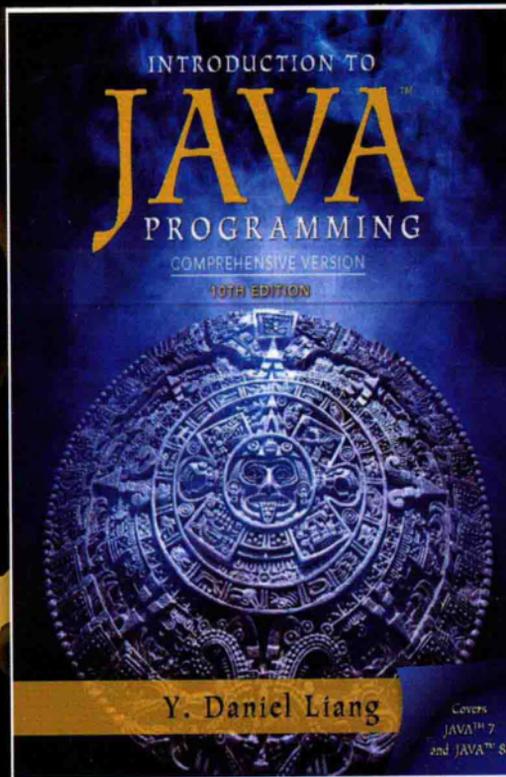


Java语言程序设计 (基础篇)

[美] 梁勇 (Y. Daniel Liang) 著 戴开宇 译
阿姆斯特朗亚特兰大州立大学 复旦大学

Introduction to Java Programming
Comprehensive Version Tenth Edition



Java语言程序设计 (基础篇) 原书第10版

Introduction to Java Programming Comprehensive Version Tenth Edition

本书是Java语言的经典教材,多年来畅销不衰。本书全面整合了Java 8的特性,采用“基础优先,问题驱动”的教学方式,循序渐进地介绍了程序设计基础、解决问题的方法、面向对象程序设计、图形用户界面设计、异常处理、I/O和递归等内容。此外,本书还全面且深入地覆盖了一些高级主题,包括算法和数据结构、多线程、网络、国际化、高级GUI等内容。

本书中文版由《Java语言程序设计 基础篇》和《Java语言程序设计 进阶篇》组成。基础篇对应原书的第1~18章,进阶篇对应原书的第19~33章。为满足对Web设计有浓厚兴趣的同学,本版在配套网站上增加了第34~42章的内容,以提供更多的相关信息。

本书特点

- 基础篇介绍基础内容,进阶篇介绍高级内容,便于教师按需选择理想的教材。
- 全面整合了Java 8的特性,对全书的内容进行了修订和更新,以反映Java程序设计的最新技术发展。
- 对面向对象程序设计进行了深入论述,包含GUI程序设计的基础和扩展。
- 提供的大量示例中都包括了对问题求解的详细步骤,很多示例都是随着Java技术的引入不断地进行增强,这种循序渐进的讲解方式更易于学生学习。
- 用JavaFX取代了Swing,极大地简化了GUI编程,比Swing更易于学习。
- 更多有趣示例和练习,激发学生兴趣。在配套网站上还为教师提供了100多道的编程练习题。

作者简介

梁勇 (Y. Daniel Liang) 现为阿姆斯特朗亚特兰大州立大学计算机科学系教授。之前曾是普度大学计算机科学系副教授,并两次获得普度大学杰出研究奖。他所编写的Java教程在美国大学Java课程中采用率极高,同时他还兼任Prentice Hall Java系列丛书的编辑。他是“Java Champion”荣誉得主,并在世界各地为在校学生和程序员做JAVA程序设计方法及技术方面的讲座。

译者简介

戴开宇 复旦大学软件学院教师,工程硕士导师,中国计算机学会会员。博士毕业于上海交通大学计算机应用专业,2011~2012年在美国佛罗里达大学作访问学者。承担多门本科专业课程、通识教育课程以及工程硕士课程,这些课程被评为校精品课程,上海市重点建设课程,IBM-教育部精品课程等。



PEARSON

投稿热线: (010) 88379604
客服热线: (010) 88378991 88361066
购书热线: (010) 68326294 88379649 68995259

华章网站: www.hzbook.com
网上购书: www.china-pub.com
数字阅读: www.hzmedia.com.cn



上架指导: 计算机程序设计

ISBN 978-7-111-50690-4



9 787111 506904 >

定价: 85.00元

计 算 机 科 学 丛 书

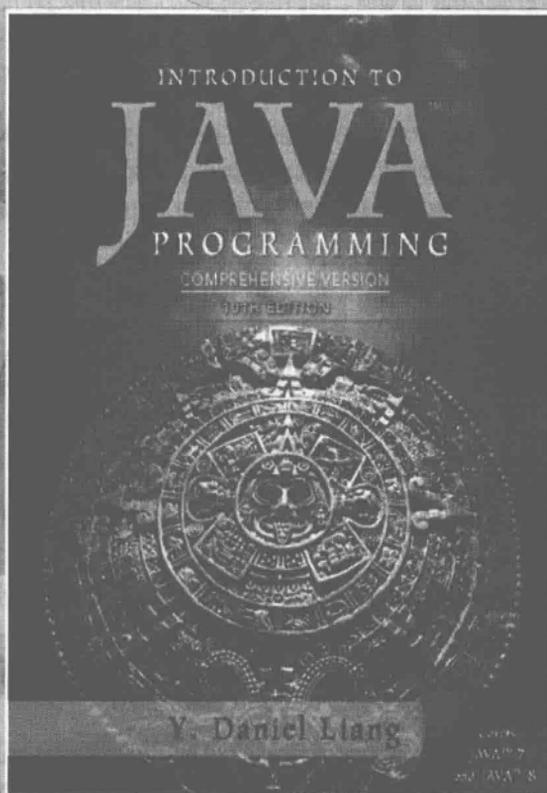
原书第10版

Java语言程序设计

(基础篇)

[美] 梁勇 (Y. Daniel Liang) 著 戴开宇 译
阿姆斯特朗亚特兰大州立大学 复旦大学

Introduction to Java Programming
Comprehensive Version Tenth Edition



机械工业出版社
China Machine Press

图书在版编目 (CIP) 数据

Java 语言程序设计 (基础篇) (原书第 10 版) / (美) 梁勇 (Liang, Y. D.) 著, 戴开宇译.
—北京: 机械工业出版社, 2015.6

(计算机科学丛书)

书名原文: Introduction to Java Programming, Comprehension Version, Tenth Edition

ISBN 978-7-111-50690-4

I. J… II. ① 梁… ② 戴… III. JAVA 语言—程序设计 IV. TP312

中国版本图书馆 CIP 数据核字 (2015) 第 131479 号

本书版权登记号: 图字: 01-2014-5466

Authorized translation from the English language edition, entitled *Introduction to Java Programming, Comprehension Version, Tenth Edition*, 978-0-13-376131-3 by Y. Daniel Liang, published by Pearson Education, Inc., Copyright © 2015, 2013, 2011.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

Chinese simplified language edition published by Pearson Education Asia Ltd., and China Machine Press Copyright © 2015.

本书中文简体字版由 Pearson Education (培生教育出版集团) 授权机械工业出版社在中华人民共和国境内 (不包括中国台湾地区和中国香港、澳门特别行政区) 独家出版发行。未经出版者书面许可, 不得以任何方式抄袭、复制或节录本书中的任何部分。

本书封底贴有 Pearson Education (培生教育出版集团) 激光防伪标签, 无标签者不得销售。

本书是 Java 语言的经典教材, 中文版分为基础篇和进阶篇, 主要介绍程序设计基础、面向对象程序设计、GUI 程序设计、数据结构和算法、高级 Java 程序设计等内容。本书以示例讲解解决问题的技巧, 提供大量的程序清单, 每章配有大量复习题和编程练习题, 帮助读者掌握编程技术, 并应用所学技术解决实际应用开发中遇到的问题。

基础篇主要介绍基本程序设计、语法结构、面向对象程序设计、继承和多态、异常处理和文本 I/O、抽象类和接口等内容。

本书可作为高等院校相关专业程序设计课程的基础教材, 也可作为 Java 语言及编程爱好者的参考资料。

出版发行: 机械工业出版社 (北京市西城区百万庄大街 22 号 邮政编码 100037)

责任编辑: 李 艺

责任校对: 殷 虹

印 刷: 三河市宏图印务有限公司

版 次: 2015 年 7 月第 1 版第 1 次印刷

开 本: 185mm × 260mm 1/16

印 张: 42.25

书 号: ISBN 978-7-111-50690-4

定 价: 85.00 元

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

客服热线: (010) 88378991 88361066

投稿热线: (010) 88379604

购书热线: (010) 68326294 88379649 68995259

读者信箱: hzjsj@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问: 北京大成律师事务所 韩光 / 邹晓东

文艺复兴以来，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的优势，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅肇划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自1998年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与 Pearson, McGraw-Hill, Elsevier, MIT, John Wiley & Sons, Cengage 等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出 Andrew S. Tanenbaum, Bjarne Stroustrup, Brain W. Kernighan, Dennis Ritchie, Jim Gray, Afred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, Abraham Silberschatz, William Stallings, Donald E. Knuth, John L. Hennessy, Larry L. Peterson 等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力相助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专门为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方式如下：

华章网站：www.hzbook.com

电子邮件：hzsj@hzbook.com

联系电话：(010) 88379604

联系地址：北京市西城区百万庄南街1号

邮政编码：100037



华章科技图书出版中心

中文版序

Introduction to Java Programming, Comprehension Version, Tenth Edition

Welcome to the Chinese translation of Introduction to Java Programming Tenth Edition. The first edition of the English version was published in 1998. Since then ten editions of the book have been published in the last seventeen years. Each new edition substantially improved the book in contents, presentation, organization, examples, and exercises. This book is now the #1 selling computer science textbook in the US. Hundreds and thousands of students around the world have learned programming and problem solving using this book.

I thank Dr. Kaiyu Dai of Fudan University for translating this latest edition. It is a great honor to reconnect with Fudan through this book. I personally benefited from teachings of many great professors at Fudan. Professor Meng Bin made Calculus easy with many insightful examples. Professor Liu Guangqi introduced multidimensional mathematic modeling in the Linear Algebra class. Professor Zhang Aizhu laid a solid mathematical foundation for computer science in the discrete mathematics class. Professor Xia Kuanli paid a great attention to small details in the PASCAL course. Professor Shi Bole showed many interesting sort algorithms in the data structures course. Professor Zhu Hong required an English text for the algorithm design and analysis course. Professor Lou Rongsheng taught the database course and later supervised my master's thesis.

My study at Fudan and teaching in the US prepared me to write the textbook. The Chinese teaching emphasizes on the fundamental concepts and basic skills, which is exactly I used to write this book. The book is fundamentals first by introducing basic programming concepts and techniques before designing custom classes. The fundamental-first approach is now widely adopted by the universities in the US. With the excellent translation from Dr. Dai, I hope more students will benefit from this book and excel in programming and problem solving.

欢迎阅读本书第 10 版的中文版。本书英文版的第 1 版于 1998 年出版。自那之后的 17 年中，本书共出版了 10 个版本。每个新的版本都在内容、表述、组织、示例以及练习题等方面进行了大量的改进。本书目前在美国计算机科学类教材中销量排名第一。全世界无数的学生通过本书学习程序设计以及问题求解。

感谢复旦大学的戴开宇博士翻译了这一最新版本。非常荣幸通过这本书和复旦大学重建联系，我本人曾经受益于复旦大学的许多杰出教授：孟斌教授采用许多富有洞察力的示例将微积分变得清晰易懂；刘光奇教授在线性代数课堂上介绍了多维度数学建模；张霁珠教授的离散数学课程为计算机科学的学习打下了坚实的数学基础；夏宽理教授在 Pascal 课程中对许多小的细节给予了极大的关注；施伯乐教授在数据结构课程中演示了许多有趣的排序算法；朱洪教授在算法设计和分析课程中使用了英文教材；楼荣生教授讲授了数据库课程，并且指导了我的硕士论文。

我在复旦大学的学习经历以及美国的授课经验为撰写本书奠定了基础。中国的教学重视基本概念和基础技能，这也是我写这本书所采用的方法。本书采用基础为先的方法，在介绍设计自定义类之前首先介绍了基本的程序设计概念和方法。目前，基础为先的方法也被美国的大学广泛采用。我希望通过戴博士的优秀翻译，让更多的学生从中受益，并在程序设计和问题求解方面出类拔萃。

梁勇

y.daniel.liang@gmail.com

www.cs.armstrong.edu/liang

Java 是一门伟大的程序设计语言，同时，它还是基于 Java 语言从嵌入式开发到企业级开发的平台。在风起云涌的计算机技术发展历程中，Java 的身影随处可见，而且生命力极其强大。1995 年，Java Applet 使得 Web 网页可以表现精彩和互动的多媒体内容，促进了 Web 的蓬勃发展。之后随着 Web 的发展，应用 Web 成为大型应用开发的主流方式，Java 凭借其“一次编译，到处运行”的特性很好地支持了互联网应用所要求的跨平台能力，成为服务器端开发的主流语言。Java EE 至今依然是最重要的企业开发服务器端平台。2004 年再次产生了对 Web 客户端体验的强烈需求，促使富因特网应用技术广泛流行，从 Java Web Start 到现在的 JavaFX，都是重要的富因特网应用技术。现在我们进入了移动互联网时代，而 Java 依然是当之无愧的主角。从第一阶段移动互联网中的 J2ME，到目前移动操作系统中全球占据份额最大的 Android 系统上的 App 开发，都采用的是 Java 语言和平台。云计算、大数据、物联网、可穿戴设备等技术的应用，都需要可以跨平台、跨设备的分布式计算环境，我们依然会看到 Java 语言在其中的关键作用。除此之外，Java 还是一门非常优秀的教学语言。它是一门经典的面向对象编程语言，拥有优雅和尽量简明的语法以及丰富的实用类库，让编程人员可以尽可能地将精力集中在业务领域的问题求解上。许多开源项目和科研中的原型系统都是采用 Java 实现的。课堂教学采用的语言同时在工业界和学术领域具有如此广泛的应用，对于学生今后的科研和工作都有直接帮助。我曾经对美国计算机专业排名靠前的几十所大学的相关课程进行调研，这些著名大学的编程课程中绝大部分选用了 Java 语言进行教学。

在多年前机械工业出版社举办的一次教学研讨会上，我有幸认识了原书的作者梁勇（Y. Daniel Liang）教授并进行了交流。那次会议之后我开始在主讲的程序设计课程中采用本书英文版作为教材，在同行和学生中得到了良好反响。作为复旦校友，梁教授对中国学生的情况非常了解，书中没有过于晦涩的词汇和表达，所以本英文教材非常适合中国学生的英文基础。更重要的是，本书知识点全面，体系结构清晰，重点突出、文字准确，内容组织循序渐进，并有大量精选的示例和配套素材，比如精心设计的大量练习题，甚至在配套网站中有支持教学的大量动画演示。本书采用基础优先的方式，从编程基础开始，逐步引入面向对象思想，最后介绍应用框架，这样很适合程序设计入门的学生。另外，强调面向问题求解的教学方法是本书特色，这也是我在课堂上一一直遵循的教学方法。通过生动实用的例子来引导学生学习程序设计课程，避免了枯燥的语法学习，让学生学以致用，并且可以举一反三。程序设计课堂最重要的是要培养学生的计算思维，这对学生综合素质的培养以及其他知识的学习，都是很有裨益的。掌握了程序设计的思维，可以很方便地学习和使用其他编程语言。该版本的另一特色是对最新 Java 语言特色的跟进，即基于 Java 最新版本 8 进行介绍。这是 Java 语言变动非常大的一个版本，比如对 JavaFX 的全面引入以及并行计算的支持等，都反映了最新的计算机技术和应用特点。相应地，教材也进行了大幅更新。我很荣幸成为本书第 10 版的译者，让中国的读者可以通过这一最新版本的中文版方便地学习程序设计相关知识。

在本书的翻译过程中，我得到了原书作者梁勇教授的大力支持。非常感谢他不仅对我邮件中的一些问题进行快速回复和详细解答，还拨冗写了中文版序，其一丝不苟的学术精神让人感动。感谢机械工业出版社的朱劼编辑，她在本书的整个翻译过程中提供了许多帮助。感谢李艺编辑等其他出版社工作人员以及本书前一版的译者，本书的出版也得益于他们的工作。最后要感谢我的家人在翻译过程中给予的支持和鼓励。由于经验不足和水平有限，书中一定会存在许多问题，敬请得到大家的指正。你们善意的指正，对我和阅读本书的许多读者是有益的。

戴开宇

2015年4月

许多读者就本书之前的版本给出了很多反馈。这些评论和建议极大地改进了本书。这一版从表述、组织、示例、练习题以及附录方面都进行了极大的增强，包括：

- 用 JavaFX 取代了 Swing。JavaFX 是一个用于开发 Java GUI 程序的新框架，它极大地简化了 GUI 程序设计，比 Swing 更易于学习。
- 在 GUI 程序设计之前介绍异常处理、抽象类和接口，若教师选择不教授 GUI 的内容，可以直接跳过第 14 ~ 16 章。
- 在第 4 章便开始介绍对象和字符串，从而使得学生可以较早地使用对象和字符串来开发有趣的程序。
- 包含更多新的有趣示例和练习题，用于激发学生兴趣。在配套网站 (www.cs.armstrong.edu/liang/intro10e/ 或 www.pearsonhighered.com/liang) 上还还为教师提供了 100 多道编程练习题。

本书采用基础优先的方法，在设计自定义类之前，首先介绍基本的程序设计概念和技术。选择语句、循环、方法和数组这样的基本概念和技术是程序设计的基础，它们为学生进一步学习面向对象程序设计和高级 Java 程序设计做好准备。

本书以问题驱动的方式来教授程序设计，将重点放在问题的解决而不是语法上。我们通过使用在各种应用情景中引发思考的问题，使得程序设计的介绍也变得更加有趣。前面章节的主线放在问题的解决上，引入合适的语法和库以支持编写解决问题的程序。为了支持以问题驱动的方式来教授程序设计，本书提供了大量不同难度的问题来激发学生的积极性。为了吸引各个专业的学生来学习，这些问题涉及很多应用领域，包括数学、科学、商业、金融、游戏、动画以及多媒体等。

本书将程序设计、数据结构和算法无缝集成在一起，采用一种实用性的方式来教授数据结构。首先介绍如何使用各种数据结构来开发高效的算法，然后演示如何实现这些数据结构。通过实现，学生获得关于数据结构效率，以及如何和何时使用某种数据结构的深入理解。最后，我们设计和实现了针对树和图的自定义数据结构。

本书广泛应用于全球各大学的程序设计入门、数据结构和算法课程中。完全版[⊖]包括程序设计基础、面向对象程序设计、GUI 程序设计、数据结构、算法、并行、网络、数据库和 Web 程序设计。这个版本旨在把学生培养成精通 Java 的程序员。基础篇可用于程序设计的第一门课程（通常称为 CS1）。基础篇包含完全版的前 18 章内容，前 13 章适合准备 AP 计算机科学考试（AP Computer Science Exam）的人员使用。

教授编程的最好途径是通过示例，而学习编程的唯一途径是通过动手练习。本书通过示例对基本概念进行了解释，提供了大量不同难度的练习题供学生进行实践。在我们的程序设计课程中，每次课后都布置了编程练习。

[⊖] 本书中文版将完全版分为基础篇和进阶篇出版，基础篇对应原书第 1 ~ 18 章，进阶篇对应原书第 19 ~ 33 章，您手中的这一本是基础篇。——编辑注

我们的目标是编写一本可以通过各种应用场景中的有趣示例来教授问题求解和程序设计的教材。如果您有任何关于如何改进本书的评论或建议，请通过以下方式与我联系。

Y. Daniel Liang

y.daniel.liang@gmail.com

www.cs.armstrong.edu/liang

www.pearsonhighered.com/liang

本版新增内容

本版对各个细节都进行了全面修订，以增强其清晰性、表述、内容、例子和练习题。本版主要的改进如下：

- 更新到 Java 8 版本。
- 由于 Swing 被 JavaFX 所替代，因此所有的 GUI 示例和练习题都使用 JavaFX 改写。
- 使用 lambda 表达式来简化 JavaFX 和线程中的编程。
- 在配套网站上为教师提供了 100 多道编程练习题，并给出了答案。这些练习题没有出现在教材中。
- 在第 4 章就引入了数学方法，使得学生可以使用数学函数编写代码。
- 在第 4 章就引入了字符串，使得学生可以早点使用对象和字符串开发有趣的程序。
- GUI 编程放在抽象类和接口之后介绍，若教师选择不教授 GUI 内容的话，可以直接跳过这些章节。
- 第 4、14、15 和 16 章是全新的章节。
- 第 28 和 29 章大幅改写，对最小生成树和最短路径使用更加简化的方法实现。

教学特色

本书使用以下要素组织素材：

- **教学目标** 在每章开始处列出学生应该掌握的内容，学完这章后，学生能够判断自己是否达到这个目标。
- **引言** 提出代表性的问题，以便学生对该章内容有一个概括了解。
- **要点提示** 突出每节中涵盖的重要概念。
- **复习题** 按节组织，帮助学生复习相关内容并评估掌握的程度。
- **示例学习** 通过精心挑选示例，以容易理解的方式教授问题求解和程序设计概念。本书使用多个小的、简单的、激发兴趣的例子来演示重要的概念。
- **本章小结** 回顾学生应该理解和记住的重要主题，有助于巩固该章所学的关键概念。
- **测试题** 测试题是在线的，让学生对编程概念和技术进行自我测试。
- **编程练习题** 为学生提供独立应用所学新技能的机会。练习题的难度分为容易（没有星号）、适中（*）、难（**）和具有挑战性（***）四个级别。学习程序设计的窍门就是实践、实践、再实践。所以，本书提供了大量的编程练习题。
- **注意、提示、警告和设计指南** 贯穿全书，对程序开发的重要方面提供有价值的建议和见解。
 - **注意** 提供学习主题的附加信息，巩固重要概念。

- **提示** 教授良好的程序设计风格和实践经验。
- **警告** 帮助学生避开程序设计错误的误区。
- **设计指南** 提供设计程序的指南。

灵活的章节顺序

本书提供灵活的章节顺序，使学生可以或早或晚地了解 GUI、异常处理、递归、泛型和 Java 集合框架等内容。下页的插图显示了各章之间的相关性。

本书的组织

所有的章节分为五部分，构成 Java 程序设计、数据结构和算法、数据库和 Web 程序设计的全面介绍。因为知识是循序渐进的，前面的章节介绍了程序设计的基本概念，并且通过简单的例子和练习题指导学生；后续的章节逐步详细地介绍 Java 程序设计，最后介绍开发综合的 Java 应用程序。附录包含各种主题，包含数系、位操作、正则表达式以及枚举类型。

第一部分 程序设计基础（第 1～8 章）

本书第一部分是基石，让你开始踏上 Java 学习之旅。你将开始了解 Java（第 1 章），还将学习像基本数据类型、变量、常量、赋值、表达式以及操作符这样的基本程序设计技术（第 2 章），选择语句（第 3 章），数学函数、字符和字符串（第 4 章），循环（第 5 章），方法（第 6 章），数组（第 7～8 章）。在第 7 章之后，可以跳到第 18 章去学习如何编写递归的方法来解决本身具有递归特性的问题。

第二部分 面向对象程序设计（第 9～13 章和第 17 章）

这一部分介绍面向对象程序设计。Java 是一种面向对象程序设计语言，它使用抽象、封装、继承和多态来提供开发软件的极大灵活性、模块化和可重用性。你将学习如何使用对象和类进行程序设计（第 9～10 章）、类的继承（第 11 章）、多态性（第 11 章）、异常处理（第 12 章）、抽象类（第 13 章）以及接口（第 13 章）。文本 I/O 将在第 12 章介绍，二进制 I/O 将在第 17 章介绍。

第三部分 GUI 程序设计（第 14～16 章和奖励章节第 34 章）

JavaFX 是一个开发 Java GUI 程序的新框架。它不仅对于开发 GUI 程序有用，还是一个用于学习面向对象程序设计的优秀教学工具。这一部分中在第 14～16 章介绍使用 JavaFX 的 Java GUI 程序设计。主要的主题包括 GUI 基础（第 14 章）、容器面板（第 14 章）、绘制形状（第 14 章）、事件驱动编程（第 15 章）、动画（第 15 章）、GUI 组件（第 16 章），以及播放音频和视频（第 16 章）。你将学习采用 JavaFX 的 GUI 程序设计的架构，并且使用组件、形状、面板、图像和视频来开发有用的应用程序。第 34 章涵盖 JavaFX 的高级特性。

第四部分 数据结构和算法（第 18～29 章和奖励章节第 40～41 章）

这一部分介绍经典数据结构和算法课程中的主要内容。第 18 章介绍递归来编写解决本身具有递归特性的问题的方法。第 19 章介绍泛型来提高软件的可靠性。第 20 和 21 章介绍 Java 集合框架，它为数据结构定义了一套有用的 API。第 22 章讨论算法效率的度量以便给应用程序选择合适的算法。第 23 章介绍经典的排序算法。你将在第 24 章中学到如何实现经典的数据结构，如列表、队列和优先队列。第 25 和 26 章介绍二分查找树和 AVL 树。第 27

第一部分：程序设计基础

- 第1章 计算机、程序和Java概述
- 第2章 基本程序设计
- 第3章 选择
- 第4章 数学函数、字符和字符串
- 第5章 循环
- 第6章 方法
- 第7章 一维数组
- 第8章 多维数组

第二部分：面向对象程序设计

- 第9章 对象和类
- 第10章 面向对象思考
- 第11章 继承和多态
- 第12章 异常处理和文本I/O
- 第13章 抽象类和接口
- 第17章 二进制I/O

第三部分：GUI 程序设计

- 第14章 JavaFX 基础
- 第15章 事件驱动编程和动画
- 第16章 JavaFX UI 组件和多媒体
- 第34章 高级 GUI 程序设计

第四部分：数据结构和算法

- 第7章 → 第18章 递归
- 第13章 → 第19章 泛型
- 第20章 列表、栈、队列和优先队列
- 第21章 集合与映射
- 第22章 开发高级算法
- 第23章 排序
- 第24章 实现列表、栈、队列和优先队列
- 第25章 二分查找树
- 第26章 AVL 树
- 第27章 散列
- 第28章 图及其应用
- 第29章 加权图及其应用
- 第40章 2-4 树和 B 树
- 第41章 红黑树

第五部分：高级 Java 程序设计

- 第16章 → 第30章 多线程和并行程序设计
- 第31章 网络
- 第32章 Java 数据库程序设计
- 第33章 JSF
- 第35章 高级数据库程序设计
- 第36章 国际化
- 第37章 Servlet
- 第38章 JSP
- 第39章 Web 服务
- 第9章 → 第42章 使用JUnit 测试

注意：第34~42章是奖励章节，可以从配套网站上得到。

章介绍散列以及通过散列实现映射 (map) 和集合 (set)。第 28 和 29 章介绍图的应用。2-4 树、B 树以及红黑树在奖励章节第 40 ~ 41 章中介绍。

第五部分 高级 Java 程序设计 (第 30 ~ 33 章、奖励章节第 35 ~ 39 章及第 42 章)

这一部分介绍高级 Java 程序设计。第 30 章介绍使用多线程使程序具有更好的响应和交互性, 并介绍并行编程。第 31 章讨论如何编写程序使得 Internet 上的不同主机能够相互对话。第 32 章介绍使用 Java 来开发数据库项目。第 33 章介绍使用 JavaServer Faces 进行现代 Web 应用程序开发。第 35 章探究高级 Java 数据库程序设计。第 36 章涵盖国际化支持的使用, 以开发面向全球使用者的项目。第 37 和 38 章介绍如何使用 Java servlet 和 JSP 创建来自 Web 服务器的动态内容。第 39 章讨论 Web 服务。第 42 章介绍使用 JUnit 测试 Java 程序。

附录

附录 A 列出 Java 关键字。附录 B 给出十进制和十六进制 ASCII 字符集。附录 C 给出操作符优先级。附录 D 总结 Java 修饰符和它们的使用。附录 E 讨论特殊的浮点值。附录 F 介绍数系以及二进制、十进制和十六进制间的转换。附录 G 介绍位操作。附录 H 介绍正则表达式。附录 I 涵盖枚举类型。

Java 开发工具

可以使用 Windows 记事本 (NotePad) 或写字板 (WordPad) 这样的文本编辑器创建 Java 程序, 然后从命令窗口编译、运行这个程序。也可以使用 Java 开发工具, 例如, NetBeans 或者 Eclipse。这些工具支持快速开发 Java 应用程序的集成开发环境 (IDE), 编辑、编译、构建、运行和调试程序都集成在一个图形用户界面中。有效地使用这些工具可以极大地提高编写程序的效率。如果按照教程学习, NetBeans 和 Eclipse 也是易于使用的。关于 NetBeans 和 Eclipse 的教程, 参见配套网站。

学生资源

学生资源可以从本书的配套网站得到, 具体包括:

- 复习题的答案。
- 偶数号编程练习题的解答。
- 本书例子的源代码。
- 交互式的自测题 (按章节组织)。
- 补充材料。
- 调试技巧。
- 算法动画。
- 勘误表。

教师资源[⊖]

教师资源包括:

⊖ 关于本书教辅资源, 用书教师可向培生教育出版集团北京代表处申请, 电话: 010-57355169/57355171, 电子邮件: service.cn@pearson.com。——编辑注

- PowerPoint 教学幻灯片，通过交互性的按钮可以观看彩色并且语法项高亮显示的源代码，并可以不离幻灯片运行程序。
- 所有编程练习题的答案。学生只可以得到偶数号练习题的答案。
- 100 多道编程练习题，按章节组织。这些练习题仅对教师开放，并提供答案。
- 基于 Web 的测试题生成器。(教师可以选择章节以从 2000 多个大型题库中生成测试题。)
- 样卷。大多数试卷包含 4 个部分：
 - 多选题或者简答题。
 - 改正编程错误。
 - 跟踪程序。
 - 编写程序。
- ACM/IEEE 课程体系 2013 版。新的 ACM/IEEE 计算机科学课程体系 2013 版将知识主体组织成 18 个知识领域。为了帮助教师基于本书设计课程，我们提供了示例教学大纲来确定知识领域和知识单元。示例教学大纲用于一个三学期的课程系列，作为一个学院自定义 (institutional customization) 示例。
- 具有 ABET 课程评价的样卷。
- 课程项目。通常，每个项目给出一个描述，并且要求学生分析、设计和实现该项目。

致谢

感谢阿姆斯特朗亚特兰大州立大学给我机会讲授我所写的内容，并支持我将所教的内容编写成教材。教学是我持续改进本书的灵感之源。感谢使用本书的教师和学生提出的评价、建议、错误报告和赞扬。

由于有了对本版和以前版本的富有见解的审阅，本书得到很大的改进。感谢以下审阅人员：Elizabeth Adams (James Madison University), Syed Ahmed (North Georgia College and State University), Omar Aldawud (Illinois Institute of Technology), Stefan Andrei (Lamar University), Yang Ang (University of Wollongong, Australia), Kevin Bierre (Rochester Institute of Technology), David Champion (DeVry Institute), James Chegwidden (Tarrant County College), Anup Dargar (University of North Dakota), Charles Dierbach (Towson University), Frank Ducrest (University of Louisiana at Lafayette), Erica Eddy (University of Wisconsin at Parkside), Deena Engel (New York University), Henry A Etlinger (Rochester Institute of Technology), James Ten Eyck (Marist College), Myers Foreman (Lamar University), Olac Fuentes (University of Texas at El Paso), Edward F. Gehringer (North Carolina State University), Harold Grossman (Clemson University), Barbara Guillot (Louisiana State University), Stuart hansen (University of Wisconsin, Parkside), Dan Harvey (Southern Oregon University), Ron Hofman (Red River College, Canada), Stephen Hughes (Roanoke College), Vladan Jovanovic (Georgia Southern University), Edwin Kay (Lehigh University), Larry King (University of Texas at Dallas), Nana Kofi (Langara College, Canada), George Koutsogiannakis (Illinois Institute of Technology), Roger Kraft (Purdue University at Calumet), Norman Krumpe (Miami University), Hong Lin (DeVry Institute), Dan Lipsa (Armstrong Atlantic State University), James Madison (Rensselaer Polytechnic Institute), Frank Malinowski (Darton College), Tim Margush (University

of Akron), Debbie Masada (Sun Microsystems), Blayne Mayfield (Oklahoma State University), John McGrath (J.P. McGrath Consulting), Hugh McGuire (Grand Valley State), Shyamal Mitra (University of Texas at Austin), Michel Mitri (James Madison University), Kenrick Mock (University of Alaska Anchorage), Frank Murgolo (California State University, Long Beach), Jun Ni (University of Iowa), Benjamin Nystuen (University of Colorado at Colorado Springs), Maureen Opkins (CA State University, Long Beach), Gavin Osborne (University of Saskatchewan), Kevin Parker (Idaho State University), Dale Parson (Kutztown University), Mark Pendergast (Florida Gulf Coast University), Richard Povinelli (Marquette University), Roger Priebe (University of Texas at Austin), Mary Ann Pumphrey (De Anza Junior College), Pat Roth (Southern Polytechnic State University), Amr Sabry (Indiana University), Ben Setzer (Kennesaw State University), Carolyn Schauble (Colorado State University), David Scuse (University of Manitoba), Ashraf Shirani (San Jose State University), Daniel Spiegel (Kutztown University), Joslyn A. Smith (Florida Atlantic University), Lixin Tao (Pace University), Ronald F. Taylor (Wright State University), Russ Tront (Simon Fraser University), Deborah Trytten (University of Oklahoma), Michael Verdicchio (Citadel), Kent Vidrine (George Washington University), Bahram Zartoshty (California State University at Northridge)。

能够与 Pearson 出版社一起工作，我感到非常愉快和荣幸。感谢 Tracy Johnson 和她的同事 Marcia Horton、Yez Alayan、Carole Snyder、Scott Disanno、Bob Engelhardt、Haseen Khan，感谢他们组织、开展和积极促进本项目。

一如既往，感谢我妻子 Samantha 的爱、支持和鼓励。

出版者的话	
中文版序	
译者序	
前言	
第1章 计算机、程序和Java概述	1
1.1 引言	1
1.2 什么是计算机	2
1.2.1 中央处理器	2
1.2.2 比特和字节	3
1.2.3 内存	3
1.2.4 存储设备	4
1.2.5 输入和输出设备	4
1.2.6 通信设备	5
1.3 编程语言	6
1.3.1 机器语言	6
1.3.2 汇编语言	6
1.3.3 高级语言	7
1.4 操作系统	8
1.4.1 控制和监视系统的活动	8
1.4.2 分配和调配系统资源	8
1.4.3 调度操作	8
1.5 Java、万维网以及其他	9
1.6 Java语言规范、API、JDK和IDE ..	10
1.7 一个简单的Java程序	11
1.8 创建、编译和执行Java程序	13
1.9 程序设计风格和文档	16
1.9.1 正确的注释和注释风格	16
1.9.2 正确的缩进和空白	16
1.9.3 块的风格	17
1.10 程序设计错误	17
1.10.1 语法错误	17
1.10.2 运行时错误	18
1.10.3 逻辑错误	18
1.10.4 常见错误	19
1.11 使用NetBeans开发Java程序	20
1.11.1 创建Java工程	20
1.11.2 创建Java类	21
1.11.3 编译和运行类	22
1.12 使用Eclipse开发Java程序	22
1.12.1 创建Java工程	22
1.12.2 创建Java类	24
1.12.3 编译和运行类	24
关键术语	25
本章小结	25
测试题	26
编程练习题	26
第2章 基本程序设计	28
2.1 引言	28
2.2 编写简单的程序	28
2.3 从控制台读取输入	31
2.4 标识符	34
2.5 变量	34
2.6 赋值语句和赋值表达式	36
2.7 命名常量	37
2.8 命名习惯	37
2.9 数值数据类型和操作	38
2.9.1 数值类型	38
2.9.2 从键盘读取数值	39
2.9.3 数值操作符	39
2.9.4 幂运算	41
2.10 数值型直接量	41
2.10.1 整型直接量	42
2.10.2 浮点型直接量	42
2.10.3 科学记数法	42
2.11 表达式求值以及操作符优先级 ..	43
2.12 示例学习：显示当前时间	44
2.13 增强赋值操作符	46
2.14 自增和自减操作符	47

2.15 数值类型转换	48	4.3 字符数据类型和操作	105
2.16 软件开发过程	50	4.3.1 Unicode和ASCII码	105
2.17 示例学习: 整钱兑零	54	4.3.2 特殊字符的转义序列	106
2.18 常见错误和陷阱	56	4.3.3 字符型数据与数值型数据 之间的转换	107
关键术语	58	4.3.4 字符的比较和测试	107
本章小结	58	4.4 String类型	109
测试题	59	4.4.1 求字符串长度	110
编程练习题	59	4.4.2 从字符串中获取字符	110
第3章 选择	64	4.4.3 连接字符串	111
3.1 引言	64	4.4.4 字符串的转换	111
3.2 boolean数据类型	64	4.4.5 从控制台读取字符串	112
3.3 if语句	66	4.4.6 从控制台读取字符	112
3.4 双分支if-else语句	68	4.4.7 字符串比较	112
3.5 嵌套的if语句和多分支if-else 语句	69	4.4.8 获得子字符串	114
3.6 常见错误和陷阱	71	4.4.9 获取字符串中的字符或者 子串	115
3.7 产生随机数	74	4.4.10 字符串和数字间的转换	116
3.8 示例学习: 计算身体质量指数	76	4.5 示例学习	117
3.9 示例学习: 计算税率	77	4.5.1 猜测生日	118
3.10 逻辑操作符	80	4.5.2 将十六进制数转换为 十进制数	121
3.11 示例学习: 判定闰年	83	4.5.3 使用字符串修改彩票程序	122
3.12 示例学习: 彩票	84	4.6 格式化控制台输出	123
3.13 switch语句	85	关键术语	126
3.14 条件表达式	88	本章小结	127
3.15 操作符的优先级和结合规则	89	测试题	127
3.16 调试	90	编程练习题	127
关键术语	91	第5章 循环	133
本章小结	91	5.1 引言	133
测试题	92	5.2 while循环	134
编程练习题	92	5.2.1 示例学习: 猜数字	136
第4章 数学函数、字符和字符串	100	5.2.2 循环设计策略	138
4.1 引言	100	5.2.3 示例学习: 多个减法测 试题	138
4.2 常用数学函数	101	5.2.4 使用标记值控制循环	140
4.2.1 三角函数方法	101	5.2.5 输入和输出重定向	141
4.2.2 指数函数方法	102	5.3 do-while循环	143
4.2.3 取整方法	102	5.4 for循环	144
4.2.4 min、max和abs方法	102	5.5 采用哪种循环	147
4.2.5 random方法	103		
4.2.6 示例学习: 计算三角形的 角度	103		

5.6 嵌套循环	149	7.2.2 创建数组	208
5.7 最小化数值错误	151	7.2.3 数组大小和默认值	209
5.8 示例学习	152	7.2.4 访问数组元素	209
5.8.1 求最大公约数	152	7.2.5 数组初始化语法	210
5.8.2 预测未来学费	154	7.2.6 处理数组	210
5.8.3 将十进制数转换为 十六进制数	155	7.2.7 foreach循环	212
5.9 关键字break和continue	156	7.3 示例学习: 分析数字	214
5.10 示例学习: 判断回文串	159	7.4 示例学习: 一副牌	215
5.11 示例学习: 显示素数	160	7.5 数组的复制	217
关键术语	162	7.6 将数组传递给方法	218
本章小结	163	7.7 从方法中返回数组	221
测试题	163	7.8 示例学习: 统计每个字母 出现的次数	221
编程练习题	163	7.9 可变长参数列表	224
第6章 方法	171	7.10 数组的查找	225
6.1 引言	171	7.10.1 线性查找法	225
6.2 定义方法	172	7.10.2 二分查找法	226
6.3 调用方法	173	7.11 数组的排序	228
6.4 void方法示例	175	7.12 Arrays类	230
6.5 通过传值进行参数传递	178	7.13 命令行参数	232
6.6 模块化代码	181	7.13.1 向main方法传递字符串	232
6.7 示例学习: 将十六进制数转换 为十进制数	183	7.13.2 示例学习: 计算器	232
6.8 重载方法	185	关键术语	234
6.9 变量的作用域	187	本章小结	235
6.10 示例学习: 生成随机字符	188	测试题	235
6.11 方法抽象和逐步求精	190	编程练习题	235
6.11.1 自顶向下的设计	191	第8章 多维数组	242
6.11.2 自顶向下和自底向上的 实现	192	8.1 引言	242
6.11.3 实现细节	193	8.2 二维数组的基础知识	242
6.11.4 逐步求精的优势	196	8.2.1 声明二维数组变量并创建 二维数组	243
关键术语	196	8.2.2 获取二维数组的长度	244
本章小结	197	8.2.3 锯齿数组	244
测试题	197	8.3 处理二维数组	245
编程练习题	197	8.4 将二维数组传递给方法	247
第7章 一维数组	207	8.5 示例学习: 多选题测验评分	248
7.1 引言	207	8.6 示例学习: 找出距离最近的 点对	249
7.2 数组的基础知识	207	8.7 示例学习: 数独	251
7.2.1 声明数组变量	208	8.8 多维数组	254

8.8.1 示例学习：每日温度和湿度	255	10.4.1 关联	316
8.8.2 示例学习：猜生日	256	10.4.2 聚集和组合	317
本章小结	258	10.5 示例学习：设计Course类	318
测试题	258	10.6 示例学习：设计栈类	320
编程练习题	258	10.7 将基本数据类型值作为对象处理	322
第9章 对象和类	270	10.8 基本类型和包装类类型之间的自动转换	325
9.1 引言	270	10.9 BigInteger和BigDecimal类	326
9.2 为对象定义类	270	10.10 String类	327
9.3 示例：定义类和创建对象	272	10.10.1 构造字符串	327
9.4 使用构造方法构造对象	277	10.10.2 不可变字符串与限定字符串	328
9.5 通过引用变量访问对象	278	10.10.3 字符串的替换和分隔	329
9.5.1 引用变量和引用类型	278	10.10.4 依照模式匹配、替换和分隔	329
9.5.2 访问对象的数据和方法	279	10.10.5 字符串与数组之间的转换	330
9.5.3 引用数据域和null值	279	10.10.6 将字符和数值转换成字符串	331
9.5.4 基本类型变量和引用类型变量的区别	280	10.10.7 格式化字符串	331
9.6 使用Java库中的类	282	10.11 StringBuilder和StringBuffer类	333
9.6.1 Date类	282	10.11.1 修改StringBuilder中的字符串	334
9.6.2 Random类	283	10.11.2 toString、capacity、length、setLength和charAt方法	335
9.6.3 Point2D类	283	10.11.3 示例学习：判断回文串时忽略既非字母又非数字的字符	336
9.7 静态变量、常量和方法	284	关键术语	338
9.8 可见性修饰符	289	本章小结	339
9.9 数据域封装	291	测试题	339
9.10 向方法传递对象参数	294	编程练习题	339
9.11 对象数组	297	第11章 继承和多态	347
9.12 不可变对象和类	299	11.1 引言	347
9.13 变量的作用域	301	11.2 父类和子类	347
9.14 this引用	302	11.3 使用super关键字	353
9.14.1 使用this引用隐藏数据域	302	11.3.1 调用父类的构造方法	353
9.14.2 使用this调用构造方法	303	11.3.2 构造方法链	354
关键术语	304		
本章小结	304		
测试题	305		
编程练习题	305		
第10章 面向对象思考	309		
10.1 引言	309		
10.2 类的抽象和封装	309		
10.3 面向对象的思考	313		
10.4 类的关系	315		

11.3.3 调用父类的方法	355	12.11.5 示例学习：替换文本	412
11.4 方法重写	356	12.12 从Web上读取数据	414
11.5 方法重写与重载	357	12.13 示例学习：Web爬虫	416
11.6 Object类及其toString()方法	359	关键术语	418
11.7 多态	359	本章小结	418
11.8 动态绑定	360	测试题	419
11.9 对象转换和instanceof运算符	363	编程练习	419
11.10 Object类的equals方法	367	第13章 抽象类和接口	424
11.11 ArrayList类	368	13.1 引言	424
11.12 对于列表有用的方法	374	13.2 抽象类	424
11.13 示例学习：自定义栈类	374	13.2.1 为何要使用抽象方法	427
11.14 protected数据和方法	376	13.2.2 抽象类的几点说明	428
11.15 防止扩展和重写	378	13.3 示例学习：抽象的Number类	429
关键术语	378	13.4 示例学习：Calendar和	
本章小结	379	GregorianCalendar	431
测试题	379	13.5 接口	434
编程练习题	380	13.6 Comparable接口	436
第12章 异常处理和文本I/O	384	13.7 Cloneable接口	440
12.1 引言	384	13.8 接口与抽象类	444
12.2 异常处理概述	385	13.9 示例学习：Rational类	447
12.3 异常类型	389	13.10 类的设计原则	452
12.4 关于异常处理的更多知识	391	13.10.1 内聚性	452
12.4.1 声明异常	392	13.10.2 一致性	452
12.4.2 抛出异常	392	13.10.3 封装性	452
12.4.3 捕获异常	393	13.10.4 清晰性	453
12.4.4 从异常中获取信息	394	13.10.5 完整性	453
12.4.5 示例学习：声明、抛出和		13.10.6 实例和静态	453
捕获异常	396	13.10.7 继承与聚合	454
12.5 finally子句	399	13.10.8 接口和抽象类	454
12.6 何时使用异常	400	关键术语	454
12.7 重新抛出异常	401	本章小结	455
12.8 链式异常	402	测试题	455
12.9 创建自定义异常类	403	编程练习题	455
12.10 File类	405	第14章 JavaFX基础	459
12.11 文件输入和输出	408	14.1 引言	459
12.11.1 使用PrintWriter写数据	408	14.2 JavaFX与Swing以及AWT的	
12.11.2 使用try-with-resources		比较	459
自动关闭资源	409	14.3 JavaFX程序的基本结构	460
12.11.3 使用Scanner读数据	410	14.4 面板、UI组件以及形状	462
12.11.4 Scanner如何工作	411	14.5 属性绑定	465

14.6	节点的通用属性和方法	468	测试题	536	
14.7	Color类	469	编程练习题	536	
14.8	Font类	470	第16章 JavaFX UI组件和多媒体	542	
14.9	Image和ImageView类	472	16.1	引言	542
14.10	布局面板	474	16.2	Labeled和Label	543
14.10.1	FlowPane	475	16.3	按钮	545
14.10.2	GridPane	477	16.4	复选框	547
14.10.3	BorderPane	478	16.5	单选按钮	549
14.10.4	HBox和VBox	480	16.6	文本域	551
14.11	形状	482	16.7	文本区域	553
14.11.1	Text	482	16.8	组合框	556
14.11.2	Line	484	16.9	列表视图	559
14.11.3	Rectangle	485	16.10	滚动条	562
14.11.4	Circle和Ellipse	487	16.11	滑动条	564
14.11.5	Arc	488	16.12	示例学习: 开发一个井字 游戏	567
14.11.6	Polygon和Polyline	491	16.13	视频和音频	572
14.12	示例学习: ClockPane类	493	16.14	示例学习: 国旗和国歌	575
关键术语		497	本章小结	577	
本章小结		498	测试题	578	
测试题		498	编程练习题	578	
编程练习题		498	第17章 二进制 I/O	584	
第15章 事件驱动编程和动画		504	17.1	引言	584
15.1	引言	504	17.2	在Java中如何处理文本I/O	584
15.2	事件和事件源	506	17.3	文本I/O与二进制I/O	585
15.3	注册处理器和处理事件	507	17.4	二进制I/O类	587
15.4	内部类	511	17.4.1	FileInputStream和 FileOutputStream	588
15.5	匿名内部类处理器	512	17.4.2	FilterInputStream和 FilterOutputStream	590
15.6	使用lambda表达式简化事件 处理	514	17.4.3	DataInputStream和 DataOutputStream	590
15.7	示例学习: 贷款计算器	517	17.4.4	BufferedInputStream和 BufferedOutputStream	594
15.8	鼠标事件	519	17.5	示例学习: 复制文件	596
15.9	键盘事件	520	17.6	对象I/O	598
15.10	可观察对象的监听器	523	17.6.1	Serializable接口	600
15.11	动画	525	17.6.2	序列化数组	601
15.11.1	PathTransition	525	17.7	随机访问文件	602
15.11.2	FadeTransition	528	关键术语	606	
15.11.3	Timeline	530			
15.12	示例学习: 弹球	532			
关键术语		535			
本章小结		535			

本章小结	606	18.10 尾递归	628
测试题	606	关键术语	629
编程练习题	606	本章小结	629
第18章 递归	609	测试题	630
18.1 引言	609	编程练习题	630
18.2 示例学习: 计算阶乘	610	附录A Java关键字	637
18.3 示例学习: 计算斐波那契数	613	附录B ASCII字符集	638
18.4 使用递归解决问题	615	附录C 操作符优先级表	639
18.5 递归辅助方法	617	附录D Java修饰符	640
18.5.1 递归选择排序	618	附录E 特殊浮点值	641
18.5.2 递归二分查找	618	附录F 数系	642
18.6 示例学习: 得到目录的大小	619	附录G 位操作	646
18.7 示例学习: 汉诺塔	621	附录H 正则表达式	647
18.8 示例学习: 分形	624	附录I 枚举类型	651
18.9 递归与迭代	627		

计算机、程序和 Java 概述

教学目标

- 理解计算机基础知识、程序和操作系统（1.2 ~ 1.4 节）。
- 阐述 Java 与万维网（World Wide Web）之间的关系（1.5 节）。
- 理解 Java 语言规范、API、JDK 和 IDE 的含义（1.6 节）。
- 编写一个简单的 Java 程序（1.7 节）。
- 在控制台上显示输出（1.7 节）。
- 解释 Java 程序的基本语法（1.7 节）。
- 创建、编译和运行 Java 程序（1.8 节）。
- 使用良好的 Java 程序设计风格和编写正确的程序文档（1.9 节）。
- 解释语法错误、运行时错误和逻辑错误的区别（1.10 节）。
- 使用 NetBeans 开发 Java 程序（1.11 节）。
- 使用 Eclipse 开发 Java 程序（1.12 节）。

1.1 引言

 **要点提示：** 本书的主旨是学习如何通过编写程序来解决问题。

本书是关于程序设计（又称编程）的。那么，什么是程序设计呢？程序设计就是创建（或者开发）软件，软件也称为程序。简言之，软件包含了指令，告诉计算机（或者计算设备）做什么。

软件遍布我们的周围，甚至在一些你认为可能不需要软件的设备中。当然，你会希望在个人计算机上找到和使用软件；但软件在运行中的飞机、汽车、手机甚至烤面包机中同样起着作用。在个人计算机上，你会使用字处理程序编写文档，使用 Web 浏览器在互联网中冲浪，使用电子邮件程序收发电子邮件。这些程序都是软件的实例。软件开发人员在称为程序设计语言的强大工具的帮助下创建软件。

本书使用 Java 程序设计语言来教授如何创建程序。程序设计语言有很多种，有些语言已有几十年的历史。每种语言都是为了实现某个特定的目的而发明的，比如，构建在以前语言的长处之上，或者为程序员提供一套全新和独特的工具。当知道有如此多可用的程序设计语言后，你自然会困惑哪种程序设计语言是最好的。但是，事实上，没有“最好”的语言。每种语言有它自己的长处和短处。有经验的程序员知道某种语言可能在某种场景下工作得很好，但是在另外一个场景中可能另外一个语言会更加合适。因此，经验丰富的程序员将尽可能掌握各种不同的程序设计语言，从而利用各种强大的软件开发工具。

如果你掌握了一种程序设计语言，应该会很容易学会其他程序设计语言。关键是学习如何使用程序设计方法来解决实际问题，这是本书的主旨。

我们即将开始一段激动人心的旅程，学习如何进行程序设计。在开始之前，很有必要复习一下计算机基础、程序和操作系统等内容。如果你已经很熟悉 CPU、内存、磁盘、操作系统以及程序设计语言等术语，那么可以跳过 1.2 ~ 1.4 节中对这些内容的回顾。

1.2 什么是计算机

要点提示：计算机是存储和处理数据的电子设备。

计算机包括硬件 (hardware) 和软件 (software) 两部分。一般来说, 硬件包括计算机中可以看得见的物理部分, 而软件提供看不见的指令, 这些指令控制硬件并且使得硬件完成特定的任务。学习一种程序设计语言, 并不一定要了解计算机硬件知识, 但是如果你了解一些硬件知识的话, 它的确可以帮助你更好地理解程序中指令对于计算机及其组成部分的功效。本节介绍计算机硬件组件及其功能。

一台计算机是由以下几个主要的硬件组件构成的 (图 1-1):

- 中央处理器 (CPU)
- 内存 (主存)
- 存储设备 (例如, 磁盘和光盘)
- 输入设备 (例如, 鼠标和键盘)
- 输出设备 (例如, 显示器和打印机)
- 通信设备 (例如, 调制解调器和网卡)



这些组件通过一个称为总线 (bus) 的子系统连接。你可以将总线想象成一个连接计算机组件的道路系统, 数据和电信号通过总线在计算机的各个部分之间传输。在个人计算机中, 总线搭建在主板上, 主板是一个连接计算机各个部分的电路板。

1.2.1 中央处理器

中央处理器 (Central Processing Unit, CPU) 是计算机的大脑。它从内存中获取指令, 然后执行这些指令。CPU 通常由两部分组成: 控制单元 (control unit) 和算术 / 逻辑单元 (arithmetic/logic unit)。控制单元用于控制和协调其他组件的动作。算术 / 逻辑单元用于完成数值运算 (加法、减法、乘法、除法) 和逻辑运算 (比较)。

现在的 CPU 都是构建在一块小小的硅半导体芯片上, 这块芯片上包含数百万称为晶体管的小电路开关, 用于处理信息。

每台计算机都有一个内部时钟, 该时钟以固定速度发射电子脉冲。这些脉冲用于控制和同步各种操作的步调。时钟速度越快, 在给定时间段内执行的指令就越多。时钟速度的计量单位是赫兹 (hertz, Hz), 1 赫兹相当于每秒 1 个脉冲。20 世纪 90 年代计算机的时钟速度通常是以兆赫 (MHz) 来表示的 (1MHz 就是 100 万 Hz)。随着 CPU 的速度不断提高, 目前计算机的时钟速度通常以千兆赫 (GHz) 来表述。Intel 公司最新处理器的运行速度大约是 3GHz。

最初被开发出来的 CPU 只有一个核 (core)。核是处理器中实现指令读取和执行的部分。

为了提高 CPU 的处理能力，芯片制造厂商现在生产包含多核的 CPU。一个多核 CPU 是一个具有两个或者更多独立核的组件。现在的消费类计算机一般具有两个、三个甚至四个独立的核。相信不久后，具有几十个甚至几百个核的 CPU 将普及。

1.2.2 比特和字节

在讨论内存前，让我们看下信息（数据和程序）是如何存储在计算机中的。

计算机就是一系列的电路开关。每个开关存在两种状态：关（off）和开（on）。简单而言，在计算机中存储信息就是将一系列的开关设置为开或者关。如果电路是开的，它的值是 1。如果电路是关的，它的值是 0。这些 0 和 1 被解释为二进制数字系统中的数，并且将它们称为比特（bit，二进制数）。

计算机中字节（byte）是最小的存储单元。每个字节由 8 个比特构成。像 3 这样的小数字就可以存储在单个字节中。为了存储单个字节放不下的大数字，计算机需要使用几个字节。

各种类型的数据（例如，数字和字符）都被编码为字节序列。程序员不需要关心数据的编码和解码，这些都是系统根据编码模式（schema）来自动完成的。编码模式是一系列的规则，控制计算机将字符、数字和符号翻译成计算机可以实际工作的数据。大多数模式将每个字符翻译成预先确定的一个比特串。例如，在流行的 ASCII 编码模式中，字符 C 是用一个字节 01000011 来表示的。

计算机的存储能力是以字节和多字节来衡量的，如下：

- 千字节（kilobyte, KB）大约是 1000 字节。
- 兆字节（megabyte, MB）大约是 100 万字节。
- 千兆字节（gigabyte, GB）大约是 10 亿字节。
- 万亿字节（terabyte, TB）大约是 1 万亿字节。

一页 Word 文档可能有 20KB。因此，1MB 可以存储 50 页的文档，1GB 可以存储 50 000 页的文档。一部两小时的高清电影可能有 8GB，因此将需要 160GB 来存储 20 部电影。

1.2.3 内存

计算机的内存由一个有序的字节序列组成，用于存储程序及程序需要的数据。你可以将内存想象成计算机执行程序的工作区域。一个程序和它的数据在被 CPU 执行前必须移到计算机的内存中。

每个字节都有一个唯一的地址，如图 1-2 所示。使用这个地址确定字节的位置，以便于存储和获取数据。因为可以按任意顺序存取字节，所以内存也被称为随机访问存储器（Random-Access Memory, RAM）。

现在的个人计算机通常至少有 4GB 的 RAM，但是它们一般装有 6 ~ 8GB 的内存。通常而言，一个计算机具有的 RAM 越多，它的运行速度越快，但是这条简单的经验法则是有限制的。

内存中字节的內容永远非空，但是它的原始内容可能对于你的程序来说是毫无意义的。一旦新的信息被放

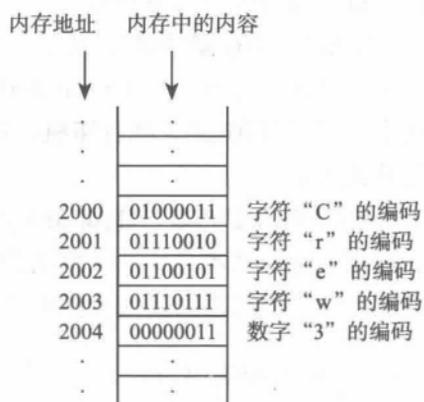


图 1-2 内存以唯一编码的内存位置来存储数据和程序指令

入内存，该字节的当前内容就会丢失。

同 CPU 一样，内存也是构建在一个表面上嵌有数百万晶体管的硅半导体芯片上。与 CPU 芯片相比，内存芯片更简单、更低速，也更便宜。

1.2.4 存储设备

计算机的内存 (RAM) 是一种易失的数据保存形式：断电时存储在内存中的信息就会丢失。程序和数据被永久地存放在存储设备上，当计算机确实要使用它们时再移入内存，因为从内存读取比从存储设备读取要快得多。

存储设备主要有以下三种类型：

- 磁盘驱动器
- 光盘驱动器 (CD 和 DVD)
- USB 闪存驱动器

驱动器 (drive) 是对存储介质进行操作的设备，例如，磁盘和光盘。存储介质物理地存储数据和程序指令。驱动器从介质读取数据并将数据写在介质上。

1. 磁盘

每台计算机至少有一个硬盘驱动器。硬盘 (hard disk) 用于永久地存储数据和程序。在较新的个人计算机上，硬盘容量一般为 500GB 到 1TB。磁盘驱动器通常安装在计算机内。此外，还有移动硬盘。

2. 光盘和数字化视频磁盘

CD 的全称是致密的盘片 (compact disc)。光盘驱动器的类型有两种：只读光盘 (CD-R) 和可读写光盘 (CD-RW)。只读光盘上的信息只能用于读取，内容一旦记录到光盘上，用户是不能修改它们的。可读写光盘可以像硬盘一样使用。也就是说，可以将数据写到光盘上，然后用新的数据覆盖掉这些数据。单张光盘的容量可以达到 700MB。大多数新型的个人计算机都安装了可读写光驱，它既支持只读光盘也支持可读写光盘。

DVD 的全称是数字化多功能碟片或者是数字化视频磁盘。DVD 和 CD 看起来很像，可以使用任意一种来存储数据。一张 DVD 上可以保存的信息要比一张 CD 上可以保存的信息多。一张标准 DVD 的存储容量是 4.7GB。如同 CD 一样，有两种类型的 DVD：DVD-R (只读) 和 DVD-RW (可重写)。

3. USB 闪存驱动器

通用串行总线 (Universal Serial Bus, USB) 接口允许用户将多种外部设备连接到计算机上。可以使用 USB 将打印机、数码相机、鼠标、外部硬盘驱动器，以及其他设备连接到计算机上。

USB 闪存驱动器 (flash drive) 是用于存储和传输数据的设备。闪存驱动器很小——大约就是一包口香糖的大小。它就像移动硬盘一样，可以插入计算机上的 USB 端口。USB 闪存驱动器目前可用的最大存储容量为 256GB。

1.2.5 输入和输出设备

输入设备和输出设备让用户可以和计算机进行通信。最常用的输入设备是键盘 (keyboard) 和鼠标 (mouse)，而最常用的输出设备是显示器 (monitor) 和打印机 (printer)。

1. 键盘

键盘是用于输入的设备。有一种便携式键盘，不带数字小键盘。

功能键 (function key) 位于键盘的最上边，而且都是以 F 为前缀。它们的功能取决于当前所使用的软件。

修饰符键 (modifier key) 是特殊键 (例如, Shift、Alt 和 Ctrl)，当它和另一个键同时按下时，会改变另一个键的常用功能。

数字小键盘 (numeric keypad) 位于键盘的右下角，是一套独立的类似计算器风格的按键集合，用于快速输入数字。

方向键 (arrow key) 位于主键盘和数字小键盘之间，在各种程序中用于上下左右地移动光标。

插入键 (Insert)、删除键 (Delete)、向上翻页键 (Page Up) 和向下翻页键 (Page Down) 分别用于在字处理和其他程序中完成插入文本和对象、删除文本和对象以及向上和向下翻页的功能。

2. 鼠标

鼠标 (mouse) 是定点设备，用来在屏幕上移动一个称为光标的图形化的指针 (通常以一个箭头的形状)，或者用于单击屏幕上的对象 (如一个按钮) 来触发它以执行动作。

3. 显示器

显示器 (monitor) 显示信息 (文本和图形)。屏幕分辨率和点距决定显示的质量。

屏幕分辨率 (screen resolution) 是指显示设备水平和垂直维度上的像素数。像素 (“图像元素”的简称) 就是构成屏幕上图像的小点。比如，对于一个 17 英寸的屏幕，分辨率一般为宽 1024 像素、高 768 像素。分辨率可以手工设置。分辨率越高，图像越锐化、越清晰。

点距 (dot pitch) 是指像素之间以毫米为单位的距离。点距越小，显示效果越好。

1.2.6 通信设备

计算机可以通过通信设备进行联网，例如，拨号调制解调器 (modulator/demodulator, 调制器 / 解调器)、DSL、电缆调制解调器、有线网络接口卡，或者无线适配器。

- 拨号调制解调器使用的是电话线，传输数据的速度可以高达 56 000bps (bps 表示每秒比特)。
- DSL (Digital Subscriber Line, 数字用户线) 使用的也是标准电话线，但是传输数据的速度比标准拨号调制解调器快 20 倍。
- 电缆调制解调器利用电缆公司维护的有线电视电缆进行数据传输，通常速度比 DSL 快。
- 网络接口卡 (NIC) 是将计算机接入局域网 (LAN) 的设备。局域网通常用于大学、商业组织和政府组织。一种称为 1000BaseT 的高速 NIC 能够以每秒 1000Mbps (Mbps 表示每秒百万比特) 的速度传输数据。
- 无线网络现在在家庭、商业和学校中极其流行。现在，每台笔记本电脑都配有无线适配器，计算机可以通过无线适配器连接到局域网和 Internet 上。

 **注意：**复习题问题的答案在配套网站上。

复习题

1.1 什么是硬件和软件？

- 1.2 列举计算机的5个主要硬件组件。
- 1.3 缩写“CPU”代表什么含义?
- 1.4 衡量CPU速度的单位是什么?
- 1.5 什么是比特? 什么是字节?
- 1.6 内存是用来做什么的? RAM代表什么? 为什么内存称为RAM?
- 1.7 用于衡量内存大小的单位是什么?
- 1.8 用于衡量磁盘大小的单位是什么?
- 1.9 内存和永久存储设备的主要不同是什么?

1.3 编程语言

要点提示: 计算机程序 (program) 称为软件 (software), 是告诉计算机该做什么的指令。

计算机不理解人类的语言, 所以, 计算机程序必须使用计算机可以使用的语言编写。现在有数百种编程语言, 对人们来说, 开发它们使编程过程更容易。但是, 所有的程序都必须转换成计算机可以执行的指令。

1.3.1 机器语言

计算机的原生语言因计算机类型的不同而有差异, 计算机的原生语言就是机器语言 (machine language), 即一套内嵌的原子指令集。因为这些指令都是以二进制代码的形式存在, 所以, 为了以机器原生语言的形式给计算机指令, 必须以二进制代码输入指令。例如, 为进行两数的相加, 可能必须写成如下的二进制形式:

```
1101101010011010
```

1.3.2 汇编语言

用机器语言进行程序设计是非常单调乏味的过程, 而且, 所编的程序也非常难以读懂和修改。为此, 在计算的早期就创建了汇编语言, 作为机器语言的替代品。汇编语言 (assembly language) 使用短的描述性单词 (称为助记符) 来表示每一条机器语言指令。例如, 助记符 `add` 一般表示数字相加, `sub` 表示数字相减。将数字 2 和数字 3 相加得到结果, 可以编写如下汇编代码:

```
add 2, 3, result
```

汇编语言的出现降低了程序设计的难度。然而, 由于计算机不理解汇编语言, 所以需要一种称为汇编器 (assembler) 的程序将汇编语言程序转换为机器代码, 如图 1-3 所示。

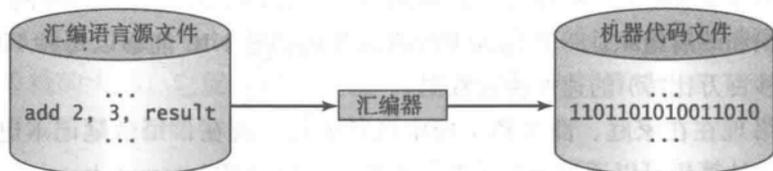


图 1-3 汇编器将汇编语言指令转换为机器代码

使用汇编语言编写代码比使用机器语言容易。然而, 用汇编语言编写代码依然很不方便。汇编语言中的一条指令对应机器代码中的一条指令。用汇编语言写代码需要知道 CPU

是如何工作的。汇编语言被认为是低级语言，因为汇编语言本质上非常接近机器语言，并且是机器相关的。

1.3.3 高级语言

20 世纪 50 年代，新一代编程语言即众所周知的高级语言出现了。它们是平台独立的，这意味着可以使用高级语言编程，然后在各种不同类型的机器上运行。高级语言很像英语，易于学习和使用。高级语言中的指令称为语句。例如，下面是计算半径为 5 的圆面积的高级语言语句：

```
area = 5 * 5 * 3.14159;
```

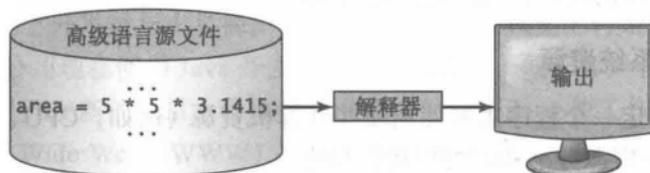
有许多高级编程语言，每种都为特定目的而设计。表 1-1 列出了一些流行的高级编程语言。

表 1-1 流行的高级编程语言

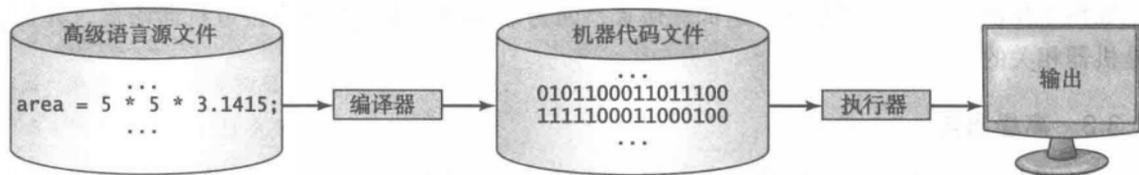
语言	描述
Ada	以 Ada Lovelace (她研究机械式的通用目的的计算机) 命名, Ada 是为美国国防部开发的, 主要用于国防项目
BASIC	初学者通用符号指令代码, 是为了让初学者易学易用而设计的
C	由贝尔实验室开发, C 语言具有汇编语言的强大功能以及高级语言的易学性和可移植性
C++	基于 C 语言开发, 是一种面向对象程序设计语言
C#	读为“C Sharp”, 由 Microsoft 公司开发的混合了 Java 和 C++ 特征的语言
COBOL	面向商业的通用语言, 是为商业应用而设计的
FORTRAN	公式翻译, 广泛用于科学和数学应用
Java	由 Sun 公司 (现在属于 Oracle) 开发, 广泛用于开发一些独立于平台的互联网应用程序
Pascal	以 Blaise Pascal (Blaise Pascal 是 17 世纪计算机器的先驱) 命名, Pascal 是一个简单的、结构化的、通用目的的语言, 主要用于编程教学
Python	一种简单的通用目的脚本语言, 适合编写小程序
Visual Basic	由 Microsoft 公司开发, 方便编程人员快速开发图形用户界面

用高级语言编写的程序称为源程序 (source program) 或源代码 (source code)。由于计算机不能运行源程序, 源程序必须被翻译成可执行的机器代码。翻译可以由另外一种称为解释器或者编译器的编程工具来完成。

- 解释器从源代码中读取一条语句, 将其翻译为机器代码或者虚拟机器代码, 然后立刻运行, 如图 1-4a 所示。请注意来自源代码的一条语句可能被翻译为多条机器指令。
- 编译器将整个源代码翻译为机器代码文件, 然后执行该机器代码文件, 如图 1-4b 所示。



a) 解释器一次翻译并且执行程序的一条语句



b) 编译器将整个源程序翻译为机器语言文件以运行

图 1-4 (续)

复习题

- 1.10 CPU 能理解什么语言?
- 1.11 什么是汇编语言?
- 1.12 什么是汇编器?
- 1.13 什么是高级编程语言?
- 1.14 什么是源程序?
- 1.15 什么是解释器?
- 1.16 什么是编译器?
- 1.17 解释语言和编译语言之间的区别是什么?

1.4 操作系统

要点提示: 操作系统 (Operating System, OS) 是运行在计算机上的最重要的程序, 它可以管理和控制计算机的活动。

流行的操作系统有 Microsoft Windows、Mac OS 以及 Linux。如果没有在计算机上安装和运行操作系统, 像 Web 浏览器或者字处理程序这样的应用程序就不能运行。硬件、操作系统、应用软件和用户之间的关系如图 1-5 所示。

操作系统的主要任务有:

- 控制和监视系统的活动
- 分配和调配系统资源
- 调度操作

1.4.1 控制和监视系统的活动

操作系统执行基本的任务, 例如, 识别来自键盘的输入, 向显示器发送输出结果, 跟踪存储设备中的文件和文件夹的动态, 控制类似硬盘驱动器和打印机这样的外部设备。操作系统还要确保不同的程序和用户同时使用计算机时不会相互干扰。另外, 操作系统还负责安全处理, 以确保未经授权的用户和程序无权访问系统。

1.4.2 分配和调配系统资源

操作系统负责确定一个程序需要使用哪些计算机资源 (例如, CPU、内存、磁盘、输入和输出设备), 并进行资源分配和调配以运行程序。

1.4.3 调度操作

操作系统负责调度程序的活动, 以便有效地利用系统资源。为了提高系统的性能, 目前

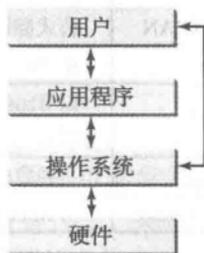


图 1-5 用户和应用程序通过操作系统访问计算机的硬件

许多操作系统都支持像多道程序设计 (multiprogramming)、多线程 (multithreading) 和多处理 (multiprocessing) 这样的技术。

多道程序设计允许多个程序通过共享 CPU 同时运行。CPU 的速度比其他组件快得多, 这样, 多数时间它都处于空闲状态, 例如, 在等待数据从磁盘或其他资源传入, 或者其他系统资源响应时。多道程序设计操作系统利用这一特点, 允许多个程序同时使用 CPU, 一旦 CPU 空闲就让别的程序使用它。例如, 在 Web 浏览器下载文件的同时, 可以用字处理程序来编辑文件。

多线程允许单个程序同时执行多个任务。例如, 字处理程序允许用户在编辑文本的同时, 将其保存到文件。在这个例子中, 编辑和保存是同一个应用程序的两个不同任务, 这两个任务可能并发运行。

多处理也称为并行处理 (parallel processing), 是指使用两个或多个处理器共同并行执行子任务, 然后将子任务的结果合并以得到整个任务的结果。它就像在外科手术中多名医生同时给一个病人做手术一样。

复习题

- 1.18 什么是操作系统? 列出一些流行的操作系统。
- 1.19 操作系统的主要任务是什么?
- 1.20 什么是多道程序设计、多线程以及多处理?

1.5 Java、万维网以及其他

要点提示: Java 是一种功能强大和多用途的编程语言, 可用于开发运行在移动设备、台式计算机以及服务器端的软件。

本书介绍 Java 程序设计。Java 是由 James Gosling 在 Sun 公司领导的小组开发的。(2010 年 Sun 公司被 Oracle 收购。) Java 最初被称为 Oak (橡树), 是 1991 年为消费类电子产品的嵌入式芯片而设计的。1995 年更名为 Java, 并重新设计用于开发 Web 应用程序。关于 Java 的历史, 参见 www.java.com/en/javahistory/index.jsp。

Java 已极其流行。Java 的快速发展以及被广泛接受都应归功于它的设计特性, 特别是它的承诺: 一次编写, 任何地方都可以运行。就像它的设计者声称的, Java 是简单的 (simple)、面向对象的 (object oriented)、分布式的 (distributed)、解释型的 (interpreted)、健壮的 (robust)、安全的 (secure)、体系结构中立的 (architecture neutral)、可移植的 (portable)、高性能的 (high performance)、多线程的 (multithreaded) 和动态的 (dynamic)。关于 Java 特性的剖析, 参见 www.cs.armstrong.edu/liang/JavaCharacteristics.pdf。

Java 是功能完善的通用程序设计语言, 可以用来开发健壮的任务关键的应用程序。现在, 它不仅用于 Web 程序设计, 而且用于在服务器、台式计算机和移动设备上开发跨平台的独立应用程序。用它开发过与火星探测器通信并控制其在火星上行走的代码。许多曾经认为 Java 言过其实的公司现在使用 Java 开发分布式应用程序, 便于客户和合作伙伴在 Internet 上访问。现在, 一旦开发新的项目, 公司都会考虑如何利用 Java 使工作变得更加容易。

万维网 (World Wide Web, WWW) 是从世界上任何地方的 Internet 都可以访问的电子信息宝库。Internet 作为万维网的基础架构已经问世四十多年。丰富多彩的万维网和设计精良的 Web 浏览器是 Internet 流行的主要原因。

Java 一开始富有吸引力是因为 Java 程序可以在 Web 浏览器中运行。这种能在 Web 浏览

器中运行的 Java 程序称为 Java 小程序 (applet)。applet 使用现代的图形用户界面与 Web 用户进行交互, 处理用户的要求, 界面中包括按钮、文本字段、文本域、单选按钮等。applet 使得 Web 更加具有响应性、交互性和趣味性。applet 内嵌在 HTML 文件中。HTML (Hypertext Markup Language) 是一种简单的脚本语言, 用于对文档布局, 链接因特网上的文档, 并且能够在万维网上提供生动的图像、声音和视频。现在, 你可以使用 Java 开发富因特网应用 (RIA)。富因特网应用作为一种 Web 应用, 被设计为可以提供通常桌面应用才具有的特性和功能。

现在, Java 广泛用于开发服务器端的应用程序。这些应用程序处理数据、执行计算, 并生成动态网页。许多商用网站后端都是采用 Java 进行开发的。

Java 是一个功能强大的程序设计语言, 可以用它来开发台式计算机、服务器以及小的手持设备上的应用程序。用于安卓手机的软件也是采用 Java 进行开发的。

复习题

- 1.21 Java 是由谁发明的? 哪个公司现在拥有 Java?
- 1.22 什么是 Java applet?
- 1.23 安卓使用的是什么编程语言?

1.6 Java 语言规范、API、JDK 和 IDE

要点提示: Java 语言规范定义了 Java 的语法, Java 库则在 Java API 中定义。JDK 是用于开发和运行 Java 程序的软件。IDE 是快速开发程序的集成开发环境。

计算机语言有严格的使用规范。如果编写程序时没有遵循这些规范, 计算机就不能理解程序。Java 语言规范和 Java API 定义 Java 的标准。

Java 语言规范 (Java language specification) 是对语言的技术定义, 包括 Java 程序设计语言的语法和语义。完整的 Java 语言规范可以在 <http://docs.oracle.com/javase/specs/> 上找到。

应用程序接口 (Application Program Interface, API) 也称为库, 包括为开发 Java 程序而预定义的类和接口。API 仍然在扩展, 在网站 <http://download.java.net/jdk8/docs/api/> 上, 可以查看和下载最新版的 Java API。

Java 是一个全面且功能强大的语言, 可用于多种用途。Java 有三个版本:

- Java 标准版 (Java Standard Edition, Java SE) 可以用来开发客户端的应用程序。应用程序可以独立运行或作为 applet 在 Web 浏览器中运行。
 - Java 企业版 (Java Enterprise Edition, Java EE) 可以用来开发服务器端的应用程序, 例如, Java servlet 和 Java Server Pages (JSP), 以及 Java Server Faces (JSF)。
 - Java 微型版 (Java Micro Edition, Java ME) 用来开发移动设备的应用程序, 例如手机。
- 本书使用 Java SE 介绍 Java 程序设计。Java SE 是基础, 其他 Java 技术都基于 Java SE。Java SE 也有很多版本, 本书采用最新的版本 Java SE 8。Oracle 发布 Java 的各个版本都带有 Java 开发工具包 (Java Development Toolkit, JDK)。Java SE 8 对应的 Java 开发工具包称为 JDK 1.8 (也称为 Java 8 或者 JDK 8)。

JDK 是由一套独立程序构成的集合, 每个程序都是从命令行调用的, 用于开发和测试 Java 程序。除了 JDK, 还可以使用某种 Java 开发工具 (例如, NetBeans、Eclipse 和 TextPad) —— 为了快速开发 Java 程序而提供集成开发环境 (Integrated Development Environment, IDE) 的软件。编辑、编译、链接、调试和在线帮助都集成在一个图形用户界

面中，这样，只需在一个窗口中输入源代码或在窗口中打开已有的文件，然后单击按钮、菜单选项或者使用功能键就可以编译和运行源代码。

☛ 复习题

- 1.24 什么是 Java 语言规范？
- 1.25 JDK 代表什么？
- 1.26 IDE 代表什么？
- 1.27 类似 NetBeans 和 Eclipse 的工具是和 Java 不同的语言吗？或者它们是 Java 的方言或者扩充？

1.7 一个简单的 Java 程序

🔑 要点提示：Java 是从类中的 main 方法开始执行的。

我们从一个简单的 Java 程序开始，该程序在控制台上显示消息“Welcome to Java!”。控制台 (console) 是一个老的计算机词汇，指计算机的文本输入和显示设备。控制台输入是指从键盘上接收输入，而控制台输出是指在显示器上显示输出。该程序如程序清单 1-1 所示。

程序清单 1-1 Welcome.java

```
1 public class Welcome {
2     public static void main(String[] args) {
3         // Display message Welcome to Java! on the console
4         System.out.println("Welcome to Java!");
5     }
6 }
```

Welcome to Java!

请注意，显示行号 (line number) 是为了引用方便，它们并不是程序的一部分。所以，不要在程序中敲入行号。

第 1 行定义了一个类。每个 Java 程序至少应该有一个类。每个类都有一个名字。按照惯例，类名都是以大写字母开头的。本例中，类名 (class name) 为 Welcome。

第 2 行定义主方法 (main method)。程序是从 main 方法开始执行的。一个类可以包含几个方法。main 方法是程序开始执行的入口。

方法是包含语句的结构体。本程序中的 main 方法包括了 System.out.println 语句。该语句在控制台上打印消息“Welcome to Java!”(第 4 行)。字符串 (string) 是一个编程术语，表示一个字符序列。一个字符串必须放入双引号中。Java 中的每条语句都以分号 (;) 结束，也称为语句结束符 (statement terminator)。

保留字 (reserved word) 或关键字 (keyword) 对编译器而言都是有特定含义的，所以不能在程序中用于其他目的。例如，当编译器看到字 class 时，它知道 class 后面的字就是这个类的名字。这个程序中的其他保留字还有 public、static 和 void。

第 3 行是注释 (comment)，它标注该程序是干什么的，以及它是如何构建的。注释帮助程序员进行相互沟通以及理解程序。注释不是程序设计语句，所以编译器编译程序时是忽略注释的。在 Java 中，在单行上用两个斜杠 (//) 引导注释，称为行注释 (line comment)；在一行或多行用 /* 和 */ 括住注释，称为块注释 (block comment)。当编译器看到 // 时，就会忽略本行 // 之后的所有文本。当看到 /* 时，它会搜索接下来的 */，并忽略掉 /* 与 */ 之间的文本。下面是这两种注释的例子：

```
// This application program displays Welcome to Java!
/* This application program displays Welcome to Java! */
/* This application program
   displays Welcome to Java! */
```

程序中的一对花括号将程序的一些组成部分组合起来，形成一个块（block）。在 Java 中，每个块以左花括号（{）开始，以右花括号（}）结束。每个类都有一个将该类的数据和放在一块的类块（class block）。每个方法都有一个将该方法中的语句放在一起的方法块（method block）。块是可以嵌套的，即一个块可以放到另一个块内，如下面代码所示。

```
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```

提示：一个左括号必须匹配一个右括号。任何时候，当输入一个左括号时，应该立即输入一个右括号来防止出现遗漏括号的错误。大多数 Java IDE 都会自动地为每个左括号插入一个右括号。

警告：Java 源程序是区分大小写的。如果在程序中将 main 替换成 Main，就会出错。

在这个程序中你可以注意到有些特殊的字符（比如，{}、//、;），它们几乎在每个程序中都会使用。表 1-2 总结了它们的用途。

表 1-2 特殊字符

字符	名称	描述
{ }	左花括号和右花括号	表示一个包含语句的块
()	左圆括号和右圆括号	和方法一起使用
[]	左方括号和右方括号	表示一个数组
//	双斜杠	表示后面是一行注释
" "	左引用符号和右引用符号	包含一个字符串（即一系列的字符）
;	分号	标识一个语句的结束

学习编程时最容易犯的错是语法错误。像其他任何一种程序设计语言一样，Java 也有自己的语法，而且你必须按照语法规则编写代码。如果你的程序违反了语法规则，例如，忘记了分号、忘记了花括号、忘记了引号，或者拼错了单词，Java 编译器会报告语法错误。可以尝试编译带有这些错误的程序，看看编译器会报告些什么。

注意：你可能想知道为什么 main 方法要以这样的方式定义，为什么使用 System.out.println(...) 在控制台上显示信息。在现阶段，你只需知道它们就是这么做的就可以。这一问题将在后续章节中得到完整的解答。

程序清单 1-1 中的程序会显示一条信息。一旦你理解了程序，很容易将该程序扩展为显示更多的信息。例如，可以改写该程序来显示三条信息，如程序清单 1-2 所示。

程序清单 1-2 WelcomeWithThreeMessages.java

```
1 public class WelcomeWithThreeMessages {
2     public static void main(String[] args) {
3         System.out.println("Programming is fun!");
4         System.out.println("Fundamentals First");
5         System.out.println("Problem Driven");
6     }
7 }
```

Programming is fun!
Fundamentals First
Problem Driven

更进一步，你可以进行数学计算，并将结果显示到控制台上。程序清单 1-3 给出一个计算 $\frac{10.5+2 \times 3}{45-3.5}$ 的例子。

程序清单 1-3 ComputeExpression.java

```
1 public class ComputeExpression {
2     public static void main(String[] args) {
3         System.out.println((10.5 + 2 * 3) / (45 - 3.5));
4     }
5 }
```

0.39759036144578314

Java 中的乘法操作符是 *。如你所看到的，将一个数学表达式翻译成 Java 表达式是一个非常直观的过程，我们将在第 2 章进一步讨论 Java 表达式。

复习题

- 1.28 什么是关键字？列举一些 Java 关键字。
- 1.29 Java 是大小写敏感的吗？Java 关键字是大写还是小写？
- 1.30 什么是注释？编译器会忽略注释吗？如何标识一行注释以及一段注释？
- 1.31 在控制台上显示一个字符串的语句是什么？
- 1.32 给出以下代码的输出：

```
public class Test {
    public static void main(String[] args) {
        System.out.println("3.5 * 4 / 2 - 2.5 is ");
        System.out.println(3.5 * 4 / 2 - 2.5);
    }
}
```

1.8 创建、编译和执行 Java 程序

要点提示：Java 源程序保存为 .java 文件，编译为 .class 文件。.class 文件由 Java 虚拟机 (JVM) 执行。

在执行程序之前，必须创建程序并进行编译。这个过程是反复执行的，如图 1-6 所示。如果程序有编译错误，必须修改程序来纠正错误，然后重新编译它。如果程序有运行时错误或者不能产生正确的结果，必须修改这个程序，重新编译，然后重新执行。

可以使用任何一个文本编辑器或者集成开发环境来创建和编辑 Java 源代码文件。本节演示如何从命令窗口创建、编译和运行 Java 程序。1.10 节和 1.11 节将介绍使用 NetBeans 和 Eclipse 来开发 Java 程序。从命令窗口，可以使用文本编辑器比如记事本 (NotePad) 来创建 Java 源代码文件，如图 1-7 所示。

注意：源文件的扩展名必须是 .java，而且文件名必须与公共类名完全相同。例如，程序清单 1-1 中源代码的文件必须命名为 Welcome.java，因为公共类的类名就是 Welcome。

Java 编译器将 Java 源文件翻译成 Java 字节码文件。下面的命令就是用来编译 Welcome.java 的：

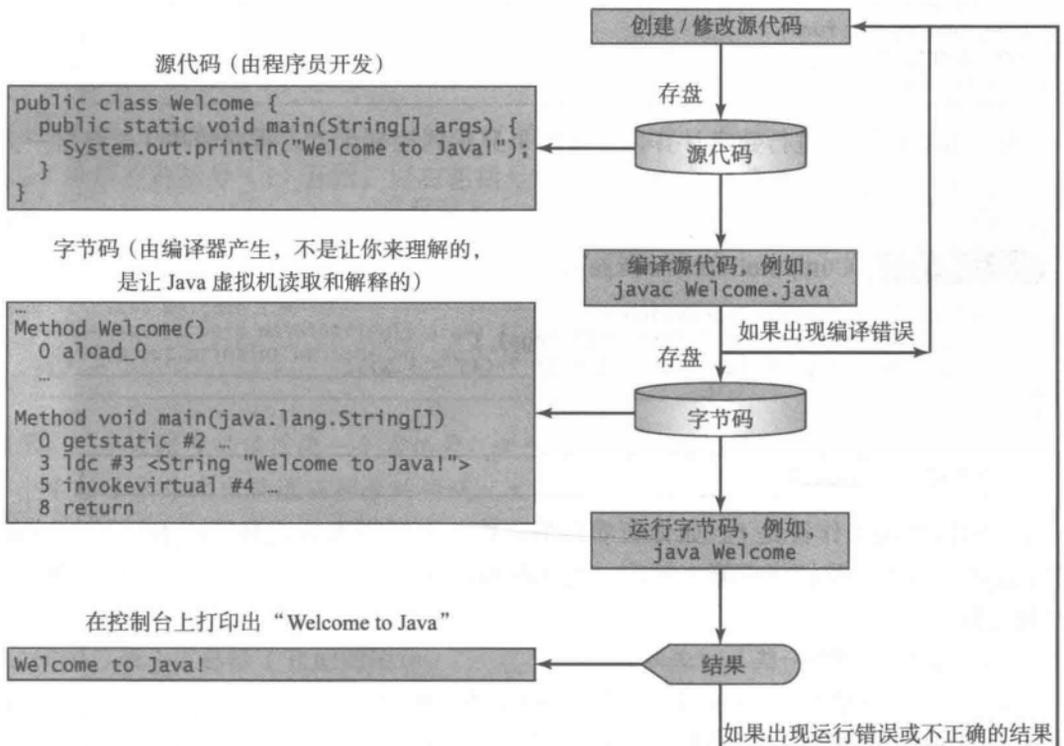


图 1-6 Java 程序开发过程就是重复地创建 / 修改源代码、编译和执行程序

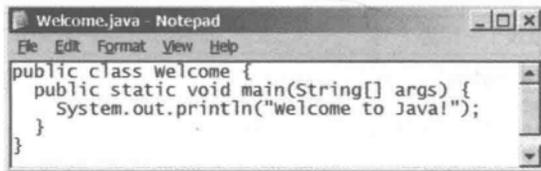


图 1-7 可以使用 Windows 记事本创建 Java 源程序文件

```
javac Welcome.java
```

注意：在编译和运行程序前必须先安装和配置 JDK。补充材料 I.B 介绍如何安装 JDK 8 以及如何设置 Java 程序的编译和运行环境。如果你在编译和运行 Java 程序的过程中遇到问题，请参考补充材料 I.C，这个补充材料还解释了如何使用基本的 DOS 命令，以及如何使用 Windows 记事本来创建和编辑文件。所有补充材料都在本书配套网站 www.cs.armstrong.edu/liang/intro10e/supplement.html 上。

如果没有语法错误，编译器（compiler）就会生成一个扩展名为 .class 的字节码文件。所以，前面的命令会生成一个名为 Welcome.class 的文件，如图 1-8a 所示。Java 语言是高级语言，而 Java 字节码是低级语言。字节码类似于机器指令，但它是体系结构中立的，是在任何带 Java 虚拟机（JVM）的平台上运行的，如图 1-8b 所示。虚拟机不是物理机器，而是一个解释 Java 字节码的程序。这正是 Java 的主要优点之一：Java 字节码可以在不同的硬件平台和操作系统上运行。Java 源代码编译成 Java 字节码，然后 Java 字节码被 JVM 解释执行。你的 Java 代码可能要用到 Java 库中的代码。JVM 将执行你的程序代码以及库中的代码。

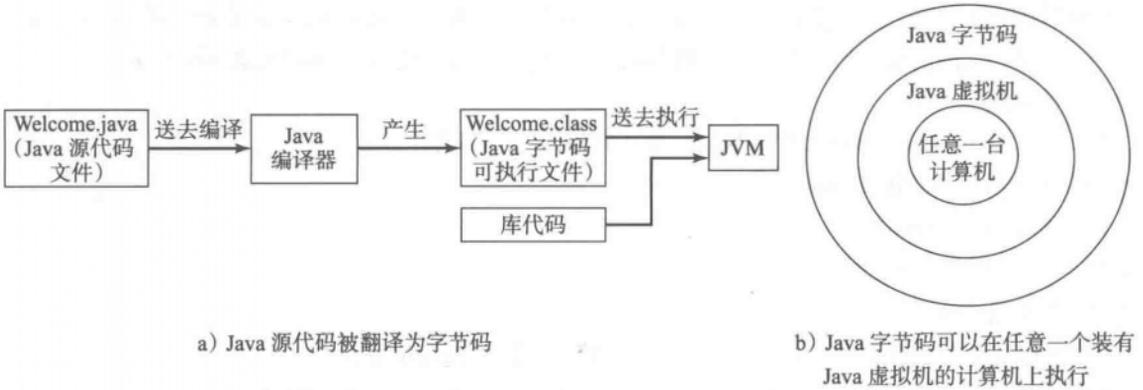


图 1-8

执行 Java 程序就是运行程序的字节码，可以在任何一个装有 JVM 的平台上运行字节码，解释 Java 字节码。解释的过程就是一次将字节码中单独的一步翻译为目标机器语言代码，而不是将整个程序翻译成单独的一块。翻译完一步之后就立即执行这一步。

下述命令用来运行程序清单 1-1 中的字节码：

```
java Welcome
```

图 1-9 显示了用于编译 Welcome.java 的命令 javac。编译器生成 Welcome.class 文件，使用命令 java 执行这个文件。

注意：为了简单性和一致性，除非特别指明，否则所有的源代码和类文件都放在 c:\book 下。

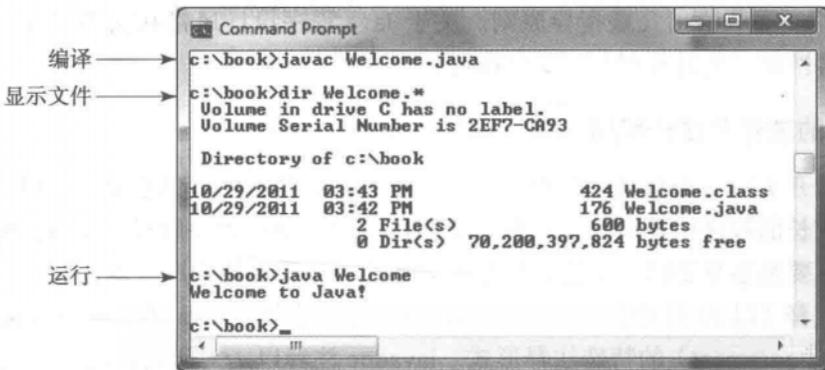


图 1-9 程序清单 1-1 的输出显示消息 “Welcome to Java!”

警告：在命令行执行程序时，不要使用扩展名 .class。使用 java ClassName 来运行程序。如果在命令行使用 java ClassName.class，系统就会尝试读取 ClassName.class.class。

提示：如果要运行一个不存在的类，就会出现 ClassNotFoundException 的错误。如果执行的类文件中没有 main 方法或敲错了 main 方法（例如，将 main 错敲成 Main），则会出现提示 NoSuchMethodError。

注意：在执行一个 Java 程序时，JVM 首先会用一个称为类加载器（class loader）的程序将类的字节码加载到内存中。如果你的程序中使用其他类，类加载程序会在需要它们之前动态地加载它们。当加载该类后，JVM 使用一个称为字节码验证器（bytecode verifier）的程序来检验字节码的合法性，确保字节码不会违反 Java 的安全规范。Java 强制执行严格的安全规范，以确保来自网络的 Java 程序不会篡改和危害你的计算机。

🔑 **教学提示：**教师可能需要学生使用包来组织程序。例如，你可能将本章的所有程序都放在一个名为 `chapter1` 的包里。要得到如何使用包的指南，请参考教材补充材料 I.F。

🔑 复习题

- 1.33 什么是 Java 源程序的文件后缀名，什么是 Java 字节码文件后缀？
- 1.34 Java 编译器的输入和输出是什么？
- 1.35 编译 Java 程序的命令是什么？
- 1.36 运行 Java 程序的命令是什么？
- 1.37 什么是 JVM？
- 1.38 Java 可以运行在任何机器上吗？在一台计算机上运行 Java 需要什么？
- 1.39 如果运行程序的时候出现 `NoClassDefFoundError` 错误，是什么原因导致了这个错误？
- 1.40 如果运行程序的时候出现 `NoSuchMethodError` 错误，是什么原因导致了这个错误？

1.9 程序设计风格和文档

🔑 **要点提示：**良好的程序设计风格和正确的文档使程序更易阅读，并且能帮助程序员避免错误。

程序设计风格 (programming style) 决定程序的外观。如果把整个程序写在一行，它也会被正确地编译和运行，但是这是非常不好的程序设计风格，因为程序的可读性很差。文档 (documentation) 是关于程序的解释性评注和注释的一个结构体。程序设计风格与文档和编写代码的作用一样重要。良好的程序设计风格和适当的文档可以减少出错的机率，并且提高程序的可读性。本节给出几条指导原则。关于 Java 程序设计风格和文档更详细的指南，可以在本书配套网站上的补充材料 I.D 中找到。

1.9.1 正确的注释和注释风格

在程序的开头写一个总结，解释一下这个程序是做什么的、其主要特点以及所用到的独特技术。在较长的程序中还要加上注释，介绍每一个主要步骤并解释每个难以读懂之处。注释写得简明扼要是很重要的，不能让整个程序都充满注释而使程序很难读懂。

除了行注释 (以 `//` 开始) 和块注释 (以 `/*` 开始) 之外，Java 还支持一种称为 Java 文档注释 (javadoc comment) 的特殊注释形式。javadoc 注释以 `/**` 开始，以 `*/` 结尾。它们能使用 JDK 的 javadoc 命令提取成一个 HTML 文件。要获得更多信息，参见配套网站的补充材料 III.Y。

使用 javadoc 注释 (`/**...*/`) 来注释整个类或整个方法。为了将这些注释提取出来放在一个 javadoc HTML 文件中，这些注释必须放在类或者方法头的前面。要注释方法中的某一步骤，使用行注释 (`//`)。

可以从 www.cs.armstrong.edu/liang/javadoc/Exercise1.html 看到一个 javadoc HTML 文件的示例。相应的 Java 代码在 www.cs.armstrong.edu/liang/javadoc/Exercise1.java 中。

1.9.2 正确的缩进和空白

保持一致的缩进风格会使程序更加清晰、易读、易于调试和维护。缩进 (indentation) 用于描述程序中组成部分或语句之间的结构性关系。即使将程序的所有语句都写在一行中，Java 也可以读懂这样的程序，但是正确的对齐能够使人们更易读懂和维护代码。在嵌套结构

中，每个内层的组成部分或语句应该比外层缩进两格。

二元操作符的两边应该各加一个空格，如下面语句所示：

```
System.out.println(3+4*4);           不好的风格
System.out.println(3 + 4 * 4);       良好的风格
```

1.9.3 块的风格

块是由花括号围起来的一组语句。块的写法有两种常用方式：次行（next-line）风格和行尾（end-of-line）风格，如下所示。

```
public class Test
{
    public static void main(String[] args)
    {
        System.out.println("Block Styles");
    }
}
```

次行风格

```
public class Test {
    public static void main(String[] args) {
        System.out.println("Block Styles");
    }
}
```

行尾风格

次行风格将括号垂直对齐，因而使程序容易阅读，而行尾风格更节省空间，并有助于避免犯一些细小的程序设计错误。这两种风格都是可以采纳的。选择哪一种完全依赖于个人或组织的偏好。应该统一采用一种风格，建议不要将这两种风格混合使用。本书与 Java API 源代码保持一致，都采用行尾风格。

☛ 复习题

1.41 使用行尾括号风格，将下面的程序根据程序设计风格 and 文档指南进行重新格式化。

```
public class Test
{
    // Main method
    public static void main(String[] args) {
        /** Display output */
        System.out.println("Welcome to Java");
    }
}
```

1.10 程序设计错误

🔑 要点提示：程序设计错误可以分为三类：语法错误、运行时错误和逻辑错误。

1.10.1 语法错误

在编译过程中出现的错误称为语法错误（syntax error）或编译错误（compile error）。语法错误是由创建代码时的错误引起的，例如：拼错关键字，忽略了一些必要的标点符号，或者左花括号没有对应的右花括号。这些错误通常很容易检测到，因为编译器会告诉你这些错误在哪儿，以及是什么原因造成的。例如：编译下面程序清单 1-4 中的程序会出现语法错误，如图 1-10 所示。

程序清单 1-4 ShowSyntaxErrors.java

```
1 public class ShowSyntaxErrors {
2     public static main(String[] args) {
3         System.out.println("Welcome to Java");
4     }
5 }
```

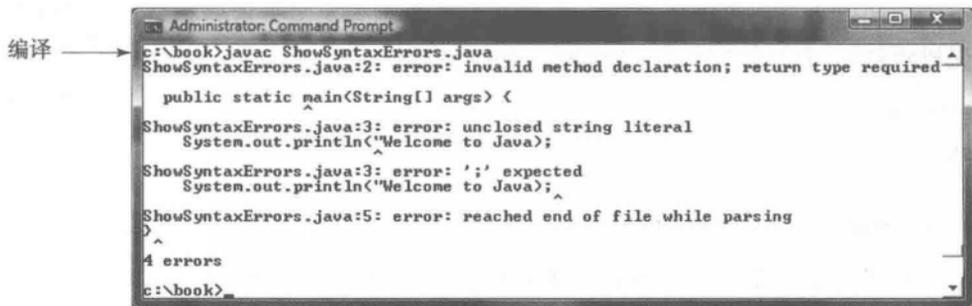


图 1-10 编译器报告语法错误

四个错误被报告，但实际上程序有两个错误：

- 第 2 行 main 方法前遗漏关键字 void。
- 第 3 行的字符串 Welcome to Java 应该加上引号。

由于一个错误常常会显示很多行的编译错误，因此，从最上面的行开始向下纠正错误是一个很好的习惯。解决程序前面出现的错误，可能就改正了程序后面出现的其他错误。

提示：如果你不知道如何纠正错误，将你的程序一个字符一个字符地仔细对照教材中的类似示例。在课程的前面几周，你可能要花许多时间纠正语法错误，但是很快你将熟悉 Java 语法，并快速纠正语法错误。

1.10.2 运行时错误

运行时错误 (runtime error) 是引起程序非正常中断的错误。运行应用程序时，当环境检测到一个不可能执行的操作时，就会出现运行时错误。输入错误是典型的运行时错误。当程序等待用户输入一个值，而用户输入了一个程序不能处理的值时，就会发生输入错误。例如：如果程序希望读入的是一个数值，而用户输入的却是一个字符串，就会导致程序出现数据类型错误。

另一个常见的运行时错误是 0 作除数。当整数除法中除数为 0 时可能引发这种情况。例如：下面程序清单 1-5 中的程序将会导致运行时错误，如图 1-11 所示。

程序清单 1-5 ShowRuntimeErrors.java

```
1 public class ShowRuntimeErrors {
2     public static void main(String[] args) {
3         System.out.println(1 / 0);
4     }
5 }
```

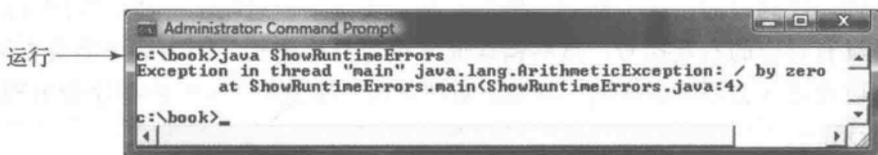


图 1-11 运行时错误导致程序非正常中断

1.10.3 逻辑错误

当程序没有按预期的方式执行时就会发生逻辑错误 (logic error)。这种错误发生的原因

有很多种。例如，假设你编写如程序清单 1-6 中的程序，将摄氏 35 度转换为华氏度：

程序清单 1-6 ShowLogicErrors.java

```
1 public class ShowLogicErrors {
2     public static void main(String[] args) {
3         System.out.println("Celsius 35 is Fahrenheit degree ");
4         System.out.println((9 / 5) * 35 + 32);
5     }
6 }
```

```
Celsius 35 is Fahrenheit degree
67
```

你将得到结果华氏 67 度，而这是错误的，结果应该是 95.0。Java 中，整数相除是返回除法的整数部分，即小数部分被截掉，因此 Java 中 9/5 的结果是 1。要得到正确的结果，需要使用 9.0/5，这样得到结果 1.8。

通常情况下，因为编译器可以明确指出错误的位置以及出错的原因，所以语法错误是很容易发现和纠正的。运行时错误也不难找，因为在程序异常中止时，错误的原因和位置都会显示在控制台上。然而，查找逻辑错误就很富有挑战性。在下面的章节中，我们将学习跟踪程序以及找到逻辑错误的技巧。

1.10.4 常见错误

对于编程新手来说，遗漏右括号、遗漏分号、遗漏字符串的引号、命名拼写错误，都是常见的错误。

常见错误 1：遗漏右括号

括号用来标识程序中的块。每个左括号必须有一个右括号匹配。常见的错误是遗漏右括号。为避免这个错误，任何时候输入左括号的时候就输入右括号，如下面的例子所示：

```
public class Welcome {
```

```
} ←—— 立刻输入右括号匹配左括号
```

如果使用 NetBeans 和 Eclipse 这样的 IDE，IDE 将自动为每个输入的左括号插入一个右括号。

常见错误 2：遗漏分号

每个语句都以一个语句结束符（；）结束。通常，编程入门者会忘了在一个块的最后一行语句后加上语句结束符，如下面例子所示：

```
public static void main(String[] args) {
    System.out.println("Programming is fun!");
    System.out.println("Fundamentals First");
    System.out.println("Problem Driven")
}
```

↑
遗漏一个分号

常见错误 3：遗漏引号

字符串必须放在引号中。通常，编程入门者会忘记在字符串结尾处加上一个引号，如下面例子所示：

```
System.out.println("Problem Driven");
```

↑
遗漏一个引号

如果使用 NetBeans 和 Eclipse 这样的 IDE，IDE 将自动为每个输入的左引号插入一个右引号。

常见错误 4：命名拼写错误

Java 是大小写敏感的。编程入门者常将名称拼写错误。例如，下面的代码中 main 错误拼写成 Main，String 错误拼写成 string。

```
1 public class Test {
2     public static void Main(string[] args) {
3         System.out.println((10.5 + 2 * 3) / (45 - 3.5));
4     }
5 }
```

复习题

- 1.42 什么是语法错误（编译错误）、运行时错误以及逻辑错误？
- 1.43 给出语法错误、运行时错误以及逻辑错误的示例。
- 1.44 如果忘记为字符串加引号了，将产生哪类错误？
- 1.45 如果程序需要读取整数，而用户输入了字符串，运行该程序的时候将产生什么错误？这是哪类错误？
- 1.46 假设编写一个计算矩形周长的程序，但是错误地写成了计算矩形面积的程序。这属于哪类错误？
- 1.47 指出和修改下面代码中的错误：

```
1 public class Welcome {
2     public void Main(String[] args) {
3         System.out.println('Welcome to Java!');
4     }
5 }
```

1.11 使用 NetBeans 开发 Java 程序

要点提示：可以使用 NetBeans 来编辑、编译、运行和调试 Java 程序。

NetBeans 和 Eclipse 是两个开发 Java 程序的免费的流行集成开发环境。如果按照简单的指南学习，可以很快掌握。我们建议你采用其中之一来开发 Java 程序。本节对于初学者给出基本的指南，在 NetBeans 环境中创建一个工程，创建类，以及编译和运行类。Eclipse 的使用将在下节中介绍。参考补充材料 II.B 以得到如何下载以及安装最新版本 NetBeans 的指南。

1.11.1 创建 Java 工程

创建 Java 程序前，首先需要创建一个工程。工程类似于一个文件夹，用于包含 Java 程序以及所有的支持文件。你只需要创建工程一次。这里是创建 Java 工程的步骤：

- 1) 选择 File → New Project 来显示 New Project 对话框，如图 1-12 所示。
- 2) 在 Categories 部分选择 Java，Projects 部分选择 Java Application，然后单击 Next 来显示 New Java Application 对话框，如图 1-13 所示。
- 3) 在 Project Name 域中输入 demo，在 Project Location 域中输入 c:\michael。去掉 Use Dedicated Folder for Storing Libraries 的勾选，并且去掉 Create Main Class 的勾选。
- 4) 单击 Finish 来创建工程，如图 1-14 所示。

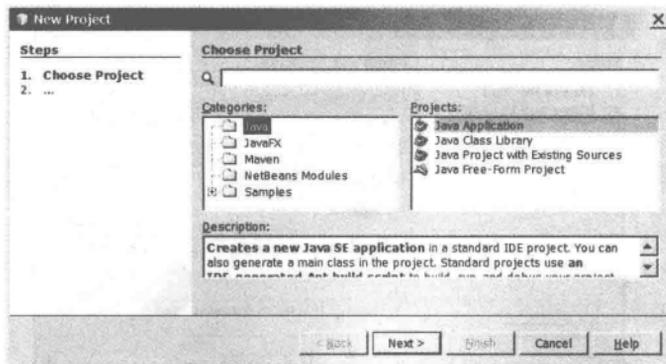


图 1-12 使用 New Project 对话框来创建一个新的工程，并且指定工程类别

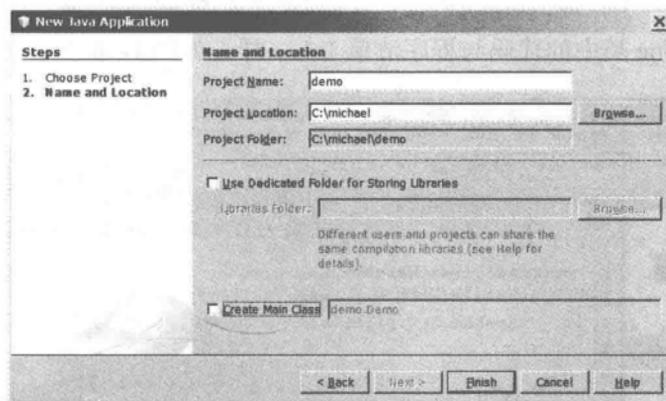


图 1-13 New Java Application 对话框用于确定工程名称和位置

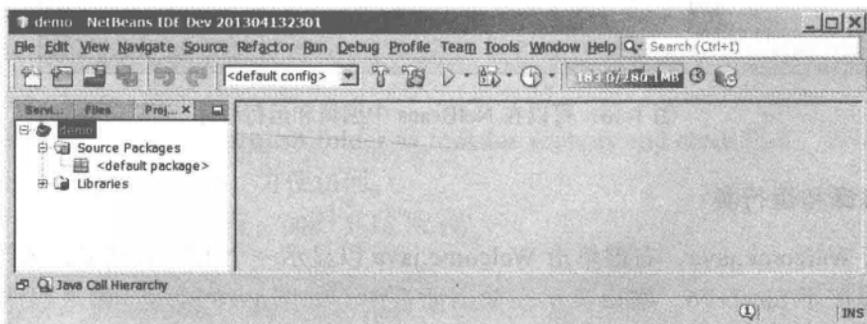


图 1-14 创建一个新的命名为 demo 的 Java 工程

1.11.2 创建 Java 类

工程创建后，可以采用以下步骤在工程中创建 Java 程序：

1) 右键单击工程面板的 demo 节点，显示一个上下文菜单。选择 New → Java Class 来显示 New Java Class 对话框，如图 1-15 所示。

2) 在 Class Name 域输入 Welcome，在 Location 域中选择 Source Packages。Package 域保留为空白，这样将在默认包中创建一个类。

3) 单击 Finish 来创建 Welcome 类。源代码文件 Welcome.java 放置在 <default package> 节点下面。

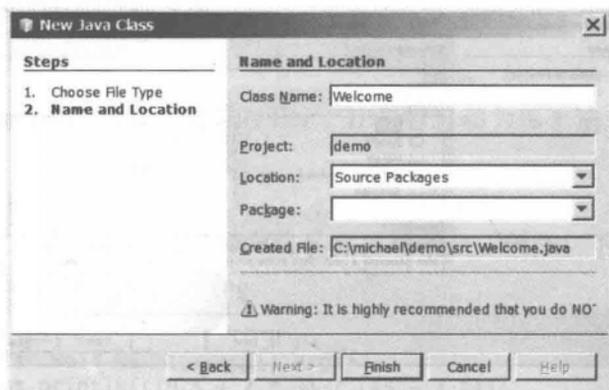


图 1-15 使用 New Java Class 对话框来创建一个新的 Java 类

4) 修改 Welcome 类中的代码与程序清单 1-1 一样, 如图 1-16 所示。

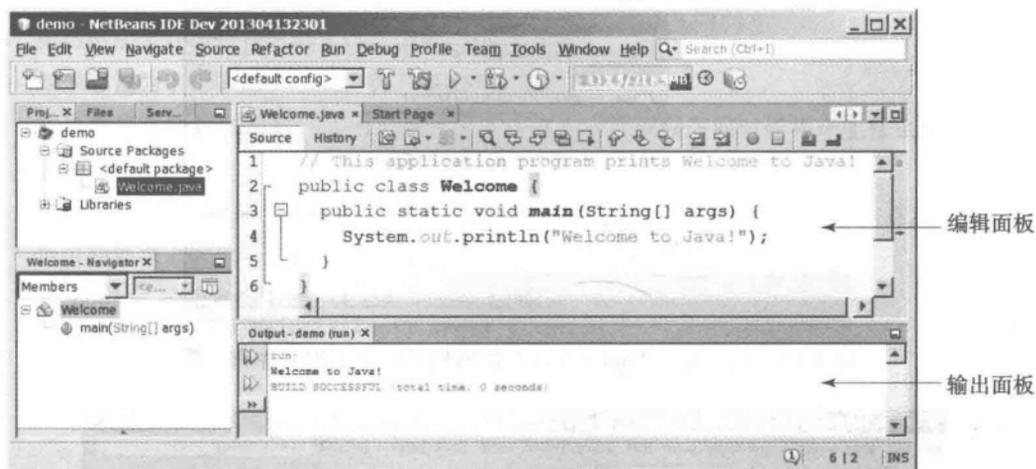


图 1-16 可以在 NetBeans 中编辑和运行程序

1.11.3 编译和运行类

要运行 Welcome.java, 右键单击 Welcome.java 以显示一个上下文菜单, 选择 Run File, 或者简单地按下 Shift+F6。输出显示在输出面板中, 如图 1-16 所示。如果程序被修改了, Run File 命令自动编译程序。

1.12 使用 Eclipse 开发 Java 程序

要点提示: 可以使用 Eclipse 来编辑、编译、运行和调试 Java 程序。

前一节介绍使用 NetBeans 开发 Java 程序, 也可以采用 Eclipse 开发 Java 程序。本节给出基本教程, 指导初学者在 Eclipse 下面创建工程, 创建类, 以及编译/运行类。参考补充材料 II.D 以得到如何下载以及安装最新版本 Eclipse 的指南。

1.12.1 创建 Java 工程

使用 Eclipse 创建 Java 程序前, 首先需要创建一个工程包含所有的文件。下面是 Eclipse

中创建 Java 工程的步骤：

1) 选择 File → New → Java Project 来显示 New Java Project 向导，如图 1-17 所示。

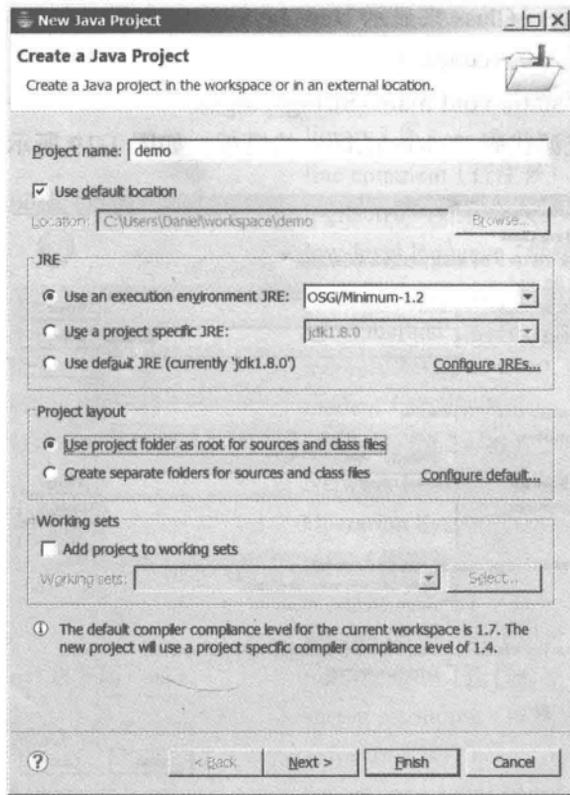


图 1-17 New Java Project 对话框用于确定工程名称和属性

2) 在 Project name 域中输入 demo，Location 域自动设置为默认。可以为你的工程自定义位置。

3) 确保选择了选项 Use project folder as root for sources and class files，从而 .java 文件和 .class 文件在同一个目录下，方便访问。

4) 单击 Finish 来创建工程，如图 1-18 所示。

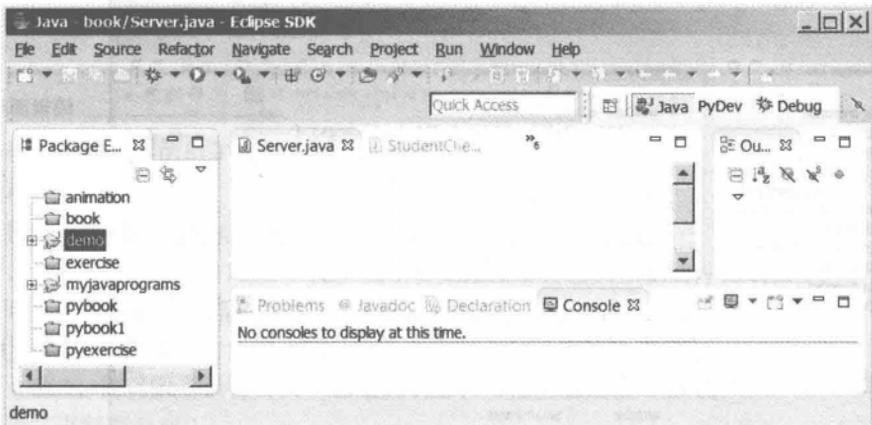


图 1-18 创建名为 demo 的新的 Java 工程

1.12.2 创建 Java 类

工程创建后，可以采用以下步骤在工程中创建 Java 程序：

- 1) 选择 File → New → Class 来显示 New Java Class 向导。
- 2) 在 Name 域中输入 Welcome。
- 3) 勾选选项 public static void main (String[] args)。
- 4) 单击 Finish 生成源代码 Welcome.java 的模板，如图 1-19 所示。

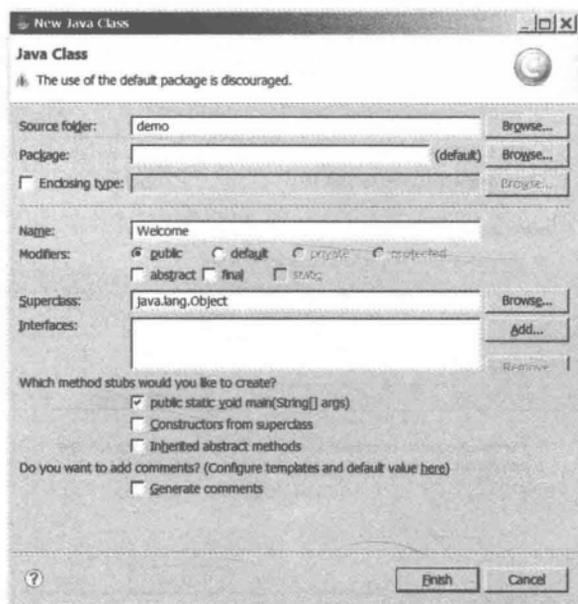


图 1-19 使用 New Java Class 对话框来创建一个新的 Java 类

1.12.3 编译和运行类

要运行程序，右键单击工程中的类，显示一个上下文菜单。在上下文菜单中选择 Run → Java Application 以运行类。输出显示在控制台面板中，如图 1-20 所示。

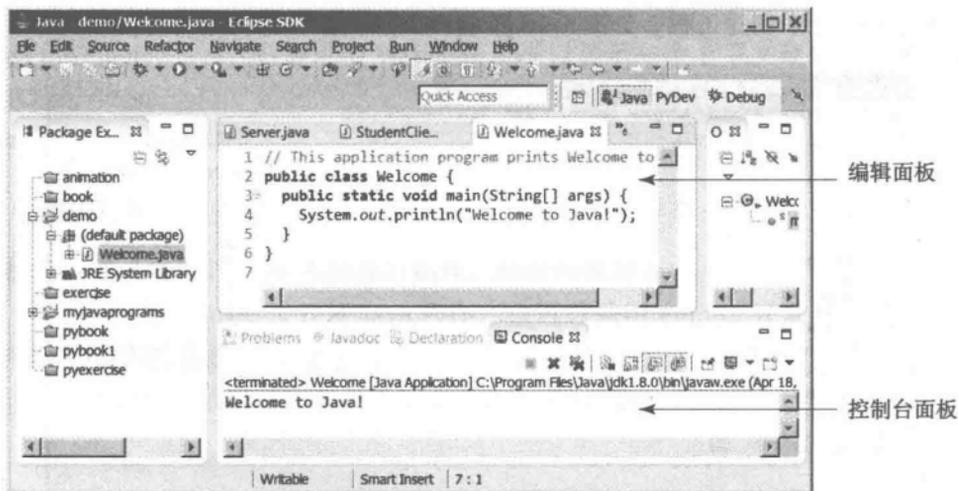


图 1-20 在 Eclipse 中编辑和运行程序

关键术语

Application Program Interface (API) (应用程序接口)	Java language specification (Java 语言规范)
assembler (汇编器)	Java Virtual Machine (JVM)(Java 虚拟机)
assembly language (汇编语言)	javac command (javac 命令)
bit (比特)	keyword or reserved word (关键字或保留字)
block (块)	library (库)
block comment (块注释)	line comment (行注释)
bus (总线)	logic error (逻辑错误)
byte (字节)	low-level language (低级语言)
bytecode (字节码)	machine language (机器语言)
bytecode verifier (字节码验证器)	main method (main 方法)
cable modem (电缆调制解调器)	memory (内存)
Central Processing Unit (CPU)(中央处理器)	modem (调制解调器)
class loader (类加载器)	motherboard (主板)
comment (注释)	Network Interface Card (NIC)(网络接口卡)
compiler (编译器)	Operation System (OS)(操作系统)
console (控制台)	pixel (像素)
dot pitch (点距)	program (程序)
DSL (Digital Subscriber Line)(数字用户线)	programming (程序设计)
encoding scheme (编码规范)	runtime error (运行时错误)
hardware (硬件)	screen resolution (屏幕分辨率)
high-level language (高级语言)	software (软件)
Integrated Development Environment (IDE) (集成开发环境)	source code (源代码)
interpreter (解释器)	source program (源程序)
java command (java 命令)	statement (语句)
Java Development Toolkit (JDK)(Java 开发工具包)	statement terminator (语句结束符)
	storage device (存储设备)
	syntax error (语法错误)

 **注意：**上面的术语都是本章所定义的。补充材料 I.A 按照章节顺序列出了本书所有的关键术语及其说明。

本章小结

1. 计算机是存储和处理数据的电子设备。
2. 计算机包括硬件和软件两部分。
3. 硬件是计算机中可以触摸到的物理部分。
4. 计算机程序，也就是通常所说的软件，是一些不可见的指令，它们控制硬件完成任务。
5. 计算机程序设计就是编写让计算机执行的指令（即代码）。
6. 中央处理器（CPU）是计算机的大脑。它从内存获取指令并且执行这些指令。
7. 计算机使用 0 或 1，因为数字设备有两个稳定的状态，习惯上就是指 0 和 1。
8. 一个比特是指二进制数 0 或 1。
9. 一个字节是指 8 比特的序列。

10. 千字节大约是 1000 字节, 兆字节大约是 100 万字节, 千兆字节大约是 10 亿字节, 万亿字节大约是 1 万亿字节。
11. 内存存储 CPU 要执行的数据和程序指令。
12. 内存单元是字节的有序序列。
13. 内存是不能长久保存数据的, 因为断电时信息就会丢失。
14. 程序和数据永久地保存在存储设备里, 当计算机确实需要使用它们时被移入内存。
15. 机器语言是一套内嵌在每台计算机的原始指令集。
16. 汇编语言是一种低级程序设计语言, 它用助记符表示每一条机器语言的指令。
17. 高级语言类似英语, 易于学习和编写程序。
18. 用高级语言编写的程序称为源程序。
19. 编译器是将源程序翻译成机器语言程序的软件。
20. 操作系统 (OS) 是管理和控制计算机活动的程序。
21. Java 是平台无关的, 这意味着只需编写一次程序, 可以在任何计算机上运行。
22. Java 程序可以内嵌在 HTML 网页内, 通过 Web 浏览器下载, 给 Web 客户带来生动的动画和灵活的交互性。
23. Java 源程序文件名必须和程序中的公共类名一致, 并且以扩展名 .java 结束。
24. 每个类都被编译成一个独立的字节码文件, 该文件名与类名相同, 扩展名为 .class。
25. 使用 javac 命令可以从命令行编译 Java 源代码文件。
26. 使用 java 命令可以从命令行运行 Java 类。
27. 每个 Java 程序都是一套类的定义集合。关键字 class 引入类的定义, 类的内容包含在块内。
28. 一个块以左花括号 { } 开始, 以右花括号 } 结束。
29. 方法包含在类中。每个可执行的 Java 程序必须有一个 main 方法。main 方法是程序开始执行的入口。
30. Java 中的每条语句都是以分号 (;) 结束的, 也称该符号为语句结束符。
31. 保留字或者称关键字, 对编译器而言都有特殊含义, 在程序中不能用于其他目的。
32. 在 Java 中, 在单行上用两个斜杠 (//) 引导注释, 称为行注释; 在一行或多行用 /* 和 */ 包含注释, 称为块注释或者段注释。编译器会忽略注释。
33. Java 源程序是区分大小写的。
34. 编程错误可以分为三类: 语法错误、运行时错误和逻辑错误。编译器报告的错误称为语法错误或者编译错误。运行时错误指引起程序非正常结束的错误。当一个程序没有按照预期的方式执行时, 产生逻辑错误。

测试题

在线回答本章测试题, 地址为 www.cs.armstrong.edu/liang/intro10e/quiz.html。

编程练习题

 **注意:** 偶数题号的答案在配套网站上。所有编程练习题的答案在教师资源网站中。额外的编程练习题以及答案提供给教师。题目的难度等级分为容易 (没有星号)、适中 (*)、难 (**) 以及具有挑战性 (***)。

- 1.1 (显示三条消息) 编写程序, 显示 Welcome to Java、Welcome to Computer Science 和 Programming is fun。
- 1.2 (显示五条消息) 编写程序, 显示 Welcome to Java 五次。
- *1.3 (显示图案) 编写一个程序, 显示下面的图案:

```

      J   A   V   V   A
      J   A A   V   V   A A
    J   J   A A A A   V V   A A A A
    J J   A   A   V   A   A

```

1.4 (打印表格) 编写程序, 显示以下表格:

a	a ²	a ³
1	1	1
2	4	8
3	9	27
4	16	64

1.5 (计算表达式) 编写程序, 显示以下公式的结果。

$$\frac{9.5 \times 4.5 - 2.5 \times 3}{45.5 - 3.5}$$

1.6 (数列求和) 编写程序, 显示 $1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9$ 的结果。

1.7 (近似求 π) 可以使用以下公式计算 π :

$$\pi = 4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \dots \right)$$

编写程序, 显示 $4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} \right)$ 和 $4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} \right)$ 的结果。在程序中用 1.0 代替 1。

1.8 (圆的面积和周长) 编写程序, 使用以下公式计算并显示半径为 5.5 的圆的面积和周长。

$$\text{周长} = 2 \times \text{半径} \times \pi$$

$$\text{面积} = \text{半径} \times \text{半径} \times \pi$$

1.9 (矩形的面积和周长) 编写程序, 使用以下公式计算并显示宽度为 4.5、高度为 7.9 的矩形的面积和周长。

$$\text{面积} = \text{宽} \times \text{高}$$

1.10 (以英里计的平均速度) 假设一个跑步者 45 分钟 30 秒内跑了 14 公里。编写一个程序显示以每小时多少英里为单位的平均速度值。(注意, 1 英里等于 1.6 公里。)

*1.11 (人口估算) 美国人口调查局基于以下假设进行人口估算:

- 每 7 秒有一个人诞生
- 每 13 秒有一个人死亡
- 每 45 秒有一个移民迁入

编写一个程序, 显示未来 5 年的每年的人口数。假设当前的人口是 312 032 486, 每年有 365 天。提示: Java 中, 两个整数相除, 结果还是整数, 小数部分被去掉。例如, $5/4$ 等于 1 (而不是 1.25), $10/4$ 等于 2 (而不是 2.5)。如果想得到有小数部分的精确结果, 进行除法运算的两个值之一必须是一个具有小数点的数值。例如, $5.0/4$ 等于 1.25, $10/4.0$ 等于 2.5。

1.12 (以公里计的平均速度) 假设一个跑步者 1 小时 40 分钟 35 秒内跑了 24 英里。编写一个程序显示以每小时多少公里为单位的平均速度值。(注意, 1 英里等于 1.6 公里。)

*1.13 (代数: 求解 2×2 线性方程) 可以使用 Cramer 规则解下面的 2×2 线性方程组:

$$\begin{cases} ax + by = e \\ cx + dy = f \end{cases} \quad x = \frac{ed - bf}{ad - bc} \quad y = \frac{af - ec}{ad - bc}$$

编写程序, 求解以下方程组并显示 x 和 y 的值。

$$3.4x + 50.2y = 44.5$$

$$2.1x + 0.55y = 5.9$$

基本程序设计

教学目标

- 编写 Java 程序完成简单的计算 (2.2 节)。
- 使用 Scanner 类从控制台获取输入 (2.3 节)。
- 使用标识符命名变量、常量、方法和类 (2.4 节)。
- 使用变量存储数据 (2.5 ~ 2.6 节)。
- 用赋值语句和赋值表达式编写程序 (2.6 节)。
- 使用常量存储永久数据 (2.7 节)。
- 按照命名习惯命名类, 方法, 变量和常量 (2.8 节)。
- 探索 Java 的基本数值类型: byte、short、int、long、float 和 double (2.9.1 节)。
- 从键盘读入一个 byte、short、int、long、float 或者 double 类型的值 (2.9.2 节)。
- 使用操作符 +、-、*、/ 和 % 来执行操作 (2.9.3 节)。
- 使用 Math.pow (a, b) 进行幂运算 (2.9.4 节)。
- 编写整数字面值、浮点数字面值, 以及科学表达式的字面值 (2.10 节)。
- 编写和计算数值表达式 (2.11 节)。
- 使用 System.currentTimeMillis() 获得当前系统时间 (2.12 节)。
- 使用增量赋值操作符 (2.13 节)。
- 区分后置递增和前置递增, 以及后置递减和前置递减 (2.14 节)。
- 将一种类型的值强制转换为另一种类型 (2.15 节)。
- 描述软件开发过程, 并将其应用于开发贷款支付额程序 (2.16 节)。
- 编写程序, 计算整钱兑零 (2.17 节)。
- 避免基础编程中常见错误和陷阱 (2.18 节)。

2.1 引言

 **要点提示:** 本章的重点是学习基础程序设计技术, 以进行问题求解。

在第 1 章里, 我们学习了如何创建、编译和运行非常基础的 Java 程序。现在, 将学习如何编程解决实际问题。通过这些问题, 你将学到如何利用基本数据类型、变量、常量、操作符、表达式以及输入/输出来进行基本的程序设计。

比如, 假设你需要计算一个学生贷款。给定贷款额度、贷款时间以及年利率, 你可以通过编写程序来计算每月支付以及整个支付额度吗? 本章将演示如何编写这样的程序。用这样的方法, 你将学习分析问题, 设计一个解决方案以及通过创建一个程序来实现这个解决方案的基本步骤。

2.2 编写简单的程序

 **要点提示:** 编写程序涉及如何设计解决问题的策略, 以及如何应用编程语言实现这个策略。

首先,我们来看一个计算圆面积的简单问题。该如何编写程序解决这个问题呢?

编写程序涉及如何设计算法以及如何将算法翻译成程序指令,即代码。算法描述的是:如果要解决问题,所需要执行的动作以及这些动作执行的顺序。算法可以帮助程序员在使用程序设计语言编写程序之前做一个规划。算法可以用自然语言或者伪代码(即自然语言和程序设计代码混在一起使用)描述。这个程序的算法描述如下:

1) 读入半径。

2) 利用下面的公式计算面积:

$$\text{面积} = \text{半径} \times \text{半径} \times \pi$$

3) 显示面积。

 **提示:** 在编写代码之前,以一种算法的形式来勾勒你的程序(或者它潜在的问题),是一个很好的做法。

当你编码,也就是当你编写一个程序时,你将一个算法翻译成程序。你已经知道每个 Java 程序都是以一个类的声明开始,在声明里类名紧跟在关键字 `class` 后面。假设你选择 `ComputeArea` 作为这个类的类名。这个程序的框架就如下所示:

```
public class ComputeArea {  
    // Details to be given later  
}
```

如你所知,每一个 Java 应用程序都必须有一个 `main` 方法,程序从该方法处开始执行。所以该程序被扩展为如下所示:

```
public class ComputeArea {  
    public static void main(String[] args) {  
        // Step 1: Read in radius  
  
        // Step 2: Compute area  
        // Step 3: Display the area  
    }  
}
```

这个程序需要读取用户从键盘输入的半径。这就产生了两个重要问题:

- 读取半径。
- 将半径存储在程序中。

我们先来解决第二个问题。为了存储半径,在程序中需要声明一个称作变量的符号。变量代表了存储在计算机内存中的一个值。

变量名应该尽量选择描述性的名字 (*descriptive name*),而不是用 `x` 和 `y` 这样的名字:在这里的例子中,用 `radius` 表示半径、用 `area` 表示面积。为了让编译器知道 `radius` 和 `area` 是什么,需要指明它们的数据类型,即存储在变量中的数据的类型,是整数、实数,或者其他。这称为声明变量。Java 提供简单数据类型来表示整数、实数、字符以及布尔类型。这些类型称为原始数据类型或基本类型。

实数(即带小数点的数字)在计算机中使用一种浮点的方法来表示。因此,实数也称为浮点数。Java 中,可以使用关键字 `double` 来声明一个浮点变量。将 `radius` 和 `area` 声明为 `double`。程序可被扩展为如下所示:

```
public class ComputeArea {  
    public static void main(String[] args) {  
        double radius;  
        double area;
```

```

// Step 1: Read in radius
// Step 2: Compute area
// Step 3: Display the area
}
}

```

程序将 `radius` 和 `area` 声明为变量。保留字 `double` 表明 `radius` 和 `area` 是以浮点数形式存储在计算机中的。

第一步是提示用户指定圆的半径。稍后我们会学习如何提示用户获得信息。现在为了学习变量是如何工作的，可以给程序中的 `radius` 赋一个固定值；之后，修改程序，以提示用户输入这个值。

第二步是计算 `area`，这是通过将表达式 `radius*radius*3.14159` 的值赋给 `area` 来实现的。在最后一步里，使用 `System.out.println` 方法在控制台上显示 `area` 的值。完整的程序如程序清单 2-1 所示。该程序的示例运行如图 2-1 所示。

程序清单 2-1 ComputeArea.java

```

1 public class ComputeArea {
2     public static void main(String[] args) {
3         double radius; // Declare radius
4         double area; // Declare area
5
6         // Assign a radius
7         radius = 20; // radius is now 20
8
9         // Compute area
10        area = radius * radius * 3.14159;
11
12        // Display results
13        System.out.println("The area for the circle of radius " +
14            radius + " is " + area);
15    }
16 }

```

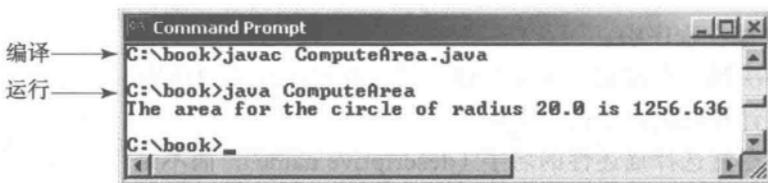


图 2-1 程序显示圆的面积

像 `radius` 和 `area` 这样的变量对应于它们在内存的位置。每个变量都有名字、类型、大小和值。第 3 行声明 `radius` 可以存储一个 `double` 型的数值。直到给它赋一个数值时，该变量才被定义。第 7 行将 `radius` 赋值为 20。类似地，第 4 行声明变量 `area`，第 10 行将 10 赋值给 `area`。下面的表格显示的是随着程序的执行，`area` 和 `radius` 在内存中的值。该表中的每一行显示的是程序中对应的每行语句执行之后变量的值。这种审查程序如何工作的方法称为跟踪一个程序。跟踪程序有助于理解一个程序是如何工作的，而且它也是一个查找程序错误的非常有用的工具。

行号	半径	面积
3	无值	
4		无值
7	20	
10		1256.636

加号 (+) 有两种意义：一种用途是做加法，另一种用途是做字符串的连接（合并）。第 13 ~ 14 行中的加号 (+) 称为字符串连接符。它把两个字符串合并为一个。如果一个字符串和一个数值连接，数值将转化为字符串然后再和另外一个字符串连接。所以，第 13 ~ 14 行的加号 (+) 会将几个字符串连成一个更长的字符串，然后将它在输出结果中显示出来。关于字符串以及字符串连接的更多内容将在第 4 章中讨论。

 **警告：**在源代码中，字符串常量不能跨行。因此，下面的语句会造成编译错误：

```
System.out.println("Introduction to Java Programming,
by Y. Daniel Liang");
```

为了改正错误，将该字符串分成几个单独的子串，然后再用连接符 (+) 将它们组合起来：

```
System.out.println("Introduction to Java Programming, " +
"by Y. Daniel Liang");
```

复习题

2.1 指出并修改以下代码中的错误：

```
1 public class Test {
2     public void main(string[] args) {
3         double i = 50.0;
4         double k = i + 50.0;
5         double j = k + 1;
6
7         System.out.println("j is " + j + " and
8             k is " + k);
9     }
10 }
```

2.3 从控制台读取输入

 **要点提示：**从控制台读取输入，使得程序可以从用户那里获得输入。

在程序清单 2-1 中，源代码中的半径是固定的。为了能使用不同的半径，必须修改源代码然后重新编译它。很显然，这是非常不方便的。可以使用 Scanner 类从控制台输入。

Java 使用 System.out 来表示标准输出设备，而用 System.in 来表示标准输入设备。默认情况下，输出设备是显示器，而输入设备是键盘。为了完成控制台输出，只需使用 println 方法就可以在控制台上显示基本值或字符串。Java 并不直接支持控制台输入，但是可以使用 Scanner 类创建它的对象，以读取来自 System.in 的输入，如下所示：

```
Scanner input = new Scanner(System.in);
```

语法 new Scanner (System.in) 表明创建了一个 Scanner 类型的对象。语法 Scanner input 声明 input 是一个 Scanner 类型的变量。整行的 Scanner input=new Scanner (System.in) 表明创建了一个 Scanner 对象，并且将它的引用值赋值给变量 input。对象可以调用它自己的方法。调用对象的方法就是让这个对象完成某个任务。可以调用

nextDouble() 方法来读取一个 double 值，如下所示：

```
double radius = input.nextDouble();
```

该语句从键盘读入一个数值，并且将该数值赋给 radius。

程序清单 2-2 重写程序清单 2-1，提示用户输入一个半径。

程序清单 2-2 ComputeAreaWithConsoleInput.java

```
1 import java.util.Scanner; // Scanner is in the java.util package
2
3 public class ComputeAreaWithConsoleInput {
4     public static void main(String[] args) {
5         // Create a Scanner object
6         Scanner input = new Scanner(System.in);
7
8         // Prompt the user to enter a radius
9         System.out.print("Enter a number for radius: ");
10        double radius = input.nextDouble();
11
12        // Compute area
13        double area = radius * radius * 3.14159;
14
15        // Display results
16        System.out.println("The area for the circle of radius " +
17            radius + " is " + area);
18    }
19 }
```

```
Enter a number for radius: 2.5 
The area for the circle of radius 2.5 is 19.6349375
```

```
Enter a number for radius: 23 
The area for the circle of radius 23.0 is 1661.90111
```

第 9 行的语句在控制台显示一个字符串 "Enter a number for radius:"，这称为一个提示，因为它指导用户键入输入。你的程序应该在希望得到键盘输入的时候，告知用户输入什么。

第 9 行的 print 方法

```
System.out.print("Enter a number for radius: ");
```

和 println 方法很类似，两者的不同之处在于：当显示完字符串之后，println 会将光标移到下一行，而 print 不会将光标移到下一行。

第 6 行创建一个 Scanner 对象。第 10 行的语句从键盘读入一个输入。

```
double radius = input.nextDouble();
```

在用户键入一个数值然后单击回车键之后，该数值就被读入并赋值给 radius。

更多关于对象的细节将在第 9 章中介绍。目前，只要知道这是如何从控制台获取输入的方式就可以了。

Scanner 类在包 java.util 里。它在第 1 行被导入。import 语句有两种类型：明确导入 (specific import) 和通配符导入 (wildcard import)。明确导入是在 import 语句中指定单个的类。例如，下面的语句就是从包 java.util 中导入 Scanner。

```
import java.util.Scanner;
```

通配符导入是指通过使用星号作为通配符，导入一个包中所有的类。例如，下面的语句导入包 `java.util` 中所有的类。

```
import java.util.*;
```

除非要在程序中使用某个类，否则关于被导入包中的这些类的信息在编译时或运行时是不被读入的。导入语句只是告诉编译器在什么地方能找到这些类。声明明确导入和声明通配符导入在性能上是没有什么差别的。

程序清单 2-3 给出从键盘读取多个输入的例子。这个例子读取三个数值，然后显示它们的平均值。

程序清单 2-3 ComputeAverage.java

```
1 import java.util.Scanner; // Scanner is in the java.util package
2
3 public class ComputeAverage {
4     public static void main(String[] args) {
5         // Create a Scanner object
6         Scanner input = new Scanner(System.in);
7
8         // Prompt the user to enter three numbers
9         System.out.print("Enter three numbers: ");
10        double number1 = input.nextDouble();
11        double number2 = input.nextDouble();
12        double number3 = input.nextDouble();
13
14        // Compute average
15        double average = (number1 + number2 + number3) / 3;
16
17        // Display results
18        System.out.println("The average of " + number1 + " " + number2
19            + " " + number3 + " is " + average);
20    }
21 }
```

```
Enter three numbers: 1 2 3 ↵
The average of 1.0 2.0 3.0 is 2.0
```

```
Enter three numbers: 10.5 ↵
11 ↵
11.5 ↵
The average of 10.5 11.0 11.5 is 11.0
```

导入 `Scanner` 类的代码（第 1 行）以及创建 `Scanner` 对象的代码（第 6 行）都是和前一个例子一样的，而且在你将编写的所有新程序中，这两行也都是一样的。

第 9 行提示用户输入三个数值。这些数值在第 10 ~ 12 行被读取。可以输入三个用空格符分隔开的数值，然后按回车键，或者每输入一个数值之后就按一次回车键，如该程序的示例运行所示。

如果输入了一个非数值的值，一个运行时错误将产生。在第 12 章中，我们将学习如何处理异常，保证程序可以继续运行下去。

注意：本书前面章节中的大多数程序分三个步骤执行，即输入、处理和输出，这被称为 IPO。输入是从用户那里获得输入，处理是使用输入产生结果，而输出是显示结果。

复习题

2.2 如何编写一条语句，让用户从键盘输入一个双精度值？在执行下面代码的时候，如果你输入 5a，将发生什么？

```
double radius = input.nextDouble();
```

2.3 下面两个 import 语句之间有什么执行的不同吗？

```
import java.util.Scanner;
import java.util.*;
```

2.4 标识符

要点提示：标识符是为了标识程序中诸如类、方法和变量的元素而采用的命名。

正如在程序清单 2-3 中看到的，ComputeAverage、main、input、number1、number2、number3 等都是出现在程序中事物的名字。在程序设计术语中，这样的名字称为标识符 (identifier)。所有的标识符必须遵从以下规则：

- 标识符是由字母、数字、下划线 (_) 和美元符号 (\$) 构成的字符序列。
- 标识符必须以字母、下划线 (_) 或美元符号 (\$) 开头，不能以数字开头。
- 标识符不能是保留字 (参见附录 A 中的保留字列表)。
- 标识符不能是 true、false 或 null。
- 标识符可以为任意长度。

例如，\$2、ComputeArea、area、radius 和 print 都是合法的标识符，而 2A 和 d+4 都是非法的，因为它们不符合标识符的命名规则。Java 编译器会检测出非法标识符，并且报语法错误。

注意：由于 Java 是区分大小写的，所以 area、Area 和 AREA 都是不同的标识符。

提示：标识符是用于命名程序中的变量、方法、类和其他项。具有描述性的标识符可提高程序的可读性。避免采用缩写作为标识符，使用完整的词汇会更具有描述性。比如，numberOfStudents 比 numStuds、numOfStuds 或者 numofStudents 要好。本教材中我们在完整的程序采用描述性的命名。然而，为了简明，我们也会偶尔在一些代码片段中采用诸如 i、j、k、x 和 y 之类的变量名。这样的命名在代码片段中也是具有一定普遍性的做法。

提示：不要用字符 \$ 命名标识符。习惯上，字符 \$ 只用在机器自动产生的源代码中。

复习题

2.4 以下标识符哪些是合法的？哪些是 Java 的关键字？

```
miles, Test, a++, --a, 4#R, $4, #44, apps
class, public, int, x, y, radius
```

2.5 变量

要点提示：变量用于表示在程序中可能被改变的值。

正如在前几节的程序中看到的，变量被用于存储程序中后面要用到的值。它们被称为变量是因为它们的值可以被改变。在程序清单 2-2 中，radius 和 area 都是双精度浮点型变量。可以将任意数值赋给 radius 和 area，并且可以对它们重新赋值。例如，在下面的代码中，radius 初始为 1.0 (第 2 行)，然后被改为 2.0 (第 7 行)，面积被设为 3.14159 (第 3 行)，然

后被重新设置为 12.56636 (第 8 行)。

```

1 // Compute the first area
2 radius = 1.0;                                radius: 1.0
3 area = radius * radius * 3.14159;           area: 3.14159
4 System.out.println("The area is " + area + " for radius " + radius);
5
6 // Compute the second area
7 radius = 2.0;                                radius: 2.0
8 area = radius * radius * 3.14159;           area: 12.56636
9 System.out.println("The area is " + area + " for radius " + radius);

```

变量用于表示特定类型的数据。为了使用变量，可以通过告诉编译器变量的名字及其可以存储的数据类型来声明该变量。变量声明告知编译器根据数据类型为变量分配合适的内存空间。声明变量的语法如下：

```
datatype variableName;
```

下面是一些变量声明的例子：

```

int count; // Declare count to be an integer variable
double radius; // Declare radius to be a double variable
double interestRate; // Declare interestRate to be a double variable

```

这个例子中使用了数据类型 `int` 和 `double`。后面还将介绍更多的数据类型，例如，`byte`、`short`、`long`、`float`、`char` 和 `boolean`。

如果几个变量为同一类型，允许一起声明它们，如下所示：

```
datatype variable1, variable2, ..., variablen;
```

变量之间用逗号分隔开。例如：

```
int i, j, k; // Declare i, j, and k as int variables
```

变量通常都有初始值。可以一步完成变量的声明和初始化。例如，考虑下面的代码：

```
int count = 1;
```

它等同于下面的两条语句：

```

int count;
count = 1;

```

也可以使用简捷的方式来同时声明和初始化同一类型的变量。例如：

```
int i = 1, j = 2;
```

提示：在赋值给变量之前，必须声明变量。方法中声明的变量在使用之前必须被赋值。

任何时候，都要尽可能一步完成变量的声明和赋初值。这会使得程序易读，同时避免程序设计错误。

每个变量都有使用范围。变量的使用范围是指变量可以被引用到的程序的部分。定义变量使用范围的规则将在本书后面逐步介绍。目前，你需要知道的是，一个变量在可以使用前，必须被声明和初始化。

复习题

2.5 请指出并修改下面代码中的错误：

```

1 public class Test {
2     public static void main(String[] args) {

```

```

3     int i = k + 2;
4     System.out.println(i);
5 }
6 }

```

2.6 赋值语句和赋值表达式

要点提示：赋值语句将一个值指定给一个变量。在 Java 中赋值语句可以作为一个表达式。

声明变量之后，可以使用赋值语句（assignment statement）给它赋一个值。在 Java 中，将等号（=）作为赋值操作符（assignment operator）。赋值语句的语法如下所示：

```
variable = expression; (变量 = 表达式;)
```

表达式（expression）表示涉及值、变量和操作符的一个运算，它们组合在一起计算出一个新值。例如，考虑下面的代码：

```

int y = 1;           // Assign 1 to variable y
double radius = 1.0; // Assign 1.0 to variable radius
int x = 5 * (3 / 2); // Assign the value of the expression to x
x = y + 1;          // Assign the addition of y and 1 to x
double area = radius * radius * 3.14159; // Compute area

```

变量也可用在表达式中。变量也可以用于 = 操作符的两边，例如：

```
x = x + 1;
```

在这个赋值语句中， $x+1$ 的结果赋值给 x 。假设在语句执行前 x 为 1，那么语句执行后它就变成了 2。

要给一个变量赋值，变量名必须在赋值操作符的左边。因此，下面的语句是错误的。

```
1 = x; // Wrong
```

注意：在数学运算中， $x=2*x+1$ 表示一个等式。但是，在 Java 中， $x=2*x+1$ 是一个赋值语句，它计算表达式 $2*x+1$ ，并且将结果赋值给 x 。

在 Java 中，赋值语句本质上就是计算出一个值并将它赋给操作符左边变量的一个表达式。由于这个原因，赋值语句常常称作赋值表达式（assignment expression）。例如，下面的语句是正确的：

```
System.out.println(x = 1);
```

它等价于语句：

```

x = 1;
System.out.println(x);

```

如果一个值要赋给多个变量，可以采用以下语法：

```
i = j = k = 1;
```

它等价于：

```

k = 1;
j = k;
i = j;

```

注意：在赋值语句中，左边变量的数据类型必须与右边值的数据类型兼容。例如，`int x=1.0` 是非法的，因为 x 的数据类型是整型 `int`。在不使用类型转换的情况下，是不能把

double 值 (1.0) 赋给 int 变量的。类型转换将在 2.15 节介绍。

复习题

2.6 请指出并修改下面代码中的错误：

```
1 public class Test {
2     public static void main(String[] args) {
3         int i = j = k = 2;
4         System.out.println(i + " " + j + " " + k);
5     }
6 }
```

2.7 命名常量

要点提示：命名常量是一个代表不变值的标识符。

一个变量的值在程序执行过程中可能会发生变化，但是命名常量 (named constant) 或简称常量，则表示从不改变的永久数据。在程序清单 2-1 中， π 是一个常量。如果频繁使用它，但又不想重复地输入 3.14159，代替的方式就是声明一个常量 π 。下面就是声明常量的语法：

```
final datatype CONSTANTNAME = value;
```

常量必须在同一条语句中声明和赋值。单词 final 是声明常量的 Java 关键字。例如，可以将 π 声明为常量，然后将程序清单 2-1 改写为程序清单 2-4：

程序清单 2-4 ComputeAreaWithConstant.java

```
1 import java.util.Scanner; // Scanner is in the java.util package
2
3 public class ComputeAreaWithConstant {
4     public static void main(String[] args) {
5         final double PI = 3.14159; // Declare a constant
6
7         // Create a Scanner object
8         Scanner input = new Scanner(System.in);
9
10        // Prompt the user to enter a radius
11        System.out.print("Enter a number for radius: ");
12        double radius = input.nextDouble();
13
14        // Compute area
15        double area = radius * radius * PI;
16
17        // Display result
18        System.out.println("The area for the circle of radius " +
19            radius + " is " + area);
20    }
21 }
```

使用常量有三个好处：1) 不必重复输入同一个值；2) 如果必须修改常量值 (例如，将 PI 的值从 3.14 改为 3.14159)，只需在源代码中的一个地方做改动；3) 给常量赋一个描述性名字会提高程序易读性。

2.8 命名习惯

要点提示：严格遵循 Java 的命名习惯可以让你的程序易于理解，以及避免错误。

应该确保程序中为变量、常量、类和方法所选择的描述性名字是直观易懂的。如前所述，命名是区分大小写的。下面列出变量、常量、方法和类的命名习惯。

- 使用小写字母命名变量和方法。如果一个名字包含多个单词，就将它们连在一起，第一个单词的字母小写，而后面的每个单词的首字母大写，例如，变量 `radius` 和 `area` 以及方法 `print`。
- 类名中的每个单词的首字母大写，例如，类名 `ComputeArea` 和 `System`。
- 大写常量中的所有字母，两个单词间用下划线连接，例如，常量 `PI` 和常量 `MAX_VALUE`。

严格遵循 Java 的命名习惯是非常重要的，这样可以让你的程序易于理解。

 **警告：**对类命名时不要选择 Java 库中已经使用的名称。例如，因为 Java 已定义了 `System` 类，就不要用 `System` 来命名自己的类。

复习题

- 2.7 使用常量的好处是什么？声明一个 `int` 类型的常量 `SIZE`，并且值为 20。
- 2.8 类名、方法名、常量和变量的命名习惯是什么？按照 Java 的命名习惯，以下哪些项可以作为常量，方法、变量或者一个类？

`MAX_VALUE`, `Test`, `read`, `readDouble`

- 2.9 将以下算法翻译成 Java 代码。

第一步：声明一个双精度型变量 `miles`，初始值为 100。

第二步：声明一个双精度型常量 `KILOMETERS_PER_MILE`，初始值为 1.609。

第三步：声明一个双精度型变量 `kilometers`，将 `miles` 和 `KILOMETERS_PER_MILE` 相乘，并且将结果赋值给 `kilometers`。

第四步：在控制台显示 `kilometers`。

第四步之后，`kilometers` 是多少？

2.9 数值数据类型和操作

 **要点提示：**Java 针对整数和浮点数有六种数值类型，以及 `+`、`-`、`*`、`/`、和 `%` 等操作符。

2.9.1 数值类型

每个数据类型都有它的取值范围。编译器会根据每个变量或常量的数据类型为其分配内存空间。Java 为数值、字符值和布尔值数据提供了八种基本数据类型。本节介绍数值数据类型和操作符。

表 2-1 列出了六种数值数据类型、它们的范围以及所占存储空间。

表 2-1 数值数据类型

类型名	范围	存储大小
<code>byte</code>	-2^7 (-128) \sim 2^7-1 (127)	8 位带符号数
<code>short</code>	-2^{15} (-32 768) \sim $2^{15}-1$ (32 767)	16 位带符号数
<code>int</code>	-2^{31} (-2 147 483 648) \sim $2^{31}-1$ (2 147 483 647)	32 位带符号数
<code>long</code>	$-2^{63} \sim 2^{63}-1$ (即 -9 223 372 036 854 775 808 \sim 9 223 372 036 854 775 807)	64 位带符号数
<code>float</code>	负数范围: $-3.4028235E+38 \sim -1.4E-45$ 正数范围: $1.4E-45 \sim 3.4028235E+38$	32 位, 标准 IEEE 754

(续)

类型名	范围	存储大小
double	负数范围: $-1.7976931348623157E+308 \sim -4.9E-324$ 正数范围: $4.9E-324 \sim 1.7976931348623157E+308$	64 位, 标准 IEEE 754

注意: IEEE 754 是美国电气电子工程师协会通过的一个标准, 用于在计算机上表示浮点数。该标准已被广泛采用。Java 采用 32 位 IEEE 754 表示 float 型, 64 位 IEEE 754 表示 double 型。IEEE 754 标准还定义了一些特殊浮点值, 这些值都在附录 E 中列出。

Java 使用四种类型的整数: byte、short、int 和 long。应该为变量选择最适合的数据类型。例如: 如果知道存储在变量中的整数是在字节范围内, 将该变量声明为 byte 型。为了简单和一致性, 我们在本书的大部分内容中都使用 int 来表示整数。

Java 使用两种类型的浮点数: float 和 double。double 型是 float 型的两倍。所以, double 型又称为双精度 (double precision), 而 float 称为单精度 (single precision)。通常情况下, 应该使用 double 型, 因为它比 float 型更精确。

2.9.2 从键盘读取数值

你知道如何使用 Scanner 类中的 nextDouble() 方法来从键盘读取一个 double 数值。你也可以使用列在表 2-2 中的方法来读取 byte、short、int、long 以及 float 类型的数值。

下面是从键盘上读取各种类型数值的例子:

```

1 Scanner input = new Scanner(System.in);
2 System.out.print("Enter a byte value: ");
3 byte byteValue = input.nextByte();
4
5 System.out.print("Enter a short value: ");
6 short shortValue = input.nextShort();
7
8 System.out.print("Enter an int value: ");
9 int intValue = input.nextInt();
10
11 System.out.print("Enter a long value: ");
12 long longValue = input.nextLong();
13
14 System.out.print("Enter a float value: ");
15 float floatValue = input.nextFloat();

```

如果你输入了一个不正确范围或者格式的值, 将产生一个运行时错误。比如, 为第 3 行你输入了 128, 将产生一个错误, 因为 128 已经超过了 byte 类型整数的范围。

2.9.3 数值操作符

数值数据类型的操作符包括标准的算术操作符: 加号 (+)、减号 (-)、乘号 (*)、除号 (/) 和求余号 (%), 如表 2-3 所示。操作数是被操作符操作的值。

表 2-3 数值操作符

运算符	名字	示例	运算结果	运算符	名字	示例	运算结果
+	加	34 + 1	35	/	除	1.0 / 2.0	0.5
-	减	34.0 - 0.1	33.9	%	求余	20 % 3	2
*	乘	300 * 30	9000				

表 2-2 Scanner 对象的方法

方法	描述
nextByte()	读取一个 byte 类型的整数
nextShort()	读取一个 short 类型的整数
nextInt()	读取一个 int 类型的整数
nextLong()	读取一个 long 类型的整数
nextFloat()	读取一个 float 类型的数
nextDouble()	读取一个 double 类型的数

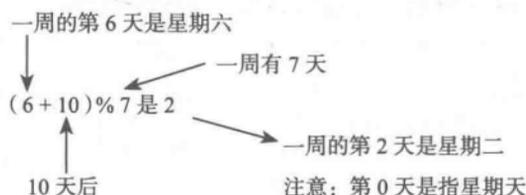
当除法的操作数都是整数时，除法的结果就是整数，小数部分被舍去。例如：5/2 的结果是 2 而不是 2.5，而 -5/2 的结果是 -2 而不是 -2.5。为了实现浮点数的除法，其中一个操作数必须是浮点数。例如：5.0/2 的结果是 2.5。

操作符 %，被称为求余或者取模操作符，可以求得除法的余数。左边的操作数是被除数，右边的操作数是除数。因此，7%3 的结果是 1，3%7 的结果是 3，12%4 的结果是 0，26%8 的结果是 2，20%13 的结果是 7。

$$\begin{array}{r}
 2 \\
 3 \overline{)7} \\
 \underline{6} \\
 1
 \end{array}
 \quad
 \begin{array}{r}
 0 \\
 7 \overline{)3} \\
 \underline{0} \\
 3
 \end{array}
 \quad
 \begin{array}{r}
 3 \\
 4 \overline{)12} \\
 \underline{12} \\
 0
 \end{array}
 \quad
 \begin{array}{r}
 0 \\
 8 \overline{)26} \\
 \underline{24} \\
 2
 \end{array}
 \quad
 \text{被除数} \longrightarrow
 \begin{array}{r}
 1 \leftarrow \text{商} \\
 13 \overline{)20} \leftarrow \text{除数} \\
 \underline{13} \\
 7 \leftarrow \text{余数}
 \end{array}$$

操作符 % 通常用在正整数上，实际上，它也可用于负整数和浮点值。只有当被除数是负数时，余数才是负的。例如：-7%3 结果是 -1，-12%4 结果是 0，-26%-8 结果是 -2，20%-13 结果是 7。

在程序设计中余数是非常有用的。例如：偶数 %2 的结果总是 0 而正奇数 %2 的结果总是 1。所以，可以利用这一特性来判定一个数是偶数还是奇数。如果今天是星期六，7 天之后就又会星期六。假设你和你的朋友计划 10 天之后见面，那么 10 天之后是星期几呢？使用下面的表达式你就能够发现那天是星期二：



程序清单 2-5 计算以秒为单位的时间量所包含的分钟数和余下的秒数。例如，500 秒就是 8 分钟 20 秒。

程序清单 2-5 DisplayTime.java

```

1 import java.util.Scanner;
2
3 public class DisplayTime {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6         // Prompt the user for input
7         System.out.print("Enter an integer for seconds: ");
8         int seconds = input.nextInt();
9
10        int minutes = seconds / 60; // Find minutes in seconds
11        int remainingSeconds = seconds % 60; // Seconds remaining
12        System.out.println(seconds + " seconds is " + minutes +
13            " minutes and " + remainingSeconds + " seconds");
14    }
15 }

```

Enter an integer for seconds: 500
500 seconds is 8 minutes and 20 seconds

line#	seconds	minutes	remainingSeconds
8	500		
10		8	
11			20

nextInt() 方法 (第 8 行) 读取 seconds 的整数值。第 10 行使用 seconds/60 获取分钟数。第 11 行 (seconds%60) 获取在减去分钟数之后得到的剩余秒数。

操作符 + 和 - 可以是一元的也可以是二元的。一元操作符仅有一个操作数; 而二元操作符有两个操作数。例如, 在 -5 中, 负号 (-) 可以认为是一元操作符, 是对正数 5 取负数值, 而在表达式 4-5 中, 负号 (-) 是二元操作符, 是从 4 中减去 5。

2.9.4 幂运算

使用方法 Math.pow(a,b) 来计算 a^b 。pow 方法定义在 Java API 的 Math 类中。运用语法 Math.pow(a,b) 可以调用 (比如, Math.pow(2,3)) 该方法, 并将返回结果 a^b (2^3)。这里, a 和 b 是 pow 方法的参数, 而数值 2 和 3 是调用方法时的真实值。比如:

```
System.out.println(Math.pow(2, 3)); // Displays 8.0
System.out.println(Math.pow(4, 0.5)); // Displays 2.0
System.out.println(Math.pow(2.5, 2)); // Displays 6.25
System.out.println(Math.pow(2.5, -2)); // Displays 0.16
```

第 5 章将介绍关于方法的更多细节。现在, 你只需要知道如何通过调用 pow 方法来执行幂运算。

复习题

2.10 找到最大和最小的 byte、short、int、long、float 以及 double。这些数据类型中, 哪个需要的内存最小?

2.11 给出以下求余计算的结果。

```
56 % 6
78 % -4
-34 % 5
-34 % -5
5 % 1
1 % 5
```

2.12 假设今天是周二, 100 天后将是周几?

2.13 25/4 的结果是多少? 如果你希望得到浮点数结果, 如何重写表达式?

2.14 给出以下代码的结果:

```
System.out.println(2 * (5 / 2 + 5 / 2));
System.out.println(2 * 5 / 2 + 2 * 5 / 2);
System.out.println(2 * (5 / 2));
System.out.println(2 * 5 / 2);
```

2.15 下面的语句正确吗? 如果正确的话, 给出输出。

```
System.out.println("25 / 4 is " + 25 / 4);
System.out.println("25 / 4.0 is " + 25 / 4.0);
System.out.println("3 * 2 / 4 is " + 3 * 2 / 4);
System.out.println("3.0 * 2 / 4 is " + 3.0 * 2 / 4);
```

2.16 写一个显示 $2^{3.5}$ 的计算结果的语句。

2.17 假设 m 和 r 是整数。编写一个 Java 表达式, 使得 mr^2 可以得到一个浮点数类型的结果。

2.10 数值型直接量

 要点提示: 一个直接量 (literal) 是一个程序中直接出现的常量值。

例如, 下面的语句中 34 和 0.305 都是直接量:

```
int numberOfYears = 34;
double weight = 0.305;
```

2.10.1 整型直接量

只要整型直接量与整型变量相匹配，就可以将整型直接量赋值给该整型变量。如果直接量太大，超出该变量的存储范围，就会出现编译错误。例如：语句 `byte b=128` 就会造成一个编译错误，因为 `byte` 型变量存放不下 128（注意：`byte` 型变量的范围是 $-128 \sim 127$ ）。

整型直接量默认是 `int` 型的，它的值在 $-2^{31}(-2\ 147\ 483\ 648) \sim 2^{31}-1(2\ 147\ 483\ 647)$ 。为了表示一个 `long` 型的整型直接量，需要在其后追加字母 `L` 或 `l`（例如：`2147483648L`）。为了在 `Java` 程序中表示整数 2147483648，必须将它写成 `2147483648L` 或者 `2147483648l`，因为 `2147483648` 超出了 `int` 型的范围。推荐使用 `L`，因为 `l` (`L` 的小写) 很容易与 `1` (数字 1) 混淆。

注意：默认情况下，整型直接量是一个十进制整数。要表示一个二进制整数直接量，使用 `0b` 或者 `0B` (零 B) 开头；表示一个八进制整数直接量，就用 `0` (零) 开头，而要表示一个十六进制整数直接量，就用 `0x` 或 `0X` (零 x) 开头。例如，

```
System.out.println(0B1111); // Displays 15
System.out.println(07777); // Displays 4095
System.out.println(0xFFFF); // Displays 65535
```

十六进制数、二进制数和八进制数都将在附录 F 中介绍。

2.10.2 浮点型直接量

浮点型直接量带小数点，默认情况下是 `double` 型的。例如：5.0 被认为是 `double` 型而不是 `float` 型。可以通过在数字后面加字母 `f` 或 `F` 表示该数为 `float` 型直接量，也可以在数字后面加 `d` 或 `D` 表示该数为 `double` 型直接量。例如：可以使用 `100.2f` 或 `100.2F` 表示 `float` 型值，用 `100.2d` 或 `100.2D` 表示 `double` 型值。

注意：`double` 型值比 `float` 型值更精确。例如：

```
System.out.println("1.0 / 3.0 is " + 1.0 / 3.0);
```

显示结果为：`1.0/3.0 is 0.3333333333333333` (小数点后 16 位)。

```
System.out.println("1.0F / 3.0F is " + 1.0F / 3.0F);
```

显示结果为：`1.0F/3.0F is 0.33333334` (小数点后 8 位)。

一个 `float` 值有 7 到 8 位小数位，一个 `double` 值有 15 到 17 位小数位。

2.10.3 科学记数法

浮点型直接量也可以用 $a \times 10^b$ 形式的科学记数法表示，例如，123.456 的科学记数法形式是 1.23456×10^2 ，0.012 345 6 的科学记数法是 1.23456×10^{-2} 。一种特定的语法可以用于表示科学记数法的数值。例如， 1.23456×10^2 可以写成 `1.23456E2` 或者 `1.23456E+2`，而 1.23456×10^{-2} 等于 `1.23456E-2`。`E` (或 `e`) 表示指数，既可以是上写的也可以是小写的。

注意：`float` 型和 `double` 型都是用来表示带有小数点的数。为什么把它们称为浮点数呢？因为这些数都是以科学记数法的形式进行内部存储的。当一个像 50.534 的数被转换成科学记数法的形式时，它就是 `5.0534E+1`，它的小数点就移到 (即浮动到) 一个新的位置。

注意：为了提高可读性，`Java` 允许在数值直接量的两个数字间使用下划线。例如，下面的直接量是正确的：

```
long ssn = 232_45_4519;
long creditCardNumber = 2324_4545_4519_3415L;
```

然而, 45_ 和 _45 是不正确的。下划线必须置于两个数字间。

复习题

2.18 在 float 和 double 类型的变量中保存了多少个精确位?

2.19 以下哪些是正确的浮点数类型直接量?

12.3, 12.3e+2, 23.4e-2, -334.4, 20.5, 39F, 40D

2.20 以下哪些和 52.534 是等价的?

5.2534e+1, 0.52534e+2, 525.34e-1, 5.2534e+0

2.21 以下哪些是正确的直接量?

5_2534e+1, _2534, 5_2, 5_

2.11 表达式求值以及操作符优先级

要点提示: Java 表达式的求值和数学表达式求值是一样的。

用 Java 编写数值表达式就是使用 Java 操作符对算术表达式进行直接的翻译。例如, 下述算术表达式:

$$\frac{3+4x}{5} - \frac{10(y-5)(a+b+c)}{x} + 9\left(\frac{4}{x} + \frac{9+x}{y}\right)$$

可以翻译成如下所示的 Java 表达式:

```
(3 + 4 * x) / 5 - 10 * (y - 5) * (a + b + c) / x +
9 * (4 / x + (9 + x) / y)
```

尽管 Java 有自己在后台计算表达式的方法, 但是, Java 表达式的结果和它对应的算术表达式的结果是一样的。因此, 可以放心地将算术运算规则应用在计算 Java 表达式上。首先执行的是包括在圆括号里的运算。圆括号可以嵌套, 嵌套时先计算内层括号。当一个表达式中有多于一个操作符时, 以下操作符的优先级规则用于确定计算的次序:

- 乘法、除法和求余运算首先计算。如果表达式中包含若干个乘法、除法和求余操作符, 可按照从左到右的顺序执行。
- 最后执行加法和减法运算。如果表达式中包含若干个加法和减法操作符, 则按照从左到右的顺序执行。

下面是一个如何计算表达式的例子:

```
3 + 4 * 4 + 5 * (4 + 3) - 1
      ↑
      (1) 圆括号里的先计算
3 + 4 * 4 + 5 * 7 - 1
      ↑
      (2) 乘法
3 + 16 + 5 * 7 - 1
      ↑
      (3) 乘法
3 + 16 + 35 - 1
      ↑
      (4) 加法
19 + 35 - 1
      ↑
      (5) 加法
54 - 1
      ↑
      (6) 减法
53
```

程序清单 2-6 给出了利用公式 $celsius = \left(\frac{5}{9}\right) (fahrenheit - 32)$ 将华氏温度转换成摄氏温度的程序。

程序清单 2-6 FahrenheitToCelsius.java

```

1 import java.util.Scanner;
2
3 public class FahrenheitToCelsius {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6
7         System.out.print("Enter a degree in Fahrenheit: ");
8         double fahrenheit = input.nextDouble();
9
10        // Convert Fahrenheit to Celsius
11        double celsius = (5.0 / 9) * (fahrenheit - 32);
12        System.out.println("Fahrenheit " + fahrenheit + " is " +
13            celsius + " in Celsius");
14    }
15 }

```

Enter a degree in Fahrenheit: 100
 Fahrenheit 100.0 is 37.7777777777778 in Celsius

line#	fahrenheit	celsius
8	100	
11		37.7777777777778

使用除法时要特别小心。在 Java 中，两个整数相除商为整数。在第 11 行，将 $\frac{5}{9}$ 转换为 $5.0/9$ 而不再是 $5/9$ ，因为在 Java 中 $5/9$ 的结果是 0。

复习题

2.22 如何在 Java 中表达以下算术表达式？

- a. $\frac{4}{3(r+34)} - 9(a+bc) + \frac{3+d(2+a)}{a+bd}$
 b. $5.5 \times (r + 2.5)^{2.5+r}$

2.12 示例学习：显示当前时间

要点提示： 可以通过调用 `System.currentTimeMillis()` 返回当前时间。

本节的问题是开发一个以 GMT（格林威治标准时间）来显示当前时间的程序，以小时：分钟：秒的格式来显示，例如 13:19:8。

`System` 类中的方法 `currentTimeMillis` 返回从 GMT 1970 年 1 月 1 日 00:00:00 开始到当前时刻的毫秒数，如图 2-2 所示。时间戳是时间开始计时的点，因为 1970 年是 UNIX 操作系统正式发布的时间，所以这一时间也称为 UNIX 时间戳（UNIX epoch）。

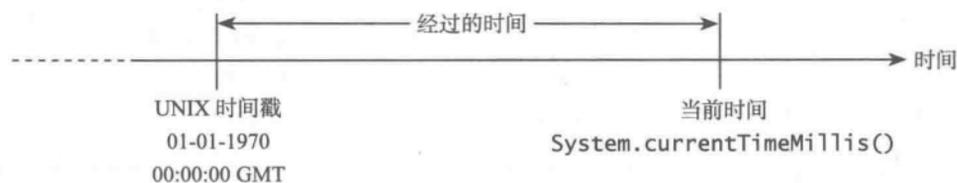


图 2-2 `System.currentTimeMillis()` 返回自 UNIX 时间戳以来的毫秒数

可以使用这个方法获取当前时间，然后按照如下步骤计算出当前的秒数、分钟数和小时数：

1) 调用 `System.currentTimeMillis()` 方法获取 1970 年 1 月 1 日午夜到现在的毫秒数 (例如：1203183086328 毫秒)，并存放在变量 `totalMilliseconds` 中。

2) 通过将总毫秒数 `totalMilliseconds` 除以 1000 得到总秒数 `totalSeconds` (例如：1203183086328 毫秒 / 1000 = 1203183068 秒)。

3) 通过 `totalSeconds % 60` 得到当前的秒数 (例如：1203183068 秒 % 60 = 8，这个值就是当前秒数)。

4) 通过将 `totalSeconds` 除以 60 得到总的分钟数 `totalMinutes` (例如：1203183068 秒 / 60 = 20053051 分钟)。

5) 通过 `totalMinutes % 60` 得到当前分钟数 (例如：20053051 分钟 % 60 = 31，这个值就是当前分钟数)。

6) 通过将总分钟数 `totalMinutes` 除以 60 获得总的小时数 `totalHours` (例如：20053051 分钟 / 60 = 334217 小时)。

7) 通过 `totalHours % 24` 得到当前的小时数 (例如：334217 小时 % 24 = 17，该值就是当前小时数)。

程序清单 2-7 给出完整的程序。

程序清单 2-7 ShowCurrentTime.java

```
1 public class ShowCurrentTime {
2     public static void main(String[] args) {
3         // Obtain the total milliseconds since midnight, Jan 1, 1970
4         long totalMilliseconds = System.currentTimeMillis();
5
6         // Obtain the total seconds since midnight, Jan 1, 1970
7         long totalSeconds = totalMilliseconds / 1000;
8
9         // Compute the current second in the minute in the hour
10        long currentSecond = totalSeconds % 60;
11
12        // Obtain the total minutes
13        long totalMinutes = totalSeconds / 60;
14
15        // Compute the current minute in the hour
16        long currentMinute = totalMinutes % 60;
17
18        // Obtain the total hours
19        long totalHours = totalMinutes / 60;
20
21        // Compute the current hour
22        long currentHour = totalHours % 24;
23
24        // Display results
25        System.out.println("Current time is " + currentHour + ":"
26            + currentMinute + ":" + currentSecond + " GMT");
27    }
28 }
```

Current time is 17:31:8 GMT

line#	4	7	10	13	16	19	22
variables							
totalMilliseconds	1203183068328						
totalSeconds		1203183068					
currentSecond			8				
totalMinutes				20053051			
currentMinute					31		
totalHours						334217	
currentHour							17

第4行调用 `System.currentTimeMillis()` 获得一个 `long` 类型的以毫秒为单位的当前时间值。因此，在该程序中的所有变量都被声明为 `long` 型。秒、分钟、小时数都从当期时间中运用 `/` 和 `%` 操作符抽取得到（第6~22行）。

在一个示例运行中，仅一位的数字8显示为秒数，而希望的输出是08。这可以运用一个方法来修复，该方法可以将单位数字格式化为加一位前缀0（参见编程练习题6.37）。

复习题

2.23 如何获得当前的秒、分钟以及小时数？

2.13 增强赋值操作符

要点提示：操作符 `+`、`-`、`*`、`/`、`%` 可以结合赋值操作符形成增强操作符。

经常会出现变量的当前值被使用、修改，然后再重新赋值给该变量的情况。例如，下面语句将变量 `count` 加1。

```
count = count + 1;
```

Java 允许使用增强赋值操作符来结合赋值和加法操作符的功能。例如，上面的语句可以写成：

```
count += 1;
```

符号 `+=` 称为加法赋值操作符（addition assignment operator）。其他简捷赋值操作符如表2-4中所示。

表 2-4 简捷赋值操作符

操作符	名称	示例	等价于
<code>+=</code>	加法赋值操作符	<code>i += 8</code>	<code>i = i + 8</code>
<code>-=</code>	减法赋值操作符	<code>i -= 8</code>	<code>i = i - 8</code>
<code>*=</code>	乘法赋值操作符	<code>i *= 8</code>	<code>i = i * 8</code>
<code>/=</code>	除法赋值操作符	<code>i /= 8</code>	<code>i = i / 8</code>
<code>%=</code>	求余赋值操作符	<code>i %= 8</code>	<code>i = i % 8</code>

增强赋值操作符在表达式中所有其他操作符计算完成后执行。例如：

```
x /= 4 + 5.5 * 1.5;
```

等同于

```
x = x / (4 + 5.5 * 1.5);
```

 **警告：**在增强操作符中是没有空格的。例如：`+ =`应该是`+=`。

 **注意：**就像赋值操作符(=)一样，操作符(+=、-=、*=、/=、%=)既可以构成赋值语句，也可以构成赋值表达式。例如，在下面的代码中，第1行的`x+=2`是一条语句，而在第2行中它就是一个表达式：

```
x += 2; // Statement
System.out.println(x += 2); // Expression
```

复习题

2.24 给出以下代码运行的结果：

```
double a = 6.5;
a += a + 1;
System.out.println(a);
a = 6;
a /= 2;
System.out.println(a);
```

2.14 自增和自减操作符

 **要点提示：**自增操作符(++)和自减操作符(--)是对变量进行加1和减1的操作。

`++`和`--`是对变量进行自增1和自减1的简写操作符。由于这是许多编程任务中经常需要改变的值，所以这两个操作符使用起来很方便。例如，下面的代码是对`i`自增1，而对`j`自减1：

```
int i = 3, j = 3;
i++; // i becomes 4
j--; // j becomes 2
```

`i++`读为`i`加加，`i--`读为`i`减减。这些操作符分别称为后置自增操作符和后置自减操作符，因为操作符`++`和`--`放在变量后面。这些操作符也可以放在变量前面，比如：

```
int i = 3, j = 3;
++i; // i becomes 4
--j; // j becomes 2
```

`++i`将`i`增加1，`--j`将`j`减去1。这些操作符称为前置自增操作符和前置自减操作符。

如你所见，前面的例子中，`i++`和`++i`的效果，或者`i--`和`--i`的效果是一样的。然而，当用在表达式中不单纯只进行自增和自减时，它们就会产生不同的效果。表2-5描述了它们的不同，并且给出了示例。

表 2-5 自增和自减操作符

操作符	名称	说明	示例(假设 <i>i</i> =1)
<code>++var</code>	前置自增操作符	变量 <code>var</code> 的值加 1 且使用 <code>var</code> 增加后的新值	<code>int j=++i;</code>
<code>var++</code>	后置自增操作符	变量 <code>var</code> 的值加 1 但使用 <code>var</code> 原来的值	<code>int j=i++;</code>
<code>--var</code>	前置自减操作符	变量 <code>var</code> 的值减 1 且使用 <code>var</code> 减少后的新值	<code>int j=--i;</code>
<code>var--</code>	后置自减操作符	变量 <code>var</code> 的值减 1 但使用 <code>var</code> 原来的值	<code>int j=i--;</code>

下面是展示前置形式的++(或者--)和后置形式的++(或者--)的补充示例。考虑以下代码:

```
int i = 10;
int newNum = 10 * i++;
System.out.print("i is " + i
    + ", newNum is " + newNum);
```

效果等同于

```
int newNum = 10 * i;
i = i + 1;
```

```
i is 11, newNum is 100
```

在此例中,首先对*i*自增1,然后返回*i*的旧值来参与乘法运算。这样,newNum的值就成为100。如果如下所示将*i*++换为++*i*:

```
int i = 10;
int newNum = 10 * (++i);
System.out.print("i is " + i
    + ", newNum is " + newNum);
```

效果等同于

```
i = i + 1;
int newNum = 10 * i;
```

```
i is 11, newNum is 110
```

*i*自增1,然后返回*i*的新值,并参与乘法运算。这样,newNum的值就成为110。

下面是另一个例子:

```
double x = 1.0;
double y = 5.0;
double z = x-- + (++y);
```

在这三行程序执行完之后,*y*的值为6.0,*z*的值为7.0,而*x*的值为0.0。

提示:使用自增操作符和自减操作符可以使表达式更加简短,但也会使它们比较复杂且难以读懂。应该避免在同一个表达式中使用这些操作符修改多个变量或多次修改同一个变量,如: `int k = ++i + i`。

复习题

2.25 下面的说法哪个为真?

- 任何表达式都可以用作一个语句。
- 表达式 `x++` 可以用作一个语句。
- 语句 `x = x + 5` 也是一个表达式。
- `x = y = x = 0` 是非法的。

2.26 给出以下代码的输出:

```
int a = 6;
int b = a++;
System.out.println(a);
System.out.println(b);
a = 6;
b = ++a;
System.out.println(a);
System.out.println(b);
```

2.15 数值类型转换

要点提示:通过显式转换,浮点数可以被转换为整数。

可以完成两个不同类型操作数的二元运算吗?当然可以。如果在一个二元运算中,其中

一个操作数是整数，而另一个操作数是浮点数，Java 会自动地将整数转换为浮点值。这样， $3*4.5$ 就成了 $3.0*4.5$ 。

总是可以将一个数值赋给支持更大数值范围类型的变量，例如，可以将 `long` 型的值赋给 `float` 型变量。但是，如果不进行类型转换，就不能将一个值赋给范围较小类型的变量。类型转换是一种将一种数据类型的值转换成另一种数据类型的操作。将一个小范围类型的变量转换为大范围类型的变量称为拓宽类型 (`widening a type`)，把大范围类型的变量转换为小范围类型的变量称为缩窄类型 (`narrowing a type`)。Java 将自动拓宽一个类型，但是，缩窄类型必须显式完成。

类型转换的语法要求目标类型放在括号内，紧跟其后的是要转换的变量名或值。例如，下面的语句：

```
System.out.println((int)1.7);
```

显示结果为 1。当 `double` 型值被转换为 `int` 型值时，小数部分被截去。

下面的语句：

```
System.out.println((double)1 / 2);
```

显示结果为 0.5，因为 1 首先被转换为 1.0，然后用 2 除 1.0。但是，语句

```
System.out.println(1 / 2);
```

显示结果为 0，因为 1 和 2 都是整数，结果也应该是整数。

🔧 **警告：** 如果要将一个值赋给一个范围较小类型的变量，例如：将 `double` 型的值赋给 `int` 型变量，就必须进行类型转换。如果在这种情况下没有使用类型转换，就会出现编译错误。使用类型转换时必须小心，丢失的信息也许会导致不精确的结果。

🔧 **注意：** 类型转换不改变被转换的变量。例如，下面代码中的 `d` 在类型转换之后值不变：

```
double d = 4.5;
int i = (int)d; // i becomes 4, but d is still 4.5
```

🔧 **注意：** Java 中，`x1 op= x2` 形式的增强赋值表达式，执行为 `x1 = (T)(x1 op x2)`，这里 `T` 是 `x1` 的类型。因此，下面代码是正确的。

```
int sum = 0;
sum += 4.5; // sum becomes 4 after this statement
sum += 4.5 等价于 sum = (int)(sum + 4.5).
```

🔧 **注意：** 将一个 `int` 型变量赋值给 `short` 型或 `byte` 型变量，必须显式地使用类型转换。例如，下述语句就会有一个编译错误：

```
int i = 1;
byte b = i; // Error because explicit casting is required
```

然而，只要整型直接量是在目标变量允许的范围內，那么将整型直接量赋给 `short` 型或 `byte` 型变量时，就不需要显式的类型转换（请参考 2.10 节）。

程序清单 2-8 中的程序将营业税值显示为小数点后两位。

程序清单 2-8 SalesTax.java

```
1 import java.util.Scanner;
2
3 public class SalesTax {
4     public static void main(String[] args) {
```

```

5 Scanner input = new Scanner(System.in);
6
7 System.out.print("Enter purchase amount: ");
8 double purchaseAmount = input.nextDouble();
9
10 double tax = purchaseAmount * 0.06;
11 System.out.println("Sales tax is $" + (int)(tax * 100) / 100.0);
12 }
13 }

```

```

Enter purchase amount: 197.55 
Sales tax is $11.85

```

line#	purchaseAmount	tax	output
8	197.55		
10		11.853	
11			11.85

变量 `purchaseAmount` 是 197.55 (第 8 行)。营业税是销售额的 6%，所以，计算得到的税款 `tax` 为 11.853 (第 10 行)。注意到：

```

tax * 100 是 1185.3
(int)(tax * 100) 是 1185
(int)(tax * 100) / 100.0 是 11.85

```

因此，第 11 行的语句显示营业税是保留小数点后两位的 11.85。

复习题

- 2.27 在一次计算中，各种类型的数值可以一起使用吗？
- 2.28 将一个 `double` 类型数值显式类型转换为 `int` 时，是如何处理 `double` 值的小数部分的？类型转换改变被类型转换的变量吗？
- 2.29 给出以下代码片的输出：

```

float f = 12.5F;
int i = (int)f;
System.out.println("f is " + f);
System.out.println("i is " + i);

```

- 2.30 在程序清单 2-8 中，如果将第 11 行的 `(int)(tax*100)/100` 改为 `(int)(tax*100)/100`，对于输入的购买量 197.55，输出会是什么？
- 2.31 给出以下代码的输出：

```

double amount = 5;
System.out.println(amount / 2);
System.out.println(5 / 2);

```

2.16 软件开发过程

要点提示：软件开发生命周期是一个多阶段的过程，包括需求规范、分析、设计、实现、测试、部署和维护。

开发一个软件产品是一个工程过程。软件产品，无论多大或者多小，具有同样的生命周期：需求规范、分析、设计、实现、测试、部署和维护，如图 2-3 所示。

需求规范是一个规范化的过程，旨在理解软件要处理的问题，以及将软件系统需要做的详细记录到文档中。这个阶段涉及用户和开发者之间紧密的接触。本书中的大多数例子是简

单的，它们的需求非常清晰地表述了。然而，在实际中，问题经常没有很好的定义。开发者需要和他们的顾客（使用软件的个人或者组织）密切合作，仔细地研究问题，以确定软件需要做什么。

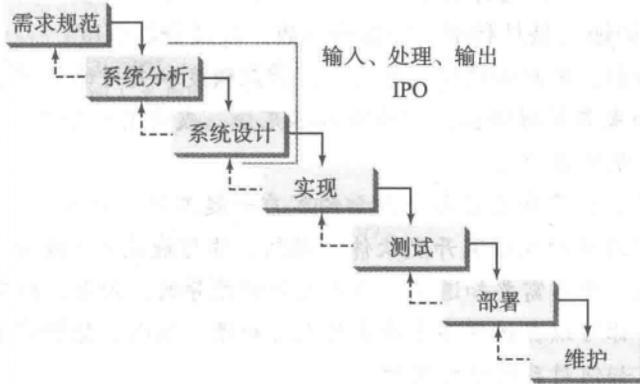


图 2-3 在软件开发生命周期的任何阶段都有可能回到之前的阶段改正错误，或者处理其他可能阻止软件按所设想的发挥功能的问题

系统分析旨在分析数据流，并且确定系统的输入和输出。当进行分析的时候，首先确定输出，然后弄清楚需要什么样子的输入从而产生结果是有帮助的。

系统设计是设计一个从输入获得输出的过程。这个阶段涉及使用多层的抽象，将问题分解为可管理的组成部分，并且设计执行每个组成部分的策略。可以将每个组成部分看作一个执行系统特定功能的子系统。系统分析和设计的本质是输入、处理和输出（IPO）。

实现是将系统设计翻译成程序。为每个组成部分编写独立的程序，然后集成在一起工作。这个过程需要使用一门编程语言，比如 Java。实现包括编码、自我测试，以及调试（即，在代码中寻找错误，称为调试）。

测试确保代码符合需求规范，并且排除错误。通常由一个没有参与产品设计和实现的独立软件工程团队完成这样的测试。

部署使得软件可以被使用。按照软件类型的不同，可能被安装到每个用户的机器上，或者安装在一个 Internet 可访问的服务器上。

维护是对软件产品进行更新和改进。软件产品必须在一直演化的环境中连续运行和改进。这要求产品的周期性改进，以修正新发现的错误，并且将更改集成到产品中。

为了了解实际过程中的软件开发过程，我们现在将创建一个计算贷款支付的程序。贷款可以是车辆贷款、学生贷款，或者一个住宅贷款。针对一个入门的编程课程，我们重点关注需求规范，分析，设计，实现和测试。

1. 需求规范

程序必须满足以下要求：

- 必须让用户输入利率、贷款额度以及支付的年数。
- 必须计算和显示月支付额度和总支付额度。

2. 系统分析

输出是月支付额度和总支付额度，可以通过下面的公式进行计算：

$$\text{月支付额度} = \frac{\text{贷款额度} \times \text{月利率}}{1 - \frac{1}{(1 + \text{月利率})^{\text{年数} \times 12}}}$$

$$\text{总支付额度} = \text{月支付额度} \times \text{年数} \times 12$$

因此，程序需要的输入是月利率、贷款的年数，以及贷款额度。

注意：需求规范给出，用户必须输入年利率、贷款额度和支付的年数。然而，在分析过程中，你可能发现如果要获得输出，有些输入不充分，或者有些值不是必需的。如果这样，你可以回过头来修改需求规范。

注意：实际应用中，你将和来自各个方面的客户一起工作。你可能为化学家、物理学家、工程师、经济学家以及心理学家开发软件。当然，你可能没有（或者不需要有）这些领域的完整知识。因此，你不需要知道这些公式是如何推导的，但是，给定月利率、年数和贷款额度的情况下，你可以在该程序中计算月支付额度。然而，你将需要和客户交流并且理解一个数学模型是如何对系统起作用的。

3. 系统设计

系统设计阶段，你确定程序中的以下步骤。

第一步：提示用户输入年利率、年数以及贷款额度。（利率通常表示为对1年时间的本金的百分比。这被称为年利率。）

第二步：对于年利率的输入是一个百分比格式的数字，比如4.5%。程序需要通过除以100将它转换成为一个十进制数。为了从年利率值得到月利率，将它除以12，因为1年有12个月。因此，为了得到十进制格式的月利率值，需要将百分比格式的年利率数除以1200。比如，如果年利率是4.5%，那么月利率则为 $4.5/1200 = 0.00375$ 。

第三步：使用前面的公式计算月支付额度。

第四步：计算总共支付额度，等于月支付额度乘以12，再乘以年数。

第五步：显示月支付额度和总共支付额度。

4. 实现

实现也称为编码（编写代码）。在公式中，你需要计算 $(1 + \text{月利率})^{\text{年数} \times 12}$ ，这可以通过使用`Math.pow(1 + monthlyInterestRate, numberOfYears * 12)`得到。

程序清单 2-9 给出完整的程序。

程序清单 2-9 ComputeLoan.java

```

1 import java.util.Scanner;
2
3 public class ComputeLoan {
4     public static void main(String[] args) {
5         // Create a Scanner
6         Scanner input = new Scanner(System.in);
7
8         // Enter annual interest rate in percentage, e.g., 7.25%
9         System.out.print("Enter annual interest rate, e.g., 7.25%: ");
10        double annualInterestRate = input.nextDouble();
11
12        // Obtain monthly interest rate
13        double monthlyInterestRate = annualInterestRate / 1200;
14
15        // Enter number of years
16        System.out.print(

```

```

17     "Enter number of years as an integer, e.g., 5: ");
18     int numberOfYears = input.nextInt();
19
20     // Enter loan amount
21     System.out.print("Enter loan amount, e.g., 120000.95: ");
22     double loanAmount = input.nextDouble();
23
24     // Calculate payment
25     double monthlyPayment = loanAmount * monthlyInterestRate / (1
26     - 1 / Math.pow(1 + monthlyInterestRate, numberOfYears * 12));
27     double totalPayment = monthlyPayment * numberOfYears * 12;
28
29     // Display results
30     System.out.println("The monthly payment is $" +
31     (int)(monthlyPayment * 100) / 100.0);
32     System.out.println("The total payment is $" +
33     (int)(totalPayment * 100) / 100.0);
34 }
35 }

```

```

Enter annual interest rate, e.g., 5.75%: 5.75 Enter
Enter number of years as an integer, e.g., 5: 15 Enter
Enter loan amount, e.g., 120000.95: 250000 Enter
The monthly payment is $2076.02
The total payment is $373684.53

```

	line#	10	13	18	22	25	27
variables							
annualInterestRate		5.75					
monthlyInterestRate			0.00479166666666				
numberOfYears				15			
loanAmount					250000		
monthlyPayment						2076.0252175	
totalPayment							373684.539

第 10 行读取年利率值，在第 13 行将其转换为月利率值。

为变量选择最合适的数据类型。比如，numberOfYear 最好声明为 int（第 18 行），尽管它也可以被声明为一个 long、float 或者 double。注意对于 numberOfYears 而言，byte 可能是最合适的。然而，为了简单起见，本书中的示例都将对整数使用 int，以及对浮点值采用 double。

第 25 ~ 27 行将计算月支付额度的公式翻译为 Java 代码。

第 31 和 33 行使用类型转换，获得更新的小数点后有两位的 monthlyPayment 和 totalPayment 值。

程序使用的 Scanner 类在第一行代码中导入。程序也使用了 Math 类，你可能会困惑为什么该类没有被导入程序。Math 类是在 java.lang 包中，而 java.lang 包中的所有类是隐式被导入的。因此，你不需要显式地导入 Math 类。

5. 测试

当实现好程序之后。使用一些样例输入数据并且验证输出是否正确。一些问题将涉及许多情况，如你在后面章节中将会看到的。对于这些类型的问题，你需要设计覆盖所有可能情

况的数据。

 **提示：**这个示例中的系统设计阶段确定了多个步骤。通过一次加入一个步骤，从而对这些步骤进行增量式的编程和测试，是一个好的方法。这个方法使得查明问题以及调试程序变得更加容易。

复习题

2.32 如何编写下面的数学表达式的代码？

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

2.17 示例学习：整钱兑零

 **要点提示：**本节展示一个程序，将一个大额的钱分成较小货币单位。

假如你希望开发一个程序，将给定的钱数分成较小的货币单位。这个程序要求用户输入一个 `double` 型的值，该值是用美元和美分表示的总钱数，然后输出一个清单，列出和总钱数等价的、最大数量的 `dollar`（1 美元）、`quarter`（2 角 5 分）、`dime`（1 角）、`nickel`（5 分）和 `penny`（1 分）的数目，按照这个顺序，从而使得硬币最少。

下面是开发这个程序的步骤：

- 1) 提示用户输入十进制数作为总钱数，例如 11.56。
- 2) 将该钱数（例如 11.56）转换为 1 分币的个数（例如 1156）。
- 3) 通过将 1 分币的个数除以 100，求出 1 美元的个数。通过对 1 分币的个数除以 100 求余数，得到剩余 1 分币的个数。
- 4) 通过将剩余的 1 分币的个数除以 25，求出 2 角 5 分币的个数。通过对剩余的 1 分币的个数除以 25 求余数，得到剩余 1 分币的个数。
- 5) 将剩余的 1 分币的个数除以 10，求出 1 角币的个数。通过对剩余的 1 分币的个数除以 10 求余数，得到剩余 1 分币的个数。
- 6) 将剩余的 1 分币的个数除以 5，求出 5 分币的个数。通过对剩余的 1 分币的个数除以 5 求余数，得到剩余 1 分币的个数。
- 7) 剩余 1 分币的个数即为所求。
- 8) 显示结果。

完整的程序如程序清单 2-10 所示。

程序清单 2-10 ComputeChange.java

```

1  import java.util.Scanner;
2
3  public class ComputeChange {
4      public static void main(String[] args) {
5          // Create a Scanner
6          Scanner input = new Scanner(System.in);
7
8          // Receive the amount
9          System.out.print(
10             "Enter an amount in double, for example 11.56: ");
11         double amount = input.nextDouble();
12
13         int remainingAmount = (int)(amount * 100);
14

```

```

15 // Find the number of one dollars
16 int numberOfOneDollars = remainingAmount / 100;
17 remainingAmount = remainingAmount % 100;
18
19 // Find the number of quarters in the remaining amount
20 int numberOfQuarters = remainingAmount / 25;
21 remainingAmount = remainingAmount % 25;
22
23 // Find the number of dimes in the remaining amount
24 int numberOfDimes = remainingAmount / 10;
25 remainingAmount = remainingAmount % 10;
26
27 // Find the number of nickels in the remaining amount
28 int numberOfNickels = remainingAmount / 5;
29 remainingAmount = remainingAmount % 5;
30
31 // Find the number of pennies in the remaining amount
32 int numberOfPennies = remainingAmount;
33
34 // Display results
35 System.out.println("Your amount " + amount + " consists of");
36 System.out.println("    " + numberOfOneDollars + " dollars");
37 System.out.println("    " + numberOfQuarters + " quarters ");
38 System.out.println("    " + numberOfDimes + " dimes");
39 System.out.println("    " + numberOfNickels + " nickels");
40 System.out.println("    " + numberOfPennies + " pennies");
41 }
42 }
    
```

Enter an amount, for example, 11.56:

Your amount 11.56 consists of

- 11 dollars
- 2 quarters
- 0 dimes
- 1 nickels
- 1 pennies

line#	11	13	16	17	20	21	24	25	28	29	32
variables											
amount	11.56										
remainingAmount		1156		56		6		6		1	
numberOfOneDollars			11								
numberOfQuarters					2						
numberOfDimes							0				
numberOfNickels									1		
numberOfPennies											1

变量 amount 存储的是从控制台上输入的钱数 (第 11 行)。由于在程序结尾处显示结果时要用到该金额，所以该变量的值是不变的。程序引入变量 remainingAmount (第 13 行) 来存储变化的余额 remainingAmount。

变量 amount 是一个 double 型的十进制数，代表美元和美分。它被转换为一个 int 型变量 remainingAmount 以代表总的 1 美分的个数。例如：如果 amount 为 11.56，那么 remainingAmount 的初始值为 1156。除法运算得到除法的整数部分，所以 1156/100 的结果

是 11。求余运算可以得到除法的余数，所以 $1156\%100$ 的结果是 56。

程序从总钱数中抽取出最大数量的 1 美元币，得到的余额存储在变量 `remainingAmount` 中（第 16 ~ 17 行）。接着从 `remainingAmount` 中抽取出最大数量 2 角 5 分币的，得到一个新的余额 `remainingAmount`（第 20 ~ 21 行）。继续同样的过程，程序就找到了余额所能包括的 1 角单位钱币的最多数量、5 分单位钱币的最多数量和 1 分单位钱币的最多数量。

本例的一个严重问题是将一个 `double` 型的总钱数转换为 `int` 型数 `remainingAmount` 时可能会损失精度，这会导致不精确的结果。如果输入的总额值为 10.03，那么 $10.03*100$ 就会变成 1002.9999999999999，程序会显示 10 个 1 美元和 2 个 1 美分。为了解决这个问题，应该输入用美分表示的整型值（参见编程练习题 2.22）。

☛ 复习题

2.33 给出输入值为 1.99 的输出。

2.18 常见错误和陷阱

🔍 **要点提示：**常见的基础编程错误经常涉及未声明变量、未初始化变量、整数溢出、超出预期的整数除法，以及数值取整错误。

常见错误 1：未声明、未初始化的变量和未使用的变量

变量必须在使用之前声明为一个类型并且赋值。一个常见的错误是没有声明变量或者初始化一个变量。考虑下面的代码：

```
double interestRate = 0.05;
double interest = interestrate * 45;
```

这个代码是错误的，因为 `interestRate` 赋值为 0.05，而 `interestrate` 并没有声明和初始化。Java 是区分大小写的，因为 `interestRate` 和 `interestrate` 被认为是两个不同的变量。

如果声明了一个变量，但是没有在程序中使用，将是一个潜在的编程错误。因此，你应该从程序中将未使用的变量移除。例如，在下面代码中，`taxRate` 从未使用。它需要从代码中去掉。

```
double interestRate = 0.05;
double taxRate = 0.05;
double interest = interestRate * 45;
System.out.println("Interest is " + interest);
```

如果你使用诸如 Eclipse 和 NetBeans 这类 IDE，你将收到关于未使用变量的警告消息。

常见错误 2：整数溢出

数字以有限的位数存储。当一个变量被赋予一个过大（以存储大小而言）的值，以至无法存储该值，这称为溢出。例如，执行下面的语句将导致溢出，因为在一个 `int` 类型变量中可以存储的最大值是 2147483647。2147483648 将超出 `int` 值的范围。

```
int value = 2147483647 + 1;
// value will actually be -2147483648
```

同样，执行下面的语句也会产生溢出，因为可以存储在 `int` 类型变量中的最小值是 -2147483648。从存储空间大小而言，-2147483649 对于一个 `int` 类型变量来说太大了。

```
int value = -2147483648 - 1;
// value will actually be 2147483647
```

Java 不会给出关于溢出的警告或者错误，因此，当处理一个与给定类型的最大和最小范

围很接近的数值时，要特别小心。

如果存储的浮点数很小（例如，接近于 0），这会引起向下溢出。Java 会将它近似为 0，所以一般情况下不用考虑向下溢出的问题。

常见错误 3：取整错误

一个取整错误，也称为凑整错误，是在计算得到的数字的近似值和确切的算术值之间的不同。例如，如果保留三位小数位数， $1/3$ 近似等于 0.333，如果保留 7 位，近似值是 0.333 333 3。因为一个变量保存的位数是有限的，因此取整错误是无法避免的。涉及浮点数的计算都是近似的，因为这些数没有以准确的精度来存储。例如：

```
System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);
```

显示的是 0.5000000000000001，而不是 0.5，而

```
System.out.println(1.0 - 0.9);
```

显示的是 0.09999999999999998，而不是 0.1。整数可以精确地存储。因此，整数计算得到的是精确的整数运算结果。

常见错误 4：超出预期的整数除法

Java 使用同样的除法操作符来执行整数和浮点数的除法。当两个操作数是整数时，/ 操作符执行一个整数除法，操作的结果是整数，小数部分被截去。要强制两个整数执行一个浮点数除法时，将其中一个整数转换为浮点数值。例如，下面 a 中的代码显示平均值为 1，而 (b) 中的代码显示平均值为 1.5。

```
int number1 = 1;
int number2 = 2;
double average = (number1 + number2) / 2;
System.out.println(average);
```

a)

```
int number1 = 1;
int number2 = 2;
double average = (number1 + number2) / 2.0;
System.out.println(average);
```

b)

常见陷阱：冗余的输入对象

编程新手经常为每个输入编写代码创建多个输入对象。例如，以下代码读取一个整数和一个双精度值。

```
Scanner input = new Scanner(System.in);
System.out.print("Enter an integer: ");
int v1 = input.nextInt();
```

```
Scanner input1 = new Scanner(System.in);
System.out.print("Enter a double value: ");
double v2 = input1.nextDouble();
```

BAD CODE

代码没有出错，但是效率低。它创建了两个不必要的输入对象，可能会导致一些不易发现的错误。应该如下重写代码：

```
Scanner input = new Scanner(System.in);
System.out.print("Enter an integer: ");
int v1 = input.nextInt();
System.out.print("Enter a double value: ");
double v2 = input.nextDouble();
```

GOOD CODE

复习题

2.34 可以将一个变量声明为 int 类型，之后重新将其声明为 double 类型吗？

- 2.35 什么是整数溢出？浮点数操作会导致溢出吗？
- 2.36 溢出会导致一个运行时错误吗？
- 2.37 什么是取整错误？整数操作会导致取整错误吗？浮点数操作会导致取整错误吗？

关键术语

- | | |
|--|----------------------------------|
| algorithm (算法) | narrowing (of types)((类型的) 缩窄) |
| assignment operator (=)(赋值操作符) | operands (操作数) |
| assignment statement (赋值语句) | operator (操作符) |
| byte type (字节类型) | overflow (上溢) |
| casting (类型转换) | postdecrement (后自减) |
| constant (常量) | postincrement (后自增) |
| data type (数据类型) | predecrement (前自减) |
| declare variables (声明变量) | preincrement (前自增) |
| decrement operator (--)(自减操作符) | primitive data type (基本数据类型) |
| double type (双精度类型) | pseudocode (伪代码) |
| expression (表达式) | requirement specification (需求规范) |
| final keyword (final 关键字) | scope of a variable (变量范围) |
| float type (浮点类型) | short type (短整型类型) |
| floating-point number (浮点数) | specific import (明确导入) |
| identifier (标识符) | system analysis (系统分析) |
| increment operator (++) (自增操作符) | system design (系统设计) |
| incremental development and testing (增量式开发和测试) | underflow (下溢) |
| int type (整数类型) | UNIX epoch (UNIX 时间戳) |
| IPO (输入 - 处理 - 输出) | variable (变量) |
| literal (直接量) | widening (of types)((类型的) 拓宽) |
| long type (长整型类型) | wildcard import (通配符导入) |

本章小结

- 标识符是程序中用于命名诸如变量、常量、方法、类、包之类元素的名称。
- 标识符是由字母、数字、下划线(_)和美元符号(\$)构成的字符序列。标识符必须以字母或下划线(_)开头,不能以数字开头。标识符不能是保留字。标识符可以为任意长度。
- 变量用于存储程序中的数据。声明变量就是告诉编译器变量可以存储何种数据类型。
- 有两种类型的 import 语句:明确导入和通配符导入。明确导入是在 import 语句中指定导入单个类;通配符导入将包中所有的类导入。
- 在 Java 中,等号(=)被用作赋值操作符。
- 方法中声明的变量必须在使用前被赋值。
- 命名常量(或简称为常量)表示从不改变的永久数据。
- 用关键字 final 声明命名常量。
- Java 提供四种整数类型(byte、short、int、long)表示四种不同长度范围的整数。
- Java 提供两种浮点类型(float、double)表示两种不同精度的浮点数。
- Java 提供操作符完成数值运算:加号(+)、减号(-)、乘号(*)、除号(/)和求余符号(%)。
- 整数运算(/)得到的结果是一个整数。

13. Java 表达式中的数值操作符和算术表达式中的使用方法是完全一致的。
14. Java 提供扩展赋值操作符: += (加法赋值)、-= (减法赋值)、*= (乘法赋值)、/= (除法赋值) 以及 %= (求余赋值)。
15. 自增操作符 (++) 和自减操作符 (--) 分别对变量加 1 或减 1。
16. 当计算的表达式中有不同类型的值, Java 会自动地将操作数转换为恰当的类型。
17. 可以使用 (type)value 这样的表示法显式地将数值从一个类型转换到另一个类型。
18. 将一个较小范围类型的变量转换为较大范围类型的变量称为拓宽类型。
19. 将一个较大范围类型的变量转换为较小范围类型的变量称为缩窄类型。
20. 拓宽类型不需要显式转换, 可以自动完成。缩窄类型必须显式完成。
21. 在计算机科学中, 1970 年 1 月 1 日午夜零点称为 UNIX 时间戳。

测试题

在线回答本章测试题, 地址为 www.cs.armstrong.edu/liang/intro10e/quiz.html。

编程练习题

- 🔧 **调试提示:** 编译器通常会给出一个语法错误的原因。如果你不知道如何改正, 将你的程序仔细地、一个字符一个字符地和教材中类似的例子进行对比检查。
- 🔧 **教学注解:** 教师可能会要求你将选中的练习题的分析和设计记录为文档。使用你自己的语言来分析问题, 包括输入、输出, 以及需要计算什么, 并且以伪代码描述如何解决问题。

2.2 ~ 2.12 节

- 2.1 (将摄氏温度转换为华氏温度) 编写程序, 从控制台读入 double 型的摄氏温度, 然后将其转换为华氏温度, 并且显示结果。转换公式如下所示:

$$\text{华氏温度} = (9/5) * \text{摄氏温度} + 32$$

提示: 在 Java 中, 9/5 的结果是 1, 但是 9.0/5 的结果是 1.8。

下面是一个运行示例:

```
Enter a degree in Celsius: 43 ↵
43 Celsius is 109.4 Fahrenheit
```

- 2.2 (计算圆柱体的体积) 编写程序, 读入圆柱体的半径和高, 并使用下列公式计算圆柱的体积:

$$\text{面积} = \text{半径} \times \text{半径} \times \pi$$

$$\text{体积} = \text{面积} \times \text{高}$$

下面是一个运行示例:

```
Enter the radius and length of a cylinder: 5.5 12 ↵
The area is 95.0331
The volume is 1140.4
```

- 2.3 (将英尺转换为米) 编写程序, 读入英尺数, 将其转换为米数并显示结果。一英尺等于 0.305 米。下面是运行示例:

```
Enter a value for feet: 16.5 ↵
16.5 feet is 5.0325 meters
```

- 2.4 (将磅转换为千克) 编写程序, 将磅数转换为千克数。程序提示用户输入磅数, 然后转换成千克并显示结果。一磅等于 0.454 千克。下面是一个运行示例:

```
Enter a number in pounds: 55.5 ↵
55.5 pounds is 25.197 kilograms
```

- *2.5 (财务应用程序: 计算小费) 编写一个程序, 读入一笔费用与酬金率, 计算酬金和总钱数。例如, 如果用户输入 10 作为费用, 15% 作为酬金率, 计算结果显示酬金为 \$1.5, 总费用为 \$11.5。下面是一个运行示例:

```
Enter the subtotal and a gratuity rate: 10 15 --Enter
The gratuity is $1.5 and total is $11.5
```

- **2.6 (求一个整数各位数的和) 编写程序, 读取一个在 0 和 1000 之间的整数, 并将该整数的各位数字相加。例如: 整数是 932, 各位数字之和为 14。
提示: 利用操作符 % 分解数字, 然后使用操作符 / 去掉分解出来的数字。例如: 932%10=2, 932/10=93。下面是一个运行示例:

```
Enter a number between 0 and 1000: 999 --Enter
The sum of the digits is 27
```

- *2.7 (求出年数) 编写程序, 提示用户输入分钟数 (例如十亿) 然后显示这些分钟代表多少年和多少天。为了简化问题, 假设一年有 365 天。下面是一个运行示例:

```
Enter the number of minutes: 1000000000 --Enter
1000000000 minutes is approximately 1902 years and 214 days
```

- *2.8 (当前时间) 程序清单 2-7 给出了显示当前格林威治时间的程序。修改这个程序, 提示用户输入相对于 GMT 的时区偏移量, 然后显示在这个特定时区的时间。下面是一个运行示例:

```
Enter the time zone offset to GMT: -5 --Enter
The current time is 4:50:34
```

- 2.9 (物理: 加速度) 平均加速度定义为速度的变化量除以这个变化所用的时间, 如下式所示:

$$a = \frac{v_1 - v_0}{t}$$

编写程序, 提示用户输入以米/秒为单位的起始速度 v_0 , 以米/秒为单位的终止速度 v_1 , 以及以秒为单位的时间段 t , 最后显示平均加速度。下面是一个运行示例:

```
Enter v0, v1, and t: 5.5 50.9 4.5 --Enter
The average acceleration is 10.0889
```

- 2.10 (科学: 计算能量) 编写程序, 计算将水从初始温度加热到最终温度所需的能量。程序应该提示用户输入水的重量 (以千克为单位), 以及水的初始温度和最终温度。计算能量的公式是:

$$Q = M \times (\text{最终温度} - \text{初始温度}) \times 4184$$

这里的 M 是以千克为单位的水的重量, 温度以摄氏度为单位, 而能量 Q 以焦耳为单位。下面是一个运行示例:

```
Enter the amount of water in kilograms: 55.5 --Enter
Enter the initial temperature: 3.5 --Enter
Enter the final temperature: 10.5 --Enter
The energy needed is 1625484.0
```

- 2.11 (人口统计) 重写编程练习题 1.11, 提示用户输入年数, 然后显示这个年数之后的人口值。将编程练习题 1.11 中的提示用于这个程序。人口数应该类型转换为一个整数。下面是一个运行示例:

```
Enter the number of years: 5 --Enter
The population in 5 years is 325932970
```

- 2.12 (物理: 求出跑道长度) 假设一个飞机的加速度是 a 而起飞速度是 v , 那么可以使用下面的公式计算出飞机起飞所需的最短跑道长度:

$$\text{跑道长度} = \frac{v^2}{2a}$$

编写程序, 提示用户输入以米/秒 (m/s) 为单位的速度 v 和以米/秒的平方 (m/s^2) 为单位的加速度 a , 然后显示最短跑道长度。下面是一个运行示例:

```
Enter speed and acceleration: 60 3.5 ↵
The minimum runway length for this airplane is 514.286
```

- **2.13 (财务应用程序: 复利值) 假设你每月向银行账户存 100 美元, 年利率为 5%, 那么每月利率是 $0.05/12=0.00417$ 。第一个月之后, 账户上的值就变成:

$$100 * (1 + 0.00417) = 100.417$$

第二个月之后, 账户上的值就变成:

$$(100 + 100.417) * (1 + 0.00417) = 201.252$$

第三个月之后, 账户上的值就变成:

$$(100 + 201.252) * (1 + 0.00417) = 302.507$$

依此类推。

编写程序显示六个月后账户上的钱数。(在编程练习题 5.30 中, 你将使用循环来简化这里的代码, 并能显示任何一个月之后的账户值。)

```
Enter the monthly saving amount: 100 ↵
After the sixth month, the account value is $608.81
```

- *2.14 (医疗应用程序: 计算 BMI) 身体质量指数 (BMI) 是对体重的健康测量。它的值可以通过将体重 (以公斤为单位) 除以身高 (以米为单位) 的平方值得到。编写程序, 提示用户输入体重 (以磅为单位) 以及身高 (以英寸为单位), 然后显示 BMI。注意: 一磅是 0.45359237 公斤, 一英寸是 0.0254 米。下面是一个运行示例:

```
Enter weight in pounds: 95.5 ↵
Enter height in inches: 50 ↵
BMI is 26.8573
```

- 2.15 (几何: 两点间距离) 编写程序, 提示用户输入两个点 (x_1, y_1) 和 (x_2, y_2), 然后显示两点间的距离。计算两点间距离的公式是 $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ 。注意: 可以使用 `Math.pow(a, 0.5)` 来计算 \sqrt{a} 。下面是一个运行示例:

```
Enter x1 and y1: 1.5 -3.4 ↵
Enter x2 and y2: 4 5 ↵
The distance between the two points is 8.764131445842194
```

- 2.16 (几何: 六边形面积) 编写程序, 提示用户输入六边形的边长, 然后显示它的面积。计算六边形面积的公式是:

$$\text{面积} = \frac{3\sqrt{3}}{2} s^2$$

这里的 s 就是边长。下面是一个运行示例:

```
Enter the side: 5.5
The area of the hexagon is 78.5918
```

- *2.17 (科学: 风寒温度) 外面到底有多冷? 只有温度是不足以提供答案的, 包括风速、相对湿度以及阳光等其他的因素在确定室外是否寒冷方面都起了很重要的作用。2001年, 国家气象服务(NWS)利用温度和风速计算新的风寒温度, 来衡量寒冷程度。计算公式如下所示:

$$t_{wc} = 35.74 + 0.6215t_a - 35.75v^{0.16} + 0.4275t_a v^{0.16}$$

这里的 t_a 是室外的温度, 以华氏摄氏度为单位, 而 v 是速度, 以每小时英里数为单位。 t_{wc} 是风寒温度。该公式不适用于风速低于 2mph, 或温度在 -58°F 下或 41°F 以上的情况。

编写程序, 提示用户输入在 -58°F 和 41°F 之间的度数, 同时大于或等于 2 的风速, 然后显示风寒温度。使用 `Math.pow(a,b)` 来计算 $v^{0.16}$ 。下面是一个运行示例:

```
Enter the temperature in Fahrenheit between -58°F and 41°F:
5.3
Enter the wind speed (>=2) in miles per hour: 6
The wind chill index is -5.56707
```

- 2.18 (打印表格) 编写程序, 显示下面的表格。将浮点数值类型转换为整数。

a	b	pow(a, b)
1	2	1
2	3	8
3	4	81
4	5	1024
5	6	15625

- *2.19 (几何: 三角形的面积) 编写程序, 提示用户输入三角形的三个点 (x_1, y_1) 、 (x_2, y_2) 和 (x_3, y_3) , 然后显示它的面积。计算三角形面积的公式是:

$$s = (\text{边} 1 + \text{边} 2 + \text{边} 3) / 2$$

$$\text{面积} = \sqrt{s(s - \text{边} 1)(s - \text{边} 2)(s - \text{边} 3)}$$

下面是一个运行示例:

```
Enter three points for a triangle: 1.5 -3.4 4.6 5 9.5 -3.4
The area of the triangle is 33.6
```

2.13 ~ 2.17 节

- *2.20 (财务应用程序: 计算利息) 如果知道收支余额和年利率的百分比, 就可以使用下面的公式计算下个月要支付的利息额:

$$\text{利息额} = \text{收支余额} \times (\text{年利率} / 1200)$$

编写程序, 读取收支余额和年百分利率, 显示两个版本的下月利息。下面是一个运行示例:

```
Enter balance and interest rate (e.g., 3 for 3%): 1000 3.5
The interest is 2.91667
```

- *2.21 (财务应用: 计算未来投资值) 编写程序, 读取投资总额、年利率和年数, 然后使用下面的公式显示未来投资金额:

$$\text{未来投资金额} = \text{投资总额} \times (1 + \text{月利率})^{\text{年数} \times 12}$$

例如: 如果输入的投资金额为 1000, 年利率为 3.25%, 年数为 1, 那么未来投资额为 1032.98。

下面是一个运行示例:

```
Enter investment amount: 1000.56 --Enter
Enter annual interest rate in percentage: 4.25 --Enter
Enter number of years: 1 --Enter
Accumulated value is $1043.92
```

- *2.22 (财务应用: 货币单位) 改写程序清单 2-10, 解决将 `double` 型值转换为 `int` 型值时可能会造成精度损失的问题。输入的输入值是一个整数, 其最后两位代表的是美分币值。例如: 1156 就表示的是 11 美元 56 美分。
- *2.23 (驾驶费用) 编写一个程序, 提示用户输入驾驶的距离、以每加仑多少英里的汽车燃油性能, 以及每加仑的价格, 然后显示旅程的费用。下面是一个运行示例:

```
Enter the driving distance: 900.5 --Enter
Enter miles per gallon: 25.5 --Enter
Enter price per gallon: 3.55 --Enter
The cost of driving is $125.36
```

选 择

教学目标

- 声明 `boolean` 类型变量，并使用关系操作符编写布尔表达式（3.2 节）。
- 使用单分支 `if` 语句实现选择控制（3.3 节）。
- 使用双分支 `if-else` 语句实现选择控制（3.4 节）。
- 使用嵌套的 `if` 语句和多分支 `if` 语句实现选择控制（3.5 节）。
- 避免 `if` 语句中的常见错误和陷阱（3.6 节）。
- 使用 `Math.random()` 方法产生随机数（3.7 节）。
- 使用选择语句编程的各种示例（`SubstractionQuiz`、`BMI`、`ComputeTax`）（3.7 ~ 3.9 节）。
- 使用逻辑操作符（`!`、`&&`、`||` 和 `^`）对条件进行组合（3.10 节）。
- 使用带组合条件的选择语句进行编程（`LeapYear`、`Lottery`）（3.11 ~ 3.12 节）。
- 使用 `switch` 语句实现选择控制（3.13 节）。
- 使用条件操作符书写表达式（3.14 节）。
- 检验控制操作符优先级和结合规律的规则（3.15 节）。
- 应用常用技术进行错误调试（3.16 节）。

3.1 引言

 **要点提示：** 程序可以基于条件决定执行哪些语句。

在程序清单 2-2 中，如果给 `radius` 赋一个负值，程序就会打印一个非法的结果。如果半径是一个负值，是不希望程序计算面积的，那么该如何处理这种情况呢？

和所有高级程序设计语言一样，Java 也提供选择语句：在可选择的执行路径中做出选择的语句。可以用下面的选择语句来替换程序清单 2-2 中的第 12 ~ 17 行：

```
if (radius < 0) {
    System.out.println("Incorrect input");
}
else {
    area = radius * radius * 3.14159;
    System.out.println("Area is " + area);
}
```

选择语句要用到采用布尔表达式的条件。布尔表达式是计算结果为 `Boolean` 值：`true` 或者 `false` 的表达式。本章首先介绍布尔类型和关系操作符。

3.2 boolean 数据类型

 **要点提示：** `boolean` 数据类型声明一个具有值 `true` 或者 `false` 的变量。

如何比较两个值呢？例如：一个半径是大于 0，等于 0，还是小于 0 呢？如表 3-1 所示，Java 提供六种关系操作符（`relational operator`）（也称为比较操作符（`comparison operator`）），用于两个值的比较（假设表中的半径值为 5）。

表 3-1 关系操作符

Java 操作符	数学符号	名称	示例 (半径为 5)	结果
<	<	小于	radius<0	false
<=	≤	小于等于	radius<=0	false
>	>	大于	radius>0	true
>=	≥	大于等于	radius>=0	true
==	=	等于	radius==0	false
!=	≠	不等于	radius!=0	true

 **警告：**相等的关系操作符是两个等号 (==)，而不是一个等号 (=)，后者是指赋值操作符。比较的结果是一个布尔值：true (真) 或 false (假)。例如，下面的语句显示 true：

```
double radius = 1;
System.out.println(radius > 0);
```

具有布尔值的变量称为布尔变量 (boolean variable)，boolean 数据类型用于声明布尔型变量。boolean 型变量可以是以下这两个值中的一个：true 和 false。例如，下述语句将 true 赋值给变量 lightsOn：

```
boolean lightsOn = true;
```

true 和 false 都是直接量，就像 10 这样的数字。它们被当作保留字一样，不能用做程序中的标识符。

假设希望开发一个程序，让一年级学生练习加法。程序随机产生两个一位整数：number1 和 number2，然后显示 “What is 1 + 7?”，如程序清单 3-1 运行示例所示。当学生在输入对话框中输入答案之后，程序显示一个消息，表明答案是真的还是假的。

产生随机数的方法有很多种。现在，使用 System.currentTimeMillis()%10 产生第一个整数，使用 System.currentTimeMillis()*7%10 产生第二个整数。程序清单 3-1 给出该程序。第 5~6 行产生两个数：number1 和 number2。第 14 行获取从用户那里得到的答案。第 18 行使用布尔表达式 number1 + number2 == answer 给答案打分。

程序清单 3-1 AdditionQuiz.java

```
1 import java.util.Scanner;
2
3 public class AdditionQuiz {
4     public static void main(String[] args) {
5         int number1 = (int)(System.currentTimeMillis() % 10);
6         int number2 = (int)(System.currentTimeMillis() / 7 % 10);
7
8         // Create a Scanner
9         Scanner input = new Scanner(System.in);
10
11        System.out.print(
12            "What is " + number1 + " + " + number2 + "? ");
13
14        int answer = input.nextInt();
15
16        System.out.println(
17            number1 + " + " + number2 + " = " + answer + " is " +
18            (number1 + number2 == answer));
19    }
20 }
```

```
What is 1 + 7? 8 
1 + 7 = 8 is true
```

```
What is 4 + 8? 9 
4 + 8 = 9 is false
```

line#	number1	number2	answer	output
5	4			
6		8		
14			9	
16				4 + 8 = 9 is false

复习题

3.1 列出 6 个关系操作符。

3.2 假设 x 等于 1, 给出下列布尔表达式的结果:

```
(x > 0)
(x < 0)
(x != 0)
(x >= 0)
(x != 1)
```

3.3 下面涉及类型转换的变换合法吗? 编写一个测试程序来验证你的结论。

```
boolean b = true;
i = (int)b;

int i = 1;
boolean b = (boolean)i;
```

3.3 if 语句

要点提示: if 语句是一个结构, 允许程序确定执行的路径。

前面的程序显示像 “6 + 2 = 7 is false” 这样的消息。如果希望显示的消息是 “6 + 2 = 7 is incorrect”, 那么必须使用条件语句实现这个小的改变。

Java 有几种类型的选择语句: 单分支 if 语句、双分支 if-else 语句、嵌套 if 语句、多分支 if-else 语句、switch 语句和条件表达式。

单分支 if 语句是指当且仅当条件为 true 时执行一个动作。单分支 if 语句的语法如下:

```
if (布尔表达式) {
    语句 (组);
}
```

图 3-1a 所示的流程图展示 Java 如何执行 if 语句的语法。流程图是描述算法或者过程的图, 以各种盒子显示步骤, 并且通过箭头连接给出次序。处理操作显示在这些盒子中, 连接它们的箭头代表控制流程。菱形的盒子表示一个布尔类型的条件, 矩形盒子代表语句。

如果布尔表达式计算的结果为 true, 则执行块内语句。作为例子, 看看下面的代码:

```
if (radius >= 0) {
    area = radius * radius * PI;
    System.out.println("The area for the circle of radius " +
        radius + " is " + area);
}
```

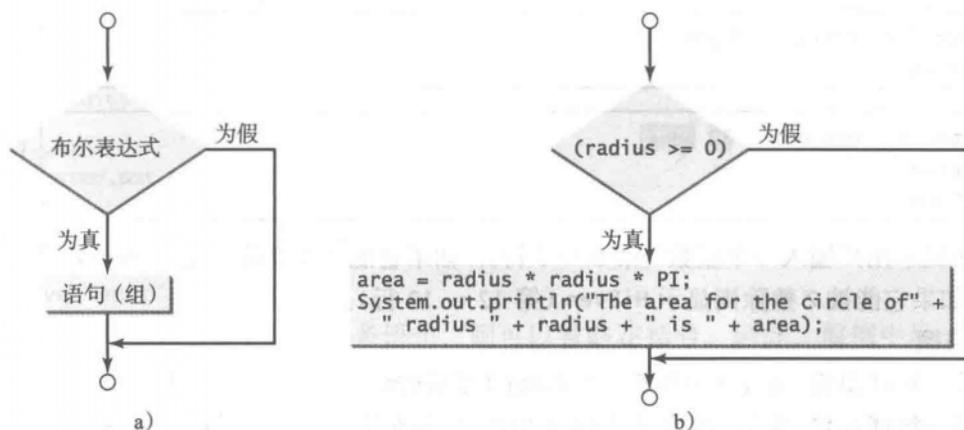


图 3-1 if 语句在布尔表达式计算结果为 true 的情况下执行语句组

上述语句的流程图参见图 3-1b。如果半径 `radius` 的值大于等于 0，则计算面积 `area` 并显示其结果；否则，不执行块内的两条语句。

布尔表达式应该用括号括住。例如：下面图 a 中的代码是错误的。应该将它改为如图 b 所示。

```
if i > 0 {
    System.out.println("i is positive");
}
```

a) 错误的

```
if (i > 0) {
    System.out.println("i is positive");
}
```

b) 正确的

如果花括号内只有一条语句，则可以省略花括号。例如：下面两个语句是等价的。

```
if (i > 0) {
    System.out.println("i is positive");
}
```

a)

等价

```
if (i > 0)
    System.out.println("i is positive");
```

b)

注意：省略括号可以让代码更加简短，但是容易产生错误。当你返回去修改略去代码的时候，容易忘记加上括号。这是一个常犯的错误。

程序清单 3-2 给出一个程序，提示用户输入一个整数。如果该数字是 5 的倍数，打印 HiFive。如果该数字能被 2 整除，打印 HiEven。

程序清单 3-2 SimpleIfDemo.java

```
1 import java.util.Scanner;
2
3 public class SimpleIfDemo {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6         System.out.println("Enter an integer: ");
7         int number = input.nextInt();
8
9         if (number % 5 == 0)
10            System.out.println("HiFive");
11
12        if (number % 2 == 0)
13            System.out.println("HiEven");
14    }
15 }
```

```
Enter an integer: 4 [Enter]
HiEven
```

```
Enter an integer: 30 [Enter]
HiFive
HiEven
```

程序提示用户输入一个整数(第6~7行),如果它能被5整除就显示 HiFive(第9~10行),而如果它能被2整除则显示 HiEven(第12~13行)。

复习题

- 3.4 编写一个 if 语句,在 y 大于等于 0 的时候将 1 赋值给 x。
 3.5 编写一个 if 语句,如果 score 大于 90 则增加 3% 的支付。

3.4 双分支 if-else 语句

要点提示: if-else 语句根据条件是真或者是假,决定执行的路径。

当指定条件为 true 时单分支 if 语句执行一个操作。而当条件为 false 时什么也不干。但是,如果你希望在条件为 false 时也能执行一些动作,该怎么办呢?你可以使用双分支 if 语句。根据条件为 true 或 false,双分支 if 语句可以指定不同的操作。

下面是双分支 if-else 语句的语法:

```
if (布尔表达式) {
    布尔表达式为真时执行的语句(组);
}
else{
    布尔表达式为假时执行的语句(组);
}
```

语句的流程图如图 3-2 所示。

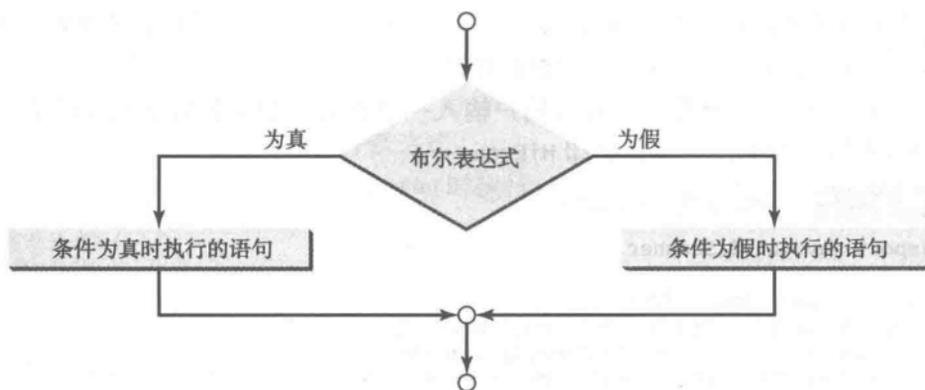


图 3-2 若布尔表达式计算结果为 true, if-else 语句运行 true 情况下的语句组; 否则,运行 false 情况下的语句组

如果布尔表达式的计算结果为 true,则执行条件为 true 时该执行的语句;否则,执行条件为 false 时该执行的语句。例如,考虑下面的代码:

```

if (radius >= 0) {
    area = radius * radius * PI;
    System.out.println("The area for the circle of radius " +
        radius + " is " + area);
}
else {
    System.out.println("Negative input");
}

```

若 $radius \geq 0$ 为 true, 则计算并显示 area; 如果 $radius \geq 0$ 为 false, 则打印信息 "Negative input".

通常, 如果花括号中只有一条语句, 则可以省略花括号。因此, 前例中用于括住语句 `System.out.println("Negative input")` 的花括号可以省略。

这里还有另外一个使用 if-else 语句的例子。这个例子检测一个数是奇数还是偶数, 如下所示:

```

if (number % 2 == 0)
    System.out.println(number + " is even.");
else
    System.out.println(number + " is odd.");

```

复习题

3.6 编写一个 if 语句, 如果 score 大于 90 则增加 3% 的支付, 否则则增加 1% 的支付。

3.7 如果 number 是 30, a 和 b 中的代码输出是什么? 如果 number 是 35 呢?

```

if (number % 2 == 0)
    System.out.println(number + " is even.");
System.out.println(number + " is odd.");

```

a)

```

if (number % 2 == 0)
    System.out.println(number + " is even.");
else
    System.out.println(number + " is odd.");

```

b)

3.5 嵌套的 if 语句和多分支 if-else 语句

要点提示: if 语句可以在另外一个 if 语句中, 形成嵌套的 if 语句。

if 或 if-else 语句中的语句可以是任意合法的 Java 语句, 甚至可以是其他的 if 或 if-else 语句。内层 if 语句称为是嵌套在外层 if 语句里的。内层 if 语句还可以包含其他的 if 语句; 事实上, 对嵌套的深度没有限制。例如, 下面就是一个嵌套的 if 语句:

```

if (i > k) {
    if (j > k)
        System.out.println("i and j are greater than k");
}
else
    System.out.println("i is less than or equal to k");

```

语句 `if(j>k)` 被嵌套在语句 `if(i>k)` 内。

嵌套的 if 语句可用于实现多重选择。例如: 图 3-3a 中所给出的语句使用了多重选择, 根据分数给变量 grade 赋一个用字母表示的级别。

这个 if 语句的执行过程如图 3-4 所示, 测试第一个条件 ($score \geq 90.0$), 如果它为 true, 级别就变成 'A'。如果它为 false, 就测试第二个条件 ($score \geq 80.0$)。如果第二个条件为 true, 级别就变成 'B'。如果第二个条件为 false, 则会继续测试第三个和剩余的条件 (如果有必要的话), 直到遇到满足的条件, 或者所有条件都为 false。如果所有条件都为 false, 级别就变成 'F'。注意: 只有在前面的所有条件都为 false 时才测试下一个条件。

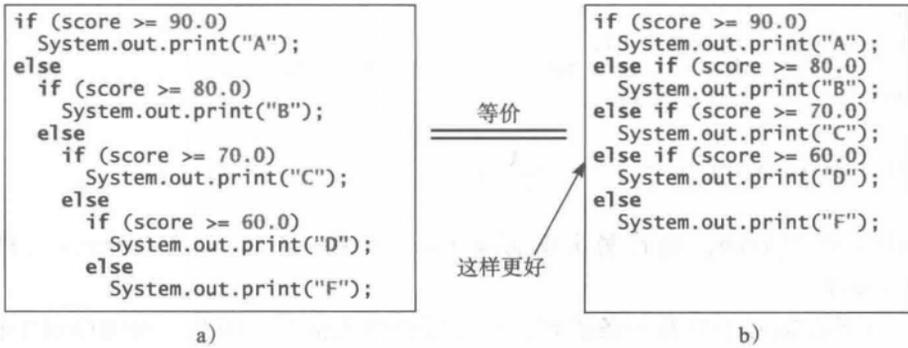


图 3-3 推荐使用如图 3-3b 中所示的多分支 if-else 语句格式

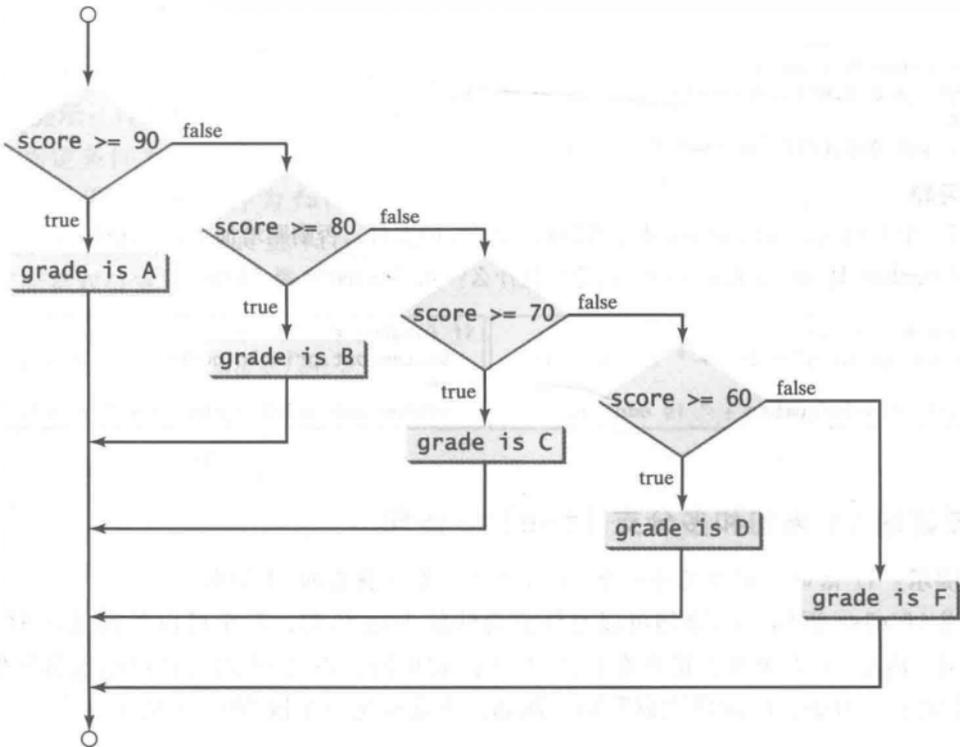


图 3-4 可以使用多分支 if-else 语句来给出一个分数

图 3-3a 中的 if 语句等价于图 3-3b 中的 if 语句。事实上，图 3-3b 是推荐使用的多重选择 if 语句的书写风格。这种风格，称为多分支 if-else 语句，可以避免深度缩进，并使程序易于阅读。

复习题

3.8 假设 $x=3$ 以及 $y=2$ ，如果下面代码有输出的话，请给出。如果 $x=3$ 并且 $y=4$ ，结果是什么呢？如果 $x=2$ 并且 $y=2$ ，结果是什么呢？绘制代码的流程图。

```

if (x > 2) {
    if (y > 2) {
        z = x + y;
        System.out.println("z is " + z);
    }
}

```

```
else
    System.out.println("x is " + x);
```

3.9 假设 $x=2$ 以及 $y=3$ ，如果下面代码有输出的话，请给出。如果 $x=3$ 并且 $y=2$ ，结果是什么呢？如果 $x=3$ 并且 $y=3$ ，结果是什么呢？

```
if (x > 2)
    if (y > 2) {
        int z = x + y;
        System.out.println("z is " + z);
    }
else
    System.out.println("x is " + x);
```

3.10 下面代码中有什么错误？

```
if (score >= 60.0)
    System.out.println("D");
else if (score >= 70.0)
    System.out.println("C");
else if (score >= 80.0)
    System.out.println("B");
else if (score >= 90.0)
    System.out.println("A");
else
    System.out.println("F");
```

3.6 常见错误和陷阱

 **要点提示：**忘记必要的括号，在错误的地方结束 if 语句，将 `==` 错当作 `=` 使用，悬空 else 分支，是选择语句中常见的错误。if-else 语句中重复的语句，以及测试双精度值的相等是常见的陷阱。

以下错误是编程新手经常会犯的错误。

常见错误 1：忘记必要的括号

如果块中只有一条语句，就可以忽略花括号。但是，当需要用花括号将多条语句括在一起时，忘记花括号是一个常见的程序设计错误。如果通过在没有花括号的 if 语句中添加一条新语句来修改代码，就必须插入花括号。例如：下面图 a 中的代码是错误的。应该用花括号将多个语句放在一起，如图 b 所示。

```
if (radius >= 0)
    area = radius * radius * PI;
    System.out.println("The area "
        + " is " + area);
```

a) 错误的

```
if (radius >= 0) {
    area = radius * radius * PI;
    System.out.println("The area "
        + " is " + area);
}
```

b) 正确的

常见错误 2：在 if 行出现错误的分号

如下面的图 a 中所示，在 if 行加上了一个分号，这是一个常见错误。

```
if (radius >= 0);
{
    area = radius * radius * PI;
    System.out.println("The area "
        + " is " + area);
}
```

a)

逻辑错误

等价于

```
if (radius >= 0) {};
{
    area = radius * radius * PI;
    System.out.println("The area "
        + " is " + area);
}
```

b)

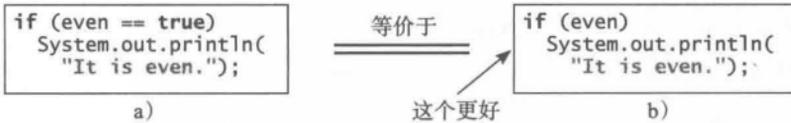
空的块

这个错误是很难发现的，因为它既不是编译错误也不是运行错误，而是一个逻辑错误。图 a 中的代码等价于一个带空块的图 b 中的代码。

当使用换行块风格时，经常会出现这个错误。所以使用行尾块风格可帮助防止出现此类错误。

常见错误 3：对布尔值的冗余测试

为了检测测试条件中的布尔型变量是 true 还是 false，像图 a 中的代码这样使用相等比较操作符是多余的：



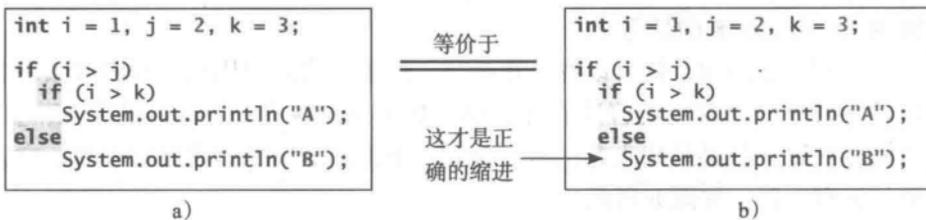
比较好的替代方法就是直接测试布尔变量，如图 b 所示。这么做的另一个原因就是避免出现难以发现的错误。使用 = 操作符而不是 == 操作符去比较测试条件中的两项是否相等是一个常见错误。它可能会导致出现下面的错误语句：

```
if (even = true)
    System.out.println("It is even.");
```

这条语句没有编译错误。它给 even 赋值 true，这样 even 永远都是 true。

常见错误 4：悬空 else 出现的歧义

下面图 a 中的代码有两个 if 子句和一个 else 子句。那么，哪个 if 子句和这个 else 匹配呢？这里的缩进表明 else 子句匹配第一个 if 子句。但是，else 实际匹配的是第二个 if 子句。这种现象就称为悬空 else 歧义 (dangling-else ambiguity)。在同一个块中，else 总是和离它最近的 if 子句匹配。这样，图 a 中的语句就等价于图 b 中的语句。



由于 $(i > j)$ 为假，所以图 a 和图 b 中的语句不打印任何东西。为强制这个 else 匹配第一个 if 子句，必须添加一对花括号。

```
int i = 1, j = 2, k = 3;

if (i > j) {
    if (i > k)
        System.out.println("A");
}
else
    System.out.println("B");
```

这条语句会打印出 B。

常见错误 5：两个浮点数值相等测试

如 2.18 节中常见错误 3 所讨论的，浮点数具有有限的计算精度；涉及浮点数的计算可能引入取整错误。因此，两个浮点数值相等测试并不可靠。比如，你期望代码显示 true，

但是会让人意外的显示 false:

```
double x = 1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1;
System.out.println(x == 0.5);
```

这里, x 并不是精确等于 0.5, 而是 0.5000000000000001。虽然不能依赖于两个浮点数值相等测试, 但是可以通过测试两个数的差距小于某个阈值, 来比较它们是否已经足够接近。也就是, 对于一个非常小的值 ε , 如果 $|x-y| < \varepsilon$, 那么 x 和 y 非常接近。 ε 是一个读为“epsilon”的希腊字母, 常用于表示一个非常小的值。通常, 将 ε 设为 10^{-14} 来比较两个 double 类型的值, 而设为 10^{-7} 来比较两个 float 类型的值。例如, 下面的代码

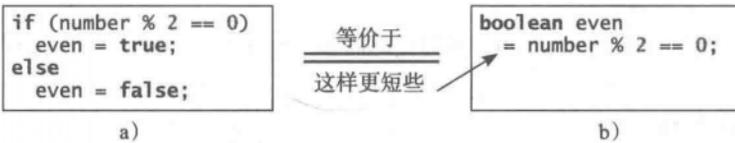
```
final double EPSILON = 1E-14;
double x = 1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1;
if (Math.abs(x - 0.5) < EPSILON)
    System.out.println(x + " is approximately 0.5");
```

将显示 0.5000000000000001 近似等于 0.5。

Math.abs(a) 方法可以用于返回 a 的绝对值。

常见陷阱 1: 简化布尔变量赋值

通常, 编程新手编写将一个条件测试赋给 boolean 变量的代码, 会如 a 中代码:



这没有错, 但是写成 b 中代码形式会更好。

常见陷阱 2: 避免不同情形中的重复代码

编程新手经常会在不同情形中写重复的代码, 这些代码应该写在一处的。例如, 下面高亮的语句是重复的。

```
if (inState) {
    tuition = 5000;
    System.out.println("The tuition is " + tuition);
}
else {
    tuition = 15000;
    System.out.println("The tuition is " + tuition);
}
```

这没有错, 但是写成如下代码形式会更好。

```
if (inState) {
    tuition = 5000;
}
else {
    tuition = 15000;
}
System.out.println("The tuition is " + tuition);
```

新的代码去掉了重复代码, 使得代码更加易于维护, 因为如果打印语句需要修改, 你只需要修改一处地方。

复习题

3.11 以下语句哪些是等价的? 哪些是合理缩进的?

```
if (i > 0) if
(j > 0)
x = 0; else
if (k > 0) y = 0;
else z = 0;
```

a)

```
if (i > 0) {
if (j > 0)
x = 0;
else if (k > 0)
y = 0;
}
else
z = 0;
```

b)

```
if (i > 0)
if (j > 0)
x = 0;
else if (k > 0)
y = 0;
else
z = 0;
```

c)

```
if (i > 0)
if (j > 0)
x = 0;
else if (k > 0)
y = 0;
else
z = 0;
```

d)

3.12 使用布尔表达式重写以下语句:

```
if (count % 10 == 0)
newLine = true;
else
newLine = false;
```

3.13 下列语句正确吗? 哪个更好?

```
if (age < 16)
System.out.println
("Cannot get a driver's license");
if (age >= 16)
System.out.println
("Can get a driver's license");
```

a)

```
if (age < 16)
System.out.println
("Cannot get a driver's license");
else
System.out.println
("Can get a driver's license");
```

b)

3.14 如果 number 值为 14、15 或者 30, 下列代码的输出是什么?

```
if (number % 2 == 0)
System.out.println
(number + " is even");
if (number % 5 == 0)
System.out.println
(number + " is multiple of 5");
```

a)

```
if (number % 2 == 0)
System.out.println
(number + " is even");
else if (number % 5 == 0)
System.out.println
(number + " is multiple of 5");
```

b)

3.7 产生随机数

 **要点提示:** 你可以使用 `Math.random()` 来获得一个 0.0 到 1.0 之间的随机 double 值, 不包括 1.0。

假设你想开发一个让一年级学生练习减法的程序。程序随机产生两个一位整数: `number1` 和 `number2`, 且满足 `number1 >= number2`。程序向学生显示问题, 例如, “What is 9-2?”。当学生输入答案之后, 程序会显示一个消息表明该答案是否正确。

前面的程序使用 `Systems.currentTimeMillis()` 产生两个随机数。更好的方法是使用 `Math` 类中的 `random()` 方法。调用这个方法会返回一个双精度的随机值 `d` 且满足 $0.0 \leq d < 1.0$ 。这样, `(int)(Math.random()*10)` 会返回一个随机的一位整数 (即 0 到 9 之间的数)。

程序可以如下工作:

- 1) 产生两个一位整数 `number1` 和 `number2`。
- 2) 如果 `number1 < number2`, 交换 `number1` 和 `number2`。
- 3) 提示学生回答 “what is number1-number2?”。
- 4) 检查学生的答案并且显示该答案是否正确。

完整的程序如程序清单 3-3 所示。

程序清单 3-3 SubtractionQuiz.java

```

1  import java.util.Scanner;
2
3  public class SubtractionQuiz {
4      public static void main(String[] args) {
5          // 1. Generate two random single-digit integers
6          int number1 = (int)(Math.random() * 10);
7          int number2 = (int)(Math.random() * 10);
8
9          // 2. If number1 < number2, swap number1 with number2
10         if (number1 < number2) {
11             int temp = number1;
12             number1 = number2;
13             number2 = temp;
14         }
15
16         // 3. Prompt the student to answer "What is number1 - number2?"
17         System.out.print
18         ("What is " + number1 + " - " + number2 + "? ");
19         Scanner input = new Scanner(System.in);
20         int answer = input.nextInt();
21
22         // 4. Grade the answer and display the result
23         if (number1 - number2 == answer)
24             System.out.println("You are correct!");
25         else {
26             System.out.println("Your answer is wrong.");
27             System.out.println(number1 + " - " + number2 +
28                 " should be " + (number1 - number2));
29         }
30     }
31 }

```

What is 6 - 6? 0 Enter
You are correct!

What is 9 - 2? 5 Enter
Your answer is wrong
9 - 2 is 7

line#	number1	number2	temp	answer	output
6	2				
7		9			
11			2		
12	9				
13		2			
20				5	
26					Your answer is wrong 9 - 2 should be 7

为了交换变量 `number1` 和 `number2`, 首先要使用一个临时变量 `temp` (第 11 行) 存储 `number1` 的值。将 `number2` 的值赋值给 `number1` (第 12 行), 然后将 `temp` 的值赋给 `number2` (第 13 行)。

复习题

3.15 以下哪些是调用 `Math.random()` 的可能输出?

323.4, 0.5, 34, 1.0, 0.0, 0.234

- 3.16 a. 如何产生一个随机的整数 i , 使得 $0 \leq i < 20$?
- b. 如何产生一个随机的整数 i , 使得 $10 \leq i < 20$?
- c. 如何产生一个随机的整数 i , 使得 $0 \leq i \leq 50$?
- d. 编写一个表达式, 随机返回 0 或者 1。

3.8 示例学习: 计算身体质量指数

 **要点提示:** 你可以使用嵌套的 `if` 语句来编写程序, 计算身体质量指数。

身体质量指数 (BMI) 是关于体重指标的健康测量。可以通过以千克为单位的体重除以以米为单位的身高的平方, 得到 BMI 的值。针对 20 岁及以上年龄的人群, 他们的 BMI 值的说明如右表所示:

BMI	说明
$BMI < 18.5$	偏瘦
$18.5 \leq BMI < 25.0$	正常
$25.0 \leq BMI < 30.0$	超重
$30.0 \leq BMI$	过胖

编写程序, 提示用户输入以英镑为单位的体重, 以及以英寸为单位的身高, 然后显示 BMI。注意: 一磅是 0.45359237 千克, 而一英寸是 0.0254 米。程序清单 3-4 给出这个程序。

程序清单 3-4 ComputeAndInterpretBMI.java

```

1  import java.util.Scanner;
2
3  public class ComputeAndInterpretBMI {
4      public static void main(String[] args) {
5          Scanner input = new Scanner(System.in);
6
7          // Prompt the user to enter weight in pounds
8          System.out.print("Enter weight in pounds: ");
9          double weight = input.nextDouble();
10
11         // Prompt the user to enter height in inches
12         System.out.print("Enter height in inches: ");
13         double height = input.nextDouble();
14
15         final double KILOGRAMS_PER_POUND = 0.45359237; // Constant
16         final double METERS_PER_INCH = 0.0254; // Constant
17
18         // Compute BMI
19         double weightInKilograms = weight * KILOGRAMS_PER_POUND;
20         double heightInMeters = height * METERS_PER_INCH;
21         double bmi = weightInKilograms /
22             (heightInMeters * heightInMeters);
23
24         // Display result
25         System.out.println("BMI is " + bmi);
26         if (bmi < 18.5)
27             System.out.println("Underweight");
28         else if (bmi < 25)
29             System.out.println("Normal");
30         else if (bmi < 30)
31             System.out.println("Overweight");
32         else
33             System.out.println("Obese");
34     }
35 }
```

```

Enter weight in pounds: 146 --Enter
Enter height in inches: 70 --Enter
BMI is 20.948603801493316
Normal

```

line#	weight	height	weightInKilograms	heightInMeters	bmi	output
9	146					
13		70				
19			66.22448602			
20				1.778		
21					20.9486	
25						BMI is 20.95
31						Normal

第 15 ~ 16 行定义两个常量 KILOGRAMS_PER_POUND 和 METERS_PER_INCH。这里使用常量可以使程序易于阅读。

你应该测试覆盖 BMI 所有可能情况的输入，保证程序对于所有情况都是工作的。

3.9 示例学习：计算税率

 **要点提示：**你可以使用嵌套的 if 语句来计算税率。

美国国家联邦个人收入所得税是基于纳税人登记的身份和可征税收入计算的。纳税人登记的身份有四种：单身纳税人、已婚共同纳税人、已婚单独纳税人和家庭户主纳税人。税率会随年变化。表 3-2 给出 2009 年的税率。也就是说，如果你是单身纳税人，可征税收入为 10 000 美元，那么可征税收入的前 8350 美元的税率为 10%，而剩下的 1650 美元的税率为 15%。所以，你该付的税金为 1082.5 美元。

表 3-2 2009 年美国国家联邦个人收入所得税税率表

税率	单身纳税人	已婚共同纳税人或证实的鳏寡	已婚单独纳税人	家庭户主纳税人
10%	\$0 ~ \$8350	\$0 ~ \$16 700	\$0 ~ \$8350	\$0 ~ \$11 950
15%	\$8351 ~ \$33 950	\$16 701 ~ \$67 900	\$8351 ~ \$33 950	\$11 951 ~ \$45 500
25%	\$33 951 ~ \$52 250	\$67 901 ~ \$137 050	\$33 951 ~ \$68 525	\$45 501 ~ \$117 450
28%	\$82 251 ~ \$171 550	\$137 051 ~ \$208 850	\$68 525 ~ \$104 425	\$117 451 ~ \$190 200
33%	\$171 551 ~ \$372 950	\$208 851 ~ \$372 950	\$104 426 ~ \$186 475	\$190 201 ~ \$372 950
35%	\$372 951+	\$372 951+	\$186 476+	\$372 951+

你将要编写一个程序来计算个人收入税。程序应该提示用户输入登记的身份以及可征税收入，然后计算出税款。输入 0 表示单身纳税人，1 表示已婚共同纳税人，2 为已婚单独纳税人，3 为家庭户主纳税人。

程序要计算基于登记身份的可征税收入。登记的身份可以使用 if 语句来决定，如下所示：

```

if (status == 0) {
    // Compute tax for single filers
}
else if (status == 1) {
    // Compute tax for married filing jointly or qualifying widow(er)
}

```

```

}
else if (status == 2) {
    // Compute tax for married filing separately
}
else if (status == 3) {
    // Compute tax for head of household
}
else {
    // Display wrong status
}

```

对每个登记的身份都有六种税率。每个税率应用于某个可征税收入范围内。例如：对于有可征税收入 400 000 美元的单身登记人来说，8350 美元的税率是 10%，从 8350 到 33 950 之间税率为 15%，从 33 950 到 82 250 之间税率是 25%，从 82 250 到 171 550 之间税率是 28%，从 171 550 到 372 950 之间税率是 33% 而从 372 950 到 400 000 之间税率是 35%。

程序清单 3-5 给出计算单身纳税人税款的解决方案，完整的解决方案留作练习。

程序清单 3-5 ComputeTax.java

```

1 import java.util.Scanner;
2
3 public class ComputeTax {
4     public static void main(String[] args) {
5         // Create a Scanner
6         Scanner input = new Scanner(System.in);
7
8         // Prompt the user to enter filing status
9         System.out.print("(0-single filer, 1-married jointly or " +
10            "qualifying widow(er), 2-married separately, 3-head of " +
11            "household) Enter the filing status: ");
12
13         int status = input.nextInt();
14
15         // Prompt the user to enter taxable income
16         System.out.print("Enter the taxable income: ");
17         double income = input.nextDouble();
18
19         // Compute tax
20         double tax = 0;
21
22         if (status == 0) { // Compute tax for single filers
23             if (income <= 8350)
24                 tax = income * 0.10;
25             else if (income <= 33950)
26                 tax = 8350 * 0.10 + (income - 8350) * 0.15;
27             else if (income <= 82250)
28                 tax = 8350 * 0.10 + (33950 - 8350) * 0.15 +
29                     (income - 33950) * 0.25;
30             else if (income <= 171550)
31                 tax = 8350 * 0.10 + (33950 - 8350) * 0.15 +
32                     (82250 - 33950) * 0.25 + (income - 82250) * 0.28;
33             else if (income <= 372950)
34                 tax = 8350 * 0.10 + (33950 - 8350) * 0.15 +
35                     (82250 - 33950) * 0.25 + (171550 - 82250) * 0.28 +
36                     (income - 171550) * 0.33;
37             else
38                 tax = 8350 * 0.10 + (33950 - 8350) * 0.15 +
39                     (82250 - 33950) * 0.25 + (171550 - 82250) * 0.28 +
40                     (372950 - 171550) * 0.33 + (income - 372950) * 0.35;
41         }
42         else if (status == 1) { // Left as an exercise
43             // Compute tax for married file jointly or qualifying widow(er)

```

```

44     }
45     else if (status == 2) { // Compute tax for married separately
46         // Left as an exercise
47     }
48     else if (status == 3) { // Compute tax for head of household
49         // Left as an exercise
50     }
51     else {
52         System.out.println("Error: invalid status");
53         System.exit(1);
54     }
55
56     // Display the result
57     System.out.println("Tax is " + (int)(tax * 100) / 100.0);
58 }
59 }

```

```

(0-single filer, 1-married jointly or qualifying widow(er),
2-married separately, 3-head of household)
Enter the filing status: 0
Enter the taxable income: 400000
Tax is 117683.5

```

line#	status	income	tax	output
13	0			
17		400000		
20			0	
38			117683.5	
57				Tax is 117683.5

这个程序接收纳税人身份和可征税收入。多分支选择 if-else 语句 (第 22、42、45、48 和 51 行) 判断登记人的身份, 并根据登记身份计算税款。

System.exit(status) (第 53 行) 是在 System 类中定义的, 调用这个方法可以终止程序。参数 status 为 0 表明程序正常结束。一个非 0 的状态代码表示非正常结束。

第 20 行赋初始值 0 给 tax。如果 tax 没有初值, 就会出现一个编译错误。因为所有其他给 tax 赋值的语句都在 if 语句中, 编译器认为这些语句可能不会执行, 因此会报告一个编译错误。

为了测试程序, 应该提供覆盖所有情况的输入。对这个程序而言, 输入应该涵盖所有的身份 (0、1、2、3)。针对每一种身份, 应对 6 个范围中的每种情况测试税款。这样, 总共会有 24 种情况。

提示: 对所有的程序都应该先编写少量代码然后进行测试, 之后再继续添加更多的代码。

这个过程称为递进式开发和测试 (incremental development and testing)。这种方法使得调试变得更加容易, 因为错误很可能就在你刚刚添加进去的新代码中。

复习题

3.17 下列两个语句等价吗?

```

if (income <= 10000)
    tax = income * 0.1;
else if (income <= 20000)
    tax = 1000 +
        (income - 10000) * 0.15;

```

```

if (income <= 10000)
    tax = income * 0.1;
else if (income > 10000 &&
        income <= 20000)
    tax = 1000 +
        (income - 10000) * 0.15;

```

3.10 逻辑操作符

🔑 要点提示：逻辑操作符 !、&&、|| 和 ^ 可以用于产生复合布尔表达式。

有时候，是否执行一条语句是由几个条件的组合来决定的。可以使用逻辑操作符组合这些条件。逻辑操作符 (logical operator) 也称为布尔操作符 (boolean operator)，是对布尔值进行的运算，它会创建新的布尔值。表 3-3 列出了布尔操作符清单。表 3-4 定义了非操作符 (!)。非操作符 (!) 对 true 取反是 false，而 false 取反之后则是 true。表 3-5 定义了与操作 (&&)。当且仅当两个操作数都为 true 时，这两个布尔型操作数的与 (&&) 为 true。表 3-6 定义了或操作符 (||)，当至少有一个操作数为 true 时，两个布尔型操作数的或 (||) 为 true。表 3-7 定义了异或操作符 (^)。当且仅当两个操作数具有不同的布尔值时，两个布尔型操作数的异或 (^) 才为 true。注意， $p1 \wedge p2$ 等同于 $p1 != p2$ 。

表 3-3 布尔操作符

操作符	名称	说明
!	非	逻辑非
&&	与	逻辑与
	或	逻辑或
^	异或	逻辑异或

表 3-4 操作符 ! 的真值表

p	!p	举例 (假设 age=24, weight=140)
true	false	!(age>18) 为 false, 因为 (age>18) 为 true
false	true	!(weight==150) 为 true, 因为 (weight==150) 为 false

表 3-5 操作符 && 的真值表

p1	p2	p1&& p2	举例 (假设 age=24, weight=140)
false	false	false	
false	true	false	(age>28)&&(weight<=140) 为 false, 因为 (age>28) 为 false
true	false	false	
true	true	true	(age>18)&&(weight>=140) 为 true, 因为 (age>18) 和 (weight>=140) 都为 true

表 3-6 或操作符 || 的真值表

p1	p2	p1 p2	举例 (假设 age=24, weight=140)
false	false	false	(age>34) (weight>150) 为 false, 因为 (age>34) 和 (weight>150) 都为 false
false	true	true	
true	false	true	(age>18) (weight<140) 为 true, 因为 (age>18) 为 true
true	true	true	

表 3-7 异或操作符 ^ 的真值表

p1	p2	p1^ p2	举例 (假设 age=24, weight=140)
false	false	false	(age>34)^(weight>140) 为 false, 因为 (age>34) 和 (weight>140) 都为 false
false	true	true	(age>34)^(weight>=140) 为 true, 因为 (age>34) 为 false, 但是 (weight>=140) 为 true
true	false	true	
true	true	false	

程序清单 3-6 给出的程序检验一个数是否能同时被 2 和 3 整除，是否被 2 或 3 整除，是否只能被 2 或 3 两者中的一个整除。

程序清单 3-6 TestBooleanOperators.java

```

1 import java.util.Scanner;
2
3 public class TestBooleanOperators {
4     public static void main(String[] args) {
5         // Create a Scanner
6         Scanner input = new Scanner(System.in);
7
8         // Receive an input
9         System.out.print("Enter an integer: ");
10        int number = input.nextInt();
11
12        if (number % 2 == 0 && number % 3 == 0)
13            System.out.println(number + " is divisible by 2 and 3.");
14
15        if (number % 2 == 0 || number % 3 == 0)
16            System.out.println(number + " is divisible by 2 or 3.");
17
18        if (number % 2 == 0 ^ number % 3 == 0)
19            System.out.println(number +
20                " is divisible by 2 or 3, but not both.");
21    }
22 }

```

```

Enter an integer: 4  Enter
4 is divisible by 2 or 3.
4 is divisible by 2 or 3, but not both.

```

```

Enter an integer: 18  Enter
18 is divisible by 2 and 3.
18 is divisible by 2 or 3.

```

(`number%2==0&&number%3==0`) (第 12 行) 检验一个数是否能被 2 和 3 整除。(`number%2==0||number%3==0`) (第 15 行) 检验一个数是否能被 2 或 3 整除。(`number%2==0^ number%3==0`) (第 18 行) 检验一个数是否能被 2 或 3 整除但不能同时被这两者整除。

警告: 从数学的角度看, 表达式 `1<=numberOfDaysInAMonth <=31` 是正确的。但是, 在 Java 中它是错的, 因为 `1<=numberOfDaysInAMonth` 得到的是一个布尔值的结果, 它是不能和 31 进行比较的。这里的两个操作数 (一个布尔值和一个数值) 是不兼容的。正确的 Java 表达式是:

```
(1 <= numberOfDaysInAMonth) && (numberOfDaysInAMonth <= 31)
```

注意: 德模佛定理是以印度出生的英国数学家和逻辑学家奥古斯都·德·模佛来命名的 (1806—1871), 这个定理可以用来简化表达式。定义表述如下:

`!(condition1 && condition2)` 和 `!condition1 || !condition2` 是等价的。

`!(condition1 || condition2)` 和 `!condition1 && !condition2` 是等价的。

例如,

```
!(number % 2 == 0 && number % 3 == 0)
```

可以简化为等价的表达式:

```
(number % 2 != 0 || number % 3 != 0)
```

另外一个例子,

```
!(number == 2 || number == 3)
```

可以写作：

```
number != 2 && number != 3
```

如果操作符 `&&` 的操作数之一为 `false`，那么表达式就是 `false`；如果操作符 `||` 的操作数之一为 `true`，那么表达式就是 `true`。Java 利用这些特性来提高这些操作符的效率。当计算 `p1&&p2` 时，Java 先计算 `p1`，如果 `p1` 为 `true` 再计算 `p2`；如果 `p1` 为 `false`，则不再计算 `p2`。当计算 `p1||p2` 时，Java 先计算 `p1`，如果 `p1` 为 `false` 再计算 `p2`；如果 `p1` 为 `true`，则不再计算 `p2`。在编程术语中，`&&` 和 `||` 被称为短路或者懒惰操作符；Java 也提供了无条件 AND (`&`) 和 OR (`|`) 操作符，更多内容请参见补充材料 III.C。

复习题

3.18 假设 `x` 为 1，给出下列布尔表达式的结果：

```
(true) && (3 > 4)
!(x > 0) && (x > 0)
(x > 0) || (x < 0)
```

```
(x != 0) || (x == 0)
(x >= 0) || (x < 0)
(x != 1) == !(x == 1)
```

3.19 (a) 编写一个布尔表达式满足：若变量 `num` 中存储的数值在 1 到 100 之间时，表达式的值为 `true`。(b) 编写一个布尔表达式满足：若变量 `num` 中存储的数值在 1 到 100 之间，或值为负数时，表达式的值为 `true`。

3.20 (a) 为 $|x-5|<4.5$ 编写一个布尔表达式。(b) 为 $|x-5|>4.5$ 编写一个布尔表达式。

3.21 假设 `x` 和 `y` 都是 `int` 类型，下面哪些是合法的 Java 表达式？

```
x > y > 0
x = y && y
x /= y
x or y
x and y
(x != 0) || (x = 0)
```

3.22 以下两个表达式等同吗？

```
a. x % 2 == 0 && x % 3 == 0
b. x % 6 == 0
```

3.23 如果 `x` 是 45、67 或者 101，表达式 `x>=50&&x<=100` 的值是多少？

3.24 假设运行下面的程序时，从控制台输入的是 2 3 6，那么输出是什么？

```
public class Test {
    public static void main(String[] args) {
        java.util.Scanner input = new java.util.Scanner(System.in);
        double x = input.nextDouble();
        double y = input.nextDouble();
        double z = input.nextDouble();

        System.out.println("(x < y && y < z) is " + (x < y && y < z));
        System.out.println("(x < y || y < z) is " + (x < y || y < z));
        System.out.println("!(x < y) is " + !(x < y));
        System.out.println("(x + y < z) is " + (x + y < z));
        System.out.println("(x + y > z) is " + (x + y > z));
    }
}
```

- 3.25 编写布尔表达式，当年龄 `age` 大于 13 且小于 18 时结果为 `true`。
- 3.26 编写布尔表达式，当体重 `weight` 大于 50 磅或者身高大于 60 英尺时结果为 `true`。
- 3.27 编写布尔表达式，当体重 `weight` 大于 50 磅并且身高大于 60 英尺时结果为 `true`。
- 3.28 编写布尔表达式，当体重 `weight` 大于 50 磅或者身高大于 60 英尺，但是不是同时满足时结果为 `true`。

3.11 示例学习：判定闰年

 **要点提示：**如果某年可以被 4 整除而不能被 100 整除，或者可以被 400 整除，那么这一年就是闰年。

可以使用下面的布尔表达式判定某年是否为闰年：

```
// A leap year is divisible by 4
boolean isLeapYear = (year % 4 == 0);

// A leap year is divisible by 4 but not by 100
isLeapYear = isLeapYear && (year % 100 != 0);

// A leap year is divisible by 4 but not by 100 or divisible by 400
isLeapYear = isLeapYear || (year % 400 == 0);
```

或者可以将这些表达式组合在一起，如下所示：

```
isLeapYear = (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
```

程序清单 3-7 给出的程序让用户输入一个年份，然后判断它是否是闰年。

程序清单 3-7 LeapYear.java

```
1 import java.util.Scanner;
2
3 public class LeapYear {
4     public static void main(String[] args) {
5         // Create a Scanner
6         Scanner input = new Scanner(System.in);
7         System.out.print("Enter a year: ");
8         int year = input.nextInt();
9
10        // Check if the year is a leap year
11        boolean isLeapYear =
12            (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
13
14        // Display the result
15        System.out.println(year + " is a leap year? " + isLeapYear);
16    }
17 }
```

```
Enter a year: 2008 
2008 is a leap year? true
```

```
Enter a year: 1900 
1900 is a leap year? false
```

```
Enter a year: 2002 
2002 is a leap year? false
```

3.12 示例学习：彩票

 **要点提示：**彩票程序涉及产生随机数、比较数字各位，以及运用布尔操作符。

假设你想开发一个玩彩票的游戏，程序随机地产生一个两位数的彩票，提示用户输入一个两位数，然后按照下面的规则判定用户是否能赢：

- 1) 如果用户的输入数匹配彩票的实际顺序，奖金为 10 000 美元。
- 2) 如果用户输入的所有数字匹配彩票的所有数字，奖金为 3000 美元。
- 3) 如果用户输入的一个数字匹配彩票的一个数字，奖金为 1000 美元。

注意，两位数字的位中可能为 0。如果一个数小于 10，我们假设这个数字以 0 开始，从而构建一个两位数。例如，程序中数字 8 被作为 08 处理，数字 0 作为 00 处理。程序清单 3-8 给出完整的程序。

程序清单 3-8 Lottery.java

```

1  import java.util.Scanner;
2
3  public class Lottery {
4      public static void main(String[] args) {
5          // Generate a lottery number
6          int lottery = (int)(Math.random() * 100);
7
8          // Prompt the user to enter a guess
9          Scanner input = new Scanner(System.in);
10         System.out.print("Enter your lottery pick (two digits): ");
11         int guess = input.nextInt();
12
13         // Get digits from lottery
14         int lotteryDigit1 = lottery / 10;
15         int lotteryDigit2 = lottery % 10;
16
17         // Get digits from guess
18         int guessDigit1 = guess / 10;
19         int guessDigit2 = guess % 10;
20
21         System.out.println("The lottery number is " + lottery);
22
23         // Check the guess
24         if (guess == lottery)
25             System.out.println("Exact match: you win $10,000");
26         else if (guessDigit2 == lotteryDigit1
27             && guessDigit1 == lotteryDigit2)
28             System.out.println("Match all digits: you win $3,000");
29         else if (guessDigit1 == lotteryDigit1
30             || guessDigit1 == lotteryDigit2
31             || guessDigit2 == lotteryDigit1
32             || guessDigit2 == lotteryDigit2)
33             System.out.println("Match one digit: you win $1,000");
34         else
35             System.out.println("Sorry, no match");
36     }
37 }

```

```

Enter your lottery pick (two digits): 15 
The lottery number is 15
Exact match: you win $10,000

```

```

Enter your lottery pick (two digits): 45 
The lottery number is 54
Match all digits: you win $3,000

```

```
Enter your lottery pick: 23 
The lottery number is 34
Match one digit: you win $1,000
```

```
Enter your lottery pick: 23 
The lottery number is 14
Sorry: no match
```

line#	6	11	14	15	18	19	33
variable							
lottery	34						
guess		23					
lotteryDigit1			3				
lotteryDigit2				4			
guessDigit1					2		
guessDigit2						3	
Output							Match one digit: you win \$1,000

程序使用 `random()` 方法 (第 6 行) 创建一个彩票, 然后提示用户输入他自己的猜测值 (第 11 行)。注意, 因为 `guess` 是一个两位数, 所以 `guess%10` 能得到 `guess` 的最后一位数, 而 `guess/10` 能得到 `guess` 的第一位数 (第 18 ~ 19 行)。

程序按照以下顺序检测猜测值和彩票:

- 1) 首先检测猜测值是否精确匹配彩票 (第 24 行)。
- 2) 如果没有匹配, 就检测猜测数的逆序是否匹配彩票 (第 26 ~ 27 行)。
- 3) 如果还不匹配, 就检测是否有一个数字在彩票中 (第 29 ~ 32 行)。
- 4) 如果还没有, 就表明都不匹配, 显示 "Sorry, no match" (第 34 ~ 35 行)。

3.13 switch 语句

 **要点提示:** `switch` 语句基于变量或者表达式的值来执行语句。

程序清单 3-5 中的 `if` 语句是根据单独的一个 `true` 或 `false` 条件做出选择的。根据变量 `status` 的值, 会有四种计算税金的情况。为了全面考虑所有的情况, 需要使用嵌套的 `if` 语句。过多地使用嵌套的 `if` 语句会使程序很难阅读。Java 提供 `switch` 语句来有效地处理多重条件的问题。可以使用下述 `switch` 语句替换程序清单 3-5 中的嵌套 `if` 语句:

```
switch (status) {
    case 0: compute tax for single filers;
           break;
    case 1: compute tax for married jointly or qualifying widow(er);
           break;
    case 2: compute tax for married filing separately;
           break;
    case 3: compute tax for head of household;
           break;
    default: System.out.println("Error: invalid status");
            System.exit(1);
}
```

上面的 switch 语句的流程图如图 3-5 所示。

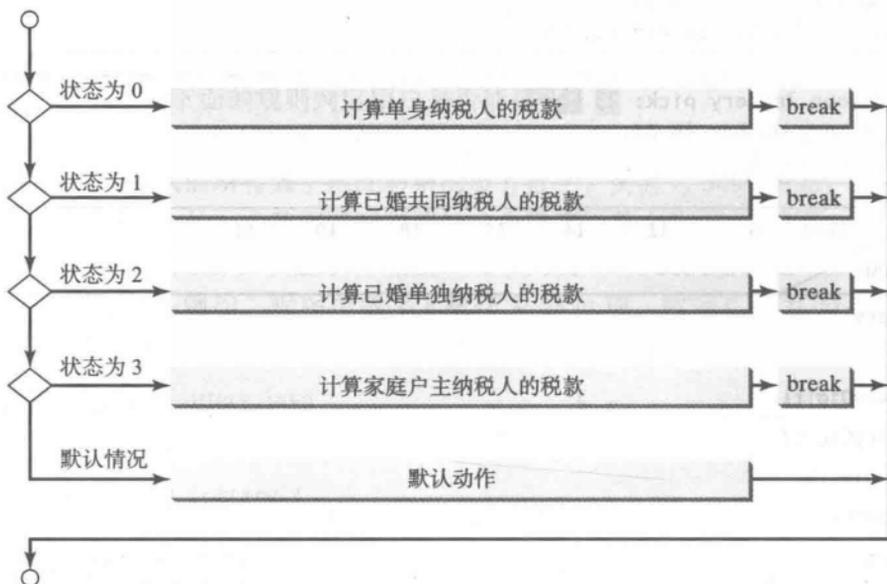


图 3-5 switch 语句检验所有的情况并执行匹配条件时的语句

这条语句依次检查 status 是否能匹配 0、1、2 或 3。如果匹配，就计算相应的税金；如果不匹配，就显示一条消息。下面是 switch 语句的完整语法：

```
switch (switch 表达式) {
    case 值 1: 语句 (组) 1;
                break;
    case 值 2: 语句 (组) 2;
                break;
    ...
    case 值 N: 语句 (组) N;
                break;
    default: 默认情况下执行的语句 (组)
}

```

switch 语句遵从下述规则：

① switch 表达式必须能计算出一个 char、byte、short、int 或者 String 型值，并且必须总是要用括号括住。(char 和 String 类型将在下一章介绍。)

② value1, ..., valueN 必须与 switch 表达式的值具有相同的数据类型。注意：value1, ..., valueN 都是常量表达式，也就是说这里的表达式是不能包含变量的，例如，不允许出现 1+x。

③ 当 switch 表达式的值与 case 语句的值相匹配时，执行从该 case 开始的语句，直到遇到一个 break 语句或到达该 switch 语句的结束。

④ 默认情况 (default) 是可选的，当没有一个给出的 case 与 switch 表达式匹配时，用来执行该操作。

⑤ 关键字 break 是可选的。break 语句会立即终止 switch 语句。

警告：不要忘记在需要的时候使用 break 语句。一旦匹配其中一个 case，就从匹配的

case 处开始执行，直到遇到 break 语句或到达 switch 语句的结束。这种现象称为落空行为 (fall-through behavior)。例如，下列代码为第 1 到 5 天显示 Weekdays，第 0 和第 6 天显示 Weekends。

```
switch (day) {
    case 1:
    case 2:
    case 3:
    case 4:
    case 5: System.out.println("Weekday"); break;
    case 0:
    case 6: System.out.println("Weekend");
}
```

提示：为了避免程序设计错误，提高代码的可维护性，如果故意省略 break，在 case 子句后添加注释是一个好的做法。

现在让我们编写一个程序，为一个给定的年份找出其中国生肖。中国生肖基于 12 年一个周期，每年用一个动物代表——猴 (monkey)、鸡 (rooster)、狗 (dog)、猪 (pig)、鼠 (rat)、牛 (ox)、虎 (tiger)、兔 (rabbit)、龙 (dragon)、蛇 (snake)、马 (horse) 或者羊 (sheep)，如图 3-6 所示。

注意：year % 12 确定生肖。1900 属鼠，因为 1900 % 12 为 4。程序清单 3-9 给出一个程序，提示用户输入一个年份，显示当年的动物。



图 3-6 中国生肖基于 12 年一个周期

程序清单 3-9 ChineseZodiac.java

```
1 import java.util.Scanner;
2
3 public class ChineseZodiac {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6
7         System.out.print("Enter a year: ");
8         int year = input.nextInt();
9
10        switch (year % 12) {
11            case 0: System.out.println("monkey"); break;
12            case 1: System.out.println("rooster"); break;
13            case 2: System.out.println("dog"); break;
14            case 3: System.out.println("pig"); break;
15            case 4: System.out.println("rat"); break;
16            case 5: System.out.println("ox"); break;
```

```

17     case 6: System.out.println("tiger"); break;
18     case 7: System.out.println("rabbit"); break;
19     case 8: System.out.println("dragon"); break;
20     case 9: System.out.println("snake"); break;
21     case 10: System.out.println("horse"); break;
22     case 11: System.out.println("sheep");
23     }
24 }
25 }

```

```

Enter a year: 1963 
rabbit

```

```

Enter a year: 1877 
ox

```

复习题

- 3.29 switch 变量需要什么类型的数据？如果在执行完 case 语句之后没有使用关键字 break，那么下一条要执行的语句是什么？可以把 switch 语句转换成等价的 if 语句吗？或者反过来可以将 if 语句转换成等价的 switch 语句吗？使用 switch 语句的优点有哪些？
- 3.30 执行下列 switch 语句之后，y 是多少？并使用 if-else 重写代码。

```

x = 3; y = 3;
switch (x + 3) {
    case 6: y = 1;
    default: y += 1;
}

```

- 3.31 执行下列 if-else 语句之后，x 是多少？使用 switch 语句重写，并且为新的 switch 语句画出流程图。

```

int x = 1, a = 3;
if (a == 1)
    x += 5;
else if (a == 2)
    x += 10;
else if (a == 3)
    x += 16;
else if (a == 4)
    x += 34;

```

- 3.32 编写 switch 语句，如果 day 是 0、1、2、3、4、5、6，分别显示 Sunday、Monday、Tuesday、Wednesday、Thursday、Friday、Saturday。

3.14 条件表达式

要点提示：条件表达式基于一个条件计算表达式的值。

有时可能需要给有特定条件限制的变量赋值。例如：下面的语句在 x 大于 0 时给 y 赋值 1；当 x 小于等于 0 时给 y 赋值 -1。

```

if (x > 0)
    y = 1;
else
    y = -1;

```

在这个例子中，还可以选择使用如下的条件表达式，也能达到同样的效果。

```

y = (x > 0) ? 1 : -1;

```

条件表达式是一种完全不同的风格，在语句中没有明确出现 if。该语法如下所示：

```
boolean-expression ? expression1 : expression2; (布尔表达式? 表达式1: 表达式2)
```

如果布尔表达式的值为 true，则条件表达式的结果为表达式 expression1；否则，结果为表达式 expression2。

假设希望将 num1 和 num2 中较大的数赋值给 max，可以利用条件表达式，只需编写一条语句：

```
max = (num1 > num2) ? num1 : num2;
```

另外举一个例子，如果 num 是偶数，下面的语句就显示信息 “num is even”；否则显示 “num is odd”：

```
System.out.println((num % 2 == 0) ? "num is even" : "num is odd");
```

如你在这些示例中所见，条件表达式使你可以编写精简的代码。

注意：符号 ? 和 : 在条件表达式中同时出现。它们构成一种条件操作符，因为操作数有三个，所以称为三元操作符 (ternary operator)。它是 Java 中唯一的三元操作符。

复习题

3.33 假设你运行下面程序的时候从控制台输入 2 3 6，将输出什么？

```
public class Test {
    public static void main(String[] args) {
        java.util.Scanner input = new java.util.Scanner(System.in);
        double x = input.nextDouble();
        double y = input.nextDouble();
        double z = input.nextDouble();

        System.out.println((x < y && y < z) ? "sorted" : "not sorted");
    }
}
```

3.34 运用条件操作符重写以下 if 语句。

```
if (ages >= 16)
    ticketPrice = 20;
else
    ticketPrice = 10;
```

3.35 使用 if-else 表达式重写下面的条件表达式。

```
a. score = (x > 10) ? 3 * scale : 4 * scale;
b. tax = (income > 10000) ? income * 0.2 : income * 0.17 + 1000;
c. System.out.println((number % 3 == 0) ? i : j);
```

3.36 编写随机返回 1 或者 -1 的条件表达式。

3.15 操作符的优先级和结合规则

要点提示：操作符的优先级和结合规则确定了操作符计算的顺序。

2.11 节介绍了涉及算术操作符的操作符优先级。本节更加详细地讨论操作符的优先级。假设有这样一个表达式：

```
3 + 4 * 4 > 5 * (4 + 3) - 1 && (4 - 3 > 5)
```

它的值是多少呢？这些操作符的执行顺序是什么呢？

应该首先计算括号中的表达式（括号可以嵌套，在嵌套的情况下，先计算里层括号中的表达式）。当计算没有括号的表达式时，操作符会依照优先级规则和结合规则进行运算。

优先级规则定义了操作符的先后次序，如表 3-8 所示，它包含了目前所学的所有操作符。它们从上到下按优先级递减的方式排列。逻辑操作符的优先级比关系操作符的低，而关系操作符的优先级比算术操作符的低。优先级相同的操作符排在同一行。（Java 操作符及其优先级的完整列表参见附录 C。）

如果优先级相同的操作符相邻，则结合规则（associativity）决定它们的执行顺序。除了赋值操作符之外，所有的二元操作符都是左结合的（left-associative）。例如，由于 + 和 - 的优先级相同并且都是左结合的，所以表达式：

$$a - b + c - d \quad \text{等价于} \quad ((a - b) + c) - d$$

赋值操作符是右结合的（right-associative）。因此，表达式：

$$a = b += c = 5 \quad \text{等价于} \quad a = (b += (c = 5))$$

假设赋值前 a、b 和 c 都是 1，在计算表达式之后，a 变成 6，b 变成 6，而 c 变成 5。注意：左结合对赋值操作符而言是说不通的。

注意：Java 有自己内部计算表达式的方式。Java 计算的结果和它对应的算术计算是一样的。感兴趣的读者可以参考补充材料 III.B，以获得更多关于 Java 的后台是如何计算表达式的讨论。

复习题

3.37 列出布尔操作符的优先级顺序。计算下面的表达式：

```
true || true && false
true && true || false
```

3.38 除 = 之外的所有二元操作符都是左结合的，这个说法是真还是假？

3.39 计算下面的表达式：

```
2 * 2 - 3 > 2 && 4 - 2 > 5
2 * 2 - 3 > 2 || 4 - 2 > 5
```

3.40 ($x > 0 \ \&\& \ x < 10$) 和 ($(x > 0) \ \&\& \ (x < 10)$) 是否一样？($x > 0 \ || \ x < 10$) 和 ($(x > 0) \ || \ (x < 10)$) 是否一样？($x > 0 \ || \ x < 10 \ \&\& \ y < 0$) 和 ($x > 0 \ || \ (x < 10 \ \&\& \ y < 0)$) 是否一样？

3.16 调试

要点提示：调试是在程序中找到和修改错误的过程。

如第 1.10.1 节所提到，因为编译器可以明确指出错误的位置以及出错的原因，所以语法错误是容易发现和纠正的。运行时错误也不难找，因为在程序异常中止时，错误的原因和位

表 3-8 操作符优先级表

优先级	操作符
最高级	var++ 和 var-- (后置操作符)
	+、- (一元加号和一元减号)、++var、--var (前置操作符)
	(type)(类型转换)
	!(非)
	*、/、% (乘法、除法和求余运算)
	+、- (二元加法和减法)
	<、<=、>、>= (比较操作符)
	==、!= (相等操作符)
	^ (异或)
	&& (条件与)
	(条件或)
最低级	=、+=、-=、*=、/=、%= (赋值操作符)

置都会显示在控制台上。然而，查找逻辑错误就很富有挑战性。

逻辑错误也称为臭虫 (bug)。查找和改正错误的过程称为调试 (debugging)。调试的一般途径是采用各种方法逐步缩小程序中 bug 所在的范围。可以手工跟踪 (hand trace) 程序 (即通过读程序找错误)，也可以插入打印语句，显示变量的值或程序的执行流程。这种方法适用于短小、简单的程序。对于庞大、复杂的程序，最有效的调试方法还是使用调试工具。

JDK 包含了一个命令行调试器 jdb，结合一个类名来调用该命令。Jdb 本身也是一个 Java 程序，运行自身的一个 Java 解释器的拷贝。所有的 Java IDE 工具，比如 Eclipse 和 NetBeans 包含集成的调试器。调试器应用让你可以跟踪一个程序的执行。它们因系统而不同，但是都支持以下有用的特征中的多数。

- 一次执行一条语句：调试器允许你一次执行一条语句，从而可以看到每条语句的效果。
- 跟踪进入或者一步运行过一个方法：如果一个方法正在被执行，你可以让调试器跟踪进入方法内部，并且一次执行方法里面的一条语句，或者你也可以让调试器一步运行过整个方法。如果你知道方法是可行的，你应该一次运行过整个的方法。比如，通常都会一步运行过系统提供的方法，比如 `System.out.println`。
- 设置断点：你也可以在一条特定的语句上面设置断点。当遇到一个断点时，你的程序将暂停。你可以设置任意多的断点。当你知道程序错误从什么地方可能开始的时候，断点特别有用。你可以将断点设置在那条语句上，让程序先执行到断点处。
- 显示变量：调试器让你选择多个变量并且显示它们的值。当你跟踪一个程序的时候，变量的内容持续更新。
- 显示调用堆栈：调试器让你跟踪所有的方法调用。当你需要看到程序执行流程的宏观图景的时候，这个特征非常有用。
- 修改变量：一些调试器允许你在调试的过程中修改变量的值。当你希望用不同的示例来测试程序，而又不希望离开调试器的时候，这是非常方便的。

提示：如果你使用诸如 Eclipse 或者 NetBeans 之类的 IDE，请参考配套网站上面的补充材料 II.C 和 II.E。补充材料给出如何使用调试器来追踪程序，以及调试过程是如何帮助你有效学习 Java 的。

关键术语

Boolean expression (布尔表达式)

Boolean data type (boolean 数据类型)

Boolean value (布尔值)

conditional operator (条件操作符)

dangling-else ambiguity (悬空 else 歧义)

debugging (调试)

fall-through behavior (落空行为)

flowchart (流程图)

lazy operator (懒惰操作符)

operator associativity (操作符结合规则)

operator precedence (操作符优先级)

selection statement (选择语句)

short-circuit evaluation (短路运算)

本章小结

1. boolean 类型变量可以存储值 true 或 false。
2. 关系操作符 (<、<=、==、!=、>、>=) 产生一个布尔值。

- 选择语句用于可选择的动作路径的编程。选择语句有以下几种类型：单分支 if 语句、双分支 if-else 语句、嵌套 if 语句、多分支 if-else 语句、switch 语句和条件表达式。
- 各种 if 语句都是基于布尔表达式来控制决定的。根据表达式的值是 true 或 false，这些语句选择两种可能路径中的一种。
- 布尔操作符 &&、||、| 和 ^ 对布尔值和布尔变量进行计算。
- 当对 $p1 \&\& p2$ 求值时，Java 先求 $p1$ 的值，如果 $p1$ 为 true，再对 $p2$ 求值；如果 $p1$ 为 false，就不再对 $p2$ 求值。当对 $p1 || p2$ 求值时，Java 先求 $p1$ 的值，如果 $p1$ 为 false，再对 $p2$ 求值；如果 $p1$ 为 true，就不再对 $p2$ 求值。因此，&& 也称为条件与操作符或短路与操作符，而 || 也称为条件或操作符或短路或操作符。
- switch 语句根据 char、byte、short、int 或者 String 类型的 switch 表达式来进行控制决定。
- 在 switch 语句中，关键字 break 是可选的，但它通常用在每个分支的结尾，以中止执行 switch 语句的剩余部分。如果没有出现 break 语句，则执行接下来的 case 语句。
- 表达式中的操作符按照括号、操作符优先级以及操作符结合规则所确定的次序进行求值。
- 括号用于强制求值的顺序以任何顺序进行。
- 具有更高级优先权的操作符更早地进行操作。对于同样优先级的操作符，它们的结合规则确定操作的顺序。
- 除开赋值操作符的所有二元操作符都是左结合的，赋值操作符是右结合的。

测试题

在线回答本章测试题，地址为 www.cs.armstrong.edu/liang/introl0e/quiz.html。

编程练习题

 **教学提示** 对于每一个练习题，都应在编码之前仔细地分析问题的需求及设计解题策略。

 **调试提示** 在寻求帮助之前，将程序读和解释给自己听。然后通过手动使用几个具有代表性的输入，或者使用某个 IDE 调试器来跟踪程序。通过调试自己编程中的错误，来学习如何编程。

3.2 节

*3.1 (代数：解一元二次方程) 可以使用下面的公式求一元二次方程 $ax^2+bx+c=0$ 的两个根：

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{和} \quad r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

b^2-4ac 称作一元二次方程的判别式。如果它是正值，那么一元二次方程就有两个实数根。如果它为 0，方程式就只有一个根。如果它是负值，方程式无实根。

编写程序，提示用户输入 a 、 b 和 c 的值，并且显示基于判别式的结果。如果这个判别式为正，显示两个根。如果判别式为 0，显示一个根。否则，显示“The equation has no real roots”(该方程式无实数根)。

 **注意：**可以使用 `Math.pow(x, 0.5)` 来计算 \sqrt{x} 。下面是一些运行示例。

```
Enter a, b, c: 1.0 3 1 
The equation has two roots -0.381966 and -2.61803
```

```
Enter a, b, c: 1 2.0 1 
The equation has one root -1
```

```
Enter a, b, c: 1 2 3 
The equation has no real roots
```

3.2 (游戏: 三个数的加法) 程序清单 3-1 中的程序产生两个整数, 并提示用户输入这两个整数的和。修改该程序使之能产生三个一位整数, 然后提示用户输入这三个整数的和。

*3.3 (代数: 求解 2×2 线性方程) 可以使用编程练习题 1.13 中给出的 Cramer 规则解线性方程组:

$$\begin{aligned} ax + by &= e \\ cx + dy &= f \end{aligned} \quad x = \frac{ed - bf}{ad - bc} \quad y = \frac{af - ec}{ad - bc}$$

编写程序, 提示用户输入 a, b, c, d, e 和 f , 然后显示结果。如果 $ad - bc$ 为 0, 报告消息 “The equation has no solution” (方程式无解)。

```
Enter a, b, c, d, e, f: 9.0 4.0 3.0 -5.0 -6.0 -21.0 ↵
x is -2.0 and y is 3.0
```

```
Enter a, b, c, d, e, f: 1.0 2.0 2.0 4.0 4.0 5.0 ↵
The equation has no solution
```

**3.4 (随机月份) 编写一个随机产生 1 和 12 之间整数的程序, 并且根据数字 1, 2, ..., 12 显示相应的英文月份: January, February, ..., December。

*3.5 (找到将来的日期) 编写一个程序, 提示用户输入代表今天日期的数字 (周日为 0, 周一为 1, ..., 周六为 6)。同时, 提示用户输入一个今天之后的天数, 作为代表将来某天的数字, 然后显示这天是星期几。下面是一个运行示例:

```
Enter today's day: 1 ↵
Enter the number of days elapsed since today: 3 ↵
Today is Monday and the future day is Thursday
```

```
Enter today's day: 0 ↵
Enter the number of days elapsed since today: 31 ↵
Today is Sunday and the future day is Wednesday
```

*3.6 (医疗应用程序: BMI) 修改程序清单 3-4, 让用户输入重量、英尺和英寸。例如: 一个人身高是 5 英尺 10 英寸, 输入的英尺值就是 5、英寸值为 10。下面是一个运行示例:

```
Enter weight in pounds: 140 ↵
Enter feet: 5 ↵
Enter inches: 10 ↵
BMI is 20.087702275404553
Normal
```

3.7 (财务应用程序: 整钱兑零) 修改程序清单 2-10, 使之只显示非零的币值单位, 用单词的单数形式显示一个单位, 例如 1 dollar and 1 penny (1 美元和 1 美分); 用单词的复数形式显示多于一个单位的值, 例如 2 dollars and 3 pennies (2 美元和 3 美分)。

*3.8 (对三个整数排序) 编写程序, 提示用户输入三个整数。以非降序的形式显示这三个整数。

**3.9 (商业: 检查 ISBN-10) ISBN-10 (国际标准书号) 以前是一个 10 位整数 $d_1d_2d_3d_4d_5d_6d_7d_8d_9d_{10}$, 最后的一位 d_{10} 是校验和, 它是使用下面的公式用另外 9 个数计算出来的:

$$(d_1 \times 1 + d_2 \times 2 + d_3 \times 3 + d_4 \times 4 + d_5 \times 5 + d_6 \times 6 + d_7 \times 7 + d_8 \times 8 + d_9 \times 9) \% 11$$

如果校验和为 10, 那么按照 ISBN-10 的习惯, 最后一位应该表示为 X。编写程序, 提示用户输入前 9 个数, 然后显示 10 位 ISBN (包括前面起始位置的 0)。程序应该读取一个整数输入。下面是一个运行示例:

```
Enter the first 9 digits of an ISBN as integer: 013601267 
The ISBN-10 number is 0136012671
```

```
Enter the first 9 digits of an ISBN as integer: 013031997 
The ISBN-10 number is 013031997X
```

3.10 (游戏: 加法测验) 程序清单 3-3 随机产生一个减法问题。修改这个程序, 随机产生一个计算两个小于 100 的整数的加法问题。

3.8 ~ 3.16 节

*3.11 (给出一个月的总天数) 编写程序, 提示用户输入月份和年份, 然后显示这个月的天数。例如: 如果用户输入的月份是 2 而年份是 2012, 那么程序应该显示 “February 2012 has 29 days” (2012 年 2 月有 29 天)。如果用户输入的月份为 3 而年份为 2015, 那么程序就应该显示 “March 2015 has 31 days” (2015 年 3 月有 31 天)。

3.12 (回文数字) 编写一个程序, 提示用户输入一个三位的整数, 然后确定它是否回文数字。当从左到右, 以及从右到左都是一样的话, 这个数字称为回文数。下面是程序的一个运行示例:

```
Enter a three-digit integer: 121 
121 is a palindrome
```

```
Enter a three-digit integer: 123 
123 is not a palindrome
```

*3.13 (财务应用程序: 计算税款) 程序清单 3-5 给出了计算单身登记人税款的源代码。将程序清单 3-5 补充完整, 从而计算所有登记的婚姻状态的税款。

3.14 (游戏: 猜硬币的正反面) 编写程序, 让用户猜一猜是硬币的正面还是反面。这个程序随机产生一个整数 0 或者 1, 它们分别表示硬币的正面和反面。程序提示用户输入一个猜测值, 然后报告这个猜测值是正确的还是错误的。

**3.15 (游戏: 彩票) 修改程序清单 3-8, 产生三位整数的彩票。程序提示用户输入一个三位整数, 然后依照下面的规则判定用户是否赢得奖金:

- 1) 如果用户输入的所有数匹配彩票的确切顺序, 奖金是 10 000 美元。
- 2) 如果用户输入的所有数匹配彩票的所有数字, 奖金是 3 000 美元。
- 3) 如果用户输入的其中一个数匹配彩票号码中的一个数, 奖金是 1 000 美元。

3.16 (随机点) 编写程序, 显示矩形中一个随机点的坐标。矩形中心位于 (0,0)、宽 100、高 200。

*3.17 (游戏: 剪刀、石头、布) 编写可以玩流行的剪刀 - 石头 - 布游戏的程序。(剪刀可以剪布, 石头可以砸剪刀, 而布可以包石头。) 程序提示用户随机产生一个数, 这个数为 0、1 或者 2, 分别表示石头、剪刀和布。程序提示用户输入值 0、1 或者 2, 然后显示一条消息, 表明用户和计算机谁赢了游戏, 谁输了游戏, 或是打成平手。下面是运行示例:

```
scissor (0), rock (1), paper (2): 1 
The computer is scissor. You are rock. You won
```

```
scissor (0), rock (1), paper (2): 2 
The computer is paper. You are paper too. It is a draw
```

*3.18 (运输成本) 一个运输公司使用下面的函数, 根据运输重量 (以磅为单位) 来计算运输成本 (以美元计算)。

$$c(w) = \begin{cases} 3.5, & \text{若 } 0 < w \leq 1 \\ 5.5, & \text{若 } 1 < w \leq 3 \\ 8.5, & \text{若 } 3 < w \leq 10 \\ 10.5, & \text{若 } 10 < w \leq 20 \end{cases}$$

编写一个程序，提示用户输入包裹重量，显示运输成本。如果重量大于 20，显示一条消息“the package cannot be shipped”。

**3.19 (计算三角形的周长) 编写程序，读取三角形的三条边，如果输入值合法就计算这个三角形的周长；否则，显示这些输入值不合法。如果任意两条边的和大于第三边，那么输入值都是合法的。

*3.20 (科学：风寒温度) 编程练习题 2.17 给出计算风寒温度的公式。这个公式适用于温度在华氏 -58° 到 41° 之间，并且风速大于或等于 2 的情况。编写一个程序，提示用户输入一个温度值和一个风速值。如果输入值是合法的，那么显示风寒温度，否则显示一条消息，表明温度或风速是不合法数值。

综合题

**3.21 (科学：某天是星期几) 泽勒一致性是由克里斯汀·泽勒开发的用于计算某天是星期几的算法。这个公式是：

$$h = \left(q + \frac{26(m+1)}{10} + k + \frac{k}{4} + \frac{j}{4} + 5j \right) \% 7$$

其中：

- h 是一个星期中的某一天 (0 为星期六；1 为星期天；2 为星期一；3 为星期二；4 为星期三；5 为星期四；6 为星期五)。
- q 是某月的第几天。
- m 是月份 (3 为三月，4 为四月，..., 12 为十二月)。一月和二月分别记为上一年度的 13 和 14 月。
- j 是世纪数 (即 $\left\lfloor \frac{\text{year}}{100} \right\rfloor$)。
- k 是该世纪的第几年 (即 $\text{year} \% 100$)。

注意，公式中的除法执行一个整数相除。编写程序，提示用户输入年、月和该月的哪一天，然后显示它是一周中的星期几。下面是一些运行示例：

```
Enter year: (e.g., 2012): 2015 Enter
Enter month: 1-12: 1 Enter
Enter the day of the month: 1-31: 25 Enter
Day of the week is Sunday
```

```
Enter year: (e.g., 2012): 2012 Enter
Enter month: 1-12: 5 Enter
Enter the day of the month: 1-31: 12 Enter
Day of the week is Saturday
```

提示：一月和二月在这个公式里是用 13 和 14 表示的。所以需要将由用户输入的月份 1 转换为 13，将由用户输入的月份 2 转换为 14，同时将年份改为前一年。

**3.22 (几何：点是否在圆内?) 编写程序，提示用户输入一个点 (x, y)，然后检查这个点是否在以原点 (0, 0) 为圆心、半径为 10 的圆内。例如：(4, 5) 是圆内的一点，而 (9, 9) 是圆外的一点，如图 3-7a 所示。

提示：如果一个点到 (0, 0) 的距离小于或等于 10，那么该点就在圆内，计算距离的公式是 $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ ，使用各种情况来测试你的程序。以下是两个运行示例。

```
Enter a point with two coordinates: 4 5 Enter
Point (4.0, 5.0) is in the circle
```

```
Enter a point with two coordinates: 9 9 Enter
Point (9.0, 9.0) is not in the circle
```

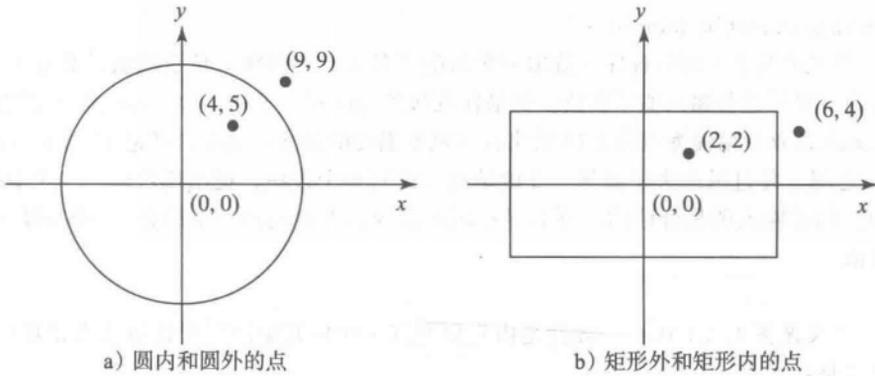


图 3-7

****3.23** (几何: 点是否在矩形内?) 编写程序, 提示用户输入点 (x, y) , 然后检测该点是否在以原点 $(0, 0)$ 为中心、宽为 10、高为 5 的矩形中。例如: $(2, 2)$ 在矩形内, 而 $(6, 4)$ 在矩形外, 如图 3-7b 所示。

提示: 如果一个点到点 $(0, 0)$ 的水平距离小于等于 $10/2$ 且到点 $(0, 0)$ 的垂直距离小于等于 $5.0/2$, 该点就在矩形内, 使用各种情况来测试你的程序。

这里有两个运行示例:

```
Enter a point with two coordinates: 2 2 Enter
Point (2.0, 2.0) is in the rectangle
```

```
Enter a point with two coordinates: 6 4 Enter
Point (6.0, 4.0) is not in the rectangle
```

****3.24** (游戏: 挑一张牌) 编写程序, 模拟从一副 52 张的牌中选择一张牌。程序应该显示牌的大小 (Ace、2、3、4、5、6、7、8、9、10、Jack、Queen、King) 以及牌的花色 (Clubs (黑梅花)、Diamonds (红方块)、Hearts (红心)、Spades (黑桃))。下面是这个程序的运行示例:

```
The card you picked is Jack of Hearts
```

***3.25** (几何: 交点) 第一条直线上面的两个点是 (x_1, y_1) 和 (x_2, y_2) , 第二条直线的两个点是 (x_3, y_3) 和 (x_4, y_4) , 如图 3-8a、图 3-8b 所示。两条直线的交点可以通过下面的线性方程组求解:

$$(y_1 - y_2)x - (x_1 - x_2)y = (y_1 - y_2)x_1 - (x_1 - x_2)y_1$$

$$(y_3 - y_4)x - (x_3 - x_4)y = (y_3 - y_4)x_3 - (x_3 - x_4)y_3$$

这个线性方程组可以应用 Cramer 规则求解 (见编程练习题 3.3)。如果方程无解, 则两条直线平行 (图 3-8c)。

编写一个程序, 提示用户输入这四个点, 然后显示它们的交点。下面是这个程序的运行示例:

```
Enter x1, y1, x2, y2, x3, y3, x4, y4: 2 2 5 -1.0 4.0 2.0 -1.0 -2.0 Enter
The intersecting point is at (2.88889, 1.1111)
```

```
Enter x1, y1, x2, y2, x3, y3, x4, y4: 2 2 7 6.0 4.0 2.0 -1.0 -2.0 Enter
The two lines are parallel
```

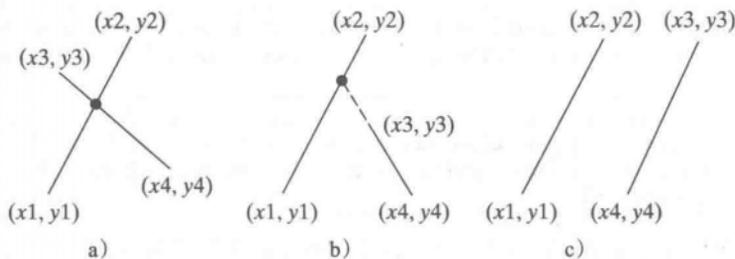
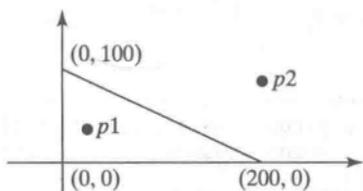


图 3-8 两条直线相交 (a 和 b), 两条直线平行 (c)

3.26 (使用操作符 `&&`、`||` 和 `^`) 编写一个程序, 提示用户输入一个整数值, 然后判定它是否能被 5 和 6 整除, 是否能被 5 或 6 整除, 以及是否能被 5 或 6 整除但是不能同时被它们整除。下面是这个程序的运行示例:

```
Enter an integer: 10 
Is 10 divisible by 5 and 6? false
Is 10 divisible by 5 or 6? true
Is 10 divisible by 5 or 6, but not both? true
```

**3.27 (几何: 点是否在三角形内?) 假设一个直角三角形放在一个平面上, 如下图所示。直角点在 $(0, 0)$ 处, 其他两个点分别在 $(200, 0)$ 和 $(0, 100)$ 处。编写程序, 提示用户输入一个点的 x 坐标和 y 坐标, 然后判定这个点是否在该三角形内。下面是运行示例:



```
Enter a point's x- and y-coordinates: 100.5 25.5 
The point is in the triangle
```

```
Enter a point's x- and y-coordinates: 100.5 50.5 
The point is not in the triangle
```

**3.28 (几何: 两个矩形) 编写一个程序, 提示用户输入两个矩形中点的 x 坐标和 y 坐标以及它们的宽度和高度, 然后判定第二个矩形是在第一个矩形内, 还是和第一个矩形重叠, 如图 3-9 所示。

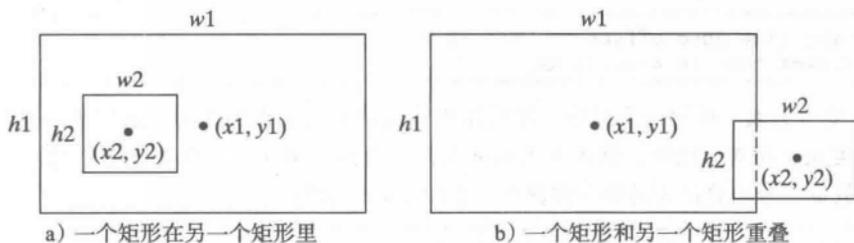


图 3-9

下面是运行示例:

```
Enter r1's center x-, y-coordinates, width, and height: 2.5 4 2.5 43 
Enter r2's center x-, y-coordinates, width, and height: 1.5 5 0.5 3 
r2 is inside r1
```

```
Enter r1's center x-, y-coordinates, width, and height: 1 2 3 5.5 --Enter
Enter r2's center x-, y-coordinates, width, and height: 3 4 4.5 5 --Enter
r2 overlaps r1
```

```
Enter r1's center x-, y-coordinates, width, and height: 1 2 3 3 --Enter
Enter r2's center x-, y-coordinates, width, and height: 40 45 3 2 --Enter
r2 does not overlap r1
```

****3.29** (几何: 两个圆) 编写程序, 提示用户输入两个圆的中心坐标和各自的半径值, 然后决定第二个圆是在第一个圆内, 还是和第一个圆重叠, 如图 3-10 所示。

提示: 如果两个圆心的距离 $\leq |r1-r2|$, 就认为 circle2 在 circle1 内; 如果两个圆心的距离 $\leq r1+r2$, 就认为 circle2 和 circle1 重叠。

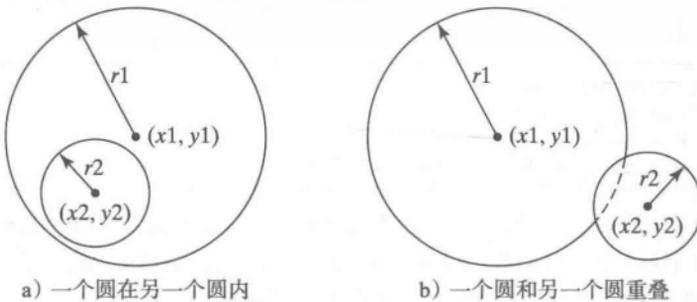


图 3-10

下面是运行示例:

```
Enter circle1's center x-, y-coordinates, and radius: 0.5 5.1 13 --Enter
Enter circle2's center x-, y-coordinates, and radius: 1 1.7 4.5 --Enter
circle2 is inside circle1
```

```
Enter circle1's center x-, y-coordinates, and radius: 3.4 5.7 5.5 --Enter
Enter circle2's center x-, y-coordinates, and radius: 6.7 3.5 3 --Enter
circle2 overlaps circle1
```

```
Enter circle1's center x-, y-coordinates, and radius: 3.4 5.5 1 --Enter
Enter circle2's center x-, y-coordinates, and radius: 5.5 7.2 1 --Enter
circle2 does not overlap circle1
```

***3.30** (当前时间) 修改编程练习题 2.8, 以 12 小时制显示小时数。这里是一个运行示例:

```
Enter the time zone offset to GMT: -5 --Enter
The current time is 4:50:34 AM
```

***3.31** (金融: 货币兑换) 编写一个程序, 提示用户输入从美元到人民币的兑换汇率。提示用户输入 0 表示从美元兑换为人民币, 输入 1 表示从人民币兑换为美元。继而提示用户输入美元数量或者人民币数量, 分别兑换为另外一种货币。下面是运行示例:

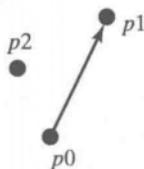
```
Enter the exchange rate from dollars to RMB: 6.81 --Enter
Enter 0 to convert dollars to RMB and 1 vice versa: 0 --Enter
Enter the dollar amount: 100 --Enter
$100.0 is 681.0 yuan
```

```
Enter the exchange rate from dollars to RMB: 6.81 --Enter
Enter 0 to convert dollars to RMB and 1 vice versa: 1 --Enter
Enter the RMB amount: 10000 --Enter
10000.0 yuan is $1468.43
```

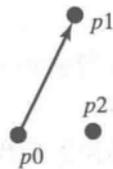
```
Enter the exchange rate from dollars to RMB: 6.81 ↵
Enter 0 to convert dollars to RMB and 1 vice versa: 5 ↵
Incorrect input
```

*3.32 (几何: 点的位置) 给定一个从点 $p_0(x_0, y_0)$ 到 $p_1(x_1, y_1)$ 的有向线段, 可以使用下面的条件来确定点 $p_2(x_2, y_2)$ 是在线段的左侧、右侧, 或者在该直线上 (见图 3-11):

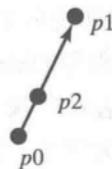
$$(x_1 - x_0) \times (y_2 - y_0) - (x_2 - x_0) \times (y_1 - y_0) \begin{cases} > 0 & p_2 \text{ 在线段的左侧} \\ = 0 & p_2 \text{ 在该线段上} \\ < 0 & p_2 \text{ 在线段的右侧} \end{cases}$$



a) p_2 在线段的左侧



b) p_2 在线段的右侧



c) p_2 在该线段上

图 3-11

编写一个程序, 提示用户输入三个点 p_0 、 p_1 和 p_2 , 显示 p_2 是否在从 p_0 到 p_1 的线段左侧、右侧, 或者在该直线上。下面是运行示例:

```
Enter three points for p0, p1, and p2: 4.4 2 6.5 9.5 -5 4 ↵
(-5.0, 4.0) is on the left side of the line from (4.4, 2.0) to (6.5, 9.5)
```

```
Enter three points for p0, p1, and p2: 1 1 5 5 2 2 ↵
(2.0, 2.0) is on the line from (1.0, 1.0) to (5.0, 5.0)
```

```
Enter three points for p0, p1, and p2: 3.4 2 6.5 9.5 5 2.5 ↵
(5.0, 2.5) is on the right side of the line from (3.4, 2.0) to (6.5, 9.5)
```

*3.33 (金融: 比较成本) 假设你要通过两种不同的包裹运输大米。你将乐于编写一个程序来比较成本, 该程序提示用户输入每个包裹的重量和价格, 然后显示具有更好价格的包裹。下面是一个运行示例:

```
Enter weight and price for package 1: 50 24.59 ↵
Enter weight and price for package 2: 25 11.99 ↵
Package 2 has a better price.
```

```
Enter weight and price for package 1: 50 25 ↵
Enter weight and price for package 2: 25 12.5 ↵
Two packages have the same price.
```

*3.34 (几何: 线段上的点) 编程练习题 3.32 显示了如何测试一个点是否在一个无限长的直线上。修改编程练习题 3.32, 测试一个点是否在一个线段上。编写一个程序, 提示用户输入三个点 p_0 、 p_1 和 p_2 , 显示 p_2 是否在从 p_0 到 p_1 的线段上。这里是一些运行示例:

```
Enter three points for p0, p1, and p2: 1 1 2.5 2.5 1.5 1.5 ↵
(1.5, 1.5) is on the line segment from (1.0, 1.0) to (2.5, 2.5) ↵
```

```
Enter three points for p0, p1, and p2: 1 1 2 2 3.5 3.5 ↵
(3.5, 3.5) is not on the line segment from (1.0, 1.0) to (2.0, 2.0)
```

数学函数、字符和字符串

教学目标

- 使用 Math 类中的方法解决数学问题 (4.2 节)。
- 使用 char 类型表示字符 (4.3 节)。
- 使用 ASCII 码和 Unicode 码来对字符编码 (4.3.1 节)。
- 用转义序列表示特殊字符 (4.3.2 节)。
- 将数值转换为字符, 以及将字符转换为整数 (4.3.3 节)。
- 使用 Character 类中的静态方法来比较和测试字符 (4.3.4 节)。
- 介绍对象和实例方法 (4.4 节)。
- 使用 String 对象表示字符串 (4.4 节)。
- 使用 length() 方法来返回字符串长度 (4.4.1 节)。
- 使用 charAt(i) 方法来返回字符串中的字符 (4.4.2 节)。
- 使用操作符 + 来连接字符串 (4.4.3 节)。
- 返回大写字符串或者小写字符串, 以及裁剪字符串 (4.4.4 节)。
- 从控制台读取字符串 (4.4.5 节)。
- 从控制台读取字符 (4.4.6 节)。
- 使用 equals 方法和 compareTo 方法比较字符串 (4.4.7 节)。
- 获取子字符串 (4.4.8 节)。
- 使用 indexOf 方法定位一个字符串中的字符或者子字符串 (4.4.9 节)。
- 使用字符和字符串编程 (GuessBirthday)(4.5.1 节)。
- 将十六进制字符转换为十进制值 (HexDigit2Dec)(4.5.2 节)。
- 使用字符串修改彩票程序 (LotteryUsingStrings)(4.5.3 节)。
- 使用 System.out.printf 方法来格式化输出 (4.6 节)。

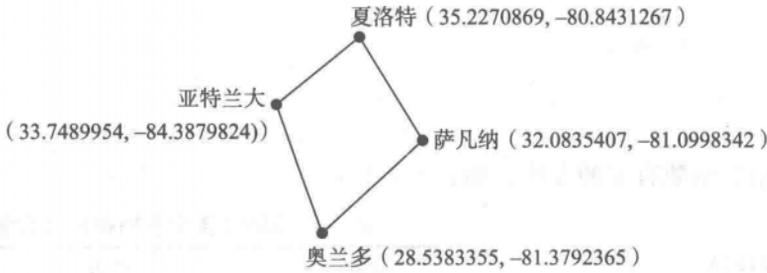
4.1 引言

 **要点提示:** 本章重点介绍数学函数、字符和字符串对象, 并使用它们来开发程序。

前面章节介绍了基础编程技术, 以及如何使用选择语句编写简单的程序来解决一些基本问题。本章介绍实现常用数学操作的方法。你将在第 6 章学到如何创建自定义方法。

假设你需要估算被四座城市包围的区域的面积, 给定这些城市的 GPS 位置 (经纬度), 如下图所示。如何编写程序来求解这个问题? 在学完本章后, 你将可以编写这样的程序。

因为字符串在编程中经常要用到, 所以尽早介绍字符串, 从而可以开始使用它们来编写实用的程序, 是有裨益的。本章给出字符串的简要介绍; 你将在第 9 章和第 10 章进一步学习对象和字符串相关知识。



4.2 常用数学函数

要点提示: Java 在 `Math` 类中提供了许多实用的方法，来计算常用的数学函数。

方法是一组语句，用于执行一个特定的任务。在 2.9.4 节中我们已经使用过方法 `pow(a,b)` 计算 a^b ，在 3.7 节中也使用过幂操作，以及 `Math.random()` 方法来产生一个随机数。本节介绍 `Math` 类中其他有用的方法。这些方法分为三类：三角函数方法 (trigonometric method)、指数函数方法 (exponent method) 和服务方法 (service method)。服务方法包括取整、求最小值、求最大值、求绝对值和随机方法。除了这些方法之外，`Math` 类还提供了两个很有用的 `double` 型常量，`PI` (π) 和 `E` (自然对数的底)。可以在任意程序中使用 `Math.PI` 和 `Math.E` 的形式来使用这两个常量。

4.2.1 三角函数方法

`Math` 类包含表 4-1 中所示的三角函数方法。

表 4-1 `Math` 类中的三角函数方法

方法	描述
<code>sin(radians)</code>	返回以弧度为单位的角度的三角正弦函数值
<code>cos(radians)</code>	返回以弧度为单位的角度的三角余弦函数值
<code>tan(radians)</code>	返回以弧度为单位的角度的三角正切函数值
<code>toRadians(degree)</code>	将以度为单位的角度值转换为以弧度表示
<code>toDegrees(radians)</code>	将以弧度为单位的角度值转换为以度表示
<code>asin(a)</code>	返回以弧度为单位的角度的反三角正弦函数值
<code>acos(a)</code>	返回以弧度为单位的角度的反三角余弦函数值
<code>atan(a)</code>	返回以弧度为单位的角度的反三角正切函数值

`sin`、`cos` 和 `tan` 的参数都是以弧度为单位的角度。`asin` 和 `atan` 的返回值是 $-\pi/2 \sim \pi/2$ 的一个弧度值，`acos` 的返回值在 0 到 π 之间。 1° 相当于 $\pi/180$ 弧度， 90° 相当于 $\pi/2$ 弧度，而 30° 相当于 $\pi/6$ 弧度。

例如：

```

Math.toDegrees(Math.PI / 2) 返回 90.0
Math.toRadians(30) 返回 0.5236 (π/6)
Math.sin(0) returns 0.0
Math.sin(Math.toRadians(270)) 返回 -1.0
Math.sin(Math.PI / 6) 返回 0.5
Math.sin(Math.PI / 2) 返回 1.0
Math.cos(0) 返回 1.0
Math.cos(Math.PI / 6) 返回 0.866
Math.cos(Math.PI / 2) 返回 0
Math.asin(0.5) 返回 0.523598333 (π/6)
    
```

Math.acos(0.5) 返回 1.0472 ($\pi/3$)
 Math.atan(1.0) 返回 0.785398 ($\pi/4$)

4.2.2 指数函数方法

Math 类中有 5 个与指数函数有关的方法，如表 4-2 所示。

例如，

Math.exp(1) 返回 2.71828
 Math.log(Math.E) 返回 1.0
 Math.log10(10) 返回 1.0
 Math.pow(2, 3) 返回 8.0
 Math.pow(3, 2) 返回 9.0
 Math.pow(4.5, 2.5) 返回 22.91765
 Math.sqrt(4) 返回 2.0
 Math.sqrt(10.5) 返回 4.24

表 4-2 Math 类中的指数函数方法

方法	描述
exp(x)	返回 e 的 x 次方
log(x)	返回 x 的自然底数
log10(x)	返回 x 的以 10 为底的对数
pow(a, b)	返回 a 的 b 次方
sqrt(x)	对于 $x \geq 0$ 的数字，返回 x 的平方根

4.2.3 取整方法

Math 类包括五个取整方法，如表 4-3 所示。

表 4-3 Math 类中的取整方法

方法	描述
ceil(x)	x 向上取整为它最接近的整数。该整数作为一个双精度值返回
floor(x)	x 向下取整为它最接近的整数。该整数作为一个双精度值返回
rint(x)	x 取整为它最接近的整数。如果 x 与两个整数的距离相等，偶数的整数作为一个双精度值返回
round(x)	如果 x 是单精度数，返回 (int) Math.floor(x+0.5)；如果 x 是双精度数，返回 (long) Math.floor(x+0.5)

例如：

Math.ceil(2.1) 返回 3.0
 Math.ceil(2.0) 返回 2.0
 Math.ceil(-2.0) 返回 -2.0
 Math.ceil(-2.1) 返回 -2.0
 Math.floor(2.1) 返回 2.0
 Math.floor(2.0) 返回 2.0
 Math.floor(-2.0) 返回 -2.0
 Math.floor(-2.1) 返回 -3.0
 Math.rint(2.1) 返回 2.0
 Math.rint(-2.0) 返回 -2.0
 Math.rint(-2.1) 返回 -2.0
 Math.rint(2.5) 返回 2.0
 Math.rint(4.5) 返回 4.0
 Math.rint(-2.5) 返回 -2.0
 Math.round(2.6f) 返回 3 // Returns int
 Math.round(2.0) 返回 2 // Returns long
 Math.round(-2.0f) 返回 -2 // Returns int
 Math.round(-2.6) 返回 -3 // Returns long
 Math.round(-2.4) 返回 -2 // Returns long

4.2.4 min、max 和 abs 方法

min 和 max 方法用于返回两个数 (int、long、float 或 double 型) 的最小值和最大值。

例如，max(4.4, 5.0) 返回 5.0，而 min(3, 2) 返回 2。

abs 方法以返回一个数 (int、long、float 或 double 型) 的绝对值。

例如:

```
Math.max(2, 3) 返回 3
Math.max(2.5, 3) 返回 3.0
Math.min(2.5, 4.6) 返回 2.5
Math.abs(-2) 返回 2
Math.abs(-2.1) 返回 2.1
```

4.2.5 random 方法

你已经使用过 random() 方法, 生成大于等于 0.0 且小于 1.0 的 double 型随机数 (0.0 ≤ Math.random() < 1.0)。可以使用它编写简单的表达式, 生成任意范围的随机数。例如:

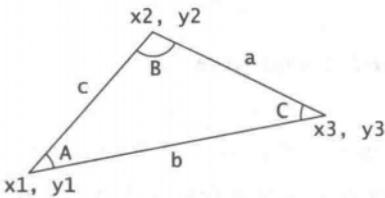
```
(int)(Math.random() * 10)  → 返回 0 ~ 9 之间的一个随机整数
50 + (int)(Math.random() * 50) → 返回 50 ~ 99 之间的一个随机整数
```

通常,

```
a + Math.random() * b  → 返回 a ~ a+b 之间的一个随机整数, 不包括 a+b
```

4.2.6 示例学习: 计算三角形的角度

可以使用数学方法求解许多计算问题。比如, 给定一个三角形的三条边, 可以通过以下公式计算角度。



$$A = \arccos((a^2 + b^2 - c^2) / (2 * a * b))$$

$$B = \arccos((b^2 + c^2 - a^2) / (2 * b * c))$$

$$C = \arccos((c^2 + a^2 - b^2) / (2 * c * a))$$

别被这个数学公式所吓住。如程序清单 2-9 所讨论的, 不需要知道数学公式是如何推导的, 依然可以计算贷款支付。在这个例子中, 给出三条边的长度, 可以运用这个公式来编写程序计算角度, 而不需要知道这个公式是如何推导的。为了计算边长, 需要知道三个顶点的坐标, 然后计算这些点之间的距离。

程序清单 4-1 是一个示例程序, 提示用户输入三角形三个顶点的 x 和 y 坐标值, 然后显示三个角。

程序清单 4-1 ComputeAngles.java

```
1 import java.util.Scanner;
2
3 public class ComputeAngles {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6
7         // Prompt the user to enter three points
8         System.out.print("Enter three points: ");
9         double x1 = input.nextDouble();
10        double y1 = input.nextDouble();
11        double x2 = input.nextDouble();
12        double y2 = input.nextDouble();
13        double x3 = input.nextDouble();
14        double y3 = input.nextDouble();
```

```

15
16 // Compute three sides
17 double a = Math.sqrt((x2 - x3) * (x2 - x3)
18     + (y2 - y3) * (y2 - y3));
19 double b = Math.sqrt((x1 - x3) * (x1 - x3)
20     + (y1 - y3) * (y1 - y3));
21 double c = Math.sqrt((x1 - x2) * (x1 - x2)
22     + (y1 - y2) * (y1 - y2));
23
24 // Compute three angles
25 double A = Math.toDegrees(Math.acos((a * a - b * b - c * c)
26     / (-2 * b * c)));
27 double B = Math.toDegrees(Math.acos((b * b - a * a - c * c)
28     / (-2 * a * c)));
29 double C = Math.toDegrees(Math.acos((c * c - b * b - a * a)
30     / (-2 * a * b)));
31
32 // Display results
33 System.out.println("The three angles are " +
34     Math.round(A * 100) / 100.0 + " " +
35     Math.round(B * 100) / 100.0 + " " +
36     Math.round(C * 100) / 100.0);
37 }
38 }

```

Enter three points: 1 1 6.5 1 6.5 2.5
The three angles are 15.26 90.0 74.74

程序提示用户输入三个顶点(第8行)。这个提示消息并不是很清晰,应该给予用户非常清晰的提示,如下所示:

```

System.out.print("Enter the coordinates of three points separated "
    + "by spaces like x1 y1 x2 y2 x3 y3: ");

```

注意,两个点 (x_1, y_1) , (x_2, y_2) 之间的距离可以通过公式 $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ 计算。程序计算两个点之间的距离(第17~22行),并且应用该公式来计算角度(第25~30行)。显示的角度值保留小数点后两位数字(第34~36行)。

Math类在程序中使用,但是并没有导入,因为它在 `java.lang` 包中。在一个Java程序中, `java.lang` 包中的所有类是隐式导入的。

复习题

4.1 计算下面的方法调用:

- | | |
|---|--|
| a. <code>Math.sqrt(4)</code> | j. <code>Math.floor(-2.5)</code> |
| b. <code>Math.sin(2 * Math.PI)</code> | k. <code>Math.round(-2.5f)</code> |
| c. <code>Math.cos(2 * Math.PI)</code> | l. <code>Math.round(-2.5)</code> |
| d. <code>Math.pow(2, 2)</code> | m. <code>Math rint(2.5)</code> |
| e. <code>Math.log(Math.E)</code> | n. <code>Math.ceil(2.5)</code> |
| f. <code>Math.exp(1)</code> | o. <code>Math.floor(2.5)</code> |
| g. <code>Math.max(2, Math.min(3, 4))</code> | p. <code>Math.round(2.5f)</code> |
| h. <code>Math rint(-2.5)</code> | q. <code>Math.round(2.5)</code> |
| i. <code>Math.ceil(-2.5)</code> | r. <code>Math.round(Math.abs(-2.5))</code> |

4.2 下述说法是否正确?三角函数方法中的参数是以弧度为单位的角。

4.3 编写一条语句,将 47° 转换为弧度值,并将结果赋给一个变量。

- 4.4 编写一条语句，将 $\pi / 7$ 转换为角度值，并将结果赋给一个变量。
- 4.5 编写一个表达式，返回 34 ~ 55 的一个随机整数。编写一个表达式，返回 0 ~ 999 的一个随机整数。编写一个表达式，返回 5.5 ~ 55.5 的一个随机数。
- 4.6 为什么 Math 类不需要导入？
- 4.7 `Math.log(Math.exp(5.5))` 等于多少？`Math.exp(Math.log(5.5))` 等于多少？`Math.asin(Math.sin(Math.PI / 6))` 等于多少？`Math.sin(Math.asin(Math.PI / 6))` 等于多少？

4.3 字符数据类型和操作

 **要点提示：**字符数据类型表示单个字符。

除了处理数值之外，Java 还可以处理字符。字符数据类型 `char` 用来表示单个字符。字符型直接量用单引号括住。考虑以下代码：

```
char letter = 'A';
char numChar = '4';
```

第一条语句将字符 A 赋值给 `char` 型变量 `letter`。第二条语句将数字字符 4 赋值给 `char` 型变量 `numChar`。

 **警告：**字符串直接量必须括在双引号中。而字符直接量是括在单引号中的单个字符。因此 "A" 是一个字符串，而 'A' 是一个字符。

4.3.1 Unicode 和 ASCII 码

计算机内部使用二进制数。一个字符在计算机中是以 0 和 1 构成的序列的形式来存储的。将字符映射到它的二进制形式的过程称为编码 (encoding)。字符有多种不同的编码方式，编码表 (encoding scheme) 定义该如何编码每个字符。

Java 支持 Unicode 码，Unicode 码是由 Unicode 协会建立的一种编码方案，它支持使用世界各种语言所书写的文本的交换、处理和显示。Unicode 码一开始被设计为 16 位的字符编码。基本数据类型 `char` 试图通过提供一种能够存放任意字符的简单数据类型来利用这个设计。但是，一个 16 位的编码所能产生的字符只有 65 536 个，它是不足以表示全世界所有字符的。因此，Unicode 标准被扩展为 1 112 064 个字符。这些字符都远远超过了原来 16 位的限制，它们称为补充字符 (supplementary character)。Java 支持这些补充字符。对补充字符的处理和表示介绍超过了本书的范围。为了简化问题，本书只考虑原来的 16 位 Unicode 字符。这些字符都可以存储在一个 `char` 型变量中。

一个 16 位 Unicode 码占两个字节，用以 `\u` 开头的 4 位十六进制数表示，范围从 `'\u0000'` 到 `'\uFFFF'`。关于十六进制数字的更多内容请参见附录 F。例如：“welcome”被翻译成中文需要两个字符“欢迎”。这两个字符的 Unicode 码为 `\u6B22\u8FCE`。希腊字母 $\alpha \beta \gamma$ 的 Unicode 码是 `\u03b1\u03b2\u03b3`。

大多数计算机采用 ASCII 码 (美国标准信息交换码)，它是表示所有大小写字母、数字、标点符号和控制字符的 8 位编码表。Unicode 码包括 ASCII 码，从 `'\u0000'` 到 `'\u007F'` 对应 128 个 ASCII 字符。表 4-4 对一些常用的字符给出了 ASCII 码。附录 B 给出了 ASCII 字符的完整清单，以及它们的十进制和十六进制编码。

表 4-4 常用字符的 ASCII 码

字符	十进制编码值	Unicode 值
'0' ~ '9'	48 ~ 57	<code>\u0030 ~ \u0039</code>
'A' ~ 'Z'	65 ~ 90	<code>\u0041 ~ \u005A</code>
'a' ~ 'z'	97 ~ 122	<code>\u0061 ~ \u007A</code>

Java 程序中，可以使用像 'x'、'1' 和 '\$' 这样的 ASCII 字符，也可以使用 Unicode 码。例如，下面的语句是等价的：

```
char letter = 'A';
char letter = '\u0041'; // Character A's Unicode is 0041
```

两条语句都将字符 A 赋值给 char 型变量 letter。

注意：自增和自减操作符也可用在 char 型变量上，这会得到该字符之前或之后的 Unicode 字符。例如，下面的语句显示字符 b。

```
char ch = 'a';
System.out.println(++ch);
```

4.3.2 特殊字符的转义序列

假如你想在输出时显示如下带引号的信息，你能编写如下所示的这条语句吗？

```
System.out.println("He said "Java is fun"");
```

答案是不能，这条语句有语法错误。编译器会认为第二个引号字符就是这个字符串的结束标志，而不知道如何处理剩余的字符。

为了解决这个问题，Java 定义了一种特殊的标记来表示特殊字符，如表 4-5 所示。这种标记称为转义序列，转义序列由反斜杠 (\) 后面加上一个字符或者一些数字位组成。比如，\t 是一个表示 Tab 字符的转义符，而诸如 \u03b1 的转义符用于表示一个 Unicode。转义序列中的序列号作为一个整体翻译，而不是分开翻译。一个转义序列被当作一个字符。

所以，现在可以使用下面的语句输出带引号的消息：

```
System.out.println("He said \"Java is fun\"");
```

它的输出是：

```
He said "Java is fun"
```

注意，符号 \ 和 "" 一起代表一个字符。

表 4-5 转义序列

转义序列	名称	Unicode 码	十进制值
\b	退格键	\u0008	8
\t	Tab 键	\u0009	9
\n	换行符	\u000A	10
\f	换页符	\u000C	12
\r	回车符	\u000D	13
\\	反斜杠	\u005C	92
\"	双引号	\u0022	34

反斜杠 \ 被称为转义字符。它是一个特殊字符。要显示这个字符，需要使用转义序列 \\。比如，下面的代码

```
System.out.println("\\t is a tab character");
```

显示

```
\t is a tab character
```

4.3.3 字符型数据与数值型数据之间的转换

char 型数据可以转换成任意一种数值类型，反之亦然。将整数转换成 char 型数据时，只用到该数据的低十六位，其余部分都被忽略。例如：

```
char ch = (char)0XAB0041; // The lower 16 bits hex code 0041 is
                        // assigned to ch
System.out.println(ch); // ch is character A
```

要将一个浮点值转换成 char 型时，首先将浮点值转换成 int 型，然后将这个整型值转换为 char 型。

```
char ch = (char)65.25; // Decimal 65 is assigned to ch
System.out.println(ch); // ch is character A
```

当一个 char 型数据转换成数值型时，这个字符的 Unicode 码就被转换成某个特定的数值类型。

```
int i = (int)'A'; // The Unicode of character A is assigned to i
System.out.println(i); // i is 65
```

如果转换结果适用于目标变量，就可以使用隐式转换方式；否则，必须使用显式转换方式。例如，因为 'a' 的 Unicode 码是 97，它在一个字节的范围内，所以就可以使用隐式转换方式：

```
byte b = 'a';
int i = 'a';
```

但是，因为 Unicode 码 \uFFFF 超过了一个字节的范围，所以下面的转换就是不正确的：

```
byte b = '\uFFFF';
```

为了强制赋值，就必须使用显式转换方式，如下所示：

```
byte b = (byte)\uFFFF;
```

0 ~ FFFF 的任何一个十六进制正整数都可以隐式地转换成字符型数据。而不在此范围内的任何其他数值都必须显式地转换为 char 型。

所有数值操作符都可以用在 char 型操作数上。如果另一个操作数是一个数字或字符，那么 char 型操作数就会被自动转换成一个数字。如果另一个操作数是一个字符串，字符就会与该字符串相连。例如，下面的语句：

```
int i = '2' + '3'; // (int)'2' is 50 and (int)'3' is 51
System.out.println("i is " + i); // i is 101
int j = 2 + 'a'; // (int)'a' is 97
System.out.println("j is " + j); // j is 99
System.out.println(j + " is the Unicode for character "
    + (char)j); // 99 is the Unicode for character c
System.out.println("Chapter " + '2');
```

显示结果

```
i is 101
j is 99
99 is the Unicode for character c
Chapter 2
```

4.3.4 字符的比较和测试

两个字符可以使用关系操作符进行比较，如同比较两个数字一样。这是通过比较两个字

符的 Unicode 值实现的。比如：

'a' < 'b' 为 true，因为 'a' (97) 的 Unicode 值比 'b' (98) 的 Unicode 值小。

'a' < 'A' 为 false，因为 'a' (97) 的 Unicode 值比 'A' (65) 的 Unicode 值大。

'I' < '8' 为 true，因为 'I' (49) 的 Unicode 值比 '8' (56) 的 Unicode 值小。

经常，程序中需要测试一个字符是数字、字母；大写字母，还是小写字母。如附录 B 的 ASCII 字符集所示，小写字母的 Unicode 是连续的整数，从 'a' 的 Unicode 开始，然后是 'b', 'c', ..., 'z'。同理，对于大写字母和数字字符也是这样的。这个特征可以用于编写测试字符的编码。比如，下列代码测试字符 ch 是大写字母、小写字母，还是数字字符。

```
if (ch >= 'A' && ch <= 'Z')
    System.out.println(ch + " is an uppercase letter");
else if (ch >= 'a' && ch <= 'z')
    System.out.println(ch + " is a lowercase letter");
else if (ch >= '0' && ch <= '9')
    System.out.println(ch + " is a numeric character");
```

为了方便，Java 的 Character 类提供了下列方法用于进行字符测试，如表 4-6 所示。

表 4-6 Character 类中的方法

方法	描述
isDigit(ch)	如果指定的字符是一个数字，返回 true
isLetter(ch)	如果指定的字符是一个字母，返回 true
isLetterOrDigit(ch)	如果指定的字符是一个字母或者数字，返回 true
isLowerCase(ch)	如果指定的字符是一个小写字母，返回 true
isUpperCase(ch)	如果指定的字符是一个大写字母，返回 true
toLowerCase(ch)	返回指定的字符的小写形式
toUpperCase(ch)	返回指定的字符的大写形式

例如：

```
System.out.println("isDigit('a') is " + Character.isDigit('a'));
System.out.println("isLetter('a') is " + Character.isLetter('a'));
System.out.println("isLowerCase('a') is "
    + Character.isLowerCase('a'));
System.out.println("isUpperCase('a') is "
    + Character.isUpperCase('a'));
System.out.println("toLowerCase('T') is "
    + Character.toLowerCase('T'));
System.out.println("toUpperCase('q') is "
    + Character.toUpperCase('q'));
```

显示

```
isDigit('a') is false
isLetter('a') is true
isLowerCase('a') is true
isUpperCase('a') is false
toLowerCase('T') is t
toUpperCase('q') is Q
```

复习题

- 4.8 使用输出语句，得到 '1'、'A'、'B'、'a' 和 'b' 的 ASCII 码。使用输出语句得到十进制码 40、59、79、85 和 90 的字符。使用输出语句得到十六进制码 40、5A、71、72 和 7A 的字符。
- 4.9 以下哪些是正确的字符直接值？

```
'l', '\u345dE', '\u3fFa', '\b', '\t'
```

4.10 如何显示字符 \ 和 " ?

4.11 求以下值:

```
int i = '1';
int j = '1' + '2' * ('4' - '3') + 'b' / 'a';
int k = 'a';
char c = 90;
```

4.12 下面涉及类型转换的转换合法吗? 如果合法, 给出转换后的结果。

```
char c = 'A';
int i = (int)c;
```

```
float f = 1000.34f;
int i = (int)f;
```

```
double d = 1000.34;
int i = (int)d;
```

```
int i = 97;
char c = (char)i;
```

4.13 给出下面程序的输出结果。

```
public class Test {
    public static void main(String[] args) {
        char x = 'a';
        char y = 'c';
        System.out.println(++x);
        System.out.println(y++);
        System.out.println(x - y);
    }
}
```

4.14 编写代码, 产生随机的小写字母。

4.15 给出下面语句的输出结果。

```
System.out.println('a' < 'b');
System.out.println('a' <= 'A');
System.out.println('a' > 'b');
System.out.println('a' >= 'A');
System.out.println('a' == 'a');
System.out.println('a' != 'b');
```

4.4 String 类型

 **要点提示:** 字符串是一个字符序列。

char 类型只能表示一个字符。为了表示一串字符, 使用称为 String (字符串) 的数据类型。例如, 下述代码将 message 声明为一个字符串, 其值为 "Welcome to Java":

```
String message = "Welcome to Java";
```

String 实际上与 System 类和 Scanner 类一样, 都是 Java 库中一个预定义的类。String 类型不是基本类型, 而是引用类型 (reference type)。任何 Java 类都可以将变量表示为引用类型。使用引用类型声明的变量称为引用变量, 它引用一个对象。这里, message 是一个引用变量, 它引用一个内容为 Welcome to Java 的字符串对象。

引用数据类型将在第 9 章中详细讨论。目前, 只需要知道如何声明 String 类型的变量,

如何将字符串赋值给该变量以及如何使用 `String` 类中的方法。关于使用字符串的更多细节将在第 10 章涵盖。

表 4-7 列出了获得字符串长度、访问字符串中字符、连接字符串、转换字符串为大写或者小写，以及裁剪字符串的 `String` 方法。

表 4-7 `String` 对象的简单方法

方法	描述
<code>length()</code>	返回字符串中的字符数
<code>charAt(index)</code>	返回字符串 <code>s</code> 中指定位置的字符
<code>concat(s1)</code>	将本字符串和字符串 <code>s1</code> 连接，返回一个新字符串
<code>toUpperCase()</code>	返回一个新字符串，其中所有的字母大写
<code>toLowerCase()</code>	返回一个新字符串，其中所有的字母小写
<code>trim()</code>	返回一个新字符串，去掉了两边的空白字符

`String` 是 Java 中的对象。表 4-7 中的方法只能从一个特定的字符串实例来调用。由于这个原因，这些方法称为实例方法。非实例方法称为静态方法。静态方法可以不使用对象来调用。定义在 `Math` 类中的所有方法都是静态方法。它们没有绑定到一个特定的对象实例上。调用一个实例方法的语法是 `reference-Variable.methodName(arguments)`。一个方法可以有多个参数，或者无参。例如，`charAt(index)` 方法具有一个参数，但是 `length()` 方法则无参。回顾曾经介绍过的，调用静态方法的语法是 `ClassName.methodName(arguments)`。例如，`Math` 类中的 `pow` 方法可以使用 `Math.pow(2,2.5)` 来调用。

4.4.1 求字符串长度

可以调用字符串的 `length()` 方法获取它的长度。例如，下面代码

```
String message = "Welcome to Java";
System.out.println("The length of " + message + " is "
    + message.length());
```

显示

The length of Welcome to Java is 15

注意：使用一个字符串时，往往是知道它的直接量值的。为方便起见，Java 允许在不创建新变量的情况下，使用字符串直接量直接引用字符串。这样，`"Welcome to Java".length()` 是正确的，它返回 15。注意，`""` 表示空字符串，并且 `"".length()` 为 0。

4.4.2 从字符串中获取字符

方法 `s.charAt(index)` 可用于提取字符串 `s` 中的某个特定字符，其中下标 `index` 的取值范围在 `0 ~ s.length()-1` 之间。例如，`message.charAt(0)` 返回字符 `W`，如图 4-1 所示。注意，字符串中第一个字符的下标值是 0。

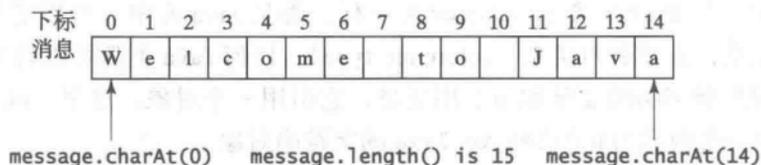


图 4-1 `String` 对象中的字符可以通过它的下标访问

警告：在字符串 `s` 中越界访问字符是一种常见的程序设计错误。为了避免此类错误，要确保使用的下标不会超过 `s.length()-1`。例如，`s.charAt(s.length())` 会造成一个 `StringIndexOutOfBoundsException` 异常。

4.4.3 连接字符串

可以使用 `concat` 方法连接两个字符串。例如，如下所示的语句将字符串 `s1` 和 `s2` 连接构成 `s3`：

```
String s3 = s1.concat(s2);
```

因为字符串连接在程序设计中应用非常广泛，所以 Java 提供了一种实现字符串连接的简便办法。可以使用加号 (+) 连接两个或多个字符串。因此，上面的语句等价于：

```
String s3 = s1 + s2;
```

下面的代码将字符串 `message`、`"and"` 和 `"HTML"` 组合成一个字符串：

```
String myString = message + " and " + "HTML";
```

回顾一下，加号 (+) 也可用于连接数字和字符串。在这种情况下，先将数字转换成字符串，然后再进行连接。注意，若要用加号实现连接功能，至少要有一个操作数必须为字符串。如果操作数之一不是字符串（比如，一个数字），非字符串值转换为字符串，并与另外一个字符串连接。这里是一些示例：

```
// Three strings are concatenated  
String message = "Welcome " + "to " + "Java";
```

```
// String Chapter is concatenated with number 2  
String s = "Chapter" + 2; // s becomes Chapter2
```

```
// String Supplement is concatenated with character B  
String s1 = "Supplement" + 'B'; // s1 becomes SupplementB
```

如果操作数都不是字符串，加号 (+) 是一个将两个数字相加的加法操作符。

增强的 += 操作符也可以用于字符串连接。例如，下面的代码将字符串 `"and Java is fun"` 添加到 `message` 变量中的字符串 `"Welcome to Java"` 后面。

```
message += " and Java is fun";
```

因此，新的 `message` 是 `"Welcome to Java and Java is fun"`。

如果 `i=1` 并且 `j=2`，下面语句的输出是什么？

```
System.out.println("i + j is " + i + j);
```

输出是 `"i+j is 12"`，因为 `"i+j is"` 首先和 `i` 的值连接。要强制 `i+j` 先执行，将 `i+j` 放在括号里，如下所示：

```
System.out.println("i + j is " + (i + j));
```

4.4.4 字符串的转换

方法 `toLowerCase()` 返回一个新字符串，其中所有字母小写；方法 `toUpperCase()` 返回一个新字符串，其中所有字母大写。例如，

"Welcome".toLowerCase() 返回一个新字符串 welcome。

"Welcome".toUpperCase() 返回一个新字符串 WELCOME。

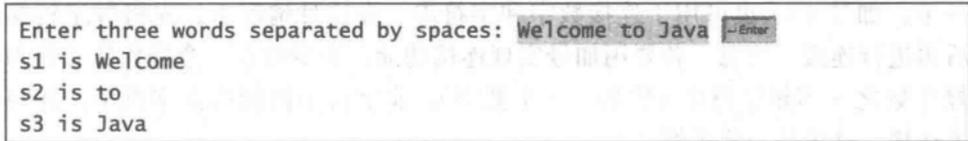
方法 trim() 通过删除字符串两端的空白字符返回一个新字符串。字符 ' '、\t、\f、\r、或者 \n 被称为空白字符。例如，

"\t Good Night \n".trim() 返回一个新字符串 Good Night。

4.4.5 从控制台读取字符串

为了从控制台读取字符串，调用 Scanner 对象上的 next() 方法。例如，下面的代码就可以从键盘读取三个字符串：

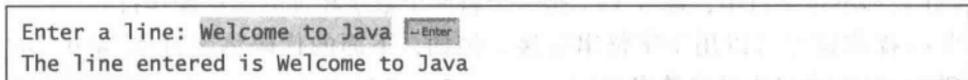
```
Scanner input = new Scanner(System.in);
System.out.print("Enter three words separated by spaces: ");
String s1 = input.next();
String s2 = input.next();
String s3 = input.next();
System.out.println("s1 is " + s1);
System.out.println("s2 is " + s2);
System.out.println("s3 is " + s3);
```



```
Enter three words separated by spaces: Welcome to Java
s1 is Welcome
s2 is to
s3 is Java
```

next() 方法读取以空白字符结束的字符串（即 ' '、'\t'、'\f'、'\r' 或 '\n'）。可以使用 nextLine() 方法读取一整行文本。nextLine() 方法读取以按下回车键为结束标志的字符串。例如，下面的语句读取一行文本：

```
Scanner input = new Scanner(System.in);
System.out.println("Enter a line: ");
String s = input.nextLine();
System.out.println("The line entered is " + s);
```



```
Enter a line: Welcome to Java
The line entered is Welcome to Java
```

重要警告：为了避免输入错误，不要在 nextByte()、nextShort()、nextInt()、nextLong()、nextFloat()、nextDouble() 和 next() 之后使用 nextLine()，原因将在 12.11.4 节中解释。

4.4.6 从控制台读取字符

为了从控制台读取字符，调用 nextLine() 方法读取一个字符串，然后在字符串上调用 charAt(0) 来返回一个字符。例如，下列代码从键盘读取一个字符：

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a character: ");
String s = input.nextLine();
char ch = s.charAt(0);
System.out.println("The character entered is " + ch);
```

4.4.7 字符串比较

String 类提供了如表 4-8 所示的方法，用于比较两个字符串。

表 4-8 String 对象的比较方法

方法	描述
<code>equals(s1)</code>	如果该字符串等于字符串 <code>s1</code> , 返回 <code>true</code>
<code>equalsIgnoreCase(s1)</code>	如果该字符串等于字符串 <code>s1</code> , 返回 <code>true</code> ; 不区分大小写
<code>compareTo(s1)</code>	返回一个大于 0、等于 0、小于 0 的整数, 表明一个字符串是否大于、等于或者小于 <code>s1</code>
<code>compareToIgnoreCase(s1)</code>	和 <code>compareTo</code> 一样, 除了比较是区分大小写的之外
<code>startsWith(prefix)</code>	如果字符串以特定的前缀开始, 返回 <code>true</code>
<code>endsWith(suffix)</code>	如果字符串以特定的后缀结束, 返回 <code>true</code>
<code>contains(s1)</code>	如果 <code>s1</code> 是该字符串的子字符串, 返回 <code>true</code>

如何比较两个字符串的内容是否相等呢? 你可能会尝试使用 `==` 操作符, 如下所示:

```
if (string1 == string2)
    System.out.println("string1 and string2 are the same object");
else
    System.out.println("string1 and string2 are different objects");
```

然而, 操作符 `==` 只能检测 `string1` 和 `string2` 是否指向同一个对象, 但它不会告诉你它们的内容是否相同。因此, 不能使用 `==` 操作符判断两个字符串变量的内容是否相同。取而代之, 应该使用 `equals` 方法。例如, 可以使用下面的代码比较两个字符串:

```
if (string1.equals(string2))
    System.out.println("string1 and string2 have the same contents");
else
    System.out.println("string1 and string2 are not equal");
```

例如, 下面的语句先显示 `true`, 然后显示 `false`。

```
String s1 = "Welcome to Java";
String s2 = "Welcome to Java";
String s3 = "Welcome to C++";
System.out.println(s1.equals(s2)); // true
System.out.println(s1.equals(s3)); // false
```

`compareTo` 方法也用来对两个字符串进行比较。例如, 考虑下述代码:

```
s1.compareTo(s2)
```

如果 `s1` 与 `s2` 相等, 那么该方法返回值 0; 如果按字典顺序 (即以 Unicode 码的顺序) `s1` 小于 `s2`, 那么方法返回值小于 0; 如果按字典顺序 `s1` 大于 `s2`, 方法返回值大于 0。

方法 `compareTo` 返回的实际值是依据 `s1` 和 `s2` 从左到右数第一个不同字符之间的距离得出的。例如, 假设 `s1` 为 "abc", `s2` 为 "abg", 那么 `s1.compareTo(s2)` 返回 -4。首先比较的是 `s1` 与 `s2` 中第一个位置的两个字符 (a 与 a)。因为它们相等, 所以比较第二个位置的两个字符 (b 与 b)。因为它们也相等, 所以比较第三个位置的两个字符 (c 与 g)。由于字符 c 比字符 g 小 4, 所以比较之后返回 -4。

 **警告:** 如果使用像 `>`、`>=`、`<` 或 `<=` 这样的比较操作符比较两个字符串, 就会发生语法错误。替代的方法就是使用 `s1.compareTo(s2)` 来进行比较。

 **注意:** 如果两个字符串相等, `equals` 方法返回 `true`; 如果它们不等, 方法返回 `false`。`compareTo` 方法会根据一个字符串是否等于、大于或小于另一个字符串, 分别返回 0、正整数或负整数。

`String` 类还提供了对字符串进行比较的方法 `equalsIgnoreCase` 和 `compareToIgnoreCase`。

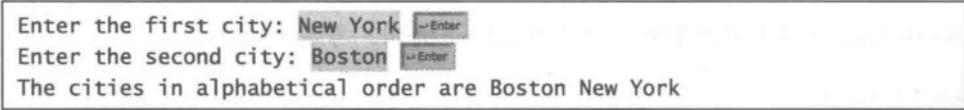
当比较两个字符串时，方法 `equalsIgnoreCase` 和 `compareToIgnoreCase` 忽略字母的大小写。还可以使用 `str.startsWith(prefix)` 来检测字符串 `str` 是否以特定前缀 (`prefix`) 开始，使用 `str.endsWith(suffix)` 来检测字符串 `str` 是否以特定后缀 (`suffix`) 结束，并且可以使用 `str.contains(s1)` 来检测是否字符串 `str` 包含字符串 `s1`。例如，

```
"Welcome to Java".startsWith("We") returns true.
"Welcome to Java".startsWith("we") returns false.
"Welcome to Java".endsWith("va") returns true.
"Welcome to Java".endsWith("v") returns false.
"Welcome to Java".contains("to") returns true.
"Welcome to Java".contains("To") returns false.
```

程序清单 4-2 给出了一个程序，提示用户输入两个城市，然后以字母表顺序进行显示。

程序清单 4-2 OrderTwoCities.java

```
1 import java.util.Scanner;
2
3 public class OrderTwoCities {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6
7         // Prompt the user to enter two cities
8         System.out.print("Enter the first city: ");
9         String city1 = input.nextLine();
10        System.out.print("Enter the second city: ");
11        String city2 = input.nextLine();
12
13        if (city1.compareTo(city2) < 0)
14            System.out.println("The cities in alphabetical order are " +
15                city1 + " " + city2);
16        else
17            System.out.println("The cities in alphabetical order are " +
18                city2 + " " + city1);
19    }
20 }
```



```
Enter the first city: New York
Enter the second city: Boston
The cities in alphabetical order are Boston New York
```

程序读取为两个城市的两个字符串 (第 9、11 行)。如果把 `input.nextLine()` 替换成 `input.next()` (第 9 行)，你不能输入一个包含空格的字符串给 `city1`。因为一个城市名字可能包含被空格隔开的多个单词，所以程序使用 `nextLine` 方法来读取一个字符串 (第 9、11 行)。调用 `city1.compareTo(city2)` 比较两个字符串 `city1` 和 `city2` (第 13 行)。一个负的返回值表明 `city1` 小于 `city2`。

4.4.8 获得子字符串

方法 `s.charAt(index)` 可用于提取字符串 `s` 中的某个特定字符。也可以使用 `String` 类中的 `substring` 方法从字符串中提取子串，如表 4-9 所示。

例如，

```
String message = "Welcome to Java";
String message = message.substring(0, 11) + "HTML";
字符串 message 变成了 Welcome to HTML.
```

表 4-9 String 类包含的获取子串的方法

方法	描述
substring(beginIndex)	返回该字符串的子串，从特定位置 beginIndex 的字符开始到字符串的结尾，如图 4-2 所示
substring(beginIndex, endIndex)	返回该字符串的子串，从特定位置 beginIndex 的字符开始到下标为 endIndex-1 的字符，如图 4-2 所示。注意，位于 endIndex 位置的字符不属于该子字符串的一部分

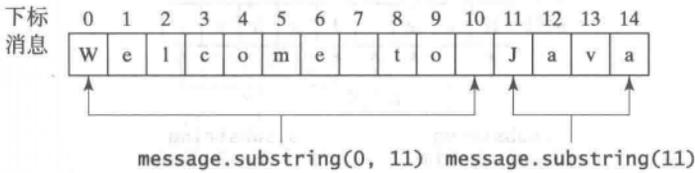


图 4-2 subString 方法从一个字符串中获得子串

4.4.9 获取字符串中的字符或者子串

String 类提供了几个版本的 indexOf 和 lastIndexOf 方法，它们可以在字符串中找出一个字符或一个子串，如表 4-10 所示。

表 4-10 String 类包含获取子串的方法

方法	描述
indexOf(ch)	返回字符串中出现的第一个 ch 的下标。如果没有匹配的，返回 -1
indexOf(ch, fromIndex)	返回字符串中 fromIndex 之后出现的第一个 ch 的下标。如果没有匹配的，返回 -1
indexOf(s)	返回字符串中出现的第一个字符串 s 的下标。如果没有匹配的，返回 -1
indexOf(s, fromIndex)	返回字符串中 fromIndex 之后出现的第一个字符串 s 的下标。如果没有匹配的，返回 -1
lastIndexOf(ch)	返回字符串中出现的最后一个 ch 的下标。如果没有匹配的，返回 -1
lastIndexOf(ch, fromIndex)	返回字符串中 fromIndex 之前出现的最后一个 ch 的下标。如果没有匹配的，返回 -1
lastIndexOf(s)	返回字符串中出现的最后一个字符串 s 的下标。如果没有匹配的，返回 -1
lastIndexOf(s, fromIndex)	返回字符串中 fromIndex 之前出现的最后一个字符串 s 的下标。如果没有匹配的，返回 -1

例如，

```

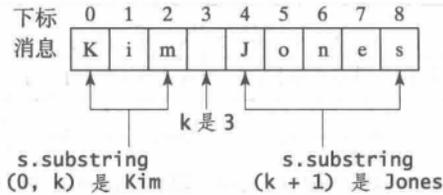
"Welcome to Java".indexOf('W') returns 0.
"Welcome to Java".indexOf('o') returns 4.
"Welcome to Java".indexOf('o', 5) returns 9.
"Welcome to Java".indexOf("come") returns 3.
"Welcome to Java".indexOf("Java", 5) returns 11.
"Welcome to Java".indexOf("java", 5) returns -1.

"Welcome to Java".lastIndexOf('W') returns 0.
"Welcome to Java".lastIndexOf('o') returns 9.
"Welcome to Java".lastIndexOf('o', 5) returns 4.
"Welcome to Java".lastIndexOf("come") returns 3.
"Welcome to Java".lastIndexOf("Java", 5) returns -1.
"Welcome to Java".lastIndexOf("Java") returns 11.
    
```

假设一个字符串 s 包含使用空格分开的姓和名。可以使用下面的代码从字符串中抽取姓和名。

```
int k = s.indexOf(' ');
String firstName = s.substring(0, k);
String lastName = s.substring(k + 1);
```

例如，如果 s 是 Kim Jones，下图显示了如何抽取出姓和名。



4.4.10 字符串和数字间的转换

可以将数值型字符串转换为数值。要将字符串转换为 `int` 值，使用 `Integer.parseInt` 方法，如下所示：

```
int intValue = Integer.parseInt(intString);
```

`intString` 是一个数值型字符串，例如 "123"。

要将字符串转换为 `double` 值，使用 `Double.parseDouble` 方法，如下所示：

```
double doubleValue = Double.parseDouble(doubleString);
```

`doubleString` 是一个数值型字符串，例如 "123.45"。

如果字符串不是数值型字符串，转换将导致一个运行时错误。`Integer` 和 `Double` 类都包含在 `java.lang` 包中，因此它们是自动导入的。

可以将数值转换为字符串，只需要简单使用字符串的连接操作符，如下所示：

```
String s = number + "";
```

复习题

4.16 假设 `s1`、`s2` 和 `s3` 是三个字符串，给定如下语句：

```
String s1 = "Welcome to Java";
String s2 = "Programming is fun";
String s3 = "Welcome to Java";
```

下列表达式的结果是什么？

- `s1 == s2`
- `s2 == s3`
- `s1.equals(s2)`
- `s1.equals(s3)`
- `s1.compareTo(s2)`
- `s2.compareTo(s3)`
- `s2.compareTo(s2)`
- `s1.charAt(0)`
- `s1.indexOf('j')`
- `s1.indexOf("to")`
- `s1.lastIndexOf('a')`
- `s1.lastIndexOf("o", 15)`
- `s1.length()`
- `s1.substring(5)`
- `s1.substring(5, 11)`
- `s1.startsWith("Wel")`
- `s1.endsWith("Java")`
- `s1.toLowerCase()`
- `s1.toUpperCase()`
- `s1.concat(s2)`
- `s1.contains(s2)`
- `"\t Wel \t".trim()`

4.17 假设 s1、s2 是两个字符串，以下语句或者表达式中哪些是错误的？

```
String s = "Welcome to Java";
String s3 = s1 + s2;
String s3 = s1 - s2;
s1 == s2;
s1 >= s2;
s1.compareTo(s2);
int i = s1.length();
char c = s1(0);
char c = s1.charAt(s1.length());
```

4.18 给出下列语句的输出（编写程序验证你的结果）。

```
System.out.println("1" + 1);
System.out.println('1' + 1);
System.out.println("1" + 1 + 1);
System.out.println("1" + (1 + 1));
System.out.println('1' + 1 + 1);
```

4.19 对下列表达式求值（编写程序验证你的结果）。

```
1 + "Welcome " + 1 + 1
1 + "Welcome " + (1 + 1)
1 + "Welcome " + ('\u0001' + 1)
1 + "Welcome " + 'a' + 1
```

4.20 假设 s1 是 "Welcome" 而 s2 是 "welcome"，为下面的陈述编写代码：

- 检查 s1 和 s2 是否相等，然后将结果赋值给一个布尔变量 isEqual。
- 在忽略大小写的情况下检查 s1 和 s2 是否相等，然后将结果赋值给一个布尔变量 isEqual。
- 比较 s1 和 s2，然后将结果赋值给一个整型变量 x。
- 在忽略大小写的情况下比较 s1 和 s2，然后将结果赋值给一个整型变量 x。
- 检查 s1 是否有前缀 "AAA"，然后将结果赋值给一个布尔变量 b。
- 检查 s1 是否有后缀 "AAA"，然后将结果赋值给一个布尔变量 b。
- 将 s1 的长度赋值给一个整型变量 x。
- 将 s1 的第一个字符赋值给一个字符型变量 x。
- 创建新字符串 s3，它是 s1 和 s2 的组合。
- 创建 s1 的子串，下标从 1 开始。
- 创建 s1 的子串，下标从 1 到 4。
- 创建新字符串 s3，它将 s1 转换为小写。
- 创建新字符串 s3，它将 s1 转换为大写。
- 创建新字符串 s3，它将 s1 两端的空白字符去掉。
- 将 s1 中第一次出现的字符 e 的下标赋值给一个 int 型变量 x。
- 将 s1 中最后一次出现的字符串 abc 的下标赋值给一个 int 型变量 x。

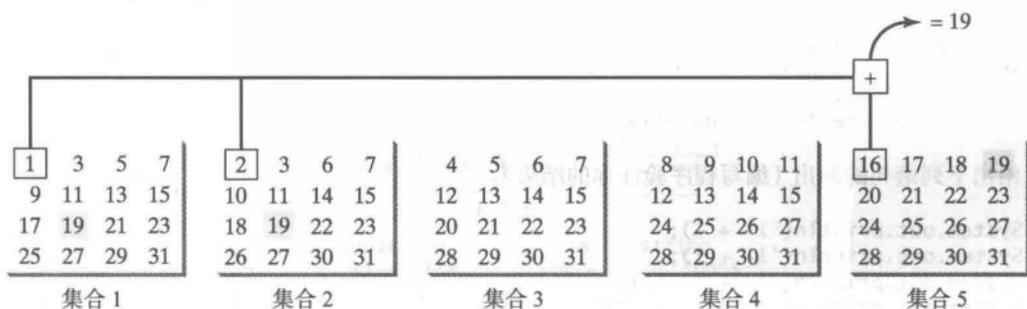
4.5 示例学习

 **要点提示：**字符串在编程中是非常基础的内容。使用字符串进行编程的能力对于学习 Java 编程非常关键。

你将经常使用字符串来编写有用的程序。本节提供三个使用字符串来求解问题的示例。

4.5.1 猜测生日

可以通过询问朋友 5 个问题，找到他出生在一个月的哪一天。每个问题都询问生日是否是 5 个数字集合中的一个。



生日是出现这一天的每个集合的第一个数字的和。例如：如果生日是 19，那么它会出现在集合 1、集合 2 和集合 5 中。这三个集合的第一个数字分别是 1、2 和 16。它们的和就是 19。

程序清单 4-3 给出程序，提示用户回答该天是否在集合 1 中（第 41 ~ 44 行），是否在集合 2 中（第 50 ~ 53 行），是否在集合 3 中（第 59 ~ 62 行），是否在集合 4 中（第 68 ~ 71 行），是否在集合 5 中（第 77 ~ 80 行）。如果这个数字在某个集合中，程序就将该集合的第一个数字加到 day 中去（第 47、56、65、74、83 行）。

程序清单 4-3 GuessBirthday.java

```

1 import java.util.Scanner;
2
3 public class GuessBirthday {
4     public static void main(String[] args) {
5         String set1 =
6             " 1 3 5 7\n" +
7             " 9 11 13 15\n" +
8             "17 19 21 23\n" +
9             "25 27 29 31";
10
11        String set2 =
12            " 2 3 6 7\n" +
13            "10 11 14 15\n" +
14            "18 19 22 23\n" +
15            "26 27 30 31";
16
17        String set3 =
18            " 4 5 6 7\n" +
19            "12 13 14 15\n" +
20            "20 21 22 23\n" +
21            "28 29 30 31";
22
23        String set4 =
24            " 8 9 10 11\n" +
25            "12 13 14 15\n" +
26            "24 25 26 27\n" +
27            "28 29 30 31";
28
29        String set5 =
30            "16 17 18 19\n" +
31            "20 21 22 23\n" +
32            "24 25 26 27\n" +

```

```
33     "28 29 30 31";
34
35     int day = 0;
36
37     // Create a Scanner
38     Scanner input = new Scanner(System.in);
39
40     // Prompt the user to answer questions
41     System.out.print("Is your birthday in Set1?\n");
42     System.out.print(set1);
43     System.out.print("\nEnter 0 for No and 1 for Yes: ");
44     int answer = input.nextInt();
45
46     if (answer == 1)
47         day += 1;
48
49     // Prompt the user to answer questions
50     System.out.print("\nIs your birthday in Set2?\n");
51     System.out.print(set2);
52     System.out.print("\nEnter 0 for No and 1 for Yes: ");
53     answer = input.nextInt();
54
55     if (answer == 1)
56         day += 2;
57
58     // Prompt the user to answer questions
59     System.out.print("Is your birthday in Set3?\n");
60     System.out.print(set3);
61     System.out.print("\nEnter 0 for No and 1 for Yes: ");
62     answer = input.nextInt();
63
64     if (answer == 1)
65         day += 4;
66
67     // Prompt the user to answer questions
68     System.out.print("\nIs your birthday in Set4?\n");
69     System.out.print(set4);
70     System.out.print("\nEnter 0 for No and 1 for Yes: ");
71     answer = input.nextInt();
72
73     if (answer == 1)
74         day += 8;
75
76     // Prompt the user to answer questions
77     System.out.print("\nIs your birthday in Set5?\n");
78     System.out.print(set5);
79     System.out.print("\nEnter 0 for No and 1 for Yes: ");
80     answer = input.nextInt();
81
82     if (answer == 1)
83         day += 16;
84
85     System.out.println("\nYour birthday is " + day + "!");
86 }
87 }
```

Is your birthday in Set1?

1 3 5 7

9 11 13 15

17 19 21 23

25 27 29 31

Enter 0 for No and 1 for Yes: 1

```

Is your birthday in Set2?
 2 3 6 7
10 11 14 15
18 19 22 23
26 27 30 31
Enter 0 for No and 1 for Yes: 1 --Enter

Is your birthday in Set3?
 4 5 6 7
12 13 14 15
20 21 22 23
28 29 30 31
Enter 0 for No and 1 for Yes: 0 --Enter

Is your birthday in Set4?
 8 9 10 11
12 13 14 15
24 25 26 27
28 29 30 31
Enter 0 for No and 1 for Yes: 0 --Enter

Is your birthday in Set5?
16 17 18 19
20 21 22 23
24 25 26 27
28 29 30 31
Enter 0 for No and 1 for Yes: 1 --Enter
Your birthday is 19!

```

line#	day	answer	output
35	0		
44		1	
47	1		
53		1	
56	3		
62		0	
71		0	
80		1	
83	19		
85			Your birthday is 19!

这个游戏是很容易编程的。你可能想知道如何创建这个游戏。实际上，这个游戏背后的数学知识是非常简单的。这些数字不是随意组成一组的。它们放在5个集合中的方式是经过深思熟虑的。这5个集合的第一个数分别是1、2、4、8和16，它们分别对应二进制数的1、10、100、1000和10000（二进制数在附录F中介绍）。从1到31的十进制数最多用5个二进制数就可以表示，如图4-3a所示。假设它是 $b_5b_4b_3b_2b_1$ ，那么 $b_5b_4b_3b_2b_1 = b_50000 + b_4000 + b_300 + b_20 + b_1$ ，如图4-3b所示。如果某天的二进制数在 b_k 位为数1，那么该数就该出现在Set k 中。例如：数字19的二进制形式是10011，所以它就该出现在集合1、集合2和集合

5 中。它就是二进制数 $1+10+10000=10011$ 或者十进制数 $1+2+16=19$ 。数字 31 的二进制形式是 11111，所以它就会出现在集合 1、集合 2、集合 3、集合 4 和集合 5 中。它就是二进制数 $1+10+100+1000+10000=11111$ ，或者十进制数 $1+2+4+8+16=31$ 。

十进制	二进制
1	00001
2	00010
3	00011
...	
19	10011
...	
31	11111

b_5 0 0 0 0		10000
b_4 0 0 0		1000
b_3 0 0	10000	100
b_2 0	10	10
b_1	$+$ $\frac{1}{10011}$	$+$ $\frac{1}{11111}$
$b_5 b_4 b_3 b_2 b_1$	19	31

a) 从 1 到 31 的数字可以用 5 位二进制数表示

b) 通过添加二进制数 1、10、100、1000 或者 10000 得到 5 位二进制数

图 4-3

复习题

4.21 如果运行程序清单 4-3，对于 Set1, Set3, Set4 输入 1，对于 Set2 和 Set5 输入 0，生日将是哪一天？

4.5.2 将十六进制数转换为十进制数

十六进制记数系统有 16 个数字：0~9, A~F。字母 A、B、C、D、E 和 F 对应于十进制数字 10、11、12、13、14 和 15。我们现在写一个程序，提示用户输入一个十六进制数字，显示它对应的十进制数，如程序清单 4-4 所示。

程序清单 4-4 HexDigit2Dec.java

```

1 import java.util.Scanner;
2
3 public class HexDigit2Dec {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6         System.out.print("Enter a hex digit: ");
7         String hexString = input.nextLine();
8
9         // Check if the hex string has exactly one character
10        if (hexString.length() != 1) {
11            System.out.println("You must enter exactly one character");
12            System.exit(1);
13        }
14
15        // Display decimal value for the hex digit
16        char ch = Character.toUpperCase(hexString.charAt(0));
17        if (ch <= 'F' && ch >= 'A') {
18            int value = ch - 'A' + 10;
19            System.out.println("The decimal value for hex digit "
20                + ch + " is " + value);
21        }
22        else if (Character.isDigit(ch)) {
23            System.out.println("The decimal value for hex digit "
24                + ch + " is " + ch);
25        }
26        else {
27            System.out.println(ch + " is an invalid input");
28        }
29    }
30 }

```

```
Enter a hex digit: AB7C ↵
You must enter exactly one character
```

```
Enter a hex digit: B ↵
The decimal value for hex digit B is 11
```

```
Enter a hex digit: 8 ↵
The decimal value for hex digit 8 is 8
```

```
Enter a hex digit: T ↵
T is an invalid input
```

程序从控制台读取一个字符串（第7行），检测该字符串是否仅包含一个字符（第10行）。如果不是，报告一个错误，然后退出程序（第12行）。

程序调用 `Character.toUpperCase` 方法得到大写字母 `ch`（第16行）。如果 `ch` 在 'A' ~ 'F'（第17行）之间，对应的十进制值为 `ch-'A'+10`（第18行）。注意，如果 `ch` 为 'A'，则 `ch-'A'` 为 0；如果 `ch` 为 'B'，则 `ch-'A'` 为 1，依次类推。当两个字符执行数值运算的时候，计算中使用的是字符的 Unicode 码。

程序调用 `Character.isDigit(ch)` 方法来检测 `ch` 是否在 '0' ~ '9' 之间（第22行）。如果在，对应的十进制数和 `ch` 相同（第23 ~ 24行）。

如果 `ch` 不在 'A' ~ 'F' 之间，或者不是一个数字字符，程序显示一个错误消息（第27行）。

4.5.3 使用字符串修改彩票程序

程序清单 3-8 中的彩票程序产生一个随机的两位数字，提示用户输入一个两位数字，根据以下规则确定用户是否中彩票：

- 1) 如果用户输入的数字完全匹配彩票中的数字，奖金为 10 000 美元。
- 2) 如果用户输入的所有数字匹配彩票中的所有数字，奖金为 3000 美元。
- 3) 如果用户输入的一个数字匹配彩票中的一个数字，奖金为 1000 美元。

程序清单 3-8 中的程序使用整数来存储数值。程序清单 4-5 给出一个新的程序，产生一个随机的两位字符串，而不是数字，并且使用字符串而不是数字来接收用户输入。

程序清单 4-5 LotteryUsingStrings.java

```
1 import java.util.Scanner;
2
3 public class LotteryUsingStrings {
4     public static void main(String[] args) {
5         // Generate a lottery as a two-digit string
6         String lottery = "" + (int)(Math.random() * 10)
7             + (int)(Math.random() * 10);
8
9         // Prompt the user to enter a guess
10        Scanner input = new Scanner(System.in);
11        System.out.print("Enter your lottery pick (two digits): ");
12        String guess = input.nextLine();
13
14        // Get digits from lottery
15        char lotteryDigit1 = lottery.charAt(0);
```

```
16     char lotteryDigit2 = lottery.charAt(1);
17
18     // Get digits from guess
19     char guessDigit1 = guess.charAt(0);
20     char guessDigit2 = guess.charAt(1);
21
22     System.out.println("The lottery number is " + lottery);
23
24     // Check the guess
25     if (guess.equals(lottery))
26         System.out.println("Exact match: you win $10,000");
27     else if (guessDigit2 == lotteryDigit1
28             && guessDigit1 == lotteryDigit2)
29         System.out.println("Match all digits: you win $3,000");
30     else if (guessDigit1 == lotteryDigit1
31             || guessDigit1 == lotteryDigit2
32             || guessDigit2 == lotteryDigit1
33             || guessDigit2 == lotteryDigit2)
34         System.out.println("Match one digit: you win $1,000");
35     else
36         System.out.println("Sorry, no match");
37 }
38 }
```

```
Enter your lottery pick (two digits): 00 
The lottery number is 00
Exact match: you win $10,000
```

```
Enter your lottery pick (two digits): 45 
The lottery number is 54
Match all digits: you win $3,000
```

```
Enter your lottery pick: 23 
The lottery number is 34
Match one digit: you win $1,000
```

```
Enter your lottery pick: 23 
The lottery number is 14
Sorry: no match
```

程序产生两个随机数字，并且将它们连接成一个字符串 `lottery`（第 6 ~ 7 行）。这样，`lottery` 包含两个随机数字。

程序提示用户以两位字符串形式输入一个猜测值（第 12 行），并且按照以下顺序，对照彩票数字检测用户的猜测值：

- 首先检测给出的猜测值是否完全匹配彩票（第 25 行）。
- 如果不匹配，检测猜测值的逆序是否匹配彩票（第 27 行）。
- 如果不匹配，检测是否有一个数字在彩票中（第 30 ~ 33 行）。
- 如果以上条件都不成立，显示“Sorry, no match”（第 36 行）。

4.6 格式化控制台输出

 **要点提示：** 可以使用 `System.out.printf` 方法在控制台上显示格式化输出。

许多情况下会希望以某一种格式来显示数值。例如，下面的代码在给定金额和年利率的情况下，计算利息。

```
double amount = 12618.98;
double interestRate = 0.0013;
double interest = amount * interestRate;
System.out.println("Interest is $" + interest);
```

```
Interest is $16.404674
```

因为利息额度是现金，所以一般希望仅显示浮点值小数点后两位数字，于是可以如下编写代码：

```
double amount = 12618.98;
double interestRate = 0.0013;
double interest = amount * interestRate;
System.out.println("Interest is $"
    + (int)(interest * 100) / 100.0);
```

```
Interest is $16.4
```

然而，格式依然不正确。这里应该在小数点后是两位小数：16.40，而不是 16.4。可以通过使用 printf 方法来修正这个问题，如下：

```
double amount = 12618.98;
double interestRate = 0.0013;
double interest = amount * interestRate;
System.out.printf("Interest is $%4.2f",
    interest);
```

```
Interest is $16.40
```

`%4.2f` ← 格式标识符

域宽度 转换码

精度

调用这个方法的语法是：

```
System.out.printf(format, item1, item2, ..., itemk)
```

这里的 format 是指一个由子串和格式标识符构成的字符串。

格式标识符指定每个条目应该如何显示。这里的条目可以是数值、字符、布尔值或字符串。简单的格式标识符是以百分号 (%) 开头的转换码。表 4-11 列出了一些常用的简单格式标识符。

表 4-11 常用的格式标识符

标识符	输出	举例	标识符	输出	举例
%b	布尔值	true 或 false	%f	浮点数	45.460000
%c	字符	'a'	%e	标准科学记数法形式的数	4.556 000e+01
%d	十进制整数	200	%s	字符串	"Java is cool"

下面是一个例子：

```
int count = 5;
double amount = 45.56;
System.out.printf("count is %d and amount is %f", count, amount);
```

display count is 5 and amount is 45.560000

条目与标识符必须在次序、数量和类型上匹配。例如：`count` 的格式标识符应该是 `%d`，而 `amount` 的格式标识符应该是 `%f`。默认情况下，浮点值显示小数点后 6 位数字。可以在标识符中指定宽度和精度，如表 4-12 中的例子所示。

表 4-12 指定宽度和精度的例子

举例	输出
<code>%5c</code>	输出字符并在这个字符条目前面加 4 个空格
<code>%6b</code>	输出布尔值，在 <code>false</code> 值前加一个空格，在 <code>true</code> 值前加两个空格
<code>%5d</code>	输出整数条目，宽度至少为 5。如果该条目的数字位数小于 5，就在数字前面加空格。如果该条目的位数大于 5，则自动增加宽度
<code>%10.2f</code>	输出的浮点条目宽度至少为 10，包括小数点和小数点后两位数字。这样，给小数点前分配了 7 位数字。如果该条目小数点前的位数小于 7，就在数字前面加空格。如果该条目小数点前的位数大于 7，则自动增加宽度
<code>%10.2e</code>	输出的浮点条目的宽度至少为 10，包括小数点、小数点后两位数字和指数部分。如果按科学记数法显示的数字小于 10，就给数字前加空格
<code>%12s</code>	输出的字符串宽度至少为 12 个字符。如果该字符串条目小于 12 个字符，就在该字符串前加空格。如果该字符串条目多于 12 个字符，则自动增加宽度

如果一个条目需要比指定宽度更多的空间，宽度自动增加。例如，下面的代码：

```
System.out.printf("%3d#%2s#%4.2f\n", 1234, "Java", 51.6653);
```

显示

```
1234#Java#51.67
```

为 `int` 条目 `1234` 指定的宽度是 3，而这个小于它的实际大小 4，宽度将自动增加到 4。为字符串条目 `Java` 指定的宽度为 2，而这个小于它的实际大小 4，宽度将自动增加到 4。为 `double` 类型条目 `51.6653` 指定的宽度为 4，但是它需要宽度 5 来显示 `51.67`，所以宽度自动增加到 5。

默认情况下，输出是右对齐的。可以在格式标识符中放一个负号 (-)，表明该条目在特定区域中的输出是左对齐的。例如，以下语句：

```
System.out.printf("%8d%8s%8.1f\n", 1234, "Java", 5.63);
System.out.printf("%-8d%-8s%-8.1f\n", 1234, "Java", 5.63);
```

显示

```

|← 8 →|← 8 →|← 8 →|
□□□ 1234 □□□ Java □□□ 5.6
1234 □□□ Java □□□ 5.6 □□□
```

这里，方框表示一个空白区域。

警告： 条目与格式标识符必须在类型上严格匹配。对应于格式标识符 `%f` 或 `%e` 的条目必须是浮点型值，例如：是 `40.0` 而不是 `40`。因此，`int` 型变量不能匹配 `%f` 或 `%e`。

提示： 使用符号 `%` 来标记格式标识符，要在格式字符串里输出直接量 `%`，需要使用 `%%`。

程序清单 4-6 给出一个使用 `printf` 来显示一个表格的程序。

程序清单 4-6 FormatDemo.java

```

1 public class FormatDemo {
2     public static void main(String[] args) {
3         // Display the header of the table
4         System.out.printf("%-10s%-10s%-10s%-10s%-10s\n", "Degrees",
5             "Radians", "Sine", "Cosine", "Tangent");
```

```

6
7 // Display values for 30 degrees
8 int degrees = 30;
9 double radians = Math.toRadians(degrees);
10 System.out.printf("%-10d%-10.4f%-10.4f%-10.4f%-10.4f\n", degrees,
11 radians, Math.sin(radians), Math.cos(radians),
12 Math.tan(radians));
13
14 // Display values for 60 degrees
15 degrees = 60;
16 radians = Math.toRadians(degrees);
17 System.out.printf("%-10d%-10.4f%-10.4f%-10.4f%-10.4f\n", degrees,
18 radians, Math.sin(radians), Math.cos(radians),
19 Math.tan(radians));
20 }
21 }

```

Degrees	Radians	Sine	Cosine	Tangent
30	0.5236	0.5000	0.8660	0.5773
60	1.0472	0.8660	0.5000	1.7320

第4~5行的语句显示表格的列名。列名是字符串。使用格式标识符%-10s来显示字符串，对字符串进行左对齐。第10~12行的语句以整数显示度数以及4个单精度浮点数。使用格式标识符%-10d来显示整数，以及使用格式标识符%-10.4f来显示单精度浮点数，来指定小数点后有四位数字。

复习题

- 4.22 输出布尔值、字符、十进制整数、浮点数和字符串的格式标识符分别是什么？
- 4.23 下面的语句错在哪里？
- System.out.printf("%5d %d", 1, 2, 3);
 - System.out.printf("%5d %f", 1);
 - System.out.printf("%5d %f", 1, 2);
- 4.24 给出下面语句的输出。
- System.out.printf("amount is %f %e\n", 32.32, 32.32);
 - System.out.printf("amount is %5.2% %5.4e\n", 32.327, 32.32);
 - System.out.printf("%6b\n", (1 > 2));
 - System.out.printf("%6s\n", "Java");
 - System.out.printf("%-6b%s\n", (1 > 2), "Java");
 - System.out.printf("%6b%-8s\n", (1 > 2), "Java");

关键术语

char type (char 类型)

encoding (编码)

escape character (转义字符)

escape sequence (转义序列)

format specifier (格式标识符)

instance method (实例方法)

static method (静态方法)

supplementary Unicode (补充 Unicode 码)

Unicode (Unicode 码)

whitespace character (空白字符)

本章小结

1. Java 提供了在 Math 类中的数学方法 `sin`、`cos`、`tan`、`asin`、`acos`、`atan`、`toRadians`、`toDegree`、`exp`、`log`、`log10`、`pow`、`sqrt`、`ceil`、`floor`、`rint`、`round`、`min`、`max`、`abs` 以及 `random`，用于执行数学函数。
2. 字符类型 `char` 表示单个字符。
3. 转义序列包含反斜杠 `\` 以及后面的字符或者数字组合。
4. 字符 `\` 称为转义字符。
5. 字符 `' '`、`\t`、`\f`、`\r` 和 `\n` 都称为空白字符。
6. 字符可以基于它们的 Unicode 码使用关系操作符进行比较。
7. `Character` 类包含方法 `isDigit`、`isLetter`、`isLetterOrDigit`、`isLowerCase`、`isUpperCase`，用于判断一个字符是否是数字、字母、小写字母还是大写字母。它也包含 `toLowerCase` 和 `toUpperCase` 方法返回小写或大写字母。
8. 字符串是一个字符序列。字符串的值包含在一对匹配的双引号 (`"`) 中。字符的值包含在一对匹配的单引号 (`'`) 中。
9. 字符串在 Java 中是对象。只能通过一个指定对象调用的方法称为实例方法。非实例方法称为静态方法，可以不使用对象来调用。
10. 可以调用字符串的 `length()` 方法获取它的长度，使用 `charAt(index)` 方法从字符串中提取特定下标位置的字符，使用 `indexOf` 和 `lastIndexOf` 方法找出一个字符串中的某个字符或某个子串。
11. 可以使用 `concat` 方法连接两个字符串，或者使用加号 (`+`) 连接两个或多个字符串。
12. 可以使用 `substring` 方法从字符串中提取子串。
13. 可以使用 `equals` 和 `compareTo` 方法比较字符串。如果两个字符串相等，`equals` 方法返回 `true`；如果它们不等，则返回 `false`。`compareTo` 方法根据一个字符串等于、大于或小于另一个字符串，分别返回 0、正整数或负整数。
14. `printf` 方法使用格式标识符来显示一个格式化的输出。

测试题

本章测试题的答案参见 www.cs.armstrong.edu/liang/intro10e/quiz.html。

编程练习题

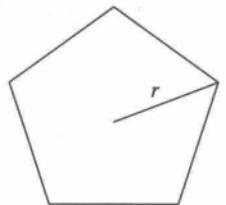
4.2 节

- 4.1 (几何：五边形的面积) 编写程序，提示用户输入从五边形中心到顶点的距离，计算五边形的面积，如右图所示。

计算五边形面积的公式为：面积 = $\frac{5 \times s^2}{4 \times \tan\left(\frac{\pi}{5}\right)}$ ，其中 s 是边长。边长可以

使用公式 $s = 2r \sin\left(\frac{\pi}{5}\right)$ 计算，其中 r 是从五边形中心到顶点的距离。结果

保留小数点后两位数字。下面是一个运行示例：



```
Enter the length from the center to a vertex: 5.5 ~Enter
The area of the pentagon is 71.92
```

- *4.2 (几何：最大圆距离) 最大圆距离是指球面上两个点之间的距离。假设 (x_1, y_1) 和 (x_2, y_2) 是两个点的地理经纬度。两个点之间的最大圆距离可以使用以下公式计算：

$$d = \text{半径} \times \arccos(\sin(x_1) \times \sin(x_2) + \cos(x_1) \times \cos(x_2) \times \cos(y_1 - y_2))$$

编写一个程序，提示用户以度为单位输入地球上两个点的经纬度，显示其最大圆距离值。地球的平均半径为 6 371.01km。注意，你需要使用 `Math.toRadians` 方法将度转换为弧度值。公式中的经纬度是相对北边和西边的，使用负数表示相对南边和东边的度数。下面是一个运行示例：

```
Enter point 1 (latitude and longitude) in degrees: 39.55, -116.25 --Enter
Enter point 2 (latitude and longitude) in degrees: 41.5, 87.37 --Enter
The distance between the two points is 10691.79183231593 km
```

- *4.3 (几何：估算面积) 从网址 www.gps-data-tem.com/map 上面找到 Georgia 州的 Atlanta、Florida 州的 Orlando、Georgia 州的 Savannah、North Carolina 的 Charlotte，计算被这四个城市所围起来的区域的面积。(提示：使用编程练习题 4.2 中的公式来计算两个城市之间的距离。将多边形分为两个三角形，使用编程练习题 2.19 中的公式计算三角形面积。)
- 4.4 (几何：六边形面积) 六边形面积可以通过下面公式计算 (s 是边长)：

$$\text{面积} = \frac{6 \times s^2}{4 \times \tan\left(\frac{\pi}{6}\right)}$$

编写程序，提示用户输入六边形的边长，然后显示它的面积。下面是一个运行示例：

```
Enter the side: 5.5 --Enter
The area of the hexagon is 78.59
```

- *4.5 (几何：正多边形的面积) 正多边形是一个 n 条边的多边形，它每条边的长度都相等，而且所有角的度数也相等 (即多边形既等边又等角)。计算正多边形面积的公式是：

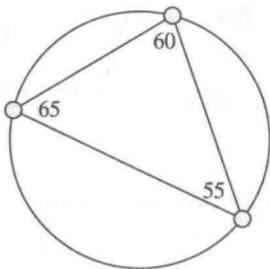
$$\text{面积} = \frac{n \times s}{4 \times \tan\left(\frac{\pi}{n}\right)}$$

这里， s 是边长。编写一个程序，提示用户输入边的个数以及正多边形的边长，然后显示它的面积。这里是一个运行示例：

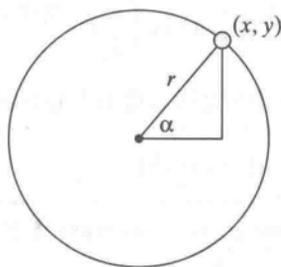
```
Enter the number of sides: 5 --Enter
Enter the side: 6.5 --Enter
The area of the polygon is 72.69017017488385
```

- *4.6 (圆上的随机点) 编写一个程序，产生一个圆心在 $(0, 0)$ 、半径为 40 的圆上面的三个随机点，显示由这三个随机点组成的三角形的三个角的度数，如图 4-4a 所示。(提示：产生 $0 \sim 2\pi$ 之间的一个以弧度为单位的随机角度 α ，如图 4-4b 所示，则由这个角度所确定的点为 $(r \times \cos(\alpha), r \times \sin(\alpha))$ 。)

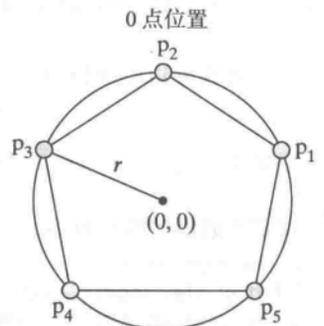
$$x = r \times \cos(\alpha) \text{ and } y = r \times \sin(\alpha)$$



a) 由圆上三个随机点构成的三角形



b) 可以从一个随机角度 α 产生圆上的随机点



c) 一个正五边形，其中心位于 $(0, 0)$ ，其中一个点位于 0 点位置

图 4-4

- *4.7 (顶点坐标) 假设一个正五边形的中心位于 (0, 0), 其中一个点位于 0 点位置, 如图 4-4c 所示。编写一个程序, 提示用户输入正五边形外切圆的半径, 显示正五边形上五个顶点的坐标。这里是一个运行示例:

```
Enter the radius of the bounding circle: 100 ↵
The coordinates of five points on the pentagon are
(95.1057, 30.9017)
(0.000132679, 100)
(-95.1056, 30.9019)
(-58.7788, -80.9015)
(58.7782, -80.902)
```

4.3 ~ 4.6 节

- *4.8 (给出 ASCII 码对应的字符) 编写一个程序, 得到一个 ASCII 码的输入 (0 ~ 127 之间的一个整数), 然后显示该字符。下面是一个运行示例:

```
Enter an ASCII code: 69 ↵
The character for ASCII code 69 is E
```

- *4.9 (给出字符的 Unicode 码) 编写一个程序, 得到一个字符的输入, 然后显示其 Unicode 值。下面是一个运行示例:

```
Enter a character: E ↵
The Unicode for the character E is 69
```

- *4.10 (猜测生日) 改写程序清单 4-3, 提示用户输入字符 Y 代表“是”, 输入 N 代表“不是”, 代替之前输入 1 表示“是”和 0 表示“不是”。

- *4.11 (十进制转十六进制) 编写一个程序, 提示用户输入 0 ~ 15 之间的一个整数, 显示其对应的十六进制数。下面是一个运行示例:

```
Enter a decimal value (0 to 15): 11 ↵
The hex value is B
```

```
Enter a decimal value (0 to 15): 5 ↵
The hex value is 5
```

```
Enter a decimal value (0 to 15): 31 ↵
31 is an invalid input
```

- 4.12 (十六进制转二进制) 编写一个程序, 提示用户输入一个十六进制数, 显示其对应的二进制数。下面是一个运行示例:

```
Enter a hex digit: B ↵
The binary value is 1011
```

```
Enter a hex digit: G ↵
G is an invalid input
```

- *4.13 (判断元音还是辅音) 编写一个程序, 提示用户输入一个字母, 判断该字母是元音还是辅音。下面是一个运行示例:

```
Enter a letter: B ↵
B is a consonant
```

```
Enter a letter grade: a 
a is a vowel
```

```
Enter a letter grade: # 
# is an invalid input
```

- *4.14 (转换字母等级为数字) 编写一个程序, 提示用户输入一个字母等级 A、B、C、D 或者 F, 显示对应的数字值 4、3、2、1 或者 0。下面是一个运行示例:

```
Enter a letter grade: B 
The numeric value for grade B is 3
```

```
Enter a letter grade: T 
T is an invalid grade
```

- *4.15 (电话键盘) 电话上的国际标准字母/数字映射如下所示:



编写一个程序, 提示用户输入一个字母, 然后显示对应的数字。

```
Enter a letter: A 
The corresponding number is 2
```

```
Enter a letter: a 
The corresponding number is 2
```

```
Enter a letter: + 
+ is an invalid input
```

- 4.16 (随机字符) 编写一个程序, 使用 `Math.random()` 方法显示一个随机的大写字母。

- *4.17 (一个月中的日期) 编写一个程序, 提示用户输入一个年份和一个月份名称的前三个字母 (第一个字母使用大写形式), 显示该月中的天数。下面是一个运行示例:

```
Enter a year: 2001 
Enter a month: Jan 
Jan 2001 has 31 days
```

```
Enter a year: 2016 
Enter a month: Feb 
Jan 2016 has 29 days
```

- *4.18 (学生的专业和状况) 编写一个程序, 提示用户输入两个字符, 显示这两个字符代表的专业以及状况。第一个字符表示专业, 第二个是一个数字字符 1、2、3、4, 分别表示该学生是大一、大二、大三还是大四。假设下面的字符用于表示专业:

M: 数学

C: 计算机科学

I: 信息技术

下面是一个运行示例:

```
Enter two characters: M1 ↵ Enter
Mathematics Freshman
```

```
Enter two characters: C3 ↵ Enter
Computer Science Junior
```

```
Enter two characters: T3 ↵ Enter
Invalid input
```

4.19 (商业: 检测 ISBN-10) 改写编程练习题 3.9, 将 ISBN 号作为一个字符串输入。

4.20 (字符串处理) 编写一个程序, 提示用户输入一个字符串, 显示它的长度和第一个字符。

*4.21 (检查 SSN) 编写一个程序, 提示用户输入一个社保号码, 它的格式是 DDD-DD-DDDD, 其中 D 是一个数字。你的程序应该判断输入是否合法。下面是一个运行示例:

```
Enter a SSN: 232-23-5435 ↵ Enter
232-23-5435 is a valid social security number
```

```
Enter a SSN: 23-23-5435 ↵ Enter
23-23-5435 is an invalid social security number
```

4.22 (检测子串) 编写一个程序, 提示用户输入两个字符串, 检测第二个字符串是否是第一个字符串的子串。

```
Enter string s1: ABCD ↵ Enter
Enter string s2: BC ↵ Enter
BC is a substring of ABCD
```

```
Enter string s1: ABCD ↵ Enter
Enter string s2: BDC ↵ Enter
BDC is not a substring of ABCD
```

*4.23 (财务应用: 薪金) 编写一个程序, 读取下面的信息, 然后输出一个薪金声明:

雇员姓名 (如: Smith)

每周的工作小时数 (如, 10 小时)

每小时的薪金 (如, 9.75 美元)

联邦所得税税率 (如, 20%)

州所得税税率 (如, 9%)

下面是一个运行示例:

```
Enter employee's name: Smith ↵ Enter
Enter number of hours worked in a week: 10 ↵ Enter
Enter hourly pay rate: 9.75 ↵ Enter
Enter federal tax withholding rate: 0.20 ↵ Enter
Enter state tax withholding rate: 0.09 ↵ Enter
```

```
Employee Name: Smith
Hours Worked: 10.0
Pay Rate: $9.75
Gross Pay: $97.5
Deductions:
  Federal Withholding (20.0%): $19.5
  State Withholding (9.0%): $8.77
  Total Deduction: $28.27
Net Pay: $69.22
```

- *4.24 (对三个城市排序) 编写一个程序，提示用户输入三个城市名称，然后以升序进行显示。下面是一个运行示例：

```
Enter the first city: Chicago --Enter
Enter the second city: Los Angeles --Enter
Enter the third city: Atlanta --Enter
The three cities in alphabetical order are Atlanta Chicago Los Angeles
```

- *4.25 (生成车牌号码) 假设一个车牌号码由三个大写字母和后面的四个数字组成。编写一个程序，生成一个车牌号码。
- *4.26 (财务应用：货币单位) 改写程序清单 2-10，解决将 float 型值转换为 int 型值时可能会造成精度损失的问题。读取的输入值是一个字符串，比如 "11.56"。你的程序应该应用 indexOf 和 substring 方法抽取小数点前的美元数量，以及小数点后面的美分数量。

循 环

教学目标

- 使用 while 循环编写重复执行语句的程序 (5.2 节)。
- 遵循循环设计策略来开发循环 (5.2.1 ~ 5.2.3 节)。
- 使用标记值控制循环 (5.2.4 节)。
- 使用输入重定向, 从文件而不是从键盘输入以获取大量输入 (5.2.5 节)。
- 使用 do-while 语句编写循环 (5.3 节)。
- 使用 for 语句编写循环 (5.4 节)。
- 了解三种类型循环语句的相似处和不同点 (5.5 节)。
- 编写嵌套循环 (5.6 节)。
- 学习最小化数值误差的技术 (5.7 节)。
- 从各种例子 (GCD、FutureTution、Dec2Hex) 中学习循环 (5.8 节)。
- 使用 break 和 continue 来实现程序的控制 (5.9 节)。
- 在一个判断回文的示例学习中, 使用循环来处理字符串中的字符 (5.10 节)。
- 编写一个程序, 显示素数 (5.11 节)。

5.1 引言

 **要点提示:** 循环可以用于让一个程序重复地执行语句。

假如你需要打印一个字符串 (例如: "Welcome to Java!") 100 次, 就需要把下面的输出语句重复写 100 遍, 这是相当繁琐的:

```
100 次 { System.out.println("Welcome to Java!");  
        System.out.println("Welcome to Java!");  
        ...  
        System.out.println("Welcome to Java!");
```

那该如何解决这个问题呢?

Java 提供了一种称为循环 (loop) 的功能强大的结构, 用来控制一个操作或操作序列重复执行的次数。使用循环语句时, 只要简单地告诉计算机输出字符串 100 次, 而无须重复打印输出语句 100 次, 如下所示:

```
int count = 0;  
while (count < 100) {  
    System.out.println("Welcome to Java!");  
    count++;  
}
```

变量 count 的初值为 0。循环检测 (count<100) 是否为 true。若为 true, 则执行循环体以输出消息 "Welcome to Java!", 然后给 count 加 1。重复执行这个循环, 直到 (count<100) 变为 false 为止。当 (count<100) 变为 false (例如: count 达到 100), 此时循环终止, 然后执行循环语句之后的下一条语句。

循环是用来控制语句块重复执行的一种结构。循环的概念是程序设计的基础。Java 提供了三种类型的循环语句：`while` 循环、`do-while` 循环和 `for` 循环。

5.2 while 循环

要点提示：`while` 循环在条件为真的情况下，重复地执行语句。

`while` 循环的语法如下：

```
while( 循环继续条件 ){
    // 循环体
    语句(组);
}
```

图 5-1a 给出了 `while` 循环的流程图。循环中包含的重复执行的语句部分称为循环体 (loop body)。循环体的每一次执行都被认为是一次循环的迭代 (或重复)。每个循环都含有循环继续条件，循环继续条件是一个布尔表达式，控制循环体的执行。在循环体执行前总是先计算循环条件以决定是否执行它。若条件为 `true`，则执行循环体；若条件为 `false`，则终止整个循环，并且程序控制转移到 `while` 循环后的下一条语句。

前面小节介绍的循环打印 “Welcome to Java!” 100 次就是 `while` 循环的一个例子。它的流程图如图 5-1b 所示。循环继续条件是 `(count < 100)`，而且循环体包含如下两条语句：

```
int count = 0;
while (count < 100) {
    System.out.println("Welcome to Java!");
    count++;
}
```

循环继续条件

循环体

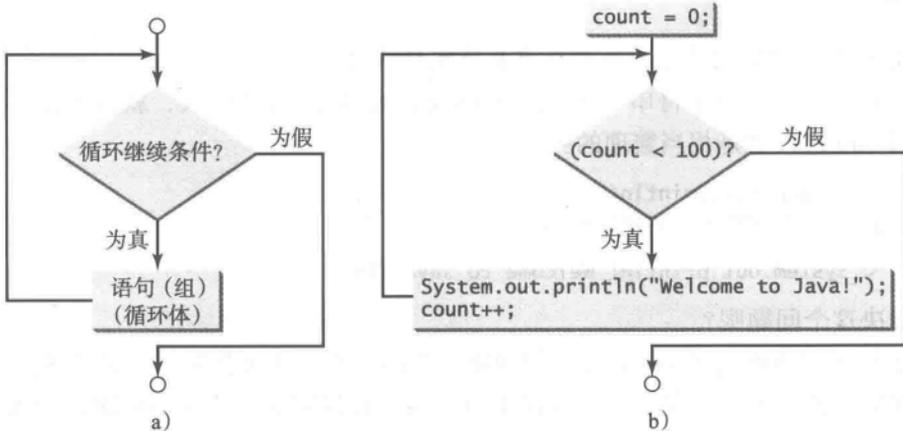


图 5-1 当循环继续条件为 `true` 时，`while` 循环重复执行循环体中的语句

在本例中，确切地知道循环体需要执行的次数。所以，使用一个控制变量 `count` 来对执行次数计数。这种类型的循环称为计数器控制的循环 (counter-controlled loop)。

注意：循环继续条件应该总是放在圆括号内。只有当循环体只包含一条语句或不包含语句时，循环体的花括号才可以省略。

下面是另外一个例子，有助于理解循环是如何工作的。

```
int sum = 0, i = 1;
while (i < 10) {
    sum = sum + i;
```

```

    i++;
}
System.out.println("sum is " + sum); // sum is 45

```

如果 $i < 10$ 为 true, 那么程序将 i 加入 sum 。变量 i 被初始化为 1, 然后自增为 2、3、直到 10。当 i 为 10 时, $i < 10$ 为 false, 退出循环。所以, 和就是 $1+2+3+\dots+9=45$ 。

如果循环被错误地写为如下所示, 那会出现什么情况?

```

int sum = 0, i = 1;
while (i < 10) {
    sum = sum + i;
}

```

该循环就会成为无限循环, 因为 i 总是 1 而 $i < 10$ 永远都为 true。

🔧 **注意:** 要保证循环继续条件最终可以变为 false, 以便程序能够结束。一个常见的程序设计错误是无限循环 (也就是说, 循环会永远执行下去)。如果程序运行了不寻常的长时间而不结束, 可能其中有无限循环。如果你是从命令窗口运行程序的, 按 CTRL+C 键来结束。

🔧 **警告:** 程序员经常会犯的的错误就是使循环多执行一次或少执行一次。这种情况通常称为差一错误 (off-by-one error)。例如: 下面的循环会将 Welcome to Java 显示 101 次, 而不是 100 次。这个错误出在条件部分, 所以条件应该是 $count < 100$ 而不是 $count \leq 100$ 。

```

int count = 0;
while (count <= 100) {
    System.out.println("Welcome to Java!");
    count++;
}

```

回顾程序清单 3-1 给出了一个程序, 提示用户为两个个位数相加的问题给出答案。使用循环, 现在你可以重写程序, 让用户重复输入新的答案, 直到答案正确为止, 如下面程序清单 5-1 所示。

程序清单 5-1 RepeatAdditionQuiz.java

```

1 import java.util.Scanner;
2
3 public class RepeatAdditionQuiz {
4     public static void main(String[] args) {
5         int number1 = (int)(Math.random() * 10);
6         int number2 = (int)(Math.random() * 10);
7
8         // Create a Scanner
9         Scanner input = new Scanner(System.in);
10
11        System.out.print(
12            "What is " + number1 + " + " + number2 + "? ";
13        int answer = input.nextInt();
14
15        while (number1 + number2 != answer) {
16            System.out.print("Wrong answer. Try again. What is "
17                + number1 + " + " + number2 + "? ");
18            answer = input.nextInt();
19        }
20
21        System.out.println("You got it!");
22    }
23 }

```

```

What is 5 + 9? 12 
Wrong answer. Try again. What is 5 + 9? 34 
Wrong answer. Try again. What is 5 + 9? 14 
You got it!

```

第 15 ~ 19 行的循环在 `number1+number2!=answer` 的情况下，重复提示用户输入一个 `answer` 值。`number1+number2!=answer` 为 `flase`，循环退出。

5.2.1 示例学习：猜数字

要解决的问题是猜测计算机“脑子”里想的是什么数。编写一个程序，随机产生一个 0 到 100 之间且包含 0 和 100 的整数。程序提示用户连续输入一个数字，直到它和计算机随机产生的数字相匹配为止。对用户每次输入的数字，程序都要告诉用户该输入值是偏大了，还是偏小了，这样用户可以明智地进行下一轮的猜测。下面是一个运行示例：

```

Guess a magic number between 0 and 100
Enter your guess: 50 ↵
Your guess is too high
Enter your guess: 25 ↵
Your guess is too low
Enter your guess: 42 ↵
Your guess is too high
Enter your guess: 39 ↵
Yes, the number is 39

```

这个魔法数在 0 到 100 之间。为了减小猜测的次数，首先输入 50。如果猜测值过高，那么这个魔法数就在 0 到 49 之间。如果猜测值过低，那么这个魔法数就在 51 到 100 之间。因此，经过一次猜测之后，下一次猜测时可以少考虑一半的数字。

该如何编写这个程序呢？要立即开始编码吗？不！编码前的思考是非常重要的。思考一下，在没有编写程序时你会如何解决这个问题。首先需要产生一个 0 到 100 之间且包含 0 和 100 的随机数，然后提示用户输入一个猜测数，最后将这个猜测数和随机数进行比较。

一次增加一个步骤地渐进编码（code incrementally）是一个很好的习惯。对涉及编写循环的程序而言，如果不知道如何立即编写循环，可以编写循环只执行一次的代码，然后规划如何在循环中重复执行这些代码。为了编写这个程序，可以打一个初稿，如程序清单 5-2 所示。

程序清单 5-2 GuessNumberOneTime.java

```

1  import java.util.Scanner;
2
3  public class GuessNumberOneTime {
4      public static void main(String[] args) {
5          // Generate a random number to be guessed
6          int number = (int)(Math.random() * 101);
7
8          Scanner input = new Scanner(System.in);
9          System.out.println("Guess a magic number between 0 and 100");
10
11         // Prompt the user to guess the number
12         System.out.print("\nEnter your guess: ");
13         int guess = input.nextInt();
14
15         if (guess == number)
16             System.out.println("Yes, the number is " + number);
17         else if (guess > number)
18             System.out.println("Your guess is too high");
19         else
20             System.out.println("Your guess is too low");
21     }
22 }

```

运行这个程序时，它只提示用户输入一次猜测值。为使用户重复输入猜测值，可将第 11 ~ 20 行的代码放入循环里，如下所示：

```
while (true) {
    // Prompt the user to guess the number
    System.out.print("\nEnter your guess: ");
    guess = input.nextInt();

    if (guess == number)
        System.out.println("Yes, the number is " + number);
    else if (guess > number)
        System.out.println("Your guess is too high");
    else
        System.out.println("Your guess is too low");
} // End of loop
```

这个循环重复提示用户输入猜测值。但是，这个循环是不正确的，因为它永远都不会结束。当 guess 和 number 匹配时，该循环就应该结束。所以，可对这个循环做如下修改：

```
while (guess != number) {
    // Prompt the user to guess the number
    System.out.print("\nEnter your guess: ");
    guess = input.nextInt();

    if (guess == number)
        System.out.println("Yes, the number is " + number);
    else if (guess > number)
        System.out.println("Your guess is too high");
    else
        System.out.println("Your guess is too low");
} // End of loop
```

程序清单 5-3 给出完整的代码。

程序清单 5-3 GuessNumber.java

```
1 import java.util.Scanner;
2
3 public class GuessNumber {
4     public static void main(String[] args) {
5         // Generate a random number to be guessed
6         int number = (int)(Math.random() * 101);
7
8         Scanner input = new Scanner(System.in);
9         System.out.println("Guess a magic number between 0 and 100");
10
11         int guess = -1;
12         while (guess != number) {
13             // Prompt the user to guess the number
14             System.out.print("\nEnter your guess: ");
15             guess = input.nextInt();
16
17             if (guess == number)
18                 System.out.println("Yes, the number is " + number);
19             else if (guess > number)
20                 System.out.println("Your guess is too high");
21             else
22                 System.out.println("Your guess is too low");
23         } // End of loop
24     }
25 }
```

	line#	number	guess	output
	6	39		
	11		-1	
iteration 1	15		50	
	20			Your guess is too high
iteration 2	15		25	
	22			Your guess is too low
iteration 3	15		42	
	20			Your guess is too high
iteration 4	15		39	
	18			Yes, the number is 39

程序在第6行创建一个魔法数，然后提示用户在一个循环中连续输入猜测值（第12~23行）。对每一次猜测，程序检查该猜测数是否正确，是偏高还是偏低了（第17-22行）。当某次猜测正确时，程序就退出这个循环（第12行）。注意：`guess`被初始化为-1。将它初始化为0到100之间的值会出错，因为它很可能就是要猜的数。

5.2.2 循环设计策略

编写一个正确的循环对编程新手来说，并不是件容易的事。编写循环时应该考虑如下三个步骤：

第一步：确定需要重复的语句。

第二步：将这些语句放在一个循环中，如下所示：

```
while(true){
    语句组；
}
```

第三步：为循环继续条件编码，并为控制循环添加适合的语句。

```
while(循环继续条件){
    语句组；
    用于控制循环的附件语句；
}
```

5.2.3 示例学习：多个减法测试题

程序清单 3-3 中的数学减法学习工具程序，每次运行只能产生一道题目。可以使用一个循环重复产生题目。那么如何编写能产生5道题目的代码呢？遵循循环设计策略。首先，确定需要重复的语句。这些语句包括：获取两个随机数，提示用户对两数做减法然后给试题打分。然后，将这些语句放在一个循环里。最后，增加一个循环控制变量和循环继续条件，然后执行循环五次。

程序清单 5-4 给出的程序可以产生5道问题，在学生回答完所有5个问题后，报告回答正确的题数。这个程序还显示该测试所花的时间，并列出了所有的题目。

程序清单 5-4 SubtractionQuizLoop.java

```
1 import java.util.Scanner;
2
3 public class SubtractionQuizLoop {
4     public static void main(String[] args) {
5         final int NUMBER_OF_QUESTIONS = 5; // Number of questions
6         int correctCount = 0; // Count the number of correct answers
7         int count = 0; // Count the number of questions
8         long startTime = System.currentTimeMillis();
9         String output = ""; // output string is initially empty
10        Scanner input = new Scanner(System.in);
11
12        while (count < NUMBER_OF_QUESTIONS) {
13            // 1. Generate two random single-digit integers
14            int number1 = (int)(Math.random() * 10);
15            int number2 = (int)(Math.random() * 10);
16
17            // 2. If number1 < number2, swap number1 with number2
18            if (number1 < number2) {
19                int temp = number1;
20                number1 = number2;
21                number2 = temp;
22            }
23
24            // 3. Prompt the student to answer "What is number1 - number2?"
25            System.out.print(
26                "What is " + number1 + " - " + number2 + "? ");
27            int answer = input.nextInt();
28
29            // 4. Grade the answer and display the result
30            if (number1 - number2 == answer) {
31                System.out.println("You are correct!");
32                correctCount++; // Increase the correct answer count
33            }
34            else
35                System.out.println("Your answer is wrong.\n" + number1
36                    + " - " + number2 + " should be " + (number1 - number2));
37
38            // Increase the question count
39            count++;
40
41            output += "\n" + number1 + "-" + number2 + "=" + answer +
42                (number1 - number2 == answer) ? " correct" : " wrong";
43        }
44
45        long endTime = System.currentTimeMillis();
46        long testTime = endTime - startTime;
47
48        System.out.println("Correct count is " + correctCount +
49            "\nTest time is " + testTime / 1000 + " seconds\n" + output);
50    }
51 }
```

```
What is 9 - 2? 7 
You are correct!

What is 3 - 0? 3 
You are correct!

What is 3 - 2? 1 
You are correct!
```

```

What is 7 - 4? 4 Enter
Your answer is wrong.
7 - 4 should be 3

What is 7 - 5? 4 Enter
Your answer is wrong.
7 - 5 should be 2

Correct count is 3
Test time is 1021 seconds

9-2=7 correct
3-0=3 correct
3-2=1 correct
7-4=4 wrong
7-5=4 wrong

```

程序使用控制变量 `count` 来控制循环的执行。`count` 被初始化为 0 (第 7 行), 并在每次迭代中加 1 (第 39 行)。每次迭代都显示并处理一个减法题目。程序中第 8 行代码获得测试开始的时间, 第 45 行获得测试结束的时间, 然后在第 46 行计算出测试所用时间。测试时间以毫秒为单位并且在第 49 行被转换为秒。

5.2.4 使用标记值控制循环

另一种控制循环的常用技术是在读取和处理一个集合的值时指派一个特殊值。这个特殊的输入值也称为标记值 (sentinel value), 用以表明循环的结束。如果一个循环使用标记值来控制它的执行, 它就称为标记位控制的循环 (sentinel-controlled loop)。

程序清单 5-5 编写了一个程序, 用来读取和计算个数不确定的整数之和, 并以输入 0 表示输入结束。需要为每次输入值声明新变量吗? 答案是: 不需要。只需要使用名为 `data` 的变量 (第 12 行) 存储输入值, 并使用名为 `sum` 的变量 (第 15 行) 存储和。每当读取一个数, 就将其赋值给 `data`, 如果它不为 0, 则将该 `data` 加到 `sum` 中 (第 17 行)。

程序清单 5-5 SentinelValue.java

```

1 import java.util.Scanner;
2
3 public class SentinelValue {
4     /** Main method */
5     public static void main(String[] args) {
6         // Create a Scanner
7         Scanner input = new Scanner(System.in);
8
9         // Read an initial data
10        System.out.print(
11            "Enter an integer (the input ends if it is 0): ");
12        int data = input.nextInt();
13
14        // Keep reading data until the input is 0
15        int sum = 0;
16        while (data != 0) {
17            sum += data;
18
19            // Read the next data
20            System.out.print(
21                "Enter an integer (the input ends if it is 0): ");
22            data = input.nextInt();
23        }

```

```

24
25     System.out.println("The sum is " + sum);
26 }
27 }

```

```

Enter an integer (the input ends if it is 0): 2 
Enter an integer (the input ends if it is 0): 3 
Enter an integer (the input ends if it is 0): 4 
Enter an integer (the input ends if it is 0): 0 
The sum is 9

```

	line#	Data	sum	output
	12	2		
	15		0	
iteration 1	17		2	
	22	3		
iteration 2	17		5	
	22	4		
iteration 3	17		9	
	22	0		
	25			The sum is 9

如果 data 不为 0，则将它加到总和 sum 中（第 17 行），然后读取下一条输入数据（第 20 ~ 22 行）。若 data 为 0，则不再执行循环体并且终止掉 while 循环。输入值 0 是该循环的标记值。注意：若第一个读取到的输入值就是 0，则永远不会执行循环体，最终的和 sum 为 0。

 **警告：**在循环控制中，不要使用浮点值来比较值是否相等。因为浮点值都是某些值的近似值，使用它们可能导致不精确的循环次数和不准确的结果。

考虑下面计算 $1+0.9+0.8+\dots+0.1$ 的代码：

```

double item = 1; double sum = 0;
while (item != 0) { // No guarantee item will be 0
    sum += item;
    item -= 0.1;
}
System.out.println(sum);

```

变量 item 从 1 开始，每执行一次循环体就减去 0.1。当 item 变为 0 时循环应该终止。但是，因为浮点数在算术上是近似的，所以不能确保 item 会变成真正的 0。从表面上看，这个循环似乎没问题，但实际上它是一个无限循环。

5.2.5 输入和输出重定向

在前面的例子中，如果要输入大量的数值，那么从键盘上输入是非常繁琐的事。可以将这些数据用空格隔开，保存在一个名为 input.txt 的文本文件中，然后使用下面的命令运行这个程序：

```
java SentinelValue < input.txt
```

这个命令称为输入重定向 (input redirection)。程序从文件 input.txt 中读取输入，而不是让用户在运行时从键盘输入数据。假设文件内容是：

```
2 3 4 5 6 7 8 9 12 23 32
23 45 67 89 92 12 34 35 3 1 2 4 0
```

程序将得到 sum 值为 518。

类似地，还有输出重定向 (output redirection)，输出重定向将输出发送给文件，而不是将它们显示在控制台上。输出重定向的命令为：

```
java ClassName > output.txt
```

可以在同一命令中同时使用输入重定向和输出重定向。例如，下面的命令从文件 input.txt 中获取输入，并将输出发送给文件 output.txt：

```
java SentinelValue <input.txt> output.txt
```

请运行这个程序，查看一下 output.txt 中的内容是什么。

复习题

5.1 分析下面的代码。在 Point A 处、Point B 处和 Point C 处，count < 0 总是 true 还是总是 false，或者有时是 true 有时是 false？

```
int count = 0;
while (count < 100) {
    // Point A
    System.out.println("Welcome to Java!");
    count++;
    // Point B
}
// Point C
```

5.2 在程序清单 5-3 中的第 11 行中，如果 guess 初始化为 0，会出现什么错误？

5.3 下面的循环体会重复多少次？这个循环的输出是什么？

```
int i = 1;
while (i < 10)
    if (i % 2 == 0)
        System.out.println(i);
```

a)

```
int i = 1;
while (i < 10)
    if (i % 2 == 0)
        System.out.println(i++);
```

b)

```
int i = 1;
while (i < 10)
    if ((i++) % 2 == 0)
        System.out.println(i);
```

c)

5.4 假设输入是 2 3 4 5 0，那么下面代码的输出结果是什么？

```
import java.util.Scanner;

public class Test {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        int number, max;
        number = input.nextInt();
        max = number;

        while (number != 0) {
            number = input.nextInt();
            if (number > max)
                max = number;
        }
    }
}
```

```

        System.out.println("max is " + max);
        System.out.println("number " + number);
    }
}

```

5.5 下面代码的输出结果是什么？解释原因。

```

int x = 80000000;

while (x > 0)
    x++;

System.out.println("x is " + x);

```

5.3 do-while 循环

要点提示：do-while 循环和 while 循环基本一样，不同的是它先执行循环体一次，然后判断循环继续条件。

do-while 循环是 while 循环的变体。它的语法如下：

```

do {
    // 循环体；
    语句（组）；
} while（循环继续条件）；

```

它的执行流程图如图 5-2 所示。

首先执行循环体，然后计算循环继续条件。如果计算结果为 true，则重复执行循环体；如果为 false，则终止 do-while 循环。while 循环与 do-while 循环的差别在于：计算循环继续条件和执行循环体的先后顺序不同。有时候，选择其中一种会比另一种更方便。例如，可以采用 do-while 循环改写程序清单 5-5 中的 while 循环，如程序清单 5-6 所示。

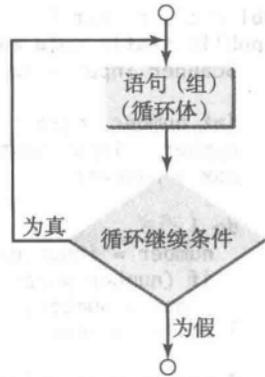


图 5-2 do-while 循环首先执行循环体，然后检查循环继续条件，以确定继续执行循环还是终止循环

程序清单 5-6 TestDoWhile.java

```

1 import java.util.Scanner;
2
3 public class TestDoWhile {
4     /** Main method */
5     public static void main(String[] args) {
6         int data;
7         int sum = 0;
8
9         // Create a Scanner
10        Scanner input = new Scanner(System.in);
11
12        // Keep reading data until the input is 0
13        do {
14            // Read the next data
15            System.out.print(
16                "Enter an integer (the input ends if it is 0): ");
17            data = input.nextInt();
18
19            sum += data;
20        } while (data != 0);

```

```

21
22     System.out.println("The sum is " + sum);
23 }
24 }

```

```

Enter an integer (the input ends if it is 0): 3
Enter an integer (the input ends if it is 0): 5
Enter an integer (the input ends if it is 0): 6
Enter an integer (the input ends if it is 0): 0
The sum is 14

```

 **提示：**如果循环中的语句至少需要执行一次，建议使用 do-while 循环。前面程序 TestDoWhile 中 do-while 循环的情形就是如此。如果使用 while 循环，那么这些语句必须在循环前和循环内都出现。

复习题

5.6 假设输入是 2 3 4 5 0，那么下面代码的输出结果是什么？

```

import java.util.Scanner;

public class Test {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        int number, max;
        number = input.nextInt();
        max = number;

        do {
            number = input.nextInt();
            if (number > max)
                max = number;
        } while (number != 0);

        System.out.println("max is " + max);
        System.out.println("number " + number);
    }
}

```

5.7 while 循环和 do-while 循环之间的区别是什么？将下面的 while 循环转换成 do-while 循环。

```

Scanner input = new Scanner(System.in);
int sum = 0;
System.out.println("Enter an integer " +
    "(the input ends if it is 0)");
int number = input.nextInt();
while (number != 0) {
    sum += number;
    System.out.println("Enter an integer " +
        "(the input ends if it is 0)");
    number = input.nextInt();
}

```

5.4 for 循环

 **要点提示：**for 循环具有编写循环的简明语法。

经常会用到下面的通用形式编写循环：

```

i = initialValue; // Initialize loop control variable
while (i < endValue)
    // Loop body
    ...
    i++; // Adjust loop control variable
}

```

可以使用 for 循环简化前面的循环：

```

for (i = initialValue; i < endValue; i++)
    // Loop body
    ...
}

```

通常，for 循环的语法如下所示：

```

for (初始操作; 循环继续条件; 每次迭代后的操作) {
// 循环体;
语句 (组);
}

```

for 循环的流程图如图 5-3a 所示。

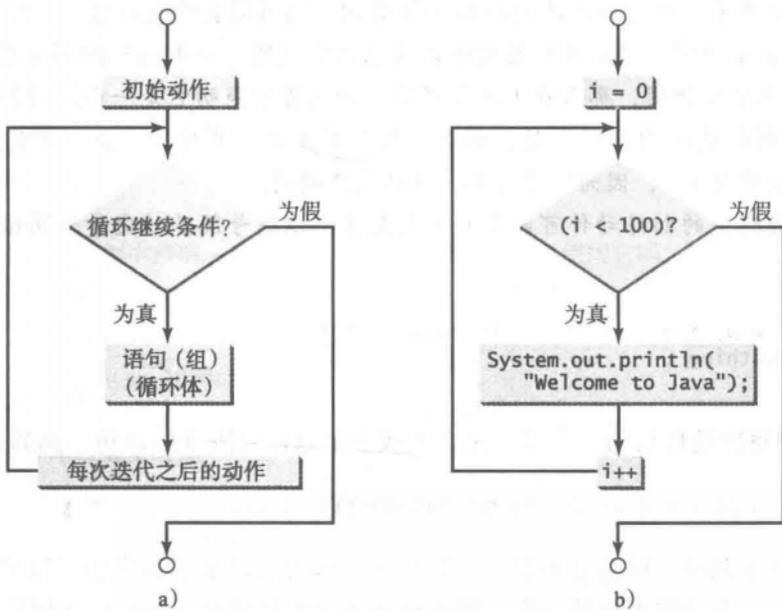


图 5-3 for 循环只执行初始动作一次，当循环继续条件为真时，重复执行循环体中的语句，然后完成每次迭代后的操作

for 循环语句从关键字 for 开始，然后是用双括号括住的循环控制结构体。这个结构体包括初始动作、循环继续条件和每次迭代后的动作。控制结构体后紧跟着花括号括起来的循环体。初始动作、循环继续条件和每次迭代后的动作都要用分号分隔。

一般情况下，for 循环使用一个变量来控制循环体的执行次数，以及什么时候循环终止。这个变量称为控制变量 (control variable)。初始化动作是指初始化控制变量，每次迭代后的动作通常会对控制变量做自增或自减，而循环继续条件检验控制变量是否达到终止值。例如，下面的 for 循环打印 Welcome to Java! 100 次：

```
int i;
for (i = 0; i < 100; i++) {
    System.out.println("Welcome to Java!");
}
```

语句的流程图如图 5-3b 所示。for 循环将控制变量 i 初始化为 0，当 i 小于 100 时，重复执行 `println` 语句并计算 $i++$ 。

初始化动作 $i=0$ 初始化控制变量 i 。循环继续条件 $i<100$ 是一个布尔表达式。这个表达式在初始化之后和每次迭代开始之前都要计算一次。如果这个条件为 `true`，则执行该循环体。如果它为 `false`，则循环终止，并且将程序控制转移到循环后的下一行。

每次迭代后的动作 $i++$ 是一个调整控制变量的语句。每次迭代结束后执行这条语句。它自增控制变量的值。最终，控制变量的值应该使循环继续条件变为 `false`，否则循环将成为无限循环。

循环控制变量可以在 for 循环中声明和初始化。下面就是一个例子：

```
for (int i = 0; i < 100; i++) {
    System.out.println("Welcome to Java!");
}
```

如果像这个例子一样，循环体内只有一条语句，则可以省略花括号。

提示：控制变量必须在循环控制结构体内或循环前说明。如果循环控制变量只在循环内使用而不在其他地方使用，那么在 for 循环的初始动作中声明它是一个很好的编程习惯。如果在循环控制结构体内声明变量，那么在循环外不能引用它。例如，不能在前面代码的 for 循环外引用变量 i ，因为它是在 for 循环内声明的。

注意：for 循环中的初始动作可以是 0 个或是多个以逗号隔开的变量声明语句或赋值表达式。例如：

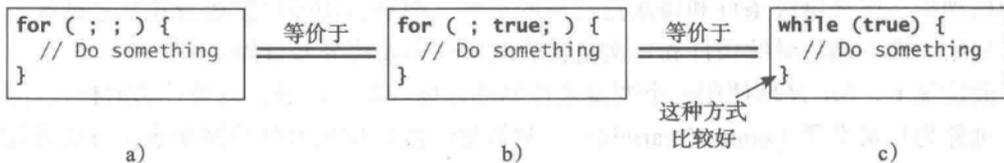
```
for (int i = 0, j = 0; i + j < 10; i++, j++) {
    // Do something
}
```

for 循环中每次迭代后的动作可以是 0 个或多个以逗号隔开的语句。例如：

```
for (int i = 1; i < 100; System.out.println(i), i++);
```

这个例子是正确的，但是它不是一个好例子，因为它增加了程序的阅读难度。通常，将声明和初始化一个变量作为初始动作，将增加或减少控制变量作为每次迭代后的操作。

注意：如果省略 for 循环中的循环继续条件，则隐含地认为循环继续条件为 `true`。因此，下面图 a 中给出的语句和图 b 中给出的语句一样，它们都是无限循环。但是，为了避免混淆，最好还是使用图 c 中的等价循环：



复习题

5.8 完成下列两个循环之后，`sum` 是否具有相同的值？

```
for (int i = 0; i < 10; ++i) {
    sum += i;
}
```

a)

```
for (int i = 0; i < 10; i++) {
    sum += i;
}
```

b)

5.9 for 循环控制的三个部分是什么？编写一个 for 循环，输出从 1 到 100 的整数。

5.10 假设输入是 2 3 4 5 0，那么下面代码的输出结果是什么？

```
import java.util.Scanner;

public class Test {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        int number, sum = 0, count;

        for (count = 0; count < 5; count++) {
            number = input.nextInt();
            sum += number;
        }

        System.out.println("sum is " + sum);
        System.out.println("count is " + count);
    }
}
```

5.11 下面的语句做什么？

```
for ( ; ; ) {
    // Do something
}
```

5.12 如果在 for 循环控制中声明一个变量，在退出循环后还可以使用它吗？

5.13 将下面的 for 循环语句转换为 while 循环和 do-while 循环：

```
long sum = 0;
for (int i = 0; i <= 1000; i++)
    sum = sum + i;
```

5.14 计算下面循环体的重复次数。

```
int count = 0;
while (count < n) {
    count++;
}
```

a)

```
for (int count = 0;
     count <= n; count++) {
}
```

b)

```
int count = 5;
while (count < n) {
    count++;
}
```

c)

```
int count = 5;
while (count < n) {
    count = count + 3;
}
```

d)

5.5 采用哪种循环

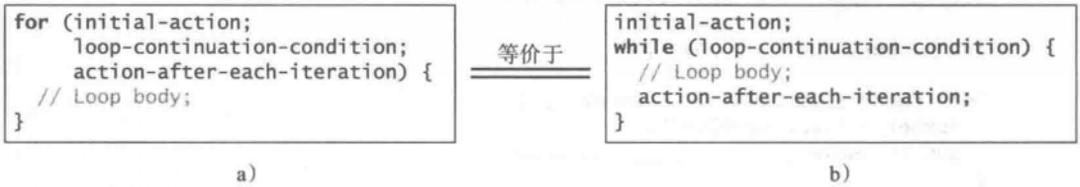
🔑 要点提示：可以根据哪个更加方便，来使用 for 循环、while 循环，或者 do-while 循环。

while 循环和 for 循环都称为前测循环 (pretest loop)，因为继续条件是在循环体执行之前检测的，do-while 循环称为后测循环 (posttest loop)，因为循环条件是在循环体执行之后

检测的。三种形式的循环语句：`while`、`do-while` 和 `for`，在表达上是等价的。也就是说，可以使用这三种形式之一来编写一个循环。例如，下面图 a 中 `while` 循环总能转化为图 b 中的 `for` 循环：

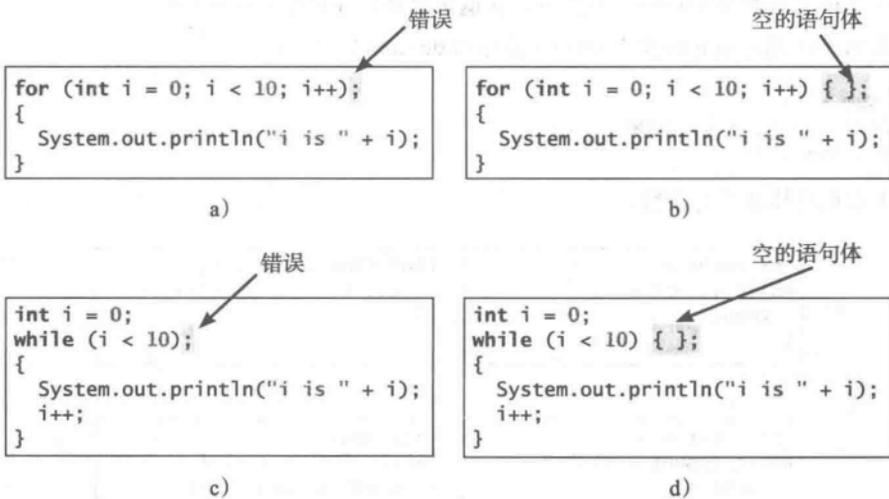


除了某些特殊情况外（参见复习题 5.25 中的情况），下面图 a 中的 `for` 循环通常都能转化为图 b 中的 `while` 循环：

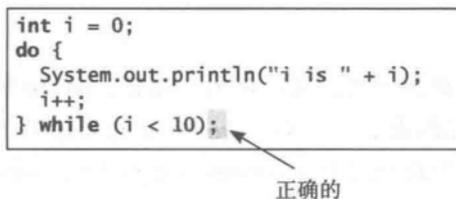


建议使用自己觉得最自然、最舒服的一种循环语句。通常，如果已经提前知道重复次数，那就采用 `for` 循环，例如，需要打印一条信息 100 次时，如果无法确定重复次数，就采用 `while` 循环，就像读入一些数值直到读入 0 为止的这种情况。如果在检验继续条件前需要执行循环体，就用 `do-while` 循环替代 `while` 循环。

警告： 在 `for` 子句的末尾和循环体之间多写分号是一个常见的错误，如下面的图 a 中所示。图 a 中分号过早地表明循环的结束。循环体实际上都是为空的，如图 b 所示。图 a 和图 b 是等价的，都是不正确的。类似地，图 c 中的循环也是错的，图 c 与图 d 等价，都是不正确的。



通常在使用次行块格式时容易发生这些错误。使用行尾块风格可以避免这种类型的错误。在 `do-while` 循环中，需要分号来结束这个循环。



复习题

5.15 可以将 for 循环转换为 while 循环吗？列出使用 for 循环的好处？

5.16 while 循环总是可以转换成 for 循环么？将下面的 while 循环转换为 for 循环。

```
int i = 1;
int sum = 0;
while (sum < 10000) {
    sum = sum + i;
    i++;
}
```

5.17 找到下面代码中的错误并且进行修正。

```
1 public class Test {
2     public void main(String[] args) {
3         for (int i = 0; i < 10; i++);
4             sum += i;
5
6         if (i < j);
7             System.out.println(i)
8         else
9             System.out.println(j);
10
11        while (j < 10);
12        {
13            j++;
14        }
15
16        do {
17            j++;
18        } while (j < 10)
19    }
20 }
```

5.18 下面的程序有什么错误？

```
1 public class ShowErrors {
2     public static void main(String[] args) {
3         int i = 0;
4         do {
5             System.out.println(i + 4);
6             i++;
7         }
8         while (i < 10)
9     }
10 }
```

a)

```
1 public class ShowErrors {
2     public static void main(String[] args) {
3         for (int i = 0; i < 10; i++);
4             System.out.println(i + 4);
5     }
6 }
```

b)

5.6 嵌套循环

要点提示：一个循环可以嵌套在另外一个循环中。

嵌套循环是由一个外层循环和一个或多个内层循环组成的。每当重复执行一次外层循环时将再次进入内部循环，然后重新开始。

程序清单 5-7 是使用嵌套 for 循环打印一个乘法表的程序。

程序清单 5-7 MultiplicationTable.java

```
1 public class MultiplicationTable {
2     /** Main method */
3     public static void main(String[] args) {
```

```

4 // Display the table heading
5 System.out.println("          Multiplication Table");
6
7 // Display the number title
8 System.out.print(" ");
9 for (int j = 1; j <= 9; j++)
10     System.out.print(" " + j);
11
12 System.out.println("\n-----");
13
14 // Display table body
15 for (int i = 1; i <= 9; i++) {
16     System.out.print(i + " | ");
17     for (int j = 1; j <= 9; j++) {
18         // Display the product and align properly
19         System.out.printf("%4d", i * j);
20     }
21     System.out.println();
22 }
23 }
24 }

```

Multiplication Table									
	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
3	3	6	9	12	15	18	21	24	27
4	4	8	12	16	20	24	28	32	36
5	5	10	15	20	25	30	35	40	45
6	6	12	18	24	30	36	42	48	54
7	7	14	21	28	35	42	49	56	63
8	8	16	24	32	40	48	56	64	72
9	9	18	27	36	45	54	63	72	81

程序在输出的第一行显示标题(第5行)。第一个 for 循环(第9~10行)在第二行显示从1到9的数字。在第三行显示横线(-)(第12行)。

下一个循环(第15~22行)是一个嵌套的 for 循环,其外层循环控制变量是*i*,而内层循环控制变量是*j*。在内层循环中,针对每个*i*,随着*j*取遍1,2,3,...,9,内层循环在每一行显示乘积*i*j*的值。

🔧 **注意:** 需要注意的是,嵌套循环将运行较长时间。考虑下面嵌套三层的循环:

```

for (int i = 0; i < 10000; i++)
    for (int j = 0; j < 10000; j++)
        for (int k = 0; k < 10000; k++)
            Perform an action

```

动作将被执行万亿次。如果需要1微秒来执行一次动作,整个循环花费的事件将大于277小时。注意,1微秒等于1秒的百万分之一。

🔪 复习题

5.19 println 语句执行了多少次?

```

for (int i = 0; i < 10; i++)
    for (int j = 0; j < i; j++)
        System.out.println(i * j)

```

5.20 给出下面程序的输出结果。(提示:绘制一个表格,在列中列出变量,对这些程序进行跟踪。)

```
public class Test {
    public static void main(String[] args) {
        for (int i = 1; i < 5; i++) {
            int j = 0;
            while (j < i) {
                System.out.print(j + " ");
                j++;
            }
        }
    }
}
```

a)

```
public class Test {
    public static void main(String[] args) {
        int i = 0;
        while (i < 5) {
            for (int j = i; j > 1; j--)
                System.out.print(j + " ");
            System.out.println("****");
            i++;
        }
    }
}
```

b)

```
public class Test {
    public static void main(String[] args) {
        int i = 5;
        while (i >= 1) {
            int num = 1;
            for (int j = 1; j <= i; j++) {
                System.out.print(num + "xxx");
                num *= 2;
            }

            System.out.println();
            i--;
        }
    }
}
```

c)

```
public class Test {
    public static void main(String[] args) {
        int i = 1;
        do {
            int num = 1;
            for (int j = 1; j <= i; j++) {
                System.out.print(num + "G");
                num += 2;
            }

            System.out.println();
            i++;
        } while (i <= 5);
    }
}
```

d)

5.7 最小化数值错误

🔑 要点提示：在循环继续条件中使用浮点数将导致数值错误。

涉及浮点数的数值误差是不可避免的，因为浮点数在计算机中本身就是近似表示的。本节将通过实例讨论如何最小化这种误差。

程序清单 5-8 给出的例子计算从 0.01 到 1.0 的数列之和，该数列中的数值以 0.01 递增，如下所示：0.01+0.02+0.03 + ...。

程序清单 5-8 TestSum.java

```
1 public class TestSum {
2     public static void main(String[] args) {
3         // Initialize sum
4         float sum = 0;
5
6         // Add 0.01, 0.02, ..., 0.99, 1 to sum
7         for (float i = 0.01f; i <= 1.0f; i = i + 0.01f)
8             sum += i;
9
10        // Display result
11        System.out.println("The sum is " + sum);
12    }
13 }
```

```
The sum is 50.499985
```

for 循环（第 7 ~ 8 行）重复地将控制变量 i 加到 sum 中。变量 i 从 0.01 开始，每次迭代增加 0.01。当 i 超过 1.0 时循环终止。

for 循环初始动作可以是任何语句，但是，它经常用来初始化控制变量。从本例中可以看到，控制变量可以是 float 型。事实上，它可以为任意数据类型。

sum 的精确结果应该是 50.50，但是答案是 50.499985。这个结果是不精确的，因为计算机使用固定位数表示浮点数，因此，它就不能精确表示某些浮点数。如果如下所示，将程序中的 float 型改成 double 型，应该可以看到精度有一些小小的改善，因为 double 型变量占 64 位而 float 型变量只占 32 位。

```
// Initialize sum
double sum = 0;

// Add 0.01, 0.02, ..., 0.99, 1 to sum
for (double i = 0.01; i <= 1.0; i = i + 0.01)
    sum += i;
```

可是你会吃惊地看到，实际的结果是 49.50000000000003。究竟哪里出错了呢？如果将循环中每次迭代的 i 打印出来，会发现最后一个 i 比 1 稍微大一点（不是精确的 1）。这就会造成最后一个 i 不能加到 sum 中。根本问题就是浮点数是用近似值表示的。为了解决这个问题，使用整数计数器以确保所有数字都被加到 sum 中。下面是一个新的循环：

```
double currentValue = 0.01;

for (int count = 0; count < 100; count++) {
    sum += currentValue;
    currentValue += 0.01;
}
```

这个循环结束后，sum 的值是 50.50000000000003。这个循环从小到大添加数字。如果如下所示从大到小（即以 1.0, 0.99, 0.98, ..., 0.02, 0.01 的顺序）添加，那会发生什么呢？

```
double currentValue = 1.0;

for (int count = 0; count < 100; count++) {
    sum += currentValue;
    currentValue -= 0.01;
}
```

在这个循环之后，sum 的值是 50.49999999999995。从大到小添加数字没有从小到大添加数字得到的值精确。这种现象是有限精度算术的产物。如果结果值要求的精度比变量可以存储的更高，那么添加一个非常小的数到一个非常大的数上可能没有什么影响。例如，100000000.0+0.000000001 的不精确的结果是 100000000.0。为了得到更精确的结果，仔细选择计算的顺序。在较大数之前先增加较小数是减小误差的一种方法。

5.8 示例学习

 **要点提示：**循环对于编程来说非常关键。编写循环的能力在学习 Java 编程中是非常重要的。如果你可以使用循环编写程序，你便知道了如何编程！因此，本节提供三个运用循环来解决问题的补充示例。

5.8.1 求最大公约数

两个整数 4 和 2 的最大公约数是 2。两个整数 16 和 24 的最大公约数是 8。如何编写程序来求最大公约数呢？是否立刻就开始编写代码？不对。在编写代码之前进行思考是非常重要的。

要的。思考让你可以在考虑如何编写代码前，生成解决问题的逻辑方案。

设输入的两个整数为 n_1 和 n_2 。已知 1 是一个公约数，但是它可能不是最大公约数。所以，可以检测 k ($k=2, 3, 4, \dots$) 是否为 n_1 和 n_2 的最大公约数，直到 k 大于 n_1 或 n_2 。公约数存储在名为 `gcd` 的变量中，`gcd` 的初值设为 1。当找到一个新的公约数时，它就成为新的 `gcd`。当检查完在 2 到 n_1 或 n_2 之间所有可能的公约数后，变量 `gcd` 的值就是最大公约数。一旦你有了一个逻辑方案，编写代码将该方案翻译成 Java 程序，如下所示：

```
int gcd = 1; // Initial gcd is 1
int k = 2; // Possible gcd

while (k <= n1 && k <= n2) {
    if (n1 % k == 0 && n2 % k == 0)
        gcd = k; // Update gcd
    k++; // Next possible gcd
}

// After the loop, gcd is the greatest common divisor for n1 and n2
```

程序清单 5-9 给出的程序，提示用户输入两个正整数，然后找到它们的最大公约数。

程序清单 5-9 GreatestCommonDivisor.java

```
1 import java.util.Scanner;
2
3 public class GreatestCommonDivisor {
4     /** Main method */
5     public static void main(String[] args) {
6         // Create a Scanner
7         Scanner input = new Scanner(System.in);
8
9         // Prompt the user to enter two integers
10        System.out.print("Enter first integer: ");
11        int n1 = input.nextInt();
12        System.out.print("Enter second integer: ");
13        int n2 = input.nextInt();
14
15        int gcd = 1; // Initial gcd is 1
16        int k = 2; // Possible gcd
17        while (k <= n1 && k <= n2) {
18            if (n1 % k == 0 && n2 % k == 0)
19                gcd = k; // Update gcd
20            k++;
21        }
22
23        System.out.println("The greatest common divisor for " + n1 +
24            " and " + n2 + " is " + gcd);
25    }
26 }
```

```
Enter first integer: 125 
Enter second integer: 2525 
The greatest common divisor for 125 and 2525 is 25
```

将逻辑方案翻译成 Java 代码的方式不是唯一的。例如，可以使用 `for` 循环改写代码，如下所示：

```
for (int k = 2; k <= n1 && k <= n2; k++) {
    if (n1 % k == 0 && n2 % k == 0)
        gcd = k;
}
```

一个问题常常有多种解决方案。最大公约数 (GCD) 问题就有许多解决方法。编程练习题 5-14 给出了另一种解决方案。一个更有效的方法是使用经典的欧几里得算法 (参见 22.6 节)。

考虑到 n_1 的除数不可能大于 $n_1/2$ ，所以，你可能会尝试用下面的循环改进该程序：

```
for (int k = 2; k <= n1 / 2 && k <= n2 / 2; k++) {
    if (n1 % k == 0 && n2 % k == 0)
        gcd = k;
}
```

上面的修改是错误的，你能找出原因吗？参见复习题 5.21 以得到答案。

5.8.2 预测未来学费

假设某个大学今年的学费是 10000 美元，而且以每年 7% 的速度增加。多少年之后学费会翻倍？

在编写解决这个问题的程序之前，首先考虑如何手工解决它。第二年的学费是第一年的学费乘以 1.07。未来一年的学费都是前一年的学费乘以 1.07。所以，每年的学费可以如下计算：

```
double tuition = 10000; int year = 0; // Year 0
tuition = tuition * 1.07; year++; // Year 1
tuition = tuition * 1.07; year++; // Year 2
tuition = tuition * 1.07; year++; // Year 3
...
```

不断地计算新一年的学费，直到学费至少是 20000 美元为止。到那时，就知道学费翻倍需要几年的时间。现在，可以将这个逻辑翻译成下面的循环：

```
double tuition = 10000; // Year 0
int year = 0;
while (tuition < 20000) {
    tuition = tuition * 1.07;
    year++;
}
```

完整的程序如程序清单 5-10 所示。

程序清单 5-10 FutureTuition.java

```
1 public class FutureTuition {
2     public static void main(String[] args) {
3         double tuition = 10000; // Year 0
4         int year = 0;
5         while (tuition < 20000) {
6             tuition = tuition * 1.07;
7             year++;
8         }
9
10        System.out.println("Tuition will be doubled in "
11            + year + " years");
12        System.out.printf("Tuition will be %.2f in %1d years",
13            tuition, year);
14    }
15 }
```

Tuition will be doubled in 11 years Tuition will be \$21048.52 in 11 years

使用 while 循环 (第 5 ~ 8 行) 重复计算新的一年的学费。当学费大于或等于 20000 美元时, 循环结束。

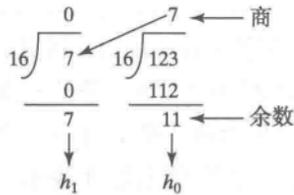
5.8.3 将十进制数转换为十六进制数

计算机系统的程序设计中会经常用到十六进制数 (参见附录 F 介绍的数字系统)。如何将一个十进制数转换为十六进制数呢? 将十进制数 d 转换为十六进制数, 就是找到满足以下条件的十六进制数 $h_n, h_{n-1}, h_{n-2}, \dots, h_2, h_1$ 和 h_0 :

$$d = h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + \dots + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0$$

这些数可以通过不断地用 d 除以 16 直到商为零而得到。依次得到的余数是 $h_n, h_{n-1}, h_{n-2}, \dots, h_2, h_1$ 和 h_0 。十六进制数字包含十进制数字 0、1、2、3、4、5、6、7、8、9 以及表示十进制数字 10 的 A, 表示十进制数字 11 的 B, 表示 12 的 C, 13 的 D, 14 的 E 和表示 15 的 F。

例如: 十进制数 123 被转换为十六进制数 7B。这个转换过程如下: 将 123 除以 16, 余数为 11 (十六进制的 B), 商为 7。继续将 7 除以 16, 余数为 7, 商为 0。因此 7B 就是 123 的十六进制数。



程序清单 5-11 给出程序, 提示用户输入一个十进制数, 然后将它转换为一个字符串形式的十六进制数。

程序清单 5-11 Dec2Hex.java

```

1 import java.util.Scanner;
2
3 public class Dec2Hex {
4     /** Main method */
5     public static void main(String[] args) {
6         // Create a Scanner
7         Scanner input = new Scanner(System.in);
8
9         // Prompt the user to enter a decimal integer
10        System.out.print("Enter a decimal number: ");
11        int decimal = input.nextInt();
12
13        // Convert decimal to hex
14        String hex = "";
15
16        while (decimal != 0) {
17            int hexValue = decimal % 16;
18
19            // Convert a decimal value to a hex digit
20            char hexDigit = (hexValue <= 9 && hexValue >= 0) ?
21                (char)(hexValue + '0') : (char)(hexValue - 10 + 'A');
22
23            hex = hexDigit + hex;
24            decimal = decimal / 16;
25        }
26
27        System.out.println("The hex number is " + hex);
28    }
29 }

```

```
Enter a decimal number: 1234 
The hex number is 4D2
```

	line#	decimal	hex	hexValue	hexDigit
	14	1234	""		
iteration 1	17			2	
	23		"2"		2
	24	77			
iteration 2	17			13	
	23		"D2"		D
	24	4			
iteration 3	17			4	
	23		"4D2"		4
	24	0			

程序提示用户输入一个十进制数字(第11行),将其转换为一个十六进制形式的字符串(第14~25行),然后显示结果(第27行)。为了将十进制转换为十六进制数,程序运用循环不断地将十进制数除以16,得到其余数(第17行)。余数转换为一个十六进制形式的字符串(第20~21行)。接下来,这个字符被追加在表示十六进制数的字符串的后面(第23行)。这个表示十六进制数的字符串初始时空(第14行)。将这个十进制数除以16,就从该数中去掉一个十六进制数字(第24行)。循环重复执行这些操作,直到商是0为止。

程序将0到15之间的十六进制数转换为一个十六进制字符。如果hexValue在0到9之间,那它就被转换为(char)(hexValue+'0')(第21行)。回顾一下,当一个字符和一个整数相加时,计算时使用的是字符的Unicode码。例如:如果hexValue为5,那么(char)(hexValue+'0')返回5。类似地,如果hexValue在10到15之间,那么它就被转换为(char)(hexValue-10+'A')(第21行)。例如,如果hexValue是11,那么(char)(hexValue-10+'A')返回B。

复习题

- 5.21 如果将程序清单5-9中第17行的n1和n2用n1/2和n2/2来替换,程序还会工作吗?
- 5.22 程序清单5-11中,如果你将第21行的代码(char)(hexValue+'0')改为hexValue+'0',为什么会出错?
- 5.23 程序清单5-11中,对于十进制数245而言,循环体将执行多少次?对于十进制数3245而言,循环体将执行多少次?

5.9 关键字 break 和 continue

要点提示: 关键字 break 和 continue 在循环中提供了额外的控制。

教学注意: 关键字 break 和 continue 都可以在循环语句中使用,为循环提供额外的控制。在某些情况下,使用 break 和 continue 可以简化程序设计。但是,过度使用或者不正确地使用它们会使得程序难以读懂也难以调试。(提醒教师:可以跳过本节,对本书的其他内容没有任何影响。)

你已经在 switch 语句中使用过关键字 break，你也可以在一个循环中使用 break 立即终止该循环。程序清单 5-12 给出的程序演示了在循环中使用 break 的效果。

程序清单 5-12 TestBreak.java

```
1 public class TestBreak {
2     public static void main(String[] args) {
3         int sum = 0;
4         int number = 0;
5
6         while (number < 20) {
7             number++;
8             sum += number;
9             if (sum >= 100)
10                break;
11        }
12
13        System.out.println("The number is " + number);
14        System.out.println("The sum is " + sum);
15    }
16 }
```

```
The number is 14
The sum is 105
```

程序清单 5-12 中的程序将从 1 到 20 的整数依次加到 sum 中，直到 sum 大于或等于 100。如果没有 if 语句（第 9 行），该程序计算从 1 到 20 之间整数的和。但是，有了 if 语句，那么当总和大于或等于 100 时，这个循环就会终止。没有 if 语句，程序输出结果将会是：

```
The number is 20
The sum is 210
```

也可以在循环中使用关键字 continue。当程序遇到 continue 时，它会结束当前的迭代。程序控制转向该循环体的末尾。换句话说，continue 只是跳出了一次迭代，而关键字 break 是跳出了整个循环。程序清单 5-13 给出的程序演示了在循环中使用 continue 的效果。

程序清单 5-13 TestContinue.java

```
1 public class TestContinue {
2     public static void main(String[] args) {
3         int sum = 0;
4         int number = 0;
5
6         while (number < 20) {
7             number++;
8             if (number == 10 || number == 11)
9                 continue;
10            sum += number;
11        }
12
13        System.out.println("The sum is " + sum);
14    }
15 }
```

```
The sum is 189
```

程序清单 5-13 中的程序，将 1 到 20 中除去 10 和 11 外的整数都加到 sum 中。程序中有 if 语句（第 8 行），当 number 为 10 或 11 时就会执行 continue 语句。continue 语句结束了当前迭代，就不再执行循环体中的其他语句，因此，当 number 为 10 或 11 时，它就没有

被加到 sum 中。若程序中没有 if 语句，程序的输出就会如下所示：

```
The sum is 210
```

在这种情况下，即使当 number 为 10 或 11 时，也要将所有的数都加到 sum 中。因此，结果为 210，这个值比有 if 语句的情况获取的值大了 21。

注意：continue 语句总是在一个循环内。在 while 和 do-while 循环中，continue 语句之后会马上计算循环继续条件；而在 for 循环中，continue 语句之后会立即先执行每次迭代后的动作，再计算循环继续条件。

总是可以编写在循环中不使用 break 和 continue 的程序，参见复习题 5.26。通常，只有在能够简化代码并使程序更容易阅读的情况下，才适合使用 break 和 continue。

假设你需要编写一个程序，找到整数 n 的除 1 外的最小因子（假设 $n >= 2$ ）。可以使用 break 语句编写简单直观的代码，如下所示：

```
int factor = 2;
while (factor <= n) {
    if (n % factor == 0)
        break;
    factor++;
}
System.out.println("The smallest factor other than 1 for "
    + n + " is " + factor);
```

你也可以不使用 break 语句重写该代码，如下所示：

```
boolean found = false;
int factor = 2;
while (factor <= n && !found) {
    if (n % factor == 0)
        found = true;
    else
        factor++;
}
System.out.println("The smallest factor other than 1 for "
    + n + " is " + factor);
```

显然，使用 break 语句可以使程序更简单和更易读。但是，应该谨慎使用 break 和 continue。过多使用 break 和 continue 会使循环有很多退出点，使程序很难阅读。

注意：很多程序设计语言都有 goto 语句。goto 语句可以随意地将控制转移到程序中的任意一条语句上，然后执行它。这使程序很容易出错。Java 中的 break 语句和 continue 语句是不同于 goto 语句的。它们只能运行在循环中或者 switch 语句中。break 语句跳出整个循环，而 continue 语句跳出循环的当前迭代。

注意：编程是一个富于创造性的工作。有许多不同的方式来编写代码。事实上，你可以通过更加简单的代码来找到最小因子，如下所示：

```
int factor = 2;
while (factor <= n && n % factor != 0)
    factor++;
```

复习题

5.24 关键字 break 的作用是什么？关键字 continue 的作用是什么？下列程序能够结束吗？如果能，给出结果。

```
int balance = 10;
while (true) {
    if (balance < 9)
        break;
    balance = balance - 9;
}

System.out.println("Balance is "
    + balance);
```

a)

```
int balance = 10;
while (true) {
    if (balance < 9)
        continue;
    balance = balance - 9;
}

System.out.println("Balance is "
    + balance);
```

b)

5.25 将下面左边的 for 循环转换成右边的 while 循环，其中有什么错误？改正该错误。

```
int sum = 0;
for (int i = 0; i < 4; i++) {
    if (i % 3 == 0) continue;
    sum += i;
}
```

a)

转换为
错误转换

```
int i = 0, sum = 0;
while (i < 4) {
    if (i % 3 == 0) continue;
    sum += i;
    i++;
}
```

b)

5.26 不使用关键字 break 和 continue，改写程序清单 5-12 和程序清单 5-13 的程序 TestBreak 和 TestContinue。

5.27 a 中 break 语句之后，执行哪条语句？给出输出。b 中 continue 语句之后，执行哪条语句？给出输出。

```
for (int i = 1; i < 4; i++) {
    for (int j = 1; j < 4; j++) {
        if (i * j > 2)
            break;

        System.out.println(i * j);
    }

    System.out.println(i);
}
```

a)

```
for (int i = 1; i < 4; i++) {
    for (int j = 1; j < 4; j++) {
        if (i * j > 2)
            continue;

        System.out.println(i * j);
    }

    System.out.println(i);
}
```

b)

5.10 示例学习：判断回文串

要点提示：本节给出了一个程序，用于判断一个字符串是否回文。

如果一个字符串从前往后，以及从后往前是一样的，那么它就是一个回文。例如，“mom”、“dad”，以及“noon”，都是回文。

要解决的问题是，编写一个程序，提示用户输入一个字符串，然后给出该字符串是否是回文。一个解决方案是，判断字符串的第一个字符是否和最后一个字符一样。如果是，判断第二个字符是否和倒数第二个字符一样。这个过程一直持续到找到不匹配的，或者字符串中所有的字符都进行了判断。如果字符串具有奇数个字符，那么中间的字符就不需要判断了。

程序清单 5-14 给出了程序。

程序清单 5-14 Palindrome.java

```
1 import java.util.Scanner;
2
3 public class Palindrome {
```

```

4  /** Main method */
5  public static void main(String[] args) {
6      // Create a Scanner
7      Scanner input = new Scanner(System.in);
8
9      // Prompt the user to enter a string
10     System.out.print("Enter a string: ");
11     String s = input.nextLine();
12
13     // The index of the first character in the string
14     int low = 0;
15
16     // The index of the last character in the string
17     int high = s.length() - 1;
18
19     boolean isPalindrome = true;
20     while (low < high) {
21         if (s.charAt(low) != s.charAt(high)) {
22             isPalindrome = false;
23             break;
24         }
25
26         low++;
27         high--;
28     }
29
30     if (isPalindrome)
31         System.out.println(s + " is a palindrome");
32     else
33         System.out.println(s + " is not a palindrome");
34 }
35 }

```

```

Enter a string: noon
noon is a palindrome

```

```

Enter a string: moon
moon is not a palindrome

```

程序使用两个变量，`low` 和 `high`，表示位于字符串 `s` 中开始和末尾的两个字符的位置（第 14、17 行）。初始时，`low` 为 0，`high` 为 `s.length()-1`。如果位于这两个位置的字符匹配，则将 `low` 加 1，`high` 减 1（第 26 ~ 27 行）。这个过程一直继续到（`low>=high`），或者找到一个不匹配（第 21 行）。

程序使用一个 `boolean` 变量 `isPalindrome` 来表示字符串 `s` 是否回文。初始时，该变量设置为 `true`（第 19 行）。当一个不匹配出现的时候（第 21 行），`isPalindrome` 设置为 `false`（第 22 行），循环由一个 `break` 语句结束（第 23 行）。

5.11 示例学习：显示素数

 **要点提示：** 本节给出了一个程序，用于分 5 行显示前 50 个素数，每行包含 10 个数字。

大于 1 的整数，如果它的正因子只有 1 和它自身，那么该整数就是素数。例如：2、3、5、7 都是素数，而 4、6、8、9 不是。

现在的问题是在 5 行中显示前 50 个素数，每行包含 10 个数。该问题可分解成以下任务：

- 判断一个给定数是否是素数。

- 针对 $number=2, 3, 4, 5, 6, \dots$, 测试它是否为素数。
- 统计素数的个数。
- 打印每个素数, 每行打印 10 个。

显然, 需要编写循环, 反复检测新的 $number$ 是否是素数。如果 $number$ 是素数, 则给计数器加 1。计数器 $count$ 被初始化为 0。当它等于 50 时, 循环终止。

下面是该问题的算法:

```
设置打印出来的素数个数为常量 NUMBER_OF_PRIMES;
使用 count 来对素数个数进行计数并将其初值设为 0;
设置 number 初始值为 2;
while (count<NUMBER_OF_PRIMES){
    测试该数是否是素数;
    if 该数是素数 {
        打印该素数并给 count 增加 1;
    }
    给 number 加 1;
}
```

为了测试某个数是否是素数, 就要检测它是否能被 2、3、4, 一直到 $number/2$ 的整数整除。如果能被整除, 那它就不是素数。这个算法可以描述如下:

```
使用布尔变量 isPrime 表示 number 是否是素数; 设置 isPrime 的初值为 true;
for(int divisor =2; divisor<=number/2; divisor++){
    if(number%divisor==0){
        将 isPrime 设置为 false
        退出循环;
    }
}
```

完整的程序在程序清单 5-15 中给出。

程序清单 5-15 PrimeNumber.java

```
1 public class PrimeNumber {
2     public static void main(String[] args) {
3         final int NUMBER_OF_PRIMES = 50; // Number of primes to display
4         final int NUMBER_OF_PRIMES_PER_LINE = 10; // Display 10 per line
5         int count = 0; // Count the number of prime numbers
6         int number = 2; // A number to be tested for primeness
7
8         System.out.println("The first 50 prime numbers are \n");
9
10        // Repeatedly find prime numbers
11        while (count < NUMBER_OF_PRIMES) {
12            // Assume the number is prime
13            boolean isPrime = true; // Is the current number prime?
14
15            // Test whether number is prime
16            for (int divisor = 2; divisor <= number / 2; divisor++) {
17                if (number % divisor == 0) { // If true, number is not prime
18                    isPrime = false; // Set isPrime to false
19                    break; // Exit the for loop
20                }
21            }
22        }
```

```

23     // Display the prime number and increase the count
24     if (isPrime) {
25         count++; // Increase the count
26
27         if (count % NUMBER_OF_PRIMES_PER_LINE == 0) {
28             // Display the number and advance to the new line
29             System.out.println(number);
30         }
31         else
32             System.out.print(number + " ");
33     }
34
35     // Check if the next number is prime
36     number++;
37 }
38 }
39 }

```

```

The first 50 prime numbers are
2 3 5 7 11 13 17 19 23 29
31 37 41 43 47 53 59 61 67 71
73 79 83 89 97 101 103 107 109 113
127 131 137 139 149 151 157 163 167 173
179 181 191 193 197 199 211 223 227 229

```

对编程新手而言这是一个复杂的例子。开发编程解决方案来解决这个问题或其他很多问题的关键之处在于要把问题分解成子问题，然后逐个地开发出每个子问题的解决方案。不要一开始就试图开发出一个完整的解决方案。而是应该首先编写代码判断一个给定的数是否是素数，然后扩展这个程序，再在循环中判断其他数是否是素数。

为了判断一个数是否是素数，检验该数是否能被2到 $\text{number}/2$ 之间并包括2和 $\text{number}/2$ 的整数整除（第16~21行）。如果能被整除，那它就不是素数（第18行）；否则，它就是一个素数。若是素数，就显示该数。若 count 能被10整除（第27~30行），就转入一个新行。当计数器 count 达到50时，程序终止。

程序在第19行使用`break`语句，一旦发现 number 不是素数，就立即退出`for`循环。也可以不用`break`语句，改写这个循环（第16~21行），如下所示：

```

for (int divisor = 2; divisor <= number / 2 && isPrime;
    divisor++) {
    // If true, the number is not prime
    if (number % divisor == 0) {
        // Set isPrime to false, if the number is not prime
        isPrime = false;
    }
}
}

```

然而，在本例中，使用`break`语句可以使程序更简单、更易读。

关键术语

`break` statement (break 语句)

input redirection (输入重定向)

`continue` statement (continue 语句)

iteration (迭代)

do-while loop (do-while 循环)

loop (循环)

for loop (for 循环)

loop body (循环体)

infinite loop (无限循环、死循环)

nested loop (嵌套循环)

off-by-one error (差一错误)

output redirection (输出重定向)

posttest loop (后测循环)

pretest loop (前测循环)

sentinel value (标志值)

while loop (while 循环)

本章小结

1. 循环语句有三类：`while` 循环、`do-while` 循环和 `for` 循环。
2. 循环中包含重复执行的语句的部分称为循环体。
3. 循环体执行一次称为循环的一次迭代。
4. 无限循环是指循环语句被无限次执行。
5. 在设计循环时，既需要考虑循环控制结构，还需要考虑循环体。
6. `while` 循环首先检查循环继续条件。如果条件为 `true`，则执行循环体；如果条件为 `false`，则循环结束。
7. `do-while` 循环与 `while` 循环类似，只是 `do-while` 循环先执行循环体，然后再检查循环继续条件，以确定是继续还是终止。
8. `while` 和 `do-while` 循环常用于循环次数不确定的情况。
9. 标记值是一个特殊的值，用来标记循环的结束。
10. `for` 循环一般用在循环体执行次数固定的情况。
11. `for` 循环控制由三部分组成。第一部分是初始操作，通常用于初始化控制变量。第二部分是循环继续条件，决定是否执行循环体。第三部分是每次迭代后执行的操作，经常用于调整控制变量。通常，在控制结构中初始化和修改循环控制变量。
12. `while` 循环和 `for` 循环都称为前测循环 (`pretest loop`)，因为在循环体执行之前，要检测一下循环继续条件。
13. `do-while` 循环称为后测循环 (`posttest loop`)，因为在循环体执行之后，要检测一下这个条件。
14. 在循环中可以使用 `break` 和 `continue` 这两个关键字。
15. 关键字 `break` 立即终止包含 `break` 的最内层循环。
16. 关键字 `continue` 只是终止当前迭代。

测试题

在线回答本章测试题，地址为 www.cs.armstrong.edu/liang/intro10e/quiz.html。

编程练习题

 **教学提示：**对每个问题都应该多读几遍，直到理解透彻为止。在编码之前，思考一下如何解决这个问题。然后将你的逻辑翻译成程序。

通常，一个问题可以有多种不同的解决方法。鼓励学生探索不同的解决方案。

5.2 ~ 5.7 节

- *5.1 (统计正数和负数的个数然后计算这些数的平均值) 编写程序，读入未指定个数的整数，判断读入的正数有多少个，读入的负数有多少个，然后计算这些输入值的总和及其平均值 (不对 0 计数)。当输入为 0 时，表明程序结束。将平均值以浮点数显示。下面是一个运行示例：

```
Enter an integer, the input ends if it is 0: 1 2 -1 3 0 
The number of positives is 3
The number of negatives is 1
The total is 5.0
The average is 1.25
```

Enter an integer, the input ends if it is 0: 0
 No numbers are entered except 0

5.2 (重复加法) 程序清单 5-4 产生了 5 个随机减法问题。改写该程序, 使它产生 10 个随机加法问题, 加数是两个 1 到 15 之间的整数。显示正确答案的个数和测验时间。

5.3 (将千克转换成磅) 编写程序, 显示下面的表格 (注意: 1 千克为 2.2 磅)。

千克	磅
1	2.2
3	6.6
...	
197	433.4
199	437.8

5.4 (将英里转换成千米) 编写程序, 显示下面的表格 (注意: 1 英里为 1.609 千米)。

英里	千米
1	1.609
2	3.218
...	
9	14.481
10	16.090

5.5 (千克与磅之间的互换) 编写一个程序, 并排显示下列两个表格。

千克	磅	磅	千克
1	2.2	20	9.09
3	6.6	25	11.36
...			
197	433.4	510	231.82
199	437.8	515	234.09

5.6 (英里与千米之间的互换) 编写一个程序, 并排显示下列两个表格。

英里	千米	千米	英里
1	1.609	20	12.430
2	3.218	25	15.538
...			
9	14.481	60	37.290
10	16.090	65	40.398

**5.7 (财务应用程序: 计算将来的学费) 假设今年某大学的学费为 10 000 美元, 学费的年增长率为 5%。一年后, 学费将是 10 500 美元。编写程序, 计算 10 年后的学费, 以及从现在开始 10 年后算起, 4 年内总学费是多少?

5.8 (找出最高分) 编写程序, 提示用户输入学生的个数、每个学生的名字及其分数, 最后显示得最高分的学生的名字。

*5.9 (找出两个分数最高的学生) 编写程序, 提示用户输入学生的个数、每个学生的名字及其分数, 最后显示获得最高分的学生和第二高分的学生。

5.10 (找出能被 5 和 6 整除的数) 编写程序, 显示从 100 到 1000 之间所有能被 5 和 6 整除的数, 每行显示 10 个。数字之间用一个空格字符隔开。

5.11 (找出能被 5 或 6 整除, 但不能被两者同时整除的数) 编写程序, 显示从 100 到 200 之间所有能被 5 或 6 整除, 但不能被两者同时整除的数, 每行显示 10 个数。数字之间用一个空格字符隔开。

5.12 (求满足 $n^2 > 12\,000$ 的 n 的最小值) 使用 while 循环找出满足 n^2 大于 12 000 的最小整数 n 。

5.13 (求满足 $n^3 < 12\,000$ 的 n 的最大值) 用 while 循环找出满足 n^3 小于 12 000 的最大整数 n 。

5.8 ~ 5.10 节

*5.14 (计算最大公约数) 下面是求两个整数 n_1 和 n_2 的最大公约数的程序清单 5-9 的另一种解法: 首先找出 n_1 和 n_2 的最小值 d , 然后依次检验 $d, d-1, d-2, \dots, 2, 1$ 是否是 n_1 和 n_2 的公约数。

第一个满足条件的公约数就是 n1 和 n2 的最大公约数。编写程序，提示用户输入两个正整数，然后显示最大公约数。

- *5.15 (显示 ASCII 码字符表) 编写一个程序，打印 ASCII 字符表从 '!' 到 '~' 的字符。每行打印 10 个字符。ASCII 码表如附录 B 所示。数字之间用一个空格字符隔开。
- *5.16 (找出一个整数的因子) 编写程序，读入一个整数，然后以升序显示它的所有最小因子。例如，若输入的整数是 120，那么输出就应该是：2，2，2，3，5。
- **5.17 (显示金字塔) 编写程序，提示用户输入一个在 1 到 15 之间的整数，然后显示一个金字塔形状的模式，如下面的运行示例所示：

```

Enter the number of lines: 7 
          1
         2 1 2
        3 2 1 2 3
       4 3 2 1 2 3 4
      5 4 3 2 1 2 3 4 5
     6 5 4 3 2 1 2 3 4 5 6
    7 6 5 4 3 2 1 2 3 4 5 6 7
  
```

- *5.18 (使用循环语句打印 4 个图案) 使用嵌套的循环语句，用四个独立的程序打印下面的图案：

图案 1	图案 2	图案 3	图案 4
1	1 2 3 4 5 6	1	1 2 3 4 5 6
1 2	1 2 3 4 5	2 1	1 2 3 4 5
1 2 3	1 2 3 4	3 2 1	1 2 3 4
1 2 3 4	1 2 3	4 3 2 1	1 2 3
1 2 3 4 5	1 2	5 4 3 2 1	1 2
1 2 3 4 5 6	1	6 5 4 3 2 1	1

- **5.19 (打印金字塔形的数字) 编写一个嵌套的 for 循环，打印下面的输出：

```

          1
        1 2 1
      1 2 4 2 1
    1 2 4 8 4 2 1
  1 2 4 8 16 8 4 2 1
1 2 4 8 16 32 16 8 4 2 1
1 2 4 8 16 32 64 32 16 8 4 2 1
  
```

- *5.20 (打印 2 到 1000 之间的素数) 修改程序清单 5-15，打印 2 到 1000 之间、包括 2 和 1000 的所有素数，每行显示 8 个素数。数字之间用一个空格字符隔开。

综合题

- **5.21 (财务应用程序：比较不同利率下的贷款) 编写程序，让用户输入贷款总额和以年为单位的贷款期限，然后显示利率从 5% 到 8%，每次递增 1/8 的过程中，每月的支付额和总支付额。下面是一个运行示例：

```

Loan Amount: 10000 
Number of Years: 5 
Interest Rate    Monthly Payment    Total Payment
5.000%          188.71                11322.74
5.125%          189.29                11357.13
5.250%          189.86                11391.59
  
```

...		
7.875%	202.17	12129.97
8.000%	202.76	12165.84

计算月支付额的公式，请参见程序清单 2-9。

- **5.22 (财务应用程序：显示分期还贷时间表) 对于给定的贷款额的月支付额包括偿还本金及利息。月利息是通过月利率乘以余额(剩余本金)计算出来的。因此，每月偿还的本金等于月支付额减去月利息。编写一个程序，让用户输入贷款总额、贷款年数以及利率，然后显示分期还贷时间表。下面是一个运行示例：

Loan Amount:	10000	<input type="button" value="Enter"/>	
Number of Years:	1	<input type="button" value="Enter"/>	
Annual Interest Rate:	7	<input type="button" value="Enter"/>	
Monthly Payment: 865.26			
Total Payment: 10383.21			
Payment#	Interest	Principal	Balance
1	58.33	806.93	9193.07
2	53.62	811.64	8381.43
...			
11	10.0	855.26	860.27
12	5.01	860.25	0.01

- 注意：最后一次偿还后，余额可能不为 0。如果是这样的话，最后一个月支付额应当是正常的月支付额加上最后的余额。
- 提示：编写一个循环来打印该表。由于每个月的还贷额都是相同的，因此，应当在循环之前计算它。开始时，余额就是贷款总额。在循环的每次迭代中，计算利息及本金，然后更新余额。这个循环可能会是这样的：

```
for (i = 1; i <= numberOfYears * 12; i++) {
    interest = monthlyInterestRate * balance;
    principal = monthlyPayment - interest;
    balance = balance - principal;
    System.out.println(i + "\t\t" + interest
        + "\t\t" + principal + "\t\t" + balance);
}
```

- *5.23 (示例抵消错误) 当处理一个很大的数字以及一个很小的数字的时候，会产生一个抵消错误(cancellation error)。例如， $100\ 000\ 000.0 + 0.000\ 000\ 001$ 等于 $100\ 000\ 000.0$ 。为了避免抵消错误，从而获得更加精确的结果，谨慎选择计算的次序。比如，在计算下面的数列时，从右到左计算要比从左到右计算得到的结果更精确：

$$1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$$

编写程序对上面的数列从左到右和从右到左计算的结果进行比较，这里取 $n=50000$ 。

- *5.24 (数列求和) 编写程序，计算下面数列的和：

$$\frac{1}{3} + \frac{3}{5} + \frac{5}{7} + \frac{7}{9} + \frac{9}{11} + \frac{11}{13} + \dots + \frac{95}{97} + \frac{97}{99}$$

- **5.25 (计算 π) 使用下面的数列可以近似计算 π ：

$$\pi = 4 \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \dots + \frac{(-1)^{i+1}}{2i-1} \right)$$

编写程序，显示当 $i=10000, 20000, \dots, 100000$ 时 π 的值。

**5.26 (计算 e) 使用下面的数列可以近似计算 e:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \dots + \frac{1}{i!}$$

编写程序，显示当 $i=10000, 20000, \dots, 100000$ 时 e 的值。

提示: 由于 $i! = i \times (i-1) \times \dots \times 2 \times 1$, 那么 $\frac{1}{i!} = \frac{1}{i(i-1)!}$ 。将 e 和通项 item 初始化为 1, 反复将新的

item 加到 e 上。新的 item 由前一个 item 除以 i 得到, 其中 $i=2, 3, 4, \dots$ 。

**5.27 (显示闰年) 编写程序, 显示从 101 到 2100 期间所有的闰年, 每行显示 10 个。数字之间用一个空格字符隔开, 同时显示这期间闰年的数目。

**5.28 (显示每月第一天是星期几) 编写程序, 提示用户输入年份和代表该年第一天是星期几的数字, 然后在控制台上显示该年每月第一天的星期。例如, 如果用户输入的年份是 2013 和代表 2013 年 1 月 1 日为星期二的 2, 程序应该显示如下输出:

January 1, 2013 is Tuesday

...

December 1, 2013 is Sunday

**5.29 (显示日历) 编写程序, 提示用户输入年份和代表该年第一天是星期几的数字, 然后在控制台上显示该年的日历表。例如, 如果用户输入年份 2013 和代表 2013 年 1 月 1 日为星期二的 2, 程序应该显示该年每个月的日历, 如下所示:

January 2013

Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

December 2013

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

*5.30 (财务应用程序: 复利值) 假设你每月在储蓄账户上存 100 美元, 年利率是 5%。那么每月利率是 $0.05/12=0.00417$ 。在第一个月之后, 账户上的值变成:

$$100 * (1 + 0.00417) = 100.417$$

第二个月之后, 账户上的值变成:

$$(100 + 100.417) * (1 + 0.00417) = 201.252$$

第三个月之后, 账户上的值变成:

$$(100 + 201.252) * (1 + 0.00417) = 302.507$$

依此类推。

编写程序提示用户输入一个数目(例如:100)、年利率(例如:5)以及月份数(例如:6),然后显示给定月份后账户上的钱数。

- *5.31 (财务应用程序:计算CD价值)假设你投资10 000美元投资一张CD,年获利率为5.75%。一个月后,这张CD价值为

$$10000 + 10000 * 5.75 / 1200 = 10047.92$$

两个月之后,这张CD价值为

$$10047.91 + 10047.91 * 5.75 / 1200 = 10096.06$$

三个月之后,这张CD价值为

$$10096.06 + 10096.06 * 5.75 / 1200 = 10144.44$$

依此类推。

编写程序,提示用户输入一个总数(例如:10 000)、年获利率(例如:5.75)以及月份数(例如:18),然后显示一个表格,如下面的运行示例所示:

```

Enter the initial deposit amount: 10000 Enter
Enter annual percentage yield: 5.75 Enter
Enter maturity period (number of months): 18 Enter

Month  CD Value
1      10047.92
2      10096.06
...
17     10846.57
18     10898.54
  
```

- **5.32 (游戏:彩票)修改程序清单3-8,产生一个两位整数的彩票。这个数中的两个整数是两个不同的数。

提示:产生第一个数,使用循环不断产生第二个数,直到它和第一个数不同为止。

- **5.33 (完全数)如果一个正整数等于除它本身之外其他所有除数之和,就称之为完全数。例如:6是第一个完全数,因为 $6=1+2+3$ 。下一个完全数是 $28=1+2+4+7+14$ 。10 000以下的完全数有四个。编写程序,找出这四个完全数。

- ***5.34 (游戏:石头、剪刀、布)编程练习题3.17给出玩石头-剪刀-布游戏的程序。修改这个程序,让用户可以连续地玩这个游戏,直到用户或者计算机赢对手两次以上为止。

- *5.35 (加法)编写程序,计算下面的和。

$$\frac{1}{1+\sqrt{2}} + \frac{1}{\sqrt{2}+\sqrt{3}} + \frac{1}{\sqrt{3}+\sqrt{4}} + \dots + \frac{1}{\sqrt{624}+\sqrt{625}}$$

- **5.36 (商业应用程序:检测ISBN)使用循环简化编程练习题3.9。

- **5.37 (十进制到二进制)编写程序,提示用户输入一个十进制整数,然后显示对应的二进制值。在这个程序中不要使用Java的Integer.toString(int)方法。

- **5.38 (十进制到八进制)编写程序,提示用户输入一个十进制整数,然后显示对应的八进制值。在这个程序中不要使用Java的Integer.toString(int)方法。

- *5.39 (财务应用程序:求出销售总额)假设你已经在某百货商店开始销售工作。你的工资包括基本工资和提成。基本工资是5000美元。使用下面的方案确定你的提成率。

销售额	提成率
0.01 ~ 5000 美元	8%
5000.01 ~ 10 000 美元	10%
10 000.01 及以上	12%

注意：这是一个渐进税率。第一个 5 000 美元的税率是 8%，下一个 5 000 美元是 10%，余下的是 12%。如果销售额是 25 000，提成则为 $5\,000 * 8\% + 5\,000 * 10\% + 15\,000 * 12\% = 2\,700$ 。

你的目标是一年挣 30 000 美元。编写程序找出为挣到 30 000 美元，你所必须完成的最小销售额。

5.40 (模拟：正面或反面) 编写程序，模拟抛硬币一百万次，显示出现正面和反面的次数。

*5.41 (最大数的出现次数) 编写程序读取整数，找出它们的最大数，然后计算该数的出现次数。假设输入是以 0 结束的。假定输入是 3 5 2 5 5 5 0，程序找出最大数 5，而 5 出现的次数是 4。

提示：维护 max 和 count 两个变量。max 存储当前最大数，而 count 存储它的出现次数。初始状态时，将第一个数赋值给 max 而将 count 赋值为 1。然后将接下来的每个数字逐个地和 max 进行比较。如果这个数大于 max，就将它赋值给 max，同时将 count 重置为 1。如果这个数等于 max，就给 count 加 1。

```
Enter numbers: 3 5 2 5 5 5 0
The largest number is 5
The occurrence count of the largest number is 4
```

*5.42 (财务应用程序：求出销售额) 如下改写编程练习题 5.39：

- 使用 for 循环替代 do-while 循环。
- 允许用户自己输入 COMMISSION_SUGHT 而不是将它固定为一个常量。

*5.43 (数学方面：组合) 编写程序，显示从整数 1 到 7 中选择两个数字的所有组合，同时显示所有组合的总数。

```
1 2
1 3
...
...

The total number of all combinations is 21
```

*5.44 (计算机体系结构：比特级的操作) 一个 short 型值用 16 位比特存储。编写程序，提示用户输入一个短整型，然后显示这个整数的 16 比特形式。下面是一个运行示例：

```
Enter an integer: 5
The bits are 0000000000000101
```

```
Enter an integer: -5
The bits are 1111111111111011
```

提示：需要使用按位右移操作符 (>>) 以及按位 AND 操作符 (&)，详见附录 G。

**5.45 (统计：计算平均值和标准方差) 在商务应用程序中经常需要计算数据的平均值和标准方差。平均值就是数字的简单平均。标准方差则是一个统计数字，给出了在一个数字集中各种数据到底离开平均值的聚集度有多紧密。例如，一个班级的学生的平均年龄是多少？年龄相差近吗？如果所有的学生都是同龄的，那么方差为 0。

编写一个程序，提示用户输入 10 个数字，然后运用下面的公式，显示这些数字的平均数以及标准方差。

$$\text{平均值} = \frac{\sum_{i=1}^n x_i}{n} = \frac{x_1 + x_2 + \dots + x_n}{n}$$

$$\text{方差} = \sqrt{\frac{\sum_{i=1}^n x_i^2 - \frac{(\sum_{i=1}^n x_i)^2}{n}}{n-1}}$$

下面是一个运行示例：

```
Enter ten numbers: 1 2 3 4.5 5.6 6 7 8 9 10
The mean is 5.61
The standard deviation is 2.99794
```

- *5.46 (倒排一个字符串) 编写一个程序, 提示用户输入一个字符串, 然后以反序显示该字符串。

```
Enter a string: ABCD ↵
The reversed string is DCBA
```

- *5.47 (商业: 检测 ISBN-13) ISBN-13 是一个标识书籍的新标准。它使用 13 位数字 $d_1d_2d_3d_4d_5d_6d_7d_8d_9d_{10}d_{11}d_{12}d_{13}$ 。最后一位数字 d_{13} 是一个校验和, 是使用下面的公式从其他数字中计算出来的:

$$10 - (d_1 + 3d_2 + d_3 + 3d_4 + d_5 + 3d_6 + d_7 + 3d_8 + d_9 + 3d_{10} + d_{11} + 3d_{12}) \% 10$$

如果校验和为 10, 将其替换为 0。程序应该将输入作为一个字符串读入。下面是一个运行示例:

```
Enter the first 12 digits of an ISBN-13 as a string: 978013213080 ↵
The ISBN-13 number is 9780132130806
```

```
Enter the first 12 digits of an ISBN-13 as a string: 978013213079 ↵
The ISBN-13 number is 9780132130790
```

```
Enter the first 12 digits of an ISBN-13 as a string: 97801320 ↵
97801320 is an invalid input
```

- *5.48 (处理字符串) 编写一个程序, 提示用户输入一个字符串, 显示奇数位置的字符。下面是一个运行示例:

```
Enter a string: Beijing Chicago ↵
BiigCiao
```

- *5.49 (对元音和辅音进行计数) 假设字母 A、E、I、O、U 为元音。编写一个程序, 提示用户输入一个字符串, 然后显示字符串中元音和辅音的数目。

```
Enter a string: Programming is fun ↵
The number of vowels is 5
The number of consonants is 11
```

- *5.50 (对大写字母计数) 编写一个程序, 提示用户输入一个字符串, 然后显示该字符串中大写字母的数目。

```
Enter a string: Welcome to Java ↵
The number of uppercase letters is 2
```

- *5.51 (最长的共同前缀) 编写一个程序, 提示用户输入两个字符串, 显示两个字符串最长的共同前缀。下面是运行示例:

```
Enter the first string: Welcome to C++ ↵
Enter the second string: Welcome to programming ↵
The common prefix is Welcome to
```

```
Enter the first string: Atlanta ↵
Enter the second string: Macon ↵
Atlanta and Macon have no common prefix
```

方 法

教学目标

- 使用形参定义方法 (6.2 节)。
- 使用实参调用方法 (6.2 节)。
- 定义带返回值的方法 (6.3 节)。
- 定义无返回值的方法 (6.4 节)。
- 按值传参 (6.5 节)。
- 开发模块化的、易读、易调试和易维护的可重用代码 (6.6 节)。
- 编写方法, 将十进制数转换为十六进制数 (6.7 节)。
- 使用方法重载, 理解歧义重载 (6.8 节)。
- 确定变量的作用域 (6.9 节)。
- 在软件开发中应用方法抽象的概念 (6.10 节)。
- 使用逐步求精的办法设计和实现方法 (6.11 节)。

6.1 引言

要点提示: 方法可以用于定义可重用的代码以及组织和简化编码。

假如需要分别求出从 1 到 10、从 20 到 37 以及从 35 到 49 的整数和, 可以编写如下代码:

```
int sum = 0;
for (int i = 1; i <= 10; i++)
    sum += i;
System.out.println("Sum from 1 to 10 is " + sum);
```

```
sum = 0;
for (int i = 20; i <= 37; i++)
    sum += i;
System.out.println("Sum from 20 to 37 is " + sum);
```

```
sum = 0;
for (int i = 35; i <= 49; i++)
    sum += i;
System.out.println("Sum from 35 to 49 is " + sum);
```

你会发现计算从 1 到 10、从 20 到 37 以及从 35 到 49 的整数和, 除了开始的数和结尾的数不同之外, 其他都是非常类似的。如果可以一次性地编写好通用的代码而无须重新编写, 难道不是更好吗? 可以通过定义和调用方法实现该功能。

上面的代码可以简化为如下所示:

```
1 public static int sum(int i1, int i2) {
2     int result = 0;
3     for (int i = i1; i <= i2; i++)
4         result += i;
5
6     return result;
```

```

7 }
8
9 public static void main(String[] args) {
10     System.out.println("Sum from 1 to 10 is " + sum(1, 10));
11     System.out.println("Sum from 20 to 37 is " + sum(20, 37));
12     System.out.println("Sum from 35 to 49 is " + sum(35, 49));
13 }

```

第1~7行定义一个名为 `sum` 的方法，该方法带有两个参数 `i1` 和 `i2`。`main` 方法中的语句调用 `sum(1,10)` 计算从1到10的整数和，`sum(20,37)` 计算从20到37的整数和，而 `sum(35,49)` 计算从35到49的整数和。

方法是为一个操作而组合在一起的语句组。在前面的章节里，已经使用过预定义的方法，例如：`System.out.println`、`System.exit`、`Math.pow` 和 `Math.random`，这些方法都在Java库中定义。在本章里，我们将学习如何定义自己的方法以及应用方法抽象来解决复杂问题。

6.2 定义方法

要点提示：方法的定义由方法名称、参数、返回值类型以及方法体组成。

定义方法的语法如下所示：

```

修饰符 返回值类型 方法名 (参数列表) {
// 方法体 ;
}

```

我们一起来看看一个方法的定义，该方法找出两个整数中哪个数比较大。这个名为 `max` 的方法有两个 `int` 型参数：`num1` 和 `num2`，方法返回两个数中较大的一个。图6-1解释了这个方法的组成。

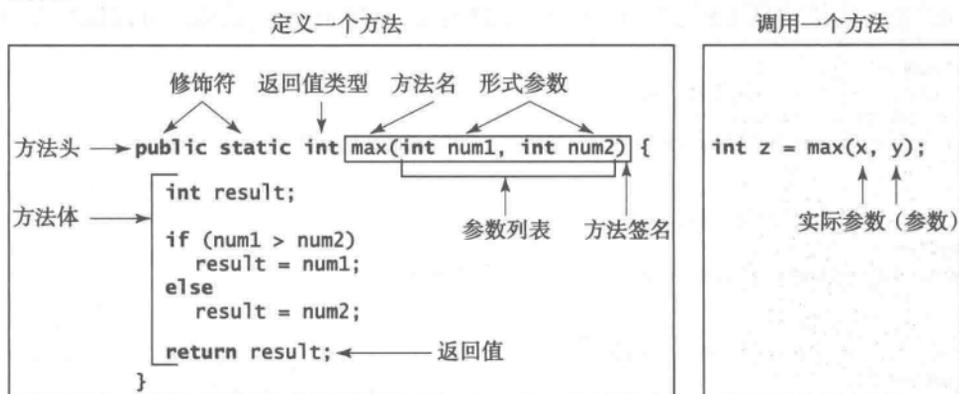


图 6-1 方法定义包括方法头和方法体

方法头 (method header) 是指方法的修饰符 (modifier)、返回值类型 (return value type)、方法名 (method name) 和方法的参数 (parameter)。本章的所有方法都使用静态修饰符 `static`，使用它的理由将在第9章中深入讨论。

方法可以返回一个值。`returnValueType` 是方法返回值的数据类型。有些方法只是完成某些要求的操作，而不返回值。在这种情况下，`returnValueType` 为关键字 `void`。例如：在 `main` 方法中 `returnValueType` 就是 `void`，在 `System.exit`、`System.out.println` 方法中返回值类型也是如此。如果方法有返回值，则称为带返回值的方法 (value-returning method)，否则就称这个该方法为 `void` 方法 (void method)。

定义在方法头中的变量称为形式参数 (formal parameter) 或者简称为形参 (parameter)。参数就像占位符。当调用方法时, 就给参数传递一个值, 这个值称为实际参数 (actual parameter) 或实参 (argument)。参数列表 (parameter list) 指明方法中参数的类型、顺序和个数。方法名和参数列表一起构成方法签名 (method signature)。参数是可选的, 也就是说, 方法可以不包含参数。例如: `Math.random()` 方法就没有参数。

方法体中包含一个执行方法的语句集合。`max` 方法的方法体使用一个 `if` 语句来判断哪个数较大, 然后返回该数的值。为使带返回值的方法能返回一个结果, 必须要使用带关键字 `return` 的返回语句。执行 `return` 语句时方法终止。

🔧 **注意:** 在其他某些语言中, 方法称为过程 (procedure) 或函数 (function)。带返回值的方法称为函数, 返回值类型为 `void` 的方法称为过程。

🔧 **警告:** 在方法头中, 需要对每一个参数进行独立的数据类型声明。例如: `max(int num1, int num2)` 是正确的, 而 `max(int num1, num2)` 是错误的。

🔧 **注意:** 我们经常会说“定义方法”和“声明变量”, 这里我们谈谈两者的细微差别。定义是指被定义的条目是什么, 而声明通常是指为被声明的条目分配内存来存储数据。

6.3 调用方法

🔧 **要点提示:** 方法的调用是执行方法中的代码。

在方法定义中, 定义方法要做什么。为了使用方法, 必须调用 (call 或 invoke) 它。根据方法是否有返回值, 调用方法有两种途径。

如果方法返回一个值, 对方法的调用通常就当作一个值处理。例如:

```
int larger = max(3, 4);
```

调用方法 `max(3,4)` 并将其结果赋给变量 `larger`。另一个把它当作值处理的调用例子是:

```
System.out.println(max(3, 4));
```

这条语句打印调用方法 `max(3,4)` 后的返回值。

如果方法返回 `void`, 对方法的调用必须是一条语句。例如, `println` 方法返回 `void`。下面的调用就是一条语句:

```
System.out.println("Welcome to Java!");
```

🔧 **注意:** 在 Java 中, 带返回值的方法也可以当作语句调用。这种情况下, 函数调用者只需忽略返回值即可。虽然这种情况很少见, 但是, 如果调用者对返回值不感兴趣, 这样也是允许的。

当程序调用一个方法时, 程序控制就转移到被调用的方法。当执行完 `return` 语句或执行到表示方法结束的右括号时, 被调用的方法将程序控制返还给调用者。

程序清单 6-1 给出了测试 `max` 方法的完整程序。

程序清单 6-1 TestMax.java

```
1 public class TestMax {
2     /** Main method */
3     public static void main(String[] args) {
4         int i = 5;
5         int j = 2;
6         int k = max(i, j);
```

```

7     System.out.println("The maximum of " + i +
8         " and " + j + " is " + k);
9     }
10
11    /** Return the max of two numbers */
12    public static int max(int num1, int num2) {
13        int result;
14
15        if (num1 > num2)
16            result = num1;
17        else
18            result = num2;
19
20        return result;
21    }
22 }

```

The maximum of 5 and 2 is 5

	line#	i	j	k	num1	num2	result
	4	5					
	5		2				
Invoking max	12				5	2	
	13						undefined
	16						5
	6			5			

这个程序包括 main 方法和 max 方法。main 方法与其他方法很类似，区别在于它是由 Java 虚拟机调用的。

main 方法的方法头永远都是一样的。就像这个例子中的 main 方法一样，它包括修饰符 public 和 static，返回值类型 void，方法名 main，String[] 类型的参数。String[] 表明参数是一个 String 型数组，数组是第 7 章将介绍的主题。

main 中的语句可以调用 main 方法所在类中定义的其他方法，也可以调用别的类中定义的方法。在本例中，main 方法调用方法 max(i, j)，该方法与 main 方法在同一个类中定义。

当调用 max 方法时（第 6 行），将变量 i 的值 5 传递给 max 方法中的 num1，变量 j 的值 2 传递给 max 方法中的 num2。控制流程转向 max 方法，执行 max 方法。当执行 max 方法中的 return 语句时，max 方法将程序控制返还给它的调用者（在此例中，调用者是 main 方法）。这个过程如图 6-2 所示。

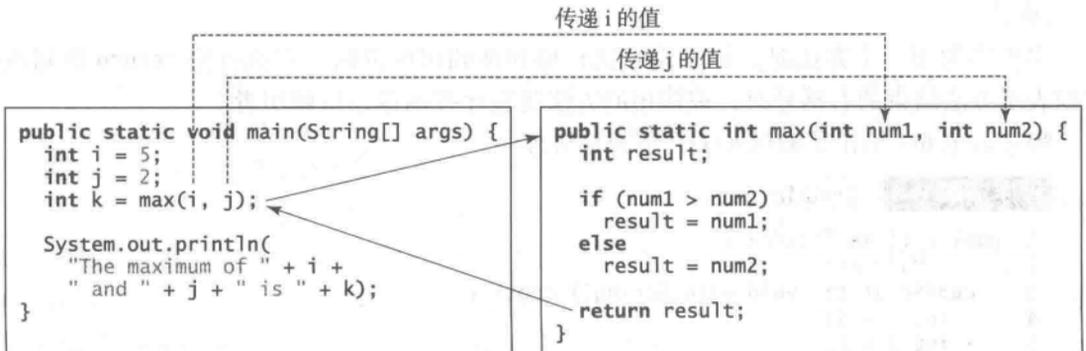
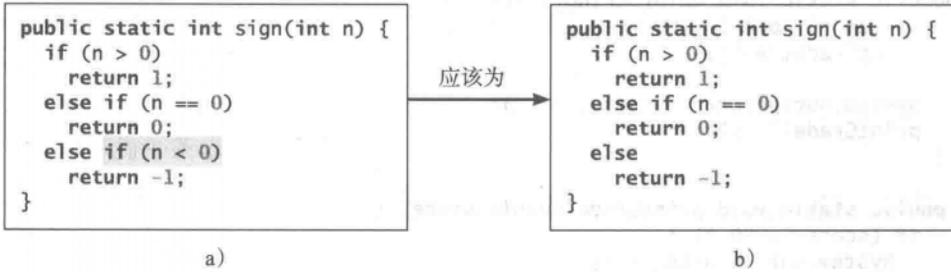


图 6-2 当调用 max 方法时，控制流程转给 max 方法。一旦 max 方法结束，将控制返还给调用者

警告：对带返回值的方法而言，return 语句是必需的。下面图 a 中显示的方法在逻辑上是正确的，但它会有编译错误，因为 Java 编译器认为该方法有可能不会返回任何值。



为解决这个问题，删除图 a 中的 if(n<0)，这样，编译器将发现不管 if 语句如何执行，总可以执行到 return 语句。

注意：方法能够带来代码的共享和重用。除了可以在 TestMax 中调用 max 方法，还可以在其他类中调用它。如果创建了一个新类，可以通过使用“类名.方法名”（即 TestMax.max）来调用 max 方法。

每当调用一个方法时，系统会创建一个活动记录（也称为活动框架），用于保存方法中的参数和变量。活动记录置于一个内存区域中，称为调用堆栈（call stack）。调用堆栈也称为执行堆栈、运行时堆栈，或者一个机器堆栈，常简称为“堆栈”。当一个方法调用另一个方法时，调用者的活动记录保持不动，一个新的活动记录被创建用于被调用的新方法。一个方法结束返回到调用者时，其相应的活动记录也被释放。

理解调用堆栈有助于理解方法是如何调用的。程序清单 6-1 中 main 方法定义的变量是 i、j 和 k；max 方法中定义的变量是 num1、num2 和 result。定义在方法签名中的变量 num1 和 num2 都是方法 max 的参数。它们的值通过方法调用进行传递。图 6-3 描述了堆栈中用于方法调用的活动记录。

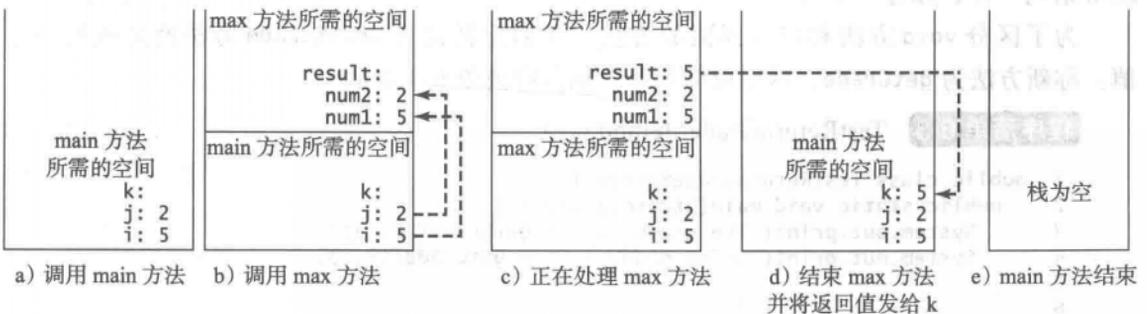


图 6-3 调用 max 方法时，程序控制流转到 max 方法；

一旦 max 方法结束，就将程序控制还给调用者

6.4 void 方法示例

要点提示：void 方法不返回值。

前一节给出一个带返回值方法的例子，本节将介绍如何定义和调用 void 方法。程序清单 6-2 给出的程序定义了一个名为 printGrade 的方法，然后调用它打印出给定分数的等级。

程序清单 6-2 TestVoidMethod.java

```
1 public class TestVoidMethod {
2     public static void main(String[] args) {
3         System.out.print("The grade is ");
4         printGrade(78.5);
5
6         System.out.print("The grade is ");
7         printGrade(59.5);
8     }
9
10    public static void printGrade(double score) {
11        if (score >= 90.0) {
12            System.out.println('A');
13        }
14        else if (score >= 80.0) {
15            System.out.println('B');
16        }
17        else if (score >= 70.0) {
18            System.out.println('C');
19        }
20        else if (score >= 60.0) {
21            System.out.println('D');
22        }
23        else {
24            System.out.println('F');
25        }
26    }
27 }
```

```
The grade is C
The grade is F
```

printGrade 方法是一个 void 方法，它不返回任何值。对 void 方法的调用必须是一条语句。因此，在 main 方法的第 4 行，printGrade 方法作为一条语句调用，这条语句同其他 Java 语句一样，以分号结束。

为了区分 void 方法和带返回值的方法，我们重新设计 printGrade 方法使之返回一个值。称新方法为 getGrade，返回成绩等级，如程序清单 6-3 所示。

程序清单 6-3 TestReturnGradeMethod.java

```
1 public class TestReturnGradeMethod {
2     public static void main(String[] args) {
3         System.out.print("The grade is " + getGrade(78.5));
4         System.out.print("\nThe grade is " + getGrade(59.5));
5     }
6
7     public static char getGrade(double score) {
8         if (score >= 90.0)
9             return 'A';
10        else if (score >= 80.0)
11            return 'B';
12        else if (score >= 70.0)
13            return 'C';
14        else if (score >= 60.0)
15            return 'D';
16        else
17            return 'F';
18    }
19 }
```

```
The grade is C
The grade is F
```

第 7 ~ 18 行定义的 `getGrade` 方法返回一个基于数字分值的字符等级。调用者在第 3 ~ 4 行调用这个方法。

调用者会在任何出现字符的地方调用 `getGrade` 方法。`printGrade` 方法不返回任何值，因此它必须作为一条语句被调用。

🔑 **注意：**`void` 方法不需要 `return` 语句，但它能用于终止方法并返回到方法的调用者。它的语法是：

```
return;
```

这种用法很少，但是对于改变 `void` 方法中的正常流程控制是很有用的。例如：当分数是无效值时，下列代码就用 `return` 语句结束方法。

```
public static void printGrade(double score) {
    if (score < 0 || score > 100) {
        System.out.println("Invalid score");
        return;
    }

    if (score >= 90.0) {
        System.out.println('A');
    }
    else if (score >= 80.0) {
        System.out.println('B');
    }
    else if (score >= 70.0) {
        System.out.println('C');
    }
    else if (score >= 60.0) {
        System.out.println('D');
    }
    else {
        System.out.println('F');
    }
}
```

🔑 复习题

- 6.1 使用方法的优点有哪些？
- 6.2 如何定义一个方法？如何调用一个方法？
- 6.3 如何使用条件操作符简化程序清单 6-1 中的 `max` 方法？
- 6.4 下面的说法是否正确：对返回值类型为 `void` 的方法的调用总是单独的一条语句，但是对带返回值类型的方法的调用本身不能作为一条语句。
- 6.5 `main` 方法的返回值类型是什么？
- 6.6 如果在一个带返回值的方法中，不写 `return` 语句会发生什么错误？在返回值类型为 `void` 的方法中可以有 `return` 语句吗？下面方法中的 `return` 语句是否会导致语法错误？

```
public static void xMethod(double x, double y) {
    System.out.println(x + y);
    return x + y;
}
```

- 6.7 给出形参、实参和方法签名的定义。
- 6.8 写出下列方法的方法头（而不是方法体）：

- 给定销售额和提成率，计算销售提成。
- 给定月份和年份，打印该月的日历。
- 计算一个数的平方根。
- 测试一个数是否是偶数，如果是，则返回 true。
- 按指定次数打印某条消息。
- 给定贷款额、还款年数和年利率，计算月支付额。
- 对于给定的小写字母，给出相应的大写字母。

6.9 确定并更正下面程序中的错误：

```

1 public class Test {
2     public static method1(int n, m) {
3         n += m;
4         method2(3.4);
5     }
6
7     public static int method2(int n) {
8         if (n > 0) return 1;
9         else if (n == 0) return 0;
10        else if (n < 0) return -1;
11    }
12 }

```

6.10 根据 1.9 节提出的程序设计风格和文档指南，使用花括号的次行风格重新编排下面的程序。

```

public class Test {
    public static double method(double i, double j)
    {
        while (i < j) {
            j--;
        }

        return j;
    }
}

```

6.5 通过传值进行参数传递

 **要点提示：**调用方法的时候是通过传值的方式将实参传给形参的。

方法的强大之处在于它处理参数的能力。可以使用方法 `println` 打印任意字符串，用 `max` 方法求任意两个 `int` 值的最大值。调用方法时，需要提供实参，它们必须与方法签名中所对应的形参次序相同。这称作参数顺序匹配（parameter order association）。例如，下面的方法打印 `message` 信息 `n` 次：

```

public static void nPrintln(String message, int n) {
    for (int i = 0; i < n; i++)
        System.out.println(message);
}

```

可以使用 `nPrintln("Hello",3)` 打印 "Hello"3 遍。语句 `nPrintln("Hello",3)` 把实际的字符串参数 "Hello" 传给参数 `message`，把 3 传给 `n`，然后打印 "Hello"3 次。然而，语句 `nPrintln(3,"Hello")` 是错误的。3 的数据类型不匹配第一个参数 `message` 的数据类型，第二个参数 "Hello" 不匹配第二个参数 `n`。

 **警告：**实参必须与方法签名中定义的参数在次序和数量上匹配，在类型上兼容。类型兼容是指不需要经过显式的类型转换，实参的值就可以传递给形参，例如，将 `int` 型的实参值传递给 `double` 型形参。

当调用带参数的方法时，实参的值传递给形参，这个过程称为按值传递 (pass-by-value)。如果实参是变量而不是直接量，则将该变量的值传递给形参。无论形参在方法中是否改变，该变量都不受影响。如程序清单 6-4 所示，x(1) 的值传给参数 n，用以调用方法 increment (第 5 行)。在该方法中 n 自增 1 (第 10 行)，而 x 的值则不论方法做了什么都保持不变。

程序清单 6-4 Increment.java

```
1 public class Increment {
2     public static void main(String[] args) {
3         int x = 1;
4         System.out.println("Before the call, x is " + x);
5         increment(x);
6         System.out.println("After the call, x is " + x);
7     }
8
9     public static void increment(int n) {
10        n++;
11        System.out.println("n inside the method is " + n);
12    }
13 }
```

```
Before the call, x is 1
n inside the method is 2
After the call, x is 1
```

程序清单 6-5 给出另一个演示按值传递参数效果的程序。程序创建了一个能实现交换两个变量的 swap 方法。调用 swap 方法时传递两个实参。有趣的是，调用方法后，这两个实参并未改变。

程序清单 6-5 TestPassByValue.java

```
1 public class TestPassByValue {
2     /** Main method */
3     public static void main(String[] args) {
4         // Declare and initialize variables
5         int num1 = 1;
6         int num2 = 2;
7
8         System.out.println("Before invoking the swap method, num1 is " +
9             num1 + " and num2 is " + num2);
10
11        // Invoke the swap method to attempt to swap two variables
12        swap(num1, num2);
13
14        System.out.println("After invoking the swap method, num1 is " +
15            num1 + " and num2 is " + num2);
16    }
17
18    /** Swap two variables */
19    public static void swap(int n1, int n2) {
20        System.out.println("\tInside the swap method");
21        System.out.println("\t\tBefore swapping, n1 is " + n1
22            + " and n2 is " + n2);
23
24        // Swap n1 with n2
25        int temp = n1;
26        n1 = n2;
27        n2 = temp;
```

```

28
29     System.out.println("\t\tAfter swapping, n1 is " + n1
30     + " and n2 is " + n2);
31 }
32 }

```

Before invoking the swap method, num1 is 1 and num2 is 2
 Inside the swap method
 Before swapping, n1 is 1 and n2 is 2
 After swapping, n1 is 2 and n2 is 1
 After invoking the swap method, num1 is 1 and num2 is 2

在调用 swap 方法（第 12 行）前，num1 为 1 而 num2 为 2。在调用 swap 方法后，num1 仍为 1，num2 仍为 2。它们的值没有因为调用 swap 方法而交换。如图 6-4 所示，实参 num1 和 num2 的值传递给 n1 和 n2，但是 n1 和 n2 有自己独立于 num1 和 num2 的存储空间。所以，n1 和 n2 的改变不影响 num1 和 num2 的内容。

另一个改变是把 swap 中形参的名称 n1 改为 num1。这样做有什么效果呢？什么也不变，因为形参和实参是否同名是没有任何分别的。形参是方法中具有自己存储空间的变量。局部变量是在调用方法时分配的，当方法返回到调用者后它就消失了。

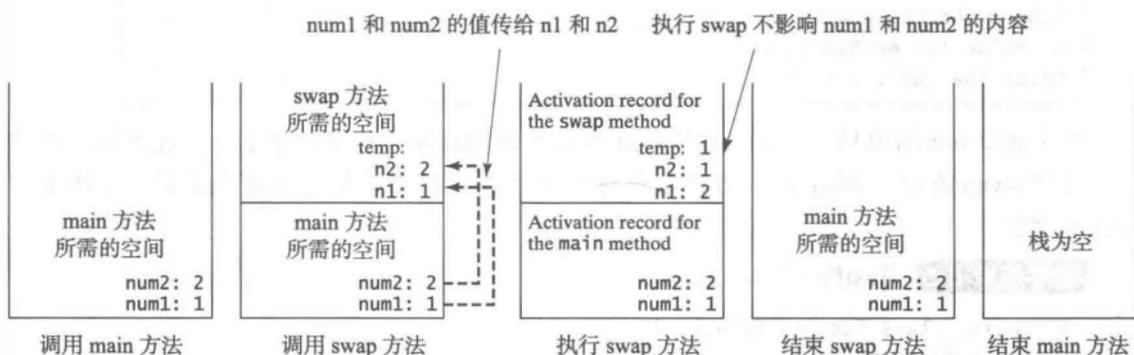


图 6-4 变量的值传递给方法中的形参

注意：为了简便，Java 程序员经常说将实参 x 传给形参 y，实际含义是指将 x 的值传递给 y。

复习题

6.11 实参是如何传递给方法的？实参可以和形参同名吗？

6.12 确定并更正下面程序中的错误：

```

1 public class Test {
2     public static void main(String[] args) {
3         nPrintln(5, "Welcome to Java!");
4     }
5
6     public static void nPrintln(String message, int n) {
7         int n = 1;
8         for (int i = 0; i < n; i++)
9             System.out.println(message);
10    }
11 }

```

6.13 什么是值传递？给出下面程序的运行结果：

```
public class Test {
    public static void main(String[] args) {
        int max = 0;
        max(1, 2, max);
        System.out.println(max);
    }

    public static void max(
        int value1, int value2, int max) {
        if (value1 > value2)
            max = value1;
        else
            max = value2;
    }
}
```

a)

```
public class Test {
    public static void main(String[] args) {
        int i = 1;
        while (i <= 6) {
            method1(i, 2);
            i++;
        }
    }

    public static void method1(
        int i, int num) {
        for (int j = 1; j <= i; j++) {
            System.out.print(num + " ");
            num *= 2;
        }

        System.out.println();
    }
}
```

b)

```
public class Test {
    public static void main(String[] args) {
        // Initialize times
        int times = 3;
        System.out.println("Before the call,"
            + " variable times is " + times);

        // Invoke nPrintln and display times
        nPrintln("Welcome to Java!", times);
        System.out.println("After the call,"
            + " variable times is " + times);
    }

    // Print the message n times
    public static void nPrintln(
        String message, int n) {
        while (n > 0) {
            System.out.println("n = " + n);
            System.out.println(message);
            n--;
        }
    }
}
```

c)

```
public class Test {
    public static void main(String[] args) {
        int i = 0;
        while (i <= 4) {
            method1(i);
            i++;

            System.out.println("i is " + i);
        }

        public static void method1(int i) {
            do {
                if (i % 3 != 0)
                    System.out.print(i + " ");
                i--;
            }
            while (i >= 1);

            System.out.println();
        }
    }
}
```

d)

6.14 在 6.13 的 a 中，分别给出调用 max 方法之前、刚进入 max 方法、max 方法刚要返回之前以及 max 方法返回之后堆栈的内容。

6.6 模块化代码

要点提示：模块化使得代码易于维护和调试，并且使得代码可以被重用。

使用方法可以减少冗余的代码，提高代码的复用性。方法也可以用来模块化代码，以提高程序的质量。

程序清单 5-9 给出的程序提示用户输入两个整数，然后显示它们的最大公约数。可以使用方法改写这个程序，如程序清单 6-6 所示。

程序清单 6-6 GreatestCommonDivisorMethod.java

```
1 import java.util.Scanner;
2
3 public class GreatestCommonDivisorMethod {
```

```

4  /** Main method */
5  public static void main(String[] args) {
6      // Create a Scanner
7      Scanner input = new Scanner(System.in);
8
9      // Prompt the user to enter two integers
10     System.out.print("Enter first integer: ");
11     int n1 = input.nextInt();
12     System.out.print("Enter second integer: ");
13     int n2 = input.nextInt();
14
15     System.out.println("The greatest common divisor for " + n1 +
16         " and " + n2 + " is " + gcd(n1, n2));
17 }
18
19 /** Return the gcd of two integers */
20 public static int gcd(int n1, int n2) {
21     int gcd = 1; // Initial gcd is 1
22     int k = 2; // Possible gcd
23
24     while (k <= n1 && k <= n2) {
25         if (n1 % k == 0 && n2 % k == 0)
26             gcd = k; // Update gcd
27         k++;
28     }
29
30     return gcd; // Return gcd
31 }
32 }

```

```

Enter first integer: 45 [Enter]
Enter second integer: 75 [Enter]
The greatest common divisor for 45 and 75 is 15

```

通过将求最大公约数的代码封装在一个方法中，这个程序就具备了以下几个优点：

1) 它将计算最大公约数的问题和 main 方法中的其他代码分隔开，这样做会使逻辑更加清晰而且程序的可读性更强。

2) 计算最大公约数的错误就限定在 gcd 方法中，这样就缩小了调试的范围。

3) 现在，其他程序就可以重复使用 gcd 方法。

程序清单 6-7 应用了代码模块化的概念来对程序清单 5-15 进行改进。

程序清单 6-7 PrimeNumberMethod.java

```

1  public class PrimeNumberMethod {
2      public static void main(String[] args) {
3          System.out.println("The first 50 prime numbers are \n");
4          printPrimeNumbers(50);
5      }
6
7      public static void printPrimeNumbers(int numberOfPrimes) {
8          final int NUMBER_OF_PRIMES_PER_LINE = 10; // Display 10 per line
9          int count = 0; // Count the number of prime numbers
10         int number = 2; // A number to be tested for primeness
11
12         // Repeatedly find prime numbers
13         while (count < numberOfPrimes) {
14             // Print the prime number and increase the count
15             if (isPrime(number)) {
16                 count++; // Increase the count
17

```

```

18     if (count % NUMBER_OF_PRIMES_PER_LINE == 0) {
19         // Print the number and advance to the new line
20         System.out.printf("%-5s\n", number);
21     }
22     else
23         System.out.printf("%-5s", number);
24 }
25
26 // Check whether the next number is prime
27 number++;
28 }
29 }
30
31 /** Check whether number is prime */
32 public static boolean isPrime(int number) {           isPrime method
33     for (int divisor = 2; divisor <= number / 2; divisor++) {
34         if (number % divisor == 0) { // If true, number is not prime
35             return false; // Number is not a prime
36         }
37     }
38
39     return true; // Number is prime
40 }
41 }

```

The first 50 prime numbers are

2	3	5	7	11	13	17	19	23	29
31	37	41	43	47	53	59	61	67	71
73	79	83	89	97	101	103	107	109	113
127	131	137	139	149	151	157	163	167	173
179	181	191	193	197	199	211	223	227	229

将一个大问题分成两个子问题：确定一个数字是否是素数以及打印素数。这样，新的程序会更易读，也更易于调试。而且，其他程序也可以重用方法 `printPrimeNumbers` 和 `isPrime`。

6.7 示例学习：将十六进制数转换为十进制数

🔑 要点提示：本节给出一个程序，将十六进制数转换为十进制数。

程序清单 5-11 给出了将十进制数转换为十六进制数的程序。那么，如何将十六进制数转换为十进制数呢？

假定一个十六进制数是 $h_n h_{n-1} h_{n-2} \cdots h_2 h_1 h_0$ ，那么等价的十进制数的值为：

$$h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + \cdots + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0$$

例如，十六进制数 AB8C 是：

$$10 \times 16^3 + 11 \times 16^2 + 8 \times 16^1 + 12 \times 16^0 = 43\,916$$

程序将提示用户将一个十六进制数作为字符串输入，然后使用下面的方法将该数转换为一个十进制数：

```
public static int hexToDecimal(String hex)
```

将每个十六进制的字符转换为一个十进制数的穷举方法就是将第 i 个位置的十六进制数乘以 16^i ，然后将所有这些项相加，就得到和这个十六进制数等价的十进制数。

注意到，

$$h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + \cdots + h_1 \times 16^1 + h_0 \times 16^0$$

$$=(\dots((h_n \times 16 + h_{n-1}) \times 16 + h_{n-2}) \times 16 + \dots + h_1) \times 16 + h_0$$

这个发现，称为霍纳算法，可以导出下面这个将十六进制字符串转换为十进制数的高效算法：

```
int decimalValue = 0;
for (int i = 0; i < hex.length(); i++) {
    char hexChar = hex.charAt(i);
    decimalValue = decimalValue * 16 + hexCharToDecimal(hexChar);
}
```

下面是将算法应用在十六进制数 AB8C 时各个变量的数值变化情况：

	i	十六进制数	十六进制数转换为十进制数	十进制数
循环开始之前				0
第一次迭代之后	0	A	10	10
第二次迭代之后	1	B	11	$10 \times 16 + 11$
第三次迭代之后	2	8	8	$(10 \times 16 + 11) \times 16 + 8$
第四次迭代之后	3	C	12	$((10 \times 16 + 11) \times 16 + 8) \times 16 + 12$

程序清单 6-8 给出完整的程序。

程序清单 6-8 Hex2Dec.java

```
1 import java.util.Scanner;
2
3 public class Hex2Dec {
4     /** Main method */
5     public static void main(String[] args) {
6         // Create a Scanner
7         Scanner input = new Scanner(System.in);
8
9         // Prompt the user to enter a string
10        System.out.print("Enter a hex number: ");
11        String hex = input.nextLine();
12
13        System.out.println("The decimal value for hex number "
14            + hex + " is " + hexToDecimal(hex.toUpperCase()));
15    }
16
17    public static int hexToDecimal(String hex) {
18        int decimalValue = 0;
19        for (int i = 0; i < hex.length(); i++) {
20            char hexChar = hex.charAt(i);
21            decimalValue = decimalValue * 16 + hexCharToDecimal(hexChar);
22        }
23
24        return decimalValue;
25    }
26
27    public static int hexCharToDecimal(char ch) {
28        if (ch >= 'A' && ch <= 'F')
29            return 10 + ch - 'A';
30        else // ch is '0', '1', ..., or '9'
31            return ch - '0';
32    }
33 }
```

Enter a hex number: AB8C
The decimal value for hex number AB8C is 43916

```
Enter a hex number: af71 
The decimal value for hex number af71 is 44913
```

程序从控制台读取一个字符串（第 11 行），然后调用 `hexToDecimal` 方法，将一个十六进制字符串转换为十进制数（第 14 行）。字符可以是小写的也可以是大写的。在调用 `hexToDecimal` 方法之前将它们都转换成大写。

在第 17 ~ 25 行定义的 `hexToDecimal` 方法返回一个整型值。这个字符串的长度是由第 19 行调用的 `hex.length()` 方法确定的。

在第 27 ~ 32 行定义的方法 `hexCharToDecimal` 返回一个十六进制字符的十进制数值。这个字符可以是小写的，也可以是大写的。回忆一下，两个字符的减法就是对它们的 Unicode 码做减法。例如，`'5'-'0'` 是 5。

6.8 重载方法

 **要点提示：**重载方法使得你可以使用同样的名字来定义不同方法，只要它们的签名是不同的。

前面用到的 `max` 方法只能用于 `int` 型数据类型。但是，如果需要决定两个浮点数中哪个较大，该怎么办呢？解决办法是创建另一个方法名相同但参数不同的方法，代码如下所示：

```
public static double max(double num1, double num2) {
    if (num1 > num2)
        return num1;
    else
        return num2;
}
```

如果调用带 `int` 型参数的 `max` 方法，就将调用需要 `int` 型参数的 `max` 方法；如果调用带 `double` 型参数的 `max` 方法，就将调用需要 `double` 型参数的 `max` 方法。这称为方法重载（method overloading）。也就是说，在一个类中有两个方法，它们具有相同的名字，但有不同的参数列表。Java 编译器根据方法签名决定使用哪个方法。

程序清单 6-9 是一个创建了三个方法的程序。第一个方法求最大整数，第二个方法求最大双精度数，而第三个方法求三个双精度数中的最大值。这三个方法都被命名为 `max`。

程序清单 6-9 TestMethodOverloading.java

```
1 public class TestMethodOverloading {
2     /** Main method */
3     public static void main(String[] args) {
4         // Invoke the max method with int parameters
5         System.out.println("The maximum of 3 and 4 is "
6             + max(3, 4));
7
8         // Invoke the max method with the double parameters
9         System.out.println("The maximum of 3.0 and 5.4 is "
10            + max(3.0, 5.4));
11
12        // Invoke the max method with three double parameters
13        System.out.println("The maximum of 3.0, 5.4, and 10.14 is "
14            + max(3.0, 5.4, 10.14));
15    }
16
17    /** Return the max of two int values */
18    public static int max(int num1, int num2) {
```

```

19     if (num1 > num2)
20         return num1;
21     else
22         return num2;
23 }
24
25 /** Find the max of two double values */
26 public static double max(double num1, double num2) {
27     if (num1 > num2)
28         return num1;
29     else
30         return num2;
31 }
32
33 /** Return the max of three double values */
34 public static double max(double num1, double num2, double num3) {
35     return max(max(num1, num2), num3);
36 }
37 }

```

```

The maximum of 3 and 4 is 4
The maximum of 3.0 and 5.4 is 5.4
The maximum of 3.0, 5.4, and 10.14 is 10.14

```

当调用 `max(3,4)` (第6行) 时, 调用的是求两个整数中较大值的 `max` 方法。当调用 `max(3.0,5.4)` (第10行) 时, 调用的是求两个双精度数中较大值的 `max` 方法。当调用 `max(3.0,5.4,10.14)` (第14行) 时, 调用的是求三个双精度数中最大数的 `max` 方法。

可以调用像 `max(2,2.5)` 这样带一个 `int` 值和一个 `double` 值的 `max` 方法吗? 如果能, 该调用哪一个 `max` 方法呢? 第一个问题的答案是肯定的。第二个问题的答案是调用求两个 `double` 数中较大值的方法。实参值 2 被自动转换为 `double` 值, 然后传递给这个方法。

你可能会感到奇怪, 为什么调用 `max(3,4)` 时不会使用 `max(double,double)` 呢? 其实, `max(double,double)` 和 `max(int,int)` 与 `max(3,4)` 都是可能的匹配。调用方法时, Java 编译器寻找最精确匹配的方法。因为方法 `max(int,int)` 比 `max(double,double)` 更精确, 所以调用 `max(3,4)` 时使用的是 `max(int,int)`。

提示: 重载方法可以使得程序更加清楚, 以及更加具有可读性。执行同样功能但是具有不同参数类型的方法应该使用同样的名字。

注意: 被重载的方法必须具有不同的参数列表。不能基于不同修饰符或返回值类型来重载方法。

注意: 有时调用一个方法时, 会有两个或更多可能的匹配, 但是, 编译器无法判断哪个是最精确的匹配。这称为歧义调用 (ambiguous invocation)。歧义调用会产生一个编译错误。考虑如下代码:

```

public class AmbiguousOverloading {
    public static void main(String[] args) {
        System.out.println(max(1, 2));
    }

    public static double max(int num1, double num2) {
        if (num1 > num2)
            return num1;
        else
            return num2;
    }
}

```

```
public static double max(double num1, int num2) {
    if (num1 > num2)
        return num1;
    else
        return num2;
}
}
```

`max(int,double)` 和 `max(double,int)` 都有可能与 `max(1,2)` 匹配。由于两个方法谁也不比谁更精确，所以这个调用是有歧义的，它会导致一个编译错误。

复习题

- 6.15 什么是方法重载？可以定义两个同名但参数类型不同的方法吗？可以在一个类中定义两个名称和参数列表相同，但返回值类型不同或修饰符不同的方法吗？
- 6.16 下面的程序有什么错误？

```
public class Test {
    public static void method(int x) {
    }

    public static int method(int y) {
        return y;
    }
}
```

- 6.17 给定两个方法定义：

```
public static double m(double x, double y)
public static double m(int x, double y)
```

给出对于下面的语句，两个方法中的哪个被调用？

- `double z = m(4, 5);`
- `double z = m(4, 5.4);`
- `double z = m(4.5, 5.4);`

6.9 变量的作用域

要点提示：变量的作用域 (scope of a variable) 是指变量可以在程序中引用的范围。

第 2.5 节介绍了变量的作用域，本节更加详细地讨论变量的范围。在方法中定义的变量称为局部变量 (local variable)。局部变量的作用域从声明变量的地方开始，直到包含该变量的块结束为止。局部变量都必须在使用之前进行声明和赋值。

参数实际上就是一个局部变量。一个方法的参数的作用域涵盖整个方法。在 `for` 循环头中初始动作部分声明的变量，其作用域是整个 `for` 循环。但是在 `for` 循环体内声明的变量，其作用域只限于循环体内，是从它的声明处开始，到包含该变量的块结束为止，如图 6-5 所示。

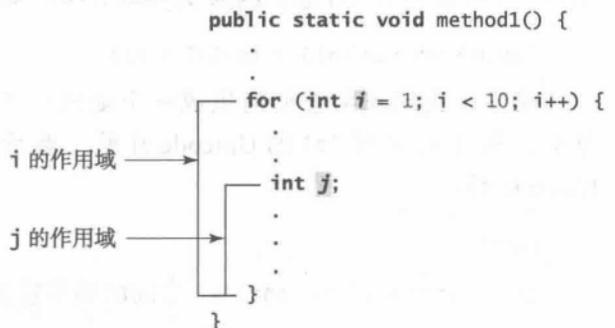


图 6-5 在 `for` 循环头中初始动作部分声明的变量，其作用域是整个 `for` 循环

可以在一个方法中的不同块里声明同名的局部变量，但是，不能在嵌套块中或同一块中两次声明同一个局部变量，如图 6-6 所示。

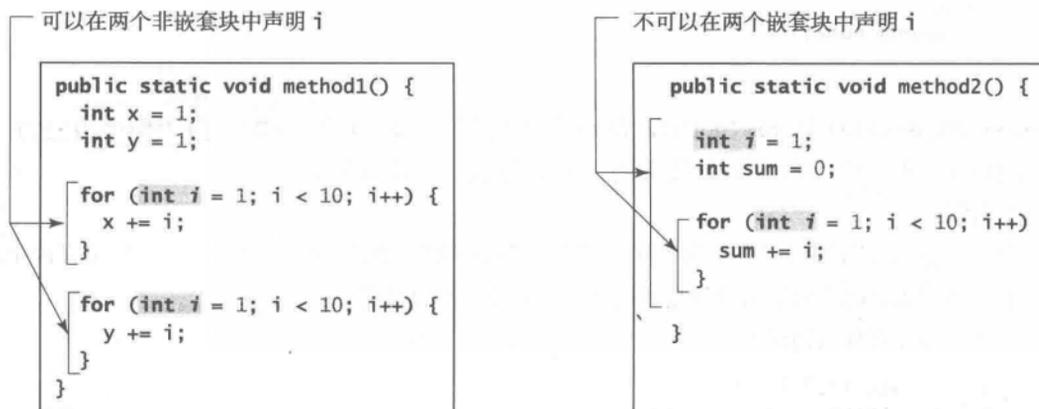


图 6-6 一个变量可以在非嵌套的块中多次声明，而在嵌套块中只能声明一次

警告：不要在块内声明一个变量然后企图在块外使用它。下面是一个常见错误的例子：

```
for (int i = 0; i < 10; i++) {
}
```

```
System.out.println(i);
```

因为变量 *i* 没有在 `for` 循环外定义，所以最后一条语句就会产生一个语法错误。

复习题

- 6.18 什么是局部变量？
- 6.19 什么是局部变量的作用域？

6.10 示例学习：生成随机字符

要点提示：字符使用整数来编码。产生一个随机字符就是产生一个随机整数。

计算机程序处理数值数据和字符。前面已经看到了许多涉及数值数据的例子。了解字符和如何处理字符也是很重要的。本节给出生成随机字符的例子。

正如 4.3 节所介绍的，每个字符都有一个唯一的在十六进制数 0 到 FFFF（即十进制的 65535）之间的 Unicode。生成一个随机字符就是使用下面的表达式，生成从 0 到 65535 之间的一个随机整数（注意：因为 $0 \leq \text{Math.random()} < 1.0$ ，必须给 65535 上加 1）：

```
(int)(Math.random() * (65535 + 1))
```

现在让我们来考虑如何生成一个随机小写字母。小写字母的 Unicode 是一串连续的整数，从小写字母 'a' 的 Unicode 开始，然后是 'b'、'c'、… 和 'z' 的 Unicode。'a' 的 Unicode 是：

```
(int)'a'
```

所以，`(int)'a'` 到 `(int)'z'` 之间的随机整数是：

```
(int)((int)'a' + Math.random() * ((int)'z' - (int)'a' + 1))
```

正如 4.3.3 节中所讨论的，所有的数字操作符都可以应用到 `char` 操作数上。如果另一个

操作数是数字或字符，那么 char 型操作数就会被转换成数字。这样，前面的表达式就可以简化为如下所示：

```
'a' + Math.random() * ('z' - 'a' + 1)
```

这样，随机的小写字母是：

```
(char)('a' + Math.random() * ('z' - 'a' + 1))
```

由此，可以生成任意两个字符 ch1 和 ch2 之间的随机字符，其中 ch1<ch2，如下所示：

```
(char)(ch1 + Math.random() * (ch2 - ch1 + 1))
```

这是一个简单但却很有用的发现。在程序清单 6-10 中创建一个名为 RandomCharacter 的类，它有五个重载的方法，随机获取某种特定类型的字符。可以在以后的项目中使用这些方法。

程序清单 6-10 RandomCharacter.java

```
1 public class RandomCharacter {
2     /** Generate a random character between ch1 and ch2 */
3     public static char getRandomCharacter(char ch1, char ch2) {
4         return (char)(ch1 + Math.random() * (ch2 - ch1 + 1));
5     }
6
7     /** Generate a random lowercase letter */
8     public static char getRandomLowerCaseLetter() {
9         return getRandomCharacter('a', 'z');
10    }
11
12    /** Generate a random uppercase letter */
13    public static char getRandomUpperCaseLetter() {
14        return getRandomCharacter('A', 'Z');
15    }
16
17    /** Generate a random digit character */
18    public static char getRandomDigitCharacter() {
19        return getRandomCharacter('0', '9');
20    }
21
22    /** Generate a random character */
23    public static char getRandomCharacter() {
24        return getRandomCharacter('\u0000', '\uFFFF');
25    }
26 }
```

程序清单 6-11 给出一个测试程序，显示 175 个随机的小写字母。

程序清单 6-11 TestRandomCharacter.java

```
1 public class TestRandomCharacter {
2     /** Main method */
3     public static void main(String[] args) {
4         final int NUMBER_OF_CHARS = 175;
5         final int CHARS_PER_LINE = 25;
6
7         // Print random characters between 'a' and 'z', 25 chars per line
8         for (int i = 0; i < NUMBER_OF_CHARS; i++) {
9             char ch = RandomCharacter.getRandomLowerCaseLetter();
10            if ((i + 1) % CHARS_PER_LINE == 0)
11                System.out.println(ch);
12            else
```

```

13     System.out.print(ch);
14     }
15 }
16 }

```

```

gmjsohezfkg tazqgmswfc lrao
pnrunulnwmaztlfjedmpchcif
lalqdgivxkxpbzulrmqmbhikr
lbnrjlsopfxahssqhwuuljvbe
xbhdotzhpehbqmuwsfktwsoli
cbuwkzgxpmtzihgatdslvbwbz
bfesoklwbhnooygiigzdxuqni

```

第9行调用定义在 `RandomCharacter` 类中的方法 `getRandomLowerCaseLetter()`。注意，虽然方法 `getRandomLowerCaseLetter()` 没有任何参数，但是在定义和调用这类方法时仍然需要使用括号。

6.11 方法抽象和逐步求精

要点提示： 开发软件的关键在于应用抽象的概念。

你将从本书中学习多种层次的抽象。方法抽象 (method abstraction) 是通过将方法的使用和它的实现分离来实现的。用户在不知道方法是如何实现的情况下，就可以使用方法。方法的实现细节封装在方法内，对使用该方法的用户来说是隐藏的。这就称为信息隐藏 (information hiding) 或封装 (encapsulation)。如果决定改变方法的实现，但只要不改变方法签名，用户的程序就不受影响。方法的实现对用户隐藏在“黑盒子”中，如图 6-7 所示。

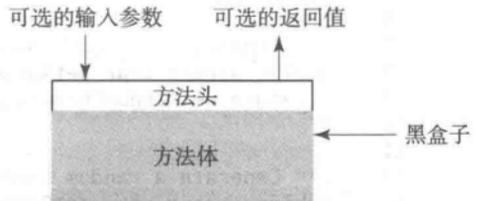


图 6-7 方法体可以看作是一个包括该方法实现细节的黑盒子

前面已经使用过方法 `System.out.print` 来显示一个字符串，用 `max` 方法求最大数。也知道了怎样在程序中编写代码来调用这些方法。但是作为这些方法的使用者，你并不需要知道它们是怎样实现的。

方法抽象的概念可以应用于程序的开发过程中。当编写一个大程序时，可以使用“分治” (divid-and-conquer) 策略，也称为逐步求精 (stepwise refinement)，将大问题分解成子问题。子问题又分解成更小、更容易处理的问题。

假设要编写一个程序，显示给定年月的日历。程序提示用户输入年份和月份，然后显示该月的整个日历，如下面的运行示例所示。

```

Enter full year (e.g., 2012): 2012 ~Enter
Enter month as number between 1 and 12: 3 ~Enter

      March 2012
-----
Sun Mon Tue Wed Thu Fri Sat
                1  2  3
 4   5  6  7  8  9 10
11  12 13 14 15 16 17
18  19 20 21 22 23 24
25  26 27 28 29 30

```

让我们用这个例子演示分治法。

6.11.1 自顶向下的设计

如何开始编写这样一个程序呢？你会立即开始编写代码吗？程序员新手常常一开始就想解决每一个细节。尽管细节对最终程序很重要，但在前期过多关注细节会阻碍解决问题的进程。为使解决问题的流程尽可能地流畅，本例先用方法抽象把细节与设计分离，只在后面才实现这些细节。

对本例来说，先把问题拆分成两个子问题：读取用户输入和打印该月的日历。在这一阶段，应该考虑还能分解成什么子问题，而不是用什么方法来读取输入和打印整个日历。可以画一个结构图，这有助于看清楚问题的分解过程（参见图 6-8a）。

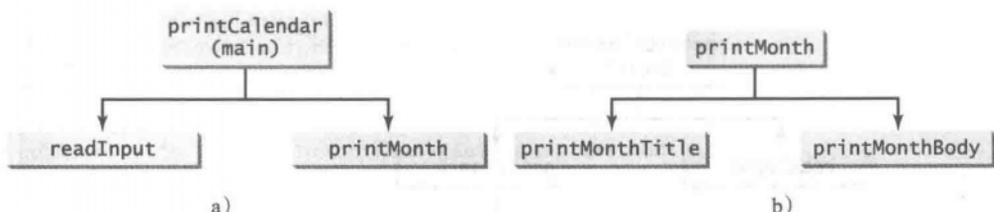
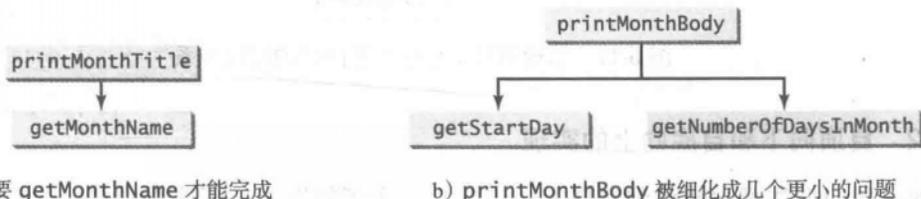


图 6-8 结构图显示将打印日历 `printCalendar` 问题分解成两个子问题——读取输入 `readInput` 和打印日历 `printMonth`，而将 `printMonth` 分解成两个更小的问题——打印日历头 `printMonthTitle` 和打印日历体 `printMonthBody`

你可以使用 `Scanner` 来读取年和月份的输入。打印给定月份的日历问题可以分解成两个子问题：打印日历的标题和日历的主体，如图 6-8b 所示。月历的标题由三行组成：年月、虚线、每周七天的星期名称。需要通过表示月份的数字（例如：1）来确定该月的全称（例如：January）。这个步骤是由 `getMonthName` 来完成的（参见图 6-9a）。



a) 需要 `getMonthName` 才能完成 `printMonthTitle`

b) `printMonthBody` 被细化成几个更小的问题

图 6-9

为了打印日历的主体，需要知道这个月的第一天是星期几（`getStartDay`），以及该月有多少天（`getNumberOfDaysInMonth`），如图 6-9b 所示。例如：2013 年 12 月有 31 天，2013 年 12 月 1 号是星期天。

怎样才能知道一个月的第一天是星期几呢？有几种方法可以求得。这里，我们采用下面的方法。假设知道 1800 年 1 月 1 日是星期三（`START_DAY_FOR_JAN_1_1800=3`），然后计算 1800 年 1 月 1 日和日历月份的第一天之间相差的总天数（`totalNumberOfDays`）。因为每个星期有 7 天，所以日历月份第一天的星期就是 $(totalNumberOfDays + START_DAY_FOR_JAN_1_1800) \% 7$ 。这样 `getStartDay` 问题就可以进一步细化为 `getTotalNumberOfDays`，如图 6-10a 所示。

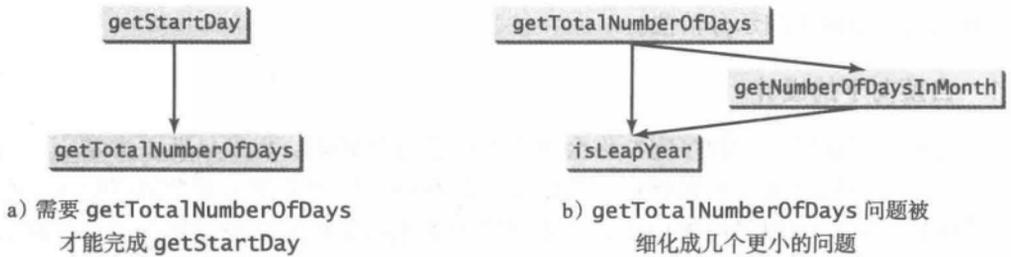


图 6-10

要计算总天数，需要知道该年是否是闰年以及每个月的天数。所以，getTotalNumberOfDays 可以进一步细化成两个子问题：isLeapYear 和 getNumberOfDaysInMonth，如图 6-10b 所示。完整的结构图如图 6-11 所示。

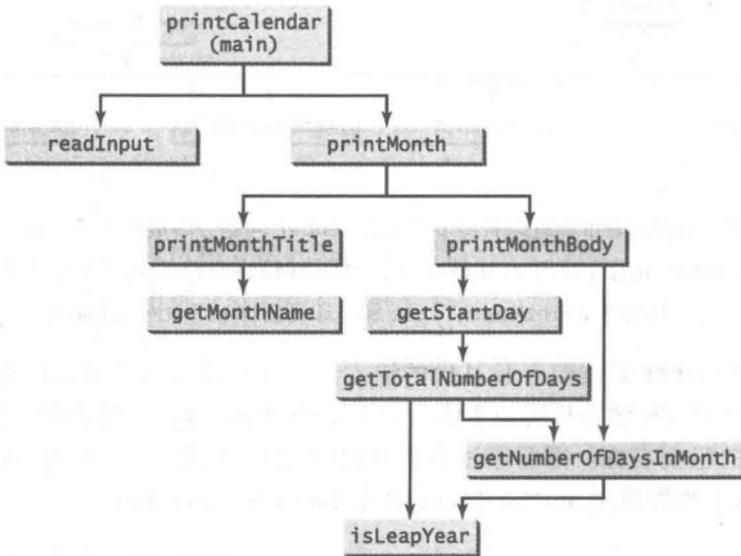


图 6-11 结构图显示程序中子问题的层次关系

6.11.2 自顶向下和自底向上的实现

现在我们把注意力转移到实现上。通常，一个子问题对应于实现中的一个方法，即使某些子问题太简单，以至于都不需要方法来实现。需要决定哪些模块要用方法实现，而哪些模块要与其他方法结合完成。这种决策应该基于所做的选择是否使整个程序更易读而做出的。在本例中，子问题 readInput 只要在 main 方法中即可实现。

可以采用“自顶向下”或“自底向上”的办法。“自顶向下”方法是自上而下，每次实现结构图中的一个方法。等待实现的方法可以用待完善方法代替。待完善方法 (stub) 是方法的一个简单但不完整的版本。使用待完善方法可以快速地构建程序的框架。首先实现 main 方法，然后使用 printMonth 方法的 stub。例如，让 printMonth 中的待完善部分显示年份和月份，那么程序就以下面的形式开始：

```
public class PrintCalendar {
    /** Main method */
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
```

```
// Prompt the user to enter year
System.out.print("Enter full year (e.g., 2012): ");
int year = input.nextInt();

// Prompt the user to enter month
System.out.print("Enter month as a number between 1 and 12: ");
int month = input.nextInt();

// Print calendar for the month of the year
printMonth(year, month);
}

/** A stub for printMonth may look like this */
public static void printMonth(int year, int month){
    System.out.print(month + " " + year);
}

/** A stub for printMonthTitle may look like this */
public static void printMonthTitle(int year, int month){
}

/** A stub for getMonthBody may look like this */
public static void printMonthBody(int year, int month){
}

/** A stub for getMonthName may look like this */
public static String getMonthName(int month) {
    return "January"; // A dummy value
}

/** A stub for getStartDay may look like this */
public static int getStartDay(int year, int month) {
    return 1; // A dummy value
}

/** A stub for getTotalNumberOfDays may look like this */
public static int getTotalNumberOfDays(int year, int month) {
    return 10000; // A dummy value
}

/** A stub for getNumberOfDaysInMonth may look like this */
public static int getNumberOfDaysInMonth(int year, int month) {
    return 31; // A dummy value
}

/** A stub for isLeapYear may look like this */
public static boolean isLeapYear(int year) {
    return true; // A dummy value
}
}
```

编译和测试这个程序，然后修改所有的错误。现在，可以实现 `printMonth` 方法。对 `printMonth` 中调用的方法，可以继续使用待完善方法。

自底向上方法是从下向上每次实现结构图中的一个方法，对每个实现的方法都写一个测试程序进行测试。自顶向下和自底向上都是不错的方法：它们都是逐步地实现方法，这有助于分离程序设计错误，使调试变得容易。有时，这两种方法可以一起使用。

6.11.3 实现细节

从 3.11 节我们知道，方法 `isLeapYear(int year)` 可以使用下列代码实现：

```
return year % 400 == 0 || (year % 4 == 0 && year % 100 != 0);
```

使用下面的事实实现 `getTotalNumberOfDaysInMonth(int year, int month)` 方法:

- 1) 一月、三月、五月、七月、八月、十月和十二月都有 31 天。
- 2) 四月、六月、九月和十一月都有 30 天。
- 3) 二月通常有 28 天，但是在闰年有 29 天。因此，一年通常有 365 天，闰年有 366 天。

要实现 `getTotalNumberOfDays(int year, int month)` 方法，需要计算 1800 年 1 月 1 日和日历月份的第一天之间的总天数 (`totalNumberOfDays`)。可以求出 1800 年到该日历年的总天数，然后求出该日历年中在日历月份之前的总天数。这两个总天数相加就是 `totalNumberOfDays`。

要打印日历体，首先在第一天之前填充一些空格，然后为每个星期打印一条线。完整的程序见程序清单 6-12。

程序清单 6-12 PrintCalendar.java

```

1  import java.util.Scanner;
2
3  public class PrintCalendar {
4      /** Main method */
5      public static void main(String[] args) {
6          Scanner input = new Scanner(System.in);
7
8          // Prompt the user to enter year
9          System.out.print("Enter full year (e.g., 2012): ");
10         int year = input.nextInt();
11
12         // Prompt the user to enter month
13         System.out.print("Enter month as a number between 1 and 12: ");
14         int month = input.nextInt();
15
16         // Print calendar for the month of the year
17         printMonth(year, month);
18     }
19
20     /** Print the calendar for a month in a year */
21     public static void printMonth(int year, int month) {
22         // Print the headings of the calendar
23         printMonthTitle(year, month);
24
25         // Print the body of the calendar
26         printMonthBody(year, month);
27     }
28
29     /** Print the month title, e.g., March 2012 */
30     public static void printMonthTitle(int year, int month) {
31         System.out.println("      " + getMonthName(month)
32             + " " + year);
33         System.out.println("-----");
34         System.out.println(" Sun Mon Tue Wed Thu Fri Sat");
35     }
36
37     /** Get the English name for the month */
38     public static String getMonthName(int month) {
39         String monthName = "";
40         switch (month) {
41             case 1: monthName = "January"; break;
42             case 2: monthName = "February"; break;
43             case 3: monthName = "March"; break;
44             case 4: monthName = "April"; break;

```

```
45     case 5: monthName = "May"; break;
46     case 6: monthName = "June"; break;
47     case 7: monthName = "July"; break;
48     case 8: monthName = "August"; break;
49     case 9: monthName = "September"; break;
50     case 10: monthName = "October"; break;
51     case 11: monthName = "November"; break;
52     case 12: monthName = "December";
53 }
54
55 return monthName;
56 }
57
58 /** Print month body */
59 public static void printMonthBody(int year, int month) {
60     // Get start day of the week for the first date in the month
61     int startDay = getStartDay(year, month)
62
63     // Get number of days in the month
64     int numberOfDaysInMonth = getNumberOfDaysInMonth(year, month);
65
66     // Pad space before the first day of the month
67     int i = 0;
68     for (i = 0; i < startDay; i++)
69         System.out.print("  ");
70
71     for (i = 1; i <= numberOfDaysInMonth; i++) {
72         System.out.printf("%4d", i);
73
74         if ((i + startDay) % 7 == 0)
75             System.out.println();
76     }
77
78     System.out.println();
79 }
80
81 /** Get the start day of month/1/year */
82 public static int getStartDay(int year, int month) {
83     final int START_DAY_FOR_JAN_1_1800 = 3;
84     // Get total number of days from 1/1/1800 to month/1/year
85     int totalNumberOfDays = getTotalNumberOfDays(year, month);
86
87     // Return the start day for month/1/year
88     return (totalNumberOfDays + START_DAY_FOR_JAN_1_1800) % 7;
89 }
90
91 /** Get the total number of days since January 1, 1800 */
92 public static int getTotalNumberOfDays(int year, int month) {
93     int total = 0;
94
95     // Get the total days from 1800 to 1/1/year
96     for (int i = 1800; i < year; i++)
97         if (isLeapYear(i))
98             total = total + 366;
99         else
100            total = total + 365;
101
102     // Add days from Jan to the month prior to the calendar month
103     for (int i = 1; i < month; i++)
104         total = total + getNumberOfDaysInMonth(year, i);
105
106     return total;
107 }
108 }
```

```

109  /** Get the number of days in a month */
110  public static int getNumberOfDaysInMonth(int year, int month) {
111      if (month == 1 || month == 3 || month == 5 || month == 7 ||
112          month == 8 || month == 10 || month == 12)
113          return 31;
114
115      if (month == 4 || month == 6 || month == 9 || month == 11)
116          return 30;
117
118      if (month == 2) return isLeapYear(year) ? 29 : 28;
119
120      return 0; // If month is incorrect
121  }
122
123  /** Determine if it is a leap year */
124  public static boolean isLeapYear(int year) {
125      return year % 400 == 0 || (year % 4 == 0 && year % 100 != 0);
126  }
127  }

```

该程序没有检测用户输入的有效性。例如：如果用户输入的月份不在 1 到 12 之间，或者年份在 1800 年之前，那么程序就会显示出错误的日历。为避免出现这样的错误，可以添加一个 if 语句在打印日历前检查输入。

该程序可以打印一个月的日历，还可以很容易地修改为打印整年的日历。尽管它现在只能处理 1800 年 1 月以后的月份，但是可以稍作修改，便能够打印 1800 年之前的月份。

6.11.4 逐步求精的优势

逐步求精将一个大问题分解为小的易于处理的子问题。每个子问题可以使用一个方法来实现。这种方法使得问题更加易于编写、重用、调试、修改和维护。

1. 更简单的程序

打印日历的程序比较长。逐步求精方法将其分解为较小的方法，而不是在一个方法中写很长的语句序列。这样简化了程序，使得整个程序易于阅读和理解。

2. 重用方法

逐步求精提高了一个程序中的方法重用。isLeapYear 方法只定义了一次，从 getTotalNumberOfDays 和 getNumberOfDaysInMonth 方法中都进行了调用。这减少了冗余的代码。

3. 易于开发、调试和测试

因为每个子问题在一个方法中解决，而一个方法可以分别的开发、调试以及测试。这隔离了错误，使得开发、调试和测试更加容易。

编写大型程序时，可以使用自顶向下或自底向上的方法。不要一次性地编写整个程序。使用这些方法似乎浪费了更多的开发时间（因为要反复编译和运行程序），但实际上，它会更节省时间并使调试更容易。

4. 更方便团队合作

当一个大问题分解为许多子问题，各个子问题可以分配给不同的编程人员。这更加易于编程人员进行团队工作。

关键术语

actual parameter (实际参数)
ambiguous invocation (歧义调用)

argument (实参)
divide and conquer (分治)

formal parameter (ie. parameter) (形式参数即形参)	modifier (修饰符)
information hiding (信息隐藏)	parameter (参数)
method (方法)	pass-by-value (按值传递)
method abstraction (方法抽象)	scope of variable (变量的作用域)
method overloading (方法重载)	stepwise refinement (逐步求精)
method signature (方法签名)	stub (待完善方法)

本章小结

1. 程序模块化和可重用性是软件工程的中心目标之一。Java 提供了很多有助于完成这一目标的有效结构。方法就是一个这样的结构。
2. 方法头指定方法的修饰符、返回值类型、方法名和参数。本章所有的方法都使用静态修饰符 `static`。
3. 方法可以返回一个值。返回值类型 `returnValueType` 是方法要返回的值的的数据类型。如果方法不返回值，则返回值类型就是关键字 `void`。
4. 参数列表是指方法中参数的类型、次序和数量。方法名和参数列表一起构成方法签名 (`method signature`)。参数是可选的，也就是说，一个方法可以不包含参数。
5. `return` 语句也可以用在 `void` 方法中，用来终止方法并返回到方法的调用者。在方法中，有时用于改变正常流程控制是很有用的。
6. 传递给方法的实际参数应该与方法签名中的形式参数具有相同的数目、类型和顺序。
7. 当程序调用一个方法时，程序控制就转移到被调用的方法。被调用的方法执行到该方法的 `return` 语句或到达方法结束的右括号时，将程序控制还给调用者。
8. 在 Java 中，带返回值的方法也可以当作语句调用。在这种情况下，调用函数只要忽略返回值即可。
9. 方法可以重载。这就意味着两个方法可以拥有相同的方法名，只要它们的方法参数列表不同即可。
10. 在方法中声明的变量称作局部变量。局部变量的作用域是从声明它的地方开始，到包含这个变量的块结束为止。局部变量在使用前必须声明和初始化。
11. 方法抽象是把方法的应用和实现分离。用户可以在不知道方法是如何实现的情况下使用方法。方法的实现细节封装在方法内，对调用该方法的用户隐藏。这称为信息隐藏或封装。
12. 方法抽象将程序模块化为整齐、层次分明的形式。将程序写成简洁的方法构成的集合会比其他方式更容易编写、调试、维护和修改。这种编写风格也会提高方法的可重用性。
13. 当实现一个大型程序时，可以使用自顶向下或自底向上的编码方法。不要一次性编写完整个程序。这种方式似乎浪费了更多的编码时间（因为要反复编译和运行这个程序），但实际上，它会更节省时间并使调试更容易。

测试题

本章节的测试题位于 www.cs.armstrong.edu/liang/intro10e/quiz.html。

编程练习题

☞ 注意：本章练习中学生常犯的错误是，没有实现符合要求的方法，尽管主程序的输出是正确的。这类错误的示例参见：www.cs.armstrong.edu/liang/CommonMethodErrorJava.pdf。

6.2 ~ 6.9 节

- 6.1 (数学：五角数) 一个五角数被定义为 $n(3n-1)/2$ ，其中 $n=1, 2, \dots$ 。所以，开始的几个数字就是 1, 5, 12, 22, ..., 编写下面的方法返回一个五角数：

```
public static int getPentagonalNumber(int n)
```

编写一个测试程序显示前 100 个五角数，每行显示 10 个。

- *6.2 (求一个整数各位数字之和) 编写一个方法，计算一个整数各位数字之和。使用下面的方法头：

```
public static int sumDigits(long n)
```

例如：sumDigits(234) 返回 9(2+3+4)。

- 提示：使用求余操作符 % 提取数字，用除号 / 去掉提取出来的数字。例如：使用 234%10 (=4) 抽取 4。然后使用 234/10 (=23) 从 234 中去掉 4。使用一个循环来反复提取和去掉每位数字，直到所有的位数都提取完为止。

编写程序提示用户输入一个整数，然后显示这个整数所有数字的和。

- **6.3 (回文整数) 使用下面的方法头编写两个方法：

```
// Return the reversal of an integer, i.e., reverse(456) returns 654
public static int reverse(int number)
```

```
// Return true if number is a palindrome
public static boolean isPalindrome(int number)
```

使用 reverse 方法实现 isPalindrome。如果一个数字的反向倒置数和它的顺向数一样，这个数就称作回文数。编写一个测试程序，提示用户输入一个整数值，然后报告这个整数是否是回文数。

- *6.4 (反向显示一个整数) 使用下面的方法头编写方法，反向显示一个整数：

```
public static void reverse(int number)
```

例如：reverse(3456) 返回 6543。编写一个测试程序，提示用户输入一个整数，然后显示它的反向数。

- *6.5 (对三个数排序) 使用下面的方法头编写方法，按升序显示三个数：

```
public static void displaySortedNumbers(
    double num1, double num2, double num3)
```

编写测试程序，提示用户输入三个数字，调用方法以升序显示他们。

- *6.6 (显示图案) 编写方法显示如下图案：

```

      1
     2 1
    3 2 1
   ...
  n n-1 ... 3 2 1
```

该方法头为：

```
public static void displayPattern(int n)
```

- *6.7 (财务应用程序：计算未来投资价值) 编写一个方法，计算按照给定的年数和利率计算未来投资价值，未来投资是用编程练习题 2.21 中的公式计算得到的。

使用下面的方法头：

```
public static double futureInvestmentValue(
    double investmentAmount, double monthlyInterestRate, int years)
```

例如：futureInvestmentValue(10000,0.05/12,5) 返回 12833.59。

编写一个测试程序，提示用户输入投资额 (例如 1000)、利率 (例如 9%)，然后打印年份从 1 到 30 年的未来投资价值，如下所示：

```

The amount invested: 1000
Annual interest rate: 9
Years      Future Value
1          1093.80
2          1196.41
...
29         13467.25
30         14730.57

```

6.8 (摄氏度和华氏度之间的转换) 编写一个类, 包含下面两个方法:

```

/** Convert from Celsius to Fahrenheit */
public static double celsiusToFahrenheit(double celsius)

/** Convert from Fahrenheit to Celsius */
public static double fahrenheitToCelsius(double fahrenheit)

```

转换公式如下:

华氏度 = (9.0 / 5) * 摄氏度 + 32
摄氏度 = (5.0 / 9) * (华氏度 - 32)

编写一个测试程序, 调用这两个方法来显示如下表格:

摄氏度	华氏度	华氏度	摄氏度
40.0	104.0	120.0	48.89
39.0	102.2	110.0	43.33
...			
32.0	89.6	40.0	4.44
31.0	87.8	30.0	-1.11

6.9 (英尺和米之间的转换) 编写一个类, 包含如下两个方法:

```

/** Convert from feet to meters */
public static double footToMeter(double foot)

/** Convert from meters to feet */
public static double meterToFoot(double meter)

```

转换公式如下:

米 = 0.305 * 英尺
英尺 = 3.279 * 米

编写一个测试程序, 调用这两个方法以显示下面的表格:

英尺	米	米	英尺
1.0	0.305	20.0	65.574
2.0	0.610	25.0	81.967
...			
9.0	2.745	60.0	196.721
10.0	3.050	65.0	213.115

6.10 (使用 isPrime 方法) 程序清单 6-7 提供了测试某个数字是否是素数的方法 isPrime(int number)。使用这个方法求小于 10000 的素数个数。

6.11 (财务应用程序: 计算佣金) 编写一个方法, 利用编程练习题 5.39 中的方案计算佣金。方法头如下所示:

```
public static double computeCommission(double salesAmount)
```

编写一个测试程序, 显示下面表格:

销售总额	酬金
10000	900.0
15000	1500.0
...	
95000	11100.0
100000	11700.0

6.12 (显示字符) 使用下面的方法头, 编写一个打印字符的方法:

```
public static void printChars(char ch1, char ch2, int
    numberPerLine)
```

该方法打印 ch1 到 ch2 之间的字符, 每行按指定个数打印。编写一个测试程序, 打印从 '1' 到 'Z' 的字符, 每行打印 10 个。字符之间使用一个空格字符隔开。

*6.13 (数列求和) 编写一个方法计算下列级数:

$$m(i) = \frac{1}{2} + \frac{2}{3} + \dots + \frac{i}{i+1}$$

编写一个测试程序, 显示下面的表格:

i	m(i)
1	0.5000
2	1.1667
...	
19	16.4023
20	17.3546

*6.14 (估算 π) π 可以使用下面的数列进行计算:

$$m(i) = 4 \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \dots + \frac{(-1)^{i+1}}{2i-1} \right)$$

编写一个方法, 对于给定的 i 返回 m(i), 并且编写一个测试程序, 显示如下表格:

i	m(i)
1	4.0000
101	3.1515
201	3.1466
301	3.1449
401	3.1441
501	3.1436
601	3.1433
701	3.1430
801	3.1428
901	3.1427

*6.15 (财务应用程序: 打印税表) 程序清单 3-5 给出计算税款的程序。使用下面的方法头编写一个计算税款的方法:

```
public static double computeTax(int status, double taxableIncome)
```

使用这个方法编写程序, 打印可征税收入从 50 000 美元到 60 000 美元, 收入间隔为 50 美元的所有婚姻状态纳税人的纳税表, 如下所示:

Taxable Income	Single	Married Joint	Married Separate	Head of a House
50000	8688	6665	8688	7353
50050	8700	6673	8700	7365
...				
59950	11175	8158	11175	9840
60000	11188	8165	11188	9853

 提示：使用 `Math.round`（即 `Math.round(computeTax(status, taxableIncome))`）将税收舍入为整数。

*6.16（一年的天数）使用下面的方法头编写一个方法，返回一年的天数：

```
public static int numberOfDaysInAYear(int year)
```

编写一个测试程序，显示从 2000 年到 2020 年每年的天数。

6.10 ~ 6.11 节

*6.17（显示 0 和 1 构成的矩阵）编写一个方法，使用下面的方法头显示 $n \times n$ 的矩阵：

```
public static void printMatrix(int n)
```

每个元素都是随机产生的 0 或 1。编写一个测试程序，提示用户输入 n ，显示如下所示的 $n \times n$ 矩阵：

```
Enter n: 3 
0 1 0
0 0 0
1 1 1
```

**6.18（检测密码）一些网站对于密码具有一些规则。编写一个方法，检测字符串是否是一个有效密码。假定密码规则如下：

- 密码必须至少 8 位字符。
- 密码仅能包含字母和数字。
- 密码必须包含至少两个数字。

编写一个程序，提示用户输入一个密码，如果符合规则，则显示 `Valid Password`，否则显示 `Invalid Password`。

*6.19（`MyTriangle` 类）创建一个名为 `MyTriangle` 的类，它包含如下两个方法：

```
/** Return true if the sum of any two sides is
 * greater than the third side. */
public static boolean isValid(
    double side1, double side2, double side3)
/** Return the area of the triangle. */
public static double area(
    double side1, double side2, double side3)
```

编写一个测试程序，读入三角形三边的值，若输入有效，则计算面积；否则显示输入无效。三角形面积的计算公式在编程练习题 2.19 中给出。

*6.20（计算一个字符串中字母的个数）编写一个方法，使用下面的方法头计算字符串中的字母个数：

```
public static int countLetters(String s)
```

编写一个测试程序，提示用户输入字符串，然后显示字符串中的字母个数。

*6.21（电话按键盘）国际标准的字母 / 数字匹配图如编程练习题 4.15 所示，编写一个方法，返回给定大写字母的数字，如下所示：

```
int getNumber(char uppercaseLetter)
```

编写一个测试程序，提示用户输入字符串形式的电话号码。输入的数字可能会包含字母。程序将字母（大写或者小写）翻译成一个数字，然后保持其他字符不变。下面是该程序的运行示例：

```
Enter a string: 1-800-Flowers 
1-800-3569377
```

```
Enter a string: 1800flowers 
18003569377
```

- **6.22（数学：平方根的近似求法）有几种实现 Math 类中 sqrt 方法的技术。其中一个称为巴比伦法。它通过使用下面公式的反复计算近似地得到：

```
nextGuess = (lastGuess + n / lastGuess) / 2
```

当 nextGuess 和 lastGuess 几乎相同时，nextGuess 就是平方根的近似值。最初的猜测值可以是任意一个正值（例如 1）。这个值就是 lastGuess 的初始值。如果 nextGuess 和 lastGuess 的差小于一个很小的数，比如 0.0001，就可以认为 nextGuess 是 n 的平方根的近似值；否则，nextGuess 就成为 lastGuess，近似过程继续执行。实现下面的方法，返回 n 的平方根。

```
public static double sqrt(long n)
```

- *6.23（指定字符的出现次数）使用下面的方法头编写一个方法，找到一个字符串中指定字符的出现次数。

```
public static int count(String str, char a)
```

例如，count("Welcome", 'e') 返回 2。编写一个测试程序，提示用户输入一个字符串以及一个字符，显示该字符在字符串中出现的次数。

6.10 ~ 6.12 节

- **6.24（显示当前日期和时间）程序清单 2-7 显示当前时间。改进这个例子，显示当前的日期和时间。在程序清单 6-12 中的日历例子，可以提供一些如何求年、月和日的思路。
- **6.25（将毫秒数转换成小时数、分钟数和秒数）使用下面的方法头，编写一个将毫秒数转换成小时数、分钟数和秒数的方法。

```
public static String convertMillis(long millis)
```

该方法返回形如“小时：分钟：秒”的字符串。例如：convertMillis(5500) 返回字符串 0:0:5，convertMillis(100000) 返回字符串 0:1:40，convertMillis(555550000) 返回字符串 154:19:10。

综合题

- **6.26（回文素数）回文素数是指一个数同时为素数和回文数。例如：131 是一个素数，同时也是一个回文素数。数字 313 和 757 也是如此。编写程序，显示前 100 个回文素数。每行显示 10 个数并且准确对齐，数字中间用空格隔开。如下所示：

```
2 3 5 7 11 101 131 151 181 191
313 353 373 383 727 757 787 797 919 929
...
```

- **6.27（反素数）反素数（反转拼写的素数）是指一个非回文素数，将其反转之后也是一个素数。例如：17 是一个素数，而 71 也是一个素数，所以 17 和 71 是反素数。编写程序，显示前 100 个反素

数。每行显示 10 个，并且数字间用空格隔开，如下所示：

```
13 17 31 37 71 73 79 97 107 113
149 157 167 179 199 311 337 347 359 389
...
```

- **6.28 (梅森素数) 如果一个素数可以写成 2^p-1 的形式，其中 p 是某个正整数，那么这个素数就称作梅森素数。编写程序，找出 $p \leq 31$ 的所有梅森素数，然后显示如下的输出结果：

p	2^p-1
2	3
3	7
5	31
...	

- **6.29 (双素数) 双素数是指一对差值为 2 的素数。例如：3 和 5 就是一对双素数，5 和 7 是一对双素数，而 11 和 13 也是一对双素数。编写程序，找出小于 1000 的所有双素数。显示结果如下所示：

```
(3, 5)
(5, 7)
...
```

- **6.30 (游戏：双骰儿赌博) 掷双骰子游戏是赌场中非常流行的骰子游戏。编写程序，玩这个游戏的一个变种，如下所示：

掷两个骰子。每个骰子有六个面，分别表示值 1, 2, ..., 6。检查这两个骰子的和。如果和为 2、3 或 12 (称为掷骰子 (craps))，你就输了；如果和是 7 或者 11 (称作自然 (natural))，你就赢了；但如果和是其他数字 (例如：4、5、6、8、9 或者 10)，就确定了一个点。继续掷骰子，直到掷出一个 7 或者掷出和刚才相同的点数。如果掷出的是 7，你就输了。如果掷出的点数和你前一次掷出的点数相同，你就赢了。

程序扮演一个独立的玩家。下面是一些运行示例。

```
You rolled 5 + 6 = 11
You win
```

```
You rolled 1 + 2 = 3
You lose
```

```
You rolled 4 + 4 = 8
point is 8
You rolled 6 + 2 = 8
You win
```

```
You rolled 3 + 2 = 5
point is 5
You rolled 2 + 5 = 7
You lose
```

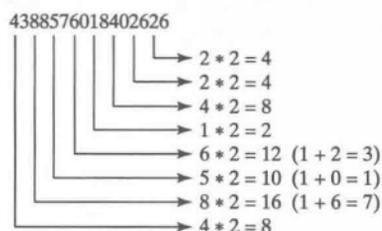
- **6.31 (财务应用程序：信用卡号的合法性) 信用卡号遵循下面的模式。一个信用卡号必须是 13 到 16 位的整数。它的开头必须是：

- 4，指 Visa 卡
- 5，指 Master 卡
- 37，指 American Express 卡
- 6，指 Discover 卡

在 1954 年，IBM 的 Hans Luhn 提出一种算法，该算法可以验证信用卡号的有效性。这个算法在确定输入的卡号是否正确，或者这张信用卡是否被扫描仪正确扫描方面是非常有用的。遵

循这个合法性检测可以生成所有的信用卡号，通常称之为 Luhn 检测或者 Mod 10 检测，可以如下描述（为了方便解释，假设卡号为 4388576018402626）：

1) 从左到右对每个数字翻倍。如果对某个数字翻倍之后的结果是一个两位数，那么就将其两位加在一起得到一位数。



2) 现在将第一步得到的所有一位数相加。

$$4+4+8+2+3+1+7+8=37$$

3) 将卡号里从左到右在奇数位上的所有数字相加。

$$6+6+0+8+0+7+8+3=38$$

4) 将第二步和第三步得到的结果相加。

$$37+38=75$$

5) 如果第四步得到的结果能被 10 整除，那么卡号是合法的；否则，卡号是不合法的。例如，号码 4388576018402626 是不合法的，但是号码 4388576018410707 是合法的。

编写程序，提示用户输入一个 long 型整数的信用卡号码，显示这个数字是合法的还是非法的。使用下面的方法设计程序：

```

/** Return true if the card number is valid */
public static boolean isValid(long number)

/** Get the result from Step 2 */
public static int sumOfDoubleEvenPlace(long number)

/** Return this number if it is a single digit, otherwise,
 * return the sum of the two digits */
public static int getDigit(int number)

/** Return sum of odd-place digits in number */
public static int sumOfOddPlace(long number)

/** Return true if the digit d is a prefix for number */
public static boolean prefixMatched(long number, int d)

/** Return the number of digits in d */
public static int getSize(long d)

/** Return the first k number of digits from number. If the
 * number of digits in number is less than k, return number. */
public static long getPrefix(long number, int k)

```

下面是程序的运行示例：（你也可以通过将输入作为一个字符串读入，以及对字符串进行处理来验证信用卡卡号。）

```

Enter a credit card number as a long integer:
4388576018410707 
4388576018410707 is valid

```

```

Enter a credit card number as a long integer:
4388576018402626 
4388576018402626 is invalid

```

- *6.32 (游戏: 赢取双骰子赌博游戏的机会) 修改编程练习题 6.30 使该程序运行 10 000 次, 然后显示赢得游戏的次数。
- *6.33 (当前日期和时间) 调用 `System.currentTimeMillis()` 返回从 1970 年 1 月 1 号 0 点开始至今为止的毫秒数。编写程序, 显示日期和时间。下面是运行示例:

```
Current date and time is May 16, 2012 10:34:23
```

- *6.34 (打印日历) 编程练习题 3.21 使用 Zeller 一致性原理来计算某天是星期几。使用 Zeller 的算法简化程序清单 6-12 以获得每月开始的第一天是星期几。
- 6.35 (几何问题: 五边形的面积) 五边形的面积可以使用下面的公式计算:

$$\text{面积} = \frac{5 \times s^2}{4 \times \tan\left(\frac{\pi}{5}\right)}$$

编写一个方法, 使用下面的方法头来返回五边形的面积。

```
public static double area(double side)
```

编写一个主方法, 提示用户输入五边形的边, 然后显示它的面积。下面是一个运行示例:

```
Enter the side: 5.5 
The area of the pentagon is 52.04444136781625
```

- *6.36 (几何问题: 正多边形的面积) 正多边形是一个 n 条边的多边形, 它的每条边的长度都相等, 而且所有角的角度也相等 (即多边形既是等边又等角的)。计算正多边形面积的公式是:

$$\text{面积} = \frac{n \times s^2}{4 \times \tan\left(\frac{\pi}{n}\right)}$$

使用下面的方法头编写方法, 返回正多边形的面积:

```
public static double area(int n, double side)
```

编写一个 `main` 方法, 提示用户输入边的个数以及正多边形的边长, 然后显示它的面积。下面是一个运行示例:

```
Enter the number of sides: 5 
Enter the side: 6.5 
The area of the polygon is 72.69017017488385
```

- 6.37 (格式化整数) 使用下面的方法头编写一个方法, 用于将整数格式化为指定宽度:

```
public static String format(int number, int width)
```

方法为数字 `number` 返回一个带有一个或多个以 0 作为前缀的字符串。字符串的位数就是宽度。比如, `format(34,4)` 返回 0034, `format(34,5)` 返回 00034。如果数字宽于指定宽度, 方法返回该数字的字符串表示。比如, `format(34,1)` 返回 34。

编写一个测试程序, 提示用户输入一个数字以及宽度, 显示通过调用 `format(number,width)` 返回的字符串。

- *6.38 (生成随机字符) 使用程序清单 6-10 中的方法 `RandomCharacter` 打印 100 个大写字母及 100 个一位数字, 每行显示 10 个。
- 6.39 (几何: 点的位置) 编程练习题 3.32 显示如何测试一个点是否在一个有向直线的左侧、右侧, 或在该直线上。使用下面的方法头编写该方法:

```

/** Return true if point (x2, y2) is on the left side of the
 * directed line from (x0, y0) to (x1, y1) */
public static boolean leftOfTheLine(double x0, double y0,
    double x1, double y1, double x2, double y2)

/** Return true if point (x2, y2) is on the same
 * line from (x0, y0) to (x1, y1) */
public static boolean onTheSameLine(double x0, double y0,
    double x1, double y1, double x2, double y2)

/** Return true if point (x2, y2) is on the
 * line segment from (x0, y0) to (x1, y1) */
public static boolean onTheLineSegment(double x0, double y0,
    double x1, double y1, double x2, double y2)

```

编写一个程序，提示用户输入三个点赋给 p0、p1 和 p2，显示 p2 是否是在从 p0 到 p1 的直线的左侧、右侧、直线上，或者线段上。下面是一些运行示例：

```

Enter three points for p0, p1, and p2: 1 1 2 2 1.5 1.5 --Enter
(1.5, 1.5) is on the line segment from (1.0, 1.0) to (2.0, 2.0)

```

```

Enter three points for p0, p1, and p2: 1 1 2 2 3 3 --Enter
(3.0, 3.0) is on the same line from (1.0, 1.0) to (2.0, 2.0)

```

```

Enter three points for p0, p1, and p2: 1 1 2 2 1 1.5 --Enter
(1.0, 1.5) is on the left side of the line
from (1.0, 1.0) to (2.0, 2.0)

```

```

Enter three points for p0, p1, and p2: 1 1 2 2 1 -1 --Enter
(1.0, -1.0) is on the right side of the line
from (1.0, 1.0) to (2.0, 2.0)

```

一维数组

教学目标

- 描述数组在程序设计中的必要性 (7.1 节)。
- 声明数组引用变量以及创建数组 (7.2.1 ~ 7.2.2 节)。
- 使用 `arrayRefVar.length` 获得数组的大小, 了解数组的默认值 (7.2.3 节)。
- 使用下标访问数组元素 (7.2.4 节)。
- 利用数组初始化语法声明、创建和初始化数组 (7.2.5 节)。
- 编写程序实现常用的数组操作 (显示数组, 对所有元素求和, 求最小和最大元素, 随机打乱和移动元素) (7.2.6 节)。
- 使用 `foreach` 循环简化程序设计 (7.2.7 节)。
- 在应用程序开发 (AnalyzeNumbers, DeckOfCards) 中应用数组 (7.3 ~ 7.4 节)。
- 将一个数组的内容复制到另一个数组 (7.5 节)。
- 开发和调用带数组参数和数组返回值的方法 (7.6 ~ 7.8 节)。
- 定义带变长参数列表的方法 (7.9 节)。
- 使用线性查找算法 (7.10.1 节) 或二分查找算法 (7.10.2 节) 查找数组的元素。
- 使用选择排序法对数组排序 (7.11 节)。
- 使用 `java.util.Arrays` 类中的方法 (7.12 节)。
- 从命令行传参数给主方法 (7.13 节)。

7.1 引言

 **要点提示:** 单个的数组变量可以引用一个大的数据集。

在执行程序的过程中, 经常需要存储大量的数据, 例如, 假设需要读取 100 个数, 计算它们的平均值, 然后找出有多少个数大于平均值。首先, 程序读入这些数并且计算它们的平均值, 然后将每个数与平均值进行比较判断它是否大于平均值。为了完成这个任务, 必须将全部的数据存储到变量中。必须声明 100 个变量, 并且重复书写 100 次几乎完全相同的代码。这样编写程序的方式似乎是不太现实的, 那么该如何解决这个问题呢?

这就需要有一个高效的有条理的方法。Java 和许多高级语言都提供了一种称作数组 (array) 的数据结构, 可以用它来存储一个元素个数固定且元素类型相同的有序集。在现在这个例子中, 可以将所有的 100 个数存储在一个数组中, 并且通过一个一维数组变量访问它。

本章介绍一维数组。下一章将介绍二维数组和多维数组。

7.2 数组的基础知识

 **要点提示:** 一旦数组被创建, 它的大小是固定的。使用一个数组引用变量, 通过下标来访问数组中的元素。

数组是用来存储数据的集合，但是，通常我们会发现把数组看作一个存储具有相同类型的变量集合会更有用。无须声明单个变量，例如：`number0`，`number1`，...，`number99`，只要声明一个数组变量 `numbers`，并且用 `numbers[0]`，`numbers[1]`，...，`numbers[99]` 来表示单个变量。本节介绍如何声明数组变量、创建数组以及使用下标变量处理数组。

7.2.1 声明数组变量

为了在程序中使用数组，必须声明一个引用数组的变量，并指明数组的元素类型。下面是声明数组变量的语法：

```
elementType[] arrayRefVar; (元素类型 [] 数组引用变量;)
```

`elementType` 可以是任意数据类型，但是数组中所有的元素都必须具有相同的数据类型。例如：下面的代码声明变量 `myList`，它引用一个具有 `double` 型元素的数组。

```
double[] myList;
```

 **注意：**也可以用 `elementType arrayRefVar[]` (元素类型 数组引用变量 []) 声明数组变量。这种来自 C/C++ 语言的风格被 Java 采纳以适用于 C/C++ 程序员。推荐使用 `elementType[] arrayRefVar` (元素类型 [] 数组引用变量) 风格。

7.2.2 创建数组

不同于基本数据类型变量的声明，声明一个数组变量时并不在内存中给数组分配任何空间。它只是创建一个对数组的引用的存储位置。如果变量不包含对数组的引用，那么这个变量的值为 `null`。除非数组已经被创建，否则不能给它分配任何元素。声明数组变量之后，可以使用下面的语法用 `new` 操作符创建数组，并且将它的引用赋给一个变量：

```
arrayRefVar = new elementType[arraySize];
```

这条语句做了两件事情：1) 使用 `new elementType[arraySize]` 创建了一个数组；2) 把这个新创建的数组的引用赋值给变量 `arrayRefVar`。

声明一个数组变量、创建数组、然后将数组引用赋值给变量这三个步骤可以合并在一个语句里，如下所示：

```
elementType[] arrayRefVar = new elementType[arraySize];
```

(元素类型 [] 数组引用变量 = new 元素类型 [数组大小] ;)

或

```
elementType arrayRefVar[] = new elementType[arraySize];
```

(元素类型 数组引用变量 = new 元素类型 [数组大小] ;)

下面是使用这条语句的一个例子：

```
double[] myList = new double[10];
```

这条语句声明了数组变量 `myList`，创建一个由 10 个 `double` 型元素构成的数组，并将该数组的引用赋值给 `myList`。使用以下语法给这些元素赋值：

```
arrayRefVar[index] = value;
```

例如，下面的代码初始化数组：

```
myList[0] = 5.6;
myList[1] = 4.5;
myList[2] = 3.3;
myList[3] = 13.2;
myList[4] = 4.0;
myList[5] = 34.33;
myList[6] = 34.0;
myList[7] = 45.45;
myList[8] = 99.993;
myList[9] = 11123;
```

图 7-1 展示了这个数组。



图 7-1 数组 `myList` 包含 10 个 `double` 型元素并且下标从 0 到 9 为 `int` 型

注意：一个数组变量看起来似乎是存储了一个数组，但实际上它存储的是指向数组的引用。严格地讲，一个数组变量和一个数组是不同的，但多数情况下它们的差别是可以忽略的。因此，为了简化，通常可以说 `myList` 是一个数组，而不用更长的陈述：`myList` 是一个含有 10 个 `double` 型元素数组的引用变量。

7.2.3 数组大小和默认值

当给数组分配空间时，必须指定该数组能够存储的元素个数，从而确定数组大小。创建数组之后就不能再修改它的大小。可以使用 `arrayRefVar.length` 得到数组的大小。例如：`myList.length` 为 10。

当创建数组后，它的元素被赋予默认值，数值型基本数据类型的默认值为 0，`char` 型的默认值为 `'\u0000'`，`boolean` 型的默认值为 `false`。

7.2.4 访问数组元素

数组元素可以通过下标访问。数组下标是基于 0 的，也就是说，其范围从 0 开始到 `arrayRefVar.length-1` 结束。例如，在图 7-1 的例子中，数组 `myList` 包含 10 个 `double` 值，而且下标从 0 到 9。

数组中的每个元素都可以使用下面的语法表示，称为下标变量 (indexed variable)：

```
arrayRefVar[index]; (数组引用变量 [下标];)
```

例如：`myList[9]` 表示数组 `myList` 的最后一个元素。

警告：一些语言使用圆括号引用数组元素，例如 `myList(9)`。而 Java 语言使用方括号，例如 `myList[9]`。

创建数组后，下标变量与正常变量的使用方法相同。例如：下面的代码是将 `myList[0]` 和 `myList[1]` 的值相加赋给 `myList[2]`。

```
myList[2] = myList[0] + myList[1];
```

下面的循环是将 0 赋给 `myList[0]`，1 赋给 `myList[1]`，...，9 赋给 `myList[9]`：

```
for (int i = 0; i < myList.length; i++) {
    myList[i] = i;
}
```

7.2.5 数组初始化语法

Java 有一个简捷的标记，称作数组初始化语法，它使用下面的语法将声明数组、创建数组和初始化数组结合到一条语句中：

```
elementType[] arrayRefVar = {value0, value1, ..., valuek}; (元素类型 [] 数组引用变量 = {值 0, 值 1, ..., 值 k};)
```

例如：

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

这条语句声明、创建并初始化包含 4 个元素的数组 `myList`，它等价于下列语句：

```
double[] myList = new double[4];
myList[0] = 1.9;
myList[1] = 2.9;
myList[2] = 3.4;
myList[3] = 3.5;
```

警告：数组初始化语法中不使用操作符 `new`。使用数组初始化语法时，必须将声明、创建和初始化数组都放在一条语句中。将它们分开会产生语法错误。因此，下面的语句是错误的：

```
double[] myList;
myList = {1.9, 2.9, 3.4, 3.5};
```

7.2.6 处理数组

处理数组元素时，经常会用到 `for` 循环，理由有以下两点：

- 1) 数组中所有元素都是同一类型的。可以使用循环以同样的方式反复处理这些元素。
- 2) 由于数组的大小是已知的，所以很自然地就使用 `for` 循环。

假设创建如下数组：

```
double[] myList = new double[10];
```

下面是一些处理数组的例子：

1) (使用输入值初始化数组) 下面的循环使用用户输入的数值初始化数组 `myList`。

```
java.util.Scanner input = new java.util.Scanner(System.in);
System.out.print("Enter " + myList.length + " values: ");
for (int i = 0; i < myList.length; i++)
    myList[i] = input.nextDouble();
```

2) (使用随机数初始化数组) 下面的循环使用 0.0 到 100.0 之间, 但小于 100.0 的随机值初始化数组 `myList`。

```
for (int i = 0; i < myList.length; i++) {
    myList[i] = Math.random() * 100;
}
```

3) (显示数组) 为了打印数组, 必须使用类似下面的循环, 打印数组中的每一个元素。

```
for (int i = 0; i < myList.length; i++) {
    System.out.print(myList[i] + " ");
}
```

提示: 对于 `char[]` 类型的数组, 可以使用一条打印语句打印。例如下面的代码显示 Dallas:

```
char[] city = {'D', 'a', 'l', 'l', 'a', 's'};
System.out.println(city);
```

4) (对所有元素求和) 使用名为 `total` 的变量存储和。`total` 的值初始化为 0。使用如下循环将数组中的每个元素加到 `total` 中:

```
double total = 0;
for (int i = 0; i < myList.length; i++) {
    total += myList[i];
}
```

5) (找出最大元素) 使用名为 `max` 的变量存储最大元素。将 `max` 的值初始化为 `myList[0]`。为了找出数组 `myList` 中的最大元素, 将每个元素与 `max` 比较, 如果该元素大于 `max`, 则更新 `max`。

```
double max = myList[0];
for (int i = 1; i < myList.length; i++) {
    if (myList[i] > max) max = myList[i];
}
```

6) (找出最大元素的最小下标值) 经常需要找出数组中的最大元素。如果数组中含有多个最大元素, 那么找出最大元素的最小下标值。假设数组 `myList` 为 {1,5,3,4,5,5}。最大元素为 5, 5 的最小下标为 1。使用名为 `max` 的变量存储最大元素, 使用名为 `indexOfMax` 的变量表示最大元素的下标。将 `max` 的值初始化为 `myList[0]`, 而将 `indexOfMax` 的值初始化为 0。将 `myList` 中的每个元素与 `max` 比较, 如果这个元素大于 `max`, 则更新 `max` 和 `indexOfMax`。

```
double max = myList[0];
int indexOfMax = 0;
for (int i = 1; i < myList.length; i++) {
    if (myList[i] > max) {
        max = myList[i];
        indexOfMax = i;
    }
}
```

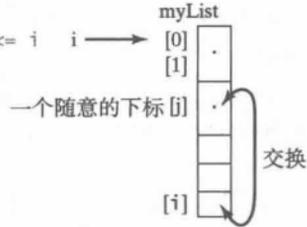
7) (随机打乱) 在很多应用程序中, 需要对数组中的元素进行任意的重新排序。这称作打乱 (shuffling)。为完成这种功能, 针对每个元素 `myList[i]`, 随意产生一个下标 `j`, 然后将 `myList[i]` 和 `myList[j]` 互换, 如下所示:

```

for (int i = myList.length - 1; i > 0; i--) {
    // Generate an index j randomly with 0 <= j <= i
    int j = (int)(Math.random()
        * (i + 1));

    // Swap myList[i] with myList[j]
    double temp = myList[i];
    myList[i] = myList[j];
    myList[j] = temp;
}

```



8) (移动元素) 有时候需要向左或向右移动元素。这里的例子就是将元素向左移动一个位置并且将第一个元素放在最后一个元素的位置:

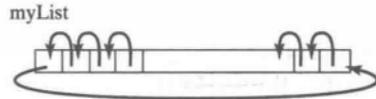
```

double temp = myList[0]; // Retain the first element

// Shift elements left
for (int i = 1; i < myList.length; i++) {
    myList[i - 1] = myList[i];
}

// Move the first element to fill in the last position
myList[myList.length - 1] = temp;

```



9) (简化编码) 对于某些任务来说, 数组可以极大简化编码。例如, 假设你想通过给定数字的月份来获得一个该月份的英文名字。如果月份名称保存在一个数组中, 给定月份的月份名可以简单通过下标获得。下面的代码提示用户输入一个月份数字, 然后显示它的月份名称:

```

String[] months = {"January", "February", ..., "December"};
System.out.print("Enter a month number (1 to 12): ");
int monthNumber = input.nextInt();
System.out.println("The month is " + months[monthNumber - 1]);

```

如果不使用 months 数组, 你将不得不使用一个很长的多分支 if-else 语句来确定月份名称, 如下所示:

```

if (monthNumber == 1)
    System.out.println("The month is January");
else if (monthNumber == 2)
    System.out.println("The month is February");
...
else
    System.out.println("The month is December");

```

7.2.7 foreach 循环

Java 支持一个简便的 for 循环, 称为 foreach 循环, 即不使用下标变量就可以顺序地遍历整个数组。例如, 下面的代码就可以显示数组 myList 的所有元素:

```

for (double e: myList) {
    System.out.println(e);
}

```

此代码可以读作“对 myList 中每个元素 e 进行以下操作”。注意, 变量 e 必须声明为与 myList 中元素相同的数据类型。

通常, foreach 循环的语法为:

```

for (elementType element: arrayRefVar) {
    // Process the element
}

```

但是, 当需要以其他顺序遍历数组或改变数组中的元素时, 还是必须使用下标变量。

警告: 越界访问数组是经常会出现的程序设计错误, 它会抛出一个运行错误 `ArrayIndexOutOfBoundsException`。为了避免错误的发生, 在使用时应确保所使用的下标不超过 `arrayRefVar.length-1`。

程序员经常错误地使用下标 1 引用数组的第一个元素, 但其实第一个元素的下标应该是 0。这称为下标过 1 错误 (off-by-one error)。它是在循环中该使用 < 的地方误用 <= 时会犯的错误的。例如, 下面的循环是错误的:

```
for (int i = 0; i <= list.length; i++)
    System.out.print(list[i] + " ");
```

应该用 < 替换 <=。

复习题

7.1 如何声明一个数组引用变量, 如何创建一个数组?

7.2 什么时候为数组分配内存?

7.3 下面代码的输出是什么?

```
int x = 30;
int[] numbers = new int[x];
x = 60;
System.out.println("x is " + x);
System.out.println("The size of numbers is " + numbers.length);
```

7.4 指出下列语句是对还是错:

- 数组中的每个元素都有相同的类型。
- 一旦数组被声明, 大小就不能改变。
- 一旦数组被创建, 大小就不能改变。
- 数组中的元素必须是基本数据类型。

7.5 以下哪些语句是有效的?

```
int i = new int(30);
double d[] = new double[30];
char[] r = new char(1..30);
int i[] = {3, 4, 3, 2};
float f[] = {2.3, 4.5, 6.6};
char[] c = new char();
```

7.6 如何访问数组的元素?

7.7 数组下标的类型是什么? 最小的下标是多少? 如何表示数组名为 a 的第三个元素?

7.8 编写语句完成:

- 创建一个含 10 个 double 值的数组。
- 将 5.5 赋值给数组中最后一个元素。
- 显示数组前两个元素的和。
- 编写循环计算数组中所有元素的和。
- 编写循环找出数组的最小值。
- 随机产生一个下标, 然后显示该下标所对应的数组元素。
- 使用数组初始化语法创建另一个初始值为 3.5、5.5、4.52 和 5.6 的数组。

7.9 当程序尝试访问下标不合法的数组元素时会发生什么?

7.10 找出错误并修改下面的代码:

```

1 public class Test {
2     public static void main(String[] args) {
3         double[100] r;
4
5         for (int i = 0; i < r.length(); i++);
6             r(i) = Math.random * 100;
7     }
8 }

```

7.11 以下代码的输出是什么?

```

1 public class Test {
2     public static void main(String[] args) {
3         int list[] = {1, 2, 3, 4, 5, 6};
4         for (int i = 1; i < list.length; i++)
5             list[i] = list[i - 1];
6
7         for (int i = 0; i < list.length; i++)
8             System.out.print(list[i] + " ");
9     }
10 }

```

7.3 示例学习：分析数字

 **要点提示：**编写一个程序，找到大于所有项平均值的那些项。

现在你可以编写程序来解决本章开始时提出的问题了。问题是，读取 100 个数，计算这些数的平均值并找到大于平均值的那些项的个数。为了更加灵活地处理任意数目的输入，我们让用户给出输入的个数，而不是将其固定为 100。程序清单 7-1 给出了一个解答。

程序清单 7-1 AnalyzeNumbers.java

```

1 public class AnalyzeNumbers {
2     public static void main(String[] args) {
3         java.util.Scanner input = new java.util.Scanner(System.in);
4         System.out.print("Enter the number of items: ");
5         int n = input.nextInt();
6         double [] numbers = new double[n];
7         double sum = 0;
8
9         System.out.print("Enter the numbers: ");
10        for (int i = 0; i < n; i++) {
11            numbers[i] = input.nextDouble();
12            sum += numbers[i];
13        }
14
15        double average = sum / n;
16
17        int count = 0; // The number of elements above average
18        for (int i = 0; i < n; i++)
19            if (numbers[i] > average)
20                count++;
21
22        System.out.println("Average is " + average);
23        System.out.println("Number of elements above the average is "
24            + count);
25    }
26 }

```

numbers[0]:	<input style="width: 40px; height: 15px;" type="text"/>
numbers[1]:	<input style="width: 40px; height: 15px;" type="text"/>
numbers[2]:	<input style="width: 40px; height: 15px;" type="text"/>
...	
numbers[i]:	
...	
numbers[n-3]:	<input style="width: 40px; height: 15px;" type="text"/>
numbers[n-2]:	<input style="width: 40px; height: 15px;" type="text"/>
numbers[n-1]:	<input style="width: 40px; height: 15px;" type="text"/>

```

Enter the number of items: 10
Enter the numbers: 3.4 5 6 1 6.5 7.8 3.5 8.5 6.3 9.5
Average is 5.75
Number of elements above the average is 6

```

程序提示用户输入数组的大小 (第 5 行), 然后根据该指定大小创建一个数组 (第 6 行)。程序读取输入, 将输入数字保存到一个数组中 (第 11 行), 并通过第 11 行代码将每个数字加到 sum 上面, 然后计算平均值 (第 15 行)。接着, 程序将每个数组中的数字与平均值比较, 从而计算大于平均值的数字个数 (17 ~ 20 行)。

7.4 示例学习: 一副牌

要点提示: 编写一个程序, 从一副牌中随机选出 4 张牌。

从一副 52 张的牌中随机挑出 4 张牌。所有的牌可以用一个名为 deck 的数组表示, 这个数组用从 0 到 51 的初始值来填充, 如下所示:

```
int[] deck = new int[52];

// Initialize cards
for (int i = 0; i < deck.length; i++)
    deck[i] = i;
```

牌号从 0 到 12、13 到 25、26 到 38 以及 39 到 51 分别表示 13 张黑桃、13 张红桃、13 张方块、13 张梅花, 如图 7-2 所示。cardNumber/13 决定牌的花色, 而 cardNumver%13 决定是具体花色中的哪张牌, 如图 7-3 所示。在打乱数组 deck 之后, 从 deck 中选出前四张牌。程序显示这四张牌号所对应的牌。

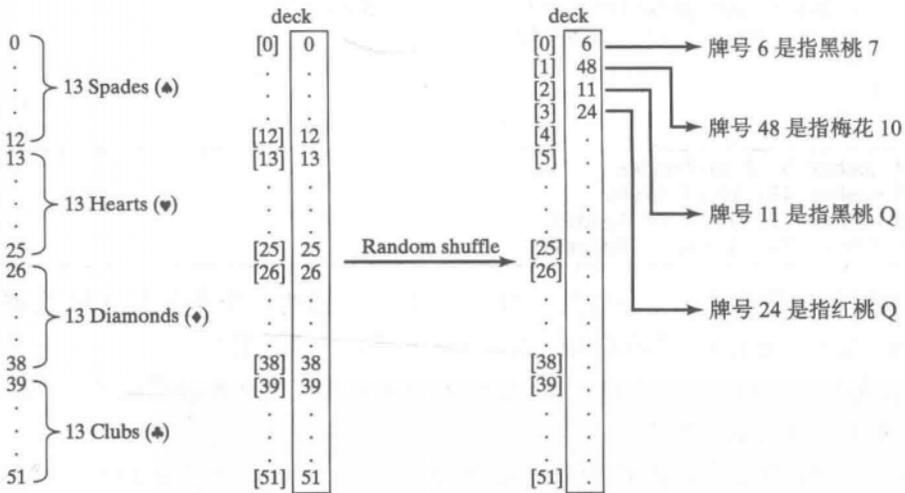


图 7-2 52 张牌存储在一个名为 deck 的数组中



图 7-3 CardNumber 标识一张牌的花色和等级数字

程序清单 7-2 给出了该问题的解决方案。

程序清单 7-2 DeckOfCards.java

```

1 public class DeckOfCards {
2     public static void main(String[] args) {
3         int[] deck = new int[52];
4         String[] suits = {"Spades", "Hearts", "Diamonds", "Clubs"};
5         String[] ranks = {"Ace", "2", "3", "4", "5", "6", "7", "8", "9",
6             "10", "Jack", "Queen", "King"};
7
8         // Initialize the cards
9         for (int i = 0; i < deck.length; i++)
10            deck[i] = i;
11
12        // Shuffle the cards
13        for (int i = 0; i < deck.length; i++) {
14            // Generate an index randomly
15            int index = (int)(Math.random() * deck.length);
16            int temp = deck[i];
17            deck[i] = deck[index];
18            deck[index] = temp;
19        }
20
21        // Display the first four cards
22        for (int i = 0; i < 4; i++) {
23            String suit = suits[deck[i] / 13];
24            String rank = ranks[deck[i] % 13];
25            System.out.println("Card number " + deck[i] + ": "
26                + rank + " of " + suit);
27        }
28    }
29 }

```

```

Card number 6: 7 of Spades
Card number 48: 10 of Clubs
Card number 11: Queen of Spades
Card number 24: Queen of Hearts

```

程序为四种花色定义了一个数组 `suits` (第 4 行), 而为一个花色中的 13 张牌定义一个数组 `ranks` (第 5 ~ 6 行)。这些数组中的每个元素都是一个字符串。

程序在第 9 ~ 10 行用 0 到 51 初始化 `deck`。`deck` 的值为 0 表示黑桃 A, 1 表示黑桃 2, 13 表示红桃 A, 14 表示红桃 2。

第 13 ~ 19 行随意地打乱这副牌。在牌被打乱后, `deck[i]` 中放的是一个任意的值。`deck[i]/13` 的值为 0、1、2 或 3, 该值确定这张牌是哪种花色 (第 23 行)。`deck[i]%13` 的值在 0 到 12 之间, 该值确定这张牌是花色中的哪张牌 (第 24 行)。如果没有定义 `suits` 数组, 你将不得不用比较冗长的多分支 `if-else` 语句来确定花色, 如下所示:

```

if (deck[i] / 13 == 0)
    System.out.print("suit is Spades");
else if (deck[i] / 13 == 1)
    System.out.print("suit is Hearts");
else if (deck[i] / 13 == 2)
    System.out.print("suit is Diamonds");
else
    System.out.print("suit is Clubs");

```

随着创建了数组 `suits = {"Spades", "Hearts", "Diamonds", "Clubs"}`, `suits[deck/13]` 给出了 `deck` 的花色。使用数组极大地简化了该程序的解决。

复习题

7.12 如果将程序清单 7-2 中的第 22~27 行替换为以下代码，程序还会挑选出四张随机的牌出来吗？

```
for (int i = 0; i < 4; i++) {
    int cardNumber = (int)(Math.random() * deck.length);
    String suit = suits[cardNumber / 13];
    String rank = ranks[cardNumber % 13];
    System.out.println("Card number " + cardNumber + ": "
        + rank + " of " + suit);
}
```

7.5 数组的复制

要点提示：要将一个数组中的内容复制到另外一个中，你需要将数组的每个元素复制到另外一个数组中。

在程序中经常需要复制一个数组或数组的一部分。这种情况下，你可能会尝试使用赋值语句 (=)，如下所示：

```
list2 = list1;
```

该语句并不能将 list1 引用的数组内容复制给 list2，而只是将 list1 的引用值复制给了 list2。在这条语句之后，list1 和 list2 都指向同一个数组，如图 7-4 所示。list2 原先所引用的数组不能再引用，它就变成了垃圾，会被 Java 虚拟机自动收回（这个过程称为垃圾回收）。

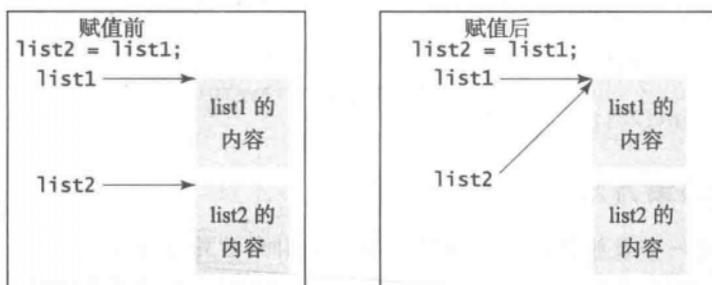


图 7-4 赋值语句执行前，list1 和 list2 指向各自的内存地址。
在赋值之后，数组 list1 的引用被传递给 list2

在 Java 中，可以使用赋值语句复制基本数据类型的变量，但不能复制数组。将一个数组变量赋值给另一个数组变量，实际上是将一个数组的引用复制给另一个变量，使两个变量都指向相同的内存地址。

复制数组有三种方法：

- 1) 使用循环语句逐个地复制数组的元素。
- 2) 使用 System 类中的静态方法 arraycopy。
- 3) 使用 clone 方法复制数组，这将在第 13 章中介绍。

可以使用循环将源数组中的每个元素复制到目标数组中的对应元素。例如，下述代码使用 for 循环将 sourceArray 复制到 targetArray：

```
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new int[sourceArray.length];
for (int i = 0; i < sourceArray.length; i++) {
```

```
    targetArray[i] = sourceArray[i];
}
```

另一种方式是使用 `java.lang.System` 类的 `arraycopy` 方法复制数组，而不是使用循环。`arraycopy` 的语法如下所示：

```
arraycopy(sourceArray, srcPos, targetArray, tarPos, length);
```

其中，参数 `srcPos` 和 `tarPos` 分别表示在源数组 `sourceArray` 和目标数组 `targetArray` 中的起始位置。从 `sourceArray` 复制到 `targetArray` 中的元素个数由参数 `length` 指定。例如，可以使用下面的语句改写上述循环：

```
System.arraycopy(sourceArray, 0, targetArray, 0, sourceArray.length);
```

`arraycopy` 方法没有给目标数组分配内存空间。复制前必须创建目标数组以及分配给它的内存空间。复制完成后，`sourceArray` 和 `targetArray` 具有相同的内容，但占有独立的内存空间。

 **注意：**`arraycopy` 方法违反了 Java 命名习惯。根据命名习惯，该方法应该命名为 `arrayCopy`（即字母 C 大写）。

复习题

7.13 使用 `arraycopy` 方法将下面的数组复制到目标数组 `t` 中：

```
int[] source = {3, 4, 5};
```

7.14 一旦数组被创建，它的大小不能被更改。那么下面的代码是否重设了数组的大小呢？

```
int[] myList;
myList = new int[10];
// Sometime later you want to assign a new array to myList
myList = new int[20];
```

7.6 将数组传递给方法

 **要点提示：**当将一个数组传递给方法时，数组的引用被传给方法。

正如前面给方法传递基本数据类型的值一样，也可以给方法传递数组。例如，下面的方法显示 `int` 型数组的元素：

```
public static void printArray(int[] array) {
    for (int i = 0; i < array.length; i++) {
        System.out.print(array[i] + " ");
    }
}
```

可以通过传递一个数组调用上面的方法。例如，下面的语句调用 `printArray` 方法显示 3、1、2、6、4 和 2：

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

 **注意：**前面的语句使用下述语法创建数组：

```
new elementType[]{value0, value1, ..., valuek};
```

该数组没有显式地引用变量，这样的数组称为匿名数组（anonymous array）。

Java 使用按值传递（`pass-by-value`）的方式将实参传递给方法。传递基本数据类型变量的值与传递数组值有很大的不同。

- 对于基本数据类型参数，传递的是实参的值。
- 对于数组类型参数，参数值是数组的引用，给方法传递的是这个引用。从语义上来讲，最好的描述就是参数传递的是共享信息 (pass-by-sharing)，即方法中的数组和传递的数组是一样的。所以，如果改变方法中的数组，将会看到方法外的数组也变化了。

例如，采用下面的代码：

```
public class Test {
    public static void main(String[] args) {
        int x = 1; // x represents an int value
        int[] y = new int[10]; // y represents an array of int values

        m(x, y); // Invoke m with arguments x and y

        System.out.println("x is " + x);
        System.out.println("y[0] is " + y[0]);
    }

    public static void m(int number, int[] numbers) {
        number = 1001; // Assign a new value to number
        numbers[0] = 5555; // Assign a new value to numbers[0]
    }
}
```

```
x is 1
y[0] is 5555
```

你会觉得困惑，为什么在调用 `m` 之后 `x` 仍然是 1，但是 `y[0]` 却变成了 5555。这是因为尽管 `y` 和 `numbers` 是两个独立的变量，但它们指向同一数组，如图 7-5 所示。当调用 `m(x,y)` 时，`x` 和 `y` 的值传递给 `number` 和 `numbers`。因为 `y` 包含数组的引用值，所以，`numbers` 现在包含的是指向同一数组的相同引用值。

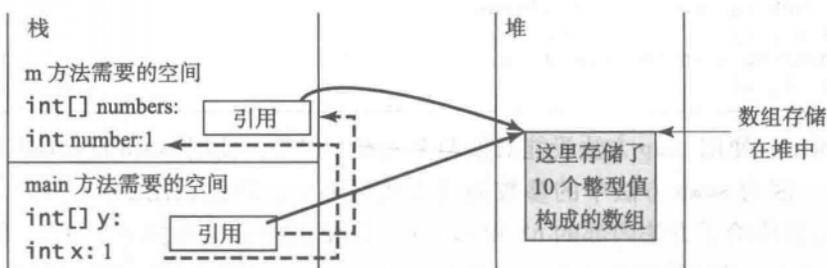


图 7-5 `x` 中的基本类型值被传递给 `number`，而 `y` 中的引用值被传递给 `numbers`

注意：数组在 Java 中是对象（对象将在第 9 章介绍）。JVM 将对象存储在一个称作堆（heap）的内存区域中，堆用于动态内存分配。

程序清单 7-3 给出另外一个例子，说明传递基本数据类型值与传递数组引用变量给方法的不同之处。

程序包含两个交换数组中元素的方法。第一个方法名为 `swap`，它没能将两个整型参数交换。第二个方法名为 `swapFirstTwoInArray`，它成功地将数组参数中前两个元素进行互换。

程序清单 7-3 TestPassArray.java

```
1 public class TestPassArray {
2     /** Main method */
3     public static void main(String[] args) {
4         int[] a = {1, 2};
```

```

5
6 // Swap elements using the swap method
7 System.out.println("Before invoking swap");
8 System.out.println("array is {" + a[0] + ", " + a[1] + "}");
9 swap(a[0], a[1]);
10 System.out.println("After invoking swap");
11 System.out.println("array is {" + a[0] + ", " + a[1] + "}");
12
13 // Swap elements using the swapFirstTwoInArray method
14 System.out.println("Before invoking swapFirstTwoInArray");
15 System.out.println("array is {" + a[0] + ", " + a[1] + "}");
16 swapFirstTwoInArray(a);
17 System.out.println("After invoking swapFirstTwoInArray");
18 System.out.println("array is {" + a[0] + ", " + a[1] + "}");
19 }
20
21 /** Swap two variables */
22 public static void swap(int n1, int n2) {
23     int temp = n1;
24     n1 = n2;
25     n2 = temp;
26 }
27
28 /** Swap the first two elements in the array */
29 public static void swapFirstTwoInArray(int[] array) {
30     int temp = array[0];
31     array[0] = array[1];
32     array[1] = temp;
33 }
34 }

```

```

Before invoking swap
array is {1, 2}
After invoking swap
array is {1, 2}
Before invoking swapFirstTwoInArray
array is {1, 2}
After invoking swapFirstTwoInArray
array is {2, 1}

```

如图 7-6 所示，使用 `swap` 方法没能对换两个元素。但是，使用 `swapFirstTwoInArray` 方法就实现了对换。因为 `swap` 方法中的参数为基本数据类型，所以调用 `swap(a[0], a[1])` 时，`a[0]` 和 `a[1]` 的值传给了方法内部的 `n1` 和 `n2`。`n1` 和 `n2` 的内存位置独立于 `a[0]` 和 `a[1]` 的内存位置。这个方法的调用没有影响数组的内容。

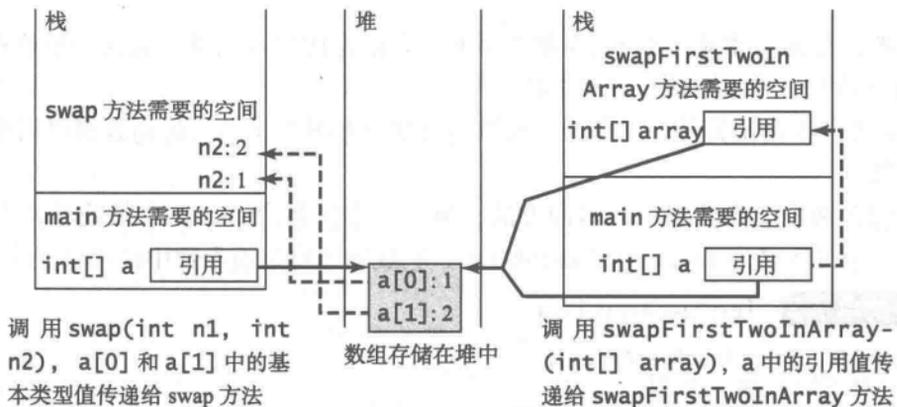


图 7-6 将数组传给方法时，传给方法的是数组的引用

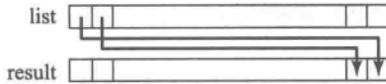
swapFirstTwoInArray 方法的参数是一个数组。如图 7-6 所示，数组的引用传给方法。这样，变量 a（方法外）和 array（方法内）都指向在同一内存位置中的同一个数组。因此，在方法 swapFirstTwoInArray 内交换 array[0] 与 array[1] 和在方法外交换 a[0] 与 a[1] 是一样的。

7.7 从方法中返回数组

要点提示：当从方法中返回一个数组时，数组的引用被返回。

可以在调用方法时向方法传递一个数组。方法也可以返回一个数组。例如，下面的方法返回一个与输入数组元素顺序相反的数组：

```
1 public static int[] reverse(int[] list) {
2     int[] result = new int[list.length];
3
4     for (int i = 0, j = result.length - 1;
5         i < list.length; i++, j--) {
6         result[j] = list[i];
7     }
8
9     return result;
10 }
```



第 2 行创建了一个新数组 result，第 4 ~ 7 行把数组 list 的元素复制到数组 result 中。第 9 行返回数组。例如，下面的语句返回元素为 6、5、4、3、2、1 的新数组 list2。

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

复习题

7.15 假设以下所写代码用于将数组中的内容进行反转，解释为什么它是错误的，以及如何修正？

```
int[] list = {1, 2, 3, 5, 4};

for (int i = 0, j = list.length - 1; i < list.length; i++, j--) {
    // Swap list[i] with list[j]
    int temp = list[i];
    list[i] = list[j];
    list[j] = temp;
}
```

7.8 示例学习：统计每个字母出现的次数

要点提示：本节给出一个程序，用于统计一个字符数组中每个字母出现的次数。

程序清单 7-4 中的程序完成下述任务：

1) 随机生成 100 个小写字母并将其放入一个字符数组中，如图 7-7a 所示。可以使用程序清单 6-10 中 RandomCharacter 类中的 getRandomLowerCaseLetter() 方法获取一个随机字母。

2) 对数组中每个字母出现的次数进行计数。为了完成这个功能，创建一个具有 26 个 int 值的数组 counts，每个值存放每个字母出现的次数，如图 7-7b 所示。也就

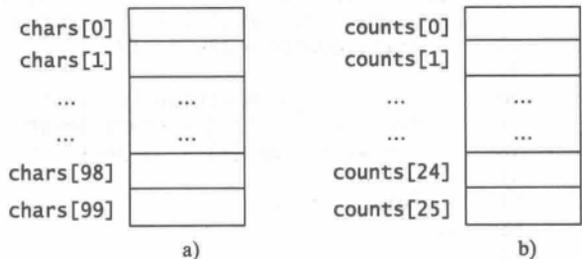


图 7-7 数组 chars 存储 100 个字符，数组 counts 存储 26 个计数器变量，每个计数器变量对一个字母进行计数

是说, counts[0] 记录 a 出现的次数, counts[1] 记录 b 出现的次数, 依此类推。

程序清单 7-4 CountLettersInArray.java

```

1  public class CountLettersInArray {
2      /** Main method */
3      public static void main(String[] args) {
4          // Declare and create an array
5          char[] chars = createArray();
6
7          // Display the array
8          System.out.println("The lowercase letters are:");
9          displayArray(chars);
10
11         // Count the occurrences of each letter
12         int[] counts = countLetters(chars);
13
14         // Display counts
15         System.out.println();
16         System.out.println("The occurrences of each letter are:");
17         displayCounts(counts);
18     }
19
20     /** Create an array of characters */
21     public static char[] createArray() {
22         // Declare an array of characters and create it
23         char[] chars = new char[100];
24
25         // Create lowercase letters randomly and assign
26         // them to the array
27         for (int i = 0; i < chars.length; i++)
28             chars[i] = RandomCharacter.getRandomLowerCaseLetter();
29
30         // Return the array
31         return chars;
32     }
33
34     /** Display the array of characters */
35     public static void displayArray(char[] chars) {
36         // Display the characters in the array 20 on each line
37         for (int i = 0; i < chars.length; i++) {
38             if ((i + 1) % 20 == 0)
39                 System.out.println(chars[i]);
40             else
41                 System.out.print(chars[i] + " ");
42         }
43     }
44
45     /** Count the occurrences of each letter */
46     public static int[] countLetters(char[] chars) {
47         // Declare and create an array of 26 int
48         int[] counts = new int[26];
49
50         // For each lowercase letter in the array, count it
51         for (int i = 0; i < chars.length; i++)
52             counts[chars[i] - 'a']++;
53
54         return counts;
55     }
56
57     /** Display counts */
58     public static void displayCounts(int[] counts) {
59         for (int i = 0; i < counts.length; i++) {
60             if ((i + 1) % 10 == 0)

```

```

61         System.out.println(counts[i] + " " + (char)(i + 'a'));
62     else
63         System.out.print(counts[i] + " " + (char)(i + 'a') + " ");
64     }
65 }
66 }

```

```

The lowercase letters are:
e y l s r i b k j v j h a b z n w b t v
s c c k r d w a m p w v u n q a m p l o
a z g d e g f i n d x m z o u l o z j v
h w i w n t g x w c d o t x h y v z y z
q e a m f w p g u q t r e n n w f c r f

The occurrences of each letter are:
5 a 3 b 4 c 4 d 4 e 4 f 4 g 3 h 3 i 3 j
2 k 3 l 4 m 6 n 4 o 3 p 3 q 4 r 2 s 4 t
3 u 5 v 8 w 3 x 3 y 6 z

```

方法 `createArray` (第 21 ~ 32 行) 生成一个存放 100 个随机小写字母的数组。第 5 行调用该方法, 并且将这个数组赋值给 `chars`。如果将代码改写成如下形式, 会出现什么错误?

```

char[] chars = new char[100];
chars = createArray();

```

这样将会创建两个数组。第一行使用 `new char[100]` 创建一个数组。第二行通过调用 `createArray()` 创建一个数组, 并将这个数组的引用赋值给 `chars`。第一行生成的数组就变成了“垃圾”, 因为它不能再被引用。Java 在后台自动回收“垃圾”。程序可以正常地编译和运行, 但它会创建一个不必要的数组。

调用 `getRandomLowerCaseLetter()` (第 28 行) 返回一个随机小写字母。该方法是在程序清单 6-10 的 `RandomCharacter` 类中定义的。

`countLetters` 方法 (第 46 ~ 55 行) 返回一个含 26 个 `int` 型值的数组, 每个整数存放的是每个字母出现的次数。该方法处理数组中的每个字母, 并将它对应的计数器加 1。统计字母出现次数的穷举方法可以如下所示:

```

for (int i = 0; i < chars.length; i++)
    if (chars[i] == 'a')
        counts[0]++;
    else if (chars[i] == 'b')
        counts[1]++;
    ...

```

但是程序第 51 ~ 52 行给出一个更好的解决方案。

```

for (int i = 0; i < chars.length; i++)
    counts[chars[i] - 'a']++;

```

如果字母 (`chars[i]`) 是 'a', 那么它对应的计数器就是 `counts['a'-'a']` (即 `counts[0]`)。如果字母是 'b', 因为 'b' 的 Unicode 码比 'a' 的统一码大 1, 所以它对应的计数器为 `counts['b'-'a']` (即 `counts[1]`)。如果字母是 'z', 因为 'z' 的 Unicode 码比 'a' 的大 25, 所以它对应的计数器为 `counts['z'-'a']` (即 `counts[25]`)。

图 7-8 显示在执行 `createArray` 的过程中和执行之后调用栈和堆的情况。参见复习题 7.18, 得到程序中其他方法调用栈堆的演示。

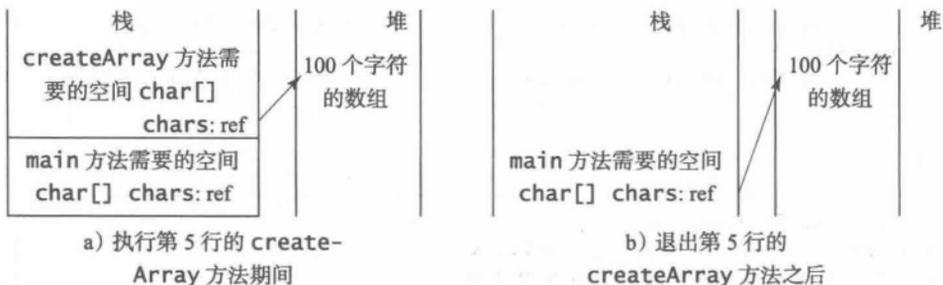


图 7-8 a) 执行 createArray 方法时, 100 个字符的数组被创建;

b) 在 main 方法中返回这个数组并赋值给变量 chars

复习题

7.16 下面说法真还是假? 当传递一个数组给方法时, 一个新的数组被创建并且传递给方法。

7.17 给出以下两个程序的输出:

```
public class Test {
    public static void main(String[] args) {
        int number = 0;
        int[] numbers = new int[1];

        m(number, numbers);

        System.out.println("number is " + number
            + " and numbers[0] is " + numbers[0]);
    }

    public static void m(int x, int[] y) {
        x = 3;
        y[0] = 3;
    }
}
```

a)

```
public class Test {
    public static void main(String[] args) {
        int[] list = {1, 2, 3, 4, 5};
        reverse(list);
        for (int i = 0; i < list.length; i++)
            System.out.print(list[i] + " ");
    }

    public static void reverse(int[] list) {
        int[] newList = new int[list.length];

        for (int i = 0; i < list.length; i++)
            newList[i] = list[list.length - 1 - i];

        list = newList;
    }
}
```

b)

7.18 在程序执行过程中, 数组保存在哪里? 给出程序清单 7-4 中执行 displayArray、countLetters、displayCounts 过程中以及之后堆栈中的内容。

7.9 可变长参数列表

要点提示: 具有同样类型的可变长度的参数可以传递给方法, 并将作为数组对待。

可以把类型相同但个数可变的参数传递给方法。方法中的参数声明如下:

typeName... parameterName (类型名 ... 参数名)

在方法声明中, 指定类型后紧接着省略号 (...). 只能给方法中指定一个可变长参数, 同时该参数必须是最后一个参数。任何常规参数必须在它之前。

Java 将可变长参数当成数组对待。可以将一个数组或数目可变的参数传递给可变长参数。当用数目可变的参数调用方法时, Java 会创建一个数组并把参数传给它。程序清单 7-5 包括了打印出个数不定的列表中最大值的方法。

程序清单 7-5 VarArgsDemo.java

```
1 public class VarArgsDemo {
2     public static void main(String[] args) {
```

```
3     printMax(34, 3, 3, 2, 56.5);
4     printMax(new double[]{1, 2, 3});
5 }
6
7 public static void printMax(double... numbers) {
8     if (numbers.length == 0) {
9         System.out.println("No argument passed");
10        return;
11    }
12
13    double result = numbers[0];
14
15    for (int i = 1; i < numbers.length; i++)
16        if (numbers[i] > result)
17            result = numbers[i];
18
19    System.out.println("The max value is " + result);
20 }
21 }
```

第3行将一个可变量参数列表传给数组 `numbers` 来调用 `printMax` 方法。如果没有传入参数，数组的长度为0（第8行）。

第4行传递一个数组调用 `printMax` 方法。

复习题

7.19 下面的方法头哪里有错误？

```
public static void print(String... strings, double... numbers)
public static void print(double... numbers, String name)
public static double... print(double d1, double d2)
```

7.20 可以使用下面的语句来调用程序清单 7-5 中的 `printMax` 方法吗？

```
printMax(1, 2, 2, 1, 4);
printMax(new double[]{1, 2, 3});
printMax(new int[]{1, 2, 3});
```

7.10 数组的查找

要点提示：如果一个数组排好序了，对于寻找数组中的一个元素，二分查找比线性查找更加高效。

查找（`searching`）是在数组中寻找特定元素的过程，例如：判断某一特定分数是否包括在成绩列表中。查找是计算机程序设计中经常要完成的任务。有很多用于查找的算法和数据结构。本节讨论两种经常使用的方法：线性查找（`linear searching`）和二分查找（`binary searching`）。

7.10.1 线性查找法

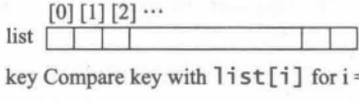
线性查找法将要查找的关键字 `key` 与数组中的元素逐个进行比较。这个过程持续到在列表中找到与关键字匹配的元素，或者查完列表也没有找到关键字为止。如果匹配成功，线性查找法返回与关键字匹配的元素在数组中的下标。如果没有匹配成功，则返回 `-1`。程序清单 7-6 中的 `linearSearch` 方法给出解决方案：

程序清单 7-6 LinearSearch.java

```

1 public class LinearSearch {
2     /** The method for finding a key in the list */
3     public static int linearSearch(int[] list, int key) {
4         for (int i = 0; i < list.length; i++) {
5             if (key == list[i])
6                 return i;
7         }
8         return -1;
9     }
10 }

```



list

[0]	[1]	[2]	...
-----	-----	-----	-----

key Compare key with list[i] for i = 0, 1, ...

为了更好地理解这个方法，对下面的语句跟踪这个方法：

```

1 int[] list = {1, 4, 4, 2, 5, -3, 6, 2};
2 int i = linearSearch(list, 4); // Returns 1
3 int j = linearSearch(list, -4); // Returns -1
4 int k = linearSearch(list, -3); // Returns 5

```

线性查找法把关键字和数组中的每一个元素进行比较。数组中的元素可以按任意顺序排列。平均来看，如果关键字存在，那么在找到关键字之前，这种算法必须与数组中一半的元素进行比较。由于线性查找法的执行时间随着数组元素个数的增长而线性增长，所以，对于大数组而言，线性查找法的效率并不高。

7.10.2 二分查找法

二分查找法是另一种常见的对数值列表的查找方法。使用二分查找法的前提条件是数组中的元素必须已经排好序。假设数组已按升序排列。二分查找法首先将关键字与数组的中间元素进行比较。考虑下面三种情况：

- 如果关键字小于中间元素，只需要在数组的前一半元素中继续查找关键字。
- 如果关键字和中间元素相等，则匹配成功，查找结束。
- 如果关键字大于中间元素，只需要在数组的后一半元素中继续查找关键字。

显然，二分法在每次比较之后就排除掉一半的数组元素，有时候是去掉一半的元素，有时候是去掉一半加1个元素。假设数组有 n 个元素。为方便起见，假设 n 是 2 的幂。经过第 1 次比较，只剩下 $n/2$ 个元素需要进一步查找；经过第 2 次比较，剩下 $(n/2)/2$ 个元素需要进一步查找。经过 k 次比较之后，需要查找的元素就剩下 $n/2^k$ 个。当 $k = \log_2 n$ 时，数组中只剩下 1 个元素，就只需要再比较 1 次。因此，在一个已经排序的数组中用二分查找法查找一个元素，即使是最坏的情况，也只需要 $\log_2 n + 1$ 次比较。对于一个有 1024 (2^{10}) 个元素的数组，在最坏情况下，二分查找法只需要比较 11 次，而在最坏的情况下线性查找要比较 1023 次。

每次比较后，数组要查找的部分就会缩小一半。用 `low` 和 `high` 分别表示当前查找数组的第一个下标和最后一个下标。初始条件下，`low` 为 0，而 `high` 为 `list.length-1`。让 `mid` 表示列表的中间元素的下标。这样，`mid` 就是 $(low + high)/2$ 。图 7-9 显示怎样使用二分法从列表 {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79} 中找出关键字 11。

现在知道了二分查找法是如何工作的。下一个任务就是在 Java 中实现它。不要急于给出一个完整的实现。逐步地实现这个程序，一次一步。可以从查找的第一次迭代开始，如图 7-10a 所示。它将关键字 `key` 和低下标 `low` 为 0、高下标 `high` 为 `list.length-1` 的列表的中间元素进行比较。如果 `key < list[mid]`，就将下标 `high` 设置为 `mid-1`；如果 `key == list[mid]`，则匹配成功并返回 `mid`；如果 `key > list[mid]`，就将下标 `low` 设置为 `mid+1`。

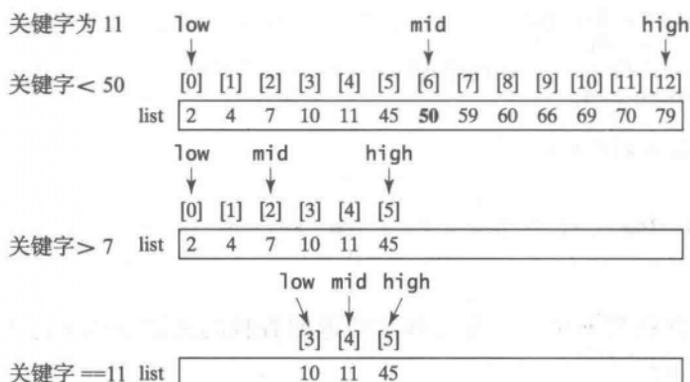


图 7-9 二分查找法在每次比较后，列表中需要进一步考虑的元素减少一半

接下来就要考虑增加一个循环，实现这个方法重复地完成查找，如图 7-10b 所示。如果找到这个关键字，或者当 $low > high$ 时还没有找到这个关键字，就结束这个查找。

```
public static int binarySearch(
    int[] list, int key) {
    int low = 0;
    int high = list.length - 1;

    int mid = (low + high) / 2;
    if (key < list[mid])
        high = mid - 1;
    else if (key == list[mid])
        return mid;
    else
        low = mid + 1;
}
```

a) 版本 1

```
public static int binarySearch(
    int[] list, int key) {
    int low = 0;
    int high = list.length - 1;

    while (high >= low) {
        int mid = (low + high) / 2;
        if (key < list[mid])
            high = mid - 1;
        else if (key == list[mid])
            return mid;
        else
            low = mid + 1;
    }

    return -1; // Not found
}
```

b) 版本 2

图 7-10 逐步实现二分查找法

当没有找到这个关键字时， low 就是一个插入点，这个位置将插入关键字以保持列表的有序性。一种更实用的方法是返回插入点减去 1。这个方法必须返回一个负值，表明这个关键字不在该序列中。可以只返回 $-low$ 吗？答案是：不可以。如果关键字小于 $list[0]$ ，那么 low 就是 0， -0 也是 0。这就表明关键字匹配 $list[0]$ 。一个好的选择是，如果关键字不在该序列中，方法返回 $-low-1$ 。返回 $-low-1$ 不仅表明关键字不在序列中，而且还给出了关键字应该插入的地方。

完整的程序在程序清单 7-7 中给出。

程序清单 7-7 BinarySearch.java

```
1 public class BinarySearch {
2     /** Use binary search to find the key in the list */
3     public static int binarySearch(int[] list, int key) {
4         int low = 0;
5         int high = list.length - 1;
6
7         while (high >= low) {
8             int mid = (low + high) / 2;
```

```

9     if (key < list[mid])
10        high = mid - 1;
11     else if (key == list[mid])
12        return mid;
13     else
14        low = mid + 1;
15 }
16
17 return -low - 1; // Now high < low, key not found
18 }
19 }

```

如果关键字包含在列表中，二分查找法就返回查找的关键字的下标（第12行）；否则，返回 $-low-1$ （第17行）。

如果用 $(high > low)$ 替换第7行的 $(high \geq low)$ ，会出现什么现象呢？这个查找也许会漏掉可能的匹配元素。假如列表只有一个元素，这个查找就会漏掉这个元素。

如果列表中有重复的元素，这个方法还能使用吗？回答是肯定的，只要列表中的元素是按递增顺序排列的。如果查找的元素在列表中，那么该方法就返回匹配元素中的一个下标。

为了更好地理解这个方法，使用下面的语句跟踪这个方法，当方法返回时确定 low 和 $high$ 。

```

int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79};
int i = BinarySearch.binarySearch(list, 2); // Returns 0
int j = BinarySearch.binarySearch(list, 11); // Returns 4
int k = BinarySearch.binarySearch(list, 12); // Returns -6
int l = BinarySearch.binarySearch(list, 1); // Returns -1
int m = BinarySearch.binarySearch(list, 3); // Returns -2

```

下面的表格列出当方法退出时 low 和 $high$ 的值，以及调用该方法的返回值。

方法	Low	High	返回值
<code>binarySearch(list,2)</code>	0	1	0
<code>binarySearch(list,11)</code>	3	5	4
<code>binarySearch(list,12)</code>	5	4	-6
<code>binarySearch(list,1)</code>	0	-1	-1
<code>binarySearch(list,3)</code>	1	0	-2

 **注意：**线性查找法适用于在较小数组或没有排序的数组中查找，但是对大数组而言效率不高。二分查找法的效率较高，但它要求数组已经排好序。

复习题

- 7.21 如果 $high$ 是一个非常大的整数，比如最大的 `int` 值 2147483647， $(low + high)/2$ 可能导致溢出。如何修改从而防止溢出？
- 7.22 以图 7-9 为例，显示如何应用二分查找法在列表 `{2,4,7,10,11,45,50,59,60,66,69,70,79}` 中查找关键字 10 和关键字 12。
- 7.23 如果二分查找方法返回 -4 ，该关键字在列表中吗？如果希望将该关键字插入到列表中，应该在什么位置？

7.11 数组的排序

 **要点提示：**如同查找一样，排序是计算机编程中非常普遍的一个任务。对于排序已经开发出很多不同的算法。本节介绍一个直观的排序算法：选择排序。

假设要按升序排列一个数列。选择排序法先找到数列中最小的数，然后将它和第一个元素交换。接下来，在剩下的数中找到最小数，将它和第二个元素交换，依此类推，直到数列中仅剩一个数为止。图 7-11 显示如何使用选择排序法对数列 {2,9,5,4,8,1,6} 进行排序。

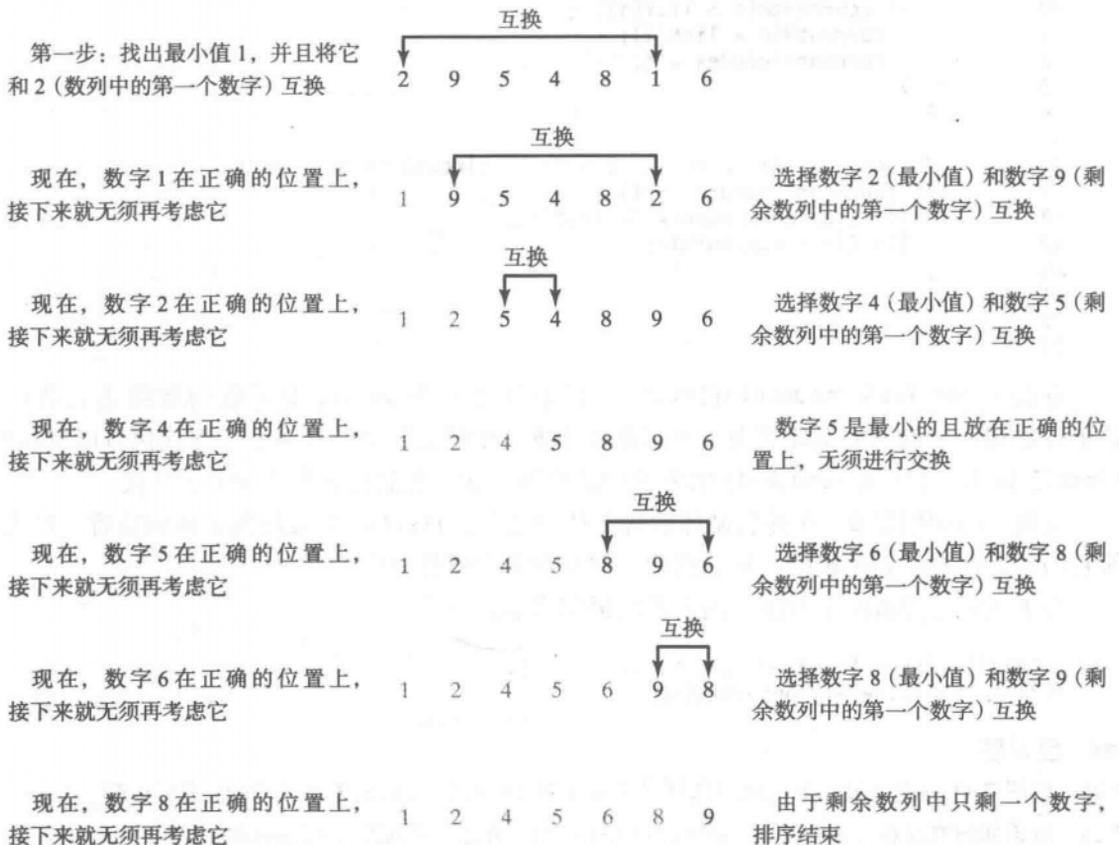


图 7-11 选择排序重复选择数列中的最小数，然后将它和数列中的第一个数字互换

已经知道了选择排序法是如何工作的。现在的任务是用 Java 语言实现它。对初学者来说，很难在第一次尝试时就开发出完整的解决方案。开始编写第一次迭代的代码，找出数列中的最大数，将其与最后一个元素互换，然后观察第二次迭代与第一次的不同之处，接着是第三次，依此类推。通过这样的观察可以写出推广到所有迭代的循环。

可以如下描述解决方案：

```
for (int i = 0; i < list.length - 1; i++) {
    select the smallest element in list[i..list.length-1];
    swap the smallest with list[i], if necessary;
    // list[i] is in its correct position.
    // The next iteration applies on list[i+1..list.length-1]
}
```

程序清单 7-8 实现了该解决方案。

程序清单 7-8 SelectionSort.java

```
1 public class SelectionSort {
2     /** The method for sorting the numbers */
3     public static void selectionSort(double[] list) {
4         for (int i = 0; i < list.length - 1; i++) {
```

```

5 // Find the minimum in the list[i..list.length-1]
6 double currentMin = list[i];
7 int currentMinIndex = i;
8
9 for (int j = i + 1; j < list.length; j++) {
10     if (currentMin > list[j]) {
11         currentMin = list[j];
12         currentMinIndex = j;
13     }
14 }
15
16 // Swap list[i] with list[currentMinIndex] if necessary
17 if (currentMinIndex != i) {
18     list[currentMinIndex] = list[i];
19     list[i] = currentMin;
20 }
21 }
22 }
23 }

```

方法 `selectionSort(double[] list)` 可以对任意一个 `double` 型元素的数组进行排序。这个方法用嵌套的 `for` 循环实现。外层循环（循环控制变量 `i`）（第 4 行）迭代执行以寻找从 `list[i]` 到 `list[list.length-1]` 的列表中最小的元素，然后将它和 `list[i]` 互换。

变量 `i` 的初值是 0。在外层循环的每次迭代之后，`list[i]` 都被放到正确的位置。最后，所有的元素都被放到正确的位置，因此，整个数列也就排好序了。

为了更好地理解这个方法，用下面的语句跟踪该方法：

```

double[] list = {1, 9, 4.5, 6.6, 5.7, -4.5};
SelectionSort.selectionSort(list);

```

复习题

- 7.24 以图 7-11 为例，显示如何应用选择排序方法对 {3.4, 5, 3, 3.5, 2.2, 1.9, 2} 进行排序。
 7.25 应该如何修改程序清单 7-8 中的 `selectionSort` 方法，实现数字按递减顺序排序？

7.12 Arrays 类

 **要点提示：** `java.util.Arrays` 类包含一些实用的方法用于常见的数组操作，比如排序和查找。

`java.util.Arrays` 类包括各种各样的静态方法，用于实现数组的排序和查找、数组的比较和填充数组元素，以及返回数组的字符串表示。这些方法都有对所有基本类型的重载方法。

可以使用 `sort` 或者 `parallelSort` 方法对整个数组或部分数组进行排序。例如，下面的代码对数值型数组和字符型数组进行排序。

```

double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};
java.util.Arrays.sort(numbers); // Sort the whole array
java.util.Arrays.parallelSort(numbers); // Sort the whole array

char[] chars = {'a', 'A', '4', 'F', 'D', 'P'};
java.util.Arrays.sort(chars, 1, 3); // Sort part of the array
java.util.Arrays.parallelSort(chars, 1, 3); // Sort part of the array

```

可以调用 `sort(numbers)` 对整个数组 `numbers` 排序。可以调用 `sort(chars, 1, 3)` 对从 `chars[1]` 到 `chars[3-1]` 的部分数组排序。如果你的计算机有多个处理器，那么 `parallelSort` 将更加高效。

可以采用二分查找法 (binarySearch 方法) 在数组中查找关键字。数组必须提前按升序排列好。如果数组中不存在关键字, 方法返回 -(插入点下标 +1)。例如, 下面的代码在整数数组和字符数组中查找关键字:

```
int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79};
System.out.println("1. Index is " +
    java.util.Arrays.binarySearch(list, 11));
System.out.println("2. Index is " +
    java.util.Arrays.binarySearch(list, 12));

char[] chars = {'a', 'c', 'g', 'x', 'y', 'z'};
System.out.println("3. Index is " +
    java.util.Arrays.binarySearch(chars, 'a'));
System.out.println("4. Index is " +
    java.util.Arrays.binarySearch(chars, 't'));
```

前面代码的输出为:

1. Index is 4
2. Index is -6
3. Index is 0
4. Index is -4

可以采用 equals 方法检测两个数组是否相等。如果它们的内容相同, 那么这两个数组相等。在下面的代码中, list1 和 list2 相等, 而 list2 和 list3 不相等。

```
int[] list1 = {2, 4, 7, 10};
int[] list2 = {2, 4, 7, 10};
int[] list3 = {4, 2, 7, 10};
System.out.println(java.util.Arrays.equals(list1, list2)); // true
System.out.println(java.util.Arrays.equals(list2, list3)); // false
```

可以使用 fill 方法填充整个数组或部分数组。例如: 下列代码将 5 填充到 list1 中, 将 8 填充到元素 list2[1] 到 list2[5-1] 中。

```
int[] list1 = {2, 4, 7, 10};
int[] list2 = {2, 4, 7, 7, 7, 10};
java.util.Arrays.fill(list1, 5); // Fill 5 to the whole array
java.util.Arrays.fill(list2, 1, 5, 8); // Fill 8 to a partial array
```

还可以使用 toString 方法来返回一个字符串, 该字符串代表了数组中的所有元素。这是一个显示数组中所有元素的快捷和简便的方法。例如, 下面代码

```
int[] list = {2, 4, 7, 10};
System.out.println(Arrays.toString(list));
```

显示 [2, 4, 7, 10]。

复习题

- 7.26 使用 java.util.Arrays.sort 方法可以对什么类型的数组进行排序? 这个 sort 方法会创建一个新数组吗?
- 7.27 为了应用 java.util.Arrays.binarySearch(array, key), 数组应按升序还是降序排列? 还是可以既非升序也非降序?
- 7.28 给出下面代码的输出结果。

```
int[] list1 = {2, 4, 7, 10};
java.util.Arrays.fill(list1, 7);
System.out.println(java.util.Arrays.toString(list1));
```

```
int[] list2 = {2, 4, 7, 10};
System.out.println(java.util.Arrays.toString(list2));
System.out.print(java.util.Arrays.equals(list1, list2));
```

7.13 命令行参数

要点提示：main 方法可以从命令行接收字符串参数。

你或许已经注意到 main 方法的声明与众不同，它具有 String[] 类型参数 args。很明显，参数 args 是一个字符串数组。main 方法就像一个带参数的普通方法。可以通过传递实参来调用一个普通方法。那能给 main 传递参数吗？当然可以。例如，在下面的示例中，TestMain 类中的 main 方法被 A 中的方法调用，如下所示：

```
public class A {
    public static void main(String[] args) {
        String[] strings = {"New York",
            "Boston", "Atlanta"};
        TestMain.main(strings);
    }
}
```

```
public class TestMain {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++)
            System.out.println(args[i]);
    }
}
```

main 方法就和普通方法一样。此外，还可以从命令行传送参数。

7.13.1 向 main 方法传递字符串

运行程序时，可以从命令行给 main 方法传递字符串参数。例如，下面的命令行用三个字符串 arg0、arg1、arg2 启动程序 TestMain：

```
java TestMain arg0 arg1 arg2
```

其中，参数 arg0、arg1 和 arg2 都是字符串，但是在命令行中出现时，不需要放在双引号中。这些字符串用空格分隔。如果字符串包含空格，那就必须使用双引号括住。考虑下面的命令行：

```
java TestMain "First num" alpha 53
```

使用三个字符串 "First num"、alpha 和 53 启动这个程序。因为 "First num" 是一个字符串，所以要用双括号括住它们。注意，53 实际上是当作字符串处理的。在命令行中可以使用 "53" 来代替 53。

当调用 main 方法时，Java 解释器会创建一个数组存储命令行参数，然后将该数组的引用传递给 args。例如，如果调用具有 n 个参数的程序，Java 解释器创建一个如下所示的数组：

```
args = new String[n];
```

然后，Java 解释器传递参数 args 去调用 main 方法。

注意：如果运行程序时没有传递字符串，那么使用 new String[0] 创建数组。在这种情况下，该数组是长度为 0 的空数组。args 是对这个空数组的引用。因此，args 不是 null，但是 args.length 是 0。

7.13.2 示例学习：计算器

假设要开发一个程序完成整型数的算术运算。程序接收三个参数：一个整数、紧随其后

的一个操作符以及另一个整数。例如，使用下面的命令对两个整数进行相加：

```
java Calculator 2 + 3
```

程序将显示下面的输出：

```
2 + 3 = 5
```

图 7-12 显示这个程序的运行示例。

传递给主程序的字符串存储在字符串数组 `args` 中。第一个字符串存储在 `arg[0]` 中，`args.length` 是传入的字符串个数。

下面是程序的步骤：

1) 利用 `args.length` 判断命令行是否提供了三个参数。如果没有，就使用 `System.exit(1)` 结束程序。

2) 运用 `args[1]` 中指定的操作符完成对操作数 `args[0]` 和 `args[2]` 的二元运算。

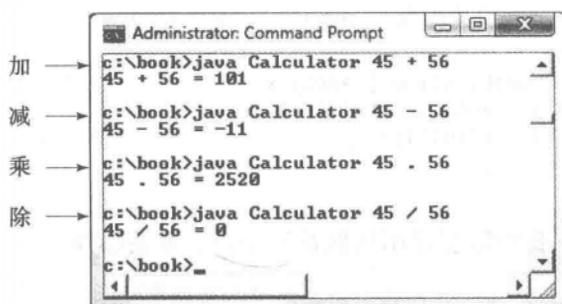


图 7-12 程序从命令行获取三个参数（操作数 1、操作符、操作数 2），然后显示这个算术运算表达式以及算术运算的结果

这个程序如程序清单 7-9 所示。

程序清单 7-9 Calculator.java

```

1 public class Calculator {
2     /** Main method */
3     public static void main(String[] args) {
4         // Check number of strings passed
5         if (args.length != 3) {
6             System.out.println(
7                 "Usage: java Calculator operand1 operator operand2");
8             System.exit(0);
9         }
10
11         // The result of the operation
12         int result = 0;
13
14         // Determine the operator
15         switch (args[1].charAt(0)) {
16             case '+': result = Integer.parseInt(args[0]) +
17                     Integer.parseInt(args[2]);
18                 break;
19             case '-': result = Integer.parseInt(args[0]) -
20                     Integer.parseInt(args[2]);
21                 break;
22             case '*': result = Integer.parseInt(args[0]) *

```

```

23         Integer.parseInt(args[2]);
24         break;
25     case '/': result = Integer.parseInt(args[0]) /
26         Integer.parseInt(args[2]);
27 }
28
29 // Display result
30 System.out.println(args[0] + ' ' + args[1] + ' ' + args[2]
31     + " = " + result);
32 }
33 }

```

`Integer.parseInt(args[0])` (第 16 行) 将一个数字字符串转换为一个整数。该字符串必须由数字构成, 否则, 程序会非正常中断。

我们使用 `.` 符号用于乘法, 而不是通常的 `*` 符号。原因是当符号 `*` 用于命令行时表示当前目录下的所有文件。在使用命令 `java Test *` 之后, 下面的程序就会显示当前目录下的所有文件:

```

public class Test {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++)
            System.out.println(args[i]);
    }
}

```

为了解决这个问题, 我们需要使用其他符号来用于乘法操作。

复习题

7.29 本书声明 `main` 方法为:

```
public static void main(String[] args)
```

它可以替换为下面行中的哪些呢?

```

public static void main(String args[])
public static void main(String[] x)
public static void main(String x[])
static void main(String x[])

```

7.30 给出使用下面命令调用时, 以下程序的输出。

1. `java Test I have a dream`
2. `java Test "1 2 3"`
3. `java Test`

```

public class Test {
    public static void main(String[] args) {
        System.out.println("Number of strings is " + args.length);
        for (int i = 0; i < args.length; i++)
            System.out.println(args[i]);
    }
}

```

关键术语

anonymous array (匿名数组)

array (数组)

array initializer (数组初始化语法)

binary search (二分查找)

garbage collection (垃圾回收)
index (下标)
indexed variable (下标变量)

linear search (线性查找)
off-by-one error (过一错误)
selection sort (选择排序)

本章小结

1. 使用语法 `elementType[] arrayRefVar` (元素类型[]数组引用变量) 或 `elementType arrayRefVar[]` (元素类型 数组引用变量[]) 声明一个数组类型的变量。尽管 `elementType arrayRefVar[]` 也是合法的, 但还是推荐使用 `elementType[] arrayRefVar` 风格。
2. 不同于基本数据类型变量的声明, 声明数组变量并不会给数组分配任何空间。数组变量不是基本数据类型变量。数组变量包含的是对数组的引用。
3. 只有创建数组后才能给数组元素赋值。可以使用 `new` 操作符创建数组, 语法如下: `new elementType[arraySize]` (数据类型 [数组大小])。
4. 数组中的每个元素都是使用语法 `arrayRefVar[index]` (数组引用变量 [下标]) 表示的。下标必须是一个整数或一个整数表达式。
5. 创建数组之后, 它的大小就不能改变, 可以使用 `arrayRefVar.length` 得到数组的大小。由于数组的下标总是从 0 开始, 所以, 最后一个下标总是 `arrayRefVar.length-1`。如果试图引用数组界外的元素, 就会发生越界错误。
6. 程序员经常会错误地用下标 1 访问数组的第一个元素, 但是, 实际上这个元素的下标应该是 0。这个错误称为下标过 1 错误 (index off-by-one error)。
7. 当创建一个数组时, 如果其中的元素的基本数据类型是数值型, 那么赋默认值 0。字符类型的默认值为 `'\u0000'`, 布尔类型的默认值为 `false`。
8. Java 有一个称为数组初始化语法 (array initializer) 的简捷表达式, 它将数组的声明、创建和初始化合并为一条语句, 其语法为:
元素类型 [] 数组引用变量 = {value0,value1,...,valuek}
9. 将数组参数传递给方法时, 实际上传递的是数组的引用; 更准确地说, 被调用的方法可以修改调用者的原始数组的元素。
10. 如果数组是排好序的, 对于查找数组中的一个元素而言, 二分查找比线性查找更加高效。
11. 选择排序找到列表中最小的数字, 并将其和第一个数字交换。然后在剩下的数字中找到最小的, 和剩下列表的第一个元素交换, 继续这个步骤, 直到列表中只剩下一个数字。

测试题

在线回答本章节的测试题, 位于 www.cs.armstrong.edu/liang/intro10e/quiz.html。

编程练习题

7.2 ~ 7.5 节

- *7.1 (指定等级) 编写一个程序, 读入学生成绩, 获取最高分 `best`, 然后根据下面的规则赋等级值:
- 如果分数 $\geq best-10$, 等级为 A
 - 如果分数 $\geq best-20$, 等级为 B
 - 如果分数 $\geq best-30$, 等级为 C
 - 如果分数 $\geq best-40$, 等级为 D
 - 其他情况下, 等级为 F

程序提示用户输入学生总数，然后提示用户输入所有的分数，最后显示等级得出结论。下面是一个运行示例：

```
Enter the number of students: 4 ↵
Enter 4 scores: 40 55 70 58 ↵
Student 0 score is 40 and grade is C
Student 1 score is 55 and grade is B
Student 2 score is 70 and grade is A
Student 3 score is 58 and grade is B
```

7.2 (倒置输入的数) 编写程序，读取 10 个整数，然后按照和读入顺序相反的顺序将它们显示出来。

**7.3 (计算数字的出现次数) 编写程序，读取在 1 到 100 之间的整数，然后计算每个数出现的次数。假定输入是以 0 结束的。下面是这个程序的一个运行示例：

```
Enter the integers between 1 and 100: 2 5 6 5 4 3 23 43 2 0 ↵
2 occurs 2 times
3 occurs 1 time
4 occurs 1 time
5 occurs 2 times
6 occurs 1 time
23 occurs 1 time
43 occurs 1 time
```

🔑 注意：如果一个数出现的次数大于一次，就在输出时使用复数“times”。

7.4 (分析成绩) 编写一个程序，读入个数不确定的考试分数，并且判断有多少个分数是大于或等于平均分，多少个分数是低于平均分的。输入一个负数表示输入的结束。假设最高分为 100。

**7.5 (打印不同的数) 编写一个程序，读入 10 个数并且显示互不相同的数（即一个数出现多次，但仅显示一次）。（提示，读入一个数，如果它是一个新数，则将它存储在数组中。如果该数已经在数组中，则忽略它。）输入之后，数组包含的都是不同的数。下面是这个程序的运行示例：

```
Enter ten numbers: 1 2 3 2 1 6 3 4 5 2 ↵
The number of distinct number is 6
The distinct numbers are: 1 2 3 6 4 5
```

*7.6 (修改程序清单 5-15) 程序清单 5-15 通过检验 $2, 3, 4, 5, 6, \dots, n/2$ 是否是数 n 的因子来判断 n 是否是素数。如果找到一个因子， n 就不是素数。判断 n 是否素数的另一个更有效的方法是：检验小于等于 \sqrt{n} 的素数是否都能整除 n 。如果不能，则 n 就是素数。使用这个方法改写程序清单 5-15 以显示前 50 个素数。需要使用一个数组存储这些素数，然后再检查它们是否是 n 的可能的因子。

*7.7 (统计一位数的个数) 编写一个程序，生成 0 和 9 之间的 100 个随机整数，然后显示每一个数出现的次数。

🔑 提示：使用 `(int)(Math.random()*10)` 产生 0 到 9 之间的随机整数。使用一个名为 `counts` 的由 10 个整数构成的数组存放 0, 1, ..., 9 的个数。

7.6 ~ 7.8 节

7.8 (求数组的平均值) 编写两个重载的方法，使用下面的方法头返回一个数组的平均数：

```
public static int average(int[] array)
public static double average(double[] array)
```

编写测试程序，提示用户输入 10 个 `double` 型值，调用这个方法，然后显示平均值。

7.9 (找出最小元素) 编写一个方法，使用下面的方法头求出一个整数数组中的最小元素：

```
public static double min(double[] array)
```

编写测试程序，提示用户输入十个数字，调用这个方法返回最小值，显示其最小值。下面是

该程序的运行示例：

```
Enter ten numbers: 1.9 2.5 3.7 2 1.5 6 3 4 5 2
The minimum number is: 1.5
```

- 7.10 (找出最小元素的下标) 编写一个方法，求出整数数组中最小元素的下标。如果这样的元素个数大于1，则返回最小的下标。使用下面的方法头：

```
public static int indexOfSmallestElement(double[] array)
```

编写测试程序，提示用户输入10个数字，调用这个方法，返回最小元素的下标，然后显示这个下标值。

- *7.11 (统计学方面：计算标准差) 编程练习题 5.45 计算数字的标准差。本题使用一个和它不同但等价的公式来计算 n 个数的标准差。

$$\text{平均值} = \frac{\sum_{i=1}^n x_i}{n} = \frac{x_1 + x_2 + \dots + x_n}{n} \quad \text{标准差} = \sqrt{\frac{\sum_{i=1}^n (x_i - \text{mean})^2}{n-1}}$$

要用这个公式计算标准差，必须使用一个数组存储每一个数，因此，可以在获取平均值后使用它们。

程序应该包含下面的方法：

```
/** Compute the deviation of double values */
public static double deviation(double[] x)

/** Compute the mean of an array of double values */
public static double mean(double[] x)
```

编写测试程序，提示用户输入10个数字，然后显示平均值和标准差，如下面的运行示例所示：

```
Enter ten numbers: 1.9 2.5 3.7 2 1.6 3 4 5 2
The mean is 3.11
The standard deviation is 1.55738
```

- *7.12 (倒置数组) 7.7 节中的 `reverse` 方法通过把数组复制到新数组中实现数组的倒置。改写方法将传递到实参的数组倒置，然后返回这个数组。编写一个测试程序，提示用户输入十个数字，调用这个方法倒置这些数字，然后显示这些数字。

7.9 节

- *7.13 (随机数选择器) 编写一个方法，返回1到54之间的随机数，不包括传递到参数中的 `numbers`。如下指定这个方法头：

```
public static int getRandom(int... numbers)
```

- 7.14 (计算 gcd) 编写一个方法，返回个数不确定的整数的最大公约数。指定这个方法头如下所示：

```
public static int gcd(int... numbers)
```

编写测试程序，提示用户输入5个数字，调用该方法找出这些数的最大公约数，并显示这个最大公约数。

7.10 ~ 7.12 节

- 7.15 (消除重复) 使用下面的方法头编写方法，消除数组中重复出现的值：

```
public static int[] eliminateDuplicates(int[] list)
```

编写一个测试程序，读取10个整数，调用该方法，然后显示结果。下面是程序的运行示例：

```
Enter ten numbers: 1 2 3 2 1 6 3 4 5 2 --Enter
The distinct numbers are: 1 2 3 6 4 5
```

7.16 (执行时间) 编写程序, 随机产生 100 000 个整数值和一个关键字。估算一下调用程序清单 7-6 中的 `linearSearch` 方法的执行时间。对该数组进行排序, 然后估算调用程序清单 7-7 中的 `binarySearch` 方法的执行时间。可以使用下面的代码模板获取执行时间:

```
long startTime = System.currentTimeMillis();
perform the task;
long endTime = System.currentTimeMillis();
long executionTime = endTime - startTime;
```

**7.17 (对学生排序) 编写一个程序, 提示用户输入学生个数、学生姓名和他们的成绩, 然后按照学生成绩的降序打印学生的姓名。

**7.18 (冒泡排序) 使用冒泡排序算法编写一个排序方法。冒泡排序算法遍历数组几次。在每次遍历中, 对相邻的两个元素进行比较。如果这一对元素是降序, 则交换它们的值; 否则, 保持值不变。由于较小的值像气泡一样逐渐“浮向”顶部, 同时较大的值“沉向”底部, 所以, 这种技术称为冒泡排序法 (bubble sort) 或下沉排序法 (sinking sort)。编写一个测试程序, 读取 10 个 `double` 型的值, 调用这个方法, 然后显示排好序的数字。

**7.19 (是否排好序了?) 编写以下方法, 如果参数中的 `list` 数组已经按照升序排好了, 则返回 `true`。

```
public static boolean isSorted(int[] list)
```

编写一个测试程序, 提示用户输入一个列表, 显示该列表是否已经排好序。下面是一个运行示例。注意, 输入中的第一个数表示列表中的元素个数。该数不是列表的一部分。

```
Enter list: 8 10 1 5 16 61 9 11 1 --Enter
The list is not sorted
```

```
Enter list: 10 1 1 3 4 4 5 7 9 11 21 --Enter
The list is already sorted
```

*7.20 (修改选择排序法) 在 7.11 节中, 使用的是选择排序法对数组排序。选择排序法重复地在当前数组中找到最小值, 然后将这个最小值与该数组中的第一个数进行交换。改写这个程序, 重复地在当前数组中找到最大值, 然后将这个最大值与该数组中的最后一个数进行交换。编写测试程序, 读取 10 个 `double` 型的数字, 调用该方法, 然后显示排好序的数字。

***7.21 (游戏: 豆机) 豆机, 也称为梅花瓶或高尔顿瓶, 它是一个用来做统计实验的设备, 是用英国科学家瑟弗兰克斯高尔顿的名字来命名的。它是一个三角形形状的均匀放置钉子 (或钩子) 的直立板子, 如图 7-13 所示。

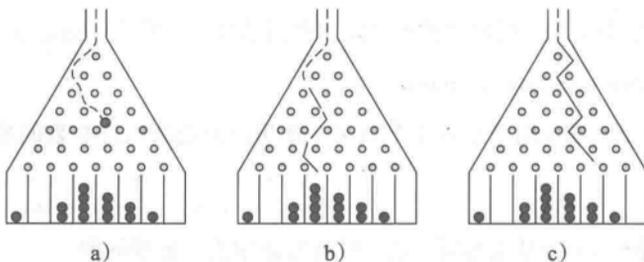


图 7-13 每个球都选取一个随机路径, 然后掉入一个槽中

球都是从板子口落下的。每当球碰到钉子, 它就有 50% 的机会落向左边或落向右边。在板子底部的槽子中都会累积一堆球。

编写程序模拟豆机。程序应该提示用户输入球的个数以及机器的槽数。打印每个球的路径模拟它的下落。例如：在图 7-13b 中球的路径是 LLRLLR，而在图 7-13c 中球的路径是 RLRRLRR。使用条形图显示槽中球的最终储备量。下面是程序的一个运行示例：

```

Enter the number of balls to drop: 5
Enter the number of slots in the bean machine: 8

LRLRLRR
RRLLLRR
LLRLLRR
RRLLLLL
LRLRRLR

  0
  0
 000
  
```

提示：创建一个名为 `slots` 的数组。数组 `slots` 中的每个元素存储的是一个槽中球的个数。每个球都经过一条路径落入一个槽中。路径上 R 的个数表示球落下的槽的位置。例如：对于路径 LRLRLRR 而言，球落到 `slots[4]` 中，而对路径 RRLLLLL 而言，球落到 `slots[2]` 中。

****7.22 (游戏：八皇后)** 经典的八皇后难题是要将八个皇后放在棋盘上，任何两个皇后都不能互相攻击（即没有两个皇后是在同一行、同一列或者同一对角上）。可能的解决方案有很多。编写程序显示一个这样的解决方案。一个示例输出如右图所示。

```

|Q| | | | | | | |
| | | | |Q| | | |
| | | | | |Q| | |
| | |Q| | | | | |
|Q| | | | | | | |
| | | |Q| | | | |
| | | | | | | |Q|
| | | | | | | | |
  
```

****7.23 (游戏：储物柜难题)** 一个学校有 100 个储物柜和 100 个学生。所有的储物柜在上学第一天都是关着的。随着学生进来，第一个学生（用 S1 表示）打开每个柜子。然后，第二个学生（用 S2 表示）从第二个柜子（用 L2 表示）开始，关闭相隔为 1 的柜子。学生 S3 从第三个柜子开始，然后改变每个第三个柜子（如果是开的就关上，如果是关的就打开）。学生 S4 从柜子 L4 开始，然后改变每个第四个柜子的开闭状态。学生 S5 从 L5 开始，然后改变每个第五个柜子的状态，依此类推，直到学生 S100 改变 L100 为止。

在所有学生都经过教学楼并且改变了柜子之后，哪些柜子是开的？编写程序找出答案。

提示：使用存放 100 个布尔型元素的数组，每个元素都表明一个柜子是开的 (`true`) 还是关的 (`false`)。初始状态时，所有的柜子都是关的。

****7.24 (仿真：优惠券收集人问题)** 优惠券收集人问题是一个经典的统计问题，它有很多实际应用。这个问题重复地从一套对象中拿出一个对象，然后找出要将所有需要拿出的对象都至少拿出来一次，需要拿多少次。从该问题衍生出的类似问题就是，从一副打乱的 52 张牌中重复选牌，找出在看到每种花色都有一张出现前，需要选多少次。假设在选下一张牌之前的那张牌是背面向上的。编写程序，模拟要得到四张不同花色的牌所需要的选取次数，然后显示选中的四张牌（有可能一张牌被选了两次）。下面是这个程序的运行示例：

```

Queen of Spades
5 of Clubs
Queen of Hearts
4 of Diamonds
Number of picks: 12
  
```

7.25 (代数问题：解二次方程式) 使用下面的方法头编写一个解二次方程式的方法：

```
public static int solveQuadratic(double[] eqn, double[] roots)
```

二次方程式 $ax^2+bx+c=0$ 的系数都传给数组 `eqn`，然后将两个非复数的根存在 `roots` 里。方法返回根的个数。参见编程练习题 3.1 了解如何解二次方程。

编写程序，提示用户输入 a 、 b 和 c 的值，然后显示实数根的个数以及所有的实数根。

- 7.26 (完全相同的数组) 如果两个数组 `list1` 和 `list2` 的长度相同，而且对于每个 i ，`list1[i]` 都等于 `list2[i]`，那么认为 `list1` 和 `list2` 是完全相同的。使用下面的方法头编写一个方法，如果 `list1` 和 `list2` 完全相同，那么这个方法返回 `true`：

```
public static boolean equals(int[] list1, int[] list2)
```

编写一个测试程序，提示用户输入两个整数列表，然后显示这两个列表是否完全相同。下面是运行示例。注意，输入的的第一个数字表明列表中元素的个数。该数字不是列表的一部分。

```
Enter list1: 5 2 5 6 1 6 ↵ Enter
Enter list2: 5 2 5 6 1 6 ↵ Enter
Two lists are strictly identical
```

```
Enter list1: 5 2 5 6 6 1 ↵ Enter
Enter list2: 5 2 5 6 1 6 ↵ Enter
Two lists are not strictly identical
```

- 7.27 (相同的数组) 如果两个数组 `list1` 和 `list2` 的内容相同，那么就说明它们是相同的。使用下面的方法头编写一个方法，如果 `list1` 和 `list2` 是相同的，该方法就返回 `true`：

```
public static boolean equals(int[] list1, int[] list2)
```

编写一个测试程序，提示用户输入两个整数列表，然后显示它们两个是否相同。下面是运行示例。注意，输入的的第一个数字表示列表中元素的个数。该数字不是列表的一部分。

```
Enter list1: 5 2 5 6 6 1 ↵ Enter
Enter list2: 5 5 2 6 1 6 ↵ Enter
Two lists are identical
```

```
Enter list1: 5 5 5 6 6 1 ↵ Enter
Enter list2: 5 2 5 6 1 6 ↵ Enter
Two lists are not identical
```

- *7.28 (数学方面：组合) 编写一个程序，提示用户输入 10 个整数，然后显示从这 10 个数中选出两个数的所有组合。
- *7.29 (游戏：选出四张牌) 编写一个程序，从一副 52 张的牌中选出四张，然后计算它们的和。Ace、King、Queen 和 Jack 分别表示 1、13、12 和 11。程序应该显示得到的和为 24 的选牌次数。
- *7.30 (模式识别方面：四个连续相等的数) 编写下面的方法，测试某个数组是否有四个连续的值相同的数字。

```
public static boolean isConsecutiveFour(int[] values)
```

编写测试程序，提示用户输入一个整数列表，如果这个列表中有四个连续的具有相同值的数，那就显示 `true`；否则，显示 `false`。程序应该首先提示用户键入输入的大小，即列表中值的个数。这里是一个运行示例。

```
Enter the number of values: 8 ↵ Enter
Enter the values: 3 4 5 5 5 4 5 ↵ Enter
The list has consecutive fours
```

```
Enter the number of values: 9 ↵ Enter
Enter the values: 3 4 5 5 6 5 5 4 5 ↵ Enter
The list has no consecutive fours
```

**7.31 (合并两个有序列表) 编写下面的方法, 将两个有序列表合并成一个新的有序列表。

```
public static int[] merge(int[] list1, int[] list2)
```

只进行 `list1.length+list2.length` 次比较来实现该方法。编写一个测试程序, 提示用户输入两个有序列表, 然后显示合并的列表。下面是一个运行示例。注意, 输入的的第一个数字表示列表中元素的个数。该数字不是列表的一部分。

```
Enter list1: 5 1 5 16 61 111
Enter list2: 4 2 4 5 6
The merged list is 1 2 4 5 5 6 16 61 111
```

**7.32 (划分列表) 编写以下方法, 使用第一个元素对列表进行划分, 该元素称为支点。

```
public static int partition(int[] list)
```

划分后, 列表中的元素被重新安排, 在支点元素之前的元素都小于或者等于该元素, 而之后的元素都大于该元素。方法返回支点元素位于新列表中的下标。例如, 假设列表是 `[5,2,9,3,8]`, 划分后, 列表变为 `[3,2,5,9,6,8]`。最多进行 `list.length` 次比较来实现该方法。编写一个测试程序, 提示用户输入一个列表, 然后显示划分后的列表。下面是一个运行示例。注意, 输入的的第一个数字表示列表中元素的个数。该数字不是列表的一部分。

```
Enter list: 8 10 1 5 16 61 9 11 1
After the partition, the list is 9 1 5 1 10 61 11 16
```

*7.33 (文化: 中国生肖) 使用一个字符串数组存储动物名称, 来简化程序清单 3-9 的程序。

**7.34 (对字符串中的字符排序) 使用以下方法头编写一个方法, 返回一个排好序的字符串。

```
public static String sort(String s)
```

例如, `sort("acb")` 返回 `abc`。编写一个测试程序, 提示用户输入一个字符串, 显示排好序的字符串。

**7.35 (游戏: 猜字游戏) 编写一个猜字游戏。随机产生一个单词, 提示用户一次猜测一个字母, 如运行示例所示。单词中的每个字母显示为一个星号。当用户猜测正确后, 正确的字母显示出来。当用户猜出一个单词, 显示猜错的次数, 并且询问用户是否继续对另外一个单词进行游戏。声明一个数组来存储单词, 如下所示:

```
// Add any words you wish in this array
String[] words = {"write", "that", ...};
```

```
(Guess) Enter a letter in word ***** > p
(Guess) Enter a letter in word p***** > r
(Guess) Enter a letter in word pr**r** > p
    p is already in the word
(Guess) Enter a letter in word pr**r** > o
(Guess) Enter a letter in word pro*r** > g
(Guess) Enter a letter in word progr** > n
    n is not in the word
(Guess) Enter a letter in word progr** > m
(Guess) Enter a letter in word progr*m > a
The word is program. You missed 1 time
Do you want to guess another word? Enter y or n>
```

多维数组

教学目标

- 给出使用二维数组表示数据的例子(8.1节)。
- 声明二维数组变量、创建数组,以及使用行下标和列下标访问二维数组中的数组元素(8.2节)。
- 编程实现常用的二维数组的操作(显示数组、对所有元素求和、找出最小元素和最大元素以及随意打乱数组)(8.3节)。
- 传递二维数组给方法(8.4节)。
- 使用二维数组编写多选题评分程序(8.5节)。
- 使用二维数组解决距离最近的点对问题(8.6节)。
- 使用二维数组检测一种九宫格的解决方案(8.7节)。
- 使用多维数组(8.8节)。

8.1 引言

 **要点提示:** 表格或矩阵中的数据可以表示为二维数组。

前一章中介绍过一维数组如何存储线性的元素集合。可以使用二维数组存储矩阵或表格。例如,使用命名为 `distances` 的二维数组就可以存储下面这个描述城市之间距离的表格。

距离表(以公里为单位)

	芝加哥	波士顿	纽约	亚特兰大	迈阿密	达拉斯	休斯敦
芝加哥	0	983	787	714	1375	967	1087
波士顿	983	0	214	1102	1763	1723	1842
纽约	787	214	0	888	1549	1548	1627
亚特兰大	714	1102	888	0	661	781	810
迈阿密	1375	1763	1549	661	0	1426	1187
达拉斯	967	1723	1548	781	1426	0	239
休斯敦	1087	1842	1627	810	1187	239	0

```
double[][] distances = {
    {0, 983, 787, 714, 1375, 967, 1087},
    {983, 0, 214, 1102, 1763, 1723, 1842},
    {787, 214, 0, 888, 1549, 1548, 1627},
    {714, 1102, 888, 0, 661, 781, 810},
    {1375, 1763, 1549, 661, 0, 1426, 1187},
    {967, 1723, 1548, 781, 1426, 0, 239},
    {1087, 1842, 1627, 810, 1187, 239, 0},
};
```

8.2 二维数组的基础知识

 **要点提示:** 二维数组中的元素通过行和列的下标来访问。

如何声明一个二维数组变量？如何创建一个二维数组？如何访问二维数组中的元素？本节将解决这些问题。

8.2.1 声明二维数组变量并创建二维数组

下面是声明二维数组的语法：

数据类型 [][] 数组名；

或者

数据类型 数组名 [][]； // 允许这种方式，但并不推荐使用它

作为例子，下面演示如何声明 int 型的二维数组变量 matrix：

```
int[][] matrix;
```

或者

```
int matrix[][]; // 允许这种方式，但并不推荐使用它
```

可以使用这个语法创建 5 × 5 的 int 型二维数组，并将它赋值给 matrix：

```
matrix = new int[5][5];
```

二维数组中使用两个下标，一个表示行，另一个表示列。同一维数组一样，每个下标索引值都是 int 型的，从 0 开始，如图 8-1a 所示。

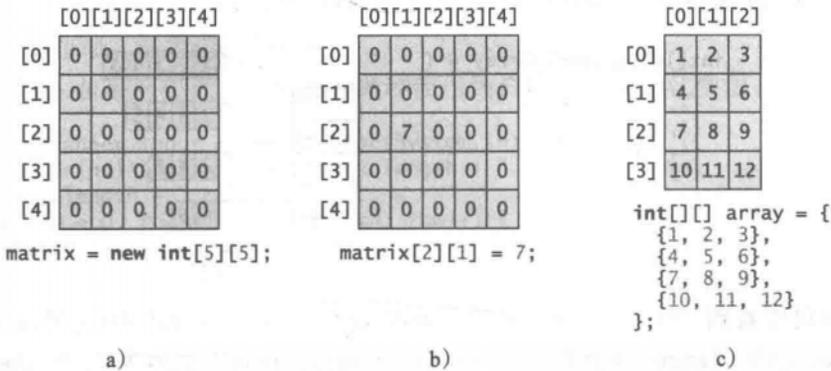


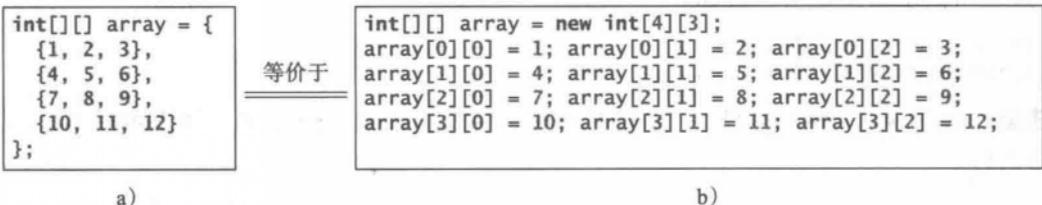
图 8-1 二维数组的每个下标索引值都是从 0 开始的 int 值

如图 8-1b 所示，要将 7 赋值给行下标为 2、列下标为 1 的特定元素，可以使用下面的语句：

```
matrix[2][1] = 7;
```

警告：使用 matrix[2,1] 访问行下标为 2、列下标为 1 的元素是一种常见错误。在 Java 中，每个下标必须放在一对方括号中。

也可以使用数组初始化来声明、创建和初始化一个二维数组。例如：下图 a 中的代码创建一个具有特定初值的数组，如图 8-1c 所示。它和图 b 中的代码是等价的。



8.2.2 获取二维数组的长度

二维数组实际上是一个数组，它的每个元素都是一个一维数组。数组 x 的长度是数组中元素的个数，可以用 $x.length$ 获取该值。元素 $x[0]$, $x[1]$, ..., $x[x.length-1]$ 也是数组。可以使用 $x[0].length$, $x[1].length$, ..., $x[x.length-1].length$ 获取它们的长度。

例如：假设 $x = \text{new int}[3][4]$ ，那么 $x[0]$ 、 $x[1]$ 和 $x[2]$ 都是一维数组，每个数组都包含 4 个元素，如图 8-2 所示。 $x.length$ 为 3， $x[0].length$ 、 $x[1].length$ 和 $x[2].length$ 都是 4。

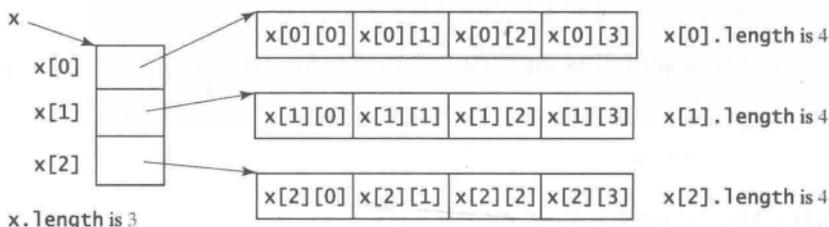
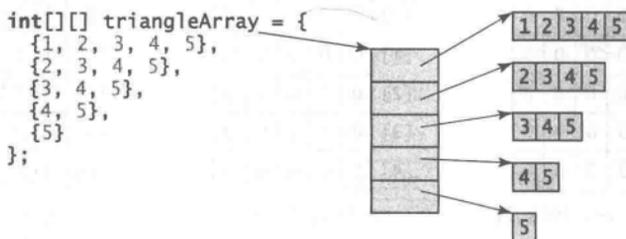


图 8-2 二维数组是一个一维数组，它的每个元素是另一个一维数组

8.2.3 锯齿数组

二维数组中的每一行本身就是一个数组，因此，各行的长度就可以不同。这样的数组称为锯齿数组 (ragged array)。下面就是一个创建锯齿数组的例子：



从上图中可以看到， $\text{triangleArray}[0].length$ 的值为 5， $\text{triangleArray}[1].length$ 的值为 4， $\text{triangleArray}[2].length$ 的值为 3， $\text{triangleArray}[3].length$ 的值为 2， $\text{triangleArray}[4].length$ 的值为 1。

如果事先不知道锯齿数组的值，但知道它的长度，正如前面讲到的，可以使用如下所示的语法创建锯齿数组：

```
int[][] triangleArray = new int[5][];
triangleArray[0] = new int[5];
triangleArray[1] = new int[4];
triangleArray[2] = new int[3];
triangleArray[3] = new int[2];
triangleArray[4] = new int[1];
```

现在可以给数组赋值，例如：

```
triangleArray[0][3] = 50;
triangleArray[4][0] = 45;
```

注意：使用语法 $\text{new int}[5][\]$ 创建数组时，必须指定第一个下标。语法 $\text{new int}[\][\]$ 是错误的。

复习题

- 8.1 为一个 4×5 的整型矩阵声明一个数组引用变量，创建该矩阵，并将其赋值给数组引用变量。
- 8.2 二维数组的行可以有不同的长度吗？
- 8.3 以下代码的输出是什么？

```
int[][] array = new int[5][6];
int[] x = {1, 2};
array[0] = x;
System.out.println("array[0][1] is " + array[0][1]);
```

- 8.4 以下哪些语句是合法的？

```
int[][] r = new int[2];
int[] x = new int[];
int[][] y = new int[3][];
int[][] z = {{1, 2}};
int[][] m = {{1, 2}, {2, 3}};
int[][] n = {{1, 2}, {2, 3}, };
```

8.3 处理二维数组

要点提示：嵌套的 for 循环常用于处理二维数组。

假设如下创建数组 matrix：

```
int[][] matrix = new int[10][10];
```

下面是一些处理二维数组的例子：

1) (使用输入值初始化数组) 下面的循环使用用户输入值初始化数组：

```
java.util.Scanner input = new Scanner(System.in);
System.out.println("Enter " + matrix.length + " rows and " +
    matrix[0].length + " columns: ");
for (int row = 0; row < matrix.length; row++) {
    for (int column = 0; column < matrix[row].length; column++) {
        matrix[row][column] = input.nextInt();
    }
}
```

2) (使用随机值初始化数组) 下面的循环使用 0 到 99 之间的随机值初始化数组：

```
for (int row = 0; row < matrix.length; row++) {
    for (int column = 0; column < matrix[row].length; column++) {
        matrix[row][column] = (int)(Math.random() * 100);
    }
}
```

3) (打印数组) 为打印一个二维数组，必须使用如下所示的循环打印数组中的每个元素：

```
for (int row = 0; row < matrix.length; row++) {
    for (int column = 0; column < matrix[row].length; column++) {
        System.out.print(matrix[row][column] + " ");
    }
    System.out.println();
}
```

4) (求所有元素的和) 使用名为 total 的变量存储和。将 total 初始化为 0。利用类似下面的循环，把数组中的每一个元素都加到 total 上：

```
int total = 0;
for (int row = 0; row < matrix.length; row++) {
    for (int column = 0; column < matrix[row].length; column++) {
        total += matrix[row][column];
    }
}
```

5) (对数组按列求和) 对于每一列, 使用名为 `total` 的变量存储它的和。利用类似下面的循环, 将该列中的每个元素加到 `total` 上:

```
for (int column = 0; column < matrix[0].length; column++) {
    int total = 0;
    for (int row = 0; row < matrix.length; row++)
        total += matrix[row][column];
    System.out.println("Sum for column " + column + " is "
        + total);
}
```

6) (哪一行的和最大?) 使用变量 `maxRow` 和 `indexOfMaxRow` 分别跟踪和的最大值以及该行的索引值。计算每一行的和, 如果计算出的新行的和更大, 就更新 `maxRow` 和 `indexOfMaxRow`。

```
int maxRow = 0;
int indexOfMaxRow = 0;

// Get sum of the first row in maxRow
for (int column = 0; column < matrix[0].length; column++) {
    maxRow += matrix[0][column];
}

for (int row = 1; row < matrix.length; row++) {
    int totalOfThisRow = 0;
    for (int column = 0; column < matrix[row].length; column++)
        totalOfThisRow += matrix[row][column];

    if (totalOfThisRow > maxRow) {
        maxRow = totalOfThisRow;
        indexOfMaxRow = row;
    }
}

System.out.println("Row " + indexOfMaxRow
    + " has the maximum sum of " + maxRow);
```

7) (随意打乱) 在 7.2.6 节中已经介绍了如何打乱一维数组的元素, 那么如何打乱二维数组中的所有元素呢? 为了实现这个功能, 对每个元素 `matrix[i][j]`, 随机产生下标 `i1` 和 `j1`, 然后互换 `matrix[i][j]` 和 `matrix[i1][j1]`, 如下所示:

```
for (int i = 0; i < matrix.length; i++) {
    for (int j = 0; j < matrix[i].length; j++) {
        int i1 = (int)(Math.random() * matrix.length);
        int j1 = (int)(Math.random() * matrix[i].length);

        // Swap matrix[i][j] with matrix[i1][j1]
        int temp = matrix[i][j];
        matrix[i][j] = matrix[i1][j1];
        matrix[i1][j1] = temp;
    }
}
```

复习题

8.5 给出下面代码的输出:

```
int[][] array = {{1, 2}, {3, 4}, {5, 6}};
for (int i = array.length - 1; i >= 0; i--) {
    for (int j = array[i].length - 1; j >= 0; j--)
        System.out.print(array[i][j] + " ");
    System.out.println();
}
```

8.6 给出下面代码的输出:

```
int[][] array = {{1, 2}, {3, 4}, {5, 6}};
int sum = 0;
for (int i = 0; i < array.length; i++)
    sum += array[i][0];
System.out.println(sum);
```

8.4 将二维数组传递给方法

要点提示: 将一个二维数组传递给方法的时候, 数组的引用传递给了方法。

可以像传递一维数组一样, 给方法传递二维数组。也可以从一个方法返回一个数组。程序清单 8-1 给出一个具有两个方法的示例。第一个方法, `getArray()`, 返回一个二维数组; 第二个方法, `sum(int[][] m)`, 返回一个矩阵中所有元素的和。

程序清单 8-1 PassTwoDimensionalArray.java

```
1 import java.util.Scanner;
2
3 public class PassTwoDimensionalArray {
4     public static void main(String[] args) {
5         int[][] m = getArray(); // Get an array
6
7         // Display sum of elements
8         System.out.println("\nSum of all elements is " + sum(m));
9     }
10
11     public static int[][] getArray() {
12         // Create a Scanner
13         Scanner input = new Scanner(System.in);
14
15         // Enter array values
16         int[][] m = new int[3][4];
17         System.out.println("Enter " + m.length + " rows and "
18             + m[0].length + " columns: ");
19         for (int i = 0; i < m.length; i++)
20             for (int j = 0; j < m[i].length; j++)
21                 m[i][j] = input.nextInt();
22
23         return m;
24     }
25
26     public static int sum(int[][] m) {
27         int total = 0;
28         for (int row = 0; row < m.length; row++) {
29             for (int column = 0; column < m[row].length; column++) {
30                 total += m[row][column];
31             }
32         }
33
34         return total;
35     }
36 }
```

```
Enter 3 rows and 4 columns:
```

```
1 2 3 4 [Enter]
```

```
5 6 7 8 [Enter]
```

```
9 10 11 12 [Enter]
```

```
Sum of all elements is 78
```

方法 `getArray` 提示用户为数组输入值 (第 11 ~ 24 行) 并且返回该数组 (第 23 行)。

方法 `sum` (第 26 ~ 35 行) 有一个二维数组参数。可以使用 `m.length` 获取行数 (第 28 行), 而使用 `m[row].column` 得到特定行的列数 (第 29 行)。

复习题

8.7 给出下面代码的输出:

```
public class Test {
    public static void main(String[] args) {
        int[][] array = {{1, 2, 3, 4}, {5, 6, 7, 8}};
        System.out.println(m1(array)[0]);
        System.out.println(m1(array)[1]);
    }

    public static int[] m1(int[][] m) {
        int[] result = new int[2];
        result[0] = m.length;
        result[1] = m[0].length;
        return result;
    }
}
```

8.5 示例学习: 多选题测验评分

要点提示: 编写一个可以进行多选题测验评分的程序。

本节介绍的问题是编写一个程序, 对多选题测验进行评分。假设这里有 8 个学生和 10 道题目, 学生的答案存储在一个二维数组中。每一行记录一名学生对所有题目的答案, 如下面数组所示:

		学生对问题的回答									
		0	1	2	3	4	5	6	7	8	9
Student 0		A	B	A	C	C	D	E	E	A	D
Student 1		D	B	A	B	C	A	E	E	A	D
Student 2		E	D	D	A	C	B	E	E	A	D
Student 3		C	B	A	E	D	C	E	E	A	D
Student 4		A	B	D	C	C	D	E	E	A	D
Student 5		B	B	E	C	C	D	E	E	A	D
Student 6		B	B	A	C	C	D	E	E	A	D
Student 7		E	B	E	C	C	D	E	E	A	D

正确答案存储在一个一维数组中:

```
问题的正确答案
0 1 2 3 4 5 6 7 8 9
Key D B D C C D A E A D
```

程序给测验评分并显示结果。它将每个学生的答案与正确答案进行比较, 统计正确答案的个数, 并将其显示出来。程序清单 8-2 给出该程序。

程序清单 8-2 GradeExam.java

```
1 public class GradeExam {
2     /** Main method */
3     public static void main(String[] args) {
4         // Students' answers to the questions
5         char[][] answers = {
6             {'A', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
7             {'D', 'B', 'A', 'B', 'C', 'A', 'E', 'E', 'A', 'D'},
8             {'E', 'D', 'D', 'A', 'C', 'B', 'E', 'E', 'A', 'D'},
9             {'C', 'B', 'A', 'E', 'D', 'C', 'E', 'E', 'A', 'D'},
10            {'A', 'B', 'D', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
11            {'B', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
12            {'B', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
13            {'E', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A', 'D'};
14
15        // Key to the questions
16        char[] keys = {'D', 'B', 'D', 'C', 'C', 'D', 'A', 'E', 'A', 'D'};
17
18        // Grade all answers
19        for (int i = 0; i < answers.length; i++) {
20            // Grade one student
21            int correctCount = 0;
22            for (int j = 0; j < answers[i].length; j++) {
23                if (answers[i][j] == keys[j])
24                    correctCount++;
25            }
26
27            System.out.println("Student " + i + "'s correct count is " +
28                correctCount);
29        }
30    }
31 }
```

```
Student 0's correct count is 7
Student 1's correct count is 6
Student 2's correct count is 5
Student 3's correct count is 4
Student 4's correct count is 8
Student 5's correct count is 7
Student 6's correct count is 7
Student 7's correct count is 7
```

第5~13行的语句声明、创建和初始化一个二维字符数组，并将它的引用赋值给char[][]型变量answers。

第16行的语句声明、创建和初始化一个char值构成的数组，并将其引用赋值给char[]型变量keys。

数组answers的每一行存储一个学生的答案，将它与数组keys中的正确答案比较之后进行评分。给一个学生评完分数后就立刻将结果显示出来。

8.6 示例学习：找出距离最近的点对

 **要点提示：**本节提供一个几何问题的解决——找到距离最近的点对。

假设有一个集合的点，找出最接近的点对问题就是找到两个点，它们到彼此的距离最近。例如：在图8-3中，点(1,1)和(2,0.5)是彼此之间距离最近的一对点。解决这个问题的方法有好几种。一种直观的方法就是计算所有点对之间的距离，并且找出最短的距离，它

的实现如程序清单 8-3 所示。

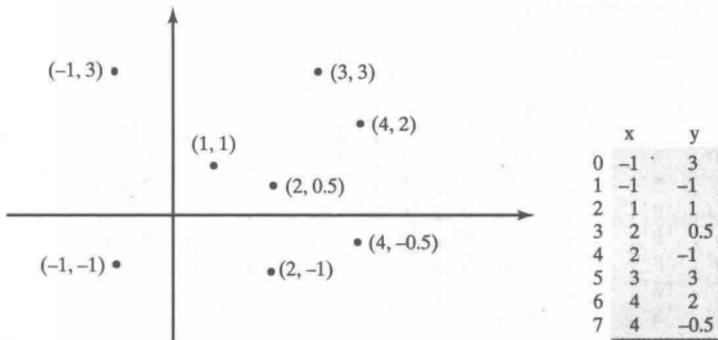


图 8-3 使用二维数组表示点

程序清单 8-3 FindNearestPoints.java

```

1  import java.util.Scanner;
2
3  public class FindNearestPoints {
4      public static void main(String[] args) {
5          Scanner input = new Scanner(System.in);
6          System.out.print("Enter the number of points: ");
7          int numberOfPoints = input.nextInt();
8
9          // Create an array to store points
10         double[][] points = new double[numberOfPoints][2];
11         System.out.print("Enter " + numberOfPoints + " points: ");
12         for (int i = 0; i < points.length; i++) {
13             points[i][0] = input.nextDouble();
14             points[i][1] = input.nextDouble();
15         }
16
17         // p1 and p2 are the indices in the points' array
18         int p1 = 0, p2 = 1; // Initial two points
19         double shortestDistance = distance(points[p1][0], points[p1][1],
20             points[p2][0], points[p2][1]); // Initialize shortestDistance
21
22         // Compute distance for every two points
23         for (int i = 0; i < points.length; i++) {
24             for (int j = i + 1; j < points.length; j++) {
25                 double distance = distance(points[i][0], points[i][1],
26                     points[j][0], points[j][1]); // Find distance
27
28                 if (shortestDistance > distance) {
29                     p1 = i; // Update p1
30                     p2 = j; // Update p2
31                     shortestDistance = distance; // Update shortestDistance
32                 }
33             }
34         }
35
36         // Display result
37         System.out.println("The closest two points are " +
38             "(" + points[p1][0] + ", " + points[p1][1] + ") and (" +
39             points[p2][0] + ", " + points[p2][1] + ")");
40     }
41
42     /** Compute the distance between two points (x1, y1) and (x2, y2)*/
43     public static double distance(

```

```

44     double x1, double y1, double x2, double y2) {
45     return Math.sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1));
46     }
47 }

```

```

Enter the number of points: 8
Enter 8 points: -1 3 -1 -1 1 1 2 0.5 2 -1 3 3 4 2 4 -0.5
The closest two points are (1, 1) and (2, 0.5)

```

程序提示用户输入点的个数(第6~7行)。从控制台读取多个点,并将它们存储在一个名为 `points` 的二维数组中(第12~15行)。程序使用变量 `shortestDistance`(第19行)来存储两个距离最近的点,而这两个点在 `points` 数组中的下标都存储在 `p1` 和 `p2` 中(第18行)。

对每一个索引值为 i 的点,程序会对所有的 $j > i$ 计算 `points[i]` 和 `points[j]` 之间的距离(第23~34行)。只要找到比当前最短距离更短的距离,就更新变量 `shortestDistance` 以及 `p1` 和 `p2`(第28~32行)。

两个点 (x_1, y_1) 和 (x_2, y_2) 之间的距离可以使用公式 $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ 计算(第43~46行)。

程序假设平面上至少有两个点。可以简单地修改程序,处理平面没有点或只有一个点的情况。

注意: 也可能会有不止一对具有相同最小距离的点对。程序找到这样的一对点。可以在编程练习题 8.8 中修改这个程序,找出所有距离最短的点对。

提示: 从键盘输入所有的点是很繁琐的。可以将输入存储在一个名为 `FindNearestPoints.txt` 的文件中,并使用下面的命令编译和运行这个程序:

```
java FindNearestPoints < FindNearestPoints.txt
```

8.7 示例学习:数独

要点提示: 要解决的问题是检查一个给定的数独解答是否正确。

本节介绍一个每天都会出现在报纸上的很有趣的问题。这是一个数字放置的难题,通常称为数独(Sudoku)。它是一个非常有挑战性的问题。为了使之能被编程新手接受,本节给出数独问题的简化版本的一个解决方案,它可以验证该解决方案是否正确。解决数独问题的完整方案放在补充材料 VI.A 中。

数独是一个 9×9 的网格,它被分为更小的 3×3 的盒子(也称为区域或者块),如图 8-4a 所示。将从 1 到 9 的数字植入一些称为固定方格(fixed cell)的格子里。该程序的目标是将从 1 到 9 的数字植入那些称为自由方格(free cell)的格子,以便能够使得每行每列以及每个 3×3 的盒子都包含从 1 到 9 的数字,如图 8-4b 所示。

为了方便起见,使用值 0 表示自由方格,如图 8-5a 所示。很自然就会使用二维数组表示网格,如图 8-5b 所示。

为了找到该难题的解决方案,必须用 1 到 9 之间合适的数字替换网格中的每个 0。对于图 8-5 中难题的解决方案,网格应该如图 8-6 所示。

一旦找到一个数独难题的解决方案,如何验证它是正确的呢?有两种方法:

- 检查是否每行都有 1 到 9 的数字以及每列都有 1 到 9 的数字,并且每个小的方块都有 1 到 9 的数字。

- 检查每个单元格。每个单元格必须是 1 到 9 的数字，单元格数字在每行、每列，以及每个小方盒中都是唯一的。

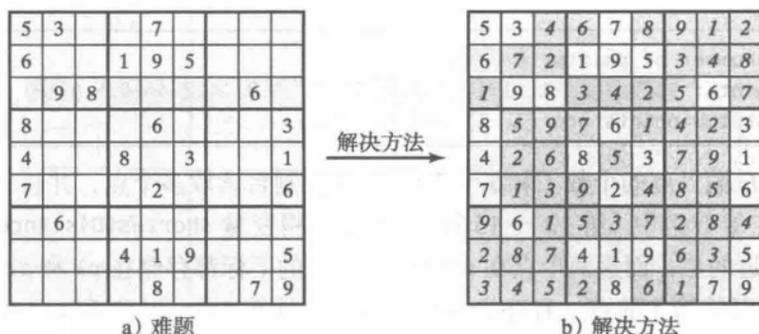


图 8-4 图 a 中的难题在图 b 中解决

5	3	0	0	7	0	0	0	0
6	0	0	1	9	5	0	0	0
0	9	8	0	0	0	0	6	0
8	0	0	0	6	0	0	0	3
4	0	0	8	0	3	0	0	1
7	0	0	0	2	0	0	0	6
0	6	0	0	0	0	0	0	0
0	0	0	4	1	9	0	0	5
0	0	0	0	8	0	0	7	9

a)

```
int[][] grid =
{{5, 3, 0, 0, 7, 0, 0, 0, 0},
{6, 0, 0, 1, 9, 5, 0, 0, 0},
{0, 9, 8, 0, 0, 0, 0, 6, 0},
{8, 0, 0, 0, 6, 0, 0, 0, 3},
{4, 0, 0, 8, 0, 3, 0, 0, 1},
{7, 0, 0, 0, 2, 0, 0, 0, 6},
{0, 6, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 4, 1, 9, 0, 0, 5},
{0, 0, 0, 0, 8, 0, 0, 7, 9}};
```

b)

图 8-5 使用一个二维数组表示网格

```
A solution grid is
{{5, 3, 4, 6, 7, 8, 9, 1, 2},
{6, 7, 2, 1, 9, 5, 3, 4, 8},
{1, 9, 8, 3, 4, 2, 5, 6, 7},
{8, 5, 9, 7, 6, 1, 4, 2, 3},
{4, 2, 6, 8, 5, 3, 7, 9, 1},
{7, 1, 3, 9, 2, 4, 8, 5, 6},
{9, 6, 1, 5, 3, 7, 2, 8, 4},
{2, 8, 7, 4, 1, 9, 6, 3, 5},
{3, 4, 5, 2, 8, 6, 1, 7, 9}};
```

图 8-6 解决方案存储在网格 grid 中

程序清单 8-4 中的程序提示用户输入一个解决方案，然后报告它是否有效。程序中采用第 2 种方法来检查解决方案是否正确。

程序清单 8-4 CheckSudokuSolution.java

```
1 import java.util.Scanner;
2
3 public class CheckSudokuSolution {
4     public static void main(String[] args) {
5         // Read a Sudoku solution
6         int[][] grid = readASolution();
7
8         System.out.println(isValid(grid) ? "Valid solution" :
9             "Invalid solution");
10    }
11
12    /** Read a Sudoku solution from the console */
13    public static int[][] readASolution() {
14        // Create a Scanner
15        Scanner input = new Scanner(System.in);
16
17        System.out.println("Enter a Sudoku puzzle solution:");
18        int[][] grid = new int[9][9];
19        for (int i = 0; i < 9; i++)
20            for (int j = 0; j < 9; j++)
21                grid[i][j] = input.nextInt();
22
23        return grid;
```

```

24 }
25
26 /** Check whether a solution is valid */
27 public static boolean isValid(int[][] grid) {
28     for (int i = 0; i < 9; i++)
29         for (int j = 0; j < 9; j++)
30             if (grid[i][j] < 1 || grid[i][j] > 9
31                 || !isValid(i, j, grid))
32                 return false;
33     return true; // The solution is valid
34 }
35
36 /** Check whether grid[i][j] is valid in the grid */
37 public static boolean isValid(int i, int j, int[][] grid) {
38     // Check whether grid[i][j] is unique in i's row
39     for (int column = 0; column < 9; column++)
40         if (column != j && grid[i][column] == grid[i][j])
41             return false;
42
43     // Check whether grid[i][j] is unique in j's column
44     for (int row = 0; row < 9; row++)
45         if (row != i && grid[row][j] == grid[i][j])
46             return false;
47
48     // Check whether grid[i][j] is unique in the 3-by-3 box
49     for (int row = (i / 3) * 3; row < (i / 3) * 3 + 3; row++)
50         for (int col = (j / 3) * 3; col < (j / 3) * 3 + 3; col++)
51             if (row != i && col != j && grid[row][col] == grid[i][j])
52                 return false;
53
54     return true; // The current value at grid[i][j] is valid
55 }
56 }

```

Enter a Sudoku puzzle solution:

9 6 3 1 7 4 2 5 8	<input type="button" value="Enter"/>
1 7 8 3 2 5 6 4 9	<input type="button" value="Enter"/>
2 5 4 6 8 9 7 3 1	<input type="button" value="Enter"/>
8 2 1 4 3 7 5 9 6	<input type="button" value="Enter"/>
4 9 6 8 5 2 3 1 7	<input type="button" value="Enter"/>
7 3 5 9 6 1 8 2 4	<input type="button" value="Enter"/>
5 8 9 7 1 3 4 6 2	<input type="button" value="Enter"/>
3 1 7 2 4 6 9 8 5	<input type="button" value="Enter"/>
6 4 2 5 9 8 1 7 3	<input type="button" value="Enter"/>

Valid solution

程序调用 `readASolution()` 方法 (第 6 行) 来读取一个数独的解决方案, 并且返回一个表示数独网格的二维数组。

`isValid(grid)` 方法通过检查每个值是否都是从 1 到 9 的数字以及每个网格中值是否都是有效的 (第 27 ~ 34 行), 来确认网格中是否放入了正确的值。

`isValid(i, j, grid)` 方法检查 `grid[i][j]` 的值是否是有效的。它检查 `grid[i][j]` 在第 `i` 行 (第 39 ~ 41 行)、第 `j` 列 (第 44 ~ 46 行), 以及 3×3 的方盒 (第 49 ~ 52 行) 中是否出现超过一次。

如何定位同一个方盒中的所有单元格呢? 对于任意的 `grid[i][j]`, 包含它的 3×3 的方盒的起始单元格是 `grid[(i/3)*3][(j/3)*3]`, 如图 8-7 所示。

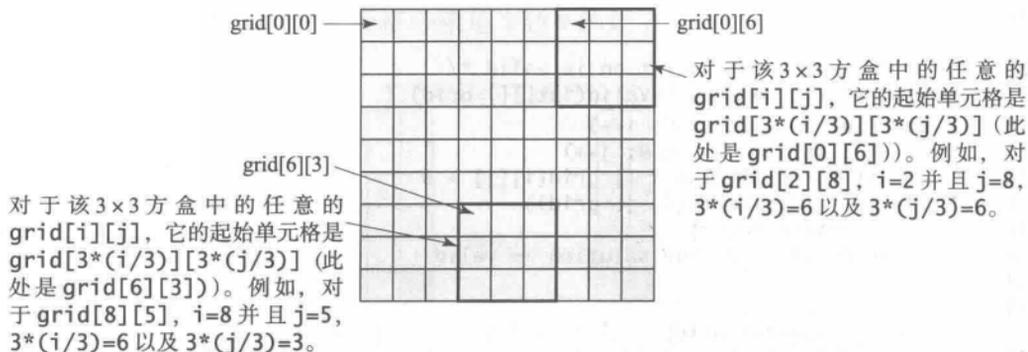


图 8-7 在 3×3 方盒的第一个单元格的位置决定了方盒中其他单元格的位置

观察到这点后，可以很容易地确定方盒中的所有单元格。例如，如果 $grid[r][c]$ 是 3×3 盒子的起始方格，这个方盒中的元素可以使用嵌套循环来遍历，如下所示：

```
// Get all cells in a 3-by-3 box starting at grid[r][c]
for (int row = r; row < r + 3; row++)
    for (int col = c; col < c + 3; col++)
        // grid[row][col] is in the box
```

从控制台输入 81 个数字是很繁琐的。测试这个程序时，可以将输入存储在一个名为 `CheckSudokuSolution.txt` 的文件中，然后使用下面的命令运行这个程序：

```
java CheckSudokuSolution < CheckSudokuSolution.txt
```

8.8 多维数组

要点提示：二维数组由一个一维数组的数组组成，而一个三维数组可以认为是由一个二维数组的数组所组成。

在前一节中，我们使用二维数组表示矩阵或者表格。有时，可能还需要表示 n 维的数据结构。在 Java 中，可以创建 n 维数组，其中 n 是任意整数。

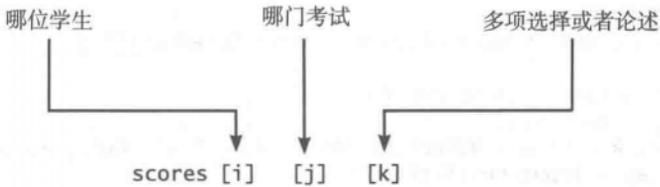
可以对二维数组变量的声明以及二维数组的创建方法进行推广，用于声明 $n \geq 3$ 的 n 维数组变量和创建 n 维数组。例如，可以采用一个三维数组来存储一个具有六个同学以及五门考试的班级成绩，其中每门考试有两部分（多选题以及论述）。下述语法声明一个三维数组变量 `scores`，创建一个数组并将它的引用赋值给 `scores`：

```
double[][][] scores = new double[6][5][2];
```

也可以采用简写方式来创建和初始化数组，如下所示：

```
double[][][] scores = {
    {{7.5, 20.5}, {9.0, 22.5}, {15, 33.5}, {13, 21.5}, {15, 2.5}},
    {{4.5, 21.5}, {9.0, 22.5}, {15, 34.5}, {12, 20.5}, {14, 9.5}},
    {{6.5, 30.5}, {9.4, 10.5}, {11, 33.5}, {11, 23.5}, {10, 2.5}},
    {{6.5, 23.5}, {9.4, 32.5}, {13, 34.5}, {11, 20.5}, {16, 7.5}},
    {{8.5, 26.5}, {9.4, 52.5}, {13, 36.5}, {13, 24.5}, {16, 2.5}},
    {{9.5, 20.5}, {9.4, 42.5}, {13, 31.5}, {12, 20.5}, {16, 6.5}}};
```

`score[0][1][0]` 代表第一个学生的第二门考试的多项选择部分的成绩，该值为 9.0。
`score[0][1][1]` 代表第一个学生的第二门考试的论述部分的成绩，该值为 22.5。在下图中给出了图示。



多维数组实际上是一个数组，它的每个元素都是另一个数组。三维数组是由二维数组构成的数组，每个二维数组又是一维数组的数组。例如，假设 `x=new int[2][2][5]`，则 `x[0]` 和 `x[1]` 是二维数组，`x[0][0]`、`x[0][1]`、`x[1][0]` 和 `x[1][1]` 都是一维数组，而且它们都含 5 个元素。`x.length` 的值为 2，`x[0].length` 和 `x[1].length` 的值为 2，`x[0][0].length`、`x[0][1].length`、`x[1][0].length` 和 `x[1][1].length` 的值为 5。

8.8.1 示例学习：每日温度和湿度

假设气象站每天每小时都会记录温度和湿度，并且将过去十天的数据都存储在一个名为 `Weather.txt` 的文本文件中（参见 `www.cs.armstrong.edu/liang/data/Weather.txt`）。文件中的每一行包含四个数字，分别表明日期、小时、温度和湿度。这个文件的内容如图 a 所示：

日期	小时	温度	湿度
1	1	76.4	0.92
1	2	77.7	0.93
...
10	23	97.7	0.71
10	24	98.7	0.74

a)

日期	小时	温度	湿度
10	24	98.7	0.74
1	2	77.7	0.93
...
10	23	97.7	0.71
1	1	76.4	0.92

b)

注意，文件的行不一定要按照日期和小时的升序排列。例如：文件可以如图 b 所示。

你的任务是编写程序，计算这十天的日均温度和日均湿度。可以使用输入重定向来读取文件，并将这些数据存在一个名为 `data` 的三维数组中。`data` 的第一个下标范围从 0 到 9，代表 10 天；第二个下标范围从 0 到 23，代表 24 小时；而第三个下标范围从 0 到 1，分别代表温度和湿度，如下图所示。



注意：在文件中，天是从 1 到 10 编号的，而小时是从 1 到 24 编号的。因为数组下标是从 0 开始的，所以 `data[0][0][0]` 存储的是第一天第一小时的温度，而 `data[9][23][1]` 存储的是第十天第二十四小时的湿度。

该程序在程序清单 8-5 中给出。

程序清单 8-5 Weather.java

```

1 import java.util.Scanner;
2
3 public class Weather {
4     public static void main(String[] args) {
5         final int NUMBER_OF_DAYS = 10;
6         final int NUMBER_OF_HOURS = 24;
    
```

```

7     double[][][] data
8       = new double[NUMBER_OF_DAYS][NUMBER_OF_HOURS][2];
9
10    Scanner input = new Scanner(System.in);
11    // Read input using input redirection from a file
12    for (int k = 0; k < NUMBER_OF_DAYS * NUMBER_OF_HOURS; k++) {
13        int day = input.nextInt();
14        int hour = input.nextInt();
15        double temperature = input.nextDouble();
16        double humidity = input.nextDouble();
17        data[day - 1][hour - 1][0] = temperature;
18        data[day - 1][hour - 1][1] = humidity;
19    }
20
21    // Find the average daily temperature and humidity
22    for (int i = 0; i < NUMBER_OF_DAYS; i++) {
23        double dailyTemperatureTotal = 0, dailyHumidityTotal = 0;
24        for (int j = 0; j < NUMBER_OF_HOURS; j++) {
25            dailyTemperatureTotal += data[i][j][0];
26            dailyHumidityTotal += data[i][j][1];
27        }
28
29        // Display result
30        System.out.println("Day " + i + "'s average temperature is "
31            + dailyTemperatureTotal / NUMBER_OF_HOURS);
32        System.out.println("Day " + i + "'s average humidity is "
33            + dailyHumidityTotal / NUMBER_OF_HOURS);
34    }
35 }
36 }

```

```

Day 0's average temperature is 77.7708
Day 0's average humidity is 0.929583
Day 1's average temperature is 77.3125
Day 1's average humidity is 0.929583
. . .
Day 9's average temperature is 79.3542
Day 9's average humidity is 0.9125

```

可以使用下面的命令来运行这个程序：

```
java Weather < Weather.txt
```

在第 8 行创建存储温度和湿度的三维数组。在第 12 ~ 19 行的循环中将输入读给数组。可以从键盘键入输入，但是这样做不是很便利。为了方便起见，将这些数据存储在一个文件中，并使用输入重定向从文件中读取数据。在第 24 ~ 27 行的循环中，将一天中每个小时的温度都加到 `dailyTemperatureTotal` 中，并将每个小时的湿度都加到 `dailyHumidityTotal` 中。第 30 ~ 33 行显示日均温度和日均湿度。

8.8.2 示例学习：猜生日

程序清单 4-3 给出一个猜生日的程序。可以通过用三维数组来存储 5 个数字集来简化程序，然后使用循环提示用户回答，如程序清单 8-6 所示。该程序的运行示例和程序清单 4-3 所显示的是一样。

程序清单 8-6 GuessBirthdayUsingArray.java

```

1 import java.util.Scanner;
2

```

```
3 public class GuessBirthdayUsingArray {
4     public static void main(String[] args) {
5         int day = 0; // Day to be determined
6         int answer;
7
8         int[][][] dates = {
9             {{ 1, 3, 5, 7},
10            { 9, 11, 13, 15},
11            {17, 19, 21, 23},
12            {25, 27, 29, 31}},
13            {{ 2, 3, 6, 7},
14            {10, 11, 14, 15},
15            {18, 19, 22, 23},
16            {26, 27, 30, 31}},
17            {{ 4, 5, 6, 7},
18            {12, 13, 14, 15},
19            {20, 21, 22, 23},
20            {28, 29, 30, 31}},
21            {{ 8, 9, 10, 11},
22            {12, 13, 14, 15},
23            {24, 25, 26, 27},
24            {28, 29, 30, 31}},
25            {{16, 17, 18, 19},
26            {20, 21, 22, 23},
27            {24, 25, 26, 27},
28            {28, 29, 30, 31}}};
29
30         // Create a Scanner
31         Scanner input = new Scanner(System.in);
32
33         for (int i = 0; i < 5; i++) {
34             System.out.println("Is your birthday in Set" + (i + 1) + "?");
35             for (int j = 0; j < 4; j++) {
36                 for (int k = 0; k < 4; k++)
37                     System.out.printf("%4d", dates[i][j][k]);
38                 System.out.println();
39             }
40
41             System.out.print("\nEnter 0 for No and 1 for Yes: ");
42             answer = input.nextInt();
43
44             if (answer == 1)
45                 day += dates[i][0][0];
46         }
47
48         System.out.println("Your birthday is " + day);
49     }
50 }
```

第 8 ~ 28 行创建三维数组 `dates`。这个数组存储 5 个数字集。每个集合都是一个 4×4 的二维数组。

循环从第 33 行开始显示每个集合的数字，然后提示用户回答生日是否在该集合中（第 41 ~ 42 行）。如果生日是在某个集合中，那么这个集合的第一个数字（`dates[i][0][0]`）就被加到变量 `day` 中（第 45 行）。

复习题

- 8.8 为一个三维数组声明一个数组变量，创建一个 $4 \times 6 \times 5$ 的 `int` 类型数组，并且将其引用赋给该变量。
- 8.9 假设 `int[][][] x = new char [12][5][2]`，该数组中有多少个元素？`x.length`、`x[2].length`，以及 `x[0][0].length` 分别是多少？

8.10 给出下面代码的输出：

```
int[][][] array = {{{1, 2}, {3, 4}}, {{5, 6},{7, 8}}};
System.out.println(array[0][0][0]);
System.out.println(array[1][1][1]);
```

本章小结

1. 可以使用二维数组来存储表格。
2. 可以使用以下语法来声明二维数组变量：

元素类型 [][] 数组变量

3. 可以使用以下语法来创建二维数组变量：

new 元素类型 [行的个数][列的个数]

4. 使用下面的语法表示二维数组中的每个元素：

数组变量 [行下标][列下标]

5. 可以使用数组初始化语法来创建和初始化二维数组：

元素类型 [][] 数组变量 = {{ 某行的值 }, ..., { 某行的值 }}

6. 可以使用数组的数组构成多维数组。例如：一个三维数组变量可以声明为“元素类型 [][][] 数组变量”，并使用“new 元素类型 [size1][size2][size3]”来创建三维数组。

测试题

在线回答本章节的测试题，位于 www.cs.armstrong.edu/liang/introl10e/quiz.html。

编程练习题

- *8.1 (求矩阵中各列数字的和) 编写一个方法，求整数矩阵中特定列的所有元素的和，使用下面的方法头：

```
public static double sumColumn(double[][] m, int columnIndex)
```

编写一个测试程序，读取一个 3×4 的矩阵，然后显示每列元素的和。下面是一个运行示例：

```
Enter a 3-by-4 matrix row by row:
1.5 2 3 4 ↵ Enter
5.5 6 7 8 ↵ Enter
9.5 1 3 1 ↵ Enter
Sum of the elements at column 0 is 16.5
Sum of the elements at column 1 is 9.0
Sum of the elements at column 2 is 13.0
Sum of the elements at column 3 is 13.0
```

- *8.2 (求矩阵主对角线元素的和) 编写一个方法，求 $n \times n$ 的 double 类型矩阵中主对角线上所有数字的和，使用下面的方法头：

```
public static double sumMajorDiagonal(double[][] m)
```

编写一个测试程序，读取一个 4×4 的矩阵，然后显示它的主对角线上的所有元素的和。下面是一个运行示例：

```
Enter a 4-by-4 matrix row by row:
1 2 3 4.0 ↵ Enter
5 6.5 7 8 ↵ Enter
9 10 11 12 ↵ Enter
```

```
13 14 15 16 
Sum of the elements in the major diagonal is 34.5
```

*8.3 (按考分对学生排序) 重写程序清单 8-2, 按照正确答案个数的升序显示学生。

**8.4 (计算每个雇员每周工作的小时数) 假定所有雇员每周工作的小时数存储在一个二维数组中。每行将一个雇员 7 天的工作时间记录在 7 列中。例如: 右面显示的数组存储了 8 个雇员的工作时间。编写一个程序, 按照总工时降序的方式显示雇员和他们的总工时。

	Su	M	T	W	Th	F	Sa
Employee 0	2	4	3	4	5	8	8
Employee 1	7	3	4	3	3	4	4
Employee 2	3	3	4	3	3	2	2
Employee 3	9	3	4	7	3	4	1
Employee 4	3	5	4	3	6	3	8
Employee 5	3	4	4	6	3	4	4
Employee 6	3	7	4	8	3	8	4
Employee 7	6	3	5	9	2	7	9

8.5 (代数方面: 两个矩阵相加) 编写两个矩阵相加的方法。方法头如下:

```
public static double[][] addMatrix(double[][] a, double[][] b)
```

为了能够进行相加, 两个矩阵必须具有相同的维数, 并且元素具有相同或兼容的数据类型。假设 c 表示最终的矩阵, 每个元素 c_{ij} 就是 $a_{ij}+b_{ij}$ 。例如, 对于两个 3×3 的矩阵 a 和 b , c 就有:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} a_{11}+b_{11} & a_{12}+b_{12} & a_{13}+b_{13} \\ a_{21}+b_{21} & a_{22}+b_{22} & a_{23}+b_{23} \\ a_{31}+b_{31} & a_{32}+b_{32} & a_{33}+b_{33} \end{pmatrix}$$

编写一个测试程序, 提示用户输入两个 3×3 的矩阵, 然后显示它们的和。下面是一个运行示例:

```
Enter matrix1: 1 2 3 4 5 6 7 8 9 
Enter matrix2: 0 2 4 1 4.5 2.2 1.1 4.3 5.2 
The matrices are added as follows
1.0 2.0 3.0      0.0 2.0 4.0      1.0 4.0 7.0
4.0 5.0 6.0 + 1.0 4.5 2.2 = 5.0 9.5 8.2
7.0 8.0 9.0      1.1 4.3 5.2      8.1 12.3 14.2
```

**8.6 (代数方面: 两个矩阵相乘) 编写两个矩阵相乘的方法。方法头如下:

```
public static double[][] multiplyMatrix(double[][] a, double[][] b)
```

为了使矩阵 a 能够和矩阵 b 相乘, 矩阵 a 的列数必须与矩阵 b 的行数相同, 并且两个矩阵的元素要具有相同或兼容的数据类型。假设矩阵 c 是相乘的结果, 而 a 的列数是 n , 那么每个元素 c_{ij} 就是 $a_{i1} \times b_{1j} + a_{i2} \times b_{2j} + \dots + a_{in} \times b_{nj}$ 。例如, 对于两个 3×3 的矩阵 a 和 b , c 就有:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

这里的 $c_{ij} = a_{i1} \times b_{1j} + a_{i2} \times b_{2j} + a_{i3} \times b_{3j}$ 。

编写一个测试程序, 提示用户输入两个 3×3 的矩阵, 然后显示它们的乘积。下面是一个运行示例:

```
Enter matrix1: 1 2 3 4 5 6 7 8 9 
Enter matrix2: 0 2 4 1 4.5 2.2 1.1 4.3 5.2 
The multiplication of the matrices is
1 2 3      0 2.0 4.0      5.3 23.9 24
4 5 6 * 1 4.5 2.2 = 11.6 56.3 58.2
7 8 9      1.1 4.3 5.2      17.9 88.7 92.4
```

*8.7 (距离最近的两个点) 程序清单 8-3 给出找到二维空间中距离最近的两个点的程序。修改该程序, 让程序能够找出在三维空间上距离最近的两个点。使用一个二维数组表示这些点。使用下面的点来测试这个程序:

```
double[][] points = {{-1, 0, 3}, {-1, -1, -1}, {4, 1, 1},
    {2, 0.5, 9}, {3.5, 2, -1}, {3, 1.5, 3}, {-1.5, 4, 2},
    {5.5, 4, -0.5}};
```

计算两个点 (x_1, y_1, z_1) 和 (x_2, y_2, z_2) 之间距离的公式是 $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$ 。

**8.8 (所有最近的点对) 修改程序清单 8-3, 找出所有具有相同最小距离的点对。下面是一个运行示例:

```
Enter the number of points: 8
Enter 8 points: 0 0 1 1 -1 -1 2 2 -2 -2 -3 -3 -4 -4 5 5
The closest two points are (0.0, 0.0) and (1.0, 1.0)
The closest two points are (0.0, 0.0) and (-1.0, -1.0)
The closest two points are (1.0, 1.0) and (2.0, 2.0)
The closest two points are (-1.0, -1.0) and (-2.0, -2.0)
The closest two points are (-2.0, -2.0) and (-3.0, -3.0)
The closest two points are (-3.0, -3.0) and (-4.0, -4.0)
Their distance is 1.4142135623730951
```

***8.9 (游戏: 井字游戏) 在井字游戏中, 两个玩家使用各自的标志 (一方用 X 则另一方就用 O), 轮流填写 3×3 的网格中的某个空格。当一个玩家在网格的水平方向、垂直方向或者对角线方向上出现了三个相同的 X 或三个相同的 O 时, 游戏结束, 该玩家获胜。平局 (没有赢家) 是指当网格中所有的空格都被填满时没有任何一方的玩家获胜的情况。创建一个玩井字游戏的程序。

程序提示两个玩家可以选择 X 和 O 作为他们的标志。当输入一个标志时, 程序在控制台上重新显示棋盘, 然后确定游戏的状态 (是获胜、平局还是继续)。下面是一个运行示例:

```

| | | |
| | | |
| | | |
Enter a row (0, 1, or 2) for player X: 1
Enter a column (0, 1, or 2) for player X: 1
| | | |
| | X | |
| | | |
Enter a row (0, 1, or 2) for player O: 1
Enter a column (0, 1, or 2) for player O: 2
| | | |
| | X | O |
| | | |
```

```

Enter a row (0, 1, or 2) for player X:
. . .

-----
| X |  |  |
-----
| 0 | X | 0 |
-----
|  |  | X |
-----
X player won
    
```

**8.10 (最大的行和列) 编写一个程序，在一个 4x4 的矩阵中随机填入 0 和 1，打印该矩阵，找到第一个具有最多 1 的行和列。下面是一个程序的运行示例：

```

0011
0011
1101
1010
The largest row index: 2
The largest column index: 2
    
```

**8.11 (游戏：九个正面和背面) 一个 3x3 的矩阵中放置了 9 个硬币，这些硬币有些面向上，有些面向下。可以使用 3x3 的矩阵中的 0 (正面) 或 1 (反面) 表示硬币的状态。下面是一些例子：

```

0 0 0   1 0 1   1 1 0   1 0 1   1 0 0
0 1 0   0 0 1   1 0 0   1 1 0   1 1 1
0 0 0   1 0 0   0 0 1   1 0 0   1 1 0
    
```

每个状态都可以使用一个二进制数表示。例如，前面的矩阵对应到数字：

```
000010000 101001100 110100001 101110100 100111110
```

总共会有 512 种可能性。所以，可以使用十进制数 0, 1, 2, 3, ..., 511 来表示这个矩阵的所有状态。编写一个程序，提示用户输入一个在 0 到 511 之间的数字，然后显示用字符 H 和 T 表示的对应的矩阵。下面是一个运行示例：

```

Enter a number between 0 and 511: 7 
H H H
H H H
T T T
    
```

用户输入 7，它代表的是 000000111。因为 0 代表 H 而 1 代表 T，所以输出正确。

**8.12 (财务应用程序：计算税款) 使用数组重写程序清单 3-5。每个登记者的身份都有六种税率。每种税率都应用在某个特定范围内的可征税收入。例如：对一个可征税税收为 400 000 美元的单身登记者来讲，8350 美元的税率是 10%，8350 ~ 33 950 之间的税率是 15%，33 950 ~ 82 250 之间的税率是 25%，82 250 ~ 171 550 之间的税率是 28%，171 550 ~ 372 550 之间的税率是 33%，而 372 950 ~ 400 000 之间的税率是 36%。这六种税率对所有的登记身份都是一样的，可以用下面的数组来表示：

```
double[] rates = {0.10, 0.15, 0.25, 0.28, 0.33, 0.35};
```

所有登记者身份的每个利率括号都可以用一个二维数组表示，如下所示：

```

int[][] brackets = {
    {8350, 33950, 82250, 171550, 372950}, // Single filer
    {16700, 67900, 137050, 20885, 372950}, // Married jointly
                                           // -or qualifying widow(er)
    {8350, 33950, 68525, 104425, 186475}, // Married separately
    {11950, 45500, 117450, 190200, 372950} // Head of household
};
    
```

假设单身身份的登记者的可征税收入是 400 000 美元，该税收可以如下计算：

```
tax = brackets[0][0] * rates[0] +
      (brackets[0][1] - brackets[0][0]) * rates[1] +
      (brackets[0][2] - brackets[0][1]) * rates[2] +
      (brackets[0][3] - brackets[0][2]) * rates[3] +
      (brackets[0][4] - brackets[0][3]) * rates[4] +
      (400000 - brackets[0][4]) * rates[5]
```

*8.13 (定位最大的元素) 编写下面的方法，返回二维数组中最大元素的位置。

```
public static int[] locateLargest(double[][] a)
```

返回值是包含两个元素的一维数组。这两个元素表示二维数组中最大元素的行下标和列下标。编写一个测试程序，提示用户输入一个二维数组，然后显示这个数组中最大元素的位置。下面是一个运行示例：

```
Enter the number of rows and columns of the array: 3 4 
Enter the array:
23.5 35 2 10 
4.5 3 45 3.5 
35 44 5.5 9.6 
The location of the largest element is at (1, 2)
```

**8.14 (探索矩阵) 编写程序，提示用户输入一个方阵的长度，随机地在矩阵中填入 0 和 1，打印这个矩阵，然后找出整行、整列或者对角线都是 0 或 1 的行、列和对角线。下面是这个程序的一个运行示例：

```
Enter the size for the matrix: 4 
0111
0000
0100
1111
All 0s on row 1
All 1s on row 3
No same numbers on a column
No same numbers on the major diagonal
No same numbers on the sub-diagonal
```

*8.15 (几何：在一条直线上吗?) 编程练习题 6.39 给出一个方法，用于测试三点是否在一条直线上。编写下面的方法，检测 points 数组中所有的点是否都在同一条直线上。

```
public static boolean sameLine(double[][] points)
```

编写一个程序，提示用户输入 5 个点，并且显示它们是否在同一直线上。下面是一个运行示例：

```
Enter five points: 3.4 2 6.5 9.5 2.3 2.3 5.5 5 -5 4 
The five points are not on the same line
```

```
Enter five points: 1 1 2 2 3 3 4 4 5 5 
The five points are on the same line
```

*8.16 (对二维数组排序) 编写一个方法，使用下面的方法头对二维数组排序：

```
public static void sort(int m[][])
```

这个方法首先按行排序，然后按列排序。

例如：数组 {{4, 2}, {1, 7}, {4, 5}, {1, 2}, {1, 1}, {4, 1}} 将被排序为 {{1, 1}, {1, 2}, {1, 7}, {4, 1}, {4, 2}, {4, 5}}。

****8.17 (金融风暴)** 银行会互相借钱。在经济艰难时期，如果一个银行倒闭，它就不能偿还贷款。一个银行的总资产是它当前的余款减去它欠其他银行的贷款。图 8-8 就是五个银行的状况图。每个银行的当前余额分别是 2500 万美元、1 亿 2500 万美元、1 亿 7500 万美元、7500 万美元和 1 亿 8100 万美元。从节点 1 到节点 2 的方向的边表示银行 1 借给银行 2 共计 4 千万美元。

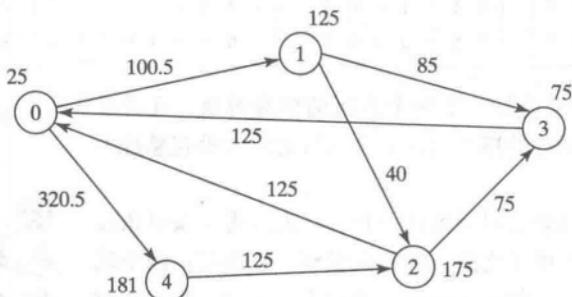


图 8-8 银行之间互相借款

如果银行的总资产在某个限定范围以下，那么这个银行就是不安全的。它借的钱就不能返还给借贷方，而且这个借贷方也不能将这个贷款算入它的总资产。因此，如果借贷方总资产在限定范围以下，那么它也不安全。编写程序，找出所有不安全的银行。程序如下读取输入。它首先读取两个整数 *n* 和 *limit*，这里的 *n* 表示银行个数，而 *limit* 表示要保持银行安全的最小总资产。然后，程序会读取描述 *n* 个银行的 *n* 行信息，银行的 id 从 0 到 *n*-1。每一行的第一个数字都是银行的余额，第二个数字表明从该银行借款的银行，其余的就都是两个数字构成的数对。每对都描述一个借款方。每一对数字的第一个数就是借款方的 id，第二个数就是所借的钱数。例如，在图 8-8 中五个银行的输入如下所示（注意：*limit* 是 201）：

```
5 201
25 2 1 100.5 4 320.5
125 2 2 40 3 85
175 2 0 125 3 75
75 1 0 125
181 1 2 125
```

银行 3 的总资产是 75+125，这个数字是在 201 以下的。所以，银行 3 是不安全的。在银行 3 变得不安全之后，银行 1 的总资产也降为 125+40。所以，银行 1 也不安全。程序的输出应该是：

```
Unsafe banks are 3 1
```

提示：使用一个二维数组 *borrowers* 来表示贷款。*borrowers*[*i*][*j*] 表明银行 *i* 贷款给银行 *j* 的贷款额。一旦银行 *j* 变得不安全，那么 *borrowers*[*i*][*j*] 就应该设置为 0。

****8.18 (打乱行)** 编写一个方法，使用下面的方法头打乱一个二维整型数组的行：

```
public static void shuffle(int[][] m)
```

编写一个测试程序，打乱下面的矩阵：

```
int[][] m = {{1, 2}, {3, 4}, {5, 6}, {7, 8}, {9, 10}};
```

****8.19 (模式识别：连续四个相等的数)** 编写下面的方法，测试一个二维数组是否有四个连续的数字具有相同的值，这四个数可以是水平方向的、垂直方向的或者对角线方向的。

```
public static boolean isConsecutiveFour(int[][] values)
```

编写一个测试程序，提示用户输入一个二维数组的行数、列数以及数组中的值。如果这个数组有四个连续的数字具有相同的值，就显示 true；否则，显示 false。下面是结果为 true 的一些例子：

0	1	0	3	1	6	1
0	1	6	8	6	0	1
5	6	2	1	8	2	9
6	5	6	1	1	9	1
1	3	6	1	4	0	7
3	3	3	3	4	0	7

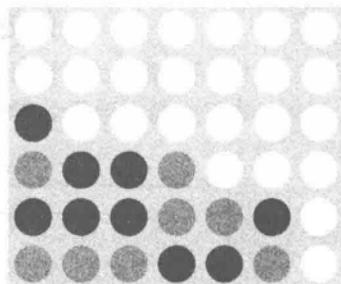
0	1	0	3	1	6	1
0	1	6	8	6	0	1
5	5	2	1	8	2	9
6	5	6	1	1	9	1
1	5	6	1	4	0	7
3	5	3	3	4	0	7

0	1	0	3	1	6	1
0	1	6	8	6	0	1
5	6	2	1	6	2	9
6	5	6	6	1	9	1
1	3	6	1	4	0	7
3	6	3	3	4	0	7

0	1	0	3	1	6	1
0	1	6	8	6	0	1
9	6	2	1	8	2	9
6	9	6	1	1	9	1
1	3	9	1	4	0	7
3	3	3	9	4	0	7

***8.20 (游戏：四子连) 四子连是一个两个人玩的棋盘游戏，在游戏中，玩家轮流将有颜色的棋子放在一个六行七列的垂直悬挂的网格中，如下所示。

这个游戏的目的是在对手实现一行、一列或者一条对角线上有四个相同颜色的棋子之前，你能先做到。程序提示两个玩家交替地下红子 Red 或黄子 Yellow。当放下一子时，程序在控制台重新显示这个棋盘，然后确定游戏的状态（赢、平局还是继续）。下面是一个运行示例：



```

| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
-----
Drop a red disk at column (0-6): 0  Enter
| | | | | | |
| | | | | | |
| | | | | | |
|R| | | | | |
| | | | | | |
-----
Drop a yellow disk at column (0-6): 3  Enter
| | | | | | | |
| | | | | | |
| | | | | | |
|R| | | Y| | | |
| | | | | | |
| | | | | | |
| | | | | | |
-----
Drop a yellow disk at column (0-6): 6  Enter
| | | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
|R|Y|R|Y|R|Y|Y|
|R|Y|R|Y|R|R|R|
-----
The yellow player won

```

*8.21 (中心城市) 给定一组城市，中心城市是和所有其他城市之间具有最短距离的城市。编写一个程序，提示用户输入城市的数目以及城市的位置（坐标），找到中心城市以及和所有其他城市的总

距离。

```
Enter the number of cities: 5 ↵
Enter the coordinates of the cities:
2.5 5 5.1 3 1 9 5.4 54 5.5 2.1 ↵
The central city is at (2.5, 5.0)
The total distance to all other cities is 60.81
```

*8.22 (偶数个 1) 编写一个程序，产生一个 6×6 的填写了 0 和 1 的二维矩阵，显示该矩阵，检测是否每行以及每列中有偶数个 1。

*8.23 (游戏：找到翻转的单元格) 假设给定一个填满 0 和 1 的 6×6 矩阵。所有的行和列都有偶数个 1。让用户翻转一个单元（即从 1 翻成 0 或者从 0 翻成 1），编写一个程序找到哪个单元格被翻转了。程序应该提示用户输入一个 6×6 的填满 0 和 1 的矩阵，并且找到第一个 r 行以及第一个 c 列具有偶数个 1 的特征是不符合的（即 1 的数目不是偶数），则该翻转的单元格位于 (r, c) 。下面是一个运行示例：

```
Enter a 6-by-6 matrix row by row:
1 1 1 0 1 1 ↵
1 1 1 1 0 0 ↵
0 1 0 1 1 1 ↵
1 1 1 1 1 1 ↵
0 1 1 1 1 0 ↵
1 0 0 0 0 1 ↵
The flipped cell is at (0, 1)
```

*8.24 (检测数独的解决方案) 程序清单 8-4 通过检测棋盘上的每个数字是否是有效的，从而检测一个解决方案是否是有效的。重写该程序，通过检测是否每行、每列以及每个小的方盒中具有数字 1 到 9 来检测解决方案的有效性。

*8.25 (马尔科夫矩阵) 一个 $n \times n$ 的矩阵被称为一个正马尔科夫矩阵，当且仅当每个元素都是正数，并且每列的元素的和为 1。编写下面的方法来检测一个矩阵是否是一个马尔科夫矩阵。

```
public static boolean isMarkovMatrix(double[][] m)
```

编写一个测试程序，提示用户输入一个 3×3 的 `double` 值的矩阵，测试它是否是一个马尔科夫矩阵。下面是一个运行示例：

```
Enter a 3-by-3 matrix row by row:
0.15 0.875 0.375 ↵
0.55 0.005 0.225 ↵
0.30 0.12 0.4 ↵
It is a Markov matrix
```

```
Enter a 3-by-3 matrix row by row:
0.95 -0.875 0.375 ↵
0.65 0.005 0.225 ↵
0.30 0.22 -0.4 ↵
It is not a Markov matrix
```

*8.26 (行排序) 用下面的方法实现一个二维数组中的行排序。返回一个新的数组，并且原数组保持不变。

```
public static double[][] sortRows(double[][] m)
```

编写一个测试程序，提示用户输入一个 3×3 的 `double` 类型值的矩阵，显示一个新的每行排好序的矩阵。下面是一个运行示例。

```
Enter a 3-by-3 matrix row by row:
```

```
0.15 0.875 0.375 ↵ Enter
```

```
0.55 0.005 0.225 ↵ Enter
```

```
0.30 0.12 0.4 ↵ Enter
```

```
The row-sorted array is
```

```
0.15 0.375 0.875
```

```
0.005 0.225 0.55
```

```
0.12 0.30 0.4
```

- *8.27 (列排序) 用下面的方法实现一个二维数组中的列排序。返回一个新的数组，并且原数组保持不变。

```
public static double[][] sortColumns(double[][] m)
```

编写一个测试程序，提示用户输入一个 3×3 的 `double` 类型值的矩阵，显示一个新的每列排好序的矩阵。下面是一个运行示例。

```
Enter a 3-by-3 matrix row by row:
```

```
0.15 0.875 0.375 ↵ Enter
```

```
0.55 0.005 0.225 ↵ Enter
```

```
0.30 0.12 0.4 ↵ Enter
```

```
The column-sorted array is
```

```
0.15 0.0050 0.225
```

```
0.3 0.12 0.375
```

```
0.55 0.875 0.4
```

- 8.28 (严格相同的数组) 如果两个二维数组 `m1` 和 `m2` 相应的元素是相等的话，则认为它们是严格相同的。编写一个方法，如果 `m1` 和 `m2` 是严格相同的话，返回 `true`。使用下面的方法头：

```
public static boolean equals(int[][] m1, int[][] m2)
```

编写一个测试程序，提示用户输入两个 3×3 的整数数组，显示两个矩阵是否是严格相同的。下面是一个运行示例。

```
Enter list1: 51 22 25 6 1 4 24 54 6 ↵ Enter
```

```
Enter list2: 51 22 25 6 1 4 24 54 6 ↵ Enter
```

```
The two arrays are strictly identical
```

```
Enter list1: 51 25 22 6 1 4 24 54 6 ↵ Enter
```

```
Enter list2: 51 22 25 6 1 4 24 54 6 ↵ Enter
```

```
The two arrays are not strictly identical
```

- 8.29 (相同的数组) 如果两个二维数组 `m1` 和 `m2` 具有相同的内容，则它们是相同的。编写一个方法，如果 `m1` 和 `m2` 是相同的话，返回 `true`。使用下面的方法头：

```
public static boolean equals(int[][] m1, int[][] m2)
```

编写一个测试程序，提示用户输入两个 3×3 的整数数组，显示两个矩阵是否是相同的。下面是一个运行示例。

```
Enter list1: 51 25 22 6 1 4 24 54 6 ↵ Enter
```

```
Enter list2: 51 22 25 6 1 4 24 54 6 ↵ Enter
```

```
The two arrays are identical
```

```
Enter list1: 51 5 22 6 1 4 24 54 6 ~Enter
Enter list2: 51 22 25 6 1 4 24 54 6 ~Enter
The two arrays are not identical
```

*8.30 (代数: 解答线性方程) 编写一个方法, 解答下面的 2×2 线性方程组系统:

$$\begin{aligned} a_{00}x + a_{01}y &= b_0 \\ a_{10}x + a_{11}y &= b_1 \end{aligned} \quad x = \frac{b_0a_{11} - b_1a_{01}}{a_{00}a_{11} - a_{01}a_{10}} \quad y = \frac{b_1a_{00} - b_0a_{10}}{a_{00}a_{11} - a_{01}a_{10}}$$

方法头为:

```
public static double[] linearEquation(double[][] a, double[] b)
```

如果 $a_{00}a_{11} - a_{01}a_{10}$ 为 0, 方法返回 null。编写一个测试程序, 提示用户输入 a_{00} 、 a_{01} 、 a_{10} 、 a_{11} 、 b_0 以及 b_1 , 并且显示结果。如果 $a_{00}a_{11} - a_{01}a_{10}$ 为 0, 报告“方程无解”。运行示例和编程练习题 3.3 的类似。

*8.31 (几何: 交点) 编写一个方法, 返回两条直线的交点。两条直线的交点可以使用编程练习题 3.25 中显示的公式找到。假设 (x_1, y_1) 和 (x_2, y_2) 是直线 1 上的两点, 而 (x_3, y_3) 和 (x_4, y_4) 位于直线 2 上。方法头是:

```
public static double[] getIntersectingPoint(double[][] points)
```

这些点保存在一个 4×2 的二维矩阵 points 中, 其中 $(points[0][0], points[0][1])$ 代表 (x_1, y_1) 。方法返回交点, 或者如果两条线平行的话就返回 null。编写一个程序, 提示用户输入四个点, 并且显示交点。运行示例参见编程练习题 3.25。

*8.32 (几何: 三角形面积) 编写一个方法, 使用下面的方法头, 返回一个三角形的面积:

```
public static double getTriangleArea(double[][] points)
```

点保存在一个 3×2 的二维矩阵 points 中, 其中 $(points[0][0], points[0][1])$ 代表 (x_1, y_1) 。三角形面积的计算可以使用编程练习题 2.19 中的公式。如果三个点在一条直线上, 方法返回 0。编写一个程序, 提示用户输入三角形的三个点, 然后显示三角形的面积。下面是一个运行示例。

```
Enter x1, y1, x2, y2, x3, y3: 2.5 2 5 -1.0 4.0 2.0 ~Enter
The area of the triangle is 2.25
```

```
Enter x1, y1, x2, y2, x3, y3: 2 2 4.5 4.5 6 6 ~Enter
The three points are on the same line
```

*8.33 (几何: 多边形的子面积) 一个具有四个顶点的凸多边形被分为四个三角形, 如图 8-9 所示。编写一个程序, 提示用户输入四个顶点的坐标, 然后以升序显示四个三角形的面积。下面是一个运行示例。

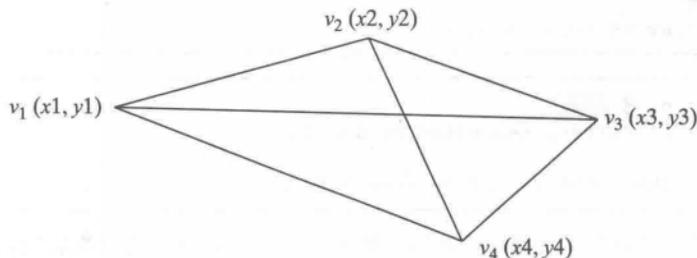


图 8-9 一个具有四个顶点的多边形被四个顶点所限定

```
Enter x1, y1, x2, y2, x3, y3, x4, y4:
-2.5 2 4 4 3 -2 -2 -3.5 Enter
The areas are 6.17 7.96 8.08 10.42
```

- *8.34 (几何: 最右下角的点) 在计算几何中经常需要从一个点集中找到最右下角的点。编写以下方法, 从一个点的集合中返回最右下角的点。

```
public static double[]
    getRightmostLowestPoint(double[][] points)
```

编写一个测试程序, 提示用户输入 6 个点的坐标, 然后显示最右下角的点。下面是一个运行示例。

```
Enter 6 points: 1.5 2.5 -3 4.5 5.6 -7 6.5 -7 8 1 10 2.5 Enter
The rightmost lowest point is (6.5, -7.0)
```

- **8.35 (最大块) 给定一个元素为 0 或者 1 的方阵, 编写一个程序, 找到一个元素都为 1 的最大的子方阵。程序提示用户输入矩阵的行数。然后显示最大的子方阵的第一个元素, 以及该子方阵中的行数。下面是一个运行示例。

```
Enter the number of rows in the matrix: 5 Enter
Enter the matrix row by row:
1 0 1 0 1 Enter
1 1 1 0 1 Enter
1 0 1 1 1 Enter
1 0 1 1 1 Enter
1 0 1 1 1 Enter
The maximum square submatrix is at (2, 2) with size 3
```

程序需要实现和使用下面的方法来找到最大的子方阵:

```
public static int[] findLargestBlock(int[][] m)
```

返回值是一个包含三个值的数组。前面两个值是子方阵中的行和列的下标, 第 3 个值是子方阵中的行数。

- **8.36 (拉丁正方形) 拉丁正方形是一个 $n \times n$ 的数组, 由 n 个不同的拉丁字母填充, 每个拉丁字母恰好只在每行和每列中出现一次。编写一个程序, 提示用户输入数字 n 以及字符数组, 如示例输出所示, 检测该输出数组是否是一个拉丁正方形。字符是从 A 开始的前面 n 个字符。

```
Enter number n: 4 Enter
Enter 4 rows of letters separated by spaces:
A B C D Enter
B A D C Enter
C D B A Enter
D C A B Enter
The input array is a Latin square
```

```
Enter number n: 3 Enter
Enter 3 rows of letters separated by spaces:
A F D Enter
Wrong input: the letters must be from A to C
```

- **8.37 (猜测首府) 编写一个程序, 重复提示用户输入一个州的首府。当接收到用户输入后, 程序报告答案是否正确。假设 50 个州以及它们的首府保存在一个二维数组中, 如图 8-10 所示。程序提

示用户回答所有州的首府，并且显示所有正确回答的数目（忽略英文字母的大小写）。

Alabama	Montgomery
Alaska	Juneau
Arizona	Phoenix
...	...
...	...

图 8-10 一个保存了州以及它们的首府的二维数组

下面是一个运行示例。

```
What is the capital of Alabama? Montogomery  Enter
The correct answer should be Montgomery
What is the capital of Alaska? Juneau  Enter
Your answer is correct
What is the capital of Arizona? ...
...
The correct count is 35
```

对象和类

教学目标

- 描述对象和类，使用类来建模对象（9.2节）。
- 使用UML图形符号来描述类和对象（9.2节）。
- 演示如何定义类以及如何创建对象（9.3节）。
- 使用构造方法创建对象（9.4节）。
- 通过对象引用变量访问对象（9.5节）。
- 使用引用类型定义引用变量（9.5.1节）。
- 使用对象成员访问操作符（.）来访问对象的数据和方法（9.5.2节）。
- 定义引用类型的数据域并给对象的数据域赋默认值（9.5.3节）。
- 区分对象引用变量和基本类型变量的不同（9.5.4节）。
- 使用Java类库中的Data类、Random类和Point2D类（9.6节）。
- 区分实例变量与静态变量、实例方法与静态方法的不同（9.7节）。
- 定义有恰当的get方法和set方法的私有数据域（9.8节）。
- 封装数据域以便于类的维护（9.9节）。
- 开发带对象参数的方法，区分基本类型参数和对象类型参数的不同（9.10节）。
- 在数组中存储和处理对象（9.11节）。
- 从不变类中创建不变对象，从而保护对象的内容。（9.12节）。
- 在一个类中确定变量的范围（9.13节）。
- 使用关键字this来引用对象自身（9.14节）。

9.1 引言

要点提示：面向对象编程可以有效地帮助开发大规模的软件以及图形用户界面。

学习过前几章的内容之后，你已经能够使用选择、循环、方法和数组解决很多程序设计问题。但是，这些Java的特性还不足够用来开发图形用户界面和大型软件系统。假设希望开发一个GUI（图形用户界面，发音为goo-ee），如图9-1所示，该如何用程序实现它呢？



图 9-1 从类中创建这些 GUI 对象

本章开始介绍面向对象程序设计，它会有助于更有效地开发 GUI 和大型软件系统。

9.2 为对象定义类

要点提示：类为对象定义属性和行为。

面向对象程序设计 (OOP) 就是使用对象进行程序设计。对象 (object) 代表现实世界中可以明确标识的一个实体。例如：一个学生、一张桌子、一个圆、一个按钮甚至一笔贷款都可以看作是一个对象。每个对象都有自己独特的标识、状态和行为。

- 一个对象的状态 (state, 也称为特征 (property) 或属性 (attribute)) 是由具有当前值的数据域来表示的。例如：圆对象具有一个数据域 `radius`，它是标识圆的属性。一个矩形对象具有数据域 `width` 和 `height`，它们都是描述矩形的属性。
- 一个对象的行为 (behavior, 也称为动作 (action)) 是由方法定义的。调用对象的一个方法就是要求对象完成一个动作。例如：可以为圆对象定义一个名为 `getArea()` 和 `getPerimeter()` 的方法。圆对象可以调用 `getArea()` 返回圆的面积，调用 `getPerimeter()` 返回它的周长。还可以定义 `setRadius(radius)` 方法。圆对象可以调用这个方法修改它的半径。

使用一个通用类来定义同一类型的对象。类是一个模板、蓝本或者说是合约，用来定义对象的数据域是什么以及方法是做什么的。一个对象是类的一个实例。可以从一个类中创建多个实例。创建实例的过程称为实例化 (instantiation)。对象 (object) 和实例 (instance) 经常是可以互换的。类和对象之间的关系类似于苹果派配方和苹果派之间的关系。可以用一种配方做出任意多的苹果派来。图 9-2 显示名为 `Circle` 的类和它的三个对象。

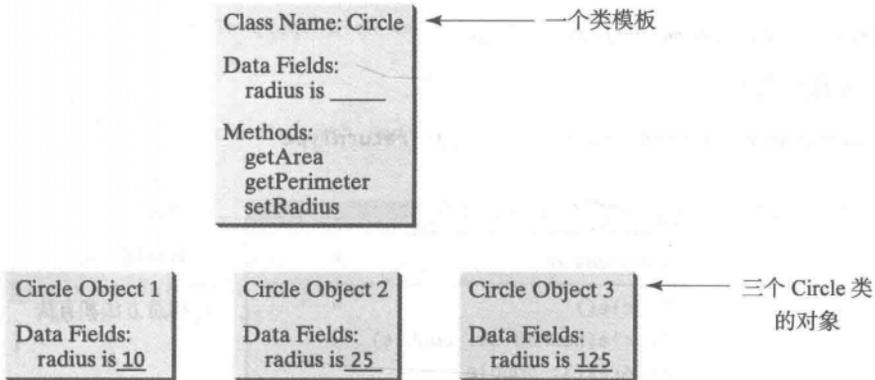


图 9-2 类是创建对象的模板

Java 类使用变量定义数据域，使用方法定义动作。除此之外，类还提供了一种称为构造方法 (constructor) 的特殊类型的方法，调用它可以创建一个新对象。构造方法本身是可以完成任何动作的，但是设计构造方法是为了完成初始化动作，例如：初始化对象的数据域。图 9-3 显示定义圆对象的类的例子。

`Circle` 类与目前所见过的所有其他类都不同，它没有 `main` 方法，因此是不能运行的；它只是对圆对象的定义。本书还将涉及包含 `main` 方法的类。为了方便，本书将包含 `main` 方法的类称为主类 (main class)。

图 9-2 中类的模板和对象的图示可以使用统一建模语言 (Unified Modeling Language, UML) 的图形符号进行标准化，如图 9-4 所示，这种表示方法称为 UML 类图 (UML class diagram)，或简称为类图 (class diagram)。在类图中，数据域表示为：

`dataFieldName: dataType`

构造方法可以表示为：

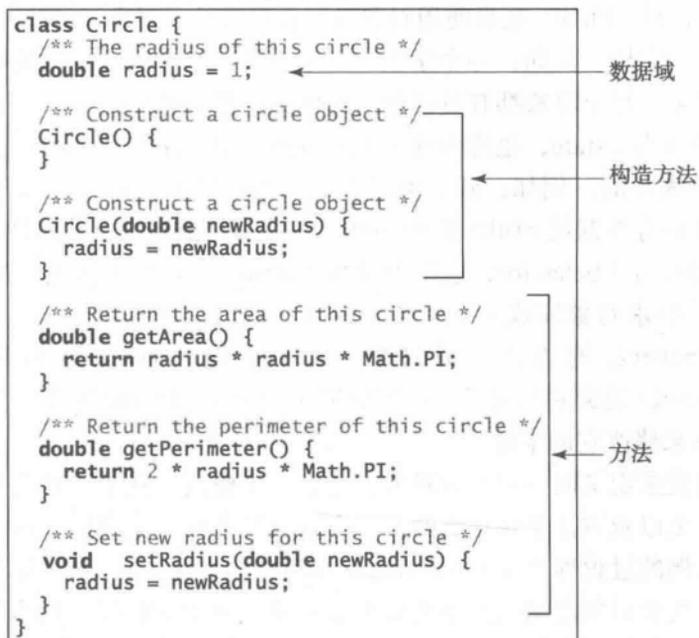


图 9-3 类是定义相同类型对象的结构

ClassName(parameterName: parameterType)

方法可以表示为:

methodName(parameterName: parameterType): returnType

UML 类图

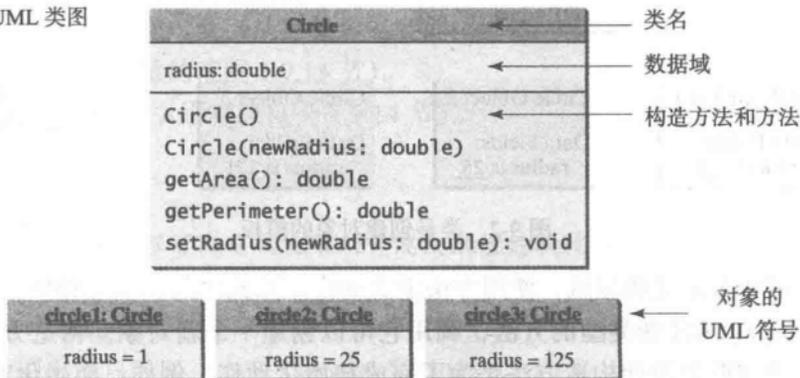


图 9-4 可以使用 UML 符号表示类 and 对象

9.3 示例：定义类和创建对象

要点提示：类是对象的定义，对象从类创建。

本节给出两个定义类和使用类创建对象的例子。程序清单 9-1 是一个定义 Circle 类并使用该类创建对象的程序。程序构造了三个圆对象，其半径分别为 1、25 和 125，然后显示这三个圆的半径和面积。然后将第二个对象的半径改为 100，并显示它的新半径和面积。

注意：为了避免与本章后续介绍的 Circle 类的改进版本有命名冲突，将本例中的 Circle 类命名为 SimpleCircle，为简单起见我们仍然将教材中的类称为 Circle。

程序清单 9-1 TestSimpleCircle.java

```
1 public class TestSimpleCircle {
2     /** Main method */
3     public static void main(String[] args) {
4         // Create a circle with radius 1
5         SimpleCircle circle1 = new SimpleCircle();
6         System.out.println("The area of the circle of radius "
7             + circle1.radius + " is " + circle1.getArea());
8
9         // Create a circle with radius 25
10        SimpleCircle circle2 = new SimpleCircle(25);
11        System.out.println("The area of the circle of radius "
12            + circle2.radius + " is " + circle2.getArea());
13
14        // Create a circle with radius 125
15        SimpleCircle circle3 = new SimpleCircle(125);
16        System.out.println("The area of the circle of radius "
17            + circle3.radius + " is " + circle3.getArea());
18
19        // Modify circle radius
20        circle2.radius = 100; // or circle2.setRadius(100)
21        System.out.println("The area of the circle of radius "
22            + circle2.radius + " is " + circle2.getArea());
23    }
24 }
25
26 // Define the circle class with two constructors
27 class SimpleCircle {
28     double radius;
29
30     /** Construct a circle with radius 1 */
31     SimpleCircle() {
32         radius = 1;
33     }
34
35     /** Construct a circle with a specified radius */
36     SimpleCircle(double newRadius) {
37         radius = newRadius;
38     }
39
40     /** Return the area of this circle */
41     double getArea() {
42         return radius * radius * Math.PI;
43     }
44
45     /** Return the perimeter of this circle */
46     double getPerimeter() {
47         return 2 * radius * Math.PI;
48     }
49
50     /** Set a new radius for this circle */
51     void setRadius(double newRadius) {
52         radius = newRadius;
53     }
54 }
```

```
The area of the circle of radius 1.0 is 3.141592653589793
The area of the circle of radius 25.0 is 1963.4954084936207
The area of the circle of radius 125.0 is 49087.385212340516
The area of the circle of radius 100.0 is 31415.926535897932
```

程序包括两个类。其中第一个类 TestSimpleCircle 是主类。它的唯一目的就是测试第

二个类 SimpleCircle。使用这样的类的程序通常称为该类的客户 (client)。运行这个程序时, Java 运行系统会调用这个主类的 main 方法。

可以把两个类放在同一个文件中, 但是文件中只能有一个类是公共 (public) 类。此外, 公共类必须与文件同名。因此, 文件名就应该是 TestSimpleCircle.java, 因为 TestSimpleCircle 是公共的。源代码中的每个类编译成 .class 文件。当编译 TestSimpleCircle.java 时, 产生两个类文件 TestSimpleCircle.class 和 SimpleCircle.class, 如图 9-5 所示。

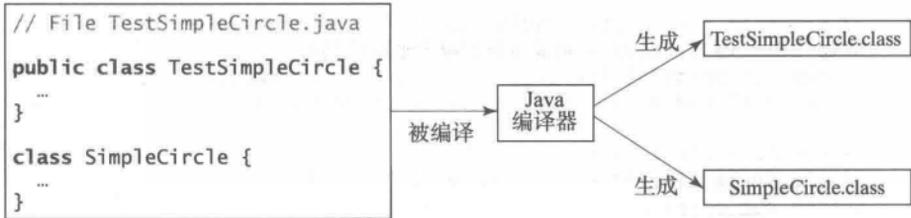


图 9-5 源代码中的每个类被编译成一个 .class 文件

主类包含 main 方法 (第 3 行), 该方法创建三个对象。和创建数组一样, 使用 new 操作符从构造方法创建一个对象。new SimpleCircle() 创建一个半径为 1 的对象 (第 5 行), new SimpleCircle(25) 创建一个半径为 25 的对象 (第 10 行), 而 new SimpleCircle (125) 创建一个半径为 125 的对象 (第 15 行)。

这三个对象 (通过 circle1、circle2 和 circle3 来引用) 有不同的数据, 但是有相同的方法。因此, 可以使用 getArea() 方法计算它们各自的面积。可以分别使用 circle1.radius、circle2.radius、circle3.radius 来通过对象引用访问数据域。对象可以分别使用 circle1.getArea()、circle2.getArea()、circle3.getArea() 来通过对象引用调用它的方法。

这三个对象是独立的。circle2 的半径在第 20 行改为 100。这个对象的新半径和新面积在第 21 ~ 22 行显示。

编写 Java 程序的方法有很多种。例如, 可以将例子中的两个类组合成一个, 如程序清单 9-2 所示。

程序清单 9-2 SimpleCircle.java

```

1 public class SimpleCircle {
2     /** Main method */
3     public static void main(String[] args) {
4         // Create a circle with radius 1
5         SimpleCircle circle1 = new SimpleCircle();
6         System.out.println("The area of the circle of radius "
7             + circle1.radius + " is " + circle1.getArea());
8
9         // Create a circle with radius 25
10        SimpleCircle circle2 = new SimpleCircle(25);
11        System.out.println("The area of the circle of radius "
12            + circle2.radius + " is " + circle2.getArea());
13
14        // Create a circle with radius 125
15        SimpleCircle circle3 = new SimpleCircle(125);
16        System.out.println("The area of the circle of radius "
17            + circle3.radius + " is " + circle3.getArea());
18
19        // Modify circle radius
20        circle2.radius = 100;
21        System.out.println("The area of the circle of radius "
22            + circle2.radius + " is " + circle2.getArea());
    
```

```

23  }
24
25  double radius;
26
27  /** Construct a circle with radius 1 */
28  SimpleCircle() {
29      radius = 1;
30  }
31
32  /** Construct a circle with a specified radius */
33  SimpleCircle(double newRadius) {
34      radius = newRadius;
35  }
36
37  /** Return the area of this circle */
38  double getArea() {
39      return radius * radius * Math.PI;
40  }
41
42  /** Return the perimeter of this circle */
43  double getPerimeter() {
44      return 2 * radius * Math.PI;
45  }
46
47  /** Set a new radius for this circle */
48  void setRadius(double newRadius) {
49      radius = newRadius;
50  }
51  }

```

由于组合后的类中有一个 main 方法，所以它可以由 Java 解释器来执行。main 方法和程序清单 9-1 中的是一样的。它演示如何通过在一个类中加入 main 方法来测试这个类。

另一个例子是关于电视机的。每台电视机都是一个对象，每个对象都有状态（当前频道、当前音量、电源开或关）以及动作（转换频道、调节音量、开启/关闭）。可以使用一个类对电视机进行建模。这个类的 UML 图如图 9-6 所示。

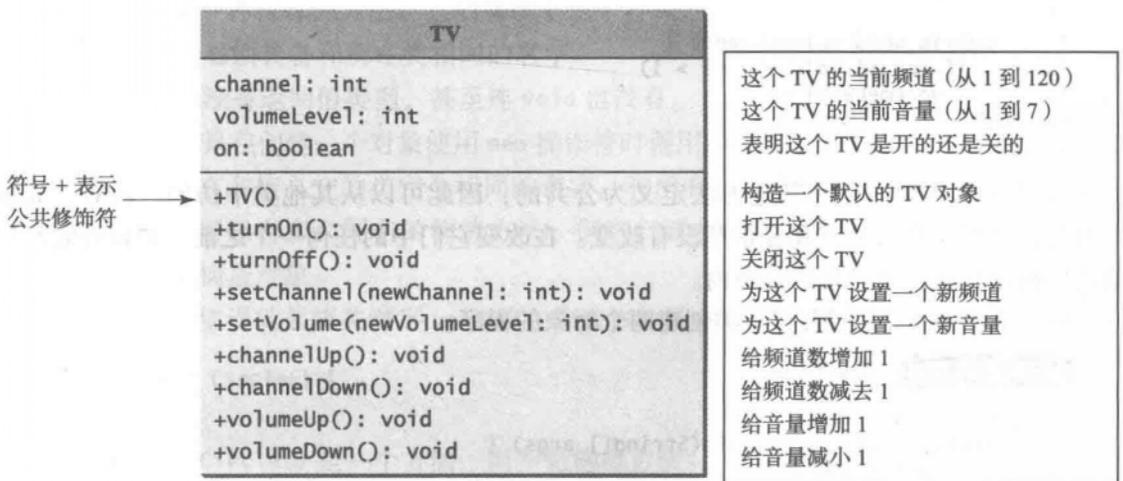


图 9-6 TV 类是对电视机的建模

程序清单 9-3 给出定义 TV 类的程序。

程序清单 9-3 TV.java

```

1 public class TV {

```

```
2 int channel = 1; // Default channel is 1
3 int volumeLevel = 1; // Default volume level is 1
4 boolean on = false; // TV is off
5
6 public TV() {
7 }
8
9 public void turnOn() {
10     on = true;
11 }
12
13 public void turnOff() {
14     on = false;
15 }
16
17 public void setChannel(int newChannel) {
18     if (on && newChannel >= 1 && newChannel <= 120)
19         channel = newChannel;
20 }
21
22 public void setVolume(int newVolumeLevel) {
23     if (on && newVolumeLevel >= 1 && newVolumeLevel <= 7)
24         volumeLevel = newVolumeLevel;
25 }
26
27 public void channelUp() {
28     if (on && channel < 120)
29         channel++;
30 }
31
32 public void channelDown() {
33     if (on && channel > 1)
34         channel--;
35 }
36
37 public void volumeUp() {
38     if (on && volumeLevel < 7)
39         volumeLevel++;
40 }
41
42 public void volumeDown() {
43     if (on && volumeLevel > 1)
44         volumeLevel--;
45 }
46 }
```

TV类中的构造方法和其他方法定义为公共的，因此可以从其他类中访问。注意，如果没有打开电视，那么频道和音量都没有改变。在改变它们中的任何一个之前，要检查它的当前值以确保它在一个正确的范围内。

程序清单 9-4 给出使用 TV 类创建两个对象的程序。

程序清单 9-4 TestTV.java

```
1 public class TestTV {
2     public static void main(String[] args) {
3         TV tv1 = new TV();
4         tv1.turnOn();
5         tv1.setChannel(30);
6         tv1.setVolume(3);
7
8         TV tv2 = new TV();
9         tv2.turnOn();
```

```
10     tv2.channelUp();
11     tv2.channelUp();
12     tv2.volumeUp();
13
14     System.out.println("tv1's channel is " + tv1.channel
15         + " and volume level is " + tv1.volumeLevel);
16     System.out.println("tv2's channel is " + tv2.channel
17         + " and volume level is " + tv2.volumeLevel);
18 }
19 }
```

```
tv1's channel is 30 and volume level is 3
tv2's channel is 3 and volume level is 2
```

程序在第3行和第8行创建两个对象，然后调用对象中的方法来完成设置频道和音量的动作，以及增加频道和提高音量的动作。程序在第14~17行显示对象的状态。使用像 `tv1.turnOn()` 的语法调用这个方法（第4行）。使用像 `tv1.channel` 的语法访问数据域（第14行）。

这些例子展示了类和对象的概貌。你可能已经有很多关于构造方法、对象、引用变量、访问数据域以及调用对象方法方面的问题，随后将会详细讨论这些话题。

复习题

- 9.1 描述对象和它的定义类之间的关系。
- 9.2 如何定义一个类？
- 9.3 如何声明一个对象引用变量？
- 9.4 如何创建一个对象？

9.4 使用构造方法构造对象

要点提示：构造方法在使用 `new` 操作符创建对象的时候被调用。

构造方法是一种特殊的方法。它们有以下三个特殊性：

- 构造方法必须具备和所在类相同的名字。
- 构造方法没有返回值类型，甚至连 `void` 也没有。
- 构造方法是在创建一个对象使用 `new` 操作符时调用的。构造方法的作用是初始化对象。

构造方法具有和定义它的类完全相同的名字。和所有其他方法一样，构造方法也可以重载（也就是说，可以有多个同名的构造方法，但它们要有不同的签名），这样更易于用不同的初始数据值来构造对象。

一个常见的错误就是将关键字 `void` 放在构造方法的前面。例如：

```
public void Circle() {
}
```

在这种情况下，`Circle()` 是一个方法，而不是构造方法。

构造方法是用来构造对象的。为了能够从一个类构造对象，使用 `new` 操作符调用这个类的构造方法，如下所示：

```
new ClassName(arguments);
```

例如：`new Circle()` 使用 `Circle` 类中定义的第一个构造方法创建一个 `Circle` 对象。

`new Circle(25)` 调用 `Circle` 类中定义的第二个构造方法创建一个 `Circle` 对象。

通常，一个类会提供一个没有参数的构造方法（例如：`Circle()`）。这样的构造方法称为无参构造方法（no-arg 或 no-argument constructor）。

一个类可以不定义构造方法。在这种情况下，类中隐含定义一个方法体为空的无参构造方法。这个构造方法称为默认构造方法（default constructor），当且仅当类中没有明确定义任何构造方法时才会自动提供它。

复习题

9.5 构造方法和普通方法之间的区别是什么？

9.6 什么时候类将有一个默认构造方法？

9.5 通过引用变量访问对象

要点提示：对象的数据和方法可以运用点操作符（.）通过对象的引用变量进行访问。

新创建的对象在内存中被分配空间。它们可以通过引用变量来访问。

9.5.1 引用变量和引用类型

对象是通过对象引用变量（reference variable）来访问的，该变量包含对对象的引用，使用如下语法格式声明这样的变量：

```
ClassName objectRefVar;
```

本质上来说，一个类是一个程序员定义的类型。类是一种引用类型（reference type），这意味着该类类型的变量都可以引用该类的一个实例。下面的语句声明变量 `myCircle` 的类型是 `Circle` 类型：

```
Circle myCircle;
```

变量 `myCircle` 能够引用一个 `Circle` 对象。下面的语句创建一个对象，并且将它的引用赋给变量 `myCircle`：

```
myCircle = new Circle();
```

采用如下所示的语法，可以写一条包括声明对象引用变量、创建对象以及将对象的引用赋值给这个变量的语句。

```
ClassName objectRefVar = new ClassName();
```

下面是一个例子：

```
Circle myCircle = new Circle();
```

变量 `myCircle` 中放的是对 `Circle` 对象的一个引用。

注意：从表面上看，对象引用变量中似乎存放了一个对象，但事实上，它只是包含了对该对象的引用。严格地讲，对象引用变量和对象是不同的，但是大多数情况下，这种差异是可以忽略的。因此，可以简单地说 `myCircle` 是一个 `Circle` 对象，而不用冗长地描述说，`myCircle` 是一个包含对 `Circle` 对象引用的变量。

注意：在 Java 中，数组被看作是对象。数组是用 `new` 操作符创建的。一个数组变量实际上是一个包含数组引用的变量。

9.5.2 访问对象的数据和方法

在面向对象编程中，对象成员可以引用该对象的数据域和方法。在创建一个对象之后，它的数据和方法可以使用点操作符 (.) 来访问和调用，该操作符也称为对象成员访问操作符 (object member access operator):

- `objectRefVar.dataField` 引用对象的数据域。
- `objectRefVar.method(arguments)` 调用对象的方法。

例如：`myCircle.radius` 引用 `myCircle` 的半径，而 `myCircle.getArea()` 调用 `myCircle` 的 `getArea` 方法。方法作为对象上的操作被调用。

数据域 `radius` 称作实例变量 (instance variable)，因为它依赖于某个具体的实例。基于同样的原因，`getArea` 方法称为实例方法 (instance method)，因为只能在具体的实例上调用它。调用对象上的实例方法的过程称为调用对象 (calling object)。

 **警告：**回想一下，我们曾经使用过 `Math.methodName(参数)` (例如：`Math.pow(3,2.5)`) 来调用 `Math` 类中的方法。那么能否用 `Circle.getArea()` 来调用 `getArea` 方法呢？答案是不能。`Math` 类中的所有方法都是用关键字 `static` 定义的静态方法。但是，`getArea()` 是实例方法，因此它是非静态的。它必须使用 `objectRefVar.methodName(参数)` 的方式 (例如：`myCircle.getArea()`) 从对象调用。更详细的解释将在 9.7 节中给出。

 **注意：**通常，我们创建一个对象，然后将它赋值给一个变量，之后就可以使用这个变量来引用对象。有时候，一个对象在创建之后并不需要引用。在这种情况下，可以创建一个对象，而并不将它明确地赋值给一个变量，如下所示：

```
new Circle();
```

或者

```
System.out.println("Area is " + new Circle(5).getArea());
```

前面的语句创建了一个 `Circle` 对象。后面的语句创建了一个 `Circle` 对象，然后调用它的 `getArea` 方法返回其面积。这种方式创建的对象称为匿名对象 (anonymous object)。

9.5.3 引用数据域和 null 值

数据域也可能是引用型的。例如：下面的 `Student` 类包含一个 `String` 类型的 `name` 数据域，`String` 是一个预定义的 Java 类。

```
class Student {
    String name; // name has the default value null
    int age; // age has the default value 0
    boolean isScienceMajor; // isScienceMajor has default value false
    char gender; // gender has default value '\u0000'
}
```

如果一个引用类型的数据域没有引用任何对象，那么这个数据域就有一个特殊的 Java 值 `null`。`null` 同 `true` 和 `false` 一样都是一个直接量。`true` 和 `false` 是 `boolean` 类型直接量，而 `null` 是引用类型直接量。

引用类型数据域的默认值是 `null`，数值类型数据域的默认值是 `0`，`boolean` 类型数据域的默认值是 `false`，而 `char` 类型数据域的默认值是 `'\u0000'`。但是，Java 没有给方法中的局部变量赋默认值。下面的代码显示 `Student` 对象中数据域 `name`、`age`、`isScienceMajor` 和

gender 的默认值:

```
class Test {
    public static void main(String[] args) {
        Student student = new Student();
        System.out.println("name? " + student.name);
        System.out.println("age? " + student.age);
        System.out.println("isScienceMajor? " + student.isScienceMajor);
        System.out.println("gender? " + student.gender);
    }
}
```

下面代码中的局部变量 x 和 y 都没有被初始化, 所以它会出现编译错误:

```
class Test {
    public static void main(String[] args) {
        int x; // x has no default value
        String y; // y has no default value
        System.out.println("x is " + x);
        System.out.println("y is " + y);
    }
}
```

警告: `NullPointerException` 是一种常见的运行时错误, 当调用值为 `null` 的引用变量上的方法时会发生此类异常。在通过引用变量调用一个方法之前, 确保先将对象引用赋值给这个变量 (参见复习题 9.11c)。

9.5.4 基本类型变量和引用类型变量的区别

每个变量都代表一个存储值的内存位置。声明一个变量时, 就是在告诉编译器这个变量可以存放什么类型的值。对基本类型变量来说, 对应内存所存储的值是基本类型值。对引用类型变量来说, 对应内存所存储的值是一个引用, 是对象的存储地址。例如: 如图 9-7 所示, `int` 型变量 `i` 的值就是 `int` 值 1, 而 `Circle` 对象 `c` 的值存的是一个引用, 它指明这个 `Circle` 对象的内容存储在内存中的什么位置。

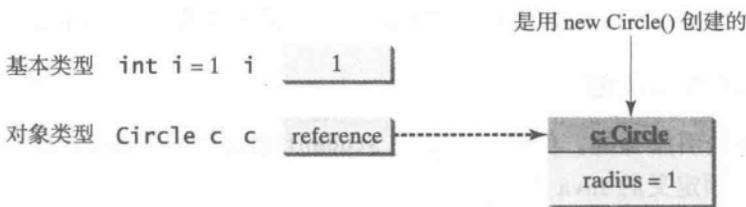


图 9-7 基本类型变量在内存中存储的是一个基本类型值, 而引用类型变量存储的是一个引用, 它指向对象在内存中的位置

将一个变量赋值给另一个变量时, 另一个变量就被赋予同样的值。对基本类型变量而言, 就是将一个变量的实际值赋给另一个变量。对引用类型变量而言, 就是将一个变量的引用赋给另一个变量。如图 9-8 所示, 赋值语句 `i=j` 将基本类型变量 `j` 的内容复制给基本类型变量 `i`。如图 9-9 所示, 对引用变量来讲, 赋值语句 `c1=c2` 是将 `c2` 的引用赋给 `c1`。赋值之后, 变量 `c1` 和 `c2` 指向同一个对象。

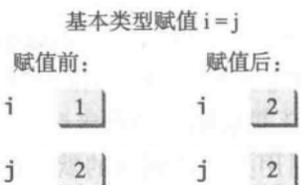
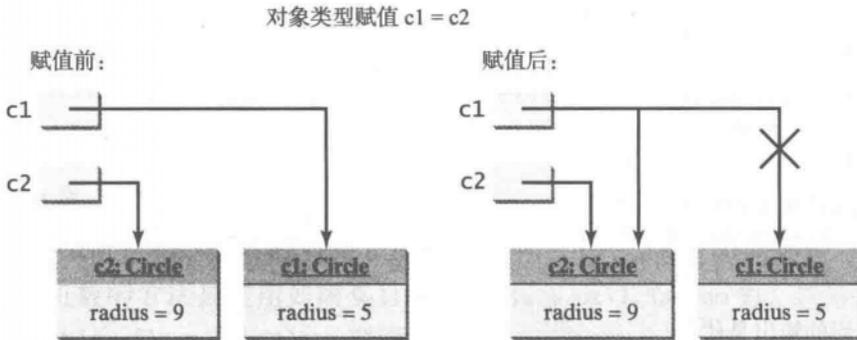


图 9-8 基本类型变量 `j` 复制到变量 `i` 中

图 9-9 引用变量 $c2$ 复制到变量 $c1$ 中

注意：如图 9-9 所示，执行完赋值语句 $c1=c2$ 之后， $c1$ 指向 $c2$ 所指的同一个对象。 $c1$ 以前引用的对象就不再有用，因此，现在它就成为垃圾 (garbage)。垃圾会占用内存空间。Java 运行系统会检测垃圾并自动回收它所占的空间，这个过程称为垃圾回收 (garbage collection)。

提示：如果你认为不再需要某个对象时，可以显式地给该对象的引用变量赋 `null` 值。如果该对象没有被任何引用变量所引用，Java 虚拟机将自动回收它所占的空间。

复习题

- 9.7 哪个操作符用于访问对象的数据域或者调用对象的方法？
- 9.8 什么是匿名对象？
- 9.9 什么是 `NullPointerException` ？
- 9.10 数组是对象还是基本类型值？数组可以包含对象类型的元素吗？描述数组元素的默认值。
- 9.11 下面每个程序中有什么错误？

```

1 public class ShowErrors {
2     public static void main(String[] args) {
3         ShowErrors t = new ShowErrors(5);
4     }
5 }

```

a)

```

1 public class ShowErrors {
2     public static void main(String[] args) {
3         ShowErrors t = new ShowErrors();
4         t.x();
5     }
6 }

```

b)

```

1 public class ShowErrors {
2     public void method1() {
3         Circle c;
4         System.out.println("What is radius "
5             + c.getRadius());
6         c = new Circle();
7     }
8 }

```

c)

```

1 public class ShowErrors {
2     public static void main(String[] args) {
3         C c = new C(5.0);
4         System.out.println(c.value);
5     }
6 }
7
8 class C {
9     int value = 2;
10 }

```

d)

- 9.12 下面代码有什么错误？

```

1 class Test {
2     public static void main(String[] args) {
3         A a = new A();
4         a.print();
5     }
6 }
7

```

```

8 class A {
9     String s;
10
11     A(String newS) {
12         s = newS;
13     }
14
15     public void print() {
16         System.out.print(s);
17     }
18 }

```

9.13 下面代码的输出是什么?

```

public class A {
    boolean x;

    public static void main(String[] args) {
        A a = new A();
        System.out.println(a.x);
    }
}

```

9.6 使用 Java 库中的类

 **要点提示:** Java API 包含了丰富的类的集合, 用于开发 Java 程序。

程序清单 9-1 声明了 SimpleCircle 类并从这个类创建了该类的对象。你将会频繁地使用到 Java 类库里的类来开发程序。本节将给出 Java 类库中一些类的例子。

9.6.1 Date 类

在程序清单 2-7 中, 我们已经学习了如何使用 `System.currentTimeMillis()` 来获得当前时间。使用除法和求余运算分解出当前的秒数、分钟数和小时数。Java 在 `java.util.Date` 类中还提供了与系统无关的对日期和时间的封装, 如图 9-10 所示。

java.util.Date	
+Date()	为当前时间创建一个 Date 对象
+Date(elapseTime: long)	为一个从格林威治时间 1970 年 1 月 1 日至今流逝的以毫秒为单位计算的给定时间创建 Date 对象
+toString(): String	返回一个代表日期和时间的字符串表示
+getTime(): long	返回从格林威治时间 1970 年 1 月 1 日至今流逝的毫秒数
+setTime(elapseTime: long): void	在对象中设置一个新的流逝时间

图 9-10 Date 对象表示特定的日期和时间

可以使用 Date 类中的无参构造方法为当前的日期和时间创建一个实例, 它的 `getTime()` 方法返回自从 GMT 时间 1970 年 1 月 1 日算起至今流逝的时间, 它的 `toString()` 方法返回日期和时间的字符串。例如, 下面的代码

```

java.util.Date date = new java.util.Date();
System.out.println("The elapsed time since Jan 1, 1970 is " +
    date.getTime() + " milliseconds");
System.out.println(date.toString());

```

显示输出为:

```
The elapsed time since Jan 1, 1970 is 1324903419651 milliseconds
Mon Dec 26 07:43:39 EST 2011
```

Date 类还有另外一个构造方法: `Date(long elapsedTime)`, 可以用它创建一个 Date 对象。该对象有一个从 GMT 时间 1970 年 1 月 1 日算起至今流逝的以毫秒为单位的给定时间。

9.6.2 Random 类

可以使用 `Math.random()` 获取一个 0.0 到 1.0 (不包括 1.0) 之间的随机 double 型值。另一种产生随机数的方法是使用如图 9-11 所示的 `java.util.Random` 类, 它可以产生一个 `int`、`long`、`double`、`float` 和 `boolean` 型值。

java.util.Random	
+Random()	以当前时间作为种子创建一个 Random 对象
+Random(seed: long)	以一个特定值作为种子创建一个 Random 对象
+nextInt(): int	返回一个随机的 int 值
+nextInt(n: int): int	返回一个 0 到 n (不包含 n) 之间的随机 int 类型的值
+nextLong(): long	返回一个随机的 long 值
+nextDouble(): double	返回一个 0.0 到 1.0 (不包含 1.0) 之间的随机 double 类型的值
+nextFloat(): float	返回一个 0.0F 到 1.0F (不包含 1.0F) 之间的随机 float 类型的值
+nextBoolean(): boolean	返回一个随机的 boolean 值

图 9-11 Random 对象可以用来产生随机值

创建一个 Random 对象时, 必须指定一个种子或者使用默认的种子。种子是一个用于初始化一个随机数字生成器的数字。无参构造方法使用当前已经逝去的时间作为种子, 创建一个 Random 对象。如果这两个 Random 对象有相同的种子, 那它们将产生相同的数列。例如: 下面的代码都用相同的种子 3 来产生两个 Random 对象。

```
Random random1 = new Random(3);
System.out.print("From random1: ");
for (int i = 0; i < 10; i++)
    System.out.print(random1.nextInt(1000) + " ");
```

```
Random random2 = new Random(3);
System.out.print("\nFrom random2: ");
for (int i = 0; i < 10; i++)
    System.out.print(random2.nextInt(1000) + " ");
```

这些代码产生相同的 int 类型的随机数列:

```
From random1: 734 660 210 581 128 202 549 564 459 961
From random2: 734 660 210 581 128 202 549 564 459 961
```

注意: 产生相同随机值序列的能力在软件测试以及其他许多应用中是很有用的。在软件测试中, 经常需要从一组固定顺序的随机数中来重复生成测试案例。

9.6.3 Point2D 类

Java API 在 `javafx.geometry` 包中有一个便于使用的 `Point2D` 类, 用于表示二维平面上的点。该类的 UML 图如图 9-12 所示。

可以为给定 x 和 y 坐标的点来创建一个 `Point2D` 对象, 使用 `distance` 方法计算该点到另外一个点之间的距离, 并且使用 `toString()` 方法来返回该点的字符串表示。程序清单 9-5

给出了一个使用该类的示例。

javafx.geometry.Point2D	
+Point2D(x: double, y: double)	用给定的 x 和 y 坐标来创建一个 Point2D 对象
+distance(x: double, y: double): double	返回该点到给定点 (x, y) 之间的距离
+distance(p: Point2D): double	返回该点到给定点 p 之间的距离
+getX(): double	返回该点的 x 坐标
+getY(): double	返回该点的 y 坐标
+toString(): String	返回该点的字符串表示

图 9-12 一个 Point2D 对象使用 x 和 y 坐标表示一个点

程序清单 9-5 TestPoint2D.java

```

1 import java.util.Scanner;
2 import javafx.geometry.Point2D;
3
4 public class TestPoint2D {
5     public static void main(String[] args) {
6         Scanner input = new Scanner(System.in);
7
8         System.out.print("Enter point1's x-, y-coordinates: ");
9         double x1 = input.nextDouble();
10        double y1 = input.nextDouble();
11        System.out.print("Enter point2's x-, y-coordinates: ");
12        double x2 = input.nextDouble();
13        double y2 = input.nextDouble();
14
15        Point2D p1 = new Point2D(x1, y1);
16        Point2D p2 = new Point2D(x2, y2);
17        System.out.println("p1 is " + p1.toString());
18        System.out.println("p2 is " + p2.toString());
19        System.out.println("The distance between p1 and p2 is " +
20            p1.distance(p2));
21    }
22 }

```

```

Enter point1's x-, y-coordinates: 1.5 5.5
Enter point2's x-, y-coordinates: -5.3 -4.4
p1 is Point2D [x = 1.5, y = 5.5]
p2 is Point2D [x = -5.3, y = -4.4]
The distance between p1 and p2 is 12.010412149464313

```

程序创建 Point2D 类的两个对象（第 15 ~ 16 行）。toString() 方法返回表述该对象的字符串（第 17 ~ 18 行）。调用 p1.distance(p2) 返回两个点之间的距离（第 20 行）。

复习题

- 9.14 如何为当前时间创建一个 Date？如何显示当前时间？
- 9.15 如何创建一个 Point2D？假设 p1 和 p2 是 Point2D 的两个实例，如何获得两点之间的距离？
- 9.16 哪些包包含类 Date、Random、Point2D、System 以及 Math？

9.7 静态变量、常量和方法

要点提示：静态变量被类中的所有对象所共享。静态方法不能访问类中的实例成员。

Circle 类的数据域 radius 称为一个实例变量。实例变量是绑定到类的某个特定实例的，

它是不能被同一个类的不同对象所共享的。例如，假设创建了如下的两个对象：

```
Circle circle1 = new Circle();
Circle circle2 = new Circle(5);
```

circle1 中的 radius 和 circle2 中的 radius 是不相关的，它们存储在不同的内存位置。circle1 中 radius 的变化不会影响 circle2 中的 radius，反之亦然。

如果想让一个类的所有实例共享数据，就要使用静态变量 (static variable)，也称为类变量 (class variable)。静态变量将变量值存储在一个公共的内存地址。因为它是公共的地址，所以如果某一个对象修改了静态变量的值，那么同一个类的所有对象都会受到影响。Java 支持静态方法和静态变量，无须创建类的实例就可以调用静态方法 (static method)。

修改 Circle 类，添加静态变量 numberOfObjects 统计创建的 Circle 对象的个数。当该类的第一个对象创建后，numberOfObjects 的值是 1。当第二个对象创建后，numberOfObjects 的值是 2。新 Circle 类的 UML 图如图 9-13 所示。Circle 类定义了实例变量 radius 和静态变量 numberOfObjects，还定义了实例方法 getRadius、setRadius 和 getArea 以及静态方法 getNumberOfObjects。(注意，在 UML 类图中，静态变量和静态方法都是以下划线标注的。)

UML 符号：

下划线：静态变量或方法

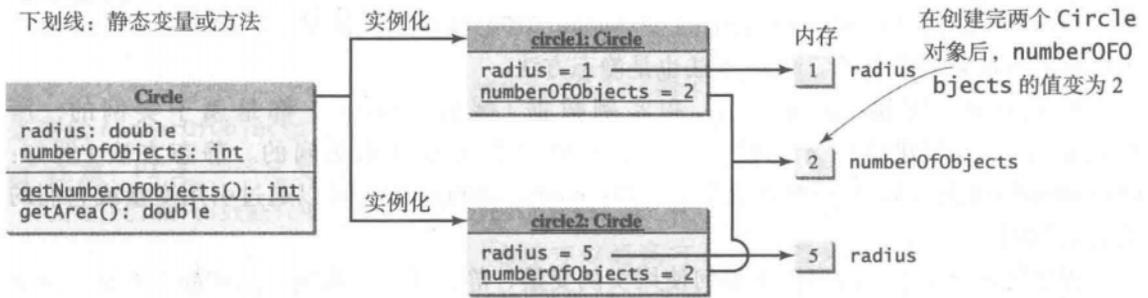


图 9-13 属于实例的实例变量存储在互不相关的内存中，静态变量是被同一个类的所有实例所共享的

要声明一个静态变量或定义一个静态方法，就要在这个变量或方法的声明中加上修饰符 static。静态变量 numberOfObjects 和静态方法 getNumberOfObjects() 可以如下声明：

```
static int numberOfObjects;

static int getNumberOfObjects() {
    return numberOfObjects;
}
```

类中的常量是被该类的所有对象所共享的。因此，常量应该声明为 final static，例如，Math 类中的常量 PI 是如下定义的：

```
final static double PI = 3.14159265358979323846;
```

新的名为 CircleWithStaticMembers 的圆类的声明如程序清单 9-6 所示。

程序清单 9-6 CircleWithStaticMembers.java

```
1 public class CircleWithStaticMembers {
2     /** The radius of the circle */
3     double radius;
```

```

4
5  /** The number of objects created */
6  static int numberOfObjects = 0;
7
8  /** Construct a circle with radius 1 */
9  CircleWithStaticMembers() {
10     radius = 1;
11     numberOfObjects++;
12 }
13
14 /** Construct a circle with a specified radius */
15 CircleWithStaticMembers(double newRadius) {
16     radius = newRadius;
17     numberOfObjects++;
18 }
19
20 /** Return numberOfObjects */
21 static int getNumberOfObjects() {
22     return numberOfObjects;
23 }
24
25 /** Return the area of this circle */
26 double getArea() {
27     return radius * radius * Math.PI;
28 }
29 }

```

CircleWithStaticMembers 类中的 getNumberOfObjects() 方法是一个静态方法。Math 类中所有的方法都是静态的。main 方法也是静态方法。

实例方法（例如：getArea()）和实例数据（例如：radius）都是属于实例的，所以它们在实例创建之后才能使用。它们是通过引用变量来访问的。静态方法（例如：getNumberOfObjects()）和静态数据（例如：numberOfObjects）可以通过引用变量或它们的类名来调用。

程序清单 9-7 中的程序演示如何使用实例变量、静态变量、实例方法和静态方法，还演示了使用它们的效果。

程序清单 9-7 TestCircleWithStaticMembers.java

```

1  public class TestCircleWithStaticMembers {
2      /** Main method */
3      public static void main(String[] args) {
4          System.out.println("Before creating objects");
5          System.out.println("The number of Circle objects is " +
6              CircleWithStaticMembers.numberOfObjects);
7
8          // Create c1
9          CircleWithStaticMembers c1 = new CircleWithStaticMembers();
10
11         // Display c1 BEFORE c2 is created
12         System.out.println("\nAfter creating c1");
13         System.out.println("c1: radius (" + c1.radius +
14             ") and number of Circle objects (" +
15             c1.numberOfObjects + ")");
16
17         // Create c2
18         CircleWithStaticMembers c2 = new CircleWithStaticMembers(5);
19
20         // Modify c1
21         c1.radius = 9;
22

```

```

23 // Display c1 and c2 AFTER c2 was created
24 System.out.println("\nAfter creating c2 and modifying c1");
25 System.out.println("c1: radius (" + c1.radius +
26     ") and number of Circle objects (" +
27     c1.numberOfObjects + ")");
28 System.out.println("c2: radius (" + c2.radius +
29     ") and number of Circle objects (" +
30     c2.numberOfObjects + ")");
31 }
32 }

```

Before creating objects

The number of Circle objects is 0

After creating c1

c1: radius (1.0) and number of Circle objects (1)

After creating c2 and modifying c1

c1: radius (9.0) and number of Circle objects (2)

c2: radius (5.0) and number of Circle objects (2)

编译 TestCircleWithStaticMembers.java 时，如果 CircleWithStaticMembers.java 在最后一次修改后之后还没有编译过的话，Java 编译器就会自动编译它。

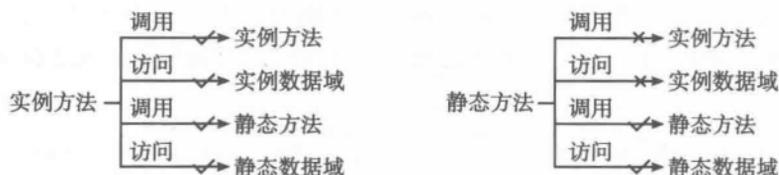
静态变量和方法可以在不创建对象的情况下访问。第 6 行显示对象的个数为 0，因为还没有创建任何对象。

main 方法创建两个圆，c1 和 c2(第 9、18 行)。c1 中的实例变量 radius 修改为 9(第 21 行)。这个变化不会影响 c2 中的实例变量 radius，因为这两个实例变量是独立的。c1 创建之后静态变量 numberOfObjects 变成 1(第 9 行)，而 c2 创建之后 numberOfObjects 变成 2(第 18 行)。

注意：PI 是一个定义在 Math 中的常量，可以使用 Math.PI 来访问这个常量。最好使用 CircleWithStaticMembers.numberOfObjects 来代替 c1.numberOfObjects(第 27 行)和 c2.numberOfObjects(第 30 行)。这样可以提高可读性，因为其他程序员可以很容易地识别静态变量。也可以用 CircleWithStaticMembers.getNumberOfObjects() 替换掉 CircleWithStaticMembers.numberOfObjects。

提示：使用“类名.方法名(参数)”的方式调用静态方法，使用“类名.静态变量”的方式访问静态变量。这会提高可读性，因为可以很容易地识别出类中的静态方法和数据。

实例方法可以调用实例方法和静态方法，以及访问实例数据域或者静态数据域。静态方法可以调用静态方法以及访问静态数据域。然而，静态方法不能调用实例方法或者访问实例数据域，因为静态方法和静态数据域不属于某个特定的对象。静态成员和实例成员的关系总结在下图中。



例如，下面给出的代码是错误的。

```

1 public class A {
2     int i = 5;
3     static int k = 2;
4
5     public static void main(String[] args) {

```

```

6     int j = i; // Wrong because i is an instance variable
7     m1(); // Wrong because m1() is an instance method
8 }
9
10    public void m1() {
11        // Correct since instance and static variables and methods
12        // can be used in an instance method
13        i = i + k + m2(i, k);
14    }
15
16    public static int m2(int i, int j) {
17        return (int)(Math.pow(i, j));
18    }
19 }

```

🔑 **注意：**如果用下面的新的代码替换上面的代码，程序就是正确的，因为实例数据域 *i* 和方法 *m1* 是通过对象 *a* 访问的（第 7~8 行）：

```

1    public class A {
2        int i = 5;
3        static int k = 2;
4
5        public static void main(String[] args) {
6            A a = new A();
7            int j = a.i; // OK, a.i accesses the object's instance variable
8            a.m1(); // OK. a.m1() invokes the object's instance method
9        }
10
11        public void m1() {
12            i = i + k + m2(i, k);
13        }
14
15        public static int m2(int i, int j) {
16            return (int)(Math.pow(i, j));
17        }
18    }

```

🔑 **设计指南：**如何判断一个变量或方法应该是实例的还是静态的？如果一个变量或方法依赖于类的某个具体实例，那就应该将它定义为实例变量或实例方法。如果一个变量或方法不依赖于类的某个具体实例，就应该将它定义为静态变量或静态方法。例如：每个圆都有自己的半径，半径都依赖于某个具体的圆。因此，半径 *radius* 就是 *Circle* 类的一个实例变量。由于 *getArea* 方法依赖于某个具体的圆，所以，它也是一个实例方法。在 *Math* 类中没有一个是依赖于一个特定实例的，例如：*random*、*pow*、*sin* 和 *cos*。因此，这些方法都是静态方法。*main* 方法也是静态的，可以从类中直接调用。

🔑 **警告：**一个常见的设计错误就是将一个本应该声明为静态的方法声明为实例方法。例如：方法 *factorial(int n)* 应该定义为静态的，如下所示，因为它不依赖于任何具体的实例。

```

public class Test {
    public int factorial(int n) {
        int result = 1;
        for (int i = 1; i <= n; i++)
            result *= i;

        return result;
    }
}

```

a) 错误的设计

```

public class Test {
    public static int factorial(int n) {
        int result = 1;
        for (int i = 1; i <= n; i++)
            result *= i;

        return result;
    }
}

```

b) 正确的设计

复习题

9.17 假设 F 类在 a 中定义，f 是 F 的一个实例，那么 b 中的哪些语句是正确的？

```
public class F {
    int i;
    static String s;

    void imethod() {
    }

    static void smethod() {
    }
}
```

a)

```
System.out.println(f.i);
System.out.println(f.s);
f.imethod();
f.smethod();
System.out.println(F.i);
System.out.println(F.s);
F.imethod();
F.smethod();
```

b)

9.18 如果合适的话，在出现 ? 的位置添加 static 关键字。

```
public class Test {
    int count;

    public ? void main(String[] args) {
        ...
    }

    public ? int getCount() {
        return count;
    }

    public ? int factorial(int n) {
        int result = 1;
        for (int i = 1; i <= n; i++)
            result *= i;
        return result;
    }
}
```

9.19 能否从静态方法中调用实例方法或引用一个实例变量？能否从实例方法中调用静态方法或引用一个静态变量？下面代码错在哪里？

```
1 public class C {
2     public static void main(String[] args) {
3         method1();
4     }
5
6     public void method1() {
7         method2();
8     }
9
10    public static void method2() {
11        System.out.println("What is radius " + c.getRadius());
12    }
13
14    Circle c = new Circle();
15 }
```

9.8 可见性修饰符

要点提示：可见性修饰符可以用于确定一个类以及它的成员的可见性。

可以在类、方法和数据域前使用 public 修饰符，表示它们可以被任何其他的类访问。如果没有使用可见性修饰符，那么则默认类、方法和数据域是可以被同一个包中的任何一个类访问的。这称作包私有 (package-private) 或包内访问 (package-access)。

🔑 **注意：**包可以用来组织类。为了完成这个目标，需要在程序中首先出现下面这行语句，在这行语句之前不能有注释也不能有空白：

```
package packageName;
```

如果定义类时没有声明包，就表示把它放在默认包中。

Java 建议最好将类放入包中，而不要使用默认包。但是，本书为了简化问题使用的是默认包。关于包的更多的信息，参见补充材料 III.E。

除了 `public` 和默认可见性修饰符，Java 还为类成员提供 `private` 和 `protected` 修饰符。本节介绍 `private` 修饰符。修饰符 `protected` 将在 11.14 节介绍。

`private` 修饰符限定方法和数据域只能在它自己的类中被访问。图 9-14 演示类 C1 中的公共的、默认的和私有的数据域或方法能否被同一个包内的类 C2 访问，以及能否被不在同一个包内的类 C3 访问。

<pre>package p1; public class C1 { public int x; int y; private int z; public void m1() { } void m2() { } private void m3() { } }</pre>	<pre>package p1; public class C2 { void aMethod() { C1 o = new C1(); can access o.x; can access o.y; cannot access o.z; can invoke o.m1(); can invoke o.m2(); cannot invoke o.m3(); } }</pre>	<pre>package p2; public class C3 { void aMethod() { C1 o = new C1(); can access o.x; cannot access o.y; cannot access o.z; can invoke o.m1(); cannot invoke o.m2(); cannot invoke o.m3(); } }</pre>
---	---	---

图 9-14 私有的修饰符将访问权限限定在它自己的类内，默认修饰符将访问权限限定在包内，而公共的修饰符可以无限制的访问

如果一个类没有被定义为公共类，那么它只能在同一个包内被访问。如图 9-15 所示，C2 可以访问 C1，而 C3 不能访问 C1。

<pre>package p1; class C1 { ... }</pre>	<pre>package p1; public class C2 { can access C1 }</pre>	<pre>package p2; public class C3 { cannot access C1; can access C2; }</pre>
--	---	--

图 9-15 一个非公共类具有包访问性

可见性修饰符指明类中的数据域和方法是否能在该类之外被访问。在该类之内，对数据域和方法的访问是没有任何限制的。如图 9-16b 所示，C 类的对象 `c` 不能引用它的私有成员，因为 `c` 在 `Test` 类中。如图 9-16a 所示，C 类的对象 `c` 可以访问它的私有成员，因为 `c` 在自己的类内定义。

🔑 **警告：**修饰符 `private` 只能应用在类的成员上。修饰符 `public` 可以应用在类或类的成员上。在局部变量上使用修饰符 `public` 和 `private` 都会导致编译错误。

🔑 **注意：**大多数情况下，构造方法应该是公共的。但是，如果想防止用户创建类的实例，就该使用私有构造方法。例如：因为 `Math` 类的所有数据域和方法都是静态的，所以没必要

创建 Math 类的实例。为了防止用户从 Math 类创建对象，在 java.lang.Math 中的构造方法定义为如下所示：

```
private Math() {
}
```

```
public class C {
    private boolean x;

    public static void main(String[] args) {
        C c = new C();
        System.out.println(c.x);
        System.out.println(c.convert());
    }

    private int convert() {
        return x ? 1 : -1;
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        C c = new C();
        System.out.println(c.x);
        System.out.println(c.convert());
    }
}
```



a) 这里没有问题，因为对象 c 在类 C 中使用

b) 这里有错误，因为 x 和 convert 在类 C 中是私有的

图 9-16 如果一个对象是在它自己的类中定义的，那么这个对象可以访问它的私有成员

9.9 数据域封装

要点提示：将数据域设为私有保护数据，并且使类易于维护。

在程序清单 9-6 中，CircleWithStaticMembers 类的数据域 radius 和 numberOfObjects 可以直接修改（例如：c1.radius = 5 或 CircleWithStaticMembers.numberOfObjects = 10）。这不是一个好的做法，原因有两点：

- 首先，数据可能被篡改。例如：numberOfObjects 是用来统计被创建的对象个数的，但是它可能会被错误地设置为一个任意值（例如：CircleWithStaticMembers.numberOfObjects = 10）。
- 其次，它使类变得难于维护，同时容易出现错误。假如在其他程序已经使用 CircleWithStaticMembers 类之后想修改半径以确保半径是一个非负数。因为使用该类的客户可以直接修改 radius（例如：myCircle.radius=-5），所以，不仅要修改 CircleWithStaticMembers，而且还要修改使用 CircleWithStaticMembers 的这些程序。

为了避免对数据域的直接修改，应该使用 private 修饰符将数据域声明为私有的，这称为数据域封装（data field encapsulation）。

在定义私有数据域类外的对象是不能访问这个数据域的。但是经常会有客户端需要存取、修改数据域的情况。为了能够访问私有数据域，可以提供一个 get 方法返回数据域的值。为了能够更新一个数据域，可以提供一个 set 方法给数据域设置新值。get 方法也被称为访问器（accessor），而 set 方法称为修改器（mutator）。

get 方法有如下签名：

```
public returnType getPropertyName()
```

如果返回值类型是 boolean 型，习惯上如下定义 get 方法：

```
public boolean isPropertyName()
```

set 方法有如下签名:

```
public void setPropertyName(dataType propertyValue)
```

现在来创建一个新的圆类, 半径设置为私有数据域, 并有相关的访问器和修改器。类图如图 9-17 所示。程序清单 9-8 中定义一个名为 CircleWithPrivateDataFields 的新的圆类。

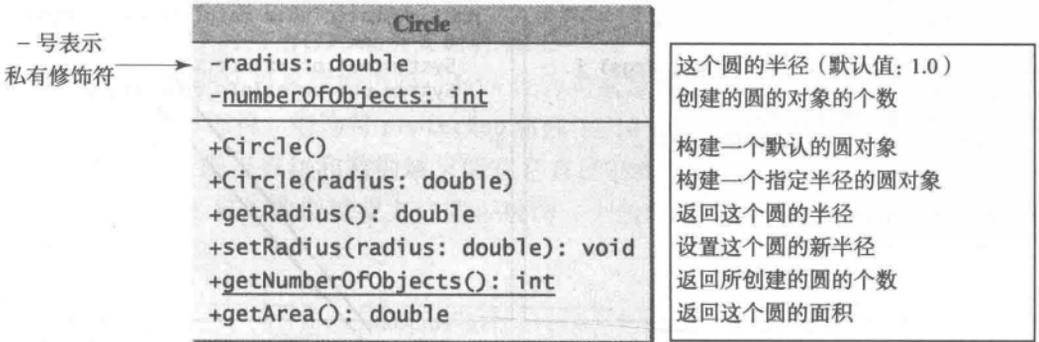


图 9-17 Circle 类封装了圆的属性并提供了 get/set 方法以及其他方法

程序清单 9-8 CircleWithPrivateDataFields.java

```
1 public class CircleWithPrivateDataFields {
2     /** The radius of the circle */
3     private double radius = 1;
4
5     /** The number of objects created */
6     private static int numberOfObjects = 0;
7
8     /** Construct a circle with radius 1 */
9     public CircleWithPrivateDataFields() {
10        numberOfObjects++;
11    }
12
13    /** Construct a circle with a specified radius */
14    public CircleWithPrivateDataFields(double newRadius) {
15        radius = newRadius;
16        numberOfObjects++;
17    }
18
19    /** Return radius */
20    public double getRadius() {
21        return radius;
22    }
23
24    /** Set a new radius */
25    public void setRadius(double newRadius) {
26        radius = (newRadius >= 0) ? newRadius : 0;
27    }
28
29    /** Return numberOfObjects */
30    public static int getNumberOfObjects() {
31        return numberOfObjects;
32    }
33
34    /** Return the area of this circle */
35    public double getArea() {
36        return radius * radius * Math.PI;
37    }
38 }
```

`getRadius()` 方法 (第 20 ~ 22 行) 返回半径值, `setRadius(newRadius)` 方法 (第 25 ~ 27 行) 给对象设置新的半径, 如果新半径为负, 就将这个对象的半径设置为 0。因为这些方法是读取和修改半径的唯一途径, 所以, 你完全控制了如何访问 `radius` 属性。如果必须改变这些方法的实现, 是不需要改变使用它们的客户程序的。这会使类更易于维护。

程序清单 9-9 给出一个客户程序, 它使用 `Circle` 类创建一个 `Circle` 对象, 然后使用 `setRadius` 方法修改它的半径。

程序清单 9-9 TestCircleWithPrivateDataFields.java

```
1 public class TestCircleWithPrivateDataFields {
2     /** Main method */
3     public static void main(String[] args) {
4         // Create a circle with radius 5.0
5         CircleWithPrivateDataFields myCircle =
6             new CircleWithPrivateDataFields(5.0);
7         System.out.println("The area of the circle of radius "
8             + myCircle.getRadius() + " is " + myCircle.getArea());
9
10        // Increase myCircle's radius by 10%
11        myCircle.setRadius(myCircle.getRadius() * 1.1);
12        System.out.println("The area of the circle of radius "
13            + myCircle.getRadius() + " is " + myCircle.getArea());
14
15        System.out.println("The number of objects created is "
16            + CircleWithPrivateDataFields.getNumberOfObjects());
17    }
18 }
```

数据域 `radius` 被声明为私有的。私有数据只能在定义它们的类中被访问。不能在客户程序中使用 `myCircle.radius`。如果试图从客户程序访问私有数据, 将会产生编译错误。

由于 `numberOfObjects` 是私有的, 所以它是不能修改的。这就制止了篡改行为。例如: 用户不能设置 `numberOfObjects` 为 100。要使这个值为 100 的唯一方法就是创建 100 个 `Circle` 类的对象。

假如通过把 `TestCircleWithPrivateDataFields` 类中的 `main` 方法移到 `Circle` 类中, 实现将 `TestCircleWithPrivateDataFields` 类和 `Circle` 类组合成一个类, 那么可以在 `main` 方法中使用 `myCircle.radius` 吗? 参见复习题 9.22 找到这个答案。

🔧 设计指南: 为防止数据被篡改以及使类更易于维护, 最好将数据域声明为私有的。

👉 复习题

- 9.20 什么是访问器方法? 什么是修改器方法? 访问器方法和修改器方法的命名习惯是什么?
- 9.21 数据域封装的优点是什么?
- 9.22 在下面的代码中, `Circle` 类中的 `radius` 是私有的, 而 `myCircle` 是 `Circle` 类的一个对象, 下面高亮的代码会导致什么问题吗? 如果有问题的话, 解释为什么。

```
public class Circle {
    private double radius = 1;

    /** Find the area of this circle */
    public double getArea() {
        return radius * radius * Math.PI;
    }

    public static void main(String[] args) {
```

```

    Circle myCircle = new Circle();
    System.out.println("Radius is " + myCircle.radius);
}
}

```

9.10 向方法传递对象参数

☞ 要点提示：给方法传递一个对象，是将对象的引用传递给方法。

可以将对象传递给方法。同传递数组一样，传递对象实际上是传递对象的引用。下面的代码将 `myCircle` 对象作为参数传递给 `printCircle` 方法：

```

1 public class Test {
2     public static void main(String[] args) {
3         // CircleWithPrivateDataFields is defined in Listing 9.8
4         CircleWithPrivateDataFields myCircle = new
5             CircleWithPrivateDataFields(5.0);
6         printCircle(myCircle);
7     }
8
9     public static void printCircle(CircleWithPrivateDataFields c) {
10        System.out.println("The area of the circle of radius "
11            + c.getRadius() + " is " + c.getArea());
12    }
13 }

```

Java 只有一种参数传递方式：值传递（pass-by-value）。在上面的代码中，`myCircle` 的值被传递给 `printCircle` 方法。这个值就是一个对 `Circle` 对象的引用值。

程序清单 9-10 中的程序展示了传递基本类型值和传递引用值的差异。

程序清单 9-10 TestPassObject.java

```

1 public class TestPassObject {
2     /** Main method */
3     public static void main(String[] args) {
4         // Create a Circle object with radius 1
5         CircleWithPrivateDataFields myCircle =
6             new CircleWithPrivateDataFields(1);
7
8         // Print areas for radius 1, 2, 3, 4, and 5.
9         int n = 5;
10        printAreas(myCircle, n);
11
12        // See myCircle.radius and times
13        System.out.println("\n" + "Radius is " + myCircle.getRadius());
14        System.out.println("n is " + n);
15    }
16
17    /** Print a table of areas for radius */
18    public static void printAreas(
19        CircleWithPrivateDataFields c, int times) {
20        System.out.println("Radius \t\tArea");
21        while (times >= 1) {
22            System.out.println(c.getRadius() + "\t\t" + c.getArea());
23            c.setRadius(c.getRadius() + 1);
24            times--;
25        }
26    }
27 }

```

Radius	Area
1.0	3.141592653589793
2.0	12.566370614359172
3.0	29.274333882308138
4.0	50.26548245743669
5.0	79.53981633974483

Radius is 6.0
n is 5

CircleWithPrivateDataFields 类是在程序清单 9-8 中定义的。这个程序使用 CircleWithPrivateDataFields 类的一个对象 myCircle 和 n 的整数值调用 printAreas (myCircle,n) 方法 (第 10 行), 打印出半径为 1、2、3、4 和 5 的圆面积所构成的表格, 如样本输出所示。

图 9-18 展示执行程序的这个方法的过程中的调用堆栈。注意, 对象是存储在堆中的 (参见第 7.6 小节)。

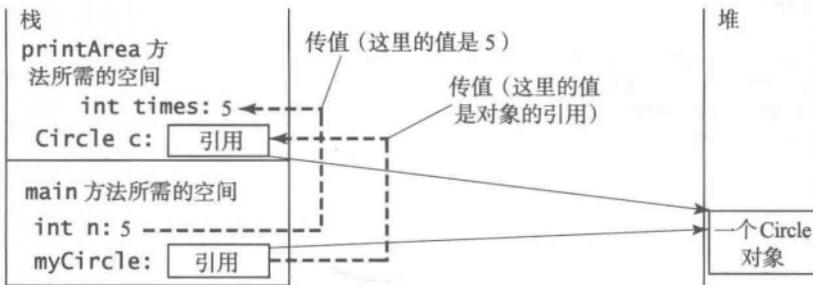


图 9-18 n 的值被传递给 times, 而 myCircle 的引用值被传递给 printAreas 方法中的 c

当传递基本数据类型参数时, 传递的是实参的值。在这种情况下, n(5) 的值就被传递给 times。在 printAreas 方法内, times 的内容改变, 这并不会影响 n 的内容。

传递引用类型的参数时, 传递的是对象的引用。在这种情况下, c 具有与 myCircle 相同的引用值。因此, 通过在 printAreas 方法内部的 c 与在方法外的 myCircle 来改变对象的属性, 效果是一样的。引用上的传值在语义上最好描述为传共享 (pass-by-sharing), 也就是说, 在方法中引用的对象和传递的对象是一样的。

复习题

9.23 描述传递基本类型参数和传递引用类型参数的区别, 并给出下面程序的输出:

```
public class Test {
    public static void main(String[] args) {
        Count myCount = new Count();
        int times = 0;

        for (int i = 0; i < 100; i++)
            increment(myCount, times);

        System.out.println("count is " + myCount.count);
        System.out.println("times is " + times);
    }

    public static void increment(Count c, int times) {
        c.count++;
        times++;
    }
}
```

```
public class Count {
    public int count;

    public Count(int c) {
        count = c;
    }

    public Count() {
        count = 1;
    }
}
```

9.24 显示下面程序的输出:

```
public class Test {
    public static void main(String[] args) {
        Circle circle1 = new Circle(1);
        Circle circle2 = new Circle(2);

        swap1(circle1, circle2);
        System.out.println("After swap1: circle1 = " +
            circle1.radius + " circle2 = " + circle2.radius);

        swap2(circle1, circle2);
        System.out.println("After swap2: circle1 = " +
            circle1.radius + " circle2 = " + circle2.radius);
    }

    public static void swap1(Circle x, Circle y) {
        Circle temp = x;
        x = y;
        y = temp;
    }

    public static void swap2(Circle x, Circle y) {
        double temp = x.radius;
        x.radius = y.radius;
        y.radius = temp;
    }
}

class Circle {
    double radius;

    Circle(double newRadius) {
        radius = newRadius;
    }
}
```

9.25 显示下面程序的输出:

```
public class Test {
    public static void main(String[] args) {
        int[] a = {1, 2};
        swap(a[0], a[1]);
        System.out.println("a[0] = " + a[0]
            + " a[1] = " + a[1]);
    }

    public static void swap(int n1, int n2) {
        int temp = n1;
        n1 = n2;
        n2 = temp;
    }
}
```

a)

```
public class Test {
    public static void main(String[] args) {
        int[] a = {1, 2};
        swap(a);
        System.out.println("a[0] = " + a[0]
            + " a[1] = " + a[1]);
    }

    public static void swap(int[] a) {
        int temp = a[0];
        a[0] = a[1];
        a[1] = temp;
    }
}
```

b)

```

public class Test {
    public static void main(String[] args) {
        T t = new T();
        swap(t);
        System.out.println("e1 = " + t.e1
            + " e2 = " + t.e2);
    }

    public static void swap(T t) {
        int temp = t.e1;
        t.e1 = t.e2;
        t.e2 = temp;
    }
}

class T {
    int e1 = 1;
    int e2 = 2;
}

```

c)

```

public class Test {
    public static void main(String[] args) {
        T t1 = new T();
        T t2 = new T();
        System.out.println("t1's i = " +
            t1.i + " and j = " + t1.j);
        System.out.println("t2's i = " +
            t2.i + " and j = " + t2.j);
    }
}

class T {
    static int i = 0;
    int j = 0;

    T() {
        i++;
        j = 1;
    }
}

```

d)

9.26 显示下面程序的输出:

```

import java.util.Date;

public class Test {
    public static void main(String[] args) {
        Date date = null;
        m1(date);
        System.out.println(date);
    }

    public static void m1(Date date) {
        date = new Date();
    }
}

```

a)

```

import java.util.Date;

public class Test {
    public static void main(String[] args) {
        Date date = new Date(1234567);
        m1(date);
        System.out.println(date.getTime());
    }

    public static void m1(Date date) {
        date = new Date(7654321);
    }
}

```

b)

```

import java.util.Date;

public class Test {
    public static void main(String[] args) {
        Date date = new Date(1234567);
        m1(date);
        System.out.println(date.getTime());
    }

    public static void m1(Date date) {
        date.setTime(7654321);
    }
}

```

c)

```

import java.util.Date;

public class Test {
    public static void main(String[] args) {
        Date date = new Date(1234567);
        m1(date);
        System.out.println(date.getTime());
    }

    public static void m1(Date date) {
        date = null;
    }
}

```

d)

9.11 对象数组

 **要点提示:** 数组既可以存储基本类型值, 也可以存储对象。

在第 7 章中描述了如何创建基本类型元素的数组。也可以创建对象数组。例如, 下面的语句声明并创建了 10 个 Circle 对象的数组:

```
Circle[] circleArray = new Circle[10];
```

为了初始化数组 `circleArray`，可以使用如下的 `for` 循环：

```
for (int i = 0; i < circleArray.length; i++) {
    circleArray[i] = new Circle();
}
```

对象的数组实际上是引用变量的数组。因此，调用 `circleArray[1].getArea()` 实际上调用了两个层次的引用，如图 9-19 所示。`circleArray` 引用了整个数组，`circleArray[1]` 引用了一个 `Circle` 对象。

注意：当使用 `new` 操作符创建对象数组后，这个数组中的每个元素都是默认值为 `null` 的引用变量。

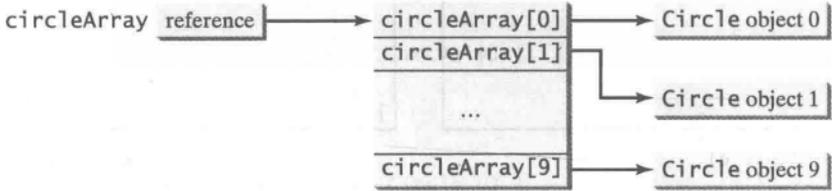


图 9-19 在对象数组中，数组的每个元素都包含一个对象的引用

程序清单 9-11 给出了一个例子，演示如何使用对象数组。这个程序求圆的数组的总面积。程序创建 5 个 `Circle` 对象组成的数组 `circleArray`，接着使用随机值初始化这些圆的半径，然后显示数组中的圆的总面积。

程序清单 9-11 TotalArea.java

```
1 public class TotalArea {
2     /** Main method */
3     public static void main(String[] args) {
4         // Declare circleArray
5         CircleWithPrivateDataFields[] circleArray;
6
7         // Create circleArray
8         circleArray = createCircleArray();
9
10        // Print circleArray and total areas of the circles
11        printCircleArray(circleArray);
12    }
13
14    /** Create an array of Circle objects */
15    public static CircleWithPrivateDataFields[] createCircleArray() {
16        CircleWithPrivateDataFields[] circleArray =
17            new CircleWithPrivateDataFields[5];
18
19        for (int i = 0; i < circleArray.length; i++) {
20            circleArray[i] =
21                new CircleWithPrivateDataFields(Math.random() * 100);
22        }
23
24        // Return Circle array
25        return circleArray;
26    }
27
28    /** Print an array of circles and their total area */
29    public static void printCircleArray(
30        CircleWithPrivateDataFields[] circleArray) {
31        System.out.printf("%-30s%-15s\n", "Radius", "Area");
32        for (int i = 0; i < circleArray.length; i++) {
```

```

33     System.out.printf("%-30f%-15f\n", circleArray[i].getRadius(),
34         circleArray[i].getArea());
35     }
36
37     System.out.println("-----");
38
39     // Compute and display the result
40     System.out.printf("%-30s%-15f\n", "The total area of circles is",
41         sum(circleArray) );
42 }
43
44 /** Add circle areas */
45 public static double sum(CircleWithDataFields[] circleArray) {
46     // Initialize sum
47     double sum = 0;
48
49     // Add areas to sum
50     for (int i = 0; i < circleArray.length; i++)
51         sum += circleArray[i].getArea();
52
53     return sum;
54 }
55 }

```

Radius	Area
70.577708	15649.941866
44.152266	6124.291736
24.867853	1942.792644
5.680718	101.380949
36.734246	4239.280350

The total area of circles is 28056.687544

程序调用 `createCircleArray()` 方法 (第 8 行) 创建一个由 5 个圆对象组成的数组。本章介绍了几个圆类。本例使用的是 9.9 节中介绍的 `CircleWithDataFields` 类。

圆的半径是使用 `Math.random()` 方法随机生成的 (第 21 行)。`createCircleArray` 方法返回一个 `CircleWithDataFields` 对象的数组 (第 25 行)。这个数组作为参数传给 `printCircleArray` 方法, 该方法显示每个圆的半径和面积以及它们的总面积。

圆的面积之和是用 `sum` 方法计算出来的 (第 41 行), 该方法以 `CircleWithDataFields` 对象的数组为参数, 返回的是 `double` 型的总面积值。

复习题

9.27 下面的代码有什么错误?

```

1 public class Test {
2     public static void main(String[] args) {
3         java.util.Date[] dates = new java.util.Date[10];
4         System.out.println(dates[0]);
5         System.out.println(dates[0].toString());
6     }
7 }

```

9.12 不可变对象和类

 **要点提示:** 可以定义不可变类来产生不可变对象。不可变对象的内容不能被改变。

通常，创建一个对象后，它的内容是允许之后改变的。有时候也需要创建一个一旦创建其内容就不能再改变的对象。我们称这种对象为一个不可变对象 (immutable object)，而它的类就称为不可变类 (immutable class)。例如：String 类就是不可变的。如果把程序清单 9-9 中 CircleWithPrivateDataFields 类的 set 方法删掉，该类就变成不可变类，因为半径是私有的，所以如果没有 set 方法，它的值就不能再改变。

如果一个类是不可变的，那么它的所有数据域必须都是私有的，而且没有对任何一个数据域提供公共的 set 方法。一个类的所有数据都是私有的且没有修改器并不意味着它一定是不可变类。例如：下面的 Student 类，它的所有数据域都是私有的，而且也没有 set 方法，但它不是一个不可变的类。

```

1  public class Student {
2      private int id;
3      private String name;
4      private java.util.Date dateCreated;
5
6      public Student(int ssn, String newName) {
7          id = ssn;
8          name = newName;
9          dateCreated = new java.util.Date();
10     }
11
12     public int getId() {
13         return id;
14     }
15
16     public String getName() {
17         return name;
18     }
19
20     public java.util.Date getDateCreated() {
21         return dateCreated;
22     }
23 }

```

如下面的代码所示，使用 getDateCreated() 方法返回数据域 dateCreated。它是对 Date 对象的一个引用。通过这个引用，可以改变 dateCreated 的值。

```

public class Test {
    public static void main(String[] args) {
        Student student = new Student(111223333, "John");
        java.util.Date dateCreated = student.getDateCreated();
        dateCreated.setTime(200000); // Now dateCreated field is changed!
    }
}

```

要使一个类成为不可变的，它必须满足下面的要求：

- 所有数据域都是私有的。
- 没有修改器方法。
- 没有一个返回指向可变数据域的引用的访问器方法。

有兴趣的读者可以参考补充材料 III.U 获得不可变对象的更多信息。

复习题

- 9.28 如果类中仅包含私有数据域并且没有设置 set 方法，该类可以改变吗？
- 9.29 如果类中的所有数据域是私有的基本数据类型，并且类中没有包含任何 set 方法，该类可以改变吗？

9.30 下面的类可以改变吗?

```
public class A {
    private int[] values;

    public int[] getValues() {
        return values;
    }
}
```

9.13 变量的作用域

🔑 要点提示: 实例变量和静态变量的作用域是整个类, 无论变量是在哪里声明的。

在 6.9 节中讨论了局部变量和它们的作用域。局部变量的声明和使用都在一个方法的内部。本节将在类的范围内讨论所有变量的作用域规则。

一个类的实例变量和静态变量称为类变量 (class's variables) 或数据域 (data field)。在方法内部定义的变量称为局部变量。无论在何处声明, 类变量的作用域都是整个类。类的变量和方法可以在类中以任意顺序出现, 如图 9-20a 所示。但是当—个数据域是基于对另一个数据域的引用来进行初始化时则不是这样。在这种情况下, 必须首先声明另一个数据域, 如图 9-20b 所示。为保持一致性, 本书在类的开头就声明数据域。

```
public class Circle {
    public double findArea() {
        return radius * radius * Math.PI;
    }

    private double radius = 1;
}
```

a) 变量 `radius` 和方法 `findArea()` 可以以任意顺序声明

```
public class F {
    private int i ;
    private int j = i + 1;
}
```

b) `i` 必须在 `j` 之前声明, 因为 `j` 的初始值依赖于 `i`

图 9-20 类的成员可以按任意顺序声明, 只有一种例外情况

类变量只能声明一次, 但是在一个方法内不同的非嵌套块中, 可以多次声明相同的变量名。

如果一个局部变量和一个类变量具有相同的名字, 那么局部变量优先, 而同名的类变量将被隐藏 (hidden)。例如: 在下面的程序中, `x` 被定义为一个实例变量, 也在方法中被定义为局部变量。

```
public class F {
    private int x = 0; // Instance variable
    private int y = 0;

    public F() {
    }

    public void p() {
        int x = 1; // Local variable
        System.out.println("x = " + x);
        System.out.println("y = " + y);
    }
}
```

假设 `f` 是 `F` 的一个实例, 那么 `f.p()` 的打印输出是什么呢? `f.p()` 的打印输出是: `x` 为 1, `y` 为 0。其原因如下:

- x 被声明为类中初值为 0 的数据域，但是它在方法 p() 中又被声明了一次，初值为 1。System.out.println 语句中引用的 x 是后者。
- y 在方法 p() 的外部声明，但在方法内部也是可访问的。

提示：为避免混淆和错误，除了方法中的参数，不要将实例变量或静态变量的名字作为局部变量名。

复习题

9.31 下面程序的输出是什么？

```
public class Test {
    private static int i = 0;
    private static int j = 0;

    public static void main(String[] args) {
        int i = 2;
        int k = 3;

        {
            int j = 3;
            System.out.println("i + j is " + i + j);
        }

        k = i + j;
        System.out.println("k is " + k);
        System.out.println("j is " + j);
    }
}
```

9.14 this 引用

要点提示：关键字 this 引用对象自身。它也可以在构造方法内部用于调用同一个类的其他构造方法。

关键字 this 是指向调用对象本身的引用名。可以用 this 关键字引用对象的实例成员。例如，下面 a 的代码使用 this 来显式地引用对象的 radius 以及调用它的 getArea() 方法。this 引用通常是省略掉的，如 b 所示。然而，在引用隐藏数据域以及调用一个重载的构造方法的时候，this 引用是必须的。

```
public class Circle {
    private double radius;

    ...

    public double getArea() {
        return this.radius * this.radius * Math.PI;
    }

    public String toString() {
        return "radius: " + this.radius
            + "area: " + this.getArea();
    }
}
```

a)

等价于

```
public class Circle {
    private double radius;

    ...

    public double getArea() {
        return radius * radius * Math.PI;
    }

    public String toString() {
        return "radius: " + radius
            + "area: " + getArea();
    }
}
```

b)

9.14.1 使用 this 引用隐藏数据域

this 关键字可以用于引用类的隐藏数据域。例如，在数据域的 set 方法中，经常将数

据域名用作参数名。在这种情况下，这个数据域在 `set` 方法中被隐藏。为了给它设置新值，需要在方法中引用隐藏的数据域名。隐藏的静态变量可以简单地通过“类名.静态变量”的方式引用。隐藏的实例变量就需要使用关键字 `this` 来引用，如图 9-21a 所示。

```
public class F {
    private int i = 5;
    private static double k = 0;

    public void setI(int i) {
        this.i = i;
    }

    public static void setK(double k) {
        F.k = k;
    }

    // Other methods omitted
}
```

a)

Suppose that `f1` and `f2` are two objects of `F`.

Invoking `f1.setI(10)` is to execute
`this.i = 10`, where `this` refers `f1`

Invoking `f2.setI(45)` is to execute
`this.i = 45`, where `this` refers `f2`

Invoking `F.setK(33)` is to execute
`F.k = 33`. `setK` is a static method

b)

图 9-21 关键字 `this` 引用调用方法的对象

关键字 `this` 给出一种引用调用实例方法的对象的方法。调用 `f1.setI(10)` 时，执行了 `this.i=i`，将参数 `i` 的值赋给调用对象 `f1` 的数据域 `i`。关键字 `this` 是指调用实例方法 `setI` 的对象，如图 9-21b 所示。`F.k=k` 这一行的意思是将参数 `k` 的值赋给这个类的静态数据域 `k`，`k` 是被类的所有对象所共享的。

9.14.2 使用 `this` 调用构造方法

关键字 `this` 可以用于调用同一个类的另一个构造方法。例如，可以如下改写 `Circle` 类：

```
public class Circle {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    public Circle() {
        this(1.0);
    }

    ...
}
```

注：图中有箭头从 `this.radius = radius;` 指向 `this` 关键字，标注为“this 关键字用于引用所构建的对象的隐藏数据域 radius”；以及从 `this(1.0);` 指向 `this` 关键字，标注为“this 关键字用于调用另外一个构造方法”。

在第二个构造方法中，`this(1.0)` 这一行调用带 `double` 值参数的第一个构造方法。

- 注意：Java 要求在构造方法中，语句 `this(参数列表)` 应在任何其他可执行语句之前出现。
- 提示：如果一个类有多个构造方法，最好尽可能使用 `this(参数列表)` 实现它们。通常，无参数或参数少的构造方法可以用 `this(参数列表)` 调用参数多的构造方法。这样做通常可以简化代码，使类易于阅读和维护。

复习题

9.32 描述 `this` 关键字的角色。

9.33 下面代码中哪里有错误？

```
1 public class C {
2     private int p;
3 }
```

```

4   public C() {
5       System.out.println("C's no-arg constructor invoked");
6       this(0);
7   }
8
9   public C(int p) {
10      p = p;
11  }
12
13  public void setP(int p) {
14      p = p;
15  }
16  }

```

9.34 下面代码中哪里有错误?

```

public class Test {
    private int id;

    public void m1() {
        this.id = 45;
    }

    public void m2() {
        Test.id = 45;
    }
}

```

关键术语

action (动作)	no-arg constructor (无参构造方法)
anonymous object (匿名对象)	null value (空值)
attribute (属性)	object (对象)
behavior (行为)	object-oriented programming (OOP)(面向对象程序设计)
class (类)	package-private (or package-access) (包私有 (或包访问))
class's variable (类变量)	private constructor (私有的构造方法)
client (客户)	property (属性)
constructor (构造方法)	public class (公共类)
date field (数据域)	reference type (引用类型)
data field encapsulation (数据域封装)	reference variable (引用变量)
default constructor (默认构造方法)	setter (or mutator)(设置方法 (修改器))
dot operator (.) (点操作符)	state (状态)
getter (or accessor)(访问器 (读取器))	static method (静态方法)
instance (实例)	static variable (静态变量)
instance method (实例方法)	this keyword (this 关键字)
instance variable (实例变量)	Unified Modeling Language (UML) (统一建模语言)
instantiation (实例化)	
immutable class (不可变类)	
immutable object (不可变对象)	

本章小结

1. 类是对象的模板。它定义对象的属性, 并提供用于以创建对象的构造方法以及操作对象的普通方法。

2. 类也是一种数据类型。可以用它声明对象引用变量。对象引用变量中似乎存放了一个对象，但事实上，它包含的只是对该对象的引用。严格地讲，对象引用变量和对象是不同的，但是大多数情况下，它们的区别是可以忽略的。
3. 对象是类的实例。可以使用 `new` 操作符创建对象，使用点操作符 (`.`) 通过对象的引用变量来访问该对象的成员。
4. 实例变量或方法属于类的一个实例。它的使用与各自的实例相关联。静态变量是被同一个类的所有实例所共享的。可以在不使用实例的情况下调用静态方法。
5. 类的每个实例都能访问这个类的静态变量和静态方法。然而，为清晰起见，最好使用“类名.变量”和“类名.方法”来调用静态变量和静态方法。
6. 可见性修饰符指定类、方法和数据是如何被访问的。公共的 (`public`) 类、方法或数据可以被任何客户访问，私有的 (`private`) 方法或数据只能在本类中被访问。
7. 可以提供 `get` (访问器) 方法或者 `set` (修改器) 方法使客户程序能够看到或修改数据。
8. `get` 方法具有方法签名 `public returnType getPropertyname()`。如果返回值类型 (`returnType`) 是 `boolean` 型，则 `get` 方法应该定义为 `public boolean isPropertyName()`。`set` 方法具有方法签名 `public void setPropertyName(dataType propertyValue)`。
9. 所有传递给方法的参数都是值传递的。对于基本类型的参数，传递的是实际值；而若参数是引用数据类型，则传递的是对象的引用。
10. Java 数组是一个可以包含基本类型值或对象类型值的对象。当创建一个对象数组时，它的元素被赋予默认值 `null`。
11. 一旦被创建，不可变对象 (`immutable object`) 就不能被改变了。为了防止用户修改一个对象，可以定义该对象为不可变类。
12. 实例变量和静态变量的作用域是整个类，无论该变量在什么位置定义。实例变量和静态变量可以在类中的任何位置定义。为一致性考虑，本书都在类的开始部分定义。
13. `this` 关键字可以用于引用进行调用的对象。它也可以用于在构造方法中来调用同一个类的另外一个构造方法。

测试题

在线回答本章节的测试题，位于 www.cs.armstrong.edu/liang/intro10e/quiz.html。

编程练习题

 **教学提示：**第 9~13 章的练习题要达到下面三个目标：

- 设计类并画出 UML 类图。
- 实现 UML 中的类。
- 使用类开发应用程序。

学生可以从配套网站上下载偶数题号练习题的答案，教师可以从同一个网站下载所有答案。

9.2~9.5 节

9.1 (矩形类 `Rectangle`) 遵照 9.2 节中 `Circle` 类的例子，设计一个名为 `Rectangle` 的类表示矩形。这个类包括：

- 两个名为 `width` 和 `height` 的 `double` 型数据域，它们分别表示矩形的宽和高。`width` 和 `height` 的默认值都为 1。
- 创建默认矩形的无参构造方法。
- 一个创建 `width` 和 `height` 为指定值的矩形的构造方法。
- 一个名为 `getArea()` 的方法返回这个矩形的面积。

- 一个名为 `getPerimeter()` 的方法返回周长。

画出该类的 UML 图并实现这个类。编写一个测试程序，创建两个 `Rectangle` 对象——一个矩形的宽为 4 而高为 40，另一个矩形的宽为 3.5 而高为 35.9。按照这个顺序显示每个矩形的宽、高、面积和周长。

9.2 (股票类 `Stock`) 遵照 9.2 节中 `Circle` 类的例子，设计一个名为 `Stock` 的类。这个类包括：

- 一个名为 `symbol` 的字符串数据域表示股票代码。
- 一个名为 `name` 的字符串数据域表示股票名字。
- 一个名为 `previousClosingPrice` 的 `double` 型数据域，它存储的是前一日的股票值。
- 一个名为 `currentPrice` 的 `double` 型数据域，它存储的是当时的股票值。
- 创建一支有特定代码和名字的股票构造方法。
- 一个名为 `getChangePercent()` 的方法，返回从 `previousClosingPrice` 变化到 `currentPrice` 的百分比。

画出该类的 UML 图并实现这个类。编写一个测试程序，创建一个 `Stock` 对象，它的股票代码是 ORCL，股票名字为 Oracle Corporation，前一日收盘价是 34.5。设置新的当前值为 34.35，然后显示市值变化的百分比。

9.6 节

*9.3 (使用日期类 `Date`) 编写程序创建一个 `Date` 对象，设置它的流逝时间分别为 10000、100000、1000000、10000000、100000000、1000000000、10000000000、100000000000，然后使用 `toString()` 方法分别显示上述日期。

*9.4 (使用随机类 `Random`) 编写一个程序，创建种子是 1000 的 `Random` 对象，然后使用 `nextInt(100)` 方法显示 0 到 100 之间前 50 个随机整数。

*9.5 (使用公历类 `GregorianCalendar`) Java API 有一个在包 `java.util` 中的类 `GregorianCalendar`，可以使用它获得某个日期的年、月、日。它的无参构造方法构建一个当前日期的实例，`get(GregorianCalendar.YEAR)`、`get(GregorianCalendar.MONTH)` 和 `get(GregorianCalendar.DAY_OF_MONTH)` 方法返回年、月和日。编写一个程序完成两个任务：

- 显示当前的年、月和日。
- `GregorianCalendar` 类有方法 `setTimeInMillis(long)`，可以用它来设置从 1970 年 1 月 1 日算起的一个特定时间。将这个值设置为 1234567898765L，然后显示这个年、月和日。

9.7 ~ 9.9 节

*9.6 (秒表) 设计一个名为 `StopWatch` 的类，该类包含：

- 具有访问器方法的私有数据域 `startTime` 和 `endTime`。
- 一个无参构造方法，使用当前时间来初始化 `startTime`。
- 一个名为 `start()` 的方法，将 `startTime` 重设为当前时间。
- 一个名为 `stop()` 的方法，将 `endTime` 设置为当前时间。
- 一个名为 `getElapsedTime()` 的方法，以毫秒为单位返回秒表记录的流逝时间。

画出该类的 UML 图并实现这个类。编写一个测试程序，用于测量使用选择排序对 100 000 个数字进行排序的执行时间。

9.7 (账户类 `Account`) 设计一个名为 `Account` 的类，它包括：

- 一个名为 `id` 的 `int` 类型私有数据域 (默认值为 0)。
- 一个名为 `balance` 的 `double` 类型私有数据域 (默认值为 0)。
- 一个名为 `annualInterestRate` 的 `double` 类型私有数据域存储当前利率 (默认值为 0)。假设所有的账户都有相同的利率。
- 一个名为 `dateCreated` 的 `Date` 类型的私有数据域，存储账户的开户日期。
- 一个用于创建默认账户的无参构造方法。

- 一个用于创建带特定 id 和初始余额的账户的构造方法。
- id、balance 和 annualInterestRate 的访问器和修改器。
- dateCreated 的访问器。
- 一个名为 getMonthlyInterestRate() 的方法，返回月利率。
- 一个名为 withdraw 的方法，从账户提取特定数额。
- 一个名为 deposit 的方法向账户存储特定数额。

画出该类的 UML 图并实现这个类。

 **提示：**方法 getMonthlyInterest() 用于返回月利息，而不是利率。月利息是 $\text{balance} \times \text{monthlyInterestRate}$ 。monthlyInterestRate 是 $\text{annualInterestRate}/12$ 。注意，annualInterestRate 是一个百分数，比如 4.5%。你需要将其除以 100。

编写一个测试程序，创建一个账户 ID 为 1122、余额为 20 000 美元、年利率为 4.5% 的 Account 对象。使用 withdraw 方法取款 2500 美元，使用 deposit 方法存款 3000 美元，然后打印余额、月利息以及这个账户的开户日期。

9.8 (风扇类 Fan) 设计一个名为 Fan 的类来表示一个风扇。这个类包括：

- 三个名为 SLOW、MEDIUM 和 FAST 而值为 1、2 和 3 的常量，表示风扇的速度。
- 一个名为 speed 的 int 类型私有数据域，表示风扇的速度（默认值为 SLOW）。
- 一个名为 on 的 boolean 类型私有数据域，表示风扇是否打开（默认值为 false）。
- 一个名为 radius 的 double 类型私有数据域，表示风扇的半径（默认值为 5）。
- 一个名为 color 的 string 类型数据域，表示风扇的颜色（默认值为 blue）。
- 这四个数据域的访问器和修改器。
- 一个创建默认风扇的无参构造方法。
- 一个名为 toString() 的方法返回描述风扇的字符串。如果风扇是打开的，那么该方法在一个组合的字符串中返回风扇的速度、颜色和半径。如果风扇没有打开，该方法就会返回一个由“fan is off”和风扇颜色及半径组合成的字符串。

画出该类的 UML 图并实现这个类。编写一个测试程序，创建两个 Fan 对象。将第一个对象设置为最大速度、半径为 10、颜色为 yellow、状态为打开。将第二个对象设置为中等速度、半径为 5、颜色为 blue、状态为关闭。通过调用它们的 toString 方法显示这些对象。

**9.9 (几何：正 n 边形) 在一个正 n 边形中，所有边的长度都相同，且所有角的度数都相同（即这个多边形是等边等角的）。设计一个名为 RegularPolygon 的类，该类包括：

- 一个名为 n 的 int 型私有数据域定义多边形的边数，默认值为 3。
- 一个名为 side 的 double 型私有数据域存储边的长度，默认值为 1。
- 一个名为 x 的 double 型私有数据域定义多边形中点的 x 坐标，默认值为 0。
- 一个名为 y 的 double 型私有数据域定义多边形中点的 y 坐标，默认值为 0。
- 一个创建带默认值的正多边形的无参构造方法。
- 一个能创建带指定边数和边长度、中心在 (0,0) 的正多边形的构造方法。
- 一个能创建带指定边数和边长度、中心在 (x,y) 的正多边形的构造方法。
- 所有数据域的访问器和修改器。
- 一个返回多边形周长的方法 getPerimeter()。
- 一个返回多边形面积的方法 getArea()。计算正多边形面积的公式是：

$$\text{面积} = \frac{n \times s^2}{4 \times \tan\left(\frac{\pi}{n}\right)}$$

画出该类的 UML 图并实现这个类。编写一个测试程序，分别使用无参构造方法、RegularPolygon(6,4) 和 RegularPolygon(10,4,5.6,7.8) 创建三个 RegularPolygon 对象。

显示每个对象的周长和面积。

*9.10 (代数: 二次方程式) 为二次方程式 $ax^2+bx+c=0$ 设计一个名为 `QuadraticEquation` 的类。这个类包括:

- 代表三个系数的私有数据域 `a`、`b` 和 `c`。
- 一个参数为 `a`、`b` 和 `c` 的构造方法。
- `a`、`b`、`c` 的三个 `get` 方法。
- 一个名为 `getDiscriminant()` 的方法返回判别式, b^2-4ac 。
- 名为 `getRoot1()` 和 `getRoot2()` 的方法返回等式的两个根:

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{和} \quad r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

这些方法只有在判别式为非负数时才有用。如果判别式为负, 这些方法返回 0。

画出该类的 UML 图并实现这个类。编写一个测试程序, 提示用户输入 `a`、`b` 和 `c` 的值, 然后显示判别式的结果。如果判别式为正数, 显示两个根; 如果判别式为 0, 显示一个根; 否则, 显示 “The equation has no roots.” (这个方程无根)。参见编程练习题 3.1 的运行示例。

*9.11 (代数: 2×2 的线性方程) 为一个 2×2 的线性方程设计一个名为 `LinearEquation` 的类:

$$\begin{array}{l} ax + by = e \\ cx + dy = f \end{array} \quad x = \frac{ed - bf}{ad - bc} \quad y = \frac{af - ec}{ad - bc}$$

这个类包括:

- 私有数据域 `a`、`b`、`c`、`d`、`e` 和 `f`。
- 一个参数为 `a`、`b`、`c`、`d`、`e`、`f` 的构造方法。
- `a`、`b`、`c`、`d`、`e`、`f` 的六个 `get` 方法。
- 一个名为 `isSolvable()` 的方法, 如果 $ad-bc$ 不为 0 则返回 `true`。
- 方法 `getX()` 和 `getY()` 返回这个方程的解。

画出该类的 UML 图并实现这个类。编写一个测试程序, 提示用户输入 `a`、`b`、`c`、`d`、`e`、`f` 的值, 然后显示它的结果。如果 $ad-bc$ 为 0, 就报告 “The equation has no solution.”。参见编程练习题 3.3 的运行示例。

**9.12 (几何: 交点) 假设两条线段相交。第一条线段的两个端点是 (x_1, y_1) 和 (x_2, y_2) , 第二条线段的两个端点是 (x_3, y_3) 和 (x_4, y_4) 。编写一个程序, 提示用户输入这四个端点, 然后显示它们的交点。如编程练习题 3.25 所讨论的, 可以通过对一个线性方程求解来得到。使用编程练习题 9.11 中的 `LinearEquation` 类来求解该方程。参见编程练习题 3.25 的运行示例。

**9.13 (位置类 `Location`) 设计一个名为 `Location` 的类, 定位二维数组中的最大值及其位置。这个类包括公共的数据域 `row`、`column` 和 `maxValue`, 二维数组中的最大值及其下标用 `int` 型的 `row` 和 `column` 以及 `double` 型的 `maxValue` 存储。

编写下面的方法, 返回一个二维数组中最大值的位置。

```
public static Location locateLargest(double[][] a)
```

返回值是一个 `Location` 的实例。编写一个测试程序, 提示用户输入一个二维数组, 然后显示这个数组中最大元素的位置。下面是一个运行示例:

```
Enter the number of rows and columns in the array: 3 4 [Enter]
Enter the array:
23.5 35 2 10 [Enter]
4.5 3 45 3.5 [Enter]
35 44 5.5 9.6 [Enter]
The location of the largest element is 45 at (1, 2)
```

面向对象思考

教学目标

- 应用类的抽象来开发软件 (10.2 节)。
- 探索面向过程范式和面向对象范式的不同之处 (10.3 节)。
- 发现类之间的关系 (10.4 节)。
- 使用面向对象范式设计程序 (10.5 ~ 10.6 节)。
- 使用包装类 (Byte、Short、Integer、Long、Float、Double、Character 以及 Boolean) 为基本类型值创建对象 (10.7 节)。
- 使用基本类型与包装类类型之间的自动转化来简化程序设计 (10.8 节)。
- 使用 BigInteger 和 BigDecimal 类计算任意精度的大数字 (10.9 节)。
- 使用 String 类处理不可改变的字符串 (10.10 节)。
- 使用 StringBuilder 类和 StringBuffer 类来处理可以改变的字符串 (10.11 节)。

10.1 引言

要点提示：本章重点在类设计以及探索面向过程编程和面向对象编程的不同。

前面章节介绍了对象和类。我们也学习了如何定义类、创建对象以及使用 Java API 中的一些类的对象 (例如: Circle、Date、Random 以及 Point2D)。本书的方法是在教授面向对象程序设计之前,先讲述问题求解和基本程序设计的技术。本章将给出面向过程和面向对象程序设计的不同之处。你将会看到面向对象程序设计的优点,并学习如何高效地使用它。

这里,我们的焦点放在类的设计上。我们将使用几个例子来诠释面向对象方法的优点。这些例子包括如何在应用程序中设计新类、如何使用这些类,以及介绍 Java API 中的一些新的类。

10.2 类的抽象和封装

要点提示：类的抽象是指将类的实现和类的使用分离开,实现的细节被封装并且对用户隐藏,这被称为类的封装。

在第 6 章中已经学习了方法的抽象以及如何在逐步求精中使用它。Java 提供了多层次的抽象。类抽象 (class abstraction) 是将类的实现和使用分离。类的创建者描述类的功能,让使用者明白如何才能使用类。从类外可以访问的方法和数据域的集合以及预期这些成员如何行为的描述,合称为类的合约 (class's contract)。如图 10-1 所示,类的使用者不需要知道类是如何实现的。实现的细节经过封装,对用户隐藏起来,这称为类的封装 (class encapsulation)。例如:可以创建一个 Circle 对象,并且可以在不知道面积是如何计算出来的情况下,求出这个圆的面积。由于这个原因,类也称为抽象数据类型 (Abstract Data Type, ADT)。

类的抽象和封装是一个问题的两个方面。现实生活中的许多例子都可以说明类抽象的概念。例如:考虑建立一个计算机系统。个人计算机有很多组件——CPU、内存、磁盘、主板和风扇等。每个组件都可以看作是一个有属性和方法的对象。要使各个组件一起工作,只需

要知道每个组件是怎么用的以及如何与其他组件进行交互的，而无须了解这些组件内部是如何工作的。内部功能的实现被封装起来，对你是隐藏的。所以，你可以组装一台计算机，而不需要了解每个组件的功能是如何实现的。

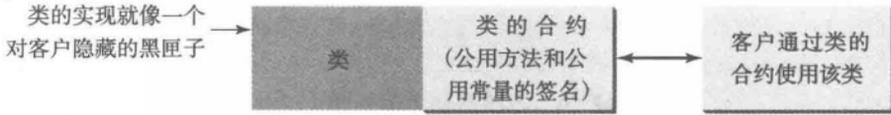


图 10-1 类抽象将类的实现与类的使用分离

对计算机系统的模拟准确地反映了面向对象方法。每个组件可以看成组件类的对象。例如，你可能已经建立了一个类，模拟用在计算机上的各种类型的风扇，它具有风扇尺寸和速度等属性，还有像开始和停止这样的方法。一个具体的风扇就是该类具有特定属性值的实例。

将得到一笔贷款作为另一个例子。一笔具体的贷款可以看作贷款类 `Loan` 的一个对象，利率、贷款额以及还贷周期都是它的数据属性，计算每月偿还额和总偿还额是它的方法。当你购买一辆汽车时，就用贷款利率、贷款额和还贷周期实例化这个类，创建一个贷款对象。然后，就可以使用这些方法计算贷款的月偿还额和总偿还额。作为一个贷款类 `Loan` 的用户，是不需要知道这些方法是如何实现的。

程序清单 2-9 给出计算贷款偿还额的程序。这个程序不能在其他程序中重用，因为计算支付的代码放在 `main` 方法中。解决这个问题的一种方式就是定义计算月偿还额和总偿还额的静态方法。但是，这个解决方案是有局限性的。假设希望将一个日期和这个贷款联系起来。没有一个好的办法可以不通过对象的使用来将一个日期和贷款联系起来。传统的面向过程式编程是动作驱动的，数据和动作是分离的。面向对象编程的范式重点在于对象，动作和数据一起定义在对象中。为了将日期和贷款联系起来，可以定义一个贷款类，将日期和贷款的其他属性一起作为数据域，并且贷款数据和动作在一个对象中集成。图 10-2 给出 `Loan` 类的 UML 类图。

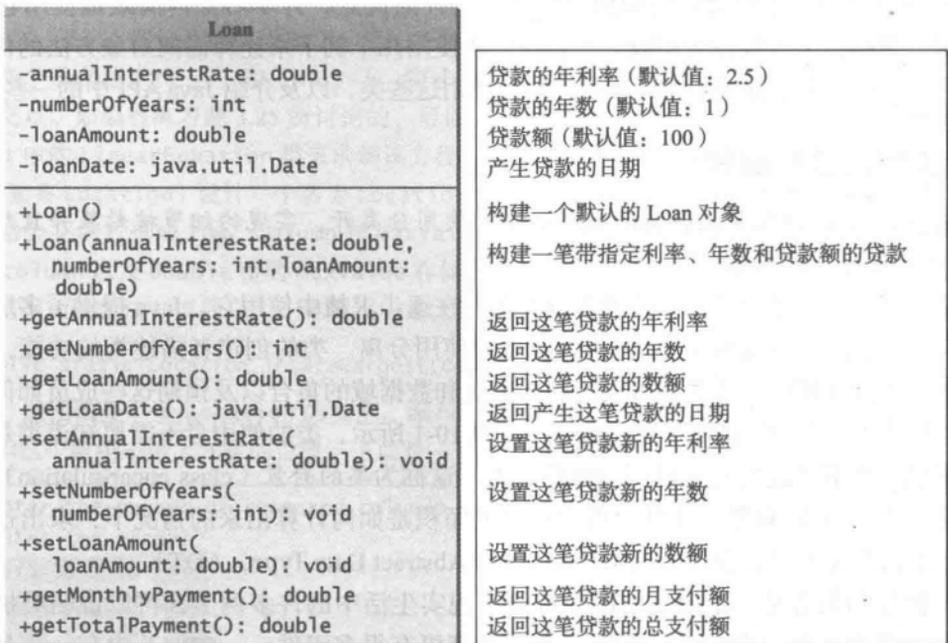


图 10-2 `Loan` 类对贷款的属性 and 行为建模

将图 10-2 的 UML 图看作 Loan 类的合约。贯穿本书，你将扮演两个角色，一个是类的用户，一个是类的开发者。记住用户可以在不知道类是如何实现的情况下使用类。

假设 Loan 类是可用的。程序清单 10-1 中的程序使用该类。

程序清单 10-1 TestLoanClass.java

```
1 import java.util.Scanner;
2
3 public class TestLoanClass {
4     /** Main method */
5     public static void main(String[] args) {
6         // Create a Scanner
7         Scanner input = new Scanner(System.in);
8
9         // Enter annual interest rate
10        System.out.print(
11            "Enter annual interest rate, for example, 8.25: ");
12        double annualInterestRate = input.nextDouble();
13
14        // Enter number of years
15        System.out.print("Enter number of years as an integer: ");
16        int numberOfYears = input.nextInt();
17
18        // Enter loan amount
19        System.out.print("Enter loan amount, for example, 120000.95: ");
20        double loanAmount = input.nextDouble();
21
22        // Create a Loan object
23        Loan loan =
24            new Loan(annualInterestRate, numberOfYears, loanAmount);
25
26        // Display loan date, monthly payment, and total payment
27        System.out.printf("The loan was created on %s\n" +
28            "The monthly payment is %.2f\nThe total payment is %.2f\n",
29            loan.getLoanDate().toString(), loan.getMonthlyPayment(),
30            loan.getTotalPayment());
31    }
32 }
```

```
Enter annual interest rate, for example, 8.25: 2.5 ↵ Enter
Enter number of years as an integer: 5 ↵ Enter
Enter loan amount, for example, 120000.95: 1000 ↵ Enter
The loan was created on Sat Jun 16 21:12:50 EDT 2012
The monthly payment is 17.75
The total payment is 1064.84
```

main 方法读取利率和还贷时间（以年为单位）以及贷款总额，创建一个 Loan 对象，然后使用 Loan 类中的实例方法获取月偿还款（第 29 行）和总偿还款（第 30 行）。

Loan 类可以如程序清单 10-2 实现。

程序清单 10-2 Loan.java

```
1 public class Loan {
2     private double annualInterestRate;
3     private int numberOfYears;
4     private double loanAmount;
5     private java.util.Date loanDate;
6
7     /** Default constructor */
```

```
8 public Loan() {
9     this(2.5, 1, 1000);
10 }
11
12 /** Construct a loan with specified annual interest rate,
13     number of years, and loan amount
14     */
15 public Loan(double annualInterestRate, int numberOfYears,
16     double loanAmount) {
17     this.annualInterestRate = annualInterestRate;
18     this.numberOfYears = numberOfYears;
19     this.loanAmount = loanAmount;
20     loanDate = new java.util.Date();
21 }
22
23 /** Return annualInterestRate */
24 public double getAnnualInterestRate() {
25     return annualInterestRate;
26 }
27
28 /** Set a new annualInterestRate */
29 public void setAnnualInterestRate(double annualInterestRate) {
30     this.annualInterestRate = annualInterestRate;
31 }
32
33 /** Return numberOfYears */
34 public int getNumberOfYears() {
35     return numberOfYears;
36 }
37
38 /** Set a new numberOfYears */
39 public void setNumberOfYears(int numberOfYears) {
40     this.numberOfYears = numberOfYears;
41 }
42
43 /** Return loanAmount */
44 public double getLoanAmount() {
45     return loanAmount;
46 }
47
48 /** Set a new loanAmount */
49 public void setLoanAmount(double loanAmount) {
50     this.loanAmount = loanAmount;
51 }
52
53 /** Find monthly payment */
54 public double getMonthlyPayment() {
55     double monthlyInterestRate = annualInterestRate / 1200;
56     double monthlyPayment = loanAmount * monthlyInterestRate / (1 -
57     (1 / Math.pow(1 + monthlyInterestRate, numberOfYears * 12)));
58     return monthlyPayment;
59 }
60
61 /** Find total payment */
62 public double getTotalPayment() {
63     double totalPayment = getMonthlyPayment() * numberOfYears * 12;
64     return totalPayment;
65 }
66
67 /** Return loan date */
68 public java.util.Date getLoanDate() {
69     return loanDate;
70 }
71 }
```

从类的开发者的角度来看,设计类是为了让很多不同的用户所使用。为了在更大的应用范围内使用类,类应该通过构造方法、属性和方法提供各种方式的定制。

Loan类包含两个构造方法、四个get方法、三个set方法,以及求月偿还额和总偿还额的方法。可以通过使用无参构造方法或者带三个参数(年利率、年数和贷款额)的构造方法来构造一个Loan对象。当创建一个贷款对象时,它的数据存储在loanDate域中,getLoanDate方法返回日期。方法getAnnualInterest、getNumberOfYears和getLoanAmount分别返回年利率、还款时间以及贷款总额。这个类的所有数据属性和方法都被绑定到Loan类的某个特定实例。因此,它们都是实例变量或者方法。

 **重要教学提示:** Loan类的UML图如图10-2所示,使用该图编写使用Loan类的测试程序,即使不知道这个Loan类是如何执行的。这有三个优点:

- 1) 揭示了开发类和使用类是两个不同的任务。
- 2) 能使你跳过某个类的复杂实现,而不打乱整本书的顺序。
- 3) 如果通过使用它熟悉了该类,那么你将更容易学会如何实现这个类。

从现在开始的所有例子,在将注意力放在它的实现上之前,你都可以先在这个类中创建一个对象,并且尝试使用它的方法。

复习题

10.1 如果重新定义程序清单10-2中的Loan类,去掉其中的设置方法,这个类是不可改变的吗?

10.3 面向对象的思考

 **要点提示:** 面向过程的范式重点在于设计方法。面向对象的范式将数据和方法耦合在一起构成对象。使用面向对象范式的软件设计重点在对象以及对对象的操作上。

第1~8章介绍使用循环、方法和数组来解决问题的基本程序设计技术。这些技术的学习为面向对象程序设计打下坚实的基础。类为构建可重用软件提供了更高的灵活性和更多的模块化。本节使用面向对象方法来改进第3章中介绍的一个问题的解决方案。在这个改进的过程中,可以洞察面向过程程序设计和面向对象程序设计的不同,也可以看出使用对象和类来开发可重用代码的优势。

程序清单3-4给出了计算身体质量指数的程序。因为它的代码在main方法中,所以不能在其他程序中重用。为使之具备可重用性,定义一个静态方法计算身体质量指数,如下所示:

```
public static double getBMI(double weight, double height)
```

这个方法对于计算给定体重和身高的身体质量指数是很有用的。但是,它是有局限性的。假设需要将体重和身高同一个人的名字与出生日期相关联,虽然可以分别声明几个变量来存储这些值,但是这些值不是紧密耦合在一起的。将它们耦合在一起的理想方法就是创建一个包含它们的对象。因为这些值都被绑定到单独的对象上,所以它们应该存储在实例数据域中。可以定义一个名为BMI的类,如图10-3所示。

假设BMI类是可用的。程序清单10-3给出使用这个类的测试程序。

程序清单 10-3 UseBMIClass.java

```
1 public class UseBMIClass {
2     public static void main(String[] args) {
3         BMI bmi1 = new BMI("Kim Yang", 18, 145, 70);
4         System.out.println("The BMI for " + bmi1.getName() + " is "
```

```

5     + bmi1.getBMI() + " " + bmi1.getStatus());
6
7     BMI bmi2 = new BMI("Susan King", 215, 70);
8     System.out.println("The BMI for " + bmi2.getName() + " is "
9     + bmi2.getBMI() + " " + bmi2.getStatus());
10  }
11  }

```

```

The BMI for Kim Yang is 20.81 Normal
The BMI for Susan King is 30.85 Obese

```

在类中提供这些数据域的 get 方法，为了简洁在 UML 图中省略这些方法

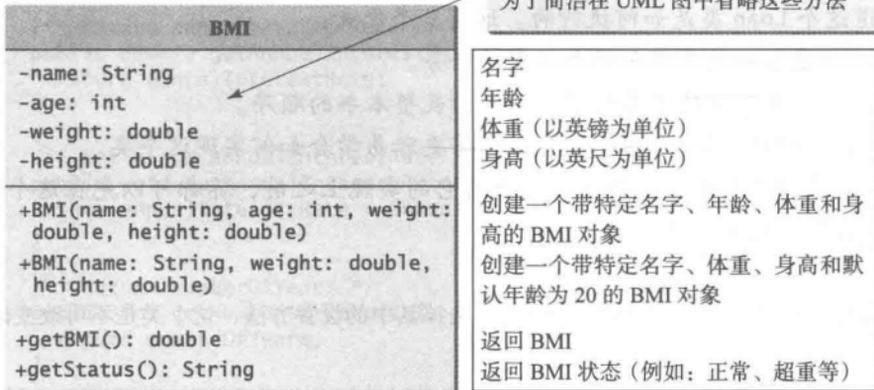


图 10-3 BMI 类封装 BMI 信息

第 3 行为 Kim Yang 创建一个对象 bmi1，而第 7 行为 Susan King 创建一个对象 bmi2。可以使用实例方法 getName()、getBMI() 和 getStatus() 返回一个 BMI 对象中的 BMI 信息。

BMI 类可以如程序清单 10-4 中的实现。

程序清单 10-4 BMI.java

```

1  public class BMI {
2      private String name;
3      private int age;
4      private double weight; // in pounds
5      private double height; // in inches
6      public static final double KILOGRAMS_PER_POUND = 0.45359237;
7      public static final double METERS_PER_INCH = 0.0254;
8
9      public BMI(String name, int age, double weight, double height) {
10         this.name = name;
11         this.age = age;
12         this.weight = weight;
13         this.height = height;
14     }
15
16     public BMI(String name, double weight, double height) {
17         this(name, 20, weight, height);
18     }
19
20     public double getBMI() {
21         double bmi = weight * KILOGRAMS_PER_POUND /
22             ((height * METERS_PER_INCH) * (height * METERS_PER_INCH));
23         return Math.round(bmi * 100) / 100.0;
24     }
25 }

```

```
26 public String getStatus() {
27     double bmi = getBMI();
28     if (bmi < 18.5)
29         return "Underweight";
30     else if (bmi < 25)
31         return "Normal";
32     else if (bmi < 30)
33         return "Overweight";
34     else
35         return "Obese";
36 }
37
38 public String getName() {
39     return name;
40 }
41
42 public int getAge() {
43     return age;
44 }
45
46 public double getWeight() {
47     return weight;
48 }
49
50 public double getHeight() {
51     return height;
52 }
53 }
```

使用体重和身高来计算 BMI 的数学公式已经在 3.8 节中给出。实例方法 `getBMI()` 返回 BMI。因为体重和身高是对象的实例数据域，`getBMI()` 方法可以使用这些属性来计算对象的 BMI 值。

实例方法 `getStatus()` 返回解释 BMI 的字符串。这个解释也已经在 3.8 节中给出。

这个例子演示了面向对象范式比面向过程范式有优势的地方。面向过程范式重在设计方法。面向对象范式将数据和方法都组合在对象中。使用面向对象范式的软件设计重在对象和对象上的操作。面向对象方法结合了面向过程范式的功能以及将数据和操作集成在对象中的特性。

在面向过程程序设计中，数据和数据上的操作是分离的，而且这种做法要求传递数据给方法。面向对象程序设计将数据和对它们的操作都放在一个对象中。这个方法解决了很多面向过程程序设计固有的问题。面向对象程序设计方法以一种反映真实世界的方式组织程序，在真实世界中，所有的对象和属性及动作都相关联。使用对象提高了软件的可重用性，并且使程序更易于开发和维护。Java 程序设计涉及对对象的思考，一个 Java 程序可以看作是一个相互操作的对象集合。

🔪 复习题

10.2 程序清单 10-4 中的 BMI 类是不可改变的吗？

10.4 类的关系

🔪 要点提示：为了设计类，需要探究类之间的关系。类中间的关系通常是关联、聚合、组合以及继承。

本节探讨关联、聚合以及组合关系。继承关系将在下一章中介绍。

10.4.1 关联

关联是一种常见的二元关系，描述两个类之间的活动。例如，学生选取课程是 Student 类和 Course 类之间的一种关联，而教师教授课程是 Faculty 类和 Course 类之间的关联。这些关联可以使用 UML 图形标识来表达，如图 10-4 所示。

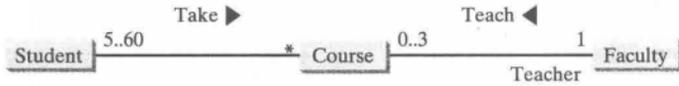


图 10-4 该 UML 图显示学生可以选择任意数量的课程，教师最多可以教授 3 门课程，每门课程可以有 5 到 60 个学生，并且每门课程只由一位教师来教授

关联由两个类之间的实线表示，可以有一个可选的标签描述关系。图 10-4 中，标签是 Take 和 Teach。每个关系可以有一个可选的小的黑色三角形表明关系的方向。在该图中，方向表明学生选取课程（而不是相反方向的课程选取学生）。

关系中涉及的每个类可以有一个角色名称，描述在该关系中担当的角色。图 10-4 中，Teacher 是 Faculty 的角色名。

关联中涉及的每个类可以给定一个多重性（multiplicity），放置在类的边上用于给定 UML 图中关系所涉及的类的对象数。多重性可以是一个数字或者一个区间，决定在关系中涉及类的多少个对象。字符 * 意味着无数多个对象，而 $m..n$ 表示对象数处于 m 和 n 之间，并且包括 m 和 n 。图 10-4 中，每个学生可以选取任意数量的课程数，每门课程可以有至少 5 个最多 60 个学生。每门课程只由一位教师教授，并且每位教师每学期可以教授 0 到 3 门课程。

在 Java 代码中，可以通过使用数据域以及方法来实现关联。例如，图 10-4 中的关系可以使用图 10-5 中的类来实现。关系“一个学生选取一门课程”使用 Student 类中的 addCourse 方法和 Course 类中的 addStudent 方法实现。关系“一位教师教授一门课程”使用 Faculty 类中的 addCourse 方法和 Course 类中的 setFaculty 方法实现。Student 类可以使用一个列表来存储学生选取的课程，Faculty 类可以使用一个列表来存储教师教授的课程，Course 类可以使用一个列表来存储课程中登记的学生以及一个数据域来存储教授该课程的教师。

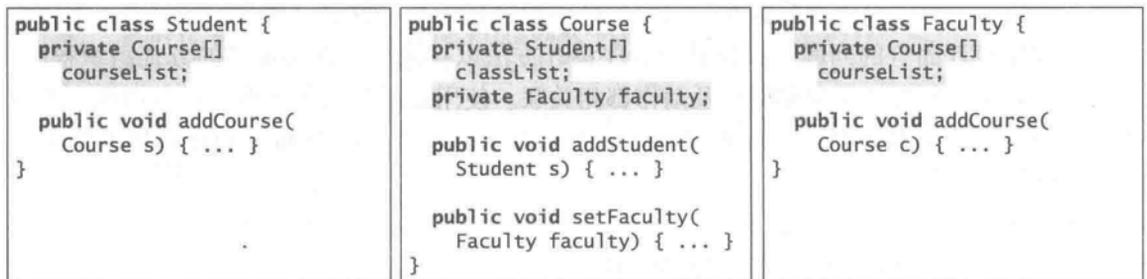


图 10-5 关联关系使用类中的数据域和方法来实现

注意：实现类之间的关系可以有很多种可能的方法。例如，Course 类中的学生和教师信息可以省略，因为它们已经在 Student 和 Faculty 类中了。同样的，如果不需要知道一个学生选取的课程或者教师教授的课程，Student 或者 Faculty 类中的数据域 courseList 和 addCourse 方法也可以省略。

10.4.2 聚集和组合

聚集是关联的一种特殊形式，代表了两个对象之间的归属关系。聚集建模 has-a 关系。所有者对象称为聚集对象，它的类称为聚集类。而从属对象称为被聚集对象，它的类称为被聚集类。

一个对象可以被多个其他的聚集对象所拥有。如果一个对象只归属于一个聚集对象，那么它和聚集对象之间的关系就称为组合 (composition)。例如：“一个学生有一个名字”就是学生类 Student 与名字类 Name 之间的一个组合关系，而“一个学生有一个地址”是学生类 Student 与地址类 Address 之间的一个聚集关系，因为一个地址可以被几个学生所共享。在 UML 中，附加在聚集类 (例如：Student) 上的实心菱形表示它和被聚集类 (例如：Name) 之间具有组合关系；而附加在聚集类 (例如：Student) 上的空心菱形表示它与被聚集类 (例如：Address) 之间具有聚集关系，如图 10-6 所示。

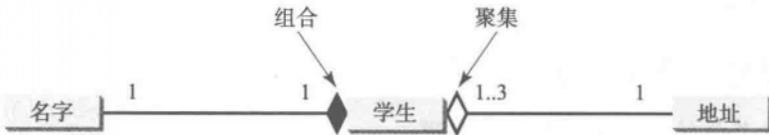


图 10-6 每个学生有一个名字和一个地址

在图 10-6 中，每个学生只能有一个地址，而每个地址最多可以被 3 个学生共享。每个学生都有一个名字，而每个学生的名字都是唯一的。

聚集关系通常被表示为聚集类中的一个数据域。例如：图 10-6 中的关系可以使用图 10-7 中的类来实现。关系“一个学生拥有一个名字”以及“一个学生有一个地址”在 Student 类中的数据域 name 和 address 中实现。

```

public class Name {
  ...
}
public class Student {
  private Name name;
  private Address address;
  ...
}
public class Address {
  ...
}
  
```

被聚集类 聚集类 被聚集类

图 10-7 组合关系使用类中的数据域来实现

聚集可以存在于同一类的多个对象之间。例如：一个人可能有一个管理者，如图 10-8 所示。

在关系“一个人有一个管理者”中，管理者可以如下表示为 Person 类的一个数据域：

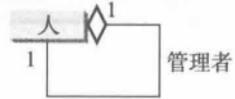


图 10-8 一个人可以有一个管理者

```

public class Person {
  // The type for the data is the class itself
  private Person supervisor;
  ...
}
  
```

如果一个人可以有几个管理者，如图 10-9a 所示，可以用一个数组存储管理者，如图 10-9b 所示。

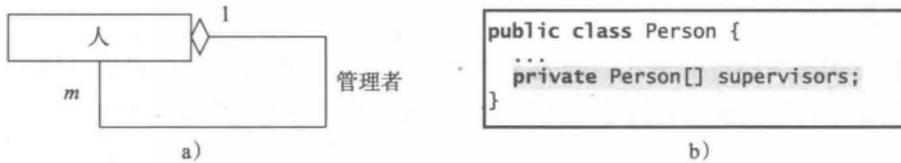


图 10-9 一个人可以有几个管理者

注意：由于聚集和组合关系都以同样的方式用类来表示，我们不区分它们，将两者都称为组合。

复习题

- 10.3 类之间的常用关系是什么？
- 10.4 什么是关联？什么是聚集？什么是组合？
- 10.5 聚集和组合的 UML 图标识是什么？
- 10.6 为什么聚集和组合都一起被称为组合？

10.5 示例学习：设计 Course 类

要点提示：本节设计一个类来对课程建模。

本书的宗旨是“通过例子来教学，通过动手来学习（teaching by example and learning by doing）”。本书提供了各种例子来演示面向对象程序设计。本节以及下一节将给出设计类的补充示例。

假设需要处理课程信息。每门课程都有一个名字以及选课的学生，要能够向 / 从这个课程添加 / 删除一个学生。可以使用一个类来对课程建模，如图 10-10 所示。

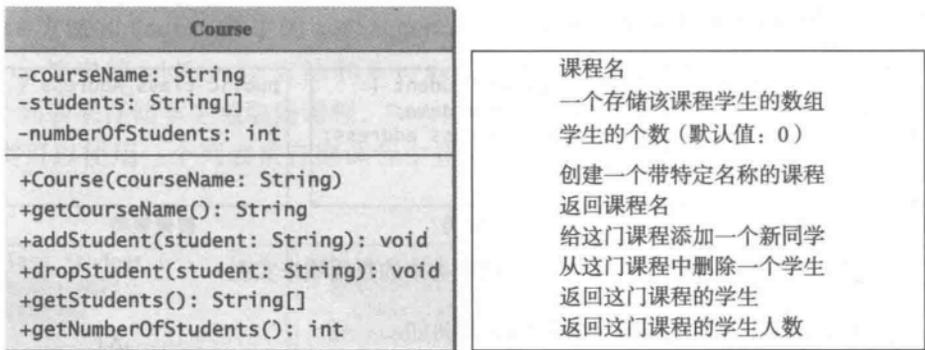


图 10-10 Course 类对课程建模

可以向构造方法 `Course(String name)` 传递一门课程的名称来创建一个 `Course` 对象。可以使用 `addStudent(String student)` 方法向某门课程添加学生，使用 `dropStudent(String student)` 方法从某门课程中删除一个学生，而使用 `getStudents()` 方法可以返回选这门课程的所有学生。假设 `Course` 类是可用的。程序清单 10-5 给出了一个测试类，这个测试类创建了两门课程，并向课程中添加学生。

程序清单 10-5 TestCourse.java

```

1 public class TestCourse {
2     public static void main(String[] args) {
3         Course course1 = new Course("Data Structures");

```

```

4     Course course2 = new Course("Database Systems");
5
6     course1.addStudent("Peter Jones");
7     course1.addStudent("Kim Smith");
8     course1.addStudent("Anne Kennedy");
9
10    course2.addStudent("Peter Jones");
11    course2.addStudent("Steve Smith");
12
13    System.out.println("Number of students in course1: "
14        + course1.getNumberOfStudents());
15    String[] students = course1.getStudents();
16    for (int i = 0; i < course1.getNumberOfStudents(); i++)
17        System.out.print(students[i] + ", ");
18
19    System.out.println();
20    System.out.print("Number of students in course2: "
21        + course2.getNumberOfStudents());
22 }
23 }

```

```

Number of students in course1: 3
Peter Jones, Kim Smith, Anne Kennedy,
Number of students in course2: 2

```

Course 类在程序清单 10-6 中实现。它使用一个数组存储选该门课的学生。为简单起见，假设选课的人数最多为 100。在第 3 行使用 `new String[100]` 创建数组。`addStudent` 方法（第 10 行）向这个数组中添加学生。只要有新的学生加入课程，`numberOfStudents` 就增加 1（第 12 行）。`getStudents` 方法返回这个数组。`dropStudent` 方法（第 27 行）留作练习。

程序清单 10-6 Course.java

```

1  public class Course {
2      private String courseName;
3      private String[] students = new String[100];
4      private int numberOfStudents;
5
6      public Course(String courseName) {
7          this.courseName = courseName;
8      }
9
10     public void addStudent(String student) {
11         students[numberOfStudents] = student;
12         numberOfStudents++;
13     }
14
15     public String[] getStudents() {
16         return students;
17     }
18
19     public int getNumberOfStudents() {
20         return numberOfStudents;
21     }
22
23     public String getCourseName() {
24         return courseName;
25     }
26
27     public void dropStudent(String student) {
28         // Left as an exercise in Programming Exercise 10.9
29     }
30 }

```

数组的大小固定为 100 (第 3 行), 所以在一门课程中不能有多于 100 个学生。可以在编程练习题 10.9 中改进它, 使数组尺寸可以自动增加。

创建一个 `Course` 对象时就创建了一个数组对象。`Course` 对象包含对数组的引用。简洁起见, 可以说这个 `Course` 对象包含了这个数组。

用户可以创建一个 `Course` 对象, 然后通过公共方法 `addStudent`、`dropStudent`、`getNumberOfStudents` 和 `getStudents` 来操作它。然而, 用户不需要知道这些方法是如何实现的。`Course` 类封装了内部的实现。该例是使用一个数组存储学生的, 但也可以使用不同的数据结构存储 `students`。只要公共方法的合约保持不变, 那么使用 `Course` 类的程序也无须修改。

10.6 示例学习: 设计栈类

 **要点提示:** 本节设计一个类来对栈建模。

回顾一下, 栈 (stack) 是一种以“后进先出”的方式存放数据的数据结构, 如图 10-11 所示。

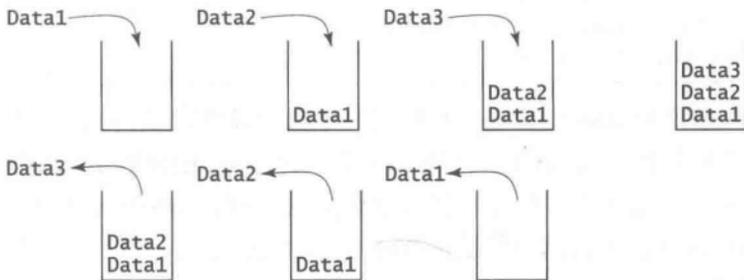


图 10-11 栈是用后进先出的方式存放数据的

栈有很多应用。例如: 编译器使用栈来处理方法的调用。当调用某个方法时, 方法的参数和局部变量都被压入栈中。当一个方法调用另一个方法时, 新方法的参数和局部变量被压入栈中。当方法完成它的工作, 返回它的调用者时, 从栈中释放与它相关的空间。

可以定义一个类建模栈。为简单起见, 假设该栈存储 `int` 数值。因此, 命名这个栈类为 `StackOfIntegers`。这个类的 UML 图如图 10-12 所示。

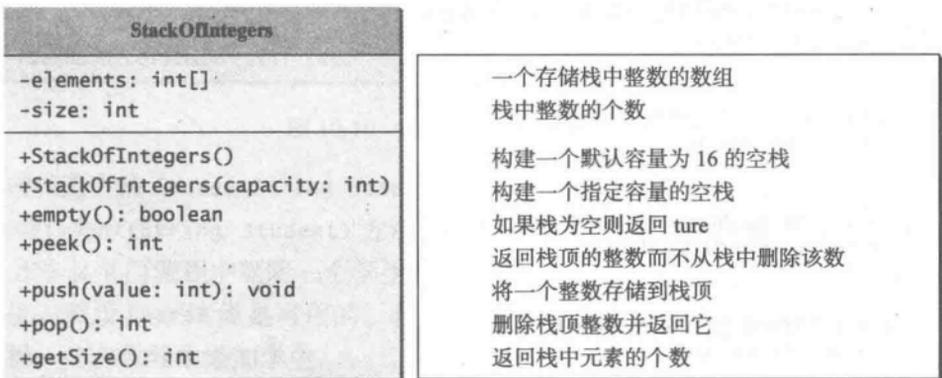


图 10-12 `StackOfIntegers` 类封装栈的存储并提供处理栈的操作

假设该类是可用的。在程序清单 10-7 中编写一个测试程序, 它使用该创建类创建一个栈

(第3行), 其中存储了 0, 1, 2, ..., 9 这 10 个整数 (第6行), 然后按逆序显示它们 (第9行)。

程序清单 10-7 TestStackOfIntegers.java

```

1 public class TestStackOfIntegers {
2     public static void main(String[] args) {
3         StackOfIntegers stack = new StackOfIntegers();
4
5         for (int i = 0; i < 10; i++)
6             stack.push(i);
7
8         while (!stack.empty())
9             System.out.print(stack.pop() + " ");
10    }
11 }

```

9 8 7 6 5 4 3 2 1 0

如何实现 StackOfIntegers 类呢? 栈中的元素都存储在一个名为 elements 的数组中。创建一个栈的时候, 同时也创建了这个数组。类的无参构造方法创建一个默认容量为 16 的数组。变量 size 记录了栈中元素的个数, 而 size-1 是栈顶元素的下标, 如图 10-13 所示。对空栈来说, size 为 0。

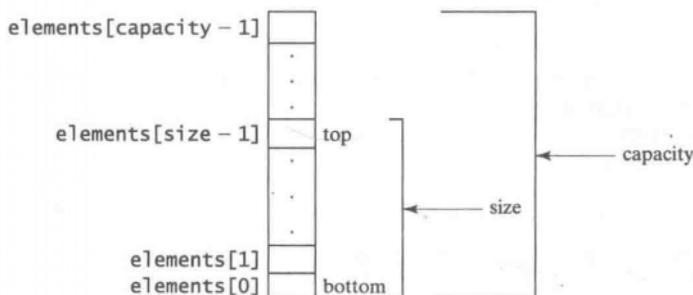


图 10-13 StackOfIntegers 类封装栈的存储并提供处理栈的操作

StackOfIntegers 类在程序清单 10-8 中实现。方法 empty()、peek()、pop() 和 getSize() 都容易实现。为了实现方法 push(int value), 如果 size < capacity, 则将 value 赋值给 elements[size] (第 24 行)。如果栈已满 (即 size ≥ capacity), 则创建一个容量为当前容量两倍的新数组 (第 19 行), 将当前数组的内容复制到新数组中 (第 20 行), 并将新数组的引用赋值给栈中当前数组 (第 21 行)。现在, 可以给这个数组添加新值了 (第 24 行)。

程序清单 10-8 StackOfIntegers.java

```

1 public class StackOfIntegers {
2     private int[] elements;
3     private int size;
4     public static final int DEFAULT_CAPACITY = 16;
5
6     /** Construct a stack with the default capacity 16 */
7     public StackOfIntegers() {
8         this(DEFAULT_CAPACITY);
9     }
10
11    /** Construct a stack with the specified maximum capacity */
12    public StackOfIntegers(int capacity) {
13        elements = new int[capacity];
14    }

```

```
15
16  /** Push a new integer to the top of the stack */
17  public void push(int value) {
18      if (size >= elements.length) {
19          int[] temp = new int[elements.length * 2];
20          System.arraycopy(elements, 0, temp, 0, elements.length);
21          elements = temp;
22      }
23
24      elements[size++] = value;
25  }
26
27  /** Return and remove the top element from the stack */
28  public int pop() {
29      return elements[--size];
30  }
31
32  /** Return the top element from the stack */
33  public int peek() {
34      return elements[size - 1];
35  }
36
37  /** Test whether the stack is empty */
38  public boolean empty() {
39      return size == 0;
40  }
41
42  /** Return the number of elements in the stack */
43  public int getSize() {
44      return size;
45  }
46 }
```

10.7 将基本数据类型值作为对象处理

 **要点提示：**基本数据类型值不是一个对象，但是可以使用 Java API 中的包装类来包装成一个对象。

出于对性能的考虑，在 Java 中基本数据类型不作为对象使用。因为处理对象需要额外的系统开销，所以，如果将基本数据类型当作对象，就会给语言性能带来负面影响。然而，Java 中的许多方法需要将对象作为参数。Java 提供了一个方便的办法，即将基本数据类型并入对象或包装成对象（例如，将 int 包装成 Integer 类，将 double 包装成 Double 类，将 char 包装成 Character 类）。通过使用包装类，可以将基本数据类型值作为对象处理。Java 为基本数据类型提供了 Boolean、Character、Double、Float、Byte、Short、Integer 和 Long 等包装类。这些包装类都打包在 java.lang 包里。Boolean 类包装了布尔值 true 或者 false。本节使用 Integer 和 Double 类为例介绍数值包装类。

 **注意：**大多数基本类型的包装类的名称与对应的基本数据类型名称一样，第一个字母要大写。Integer 和 Character 例外。

数值包装类相互之间都非常相似。每个都包含了 doubleValue()、floatValue()、intValue()、longValue()、shortValue() 和 byteValue() 方法。这些方法将对象“转换”为基本类型值。Integer 类和 Double 类的主要特征如图 10-14 所示。

既可以用基本数据类型值也可以用表示数值的字符串来构造包装类。例如，new Double(5.0)、new Double("5.0")、new Integer(5) 和 new Integer("5")。

java.lang.Integer	java.lang.Double
-value: int	-value: double
+MAX_VALUE: int	+MAX_VALUE: double
+MIN_VALUE: int	+MIN_VALUE: double
+Integer(value: int)	+Double(value: double)
+Integer(s: String)	+Double(s: String)
+byteValue(): byte	+byteValue(): byte
+shortValue(): short	+shortValue(): short
+intValue(): int	+intValue(): int
+longValue(): long	+longValue(): long
+floatValue(): float	+floatValue(): float
+doubleValue(): double	+doubleValue(): double
+compareTo(o: Integer): int	+compareTo(o: Double): int
+toString(): String	+toString(): String
+valueOf(s: String): Integer	+valueOf(s: String): Double
+valueOf(s: String, radix: int): Integer	+valueOf(s: String, radix: int): Double
+parseInt(s: String): int	+parseDouble(s: String): double
+parseInt(s: String, radix: int): int	+parseDouble(s: String, radix: int): double

图 10-14 包装类提供构造方法、常量和处理各种数据类型的转换方法

包装类没有无参构造方法。所有包装类的实例都是不可变的，这意味着一旦创建对象后，它们的内部值就不能再改变。

每一个数值包装类都有常量 MAX_VALUE 和 MIN_VALUE。MAX_VALUE 表示对应的基本数据类型的最大值。对于 Byte、Short、Integer 和 Long 而言，MIN_VALUE 表示对应的基本类型 byte、short、int 和 long 的最小值。对 Float 和 Double 类而言，MIN_VALUE 表示 float 型和 double 型的最小正值。下面的语句显示最大整数 (2 147 483 647)、最小正浮点数 (1.4E-45)，以及双精度浮点数的最大值 (1.79769313486231570e+308d)：

```
System.out.println("The maximum integer is " + Integer.MAX_VALUE);
System.out.println("The minimum positive float is " +
    Float.MIN_VALUE);
System.out.println(
    "The maximum double-precision floating-point number is " +
    Double.MAX_VALUE);
```

每个数值包装类都会包含方法 doubleValue()、floatValue()、intValue()、longValue() 和 shortValue()。这些方法返回包装对象的 double、float、int、long 或 short 值。例如，

```
new Double(12.4).intValue() returns 12;
new Integer(12).doubleValue() returns 12.0;
```

回顾下 String 类中包含 compareTo 方法用于比较两个字符串。数值包装类中包含 compareTo 方法用于比较两个数值，并且如果该数值大于、等于或者小于另外一个数值时，分别返回 1、0、-1。例如，

```
new Double(12.4).compareTo(new Double(12.3)) returns 1;
new Double(12.3).compareTo(new Double(12.3)) returns 0;
new Double(12.3).compareTo(new Double(12.51)) returns -1;
```

数值包装类有一个有用的静态方法 valueOf(String s)。该方法创建一个新对象，并将它初始化为指定字符串表示的值。例如，

```
Double doubleObject = Double.valueOf("12.4");
Integer integerObject = Integer.valueOf("12");
```

我们已经使用过 `Integer` 类中的 `parseInt` 方法将一个数值字符串转换为一个 `int` 值，而且使用过 `Double` 类中的 `parseDouble` 方法将一个数值字符串转变为一个 `double` 值。每个数值包装类都有两个重载的方法，将数值字符串转换为正确的以 10（十进制）或指定值为基数（例如，2 为二进制，8 为八进制，16 为十六进制）的数值。

```
// These two methods are in the Byte class
public static byte parseByte(String s)
public static byte parseByte(String s, int radix)

// These two methods are in the Short class
public static short parseShort(String s)
public static short parseShort(String s, int radix)

// These two methods are in the Integer class
public static int parseInt(String s)
public static int parseInt(String s, int radix)

// These two methods are in the Long class
public static long parseLong(String s)
public static long parseLong(String s, int radix)

// These two methods are in the Float class
public static float parseFloat(String s)
public static float parseFloat(String s, int radix)

// These two methods are in the Double class
public static double parseDouble(String s)
public static double parseDouble(String s, int radix)
```

例如，

```
Integer.parseInt("11", 2) returns 3;
Integer.parseInt("12", 8) returns 10;
Integer.parseInt("13", 10) returns 13;
Integer.parseInt("1A", 16) returns 26;
```

`Integer.parseInt("12", 2)` 会引起一个运行时错误，因为 12 不是二进制数。

注意，可以使用 `format` 方法将一个十进制数转换为十六进制数，例如，

```
String.format("%x", 26) returns 1A;
```

复习题

10.7 描述基本类型的包装类。

10.8 下面的每个语句可以编译成功么？

- `Integer i = new Integer("23");`
- `Integer i = new Integer(23);`
- `Integer i = Integer.valueOf("23");`
- `Integer i = Integer.parseInt("23", 8);`
- `Double d = new Double();`
- `Double d = Double.valueOf("23.45");`
- `int i = (Integer.valueOf("23")).intValue();`
- `double d = (Double.valueOf("23.4")).doubleValue();`
- `int i = (Double.valueOf("23.4")).intValue();`

j. `String s = (Double.valueOf("23.4")).toString();`

10.9 如何将一个整数转换为一个字符串？如何将一个数值字符串转换为一个整数？如何将一个 `double` 值转换为字符串？如何将一个数值型字符串转换为 `double` 值？

10.10 给出下面代码的输出。

```
public class Test {
    public static void main(String[] args) {
        Integer x = new Integer(3);
        System.out.println(x.intValue());
        System.out.println(x.compareTo(new Integer(4)));
    }
}
```

10.11 下面代码的输出是什么？

```
public class Test {
    public static void main(String[] args) {
        System.out.println(Integer.parseInt("10"));
        System.out.println(Integer.parseInt("10", 10));
        System.out.println(Integer.parseInt("10", 16));
        System.out.println(Integer.parseInt("11"));
        System.out.println(Integer.parseInt("11", 10));
        System.out.println(Integer.parseInt("11", 16));
    }
}
```

10.8 基本类型和包装类类型之间的自动转换

 **要点提示：**根据上下文环境，基本数据类型值可以使用包装类自动转换成一个对象，反过来的自动转换也可以。

将基本类型值转换为包装类对象的过程称为装箱 (boxing)，相反的过程称为开箱 (unboxing)。Java 允许基本类型和包装类类型之间进行自动转换。如果一个基本类型值出现在需要对象的环境中，编译器会将基本类型值进行自动装箱；如果一个对象出现在需要基本类型值的环境中，编译器会将对象进行自动开箱。这称为自动装箱和自动开箱。

例如，可以用自动装箱将图 a 中的语句简化为图 b 中的语句：

<code>Integer intObject = new Integer (2);</code>	等价于	<code>Integer intObject = 2;</code>
a)		b)
	 自动装箱	

考虑下面的例子：

```
1 Integer[] intArray = {1, 2, 3};
2 System.out.println(intArray[0] + intArray[1] + intArray[2]);
```

在第一行中，基本类型值 1、2 和 3 被自动装箱成对象 `new Integer(1)`、`new Integer(2)` 和 `new Integer(3)`。第二行中，对象 `intArray[0]`、`intArray[1]` 和 `intArray[2]` 被自动转换为 `int` 值，然后进行相加。

复习题

10.12 什么是自动装箱和自动开箱？下面的语句正确吗？

- a. `Integer x = 3 + new Integer(5);`
- b. `Integer x = 3;`
- c. `Double x = 3;`

- d. `Double x = 3.0;`
- e. `int x = new Integer(3);`
- f. `int x = new Integer(3) + new Integer(4);`

10.13 给出下面代码的输出结果。

```
public class Test {
    public static void main(String[] args) {
        Double x = 3.5;
        System.out.println(x.intValue());
        System.out.println(x.compareTo(4.5));
    }
}
```

10.9 BigInteger 和 BigDecimal 类

 **要点提示：** `BigInteger` 类和 `BigDecimal` 类可以用于表示任意大小和精度的整数或者十进制数。

如果要进行非常大的数的计算或者高精度浮点值的计算，可以使用 `java.math` 包中的 `BigInteger` 类和 `BigDecimal` 类。它们都是不可变的。`long` 类型的最大整数值为 `long.MAX_VALUE`（即 9223372036854775807）。`BigInteger` 的实例可以表示任意大小的整数。可以使用 `new BigInteger(String)` 和 `new BigDecimal(String)` 来创建 `BigInteger` 和 `BigDecimal` 的实例，使用 `add`、`subtract`、`multiply`、`divide` 和 `remainder` 方法完成算术运算，使用 `compareTo` 方法比较两个大数字。例如，下面的代码创建两个 `BigInteger` 对象并且将它们进行相乘：

```
BigInteger a = new BigInteger("9223372036854775807");
BigInteger b = new BigInteger("2");
BigInteger c = a.multiply(b); // 9223372036854775807 * 2
System.out.println(c);
```

它的输出为 18446744073709551614。

对 `BigDecimal` 对象的精度没有限制。如果结果不能终止，那么 `divide` 方法会抛出 `ArithmeticException` 异常。但是，可以使用重载的 `divide(BigDecimal d, int scale, int roundingMode)` 方法来指定尺度和舍入方式来避免这个异常，这里的 `scale` 是指小数点后最小的整数位数。例如，下面的代码创建两个尺度为 20、舍入方式为 `BigDecimal.ROUND_UP` 的 `BigDecimal` 对象。

```
BigDecimal a = new BigDecimal(1.0);
BigDecimal b = new BigDecimal(3);
BigDecimal c = a.divide(b, 20, BigDecimal.ROUND_UP);
System.out.println(c);
```

输出为 0.33333333333333333334。

注意，一个整数的阶乘可能会非常大。程序清单 10-9 给出可以返回任意整数阶乘的方法。

程序清单 10-9 LargeFactorial.java

```
1 import java.math.*;
2
3 public class LargeFactorial {
4     public static void main(String[] args) {
5         System.out.println("50! is \n" + factorial(50));
6     }
}
```

```
7
8 public static BigInteger factorial(long n) {
9     BigInteger result = BigInteger.ONE;
10    for (int i = 1; i <= n; i++)
11        result = result.multiply(new BigInteger(i + ""));
12
13    return result;
14 }
15 }
```

```
50! is
30414093201713378043612608166064768844377641568960512000000000000
```

`BigInteger.ONE` (第9行) 是一个定义在 `BigInteger` 类中的常量。`BigInteger.ONE` 和 `new BigInteger("1")` 是一样的。

通过调用 `multiply` 方法 (第11行), 得到了一个新的结果。

复习题

10.14 下面代码的输出是什么?

```
public class Test {
    public static void main(String[] args) {
        java.math.BigInteger x = new java.math.BigInteger("3");
        java.math.BigInteger y = new java.math.BigInteger("7");
        java.math.BigInteger z = x.add(y);
        System.out.println("x is " + x);
        System.out.println("y is " + y);
        System.out.println("z is " + z);
    }
}
```

10.10 String 类

要点提示: `String` 对象是不可改变的。字符串一旦创建, 内容不能再改变。

在 4.4 节中介绍了字符串。你已经知道字符串是对象, 可以通过 `charAt(index)` 方法从字符串中得到某个指定位置的字符。`length()` 方法返回字符串的大小, `substring` 方法返回字符串中的子串, `indexOf` 和 `lastIndexOf` 方法返回第一个或者最后一个匹配的字符或者子字符串。本节中我们将更加细致地对字符串进行讨论。

`String` 类中有 13 个构造方法以及 40 多个处理字符串的方法。这不仅在程序设计中非常有用, 而且也是一个学习类和对象的很好的例子。

10.10.1 构造字符串

可以用字符串直接量或字符数组创建一个字符串对象。使用如下语法, 用字符串直接量创建一个字符串:

```
String newString = new String(stringLiteral);
```

参数 `StringLiteral` 是一个括在双引号内的字符序列。下面的语句为字符串直接量 "Welcome to Java" 创建一个 `String` 对象 `message`:

```
String message = new String("Welcome to Java");
```

Java 将字符串直接量看作 `String` 对象。所以, 下面的语句是合法的:

```
String message = "Welcome to Java";
```

还可以用字符数组创建一个字符串。例如，下述语句构造一个字符串 "Good Day"：

```
char[] charArray = {'G', 'o', 'o', 'd', ' ', ' ', 'D', 'a', 'y'};
String message = new String(charArray);
```

注意：String 变量存储的是对 String 对象的引用，String 对象里存储的才是字符串的值。严格地讲，术语 String 变量、String 对象和字符串值是不同的。但在大多数情况下，它们之间的区别是可以忽略的。为简单起见，术语字符串将经常被用于指 String 变量、String 对象和字符串的值。

10.10.2 不可变字符串与限定字符串

String 对象是不可变的，它的内容是不能改变的。下列代码会改变字符串的内容吗？

```
String s = "Java";
s = "HTML";
```

答案是不能。第一条语句创建了一个内容为 "Java" 的 String 对象，并将其引用赋值给 s。第二条语句创建了一个内容为 "HTML" 的新 String 对象，并将其引用赋值给 s。赋值后第一个 String 对象仍然存在，但是不能再访问它，因为变量 s 现在指向了新的对象，如图 10-15 所示。

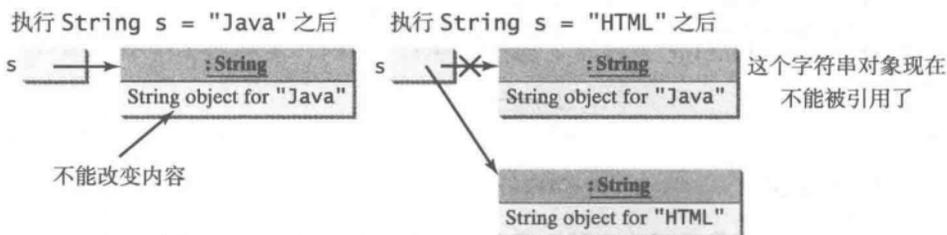
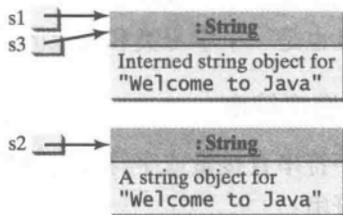


图 10-15 字符串是不可变的；一旦创建，它们的内容是不能修改的

因为字符串在程序设计中是不可变的，但同时又会频繁地使用，所以 Java 虚拟机为了提高效率并节约内存，对具有相同字符序列的字符串直接量使用同一个实例。这样的实例称为限定的 (interned) 字符串。例如，下面的语句：

```
String s1 = "Welcome to Java";
String s2 = new String("Welcome to Java");
String s3 = "Welcome to Java";
System.out.println("s1 == s2 is " + (s1 == s2));
System.out.println("s1 == s3 is " + (s1 == s3));
```



程序结果显示：

```
s1 == s2 is false
s1 == s3 is true
```

在上述语句中，由于 s1 和 s3 指向相同的限定字符串 "Welcome to Java"，因此，s1==s3 为 true。但是，s1==s2 为 false，这是因为尽管 s1 和 s2 的内容相同，但它们是不同的字符串对象。

10.10.3 字符串的替换和分隔

String 类提供替换和分隔字符串的方法，如图 10-16 所示。

java.lang.String	
+replace(oldChar: char, newChar: char): String	将字符串中所有匹配的字符替换成新的字符，然后返回新的字符串
+replaceFirst(oldString: String, newString: String): String	将字符串中第一个匹配的子字符串替换成新的子字符串，然后返回新的字符串
+replaceAll(oldString: String, newString: String): String	将字符串中所有匹配的子字符串替换成新的子字符串，然后返回新的字符串
+split(delimiter: String): String[]	返回一个字符串数组，其中包含被分隔符分隔的子字符串集

图 10-16 String 类包含替换和分隔字符串的方法

一旦创建了字符串，它的内容就不能改变。但是，方法 `replace`、`replaceFirst` 和 `replaceAll` 会返回一个源自原始字符串的新字符串（并未改变原始字符串！）。方法 `replace` 有好几个版本，它们实现用新的字符或子串替换字符串中的某个字符或子串。

例如：

```
"Welcome".replace('e', 'A') 返回一个新的字符串, WAlcomA.
"Welcome".replaceFirst("e", "AB") 返回一个新的字符串, WABlcome.
"Welcome".replace("e", "AB") 返回一个新的字符串, WABlcomAB.
"Welcome".replace("el", "AB") 返回一个新的字符串, WABcome.
```

`split` 方法可以从一个指定分隔符的字符串中提取标识。例如，下面的代码：

```
String[] tokens = "Java#HTML#Perl".split("#");
for (int i = 0; i < tokens.length; i++)
    System.out.print(tokens[i] + " ");
```

显示

```
Java HTML Perl
```

10.10.4 依照模式匹配、替换和分隔

我们经常需要编写代码来验证用户输入，比如检测输入是否是一个数字，或者是否是一个全部小写字母的字符串，或者是否是一个社会安全号。如何编写这类代码呢？一个简单有效的完成该任务的方法是使用正则表达式。

正则表达式 (regular expression) (缩写 `regex`) 是一个字符串，用于描述匹配一个字符串集的模式。可以通过指定某个模式来匹配、替换或分隔一个字符串。这是一种非常有用且功能强大的特性。

从 `String` 类中的 `matches` 方法开始。乍一看，`matches` 方法和 `equals` 方法非常相似。例如，下面两条语句的值均为 `true`：

```
"Java".matches("Java");
"Java".equals("Java");
```

但是，`matches` 方法的功能更强大。它不仅可以匹配定长的字符串，还能匹配一套遵从某种模式的字符串。例如，下面语句的结果均为 `true`：

```
"Java is fun".matches("Java.*")
"Java is cool".matches("Java.*")
"Java is powerful".matches("Java.*")
```

在前面语句中的 "Java.*" 是一个正则表达式。它描述的字符串模式是以字符串 Java 开始的，后面紧跟任意 0 个或多个字符。这里，子串 .* 与 0 个或多个字符相匹配。

下面语句结果为 true。

```
"440-02-4534".matches("\\d{3}-\\d{2}-\\d{4}")
```

这里 \\d 表示单个数字位，\\d{3} 表示三个数字位。

方法 replaceAll、replaceFirst 和 split 也可以和正则表达式结合在一起使用。例如，下面的语句用字符串 NNN 替换 "a+b\$#c" 中的 \$、+ 或者 #，然后返回一个新字符串。

```
String s = "a+b$#c".replaceAll("[$+#]", "NNN");
System.out.println(s);
```

这里的正则表达式 [\$+#] 表示能够匹配 \$、+ 或者 # 的模式。所以，输出是 aNNNbNNNNNNc。

下面的语句将字符串分隔为由标点符号分隔开的字符串数组。

```
String[] tokens = "Java,C?C#,C++".split("[.,:;?]*");

for (int i = 0; i < tokens.length; i++)
    System.out.println(tokens[i]);
```

这里的正则表达式 [.,:;?*] 指定的模式是指匹配 .、,、:、; 或者 ?。这里的每个字符都是分隔字符串的分隔符。因此，这个字符串就被分割成 Java、C、C# 和 C++，它们都存储在数组 tokens 中。

正则表达式对起步阶段的学生来讲可能会比较复杂。基于这个原因，本节只使用两个简单的模式。若要进行进一步的学习，请参照补充材料 H。

10.10.5 字符串与数组之间的转换

字符串不是数组，但是字符串可以转换成数组，反之亦然。为了将字符串转换成一个字符串数组，可以使用 toCharArray 方法。例如，下述语句将字符串 "Java" 转换成一个数组：

```
char[] chars = "Java".toCharArray();
```

因此，chars[0] 是 'J'，chars[1] 是 'a'，chars[2] 是 'v'，chars[3] 是 'a'。

还可以使用方法 getChars(int srcBegin,int srcEnd,char[]dst,int dstBegin) 将下标从 srcBegin 到 srcEnd-1 的子串复制到字符数组 dst 中下标从 dstBegin 开始的位置。例如，下面的代码将字符串 "CS3720" 中下标从 2 到 6-1 的子串 "3720" 复制到字符数组 dst 中下标从 4 开始的位置：

```
char[] dst = {'J', 'A', 'V', 'A', '1', '3', '0', '1'};
"CS3720".getChars(2, 6, dst, 4);
```

这样，dst 就变成了 {'J','A','V','A','3','7','2','0'}。

为了将一个字符数组转换成一个字符串，应该使用构造方法 String(char[]) 或者方法 valueOf(char[])。例如，下面的语句使用 String 构造方法由一个数组构造一个字符串：

```
String str = new String(new char[]{'J', 'a', 'v', 'a'});
```

下面的语句使用 valueOf 方法由一个数组构造一个字符串：

```
String str = String.valueOf(new char[]{'J', 'a', 'v', 'a'});
```

10.10.6 将字符和数值转换成字符串

回顾下，可以使用 `Double.parseDouble(str)` 或者 `Integer.parseInt(str)` 将一个字符串转换为一个 `double` 值或者一个 `int` 值，也可以使用字符串的连接操作符来将字符或者数字转换为字符串。另外一种将数字转换为字符串的方法是使用重载的静态 `valueOf` 方法。该方法可以用于将字符和数值转换成字符串，如图 10-17 所示。

java.lang.String	
<code>+valueOf(c: char): String</code>	返回包含字符 <code>c</code> 的字符串
<code>+valueOf(data: char[]): String</code>	返回包含数组中字符的字符串
<code>+valueOf(d: double): String</code>	返回表示 <code>double</code> 值的字符串表述
<code>+valueOf(f: float): String</code>	返回表示 <code>float</code> 值的字符串表述
<code>+valueOf(i: int): String</code>	返回表示 <code>int</code> 值的字符串表述
<code>+valueOf(l: long): String</code>	返回表示 <code>long</code> 值的字符串表述
<code>+valueOf(b: boolean): String</code>	返回表示 <code>boolean</code> 值的字符串表述

图 10-17 String 类包含从基本类型值创建字符串的静态方法

例如，为了将 `double` 值 5.44 转换成字符串，可以使用 `String.valueOf(5.44)`。返回值是由字符 '5'、'.'、'4' 和 '4' 构成的字符串。

10.10.7 格式化字符串

`String` 类包含静态 `format` 方法，它可以创建一个格式化的字符串。调用该方法的语法是：

```
String.format(format, item1, item2, ..., itemk)
```

这个方法很像 `printf` 方法，只是 `format` 方法返回一个格式化的字符串，而 `printf` 方法显示一个格式化的字符串。例如：

```
String s = String.format("%7.2f%6d%-4s", 45.556, 14, "AB");
System.out.println(s);
```

显示

```
□□45.56□□□□14AB□□
```

注意

```
System.out.printf(format, item1, item2, ..., itemk);
```

等价于

```
System.out.print(
    String.format(format, item1, item2, ..., itemk));
```

这里，小方形框表示一个空格。

复习题

10.15 假设 `s1`、`s2`、`s3`、`s4` 是四个字符串，给定如下语句：

```
String s1 = "Welcome to Java";
String s2 = s1;
String s3 = new String("Welcome to Java");
String s4 = "Welcome to Java";
```

下面表达式的结果是什么？

- a. `s1 == s2`
- b. `s1 == s3`
- c. `s1 == s4`
- d. `s1.equals(s3)`
- e. `s1.equals(s4)`
- f. `"Welcome to Java".replace("Java", "HTML")`
- g. `s1.replace('o', 'T')`
- h. `s1.replaceAll("o", "T")`
- i. `s1.replaceFirst("o", "T")`
- j. `s1.toCharArray()`

10.16 为了创建一个字符串 `Welcome to java`，可能采用下面的语句：

```
String s = "Welcome to Java";
```

或者

```
String s = new String("Welcome to Java");
```

哪个更好？为什么？

10.17 下面代码的输出是什么？

```
String s1 = "Welcome to Java";
String s2 = s1.replace("o", "abc");
System.out.println(s1);
System.out.println(s2);
```

10.18 假设 `s1` 是 `"Welcome"` 而 `s2` 是 `"welcome"`，为下面的陈述编写代码：

- a. 用 `E` 替换 `s1` 中所有出现字符 `e` 的地方，然后将新字符串赋值给 `s2`。
- b. 将 `"Welcome to Java and HTML"` 按空格分隔为一个数组 `tokens`，将前面两个标识赋值给 `s1` 和 `s2`。

10.19 `String` 类中是否有可以改变字符串内容的方法？

10.20 假设字符串 `s` 是用 `new String()` 创建的，那么 `s.length()` 是多少？

10.21 如何将一个 `char` 值、一个字符数组或一个数值转换为一个字符串？

10.22 为什么下面的代码会造成 `NullPointerException` 异常？

```
1 public class Test {
2     private String text;
3
4     public Test(String s) {
5         String text = s;
6     }
7
8     public static void main(String[] args) {
9         Test test = new Test("ABC");
10        System.out.println(test.text.toLowerCase());
11    }
12 }
```

10.23 下面程序的错误是什么？

```
1 public class Test {
2     String text;
3
4     public void Test(String s) {
5         text = s;
6     }
7 }
```

```

7
8 public static void main(String[] args) {
9     Test test = new Test("ABC");
10    System.out.println(test);
11 }
12 }

```

10.24 给出下面代码的输出结果。

```

public class Test {
    public static void main(String[] args) {
        System.out.println("Hi, ABC, good".matches("ABC "));
        System.out.println("Hi, ABC, good".matches(".*ABC.*"));
        System.out.println("A,B;C".replaceAll(",;", "#"));
        System.out.println("A,B;C".replaceAll("[,;]", "#"));

        String[] tokens = "A,B;C".split("[,;]");
        for (int i = 0; i < tokens.length; i++)
            System.out.print(tokens[i] + " ");
    }
}

```

10.25 给出下面代码的输出结果。

```

public class Test {
    public static void main(String[] args) {
        String s = "Hi, Good Morning";
        System.out.println(m(s));
    }

    public static int m(String s) {
        int count = 0;
        for (int i = 0; i < s.length(); i++)
            if (Character.isUpperCase(s.charAt(i)))
                count++;

        return count;
    }
}

```

10.11 StringBuilder 和 StringBuffer 类

 **要点提示:** StringBuilder 和 StringBuffer 类似于 String 类，区别在于 String 类是不可改变的。

一般来说，只要使用字符串的地方，都可以使用 StringBuilder/StringBuffer 类。StringBuilder/StringBuffer 类比 String 类更灵活。可以给一个 StringBuilder 或 StringBuffer 中添加、插入或追加新的内容，但是 String 对象一旦创建，它的值就确定了。

除了 StringBuffer 中修改缓冲区的方法是同步的，这意味着只有一个任务被允许执行方法之外，StringBuilder 类与 StringBuffer 类是很相似的。如果是多任务并发访问，就使用 StringBuffer，因为这种情况下需要同步以防止 StringBuffer 崩溃。并发编程将在第 30 章介绍。而如果是单任务访问，使用 StringBuilder 会更有效。StringBuffer 和 StringBuilder 中的构造方法和其他方法几乎是完全一样的。本节介绍 StringBuilder。在本节的所有地方 StringBuilder 都可以替换为 StringBuffer。程序可以不经任何修改进行编译和运行。

StringBuilder 类有 3 个构造方法和 30 多个用于管理构建器或修改构建器内字符串的方法。可以使用构造方法创建一个空的构建器或从一个字符串创建一个构建器，如图 10-18 所示。

java.lang.StringBuilder	
+StringBuilder()	构建一个容量为 16 的空字符串构建器
+StringBuilder(capacity: int)	构建一个指定容量的字符串构建器
+StringBuilder(s: String)	构建一个指定字符串的字符串构建器

图 10-18 StringBuilder 类包含创建 StringBuilder 实例的构造方法

10.11.1 修改 StringBuilder 中的字符串

可以使用图 10-19 中列出的方法，在字符串构建器的末尾追加新内容，在字符串构建器的特定位置插入新的内容，还可以删除或替换字符串构建器中的字符。

java.lang.StringBuilder	
+append(data: char[]): StringBuilder	追加一个字符数组到字符串构建器
+append(data: char[], offset: int, len: int): StringBuilder	追加 data 中的子数组到字符串构建器
+append(v: aPrimitiveType): StringBuilder	将一个基本类型值作为字符串追加到字符串构建器
+append(s: String): StringBuilder	追加一个字符串到字符串构建器
+delete(startIndex: int, endIndex: int): StringBuilder	删除从 startIndex 到 endIndex-1 的字符
+deleteCharAt(index: int): StringBuilder	删除给定索引位置的字符
+insert(index: int, data: char[], offset: int, len: int): StringBuilder	在字符串构建器的给定索引位置插入数组 data 的子数组
+insert(offset: int, data: char[]): StringBuilder	向构建器的偏移位置插入数据
+insert(offset: int, b: aPrimitiveType): StringBuilder	向该字符串构建器插入一个转换为字符串的值
+insert(offset: int, s: String): StringBuilder	在该构建器指定的偏移位置插入一个字符串
+replace(startIndex: int, endIndex: int, s: String): StringBuilder	将该构建器从 startIndex 到 endIndex-1 的位置的字符替换为给定的字符串
+reverse(): StringBuilder	倒置构建器中的字符
+setCharAt(index: int, ch: char): void	将该构建器的指定索引位置设为新的字符

图 10-19 StringBuilder 类包含修改字符串构建器的方法

StringBuilder 类提供了几个重载方法，可以将 boolean、char、char 数组、double、float、int、long 和 String 类型值追加到字符串构建器。例如，下面的代码将字符串和字符追加到 stringBuilder，构成新的字符串 "Welcome to Java"。

```
StringBuilder stringBuilder = new StringBuilder();
stringBuilder.append("Welcome");
stringBuilder.append(' ');
stringBuilder.append("to");
stringBuilder.append(' ');
stringBuilder.append("Java");
```

StringBuilder 类也包括几个重载的方法，可以将 boolean、char、char 数组、double、float、int、long 和 String 类型值插入到字符串构建器。考虑下面的代码：

```
stringBuilder.insert(11, "HTML and ");
```

假设在应用 insert 方法之前，stringBuilder 包含的字符串是 "Welcome to Java"。上

面的代码就在 `stringBuilder` 的第 11 个位置（就在 J 之前）插入 "HTML and"。新的 `stringBuilder` 就变成 "Welcome to HTML and Java"。

也可以使用两个 `delete` 方法将字符从构建器中的字符串中删除，使用 `reverse` 方法倒置字符串，使用 `replace` 方法替换字符串中的字符，或者使用 `setCharAt` 方法在字符串中设置一个新字符。

例如，假设在应用下面的每个方法之前，`stringBuilder` 包含的是 "Welcome to Java"。

`stringBuilder.delete(8,11)` 将构建器变为 `Welcome Java`。

`stringBuilder.deleteCharAt(8)` 将构建器变为 `Welcome o Java`。

`stringBuilder.reverse()` 将构建器变为 `avaJ ot emocleW`。

`stringBuilder.replace(11,15,"HTML")` 将构建器变为 `Welcome to HTML`。

`stringBuilder.setCharAt(0,'w')` 将构建器变为 `welcome to Java`。

除了 `setCharAt` 方法之外，所有这些进行修改的方法都做两件事：

- 改变字符串构建器的内容。
- 返回字符串构建器的引用。

例如，下面的语句：

```
StringBuilder stringBuilder1 = stringBuilder.reverse();
```

将构建器中的字符倒置并把构建器的引用赋值给 `stringBuilder1`。这样，`stringBuilder` 和 `stringBuilder1` 都指向同一个 `StringBuffer` 对象。回顾一下，如果对方法的返回值不感兴趣，所有带返回值类的方法都可以被当作语句调用。在这种情况下，Java 就简单地忽略掉返回值。例如，下面的语句

```
stringBuilder.reverse();
```

它的返回值就被忽略了。

 **提示：**如果一个字符串不需要任何改变，则使用 `String` 类而不使用 `StringBuffer` 类。Java 可以完成对 `String` 类的优化，例如，共享限定字符串等。

10.11.2 `toString`、`capacity`、`length`、`setLength` 和 `charAt` 方法

`StringBuilder` 类提供了许多其他处理字符串构建器和获取它的属性的方法，如图 10-20 所示。

java.lang.StringBuilder	
+toString(): String	从字符串构建器返回一个字符串对象
+capacity(): int	返回该字符串构建器的容量
+charAt(index: int): char	返回指定索引位置的字符
+length(): int	返回该构建器中的字符数
+setLength(newLength: int): void	设置该构建器的新的长度
+substring(startIndex: int): String	返回从 <code>startIndex</code> 开始的子字符串
+substring(startIndex: int, endIndex: int): String	返回从 <code>startIndex</code> 到 <code>endIndex-1</code> 的子字符串
+trimToSize(): void	减少用于字符串构建器的存储大小

图 10-20 `StringBuilder` 类包括修改字符串构建器的方法

`capacity()` 方法返回字符串构建器当前的容量。容量是指在不增加构建器大小的情况

下能够存储的字符数量。

`length()` 方法返回字符串构建器中实际存储的字符数量。`setLength(newLength)` 方法设置字符串构建器的长度。如果参数 `newLength` 小于字符串构建器的当前长度，则字符串构建器会被截短到恰好能包含由参数 `newLength` 给定的字符个数。如果参数 `newLength` 大于或等于当前长度，则给字符串构建器追加足够多的空字符 (`'\u0000'`)，使其长度 `length` 变成新参数 `newLength`。参数 `newLength` 必须大于等于 0。

`charAt(index)` 方法返回字符串构建器中某个特定下标 `index` 的字符。下标是基于 0 的，字符串构建器中的第一个字符的下标为 0，第二个字符的下标为 1，依此类推。参数 `index` 必须大于或等于 0，并且小于字符串构建器的长度。

注意：字符串的长度总是小于或等于构建器的容量。长度是存储在构建器中的字符串的实际大小，而容量是构建器的当前大小。如果有更多的字符添加到字符串构建器，超出它的容量，则构建器的容量就会自动增加。在计算机内部，字符串构建器是一个字符数组，因此，构建器的容量就是数组的大小。如果超出构建器的容量，就用新的数组替换现有数组。新数组的大小为 $2 \times (\text{之前数组的长度} + 1)$ 。

提示：可以使用 `new StringBuilder(initialCapacity)` 创建指定初始容量的 `StringBuilder`。通过仔细选择初始容量能够使程序更有效。如果容量总是超过构建器的实际使用长度，JVM 将永远不需要为构建器重新分配内存。另一方面，如果容量过大将会浪费内存空间。可以使用 `trimToSize()` 方法将容量降到实际的大小。

10.11.3 示例学习：判断回文串时忽略既非字母又非数字的字符

程序清单 5-14 考虑字符串中的所有字符来检测字符串是否是回文串。编写一个新程序，检测一个字符串在忽略既非字母又非数字的字符时是否是一个回文串。

下面是解决这个问题的步骤：

1) 通过删除既非字母又非数字的字符过滤这个字符串。要做到这一点，需要创建一个空字符串构建器，将字符串中每一个字母或数字字符添加到字符串构建器中，然后从这个构建器返回所求的字符串。可以使用 `Character` 类中的 `isLetterOrDigit(ch)` 方法来检测字符 `ch` 是否是字母或数字。

2) 倒置过滤后的字符串得到一个新字符串。使用 `equals` 方法对倒置后的字符串和过滤后的字符串进行比较。

完整的程序如程序清单 10-10 所示。

程序清单 10-10 PalindromeIgnoreNonAlphanumeric.java

```
1 import java.util.Scanner;
2
3 public class PalindromeIgnoreNonAlphanumeric {
4     /** Main method */
5     public static void main(String[] args) {
6         // Create a Scanner
7         Scanner input = new Scanner(System.in);
8
9         // Prompt the user to enter a string
10        System.out.print("Enter a string: ");
11        String s = input.nextLine();
12
13        // Display result
```

```
14     System.out.println("Ignoring nonalphanumeric characters, \nis "
15         + s + " a palindrome? " + isPalindrome(s));
16 }
17
18 /** Return true if a string is a palindrome */
19 public static boolean isPalindrome(String s) {
20     // Create a new string by eliminating nonalphanumeric chars
21     String s1 = filter(s);
22
23     // Create a new string that is the reversal of s1
24     String s2 = reverse(s1);
25
26     // Check if the reversal is the same as the original string
27     return s2.equals(s1);
28 }
29
30 /** Create a new string by eliminating nonalphanumeric chars */
31 public static String filter(String s) {
32     // Create a string builder
33     StringBuilder stringBuilder = new StringBuilder();
34
35     // Examine each char in the string to skip alphanumeric char
36     for (int i = 0; i < s.length(); i++) {
37         if (Character.isLetterOrDigit(s.charAt(i))) {
38             stringBuilder.append(s.charAt(i));
39         }
40     }
41
42     // Return a new filtered string
43     return stringBuilder.toString();
44 }
45
46 /** Create a new string by reversing a specified string */
47 public static String reverse(String s) {
48     StringBuilder stringBuilder = new StringBuilder(s);
49     stringBuilder.reverse(); // Invoke reverse in StringBuilder
50     return stringBuilder.toString();
51 }
52 }
```

```
Enter a string: ab<c>cb?a 
Ignoring nonalphanumeric characters,
is ab<c>cb?a a palindrome? true
```

```
Enter a string: abcc><?cab 
Ignoring nonalphanumeric characters,
is abcc><?cab a palindrome? false
```

`filter(String s)` 方法 (第 31 ~ 44 行) 逐个地检测字符串 `s` 中的每个字符, 如果字符是字母或数字字符, 就将它复制到字符串构建器。`filter` 方法返回构建器中的字符串。`reverse(String s)` 方法 (第 47 ~ 51 行) 创建一个新字符串, 这个新串是对给定字符串 `s` 的倒置。`filter` 方法和 `reverse` 方法都会返回一个新字符串。原始字符串并没有改变。

程序清单 5-14 中的程序通过比较字符串两端的一对字符来检测一个字符串是否是回文串。程序清单 10-10 使用 `StringBuilder` 类中的 `reverse` 方法倒置字符串, 然后比较两个字符串是否相等以判断原始字符串是否是回文串。

复习题

10.26 `StringBuilder` 和 `StringBuffer` 之间的区别是什么?

- 10.27 如何为一个字符串创建字符串构建器? 如何从一个字符串构建器获取字符串?
- 10.28 使用 `StringBuilder` 类中的 `reverse` 方法编写三条语句, 倒置字符串 `s`。
- 10.29 编写三条语句, 从包含 20 个字符的字符串 `s` 中删除下标从 4 到 10 的子串。使用 `StringBuilder` 类中的 `delete` 方法。
- 10.30 字符串和字符串构建器内部用什么存储字符?
- 10.31 假设给出如下所示的 `s1` 和 `s2`:

```
StringBuilder s1 = new StringBuilder("Java");
StringBuilder s2 = new StringBuilder("HTML");
```

显示执行下列每条语句之后 `s1` 的结果。假定这些表达式都是相互独立的。

- a. `s1.append(" is fun");`
- b. `s1.append(s2);`
- c. `s1.insert(2, "is fun");`
- d. `s1.insert(1, s2);`
- e. `s1.charAt(2);`
- f. `s1.length();`
- g. `s1.deleteCharAt(3);`
- h. `s1.delete(1, 3);`
- i. `s1.reverse();`
- j. `s1.replace(1, 3, "Computer");`
- k. `s1.substring(1, 3);`
- l. `s1.substring(2);`

- 10.32 给出下面程序的输出结果:

```
public class Test {
    public static void main(String[] args) {
        String s = "Java";
        StringBuilder builder = new StringBuilder(s);
        change(s, builder);

        System.out.println(s);
        System.out.println(builder);
    }

    private static void change(String s, StringBuilder builder) {
        s = s + " and HTML";
        builder.append(" and HTML");
    }
}
```

关键术语

Abstract data type (ADT) 抽象数据类型 (ADT)	composition (组合)
Aggregation (聚集)	has-a relationship (拥有关系)
Boxing (装箱)	multiplicity (多重性)
class abstraction (类抽象)	stack (栈)
class encapsulation (类封装)	unboxing (开箱)
class contract (类的合约)	

本章小结

1. 面向过程范式重在设计方法。面向对象范式将数据和方法耦合在对象中。使用面向对象范式的软件设计重在对象和对象上的操作。面向对象方法结合了面向过程范式的功能以及将数据和操作集成在对象中的特点。
2. 许多 Java 方法要求使用对象作为参数。Java 提供了一个便捷的办法，将基本数据类型合并或包装到一个对象中（例如，包装 int 值到 Integer 类中，包装 double 值到 Double 类中）。
3. Java 可以根据上下文自动地将基本类型值转换为对应的包装对象，反之亦然。
4. BigInteger 类在计算和处理任意大小的整数方面是很有用的。BigDecimal 类可以用作计算和处理带任意精度的浮点数。
5. String 对象是不可变的，它的内容不能改变。为了提高效率和节省内存，如果两个直接量字符串有相同的字符序列，Java 虚拟机就将它们存储在一个对象中。这个独特的对象称为限定字符串对象。
6. 正则表达式（缩写 regex）是一个描述模板的字符串，用于匹配一系列字符串。可以通过指定一个模板来匹配、替代或者分隔字符串。
7. StringBuilder/StringBuffer 类可以用来替代 String 类。String 对象是不可变的，但是可以向 StringBuilder/StringBuffer 对象中添加、插入或追加新的内容。如果字符串的内容不需要任何改变，就使用 String 类；如果可能改变的话，则使用 StringBuilder/StringBuffer 类。

测试题

在线回答本章节的测试题，位于 www.cs.armstrong.edu/liang/intro10e/quiz.html。

编程练习题

10.2 ~ 10.3 节

*10.1（时间类 Time）设计一个名为 Time 的类。这个类包含：

- 表示时间的数据域 hour、minute 和 second。
- 一个以当前时间创建 Time 对象的无参构造方法（数据域的值表示当前时间）。
- 一个构造 Time 对象的构造方法，这个对象有一个特定的时间值，这个值是以毫秒表示的、从 1970 年 1 月 1 日午夜开始到现在流逝的时间段（数据域的值表示这个时间）。
- 一个构造带特定的小时、分钟和秒的 Time 对象的构造方法。
- 三个数据域 hour、minute 和 second 各自的 get 方法。
- 一个名为 setTime(long elapsedTime) 的方法使用流逝的时间给对象设置一个新时间。例如，如果流逝的时间为 555550000 毫秒，则转换为 10 小时、10 分钟、10 秒。

画出该类的 UML 图并实现这个类。编写一个测试程序，创建两个 Time 对象（使用 new Time() 和 new Time(555550000)），然后显示它们的小时、分钟和秒。

提示：前两个构造方法可以从流逝的时间中提取出小时、分钟和秒。对于无参构造方法，当前时间可以使用 System.currentTimeMillis() 获取当前时间，如程序清单 2-7 所示。

10.2（BMI 类）将下面的新构造方法加入到 BMI 类中：

```
/** Construct a BMI with the specified name, age, weight,
 * feet, and inches
 */
public BMI(String name, int age, double weight, double feet,
           double inches)
```

10.3（MyInteger 类）设计一个名为 MyInteger 的类。这个类包括：

- 一个名为 value 的 int 型数据域，存储这个对象表示的 int 值。

- 一个为指定的 `int` 值创建 `MyInteger` 对象的构造方法。
- 一个返回 `int` 值的 `get` 方法。
- 如果值分别为偶数、奇数或素数，那么 `isEven()`、`isOdd()` 和 `isPrime()` 方法都会返回 `true`。
- 如果指定值分别为偶数、奇数或素数，那么相应的静态方法 `isEven(int)`、`isOdd(int)` 和 `isPrime(int)` 会返回 `true`。
- 如果指定值分别为偶数、奇数或素数，那么相应的静态方法 `isEven(MyInteger)`、`isOdd(MyInteger)` 和 `isPrime(MyInteger)` 会返回 `true`。
- 如果该对象的值与指定的值相等，那么 `equals(int)` 和 `equals(MyInteger)` 方法返回 `true`。
- 静态方法 `parseInt(char[])` 将数字字符构成的数组转换为一个 `int` 值。
- 静态方法 `parseInt(String)` 将一个字符串转换为一个 `int` 值。

画出该类的 UML 图并实现这个类。编写客户程序测试这个类中的所有方法。

10.4 (MyPoint 类) 设计一个名为 `MyPoint` 的类，表示一个带 `x` 坐标和 `y` 坐标的点。该类包括：

- 两个带 `get` 方法的数据域 `x` 和 `y` 分别表示它们的坐标。
- 一个创建点 (0,0) 的无参构造方法。
- 一个创建特定坐标点的构造方法。
- 一个名为 `distance` 的方法，返回从该点到 `MyPoint` 类型的指定点之间的距离。
- 一个名为 `distance` 的方法，返回从该点到指定 `x` 和 `y` 坐标的指定点之间的距离。

画出该类的 UML 图并实现这个类。编写一个测试程序，创建两个点 (0,0) 和 (10,30.5)，并显示它们之间的距离。

10.4 ~ 10.8 节

- *10.5 (显示素数因子) 编写一个程序，提示用户输入一个正整数，然后以降序显示它的所有最小因子。例如：如果整数为 120，那么显示的最小因子为 5、3、2、2、2。使用 `StackOfIntegers` 类存储因子（例如：2、2、2、3、5），获取之后按倒序显示这些因子。
- *10.6 (显示素数) 编写一个程序，然后以降序显示小于 120 的所有素数。使用 `StackOfIntegers` 类存储这些素数（例如：2、3、5、...），获取之后按倒序显示它们。
- **10.7 (游戏：ATM 机) 使用编程练习题 9.7 中创建的 `Account` 类来模拟一台 ATM 机。创建一个有 10 个账户的数组，其 `id` 为 0, 1, ..., 9，并初始化收支为 100 美元。系统提示用户输入一个 `id`。如果输入的 `id` 不正确，就要求用户输入正确的 `id`。一旦接受一个 `id`，就显示如运行示例所示的主菜单。可以选择 1 来查看当前的收支，选择 2 表示取钱，选择 3 表示存钱，选择 4 表示退出主菜单。一旦退出，系统就会提示再次输入 `id`。所以，系统一旦启动就不会停止。

```

Enter an id: 4 
Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 1 
The balance is 100.0

```

```

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 2 ↵
Enter an amount to withdraw: 3 ↵

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 1 ↵
The balance is 97.0

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 3 ↵
Enter an amount to deposit: 10 ↵

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 1 ↵
The balance is 107.0

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 4 ↵

Enter an id:

```

***10.8 (财务：税款类 Tax) 编程练习题 8.12 使用数组编写一个计算税款的程序。设计一个名为 Tax 的类，该类包含下面的实例数据域。

- `int filingStatus` (四种纳税人状态之一)：0——单身纳税人、1——已婚共缴纳税人或合法寡妇、2——已婚单缴纳税人、3——家庭纳税人。使用公共静态常量 `SINGLE_FILER(0)`、`MARRIED_JOINTLY_OR_QUALIFYING_WIDOW(ER)(1)`、`MARRIED_SEPARATELY(2)` 和 `HEAD_OF_HOUSEHOLD(3)` 表示这些状态。
- `int[][] brackets`：存储每种纳税人的纳税等级。
- `double[] rates`：存储每种纳税等级的税率。
- `double taxableIncome`：存储可征税收入。

给每个数据域提供 `get` 和 `set` 方法，并提供返回税款的 `getTax()` 方法。该类还提供一个无参构造方法和构造方法 `Tax(filingStatus, brackets, rates, taxableIncome)`。

画出该类的 UML 图并实现这个类。编写一个测试程序，使用 Tax 类对所给四种纳税人打印 2001 年和 2009 年的税款表，可征税收入范围在 50 000 美元和 60 000 美元之间，间隔区间为 1000 美元。2009 年的税率参见表 3-2，2001 年的税率参见表 10-1。

表 10-1 2001 年美国联邦个人所得税税率表

税率	单身纳税人	已婚共缴纳税人或符合条件的丧偶人士	已婚单缴纳税人	家庭纳税人
15%	\$27 050 以下	\$45 200 以下	\$22 600 以下	\$36 250 以下
27.5%	\$27 051 ~ \$65 550	\$45 201 ~ \$109 250	\$22 601 ~ \$54 625	\$36 251-\$93 650
30.5%	\$65 551 ~ \$136 750	\$109 251 ~ 166 500	\$54 626 ~ \$83 250	\$93 651-\$151 650
35.5%	\$136 751 ~ \$29 7350	\$166 501 ~ \$297 350	\$83 251 ~ \$148 675	\$151 651-\$297 350
39.1%	\$297 351 及以上	\$297 351 及以上	\$148 676 及以上	\$297 351 及以上

**10.9 (课程类 Course) 如下改写 Course 类:

- 程序清单 10-6 中数组的大小是固定的。对它进行改进, 通过创建一个新的更大的数组并复制当前数组的内容来实现数组大小的自动增长。
- 实现 dropStudent 方法。
- 添加一个名为 clear() 的新方法, 然后删掉选某门课程的所有学生。

编写一个测试程序, 创建一门课程, 添加三个学生, 删除一个学生, 然后显示这门课程的学生。

*10.10 (Queue 类) 10.6 节给出了一个用于 Stack 的类。设计一个名为 Queue 的类用于存储整数。像栈一样, 队列具有元素。在栈中, 元素以“后进先出”的方式获得。在队列中, 元素以“先进先出”的方式获取。该类包含:

- 一个名为 element 的 int[] 类型的数据域, 保存队列中的 int 值。
- 一个名为 size 的数据域, 保存队列中的元素个数。
- 一个构造方法, 使用默认的容量 8 来创建一个 Queue 对象。
- 方法 enqueue(int v), 用于将 v 加入到队列中。
- 方法 dequeue(), 用于从队列中移除元素并返回该元素。
- 方法 empty(), 如果队列是空的话, 该方法返回 true。
- 方法 getSize(), 返回队列的大小。

画出该类的 UML 图并实现这个类, 使之初始数组的大小为 8。一旦元素个数超过了大小, 数组大小将会翻倍。如果一个元素从数组的开始部分移除, 你需要将数组中的所有元素往左边改变一个位置。编写一个测试程序, 增加从 1 到 20 的 21 个成员, 然后将这些数字移除并显示它们。

*10.11 (几何: Circle2D 类) 定义 Circle2D 类, 包括:

- 两个带有 get 方法的名为 x 和 y 的 double 型数据域, 表明圆的中心点。
- 一个带 get 方法的数据域 radius。
- 一个无参构造方法, 该方法创建一个 (x,y) 值为 (0,0) 且 radius 为 1 的默认圆。
- 一个构造方法, 创建带指定的 x、y 和 radius 的圆。
- 一个返回圆面积的方法 getArea()。
- 一个返回圆周长的方法 getPerimeter()。
- 如果给定的点 (x,y) 在圆内, 那么方法 contains(double x, double y) 返回 true, 如图 10-21a 所示。
- 如果给定的圆在这个圆内, 那么方法 contains(Circle2D circle) 返回 true, 如图 10-21b 所示。
- 如果给定的圆和这个圆重叠, 那么方法 overlaps(Circle2D circle) 返回 true, 如图 10-21c 所示。

画出该类的 UML 图并实现这个类。编写测试程序, 创建一个 Circle2D 对象 c1(new Circle2D(2,2,5.5)), 显示它的面积和周长, 还要显示 c1.contains(3,3)、c1.contains(new Circle2D(4,5,10.5)) 和 c1.overlaps(new Circle2D(3,5,2.3))。



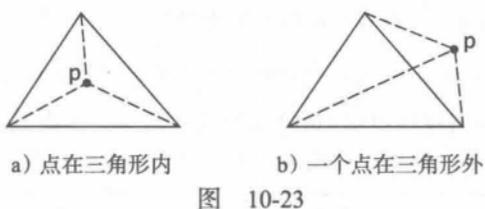
***10.12 (几何: Triangle2D 类) 定义 Triangle2D 类, 包含:

- 三个名为 p1、p2 和 p3 的 MyPoint 类型数据域, 这三个数据域都带有 get 和 set 方法。MyPoint 在编程练习题 10.4 中定义。
- 一个无参构造方法, 该方法创建三个坐标为 (0,0)、(1,1) 和 (2,5) 的点组成的默认三角形。
- 一个创建带指定点的三角形的构造方法。
- 一个返回三角形面积的方法 getArea()。
- 一个返回三角形周长的方法 getPerimeter()。
- 如果给定的点 p 在这个三角形内, 那么方法 contains(MyPoint p) 返回 true, 如图 10-22a 所示。
- 如果给定的三角形在这个三角形内, 那么方法 contains(Triangle2D t) 返回 true, 如图 10-22b 所示。
- 如果给定的三角形和这个三角形重叠, 那么方法 overlaps(Triangle2D t) 返回 true, 如图 10-22c 所示。



画出该类的 UML 图并实现这个类。编写测试程序, 使用构造方法 `new Triangle2D(new MyPoint(2.5,2), new MyPoint(4.2,3), new MyPoint(5,3.5))` 创建一个 Triangle2D 对象 t1, 显示它的面积和周长, 并显示 `t1.contains(3,3)`、`t1.contains(new Triangle2D(new MyPoint(2.9,2), new MyPoint(4,1), MyPoint(1,3.4)))` 和 `t1.overlaps(new Triangle2D(new MyPoint(2,5.5), new MyPoint(4,-3), MyPoint(2,6.5)))` 的结果。

提示: 关于计算三角形面积的公式请参见编程练习题 2.19。为了检测一个点是否在三角形中, 画三条虚线, 如图 10-23 所示。如果点在三角形中, 每条虚线应该和边相交一次。如果虚线和边相交两次, 那么这个点肯定在这个三角形外。找到两条线交点的算法, 参加编程练习题 3.25。



*10.13 (几何: MyRectangle2D 类) 定义 MyRectangle2D 类, 包含:

- 两个名为 x 和 y 的 double 型数据域表明矩形的中心点, 这两个数据域都带有 get 和 set 方法 (假设这个矩形的边与 x 轴和 y 轴平行)。
- 带 get 和 set 方法的数据域 width 和 height。

- 一个无参构造方法，该方法创建一个 (x,y) 值为 (0,0) 且 width 和 height 为 1 的默认矩形。
- 一个构造方法，创建带指定的 x、y、width 和 height 的矩形。
- 方法 `getArea()` 返回矩形的面积。
- 方法 `getPerimeter()` 返回矩形的周长。
- 如果给定的点 (x,y) 在矩形内，那么方法 `contains(double x, double y)` 返回 true，如图 10-24a 所示。
- 如果给定的矩形在这个矩形内，那么方法 `contains(MyRectangle2D r)` 返回 true，如图 10-24b 所示。
- 如果给定的矩形和这个矩形重叠，那么方法 `overlaps(MyRectangle2D r)` 返回 true，如图 10-24c 所示。

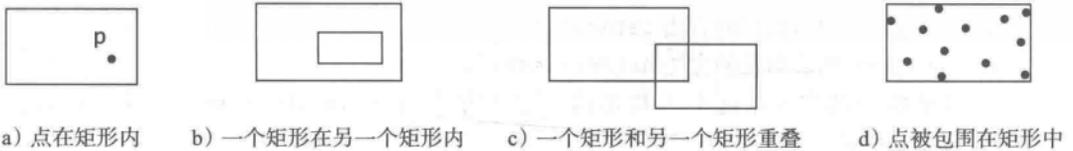


图 10-24

画出该类的 UML 图并实现这个类。编写测试程序，创建一个 `MyRectangle2D` 对象 `r1(new MyRectangle2D(2,2,5.5,4.9))`，显示它的面积和周长，然后显示 `r1.contains(3,3)`、`r1.contains(new MyRectangle2D(4,5,10.5,3.2))` 和 `r1.overlaps(new MyRectangle2D(3,5,2.3,5.4))` 的结果。

*10.14 (MyDate 类) 设计一个名为 `MyDate` 的类。该类包含：

- 表示日期的数据域 `year`、`month` 和 `day`。月份是从 0 开始的，即 0 表示一月份。
- 一个无参构造方法，该方法创建当前日期的 `MyDate` 对象。
- 一个构造方法，创建以从 1970 年 1 月 1 日午夜开始流逝的毫秒数为时间的 `MyDate` 对象。
- 一个构造方法，创建一个带指定年、月、日的 `MyDate` 对象。
- 三个数据域 `year`、`month` 和 `day` 的 `get` 方法。
- 一个名为 `setDate(long elapsedTime)` 使用流逝的时间为对象设置新数据的方法。

画出该类的 UML 图并实现这个类。编写测试程序，创建一个测试程序，创建两个 `Date` 对象（使用 `new Date()` 和 `new Date(3435555133101L)`），然后显示它们的小时、分钟和秒。

 提示：前两个构造方法将从逝去的时间中提取出年、月、日。例如：如果逝去的时间是 56155550000 毫秒，那么年就是 1987，月就是 9，而天是 18。可以使用编程练习题 9.5 中讨论的 `GregorianCalendar` 类来简化编程。

*10.15 (几何：边界矩形) 边界矩形是指包围一个二维平面上一系列点的矩形，如图 10-24d 所示。编写一个方法，为二维平面上一系列点返回一个边界矩形，如下所示：

```
public static MyRectangle2D getRectangle(double[][] points)
```

`Rectangle2D` 类在编程练习题 10.13 中定义。编写一个测试程序，提示用户输入 5 个点，然后显示边界矩形的中心、宽度以及高度。下面是一个运行示例：

```
Enter five points: 1.0 2.5 3 4 5 6 7 8 9 10 [Enter]
The bounding rectangle's center (5.0, 6.25), width 8.0, height 7.5
```

10.9 节

*10.16 (被 2 或 3 整除) 找出能被 2 或 3 整除的前 10 个数字，这些数字有 50 个十进制位数。

- *10.17 (平方数) 找出大于 `Long.MAX_VALUE` 的前 10 个平方数。平方数是指形式为 n^2 的数。例如, 4、9 以及 16 都是平方数。找到一种方法, 使你的程序能快速运行。
- *10.18 (大素数) 编写程序找出五个大于 `Long.MAX_VALUE` 的素数。
- *10.19 (Mersenne 素数) 如果一个素数可以写成 $2^p - 1$ 的形式, 那么该素数就称为 Mersenne 素数, 其中的 p 是一个正整数。编写程序找出 $p \leq 100$ 的所有 Mersenne 素数, 然后显示如下所示的输出。(必须使用 `BigInteger` 来存储数字, 因为它太大了, 不能用 `long` 来存储。程序可能需要运行几个小时。)

p	$2^p - 1$
2	3
3	7
5	31
...	

- *10.20 (近似 e) 编程练习题 5.26 使用下面数列近似计算 e :

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \dots + \frac{1}{i!}$$

为了得到更好的精确度, 在计算中使用 25 位精度的 `BigDecimal`。编写程序显示当 $i=100, 200, \dots, 1000$ 时 e 的值。

- 10.21 (被 5 或 6 整除) 找出能被 5 或 6 整除的大于 `Long.MAX_VALUE` 的前 10 个数字。

第 10.10 ~ 10.11 节

- **10.22 (实现 `String` 类) Java 库中提供了 `String` 类, 给出你自己对下面方法的实现 (将新类命名为 `MyString1`):

```
public MyString1(char[] chars);
public char charAt(int index);
public int length();
public MyString1 substring(int begin, int end);
public MyString1 toLowerCase();
public boolean equals(MyString1 s);
public static MyString1 valueOf(int i);
```

- **10.23 (实现 `String` 类) 在 Java 库中提供了 `String` 类, 给出你自己对下面方法的实现 (将新类命名为 `MyString2`):

```
public MyString2(String s);
public int compare(String s);
public MyString2 substring(int begin);
public MyString2 toUpperCase();
public char[] toChars();
public static MyString2 valueOf(boolean b);
```

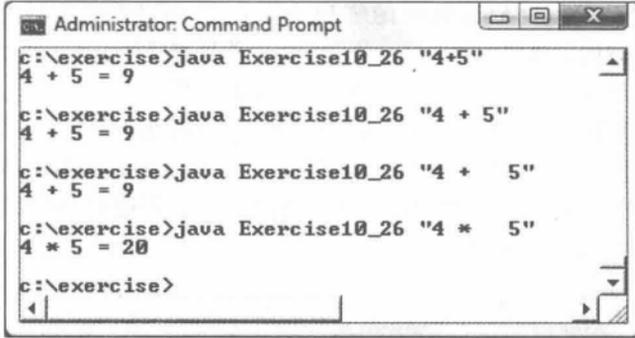
- 10.24 (实现 `Character` 类) 在 Java 库中提供了 `Character` 类, 给出你自己对这个类的实现 (将新类命名为 `MyCharacter`)。

- **10.25 (新的字符串 `split` 方法) `String` 类中的 `split` 方法会返回一个字符串数组, 该数组是由分隔符分隔开的子串构成的。但是, 这个分隔符是不返回的。实现下面的新方法, 方法返回字符串数组, 这个数组由用匹配字符分隔开的子串构成, 子串也包括匹配字符。

```
public static String[] split(String s, String regex)
```

例如, `split("ab#12#453", "#")` 会返回 `ab`、`#`、`12`、`#` 和 `453` 构成的 `String` 数组, 而 `split("a?b?gf#e", "[?#]")` 会返回 `a`、`?`、`b`、`?`、`gf`、`#` 和 `e` 构成的字符串数组。

- *10.26 (计算器) 修改程序清单 7-9, 接收一个字符串的表达式, 其中操作符和操作数由 0 到多个空格隔开。例如, `3+4` 和 `3 + 4` 都是可以接受的表达式。下面是一个运行示例:



```
Administrator: Command Prompt
c:\exercise>java Exercise10_26 "4+5"
4 + 5 = 9

c:\exercise>java Exercise10_26 "4 + 5"
4 + 5 = 9

c:\exercise>java Exercise10_26 "4 + 5"
4 + 5 = 9

c:\exercise>java Exercise10_26 "4 * 5"
4 * 5 = 20

c:\exercise>
```

****10.27** (实现 `StringBuilder` 类) 在 Java 库中提供了 `StringBuilder` 类。给出你自己对下面方法的实现 (将新类命名为 `MyStringBuilder1`):

```
public MyStringBuilder1(String s);
public MyStringBuilder1 append(MyStringBuilder1 s);
public MyStringBuilder1 append(int i);
public int length();
public char charAt(int index);
public MyStringBuilder1 toLowerCase();
public MyStringBuilder1 substring(int begin, int end);
public String toString();
```

****10.28** (实现 `StringBuilder` 类) 在 Java 库中提供了 `StringBuilder` 类。给出你自己对下面方法的实现 (将新类命名为 `MyStringBuilder2`):

```
public MyStringBuilder2();
public MyStringBuilder2(char[] chars);
public MyStringBuilder2(String s);
public MyStringBuilder2 insert(int offset, MyStringBuilder2 s);
public MyStringBuilder2 reverse();
public MyStringBuilder2 substring(int begin);
public MyStringBuilder2 toUpperCase();
```

继承和多态

教学目标

- 通过继承由父类定义子类 (11.2 节)。
- 使用关键字 `super` 调用父类的构造方法和方法 (11.3 节)。
- 在子类中重写方法 (11.4 节)。
- 区分重写和重载的不同 (11.5 节)。
- 探究 `Object` 类中的 `toString()` 方法 (11.6 节)。
- 理解多态和动态绑定 (11.7 ~ 11.8 节)。
- 描述转换并解释显式向下转换的必要性 (11.9 节)。
- 探究 `Object` 类中的 `equals` 方法 (11.10 节)。
- 存储、提取和操作 `ArrayList` 中的对象 (11.11 节)。
- 用数组来构建一个 `ArrayList`，排序和打乱一个列表，以及得到列表中的最大和最小元素 (11.12 节)。
- 使用 `ArrayList` 来实现一个 `Stack` (11.13 节)。
- 使用可见性修饰符 `protected` 使父类中的数据和方法可以被子类访问 (11.14 节)。
- 使用修饰符 `final` 防止类的继承以及方法的覆盖 (11.14 节)。

11.1 引言

要点提示：面向对象的编程允许你从已经存在的类中定义新的类，这称为继承。

如本书之前所讨论的，面向过程的范式重点在于方法的设计，而面向对象的范式将数据和方法结合在对象中。面向对象范式的软件设计着重于对象以及对象上的操作。面向对象的方法结合了面向过程范式的强大之处，并且进一步将数据和操作集成在对象中。

继承是 Java 在软件重用方面一个重要且功能强大的特征。假设要定义一个类，对圆、矩形和三角形建模。这些类有很多共同的特性。设计这些类来避免冗余并使系统更易于理解和易于维护的最好方式是什么？答案就是使用继承。

11.2 父类和子类

要点提示：继承使得你可以定义一个通用的类（即父类），之后扩充该类为一个更加特定的类（即子类）。

使用类来对同一类型的对象建模。不同的类也可能会有一些共同的特征和行为，这些共同的特征和行为都统一放在一个类中，它是可以被其他类所共享的。你可以定义特定的类继承自通用类。这些特定的类继承通用类中的特征和方法。

考虑一下几何对象。假设要设计类建模像圆和矩形这样的几何对象。几何对象有许多共同的属性和行为。它们可以用某种颜色画出来的、填充的或者不填充的。这样，一个通用类 `GeometricObject` 可以用来建模所有的几何对象。这个类包括属性 `color` 和

filled, 以及适用于这些属性的 get 和 set 方法。假设该类还包括 dateCreated 属性以及 getDateCreated() 和 toString() 方法。toString() 方法返回代表该对象的字符串。由于圆是一个特殊类型的几何对象, 所以它和其他几何对象共享共同的属性和方法。因此, 通过继承自 GeometricObject 类来定义 Circle 类是合理的。同理, Rectangle 也可以定义为 GeometricObject 的子类。图 11-1 显示了这些类之间的关系, 指向父类的三角箭头用来表示相关的两个类之间的继承关系。

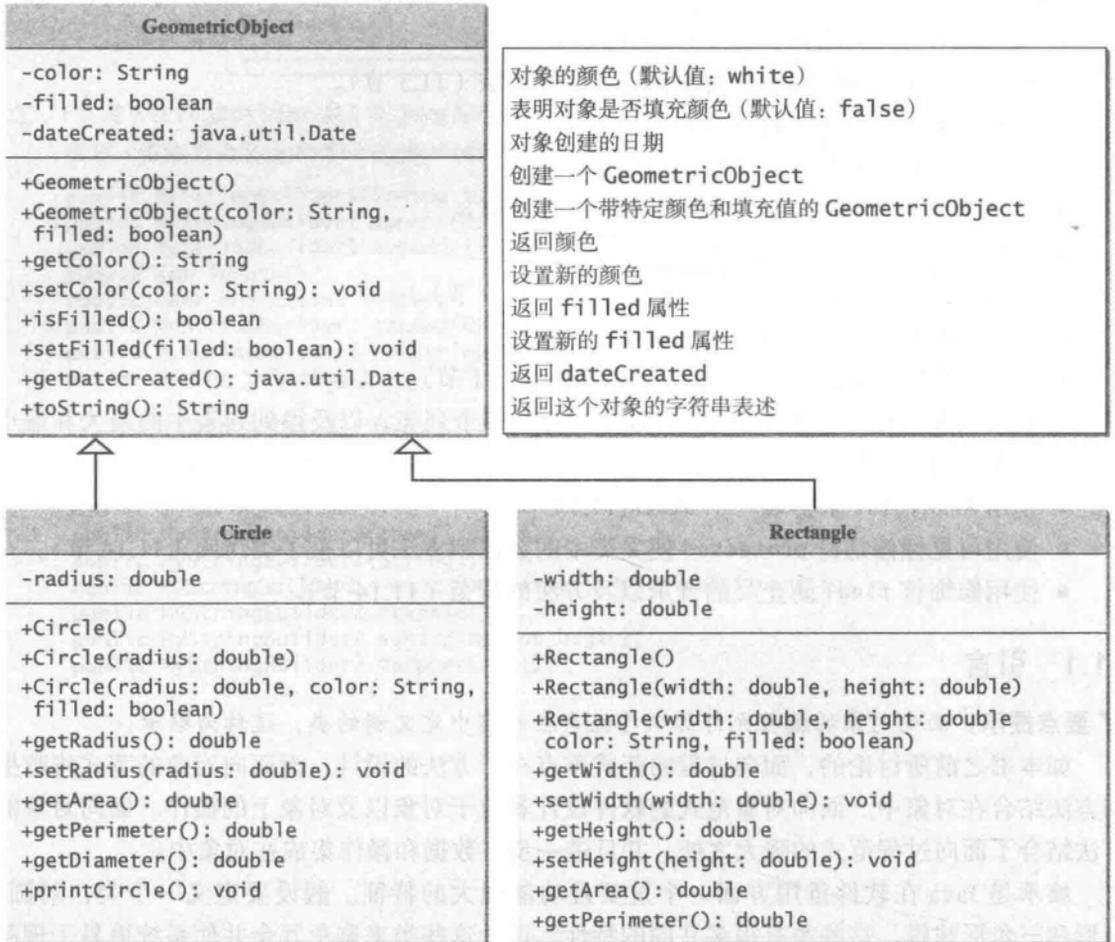


图 11-1 GeometricObject 类是 Circle 类和 Rectangle 类的父类

在 Java 术语中, 如果类 C1 扩展自另一个类 C2, 那么就将 C1 称为次类 (subclass), 将 C2 称为超类 (superclass)。超类也称为父类 (parent class) 或基类 (base class), 次类又称为子类 (child class)、扩展类 (extended class) 或派生类 (derived class)。子类从它的父类中继承可访问的数据域和方法, 还可以添加新数据域和新方法。

Circle 类继承了 GeometricObject 类所有可以访问的数据域和方法。除此之外, 它还有一个新的数据域 radius, 以及与 radius 相关的 get 和 set 方法。它还包括 getArea()、getPerimeter() 和 getDiameter() 方法以返回圆的面积、周长和直径。

Rectangle 类从 GeometricObject 类继承所有可访问的数据域和方法。此外, 它还有 width 和 height 数据域, 以及和它们相关的 get 和 set 方法。它还包括 getArea() 和

getPerimeter() 方法返回矩形的面积和周长。

GeometricObject 类、Circle 类和 Rectangle 类分别在程序清单 11-1、程序清单 11-2 和程序清单 11-3 中给出。

注意：为了避免与第 13 章介绍的改进版的 GeometricObject 类、Circle 类和 Rectangle 类发生命名冲突，这里将本章的类命名为 SimpleGeometricObject、CircleFromSimpleGeometricObject 和 RectangleFromSimpleGeometricObject。为简单起见，在文字描述上仍称它们为 GeometricObject、Circle 和 Rectangle 类。避免命名冲突最好的方法应该是将这些类放到不同的包中。但为了简单性和一致性，本书中所有的类都放在某个默认的包内。

程序清单 11-1 SimpleGeometricObject.java

```
1 public class SimpleGeometricObject {
2     private String color = "white";
3     private boolean filled;
4     private java.util.Date dateCreated;
5
6     /** Construct a default geometric object */
7     public SimpleGeometricObject() {
8         dateCreated = new java.util.Date();
9     }
10
11    /** Construct a geometric object with the specified color
12     * and filled value */
13    public SimpleGeometricObject(String color, boolean filled) {
14        dateCreated = new java.util.Date();
15        this.color = color;
16        this.filled = filled;
17    }
18
19    /** Return color */
20    public String getColor() {
21        return color;
22    }
23
24    /** Set a new color */
25    public void setColor(String color) {
26        this.color = color;
27    }
28
29    /** Return filled. Since filled is boolean,
30     its getter method is named isFilled */
31    public boolean isFilled() {
32        return filled;
33    }
34
35    /** Set a new filled */
36    public void setFilled(boolean filled) {
37        this.filled = filled;
38    }
39
40    /** Get dateCreated */
41    public java.util.Date getDateCreated() {
42        return dateCreated;
43    }
44
45    /** Return a string representation of this object */
46    public String toString() {
47        return "created on " + dateCreated + "\ncolor: " + color +
48            " and filled: " + filled;
49    }
50 }
```

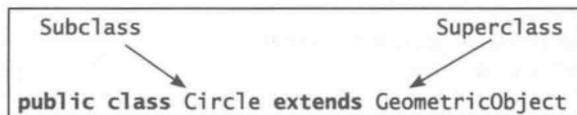
程序清单 11-2 CircleFromSimpleGeometricObject.java

```

1 public class CircleFromSimpleGeometricObject
2     extends SimpleGeometricObject
3     private double radius;
4
5 public CircleFromSimpleGeometricObject() {
6 }
7
8 public CircleFromSimpleGeometricObject(double radius) {
9     this.radius = radius;
10 }
11
12 public CircleFromSimpleGeometricObject(double radius,
13     String color, boolean filled) {
14     this.radius = radius;
15     setColor(color);
16     setFilled(filled);
17 }
18
19 /** Return radius */
20 public double getRadius() {
21     return radius;
22 }
23
24 /** Set a new radius */
25 public void setRadius(double radius) {
26     this.radius = radius;
27 }
28
29 /** Return area */
30 public double getArea() {
31     return radius * radius * Math.PI;
32 }
33
34 /** Return diameter */
35 public double getDiameter() {
36     return 2 * radius;
37 }
38
39 /** Return perimeter */
40 public double getPerimeter() {
41     return 2 * radius * Math.PI;
42 }
43
44 /** Print the circle info */
45 public void printCircle() {
46     System.out.println("The circle is created " + getDateCreated() +
47         " and the radius is " + radius);
48 }
49 }

```

Circle 类（程序清单 11-2）使用下面的语法扩展 GeometricObject 类（程序清单 11-1）：



关键字 extends（第 1 ~ 2 行）告诉编译器，Circle 类扩展自 GeometricObject 类，这样，它就继承了 getColor、setColor、isFilled、setFilled 和 toString 方法。

重载的构造方法 Circle(double radius,String color,boolean filled) 是通过调用

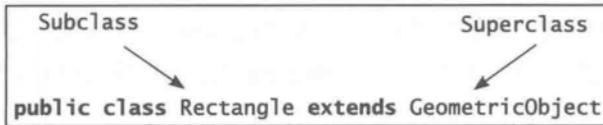
setColor 和 setFilled 方法设置 color 和 filled 属性来执行的 (第 12 ~ 17 行)。这两个公共方法是在基类 GeometricObject 中定义的, 并在 Circle 中继承, 因此可以在 Circle 类中使用它们。

你可能会尝试在构造方法中使用数据域 color 和 filled, 如下所示:

```
public CircleFromSimpleGeometricObject(
    double radius, String color, boolean filled) {
    this.radius = radius;
    this.color = color; // Illegal
    this.filled = filled; // Illegal
}
```

这是错误的, 因为 GeometricObject 类中的私有数据域 color 和 filled 是不能被除了 GeometricObject 类本身之外的其他任何类访问的。唯一读取和改变 color 与 filled 的方法就是通过它们的 get 和 set 方法。

Rectangle 类 (程序清单 11-3) 使用下面的语法继承 GeometricObject 类 (程序清单 11-1):



关键字 extends (第 1 ~ 2 行) 告诉编译器 Rectangle 类继承自 GeometricObject 类, 也就是继承了 getColor、setColor、isFilled、setFilled 和 toString 等方法。

程序清单 11-3 RectangleFromSimpleGeometricObject.java

```
1 public class RectangleFromSimpleGeometricObject
2     extends SimpleGeometricObject {
3     private double width;
4     private double height;
5
6     public RectangleFromSimpleGeometricObject() {
7     }
8
9     public RectangleFromSimpleGeometricObject(
10        double width, double height) {
11        this.width = width;
12        this.height = height;
13    }
14
15    public RectangleFromSimpleGeometricObject(
16        double width, double height, String color, boolean filled) {
17        this.width = width;
18        this.height = height;
19        setColor(color);
20        setFilled(filled);
21    }
22
23    /** Return width */
24    public double getWidth() {
25        return width;
26    }
27
28    /** Set a new width */
29    public void setWidth(double width) {
30        this.width = width;
31    }
32
```

```

33  /** Return height */
34  public double getHeight() {
35      return height;
36  }
37
38  /** Set a new height */
39  public void setHeight(double height) {
40      this.height = height;
41  }
42
43  /** Return area */
44  public double getArea() {
45      return width * height;
46  }
47
48  /** Return perimeter */
49  public double getPerimeter() {
50      return 2 * (width + height);
51  }
52 }

```

程序清单 11-4 中的代码创建了 Circle 和 Rectangle 的对象，并调用这些对象上的方法。toString() 方法继承自 GeometricObject 类，并且由 Circle 对象（第 5 行）和 Rectangle 对象（第 13 行）调用。

程序清单 11-4 TestCircleRectangle.java

```

1  public class TestCircleRectangle {
2      public static void main(String[] args) {
3          CircleFromSimpleGeometricObject circle =
4              new CircleFromSimpleGeometricObject(1);
5          System.out.println("A circle " + circle.toString());
6          System.out.println("The color is " + circle.getColor());
7          System.out.println("The radius is " + circle.getRadius());
8          System.out.println("The area is " + circle.getArea());
9          System.out.println("The diameter is " + circle.getDiameter());
10
11         RectangleFromSimpleGeometricObject rectangle =
12             new RectangleFromSimpleGeometricObject(2, 4);
13         System.out.println("\nA rectangle " + rectangle.toString());
14         System.out.println("The area is " + rectangle.getArea());
15         System.out.println("The perimeter is " +
16             rectangle.getPerimeter());
17     }
18 }

```

```

A circle created on Thu Feb 10 19:54:25 EST 2011
color: white and filled: false
The color is white
The radius is 1.0
The area is 3.141592653589793
The diameter is 2.0
A rectangle created on Thu Feb 10 19:54:25 EST 2011
color: white and filled: false
The area is 8.0
The perimeter is 12.0

```

下面是关于继承应该注意的几个关键点：

- 和传统的理解不同，子类并不是父类的一个子集。实际上，一个子类通常比它的父类包含更多的信息和方法。

- 父类中的私有数据域在该类之外是不可访问的。因此，不能在子类中直接使用。但是，如果父类中定义了公共的访问器 / 修改器，那么可以通过这些公共的访问器 / 修改器来访问和修改它们。
- 不是所有的“是一种”(is-a)关系都该用继承来建模。例如：正方形是一种矩形，但是不应该定义一个 Square 类来扩展 Rectangle 类，因为 width 和 height 属性并不适合于正方形。应该定义一个继承自 GeometricObject 类的 Square 类，并为正方形的边定义一个 side 属性。
- 继承是用来为“是一种”关系(is-a)建模的。不要仅仅为了重用方法这个原因而盲目地扩展一个类。例如：尽管 Person 类和 Tree 类可以共享类似高度和重量这样的通用特性，但是从 Person 类扩展出 Tree 类是毫无意义的。一个父类和它的子类之间必须存在“是一种”(is-a)关系。
- 某些程序设计语言是允许从几个类派生出一个子类的。这种能力称为多重继承(multiple inheritance)。但是在 Java 中是不允许多重继承的。一个 Java 类只可能直接继承自一个父类。这种限制称为单一继承(single inheritance)。如果使用 extends 关键字来定义一个子类，它只允许有一个父类。然而，多重继承是可以通过接口来实现的，这部分内容将在 13.4 节中介绍。

复习题

- 11.1 下面说法是真是假？一个子类是父类的子集。
- 11.2 使用什么关键字来定义一个子类？
- 11.3 什么是单一继承？什么是多重继承？Java 支持多重继承吗？

11.3 使用 super 关键字

 **要点提示：**关键字 super 指代父类，可以用于调用父类中的普通方法和构造方法。

子类继承它的父类中所有可访问的数据域和方法。它能继承构造方法吗？父类的构造方法能够从子类调用吗？本节就来解决这些问题以及衍生出来的问题。

9.14 节中介绍了关键字 this 的作用，它是对调用对象的引用。关键字 super 是指这个 super 关键字所在的类的父类。关键字 super 可以用于两种途径：

- 1) 调用父类的构造方法。
- 2) 调用父类的方法。

11.3.1 调用父类的构造方法

构造方法用于构建一个类的实例。不同于属性和普通方法，父类的构造方法不会被子类继承。它们只能使用关键字 super 从子类的构造方法中调用。

调用父类构造方法的语法是：

```
super() 或者 super(parameters);
```

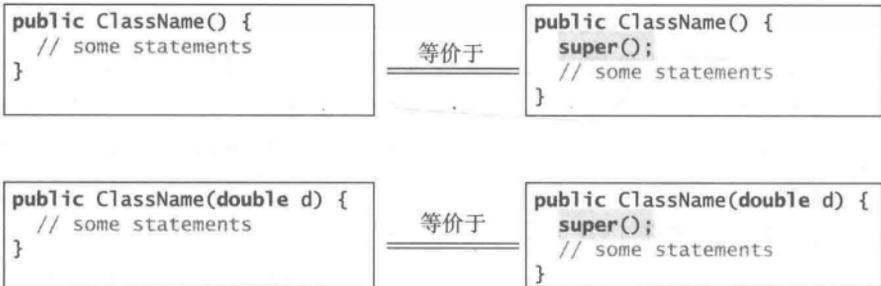
语句 super() 调用父类的无参构造方法，而语句 super(arguments) 调用与参数匹配的父类的构造方法。语句 super() 和 super(arguments) 必须出现在子类构造方法的第一行，这是显式调用父类构造方法的唯一方式。例如，在程序清单 11-2 中的第 12 ~ 17 行的构造方法可以用下面的代码替换：

```
public CircleFromSimpleGeometricObject(
    double radius, String color, boolean filled) {
    super(color, filled);
    this.radius = radius;
}
```

 **警告：**要调用父类构造方法就必须使用关键字 `super`，而且这个调用必须是构造方法的第一条语句。在子类中调用父类构造方法的名字会引起一个语法错误。

11.3.2 构造方法链

构造方法可以调用重载的构造方法或父类的构造方法。如果它们都没有被显式地调用，编译器就会自动地将 `super()` 作为构造方法的第一条语句。例如：



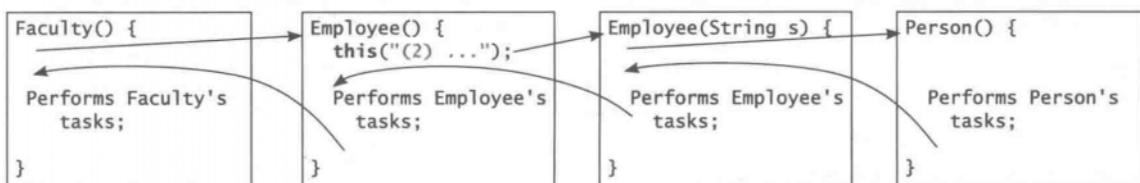
在任何情况下，构造一个类的实例时，将会调用沿着继承链的所有父类的构造方法。当构造一个子类的对象时，子类构造方法会在完成自己的任务之前，首先调用它的父类的构造方法。如果父类继承自其他类，那么父类构造方法又会在完成自己的任务之前，调用它自己的父类的构造方法。这个过程持续到沿着这个继承体系结构的最后一个构造方法被调用为止。这就是构造方法链（constructor chaining）。

思考下面的代码：

```
1 public class Faculty extends Employee {
2     public static void main(String[] args) {
3         new Faculty();
4     }
5
6     public Faculty() {
7         System.out.println("(4) Performs Faculty's tasks");
8     }
9 }
10
11 class Employee extends Person {
12     public Employee() {
13         this("(2) Invoke Employee's overloaded constructor");
14         System.out.println("(3) Performs Employee's tasks ");
15     }
16
17     public Employee(String s) {
18         System.out.println(s);
19     }
20 }
21
22 class Person {
23     public Person() {
24         System.out.println("(1) Performs Person's tasks");
25     }
26 }
```

- (1) Performs Person's tasks
- (2) Invoke Employee's overloaded constructor
- (3) Performs Employee's tasks
- (4) Performs Faculty's tasks

该程序会产生上面的输出。为什么呢？我们讨论一下这个原因。在第3行，`new Faculty()`调用 `Faculty` 的无参构造方法。由于 `Faculty` 是 `Employee` 的子类，所以，在 `Faculty` 构造方法中的所有语句执行之前，先调用 `Employee` 的无参构造方法。`Employee` 的无参构造方法调用 `Employee` 的第二个构造方法（第13行）。由于 `Employee` 是 `Person` 的子类，所以，在 `Employee` 的第二个构造方法中所有语句执行之前，先调用 `Person` 的无参构造方法。这个过程如下图所示：



警告：如果要设计一个可以被继承的类，最好提供一个无参构造方法以避免程序设计错误。思考下面的代码：

```

1 public class Apple extends Fruit {
2 }
3
4 class Fruit {
5     public Fruit(String name) {
6         System.out.println("Fruit's constructor is invoked");
7     }
8 }
  
```

由于在 `Apple` 中没有显式定义的构造方法，因此，`Apple` 的默认无参构造方法被隐式调用。因为 `Apple` 是 `Fruit` 的子类，所以 `Apple` 的默认构造方法会自动调用 `Fruit` 的无参构造方法。然而，`Fruit` 没有无参构造方法，因为 `Fruit` 显式地定义了构造方法。因此，程序不能被成功编译。

设计指南：一般情况下，最好能为每个类提供一个无参构造方法，以便于对该类进行扩展，同时避免错误。

11.3.3 调用父类的方法

关键字 `super` 不仅可以引用父类的构造方法，也可以引用父类的方法。所用语法如下：

```
super.方法名(参数);
```

可以如下改写 `Circle` 类中的 `printCircle()` 方法：

```

public void printCircle() {
    System.out.println("The circle is created " +
        super.getDateCreated() + " and the radius is " + radius);
}
  
```

在这种情况下，没有必要在 `getDateCreated()` 前放置 `super`，因为 `getDateCreated` 是 `GeometricObject` 类中的一个方法并被 `Circle` 类继承。然而，在某些情况下，如 11.4 节所

示，关键字 `super` 是必不可少的。

复习题

11.4 下面 a 中类 C 的运行结果输出什么？编译 b 中的程序的时候将出现什么问题？

```
class A {
    public A() {
        System.out.println(
            "A's no-arg constructor is invoked");
    }
}

class B extends A {
}

public class C {
    public static void main(String[] args) {
        B b = new B();
    }
}
```

a)

```
class A {
    public A(int x) {
    }
}

class B extends A {
    public B() {
    }
}

public class C {
    public static void main(String[] args) {
        B b = new B();
    }
}
```

b)

11.5 子类如何调用它的父类的构造方法？

11.6 下面的说法是真是假：当从子类调用构造方法时，它的父类的无参构造方法总是会被调用？

11.4 方法重写

要点提示：要重写一个方法，需要在子类中使用和父类一样的签名以及一样的返回值类型来对该方法进行定义。

子类从父类中继承方法。有时，子类需要修改父类中定义的方法的实现，这称作方法重写 (method overriding)。

`GeometricObject` 类中的 `toString` 方法 (程序清单 11-1 的第 46 ~ 49 行) 返回表示几何对象的字符串。这个方法可以被重写，返回表示圆的字符串。为了重写它，在程序清单 11-2 中加入下面的新方法：

```
1 public class CircleFromSimpleGeometricObject
2     extends SimpleGeometricObject {
3     // Other methods are omitted
4
5     // Override the toString method defined in the superclass
6     public String toString() {
7         return super.toString() + "\nradius is " + radius;
8     }
9 }
```

`toString()` 方法在 `GeometricObject` 类中定义，在 `Circle` 类中修改。在这两个类中定义的方法都可以在 `Circle` 类中使用。要在 `Circle` 类中调用定义在 `GeometricObject` 中的 `toString` 方法，使用 `super.toString()` (第 7 行)。

`Circle` 的子类能用语法规 `super.super.toString()` 访问定义在 `GeometricObject` 中的 `toString` 方法吗？答案是不能，这是一个语法错误。

以下几点值得注意：

- 仅当实例方法是可访问时，它才能被覆盖。因为私有方法在它的类本身以外是不能访问的，所以它不能被覆盖。如果子类中定义的方法在父类中是私有的，那么这两

个方法完全没有关系。

- 与实例方法一样，静态方法也能被继承。但是，静态方法不能被覆盖。如果父类中定义的静态方法在子类中被重新定义，那么在父类中定义的静态方法将被隐藏。可以使用语法：父类名.静态方法名 (SuperClassName.staticMethodName) 调用隐藏的静态方法。

复习题

- 11.7 下面说法是真是假：可以重写父类中定义的私有方法？
- 11.8 下面说法是真是假：可以重写父类中定义的静态方法？
- 11.9 如何从子类中显式的调用父类的构造方法？
- 11.10 如何从子类中调用一个被重写的父类的方法？

11.5 方法重写与重载

要点提示：重载意味着使用同样的名字但是不同的签名来定义多个方法。重写意味着在子类中提供一个对方法的新的实现。

在 6.8 节中已经学过关于方法重载的内容。方法重写是指该方法必须使用相同的签名和相同的返回值类型在子类中定义。

让我们用一个例子来显示重写和重载的不同。在图 a 中，类 A 中的方法 `p(double i)` 重写了在类 B 中定义的方法。但是，在图 b 中，类 A 中有两个重载的方法 `p(double i)` 和 `p(int i)`。方法 `p(double i)` 继承自类 B。

```
public class Test {
    public static void main(String[] args) {
        A a = new A();
        a.p(10);
        a.p(10.0);
    }
}

class B {
    public void p(double i) {
        System.out.println(i * 2);
    }
}

class A extends B {
    // This method overrides the method in B
    public void p(double i) {
        System.out.println(i);
    }
}
```

a)

```
public class Test {
    public static void main(String[] args) {
        A a = new A();
        a.p(10);
        a.p(10.0);
    }
}

class B {
    public void p(double i) {
        System.out.println(i * 2);
    }
}

class A extends B {
    // This method overloads the method in B
    public void p(int i) {
        System.out.println(i);
    }
}
```

b)

运行 a 中的 Test 类时，`a.p(10)` 和 `a.p(10.0)` 调用的都是定义在类 A 中的 `p(double i)` 方法，所以程序都显示 10.0。运行 b 中的 Test 类时，`a.p(10)` 调用类 A 中定义的 `p(int i)` 方法，显示输出为 10，而 `a.p(10.0)` 调用定义在类 B 中的 `p(double i)` 方法，显示输出为 20.0。

注意以下问题：

- 方法重写发生在通过继承而相关的不同类中；方法重载可以发生在同一个类中，也可以发生在由于继承而相关的不同类中。

- 方法重写具有同样的签名和返回值类型；方法重载具有同样的名字，但是不同的参数列表。

为了避免错误，可以使用一个特殊的 Java 语法，称为重写标注（override annotation），在子类的方法前面放一个 `@Override`。例如：

```

1 public class CircleFromSimpleGeometricObject
2     extends SimpleGeometricObject {
3     // Other methods are omitted
4
5     @Override
6     public String toString() {
7         return super.toString() + "\nradius is " + radius;
8     }
9 }
```

该标注表示被标注的方法必须重写父类的一个方法。如果具有该标注的方法没有重写父类的方法，编译器将报告一个错误。例如，如果 `toString` 被错误地输入为 `tostring`，将报告一个编译错误。如果没有使用重写标注，编译器不会报告错误。使用标注可以避免错误。

复习题

11.11 指出下面代码的错误：

```

1 public class Circle {
2     private double radius;
3
4     public Circle(double radius) {
5         radius = radius;
6     }
7
8     public double getRadius() {
9         return radius;
10    }
11
12    public double getArea() {
13        return radius * radius * Math.PI;
14    }
15 }
16
17 class B extends Circle {
18     private double length;
19
20     B(double radius, double length) {
21         Circle(radius);
22         length = length;
23     }
24
25     @Override
26     public double getArea() {
27         return getArea() * length;
28     }
29 }
```

- 11.12 解释方法重载和方法重写的不同之处。
- 11.13 如果子类中的方法具有和它父类中的方法完全相同的方法签名，且返回值类型也相同，那么这是方法重写还是方法重载呢？
- 11.14 如果子类中的一个方法具有和它父类中的方法完全相同的方法签名，但返回值类型不相同，这会存在问题吗？
- 11.15 如果子类中的一个方法具有和它父类中的方法相同的名字，但参数类型不同，那么这是方法重

写还是方法重载呢?

11.16 使用 @Override 标注的好处是什么?

11.6 Object 类及其 toString() 方法

要点提示: Java 中的所有类都继承自 java.lang.Object 类。

如果在定义一个类时没有指定继承性,那么这个类的父类就被默认为是 Object。例如,下面两个类的定义是一样的:

<pre>public class ClassName { ... }</pre>	等价于	<pre>public class ClassName extends Object { ... }</pre>
---	-----	--

诸如 String、StringBuilder、Loan 和 GeometricObject 这样的类都是 Object 的隐含子类(此前在本书中见到的所有主类也是如此)。熟悉 Object 类提供的方法是非常重要的,因为这样就可以在自己的类中使用它们。本节将介绍 Object 类中的 toString() 方法。

toString() 方法的签名是:

```
public String toString()
```

调用一个对象的 toString() 会返回一个描述该对象的字符串。默认情况下,它返回一个由该对象所属的类名、at 符号 (@) 以及该对象十六进制形式的内存地址组成的字符串。例如,考虑下面在程序清单 10-2 中定义 Loan 类的代码:

```
Loan loan = new Loan();  
System.out.println(loan.toString());
```

这些代码会显示像 Loan@15037e5 的字符串。这个信息不是很有用,或者说没有什么信息量。通常,应该重写这个 toString 方法,这样,它可以返回一个代表该对象的描述性字符串。例如, Object 类中的 toString 方法在 GeometricObject 类中重写,如程序清单 11-1 中第 46 ~ 49 行所示:

```
public String toString() {  
    return "created on " + dateCreated + "\ncolor: " + color +  
        " and filled: " + filled;  
}
```

注意: 也可以传递一个对象来调用 System.out.println(object) 或者 System.out.print(object)。这等价于调用 System.out.println(object.toString()) 或者 System.out.print(object.toString())。因此,可以使用 System.out.println(loan) 来替换 System.out.println(loan.toString())。

11.7 多态

要点提示: 多态意味着父类的变量可以指向子类对象。

面向对象程序设计的三大支柱是封装、继承和多态。我们已经学习了前两个,本节将介绍多态。

首先,定义两个有用的术语:子类型和父类型。一个类实际上定义了一种类型。子类定义的类型称为子类型(subtype),而父类定义的类型称为父类型(supertype)。因此,可以说 Circle 是 GeometricObject 的子类型,而 GeometricObject 是 Circle 的父类型。

继承关系使一个子类继承父类的特征，并且附加一些新特征。子类是它的父类的特殊化，每个子类的实例都是其父类的实例，但是反过来就不成立。例如：每个圆都是一个几何对象，但并非每个几何对象都是圆。因此，总可以将子类的实例传给需要父类型的参数。考虑程序清单 11-5 中的代码。

程序清单 11-5 PolymorphismDemo.java

```

1 public class PolymorphismDemo {
2     /** Main method */
3     public static void main(String[] args) {
4         // Display circle and rectangle properties
5         displayObject(new CircleFromSimpleGeometricObject
6             (1, "red", false));
7         displayObject(new RectangleFromSimpleGeometricObject
8             (1, 1, "black", true));
9     }
10
11     /** Display geometric object properties */
12     public static void displayObject(SimpleGeometricObject object) {
13         System.out.println("Created on " + object.getDateCreated() +
14             ". Color is " + object.getColor());
15     }
16 }

```

```

Created on Mon Mar 09 19:25:20 EDT 2011. Color is red
Created on Mon Mar 09 19:25:20 EDT 2011. Color is black

```

方法 `displayObject` (第 12 行) 具有 `GeometricObject` 类型的参数。可以通过传递任何一个 `GeometricObject` 的实例 (例如: 在第 5 ~ 8 行的 `new CircleFromSimpleGeometricObject(1, "red", false)` 和 `new RectangleFromSimpleGeometricObject(1, 1, "black", false)`) 来调用 `displayObject`。使用父类对象的地方都可以使用子类的对象。这就是通常所说的多态 (polymorphism, 它源于希腊文字, 意思是“多种形式”)。简单来说, 多态意味着父类型的变量可以引用子类型的对象。

11.8 动态绑定

 **要点提示:** 方法可以在沿着继承链的多个类中实现。JVM 决定运行时调用哪个方法。

方法可以在父类中定义而在子类中重写。例如: `toString()` 方法是在 `Object` 类中定义的, 而在 `GeometricObject` 类中重写。思考下面的代码:

```

Object o = new GeometricObject();
System.out.println(o.toString());

```

这里的 `o` 调用哪个 `toString()` 呢? 为了回答这个问题, 我们首先介绍两个术语: 声明类型和实际类型。一个变量必须被声明为某种类型。变量的这个类型称为它的声明类型 (declared type)。这里, `o` 的声明类型是 `Object`。一个引用类型变量可以是一个 `null` 值或者是一个对声明类型实例的引用。实例可以使用声明类型或它的子类型的构造方法创建。变量的实际类型 (actual type) 是被变量引用的对象的实际类。这里, `o` 的实际类型是 `GeometricObject`, 因为 `o` 指向使用 `new GeometricObject()` 创建的对象。`o` 调用哪个 `toString()` 方法由 `o` 的实际类型决定。这称为动态绑定 (dynamic binding)。

动态绑定工作机制如下: 假设对象 `o` 是类 `C1`, `C2`, ..., `Cn-1`, `Cn` 的实例, 其中 `C1` 是 `C2` 的子类, `C2` 是 `C3` 的子类, ..., `Cn-1` 是 `Cn` 的子类, 如图 11-2 所示。也就是说, `Cn` 是最通

用的类, C_1 是最特殊的类。在 Java 中, C_n 是 `Object` 类。如果对象 o 调用一个方法 p , 那么 JVM 会依次在类 $C_1, C_2, \dots, C_{n-1}, C_n$ 中查找方法 p 的实现, 直到找到为止。一旦找到一个实现, 就停止查找, 然后调用这个首先找到的实现。



图 11-2 被调用的方法是在运行时动态绑定的

程序清单 11-6 给出一个演示动态绑定的例子。

程序清单 11-6 DynamicBindingDemo.java

```

1 public class DynamicBindingDemo {
2     public static void main(String[] args) {
3         m(new GraduateStudent());
4         m(new Student());
5         m(new Person());
6         m(new Object());
7     }
8
9     public static void m(Object x) {
10        System.out.println(x.toString());
11    }
12 }
13
14 class GraduateStudent extends Student {
15 }
16
17 class Student extends Person {
18     @Override
19     public String toString() {
20         return "Student" ;
21     }
22 }
23
24 class Person extends Object {
25     @Override
26     public String toString() {
27         return "Person" ;
28     }
29 }

```

```

Student
Student
Person
java.lang.Object@130c19b

```

方法 m (第 9 行) 采用 `Object` 类型的参数。可以用任何对象 (例如: 在第 3 ~ 6 行的 `new GraduateStudent()`、`new Student()`、`new Person()` 和 `new Object()`) 作为参数来调用 m 方法。

当执行方法 `m(Object x)` 时, 调用参数 x 的 `toString` 方法。 x 可能会是 `GraduateStudent`、`Student`、`Person` 或者 `Object` 的实例。类 `GraduateStudent`、`Student`、`Person` 以及 `Object` 都有它们自己对 `toString` 方法的实现。使用哪个实现取决于运行时 x 的实际类型。调用 `m(new GraduateStudent())` (第 3 行) 会导致定义在 `Student` 类中的 `toString` 方法被调用。

调用 `m(new Student())` (第 4 行) 会调用在 `Student` 类中定义的 `toString` 方法。调用 `m(new Person())` (第 5 行) 会调用在 `Person` 类中定义的 `toString` 方法。调用 `m(new Object())` (第 6 行) 会调用在 `Object` 类中定义的 `toString` 方法。

匹配方法的签名和绑定方法的实现是两个不同的问题。引用变量的声明类型决定了编译时匹配哪个方法。在编译时, 编译器会根据参数类型、参数个数和参数顺序找到匹配的方法。一个方法可能在沿着继承链的多个类中实现。Java 虚拟机在运行时动态绑定方法的实现, 这是由变量的实际类型决定的。

复习题

- 11.17 什么是多态? 什么是动态绑定?
- 11.18 描述方法匹配和方法绑定之间的不同。
- 11.19 可以将以下实例赋值给 `Object[]` 类型的变量吗, `new int[50]`、`new Integer[50]`、`new String[50]` 或者 `new Object[50]`?
- 11.20 下面代码中哪里有错误?

```

1  public class Test {
2      public static void main(String[] args) {
3          Integer[] list1 = {12, 24, 55, 1};
4          Double[] list2 = {12.4, 24.0, 55.2, 1.0};
5          int[] list3 = {1, 2, 3};
6          printArray(list1);
7          printArray(list2);
8          printArray(list3);
9      }
10
11     public static void printArray(Object[] list) {
12         for (Object o: list)
13             System.out.print(o + " ");
14         System.out.println();
15     }
16 }

```

- 11.21 给出下面代码的输出。

```

public class Test {
    public static void main(String[] args) {
        new Person().printPerson();
        new Student().printPerson();
    }
}

class Student extends Person {
    @Override
    public String getInfo() {
        return "Student";
    }
}

class Person {
    public String getInfo() {
        return "Person";
    }

    public void printPerson() {
        System.out.println(getInfo());
    }
}

```

a)

```

public class Test {
    public static void main(String[] args) {
        new Person().printPerson();
        new Student().printPerson();
    }
}

class Student extends Person {
    private String getInfo() {
        return "Student";
    }
}

class Person {
    private String getInfo() {
        return "Person";
    }

    public void printPerson() {
        System.out.println(getInfo());
    }
}

```

b)

11.22 给出下面程序的输出。

```
1 public class Test {
2     public static void main(String[] args) {
3         A a = new A(3);
4     }
5 }
6
7 class A extends B {
8     public A(int t) {
9         System.out.println("A's constructor is invoked");
10    }
11 }
12
13 class B {
14     public B() {
15         System.out.println("B's constructor is invoked");
16     }
17 }
```

当调用 `new A(3)` 时, `Object` 的无参构造方法被调用了吗?

11.23 给出下面程序的输出:

```
public class Test {
    public static void main(String[] args) {
        new A();
        new B();
    }
}
class A {
    int i = 7;

    public A() {
        setI(20);
        System.out.println("i from A is " + i);
    }

    public void setI(int i) {
        this.i = 2 * i;
    }
}
class B extends A {
    public B() {
        System.out.println("i from B is " + i);
    }

    public void setI(int i) {
        this.i = 3 * i;
    }
}
```

11.9 对象转换和 `instanceof` 运算符

🔑 要点提示: 对象的引用可以类型转换为对另外一种对象的引用, 这称为对象转换。

在上一节中, 语句

```
m(new Student());
```

将对象 `new Student()` 赋值给一个 `Object` 类型的参数。这条语句等价于

```
Object o = new Student(); // Implicit casting
m(o);
```

由于 Student 的实例也是 Object 的实例，所以，语句 `Object o = new Student()` 是合法的，它称为隐式转换 (implicit casting)。

假设想使用下面的语句把对象引用 `o` 赋值给 Student 类型的变量：

```
Student b = o;
```

在这种情况下，将会发生编译错误。为什么语句 `Object o = new Student()` 可以运行，而语句 `Student b = o` 不行呢？原因是 Student 对象总是 Object 的实例，但是，Object 对象不一定是 Student 的实例。即使可以看到 `o` 实际上是一个 Student 的对象，但是编译器还没有聪明到知道这一点。为了告诉编译器 `o` 就是一个 Student 对象，就要使用显式转换 (explicit casting)。它的语法与基本类型转换的语法很类似，用圆括号把目标对象的类型括住，然后放到要转换的对象前面，如下所示：

```
Student b = (Student)o; // Explicit casting
```

总是可以将一个子类的实例转换为一个父类的变量，称为向上转换 (upcasting)，因为子类的实例永远是它的父类的实例。当把一个父类的实例转换为它的子类变量 (称为向下转换 (downcasting)) 时，必须使用转换记号“(子类名)”进行显式转换，向编译器表明你的意图。为使转换成功，必须确保要转换的对象是子类的一个实例。如果父类对象不是子类的一个实例，就会出现一个运行异常 `ClassCastException`。例如：如果一个对象不是 Student 的实例，它就不能转换成 Student 类型的变量。因此，一个好的经验是，在尝试转换之前确保该对象是另一个对象的实例。这是可以利用运算符 `instanceof` 来实现的。考虑下面的代码：

```
Object myObject = new Circle();
... // Some lines of code
/** Perform casting if myObject is an instance of Circle */
if (myObject instanceof Circle) {
    System.out.println("The circle diameter is " +
        ((Circle)myObject).getDiameter());
    ...
}
```

你可能会奇怪为什么必须进行类型转换。变量 `myObject` 被声明为 `Object`。声明类型决定了在编译时匹配哪个方法。使用 `myObject.getDiameter()` 会引起一个编译错误，因为 `Object` 类没有 `getDiameter` 方法。编译器无法找到和 `myObject.getDiameter()` 匹配的方法。所以，有必要将 `myObject` 转换成 `Circle` 类型，来告诉编译器 `myObject` 也是 `Circle` 的一个实例。

为什么没有在一开始就把 `myObject` 定义为 `Circle` 类型呢？为了能够进行通用程序设计，一个好的经验是把变量定义为父类型，这样，它就可以接收任何子类型的值。

🔧 注意：instanceof 是 Java 的关键字。在 Java 关键字中的每个字母都是小写的。

🔧 提示：为了更好地理解类型转换，可以认为它们类似于水果、苹果、橘子之间的关系，其中水果类 `Fruit` 是苹果类 `Apple` 和橘子类 `Orange` 的父类。苹果是水果，所以，总是可以将 `Apple` 的实例安全地赋值给 `Fruit` 变量。但是，水果不一定是苹果，所以，必须进行显式转换才能将 `Fruit` 的实例赋值给 `Apple` 的变量。

程序清单 11-7 演示了多态和类型转换。程序创建两个对象 (第 5 ~ 6 行)，一个圆和一个矩形，然后调用 `displayObject` 方法显示它们 (第 9 ~ 10 行)。如果对象是一个圆，`displayObject` 方法显示它的面积和周长 (第 15 行)；而如果对象是一个矩形，这个方法显

示它的面积 (第 21 ~ 22 行)。

程序清单 11-7 CastingDemo.java

```

1 public class CastingDemo {
2     /** Main method */
3     public static void main(String[] args) {
4         // Create and initialize two objects
5         Object object1 = new CircleFromSimpleGeometricObject(1);
6         Object object2 = new RectangleFromSimpleGeometricObject(1, 1);
7
8         // Display circle and rectangle
9         displayObject(object1);
10        displayObject(object2);
11    }
12
13    /** A method for displaying an object */
14    public static void displayObject(Object object) {
15        if (object instanceof CircleFromSimpleGeometricObject) {
16            System.out.println("The circle area is " +
17                ((CircleFromSimpleGeometricObject)object).getArea());
18            System.out.println("The circle diameter is " +
19                ((CircleFromSimpleGeometricObject)object).getDiameter());
20        }
21        else if (object instanceof
22            RectangleFromSimpleGeometricObject) {
23            System.out.println("The rectangle area is " +
24                ((RectangleFromSimpleGeometricObject)object).getArea());
25        }
26    }
27 }

```

```

The circle area is 3.141592653589793
The circle diameter is 2.0
The rectangle area is 1.0

```

`displayObject(Object object)` 方法是一个通用程序设计的例子。它可以通过传入 `Object` 的任何实例被调用。

程序使用隐式转换将一个 `Circle` 对象赋值给 `object1` 并且将一个 `Rectangle` 对象赋值给 `object2` (第 5 ~ 6 行), 然后调用 `displayObject` 方法显示这些对象的信息 (第 9 ~ 10 行)。

在 `displayObject` 方法中 (第 14 ~ 26 行), 如果对象是 `Circle` 的一个实例, 则用显式转换将这个对象转换为 `Circle` 对象, 并使用 `getArea` 和 `getDiameter` 方法显示这个圆的面积和直径。

只有源对象是目标类的实例时才能进行类型转换。在执行转换前, 程序使用 `instanceof` 运算符来确保源对象是否是目标类的实例 (第 15 行)。

由于 `getArea` 和 `getDiameter` 方法在 `Object` 类中是不可用的, 所以, 有必要显式地转换成 `Circle` 类型 (第 17、19 行) 和 `Rectangle` 类型 (第 24 行)。

警告: 对象成员访问运算符 (`.`) 优先于类型转换运算符。使用圆括号保证在点运算符 (`.`) 之前进行转换, 例如:

```
((Circle)object).getArea();
```

对基本类型值进行转换不同于对对象引用进行转换。转换基本类型值返回一个新的值。例如:

```
int age = 45;
byte newAge = (byte)age; // A new value is assigned to newAge
```

而转换一个对象引用不会创建一个新的对象，例如：

```
Object o = new Circle();
Circle c = (Circle)o; // No new object is created
```

现在，引用变量 `o` 和 `c` 指向同一个对象。

复习题

11.24 下面的说法是对还是错？

- 总可以成功地将子类的实例转换为父类。
- 总可以成功地将父类的实例转换为子类。

11.25 对于程序清单 11-1 和程序清单 11-2 中的 `GeometricObject` 类和 `Circle` 类，回答下面的问题：

a. 假设 `circle` 和 `object` 如下创建：

```
Circle circle = new Circle(1);
GeometricObject object = new GeometricObject();
```

下面的布尔表达式的值是 `true` 还是 `false`？

```
(circle instanceof GeometricObject)
(object instanceof GeometricObject)
(circle instanceof Circle)
(object instanceof Circle)
```

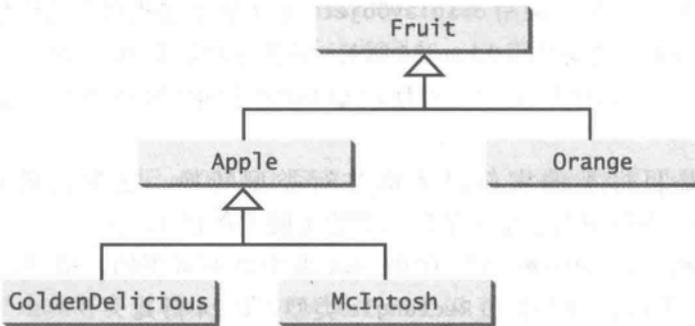
b. 下面的语句能够成功编译吗？

```
Circle circle = new Circle(5);
GeometricObject object = circle;
```

c. 下面的语句能够成功编译吗？

```
GeometricObject object = new GeometricObject();
Circle circle = (Circle)object;
```

11.26 假设 `Fruit`、`Apple`、`Orange`、`GoldenDelicious` 和 `McIntosh` 如下面的继承层次定义：



假设给出以下代码：

```
Fruit fruit = new GoldenDelicious();
Orange orange = new Orange();
```

回答下面的问题：

- `fruit instanceof Fruit` 的值为 `true` 吗？
- `fruit instanceof Orange` 的值为 `true` 吗？

- c. `fruit instanceof Apple` 的值为 `true` 吗?
- d. `fruit instanceof GoldenDelicious` 的值为 `true` 吗?
- e. `fruit instanceof Macintosh` 的值为 `true` 吗?
- f. `orange instanceof Orange` 的值为 `true` 吗?
- g. `orange instanceof Fruit` 的值为 `true` 吗?
- h. `orange instanceof Apple` 的值为 `true` 吗?
- i. 假设 `makeApple Cider` 方法定义在 `Apple` 类中。`fruit` 可以调用这个方法吗? `orange` 可以调用这个方法吗?
- j. 假设 `makeOrangeJuice` 方法定义在 `Orange` 类中。`orange` 可以调用这个方法吗? `fruit` 可以调用这个方法吗?
- k. 语句 `Orange p=new Apple()` 是否合法?
- l. 语句 `Macintosh p=new Apple()` 是否合法?
- m. 语句 `Apple p=new Macintosh()` 是否合法?

11.27 下面代码中的错误是什么?

```
1 public class Test {
2     public static void main(String[] args) {
3         Object fruit = new Fruit();
4         Object apple = (Apple)fruit;
5     }
6 }
7
8 class Apple extends Fruit {
9 }
10
11 class Fruit {
12 }
```

11.10 Object 类的 equals 方法

🔑 要点提示: 如同 `toString()` 方法, `equals(Object)` 方法是定义在 `Object` 类中的另外一个有用的方法。

在 `Object` 类中定义的另外一个经常使用的方法是 `equals` 方法。它的签名是:

```
public boolean equals(Object o)
```

这个方法测试两个对象是否相等。调用它的语法是:

```
object1.equals(object2);
```

`Object` 类中 `equals` 方法的默认实现是:

```
public boolean equals(Object obj) {
    return (this == obj);
}
```

这个实现使用 `==` 运算符检测两个引用变量是否指向同一个对象。因此,应该在自己的客户类中重写这个方法,以测试两个不同的对象是否具有相同的内容。

`equals` 方法在 Java API 的许多类中被重写,比如 `java.lang.String` 和 `java.util.Date`,用于比较两个对象的内容是否相等。在 4.4.7 节中已经用过 `equals` 方法比较两个字符串。`String` 类中的 `equals` 方法继承自 `Object` 类,然后在 `String` 类中被重写,使之能够检

验两个字符串的内容是否相等。

可以重写 Circle 类中的 equals 方法，基于圆的半径比较两个圆是否相等，如下所示：

```
public boolean equals(Object o) {
    if (o instanceof Circle)
        return radius == ((Circle)o).radius;
    else
        return this == o;
}
```

🔪 **注意：**比较运算符 == 用来比较两个基本数据类型的值是否相等，或者判断两个对象是否具有相同的引用。如果想让 equals 方法能够判断两个对象是否具有相同的内容，可以在定义这些对象的类时，重写 Circle 类中的 equals 方法。运算符 == 要比 equals 方法的功能强大些，因为 == 运算符可以检测两个引用变量是否指向同一个对象。

🔪 **警告：**在子类中，使用签名 equals(SomeClassName obj)（例如：equals(Circle c)）重写 equals 方法是一个常见错误，应该使用 equals(Object obj)。参见复习题 11.29。

🔪 复习题

- 11.28 每个对象都有 toString 方法和 equals 方法吗？它们从何而来？如何使用？重写这些方法合适吗？
- 11.29 当覆盖 equals 方法时，常见的错误就是在子类中输错它的签名。例如：equals 方法被错误地写成 equals(Circle circle)，如下面 a 中的代码所示；应该使用如 b 中所示的 equals(Object circle) 替换它。分别给出运行 a 和 b 中的 Test 类和 Circle 类的输出。

```
public class Test {
    public static void main(String[] args) {
        Object circle1 = new Circle();
        Object circle2 = new Circle();
        System.out.println(circle1.equals(circle2));
    }
}
```

```
class Circle {
    double radius;

    public boolean equals(Circle circle) {
        return this.radius == circle.radius;
    }
}
```

a)

```
class Circle {
    double radius;

    public boolean equals(Object circle) {
        return this.radius ==
            ((Circle)circle).radius;
    }
}
```

b)

如果 Test 类中的 Object 换成 Circle，那么分别使用 a 和 b 中的 Circle 类来运行 Test 将输出什么？

11.11 ArrayList 类

🔪 **要点提示：**ArrayList 对象可以用于存储一个对象列表。

现在，我们介绍一个很有用的用于存储对象的类了。可以创建一个数组存储对象，但是这个数组一旦创建，它的大小就固定了。Java 提供 ArrayList 类来存储不限定个数的对象。图 11-3 给出了 ArrayList 中的一些方法。

ArrayList 是一种泛型类，具有一个泛型类型 E。创建一个 ArrayList 时，可以指定一个具体的类型来替换 E。例如，下面语句创建一个 ArrayList，并且将其引用赋值给变量

cities。该 ArrayList 对象可以用于存储字符串。

java.util.ArrayList<E>	
+ArrayList()	创建一个空的列表
+add(o: E): void	增加一个新元素 o 到该列表的末尾
+add(index: int, o: E): void	增加一个新元素 o 到该列表的指定下标处
+clear(): void	清除列表中的所有元素
+contains(o: Object): boolean	如果该列表包含元素 o, 则返回 true
+get(index: int): E	返回该列表指定下标位置的元素
+indexOf(o: Object): int	返回列表中第一个匹配元素的下标
+isEmpty(): boolean	如果该列表不包含任何元素, 则返回 true
+lastIndexOf(o: Object): int	返回列表中匹配的最后一个元素的下标
+remove(o: Object): boolean	去除列表中的第一个元素。如果该元素被去除, 则返回 true
+size(): int	返回列表中的元素个数
+remove(index: int): E	去除指定下标位置的元素。如果该元素被去除, 则返回 true
+set(index: int, o: E): E	设置指定下标位置的元素

图 11-3 ArrayList 中存储不限定个数的对象

```
ArrayList<String> cities = new ArrayList<String>();
```

下面语句创建一个 ArrayList 并且将其引用赋值给变量 dates。该 ArrayList 对象可以用于存储日期。

```
ArrayList<java.util.Date> dates = new ArrayList<java.util.Date> ();
```

注意：从 JDK1.7 开始，语句

```
ArrayList<AConcreteType> list = new ArrayList<AConcreteType>();
```

可以简化为

```
ArrayList<AConcreteType> list = new ArrayList<>();
```

由于使用了称为类型推导的特征，构造方法中不再要求给出具体类型。编译器可以从变量的声明中推导出类型。关于泛型的更多讨论，包括如何定义自定义的泛型类和方法，将在第 19 章中做介绍。

程序清单 11-8 给出了使用 ArrayList 来存储对象的一个示例。

程序清单 11-8 TestArrayList.java

```
1 import java.util.ArrayList;
2
3 public class TestArrayList {
4     public static void main(String[] args) {
5         // Create a list to store cities
6         ArrayList<String> cityList = new ArrayList<>();
7
8         // Add some cities in the list
9         cityList.add("London");
10        // cityList now contains [London]
11        cityList.add("Denver");
12        // cityList now contains [London, Denver]
```

```

13     cityList.add("Paris");
14     // cityList now contains [London, Denver, Paris]
15     cityList.add("Miami");
16     // cityList now contains [London, Denver, Paris, Miami]
17     cityList.add("Seoul");
18     // Contains [London, Denver, Paris, Miami, Seoul]
19     cityList.add("Tokyo");
20     // Contains [London, Denver, Paris, Miami, Seoul, Tokyo]
21
22     System.out.println("List size? " + cityList.size());
23     System.out.println("Is Miami in the list? " +
24         cityList.contains("Miami"));
25     System.out.println("The location of Denver in the list? "
26         + cityList.indexOf("Denver"));
27     System.out.println("Is the list empty? " +
28         cityList.isEmpty()); // Print false
29
30     // Insert a new city at index 2
31     cityList.add(2, "Xian");
32     // Contains [London, Denver, Xian, Paris, Miami, Seoul, Tokyo]
33
34     // Remove a city from the list
35     cityList.remove("Miami");
36     // Contains [London, Denver, Xian, Paris, Seoul, Tokyo]
37
38     // Remove a city at index 1
39     cityList.remove(1);
40     // Contains [London, Xian, Paris, Seoul, Tokyo]
41
42     // Display the contents in the list
43     System.out.println(cityList.toString());
44
45     // Display the contents in the list in reverse order
46     for (int i = cityList.size() - 1; i >= 0; i--)
47         System.out.print(cityList.get(i) + " ");
48     System.out.println();
49
50     // Create a list to store two circles
51     ArrayList<CircleFromSimpleGeometricObject> list
52         = new ArrayList<>();
53
54     // Add two circles
55     list.add(new CircleFromSimpleGeometricObject(2));
56     list.add(new CircleFromSimpleGeometricObject(3));
57
58     // Display the area of the first circle in the list
59     System.out.println("The area of the circle? " +
60         list.get(0).getArea());
61 }
62 }

```

```

List size? 6
Is Miami in the list? true
The location of Denver in the list? 1
Is the list empty? false
[London, Xian, Paris, Seoul, Tokyo]
Tokyo Seoul Paris Xian London
The area of the circle? 12.566370614359172

```

由于 ArrayList 位于 java.util 包中，所以在第一行导入该包。程序使用无参构造方法创建一个存储字符串的 ArrayList，将引用赋值给 cityList（第 6 行）。add 方法（第 9 ~ 19 行）将字符串增加到数组列表末尾。因此，在执行完 cityList.add("London")（第 9 行）之

后，这个数组列表包含

```
[London]
```

执行完 `cityList.add("New Denver")` (第 11 行) 后，这个数组列表包含

```
[London, Denver]
```

在加入 Paris、Miami、Seoul 和 Tokyo 之后 (第 13 ~ 19 行)，这个数组列表包含

```
[London, Denver, Paris, Miami, Seoul, Tokyo]
```

调用 `size()` (第 22 行) 返回这个数组列表的大小，数组列表的当前大小为 6。调用 `contains("Miami")` (第 24 行) 检测这个对象是否在这个数组列表中。在这种情况下，它返回 `true`，因为 Miami 在这个数组列表中。调用 `indexOf("Denver")` (第 26 行) 返回该对象在数组列表中的索引值，这里它的值为 1。如果对象不在这个数组列表中，它返回 -1。`isEmpty()` 方法 (第 28 行) 检测这个数组列表是否为空。因为当前列表不为空，所以它返回 `false`。

语句 `cityList.add(2,"Xian")` (第 31 行) 在这个数组列表的指定下标位置插入一个对象。该语句执行完之后，数组列表变成

```
[London, Denver, Xian, Paris, Miami, Seoul, Tokyo]
```

语句 `cityList.remove("Miami")` (第 35 行) 从数组列表中删除该对象。该语句执行后，数组列表就变成

```
[London, Denver, Xian, Paris, Seoul, Tokyo]
```

语句 `cityList.remove(1)` 语句 (第 39 行) 从数组列表中删除指定下标位置的元素，该语句执行后，数组列表变成

```
[London, Xian, Paris, Seoul, Tokyo]
```

第 43 行的语句相当于

```
System.out.println(cityList);
```

方法 `toString()` 返回数组列表的字符串表示，其形式为 `[e0.toString(), e1.toString(), ..., ek.toString()]`，这里的 `e0`, `e1`, ..., `ek` 都是数组列表中的元素。

方法 `get(index)` (第 47 行) 返回指定下标位置处的对象。

可以像使用数组一样使用 `ArrayList` 对象，但是两者还是有很多不同之处。表 11-1 列出了它们的异同点。

表 11-1 数组和 ArrayList 之间的异同

操作	数组	ArrayList
创建数组 / 数组列表	<code>String[] a = new String [10]</code>	<code>ArrayList list< String > = new . ArrayList()</code>
引用元素	<code>a [index]</code>	<code>list.get(index)</code>
更新元素	<code>a [index] = "London";</code>	<code>list.set(index, "London");</code>
返回大小	<code>a.length</code>	<code>list.size()</code>
添加一个新元素		<code>list.add("London")</code>
插入一个新元素		<code>list.add(index, "London")</code>

(续)

操作	数组	ArrayList
删除一个元素		<code>list.remove(index)</code>
删除一个元素		<code>list.remove(Object)</code>
删除所有元素		<code>list.clear()</code>

一旦创建了一个数组，它的大小就确定下来了。可以使用方括号访问数组元素（例如：`a[index]`）。当创建 `ArrayList` 后，它的大小为 0。如果元素不在数组列表中，就不能使用 `get(index)` 和 `set(index, element)` 方法。向数组列表中添加、插入和删除元素是比较容易的，而向数组中添加、插入和删除元素是比较复杂的。为了实现这些操作，必须编写代码操纵这个数组。注意，可以使用 `java.util.Arrays.sort(array)` 方法来对一个数组排序。如果要对一个数组列表排序，使用 `java.util.Collections.sort(arrayList)` 方法。

假设想创建一个用于存储整数的 `ArrayList`，可以使用下面代码来创建一个列表吗？

```
ArrayList<int> list = new ArrayList<>();
```

答案是不行。这样行不通，因为存储在 `ArrayList` 中的元素必须是一种对象。不能使用诸如 `int` 的基本数据类型来代替一个泛型类型。然而，你可以创建一个存储 `Integer` 对象的 `ArrayList`，如下所示：

```
ArrayList<Integer> list = new ArrayList<>();
```

程序清单 11-9 给出了一个程序，提示用户输入一个数字序列，然后显示该序列中的不同数字。假设输入 0 表示结束输入，并且 0 不计入序列中的数字。

程序清单 11-9 DistinctNumbers.java

```

1 import java.util.ArrayList;
2 import java.util.Scanner;
3
4 public class DistinctNumbers {
5     public static void main(String[] args) {
6         ArrayList<Integer> list = new ArrayList<>();
7
8         Scanner input = new Scanner(System.in);
9         System.out.print("Enter integers (input ends with 0): ");
10        int value;
11
12        do {
13            value = input.nextInt(); // Read a value from the input
14
15            if (!list.contains(value) && value != 0)
16                list.add(value); // Add the value if it is not in the list
17        } while (value != 0);
18
19        // Display the distinct numbers
20        for (int i = 0; i < list.size(); i++)
21            System.out.print(list.get(i) + " ");
22    }
23 }
```

```

Enter numbers (input ends with 0): 1 2 3 2 1 6 3 4 5 4 5 1 2 3 0
The distinct numbers are: 1 2 3 6 4 5
```

程序创建了一个存储 `Integer` 对象的 `ArrayList` (第 6 行)，然后在循环中重复读入值 (第

12~17行)。对于每个值，如果不在列表中（第15行），则将其添加到列表中（第16行）。可以重写该程序，使用数组替代 ArrayList 来存储元素。然而，使用 ArrayList 来实现该程序将更简单，有以下两个原因。

- ArrayList 的大小是灵活的，所以无须提前给定它的大小。而当创建一个数组时，它的大小必须给定。
- ArrayList 包含许多有用的方法。比如，可以使用 contains 方法来测试某个元素是否在列表中。如果使用数组，则需要编写额外代码来实现该方法。

可以在数组里使用 foreach 循环来遍历元素。数组列表中的元素也可以使用 foreach 循环来进行遍历，语法如下：

```
for (elementType element: arrayList) {
    // Process the element
}
```

例如，可以使用下面代码来替代第20-21行的代码：

```
for (int number: list)
    System.out.print(number + " ");
```

☛ 复习题

11.30 如何实现以下功能？

- 创建一个存储双精度值的 ArrayList。
- 向数组列表中追加一个对象。
- 在数组列表的开始位置插入一个对象。
- 找出数组列表中所包含对象的个数。
- 从数组列表中删除给定对象。
- 从数组列表中删除最后一个对象。
- 检测一个给定的对象是否在数组列表中。
- 从数组列表中获取指定下标位置的元素。

11.31 请找出下面代码中的错误：

```
ArrayList<String> list = new ArrayList<>();
list.add("Denver");
list.add("Austin");
list.add(new java.util.Date());
String city = list.get(0);
list.set(3, "Dallas");
System.out.println(list.get(3));
```

11.32 假定 ArrayList list 包含 {"Dallas", "Dallas", "Houston", "Dallas"}。调用 list.remove("Dallas") 一次之后的列表是什么？下面语句可以从列表中删除所有具有值 "Dallas" 的元素么？如果不能，修改代码。

```
for (int i = 0; i < list.size(); i++)
    list.remove("Dallas");
```

11.33 解释为什么下面代码显示 [1, 3]，而不是 [2, 3]。

```
ArrayList<Integer> list = new ArrayList<>();
list.add(1);
list.add(2);
list.add(3);
list.remove(1);
System.out.println(list);
```

11.34 解释为什么下面代码是错误的。

```
ArrayList<Double> list = new ArrayList<>();
list.add(1);
```

11.12 对于列表有用的方法

 **要点提示:** Java 提供了方法，用于从数组创建列表、对列表排序、找到列表中的最大和最小元素，以及打乱一个列表。

我们经常需要从一个对象数组中创建一个数组列表，或者相反。可以使用循环来实现，但是更容易的方法是使用 Java API 中的方法。这里是一个从数组中创建一个数组列表的例子：

```
String[] array = {"red", "green", "blue"};
ArrayList<String> list = new ArrayList<>(Arrays.asList(array));
```

Arrays 类中的静态方法 `asList` 返回一个列表，该列表传递给 `ArrayList` 的构造方法用于创建一个 `ArrayList`。反过来，可以使用下面代码从一个数组列表来创建一个对象数组。

```
String[] array1 = new String[list.size()];
list.toArray(array1);
```

调用 `list.toArray(array1)` 将 `list` 中的内容复制到 `array1` 中。

如果列表中的元素是可比较的，比如整数、双精度浮点数或者字符串，则可以使用 `java.util.Collections` 类中的静态的 `sort` 方法来对元素进行排序。这里是一些例子：

```
Integer[] array = {3, 5, 95, 4, 15, 34, 3, 6, 5};
ArrayList<Integer> list = new ArrayList<>(Arrays.asList(array));
java.util.Collections.sort(list);
System.out.println(list);
```

可以使用 `java.util.Collections` 类中的静态的 `max` 和 `min` 方法来返回列表中的最大和最小元素。这里是一些例子：

```
Integer[] array = {3, 5, 95, 4, 15, 34, 3, 6, 5};
ArrayList<Integer> list = new ArrayList<>(Arrays.asList(array));
System.out.println(java.util.Collections.max(list));
System.out.println(java.util.Collections.min(list));
```

可以使用 `java.util.Collections` 类中的静态的 `shuffle` 方法来随机打乱列表的元素。这里是一些例子：

```
Integer[] array = {3, 5, 95, 4, 15, 34, 3, 6, 5};
ArrayList<Integer> list = new ArrayList<>(Arrays.asList(array));
java.util.Collections.shuffle(list);
System.out.println(list);
```

复习题

11.35 改正下面语句中的错误：

```
int[] array = {3, 5, 95, 4, 15, 34, 3, 6, 5};
ArrayList<Integer> list = new ArrayList<>(Arrays.asList(array));
```

11.36 改正下面语句中的错误：

```
int[] array = {3, 5, 95, 4, 15, 34, 3, 6, 5};
System.out.println(java.util.Collections.max(array));
```

11.13 示例学习：自定义栈类

 **要点提示:** 本节设计一个栈类，用于存放对象。

10.6 节给出了一个存储 `int` 值的栈类。本节介绍一个存储对象的栈类。可以使用一个 `ArrayList` 来实现 `Stack`，如程序清单 11-10 所示。该类的 UML 图显示在图 11-4 中。

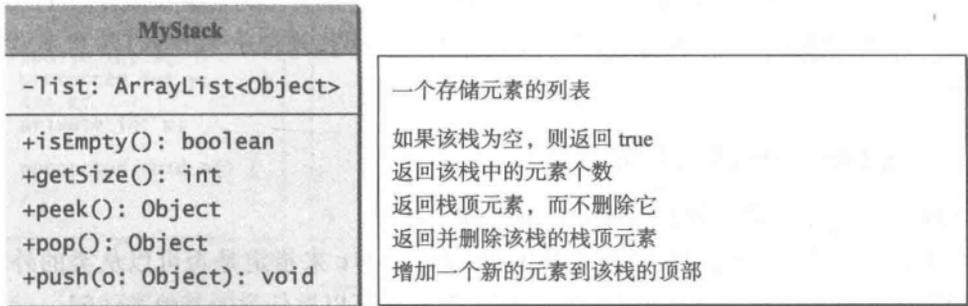


图 11-4 MyStack 类封装了栈的存储，提供了操作栈的操作

程序清单 11-10 MyStack.java

```

1  import java.util.ArrayList;
2
3  public class MyStack {
4      private ArrayList<Object> list = new ArrayList<>();
5
6      public boolean isEmpty() {
7          return list.isEmpty();
8      }
9
10     public int getSize() {
11         return list.size();
12     }
13
14     public Object peek() {
15         return list.get(getSize() - 1);
16     }
17
18     public Object pop() {
19         Object o = list.get(getSize() - 1);
20         list.remove(getSize() - 1);
21         return o;
22     }
23
24     public void push(Object o) {
25         list.add(o);
26     }
27
28     @Override
29     public String toString() {
30         return "stack: " + list.toString();
31     }
32 }

```

创建一个数组列表用于存储栈中的元素（第 4 行）。`isEmpty()` 方法（第 6 ~ 8 行）返回 `list.isEmpty()`。`getSize()` 方法（第 10 ~ 12 行）返回 `list.size()`。`peek()` 方法（第 14 ~ 16 行）可以获取栈顶元素而不删除它，线性表末尾的元素作为栈顶的元素。`pop()` 方法（第 18 ~ 22 行）删除栈顶元素并返回该元素。`push(Object element)` 方法（第 24 ~ 26 行）将指定元素添加到这个栈中。通过调用 `list.toString()` 重写 `Object` 类中定义的 `toString()` 方法（第 28 ~ 31 行）显示这个栈中的内容。`ArrayList` 中实现的 `toString()` 返回表示一个数组线性表中所有元素的字符串表示。

设计指南 在程序清单 11-10 中, MyStack 中包含 ArrayList。MyStack 和 ArrayList 之间的关系为组合。因为继承是对“是一种”(is-a)关系建模,而组合是对“是一部分”(has-a)关系建模。也可以将 MyStack 实现为 ArrayList 的一个子类(参见编程练习题 11.10)。使用组合关系更好些,因为它可以定义一个全新的类,而无须继承 ArrayList 中不必要和不恰当的方法。

11.14 protected 数据和方法

要点提示: 一个类中的受保护成员可以从子类中访问。

迄今为止,我们已经使用过关键字 `private` 和 `public` 来指定是否可以从类的外部访问数据域和方法。私有成员只能在类内访问,而公共成员可以被任意的其他类访问。

经常需要允许子类访问定义在父类中的数据域或方法,但不允许非子类访问这些数据域和方法。可以使用关键字 `protected` 完成该功能。父类中被保护的数据域或方法可以在它的子类中访问。

修饰符 `private`、`protected` 和 `public` 都称为可见性修饰符(visibility modifier)或可访问性修饰符(accessibility modifier),因为它们指定如何访问类和类的成员。这些修饰符的可见性按下面的顺序递增:

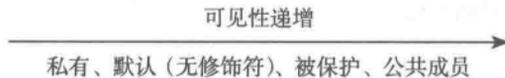


表 11-2 总结了类中成员的可访问性。图 11-5 描述了 C1 类中的 `public`、`protected`、默认的和 `private` 数据或方法是如何被 C2、C3、C4 和 C5 类访问的,其中, C2 类与 C1 类在同一个包中、C3 类是 C1 类在同一个包中的子类、C4 类是 C1 类在不同包中的子类、C5 类与 C1 类在不同包中。

表 11-2 数据和方法的可见性

类中成员的修饰符	在同一类内可访问	在同一包内可访问	在子类内可访问	在不同包可访问
<code>public</code>	√	√	√	√
<code>protected</code>	√	√	√	—
(default)	√	√	—	—
<code>private</code>	√	—	—	—

使用 `private` 修饰符可以完全隐藏类的成员,这样,就不能从类外直接访问它们。不使用修饰符就表示允许同一个包里的任何类直接访问类的成员,但是其他包中的类不可以访问。使用 `protected` 修饰符允许任何包中的子类或同一包中的类访问类的成员。使用 `public` 修饰符允许任意类访问类的成员。

类可以以两种方式使用:一种是用于创建该类的实例;另一种是通过扩展该类创建它的子类。如果不想从类的外部使用类的成员,就把成员声明成 `private`。如果想让该类的用户都能使用类的成员,就把成员声明成 `public`。如果想让该类的扩展者使用数据和方法,而不想让该类的用户使用,则把成员声明成 `protected`。

修饰符 `private` 和 `protected` 只能用于类的成员。`public` 修饰符和默认修饰符(也就是没有修饰符)既可以用于类的成员,也可以用于类。一个没有修饰符的类(即非公共类)是

不能被其他包中的类访问的。

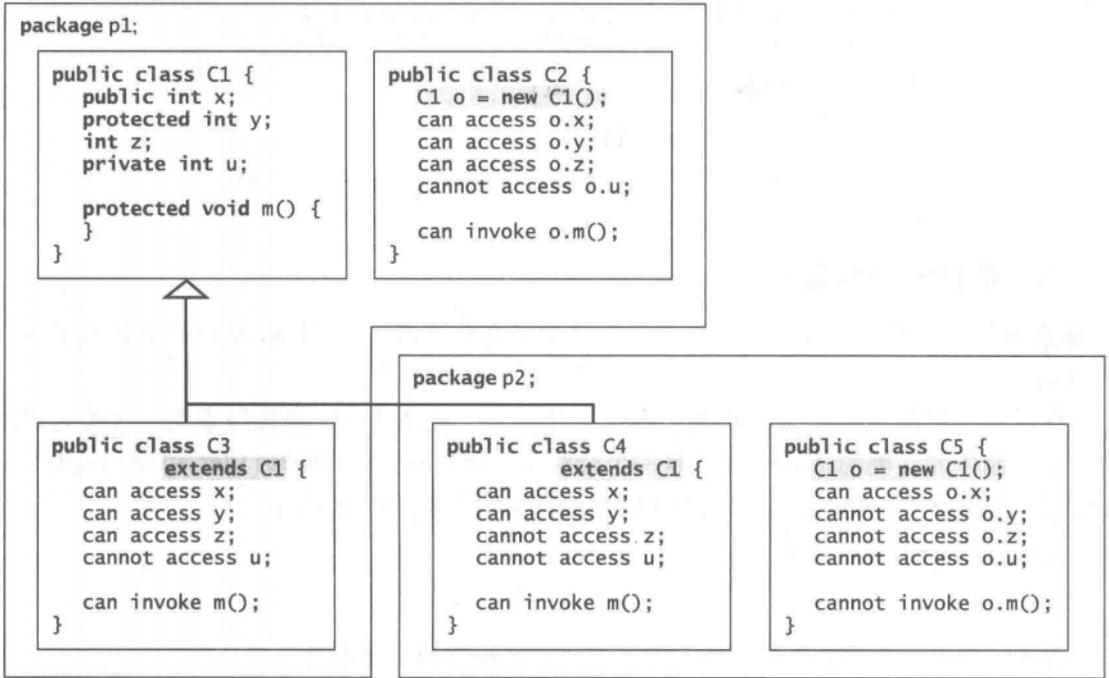


图 11-5 使用可见性修饰符控制如何访问数据和方法

注意：子类可以重写它的父类的 `protected` 方法，并把它的可见性改为 `public`。但是，子类不能削弱父类中定义的方法的可访问性。例如：如果一个方法在父类中定义为 `public`，在子类中也必须定义为 `public`。

复习题

- 11.37 应该在类上使用什么修饰符才能使同一个包中的类可以访问它，而不同包中的类不能访问它？
- 11.38 应该用什么修饰符才能使不同包中的类不能访问这个类，而任何包中的子类都可以访问它？
- 11.39 在下面的代码中，类 A 和类 B 在同一个包中。如果 a 中的问号被空白代替，那么类 B 能编译吗？如果问号被 `private` 代替，那么类 B 能编译吗？如果问号被 `protected` 代替，类 B 能编译吗？

```

package p1;

public class A {
    ? int i;

    ? void m() {
        ...
    }
}

```

a)

```

package p1;

public class B extends A {
    public void m1(String[] args) {
        System.out.println(i);
        m();
    }
}

```

b)

- 11.40 在下面的代码中，类 A 和类 B 在不同的包中。如果 a 中的问号被空白代替，那么类 B 能编译吗？如果问号被 `private` 代替，那么类 B 能编译吗？如果问号被 `protected` 代替，那么类 B 能编译吗？

```
package p1;

public class A {
    ? int i;

    ? void m() {
        ...
    }
}
```

a)

```
package p2;

public class B extends A {
    public void m1(String[] args) {
        System.out.println(i);
        m();
    }
}
```

b)

11.15 防止扩展和重写

要点提示：一个被 `final` 修饰的类和方法都不能被扩展。被 `final` 修饰的数据域是一个常数。

有时候，可能希望防止类扩展。在这种情况下，使用 `final` 修饰符表明一个类是最终的，是不能作为父类的。`Math` 类就是一个最终类。`String`、`StringBuilder` 和 `StringBuffer` 类也可以是最终类。例如，下面的类 `A` 就是最终类，是不能被继承的：

```
public final class A {
    // Data fields, constructors, and methods omitted
}
```

也可以定义一个方法为最终的，最终方法不能被它的子类重写。

例如，下面的方法是最终的，是不能重写的：

```
public class Test {
    // Data fields, constructors, and methods omitted

    public final void m() {
        // Do something
    }
}
```

注意：修饰符 `public`、`protected`、`private`、`static`、`abstract` 以及 `final` 可以用在类和类的成员（数据和方法）上，只有 `final` 修饰符还可以用在方法中的局部变量上。方法内的最终局部变量就是常量。

复习题

11.41 如何防止一个类被扩展？如何防止一个方法被重写？

11.42 指出下面语句是对还是错：

- 被保护的数据或方法可以被同一包中的任何类访问。
- 被保护的数据或方法可以被不同包中的任何类访问。
- 被保护的数据或方法可以被任意包中它的子类访问。
- 最终类可以有实例。
- 最终类可以被扩展。
- 最终方法可以被重写。

关键术语

actual type (实际类型)

casting object (转换对象)

constructor chaining (构造方法链)

declared type (声明类型)

dynamic binding (动态绑定)	protected (受保护修饰符)
inheritance (继承)	single inheritance (单一继承)
instanceof (运算符, 是……类型的实例)	subclass (子类)
is-a relationship (是关系)	subtype (子类型)
method overriding (方法重写)	superclass (父类)
multiple inheritance (多重继承)	supertype (父类型)
override (重写)	type inference (类型推导)
polymorphism (多态)	

本章小结

1. 可以从现有的类定义新的类, 这称为类的继承。新类称为次类、子类或继承类; 现有的类称为超类、父类或基类。
2. 构造方法用来构造类的实例。不同于属性和方法, 子类不继承父类的构造方法。它们只能用关键字 `super` 从子类的构造方法中调用。
3. 构造方法可以调用重载的构造方法或它的父类的构造方法。这种调用必须是构造方法的第一条语句。如果没有显式地调用它们中的任何一个, 编译器就会把 `super()` 作为构造方法的第一条语句, 它调用的是父类的无参构造方法。
4. 为了重写一个方法, 必须使用与它的父类中的方法相同的签名来定义子类中的方法。
5. 实例方法只有在可访问时才能重写。这样, 私有方法是不能重写的, 因为它是不能在类本身之外访问的。如果子类中定义的方法在父类中是私有的, 那么这两个方法是完全没有关系的。
6. 静态方法与实例方法一样可以继承。但是, 静态方法不能重写, 如果父类中定义的静态方法在子类中重新定义, 那么父类中定义的方法被隐藏。
7. Java 中的每个类都继承自 `java.lang.Object` 类。如果一个类在定义时没有指定继承关系, 那么它的父类就是 `Object`。
8. 如果一个方法的参数类型是父类 (例如: `Object`), 可以向该方法的参数传递任何子类 (例如: `Circle` 类或 `String` 类) 的对象。这称为多态。
9. 因为子类的实例总是它的父类的实例, 所以, 总是可以将一个子类的实例转换成一个父类的变量。当把父类实例转换成它的子类变量时, 必须使用转换记号 (子类名) 进行显式转换, 向编译器表明你的意图。
10. 一个类定义一个类型。子类定义的类型称为子类型, 而父类定义的类型称为父类型。
11. 当从引用变量调用实例方法时, 该变量的实际类型在运行时决定使用该方法的哪个实现。这称为动态绑定。
12. 可以使用表达式 `obj instanceof AClass` (对象名 instanceof 类名) 测试一个对象是否是一个类的实例。
13. 可以使用 `ArrayList` 类来创建一个对象, 用于存储一个对象列表。
14. 可以使用 `protected` 修饰符来防止方法和数据被不同包的子类访问。
15. 可以用 `final` 修饰符来表明一个类是最终类, 是不能被继承的; 也可以表明一个方法是最终的, 是不能重写的。

测试题

回答位于 www.cs.armstrong.edu/liang/intro10e/quiz.html 中本章的测试题。

编程练习题

11.2 ~ 11.4 节

11.1 (三角形类 Triangle) 设计一个名为 Triangle 的类来扩展 GeometricObject 类。该类包括:

- 三个名为 side1、side2 和 side3 的 double 数据域表示这个三角形的三条边, 它们的默认值是 1.0。
- 一个无参构造方法创建默认的三角形。
- 一个能创建带指定 side1、side2 和 side3 的三角形的构造方法。
- 所有三个数据域的访问器方法。
- 一个名为 getArea() 的方法返回这个三角形的面积。
- 一个名为 getPerimeter() 的方法返回这个三角形的周长。
- 一个名为 toString() 的方法返回这个三角形的字符串描述。

计算三角形面积的公式参见编程练习题 2.19。toString() 方法的实现如下所示:

```
return "Triangle: side1 = " + side1 + " side2 = " + side2 +
    " side3 = " + side3;
```

画出 Triangle 类和 GeometricObject 类的 UML 图, 并实现这些类。编写一个测试程序, 提示用户输入三角形的三条边、颜色以及一个 Boolean 值表明该三角形是否填充。程序应该使用输入创建一个具有这些边并设置 color 和 filled 属性的三角形。程序应该显示面积、边长、颜色以及表明是否填充的真或者假的值。

11.5 ~ 11.14 节

11.2 (Person、Student、Employee、Faculty 和 Staff 类) 设计一个名为 Person 的类和它的两个名为 Student 和 Employee 的子类。Employee 类又有子类: 教员类 Faculty 和职员类 Staff。每个人都有姓名、地址、电话号码和电子邮件地址。学生有班级状态(大一、大二、大三或大四)。将这些状态定义为常量。一个雇员涉及办公室、工资和受聘日期。使用编程练习题 10.14 中定义的 MyDate 类为受聘日期创建一个对象。教员有办公时间和级别。职员有职务称号。覆盖每个类中的 toString 方法, 显示相应的类别名字和人名。

画出这些类的 UML 图并实现这些类。编写一个测试程序, 创建 Person、Student、Employee、Faculty 和 Staff, 并且调用它们的 toString() 方法。

11.3 (账户类 Account 的子类) 在编程练习题 9.7 中定义了一个 Account 类来建模一个银行账户。一个账户有账号、余额、年利率、开户日期等属性, 以及存款和取款等方法。创建两个检测支票账户 (checking account) 和储蓄账户 (saving account) 的子类。支票账户有一个透支限定额, 但储蓄账户不能透支。

画出这些类的 UML 图并实现这些类。编写一个测试程序, 创建 Account、SavingsAccount 和 CheckingAccount 的对象, 然后调用它们的 toString() 方法。

11.4 (ArrayList 的最大元素) 编写以下方法, 返回一个整数 ArrayList 的最大值。如果列表为 null 或者列表的大小为 0, 则返回 null 值。

```
public static Integer max(ArrayList<Integer> list)
```

编写一个测试程序, 提示用户输入一个以 0 结尾的数值序列, 调用该方法返回输入的最大数值。

11.5 (课程类 Course) 重写程序清单 10-6 中的 Course 类, 使用 ArrayList 代替数组来存储学生。为该类绘制新的 UML 图。不应该改变 Course 类的原始合约 (即, 构造方法和方法的定义都不应该改变, 但私有的成员可以改变)。

11.6 (使用 ArrayList) 编写程序, 创建一个 ArrayList, 然后向这个列表中添加一个 Loan 对象、

一个 Date 对象、一个字符串和一个 Circle 对象，然后使用循环调用对象的 toString() 方法，来显示列表中所有的元素。

11.7 (打乱 ArrayList) 编写以下方法，打乱一个整数 ArrayList 中的元素。

```
public static void shuffle(ArrayList<Integer> list)
```

**11.8 (新的 Account 类) 编程练习题 9.7 中给出了一个 Account 类，如下设计一个新的 Account 类：

- 添加一个 String 类型的新数据域 name 来存储客户的名字。
- 添加一个新的构造方法，该方法创建一个具有指定名字、id 和收支额的账户。
- 添加一个名为 transactions 的 ArrayList 类型的新数据域，用于为账户存储交易。每笔交易都是一个 Transaction 类的实例。Transaction 类的定义如图 11-6 所示。
- 修改 withdraw 和 deposit 方法，向 transactions 数组线性表添加一笔交易。
- 其他所有属性和方法都和编程练习题 9.7 中的一样。

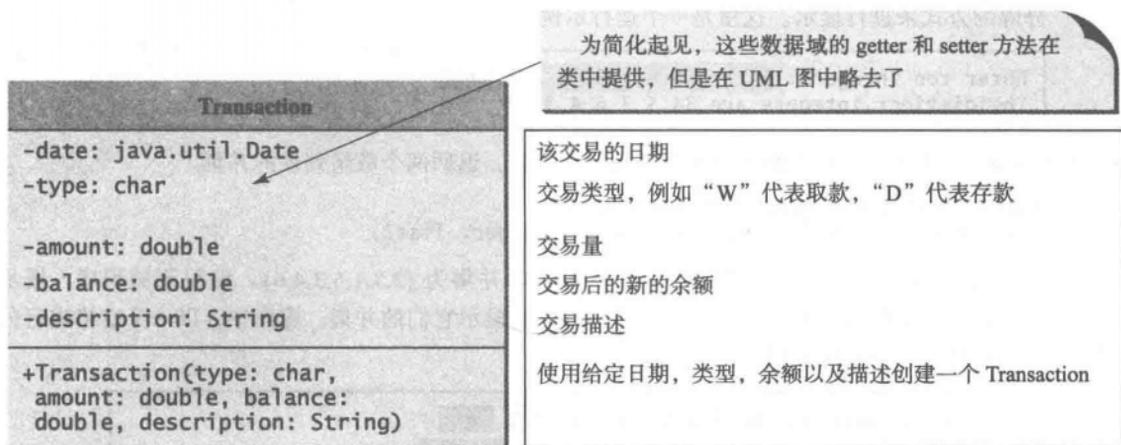


图 11-6 Transaction 类描述银行账户的一笔交易

编写一个测试程序，创建一个年利率为 1.5%、收支额为 1000、id 为 1122 而名字为 George 的 Account。向该账户存入 30 美元、40 美元和 50 美元并从该账户中取出 5 美元、4 美元和 2 美元。打印账户清单，显示账户持有者名字、利率、收支额和所有的交易。

*11.9 (最大的行和列) 编写程序，随机将 0 和 1 填入一个 $n \times n$ 的矩阵，打印该矩阵，并且找出具有最多 1 的行和列。

提示：使用两个 ArrayList 来存储具有最多 1 的行和列的下标。

这里是程序的一个运行示例：

```
Enter the array size n: 4
The random array is
0011
0011
1101
1010
The largest row index: 2
The largest column index: 2, 3
```

11.10 (利用继承实现 MyStack) 在程序清单 11-10 中，MyStack 是用组合实现的。扩展 ArrayList 创建一个新的栈类。

画出这些类的 UML 图并实现 MyStack 类。编写一个测试程序，提示用户输入五个字符串，然后以逆序显示这些字符串。

11.11 (对 ArrayList 排序) 编写以下方法, 对一个数值的 ArrayList 进行排序:

```
public static void sort(ArrayList<Integer> list)
```

编写测试程序, 提示用户输入 5 个数字, 将其存储在一个数组列表中, 并且以升序进行显示。

11.12 (对 ArrayList 求和) 编写以下方法, 返回 ArrayList 中所有数字的和:

```
public static double sum(ArrayList<Double> list)
```

编写测试程序, 提示用户输入 5 个数字, 将其存储在一个数组列表中, 并且显示它们的和。

*11.13 (去掉重复元素) 使用下面的方法头编写方法, 从一个整数的数组列表中去掉重复元素:

```
public static void removeDuplicate(ArrayList<Integer> list)
```

编写测试程序, 提示用户输入 10 个整数到列表中, 显示其中不同的整数, 并以一个空格分隔的方式来进行显示。这里是一个运行示例:

```
Enter ten integers: 34 5 3 5 6 4 33 2 2 4 --Enter
The distinct integers are 34 5 3 6 4 33 2
```

11.14 (结合两个列表) 使用下面的方法头编写一个方法, 返回两个数组列表的并集。

```
public static ArrayList<Integer> union(
    ArrayList<Integer> list1, ArrayList<Integer> list2)
```

例如, 两个数组列表 {2,3,1,5} 和 {3,4,6} 的并集为 {2,3,1,5,3,4,6}。编写测试程序, 提示用户输入两个列表, 每个列表有 5 个整数, 然后显示它们的并集。输出中, 以一个空格进行分隔。这里是一个运行示例:

```
Enter five integers for list1: 3 5 45 4 3 --Enter
Enter five integers for list2: 33 51 5 4 13 --Enter
The combined list is 3 5 45 4 3 33 51 5 4 13
```

*11.15 (凸多边形面积) 如果一个多边形中连接任意两个顶点的线段都包含在多边形中, 则称为凸多边形。编写一个程序, 提示用户输入一个凸多边形中的顶点数, 并顺时针输入点, 然后程序显示多边形的面积信息。这里是一个程序的运行示例:

```
Enter the number of the points: 7 --Enter
Enter the coordinates of the points:
-12 0 -8.5 10 0 11.4 5.5 7.8 6 -5.5 0 -7 -3.5 -13.5 --Enter
The total area is 292.575
```

**11.16 (加法测试) 重写程序清单 5-1, 如果用户重复输入了相同的答案, 则给出用户警告。

 提示: 使用一个数组列表来存储答案。

这里是一个运行示例:

```
What is 5 + 9? 12 --Enter
Wrong answer. Try again. What is 5 + 9? 34 --Enter
Wrong answer. Try again. What is 5 + 9? 12 --Enter
You already entered 12
Wrong answer. Try again. What is 5 + 9? 14 --Enter
You got it!
```

**11.17 (代数: 完全平方) 编写一个程序, 提示用户输入一个整数 m , 然后找到最小的整数 n , 使得 $m*n$ 是一个完全平方。

提示：存储所有 m 的最小因子到一个数组列表，则 n 是列表中出现奇数次的因子的乘积。例如，考虑 $m=90$ 的情况，保存因子 2,3,3,5 到一个数组列表中。列表中 2 和 5 出现了奇数次数，因此， n 是 10。)

这里是一个运行示例：

```
Enter an integer m: 1500 
The smallest number n for m * n to be a perfect square is 15
m * n is 22500
```

```
Enter an integer m: 63 
The smallest number n for m * n to be a perfect square is 7
m * n is 441
```

异常处理和文本 I/O

教学目标

- 了解异常和异常处理的概况 (12.2 节)。
- 探究使用异常处理的优点 (12.2 节)。
- 区别异常的类型: Error(致命的) 和 Exception(非致命的) 以及必检和免检异常 (12.3 节)。
- 在方法头中声明异常 (12.4.1 节)。
- 在方法中抛出异常 (12.4.2 节)。
- 编写 try-catch 块处理异常 (12.4.3 节)。
- 解释异常是如何传播的 (12.4.3 节)。
- 从异常对象中获得信息 (12.4.4 节)。
- 开发具有异常处理的应用 (12.4.5 节)。
- 在 try-catch 块中使用 finally 子句 (12.5 节)。
- 只为非预期错误使用异常 (12.6 节)。
- 在 catch 块中重新抛出异常 (12.7 节)。
- 创建链式异常 (12.8 节)。
- 定义自定义的异常类 (12.9 节)。
- 使用 File 类获取文件/目录的属性, 删除和重命名文件/目录, 以及创建目录 (12.10 节)。
- 使用 PrintWriter 类向文件写数据 (12.11.1 节)。
- 使用 try-with-resources 来保证资源自动关闭了。(12.11.2 节)。
- 使用 Scanner 类从文件读取数据 (12.11.3 节)。
- 理解如何使用 Scanner 来读取数据 (12.11.4 节)。
- 开发一个替换文件中文本的程序 (12.11.5 节)。
- 从 Web 读取数据 (12.12 节)。
- 开发一个 Web 抓取程序 (12.13 节)。

12.1 引言

 **要点提示:** 异常处理使得程序可以处理非预期的情景, 并且继续正常的处理。

在程序运行过程中, 如果 JVM 检测出一个不可能执行的操作, 就会出现运行时错误 (runtime error)。例如, 如果使用一个越界的下标访问数组, 程序就会产生一个 `ArrayIndexOutOfBoundsException` 的运行时错误。如果程序需要输入一个整数的时候用户输入了一个 `double` 值, 会得到一个 `InputMismatchException` 的运行时错误。

在 Java 中, 运行时错误会作为异常抛出。异常就是一种对象, 表示阻止正常进行程序执行的错误或者情况。如果异常没有被处理, 那么程序将会非正常终止。该如何处理这个异常, 以使程序可以继续运行或者优雅终止呢? 本章介绍该主题以及文本的输入和输出。

12.2 异常处理概述

🔑 **要点提示：**异常是从方法抛出的。方法的调用者可以捕获以及处理该异常。

为了演示异常处理，包括异常是如何创建以及如何抛出的，我们从一个读取两个整数并显示它们的商的例子（程序清单 12-1）开始。

程序清单 12-1 Quotient.java

```
1 import java.util.Scanner;
2
3 public class Quotient {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6
7         // Prompt the user to enter two integers
8         System.out.print("Enter two integers: ");
9         int number1 = input.nextInt();
10        int number2 = input.nextInt();
11
12        System.out.println(number1 + " / " + number2 + " is " +
13            (number1 / number2));
14    }
15 }
```

```
Enter two integers: 5 2 
5 / 2 is 2
```

```
Enter two integers: 3 0 
Exception in thread "main" java.lang.ArithmeticException: / by zero
at Quotient.main(Quotient.java:11)
```

如果为第二个数字输入的是 0，那就会产生一个运行时错误，因为不能用一个整数除以 0（注意，一个浮点数除以 0 是不会产生异常的）。解决这个错误的一个简单方法就是添加一个 if 语句来测试第二个数字，如程序清单 12-2 所示。

程序清单 12-2 QuotientWithIf.java

```
1 import java.util.Scanner;
2
3 public class QuotientWithIf {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6
7         // Prompt the user to enter two integers
8         System.out.print("Enter two integers: ");
9         int number1 = input.nextInt();
10        int number2 = input.nextInt();
11
12        if (number2 != 0)
13            System.out.println(number1 + " / " + number2
14                + " is " + (number1 / number2));
15        else
16            System.out.println("Divisor cannot be zero");
17    }
18 }
```

```
Enter two integers: 5 0 
Divisor cannot be zero
```

为了介绍异常处理，我们重写程序清单 12-2 来使用一个方法计算商，如程序清单 12-3 所示：

程序清单 12-3 QuotientWithMethod.java

```

1  import java.util.Scanner;
2
3  public class QuotientWithMethod {
4      public static int quotient(int number1, int number2) {
5          if (number2 == 0) {
6              System.out.println("Divisor cannot be zero");
7              System.exit(1);
8          }
9
10         return number1 / number2;
11     }
12
13     public static void main(String[] args) {
14         Scanner input = new Scanner(System.in);
15
16         // Prompt the user to enter two integers
17         System.out.print("Enter two integers: ");
18         int number1 = input.nextInt();
19         int number2 = input.nextInt();
20
21         int result = quotient(number1, number2);
22         System.out.println(number1 + " / " + number2 + " is "
23             + result);
24     }
25 }

```

Enter two integers: 5 3
5 / 3 is 1

Enter two integers: 5 0
Divisor cannot be zero

方法 `quotient` (第 4 ~ 11 行) 返回两个整数的商。如果 `number2` 为 0，则不能返回一个值，因此程序在第 7 行终止。这显然是一个问题。不应该让方法来终止程序——应该由调用者决定是否终止程序。

方法如何通知它的调用者一个异常产生了呢？Java 可以让一个方法可以抛出一个异常，该异常可以被调用者捕获和处理。程序清单 12-3 可以如程序清单 12-4 重写。

程序清单 12-4 QuotientWithException.java

```

1  import java.util.Scanner;
2
3  public class QuotientWithException {
4      public static int quotient(int number1, int number2) {
5          if (number2 == 0)
6              throw new ArithmeticException("Divisor cannot be zero");
7
8          return number1 / number2;
9      }
10
11     public static void main(String[] args) {
12         Scanner input = new Scanner(System.in);
13
14         // Prompt the user to enter two integers
15         System.out.print("Enter two integers: ");

```

```

16     int number1 = input.nextInt();
17     int number2 = input.nextInt();
18
19     try {
20         int result = quotient(number1, number2);
21         System.out.println(number1 + " / " + number2 + " is "
22             + result);
23     }
24     catch (ArithmeticException ex) {
25         System.out.println("Exception: an integer " +
26             "cannot be divided by zero ");
27     }
28
29     System.out.println("Execution continues ...");
30 }
31 }

```

```

Enter two integers: 5 3 
5 / 3 is 1
Execution continues ...

```

```

Enter two integers: 5 0 
Exception: an integer cannot be divided by zero
Execution continues ...

```

如果 number2 为 0，方法通过执行下面语句抛出一个异常（第 6 行）：

```
throw new ArithmeticException("Divisor cannot be zero");
```

在这种情况下，抛出的值为 `new ArithmeticException("Divisor cannot be zero")`，称为一个异常（exception）。`throw` 语句的执行称为抛出一个异常（throwing an exception）。异常就是一个从异常类创建的对象。在这种情况下，异常类就是 `java.lang.ArithmeticException`。构造方法 `ArithmeticException(str)` 被调用以构建一个异常，其中 `str` 是描述异常的消息。

当异常被抛出时，正常的执行流程就被中断。就像它的名字所提示的，“抛出异常”就是将异常从一个地方传递到另一个地方。调用方法的语句包含在一个 `try` 块和一个 `catch` 块中。`try` 块（第 19 ~ 23 行）包含了正常情况下执行的代码。异常被 `catch` 块所捕获。`catch` 块中的代码执行以处理异常。之后，`catch` 块之后的语句（第 29 行）被执行。

`throw` 语句类似于方法的调用，但不同于调用方法的是，它调用的是 `catch` 块。从某种意义上讲，`catch` 块就像带参数的方法定义，这些参数匹配抛出的值的类型。但是，它不像方法，在执行完 `catch` 块之后，程序控制不返回到 `throw` 语句；而是执行 `catch` 块后的下一条语句。

`catch` 块的头部

```
catch (ArithmeticException ex)
```

标识符 `ex` 的作用很像是方法中的参数。所以，这个参数称为 `catch` 块的参数。`ex` 之前的类型（例如，`ArithmeticException`）指定了 `catch` 块可以捕获的异常类型。一旦捕获该异常，就能从 `catch` 块体中的参数访问这个抛出的值。

总之，一个 `try-throw-catch` 块的模板可能会如下所示：

```

try {
    Code to run;
    A statement or a method that may throw an exception;
}

```

```

    More code to run;
}
catch (type ex) {
    Code to process the exception;
}

```

一个异常可能是通过 try 块中的 throw 语句直接抛出，或者调用一个可能会抛出异常的方法而抛出。

main 方法调用 quotient(第 20 行)。如果求商方法正常执行，它会返回一个值给调用者。如果 quotient 方法遇到一个异常，它会抛出一个异常给它的调用者。这个调用者的 catch 块处理该异常。

现在，你看到了使用异常处理的优点。它能使方法抛出一个异常给它的调用者，并由调用者处理该异常。如果没有这个能力，那么被调用的方法就必须自己处理异常或者终止该程序。被调用的方法通常不知道在出错的情况下该做些什么，这是库方法的一般情况。库方法可以检测出错误，但是只有调用者才知道出现错误时需要做些什么。异常处理最根本的优势就是将检测错误（由被调用的方法完成）从处理错误（由调用方法完成）中分离出来。

很多库方法都会抛出异常。程序清单 12-5 给出一个读入一个输入时处理 InputMismatchException 的例子。

程序清单 12-5 InputMismatchExceptionDemo.java

```

1  import java.util.*;
2
3  public class InputMismatchExceptionDemo {
4      public static void main(String[] args) {
5          Scanner input = new Scanner(System.in);
6          boolean continueInput = true;
7
8          do {
9              try {
10                 System.out.print("Enter an integer: ");
11                 int number = input.nextInt();
12                 // If an InputMismatchException occurs
13                 // Display the result
14                 System.out.println(
15                     "The number entered is " + number);
16
17                 continueInput = false;
18             }
19             catch (InputMismatchException ex) {
20                 System.out.println("Try again. (" +
21                     "Incorrect input: an integer is required");
22                 input.nextLine(); // Discard input
23             }
24         } while (continueInput);
25     }
26 }

```

```

Enter an integer: 3.5 [Enter]
Try again. (Incorrect input: an integer is required)
Enter an integer: 4 [Enter]
The number entered is 4

```

当执行 input.nextInt() (第 11 行) 时，如果键入的输入不是一个整数，就会出现一个 InputMismatchException 异常。假设输入的是 3.5，就会出现一个 InputMismatchException 异常，而且控制被转移到 catch 块。现在，执行 catch 块中的语句。第 22 行的语句 input.

nextLine() 丢弃当前的输入行，所以，用户就可以键入一个新行。变量 continueInput 来控制循环。它的初始值为 true (第 6 行)，当接收到的是一个合法值时，该值就变成 false (第 17 行)。一旦获得一个有效输入，就没有必要继续输入了。

复习题

12.1 使用异常处理的优势是什么？

12.2 下面哪些语句会抛出一个异常？

```
System.out.println(1 / 0);
System.out.println(1.0 / 0);
```

12.3 指出下面代码中的问题。代码会抛出任何异常吗？

```
long value = Long.MAX_VALUE + 1;
System.out.println(value);
```

12.4 当产生一个异常的时候，JVM 会做什么？如何捕获一个异常？

12.5 下面代码的输出是什么？

```
public class Test {
    public static void main(String[] args) {
        try {
            int value = 30;
            if (value < 40)
                throw new Exception("value is too small");
        }
        catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
        System.out.println("Continue after the catch block");
    }
}
```

如果把语句

```
int value = 30;
```

换成

```
int value = 50;
```

会输出什么结果？

12.6 给出下面代码的输出。

```
public class Test {
    public static void main(String[] args) {
        for (int i = 0; i < 2; i++) {
            System.out.print(i + " ");
            try {
                System.out.println(1 / 0);
            }
            catch (Exception ex) {
            }
        }
    }
}
```

a)

```
public class Test {
    public static void main(String[] args) {
        try {
            for (int i = 0; i < 2; i++) {
                System.out.print(i + " ");
                System.out.println(1 / 0);
            }
        }
        catch (Exception ex) {
        }
    }
}
```

b)

12.3 异常类型

要点提示：异常是对象，而对象都采用类来定义。异常的根类是 java.lang.Throwable。

前面小节使用了类 `ArithmeticException` 和 `InputMismatchException`。是否还有可以使用的其他类型的异常？可以定义自己的异常类吗？回答是肯定的。在 Java API 中有很多预定义的异常类。图 12-1 给出它们中的一部分。12.9 节中，你将学到如何定义自己的异常类。

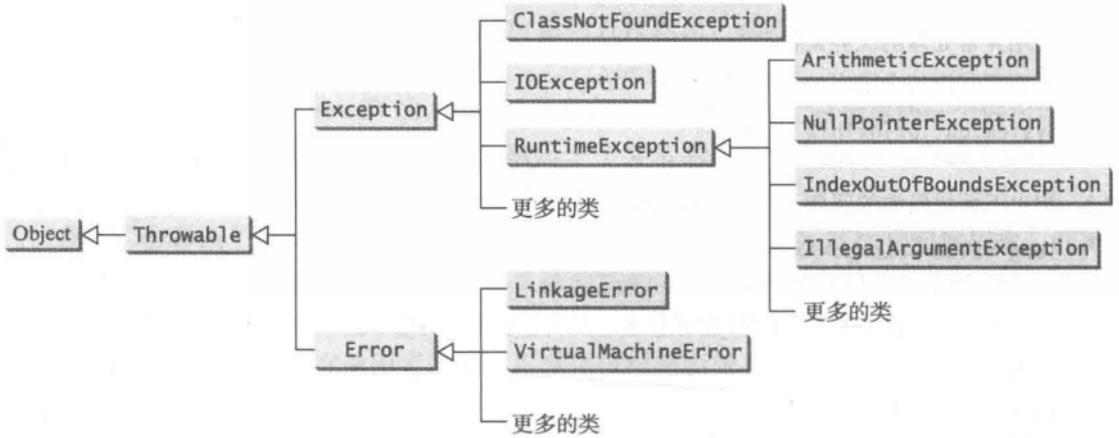


图 12-1 抛出的异常都是这个图中给出的类的实例，或者是这些类的子类的实例

注意：类名 `Error`、`Exception` 和 `RuntimeException` 有时候容易引起混淆。这三种类都是异常，这里讨论的错误都发生在运行时。

`Throwable` 类是所有异常类的根。所有的 Java 异常类都直接或者间接地继承自 `Throwable`。可以通过继承 `Exception` 或者 `Exception` 的子类来创建自己的异常类。

这些异常类可以分为三种主要类型：系统错误、异常和运行时异常。

- 系统错误（system error）是由 Java 虚拟机抛出的，用 `Error` 类表示。`Error` 类描述的是内部系统错误。这样的错误很少发生。如果发生，除了通知用户以及尽量稳妥地终止程序外，几乎什么也不能做。表 12-1 列出了 `Error` 的子类的一些例子。

表 12-1 `Error` 类的子类的例子

类	可能引起异常的原因
<code>LinkageError</code>	一个类对另一个类有某种依赖性，但是在编译前者后，后者进行了修改，变得不兼容
<code>VirtualMachineError</code>	Java 虚拟机崩溃，或者运行所必需的资源已经耗尽

- 异常（exception）是用 `Exception` 类表示的，它描述的是由程序和外部环境所引起的错误，这些错误能被程序捕获和处理。表 12-2 列出 `Exception` 类的子类的一些例子。

表 12-2 `Exception` 类的子类的例子

类	可能引起异常的原因
<code>ClassNotFoundException</code>	试图使用一个不存在的类。例如，如果试图使用命令 <code>java</code> 来运行一个不存在的类，或者程序要调用三个类文件而只能找到两个，都会发生这种异常
<code>IOException</code>	同输入 / 输出相关的操作，例如，无效的输入、读文件时超过文件尾、打开一个不存在的文件等。 <code>IOException</code> 的子类的例子有 <code>InterruptedException</code> 、 <code>EOFException</code> （ <code>EOF</code> 是 <code>End Of File</code> 的缩写）和 <code>FileNotFoundException</code>

- 运行时异常（runtime exception）是用 `RuntimeException` 类表示的，它描述的是程序设计错误，例如，错误的类型转换、访问一个越界数组或数值错误。运行时异常通常是由 Java 虚拟机抛出的。表 12-3 列出 `RuntimeException` 的子类的一些例子。

表 12-3 RuntimeException 类的子类的例子

类	可能引起异常的原因
ArithmeticException	一个整数除以 0。注意，浮点数的算术运算不抛出异常。参见附录 E
NullPointerException	试图通过一个 null 引用变量访问一个对象
IndexOutOfBoundsException	数组的下标超出范围
IllegalArgumentException	传递给方法的参数非法或不合适

RuntimeException、Error 以及它们的子类都称为免检异常 (unchecked exception)。所有其他异常都称为必检异常 (checked exception)，意思是指编译器会强制程序员检查并通过 try-catch 块处理它们，或者在方法头进行声明。在方法头声明一个异常将在 12.4 节中讨论到。

在大多数情况下，免检异常都会反映出程序设计上不可恢复的逻辑错误。例如，如果通过一个引用变量访问一个对象之前并未将一个对象赋值给它，就会抛出 NullPointerException 异常；如果访问一个数组的越界元素，就会抛出 IndexOutOfBoundsException 异常。这些都是程序中必须纠正的逻辑错误。免检异常可能在程序的任何一个地方出现。为避免过多地使用 try-catch 块，Java 语言不强制要求编写代码捕获或声明免检异常。

复习题

12.7 描述 Java 的 Throwable 类，它的子类以及异常的类型。

12.8 下面的程序如果将抛出 RuntimeException，那么会抛出哪种？

```
public class Test {
    public static void main(String[] args) {
        System.out.println(1 / 0);
    }
}
```

a)

```
public class Test {
    public static void main(String[] args) {
        int[] list = new int[5];
        System.out.println(list[5]);
    }
}
```

b)

```
public class Test {
    public static void main(String[] args) {
        String s = "abc";
        System.out.println(s.charAt(3));
    }
}
```

c)

```
public class Test {
    public static void main(String[] args) {
        Object o = new Object();
        String d = (String)o;
    }
}
```

d)

```
public class Test {
    public static void main(String[] args) {
        Object o = null;
        System.out.println(o.toString());
    }
}
```

e)

```
public class Test {
    public static void main(String[] args) {
        System.out.println(1.0 / 0);
    }
}
```

f)

12.4 关于异常处理的更多知识

要点提示：异常的处理是通过从当前的方法开始，沿着方法调用链，按照异常的反向传播方向找到的。

前面给出了异常处理的概况，同时介绍了几个预定义的异常类型。本节对异常处理进行深入讨论。

Java 的异常处理模型基于三种操作：声明一个异常 (declaring an exception)、抛出一个

异常 (throwing an exception) 和捕获一个异常 (catching an exception), 如图 12-2 所示。

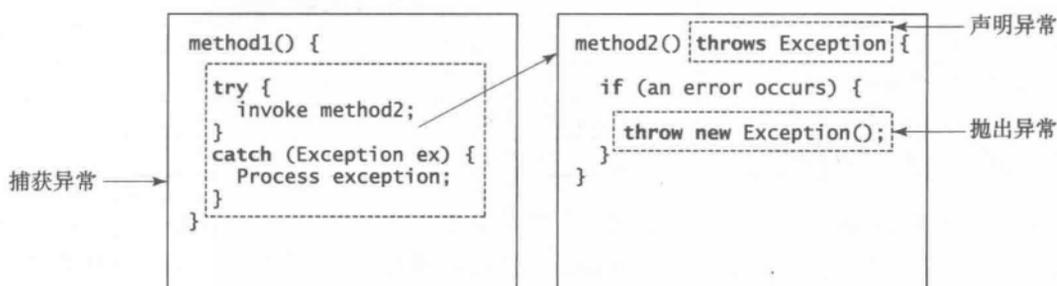


图 12-2 Java 中的异常处理包括声明异常、抛出异常以及捕获和处理异常

12.4.1 声明异常

在 Java 中, 当前执行的语句必属于某个方法。Java 解释器调用 main 方法开始执行一个程序。每个方法都必须声明它可能抛出的必检异常的类型。这称为声明异常 (declaring exception)。因为任何代码都可能发生系统错误和运行时错误, 因此, Java 不要求在方法中显式声明 Error 和 RuntimeException (免检异常)。但是, 方法要抛出的其他异常都必须在方法头中显式声明, 这样, 方法的调用者会被告知有异常。

为了在方法中声明一个异常, 就要在方法头中使用关键字 throws, 如下所示:

```
public void myMethod() throws IOException
```

关键字 throws 表明 myMethod 方法可能会抛出异常 IOException。如果方法可能会抛出多个异常, 就可以在关键字 throws 后添加一个用逗号分隔的异常列表:

```
public void myMethod()
    throws Exception1, Exception2, ..., ExceptionN
```

注意: 如果方法没有在父类中声明异常, 那么就不能在子类中对其进行继承来声明异常。

12.4.2 抛出异常

检测到错误的程序可以创建一个合适的异常类型的实例并抛出它, 这就称为抛出一个异常 (throwing an exception)。这里有一个例子, 假如程序发现传递给方法的参数与方法的合约不符 (例如, 方法中的参数必须是非负的, 但是传入的是一个负参数), 这个程序就可以创建 IllegalArgumentException 的一个实例并抛出它, 如下所示:

```
IllegalArgumentException ex =
    new IllegalArgumentException("Wrong Argument");
throw ex;
```

或者, 根据你的偏好, 也可以使用下面的语句:

```
throw new IllegalArgumentException("Wrong Argument");
```

注意: IllegalArgumentException 是 Java API 中的一个异常类。通常, Java API 中的每个异常类至少有两个构造方法: 一个无参构造方法和一个带可描述这个异常的 String 参数的构造方法。该参数称为异常消息 (exception message), 它可以用 getMessage() 获取。

提示: 声明异常的关键字是 throws, 抛出异常的关键字是 throw。

12.4.3 捕获异常

现在我们知道了如何声明一个异常以及如何抛出一个异常。当抛出一个异常时，可以在 try-catch 块中捕获和处理它，如下所示：

```
try {
    statements; // Statements that may throw exceptions
}
catch (Exception1 exVar1) {
    handler for exception1;
}
catch (Exception2 exVar2) {
    handler for exception2;
}
...
catch (ExceptionN exVarN) {
    handler for exceptionN;
}
```

如果在执行 try 块的过程中没有出现异常，则跳过 catch 子句。

如果 try 块中的某条语句抛出一个异常，Java 就会跳过 try 块中剩余的语句，然后开始查找处理这个异常的代码的过程。处理这个异常的代码称为异常处理器 (exception handler)；可以从当前的方法开始，沿着方法调用链，按照异常的反向传播方向找到这个处理器。从第一个到最后一个逐个检查 catch 块，判断在 catch 块中的异常类实例是否是该异常对象的类型。如果是，就将该异常对象赋值给所声明的变量，然后执行 catch 块中的代码。如果没有发现异常处理器，Java 会退出这个方法，把异常传递给调用这个方法的方法，继续同样的过程来查找处理器。如果在调用的方法链中找不到处理器，程序就会终止并且在控制台上打印出错信息。寻找处理器的过程称为捕获一个异常 (catching an exception)。

假设 main 方法调用 method1，method1 调用 method2，method2 调用 method3，method3 抛出一个异常，如图 12-3 所示。考虑下面的情形：

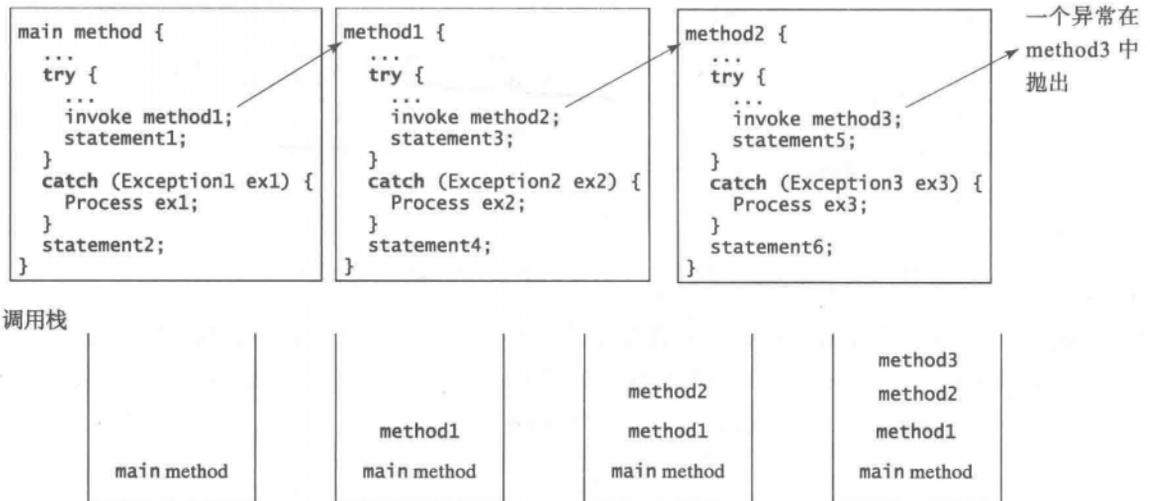


图 12-3 如果异常没有在当前的方法中被捕获，就被传给该方法的调用者
这个过程一直重复，直到异常被捕获或被传给 main 方法

- 如果异常类型是 `Exception3`，它就会被 `method2` 中处理异常 `ex3` 的 catch 块捕获。跳过 `statement5`，然后执行 `statement6`。

- 如果异常类型是 `Exception2`，则退出 `method2`，控制被返回给 `method1`，而这个异常就会被 `method1` 中处理异常 `ex2` 的 `catch` 块捕获。跳过 `statement3`，然后执行 `statement4`。
- 如果异常类型是 `Exception1`，则退出 `method1`，控制被返回给 `main` 方法，而这个异常就会被 `main` 方法中处理异常 `ex1` 的 `catch` 块捕获。跳过 `statement1`，然后执行 `statement2`。
- 如果异常类型没有在 `method2`、`method1` 和 `main` 方法中被捕获，程序就会终止。不执行 `statement1` 和 `statement2`。

- 🔧 注意：从一个通用的父类可以派生出各种异常类。如果一个 `catch` 块可以捕获一个父类的异常对象，它就能捕获那个父类的所有子类的异常对象。
- 🔧 注意：在 `catch` 块中异常被指定的顺序是非常重要的。如果父类的 `catch` 块出现在子类的 `catch` 块之前，就会导致编译错误。例如，a 中的顺序是错误的，因为 `RuntimeException` 是 `Exception` 的一个子类。正确的顺序应该如 b 中所示。

```
try {
    ...
}
catch (Exception ex) {
    ...
}
catch (RuntimeException ex) {
    ...
}
```

a) 错误的顺序

```
try {
    ...
}
catch (RuntimeException ex) {
    ...
}
catch (Exception ex) {
    ...
}
```

b) 正确的顺序

- 🔧 注意：Java 强迫程序员处理必检异常。如果方法声明了一个必检异常（即 `Error` 或 `RuntimeException` 之外的异常），就必须在 `try-catch` 块中调用它，或者在调用方法中声明要抛出异常。例如，假定方法 `p1` 调用方法 `p2`，而 `p2` 可能会抛出一个必检异常（例如，`IOException`），就必须如 a) 和 b) 所示编写代码。

```
void p1() {
    try {
        p2();
    }
    catch (IOException ex) {
        ...
    }
}
```

a) 捕获异常

```
void p1() throws IOException {
    p2();
}
```

b) 抛出异常

- 🔧 注意：对于使用同样的处理代码处理多个异常的情况，可以使用新的 JDK7 的多捕获特征（multi-catch feature）简化异常的代码编写。语法是：

```
catch (Exception1 | Exception2 | ... | Exceptionk ex) {
    // Same code for handling these exceptions
}
```

每个异常类型使用竖线（|）与下一个分隔。如果其中一个异常被捕获，则执行处理的代码。

12.4.4 从异常中获取信息

异常对象包含关于异常的有价值的信息。可以利用下面这些 `java.lang.Throwable` 类中

的实例方法获取有关异常的信息，如图 12-4 所示。printStackTrace() 方法在控制台上打印栈跟踪信息。getStackTrace() 方法提供编程的方式，来访问由 printStackTrace() 打印输出的栈跟踪信息。

java.lang.Throwable	
+getMessage(): String	返回描述该异常对象的信息
+toString(): String	返回三个字符串的连接: 1) 异常类的全名; 2) ": "(一个冒号和空白)
+printStackTrace(): void	3) getMessage (方法) 在控制台上打印 Throwable 对象和它的调用堆栈信息。
+getStackTrace(): StackTraceElement[]	返回和该异常对象相关的代表堆栈跟踪的一个堆栈跟踪元素的数组

图 12-4 Throwable 是所有异常类的根类

程序清单 12-6 给出了一个例子，它使用 Throwable 中的方法来显示异常信息。第 4 行调用 sum 方法返回数组中所有元素的和。第 23 行有一个错误，该错误引起一个异常 ArrayIndexOutOfBoundsException，它是 IndexOutOfBoundsException 的子类。该异常在 try-catch 块中被捕获。第 7、8、9 行使用 printStackTrace()、getMessage() 和 toString() 方法显示栈跟踪、异常信息、异常对象和信息，如图 12-5 所示。第 12 行将栈跟踪元素放入一个数组。每个元素表示一个方法调用。可以获得每个元素的方法（第 14 行）、类名（第 15 行）和异常行号（第 16 行）。

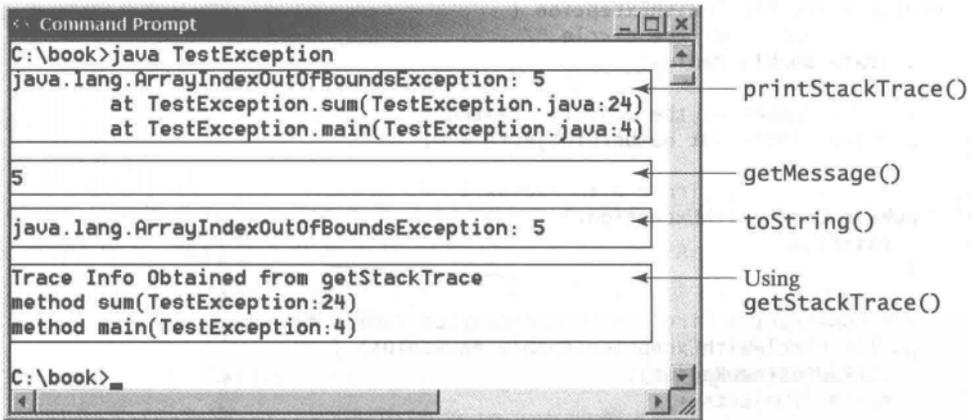


图 12-5 可以使用 printStackTrace()、getMessage()、toString() 和 getStackTrace() 方法从异常对象获取信息

程序清单 12-6 TestException.java

```

1 public class TestException {
2     public static void main(String[] args) {
3         try {
4             System.out.println(sum(new int[] {1, 2, 3, 4, 5}));
5         }
6         catch (Exception ex) {
7             ex.printStackTrace();
8             System.out.println("\n" + ex.getMessage());
9             System.out.println("\n" + ex.toString());
10
11             System.out.println("\nTrace Info Obtained from getStackTrace");
  
```

```

12     StackTraceElement[] traceElements = ex.getStackTrace();
13     for (int i = 0; i < traceElements.length; i++) {
14         System.out.print("method " + traceElements[i].getMethodName());
15         System.out.print("(" + traceElements[i].getClassName() + "):");
16         System.out.println(traceElements[i].getLineNumber() + ")");
17     }
18 }
19 }
20
21 private static int sum(int[] list) {
22     int result = 0;
23     for (int i = 0; i <= list.length; i++)
24         result += list[i];
25     return result;
26 }
27 }

```

12.4.5 示例学习：声明、抛出和捕获异常

本例改写程序清单 9-8 中 Circle 类的 setRadius 方法来演示如何声明、抛出和捕获异常。如果半径是负数，那么新的 setRadius 方法会抛出一个异常。

程序清单 12-7 定义了一个名为 CircleWithException 的新的圆类，除了 setRadius(double newRadius) 方法在参数 newRadius 为负时会抛出一个 IllegalArgumentException 异常之外，它与 CircleWithPrivateDataFields 类是一样的。

程序清单 12-7 CircleWithException.java

```

1 public class CircleWithException {
2     /** The radius of the circle */
3     private double radius;
4
5     /** The number of the objects created */
6     private static int numberOfObjects = 0;
7
8     /** Construct a circle with radius 1 */
9     public CircleWithException() {
10        this(1.0);
11    }
12
13    /** Construct a circle with a specified radius */
14    public CircleWithException(double newRadius) {
15        setRadius(newRadius);
16        numberOfObjects++;
17    }
18
19    /** Return radius */
20    public double getRadius() {
21        return radius;
22    }
23
24    /** Set a new radius */
25    public void setRadius(double newRadius)
26        throws IllegalArgumentException {
27        if (newRadius >= 0)
28            radius = newRadius;
29        else
30            throw new IllegalArgumentException(
31                "Radius cannot be negative");
32    }
33
34    /** Return numberOfObjects */

```

```

35 public static int getNumberOfObjects() {
36     return numberOfObjects;
37 }
38
39 /** Return the area of this circle */
40 public double findArea() {
41     return radius * radius * 3.14159;
42 }
43 }

```

程序清单 12-8 给出使用新 Circle 类的测试程序。

程序清单 12-8 TestCircleWithException.java

```

1 public class TestCircleWithException {
2     public static void main(String[] args) {
3         try {
4             CircleWithException c1 = new CircleWithException(5);
5             CircleWithException c2 = new CircleWithException(-5);
6             CircleWithException c3 = new CircleWithException(0);
7         }
8         catch (IllegalArgumentException ex) {
9             System.out.println(ex);
10        }
11
12        System.out.println("Number of objects created: " +
13            CircleWithException.getNumberOfObjects());
14    }
15 }

```

```

java.lang.IllegalArgumentException: Radius cannot be negative
Number of objects created: 1

```

原始的 Circle 类除了下面三点之外其他保持不变：将类名改为 CircleWithException，加入一个新的构造方法 CircleWithException(newRadius) 以及如果半径为负则 setRadius 方法声明一个异常并抛出它。

setRadius 方法在方法头中声明为抛出 IllegalArgumentException 异常（程序清单 12-7 中的第 25 ~ 32 行）。即使在方法声明中删除 throws IllegalArgumentException 子句（第 26 行），CircleWithException 类也仍然会编译，因为该异常是 RuntimeException 的子类，而且不管是否在方法头中声明，每个方法都能抛出 RuntimeException 异常（免检异常）。

测试程序创建三个 CircleWithException 对象：c1、c2 和 c3，测试如何处理异常。调用 new CircleWithException(-5)（程序清单 12-8 中的第 5 行）会导致对 setRadius 方法的调用，因为半径为负，所以 setRadius 方法会抛出 IllegalArgumentException 异常。在 catch 块中，对象 ex 的类型是 IllegalArgumentException，它与 setRadius 方法抛出的异常对象相匹配，因此，这个异常被 catch 块捕获。

异常处理器使用 System.out.println(ex) 打印一个有关异常的短消息 ex.toString()（程序清单 12-8 中第 9 行）。

注意：在异常事件中，执行仍然会继续。如果处理器没有捕获到这个异常，程序就会突然中断。

由于这个方法抛出 RuntimeException（免检异常）子类 IllegalArgumentException 的一个实例，所以，如果不使用 try 语句，这个测试程序也能编译。如果方法抛出 RuntimeException 和 Error 之外的异常，那么此方法就必须在 try-catch 块内调用。

复习题

- 12.9 声明异常的目的是什么? 怎样声明一个异常, 在哪里声明? 在一个方法头中可以声明多个异常吗?
- 12.10 什么是必检异常? 什么是免检异常?
- 12.11 如何抛出一个异常? 可以在一个 `throw` 语句中抛出多个异常吗?
- 12.12 关键字 `throw` 的作用是什么? 关键字 `throws` 的作用是什么?
- 12.13 假设下面的 `try-catch` 块中的 `statement2` 引起一个异常:

```
try {
    statement1;
    statement2;
    statement3;
}
catch (Exception1 ex1) {
}
catch (Exception2 ex2) {
}
```

`statement4;`

回答下列问题:

- 会执行 `statement3` 吗?
 - 如果异常未被捕获, 会执行 `statement4` 吗?
 - 如果在 `catch` 块中捕获了异常, 会执行 `statement4` 吗?
- 12.14 运行下面程序时会显示什么?

```
public class Test {
    public static void main(String[] args) {
        try {
            int[] list = new int[10];
            System.out.println("list[10] is " + list[10]);
        }
        catch (ArithmeticException ex) {
            System.out.println("ArithmeticException");
        }
        catch (RuntimeException ex) {
            System.out.println("RuntimeException");
        }
        catch (Exception ex) {
            System.out.println("Exception");
        }
    }
}
```

- 12.15 运行下面程序时会显示什么?

```
public class Test {
    public static void main(String[] args) {
        try {
            method();
            System.out.println("After the method call");
        }
        catch (ArithmeticException ex) {
            System.out.println("ArithmeticException");
        }
        catch (RuntimeException ex) {
            System.out.println("RuntimeException");
        }
        catch (Exception e) {
```

```

        System.out.println("Exception");
    }
}

static void method() throws Exception {
    System.out.println(1 / 0);
}
}

```

12.16 运行下面程序时会显示什么?

```

public class Test {
    public static void main(String[] args) {
        try {
            method();
            System.out.println("After the method call");
        }
        catch (RuntimeException ex) {
            System.out.println("RuntimeException in main");
        }
        catch (Exception ex) {
            System.out.println("Exception in main");
        }
    }

    static void method() throws Exception {
        try {
            String s = "abc";
            System.out.println(s.charAt(3));
        }
        catch (RuntimeException ex) {
            System.out.println("RuntimeException in method()");
        }
        catch (Exception ex) {
            System.out.println("Exception in method()");
        }
    }
}

```

12.17 方法 `getMessage()` 可以做什么?

12.18 方法 `printStackTrace()` 可以做什么?

12.19 没有异常发生时, `try-catch` 块的存在会引起额外的系统开销吗?

12.20 修改下面代码中的编译错误:

```

public void m(int value) {
    if (value < 40)
        throw new Exception("value is too small");
}

```

12.5 finally 子句

 **要点提示:** 无论异常是否产生, `finally` 子句总是会被执行的。

有时候, 不论异常是否出现或者是否被捕获, 都希望执行某些代码。Java 有一个 `finally` 子句, 可以用来达到这个目的。 `finally` 子句的语法如下所示:

```

try {
    statements;
}
catch (TheException ex) {
    handling ex;
}

```

```
finally {
    finalStatements;
}
```

在任何情况下，finally 块中的代码都会执行，不论 try 块中是否出现异常或者是否被捕获。考虑下面三种可能出现的情况：

- 如果 try 块中没有出现异常，执行 finalStatements，然后执行 try 语句的下一条语句。
- 如果 try 块中有一条语句引起异常，并被 catch 块捕获，然后跳过 try 块的其他语句，执行 catch 块和 finally 子句。执行 try 语句之后的下一条语句。
- 如果 try 块中有一条语句引起异常，但是没有被任何 catch 块捕获，就会跳过 try 块中的其他语句，执行 finally 子句，并且将异常传递给这个方法的调用者。

即使在到达 finally 块之前有一个 return 语句，finally 块还是会执行。

 **注意：**使用 finally 子句时可以省略掉 catch 块。

复习题

12.21 假设下面的语句中，statement2 会引起一个异常：

```
try {
    statement1;
    statement2;
    statement3;
}
catch (Exception1 ex1) {
}
finally {
    statement4;
}
statement5;
```

回答以下问题：

- 如果没有异常发生，那么会执行 statement4 吗？会执行 statement5 吗？
- 如果异常类型是 Exception1，那么会执行 statement4 吗？会执行 statement5 吗？
- 如果异常不是类型 Exception1，那么会执行 statement4 吗？会执行 statement5 吗？

12.6 何时使用异常

 **要点提示：**当错误需要被方法的调用者处理的时候，方法应该抛出一个异常。

try 块包含正常情况下执行的代码。catch 块包含异常情况下执行的代码。异常处理将错误处理代码从正常的程序设计任务中分离出来，这样，可以使程序更易读、更易修改。但是，应该注意，由于异常处理需要初始化新的异常对象，需要从调用栈返回，而且还需要沿着方法调用链来传播异常以便找到它的异常处理器，所以，异常处理通常需要更多的时间和资源。

异常出现在方法中。如果想让该方法的调用者处理异常，应该创建一个异常对象并将其抛出。如果能在发生异常的方法中处理异常，那么就不需要抛出或使用异常。

一般来说，一个项目中多个类都会发生的共同异常应该考虑作为一种异常类。对于发生在个别方法中的简单错误最好进行局部处理，无须抛出异常。

在代码中，应该什么时候使用 try-catch 块呢？当必须处理不可预料的错误状况时应该使用它。不要用 try-catch 块处理简单的、可预料的情况。例如，下面的代码：

```
try {
    System.out.println(refVar.toString());
}
catch (NullPointerException ex) {
    System.out.println("refVar is null");
}
```

最好用以下代码代替：

```
if (refVar != null)
    System.out.println(refVar.toString());
else
    System.out.println("refVar is null");
```

哪些情况是异常的，哪些情况是可预料的，有时很难判断。但有一点要把握住，不要把异常处理用作简单的逻辑测试。

☛ 复习题

12.22 下面的方法检查一个字符串是否是数值字符串：

```
public static boolean isNumeric(String token) {
    try {
        Double.parseDouble(token);
        return true;
    }
    catch (java.lang.NumberFormatException ex) {
        return false;
    }
}
```

该方法是否正确？不使用异常重写该方法。

12.7 重新抛出异常

🔍 **要点提示：**如果异常处理器不能处理一个异常，或者只是简单地希望它的调用者注意到该异常，Java 允许该异常处理器重新抛出异常。

重新抛出异常的语法如下所示：

```
try {
    statements;
}
catch (TheException ex) {
    perform operations before exits;
    throw ex;
}
```

语句 `throw ex` 重新抛出异常给调用者，以便调用者的其他处理器获得处理异常 `ex` 的机会。

☛ 复习题

12.23 假设下面的语句中，`statement2` 会引起一个异常：

```
try {
    statement1;
    statement2;
    statement3;
}
catch (Exception1 ex1) {
}
catch (Exception2 ex2) {
    throw ex2;
}
```

```

}
finally {
    statement4;
}
statement5;

```

回答以下问题：

- 如果没有异常发生，会执行语句 `statement4` 吗？会执行语句 `statement5` 吗？
- 如果异常类型是 `Exception1`，那么会执行 `statement4` 吗？会执行 `statement5` 吗？
- 如果异常类型是 `Exception2`，那么会执行 `statement4` 吗？会执行 `statement5` 吗？
- 如果异常类型不是 `Exception1` 以及 `Exception2` 类型的，那么会执行 `statement4` 吗？会执行 `statement5` 吗？

12.8 链式异常

 **要点提示：** 和其他异常一起抛出一个异常，构成了链式异常。

在 12.7 节中，`catch` 块重新抛出原始的异常。有时候，可能需要同原始异常一起抛出一个新异常（带有附加信息），这称为链式异常（`chained exception`）。程序清单 12-9 解释了如何产生和抛出链式异常。

程序清单 12-9 ChainedExceptionDemo.java

```

1 public class ChainedExceptionDemo {
2     public static void main(String[] args) {
3         try {
4             method1();
5         }
6         catch (Exception ex) {
7             ex.printStackTrace();
8         }
9     }
10
11    public static void method1() throws Exception {
12        try {
13            method2();
14        }
15        catch (Exception ex) {
16            throw new Exception("New info from method1", ex);
17        }
18    }
19
20    public static void method2() throws Exception {
21        throw new Exception("New info from method2");
22    }
23 }

```

```

java.lang.Exception: New info from method1
    at ChainedExceptionDemo.method1(ChainedExceptionDemo.java:16)
    at ChainedExceptionDemo.main(ChainedExceptionDemo.java:4)
Caused by: java.lang.Exception: New info from method2
    at ChainedExceptionDemo.method2(ChainedExceptionDemo.java:21)
    at ChainedExceptionDemo.method1(ChainedExceptionDemo.java:13)
    ... 1 more

```

`main` 方法调用 `method1`（第 4 行），`method1` 调用 `method2`（第 13 行），`method2` 抛出一个异常（第 21 行）。该异常被 `method1` 的 `catch` 块所捕获，并在第 16 行被包装成一个新异常。该新异常被抛出，并在 `main` 方法中的 `catch` 块中被捕获（第 6 行）。示例输出在第 7 行

中 `printStackTrace()` 方法的结果。首先，显示从 `method1` 中抛出的新异常，然后显示从 `method2` 中抛出的原始异常。

复习题

12.24 如果第 16 行被下面一行所替代，将输出什么？

```
throw new Exception("New info from method1");
```

12.9 创建自定义异常类

要点提示：可以通过派生 `java.lang.Exception` 类来定义一个自定义异常类。

Java 提供相当多的异常类，尽量使用它们而不要创建自己的异常类。然而，如果遇到一个不能用预定义异常类恰当描述的问题，那就可以通过派生 `Exception` 类或其子类，例如，`IOException`，来创建自己的异常类。

在程序清单 12-7 中，当半径为负时，`setRadius` 方法会抛出一个异常。假设希望把这个半径传递给处理器。在这种情况下，就必须创建自定义异常类，如程序清单 12-10 所示。

程序清单 12-10 `InvalidRadiusException.java`

```
1 public class InvalidRadiusException extends Exception {
2     private double radius;
3
4     /** Construct an exception */
5     public InvalidRadiusException(double radius) {
6         super("Invalid radius " + radius);
7         this.radius = radius;
8     }
9
10    /** Return the radius */
11    public double getRadius() {
12        return radius;
13    }
14 }
```

这个自定义异常类继承自 `java.lang.Exception` (第 1 行)。而 `Exception` 类扩展自 `java.lang.Throwable`。`Exception` 类中的所有方法 (例如，`getMessage()`、`toString()` 和 `printStackTrace()`) 都是从 `Throwable` 继承而来的。`Exception` 类包括四个构造方法，其中经常使用的是下面两个构造方法：

java.lang.Exception	
+Exception()	构建一个没有消息的异常
+Exception(message: String)	构建一个给定消息的异常

第 6 行调用父类的带有一条消息的构造方法。这条消息将会被设置在异常对象中，并且可以通过在该对象上调用 `getMessage()` 获得。

提示：Java API 中的大多数异常类都包含两个构造方法：一个无参构造方法和一个带消息参数的构造方法。

要创建一个 `InvalidRadiusException` 类，必须传递一个半径。所以，程序清单 12-7 中的 `setRadius` 方法可以修改如程序清单 12-11 所示：

程序清单 12-11 TestCircleWithCustomException.java

```

1 public class TestCircleWithCustomException {
2     public static void main(String[] args) {
3         try {
4             new CircleWithCustomException(5);
5             new CircleWithCustomException(-5);
6             new CircleWithCustomException(0);
7         }
8         catch (InvalidRadiusException ex) {
9             System.out.println(ex);
10        }
11
12        System.out.println("Number of objects created: " +
13            CircleWithCustomException.getNumberOfObjects());
14    }
15 }
16
17 class CircleWithCustomException {
18     /** The radius of the circle */
19     private double radius;
20
21     /** The number of objects created */
22     private static int numberOfObjects = 0;
23
24     /** Construct a circle with radius 1 */
25     public CircleWithCustomException() throws InvalidRadiusException { declare exception
26         this(1.0);
27     }
28
29     /** Construct a circle with a specified radius */
30     public CircleWithCustomException(double newRadius)
31         throws InvalidRadiusException {
32         setRadius(newRadius);
33         numberOfObjects++;
34     }
35
36     /** Return radius */
37     public double getRadius() {
38         return radius;
39     }
40
41     /** Set a new radius */
42     public void setRadius(double newRadius)
43         throws InvalidRadiusException {
44         if (newRadius >= 0)
45             radius = newRadius;
46         else
47             throw new InvalidRadiusException(newRadius);
48     }
49
50     /** Return numberOfObjects */
51     public static int getNumberOfObjects() {
52         return numberOfObjects;
53     }
54
55     /** Return the area of this circle */
56     public double findArea() {
57         return radius * radius * 3.14159;
58     }
59 }

```

```

InvalidRadiusException: Invalid radius -5.0
Number of objects created: 1

```

当半径为负时，CircleWithCustomException 中的 setRadius 方法会抛出一个 InvalidRadiusException (第 47 行)。由于 InvalidRadiusException 是一个必检异常，setRadius 方法必须在方法头部进行声明 (第 43 行)。由于 CircleWithCustomException 的构造方法调用了 setRadius 方法来设置一个新的半径，而该方法可能会抛出一个 InvalidRadiusException，构造方法需要声明抛出 InvalidRadiusException (第 25、31 行)。

调用 new CircleWithCustomException(-5) 方法会抛出一个 InvalidRadiusException 异常，它被处理器捕获。处理器在异常对象 ex 中显示半径。

提示：可以扩展 RuntimeException 声明一个自定义异常类吗？可以，但这不是一个好方法，因为这会使自定义异常成为免检异常。最好使自定义异常必检，这样，编译器就可以在程序中强制捕获这些异常。

复习题

12.25 如何定义一个自定义异常类？

12.26 假定 setRadius 方法抛出程序清单 12-10 中定义的 InvalidRadiusException 异常，那么运行下面的程序时会显示什么？

```
public class Test {
    public static void main(String[] args) {
        try {
            method();
            System.out.println("After the method call");
        }
        catch (RuntimeException ex) {
            System.out.println("RuntimeException in main");
        }
        catch (Exception ex) {
            System.out.println("Exception in main");
        }
    }

    static void method() throws Exception {
        try {
            Circle c1 = new Circle(1);
            c1.setRadius(-1);
            System.out.println(c1.getRadius());
        }
        catch (RuntimeException ex) {
            System.out.println("RuntimeException in method()");
        }
        catch (Exception ex) {
            System.out.println("Exception in method()");
            throw ex;
        }
    }
}
```

12.10 File 类

要点提示：File 类包含了获得一个文件 / 目录的属性，以及对文件 / 目录进行改名和删除的方法。

在学完异常处理后，我们来学习文件处理了。存储在程序中的数据是暂时的，当程序终止时它们就会丢失。为了能够永久地保存程序中创建的数据，需要将它们存储到磁盘或其他永久存储设备的文件中。这样，这些文件其后可以被其他程序传送和读取。由于数据存储在文件中，所以本节就介绍如何使用 File 类获取文件 / 目录的属性以及删除和重命名文件 / 目

录，以及创建目录。下一节将介绍如何从 / 向一个文本文件读 / 写数据。

在文件系统中，每个文件都存放在一个目录下。绝对文件名（absolute file name）是由文件名和它的完整路径以及驱动器字母组成。例如，c:\book\Welcome.java 是文件 Welcome.java 在 Windows 操作系统上的绝对文件名。这里的 c:\book 称为该文件的目录路径（directory path）。绝对文件名是依赖机器的，在 UNIX 平台上，绝对文件名可能会是 /home/liang/book/Welcome.java，其中 /home/liang/book 是文件 Welcome.java 的目录路径。

相对文件名是相对于当前工作目录的。对于相对文件名而言，完整目录被忽略。例如，Welcome.java 是一个相对文件名。如果当前工作目录是 c:\book，绝对文件名将是 c:\book\Welcome.java。

File 类意图提供了一种抽象，这种抽象是指以不依赖机器的方式来处理很多依赖于机器的文件和路径名的复杂性。File 类包含许多获取文件属性的方法，以及重命名和删除文件和目录的方法，如图 12-6 所示。但是，File 类不包含读写文件内容的方法。

java.io.File	
+File(pathname: String)	为一个指定的路径名创建一个 File 对象。路径名可能是一个目录或者一个文件
+File(parent: String, child: String)	在目录 parent 下创建一个子路径的 File 对象，子路径可能是一个目录或者一个文件
+File(parent: File, child: String)	在目录 parent 下创建一个子路径的 File 对象。parent 是一个 File 对象。之前的构造方法中，parent 是一个字符串
+exists(): boolean	File 对象代表的文件和目录存在，返回 true
+canRead(): boolean	File 对象代表的文件存在且可读，返回 true
+canWrite(): boolean	File 对象代表的文件存在且可写，返回 true
+isDirectory(): boolean	File 对象代表的是一个目录，返回 true
+isFile(): boolean	File 对象代表的是一个文件，返回 true
+isAbsolute(): boolean	File 对象是采用绝对路径名创建，返回 true
+isHidden(): boolean	如果 File 对象代表的文件是隐藏的，返回 true。隐藏的确切定义是系统相关的。Windows 系统中，可以在文件属性对话框中标记一个文件隐藏。Unix 系统中，如果文件名以点字符（.）开始，则文件是隐藏的
+getAbsolutePath(): String	返回 File 对象代表的文件和目录的完整绝对路径名
+getCanonicalPath(): String	和 getAbsolutePath() 返回相同，除了从路径名中去掉了冗余的名字，比如 "." 和 ".."，以及解析符号链接（Unix 中），将盘符转化为标准的大写形式（Windows 中）
+getName(): String	返回 File 对象代表的目录和文件名的最后名字。例如，new File("c:\\book\\test.dat").getName() 返回 test.dat
+getPath(): String	返回 File 对象代表的完整的目录和文件名。例如，new File("c:\\book\\test.dat").getPath() 返回 c:\book\test.dat
+getParent(): String	返回 File 对象代表的当前目录和文件的完整父目录。例如，new File("c:\\book\\test.dat").getParent() 返回 c:\book
+lastModified(): long	返回文件最后修改时间
+length(): long	返回文件的大小，如果不存在的或者是一个目录的话，返回 0
+listFile(): File[]	返回一个目录 File 对象下面的文件
+delete(): boolean	删除 File 对象代表的文件或者目录。如果删除成功，方法返回 true
+renameTo(dest: File): boolean	将该 File 对象代表的文件或者目录改名为 dest 中指定的名字。如果操作成功，方法返回 true
+mkdir(): boolean	创建该 File 对象代表的目录。如果目录成功创建，则返回 true
+mkdirs(): boolean	和 mkdir() 相同，除开在父目录不存在的情况下，将和父目录一起创建

图 12-6 File 类可以用来获取文件和目录的属性，删除和重命名文件和目录，以及创建目录

文件名是一个字符串。File 类是文件名及其目录路径的一个包装类。例如，在 Windows 中，语句 new File("c:\\book") 在目录 c:\book 下创建一个 File 对象，而语句 new File("c:\\book\\test.dat") 为文件 c:\book\test.dat 创建一个 File 对象。可以用 File 类

的 `isDirectory()` 方法来判断这个对象是否表示一个目录，还可以用 `isFile()` 方法来判断这个对象是否表示一个文件名。

警告：在 Windows 中目录的分隔符是反斜杠 (\)。但是在 Java 中，反斜杠是一个特殊的字符，应该写成 \\ 的形式（参见表 4-5）。

注意：构建一个 `File` 实例并不会在机器上创建一个文件。不管文件是否存在，都可以创建任意文件名的 `File` 实例。可以调用 `File` 实例上的 `exists()` 方法来判断这个文件是否存在。

在程序中，不要直接使用绝对文件名。如果使用了像 `c:\\book\\Welcome.java` 之类的文件名，那么它能在 Windows 上工作，但是不能在其他平台上工作。应该使用与当前目录相关的文件名。例如，可以使用 `new File("Welcome.java")` 为在当前目录下的文件 `Welcome.java` 创建一个 `File` 对象。可以使用 `new File("image/us.gif")` 为在当前目录下的 `image` 目录下的文件 `us.gif` 创建一个 `File` 对象。斜杠 (/) 是 Java 的目录分隔符，这点和 UNIX 是一样的。语句 `new File("image/us.gif")` 在 Windows、UNIX 或任何其他系统上都能工作。

程序清单 12-12 演示如何创建一个 `File` 对象，以及如何使用 `File` 类中的方法获取它的属性。这个程序为文件 `us.gif` 创建了一个 `File` 对象。这个文件存储在当前目录的 `image` 目录下。

程序清单 12-12 TestFileClass.java

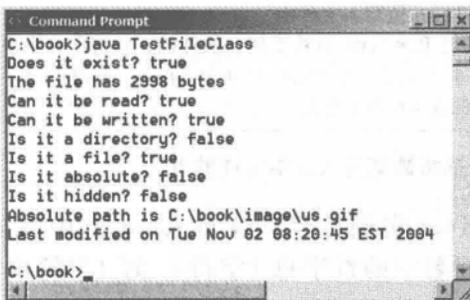
```

1 public class TestFileClass {
2     public static void main(String[] args) {
3         java.io.File file = new java.io.File("image/us.gif");
4         System.out.println("Does it exist? " + file.exists());
5         System.out.println("The file has " + file.length() + " bytes");
6         System.out.println("Can it be read? " + file.canRead());
7         System.out.println("Can it be written? " + file.canWrite());
8         System.out.println("Is it a directory? " + file.isDirectory());
9         System.out.println("Is it a file? " + file.isFile());
10        System.out.println("Is it absolute? " + file.isAbsolute());
11        System.out.println("Is it hidden? " + file.isHidden());
12        System.out.println("Absolute path is " +
13            file.getAbsolutePath());
14        System.out.println("Last modified on " +
15            new java.util.Date(file.lastModified()));
16    }
17 }

```

`lastModified()` 方法返回文件最后被修改的日期和时间，它计算的是从 UNIX 时间（1970 年 1 月 1 日 0 时 0 分 0 秒）开始的毫秒数，第 14 ~ 15 行使用 `Date` 类以一种易读的格式显示它。

图 12-7a 显示程序在 Windows 平台上的运行示例，而图 12-7b 显示程序在 UNIX 平台上的运行示例。如图所示，Windows 平台和 UNIX 平台的路径命名习惯是不一样的。



a) Windows 平台



b) UNIX 平台

图 12-7 程序创建一个 `File` 对象然后显示文件属性

复习题

12.27 使用下面的语句创建 File 对象时，错在哪里？

```
new File("c:\book\test.dat");
```

12.28 如何检查一个文件是否已经存在？如何删除一个文件？如何重命名一个文件？使用 File 类能够获得文件的大小（字节数）吗？如何创建一个目录？

12.29 能够使用 File 类进行输入/输出吗？创建一个 File 对象就是在磁盘上创建一个文件吗？

12.11 文件输入和输出

要点提示：使用 Scanner 类从文件中读取文本数据，使用 PrintWriter 类向文本文件写入数据。

File 对象封装了文件或路径的属性，但是它既不包括创建文件的方法，也不包括从/向文件读/写数据（称为数据输入输出，简称 I/O）的方法。为了完成 I/O 操作，需要使用恰当的 Java I/O 类创建对象。这些对象包含从/向文件读/写数据的方法。文本文件本质上是存储在磁盘上的字符。本节介绍如何使用 Scanner 和 PrintWriter 类从（向）文本文件读（写）字符串和数值信息。二进制文件将在 17 章介绍。

12.11.1 使用 PrintWriter 写数据

java.io.PrintWriter 类可用来创建一个文件并向文本文件写入数据。首先，必须为一个文本文件创建一个 PrintWriter 对象，如下所示：

```
PrintWriter output = new PrintWriter(filename);
```

然后，可以调用 PrintWriter 对象上的 print、println 和 printf 方法向文件写入数据。图 12-8 总结了 PrintWriter 中的常用方法。

java.io.PrintWriter	
+PrintWriter(file: File)	为指定的文件对象创建一个 PrintWriter 对象
+PrintWriter(filename: String)	为指定的文件名字符串创建一个 PrintWriter 对象
+print(s: String): void	将字符串写入文件中
+print(c: char): void	将字符写入文件中
+print(cArray: char[]): void	将字符数组写入文件中
+print(i: int): void	将一个 int 值写入文件中
+print(l: long): void	将一个 long 值写入文件中
+print(f: float): void	将一个 float 值写入文件中
+print(d: double): void	将一个 double 值写入文件中
+print(b: boolean): void	将一个 boolean 值写入文件中
也包含重载的 println 方法	println 方法和 print 方法类似；额外的，它打印一个换行。换行字符串由系统定义。在 Windows 上位 \r\n，在 Unix 上为 \n
也包含重载的 printf 方法	printf 方法在 4.6 节中介绍

图 12-8 PrintWriter 类包括将数据写入文本文件的方法

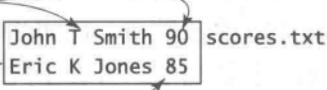
程序清单 12-13 给出创建一个 PrintWriter 实例并且向文件 score.txt 中写入两行数据的例子。每行都包括名字（字符串）、中间名字的首字母（字符）、姓（字符串）和分数（整数）。

程序清单 12-13 WriteData.java

```

1 public class WriteData {
2     public static void main(String[] args) throws IOException {
3         java.io.File file = new java.io.File("scores.txt");
4         if (file.exists()) {
5             System.out.println("File already exists");
6             System.exit(1);
7         }
8
9         // Create a file
10        java.io.PrintWriter output = new java.io.PrintWriter(file);
11
12        // Write formatted output to the file
13        output.print("John T Smith ");
14        output.println(90);
15        output.print("Eric K Jones ");
16        output.println(85);
17
18        // Close the file
19        output.close();
20    }
21 }

```



第 4 ~ 7 行检查文件 score.txt 是否存在。如果存在，则退出该程序（第 6 行）。

如果文件不存在，调用 `PrintWriter` 的构造方法会创建一个新文件。如果文件已经存在，那么文件的当前内容将在不和用户确认的情况下被废弃。

调用 `PrintWriter` 的构造方法可能会抛出某种 I/O 异常。Java 强制要求编写代码来处理这类异常。在 13 章中将学习如何处理它。为简单起见，只要在方法头声明中声明 `throws Exception`（第 2 行）即可。

我们已经使用过 `System.out.print`、`System.out.println` 和 `System.out.printf` 方法向控制台输出文本。`System.out` 是控制台的标准 Java 对象。可以创建对象，然后使用 `print`、`println` 和 `printf` 向文件中写入文本（第 13 ~ 16 行）。

必须使用 `close()` 方法关闭文件（第 19 行）。如果没有调用该方法，数据就不能正确地保存在文件中。

12.11.2 使用 try-with-resources 自动关闭资源

程序员经常会忘记关闭文件。JDK 7 提供了下面的新的 `try-with-resources` 语法来自动关闭文件。

```

try(声明和创建资源){
    使用资源来处理文件；
}

```

使用 `try-with-resources` 语法，我们重写程序清单 12-14 中的代码。

程序清单 12-14 WriteDataWithAutoClose.java

```

1 public class WriteDataWithAutoClose {
2     public static void main(String[] args) throws Exception {
3         java.io.File file = new java.io.File("scores.txt");
4         if (file.exists()) {
5             System.out.println("File already exists");
6             System.exit(0);
7         }
8     }

```

```

9     try (
10        // Create a file
11        java.io.PrintWriter output = new java.io.PrintWriter(file);
12    ) {
13        // Write formatted output to the file
14        output.print("John T Smith ");
15        output.println(90);
16        output.print("Eric K Jones ");
17        output.println(85);
18    }
19 }
20 }

```

关键字 `try` 后声明和创建了一个资源。注意，资源放在括号中（第 9 ~ 12 行）。资源必须是 `AutoCloseable` 的子类型，比如 `PrintWriter`，具有一个 `close()` 方法。资源的声明和创建必须在同一行语句中，可以在括号中进行多个资源的声明和创建。紧接着资源声明的块中的语句（第 12 ~ 18 行）使用资源。块结束后，资源的 `close()` 方法自动调用以关闭资源。使用 `try-with-resource` 不仅可以避免错误，而且可以简化代码。

12.11.3 使用 Scanner 读数据

在 2.3 节中，`java.util.Scanner` 类用来从控制台读取字符串和基本类型数值。`Scanner` 可以将输入分为由空白字符分隔的标记。为了能从键盘读取，需要为 `System.in` 创建一个 `Scanner`，如下所示：

```
Scanner input = new Scanner(System.in);
```

为了从文件中读取，为文件创建一个 `Scanner`，如下所示：

```
Scanner input = new Scanner(new File(filename));
```

图 12-9 总结出 `Scanner` 中的常用方法。

java.util.Scanner	
+Scanner(source: File)	创建一个 Scanner，从指定的文件中扫描标记
+Scanner(source: String)	创建一个 Scanner，从指定的字符串中扫描标记
+close()	关闭该 Scanner
+hasNext(): boolean	如果 Scanner 还有更多数据读取，则返回 true
+next(): String	从该 Scanner 中读取下一个标记作为字符串返回
+nextLine(): String	从该 Scanner 中读取一行，以换行结束
+nextByte(): byte	从该 Scanner 中读取下一个标记作为 byte 值返回
+nextShort(): short	从该 Scanner 中读取下一个标记作为 short 值返回
+nextInt(): int	从该 Scanner 中读取下一个标记作为 int 值返回
+nextLong(): long	从该 Scanner 中读取下一个标记作为 long 值返回
+nextFloat(): float	从该 Scanner 中读取下一个标记作为 float 值返回
+nextDouble(): double	从该 Scanner 中读取下一个标记作为 double 值返回
+useDelimiter(pattern: String): Scanner	设置改 Scanner 的分割符，并且返回该 Scanner

图 12-9 Scanner 类包含扫描数据的方法

程序清单 12-15 给出的例子创建了一个 `Scanner` 的实例，并从文件 `scores.txt` 中读取数据。

程序清单 12-15 ReadData.java

```

1 import java.util.Scanner;
2
3 public class ReadData {
4     public static void main(String[] args) throws Exception {
5         // Create a File instance
6         java.io.File file = new java.io.File("scores.txt");
7
8         // Create a Scanner for the file
9         Scanner input = new Scanner(file);
10
11        // Read data from a file
12        while (input.hasNext()) {
13            String firstName = input.next();
14            String mi = input.next();
15            String lastName = input.next();
16            int score = input.nextInt();
17            System.out.println(
18                firstName + " " + mi + " " + lastName + " " + score);
19        }
20
21        // Close the file
22        input.close();
23    }
24 }

```

注意，`new Scanner(String)` 为给定的字符串创建一个 `Scanner`。为创建 `Scanner` 从文件中读取数据，必须使用构造方法 `new File(filename)` 利用 `java.io.File` 类创建 `File` 的一个实例（第 6 行），然后使用 `new Scanner(File)` 为文件创建一个 `Scanner`（第 9 行）。

调用构造方法 `new Scanner(File)` 可能会抛出一个 I/O 异常。因此，`main` 方法在第 4 行声明了 `throws Exception`。

`while` 循环中的每次迭代都从文本文件中读取名字、中间名、姓和分数（第 12 ~ 19 行）。文件在第 22 行关闭。

没有必要关闭输入文件（第 22 行），但这样做是一种释放被文件占用的资源的好方法。可以使用 `try-with-resources` 语法重写该程序。参见：www.cs.armstrong.edu/liang/intro10e/html/ReadDataWithAutoClose.html。

12.11.4 Scanner 如何工作

方法 `nextByte()`、`nextShort()`、`nextInt()`、`nextLong()`、`nextFloat()`、`nextDouble()` 和 `next()` 等都称为标记读取方法（token-reading method），因为它们会读取用分隔符分隔开的标记。默认情况下，分隔符是空格。可以使用 `useDelimiter(String regex)` 方法设置新的分隔符模式。

一个输入方法是如何工作的呢？一个标记读取方法首先跳过任意分隔符（默认情况下是空格），然后读取一个以分隔符结束的标记。然后，对应于 `nextByte()`、`nextShort()`、`nextInt()`、`nextLong()`、`nextFloat()` 和 `nextDouble()`，这个标记就分别被自动地转换为一个 `byte`、`short`、`int`、`long`、`float` 或 `double` 型的值。对于 `next()` 方法而言是无须做转换的。如果标记和期望的类型不匹配，就会抛出一个运行异常 `java.util.InputMismatchException`。

方法 `next()` 和 `nextLine()` 都会读取一个字符串。`next()` 方法读取一个由分隔符分隔的字符串，但是 `nextLine()` 读取一个以换行符结束的行。

 **注意：**行分隔符字符串是由系统定义的，在 Windows 平台上是 `\r\n`，而在 UNIX 平台上是 `\n`。为了得到特定平台上的行分隔符，使用

```
String lineSeparator = System.getProperty("line.separator");
```

如果从键盘输入，每行就以回车键 (Enter key) 结束，它对应于 `\n` 字符。

标记读取方法不能读取标记后面的分隔符。如果在标记读取方法之后调用 `nextLine()`，该方法读取从这个分隔符开始，到这行的行分隔符结束的字符。这个行分隔符也被读取，但是它不是 `nextLine()` 返回的字符串部分。

假设一个名为 `test.txt` 的文本文件包含一行

```
34 567
```

在执行完下面的代码之后，

```
Scanner input = new Scanner(new File("test.txt"));
int intValue = input.nextInt();
String line = input.nextLine();
```

`intValue` 的值为 34，而 `line` 包含的字符是 ' '、'5'、'6'、'7'。

如果输入是从键盘键入，那会发生什么呢？假设为下面的代码输入 34，然后按回车键，接着输入 567，然后再按回车键：

```
Scanner input = new Scanner(System.in);
int intValue = input.nextInt();
String line = input.nextLine();
```

将会得到 `intValue` 值是 34，而 `line` 中是一个空的字符串。这是为什么呢？原因如下。标记读取方法 `nextInt()` 读取 34，然后在分隔符处停止，这里的分隔符是行分隔符 (回车键)。`nextLine()` 方法会在读取行分隔符之后结束，然后返回在行分隔符之前的字符串。因为在行分隔符之前没有字符，所以 `line` 是空的。

可以使用 `Scanner` 类从文件或者键盘读取数据。也可以使用 `Scanner` 类从一个字符串扫描数据。例如，下面代码

```
Scanner input = new Scanner("13 14");
int sum = input.nextInt() + input.nextInt();
System.out.println("Sum is " + sum);
```

显示

```
The sum is 27
```

12.11.5 示例学习：替换文本

假设要编写一个名为 `ReplaceText` 的程序，用一个新字符串替换文本文件中所有出现某个字符串的地方。文件名和字符串都作为命令行参数传递，如下所示：

```
java ReplaceText sourceFile targetFile oldString newString
```

例如，调用

```
java ReplaceText FormatString.java t.txt StringBuilder StringBuffer
```

就会用 `StringBuffer` 替换 `FormatString.java` 中所有出现的 `StringBuilder`，然后将新文件保存在 `t.txt` 中。

程序清单 12-16 给出该程序。程序检查传给 main 方法的参数个数 (第 7~11 行), 检查源文件和目标文件是否存在 (第 14~25 行), 为源文件创建一个 Scanner (第 29 行), 为目标文件创建一个 PrintWriter (第 30 行), 然后重复从源文件读入一行 (第 33 行), 替换文本 (第 34 行), 向目标文件中写入新的一行 (第 35 行)。

程序清单 12-16 ReplaceText.java

```
1 import java.io.*;
2 import java.util.*;
3
4 public class ReplaceText {
5     public static void main(String[] args) throws Exception {
6         // Check command line parameter usage
7         if (args.length != 4) {
8             System.out.println(
9                 "Usage: java ReplaceText sourceFile targetFile oldStr newStr");
10            System.exit(1);
11        }
12
13        // Check if source file exists
14        File sourceFile = new File(args[0]);
15        if (!sourceFile.exists()) {
16            System.out.println("Source file " + args[0] + " does not exist");
17            System.exit(2);
18        }
19
20        // Check if target file exists
21        File targetFile = new File(args[1]);
22        if (targetFile.exists()) {
23            System.out.println("Target file " + args[1] + " already exists");
24            System.exit(3);
25        }
26
27        try {
28            // Create input and output files
29            Scanner input = new Scanner(sourceFile);
30            PrintWriter output = new PrintWriter(targetFile);
31        } {
32            while (input.hasNext()) {
33                String s1 = input.nextLine();
34                String s2 = s1.replaceAll(args[2], args[3]);
35                output.println(s2);
36            }
37        }
38    }
39 }
```

通常情况下, 程序会在一个文件被复制后终止。但是, 如果命令行参数没有正确使用 (第 7~11 行), 或者如果源文件不存在 (第 14~18 行), 或者目标文件已经存在 (第 22~25 行), 程序将异常终止。退出的状态代码 1、2 以及 3 用于表明这些异常的终止 (第 10、17、24 行)。

复习题

- 12.30 如何创建一个 PrintWriter 以向文件写数据? 在程序清单 12-13 中, 为什么要在 main 方法中声明 throws Exception? 在程序清单 12-13 中, 如果不调用 close() 方法, 将会发生什么?
- 12.31 给出下面的程序执行之后, 文件 temp.txt 的内容。

```
public class Test {
    public static void main(String[] args) throws Exception {
        java.io.PrintWriter output = new
```

```

        java.io.PrintWriter("temp.txt");
        output.printf("amount is %f %e\r\n", 32.32, 32.32);
        output.printf("amount is %5.4f %5.4e\r\n", 32.32, 32.32);
        output.printf("%6b\r\n", (1 > 2));
        output.printf("%6s\r\n", "Java");
        output.close();
    }
}

```

- 12.32 使用 try-with-resource 语法重写前一题中的代码。
- 12.33 如何创建一个 Scanner 从文件读数据？在程序清单 12-15 中，为什么要在 main 方法中声明 throws Exception？在程序清单 12-15 中，如果不调用 close() 方法，将会发生什么？
- 12.34 如果试图对一个不存在的文件创建 Scanner，将会发生什么情况？如果试图对一个已经存在的文件创建 PrintWriter，会发生什么情况？
- 12.35 是否所有平台上的行分隔符都是一样的？Windows 平台上的行分隔符是什么？
- 12.36 假设输入 45 57.8 789，然后点击回车键。显示执行完下面的代码之后变量的内容。

```

Scanner input = new Scanner(System.in);
int intValue = input.nextInt();
double doubleValue = input.nextDouble();
String line = input.nextLine();

```

- 12.37 假设输入 45、回车键、57.8、回车键、789、回车键。显示执行完下面的代码之后变量的内容。

```

Scanner input = new Scanner(System.in);
int intValue = input.nextInt();
double doubleValue = input.nextDouble();
String line = input.nextLine();

```

12.12 从 Web 上读取数据

要点提示：如同从电脑中的文件中读取数据一样，也可以从 Web 上的文件中读取数据。

除开从电脑中的本地文件或者文件服务器中读取数据，如果知道 Web 上文件的 URL (Uniform Resource Locator, 统一资源定位器, 即为 Web 上的文件提供唯一的地址), 也可以从 Web 上访问数据。例如, www.google.com/index.html 是位于 Google Web 服务器上的文件 index.html 的 URL。当在一个 Web 浏览器中输入 URL 之后, Web 服务器将数据传送给浏览器, 浏览器则将数据渲染成图形。图 12-10 演示了这个过程是如何工作的。

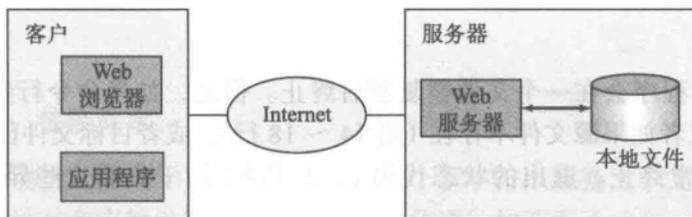


图 12-10 客户端从一个 Web 服务器中获取文件

为了读取一个文件, 首先要使用 java.net.URL 类的这个构造方法, 为该文件创建一个 URL 对象。

```
public URL(String spec) throws MalformedURLException
```

例如, 下面给出的语句为 http://www.google.com/index.html 创建一个 URL 对象。

```

1 try {
2     URL url = new URL("http://www.google.com/index.html");
3 }
4 catch (MalformedURLException ex) {
5     ex.printStackTrace();
6 }

```

如果 URL 字符串出现语法错误的话，将会有有一个 `MalformedURLException` 被抛出。例如，URL 字符串 “`http:www.google.com/index.html`” 将会引起 `MalformedURLException` 运行错误，因为在冒号 (:) 之后要求带有双斜杠 (//)。注意，要让 URL 类来识别一个有效的 URL，前缀 `http://` 是必需的。如果将第 2 行替换为下面代码，将会出错：

```
URL url = new URL("www.google.com/index.html");
```

创建一个 URL 对象后，可以使用 URL 类中定义的 `openStream()` 方法来打开输入流和用输入流创建如下 `Scanner` 对象。

```
Scanner input = new Scanner(url.openStream());
```

现在可以从输入流中读取数据了，如同从本地文件中读取一样。程序清单 12-17 中的示例提示用户输入一个 URL，然后显示文件的大小。

程序清单 12-17 ReadFileFromURL.java

```

1 import java.util.Scanner;
2
3 public class ReadFileFromURL {
4     public static void main(String[] args) {
5         System.out.print("Enter a URL: ");
6         String urlString = new Scanner(System.in).next();
7
8         try {
9             java.net.URL url = new java.net.URL(urlString);
10            int count = 0;
11            Scanner input = new Scanner(url.openStream());
12            while (input.hasNext()) {
13                String line = input.nextLine();
14                count += line.length();
15            }
16
17            System.out.println("The file size is " + count + " characters");
18        }
19        catch (java.net.MalformedURLException ex) {
20            System.out.println("Invalid URL");
21        }
22        catch (java.io.IOException ex) {
23            System.out.println("I/O Errors: no such file");
24        }
25    }
26 }

```

```

Enter a URL: http://cs.armstrong.edu/liang/data/Lincoln.txt 
The file size is 1469 characters

```

```

Enter a URL: http://www.yahoo.com 
The file size is 190006 characters

```

程序提示用户输入一个 URL 字符串 (第 6 行)，然后创建一个 URL 对象 (第 9 行)。如果 URL 的表示没有正确给出，则构造方法将抛出一个 `java.net.MalformedURLException` (第 19 行)。

程序为 URL 代表的输入流创建一个 Scanner 对象 (第 11 行)。如果正确给出了 URL 的表示但是不存在, 将抛出一个 IOException (第 22 行)。例如, `http:// google.com/index1.html` 使用了合适的形式, 但是 URL 本身不存在。如果该 URL 用于这个程序的话, 一个 IOException 将抛出。

复习题

12.38 如何为从一个 URL 读取的文本创建一个 Scanner 对象?

12.13 示例学习: Web 爬虫

要点提示: 本节的示例学习开发一个程序, 可以跟随超链接来遍历 Web。

World Wide Web, 缩写为 WWW、W3 或者 Web, 是一个因特网上的相互链接的超文本文档。使用 Web 浏览器, 可以查看一个文档, 以及跟随超链接查看其他文档。在本示例学习中, 我们将开发一个程序, 可以跟随超链接来自动遍历 Web。这类程序通常称为 Web 爬虫。为简化起见, 我们的程序跟随以 `http://` 开始的超链接。图 12-11 给出了一个遍历 Web 的例子。从一个包含了三个分别名为 URL1、URL2、URL3 的网址的页面开始, 跟随 URL2 将到达一个包含两个名为 URL21 和 URL22 的网址的页面, 跟随 URL3 将到达一个包含名为 URL31、URL32、URL33、URL34 的网址的页面。可以继续跟随着新的链接对 Web 进行遍历。如你所见, 这个过程可以一直进行下去, 但是我们将在遍历了 100 个页面后退出程序。

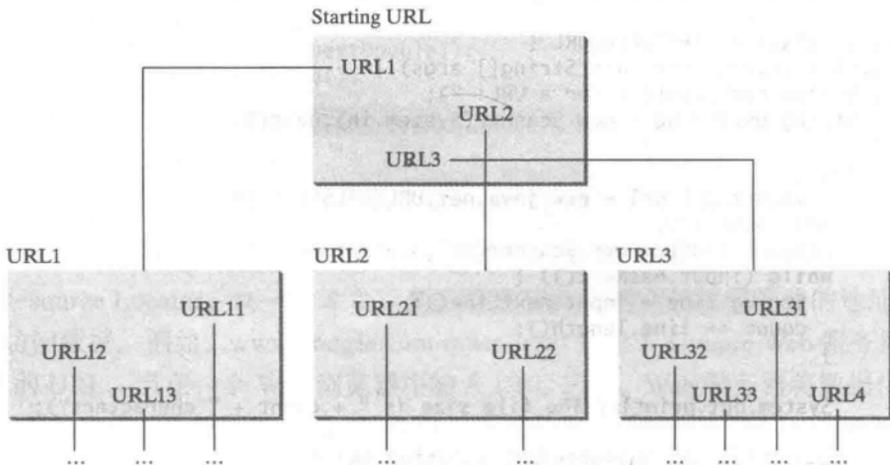


图 12-11 客户程序从一个 Web 服务器上获取文件

程序跟随 URL 来遍历 Web。为了保证每个 URL 只被遍历一次, 程序包含两个网址的列表。一个列表保存将被遍历的网址, 另外一个保存已经被遍历的网址。程序的算法如下描述:

```

将起始 URL 添加到名为 listOfPendingURLs 的列表中;
当 listOfPendingURLs 不为空并且 listOfTraversedURLs 的长度 <=100{
    从 listOfPendingURLs 移除一个 URL;
    如果该 URL 不在 listOfTraversedURLs 中 {
        将其添加到 listOfTraversedURLs 中;
        显示该 URL;
        读取该 URL 的页面, 并且对该页面中包含的每个 URL 进行如下操作 {
            如果不在 listOfTraversedURLs 中, 则将其添加到 listOfPendingURLs 中;
        }
    }
}

```

程序清单 12-18 给出了实现该算法的程序。

程序清单 12-18 WebCrawler.java

```
1 import java.util.Scanner;
2 import java.util.ArrayList;
3
4 public class WebCrawler {
5     public static void main(String[] args) {
6         java.util.Scanner input = new java.util.Scanner(System.in);
7         System.out.print("Enter a URL: ");
8         String url = input.nextLine();
9         crawler(url); // Traverse the Web from the a starting url
10    }
11
12    public static void crawler(String startingURL) {
13        ArrayList<String> listOfPendingURLs = new ArrayList<>();
14        ArrayList<String> listOfTraversedURLs = new ArrayList<>();
15
16        listOfPendingURLs.add(startingURL);
17        while (!listOfPendingURLs.isEmpty() &&
18            listOfTraversedURLs.size() <= 100) {
19            String urlString = listOfPendingURLs.remove(0);
20            if (!listOfTraversedURLs.contains(urlString)) {
21                listOfTraversedURLs.add(urlString);
22                System.out.println("Crawl " + urlString);
23
24                for (String s: getSubURLs(urlString)) {
25                    if (!listOfTraversedURLs.contains(s))
26                        listOfPendingURLs.add(s);
27                }
28            }
29        }
30    }
31
32    public static ArrayList<String> getSubURLs(String urlString) {
33        ArrayList<String> list = new ArrayList<>();
34
35        try {
36            java.net.URL url = new java.net.URL(urlString);
37            Scanner input = new Scanner(url.openStream());
38            int current = 0;
39            while (input.hasNext()) {
40                String line = input.nextLine();
41                current = line.indexOf("http:", current);
42                while (current > 0) {
43                    int endIndex = line.indexOf("\"", current);
44                    if (endIndex > 0) { // Ensure that a correct URL is found
45                        list.add(line.substring(current, endIndex));
46                        current = line.indexOf("http:", endIndex);
47                    }
48                    else
49                        current = -1;
50                }
51            }
52        }
53        catch (Exception ex) {
54            System.out.println("Error: " + ex.getMessage());
55        }
56
57        return list;
58    }
59 }
```

```

Enter a URL: http://cs.armstrong.edu/liang 
Enter a URL: http://www.cs.armstrong.edu/liang
Crawl http://www.cs.armstrong.edu/liang
Crawl http://www.cs.armstrong.edu
Crawl http://www.armstrong.edu
Crawl http://www.pearsonhighered.com/liang
...

```

程序提示用户输入一个起始 URL (第 7 ~ 8 行), 然后调用 `crawler(url)` 方法来遍历 Web (第 9 行)。

`crawler(url)` 方法将起始 url 添加到 `listOfPendingURLs` (第 16 行), 然后通过一个 `while` 循环重复处理 `listOfPendingURLs` 中的每个 URL (第 17 ~ 29 行)。程序将列表的第一个 URL 去除 (第 19 行), 如果该 URL 没有被处理过, 则对其进行处理 (第 20 ~ 28 行)。处理每个 URL 时, 程序首先将 URL 添加到 `listOfTraversedURLs` 中 (第 21 行)。该列表存储了所有处理过的 URL。`getSubURLs(url)` 方法为每个给定的 URL 返回一个 URL 列表 (第 24 行)。程序使用一个 `foreach` 循环, 将页面中的每个不存在于 `listOfTraversedURLs` 中的 URL 添加到 `listOfPendingURLs` 中 (第 24 ~ 26 行)。

`getSubURLs(url)` 方法从 Web 页面中读取每行 (第 40 行), 并且寻找该行中的 URL (第 41 行)。注意到正确的 URL 不能包含分行符, 因此只要在 Web 页面中的一行文本中寻找 URL 就足够了。为了简化起见, 假设一个 URL 以引号 " 结束 (第 43 行)。方法获取一个 URL 并且将其添加到列表中 (第 45 行)。一行中可能包含多个 URL。方法接着继续寻找下一个 URL (第 46 行)。如果在该行中没有发现 URL, `current` 设为 -1 (第 49 行)。页面中包含的 URL 以一个列表的形式返回 (第 57 行)。

当遍历的 URL 数目达到 100 的时候, 程序结束 (第 18 行)。

这是一个遍历 Web 的简单程序。后面将学习到让该程序更加有效和健壮的技术。

复习题

12.39 在一个 URL 添加到 `listOfPendingURLs` 之前, 第 25 行检查它是否被遍历过了。

`listOfPendingURLs` 是否可能包含重复的 URLs 呢? 如果是, 给出一个例子。

关键术语

absolute file name (绝对文件名)

chained exception (链式异常)

checked exception (必检异常)

declare exception (声明异常)

directory path (目录路径)

exception (异常)

exception propagation (异常传播)

relative file name (相对文件名)

throw exception (抛出异常)

unchecked exception (免检异常)

本章小结

1. 异常处理使一个方法能够抛出一个异常给它的调用者。
2. Java 异常是扩展自 `java.lang.Throwable` 的类的实例。Java 提供大量预定义的异常类, 例如, `Error`、`Exception`、`RuntimeException`、`ClassNotFoundException`、`NullPointerException` 和 `ArithmeticException`。也可以通过扩展 `Exception` 类来定义自己的异常类。
3. 异常发生在一个方法的执行过程中。`RuntimeException` 和 `Error` 都是免检异常, 所有其他的异常都是必检的。

4. 当声明一个方法时, 如果这个方法可能抛出一个必检异常, 则必须进行声明, 从而告诉编译器可能会出现什么错误。
5. 声明异常的关键字是 `throws`, 而抛出异常的关键字是 `throw`。
6. 如果调用声明了必检异常的方法, 必须将该方法调用放在 `try` 语句中。在方法执行过程中出现异常时, `catch` 块会捕获并处理异常。
7. 如果一个异常没有被当前方法捕获, 则该异常被传给调用者。这个过程不断重复直到异常被捕获或者传递给 `main` 方法。
8. 可以从一个共同的父类派生出各种不同的异常类。如果一个 `catch` 块捕获到父类的异常对象, 它也能捕捉这个父类的子类的所有异常对象。
9. 在 `catch` 块中, 异常的指定顺序是非常重要的。如果在指定一个类的异常对象之前, 指定了这个异常类的父类的异常对象, 就会导致一个编译错误。
10. 当方法中发生异常时, 如果异常没有被捕获, 方法将会立刻退出。如果想在方法退出前执行一些任务, 可以在方法中捕获这个异常, 然后再重新抛给它的调用者。
11. 任何情况下都会执行 `finally` 块中的代码, 不管 `try` 块中是否出现了异常, 或者出现异常后是否捕获了该异常。
12. 异常处理将错误处理代码从正常的程序设计任务中分离出来, 这样, 就会使得程序更易于阅读和修改。
13. 不应该使用异常处理代替简单的测试。应该尽可能地使用 `if` 语句来进行简单的测试, 将异常处理留作处理那些无法用 `if` 语句处理的场景。
14. `File` 类用于获得文件属性和操作文件。它不包含创建文件的方法, 或者从 / 向文件读 / 写数据。
15. 可以使用 `Scanner` 来从一个文本文件中读取字符串和基本数据类型的值, 使用 `PrintWriter` 来创建一个文件并且将数据写入文本文件。
16. 可以使用 `URL` 类来读取一个 Web 上的文件内容。

测试题

在线回答本章的测试题, 地址为: www.cs.armstrong.edu/liang/intro10e/quiz.html。

编程练习题

12.2 ~ 12.9 节

- *12.1 (`NumberFormatException` 异常) 程序清单 7-9 是一个简单的命令行计算器。注意, 如果某个操作数是非数值的, 程序就会中止。编写一个程序, 利用异常处理器来处理非数值操作数; 然后编写另一个不使用异常处理器的程序, 达到相同的目的。程序在退出之前应该显示一条消息, 通知用户发生了操作数类型错误 (参见图 12-12)。
- *12.2 (`InputMismatchException` 异常) 编写一个程序, 提示用户读取两个整数, 然后显示它们的和。程序应该在输入不正确时提示用户再次读取数字。
- *12.3 (`ArrayIndexOutOfBoundsException` 异常) 编写一个满足下面要求的程序:
 - 创建一个由 100 个随机选取的整数构成的数组。
 - 提示用户输入数组的下标, 然后显示对应的元素值。如果指定的下标越界, 就显示消息 `Out of Bounds`。

```

c:\exercise>java Exercise12_01 4 + 5
4 + 5 = 9

c:\exercise>java Exercise12_01 4 - 5
4 - 5 = -1

c:\exercise>java Exercise12_01 4x - 5
Wrong Input: 4x

c:\exercise>

```

图 12-12 程序执行算术运算并检查输入错误

- *12.4 (IllegalArgumentException 异常) 修改程序清单 10-2 中的 Loan 类, 如果贷款总额、利率、年数小于或等于零, 则抛出 IllegalArgumentException 异常。
- *12.5 (IllegalTriangleException 异常) 编程练习题 11.1 定义了带三条边的 Triangle 类。在三角形中, 任意两边之和总大于第三边, 三角形类 Triangle 必须遵从这一规则。创建一个 IllegalTriangleException 类, 然后修改 Triangle 类的构造方法, 如果创建的三角形的边违反了这一规则, 抛出一个 IllegalTriangleException 对象, 如下所示:


```
/** Construct a triangle with the specified sides */
public Triangle(double side1, double side2, double side3)
    throws IllegalTriangleException {
    // Implement it
}
```
- *12.6 (NumberFormatException 异常) 程序清单 6-8 实现了 hexToDec (String hexString) 方法, 它将一个十六进制字符串转换为一个十进制数。实现这个 hexToDec 方法, 在字符串不是一个十六进制字符串时抛出 NumberFormatException 异常。
- *12.7 (NumberFormatException 异常) 编写 bin2Dec(String binaryString) 方法, 将一个二进制字符串转换为一个十进制数。实现 bin2Dec 方法, 在字符串不是一个二进制字符串时抛出 NumberFormatException 异常。
- *12.8 (HexFormatException 异常) 编程练习题 12.6 实现 hex2Dec 方法, 在字符串不是一个十六进制字符串时抛出 NumberFormatException 异常。定义一个名为 HexFormatException 的自定义异常。实现 hex2Dec 方法, 在字符串不是一个十六进制字符串时抛出 HexFormatException 异常。
- *12.9 (BinaryFormatException 异常) 编程练习题 12.7 实现 bin2Dec 方法, 在字符串不是一个二进制字符串时抛出 BinaryFormatException 异常。定义一个名为 BinaryFormatException 的自定义异常。实现 bin2Dec 方法, 在字符串不是一个二进制字符串时抛出 BinaryFormatException 异常。
- *12.10 (OutOfMemoryError 错误) 编写一个程序, 它能导致 JVM 抛出一个 OutOfMemoryError, 然后捕获和处理这个错误。

12.10 ~ 12.12 节

- **12.11 (删除文本) 编写一个程序, 从一个文本文件中删掉所有指定的某个字符串。例如, 调用

```
java Exercise12_11 John filename
```

从指定文件中删掉字符串 John。程序从命令行获得参数。

- **12.12 (重新格式化 Java 源代码) 编写一个程序, 将 Java 源代码的次行块风格转换成行尾块风格。例如, 图 a 中的 Java 源代码使用的是次行块风格。程序将它转换成图 b 中所示的行尾块形式。

```
public class Test
{
    public static void main(String[] args)
    {
        // Some statements
    }
}
```

a) 次行块风格

```
public class Test {
    public static void main(String[] args) {
        // Some statements
    }
}
```

b) 行尾块风格

程序可以从命令行调用, 以 Java 源代码文件作为其参数。它会将这个 Java 源代码变成新的格式。例如, 下面的命令将 Java 源代码文件 Test.java 转变成行尾块风格:

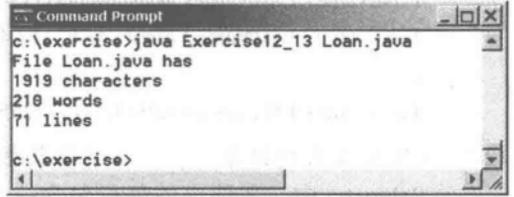
```
java Exercise12_12 Test.java
```

- *12.13 (统计一个文件中的字符数、单词数和行数) 编写一个程序, 统计一个文件中的字符数、单词数

以及行数。单词由空格符分隔，文件名应该作为命令行参数被传递，如图 12-13 所示。

*12.14 (处理文本文件中的分数) 假定一个文本文件中包含未指定个数的分数，用空格分开。编写一个程序，提示用户输入文件，然后从文件中读入分数，并且显示它们的和以及平均值。

*12.15 (写/读数据) 编写一个程序，如果名为 Exercise12_15.txt 的文件不存在，则创建该文件。使用文本 I/O 将随机产生的 100 个整数写入文件，文件中的整数由空格分开。从文件中读回数据并以升序显示数据。



```

c:\exercise>java Exercise12_13 Loan.java
File Loan.java has
1919 characters
218 words
71 lines
c:\exercise>

```

图 12-13 程序显示给定文件中的字符数、单词数和行数

**12.16 (替换文本) 程序清单 12-16 给出一个程序，替换源文件中的文本，然后将这个变化存储到一个新文件中。改写程序，将这个变化存储到原始文件中。例如：调用

```
java Exercise12_16 file oldString newString
```

用 newString 代替源文件中的 oldString。

***12.17 (游戏：刽子手) 改写编程练习题 7.35。程序读取存储在一个名为 hangman.txt 的文本文件中的单词。这些单词用空格分隔。

**12.18 (添加包语句) 假设在目录 chapter1, chapter2, ..., chapter34 下面有 Java 源文件。编写一个程序，对在目录 chapteri 下面的 Java 源文件的第一行添加语句 “package chapteri;”。假定 chapter1, chapter2, ..., chapter34 在根目录 srcRootDirectory 下面。根目录和 chapteri 目录可能包含其他目录和文件。使用下面命令来运行程序：

```
java Exercise12_18 srcRootDirectory
```

*12.19 (统计单词) 编写一个程序，统计 Abraham Lincoln 总统的 Gettysburg 演讲中的单词数，该演讲的网址为 <http://cs.armstrong.edu/liang/data/Lincoln.txt>。

**12.20 (删除包语句) 假设在目录 chapter1, chapter2, ..., chapter34 下面有 Java 源文件。编写一个程序，对在目录 chapteri 下面的 Java 源文件删除其第一行包语句 “package chapteri;”。假定 chapter1, chapter2, ..., chapter34 在根目录 srcRootDirectory 下面。根目录和 chapteri 目录可能包含其他目录和文件。使用下面命令来运行程序：

```
java Exercise12_20 srcRootDirectory
```

*12.21 (数据排好了吗?) 编写一个程序，从文件 SortedStrings.txt 中读取字符串，并且给出报告，文件中的字符串是否以升序的方式进行存储。如果文件中的字符串没有排好序，显示没有遵循排序的前面两个字符串。

**12.22 (替换文本) 修改编程练习题 12.16，使用下面的命令用一个新字符串替换某个特定目录下所有文件中的一个字符串：

```
java Exercise12_22 dir oldString newString
```

**12.23 (处理 Web 上的文本文件中的分数) 假定 Web 上的一个文本文件 <http://cs.armstrong.edu/liang/data/Scores.txt> 中包含了不确定数目的成绩。编写一个程序，从该文件中读取分数，并且显示它们的总数以及平均数。分数使用空格进行分隔。

*12.24 (创建大的数据集) 创建一个具有 1000 行的数据文件。文件中的每行包含了一个教职员工的姓、名、级别以及薪水。第 i 行的教职员工的姓和名为 FirstName i 和 LastName i 。级别随机产生为 assistant (助理)、associate (副) 以及 full (正)。薪水为随机产生的数字，并且小数点后保留两位数字。对于助理教授而言，薪水应该在 50 000 到 80 000 的范围内，对于副教授为 60 000 到

110 000, 对于正教授为 75 000 到 130 000。保存文件为 Salary.txt。下面是一些示例数据:

```
FirstName1 LastName1 assistant 60055.95
FirstName2 LastName2 associate 81112.45
...
FirstName1000 LastName1000 full 92255.21
```

*12.25 (处理大的数据集) 一个大学将其教职员工的薪水发布在 <http://cs.armstrong.edu/liang/data/Salary.txt> 中。文件中的每行包含一个教职员工的姓、名、级别以及薪水(见编程练习题 12.24)。编写一个程序, 分别显示助理教授、副教授、正教授, 所有教职员工各个类别的总薪水, 以及上述类别的平均薪水。

**12.26 (创建一个目录) 编写一个程序, 提示用户输入一个目录名称, 然后使用 File 的 mkdirs 方法创建相应的目录。如果目录创建成功则显示 “Directory created successfully”, 如果目录已经存在, 则显示 “Directory already exists”。

**12.27 (替换文本) 假定在某个目录下面的多个文件中包含了单词 Exercise i_j , 其中 i 和 j 是数字。编写一个程序, 如果 i 是个位数, 则在 i 前面插入一个 0, 同理如果 j 是个位数, 则在 j 前面插入一个 0。例如, 文件中的单词 Exercise2_1 将被替换为 Exercise02_01。Java 中, 当从命令行传递符号 * 的时候, 指代该目录下的所有文件(参见附录 III.V)。使用下面的命令来运行程序。

```
java Exercise12_27 *
```

**12.28 (更改文件名) 假定在某个目录下面有多个文件, 命名为 Exercise i_j , 其中 i 和 j 是数字。编写一个程序, 如果 i 是个位数, 则在 i 前面插入一个 0。例如, 目录中的文件 Exercise2_1 将被改名为 Exercise02_1。Java 中, 当从命令行传递符号 * 的时候, 指代该目录下的所有文件(参见附录 III.V)。使用下面的命令来运行程序。

```
java Exercise12_28 *
```

**12.29 (更改文件名) 假定在某个目录下面有多个文件, 命名为 Exercise i_j , 其中 i 和 j 是数字。编写一个程序, 如果 j 是个位数, 则在 j 前面插入一个 0。例如, 目录中的文件 Exercise2_1 将被改名为 Exercise2_01。Java 中, 当从命令行传递符号 * 的时候, 指代该目录下的所有文件(参见附录 III.V)。使用下面的命令来运行程序。

```
java Exercise12_29 *
```

**12.30 (每个字母出现的次数) 编写一个程序, 提示用户输入一个文件名, 然后显示该文件中每个字母出现的次数。字母是大小写敏感的。下面是一个运行示例:

```
Enter a filename: Lincoln.txt 
Number of A's: 56
Number of B's: 134
...
Number of Z's: 9
```

*12.31 (小孩名字流行度排名) 从 2001 年到 2010 年的小孩取名的流行度排名可以从 www.ssa.gov/oact/babynames 下载并保存在 babynameranking2001.txt, babynameranking2002.txt, ..., babynameranking2010.txt。每个文件包含了一千行。每行包含一个排名, 一个男孩的名字, 取该名字的数目, 一个女孩子的名字, 取该名字的数目。例如, 文件 babynameranking2010.txt 的前面两行如下所示:

```
1      Jacob      21,875      Isabella      22,731
2      Ethan      17,866      Sophia        20,477
```

因此，男孩名 Jacob 和女孩名 Isabella 排第一位，男孩名 Ethan 和女孩名 Sophia 排名第二。有 21 875 名男孩取名 Jacob，22 731 名女孩取名 Isabella。编写一个程序，提示用户输入年份、性别，接着输入名字，程序可以显示该年份该名字的排名。这里是一个运行示例：

```
Enter the year: 2010 ↵ Enter
Enter the gender: M ↵ Enter
Enter the name: Javier ↵ Enter
Javier is ranked #190 in year 2010
```

```
Enter the year: 2010 ↵ Enter
Enter the gender: F ↵ Enter
Enter the name: ABC ↵ Enter
The name ABC is not ranked in year 2010
```

*12.32 (排名总结) 编写一个程序，使用编程练习 12.31 中所描述的文件，显示前 5 位的女孩和男孩名字的排名总结表格：

Year	Rank 1	Rank 2	Rank 3	Rank 4	Rank 5	Rank 1	Rank 2	Rank 3	Rank 4	Rank 5
2010	Isabella	Sophia	Emma	Olivia	Ava	Jacob	Ethan	Michael	Jayden	William
2009	Isabella	Emma	Olivia	Sophia	Ava	Jacob	Ethan	Michael	Alexander	William
...										
2001	Emily	Madison	Hannah	Ashley	Alexis	Jacob	Michael	Matthew	Joshua	Christopher

**12.33 (搜索 Web) 修改程序清单 12-18，从网址 <http://cs.armstrong.edu/liang> 开始搜索单词 Computer Programming，一旦搜索到，程序终止。显示包含了单词的页面的 URL 地址。

抽象类和接口

教学目标

- 设计和使用抽象类 (13.2 节)。
- 使用抽象的 `Number` 类来将数值包装类、`BigInteger` 以及 `BigDecimal` 类通用化 (13.3 节)。
- 使用 `Calendar` 类和 `GregorianCalendar` 类处理日历 (13.4 节)。
- 使用接口指定对象共有的行动 (13.5 节)。
- 定义接口以及实现接口的类 (13.5 节)。
- 使用 `Comparable` 接口定义自然的顺序 (13.6 节)。
- 使用 `Cloneable` 接口使对象成为可克隆的 (13.7 节)。
- 探究具体类、抽象类和接口的相同点与不同点 (13.8 节)。
- 设计 `Rational` 类来处理有理数 (13.9 节)。
- 遵循类设计的准则来设计类 (13.10 节)。

13.1 引言

 **要点提示：**父类中定义了相关子类中的共同行为。接口可以用于定义类的共同行为 (包括非相关的类)。

可以使用 `java.util.Arrays.sort` 方法来对数值和字符串进行排序。那么可以应用同样的 `sort` 方法对一个几何对象的数组进行排序吗？为了编写这样的代码，必须要了解接口。接口是为了定义多个类 (包括非相关的类) 的共同行为。在讨论接口之前，我们介绍一个非常接近的相关主题：抽象类。

13.2 抽象类

 **要点提示：**抽象类不可以用于创建对象。抽象类可以包含抽象方法，这些方法将在具体的子类中实现。

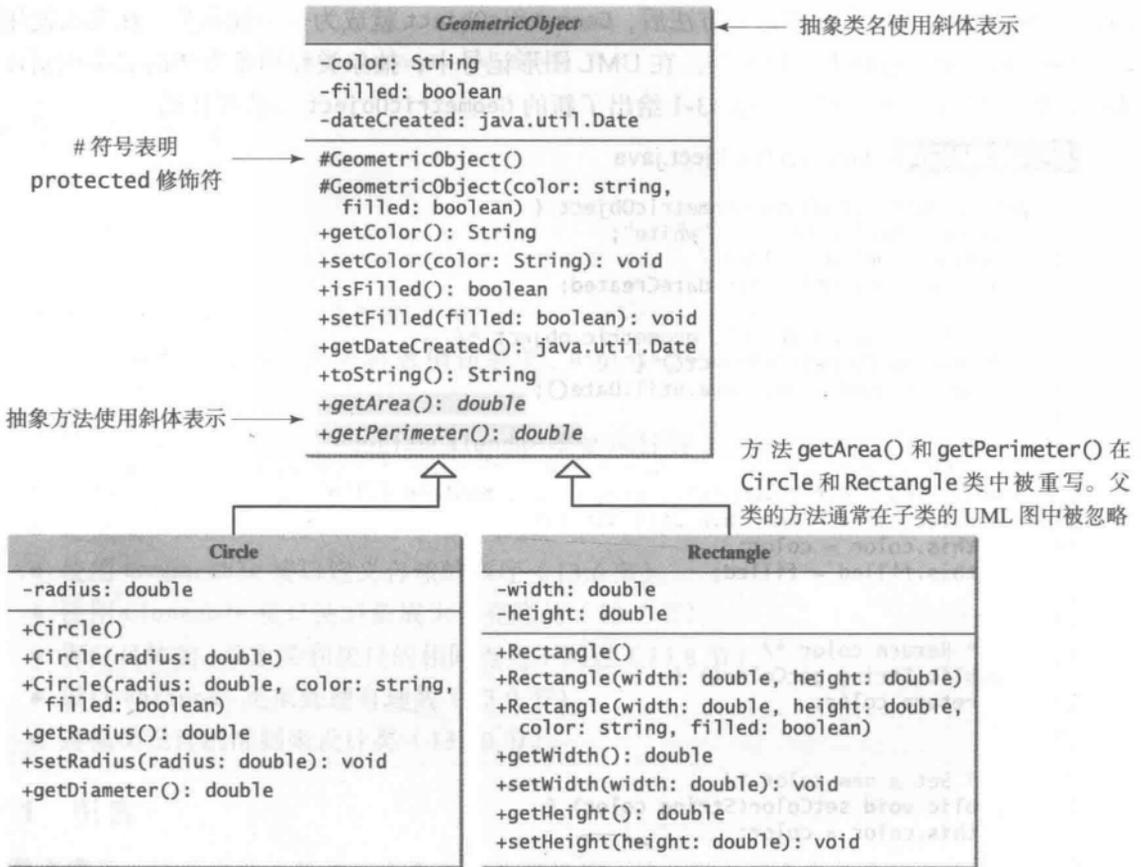
在继承的层次结构中，每个新子类都使类变得越来越明确和具体。如果从一个子类追溯到父类，类就会变得更通用、更加不明确。类的设计应该确保父类包含它的子类的共同特征。有时候，一个父类设计得非常抽象，以至于它都没有任何具体的实例。这样的类称为抽象类 (abstract class)。

在第 11 章中，`GeometricObject` 类定义成 `Circle` 类和 `Rectangle` 类的父类。`GeometricObject` 类模拟了几何对象的共同特征。`Circle` 类和 `Rectangle` 类分别包含计算圆和矩形的面积和周长的方法 `getArea()` 和 `getPerimeter()`。因为可以计算所有几何对象的面积和周长，所以最好在 `GeometricObject` 类中定义 `getArea()` 和 `getPerimeter()` 方法。但是，这些方法不能在 `GeometricObject` 类中实现，因为它们实现取决于几何对象的具体类型。这样的方法称为抽象方法 (abstract method)，在方法头中使用 `abstract` 修饰符表示。在

GeometricObject 类中定义了这些方法后, GeometricObject 就成为一个抽象类。在类头使用 `abstract` 修饰符表示该类为抽象类。在 UML 图形记号中, 抽象类和抽象方法的名字用斜体表示, 如图 13-1 所示。程序清单 13-1 给出了新的 GeometricObject 类的源代码。

程序清单 13-1 GeometricObject.java

```
1 public abstract class GeometricObject {
2     private String color = "white";
3     private boolean filled;
4     private java.util.Date dateCreated;
5
6     /** Construct a default geometric object */
7     protected GeometricObject() {
8         dateCreated = new java.util.Date();
9     }
10
11    /** Construct a geometric object with color and filled value */
12    protected GeometricObject(String color, boolean filled) {
13        dateCreated = new java.util.Date();
14        this.color = color;
15        this.filled = filled;
16    }
17
18    /** Return color */
19    public String getColor() {
20        return color;
21    }
22
23    /** Set a new color */
24    public void setColor(String color) {
25        this.color = color;
26    }
27
28    /** Return filled. Since filled is boolean,
29     * the get method is named isFilled */
30    public boolean isFilled() {
31        return filled;
32    }
33
34    /** Set a new filled */
35    public void setFilled(boolean filled) {
36        this.filled = filled;
37    }
38
39    /** Get dateCreated */
40    public java.util.Date getDateCreated() {
41        return dateCreated;
42    }
43
44    @Override
45    public String toString() {
46        return "created on " + dateCreated + "\ncolor: " + color +
47            " and filled: " + filled;
48    }
49
50    /** Abstract method getArea */
51    public abstract double getArea();
52
53    /** Abstract method getPerimeter */
54    public abstract double getPerimeter();
55 }
```

图 13-1 新的 `GeometricObject` 类包含抽象方法

抽象类和常规类很像，但是不能使用 `new` 操作符创建它的实例。抽象方法只有定义而没有实现。它的实现由子类提供。一个包含抽象方法的类必须声明为抽象类。

抽象类的构造方法定义为 `protected`，因为它只被子类使用。创建一个具体子类的实例时，它的父类的构造方法被调用以初始化父类中定义的数据域。

抽象类 `GeometricObject` 为几何对象定义了共同特征（数据和方法），并且提供了合适的构造方法。因为不知道如何计算几何对象的面积和周长，所以，`getArea()` 和 `getPerimeter()` 定义为抽象方法。这些方法在子类中实现。`Circle` 类和 `Rectangle` 类的实现除了扩展本章定义的 `GeometricObject` 类之外，其他都是同程序清单 11-2 和程序清单 11-3 一样的。可以分别从 www.cs.armstrong.edu/liang/intro10e/html/Circle.html 和 www.cs.armstrong.edu/liang/intro10e/html/Rectangle.html 得到两个程序的完整代码。

程序清单 13-2 Circle.java

```
1 public class Circle extends GeometricObject {
2     // Same as lines 3-48 in Listing 11.2, so omitted
3 }
```

程序清单 13-3 Rectangle.java

```
1 public class Rectangle extends GeometricObject {
2     // Same as lines 3-51 in Listing 11.3, so omitted
3 }
```

13.2.1 为何要使用抽象方法

你可能会疑惑在 `GeometricObject` 类中定义方法 `getArea()` 和 `getPerimeter()` 为抽象的而不是在每个子类中定义它们会有什么好处。下面程序清单 13-4 的例子就能看出在 `GeometricObject` 中定义它们的好处。程序创建了两个几何对象：一个圆和一个矩形，调用 `equalArea` 方法来检查它们的面积是否相同，然后调用 `displayGeometricObject` 方法来显示它们。

程序清单 13-4 TestGeometricObject.java

```
1 public class TestGeometricObject {
2     /** Main method */
3     public static void main(String[] args) {
4         // Create two geometric objects
5         GeometricObject geoObject1 = new Circle(5);
6         GeometricObject geoObject2 = new Rectangle(5, 3);
7
8         System.out.println("The two objects have the same area? " +
9             equalArea(geoObject1, geoObject2));
10
11        // Display circle
12        displayGeometricObject(geoObject1);
13
14        // Display rectangle
15        displayGeometricObject(geoObject2);
16    }
17
18    /** A method for comparing the areas of two geometric objects */
19    public static boolean equalArea(GeometricObject object1,
20        GeometricObject object2) {
21        return object1.getArea() == object2.getArea();
22    }
23
24    /** A method for displaying a geometric object */
25    public static void displayGeometricObject(GeometricObject object) {
26        System.out.println();
27        System.out.println("The area is " + object.getArea());
28        System.out.println("The perimeter is " + object.getPerimeter());
29    }
30 }
```

```
The two objects have the same area? false

The area is 78.53981633974483
The perimeter is 31.41592653589793

The area is 13.0
The perimeter is 16.0
```

`Circle` 类和 `Rectangle` 类中覆盖了定义在 `GeometricObject` 类中的 `getArea()` 和 `getPerimeter()` 方法。语句（第 5 ~ 6 行）：

```
GeometricObject geoObject1 = new Circle(5);
GeometricObject geoObject2 = new Rectangle(5, 3);
```

创建了一个新圆和一个新矩形，并把它们赋值给变量 `geoObject1` 和 `geoObject2`。这两个变量都是 `GeometricObject` 类型的。

当调用 `equalArea(geoObject1, geoObject2)` 时（第 9 行），由于 `geoObject1` 是一个圆，所以 `object1.getArea()` 使用的是 `Circle` 类定义的 `getArea()` 方法，而 `geoObject2` 是一个

矩形，所以 `object2.getArea()` 使用的是 `Rectangle` 类的 `getArea()` 方法。

类似地，当调用 `displayGeometricObject(geoObject1)` 时（第 12 行），使用在 `Circle` 类中定义的 `getArea()` 和 `getPerimeter()` 方法，而当调用 `displayGeometricObject(geoObject2)`（第 15 行）时，使用的是在 `Rectangle` 类中定义的 `getArea()` 和 `getPerimeter()` 方法。JVM 在运行时根据对象的类型动态地决定调用哪一个方法。

注意，如果 `GeometricObject` 里没有定义 `getArea()` 方法，就不能在该程序中定义 `equalArea` 方法来计算这两个几何对象的面积是否相同。所以，现在可以看出在 `GeometricObject` 中定义抽象方法的好处。

13.2.2 抽象类的几点说明

下面是关于抽象类值得注意的几点：

- 抽象方法不能包含在非抽象类中。如果抽象父类的子类不能实现所有的抽象方法，那么子类也必须定义为抽象的。换句话说，在抽象类扩展的非抽象子类中，必须实现所有的抽象方法。还要注意，抽象方法是非静态的。
- 抽象类是不能使用 `new` 操作符来初始化的。但是，仍然可以定义它的构造方法，这个构造方法在它的子类的构造方法中调用。例如，`GeometricObject` 类的构造方法在 `Circle` 类和 `Rectangle` 类中调用。
- 包含抽象方法的类必须是抽象的。但是，可以定义一个不包含抽象方法的抽象类。在这种情况下，不能使用 `new` 操作符创建该类的实例。这种类是用来定义新子类的基类的。
- 子类可以覆盖父类的方法并将它定义为 `abstract`。这是很少见的，但是它在当父类的方法实现在子类中变得无效时是很有用的。在这种情况下，子类必须定义为 `abstract`。
- 即使子类的父类是具体的，这个子类也可以是抽象的。例如，`Object` 类是具体的，但是它的子类如 `GeometricObject` 可以是抽象的。
- 不能使用 `new` 操作符从一个抽象类创建一个实例，但是抽象类可以用作一种数据类型。因此，下面的语句创建一个元素是 `GeometricObject` 类型的数组是正确的：

```
GeometricObject[] objects = new GeometricObject[10];
```

然后可以创建一个 `GeometricObject` 的实例，并将它的引用赋值给数组，如下所示：

```
objects[0] = new Circle();
```

复习题

13.1 在下面类的定义中，哪些定义了合法的抽象类？

```
class A {
    abstract void unfinished() {
    }
}
```

a)

```
public class abstract A {
    abstract void unfinished();
}
```

b)

```
class A {
    abstract void unfinished();
}
```

c)

```
abstract class A {
    protected void unfinished();
}
```

d)

```
abstract class A {
    abstract void unfinished();
}
```

e)

```
abstract class A {
    abstract int unfinished();
}
```

f)

13.2 `getArea()` 方法和 `getPerimeter()` 方法可以从 `GeometricObject` 类中删除。在 `GeometricObject` 类中将这两个方法定义为抽象方法的好处是什么？

13.3 下面说法为真还是为假？

- 除了不能使用 `new` 操作符创建抽象类的实例之外，一个抽象类可以像非抽象类一样使用。
- 抽象类可以被继承。
- 非抽象的父类的子类不能是抽象的。
- 子类不能将父类中的具体方法重写，并定义为抽象的。
- 抽象方法必须是非静态的。

13.3 示例学习：抽象的 Number 类

要点提示： `Number` 类是数值包装类、`BigInteger` 以及 `BigDecimal` 的抽象父类。

10.7 节介绍了数值包装类，10.9 节介绍了 `BigInteger` 以及 `BigDecimal` 类。这些类有共同的方法 `byteValue()`、`shortValue()`、`intValue()`、`longValue()`、`floatValue()` 和 `doubleValue()`，分别从这些类的对象返回 `byte`、`short`、`int`、`long`、`float` 以及 `double` 值。这些共同的方法实际上在 `Number` 类中定义，该类是数值包装类、`BigInteger` 和 `BigDecimal` 类的父类，如图 13-2 所示。

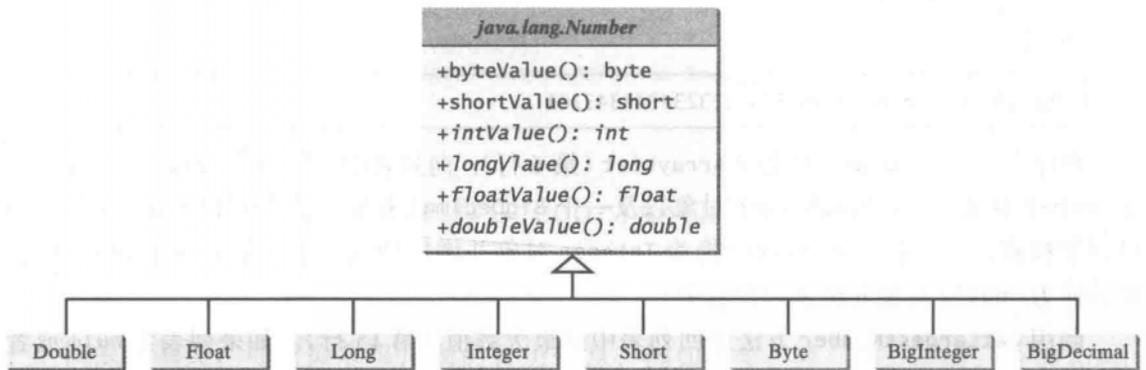


图 13-2 `Number` 类是 `Double`、`Float`、`Long`、`Integer`、`Short`、`Byte`，以及 `BigInteger` 和 `BigDecimal` 类的抽象父类

由于 `intValue()`、`longValue()`、`floatValue()` 以及 `doubleValue()` 等方法不能在 `Number` 类中给出实现，它们在 `Number` 类中被定义为抽象方法。因此 `Number` 类是一个抽象类。`byteValue()` 和 `shortValue()` 方法的实现从 `intValue()` 方法而来，如下所示：

```
public byte byteValue() {
    return (byte)intValue();
}
```

```
public short shortValue() {
    return (short)intValue();
}
```

Number 定义为数值类的父类，这样可以定义方法来执行数值的共同操作。程序清单 13-5 给出了一个程序，找到一个 Number 对象列表中的最大数。

程序清单 13-5 LargestNumbers.java

```

1  import java.util.ArrayList;
2  import java.math.*;
3
4  public class LargestNumbers {
5      public static void main(String[] args) {
6          ArrayList<Number> list = new ArrayList<>();
7          list.add(45); // Add an integer
8          list.add(3445.53); // Add a double
9          // Add a BigInteger
10         list.add(new BigInteger("3432323234344343101"));
11         // Add a BigDecimal
12         list.add(new BigDecimal("2.0909090989091343433344343"));
13
14         System.out.println("The largest number is " +
15             getLargestNumber(list));
16     }
17
18     public static Number getLargestNumber(ArrayList<Number> list) {
19         if (list == null || list.size() == 0)
20             return null;
21
22         Number number = list.get(0);
23         for (int i = 1; i < list.size(); i++)
24             if (number.doubleValue() < list.get(i).doubleValue())
25                 number = list.get(i);
26
27         return number;
28     }
29 }

```

The largest number is 3432323234344343101

程序创建一个 Number 对象的 ArrayList (第 6 行)，向列表中增加一个 Integer 对象、一个 Double 对象、一个 BigInteger 对象以及一个 BigDecimal 对象 (第 7 ~ 12 行)。注意，通过拆箱操作，第 7 行中 45 自动转换为 Integer 对象并增加到列表中，第 8 行中 3445.53 自动转换为 Double 对象并增加到列表中。

调用 getLargestNumber 方法返回列表中的最大数值 (第 15 行)。如果列表为 null 或者列表大小为 0，则 getLargestNumber 方法返回 null (第 19 ~ 20 行)。为了找到列表中的最大数值，通过调用数值对象上面的 doubleValue() 方法 (第 24 行)。doubleValue() 方法定义在 Number 类中，并在 Number 类的具体子类中得到实现。如果一个数值是一个 Integer 对象，Integer 的 doubleValue() 方法被调用。如果数值是一个 BigDecimal 对象，BigDecimal 的 doubleValue() 方法被调用。

如果 doubleValue() 方法没有在 Number 类中定义，将不能使用 Number 类从各种不同类型的数值中找到最大数值。

复习题

13.4 为什么下面两行代码可以编译成功，但是会导致运行错误？

```

Number numberRef = new Integer(0);
Double doubleRef = (Double)numberRef;

```

13.5 为什么下面两行代码可以编译成功，但是会导致运行错误？

```
Number[] numberArray = new Integer[2];
numberArray[0] = new Double(1.5);
```

13.6 给出下面代码的输出。

```
public class Test {
    public static void main(String[] args) {
        Number x = 3;
        System.out.println(x.intValue());
        System.out.println(x.doubleValue());
    }
}
```

13.7 下面代码有什么错误？（注意，Integer 和 Double 类的 compareTo 方法在第 10.7 节中进行了介绍）

```
public class Test {
    public static void main(String[] args) {
        Number x = new Integer(3);
        System.out.println(x.intValue());
        System.out.println(x.compareTo(new Integer(4)));
    }
}
```

13.8 下面代码中有什么错误？

```
public class Test {
    public static void main(String[] args) {
        Number x = new Integer(3);
        System.out.println(x.intValue());
        System.out.println((Integer)x.compareTo(new Integer(4)));
    }
}
```

13.4 示例学习：Calendar 和 GregorianCalendar

🔑 要点提示：GregorianCalendar 是抽象类 Calendar 的一个具体子类。

一个 java.util.Date 的实例表示以毫秒为精度的特定时刻。java.util.Calendar 是一个抽象的基类，可以提取出详细的日历信息，例如，年、月、日、小时、分钟和秒。Calendar 类的子类可以实现特定的日历系统，例如，公历（Gregorian 历）、农历和犹太历。目前，Java 支持公历类 java.util.GregorianCalendar，如图 13-3 所示。Calendar 类中的 add 方法是抽象的，因为它的实现依赖于某个具体的日历系统。

可以使用 new GregorianCalendar() 利用当前时间构造一个默认的 GregorianCalendar 对象，可以使用 new GregorianCalendar(year, month, date) 利用指定的 year（年）、month（月）和 date（日）构造一个 GregorianCalendar 对象。参数 month 是基于 0 的，即 0 代表 1 月（January）。

在 Calendar 类中定义的 get(int field) 方法在从 Calendar 类中提取日期和时间信息方面是很有用的。日期和时间域都被定义为常量，如表 13-1 所示。

程序清单 13-6 给出的例子显示了当前时间的日期和时间信息。

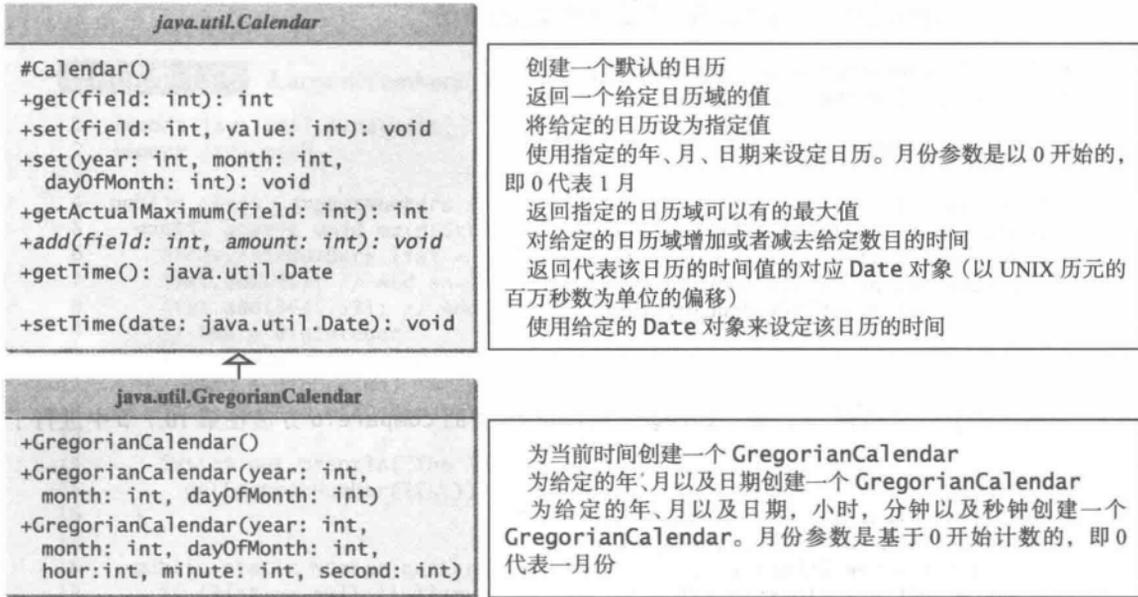


图 13-3 抽象的 Calendar 类定义了各种日历的共同特点

表 13-1 Calendar 类的域常量

常量	说明
YEAR	日历的年份
MONTH	日历的月份，0 表示一月
DATE	日历的天
HOUR	日历的小时（12 小时制）
HOUR_OF_DAY	日历的小时（24 小时制）
MINUTE	日历的分钟
SECOND	日历的秒
DAY_OF_WEEK	一周的天数，1 是星期日
DAY_OF_MONTH	和 DATE 一样
DAY_OF_YEAR	当前年的天数，1 是一年的第一天
WEEK_OF_MONTH	当前月内的星期数，1 是该月的第一个星期
WEEK_OF_YEAR	当前年内的星期数，1 是该年的第一个星期
AM_PM	表明是上午还是下午（0 表示上午，1 表示下午）

程序清单 13-6 TestCalendar.java

```
1 import java.util.*;
2
3 public class TestCalendar {
4     public static void main(String[] args) {
5         // Construct a Gregorian calendar for the current date and time
6         Calendar calendar = new GregorianCalendar();
7         System.out.println("Current time is " + new Date());
8         System.out.println("YEAR: " + calendar.get(Calendar.YEAR));
9         System.out.println("MONTH: " + calendar.get(Calendar.MONTH));
10        System.out.println("DATE: " + calendar.get(Calendar.DATE));
11        System.out.println("HOUR: " + calendar.get(Calendar.HOUR));
12        System.out.println("HOUR_OF_DAY: " +
```

```

13     calendar.get(Calendar.HOUR_OF_DAY));
14     System.out.println("MINUTE: " + calendar.get(Calendar.MINUTE));
15     System.out.println("SECOND: " + calendar.get(Calendar.SECOND));
16     System.out.println("DAY_OF_WEEK: " +
17         calendar.get(Calendar.DAY_OF_WEEK));
18     System.out.println("DAY_OF_MONTH: " +
19         calendar.get(Calendar.DAY_OF_MONTH));
20     System.out.println("DAY_OF_YEAR: " +
21         calendar.get(Calendar.DAY_OF_YEAR));
22     System.out.println("WEEK_OF_MONTH: " +
23         calendar.get(Calendar.WEEK_OF_MONTH));
24     System.out.println("WEEK_OF_YEAR: " +
25         calendar.get(Calendar.WEEK_OF_YEAR));
26     System.out.println("AM_PM: " + calendar.get(Calendar.AM_PM));
27
28     // Construct a calendar for September 11, 2001
29     Calendar calendar1 = new GregorianCalendar(2001, 8, 11);
30     String[] dayNameOfWeek = {"Sunday", "Monday", "Tuesday", "Wednesday",
31         "Thursday", "Friday", "Saturday"};
32     System.out.println("September 11, 2001 is a " +
33         dayNameOfWeek[calendar1.get(Calendar.DAY_OF_WEEK) - 1]);
34 }
35 }

```

```

Current time is Sun Nov 27 17:48:15 EST 2011
YEAR: 2011
MONTH: 10
DATE: 27
HOUR: 5
HOUR_OF_DAY: 17
MINUTE: 48
SECOND: 15
DAY_OF_WEEK: 1
DAY_OF_MONTH: 27
DAY_OF_YEAR: 331
WEEK_OF_MONTH: 5
WEEK_OF_YEAR: 49
AM_PM: 1
September 11, 2001 is a Tuesday

```

Calendar 类中定义的 `set(int field,value)` 方法用来设置一个域。例如，可以使用 `calendar.set(Calendar.DAY_OF_MONTH,1)` 将 `calendar` 设置为当月的第一天。

`add(field,value)` 方法为某个特定域增加指定的量。例如，`add(Calendar.DAY_OF_MONTH,5)` 给日历的当前时间加五天，而 `add(Calendar.DAY_OF_MONTH,-5)` 从日历的当前时间减去五天。

为了获得一个月中的天数，使用 `calendar.getActualMaximum(Calendar.DAY_OF_MONTH)` 方法。例如，如果是三月的 `calendar`，那么这个方法将返回 31。

可以通过调用 `calendar.setTime(date)` 为 `calendar` 设置一个用 `Date` 对象表示的时间，通过调用 `calendar.getTime()` 获取时间。

复习题

13.9 可以使用 `Calendar` 类来创建一个 `Calendar` 对象吗？

13.10 Calendar 中哪个方法是抽象的?

13.11 如何为当前时间创建一个 Calendar 对象?

13.12 对于一个 Calendar 对象 c 而言, 如何得到它的年、月、日期、小时、分钟以及秒钟?

13.5 接口

 **要点提示:** 接口是一种与类相似的结构, 只包含常量和抽象方法。

接口在许多方面都与抽象类很相似, 但是它的目的是指明相关或者不相关类的多个对象的共同行为。例如, 使用正确的接口, 可以指明这些对象是可比较的、可食用的, 以及可克隆的。

为了区分接口和类, Java 采用下面的语法来定义接口:

```
修饰符 interface 接口名 {
    /** 常量声明 */
    /** 方法签名 */
}
```

下面是一个接口的例子:

```
modifier interface InterfaceName {
    /** Constant declarations */
    /** Abstract method signatures */
}
```

在 Java 中, 接口被看作是一种特殊的类。就像常规类一样, 每个接口都被编译为独立的字节码文件。使用接口或多或少有点像使用抽象类。例如, 可以使用接口作为引用变量的数据类型或类型转换的结果等。与抽象类相似, 不能使用 new 操作符创建接口的实例。

可以使用 Edible 接口来明确一个对象是否是可食用的。这需要使使用 implements 关键字让对象的类实现这个接口来完成。例如, 程序清单 13-7 中的 Chicken 类和 Fruit 类 (第 20、39 行) 实现 Edible 接口。类和接口之间的关系称为接口继承 (interface inheritance)。因为接口继承和类继承本质上是相同的, 所以我们将它们都简称为继承。

程序清单 13-7 TestEdible.java

```
1 public class TestEdible {
2     public static void main(String[] args) {
3         Object[] objects = {new Tiger(), new Chicken(), new Apple()};
4         for (int i = 0; i < objects.length; i++) {
5             if (objects[i] instanceof Edible)
6                 System.out.println(((Edible)objects[i]).howToEat());
7
8             if (objects[i] instanceof Animal) {
9                 System.out.println(((Animal)objects[i]).sound());
10            }
11        }
12    }
13 }
14
15 abstract class Animal {
16     /** Return animal sound */
17     public abstract String sound();
18 }
19
20 class Chicken extends Animal implements Edible {
21     @Override
22     public String howToEat() {
23         return "Chicken: Fry it";
```

```

24 }
25
26 @Override
27 public String sound() {
28     return "Chicken: cock-a-doodle-doo";
29 }
30 }
31
32 class Tiger extends Animal {
33     @Override
34     public String sound() {
35         return "Tiger: RROOAARR";
36     }
37 }
38
39 abstract class Fruit implements Edible {
40     // Data fields, constructors, and methods omitted here
41 }
42
43 class Apple extends Fruit {
44     @Override
45     public String howToEat() {
46         return "Apple: Make apple cider";
47     }
48 }
49
50 class Orange extends Fruit {
51     @Override
52     public String howToEat() {
53         return "Orange: Make orange juice";
54     }
55 }
    
```

Tiger: RROOAARR
 Chicken: Fry it
 Chicken: cock-a-doodle-doo
 Apple: Make apple cider

这个例子使用了多个类和接口。它们的继承关系如图 13-4 所示。

Animal 类定义了 sound 方法（第 17 行）。这是个抽象方法，将被具体的动物类所实现。

Chicken 类实现了 Edible 接口以表明小鸡是可食用的。当一个类实现接口时，该类用同样的签名和返回值类型实现定义在接口中的所有方法。Chicken 类实现了 howToEat 方法（第 22 ~ 24 行）。Chicken 也继承 Animal 类并实现 sound 方法（第 27 ~ 29 行）。

接口名字和方法名字
 使用斜体。虚线和空心
 三角形用于指向接口

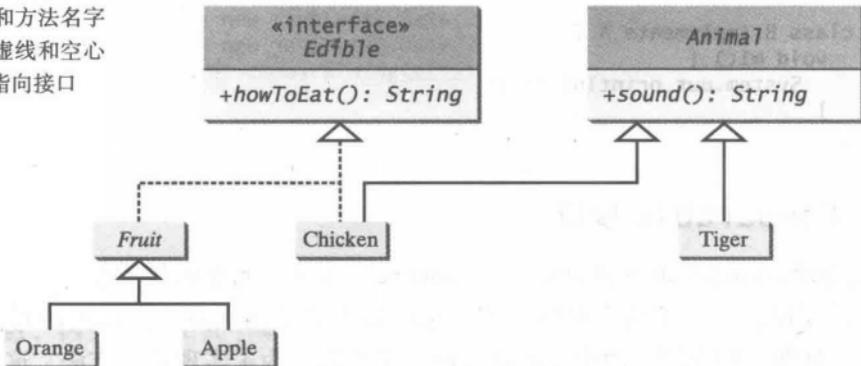


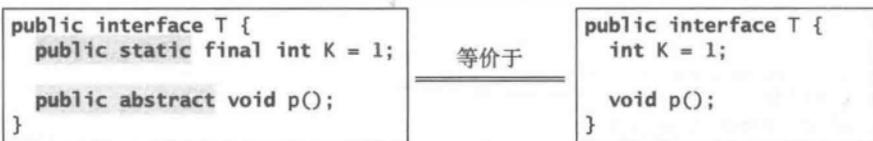
图 13-4 Edible 是 Chicken 和 Fruit 的父类型。Animal 是 Chicken 和 Tiger 的父类型。Fruit 是 Orange 和 Apple 的父类型

Fruit 类实现 Edible。因为它不实现 howToEat 方法，所以 Fruit 必须表示为 abstract (第 39 行)。Fruit 的具体子类必须实现 hotToEat 方法。Apple 类和 Orange 类实现 howToEat 方法 (第 45、52 行)。

main 方法创建由 Tiger、Chicken 和 Apple 类型的三个对象构成的数组 (第 3 行)，如果某元素是可食用的，则调用 howToEat 方法 (第 6 行)，如果某元素是一种动物，则调用 sound 方法 (第 9 行)。

本质上，Edible 接口定义了可食用对象的公共行为。所有可食用的对象都有 howToEat 方法。

注意：由于接口中所有的数据域都是 public static final 而且所有的方法都是 public abstract，所以 Java 允许忽略这些修饰符。因此，下面的接口定义是等价的：



复习题

- 13.13 假设 A 是一个接口，可以使用 new A() 创建一个实例吗？
- 13.14 假设 A 是一个接口，可以如下声明一个类型 A 的引用变量 x 吗？
A x;
- 13.15 下面哪个是正确的接口？

```
interface A {
    void print() { };
}
```

a)

```
abstract interface A extends I1, I2 {
    abstract void print() { };
}
```

b)

```
abstract interface A {
    print();
}
```

c)

```
interface A {
    void print();
}
```

d)

- 13.16 指出下面代码中的错误。

```
interface A {
    void m1();
}

class B implements A {
    void m1() {
        System.out.println("m1");
    }
}
```

13.6 Comparable 接口

要点提示：Comparable 接口定义了 compareTo 方法，用于比较对象。

假设要设计一个求两个相同类型对象中较大者的通用方法。这里的对象可以是两个学生、两个日期、两个圆、两个矩形或者两个正方形。为了实现这个方法，这两个对象必须是可比较的。因此，这两个对象都该有的共同方法就是 comparable (可比较的)。为此，Java

提供了 Comparable 接口。接口的定义如下所示：

```
// Interface for comparing objects, defined in java.lang
package java.lang;

public interface Comparable<E> {
    public int compareTo(E o);
}
```

compareTo 方法判断这个对象相对于给定对象 o 的顺序，并且当这个对象小于、等于或大于给定对象 o 时，分别返回负整数、0 或正整数。

Comparable 接口是一个泛型接口。在实现该接口时，泛型类型 E 被替换成一种具体的类型。Java 类库中的许多类实现了 Comparable 接口以定义对象的自然顺序。Byte、Short、Integer、Long、Float、Double、Character、BigInteger、BigDecimal、Calendar、String 以及 Date 类都实现了 Comparable 接口。例如，在 Java API 中，Integer、BigInteger、String 以及 Date 类都如下定义：

```
public class Integer extends Number
    implements Comparable<Integer> {
    // class body omitted

    @Override
    public int compareTo(Integer o) {
        // Implementation omitted
    }
}
```

```
public class BigInteger extends Number
    implements Comparable<BigInteger> {
    // class body omitted

    @Override
    public int compareTo(BigInteger o) {
        // Implementation omitted
    }
}
```

```
public class String extends Object
    implements Comparable<String> {
    // class body omitted

    @Override
    public int compareTo(String o) {
        // Implementation omitted
    }
}
```

```
public class Date extends Object
    implements Comparable<Date> {
    // class body omitted

    @Override
    public int compareTo(Date o) {
        // Implementation omitted
    }
}
```

因此，数字是可比较的，字符串是可比较的，日期也是如此。可以使用 compareTo 方法来比较两个数字、两个字符串以及两个日期。例如，下面代码

```
1 System.out.println(new Integer(3).compareTo(new Integer(5)));
2 System.out.println("ABC".compareTo("ABE"));
3 java.util.Date date1 = new java.util.Date(2013, 1, 1);
4 java.util.Date date2 = new java.util.Date(2012, 1, 1);
5 System.out.println(date1.compareTo(date2));
```

显示

```
-1
-2
1
```

第 1 行显示一个负数，因为 3 小于 5。第 2 行显示一个负数，因为 ABC 小于 ABE。第 5 行显示一个正数，因为 date1 大于 date2。

将 n 赋值为一个 Integer 对象，s 为一个 string 对象，d 为一个 Date 对象。下面的所有表达式都为 true。

```
n instanceof Integer
n instanceof Object
n instanceof Comparable
```

```
s instanceof String
s instanceof Object
s instanceof Comparable
```

```
d instanceof java.util.Date
d instanceof Object
d instanceof Comparable
```

由于所有 Comparable 对象都有 compareTo 方法，如果对象是 Comparable 接口类型的实例的话，Java API 中的 java.util.Arrays.sort(Object[]) 方法就可以使用 compareTo 方法来对数组中的对象进行比较和排序。程序清单 13-8 给出了一个对字符串数组和 BigInteger 对象数组进行排序的示例：

程序清单 13-8 SortComparableObjects.java

```
1 import java.math.*;
2
3 public class SortComparableObjects {
4     public static void main(String[] args) {
5         String[] cities = {"Savannah", "Boston", "Atlanta", "Tampa"};
6         java.util.Arrays.sort(cities);
7         for (String city: cities)
8             System.out.print(city + " ");
9         System.out.println();
10
11        BigInteger[] hugeNumbers = {new BigInteger("2323231092923992"),
12            new BigInteger("432232323239292"),
13            new BigInteger("54623239292")};
14        java.util.Arrays.sort(hugeNumbers);
15        for (BigInteger number: hugeNumbers)
16            System.out.print(number + " ");
17    }
18 }
```

```
Atlanta Boston Savannah Tampa
54623239292 432232323239292 2323231092923992
```

程序创建一个字符串数组（第 5 行），并且调用 sort 方法来对字符串进行排序（第 6 行）。程序创建一个 BigInteger 对象的数组（第 11 ~ 13 行），并且调用 sort 方法来对 BigInteger 对象进行排序（第 14 行）。

不能使用 sort 方法来对一个 Rectangle 对象数组进行排序，因为 Rectangle 类没有实现接口 Comparable。然而，可以定义一个新的 Rectangle 类来实现 Comparable。这个新类的实例是可比较的。将这个新类命名为 ComparableRectangle，如程序清单 13-9 所示。

程序清单 13-9 ComparableRectangle.java

```
1 public class ComparableRectangle extends Rectangle
2     implements Comparable<ComparableRectangle> {
3     /** Construct a ComparableRectangle with specified properties */
4     public ComparableRectangle(double width, double height) {
5         super(width, height);
6     }
7
8     @Override // Implement the compareTo method defined in Comparable
9     public int compareTo(ComparableRectangle o) {
10        if (getArea() > o.getArea())
11            return 1;
12        else if (getArea() < o.getArea())
13            return -1;
14        else
15            return 0;
16    }
17 }
```

```

18  @Override // Implement the toString method in GeometricObject
19  public String toString() {
20      return super.toString() + " Area: " + getArea();
21  }
22  }

```

ComparableRectangle 类扩展自 Rectangle 类并实现 Comparable 方法，如图 13-5 所示。关键字 implements 表示 ComparableRectangle 类继承 Comparable 接口的所有常量，并实现该接口的方法。compareTo 方法比较两个矩形的面积。ComparableRectangle 类的一个实例也是 Rectangle、GeometricObject、Object 和 Comparable 的实例。

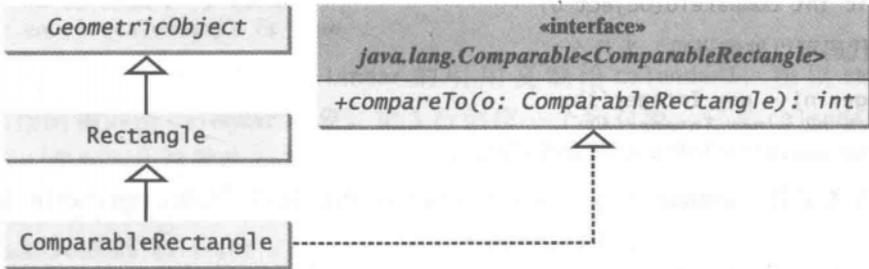


图 13-5 ComparableRectangle 类扩展 Rectangle 类并实现 Comparable 接口

现在，可以使用 sort 方法来对 ComparableRectangle 对象数组进行排序了，如程序清单 13-10 所示：

程序清单 13-10 SortRectangles.java

```

1  public class SortRectangles {
2      public static void main(String[] args) {
3          ComparableRectangle[] rectangles = {
4              new ComparableRectangle(3.4, 5.4),
5              new ComparableRectangle(13.24, 55.4),
6              new ComparableRectangle(7.4, 35.4),
7              new ComparableRectangle(1.4, 25.4)};
8          java.util.Arrays.sort(rectangles);
9          for (Rectangle rectangle: rectangles) {
10             System.out.print(rectangle + " ");
11             System.out.println();
12         }
13     }
14 }

```

```

Width: 3.4 Height: 5.4 Area: 18.36
Width: 1.4 Height: 25.4 Area: 35.559999999999995
Width: 7.4 Height: 35.4 Area: 261.96
Width: 13.24 Height: 55.4 Area: 733.496

```

接口提供通用程序设计的另一种形式。在这个例子中，如果不用接口，很难使用通用的 sort 方法来排序对象，因为必须使用多重继承才能同时继承 Comparable 和另一个类，例如 Rectangle。

Object 类包含 equals 方法，它的目的就是为了让 Object 类的子类来覆盖它，以比较对象的内容是否相同。假设 Object 类包含一个类似于 Comparable 接口中所定义的 compareTo 方法，那么 sort 方法可以用来比较一组任意的对象。Object 类中是否应该包含一个 compareTo 方法尚有争论。由于在 Object 类中没有定义 compareTo 方法，所以 Java 中定义

了 Comparable 接口，以便能够对两个 Comparable 接口的实例对象进行比较。强烈建议（尽管不要求）compareTo 应该与 equals 保持一致。也就是说，对于两个对象 o1 和 o2，应该确保当且仅当 o1.equals(o2) 为 true 时 o1.compareTo(o2)==0 成立。

☛ 复习题

13.17 下面说法为真还是为假？如果一个类实现了 Comparable，那么该类的对象可以调用 compareTo 方法。

13.18 下面哪个是正确的 String 类中的 compareTo 方法的方法头？

```
public int compareTo(String o)
public int compareTo(Object o)
```

13.19 下面代码可以被编译吗？为什么？

```
Integer n1 = new Integer(3);
Object n2 = new Integer(4);
System.out.println(n1.compareTo(n2));
```

13.20 可以在类中定义 compareTo 方法而不实现 Comparable 接口。实现 Comparable 接口的好处是什么？

13.21 下面的代码有什么错误？

```
public class Test {
    public static void main(String[] args) {
        Person[] persons = {new Person(3), new Person(4), new Person(1)};
        java.util.Arrays.sort(persons);
    }
}

class Person {
    private int id;

    Person(int id) {
        this.id = id;
    }
}
```

13.7 Cloneable 接口

🔑 要点提示：Cloneable 接口给出了一个可克隆的对象。

经常会出现需要创建一个对象拷贝的情况。为了实现这个目的，需要使用 clone 方法并理解 Cloneable 接口。

接口包括常量和抽象方法，但是 Cloneable 接口是一个特殊情况。在 java.lang 包中的 Cloneable 接口的定义如下所示：

```
package java.lang;

public interface Cloneable {
}
```

这个接口是空的。一个带空体的接口称为标记接口（marker interface）。一个标记接口既不包括常量也不包括方法。它用来表示一个类拥有某些特定的属性。实现 Cloneable 接口的类标记为可克隆的，而且它的对象可以使用在 Object 类中定义的 clone() 方法克隆。

Java 库中的很多类（例如，Date、Calendar 和 ArrayList）实现 Cloneable。这样，这些类的实例可以被克隆。例如，下面的代码

```
1 Calendar calendar = new GregorianCalendar(2013, 2, 1);
2 Calendar calendar1 = calendar;
3 Calendar calendar2 = (Calendar)calendar.clone();
4 System.out.println("calendar == calendar1 is " +
5   (calendar == calendar1));
6 System.out.println("calendar == calendar2 is " +
7   (calendar == calendar2));
8 System.out.println("calendar.equals(calendar2) is " +
9   calendar.equals(calendar2));
```

显示

```
calendar == calendar1 is true
calendar == calendar2 is false
calendar.equals(calendar2) is true
```

在前面的代码中，第2行将 calendar 的引用复制给 calendar1，所以 calendar 和 calendar1 都指向相同的 Calendar 对象。第3行创建一个新对象，它是 calendar 的克隆，然后将这个新对象的引用赋值给 calendar2。calendar2 和 calendar 是内容相同的不同对象。

下面的代码

```
1 ArrayList<Double> list1 = new ArrayList<>();
2 list1.add(1.5);
3 list1.add(2.5);
4 list1.add(3.5);
5 ArrayList<Double> list2 = (ArrayList<Double>)list1.clone();
6 ArrayList<Double> list3 = list1;
7 list2.add(4.5);
8 list3.remove(1.5);
9 System.out.println("list1 is " + list1);
10 System.out.println("list2 is " + list2);
11 System.out.println("list3 is " + list3);
```

显示

```
list1 is [2.5, 3.5]
list2 is [1.5, 2.5, 3.5, 4.5]
list3 is [2.5, 3.5]
```

前面的代码中，第5行创建了一个新对象作为 list1 的克隆，并且将新对象的引用赋值给 list2。list2 和 list1 是具有同样内容的不同对象。第6行复制 list1 的引用给 list3，因此 list1 和 list3 指向同一个 ArrayList 对象。第7行将 4.5 添加到 list2 中。第8行从 list3 中移除 1.5。由于 list1 和 list3 指向同一个 ArrayList，第9行和第11行显示同样的内容。

可以使用 clone 方法克隆一个数组。例如，下面的代码

```
1 int[] list1 = {1, 2};
2 int[] list2 = list1.clone();
3 list1[0] = 7;
4 list2[1] = 8;
5 System.out.println("list1 is " + list1[0] + ", " + list1[1]);
6 System.out.println("list2 is " + list2[0] + ", " + list2[1]);
```

显示

```
list1 is 7, 2
list2 is 1, 8
```

为了定义一个自定义类来实现 Cloneable 接口，这个类必须覆盖 Object 类中的 clone() 方法。程序清单 13-11 定义一个实现 Cloneable 和 Comparable 的名为 House 的类。

程序清单 13-11 House.java

```

1 public class House implements Cloneable, Comparable<House> {
2     private int id;
3     private double area;
4     private java.util.Date whenBuilt;
5
6     public House(int id, double area) {
7         this.id = id;
8         this.area = area;
9         whenBuilt = new java.util.Date();
10    }
11
12    public int getId() {
13        return id;
14    }
15
16    public double getArea() {
17        return area;
18    }
19
20    public java.util.Date getWhenBuilt() {
21        return whenBuilt;
22    }
23
24    @Override /** Override the protected clone method defined in
25                the Object class, and strengthen its accessibility */
26    public Object clone() throws CloneNotSupportedException {
27        return super.clone();
28    }
29
30    @Override // Implement the compareTo method defined in Comparable
31    public int compareTo(House o) {
32        if (area > o.area)
33            return 1;
34        else if (area < o.area)
35            return -1;
36        else
37            return 0;
38    }
39 }

```

House 类实现在 Object 类中定义的 clone 方法 (第 26 ~ 28 行), 方法头是:

```
protected native Object clone() throws CloneNotSupportedException;
```

关键字 native 表明这个方法不是用 Java 写的, 但它是 JVM 针对自身平台实现的。关键字 protected 限定方法只能在同一个包内或在其子类中访问。由于这个原因, House 类必须覆盖该方法并将它的可见性修饰符改为 public, 这样, 该方法就可以在任何一个包中使用。因为 Object 类中针对自身平台实现的 clone 方法完成了克隆对象的任务, 所以, 在 House 类中的 clone 方法只要简单调用 super.clone() 即可。在 Object 类中定义的 clone 方法会抛出 CloneNotSupportedException 异常。

House 类实现定义在 Comparable 接口中的 compareTo 方法 (第 31 ~ 38 行)。该方法比较两个房子的面积。

现在, 可以创建一个 House 类的对象, 然后从这个对象创建一个完全一样的拷贝, 如下所示:

```
House house1 = new House(1, 1750.50);
House house2 = (House)house1.clone();
```

house1 和 house2 是两个内容相同的不同对象。Object 类中的 clone 方法将原始对象的每个数据域复制给目标对象。如果一个数据域是基本类型，复制的就是它的值。例如，area (double 类型) 的值从 house1 复制到 house2。如果一个数据域是对象，复制的就是该域的引用。例如，域 whenBuilt 是 Date 类，所以，它的引用被复制给 house2，如图 13-6 所示。因此，尽管 house1==house2 为假，但是 house1.whenBuilt==house2.whenBuilt 为真。这称为浅复制 (shallow copy) 而不是深复制 (deep copy)，这意味着如果数据域是对象类型，那么复制的是对象的引用，而不是它的内容。

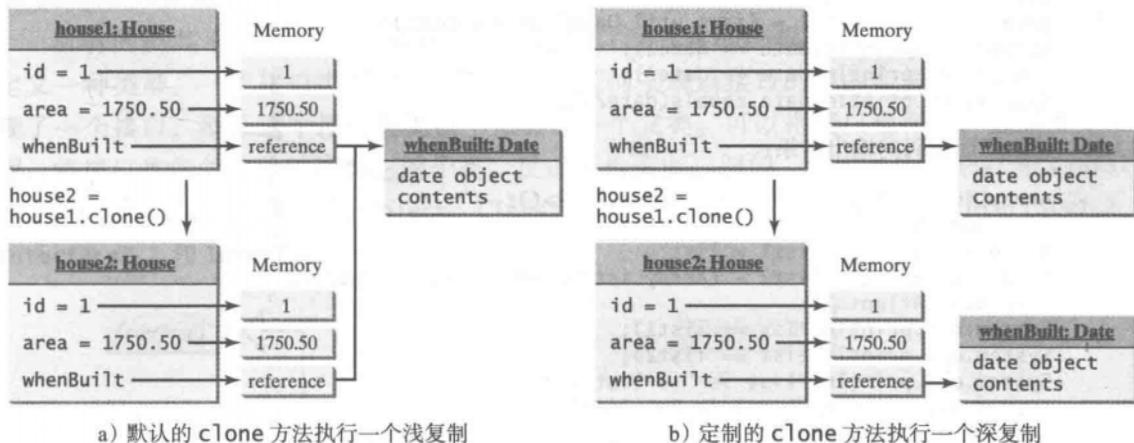


图 13-6

如果希望为 House 对象执行深复制，将 clone() 方法中的 26 ~ 28 行替换为下面代码：

```
public Object clone() throws CloneNotSupportedException {
    // Perform a shallow copy
    House houseClone = (House)super.clone();
    // Deep copy on whenBuilt
    houseClone.whenBuilt = (java.util.Date)(whenBuilt.clone());
    return houseClone;
}
```

或者

```
public Object clone() {
    try {
        // Perform a shallow copy
        House houseClone = (House)super.clone();
        // Deep copy on whenBuilt
        houseClone.whenBuilt = (java.util.Date)(whenBuilt.clone());
        return houseClone;
    }
    catch (CloneNotSupportedException ex) {
        return null;
    }
}
```

现在如果使用下面代码复制一个 House 对象：

```
House house1 = new House(1, 1750.50);
House house2 = (House)house1.clone();
```

House1.whenBuilt == house2.whenBuilt 将为 false。house1 和 house2 包含两个不同的 Date 对象，如图 13-6b 所示。

复习题

- 13.22 如果一个对象的类没有实现 `java.lang.Cloneable`，可以调用 `clone()` 方法来克隆这个对象吗？`Date` 类实现了 `Cloneable` 接口吗？
- 13.23 如果 `House` 类（在程序清单 13-11 中定义）没有覆盖 `clone()` 方法，或者如果 `House` 类没有实现 `java.lang.Cloneable`，会发生什么？
- 13.24 给出下面代码的输出结果：

```
java.util.Date date = new java.util.Date();
java.util.Date date1 = date;
java.util.Date date2 = (java.util.Date)(date.clone());
System.out.println(date == date1);
System.out.println(date == date2);
System.out.println(date.equals(date2));
```

- 13.25 给出下面代码的输出结果：

```
ArrayList<String> list = new ArrayList<>();
list.add("New York");
ArrayList<String> list1 = list;
ArrayList<String> list2 = (ArrayList<String>)(list.clone());
list.add("Atlanta");
System.out.println(list == list1);
System.out.println(list == list2);
System.out.println("list is " + list);
System.out.println("list1 is " + list1);
System.out.println("list2.get(0) is " + list2.get(0));
System.out.println("list2.size() is " + list2.size());
```

- 13.26 下面的代码有什么错误？

```
public class Test {
    public static void main(String[] args) {
        GeometricObject x = new Circle(3);
        GeometricObject y = x.clone();
        System.out.println(x == y);
    }
}
```

13.8 接口与抽象类

 **要点提示：**一个类可以实现多个接口，但是只能继承一个父类。

接口的使用和抽象类的使用基本类似，但是，定义一个接口与定义一个抽象类有所不同。表 13-2 总结了这些不同点。

表 13-2 接口与抽象类

	变量	构造方法	方法
抽象类	无限制	子类通过构造方法链调用构造方法，抽象类不能用 <code>new</code> 操作符实例化	无限制
接口	所有的变量必须是 <code>public static final</code>	没有构造方法。接口不能用 <code>new</code> 操作符实例化	所有方法必须是公共的抽象实例方法

Java 只允许为类的扩展做单一继承，但是允许使用接口做多重扩展。例如，

```
public class NewClass extends BaseClass
    implements Interface1, ..., InterfaceN {
    ...
}
```

利用关键字 `extends`，接口可以继承其他接口。这样的接口称为子接口（subinterface）。例如，在下面代码中，`NewInterface` 是 `Interface1`，...，`InterfaceN` 的子接口。

```
public interface NewInterface extends Interface1, ... , InterfaceN {
    // constants and abstract methods
}
```

一个实现 `NewInterface` 的类必须实现在 `NewInterface`，`Interface1`，...，`InterfaceN` 中定义的抽象方法。接口可以扩展其他接口而不是类。一个类可以扩展它的父类同时实现多个接口。

所有的类共享同一个根类 `Object`，但是接口没有共同的根。与类相似，接口也可以定义一种类型。一个接口类型的变量可以引用任何实现该接口的类的实例。如果一个类实现了一个接口，那么这个接口就类似于该类的一个父类。可以将接口当作一种数据类型使用，将接口类型的变量转换为它的子类，反过来也可以。例如，假设 `c` 是图 13-7 中 `Class2` 的一个实例，那么 `c` 也是 `Object`、`Class1`、`Interface1`、`Interface1_1`、`Interface1_2`、`Interface2_1` 和 `Interface2_2` 的实例。

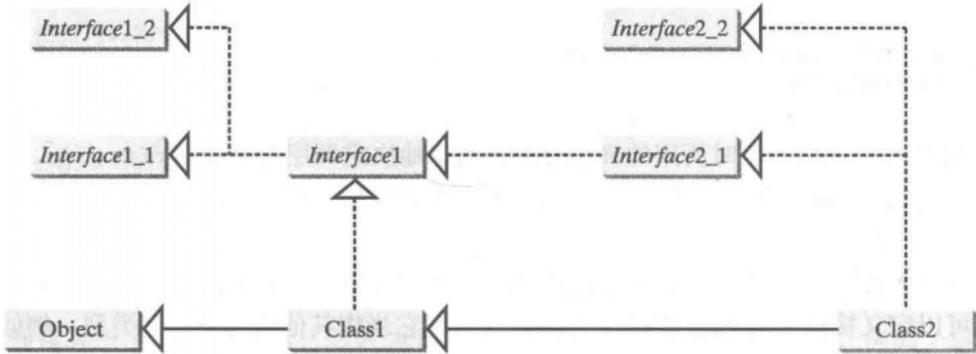


图 13-7 `Class1` 实现接口 `Interface1`；`Interface1` 扩展接口 `Interface1_1` 和 `Interface1_2`。`Class2` 扩展 `Class1` 并实现接口 `Interface2_1` 和 `Interface2_2`

🔑 注意：类名是一个名词。接口名可以是形容词或名词。

🔑 设计指南：抽象类和接口都是用来明确多个对象的共同特征的。那么该如何确定在什么情况下应该使用接口，什么情况下应该使用类呢？一般来说，清晰描述父子关系的强的“是一种”的关系（strong is-a relationship）应该用类建模。例如，因为公历是一种日历，所以，类 `java.util.GregorianCalendar` 和 `java.util.Calendar` 是用类继承建模的。弱的“是一种”的关系（weak is-a relationship）也称为类属关系（is-kind-of relationship），它表明对象拥有某种属性，可以用接口来建模。例如，所有的字符串都是可比较的，因此，`String` 类实现 `Comparable` 接口。

通常，推荐使用接口而非抽象类，因为接口可以定义非相关类共有的父类型。接口比类更加灵活。考虑 `Animal` 类。假设 `Animal` 类中定义了 `howToEat` 方法，如下所示：

```
abstract class Animal {
    public abstract String howToEat();
}
```

`Animal` 的两个子类定义如下：

```

class Chicken extends Animal {
    @Override
    public String howToEat() {
        return "Fry it";
    }
}

class Duck extends Animal {
    @Override
    public String howToEat() {
        return "Roast it";
    }
}

```

假设给定这个继承体系结构，多态会让你在一个类型为 `Animal` 的变量中保存 `Chicken` 对象或 `Duck` 对象的引用，如下面代码所示：

```

public static void main(String[] args) {
    Animal animal = new Chicken();
    eat(animal);

    animal = new Duck();
    eat(animal);
}

public static void eat(Animal animal) {
    animal.howToEat();
}

```

JVM 会基于调用方法时所用的确切对象来动态地决定调用哪个 `howToEat` 方法。

可以定义 `Animal` 的一个子类。但是，这里有个限制条件。该子类必须是另一种动物（例如，`Turkey`）。

接口就无此限制。接口比类拥有更多的灵活性，因为不用使所有东西都属于同一个类型的类。可以定义接口中的 `howToEat()` 方法，然后把它当作其他类的公用父类型。例如，

```

public static void main(String[] args) {
    Edible stuff = new Chicken();
    eat(stuff);

    stuff = new Duck();
    eat(stuff);

    stuff = new Broccoli();
    eat(stuff);
}

public static void eat(Edible stuff) {
    stuff.howToEat();
}

interface Edible {
    public String howToEat();
}

class Chicken implements Edible {
    @Override
    public String howToEat() {
        return "Fry it";
    }
}

class Duck implements Edible {
    @Override

```

```
public String howToEat() {  
    return "Roast it";  
}  
}  
  
class Broccoli implements Edible {  
    @Override  
    public String howToEat() {  
        return "Stir-fry it";  
    }  
}
```

为了定义表示可食用对象的一个类，只须让该类实现 `Edible` 接口即可。现在，这个类就成为 `Edible` 类型的子类型。任何 `Edible` 对象都可以被传递以调用 `howToEat` 方法。

复习题

- 13.27 给出一个例子显示接口相对于抽象类的优势。
- 13.28 给出抽象类和接口的定义。抽象类和接口之间的相同点和不同点是什么？
- 13.29 真或者假？
 - a. 接口被编译为独立的字节码文件。
 - b. 接口可以有静态方法。
 - c. 接口可以扩展一个或多个接口。
 - d. 接口可以扩展抽象类。
 - e. 抽象类可以扩展接口。

13.9 示例学习：Rational 类

要点提示：本节演示如何设计一个 `Rational` 类，用于表示和处理有理数。

有理数有一个分子和分母，形式为 a/b ，这里的 a 是分子而 b 是分母。例如， $1/3$ 、 $3/4$ 和 $10/4$ 都是有理数。

有理数的分母不能为 0，但是分子可以为 0。每个整数 i 等价于一个有理数 $i/1$ 。有理数用于涉及分数的准确计算，例如， $1/3=0.33333\dots$ 。这个数字不能用 `double` 或 `float` 数据类型精确地表示为浮点形式。为了获取准确的结果，必须使用有理数。

Java 提供了表示整数和浮点数的数据类型，但是没有提供表示有理数的数据类型。本节给出如何设计一个表示有理数的类。

因为有理数共享了很多整数和浮点数的通用特性，而且 `Number` 是数值包装类的根类，所以将 `Rational` 类定义为 `Number` 类的子类是合适的。因为有理数是可以比较的，所以 `Rational` 类应该也能实现 `Comparable` 接口。图 13-8 说明了 `Rational` 类以及它和 `Number` 类及 `Comparable` 接口的关系。

一个有理数包括一个分子和一个分母。有很多有理数是等价的，例如， $1/3=2/6=3/9=4/12$ 。 $1/3$ 的分子和分母除了 1 之外没有公约数，所以， $1/3$ 称为最简形式。

为了将一个有理数约减为它的最低形式，需要找到分子和分母绝对值的最大公约数 (GCD)，然后将分子和分母都除以这个值。可以使用程序清单 5-9 中计算两个整数 n 和 d 的 GCD 方法。在 `Rational` 对象中的分子和分母都可以降为它们的最简形式。

与往常一样，我们首先编写一个测试程序来创建两个 `Rational` 对象，测试它的方法。程序清单 13-12 是一个测试程序。

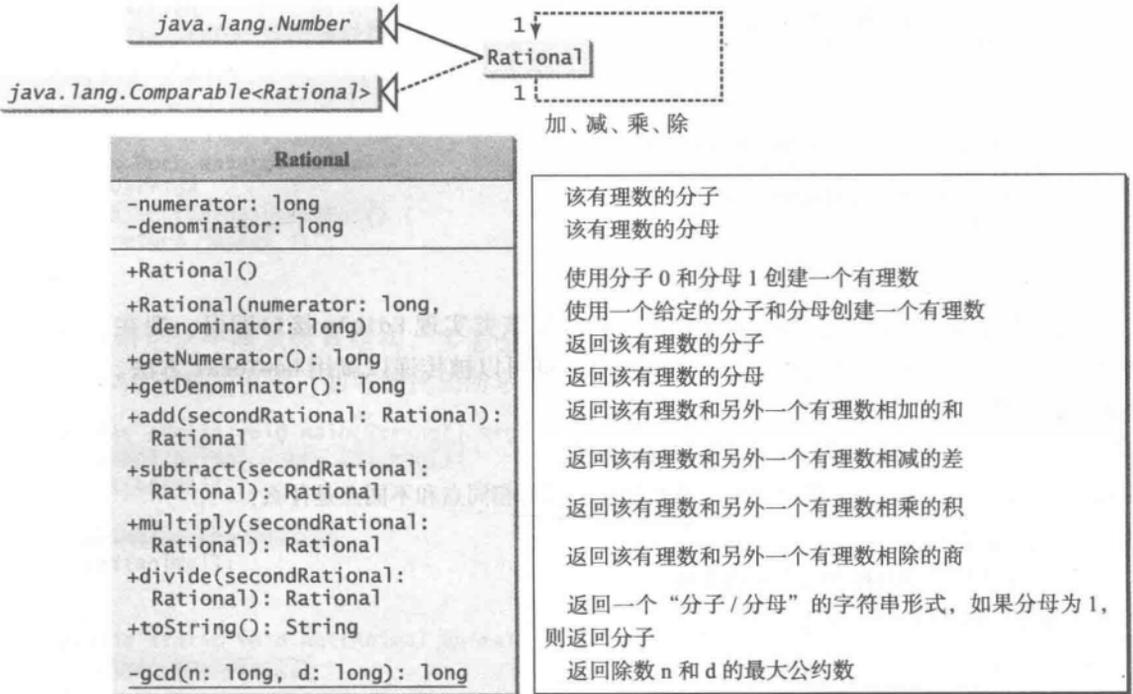


图 13-8 Rational 类的属性、构造方法和方法在 UML 中的图解

程序清单 13-12 TestRationalClass.java

```

1 public class TestRationalClass {
2     /** Main method */
3     public static void main(String[] args) {
4         // Create and initialize two rational numbers r1 and r2
5         Rational r1 = new Rational(4, 2);
6         Rational r2 = new Rational(2, 3);
7
8         // Display results
9         System.out.println(r1 + " + " + r2 + " = " + r1.add(r2));
10        System.out.println(r1 + " - " + r2 + " = " + r1.subtract(r2));
11        System.out.println(r1 + " * " + r2 + " = " + r1.multiply(r2));
12        System.out.println(r1 + " / " + r2 + " = " + r1.divide(r2));
13        System.out.println(r2 + " is " + r2.doubleValue());
14    }
15 }

```

```

2 + 2/3 = 8/3
2 - 2/3 = 4/3
2 * 2/3 = 4/3
2 / 2/3 = 3
2/3 is 0.6666666666666666

```

main 方法创建了两个有理数：r1 和 r2（第 5～6 行），然后显示 r1+r2、r1-r2、r1xr2 和 r1/r2 的结果（第 9～12 行）。为了计算 r1+r2，调用 r1.add(r2) 返回一个新的 Rational 对象。同样地，r1.subtract(r2) 用以计算 r1-r2，r1.multiply(r2) 用以计算 r1xr2，而 r1.divide(r2) 用以计算 r1/r2。

doubleValue() 方法显示 r2 的 double 值（第 13 行）。doubleValue() 方法在 java.lang.Number 中定义并且在 Rational 中被覆盖。

注意，当使用加号(+)将一个字符串和一个对象进行链接时，使用的是来自toString()方法的这个对象的字符串表示同这个字符串进行链接。因此，r1+" "+r2+" "+r1.add(r2)等价于r1.toString()+" "+r2.toString()+" "+r1.add(r2).toString()。

Rational类在程序清单13-13中实现。

程序清单 13-13 Rational.java

```

1 public class Rational extends Number implements Comparable<Rational> {
2     // Data fields for numerator and denominator
3     private long numerator = 0;
4     private long denominator = 1;
5
6     /** Construct a rational with default properties */
7     public Rational() {
8         this(0, 1);
9     }
10
11    /** Construct a rational with specified numerator and denominator */
12    public Rational(long numerator, long denominator) {
13        long gcd = gcd(numerator, denominator);
14        this.numerator = ((denominator > 0) ? 1 : -1) * numerator / gcd;
15        this.denominator = Math.abs(denominator) / gcd;
16    }
17
18    /** Find GCD of two numbers */
19    private static long gcd(long n, long d) {
20        long n1 = Math.abs(n);
21        long n2 = Math.abs(d);
22        int gcd = 1;
23
24        for (int k = 1; k <= n1 && k <= n2; k++) {
25            if (n1 % k == 0 && n2 % k == 0)
26                gcd = k;
27        }
28
29        return gcd;
30    }
31
32    /** Return numerator */
33    public long getNumerator() {
34        return numerator;
35    }
36
37    /** Return denominator */
38    public long getDenominator() {
39        return denominator;
40    }
41
42    /** Add a rational number to this rational */
43    public Rational add(Rational secondRational) {
44        long n = numerator * secondRational.getDenominator() +
45            denominator * secondRational.getNumerator();
46        long d = denominator * secondRational.getDenominator();
47        return new Rational(n, d);
48    }
49
50    /** Subtract a rational number from this rational */
51    public Rational subtract(Rational secondRational) {
52        long n = numerator * secondRational.getDenominator()
53            - denominator * secondRational.getNumerator();
54        long d = denominator * secondRational.getDenominator();
55        return new Rational(n, d);

```

```

56     }
57
58     /** Multiply a rational number by this rational */
59     public Rational multiply(Rational secondRational) {
60         long n = numerator * secondRational.getNumerator();
61         long d = denominator * secondRational.getDenominator();
62         return new Rational(n, d);
63     }
64
65     /** Divide a rational number by this rational */
66     public Rational divide(Rational secondRational) {
67         long n = numerator * secondRational.getDenominator();
68         long d = denominator * secondRational.numerator();
69         return new Rational(n, d);
70     }
71
72     @Override
73     public String toString() {
74         if (denominator == 1)
75             return numerator + "";
76         else
77             return numerator + "/" + denominator;
78     }
79
80     @Override // Override the equals method in the Object class
81     public boolean equals(Object other) {
82         if ((this.subtract((Rational)(other))).getNumerator() == 0)
83             return true;
84         else
85             return false;
86     }
87
88     @Override // Implement the abstract intValue method in Number
89     public int intValue() {
90         return (int)doubleValue();
91     }
92
93     @Override // Implement the abstract floatValue method in Number
94     public float floatValue() {
95         return (float)doubleValue();
96     }
97
98     @Override // Implement the doubleValue method in Number
99     public double doubleValue() {
100         return numerator * 1.0 / denominator;
101     }
102
103     @Override // Implement the abstract longValue method in Number
104     public long longValue() {
105         return (long)doubleValue();
106     }
107
108     @Override // Implement the compareTo method in Comparable
109     public int compareTo(Rational o) {
110         if (this.subtract(o).getNumerator() > 0)
111             return 1;
112         else if (this.subtract(o).getNumerator() < 0)
113             return -1;
114         else
115             return 0;
116     }
117 }

```

有理数封装在 Rational 对象中。内部表示中，一个有理数表示为它的最简形式（第 13

行), 分子决定有理数的符号 (第 14 行)。分母总是正数 (第 15 行)。

gcd() 方法 (Rational 类中的第 19 ~ 30 行) 是私有的, 它是不能被其他客户程序使用的。gcd() 方法只能在 Rational 类的内部使用。gcd() 方法也是静态的, 因为它不依赖于任何一个特定的 Rational 对象。

abs(x) 方法 (Rational 类中的第 20 ~ 21 行) 在 Math 类中定义, 并返回 x 的绝对值。

两个 Rational 对象可以相互作用来完成加、减、乘、除操作。这些方法返回一个新的 Rational 对象 (第 43 ~ 70 行)。

Object 类中的 toString 方法和 equals 方法在 Rational 类中被覆盖 (第 72 ~ 86 行)。toString() 方法以 numerator/denominator (分子/分母) 的形式返回一个 Rational 对象的字符串表示, 如果分母为 1 就将它简化为 numerator。如果该有理数和另一个有理数相等, 那么方法 equals(Object other) 返回值为真。

Number 类中的抽象方法 intValue、longValue、floatValue 和 doubleValue 在 Rational 类中被实现 (第 88 ~ 106 行)。这些方法返回该有理数的 int、float 和 double 值。

Comparable 接口中的 compareTo(Object other) 方法在 Rational 类中被实现 (第 108 ~ 116 行), 用于将该有理数与另一个有理数进行比较。

提示: 在 Rational 类中提供了属性 numerator (分子) 和 denominator (分母) 的 get 方法, 但是没有提供 set 方法, 因此, 一旦创建 Rational 对象, 那么它的内容就不能改变。Rational 类是不可变的。String 类和基本类型值的包装类也都是不可变的。

提示: 可以使用两个变量表示分子和分母。也可以使用两个整数构成的数组表示分子和分母 (参见编程练习题 13.14)。尽管有理数的内部表示改变, 但是 Rational 类中的公共方法的签名是不变的。这是一个演示类的数据域应该保持私有, 以确保将类的实现和类的使用分隔开的很好的例子。

Rational 类有较为严格的限制, 容易溢出。例如, 下面的代码将显示不正确的结果, 因为分母太大了。

```
public class Test {
    public static void main(String[] args) {
        Rational r1 = new Rational(1, 123456789);
        Rational r2 = new Rational(1, 123456789);
        Rational r3 = new Rational(1, 123456789);
        System.out.println("r1 * r2 * r3 is " +
            r1.multiply(r2.multiply(r3)));
    }
}
```

```
r1 * r2 * r3 is -1/2204193661661244627
```

为了修正这个问题, 可以使用 BigInteger 表示分子和分母来实现 Rational 类 (参见编程练习题 13.15)。

复习题

13.30 给出下面代码的输出。

```
Rational r1 = new Rational(-2, 6);
System.out.println(r1.getNumerator());
System.out.println(r1.getDenominator());
System.out.println(r1.intValue());
System.out.println(r1.doubleValue());
```

13.31 下面的代码错在何处?

```
Rational r1 = new Rational(-2, 6);  
Object r2 = new Rational(1, 45);  
System.out.println(r2.compareTo(r1));
```

13.32 下面的代码错在何处?

```
Object r1 = new Rational(-2, 6);  
Rational r2 = new Rational(1, 45);  
System.out.println(r2.compareTo(r1));
```

13.33 如何使用一行代码, 而不使用 if 语句来简化程序清单 13-13 中 82 ~ 85 行的代码?

13.34 仔细地跟踪程序的执行, 给出下面代码的输出。

```
Rational r1 = new Rational(1, 2);  
Rational r2 = new Rational(1, -2);  
System.out.println(r1.add(r2));
```

13.10 类的设计原则

 **要点提示:** 类的设计原则有助于设计出合理的类。

从前面两个例子以及前面几章中的其他许多例子中, 我们已经学习了如何设计类。本节对一些设计原则进行总结。

13.10.1 内聚性

类应该描述一个单一的实体, 而所有的类操作应该在逻辑上相互配合, 支持一个一致的目的。例如: 可以设计一个类用于学生, 但不应该将学生与教职工组合在同一个类中, 因为学生和教职工是不同的实体。

如果一个实体担负太多的职责, 就应该按各自的职责分成几个类。例如: `String` 类、`StringBuffer` 类和 `StringBuilder` 类都用于处理字符串, 但是它们的职责不同。`String` 类处理不可变字符串, `StringBuilder` 类创建可变量字符串, `StringBuffer` 与 `StringBuilder` 类似, 只是 `StringBuffer` 类还包含更新字符串的同步方法。

13.10.2 一致性

遵循标准 Java 程序设计风格和命名习惯。为类、数据域和方法选取具有信息的名字。通常的风格是将数据声明置于构造方法之前, 并且将构造方法置于方法之前。

选择名字要保持一致。给类似的操作选择不同的名字并非良好的实践。例如: `length()` 方法返回 `String`、`StringBuilder` 和 `StringBuffer` 的大小。如果在这些类中给这个方法用不同的名字就不一致了。

一般来说, 应该具有一致性地提供一个公共无参构造方法, 用于构建默认实例。如果一个类不支持无参的构造方法, 要用文档写出原因。如果没有显式定义构造方法, 即假定有一个空方法体的公共默认无参构造方法。

如果不想让用户创建类的对象, 可以在类中声明一个私有的构造方法, `Math` 类就是如此。

13.10.3 封装性

一个类应该使用 `private` 修饰符隐藏其数据, 以免用户直接访问它。这使得类更易于维护。

只在希望数据域可读的情况下，才提供 `get` 方法；也只在希望数据域可更新的情况下，才提供 `set` 方法。例如：`Rational` 类为 `numerator` 和 `denominator` 提供了 `get` 方法，但是没有提供 `set` 方法，因为 `Rational` 对象是不可改变的。

13.10.4 清晰性

为使设计清晰，内聚性、一致性和封装性都是很好的设计原则。除此之外，类应该有一个很清晰的合约，从而易于解释和理解。

用户可以以各种不同组合、顺序，以及在各种环境中结合使用多个类。因此，在设计一个类时，这个类不应该限制用户如何以及何时使用该类；以一种方式设计属性，以容许用户按值的任何顺序和任何组合来设置；设计方法应该使得实现的功能与它们出现的顺序无关。例如：`Loan` 类包含属性 `loanAmount`、`numberOfYears` 和 `annualInterestRate`，这些属性的值可以按任何顺序来设置。

方法应在不产生混淆的情况下进行直观定义。例如：`String` 类中的 `substring(int beginIndex, int endIndex)` 方法就有一点混乱。这个方法返回从 `beginIndex` 到 `endIndex-1` 而不是 `endIndex` 的子串。该方法应该返回从 `beginIndex` 到 `endIndex` 的子字符串，从而更加直观。

不应该声明一个来自其他数据域的数据域。例如，下面的 `Person` 类有两个数据域：`birthDate` 和 `age`。由于 `age` 可以从 `birthDate` 导出，所以 `age` 不应该声明为数据域。

```
public class Person {
    private java.util.Date birthDate;
    private int age;
    ...
}
```

13.10.5 完整性

类是为许多不同用户的使用而设计的。为了能在一个广泛的应用中使用，一个类应该通过属性和方法提供多种方案以适应用户的不同需求。例如：为满足不同的应用需求，`String` 类包含了 40 多种很实用的方法。

13.10.6 实例和静态

依赖于类的具体实例的变量或方法必须是一个实例变量或方法。如果一个变量被类的所有实例所共享，那就应该将它声明为静态的。例如：在程序清单 9-8 中，`CircleWithPrivateDataFields` 中的变量 `numberOfObjects` 被 `CircleWithPrivateDataFields` 类的所有对象共享。因此，它被声明为静态的。如果方法不依赖于某个具体的实例，那就应该将它声明为静态方法。例如：`CircleWithPrivateDataFields` 中的 `getNumberOfObjects()` 方法没有绑定到任何具体实例，因此，它被声明为静态方法。

应该总是使用类名（而不是引用变量）引用静态变量和方法，以增强可读性并避免错误。

不要从构造方法中传入参数来初始化静态数据域。最好使用 `set` 方法改变静态数据域。图 a 中的类最好用图 b 中的代替。

```
public class Something {
    private int t1;
    private static int t2;

    public Something(int t1, int t2) {
        ...
    }
}
```

a)

```
public class Something {
    private int t1;
    private static int t2;

    public Something(int t1) {
        ...
    }

    public static void setT2(int t2) {
        Something.t2 = t2;
    }
}
```

b)

实例和静态是面向对象程序设计不可或缺的部分。数据域或方法要么是实例的，要么是静态的。不要错误地忽视了静态数据域或方法。常见的设计错误是将本应该声明为静态方法的方法声明为实例方法。例如：用于计算 n 的阶乘的 `factorial(int n)` 方法应该定义为静态的，因为它不依赖于任何具体实例。

构造方法永远都是实例方法，因为它用来创建具体实例的。一个静态变量或方法可以从实例方法中调用，但是不能从静态方法中调用实例变量或方法。

13.10.7 继承与聚合

继承和聚合之间的差异，就是 `is-a`（是一种）和 `has-a`（具有）之间的关系。例如，苹果是一种水果；因此，可以使用继承来对 `Apple` 类和 `Fruit` 类之间的关系进行建模。人具有名字；因此，可以使用聚合来对 `Person` 类和 `Name` 类之间的关系建模。

13.10.8 接口和抽象类

接口和抽象类都可以用于为对象指定共同的行为。如何决定是采用接口还是类呢？通常，比较强的 `is-a`（是一种）关系清晰地描述了父子关系，应该采用类来建模。例如，因为桔子是一种水果，它们的关系就应该采用类的继承关系来建模。弱的 `is-a` 关系，也称为 `is-kind-of`（是一类）关系，表明一个对象拥有某种属性。弱的 `is-a` 关系可以使用接口建模。例如，所有的字符串都是可以比较的，因此 `String` 类实现了 `Comparable` 接口。圆或者矩形是一个几何对象，因此 `Circle` 可以设计为 `GeometricObject` 的子类。圆有不同的半径，并且可以基于半径进行比较，因此 `Circle` 可以实现 `Comparable` 接口。

接口比抽象类更加灵活，因为一个子类只能继承一个父类，但是却可以实现任意个数的接口。然而，接口不能具有具体的方法。可以结合接口和抽象类的优点，创建一个接口，使用一个抽象类来实现它。可以视其方便使用接口或者抽象类。我们将在第 20 章给出这类设计的实例。

复习题

13.35 描述类的设计原则。

关键术语

abstract class (抽象类)

abstract method (抽象方法)

deep copy (深复制)

interface (接口)

marker interface (标记接口)

shallow copy (浅复制)

subinterface (子接口)

本章小结

1. 抽象类和常规类一样，都有数据和方法，但是不能用 `new` 操作符创建抽象类的实例。
2. 非抽象类中不能包含抽象方法。如果抽象类的子类没有实现所有被继承的父类抽象方法，就必须将该子类也定义为抽象类。
3. 包含抽象方法的类必须是抽象类。但是，抽象类可以不包含抽象的方法。
4. 即使父类是具体的，子类也可以是抽象的。
5. 接口是一种与类相似的结构，只包含常量和抽象方法。接口在许多方面与抽象类很相近，但抽象类除了包含常量和抽象方法外，还可以包含变量和具体方法。
6. 在 Java 中，接口被认为是一种特殊的类。就像常规类一样，每个接口都被编译为独立的字节码文件。
7. 接口 `java.lang.Comparable` 定义了 `compareTo` 方法。Java 类库中的许多类都实现了 `Comparable`。
8. 接口 `java.lang.Cloneable` 是一个标记接口。实现 `Cloneable` 接口的类的对象是可克隆的。
9. 个类仅能继承一个父类，但一个类却可以实现一个或多个接口。
10. 一个接口可以继承一个或多个接口。

测试题

回答本章位于 www.cs.armstrong.edu/liang/intro10e/quiz.html 中的测试题。

编程练习题

13.2 ~ 13.3 节

**13.1 (三角形类) 设计一个扩展自抽象类 `GeometricObject` 的新的 `Triangle` 类。绘制 `Triangle` 类和 `GeometricObject` 类的 UML 图并实现 `Triangle` 类。编写一个测试程序，提示用户输入三角形的三条边、一种颜色以及一个表明该三角形是否填充的布尔值。程序应该根据用户的输入，使用这些边以及颜色和是否填充的信息，创建一个 `Triangle` 对象。程序应该显示面积、周长、颜色以及真或者假来表明是否被填充。

**13.2 (打乱 `ArrayList`) 编写以下方法，打乱 `ArrayList` 里面保存的数字。

```
public static void shuffle(ArrayList<Number> list)
```

**13.3 (排序 `ArrayList`) 编写以下方法，对 `ArrayList` 里面保存的数字进行排序。

```
public static void sort(ArrayList<Number> list)
```

**13.4 (显示日历) 重写程序清单 6-12 中的 `PrintCalendar` 类，使用 `Calendar` 和 `GregorianCalendar` 类显示一个给定月份的日历。你的程序从命令行得到月份和年份的输入，例如：

```
java Exercise13_04 5 2016
```

这个会显示如图 13-9 中的日历。

也可以不输入年份来运行程序。这种情况下，年份就是当前年份。如果不指定月份和年份来运行程序，那么就是指当前月份。

13.4 ~ 13.8 节

**13.5 (将 `GeometricObject` 类变成可比较的) 修改 `GeometricObject` 类以实现 `Comparable` 接口，并且在 `GeometricObject` 类中定义一个静态的求两个

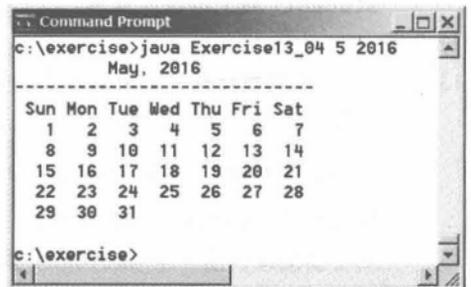


图 13-9 程序显示 2016 年五月的日历

GeometricObject 对象中较大者的 max 方法。画出 UML 图并实现这个新的 GeometricObject 类。编写一个测试程序，使用 max 方法求两个圆中的较大者和两个矩形中的较大者。

- *13.6 (ComparableCircle 类) 创建名为 ComparableCircle 的类，它继承自 Circle 类，并实现 Comparable 接口。画出 UML 图并实现 compareTo 方法，使其根据面积比较两个圆。编写一个测试程序求出 ComparableCircle 对象的两个实例中的较大者。
- *13.7 (可着色接口 Colorable) 设计一个名为 Colorable 的接口，其中有名为 howToColor() 的 void 方法。可着色对象的每个类必须实现 Colorable 接口。设计一个名为 Square 的类，继承自 GeometricObject 类并实现 Colorable 接口。实现 howToColor 方法，显示一个消息 Color all four sides (给所有的四条边着色)。

画出包含 Colorable、Square 和 GeometricObject 的 UML 图。编写一个测试程序，创建有五个 GeometricObject 对象的数组。对于数组中的每个对象而言，如果对象是可着色的，那就调用 howToColor 方法。

- *13.8 (修改 MyStack 类) 重写程序清单 11-10 中的 MyStack 类，执行 list 域的深度复制。
- *13.9 (将 Circle 类改成可比较的) 改写程序清单 13-2 中的 Circle 类，它继承自 GeometricObject 类并实现 Comparable 接口。覆盖 Object 类中的 equals 方法。当两个 Circle 对象半径相等时，则这两个 Circle 对象是相同的。画出包括 Circle、GeometricObject 和 Comparable 的 UML 图。
- *13.10 (将 Rectangle 类变成可比较的) 改写程序清单 13-3 的 Rectangle 类，它继承自 GeometricObject 类并实现 Comparable 接口。覆盖 Object 类中的 equals 方法。当两个 Rectangle 对象面积相同时，则这两个对象是相同的。画出包括 Rectangle、GeometricObject 和 Comparable 的 UML 图。
- *13.11 (八边形类 Octagon) 编写一个名为 Octagon 的类，它继承自 GeometricObject 类并实现 Comparable 和 Cloneable 接口。假设八边形八条边的边长都相等。它的面积可以使用下面的公式计算：

$$\text{面积} = (2 + 4/\sqrt{2}) \times \text{边长} \times \text{边长}$$

画出包括 Octagon、GeometricObject、Comparable 和 Cloneable 的 UML 图。编写一个测试程序，创建一个边长值为 5 的 Octagon 对象，然后显示它的面积和周长。使用 clone 方法创建一个新对象，并使用 compareTo 方法比较这两个对象。

- *13.12 (求几何对象的面积之和) 编写一个方法，求数组中所有几何对象的面积之和。方法签名如下：

```
public static double sumArea(GeometricObject[] a)
```

编写测试程序，创建四个对象（两个圆和两个矩形）的数组，然后使用 sumArea 方法求它们的总面积。

- *13.13 (使得 Course 类可复制) 重写程序清单 10-6 中的 Course 类，增加一个 clone 方法，执行 students 域上的深度复制。

13.9 节

- *13.14 (演示封装的好处) 使用新的分子分母的内部表达改写 13.13 节中的 Rational 类。创建有两个整数的数组，如下所示：

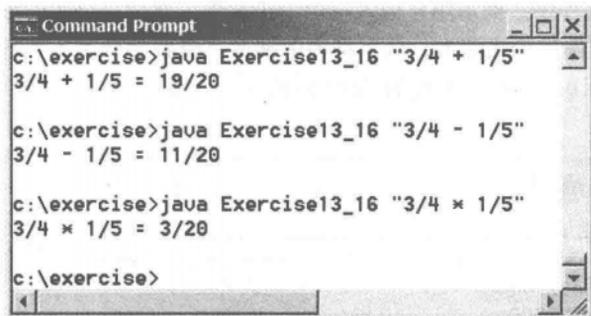
```
private long[] r = new long[2];
```

使用 r[0] 表示分子，使用 r[1] 表示分母。在 Rational 类中的方法签名没有改变，因此，无须重新编译，前一个 Rational 类的客户端应用程序可以继续使用这个新的 Rational 类。

- *13.15 (在 Rational 类中使用 BigInteger) 使用 BigInteger 表示分子和分母，重新设计和实现 13.13 节中的 Rational 类。

*13.16 (创建一个有理数的计算器) 编写一个类似于程序清单 7-9 的程序。这里不使用整数, 而是使用有理数, 如图 13-10a 所示。

需要使用在 10.10.3 节中介绍的 String 类中的 split 方法来获取分子字符串和分母字符串, 并使用 Integer.parseInt 方法将字符串转换为整数。



```

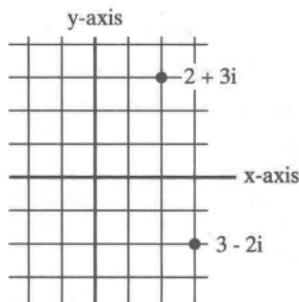
c:\exercise>java Exercise13_16 "3/4 + 1/5"
3/4 + 1/5 = 19/20

c:\exercise>java Exercise13_16 "3/4 - 1/5"
3/4 - 1/5 = 11/20

c:\exercise>java Exercise13_16 "3/4 * 1/5"
3/4 * 1/5 = 3/20

c:\exercise>
  
```

a) 程序从命令行得到三个参数(操作数 1、操作符、操作数 2), 显示该表达式以及算数运算的结果



b) 复数可以解释为一个平面上的点

图 13-10

*13.17 (数学: Complex 类) 一个复数是一个形式为 $a+bi$ 的数, 这里的 a 和 b 都是实数, i 是 $\sqrt{-1}$ 的平方根。数字 a 和 b 分别称为复数的实部和虚部。可以使用下面的公式完成复数的加、减、乘、除:

$$a + bi + c + di = (a + c) + (b + d)i$$

$$a + bi - (c + di) = (a - c) + (b - d)i$$

$$(a + bi) * (c + di) = (ac - bd) + (bc + ad)i$$

$$(a + bi) / (c + di) = (ac + bd) / (c^2 + d^2) + (bc - ad)i / (c^2 + d^2)$$

还可以使用下面的公式得到复数的绝对值:

$$|a + bi| = \sqrt{a^2 + b^2}$$

(复数可以解释为一个平面上的点, 将 (a, b) 值作为该点的坐标。复数的绝对值是该点到原点的距离, 如图 13-10b 所示。)

设计一个名为 Complex 的复数来表示复数以及完成复数运算的 add、subtract、multiply、divide 和 abs 方法, 并且覆盖 toString 方法以返回一个表示复数的字符串。方法 toString 返回字符串 a+bi。如果 b 是 0, 那么它只返回 a。Complex 类应该也实现 Cloneable 接口。

提供三个构造方法 Complex(a,b)、Complex(a) 和 Complex()。Complex() 创建数字 0 的 Complex 对象, 而 Complex(a) 创建一个 b 为 0 的 Complex 对象。还提供 getRealPart() 和 getImaginaryPart() 方法以分别返回复数的实部和虚部。

编写一个测试程序, 提示用户输入两个复数, 然后显示它们做加、减、乘、除之后的结果。下面是一个运行示例:

```

Enter the first complex number: 3.5 5.5
Enter the second complex number: -3.5 1
(3.5 + 5.5i) + (-3.5 + 1.0i) = 0.0 + 6.5i
(3.5 + 5.5i) - (-3.5 + 1.0i) = 7.0 + 4.5i
(3.5 + 5.5i) * (-3.5 + 1.0i) = -17.75 + -13.75i
(3.5 + 5.5i) / (-3.5 + 1.0i) = -0.5094 + -1.7i
|(3.5 + 5.5i)| = 6.519202405202649
  
```

13.18 (使用 Rational 类) 编写程序, 使用 Rational 类计算下面的求和数列:

$$\frac{1}{2} + \frac{2}{3} + \frac{3}{4} + \dots + \frac{98}{99} + \frac{99}{100}$$

你将会发现输出是不正确的，因为整数溢出（太大了）。为了解决这个问题，参见编程练习题 13.15。

- 13.19 (将十进制数转化为分数) 编写一个程序，提示用户输入一个十进制数，然后以分数的形式显示该数字。提示：将十进制数以字符串的形式读入，从字符串中抽取其整数部分和小数部分，然后运用编程练习题 13.15 中使用 `BigInteger` 实现的 `Rational` 类，来获得该十进制数的有理数。这里是一些运行示例：

```
Enter a decimal number: 3.25 --Enter
The fraction number is 13/4
```

```
Enter a decimal number: -0.45452 --Enter
The fraction number is -11363/25000
```

- 13.20 (数学：求解二元方程) 重写编程练习题 3.1，如果行列式小于 0，则使用编程练习题 13.17 中的 `Complex` 类来得到虚根。这里是一些运行示例：

```
Enter a, b, c: 1 3 1 --Enter
The roots are -0.381966 and -2.61803
```

```
Enter a, b, c: 1 2 1 --Enter
The root is -1
```

```
Enter a, b, c: 1 2 3 --Enter
The roots are -1.0 + 1.4142i and -1.0 + -1.4142i
```

- 13.21 (代数：顶点式方程) 抛物线方程可以表达为标准形式 ($y = ax^2 + bx + c$) 或者顶点式 ($y = a(x-h)^2 + k$)。编写一个程序，提示用户输入标准形式下的整数 a 、 b 和 c 值，显示顶点式下面的 h 和 k 值。这里是一些运行示例：

```
Enter a, b, c: 1 3 1 --Enter
h is -3/2 k is -5/4
```

```
Enter a, b, c: 2 3 4 --Enter
h is -3/4 k is 23/8
```

JavaFX 基础

教学目标

- 区分 JavaFX, Swing 和 AWT (14.2 节)。
- 编写一个简单的 JavaFX 程序, 理解舞台、场景和节点之间的关系 (14.3 节)。
- 使用面板、UI 组件和形状创建用户界面 (14.4 节)。
- 通过属性绑定自动更新属性值 (14.5 节)。
- 使用节点的通用属性 `style` 和 `rotate` (14.6 节)。
- 使用 `Color` 类创建颜色 (14.7 节)。
- 使用 `Font` 类创建字体 (14.8 节)。
- 使用 `Image` 类创建图像以及使用 `ImageView` 类创建图像视图 (14.9 节)。
- 使用 `Pane`、`StackPane`、`FlowPane`、`GridPane`、`BorderPane`、`HBox` 和 `VBox` 布局节点 (14.10 节)。
- 使用 `Text` 类显示文本以及使用 `Line`、`Circle`、`Rectangle`、`Ellipse`、`Arc`、`Polygon` 和 `Polyline` 创建形状 (14.11 节)。
- 开发一个可重用的 GUI 组件 `CLockPane` 显示一个模拟时钟 (14.12 节)。

14.1 引言

 **要点提示:** JavaFX 是学习面向对象编程的优秀教学工具。

JavaFX 是开发 Java GUI 程序的新框架。JavaFX API 是如何应用面向对象原则的优秀范例。本章起到两方面的作用。首先, 给出了 JavaFX 编程的基础。其次, 使用 JavaFX 来展示面向对象设计和编程。具体而言, 本章介绍 JavaFX 框架, 并讨论 JavaFX GUI 组件以及它们的关系。你将学到如何采用布局面板、按钮、标签、文本域、颜色、字体、图像、图像视图以及形状来开发简单的 GUI 程序。

14.2 JavaFX 与 Swing 以及 AWT 的比较

 **要点提示:** JavaFX 平台取代了 Swing 和 AWT, 用于开发富因特网应用。

当引入 Java 时, GUI 类使用一个称为抽象窗体工具包 (AWT) 的库。AWT 开发简单的图形用户界面尚可, 但是不适合开发综合的 GUI 项目。另外, AWT 容易被特定于平台的错误影响。之后 AWT 用户界面组件被一个更健壮、功能更齐全和更灵活的库所替代, 即 Swing 组件。Swing 组件使用 Java 代码在画布上直接绘制。Swing 组件更少依赖目标平台, 且使用更少的本地 GUI 资源。Swing 用于开发桌面 GUI 应用。现在, 它被一个全新的 GUI 平台 JavaFX 所替代。JavaFX 融入了现代 GUI 技术以方便开发富因特网应用 (RIA)。富因特网应用是一种 Web 应用, 可以表现一般桌面应用具有的特点和功能。JavaFX 应用可以无缝地在桌面或者 Web 浏览器中运行。另外, JavaFX 为支持触摸的设备提供多点触控支持, 如平板和智能手机。JavaFX 具有内建的 2D、3D、动画支持, 以及视频和音频的回放功能, 可以作为一个应用独立运行或者在浏览器中运行。

本书采用 JavaFX 讲解 Java GUI 编程出于以下两个原因。首先,对于 Java 编程入门者而言,JavaFX 更容易学习和使用。其次,Swing 原则上已消亡,因为它不会再得到任何增强。JavaFX 是一个新的 GUI 工具,用于在台式计算机、手持设备和 Web 上开发跨平台的富因特网应用。

☛ 复习题

- 14.1 解释 JavaGUI 技术的演变。
- 14.2 解释为何本书采用 JavaFX 教授 Java GUI。

14.3 JavaFX 程序的基本结构

🔑 要点提示: 抽象类 `javafx.application.Application` 定义编写 JavaFX 程序的基本框架。

从编写一个简单的 JavaFX 程序着手,来演示一个 JavaFX 程序的基本结构。每个 JavaFX 程序定义在一个继承自 `javafx.application.Application` 的类中,如程序清单 14-1 所示。

程序清单 14-1 MyJavaFX.java

```

1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Button;
4  import javafx.stage.Stage;
5
6  public class MyJavaFX extends Application {
7      @Override // Override the start method in the Application class
8      public void start(Stage primaryStage) {
9          // Create a scene and place a button in the scene
10         Button btOK = new Button("OK");
11         Scene scene = new Scene(btOK, 200, 250);
12         primaryStage.setTitle("MyJavaFX"); // Set the stage title
13         primaryStage.setScene(scene); // Place the scene in the stage
14         primaryStage.show(); // Display the stage
15     }
16
17     /**
18     * The main method is only needed for the IDE with limited
19     * JavaFX support. Not needed for running from the command line.
20     */
21     public static void main(String[] args) {
22         Application.launch(args);
23     }
24 }

```

可以从命令行窗体或者从一个 IDE (如 NetBeans 或者 Eclipse) 中测试和运行程序。程序的运行示例如图 14-1 所示。补充材料 II.F ~ H 给出了从一个命令窗体、NetBeans 以及 Eclipse 中运行 JavaFX 程序的提示。JavaFX 程序可以独立运行或者在 Web 浏览器中运行。在 Web 浏览器中运行 JavaFX 程序请参考补充材料 III.Z。

`launch` 方法 (第 22 行) 是一个定义在 `Application` 类中的静态方法,用于启动一个独立的 JavaFX 应用。如果你从命令行运行程序, `main` 方法 (第 21 ~ 23 行) 不是必需的。当从一个不完全支持 JavaFX 的 IDE 中启动 JavaFX 程序的时候,可能会需要 `main` 方法。当运行一个没有 `main` 方法的 JavaFX 应



图 14-1 一个在窗体中显示一个按钮的简单 JavaFX 程序

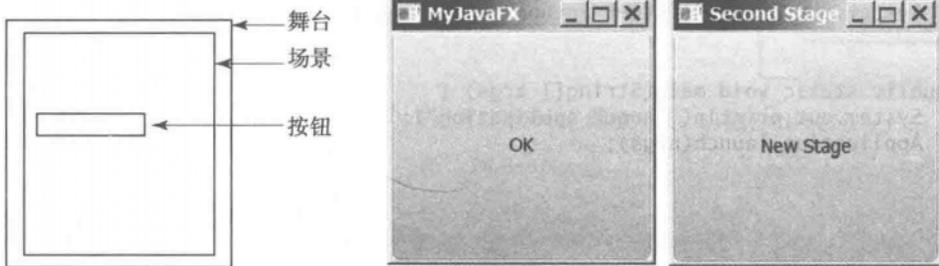
用时, JVM 自动调用 `launch` 方法以运行应用程序。

主类重写了定义在 `javafx.application.Application` 类中的 `start` 方法 (第 8 行)。当一个 JavaFX 应用启动时, JVM 使用它的无参构造方法来创建类的一个实例, 同时调用其 `start` 方法。`start` 方法一般用于将 UI 组件放入一个场景, 并且在舞台中显示该场景, 如图 14-2a 所示。

第 10 行创建一个 `Button` 对象并将其置于一个 `Scene` 对象中 (第 11 行)。一个 `Scene` 对象可以使用构造方法 `Scene(node, width, height)` 创建。这个构造方法指定了场景的宽度和高度并且将节点置于一个场景中。

一个 `Stage` 对象是一个窗体。当应用程序启动的时候, 一个称为主舞台的 `Stage` 对象由 JVM 自动创建。第 13 行将场景设定在主舞台中, 第 14 行显示主舞台。JavaFX 应用剧院的类比来命名 `Stage` 和 `Scene` 类。可以认为舞台是一个支持场景的平台, 节点如同在场景中演出的演员。

根据需要, 可以创建其他舞台。程序清单 14-2 中的 JavaFX 程序显示了两个舞台, 如图 14-2b 所示。



a) Stage 是一个窗体, 用于显示包含了节点的场景

b) 一个 JavaFX 程序可以显示多个舞台

图 14-2

程序清单 14-2 MultipleStageDemo.java

```

1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Button;
4  import javafx.stage.Stage;
5
6  public class MultipleStageDemo extends Application {
7      @Override // Override the start method in the Application class
8      public void start(Stage primaryStage) {
9          // Create a scene and place a button in the scene
10         Scene scene = new Scene(new Button("OK"), 200, 250);
11         primaryStage.setTitle("MyJavaFX"); // Set the stage title
12         primaryStage.setScene(scene); // Place the scene in the stage
13         primaryStage.show(); // Display the stage
14
15         Stage stage = new Stage(); // Create a new stage
16         stage.setTitle("Second Stage"); // Set the stage title
17         // Set a scene with a button in the stage
18         stage.setScene(new Scene(new Button("New Stage"), 100, 100));
19         stage.show(); // Display the stage
20     }
21 }

```

请注意, 在程序清单中 `main` 方法被省略了, 因为对于每一个 JavaFX 应用它都是一样

的。从现在开始，为简明起见，main 方法都不会列在 JavaFX 源代码中。

默认情况下，用户可以改变舞台的大小。如要防止用户改变舞台大小，调用 `stage.setResizable(false)` 实现。

复习题

- 14.3 如何定义 JavaFX 主类？start 方法的签名是什么？什么是舞台？什么是主舞台？主舞台是自动生成的吗？如何显示一个舞台？可以阻止用户改变舞台大小吗？在程序清单 14-1 中，可以将第 22 行的 `Application.launch(args)` 替代为 `launch(args)` 吗？
- 14.4 请给出下面 JavaFX 程序的输出结果：

```
import javafx.application.Application;
import javafx.stage.Stage;

public class Test extends Application {
    public Test() {
        System.out.println("Test constructor is invoked");
    }
    @Override // Override the start method in the Application class
    public void start(Stage primaryStage) {
        System.out.println("start method is invoked");
    }

    public static void main(String[] args) {
        System.out.println("launch application");
        Application.launch(args);
    }
}
```

14.4 面板、UI 组件以及形状

要点提示：面板、UI 组件和形状是 Node 的子类型。

当运行程序清单 14-1 中的 MyJavaFX 时，窗体如图 14-1 所示。按钮总是位于场景的中间并且总是占据整个窗体，无论你是否改变窗体的大小。可以通过设置按钮的位置和大小属性来解决这个问题。然而，一个更好的方法是使用称为面板的容器类，从而自动地将节点布局在一个希望的位置和大小。将节点置于一个面板中，然后将面板再置于一个场景中。节点是可视化组件，比如一个形状、一个图像视图、一个 UI 组件或者一个面板。形状是指文字、直线、圆、椭圆、矩形、弧、多边形、折线等。UI 组件是指标签、按钮、复选框、单选按钮、文本域、文本输入区域等。一个场景可以显示在一个舞台中，如图 14-3a 所示。Stage、Scene、Node、Control 以及 Pane 之间的关系可以采用 UML 图来表达，如图 14-3b 所示。请注意，Scene 可以包含 Control 或者 Pane，但是不能包含 Shape 和 ImageView。Pane 可以包含 Node 的任何子类型。可以使用构造方法 `Scene(Parent, width, height)` 或者 `Scene(Parent)` 创建 Scene。后一个构造方法中场景的尺寸将自动确定。Node 的每个子类都有一个无参的构造方法，用于创建一个默认节点。

程序清单 14-3 给出了一个程序示例，将一个按钮置于一个面板中，如图 14-4 所示。

程序创建一个 StackPane (第 11 行)，然后将一个按钮作为面板的组成部分 (child) 加入 (第 12 行)。getChildren() 方法返回 `javafx.collections.ObservableList` 的一个实例。ObservableList 类似于 ArrayList，用于存储一个元素集合。调用 `add(e)` 将一个元素加入

列表。StackPane 将节点置于面板中央，并且置于其他节点之上。这里只有一个节点在面板中。StackPane 会得到一个节点的偏好尺寸。所以我们看到按钮以它的偏好尺寸显示。

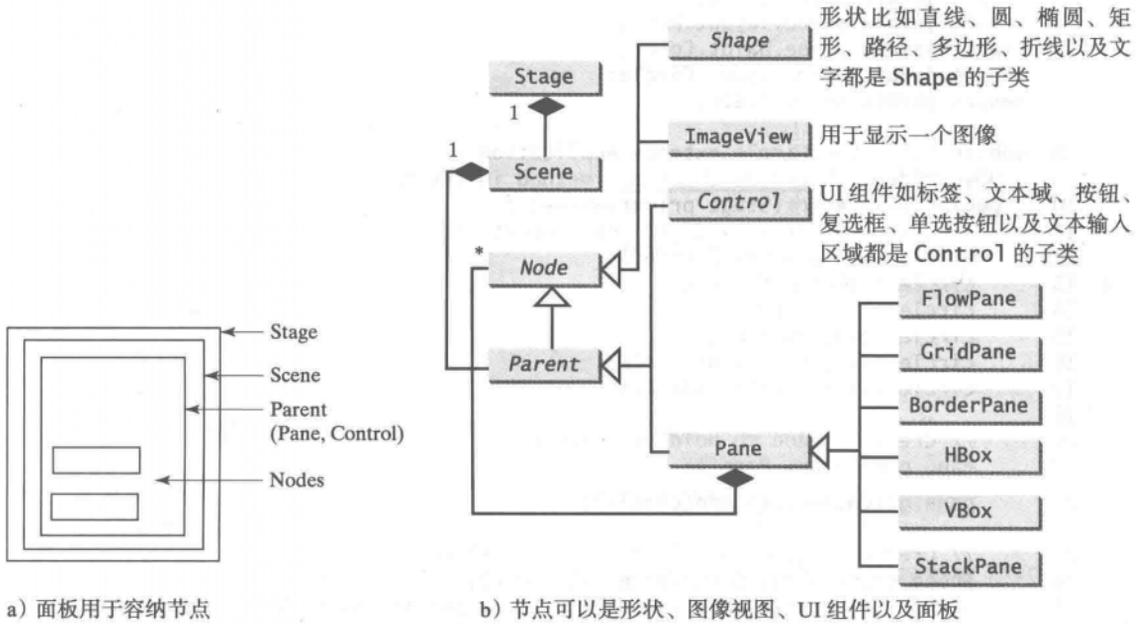


图 14-3

程序清单 14-3 ButtonInPane.java

```

1 import javafx.application.Application;
2 import javafx.scene.Scene;
3 import javafx.scene.control.Button;
4 import javafx.stage.Stage;
5 import javafx.scene.layout.StackPane;
6
7 public class ButtonInPane extends Application {
8     @Override // Override the start method in the Application class
9     public void start(Stage primaryStage) {
10         // Create a scene and place a button in the scene
11         StackPane pane = new StackPane();
12         pane.getChildren().add(new Button("OK"));
13         Scene scene = new Scene(pane, 200, 50);
14         primaryStage.setTitle("Button in a pane"); // Set the stage title
15         primaryStage.setScene(scene); // Place the scene in the stage
16         primaryStage.show(); // Display the stage
17     }
18 }
    
```

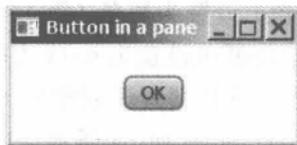


图 14-4 一个按钮置于面板的中间

程序清单 14-4 给出了一个在面板中央显示圆的示例，如图 14-5a 所示。

程序清单 14-4 ShowCircle.java

```

1 import javafx.application.Application;
2 import javafx.scene.Scene;
3 import javafx.scene.layout.Pane;
4 import javafx.scene.paint.Color;
5 import javafx.scene.shape.Circle;
6 import javafx.stage.Stage;
7
8 public class ShowCircle extends Application {
9     @Override // Override the start method in the Application class
10    public void start(Stage primaryStage) {
11        // Create a circle and set its properties
12        Circle circle = new Circle();
13        circle.setCenterX(100);
14        circle.setCenterY(100);
15        circle.setRadius(50);
16        circle.setStroke(Color.BLACK);
17        circle.setFill(Color.WHITE);
18
19        // Create a pane to hold the circle
20        Pane pane = new Pane();
21        pane.getChildren().add(circle);
22
23        // Create a scene and place it in the stage
24        Scene scene = new Scene(pane, 200, 200);
25        primaryStage.setTitle("ShowCircle"); // Set the stage title
26        primaryStage.setScene(scene); // Place the scene in the stage
27        primaryStage.show(); // Display the stage
28    }
29 }

```

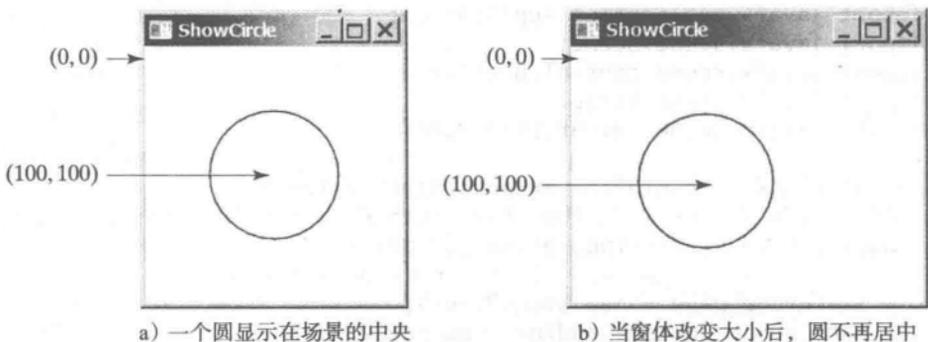


图 14-5

程序创建了一个 `Circle` (第 12 行) 并将它的圆心设置在 (100, 100) (第 13 ~ 14 行), 同时这里也是场景的中央, 因为创建场景时给出的宽度和高度都是 200 (第 24 行)。圆的半径设为 50 (第 15 行)。请注意, Java 图形的尺寸单位都使用像素。

笔划颜色 (即画圆所采用的颜色) 设置为黑色 (第 16 行)。填充颜色 (即用于填充圆的颜色) 设置为白色 (第 17 行)。可以将颜色设置为 `null` 表明采用无色。

程序创建了一个 `Pane` (第 20 行) 并将圆置于面板中 (第 21 行)。请注意, 在 Java 的坐标系中, 面板左上角的坐标是 (0,0), 如图 14-6a 所示, 这不同于传统坐标系中 (0,0) 位于窗体的中央, 如图 14-6b 所示。在 Java 坐标系中, x 坐标从左到右递增, y 坐标从上到下递增。

面板置于场景中 (第 24 行), 然后场景设置于舞台中 (第 26 行)。圆显示在舞台中央,

如图 14-5a 所示。然而，如果改变窗体的大小，圆不再居中，如图 14-5b 所示。当窗体改变大小的时候为了依然显示圆居中，圆心的 x 和 y 坐标需要重新设置在面板的中央。可以通过设置属性绑定来达到效果，具体方式将在下一节中介绍。

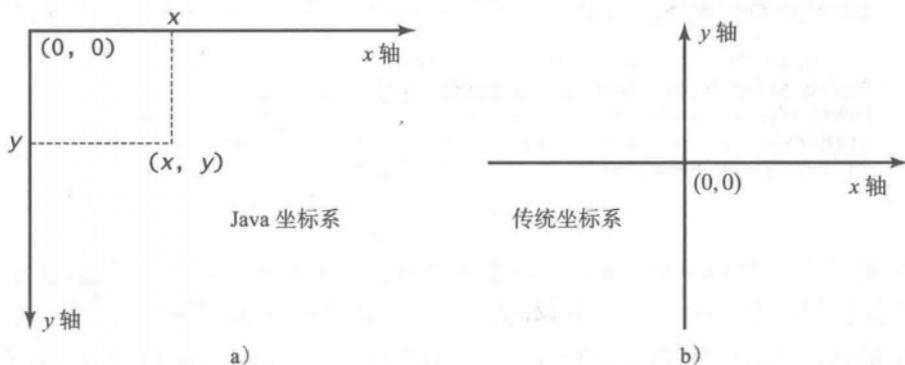


图 14-6 Java 坐标系统使用像素作为尺寸单位，(0, 0) 位于左上角

复习题

- 14.5 如何创建 Scene 对象？如何在舞台中设置场景？如何将一个圆置于场景中？
- 14.6 什么是面板？什么是节点？如何将一个节点置于面板中？可以直接将 Shape 或者 ImageView 置于 Scene 中吗？可以将 Control 或者 Pane 直接置于 Scene 中吗？
- 14.7 如何创建 Circle？如何设置它的圆心位置以及半径？如何设置它的笔划颜色以及填充颜色？

14.5 属性绑定

要点提示：可以将一个目标对象绑定到源对象中。源对象的修改将自动反映到目标对象中。

JavaFX 引入了一个称为属性绑定的新概念，可以将一个目标对象和一个源对象绑定。如果源对象中的值改变了，目标对象也将自动改变。目标对象称为绑定对象或者绑定属性，源对象称为可绑定对象或者可观察对象。如前面程序清单所讨论的，当窗体改变大小的时候，圆不再居中。窗体改变大小后为了圆依然显示在中央，圆心的 x 坐标和 y 坐标需要重新设置到面板的中央。可以通过将 `centerX` 和 `centerY` 分别绑定到面板的 `width/2` 以及 `height/2` 上面实现，如程序清单 14-5 所示。

程序清单 14-5 ShowCircleCentered.java

```

1 import javafx.application.Application;
2 import javafx.scene.Scene;
3 import javafx.scene.layout.Pane;
4 import javafx.scene.paint.Color;
5 import javafx.scene.shape.Circle;
6 import javafx.stage.Stage;
7
8 public class ShowCircleCentered extends Application {
9     @Override // Override the start method in the Application class
10    public void start(Stage primaryStage) {
11        // Create a pane to hold the circle
12        Pane pane = new Pane();
13
14        // Create a circle and set its properties
15        Circle circle = new Circle();

```

```

16 circle.centerXProperty().bind(pane.widthProperty().divide(2));
17 circle.centerYProperty().bind(pane.heightProperty().divide(2));
18 circle.setRadius(50);
19 circle.setStroke(Color.BLACK);
20 circle.setFill(Color.WHITE);
21 pane.getChildren().add(circle); // Add circle to the pane
22
23 // Create a scene and place it in the stage
24 Scene scene = new Scene(pane, 200, 200);
25 primaryStage.setTitle("ShowCircleCentered"); // Set the stage title
26 primaryStage.setScene(scene); // Place the scene in the stage
27 primaryStage.show(); // Display the stage
28 }
29 }

```

Circle 类具有一个 `centerX` 属性，用于表示圆心的 x 坐标。如同许多 JavaFX 类中的属性一样，在属性绑定中，该属性既可以作为目标，也可以作为源。目标监听源中的变化，一旦源中发生变化，目标将自动更新自身。一个目标采用 `bind` 方法和源进行绑定，如下所示：

```
target.bind(source);
```

`bind` 方法在 `javafx.beans.property.Property` 接口中定义。绑定属性是 `javafx.beans.property.Property` 的一个实例。源对象是 `javafx.beans.value.ObservableValue` 接口的一个实例。`ObservableValue` 是一个包装了值的实体，并且允许值发生改变时被观察到。

JavaFX 为基本类型和字符串定义绑定属性。对于 `double/float/long/int/boolean` 类型的值，它的绑定属性类型是 `DoubleProperty/FloatProperty/LongProperty/IntegerProperty/BooleanProperty`。对于字符串而言，它的绑定属性类型是 `StringProperty`。这些属性同时也是 `ObservableValue` 的子类型。因此它们也可以作为源对象来进行属性绑定。

一般而言，JavaFX 类（如 `Circle`）中的每个绑定属性（如 `centerX`）都有一个获取方法（如 `getCenterX()`）和设置方法（如 `setCenterX(double)`）用于返回和设置属性的值。同时还有一个获取方法返回属性本身。这个方法的命名习惯是在属性名称后面加上单词 `Property`。举例来说，`centerX` 的属性获取方法是 `centerXProperty()`。我们将 `getCenterX()` 称为值的获取方法，将 `setCenterX(double)` 称为值的设置方法，而将 `centerXProperty()` 称为属性获取方法。请注意，`getCenterX()` 返回一个 `double` 值，而 `centerXProperty()` 返回一个 `DoubleProperty` 类型的对象。图 14-7a 演示了在类中定义一个绑定属性的习惯用法，图 14-7b 演示了一个具体的示例，其中 `centerX` 是 `DoubleProperty` 类型的一个绑定属性。

```

public class SomeClassName {
    private PropertyType x;

    /** Value getter method */
    public propertyValueType getX() { ... }

    /** Value setter method */
    public void setX(propertyValueType value) { ... }

    /** Property getter method */
    public PropertyType
    xProperty() { ... }
}

```

a) `x` 是一个绑定属性

```

public class Circle {
    private DoubleProperty centerX;

    /** Value getter method */
    public double getCenterX() { ... }

    /** Value setter method */
    public void setCenterX(double value) { ... }

    /** Property getter method */
    public DoubleProperty centerXProperty() { ... }
}

```

b) `centerX` 是一个绑定属性

图 14-7 一个绑定属性具有一个值获取方法、设置方法以及属性获取方法

程序清单 14-5 和程序清单 14-4 的唯一不同之处是，它将 `circle` 的 `centerX` 和 `centerY` 属性绑定到了 `pane` 的宽度和高度的一半上（第 16 ~ 17 行）。请注意，`circle.centerXProperty()` 返回 `centerX`，`pane.widthProperty()` 返回 `width`。`centerX` 和 `width` 都是 `DoubleProperty` 类型的绑定属性。数值类型的绑定属性类（如 `DoubleProperty` 和 `IntegerProperty`）具有 `add`、`subtract`、`multiply` 以及 `divide` 方法，用于对一个绑定属性中的值进行加、减、乘、除，并返回一个新的可观察属性。因此，`pane.widthProperty().divide(2)` 返回一个代表 `pane` 的一半宽度的新的可观察属性。语句

```
circle.centerXProperty().bind(pane.widthProperty().divide(2));
```

和下面的语句相同：

```
centerX.bind(width.divide(2));
```

由于 `centerX` 绑定到 `width.divide(2)` 上，因此当 `pane` 的宽度改变的时候，`centerX` 自动更新自身以匹配 `pane` 的一半宽度。

程序清单 14-6 给出了演示绑定的另外一个示例。

程序清单 14-6 BindingDemo.java

```
1 import javafx.beans.property.DoubleProperty;
2 import javafx.beans.property.SimpleDoubleProperty;
3
4 public class BindingDemo {
5     public static void main(String[] args) {
6         DoubleProperty d1 = new SimpleDoubleProperty(1);
7         DoubleProperty d2 = new SimpleDoubleProperty(2);
8         d1.bind(d2);
9         System.out.println("d1 is " + d1.getValue()
10             + " and d2 is " + d2.getValue());
11         d2.setValue(70.2);
12         System.out.println("d1 is " + d1.getValue()
13             + " and d2 is " + d2.getValue());
14     }
15 }
```

```
d1 is 2.0 and d2 is 2.0
d1 is 70.2 and d2 is 70.2
```

程序使用 `SimpleDoubleProperty(1)`（第 6 行）创建了 `DoubleProperty` 的一个实例。请注意，`DoubleProperty`、`FloatProperty`、`LongProperty`、`IntegerProperty` 以及 `BooleanProperty` 都是抽象类。它们的具体子类 `SimpleDoubleProperty`、`SimpleFloatProperty`、`SimpleLongProperty`、`SimpleIntegerProperty` 以及 `SimpleBooleanProperty` 用于产生这些属性的实例。这些类很类似于包装类 `Double`、`Float`、`Long`、`Integer` 以及 `Boolean`，提供了绑定到一个源对象的额外特征。

程序将 `d1` 和 `d2` 绑定（第 8 行）。现在 `d1` 和 `d2` 中的值相同了。将 `d2` 设为 70.2 后（第 11 行），`d1` 也同样变成了 70.2（第 13 行）。

在这个例子中展示的绑定称为单向绑定。有时候，同步两个属性非常有用，这样，一个属性的改变将反映到另一个对象上，反过来也一样，这称为双向绑定。如果目标和源同时都是绑定属性和可观察属性，它们就可以使用 `bindBidirectional` 方法进行双向绑定。

复习题

- 14.8 什么是绑定属性？什么接口定义绑定属性？什么接口定义源对象？int、long、float、double，以及boolean的绑定对象类型是什么？Integer和Double是绑定属性吗？Integer和Double可以在一个绑定中作为源对象吗？
- 14.9 遵循JavaFX的绑定属性命名习惯，对于一个IntegerProperty类型且名为age的绑定属性，它的值获取方法、值设置方法以及属性获取方法分别是什么？
- 14.10 可以采用new IntegerProperty(3)来创建IntegerProperty类型的对象吗？如果不可以，正确的创建方法是什么？程序清单14-6中，如果第8行换成d1.bind(d2.multiply(2))，输出将是什么？程序清单14-6中，如果第8行换成d1.bind(d2.add(2))，输出将是什么？
- 14.11 什么是单向绑定和双向绑定？是否所有的属性都可以进行双向绑定？请写一个语句，将属性d1和d2进行双向绑定。

14.6 节点的通用属性和方法

 **要点提示：**抽象类Node定义了许多对于节点而言通用的属性和方法。

节点具有许多通用的属性。本节介绍两个这样的属性：style和rotate。

JavaFX的样式属性类似于用于在Web页面中指定HTML元素样式的层叠样式表(CSS)。因此，JavaFX的样式属性称为JavaFX CSS。JavaFX中，样式属性使用前缀-fx-进行定义。每个节点拥有它自己的样式属性。可以从<http://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html>找到这些属性。对于HTML和CSS的信息，请参见补充材料V.A和V.B。即使你不熟悉HTML和CSS，你也依然可以使用JavaFX CSS。

设定样式的语法是styleName:value。一个节点的多个样式属性可以一起设置，通过分号(;)进行分隔。比如，以下语句

```
circle.setStyle("-fx-stroke: black; -fx-fill: red;");
```

设置了一个圆的两个JavaFX CSS属性。该语句等价于下面两个语句：

```
circle.setStroke(Color.BLACK);
circle.setFill(Color.RED);
```

如果使用了一个不正确的JavaFX CSS，程序依然可以编译和运行，但是样式将被忽略。

rotate属性可以设定一个以度为单位的角度，让节点围绕它的中心旋转该角度。如果设置的角度是正的，表示旋转是顺时针；否则，逆时针。例如，下面的代码将一个按钮旋转80°。

```
button.setRotate(80);
```

程序清单14-7给出了一个示例，创建了一个按钮，设置它的样式并将它加入到一个面板中。然后将面板旋转45°，设置它的样式为边框颜色为红色，背景颜色为淡灰色，如图14-8所示。

程序清单 14-7 NodeStyleRotateDemo.java

```
1 import javafx.application.Application;
2 import javafx.scene.Scene;
3 import javafx.scene.control.Button;
4 import javafx.stage.Stage;
5 import javafx.scene.layout.StackPane;
6
7 public class NodeStyleRotateDemo extends Application {
8     @Override // Override the start method in the Application class
```

```

9 public void start(Stage primaryStage) {
10     // Create a scene and place a button in the scene
11     StackPane pane = new StackPane();
12     Button btOK = new Button("OK");
13     btOK.setStyle("-fx-border-color: blue;");
14     pane.getChildren().add(btOK);
15
16     pane.setRotate(45);
17     pane.setStyle(
18         "-fx-border-color: red; -fx-background-color: lightgray;");
19
20     Scene scene = new Scene(pane, 200, 250);
21     primaryStage.setTitle("NodeStyleRotateDemo"); // Set the stage title
22     primaryStage.setScene(scene); // Place the scene in the stage.
23     primaryStage.show(); // Display the stage
24 }
25 }

```

如图 14-8 所示，旋转一个面板导致了它包含的节点也进行了旋转。

Node 类包含了许多有用的方法，可以应用于所有节点。例如，可以使用 `contains(double x, double y)` 方法来检测一个点 (x, y) 是否位于一个节点的边界之内。

复习题

- 14.12 如何设置一个节点的样式，使之边框颜色为红色？请修改代码，设置按钮的文本颜色为红色。
- 14.13 可以旋转面板、文本或者按钮吗？请修改代码使按钮逆时针旋转 15° 。

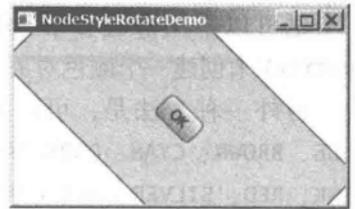


图 14-8 设置一个面板的样式并将其旋转 45°

14.7 Color 类

要点提示：Color 类可以用于创建颜色。

JavaFX 定义了抽象类 `Paint` 用于绘制节点。`javafx.scene.paint.Color` 是 `Paint` 的具体子类，用于封装颜色信息，如图 14-9 所示。

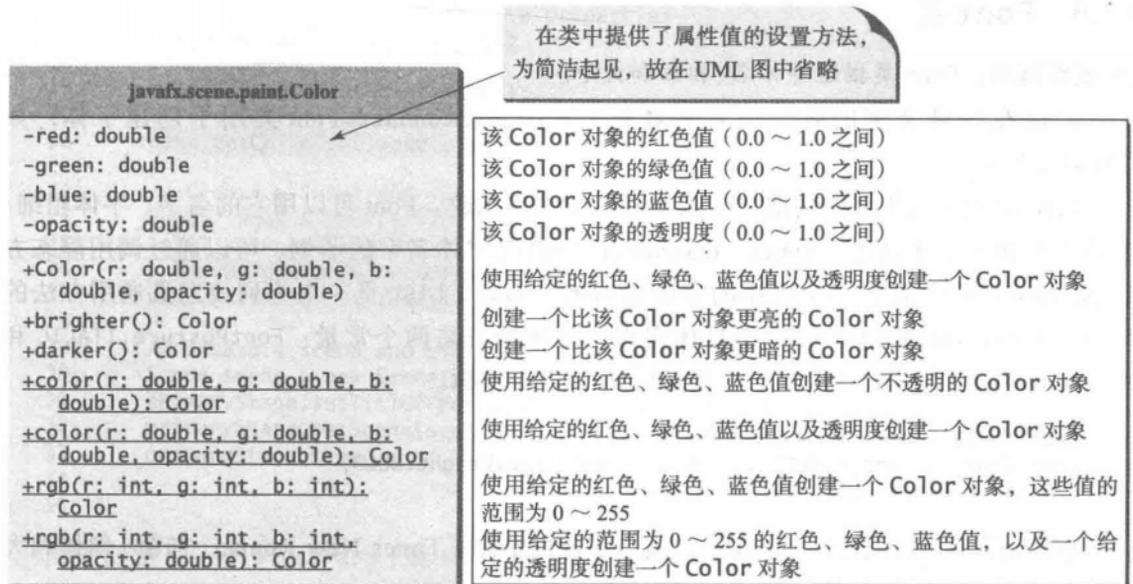


图 14-9 Color 封装了颜色信息

可以通过以下构造方法构建颜色实例：

```
public Color(double r, double g, double b, double opacity);
```

其中 *r*、*g*、*b* 通过红色、绿色、蓝色分量值来定义一个颜色，其值从 0.0（最深色）到 1.0（最浅色）。*opacity* 值定义了一个颜色的透明度，从 0.0（完全透明）到 1.0（完全不透明）。这称为 RGBA 模型，其中 RGBA 分别表示红色、绿色、蓝色和 alpha 值，alpha 值表示透明度。例如，

```
Color color = new Color(0.25, 0.14, 0.333, 0.51);
```

`Color` 类是不可修改的。当一个 `Color` 对象创建后，它的属性不能再修改。`brighter()` 方法返回一个具有更大的红、绿、蓝值的新的 `Color` 对象，而 `darker()` 方法返回一个具有更小的红、绿、蓝值的新的 `Color` 对象。*opacity* 值与原来的 `Color` 对象中的值相同。

也可以采用静态方法 `color(r,g,b)`、`color(r,g,b,opacity)`、`rgb(r,g,b)` 以及 `rgb(r,g,b,opacity)` 来创建一个颜色对象。

另外一种方法是，可以采用 `Color` 类中定义的许多标准颜色之一，如 `BEIGE`、`BLACK`、`BLUE`、`BROWN`、`CYAN`、`DARKGRAY`、`GOLD`、`GRAY`、`GREEN`、`LIGHTGRAY`、`MAGENTA`、`NAVY`、`ORANGE`、`PINK`、`RED`、`SILVER`、`WHITE` 和 `YELLOW`。例如，下面的代码设置一个圆的填充颜色为红色：

```
circle.setFill(Color.RED);
```

复习题

- 14.14 如何创建颜色？下面创建 `Color` 的代码哪里有错：`new Color(1.2,2.3,3.5, 4)`？下面的两个颜色哪个更深，`new Color(0,0,0,1)` 还是 `new Color(1,1,1,1)`？调用 `c.darker()` 改变 `c` 中的颜色值吗？
- 14.15 如何创建具有随机颜色的 `Color` 对象？
- 14.16 如何通过 `setFill` 方法和使用 `setStyle` 方法设置圆对象 `c` 的填充颜色为蓝色？

14.8 Font 类

 **要点提示：** `Font` 类描述字体名、粗细和大小。

可以在渲染文字的时候设置字体信息。`javafx.scene.text.Font` 类用于创建字体，如图 14-10 所示。

`Font` 实例可以用它的构造方法或者静态方法来构建。`Font` 可以用它的名字、字体粗细、字体形态和大小来描述。`Times`、`Courier` 和 `Arial` 是字体名字的示例。可以通过调用静态方法 `getFamilies()` 获得一个可用的字体系列名字列表。`List` 是一个为列表定义通用方法的接口。`ArrayList` 是 `List` 的一个具体实现。字体形态是两个常量：`FontPosture.ITALIC` 和 `FontPosture.REGULAR`。例如，下面的语句生成两个字体。

```
Font font1 = new Font("SansSerif", 16);
Font font2 = Font.font("Times New Roman", FontWeight.BOLD,
    FontPosture.ITALIC, 12);
```

程序清单 14-8 给出了一个程序，演示了使用字体（`Times New Roman`、加粗、斜体和大小为 20）来显示一个标签，如图 14-11 所示。

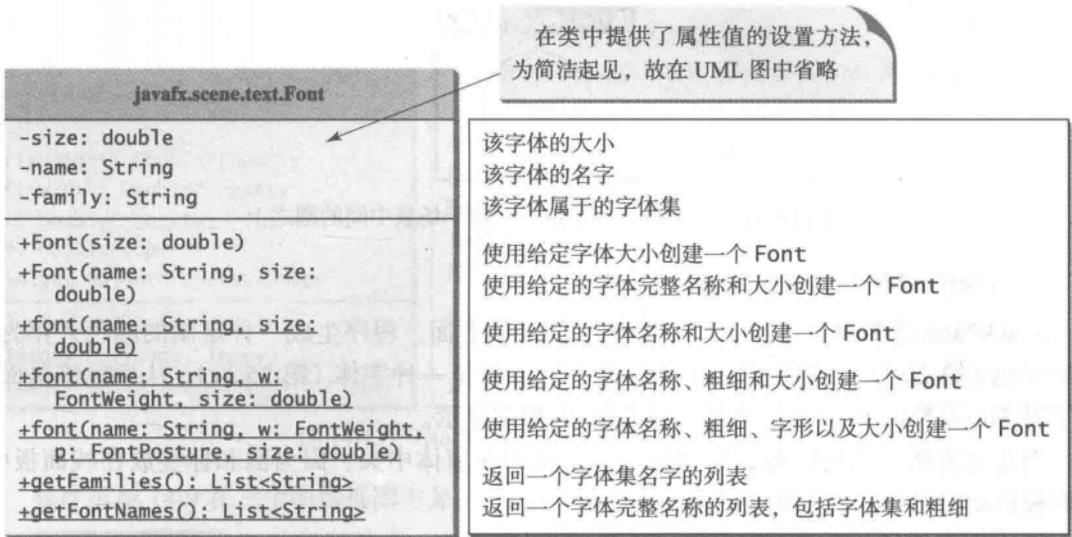


图 14-10 Font 封装了字体信息

程序清单 14-8 FontDemo.java

```

1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.layout.*;
4  import javafx.scene.paint.Color;
5  import javafx.scene.shape.Circle;
6  import javafx.scene.text.*;
7  import javafx.scene.control.*;
8  import javafx.stage.Stage;
9
10 public class FontDemo extends Application {
11     @Override // Override the start method in the Application class
12     public void start(Stage primaryStage) {
13         // Create a pane to hold the circle
14         Pane pane = new StackPane();
15
16         // Create a circle and set its properties
17         Circle circle = new Circle();
18         circle.setRadius(50);
19         circle.setStroke(Color.BLACK);
20         circle.setFill(new Color(0.5, 0.5, 0.5, 0.1));
21         pane.getChildren().add(circle); // Add circle to the pane
22
23         // Create a label and set its properties
24         Label label = new Label("JavaFX");
25         label.setFont(Font.font("Times New Roman",
26             FontWeight.BOLD, FontPosture.ITALIC, 20));
27         pane.getChildren().add(label);
28
29         // Create a scene and place it in the stage
30         Scene scene = new Scene(pane);
31         primaryStage.setTitle("FontDemo"); // Set the stage title
32         primaryStage.setScene(scene); // Place the scene in the stage
33         primaryStage.show(); // Display the stage
34     }
35 }

```

程序创建了一个 StackPane (第 14 行) 并将一个圆和标签添加到其中 (第 21、27 行)。这两个语句可以使用以下一行语句来整合：

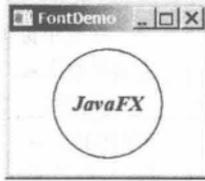


图 14-11 一个标签显示在一个位于场景中间的圆之上

```
pane.getChildren().addAll(circle, label);
```

StackPane 将节点置于中央，节点依次位于最上面。程序生成一种定制的颜色并作为圆的填充色（第 20 行）。程序创建一种标签并且设置了一种字体（第 25 行），从而标签里面的文字以 Times New Roman、加粗、斜体和 20 像素显示。

当改变窗体大小的时候，圆和标签依然显示在窗体中央。因为圆和标签放在栈面板中。栈面板自动将节点放在面板中央。

Font 对象是不可改变的。一旦一个 Font 对象创建，它的属性就不能改变。

复习题

14.17 如何创建一个字体名为 Courier，大小为 20，字体重量为黑体的 Font 对象？

14.18 如何找到系统中所有可用的字体？

14.9 Image 和 ImageView 类

要点提示：Image 类表示一个图像，ImageView 类可以用于显示一个图像。

javafx.scene.image.Image 类表示一个图像，用于从一个特定的文件名或者一个 URL 载入一个图像。例如，new Image("image/us.gif") 为位于 Java 类路径的 image 目录下的 us.gif 图像文件创建一个 Image 对象；new Image("http://www.cs.armstrong.edu/liang/image/us.gif") 为 Web 上相应 URL 中的图像文件创建一个 Image 对象。

javafx.scene.image.ImageView 是一个用于显示图像的节点。ImageView 可以从一个 Image 对象产生。例如，以下代码从一个图像文件创建一个 ImageView：

```
Image image = new Image("image/us.gif");
ImageView imageView = new ImageView(image);
```

另外，也可以直接从一个文件或者一个 URL 来创建一个 ImageView，如下所示：

```
ImageView imageView = new ImageView("image/us.gif");
```

图 14-12 和图 14-13 展示了 Image 和 ImageView 的 UML 图。

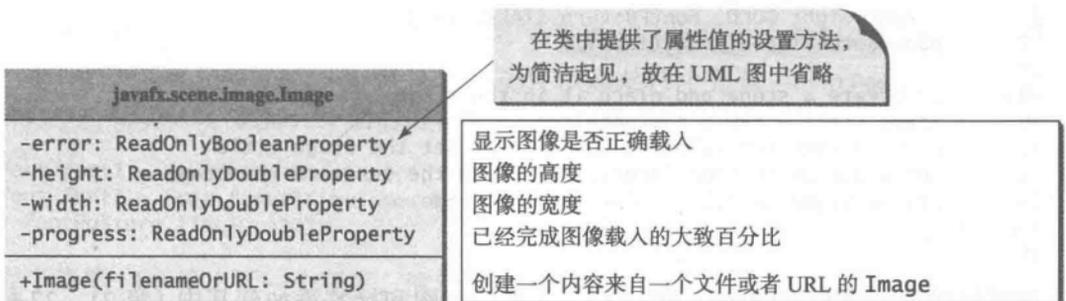


图 14-12 Image 封装了图像信息

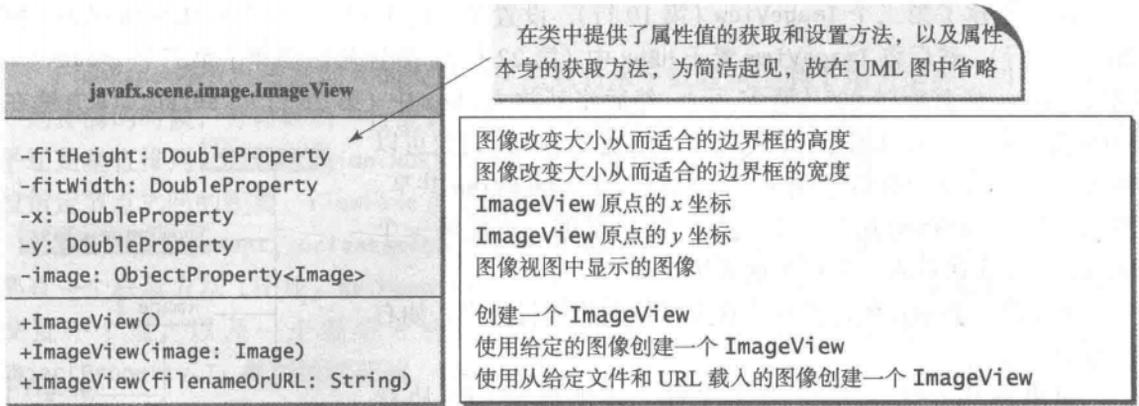


图 14-13 ImageView 是用于显示图像的节点

程序清单 14-9 在三个图像视图中显示一幅图像，如图 14-14 所示。

程序清单 14-9 ShowImage.java

```

1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.layout.HBox;
4  import javafx.scene.layout.Pane;
5  import javafx.geometry.Insets;
6  import javafx.stage.Stage;
7  import javafx.scene.image.Image;
8  import javafx.scene.image.ImageView;
9
10 public class ShowImage extends Application {
11     @Override // Override the start method in the Application class
12     public void start(Stage primaryStage) {
13         // Create a pane to hold the image views
14         Pane pane = new HBox(10);
15         pane.setPadding(new Insets(5, 5, 5, 5));
16         Image image = new Image("image/us.gif");
17         pane.getChildren().add(new ImageView(image));
18
19         ImageView imageView2 = new ImageView(image);
20         imageView2.setFitHeight(100);
21         imageView2.setFitWidth(100);
22         pane.getChildren().add(imageView2);
23
24         ImageView imageView3 = new ImageView(image);
25         imageView3.setRotate(90);
26         pane.getChildren().add(imageView3);
27
28         // Create a scene and place it in the stage
29         Scene scene = new Scene(pane);
30         primaryStage.setTitle("ShowImage"); // Set the stage title
31         primaryStage.setScene(scene); // Place the scene in the stage
32         primaryStage.show(); // Display the stage
33     }
34 }
  
```

程序创建了一个 HBox (第 14 行)。HBox 是一种面板，它将所有的节点排列在水平的一行上。程序创建一个 Image，接着创建一个 ImageView 用于显示图像，然后将 ImageView 放在 HBox 中 (第 17 行)。

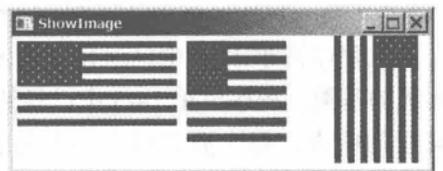


图 14-14 一个图像通过面板中的三个图像视图显示

程序创建了第二个 `ImageView` (第 19 行), 设置了它的 `fitHeight` 和 `fitWidth` 属性 (第 20 ~ 21 行), 然后将 `ImageView` 置于 `HBox` 中 (第 22 行)。程序然后创建了第三个 `ImageView` (第 24 行), 将其旋转 90° (第 25 行), 然后将其放入 `HBox` 中 (第 26 行)。`setRotate` 方法在 `Node` 类中定义, 可以用于任何节点。请注意, `Image` 对象可以被多个节点共享。在这个例子中, 它被三个 `ImageView` 共享。然而, 像 `ImageView` 这样的节点是不能共享的。不能将一个 `ImageView` 多次放入一个面板或者场景中。

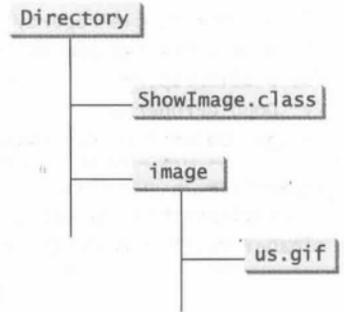
请注意, 必须将图像文件放在类文件的相同目录中, 如右图所示。

如果使用 URL 来定位图像文件, 必须提供 URL 协议 `http://`。因此下面的代码是错误的:

```
new Image("www.cs.armstrong.edu/liang/image/us.gif");
```

它应该写成如下语句:

```
new Image("http://www.cs.armstrong.edu/liang/image/us.gif");
```



复习题

- 14.19 如何从一个 URL 和一个文件名来创建一个 `Image` 对象?
- 14.20 如何从一个 `Image` 创建一个 `ImageView`, 或者直接从一个文件或 URL 创建?
- 14.21 可以将一个 `Image` 设到多个 `ImageView` 上吗? 可以将一个 `ImageView` 显示多次吗?

14.10 布局面板

要点提示: JavaFX 提供了多种类型的面板, 用于自动地将节点布局在希望的位置和大小。

JavaFX 提供了多种类型的面板, 用于在一个容器中组织节点, 如表 14-1 所示。在前面小节中我们已经用过布局面板 `Pane`、`StackPane` 和 `HBox` 来包含节点。本节更加详细地介绍这些面板。

表 14-1 用于包含和组织节点的面板

类	描述
<code>Pane</code>	布局面板的基类, 它有 <code>getChildren()</code> 方法来返回面板中的节点列表
<code>StackPane</code>	节点放置在面板中央, 并且叠加在其他节点之上
<code>FlowPane</code>	节点以水平方式一行一行放置, 或者垂直方式一列一列放置
<code>GridPane</code>	节点放置在一个二维网格的单元格中
<code>BorderPane</code>	将节点放置在顶部、右边、底部、左边以及中间区域
<code>HBox</code>	节点放在单行中
<code>VBox</code>	节点放在单列中

在程序清单 14-4 中已经使用过 `Pane`。`Pane` 通常用作显示形状的画布。`Pane` 是所有特定面板的基类。程序清单 14-3 中已经使用了一个特定的面板 `StackPane`。节点放置在 `StackPane` 面板的中央。每个面板包含一个列表用于容纳面板中的节点。这个列表是 `ObservableList` 的实例, 可以通过面板的 `getChildren()` 方法得到。可以使用 `add(node)` 方法将一个元素加到列表中, 也可以使用 `addAll(node1,node2,...)` 来添加一系列的节点到面板中。

14.10.1 FlowPane

FlowPane 将节点按照加入的次序，从左到右水平或者从上到下垂直组织。当一行或者一列排满的时候，开始新的一行或者一列。可以使用以下两个常数中的一个来确定节点是水平还是垂直排列：Orientation.HORIZONTAL 或者 Orientation.VERTICAL。可以使用像素为单位指定节点之间的距离。FlowPane 的类图如图 14-15 所示。

数据域 alignment、orientation、hgap 和 vgap 是绑定属性。JavaFX 中的每个绑定属性都有一个获取方法（比如，getHgap()）返回其值，一个设置方法（比如，setHgap(double)）设置一个值，以及一个获取方法返回属性本身（比如，hGapProperty()）。对于一个 ObjectProperty<T> 类型的数据域，值的获取方法返回一个 T 类型的值，属性获取方法返回一个 ObjectProperty<T> 类型的属性值。

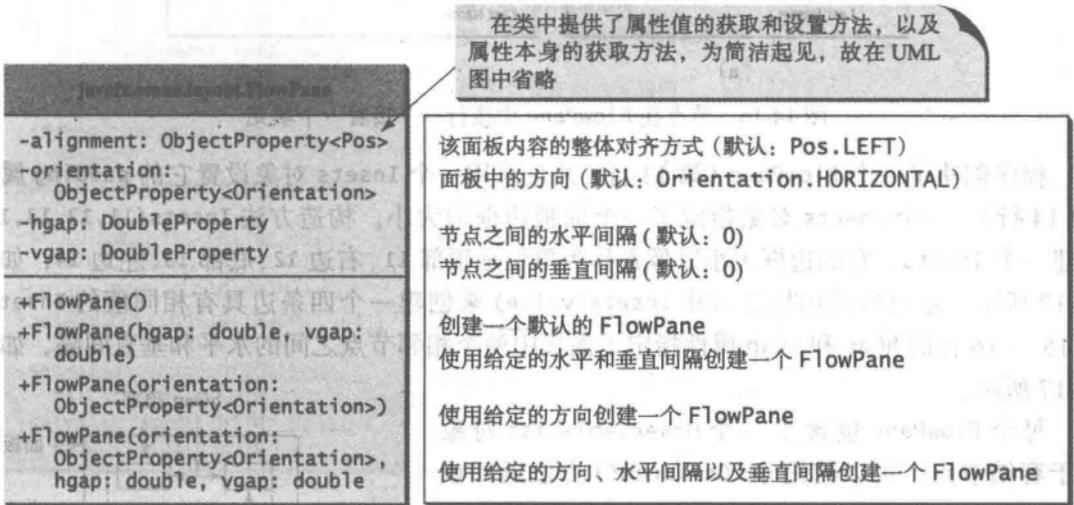


图 14-15 FlowPane 将节点按照水平方向一行一行，或者垂直方向一列一列布局

程序清单 14-10 给出了一个演示 FlowPane 用法的程序。程序添加标签和文本域到一个 FlowPane 中，如图 14-16 所示。

程序清单 14-10 ShowFlowPane.java

```

1 import javafx.application.Application;
2 import javafx.geometry.Insets;
3 import javafx.scene.Scene;
4 import javafx.scene.control.Label;
5 import javafx.scene.control.TextField;
6 import javafx.scene.layout.FlowPane;
7 import javafx.stage.Stage;
8
9 public class ShowFlowPane extends Application {
10     @Override // Override the start method in the Application class
11     public void start(Stage primaryStage) {
12         // Create a pane and set its properties
13         FlowPane pane = new FlowPane();
14         pane.setPadding(new Insets(11, 12, 13, 14));
15         pane.setHgap(5);
16         pane.setVgap(5);
17
18         // Place nodes in the pane
19         pane.getChildren().addAll(new Label("First Name:"),
  
```

```

20     new TextField(), new Label("MI:"));
21     TextField tfMi = new TextField();
22     tfMi.setPrefColumnCount(1);
23     pane.getChildren().addAll(tfMi, new Label("Last Name:"),
24         new TextField());
25
26     // Create a scene and place it in the stage
27     Scene scene = new Scene(pane, 200, 250);
28     primaryStage.setTitle("ShowFlowPane"); // Set the stage title
29     primaryStage.setScene(scene); // Place the scene in the stage
30     primaryStage.show(); // Display the stage
31 }
32 }

```

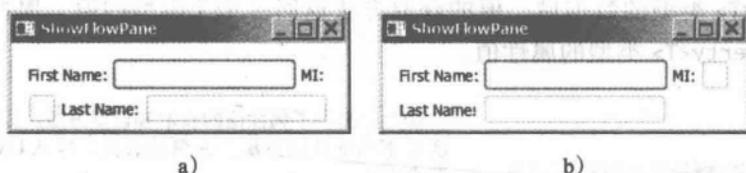


图 14-16 节点在 FlowPane 中按行一个接着一个填充

程序创建了一个 FlowPane (第 13 行) 并且采用一个 Insets 对象设置它的 padding 属性 (第 14 行)。一个 Insets 对象指定了一个面板边框的大小。构造方法 Insets(11,12,13,14) 创建一个 Insets, 它的边框大小以像素作为单位是顶部 11、右边 12、底部 13、左边 14, 如图 14-17 所示。还可以使用构造方法 Insets(value) 来创建一个四条边具有相同值的 Insets。第 15 ~ 16 行的 hGap 和 vGap 属性指定了面板中两个相邻节点之间的水平和垂直间隔, 如图 14-17 所示。

每个 FlowPane 包含了一个 ObservableList 对象用于容纳节点。可以使用 getChildren() 方法返回该列表 (第 19 行)。将一个节点添加到 FlowPane 是使用 add(node) 或者 addAll(node1,node2,...) 将其添加到列表中。也可以使用 remove(node) 来从列表中移除一个节点, 或者使用 removeAll() 方法将面板中的所有节点移除。程序将标签和文本域添加到面板中 (第 19 ~ 24 行)。调用 tfMi.setPrefColumnCount(1) 将 MI 文本域的期望列数设置为 1 (第 22 行)。程序为 MI 的 TextField 对象声明了一个显式的引用 tfMi。这个显式的引用是必要的, 因为我们需要直接引用这个对象来设置它的 prefColumnCount 属性。

程序将面板加入到场景中 (第 27 行), 将场景设置到舞台中 (第 29 行) 并显示该舞台 (第 30 行)。请注意, 如果修改窗体的大小, 这些节点自动地重新组织来适应面板。图 14-16a 中, 第一行有三个节点, 但是在图 14-16b 中, 第一行有四个节点, 因为宽度增加了。

假设希望将对象 tfMi 加入到一个面板 10 次; 是否会有 10 个文本域出现在面板中呢? 不会, 像文本域这样的节点只能加到一个面板中一次。将一个节点加入到一个面板中多次或者不同面板中将引起运行时错误。

注意: 一个节点只能放在一个面板中。因此, 面板和节点的关系是组合关系, 使用一个填充的棱形表示, 如图 14-3b 所示。

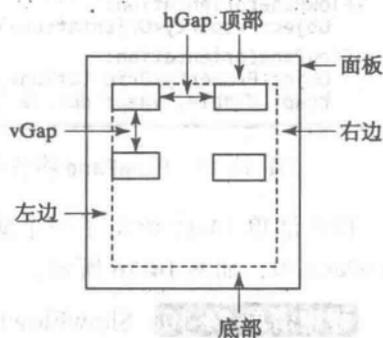


图 14-17 可以在 FlowPane 中指定节点之间的 hGap 和 vGap 间隔值

14.10.2 GridPane

GridPane 将节点布局在一个网格（矩阵）中。节点放在一个指定的列和行索引中。GridPane 的类图如图 14-18 所示。

在类中提供了属性值的获取和设置方法，以及属性本身的获取方法，为简洁起见，故在 UML 图中省略

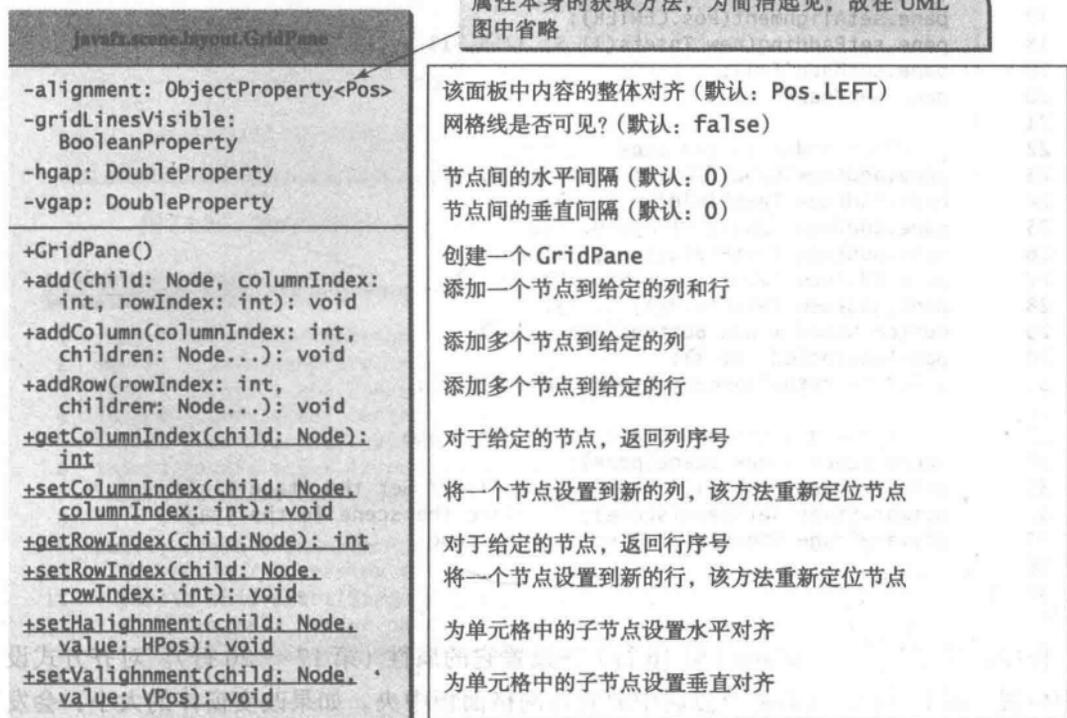


图 14-18 GridPane 将节点布局在一个网格中特定的单元格里

程序清单 14-11 给出了一个演示 GridPane 的程序。程序类似于程序清单 14-10。不同之处在于将三个标签、三个文本域，以及一个按钮添加到一个网格的特定位置，如图 14-19 所示。

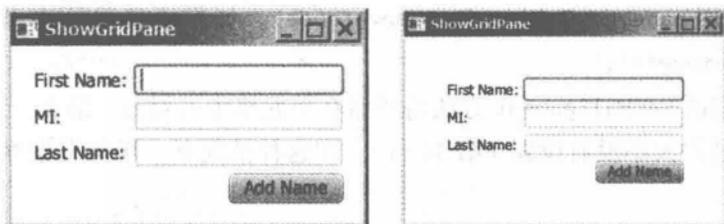


图 14-19 GridPane 将节点按照特定的行和列索引放置在一个网格中

程序清单 14-11 ShowGridPane.java

```

1 import javafx.application.Application;
2 import javafx.geometry.HPos;
3 import javafx.geometry.Insets;
4 import javafx.geometry.Pos;
5 import javafx.scene.Scene;
6 import javafx.scene.control.Button;
7 import javafx.scene.control.Label;
8 import javafx.scene.control.TextField;

```

```

9 import javafx.scene.layout.GridPane;
10 import javafx.stage.Stage;
11
12 public class ShowGridPane extends Application {
13     @Override // Override the start method in the Application class
14     public void start(Stage primaryStage) {
15         // Create a pane and set its properties
16         GridPane pane = new GridPane();
17         pane.setAlignment(Pos.CENTER);
18         pane.setPadding(new Insets(11.5, 12.5, 13.5, 14.5));
19         pane.setHgap(5.5);
20         pane.setVgap(5.5);
21
22         // Place nodes in the pane
23         pane.add(new Label("First Name:"), 0, 0);
24         pane.add(new TextField(), 1, 0);
25         pane.add(new Label("MI:"), 0, 1);
26         pane.add(new TextField(), 1, 1);
27         pane.add(new Label("Last Name:"), 0, 2);
28         pane.add(new TextField(), 1, 2);
29         Button btAdd = new Button("Add Name");
30         pane.add(btAdd, 1, 3);
31         GridPane.setHalignment(btAdd, HPos.RIGHT);
32
33         // Create a scene and place it in the stage
34         Scene scene = new Scene(pane);
35         primaryStage.setTitle("ShowGridPane"); // Set the stage title
36         primaryStage.setScene(scene); // Place the scene in the stage
37         primaryStage.show(); // Display the stage
38     }
39 }

```

程序创建了一个 GridPane (第 16 行) 并设置它的属性 (第 17 ~ 20 行)。对齐方式设为居中位置 (第 17 行), 从而将节点居中放置在网格面板中央。如果改变窗体的大小, 会发现节点依然保持在网格面板的居中位置。

程序将标签放置在第 0 列和第 0 行 (第 23 行)。列和行索引从 0 开始。add 方法将一个节点放置在特定的列和行中。不是网格中的每个单元格都需要被填充。一个按钮被放置在第 1 列和第 3 行 (第 30 行), 但是第 0 列和第 3 行没有节点。如果要从 GridPane 移除一个节点, 使用 pane.getChildren().remove(node)。如果要移除所有节点, 使用 pane.getChildren().removeAll()。

程序调用静态的 setHalignment 方法将按钮在单元格中右对齐 (第 31 行)。

请注意, 场景的大小没有设置 (第 34 行)。在这种情况下, 场景会根据其中的节点大小自动计算。

14.10.3 BorderPane

BorderPane 可以将节点放置在五个区域: 顶部、底部、左边、右边以及中间, 分别使用 setTop(node)、setBottom(node)、setLeft(node)、setRight(node) 和 setCenter(node) 方法。BorderPane 的类图如图 14-20 所示。

程序清单 14-12 给出了一个程序以演示 BorderPane 的使用。程序将五个按钮分别放置在面板的五个区域, 如图 14-21 所示。

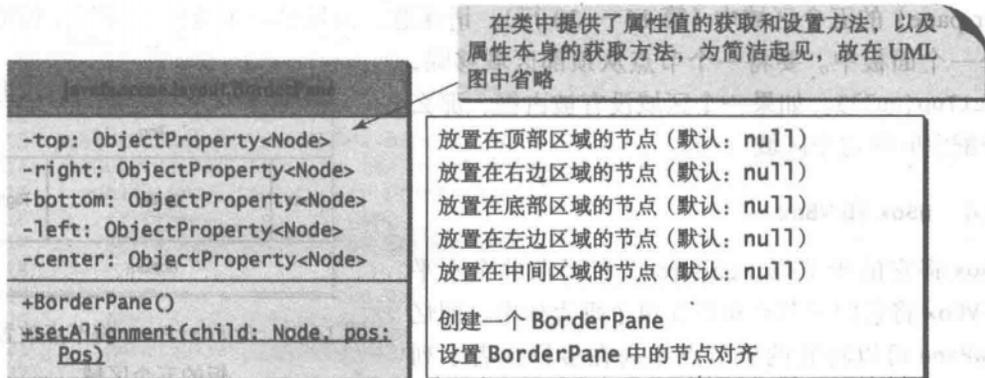


图 14-20 BorderPane 将节点放置在顶部、底部、左边、右边以及中间区域

程序清单 14-12 ShowBorderPane.java

```

1 import javafx.application.Application;
2 import javafx.geometry.Insets;
3 import javafx.scene.Scene;
4 import javafx.scene.control.Label;
5 import javafx.scene.layout.BorderPane;
6 import javafx.scene.layout.StackPane;
7 import javafx.stage.Stage;
8
9 public class ShowBorderPane extends Application {
10     @Override // Override the start method in the Application class
11     public void start(Stage primaryStage) {
12         // Create a border pane
13         BorderPane pane = new BorderPane();
14
15         // Place nodes in the pane
16         pane.setTop(new CustomPane("Top"));
17         pane.setRight(new CustomPane("Right"));
18         pane.setBottom(new CustomPane("Bottom"));
19         pane.setLeft(new CustomPane("Left"));
20         pane.setCenter(new CustomPane("Center"));
21
22         // Create a scene and place it in the stage
23         Scene scene = new Scene(pane);
24         primaryStage.setTitle("ShowBorderPane"); // Set the stage title
25         primaryStage.setScene(scene); // Place the scene in the stage
26         primaryStage.show(); // Display the stage
27     }
28 }
29
30 // Define a custom pane to hold a label in the center of the pane
31 class CustomPane extends StackPane {
32     public CustomPane(String title) {
33         getChildren().add(new Label(title));
34         setStyle("-fx-border-color: red");
35         setPadding(new Insets(11.5, 12.5, 13.5, 14.5));
36     }
37 }

```

程序定义了继承自 StackPane 的 CustomPane 类 (第 31 行)。CustomPane 的构造方法加入了一个具有特定标题的标签 (第 33 行)，为边框颜色设置样式，并采用 insets 设置内边距 (第 35 行)。

程序创建了一个 BorderPane (第 13 行) 并将 CustomPane 的五个实例分别放入边框面板

(border pane) 的五个区域中 (第 16 ~ 20 行)。请注意, 面板是一个节点, 所以面板可以加入另外一个面板中。要将一个节点从顶部区域移除, 调用 `setTop(null)`。如果一个区域没有被占据, 那么不会分配空间给这个区域。

14.10.4 HBox 和 VBox

HBox 将它的子节点 (children) 布局在单个水平行中。VBox 将它的子节点布局在单个垂直列中。回忆下 FlowPane 可以将它的子节点布局在多行或者多列中, 但是一个 HBox 或者 VBox 只能把子节点布局在一行或者一列中。HBox 和 VBox 的类图如图 14-22 和图 14-23 所示。

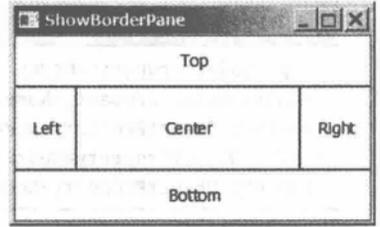


图 14-21 BorderPane 将节点放置在面板的五个区域

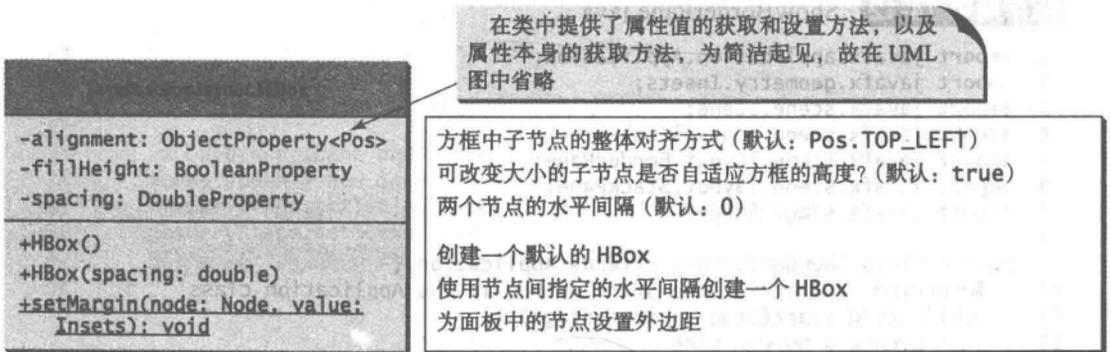


图 14-22 HBox 将节点置于一行

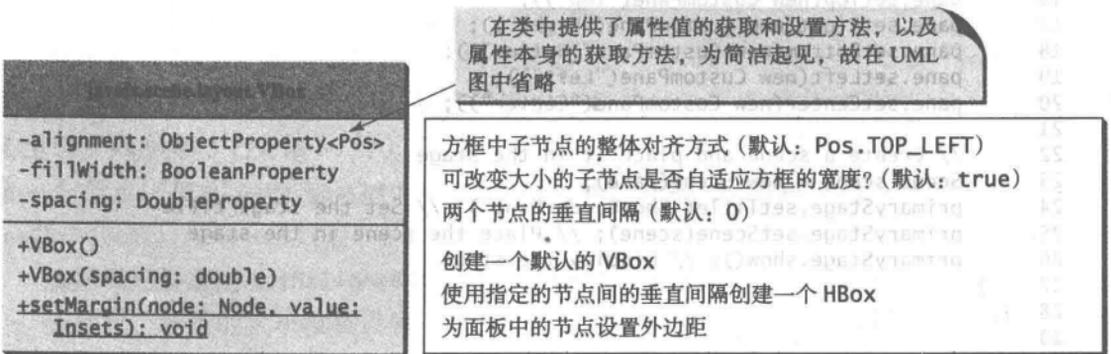


图 14-23 VBox 将节点置于一列

程序清单 14-13 给出了一个演示 HBox 和 VBox 的程序。程序将两个按钮放在一个 HBox 中, 将五个标签放在一个 VBox 中, 如图 14-24 所示。

程序清单 14-13 ShowHBoxVBox.java

```

1 import javafx.application.Application;
2 import javafx.geometry.Insets;
3 import javafx.scene.Scene;
4 import javafx.scene.control.Button;
5 import javafx.scene.control.Label;
6 import javafx.scene.layout.BorderPane;
7 import javafx.scene.layout.HBox;
                
```

```
8 import javafx.scene.layout.VBox;
9 import javafx.stage.Stage;
10 import javafx.scene.image.Image;
11 import javafx.scene.image.ImageView;
12
13 public class ShowHBoxVBox extends Application {
14     @Override // Override the start method in the Application class
15     public void start(Stage primaryStage) {
16         // Create a border pane
17         BorderPane pane = new BorderPane();
18
19         // Place nodes in the pane
20         pane.setTop(getHBox());
21         pane.setLeft(getVBox());
22
23         // Create a scene and place it in the stage
24         Scene scene = new Scene(pane);
25         primaryStage.setTitle("ShowHBoxVBox"); // Set the stage title
26         primaryStage.setScene(scene); // Place the scene in the stage
27         primaryStage.show(); // Display the stage
28     }
29
30     private HBox getHBox() {
31         HBox hBox = new HBox(15);
32         hBox.setPadding(new Insets(15, 15, 15, 15));
33         hBox.setStyle("-fx-background-color: gold");
34         hBox.getChildren().add(new Button("Computer Science"));
35         hBox.getChildren().add(new Button("Chemistry"));
36         ImageView imageView = new ImageView(new Image("image/us.gif"));
37         hBox.getChildren().add(imageView);
38         return hBox;
39     }
40
41     private VBox getVBox() {
42         VBox vBox = new VBox(15);
43         vBox.setPadding(new Insets(15, 5, 5, 5));
44         vBox.getChildren().add(new Label("Courses"));
45
46         Label[] courses = {new Label("CSCI 1301"), new Label("CSCI 1302"),
47                             new Label("CSCI 2410"), new Label("CSCI 3720")};
48
49         for (Label course: courses) {
50             VBox.setMargin(course, new Insets(0, 0, 0, 15));
51             vBox.getChildren().add(course);
52         }
53
54         return vBox;
55     }
56 }
```

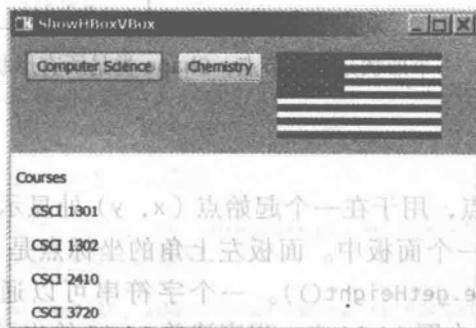


图 14-24 HBox 将节点置于一行，而 VBox 将节点置于一列

程序定义了 `getHBox()` 方法。该方法返回一个包含了两个按钮和一个图像视图的 `HBox` (第 30 ~ 39 行)。`HBox` 的背景颜色采用 Java CSS 设置为金色 (第 33 行)。程序还定义了 `getVBox()` 方法。该方法返回一个包含了五个标签的 `VBox` (第 41 ~ 55 行)。第一个标签在第 44 行加入 `VBox`, 其他四个在第 51 行加入。`setMargin` 方法用于将节点加入 `VBox` 的时候设置节点的外边距。

复习题

- 14.22 如何将一个节点加入到一个 `Pane`、`StackPane`、`FlowPane`、`GridPane`、`BorderPane`、`HBox`、`VBox` 中? 如何从这些面板中移除一个节点?
- 14.23 如何在一个 `FlowPane`、`GridPane`、`HBox`、`VBox` 中设置节点右对齐?
- 14.24 如何在一个 `FlowPane` 和 `GridPane` 中设置节点间的水平间隔和垂直间隔为 8 像素, 以及在 `HBox` 和 `VBox` 中设置间距为 8 像素?
- 14.25 如何得到 `GridPane` 面板中节点的列和行索引? 如何重新设定 `GridPane` 中节点的位置?
- 14.26 `FlowPane` 和 `HBox` 或者 `VBox` 之间的区别是什么?

14.11 形状

要点提示: JavaFX 提供了多种形状类, 用于绘制文本、直线、圆、矩形、椭圆、弧、多边形以及折线。

`Shape` 类是一个抽象基类, 定义了所有形状的共同属性。这些属性有 `fill`、`stroke`、`strokeWidth`。`fill` 属性指定一个填充形状内部区域的颜色。`Stroke` 属性指定用于绘制形状边缘的颜色。`strokeWidth` 属性指定形状边缘的宽度。本节介绍用于绘制文本和简单形状的 `Text`、`Line`、`Rectangle`、`Circle`、`Ellipse`、`Arc`、`Polygon` 以及 `PolyLine` 类。这些都是 `Shape` 的子类, 如图 14-25 所示。

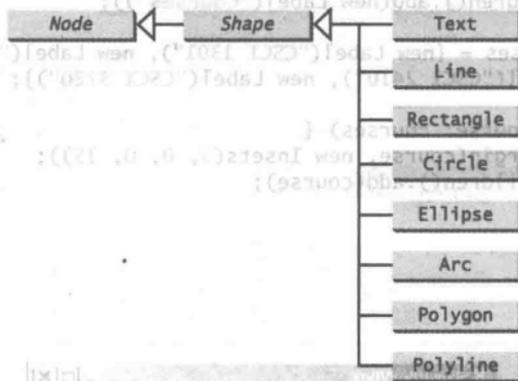


图 14-25 一个形状是一个节点, `Shape` 类是所有形状类的根

14.11.1 Text

`Text` 类定义了一个节点, 用于在一个起始点 (x, y) 处显示一个字符串, 如图 14-27a 所示。`Text` 对象通常置于一个面板中。面板左上角的坐标点是 $(0, 0)$, 右下角的坐标点是 $(pane.getWidth(), pane.getHeight())$ 。一个字符串可以通过 `\n` 分隔从而显示在多行。`Text` 类的 UML 图显示在图 14-26 中。程序清单 14-13 给出了一个演示文本的示例, 如图 14-27b 所示。

在类中提供了属性值的获取和设置方法，以及属性本身的获取方法，为简洁起见，故在UML图中省略

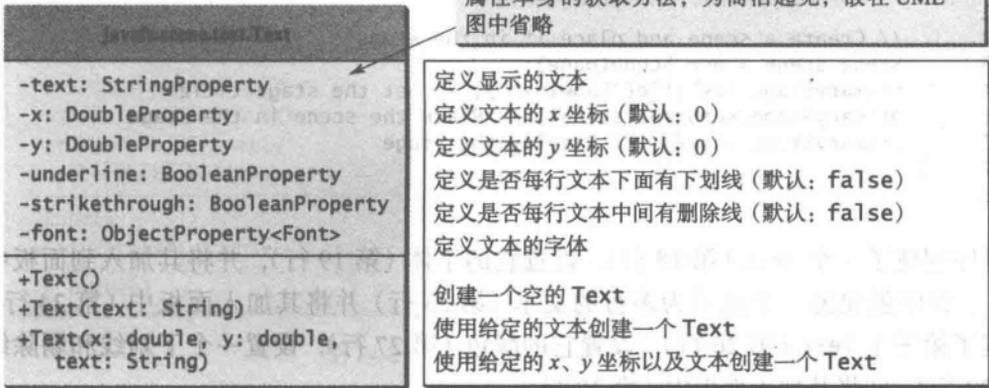


图 14-26 Text 定义了显示一个文本的节点

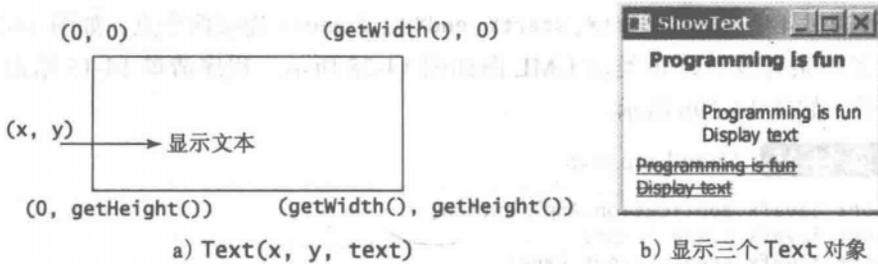


图 14-27 创建一个 Text 对象用于显示一个文本

程序清单 14-14 ShowText.java

```

1 import javafx.application.Application;
2 import javafx.scene.Scene;
3 import javafx.scene.layout.Pane;
4 import javafx.scene.paint.Color;
5 import javafx.geometry.Insets;
6 import javafx.stage.Stage;
7 import javafx.scene.text.Text;
8 import javafx.scene.text.Font;
9 import javafx.scene.text.FontWeight;
10 import javafx.scene.text.FontPosture;
11
12 public class ShowText extends Application {
13     @Override // Override the start method in the Application class
14     public void start(Stage primaryStage) {
15         // Create a pane to hold the texts
16         Pane pane = new Pane();
17         pane.setPadding(new Insets(5, 5, 5, 5));
18         Text text1 = new Text(20, 20, "Programming is fun");
19         text1.setFont(Font.font("Courier", FontWeight.BOLD,
20             FontPosture.ITALIC, 15));
21         pane.getChildren().add(text1);
22
23         Text text2 = new Text(60, 60, "Programming is fun\nDisplay text");
24         pane.getChildren().add(text2);
25
26         Text text3 = new Text(10, 100, "Programming is fun\nDisplay text");
27         text3.setFill(Color.RED);
28         text3.setUnderline(true);
                
```

```

29     text3.setStrikethrough(true);
30     pane.getChildren().add(text3);
31
32     // Create a scene and place it in the stage
33     Scene scene = new Scene(pane);
34     primaryStage.setTitle("ShowText"); // Set the stage title
35     primaryStage.setScene(scene); // Place the scene in the stage
36     primaryStage.show(); // Display the stage
37 }
38 }

```

程序创建了一个 Text (第 18 行), 设置它的字体 (第 19 行), 并将其加入到面板中 (第 21 行)。程序创建另一个显示为多行的文本 (第 23 行) 并将其加入面板中 (第 24 行)。程序创建了第三个 Text (第 26 行), 设置它的颜色 (第 27 行), 设置一个下划线和删除线 (第 28 ~ 29 行), 并将其加入面板中 (第 30 行)。

14.11.2 Line

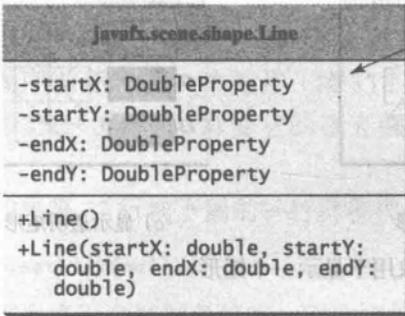
一条直线通过 4 个参数 (startX、startY、endX 以及 endY) 连接两个点, 如图 14-29a 所示。Line 类定义了一条直线。Line 类的 UML 图如图 14-28 所示。程序清单 14-15 给出了一个演示直线的例子, 如图 14-29b 所示。

程序清单 14-15 ShowLine.java

```

1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.layout.Pane;
4  import javafx.scene.paint.Color;
5  import javafx.stage.Stage;
6  import javafx.scene.shape.Line;
7
8  public class ShowLine extends Application {
9      @Override // Override the start method in the Application class
10     public void start(Stage primaryStage) {
11         // Create a scene and place it in the stage
12         Scene scene = new Scene(new LinePane(), 200, 200);
13         primaryStage.setTitle("ShowLine"); // Set the stage title
14         primaryStage.setScene(scene); // Place the scene in the stage
15         primaryStage.show(); // Display the stage
16     }
17 }
18
19 class LinePane extends Pane {
20     public LinePane() {
21         Line line1 = new Line(10, 10, 10, 10);
22         line1.endXProperty().bind(widthProperty().subtract(10));
23         line1.endYProperty().bind(heightProperty().subtract(10));
24         line1.setStrokeWidth(5);
25         line1.setStroke(Color.GREEN);
26         getChildren().add(line1);
27
28         Line line2 = new Line(10, 10, 10, 10);
29         line2.startXProperty().bind(widthProperty().subtract(10));
30         line2.endYProperty().bind(heightProperty().subtract(10));
31         line2.setStrokeWidth(5);
32         line2.setStroke(Color.GREEN);
33         getChildren().add(line2);
34     }
35 }

```



在类中提供了属性值的获取和设置方法以及属性本身的获取方法，为简洁起见，故在 UML 图中省略

起点的 x 坐标
起点的 y 坐标
终点的 x 坐标
终点的 y 坐标

创建一个空 Line
使用指定起点和终点创建一个 Line

图 14-28 Line 类定义了一条直线

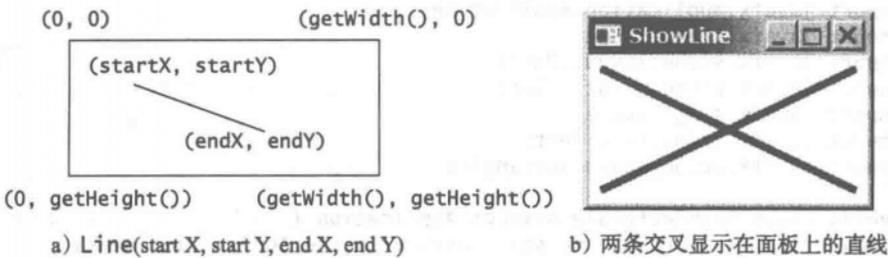


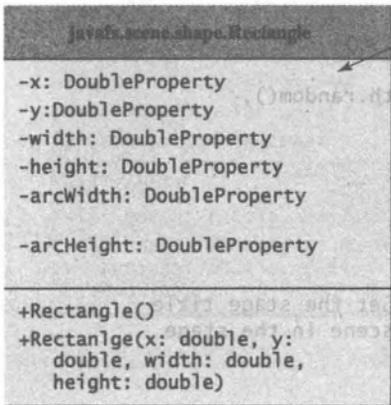
图 14-29 创建一个 Line 对象用于显示一条直线

程序定义了一个命名为 LinePane 的自定义面板类（第 19 行）。自定义面板类创建了两条直线，并将直线的起点和终点与面板的宽度和高度绑定（第 22 ~ 23 行，第 29 ~ 30 行），这样，当调整面板大小的时候直线上两个点的位置也发生相应变化。

14.11.3 Rectangle

一个矩形通过参数 x、y、width、height、arcWidth 以及 arcHeight 定义，如图 14-31a 所示。矩形的左上角点处于 (x,y)，参数 aw (arcWidth) 表示圆角处弧的水平直径，ah (arcHeight) 表示圆角处弧的垂直直径。

Rectangle 类定义了一个矩形。Rectangle 类的 UML 图如图 14-30 所示。程序清单 14-16 给出了一个演示矩形的例子，如图 14-31b 所示。



在类中提供了属性值的获取和设置方法，以及属性本身的获取方法，为简洁起见，故在 UML 图中省略

矩形左上角的 x 坐标（默认：0）
矩形左上角的 y 坐标（默认：0）
矩形的宽度（默认：0）
矩形的高度（默认：0）
矩形的 arcWidth 值（默认：0），arcWidth 是圆角处圆弧的水平直径（参见图 14-31a）
矩形的 arcHeight 值（默认：0），arcHeight 是圆角处圆弧的垂直直径（参见图 14-31a）
创建一个空的 Rectangle
使用给定的左上角点、宽度和高度创建一个 Rectangle

图 14-30 Rectangle 类定义了一个矩形

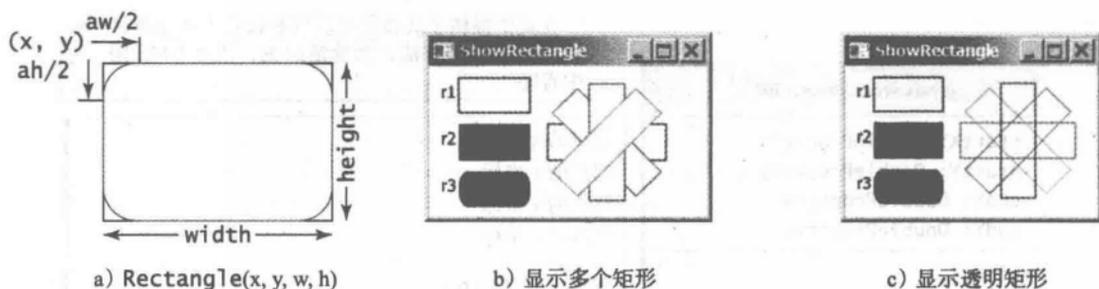


图 14-31 创建一个 Rectangle 对象用于显示一个矩形

程序清单 14-16 ShowRectangle.java

```

1 import javafx.application.Application;
2 import javafx.scene.Scene;
3 import javafx.scene.layout.Pane;
4 import javafx.scene.paint.Color;
5 import javafx.stage.Stage;
6 import javafx.scene.text.Text;
7 import javafx.scene.shape.Rectangle;
8
9 public class ShowRectangle extends Application {
10     @Override // Override the start method in the Application class
11     public void start(Stage primaryStage) {
12         // Create a pane
13         Pane pane = new Pane();
14
15         // Create rectangles and add to pane
16         Rectangle r1 = new Rectangle(25, 10, 60, 30);
17         r1.setStroke(Color.BLACK);
18         r1.setFill(Color.WHITE);
19         pane.getChildren().add(new Text(10, 27, "r1"));
20         pane.getChildren().add(r1);
21
22         Rectangle r2 = new Rectangle(25, 50, 60, 30);
23         pane.getChildren().add(new Text(10, 67, "r2"));
24         pane.getChildren().add(r2);
25
26         Rectangle r3 = new Rectangle(25, 90, 60, 30);
27         r3.setArcWidth(15);
28         r3.setArcHeight(25);
29         pane.getChildren().add(new Text(10, 107, "r3"));
30         pane.getChildren().add(r3);
31
32         for (int i = 0; i < 4; i++) {
33             Rectangle r = new Rectangle(100, 50, 100, 30);
34             r.setRotate(i * 360 / 8);
35             r.setStroke(Color.color(Math.random(), Math.random(),
36                 Math.random()));
37             r.setFill(Color.WHITE);
38             pane.getChildren().add(r);
39         }
40
41         // Create a scene and place it in the stage
42         Scene scene = new Scene(pane, 250, 150);
43         primaryStage.setTitle("ShowRectangle"); // Set the stage title
44         primaryStage.setScene(scene); // Place the scene in the stage
45         primaryStage.show(); // Display the stage
46     }
47 }

```

程序创建了多个矩形。默认的填充颜色是黑色。所以矩形填充为黑色。笔划颜色默认是白色。第 17 行设置矩形 r1 的笔划颜色为黑色。程序创建了矩形 r3 (第 26 行) 并设置它的弧的宽度和高度 (第 27 ~ 28 行)。从而 r3 显示为一个圆角矩形。

程序循环创建一个矩形 (第 33 行), 并将其旋转 (第 34 行), 设置一种随机的笔划颜色 (第 35 ~ 36 行), 设置它的填充颜色为白色 (第 37 行), 然后将矩形添加到面板上 (第 38 行)。

如果第 37 行被下面的一行所替代

```
r.setFill(null);
```

那么矩形将不会被颜色填充, 因此它们如图 14-31c 所示。

14.11.4 Circle 和 Ellipse

我们已经在本章前面的几个例子中使用了圆。一个圆由其参数 centerX、centerY 以及 radius 定义。Circle 类定义了一个圆。Circle 类的 UML 图如图 14-32 所示。

一个椭圆由其参数 centerX、centerY、radiusX 以及 radiusY 定义, 如图 14-34a 所示。Ellipse 类定义了一个椭圆。Ellipse 类的 UML 图如图 14-33 所示。程序清单 14-17 给出了一个演示椭圆的例子, 如图 14-34b 所示。

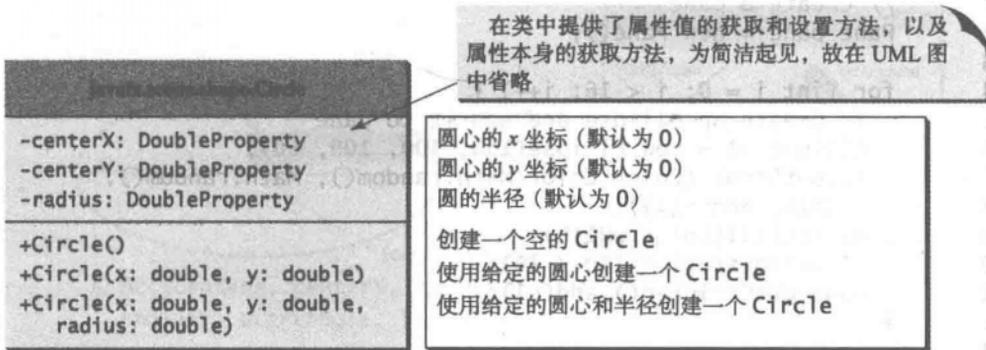


图 14-32 Circle 类定义圆

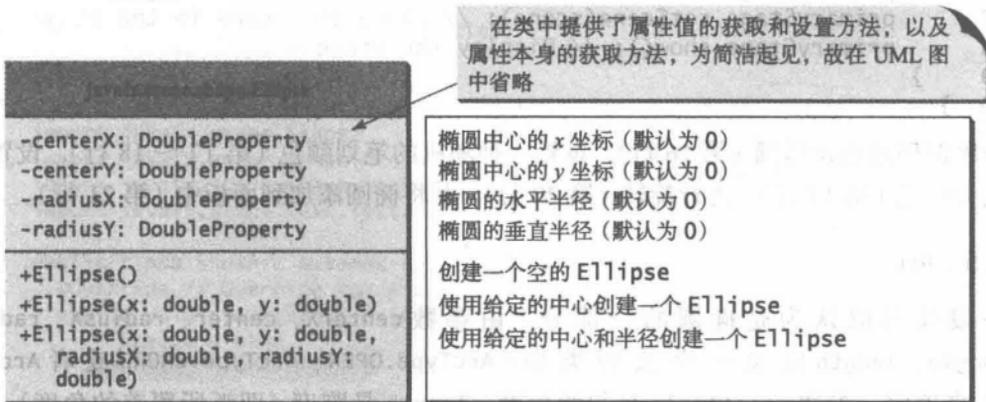


图 14-33 Ellipse 类定义椭圆

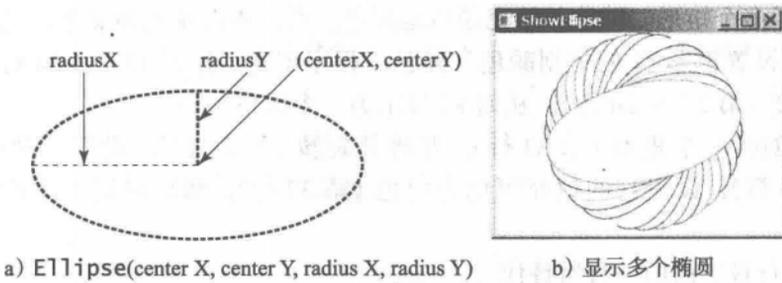


图 14-34 创建一个 Ellipse 对象用于显示椭圆

程序清单 14-17 ShowEllipse.java

```

1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.layout.Pane;
4  import javafx.scene.paint.Color;
5  import javafx.stage.Stage;
6  import javafx.scene.shape.Ellipse;
7
8  public class ShowEllipse extends Application {
9      @Override // Override the start method in the Application class
10     public void start(Stage primaryStage) {
11         // Create a pane
12         Pane pane = new Pane();
13
14         for (int i = 0; i < 16; i++) {
15             // Create an ellipse and add it to pane
16             Ellipse e1 = new Ellipse(150, 100, 100, 50);
17             e1.setStroke(Color.color(Math.random(), Math.random(),
18                 Math.random()));
19             e1.setFill(Color.WHITE);
20             e1.setRotate(i * 180 / 16);
21             pane.getChildren().add(e1);
22         }
23
24         // Create a scene and place it in the stage
25         Scene scene = new Scene(pane, 300, 200);
26         primaryStage.setTitle("ShowEllipse"); // Set the stage title
27         primaryStage.setScene(scene); // Place the scene in the stage
28         primaryStage.show(); // Display the stage
29     }
30 }

```

程序循环地创建椭圆（第 16 行），设置一种随机的笔划颜色（第 17～18 行），设置其填充颜色为白色（第 19 行），进行旋转（第 20 行），并将椭圆添加到面板中（第 21 行）。

14.11.5 Arc

一段弧可以认为是椭圆的一部分，由参数 `centerX`、`centerY`、`radiusX`、`radiusY`、`startAngle`、`length` 以及一个弧的类型（`ArcType.OPEN`、`ArcType.CHORD` 或者 `ArcType.ROUND`）来确定。参数 `startAngle` 是起始角度，`length` 是跨度（即弧所覆盖的角度）。角度使用度来作为单位，并且遵循通常的数学约定（即， 0° 是最东的方向，正的角度表示从最东方向开始顺时针方向的旋转角度），如图 14-36a 所示。

Arc 类定义一段弧。Arc 类的 UML 图如图 14-35 所示。程序清单 14-18 给出了一个演示弧的例子，如图 14-36b 所示。

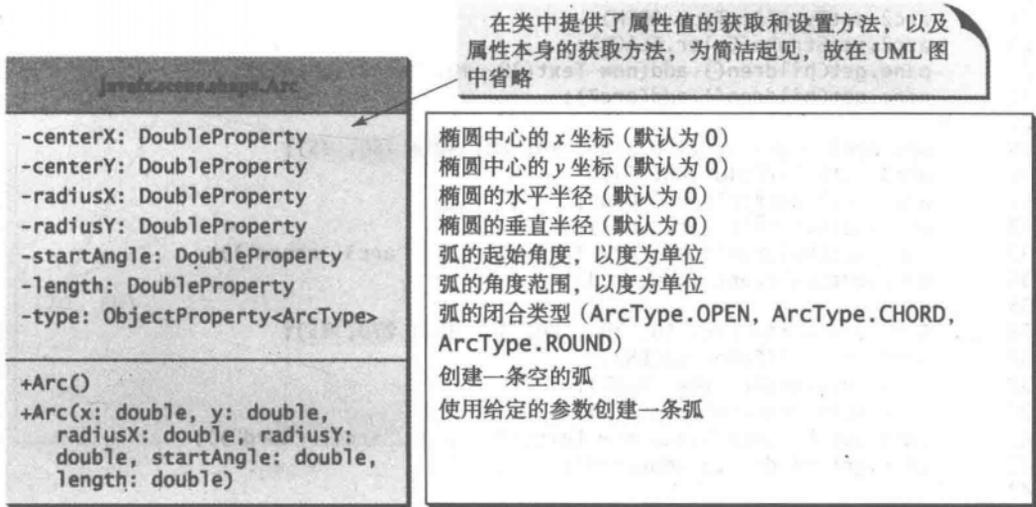


图 14-35 Arc 类定义弧

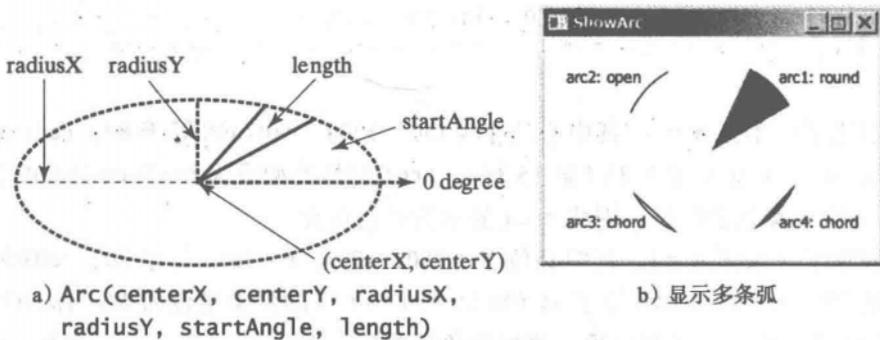


图 14-36 创建一个 Arc 对象用于显示弧

程序清单 14-18 ShowArc.java

```

1 import javafx.application.Application;
2 import javafx.scene.Scene;
3 import javafx.scene.layout.Pane;
4 import javafx.scene.paint.Color;
5 import javafx.stage.Stage;
6 import javafx.scene.shape.Arc;
7 import javafx.scene.shape.ArcType;
8 import javafx.scene.text.Text;
9
10 public class ShowArc extends Application {
11     @Override // Override the start method in the Application class
12     public void start(Stage primaryStage) {
13         // Create a pane
14         Pane pane = new Pane();
15
16         Arc arc1 = new Arc(150, 100, 80, 80, 30, 35); // Create an arc
17         arc1.setFill(Color.RED); // Set fill color
18         arc1.setType(ArcType.ROUND); // Set arc type
19         pane.getChildren().add(new Text(210, 40, "arc1: round"));

```

```

20     pane.getChildren().add(arc1); // Add arc to pane
21
22     Arc arc2 = new Arc(150, 100, 80, 80, 30 + 90, 35);
23     arc2.setFill(Color.WHITE);
24     arc2.setType(ArcType.OPEN);
25     arc2.setStroke(Color.BLACK);
26     pane.getChildren().add(new Text(20, 40, "arc2: open"));
27     pane.getChildren().add(arc2);
28
29     Arc arc3 = new Arc(150, 100, 80, 80, 30 + 180, 35);
30     arc3.setFill(Color.WHITE);
31     arc3.setType(ArcType.CHORD);
32     arc3.setStroke(Color.BLACK);
33     pane.getChildren().add(new Text(20, 170, "arc3: chord"));
34     pane.getChildren().add(arc3);
35
36     Arc arc4 = new Arc(150, 100, 80, 80, 30 + 270, 35);
37     arc4.setFill(Color.GREEN);
38     arc4.setType(ArcType.CHORD);
39     arc4.setStroke(Color.BLACK);
40     pane.getChildren().add(new Text(210, 170, "arc4: chord"));
41     pane.getChildren().add(arc4);
42
43     // Create a scene and place it in the stage
44     Scene scene = new Scene(pane, 300, 200);
45     primaryStage.setTitle("ShowArc"); // Set the stage title
46     primaryStage.setScene(scene); // Place the scene in the stage
47     primaryStage.show(); // Display the stage
48 }
49 }

```

程序创建了一条弧 arc1，其中心位于 (150, 100)，radiusX 等于 80，radiusY 等于 80。起始角度是 30°，length 等于 35 (第 15 行)。arc1 的弧类型设为 ArcType.ROUND (第 18 行)。由于 arc1 的填充颜色是红色，因此 arc1 显示为红色填充。

程序创建了一条弧 arc3，其中心位于 (150, 100)，radiusX 等于 80，radiusY 等于 80。起始角度是 30°+180°，length 等于 35 (第 29 行)。arc3 的弧类型设为 ArcType.CHORD (第 31 行)。由于 arc3 的填充颜色是白色，笔划颜色是黑色，因此 arc3 显示为一条黑色轮廓的弦。

角度可以是负数。一个负的起始角度从最东的方向顺时针旋转一个角度，如图 14-37 所示。一个负的跨度角度从起始角度开始顺时针旋转一个角度。下面的语句定义了同样一条弧。

```

new Arc(x, y, radiusX, radiusY, -30, -20);
new Arc(x, y, radiusX, radiusY, -50, 20);

```

第 1 个语句使用负的起始角 -30°，以及负的跨度角度 -20°，如图 14-37a 所示。第 2 个语句使用负的起始角度 -50°，以及正的跨度角度 20°，如图 14-37b 所示。

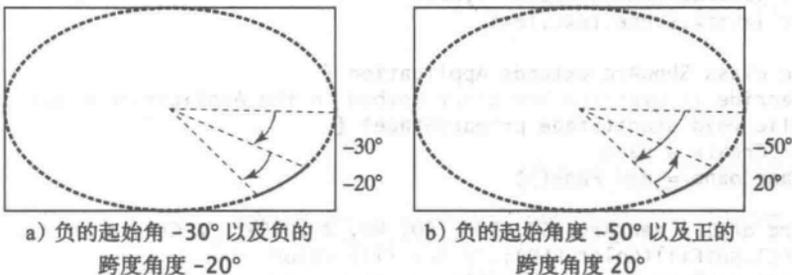


图 14-37 角度可以为负

14.11.6 Polygon 和 Polyline

Polygon 类定义一个连接一个点序列的多边形，如图 14-38a 所示。Polyline 类类似于 Polygon 类，不同之处是 Polyline 类不会自动闭合，如图 14-38b 所示。

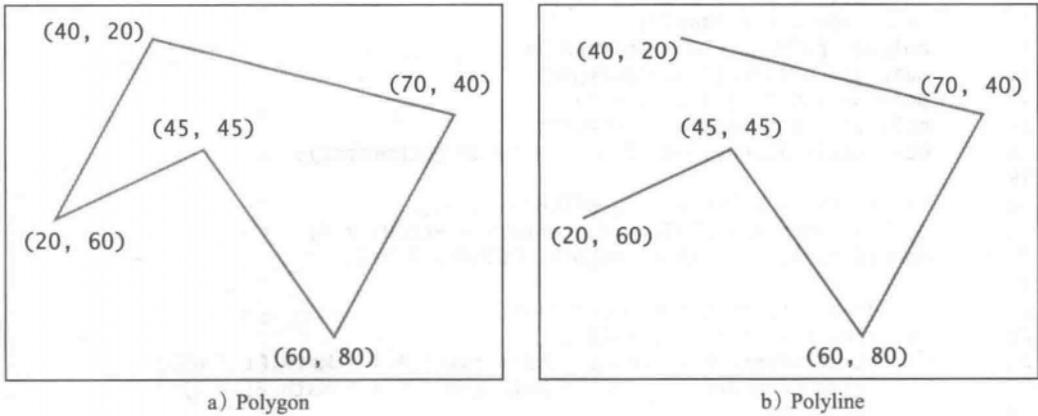


图 14-38 Polygon 是闭合的，而 Polyline 类不是闭合的

Polygon 类的 UML 图如图 14-39 所示。程序清单 14-19 给出了一个创建六边形的例子，如图 14-40 所示。

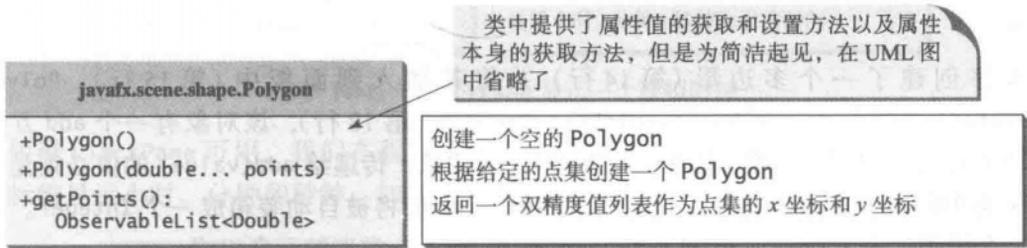


图 14-39 Polygon 类定义多边形

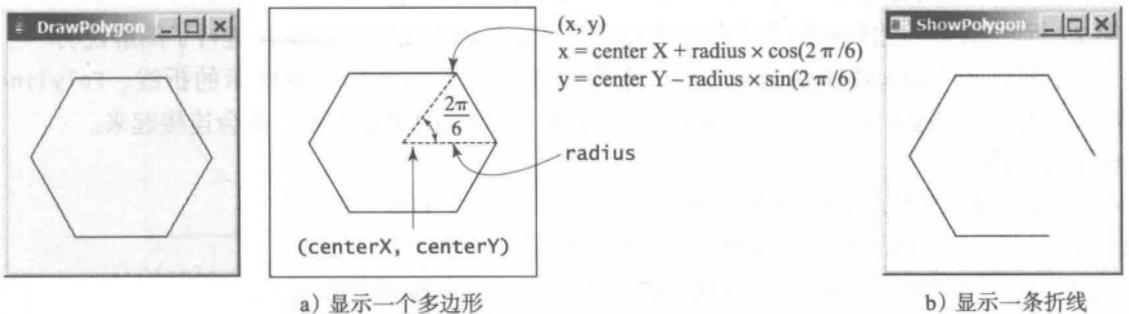


图 14-40

程序清单 14-19 ShowPolygon.java

```

1 import javafx.application.Application;
2 import javafx.collections.ObservableList;
3 import javafx.scene.Scene;
4 import javafx.scene.layout.Pane;
5 import javafx.scene.paint.Color;
6 import javafx.stage.Stage;
    
```

```

7 import javafx.scene.shape.Polygon;
8
9 public class ShowPolygon extends Application {
10     @Override // Override the start method in the Application class
11     public void start(Stage primaryStage) {
12         // Create a pane, a polygon, and place polygon to pane
13         Pane pane = new Pane();
14         Polygon polygon = new Polygon();
15         pane.getChildren().add(polygon);
16         polygon.setFill(Color.WHITE);
17         polygon.setStroke(Color.BLACK);
18         ObservableList<Double> list = polygon.getPoints();
19
20         final double WIDTH = 200, HEIGHT = 200;
21         double centerX = WIDTH / 2, centerY = HEIGHT / 2;
22         double radius = Math.min(WIDTH, HEIGHT) * 0.4;
23
24         // Add points to the polygon list
25         for (int i = 0; i < 6; i++) {
26             list.add(centerX + radius * Math.cos(2 * i * Math.PI / 6));
27             list.add(centerY - radius * Math.sin(2 * i * Math.PI / 6));
28         }
29
30         // Create a scene and place it in the stage
31         Scene scene = new Scene(pane, WIDTH, HEIGHT);
32         primaryStage.setTitle("ShowPolygon"); // Set the stage title
33         primaryStage.setScene(scene); // Place the scene in the stage
34         primaryStage.show(); // Display the stage
35     }
36 }

```

程序创建了一个多边形(第14行)并将其加入到面板中(第15行)。Polygon.getPoints()方法返回一个ObservableList<Double>(第18行),该对象有一个add方法用于将一个元素添加到列表中(第26~27行)。请注意,传递给add(value)的值必须是一个double类型的值。如果传递一个int类型的值,int值将被自动装箱成一个Integer。这样触发一个错误,因为ObservableList<Double>由Double类型的元素组成。

循环添加了6个点到多边形中(第25~28行)。每个点由其x和y坐标来表示。对于每个点,它的x坐标添加到多边形的列表中(第26行),然后它的y坐标添加到多边形的列表中(第27行)。计算六边形中每个点的x坐标和y坐标的公式在图14-40a中进行了图解说明。

如果将Polygon替换成Polyline,程序将显示一条如图14-40b所示的折线。Polyline类的使用和Polygon基本一样,不同之处是Polyline中的起点和终点不会连接起来。

🔪 复习题

- 14.27 如何显示文本、直线、矩形、圆、椭圆、弧、多边形、折线?
- 14.28 请写一段代码,显示围绕面板中心中旋转45°的一个字符串。
- 14.29 请写一段代码,显示一条从(10, 10)到(70, 30)的10像素宽的粗线。
- 14.30 请写一段代码,将一个矩形使用红色填充,该矩形的左上角位于(10, 10),宽度为100,高度为50。
- 14.31 请写一段代码,显示一个圆角的矩形,宽度为100,高度为200,左上角位于(10, 10),圆角处的水平直径为40,垂直直径为20。
- 14.32 请写一段代码,显示一个水平半径为50,垂直半径为100的椭圆。
- 14.33 请写一段代码,显示一个半径为50的圆的上半部轮廓。
- 14.34 请写一段代码,显示一个半径为50的圆的下半部,并使用红色填充。

- 14.35 请写一段代码，显示一个连接以下点并用绿色填充的多边形：(20, 40)、(30, 50)、(40, 90)、(90, 10)、(10, 30)。
- 14.36 请写一段代码，显示一条连接以下点的折线：(20, 40)、(30, 50)、(40, 90)、(90, 10)、(10, 30)。

14.12 示例学习：ClockPane 类

要点提示：本示例学习开发一个类，在一个面板中显示一个时钟。

ClockPane 类的合约显示在图 14-41 中。

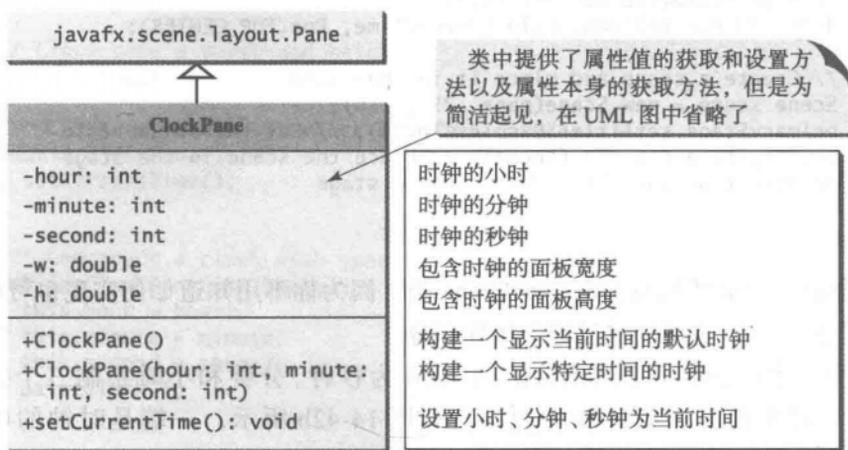
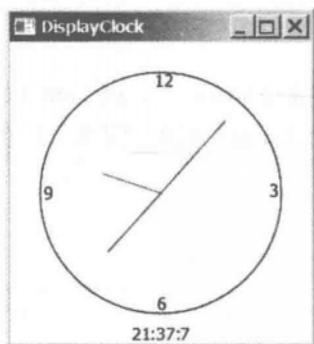
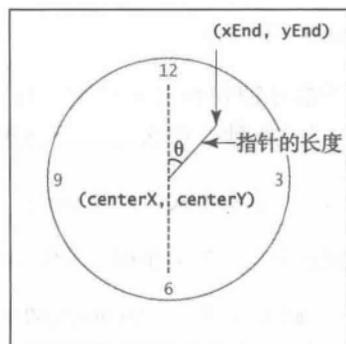


图 14-41 ClockPane 显示一个模拟时钟

假设 ClockPane 可用，我们在程序清单 14-20 中写一个测试程序来显示一个模拟时钟，使用标签显示小时、分钟和秒钟，如图 14-42 所示。



a) DisplayClock 程序显示一个时钟，可以给出当前时间



b) 给定跨越的角度、指针的长度以及中心点，可以确定一个时钟的指针终点

图 14-42

程序清单 14-20 DisplayClock.java

```

1 import javafx.application.Application;
2 import javafx.geometry.Pos;
3 import javafx.stage.Stage;
4 import javafx.scene.Scene;
5 import javafx.scene.control.Label;
6 import javafx.scene.layout.BorderPane:
  
```

```

7
8 public class DisplayClock extends Application {
9     @Override // Override the start method in the Application class
10    public void start(Stage primaryStage) {
11        // Create a clock and a label
12        ClockPane clock = new ClockPane();
13        String timeString = clock.getHour() + ":" + clock.getMinute()
14            + ":" + clock.getSecond();
15        Label lblCurrentTime = new Label(timeString);
16
17        // Place clock and label in border pane
18        BorderPane pane = new BorderPane();
19        pane.setCenter(clock);
20        pane.setBottom(lblCurrentTime);
21        BorderPane.setAlignment(lblCurrentTime, Pos.TOP_CENTER);
22
23        // Create a scene and place it in the stage
24        Scene scene = new Scene(pane, 250, 250);
25        primaryStage.setTitle("DisplayClock"); // Set the stage title
26        primaryStage.setScene(scene); // Place the scene in the stage
27        primaryStage.show(); // Display the stage
28    }
29 }

```

本节其余的部分解释如何实现 `ClockPane` 类。因为你不用知道如何实现也可以使用这个类，所以如果你愿意，也可以跳过这个实现部分。

若要绘制一个时钟，你需要绘制一个圆并为秒钟、分钟和小时绘制三个指针。为了画一个指针，需要确定一条直线的两端。如图 14-42b 所示，一端是时钟的中央，位于 $(centerX, centerY)$ ；另外一端位于 $(endX, endY)$ ，由以下公式来确定：

$$\begin{aligned}
 endX &= centerX + handLength \times \sin(\theta) \\
 endY &= centerY - handLength \times \cos(\theta)
 \end{aligned}$$

因为 1 分钟有 60 秒，所以第 2 个指针的角度是：

$$second \times (2\pi/60)$$

分针的位置由分钟和秒钟来决定。包含秒数的确切分钟数是 $minute + second/60$ 。例如，如果时间是 3 分 30 秒，那么总的分钟数是 3.5。由于 1 小时有 60 分钟，因此分针的角度是：

$$(minute + second/60) \times (2\pi/60)$$

由于一个圆被分为 12 个小时，所以时针的角度是：

$$(hour + minute/60 + second/(60 \times 60)) \times (2\pi/12)$$

为了简化，在计算分针和时针角度的时候，可以忽略秒针，因为它们数字太小，基本可以忽略。因此，秒针、分针以及时针的端点可以如下计算：

$$\begin{aligned}
 secondX &= centerX + secondHandLength \times \sin(second \times (2\pi/60)) \\
 secondY &= centerY - secondHandLength \times \cos(second \times (2\pi/60)) \\
 minuteX &= centerX + minuteHandLength \times \sin(minute \times (2\pi/60)) \\
 minuteY &= centerY - minuteHandLength \times \cos(minute \times (2\pi/60)) \\
 hourX &= centerX + hourHandLength \times \sin((hour + minute/60) \times (2\pi/12)) \\
 hourY &= centerY - hourHandLength \times \cos((hour + minute/60) \times (2\pi/12))
 \end{aligned}$$

`ClockPane` 类的实现如程序清单 14-21 所示：

程序清单 14-21 ClockPane.java

```
1 import java.util.Calendar;
2 import java.util.GregorianCalendar;
3 import javafx.scene.layout.Pane;
4 import javafx.scene.paint.Color;
5 import javafx.scene.shape.Circle;
6 import javafx.scene.shape.Line;
7 import javafx.scene.text.Text;
8
9 public class ClockPane extends Pane {
10     private int hour;
11     private int minute;
12     private int second;
13
14     // Clock pane's width and height
15     private double w = 250, h = 250;
16
17     /** Construct a default clock with the current time*/
18     public ClockPane() {
19         setCurrentTime();
20     }
21
22     /** Construct a clock with specified hour, minute, and second */
23     public ClockPane(int hour, int minute, int second) {
24         this.hour = hour;
25         this.minute = minute;
26         this.second = second;
27         paintClock();
28     }
29
30     /** Return hour */
31     public int getHour() {
32         return hour;
33     }
34
35     /** Set a new hour */
36     public void setHour(int hour) {
37         this.hour = hour;
38         paintClock();
39     }
40
41     /** Return minute */
42     public int getMinute() {
43         return minute;
44     }
45
46     /** Set a new minute */
47     public void setMinute(int minute) {
48         this.minute = minute;
49         paintClock();
50     }
51
52     /** Return second */
53     public int getSecond() {
54         return second;
55     }
56
57     /** Set a new second */
58     public void setSecond(int second) {
59         this.second = second;
60         paintClock();
61     }
62 }
```

```

63  /** Return clock pane's width */
64  public double getW() {
65      return w;
66  }
67
68  /** Set clock pane's width */
69  public void setW(double w) {
70      this.w = w;
71      paintClock();
72  }
73
74  /** Return clock pane's height */
75  public double getH() {
76      return h;
77  }
78
79  /** Set clock pane's height */
80  public void setH(double h) {
81      this.h = h;
82      paintClock();
83  }
84
85  /** Set the current time for the clock */
86  public void setCurrentTime() {
87      // Construct a calendar for the current date and time
88      Calendar calendar = new GregorianCalendar();
89
90      // Set current hour, minute and second
91      this.hour = calendar.get(Calendar.HOUR_OF_DAY);
92      this.minute = calendar.get(Calendar.MINUTE);
93      this.second = calendar.get(Calendar.SECOND);
94
95      paintClock(); // Repaint the clock
96  }
97
98  /** Paint the clock */
99  protected void paintClock() {
100     // Initialize clock parameters
101     double clockRadius = Math.min(w, h) * 0.8 * 0.5;
102     double centerX = w / 2;
103     double centerY = h / 2;
104
105     // Draw circle
106     Circle circle = new Circle(centerX, centerY, clockRadius);
107     circle.setFill(Color.WHITE);
108     circle.setStroke(Color.BLACK);
109     Text t1 = new Text(centerX - 5, centerY - clockRadius + 12, "12");
110     Text t2 = new Text(centerX - clockRadius + 3, centerY + 5, "9");
111     Text t3 = new Text(centerX + clockRadius - 10, centerY + 3, "3");
112     Text t4 = new Text(centerX - 3, centerY + clockRadius - 3, "6");
113
114     // Draw second hand
115     double sLength = clockRadius * 0.8;
116     double secondX = centerX + sLength *
117         Math.sin(second * (2 * Math.PI / 60));
118     double secondY = centerY - sLength *
119         Math.cos(second * (2 * Math.PI / 60));
120     Line sLine = new Line(centerX, centerY, secondX, secondY);
121     sLine.setStroke(Color.RED);
122
123     // Draw minute hand
124     double mLength = clockRadius * 0.65;
125     double xMinute = centerX + mLength *
126         Math.sin(minute * (2 * Math.PI / 60));

```

```

127     double minuteY = centerY - mLength *
128         Math.cos(minute * (2 * Math.PI / 60));
129     Line mLine = new Line(centerX, centerY, xMinute, minuteY);
130     mLine.setStroke(Color.BLUE);
131
132     // Draw hour hand
133     double hLength = clockRadius * 0.5;
134     double hourX = centerX + hLength *
135         Math.sin((hour % 12 + minute / 60.0) * (2 * Math.PI / 12));
136     double hourY = centerY - hLength *
137         Math.cos((hour % 12 + minute / 60.0) * (2 * Math.PI / 12));
138     Line hLine = new Line(centerX, centerY, hourX, hourY);
139     hLine.setStroke(Color.GREEN);
140
141     getChildren().clear();
142     getChildren().addAll(circle, t1, t2, t3, t4, sLine, mLine, hLine);
143 }
144 }

```

本程序使用无参构造方法显示一个指示当前时间的时钟（第 18 ~ 20 行），使用其他构造方法来显示一个指示给定小时、分钟和秒钟的时钟（第 23 ~ 28 行）。当前小时、分钟和秒钟通过 `GregorianCalendar` 类获得（第 86 ~ 96 行）。Java API 中的 `GregorianCalendar` 类可以使用它的无参构造方法来生成一个具有当前时间的 `Calendar` 实例。可以从一个 `Calendar` 对象，通过调用它的 `get(Calendar.HOUR)`、`get(Calendar.MINUTE)` 和 `get(Calendar.SECOND)` 方法返回小时、分钟以及秒钟。

该类定义了属性 `hour`、`minute` 以及 `second` 来存储该时钟表示的时间（第 10 ~ 12 行），使用 `w` 和 `h` 属性来表示时钟面板的宽度和高度（第 15 行）。`w` 和 `h` 的初始值设为 250。`w` 和 `h` 的值还可以使用 `setW` 和 `setH` 方法来重新设置（第 69 和 80 行）。`paintClock()` 方法中这些值用于在面板中绘制一个时钟。

`paintClock()` 方法绘制时钟（第 99 到 143 行）。时钟的半径和面板的宽度以及高度成正比（第 101 行）。一个代表时钟的圆绘制在面板中央（第 106 行）。显示小时数 12、3、6、9 的文本通过第 109 ~ 112 行代码生成。秒针、分针以及时针是第 114 ~ 139 行代码生成的直线。`paintClock()` 方法使用列表的 `addAll` 方法将所有这些形状加入到面板中（第 142 行）。因为 `paintClock()` 方法在任何一个新的属性值（`hour`、`minute`、`second`、`w` 以及 `h`）被设置（第 27、38、49、60、71、82、95 行）的时候调用，所以以前的内容从面板中被清除（第 141 行）。

关键术语

AWT (抽象窗体工具包)

bidirectional binding (双向绑定)

bindable object (可绑定对象)

binding object (绑定对象)

binding property (绑定属性)

JavaFX

node (节点)

observable object (可观察对象)

pane (面板)

property getter method (属性获取方法)

primary stage (主舞台)

shape (形状)

Swing

value getter method (值获取方法)

value setter method (值设置方法)

UI control (UI 组件)

unidirectional binding (单向绑定)

本章小结

1. JavaFX 是用于开发富因特网应用的新框架。JavaFX 完全替代了 Swing 和 AWT。
2. 一个 JavaFX 主类必须继承自 `javafx.application.Application` 并且实现 `start` 方法。主舞台由 JVM 自动生成并传递给 `start` 方法。
3. 舞台是用于显示一个场景的窗体。可以将一个节点加入到一个场景中。面板、组件以及形状都是节点。面板可以用作节点的容器。
4. 一个绑定属性可以绑定到一个可观察源对象上。源对象中的改变会自动反映到绑定属性上。一个绑定属性具有值获取方法、值设置方法以及属性获取方法。
5. `Node` 类定义了节点共同的许多属性。可以将这些属性应用于面板、组件和形状。
6. 可以使用指定的红、绿、蓝组件以及透明度值来生成一个 `Color` 对象。
7. 可以创建一个 `Font` 对象并设置它的名称、大小、粗细以及形态。
8. `javafx.scene.image.Image` 类可以用于装载一个图像，这个图像可以在一个 `ImageView` 中显示。
9. JavaFX 提供了许多类型的面板，用于自动布局其中节点到一个希望的位置和尺寸。`Pane` 类是所有面板的基类。它包含 `getChildren()` 方法以返回一个 `ObservableList`。可以使用 `ObservableList` 的 `add(node)` 和 `addAll(node1,node2,...)` 方法来添加节点到面板中。
10. `FlowPane` 将面板中的节点按照它们加入的次序，从左到右水平，或者从上到下垂直布局。`GridPane` 将节点布局在一个网格（矩阵）中。节点放置在特定的列和行序号上。`BorderPane` 可以将节点放置在 5 个区域：上、下、左、右以及居中。`HBox` 将其子节点放置在单个水平行中。`VBox` 将其子节点放置在单个垂直列中。
11. JavaFX 提供了许多的形状类用于绘制文本、直线、圆、矩形、椭圆、弧、多边形以及折线。

测试题

本章测试题的答案，位于 www.cs.armstrong.edu/liang/intro10e/quiz.html。

编程练习题

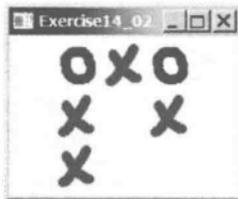
 **注意：**练习中用到的图像文件可以从 www.cs.armstrong.edu/liang/intro10e/book.zip 获得，并放置在 `image` 目录下。

14.2 ~ 14.9 节

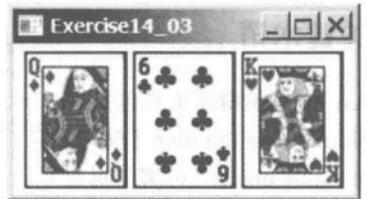
14.1（显示图像）请写一个程序，在一个网格面板里面显示 4 个图像，如图 14-43a 所示。



a) 练习题 14.1 显示 4 幅图像



b) 练习题 14.2 显示一个包含图像的井字棋盘



c) 三张扑克被随机选择

图 14-43

- *14.2（井字棋盘）请写一个程序，显示一个井字棋盘，如图 14-43b 所示。一个单元格中可能是 X、O 或者为空。每个单元格显示什么由随机决定的。X 和 O 是文件 `x.gif` 和 `o.gif` 中的图像。
- *14.3（显示三张牌）请写一个程序，显示从一副 52 张的扑克牌中随机选择的三张牌，如图 14-43c 所示。牌的图像文件命名为 `1.png`, `2.png`, ..., `52.png`，并保存在 `image/card` 目录下。三张牌都是

不同的并且是随机选取的。

提示：可以这样随机选择牌，先将数字 1 ~ 52 保存在一个数组列表中，按照 11.12 节中介绍的方法进行一次随机洗牌，然后使用数组列表中前面三个数字作为图像的文件名。

14.4 (颜色和字体) 请写一个程序，可以垂直显示 5 个文字，如图 14-44a 所示。对每个文字设置一个随机颜色和透明度，并且将每个文字的字体设置为 TimesRomes、bold、italic，大小为 22 像素。

14.5 (围绕成一个圆的字符) 请写一个程序，显示一个围绕着一个圆显示的字符串“Welcome to Java”，如图 14-44b 所示。

提示：需要使用循环来将每个字符通过合适的旋转显示在正确的位置上。

*14.6 (游戏：显示一个象棋棋盘) 请写一个程序显示一个象棋棋盘，其中每个黑白单元格都是一个填充了黑色或者白色的 Rectangle，如图 14-44c 所示。



a) 使用随机的颜色和给定的字体显示 5 个文字

b) 围绕着一个圆显示一个字符串

c) 使用矩形显示一个象棋棋盘

图 14-44

14.10 ~ 14.11 节

*14.7 (显示随机的 0 或者 1) 请写一个程序，显示一个 10×10 的方阵，如图 14-45a 所示。矩阵中的每个元素是随机产生的 0 或者 1。将每个数字居中显示在一个文本域中。使用 TextField 的 setText 方法来设置 0 和 1 作为字符串显示。

14.8 (显示 54 张牌) 扩充练习题 14.3 以显示所有 54 张牌 (包括两个王)，每行显示 9 张牌。两个王的图像文件命名为 53.jpg 和 54.jpg。

*14.9 (创建四个风扇) 请写一个程序，将四个风扇按照两行两列置于一个 GridPane 中，如图 14-45b 所示。

*14.10 (显示一个圆柱) 请写一个绘制圆柱的程序，如图 14-45c 所示。可以使用如下方法来用虚线显示弧：

```
arc.getStrokeDashArray().addAll(6.0, 21.0);
```



a) 随机产生 0 和 1 数字的程序

b) 练习题 14.9 绘制 4 个风扇

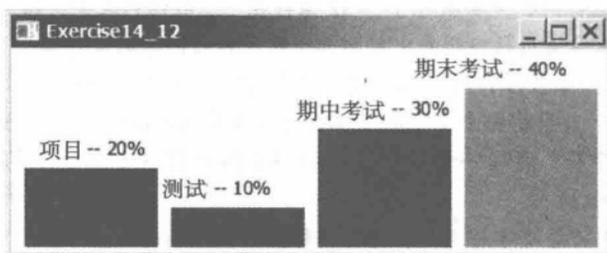
c) 练习题 14.10 绘制一个圆柱

图 14-45

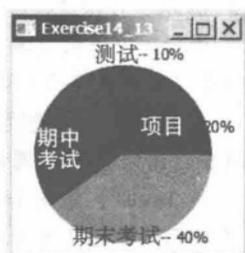
*14.11 (绘制一个笑脸) 请写一个绘制笑脸的程序, 如图 14-46a 所示。



a) 练习题 14.11 绘制一个笑脸



b) 练习题 14.12 绘制一个柱形图



c) 练习题 14.13 绘制一个饼图

图 14-46

**14.12 (显示一个柱形图) 请写一个程序, 使用柱形图来显示一个总成绩的各个组成部分的百分比, 包括项目、测试、期中考试和期末考试, 如图 14-46b 所示。假设项目占 20% 并显示为红色, 测试占 10% 并显示为蓝色, 期中考试占 30% 并显示为绿色, 期末考试占 40% 并显示为橙色。使用 `Rectangle` 类来显示柱形。有兴趣的读者可以探索使用 JavaFX 的 `BarChart` 类来进一步学习。

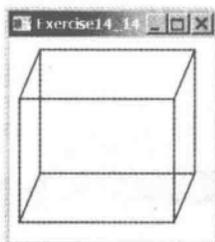
**14.13 (显示一个饼图) 请写一个程序, 使用饼图来显示一个总成绩的各个组成部分的百分比, 包括项目、测试、期中考试和期末考试, 如图 14-46c 所示。假设项目占 20% 并显示为红色, 测试占 10% 并显示为蓝色, 期中考试占 30% 并显示为绿色, 期末考试占 40% 并显示为橙色。使用 `Arc` 类来显示饼状图。有兴趣的读者可以探索使用 JavaFX 的 `PieChart` 类来进一步学习。

14.14 (显示一个立方体) 请写一个绘制立方体的程序, 如图 14-47a 所示。该立方体应该可以随着窗体的伸缩自动伸缩。

*14.15 (显示一个 STOP 标识) 请写一个绘制 STOP 标识的程序, 如图 14-47b 所示。六边形使用红色填充, 标识文字使用白色字体。

 提示: 将一个六边形和一个文本放置在一个栈面板中。

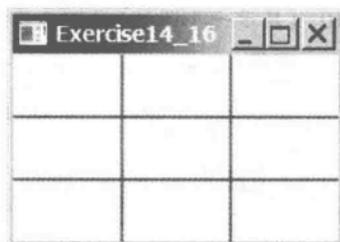
*14.16 (显示一个 3×3 的网格) 请写一个绘制 3×3 网格的程序, 如图 14-47c 所示。使用红色绘制垂直线, 蓝色绘制水平线。当窗体改变大小的时候, 这些线条自动改变大小。



a) 练习题 14.14 绘制一个立方体



b) 练习题 14.15 绘制一个 STOP 标识



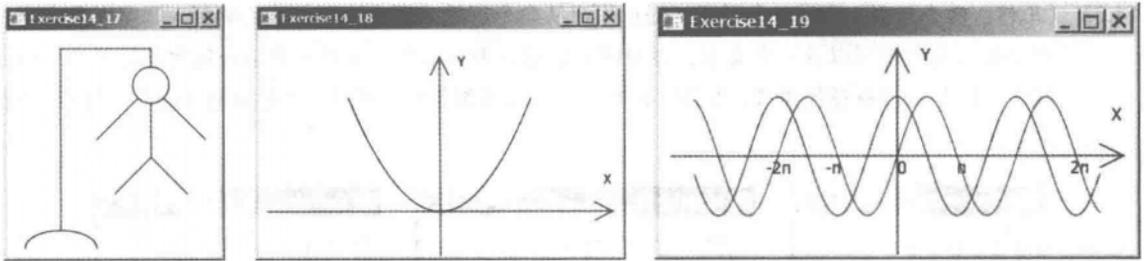
c) 练习题 14.16 绘制一个网格

图 14-47

14.17 (游戏: 猜字游戏 (hangman)) 请写一个程序, 显示一个图片用于流行的猜字游戏, 如图 14-48a 所示。

*14.18 (绘制平方函数) 请写一个程序, 绘制表示函数 $f(x)=x^2$ 的图 (参见图 14-48b)。

 提示: 使用以下代码将点加入到折线中。



a) 练习题 14.17 绘制一个用于猜字游戏的草图

b) 练习题 14.18 绘制一个二次函数

c) 练习题 14.19 绘制一个正弦 / 余弦函数

图 14-48

```
Polyline polyline = new Polyline();
ObservableList<Double> list = polyline.getPoints();
double scaleFactor = 0.0125;
for (int x = -100; x <= 100; x++) {
    list.add(x + 200.0);
    list.add(scaleFactor * x * x);
}
```

**14.19 (绘制正弦和余弦函数) 请写一个程序, 使用红色绘制正弦函数, 使用蓝色绘制余弦函数, 如图 14-48c 所示。

提示: π 的 Unicode 码龙骧虎步是 `\u03c0`。要显示 -2π , 使用 `Text(x,y,"-2\u03c0")`。对于一个像 $\sin(x)$ 这样的三角函数, 其中 x 使用弧度。使用下面的循环将点加到折线中。

```
Polyline polyline = new Polyline();
ObservableList<Double> list = polyline.getPoints();
double scaleFactor = 50;
for (int x = -170; x <= 170; x++) {
    list.add(x + 200.0);
    list.add(100 - 50 * Math.sin((x / 100.0) * 2 * Math.PI));
}
```

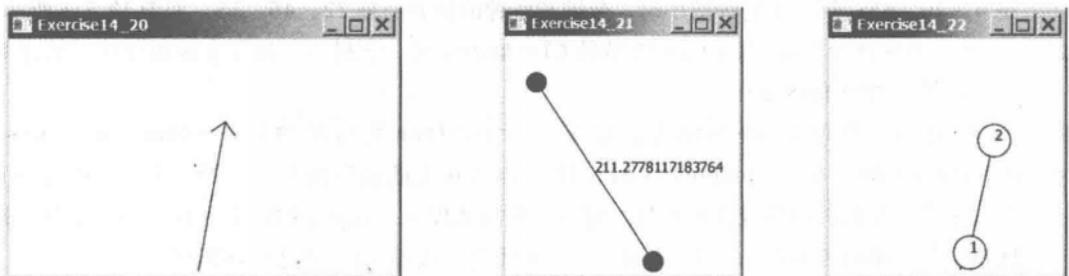
**14.20 (绘制一条箭线) 使用静态方法在面板中绘制一条从起点到终点的箭线, 使用如下方法头:

```
public static void drawArrowLine(double startX, double startY,
    double endX, double endY, Pane pane)
```

请写一个程序, 随机绘制一条箭线, 如图 14-49a 所示。

*14.21 (两个圆以及它们的距离) 请写一个程序, 绘制两个半径为 15 像素的实心圆, 圆心位于一个随机位置; 同时绘制一条直线连接两个圆。两个圆心的距离显示在直线上, 如图 14-49b 所示。

*14.22 (连接两个圆) 请写一个程序, 绘制两个半径为 15 像素的圆, 圆心位于一个随机位置; 同时绘制一条直线连接两个圆。直线不能穿到圆内, 如图 14-49c 所示。



a) 该程序显示一条箭线

b) 练习题 14.21 连接两个实心圆的圆心

c) 练习题 14.22 从外围连接两个圆

图 14-49

- *14.23 (几何: 两个矩形) 请写一个程序, 提示用户从命令行输入两个矩形的中心坐标、宽度以及高度。程序显示两个矩形以及一个文本, 表明两个矩形是否有重叠, 或者一个是否包含在另外一个内, 或者它们是否没有任何重叠, 如图 14-50 所示。参考编程练习题 10.13 是如何判断两个矩形之间关系的。

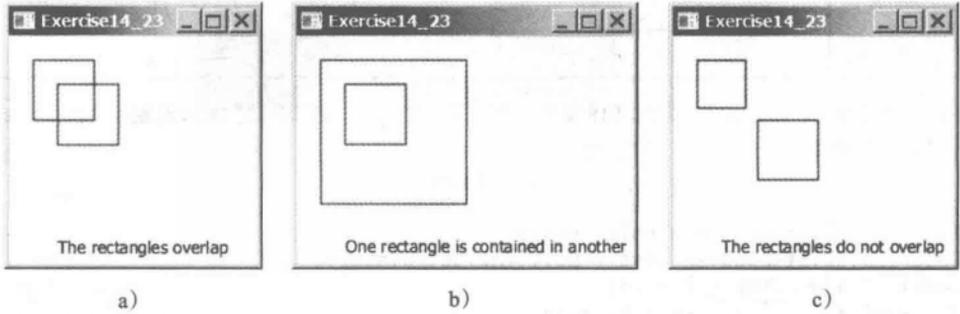
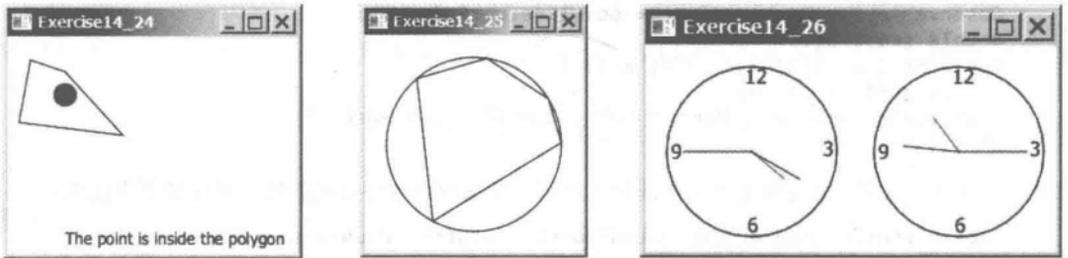


图 14-50 显示两个矩形

- *14.24 (几何: 在一个多边形内吗?) 请写一个程序, 提示用户从命令行输入 5 个点的坐标。前面 4 个点构成一个多边形, 程序显示该多边形以及一个文本指出第 5 个点是否在这个多边形中, 如图 14-51a 所示。使用 Node 的 contains 方法测试一个点是否在一个节点中。
- *14.25 (一个圆上的随机点) 修改编程练习题 4.6, 在一个圆上创建 5 个随机点, 顺时针连接这 5 个点构建一个多边形, 然后显示这个圆以及多边形, 如图 14-51b 所示。



a) 显示一个多边形和一个点 b) 练习题 14.25 连接一个圆上面的 5 个随机点 c) 练习题 14.26 显示两个时钟

图 14-51

14.12 节

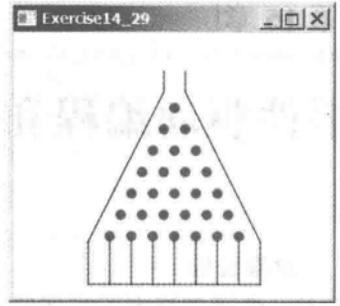
- 14.26 (使用 ClockPane 类) 请写一个程序显示两个时钟。第一个时钟的小时、分钟和秒钟的值分别是 4、20、45; 第二个时钟的小时、分钟和秒钟的值分别是 22、46、15, 如图 14-51c 所示。
- *14.27 (绘制一个详细的时钟) 修改 14.12 节的 ClockPane 类, 绘制一个包含小时和分钟更加详细信息的时钟, 如图 14-52a 所示。
- *14.28 (随机时间) 修改 ClockPane 类, 添加三个 Boolean 类型的属性——hourHandVisible、minuteHandVisible、secondHandVisible 以及相关的访问器和转变器方法。可以使用 set 方法来使一个指针可见或者不可见。请写一个测试程序, 只显示时针和分针。小时和分钟的值随机产生。小时的值在 0~11 之间, 分钟的值是 0 或者 30, 如图 14-52b 所示。
- **14.29 (游戏: 豆机) 请写一个程序, 显示编程练习题 7.21 中介绍的豆机, 如图 14-52c 所示。



a) 练习题 14.27 显示一个详细的时钟



b) 练习题 14.28 显示一个具有随机小时和分钟值的时钟



c) 练习题 14.29 显示一个豆机

图 14-52

事件驱动编程和动画

教学目标

- 尝试进行事件驱动编程 (15.1 节)。
- 描述事件、事件源以及事件类 (15.2 节)。
- 定义处理器类、注册处理器对象和源对象, 编写代码处理器事件 (15.3 节)。
- 使用内部类定义处理器类 (15.4 节)。
- 使用匿名内部类定义处理器类 (15.5 节)。
- 使用 lambda 表达式简化事件处理 (15.6 节)。
- 开发 GUI 程序完成一个借贷计算器 (15.7 节)。
- 编写处理 MouseEvent 事件的程序 (15.8 节)。
- 编写处理 KeyEvent 事件的程序 (15.9 节)。
- 创建监听器以处理一个可观察对象中值的改变 (15.10 节)。
- 使用 Animation、PathTransition、FadeTransition 和 Timeline 类开发动画 (15.11 节)。
- 开发一个模拟弹球的动画 (15.12 节)。

15.1 引言

要点提示: 可以编写代码以处理诸如单击按钮、鼠标移动以及按键盘之类的事件。

假设你希望写一个 GUI 程序, 可以让用户输入一个贷款数额、年利率以及年数, 然后单击 Calculate 按钮获得每个月的还款额以及总还款额, 如图 15-1 所示。你如何完成这个任务呢? 你需要使用事件驱动编程来编写代码, 以对按钮单击事件进行反应。

在直接进入事件驱动编程之前, 使用一个简单的例子进行尝试会比较有帮助。这个例子在一个面板中显示两个按钮, 如图 15-2 所示。

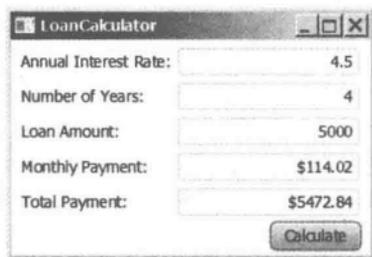


图 15-1 程序计算贷款还款额



a) 程序显示两个按钮

b) 当单击按钮后, 在控制台显示一条消息

图 15-2

为了响应一个按钮单击事件, 你需要编写代码来处理按钮单击动作。按钮是一个事件源对象, 即动作起源的地方。你需要创建一个能对一个按钮动作事件进行处理的对象。该对象

称为一个事件处理器，如图 15-3 所示。



图 15-3 一个事件处理器处理从源对象上触发出来的事件

不是所有对象都可以成为一个动作事件的处理器。要成为一个动作事件的处理器，必须满足两个要求：

1) 该对象必须是 `EventHandler <T extends Event>` 接口的一个示例。接口定义了所有处理器的共同行为。`<T extends Event>` 表示 `T` 是一个 `Event` 子类型的泛型。

2) `EventHandler` 对象 `handler` 必须使用方法 `source.setOnAction(handler)` 和事件源对象注册。

`EventHandler <ActionEvent>` 接口包含了 `handle(ActionEvent)` 方法用于处理动作事件。你的处理器类必须覆盖这个方法来响应事件。程序清单 15-1 给出了处理两个按钮上 `ActionEvent` 事件的代码。当单击 OK 按钮的时候，将显示消息“OK button clicked”。当单击 Cancel 按钮的时候，将显示消息“Cancel button clicked”，如图 15-2 所示。

程序清单 15-1 HandleEvent.java

```

1  import javafx.application.Application;
2  import javafx.geometry.Pos;
3  import javafx.scene.Scene;
4  import javafx.scene.control.Button;
5  import javafx.scene.layout.HBox;
6  import javafx.stage.Stage;
7  import javafx.event.ActionEvent;
8  import javafx.event.EventHandler;
9
10 public class HandleEvent extends Application {
11     @Override // Override the start method in the Application class
12     public void start(Stage primaryStage) {
13         // Create a pane and set its properties
14         HBox pane = new HBox(10);
15         pane.setAlignment(Pos.CENTER);
16         Button btOK = new Button("OK");
17         Button btCancel = new Button("Cancel");
18         OKHandlerClass handler1 = new OKHandlerClass();
19         btOK.setOnAction(handler1);
20         CancelHandlerClass handler2 = new CancelHandlerClass();
21         btCancel.setOnAction(handler2);
22         pane.getChildren().addAll(btOK, btCancel);
23
24         // Create a scene and place it in the stage
25         Scene scene = new Scene(pane);
26         primaryStage.setTitle("HandleEvent"); // Set the stage title
27         primaryStage.setScene(scene); // Place the scene in the stage
28         primaryStage.show(); // Display the stage
29     }
30 }
31
32 class OKHandlerClass implements EventHandler<ActionEvent> {
33     @Override
34     public void handle(ActionEvent e) {
35         System.out.println("OK button clicked");

```

```

36     }
37 }
38
39 class CancelHandlerClass implements EventHandler<ActionEvent> {
40     @Override
41     public void handle(ActionEvent e) {
42         System.out.println("Cancel button clicked");
43     }
44 }

```

第 32 ~ 44 行定义了两个处理类。每个处理类实现了 `EventHandler<ActionEvent>` 以处理 `ActionEvent`。对象 `handler1` 是一个 `OKHandlerClass` 实例 (第 18 行), 该实例通过按钮 `btOK` 注册 (第 19 行)。当单击 OK 按钮时, `OKHandlerClass` 的 `handle(ActionEvent)` 方法 (第 34 行) 被调用以处理事件。对象 `handler2` 是一个 `CancelHandlerClass` 实例 (第 20 行), 该实例通过按钮 `btCancel` 注册 (第 21 行)。当单击 Cancel 按钮时, `CancelHandlerClass` 的 `handle(ActionEvent)` 方法 (第 41 行) 被调用以处理事件。

你现在对 JavaFX 的事件驱动编程有了初步了解。你也许会有许多问题, 比如为什么一个处理器类要定义为实现 `EventHandler<ActionEvent>`。下面的章节会给出所有的答案。

15.2 事件和事件源

要点提示: 事件是从一个事件源上产生的对象。触发一个事件意味着产生一个事件并委派处理器处理该事件。

当运行一个 Java GUI 程序的时候, 程序和用户进行交互, 并且事件驱动它的执行。这被称为事件驱动编程。一个事件可以被定义为一个告知程序某件事发生的信号。事件由外部的用户动作, 比如鼠标的移动、单击和键盘按键所触发。程序可以选择响应或者忽略一个事件。前面的例子让你体验了事件驱动编程。

产生一个事件并且触发它的组件称为事件源对象, 或者简单称为源对象或者源组件。例如, 一个按钮是一个按钮单击动作事件的源对象。一个事件是一个事件类的实例。Java 事件类的根类是 `java.util.EventObject`。JavaFX 的事件类的根类是 `javafx.event.Event`。一些事件类的层次关系显示在图 15-4 中。

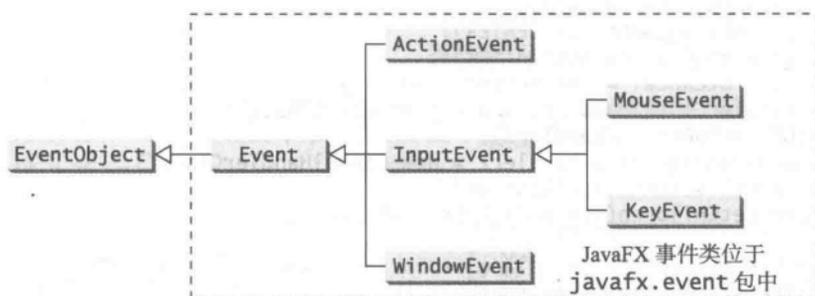


图 15-4 一个 JavaFX 中的事件是 `javafx.event.Event` 类的一个对象

一个事件对象包含与事件相关的任何属性。可以通过 `EventObject` 类中的 `getSource()` 实例方法来确定一个事件的源对象。`EventObject` 的子类处理特定类型的事件, 比如动作事件、窗口事件、鼠标事件以及键盘事件等。表 15-1 的前面三列给出了一些外部用户动作、源对象以及触发的事件类型。例如, 当单击一个按钮时, 按钮创建并触发一个 `ActionEvent`, 如表 15-1 的第一行所示。这里, 一个按钮是一个事件源对象, 一个 `ActionEvent` 是一个由

源对象触发的事件对象，如图 15-3 所示。

注意：如果一个组件可以触发一个事件，那么这个组件的任何子类都可以触发同样类型的事件。比如，每个 JavaFX 形状、布局面板和组件都可以触发 `MouseEvent` 和 `KeyEvent` 事件，因为 `Node` 是形状、布局面板和组件的超类。

表 15-1 用户动作、源对象、事件类型、处理器接口以及处理器

用户动作	源对象	触发的事件类型	事件注册方法
单击一个按钮	Button	ActionEvent	setOnAction(EventHandler<ActionEvent>)
在一个文本域中回车	TextField	ActionEvent	setOnAction(EventHandler<ActionEvent>)
勾选或者取消勾选	RadioButton	ActionEvent	setOnAction(EventHandler<ActionEvent>)
勾选或者取消勾选	CheckBox	ActionEvent	setOnAction(EventHandler<ActionEvent>)
选择一个新的项	ComboBox	ActionEvent	setOnAction(EventHandler<ActionEvent>)
按下鼠标	Node、Scene	MouseEvent	setOnMousePressed(EventHandler<MouseEvent>)
释放鼠标			setOnMouseReleased(EventHandler<MouseEvent>)
单击鼠标			setOnMouseClicked(EventHandler<MouseEvent>)
鼠标进入			setOnMouseEntered(EventHandler<MouseEvent>)
鼠标退出			setOnMouseExited(EventHandler<MouseEvent>)
鼠标移动			setOnMouseMoved(EventHandler<MouseEvent>)
鼠标拖动			setOnMouseDragged(EventHandler<MouseEvent>)
按下键	Node、Scene	KeyEvent	setOnKeyPressed(EventHandler<KeyEvent>)
释放键			setOnKeyReleased(EventHandler<KeyEvent>)
敲击键			setOnKeyTyped(EventHandler<KeyEvent>)

复习题

- 15.1 什么是事件源对象？什么是事件对象？描述事件源对象和事件对象之间的关系。
- 15.2 一个按钮可以触发一个 `MouseEvent` 事件吗？一个按钮可以触发一个 `KeyEvent` 事件吗？一个按钮可以触发一个 `ActionEvent` 事件吗？

15.3 注册处理器和处理事件

要点提示：处理器是一个对象，它必须通过一个事件源对象进行注册，并且它必须是一个恰当的事件处理接口的实例。

Java 采用一个基于委派的模型来进行事件处理：一个源对象触发一个事件，然后一个对该事件感兴趣的对象处理它。后者称为一个事件处理器或者一个事件监听者。一个对象如果要成为一个源对象上面事件的处理器，那么需要满足两个条件，如图 15-5 所示。

1) 处理器对象必须是一个对应的事件处理接口的实例，从而保证该处理器具有处理事件的正确方法。JavaFX 定义了一个对于事件 T 的统一的处理器接口 `EventHandler<T extends Event>`。该处理器接口包含 `handle(T e)` 方法用于处理事件。例如，对于 `ActionEvent` 来说，处理器接口是 `EventHandler<ActionEvent>`。`ActionEvent` 的每个处理器应该实现 `handle(ActionEvent e)` 方法从而处理一个 `ActionEvent`。

2) 处理器对象必须通过源对象进行注册。注册方法依赖于事件类型。对 `ActionEvent` 而言，方法是 `setOnAction`。对一个鼠标按下事件来说，方法是 `setOnMousePressed`。对于一个按键事件，方法是 `setOnKeyPressed`。

我们来重新看下程序清单 15-1。由于一个 `Button` 对象触发了一个 `ActionEvent`，而

ActionEvent 的处理器对象必须是 EventHandler<ActionEvent> 的实例，所以在第 32 行处理器对象实现了 EventHandler<ActionEvent>。源对象调用 setOnAction(handler) 来注册一个处理器，如下所示：

```
Button btOK = new Button("OK"); // Line 16 in Listing 15.1
OKHandlerClass handler1 = new OKHandlerClass(); // Line 18 in Listing 15.1
btOK.setOnAction(handler1); // Line 19 in Listing 15.1
```

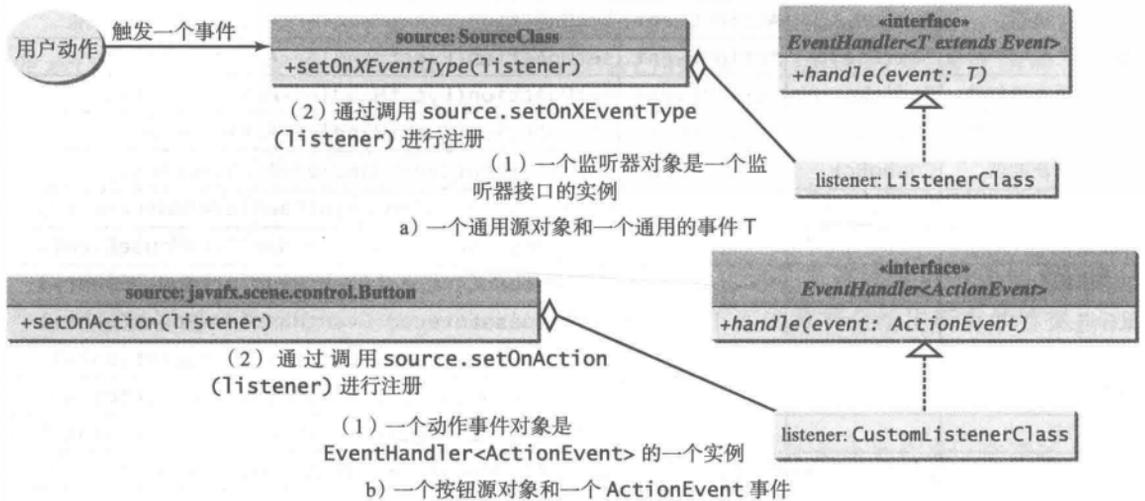


图 15-5 一个监听器必须是一个监听器接口的实例，并且必须通过一个源对象进行注册

当单击按钮时，Button 对象触发一个 ActionEvent 并将它传递给处理器的 handle (ActionEvent) 方法以处理该事件。事件对象包含了和该事件相关的信息，这些信息可以通过一些方法得到。比如，可以使用 e.getSource() 来得到触发该事件的源对象。

现在我们来编写一个程序，可以使用两个按钮来控制一个圆的大小，如图 15-6 所示。我们将逐步完善该程序。首先，我们写一个如程序清单 15-2 所示的程序来显示一个用户界面，其中包含一个居中的圆（第 15 ~ 19 行）以及位于底部的两个按钮（第 21 ~ 27 行）。

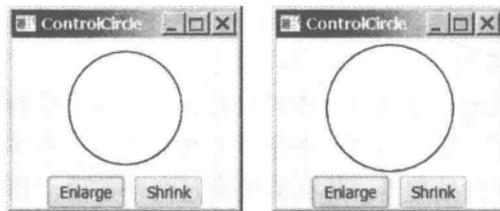


图 15-6 用户可以单击 Enlarge 和 Shrink 按钮来放大和缩小圆的尺寸

程序清单 15-2 ControlCircleWithoutEventHandling.java

```
1 import javafx.application.Application;
2 import javafx.geometry.Pos;
3 import javafx.scene.Scene;
4 import javafx.scene.control.Button;
5 import javafx.scene.layout.StackPane;
6 import javafx.scene.layout.HBox;
7 import javafx.scene.layout.BorderPane;
8 import javafx.scene.paint.Color;
9 import javafx.scene.shape.Circle;
```

```
10 import javafx.stage.Stage;
11
12 public class ControlCircleWithoutEventHandling extends Application {
13     @Override // Override the start method in the Application class
14     public void start(Stage primaryStage) {
15         StackPane pane = new StackPane();
16         Circle circle = new Circle(50);
17         circle.setStroke(Color.BLACK);
18         circle.setFill(Color.WHITE);
19         pane.getChildren().add(circle);
20
21         HBox hBox = new HBox();
22         hBox.setSpacing(10);
23         hBox.setAlignment(Pos.CENTER);
24         Button btEnlarge = new Button("Enlarge");
25         Button btShrink = new Button("Shrink");
26         hBox.getChildren().add(btEnlarge);
27         hBox.getChildren().add(btShrink);
28
29         BorderPane borderPane = new BorderPane();
30         borderPane.setCenter(pane);
31         borderPane.setBottom(hBox);
32         BorderPane.setAlignment(hBox, Pos.CENTER);
33
34         // Create a scene and place it in the stage
35         Scene scene = new Scene(borderPane, 200, 150);
36         primaryStage.setTitle("ControlCircle"); // Set the stage title
37         primaryStage.setScene(scene); // Place the scene in the stage
38         primaryStage.show(); // Display the stage
39     }
40 }
```

怎样才能使用按钮来放大和缩小圆呢？当单击 Enlarge 按钮时，你会希望圆以一个更大的半径被重画。如何来完成呢？可以扩充和修改程序清单 15-2 中的程序为程序清单 15-3，使之具有以下特点：

1) 定义一个新的类 CirclePane 用于显示面板中的圆（第 51 ~ 68 行）。这个新的类显示一个圆并且提供了 enlarge 和 shrink 方法用于增加和缩小圆的半径（第 60 ~ 62 行，第 64 ~ 67 行）。设计一个类来建模一个包含了支持方法的圆面板是一个好的策略，这样这些相关的方法和圆都耦合在一个对象中了。

2) 在 ControlCircle 类中，创建一个 CirclePane 对象，并且将 circlePane 声明为一个数据域来引用该对象（第 15 行）。ControlCircle 类中的方法现在可以通过该数据域来访问 CirclePane 的对象了。

3) 定义一个名为 EnlargeHandler 的处理器类，实现 EventHandler<ActionEvent>（第 43 ~ 48 行）。为了使得引用变量 circlePane 可以从 handle 方法中访问到，定义 EnlargeHandler 为 ControlCircle 类的一个内部类。（内部类是定义在其他类中的类。我们这里先使用内部类，下一节会完整地进行介绍。）

4) 为 Enlarge 按钮注册处理器（第 29 行）并且实现 EnlargeHandler 中的 handle 方法用于调用 circlePane.enlarge()（第 46 行）。

程序清单 15-3 ControlCircle.java

```
1 import javafx.application.Application;
2 import javafx.event.ActionEvent;
3 import javafx.event.EventHandler;
4 import javafx.geometry.Pos;
```

```

5  import javafx.scene.Scene;
6  import javafx.scene.control.Button;
7  import javafx.scene.layout.StackPane;
8  import javafx.scene.layout.HBox;
9  import javafx.scene.layout.BorderPane;
10 import javafx.scene.paint.Color;
11 import javafx.scene.shape.Circle;
12 import javafx.stage.Stage;
13
14 public class ControlCircle extends Application {
15     private CirclePane circlePane = new CirclePane();
16
17     @Override // Override the start method in the Application class
18     public void start(Stage primaryStage) {
19         // Hold two buttons in an HBox
20         HBox hBox = new HBox();
21         hBox.setSpacing(10);
22         hBox.setAlignment(Pos.CENTER);
23         Button btEnlarge = new Button("Enlarge");
24         Button btShrink = new Button("Shrink");
25         hBox.getChildren().add(btEnlarge);
26         hBox.getChildren().add(btShrink);
27
28         // Create and register the handler
29         btEnlarge.setOnAction(new EnlargeHandler());
30
31         BorderPane borderPane = new BorderPane();
32         borderPane.setCenter(circlePane);
33         borderPane.setBottom(hBox);
34         BorderPane.setAlignment(hBox, Pos.CENTER);
35
36         // Create a scene and place it in the stage
37         Scene scene = new Scene(borderPane, 200, 150);
38         primaryStage.setTitle("ControlCircle"); // Set the stage title
39         primaryStage.setScene(scene); // Place the scene in the stage
40         primaryStage.show(); // Display the stage
41     }
42
43     class EnlargeHandler implements EventHandler<ActionEvent> {
44         @Override // Override the handle method
45         public void handle(ActionEvent e) {
46             circlePane.enlarge();
47         }
48     }
49 }
50
51 class CirclePane extends StackPane {
52     private Circle circle = new Circle(50);
53
54     public CirclePane() {
55         getChildren().add(circle);
56         circle.setStroke(Color.BLACK);
57         circle.setFill(Color.WHITE);
58     }
59
60     public void enlarge() {
61         circle.setRadius(circle.getRadius() + 2);
62     }
63
64     public void shrink() {
65         circle.setRadius(circle.getRadius() > 2 ?
66             circle.getRadius() - 2 : circle.getRadius());
67     }
68 }

```

作为一个练习，请添加处理 shrink 按钮的代码，当 Shrink 按钮被单击的时候，显示一个变小的圆。

复习题

- 15.3 为什么一个处理器必须是一个恰当的处理接口实例？
- 15.4 请说明如何注册一个处理器对象，以及如何实现一个处理器接口？
- 15.5 `EventHandler<ActionEvent>` 接口的处理器方法是什么？
- 15.6 对一个按钮注册一个 `ActionEvent` 处理器的注册方法是什么？

15.4 内部类

要点提示：内部类，或者称为嵌套类，是一个定义在另外一个类范围中的类。内部类对于定义处理器类非常有用。

在前面一节中我们使用了内部类。本节详细介绍内部类。首先，让我们看下图 15-7 中的代码。图 15-7a 中的代码定义了两个分开的类，`Test` 和 `A`。图 15-7b 中的代码将 `A` 定义为 `Test` 中的一个内部类。

<pre style="border: 1px solid black; padding: 5px;">public class Test { ... } public class A { ... }</pre> <p style="text-align: center;">a)</p>	<pre style="border: 1px solid black; padding: 5px;">// OuterClass.java: inner class demo public class OuterClass { private int data; /** A method in the outer class */ public void m() { // Do something } // An inner class class InnerClass { /** A method in the inner class */ public void mi() { // Directly reference data and method, // defined in its outer class data++; m(); } } }</pre> <p style="text-align: center;">c)</p>
<pre style="border: 1px solid black; padding: 5px;">public class Test { ... // Inner class public class A { ... } }</pre> <p style="text-align: center;">b)</p>	

图 15-7 内部类将相互依赖的类结合成一个主类

图 15-7c 中示例的类 `InnerClass` 定义在 `OuterClass` 中，是内部类的另外一个例子。一个内部类可以如常规类一样使用。通常，在一个类只被它的外部类所使用的时候，才将它定义为内部类。一个内部类具有以下特征。

- 一个内部类被编译成一个名为 `OuterClassName$InnerClassName` 的类。例如，图 15-7b 中示例的 `Test` 中的内部类 `A` 被编译成 `Test$A.class`。
- 一个内部类可以引用定义在它所在的外部类中的数据和类方法。所以，你没有必要将外部类对象的引用传递给内部类的构造方法。基于这个原因，内部类可以使得程序更加精简。例如，程序清单 15-3 中 `circlePane` 定义在 `ControlCircle` 中（第 15 行）。它可以被内部类 `EnlargerHandler` 引用（第 46 行）。
- 一个内部类可以使用可见性修饰符所定义，和应用于一个类中成员的可见性规则一样。
- 一个内部类可以被定义为 `static`。一个 `static` 的内部类可以使用外部类的名字所访

问。一个 `static` 的内部类不能访问外部类中非静态的成员。

- 内部类对象通常在外部类中所创建。但是你也可以从另外一个类中来创建一个内部类的对象。如果内部类是非静态的，你必须先创建一个外部类的实例，然后使用以下语法来创建一个内部类的对象：

```
OuterClass.InnerClass innerObject = outerObject.new InnerClass();
```

- 如果内部类是静态的，使用以下语法来创建一个内部类对象：

```
OuterClass.InnerClass innerObject = new OuterClass.InnerClass();
```

一个简单的内部类的用途是将相互依赖的类结合到一个主类中。这样做减少了源文件的数量。这样也使得类文件容易组织，因为它们都将主类名作为前缀。例如，相对于图 15-7a 中创建两个源文件 `Test.java` 和 `A.java`，你可以如图 15-7b 中所示，将类 A 合并到类 Test 中，从而只创建一个源文件 `Test.java`。生成的类文件是 `Test.class` 和 `Test$A.class`。

另外一个内部类的实际用途是避免类名的冲突。在程序清单 15-2 和 15-3 中，定义了两个版本的 `CirclePane`。你可以将它们定义为内部类从而避免冲突。

一个处理器类被设计为针对一个 GUI 组件创建一个处理器对象（比如，一个按钮）。处理器类不会被其他应用所共享，所以将它定义在主类里面作为一个内部类使用是恰如其分的。

☛ 复习题

- 15.7 一个内部类可以被除它所在的嵌套类之外的类所使用吗？
 15.8 修饰符 `public`、`protected`、`private` 以及 `static` 可以用于内部类吗？

15.5 匿名内部类处理器

☛ 要点提示：一个匿名内部类是一个没有名字的内部类。它将一步实现定义一个内部类以及创建一个内部类的实例。

内部类处理器可以使用匿名内部类进行代码简化。程序清单 15-3 中的内部类可以如下所示被一个匿名内部类所替代。

```
public void start(Stage primaryStage) {
    // Omitted

    btEnlarge.setOnAction(
        new EnlargeHandler());
}

class EnlargeHandler
    implements EventHandler<ActionEvent> {
    public void handle(ActionEvent e) {
        circlePane.enlarge();
    }
}
```

a) 一个内部类 `EnlargeListener`

```
public void start(Stage primaryStage) {
    // Omitted

    btEnlarge.setOnAction(
        new class EnlargeHandler
            implements EventHandler<ActionEvent>() {
                public void handle(ActionEvent e) {
                    circlePane.enlarge();
                }
            });
}
```

b) 匿名内部类

匿名内部类的语法如下所示：

```
new SuperClassName/InterfaceName() {
    // Implement or override methods in superclass or interface

    // Other methods if necessary
}
```

由于匿名内部类是一种特殊类型的内部类，它被当作一个内部类对待，同时具有以下特征：

- 一个匿名内部类必须总是从一个父类继承或者实现一个接口，但是它不能有显式的 `extends` 或者 `implements` 子句。
- 一个匿名内部类必须实现父类或者接口中的所有抽象方法。
- 一个匿名内部类总是使用它父类的无参构造方法来创建一个实例。如果一个匿名内部类实现一个接口，构造方法是 `Object()`。
- 一个匿名内部类被编译成一个名为 `OuterClassName$n.class` 的类。例如，如果外部类 `Test` 有两个匿名的内部类，它们将被编译成 `Test$1.class` 和 `Test$2.class`。

程序清单 15-4 给出了一个示例程序，可以处理来自四个按键的事件，如图 15-8 所示。

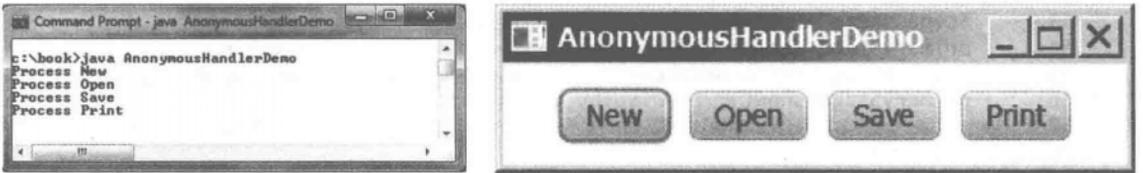


图 15-8 处理来自四个按键的事件的程序

程序清单 15-4 AnonymousHandlerDemo.java

```

1 import javafx.application.Application;
2 import javafx.event.ActionEvent;
3 import javafx.event.EventHandler;
4 import javafx.geometry.Pos;
5 import javafx.scene.Scene;
6 import javafx.scene.control.Button;
7 import javafx.scene.layout.HBox;
8 import javafx.stage.Stage;
9
10 public class AnonymousHandlerDemo extends Application {
11     @Override // Override the start method in the Application class
12     public void start(Stage primaryStage) {
13         // Hold two buttons in an HBox
14         HBox hBox = new HBox();
15         hBox.setSpacing(10);
16         hBox.setAlignment(Pos.CENTER);
17         Button btNew = new Button("New");
18         Button btOpen = new Button("Open");
19         Button btSave = new Button("Save");
20         Button btPrint = new Button("Print");
21         hBox.getChildren().addAll(btNew, btOpen, btSave, btPrint);
22
23         // Create and register the handler
24         btNew.setOnAction(new EventHandler<ActionEvent>() {
25             @Override // Override the handle method
26             public void handle(ActionEvent e) {
27                 System.out.println("Process New");
28             }
29         });
30
31         btOpen.setOnAction(new EventHandler<ActionEvent>() {
32             @Override // Override the handle method
33             public void handle(ActionEvent e) {
34                 System.out.println("Process Open");
35             }
36         });

```

```

36     });
37
38     btSave.setOnAction(new EventHandler<ActionEvent>() {
39         @Override // Override the handle method
40         public void handle(ActionEvent e) {
41             System.out.println("Process Save");
42         }
43     });
44
45     btPrint.setOnAction(new EventHandler<ActionEvent>() {
46         @Override // Override the handle method
47         public void handle(ActionEvent e) {
48             System.out.println("Process Print");
49         }
50     });
51
52     // Create a scene and place it in the stage
53     Scene scene = new Scene(hBox, 300, 50);
54     primaryStage.setTitle("AnonymousHandlerDemo"); // Set title
55     primaryStage.setScene(scene); // Place the scene in the stage
56     primaryStage.show(); // Display the stage
57 }
58 }

```

程序使用匿名内部类创建四个处理器（第 24 ~ 50 行）。如果不使用匿名内部类，你需要创建四个独立的类。一个匿名处理器如同一个内部类一样工作。使用匿名内部类使程序变得精简。

这个例子中的匿名内部类被编译成 `AnonymousHandlerDemo$1.class`、`Anonymous Handler-Demo$2.class`、`AnonymousHandlerDemo$3.class` 和 `AnonymousHandlerDemo$4.class`。

复习题

- 15.9 如果类 A 是类 B 中的一个内部类，A 的类文件名字是什么？如果类 B 包含两个匿名内部类，这两个类的 `.class` 文件名是什么？
- 15.10 下面代码中的错误是什么？

```

public class Test extends Application {
    public void start(Stage stage) {
        Button btOK = new Button("OK");
    }

    private class Handler implements
        EventHandler<ActionEvent> {
        public void handle(ActionEvent e) {
            System.out.println(e.getSource());
        }
    }
}

```

a)

```

public class Test extends Application {
    public void start(Stage stage) {
        Button btOK = new Button("OK");

        btOK.setOnAction(
            new EventHandler<ActionEvent> {
                public void handle
                    (ActionEvent e) {
                        System.out.println
                            (e.getSource());
                    }
            } // Something missing here
        )
    }
}

```

b)

15.6 使用 lambda 表达式简化事件处理

 **要点提示：** lambda 表达式可以用于极大简化事件处理的代码编写。

lambda 表达式是 Java 8 中的新特征。lambda 表达式可以被看作使用精简语法的匿名内部类。例如，下面 a 中的代码可以使用 lambda 表达式极大程度简化成如 b 中代码所示的三行。

```
btEnlarge.setOnAction(
    new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent e) {
            // Code for processing event e
        }
    }
);
```

a) 匿名内部类事件处理器

```
btEnlarge.setOnAction(e -> {
    // Code for processing event e
});
```

b) lambda 表达式事件处理器

一个 lambda 表达式的基础语法是

```
(type1 param1, type2 param2, ...) -> expression
```

或者

```
(type1 param1, type2 param2, ...) -> { statements; }
```

一个参数的数据类型既可以显式声明，也可以由编译器隐式推断。如果只有一个参数，并且没有显式的数据类型，圆括号可以被省略。在前面的例子中，lambda 表达式如下所示：

```
e -> {
    // Code for processing event e
}
```

编译器对待一个 lambda 表达式如同它是从一个匿名内部类创建的对象。这个例子中，编译器将这个对象理解为 `EventHandler<ActionEvent>` 的实例。因为 `EventHandler` 接口定义了一个具有 `ActionEvent` 类型参数的 `handle` 方法，编译器自动识别 `e` 是一个 `ActionEvent` 类型的参数，并且这些语句是 `handle` 方法的方法体。`EventHandler` 接口仅包含一个方法。lambda 表达式中的语句都用于这个方法中。如果它包含多个方法，编译器将无法编译 lambda 表达式。所以，如果要编译器理解 lambda 表达式，接口必须只包含一个抽象的方法。这样的接口称为功能接口 (functional interface) 或者一个单抽象方法 (Single Abstract Method, SAM) 接口。

程序清单 15-4 可以使用 lambda 表达式简化，如程序清单 15-5 所示。

程序清单 15-5 LambdaHandlerDemo.java

```
1 import javafx.application.Application;
2 import javafx.event.ActionEvent;
3 import javafx.geometry.Pos;
4 import javafx.scene.Scene;
5 import javafx.scene.control.Button;
6 import javafx.scene.layout.HBox;
7 import javafx.stage.Stage;
8
9 public class LambdaHandlerDemo extends Application {
10     @Override // Override the start method in the Application class
11     public void start(Stage primaryStage) {
12         // Hold two buttons in an HBox
13         HBox hBox = new HBox();
14         hBox.setSpacing(10);
15         hBox.setAlignment(Pos.CENTER);
16         Button btNew = new Button("New");
17         Button btOpen = new Button("Open");
18         Button btSave = new Button("Save");
19         Button btPrint = new Button("Print");
20         hBox.getChildren().addAll(btNew, btOpen, btSave, btPrint);
```

```

21
22     // Create and register the handler
23     btNew.setOnAction((ActionEvent e) -> {
24         System.out.println("Process New");
25     });
26
27     btOpen.setOnAction((e) -> {
28         System.out.println("Process Open");
29     });
30
31     btSave.setOnAction(e -> {
32         System.out.println("Process Save");
33     });
34
35     btPrint.setOnAction(e -> System.out.println("Process Print"));
36
37     // Create a scene and place it in the stage
38     Scene scene = new Scene(hBox, 300, 50);
39     primaryStage.setTitle("LambdaHandlerDemo"); // Set title
40     primaryStage.setScene(scene); // Place the scene in the stage
41     primaryStage.show(); // Display the stage
42 }
43 }

```

程序使用 lambda 表达式创建 4 个处理器 (第 23 ~ 35 行)。使用 lambda 表达式, 代码变得更短和更清晰。如这个例子中所见, lambda 表达式可以具有多个变种。第 23 行使用一个声明的类型。第 27 行使用一个推断的类型因为类型可以被编译器确定。第 31 行省略了圆括号, 因为只有单个可推断的类型。第 35 行忽略了花括弧, 因为方法体里面只有一个语句。

你可以通过使用内部类、匿名内部类或者 lambda 表达式定义处理器类。我们推荐使用 lambda 表达式, 因为它可以产生更加简短、清晰和整洁的代码。

复习题

- 15.11 什么是 lambda 表达式? 使用 lambda 表达式进行事件处理的好处是什么? 一个 lambda 表达式的语法是什么?
- 15.12 什么是功能接口? 为什么对于一个 lambda 表达式而言, 必须是一个功能接口?
- 15.13 请给出以下代码的输出:

```

public class Test {
    public static void main(String[] args) {
        Test test = new Test();
        test.setAction1(() -> System.out.print("Action 1! "));
        test.setAction2(e -> System.out.print(e + " "));
        System.out.println(test.setAction3(e -> e * 2));
    }

    public void setAction1(T1 t) {
        t.m();
    }

    public void setAction2(T2 t) {
        t.m(4.5);
    }

    public double setAction3(T3 t) {
        return t.m(5.5);
    }
}

interface T1 {

```

```

    public void m();
}

interface T2 {
    public void m(Double d);
}

interface T3 {
    public double m(Double d);
}

```

15.7 示例学习：贷款计算器

要点提示：本例采用事件驱动编程以及 GUI 组件开发一个贷款计算器。

现在，我们来编写本章开始提出的贷款计算器程序。这个程序中有以下关键几步：

- 1) 创建用户界面，如图 15-9 所示。
 - a) 创建一个 GridPane，添加标签、文本域和按钮到面板中。
 - b) 将按钮设置为右侧对齐。
- 2) 处理事件。

创建并注册一个处理器，用于处理按钮单击的动作事件。处理器获得用户输入的贷款额度、利率和年数。计算月支付额和总支付额，并将值显示在文本域中。

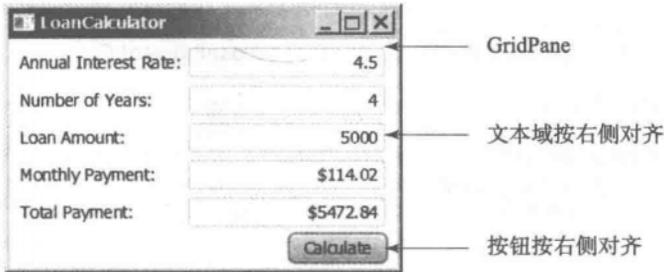


图 15-9 程序计算贷款支付

完整的程序在程序清单 15-6 中给出。

程序清单 15-6 LoanCalculator.java

```

1  import javafx.application.Application;
2  import javafx.geometry.Pos;
3  import javafx.geometry.HPos;
4  import javafx.scene.Scene;
5  import javafx.scene.control.Button;
6  import javafx.scene.control.Label;
7  import javafx.scene.control.TextField;
8  import javafx.scene.layout.GridPane;
9  import javafx.stage.Stage;
10
11 public class LoanCalculator extends Application {
12     private TextField tfAnnualInterestRate = new TextField();
13     private TextField tfNumberOfYears = new TextField();
14     private TextField tfLoanAmount = new TextField();
15     private TextField tfMonthlyPayment = new TextField();
16     private TextField tfTotalPayment = new TextField();
17     private Button btCalculate = new Button("Calculate");
18
19     @Override // Override the start method in the Application class

```

```

20 public void start(Stage primaryStage) {
21     // Create UI
22     GridPane gridPane = new GridPane();
23     gridPane.setHgap(5);
24     gridPane.setVgap(5);
25     gridPane.add(new Label("Annual Interest Rate:"), 0, 0);
26     gridPane.add(tfAnnualInterestRate, 1, 0);
27     gridPane.add(new Label("Number of Years:"), 0, 1);
28     gridPane.add(tfNumberOfYears, 1, 1);
29     gridPane.add(new Label("Loan Amount:"), 0, 2);
30     gridPane.add(tfLoanAmount, 1, 2);
31     gridPane.add(new Label("Monthly Payment:"), 0, 3);
32     gridPane.add(tfMonthlyPayment, 1, 3);
33     gridPane.add(new Label("Total Payment:"), 0, 4);
34     gridPane.add(tfTotalPayment, 1, 4);
35     gridPane.add(btCalculate, 1, 5);
36
37     // Set properties for UI
38     gridPane.setAlignment(Pos.CENTER);
39     tfAnnualInterestRate.setAlignment(Pos.BOTTOM_RIGHT);
40     tfNumberOfYears.setAlignment(Pos.BOTTOM_RIGHT);
41     tfLoanAmount.setAlignment(Pos.BOTTOM_RIGHT);
42     tfMonthlyPayment.setAlignment(Pos.BOTTOM_RIGHT);
43     tfTotalPayment.setAlignment(Pos.BOTTOM_RIGHT);
44     tfMonthlyPayment.setEditable(false);
45     tfTotalPayment.setEditable(false);
46     GridPane.setHalignment(btCalculate, HPos.RIGHT);
47
48     // Process events
49     btCalculate.setOnAction(e -> calculateLoanPayment());
50
51     // Create a scene and place it in the stage
52     Scene scene = new Scene(gridPane, 400, 250);
53     primaryStage.setTitle("Loan Calculator"); // Set title
54     primaryStage.setScene(scene); // Place the scene in the stage
55     primaryStage.show(); // Display the stage
56 }
57
58 private void calculateLoanPayment() {
59     // Get values from text fields
60     double interest =
61         Double.parseDouble(tfAnnualInterestRate.getText());
62     int year = Integer.parseInt(tfNumberOfYears.getText());
63     double loanAmount =
64         Double.parseDouble(tfLoanAmount.getText());
65
66     // Create a loan object. Loan defined in Listing 10.2
67     Loan loan = new Loan(interest, year, loanAmount);
68
69     // Display monthly payment and total payment
70     tfMonthlyPayment.setText(String.format("%.2f",
71         loan.getMonthlyPayment()));
72     tfTotalPayment.setText(String.format("%.2f",
73         loan.getTotalPayment()));
74 }
75 }

```

用户界面在 `start` 方法中创建 (第 22 ~ 46 行)。按钮是事件源。创建一个处理器并和按钮进行注册 (第 49 行)。按钮处理器调用 `calculateLoanPayment()` 方法来得到利率 (第 60 行)、年数 (第 62 行) 以及贷款额度 (第 64 行)。调用 `tfAnnualInterestRate.getText()` 返回 `tfAnnualInterestRate` 文本域中的字符串文本。`Loan` 类用于计算贷款支付。该类在程序清单 10-2 中引入。调用 `loan.getMonthlyPayment()` 返回按月支付额 (第 71 行)。在 10.10.7

节中引入的 `String.format` 方法用来将数字格式化成希望的格式，并将其作为一个字符串返回（第 70、72 行）。在一个文本域上调用 `setText` 方法将一个字符串值设置在文本域中。

15.8 鼠标事件

要点提示： 当一个鼠标按键在一个节点上或者一个场景中被按下、释放、单击、移动或者拖动时，一个 `MouseEvent` 事件被触发。

`MouseEvent` 对象捕捉事件，例如和它相关的单击数、鼠标位置（ x 和 y 坐标），或者哪个鼠标按键被按下，如图 15-10 所示。

javafx.scene.input.MouseEvent	
<code>+getButton(): MouseButton</code>	表明哪个鼠标按钮被单击
<code>+getClickCount(): int</code>	返回该事件中鼠标的单击次数
<code>+getX(): double</code>	返回事件源节点中鼠标点的 x 坐标
<code>+getY(): double</code>	返回事件源节点中鼠标点的 y 坐标
<code>+getSceneX(): double</code>	返回场景中鼠标点的 x 坐标
<code>+getSceneY(): double</code>	返回场景中鼠标点的 y 坐标
<code>+getScreenX(): double</code>	返回屏幕中鼠标点的 x 坐标
<code>+getScreenY(): double</code>	返回屏幕中鼠标点的 y 坐标
<code>+isAltDown(): boolean</code>	如果该事件中 Alt 键被按下，返回 true
<code>+isControlDown(): boolean</code>	如果该事件中 Control 键被按下，返回 true
<code>+isMetaDown(): boolean</code>	如果该事件中鼠标的 Meta 按钮被按下，返回 true
<code>+isShiftDown(): boolean</code>	如果该事件中 Shift 键被按下，返回 true

图 15-10 `MouseEvent` 类封装了鼠标事件的信息

四个常数——`PRIMARY`、`SECONDARY`、`MIDDLE` 和 `NONE` 在 `MouseButton` 中被定义，表示鼠标的左、右、中以及无按钮。可以使用 `getButton()` 方法来探测哪个按钮被按下。例如，`getButton()==MouseButton.SECONDARY` 表示右按钮被按下。

鼠标事件列举在表 15-1 中。为了演示使用鼠标事件，我们给出了一个例子，在一个面板中显示一条消息，并且可以使用鼠标来移动消息。当鼠标拖动时，消息同时移动，并且总是显示在鼠标指针处。程序清单 15-7 给出了该程序。一个程序的运行示例如图 15-11 所示。

程序清单 15-7 `MouseEventDemo.java`

```

1 import javafx.application.Application;
2 import javafx.scene.Scene;
3 import javafx.scene.layout.Pane;
4 import javafx.scene.text.Text;
5 import javafx.stage.Stage;
6
7 public class MouseEventDemo extends Application {
8     @Override // Override the start method in the Application class
9     public void start(Stage primaryStage) {
10         // Create a pane and set its properties
11         Pane pane = new Pane();
12         Text text = new Text(20, 20, "Programming is fun");
13         pane.getChildren().addAll(text);
14         text.setOnMouseDragged(e -> {
15             text.setX(e.getX());
16             text.setY(e.getY());
17         });
18     }

```

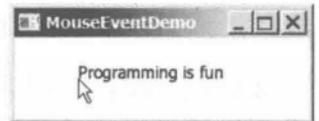


图 15-11 你可以通过拖动鼠标来移动消息

```

19     // Create a scene and place it in the stage
20     Scene scene = new Scene(pane, 300, 100);
21     primaryStage.setTitle("MouseEventDemo"); // Set the stage title
22     primaryStage.setScene(scene); // Place the scene in the stage
23     primaryStage.show(); // Display the stage
24 }
25 }

```

任何节点和场景都可以触发鼠标事件。程序创建了一个 `Text` (第 12 行) 并注册一个处理器, 用于处理鼠标拖动事件 (第 14 行)。任何时候鼠标被拖动, 文本的 x 和 y 坐标被设置到鼠标的位置 (第 15 和 16 行)。

复习题

- 15.14 对于一个鼠标事件而言, 使用什么方法来得到鼠标点的位置?
- 15.15 对于一个鼠标按下、释放、单击、进入、退出、移动和拖动事件, 使用什么方法来注册一个相应的处理器?

15.9 键盘事件

要点提示: 在一个节点或者一个场景上面只要按下、释放或者敲击键盘按键, 就会触发一个 `KeyEvent` 事件。

键盘事件使得可以采用键盘来控制 and 执行动作, 或者从键盘获得输入。`KeyEvent` 对象描述了事件的性质 (即, 一个按键被按下、释放或者敲击) 以及键值, 如图 15-12 所示。

javafx.scene.input.KeyEvent	
+getCharacter(): String	返回该事件中与该键相关的字符
+getCode(): KeyCode	返回该事件中与该键相关的键的编码
+getText(): String	返回一个描述键的编码的字符串
+isAltDown(): boolean	如果该事件中 Alt 键被按下, 返回 true
+isControlDown(): boolean	如果该事件中 Control 键被按下, 返回 true
+isMetaDown(): boolean	如果该事件中鼠标的 Meta 按钮被按下, 返回 true
+isShiftDown(): boolean	如果该事件中 Shift 键被按下, 返回 true

图 15-12 `KeyEvent` 类封装了关于键盘事件的信息

每个键盘事件有一个相关的编码, 可以通过 `KeyEvent` 的 `getCode()` 方法返回。键的编码是定义在 `KeyCode` 中的常量。表 15-2 列出了一些常量。`KeyCode` 是一个 `enum` 类型的变量。关于 `enum` 类型的使用, 参见补充材料 I。对于按下键和释放键的事件, `getCode()` 返回表中的值, `getText()` 返回一个描述键的代码的字符串, `getCharacter()` 返回一个空字符串。对于敲击键的事件, `getCode()` 返回 `UNDEFINED`, `getCharacter()` 返回相应的 Unicode 字符或者和敲击键事件相关的一个字符序列。

表 15-2 `KeyCode` 常量

常量	描述	常量	描述
HOME	The Home key	DOWN	The down-arrow key
END	The End key	LEFT	The left-arrow key
PAGE_UP	The Page Up key	RIGHT	The right-arrow key
PAGE_DOWN	The Page Down key	ESCAPE	The Esc key
UP	The up-arrow key	TAB	The Tab key

(续)

常量	描述	常量	描述
CONTROL	The Control key	ENTER	The Enter key
SHIFT	The Shift key	UNDEFINED	The keyCode unknown
BACK_SPACE	The Backspace key	F1 to F12	The function keys from F1 to F12
CAPS	The Caps Lock key	0 to 9	The number keys from 0 to 9
NUM_LOCK	The Num Lock key	A to Z	The letter keys from A to Z

程序清单 15-8 中的程序显示了一个用户输入的字符。用户可以使用上、下、左、右箭头按键来将字符做相应移动。图 15-13 包含了一个程序的运行示例。

程序清单 15-8 KeyEventDemo.java

```

1 import javafx.application.Application;
2 import javafx.scene.Scene;
3 import javafx.scene.layout.Pane;
4 import javafx.scene.text.Text;
5 import javafx.stage.Stage;
6
7 public class KeyEventDemo extends Application {
8     @Override // Override the start method in the Application class
9     public void start(Stage primaryStage) {
10         // Create a pane and set its properties
11         Pane pane = new Pane();
12         Text text = new Text(20, 20, "A");
13
14         pane.getChildren().add(text);
15         text.setOnKeyPressed(e -> {
16             switch (e.getCode()) {
17                 case DOWN: text.setY(text.getY() + 10); break;
18                 case UP: text.setY(text.getY() - 10); break;
19                 case LEFT: text.setX(text.getX() - 10); break;
20                 case RIGHT: text.setX(text.getX() + 10); break;
21                 default:
22                     if (Character.isLetterOrDigit(e.getText().charAt(0)))
23                         text.setText(e.getText());
24             }
25         });
26
27         // Create a scene and place it in the stage
28         Scene scene = new Scene(pane);
29         primaryStage.setTitle("KeyEventDemo"); // Set the stage title
30         primaryStage.setScene(scene); // Place the scene in the stage
31         primaryStage.show(); // Display the stage
32
33         text.requestFocus(); // text is focused to receive key input
34     }
35 }

```

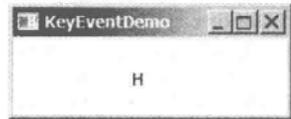


图 15-13 程序通过显示一个字符以及上、下、左、右移动字符来响应键盘事件

程序创建一个面板 (第 11 行), 然后创建一个文本 (第 12 行), 并将文本放置在面板中 (第 14 行)。在第 15 ~ 25 行文本和一个处理器注册以响应按键事件。当一个键被按下, 处理器被调用。程序使用 `e.getCode()` (第 16 行) 来获得键的编码, 使用 `e.getText()` (第 23 行) 来得到该键的字符。当一个非方向键被按下, 该字符被显示 (第 22 和 23 行)。当一个方向键被按下, 字符按照方向键所表示的方向移动 (第 17 ~ 20 行)。请注意, 在一个枚举类型值的 `switch` 语句中, `case` 后面跟的是枚举常量 (第 16 ~ 24 行)。常量是不受限的

(unqualified) 即无须加 `KeyCode` 等类限定。例如，在 `case` 子句中使用 `KeyCode.DOWN` 将出现错误（参见补充材料 I）。

只有一个被聚焦的节点可以接收 `KeyEvent` 事件。在一个 `text` 上调用 `requestFocus()` 使得 `text` 可以接收键盘输入（第 33 行）。这个方法必须在舞台被显示后调用。

现在我们为程序清单 15-3 中的 `ControlCircle` 例子加入更多的控制，比如通过单击鼠标左/右按钮，或者按下 U 和 D 键来增加/减小圆的半径。更新的程序在程序清单 15-9 中给出。`CirclePane` 类（第 12 行）已经在程序清单 15-3 中定义了，可以在本程序中重用。

程序清单 15-9 `ControlCircleWithMouseAndKey.java`

```

1  import javafx.application.Application;
2  import javafx.geometry.Pos;
3  import javafx.scene.Scene;
4  import javafx.scene.control.Button;
5  import javafx.scene.input.KeyCode;
6  import javafx.scene.input.MouseButton;
7  import javafx.scene.layout.HBox;
8  import javafx.scene.layout.BorderPane;
9  import javafx.stage.Stage;
10
11 public class ControlCircleWithMouseAndKey extends Application {
12     private CirclePane circlePane = new CirclePane();
13
14     @Override // Override the start method in the Application class
15     public void start(Stage primaryStage) {
16         // Hold two buttons in an HBox
17         HBox hBox = new HBox();
18         hBox.setSpacing(10);
19         hBox.setAlignment(Pos.CENTER);
20         Button btEnlarge = new Button("Enlarge");
21         Button btShrink = new Button("Shrink");
22         hBox.getChildren().add(btEnlarge);
23         hBox.getChildren().add(btShrink);
24
25         // Create and register the handler
26         btEnlarge.setOnAction(e -> circlePane.enlarge());
27         btShrink.setOnAction(e -> circlePane.shrink());
28
29         circlePane.setOnMouseClicked(e -> {
30             if (e.getButton() == MouseButton.PRIMARY) {
31                 circlePane.enlarge();
32             }
33             else if (e.getButton() == MouseButton.SECONDARY) {
34                 circlePane.shrink();
35             }
36         });
37
38         circlePane.setOnKeyPressed(e -> {
39             if (e.getCode() == KeyCode.U) {
40                 circlePane.enlarge();
41             }
42             else if (e.getCode() == KeyCode.D) {
43                 circlePane.shrink();
44             }
45         });
46
47         BorderPane borderPane = new BorderPane();
48         borderPane.setCenter(circlePane);
49         borderPane.setBottom(hBox);
50         BorderPane.setAlignment(hBox, Pos.CENTER);
51
52         // Create a scene and place it in the stage

```

```

53     Scene scene = new Scene(borderPane, 200, 150);
54     primaryStage.setTitle("ControlCircle"); // Set the stage title
55     primaryStage.setScene(scene); // Place the scene in the stage
56     primaryStage.show(); // Display the stage
57
58     circlePane.requestFocus(); // Request focus on circlePane
59 }
60 }

```

在代码第 29 到 36 行，针对鼠标单击事件的处理器被创建。如果鼠标左键被单击，圆将变大（第 30 ~ 32 行）；如果鼠标右键被单击，圆将缩小（第 33 ~ 35 行）。

一个针对按键事件的处理器在第 38 ~ 45 行被创建。如果 U 键被按下，圆将变大（第 39 ~ 41 行）；如果 D 键被按下，圆将缩小（第 42 ~ 44 行）。

在 `circlePane` 上面调用 `requestFocus()`（第 58 行）将使得 `circlePane` 可以接收键盘事件。请注意，当你单击一个按钮后，`circlePane` 将不再被聚焦。为了修复这个问题，可以在每次按钮被单击后，在 `circlePane` 上再次调用 `requestFocus()`。

复习题

- 15.16 使用什么方法来针对键的按下、释放以及敲击事件注册处理器？这些方法定义在哪些类中？（参见表 15-1。）
- 15.17 使用什么方法来从一个键的敲击事件中获取该键的字符？针对按下键和释放键的事件，使用什么方法来得到键的编码？
- 15.18 如何设置一个节点获取焦点，从而它可以监听键盘事件？

15.10 可观察对象的监听器

 **要点提示：**你可以通过添加一个监听器来处理一个可观察对象中的值的变化。

一个 `Observable` 类的实例被认为是一个可观察对象，它包含了一个 `addListener(InvalidationListener listener)` 方法用于添加监听器。监听器类必须实现 `InvalidationListener` 接口以重写 `invalidated(Observable o)` 方法，从而可以处理值的改变。一旦 `Observable` 中的值改变了，通过调用 `invalidated(Observable o)` 方法，监听器得到通知。每个绑定属性都是 `Observable` 的实例。程序清单 15-10 给出了示例，在一个 `DoubleProperty` 对象 `balance` 中观察和处理改变。

程序清单 15-10 ObservablePropertyDemo.java

```

1  import javafx.beans.InvalidationListener;
2  import javafx.beans.Observable;
3  import javafx.beans.property.DoubleProperty;
4  import javafx.beans.property.SimpleDoubleProperty;
5
6  public class ObservablePropertyDemo {
7      public static void main(String[] args) {
8          DoubleProperty balance = new SimpleDoubleProperty();
9          balance.addListener(new InvalidationListener() {
10             public void invalidated(Observable ov) {
11                 System.out.println("The new value is " +
12                     balance.doubleValue());
13             }
14         });
15
16         balance.set(4.5);
17     }
18 }

```

```
The new value is 4.5
```

当第 16 行被执行的时候，它引发 `balance` 中的一个改变，通过调用监听器的 `invalidated` 方法来通知监听器这一变化。

请注意，第 9 到 14 行的匿名内部类可以通过 `lambda` 表达式简化如下：

```
balance.addListener(ov -> {
    System.out.println("The new value is " +
        balance.doubleValue());
});
```

请回忆程序清单 14-20，当修改窗体大小的时候，时钟面板的大小不会改变。这个问题可以这样修复，添加一个监听器来修改时钟面板的大小，然后将这个监听器和窗体的宽度和高度属性进行注册，如程序清单 15-11 所示。

程序清单 15-11 DisplayResizableClock.java

```
1 import javafx.application.Application;
2 import javafx.geometry.Pos;
3 import javafx.stage.Stage;
4 import javafx.scene.Scene;
5 import javafx.scene.control.Label;
6 import javafx.scene.layout.BorderPane;
7
8 public class DisplayResizableClock extends Application {
9     @Override // Override the start method in the Application class
10    public void start(Stage primaryStage) {
11        // Create a clock and a label
12        ClockPane clock = new ClockPane();
13        String timeString = clock.getHour() + ":" + clock.getMinute()
14            + ":" + clock.getSecond();
15        Label lblCurrentTime = new Label(timeString);
16
17        // Place clock and label in border pane
18        BorderPane pane = new BorderPane();
19        pane.setCenter(clock);
20        pane.setBottom(lblCurrentTime);
21        BorderPane.setAlignment(lblCurrentTime, Pos.TOP_CENTER);
22
23        // Create a scene and place it in the stage
24        Scene scene = new Scene(pane, 250, 250);
25        primaryStage.setTitle("DisplayClock"); // Set the stage title
26        primaryStage.setScene(scene); // Place the scene in the stage
27        primaryStage.show(); // Display the stage
28
29        pane.widthProperty().addListener(ov ->
30            clock.setW(pane.getWidth()));
31    };
32
33    pane.heightProperty().addListener(ov ->
34        clock.setH(pane.getHeight()));
35    };
36 }
37 }
```

这个程序和程序清单 14-20 中是一样的，除开在第 29 行到 35 行添加了代码，为时钟面板注册了监听器，从而在场景的宽度和高度改变的情况下可以重新设置面板大小。代码保证了时钟面板的大小和场景大小是同步的。

复习题

15.19 如果在第 29 行和第 33 行将 `pane` 替换为 `scene` 或者 `primaryStage`, 会出现什么情况?

15.11 动画

要点提示: JavaFX 中的 `Animation` 类为所有的动画制作提供了核心功能。

假设你想写一个程序来实现一个升旗的动画, 如图 15-14 所示。如何完成这个任务呢? 有几种编程方法。一个有效的方法是使用 JavaFX 的 `Animation` 类的子类。这就是本节讨论的主题。



图 15-14 该动画模拟了升旗

抽象类 `Animation` 提供了 JavaFX 中动画制作的核心功能, 如图 15-15 所示。JavaFX 提供了许多 `Animation` 的具体子类。本节介绍 `PathTransition`、`FadeTransition` 和 `Timeline`。

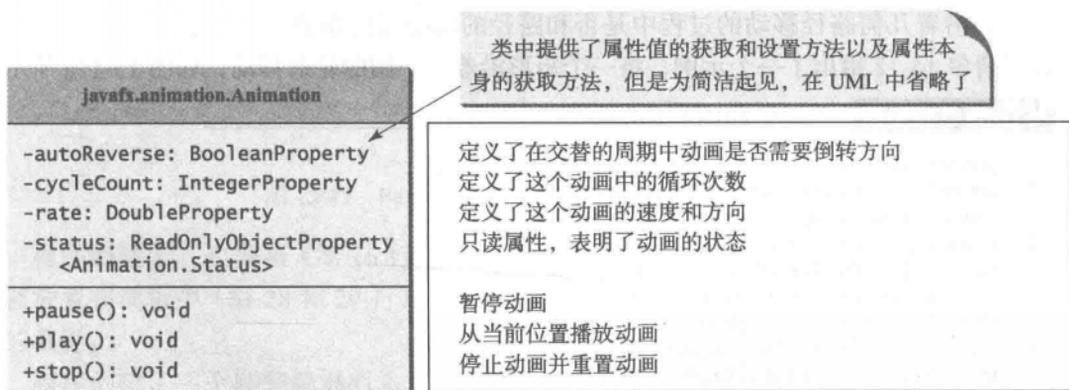


图 15-15 抽象类 `Animation` 是 JavaFX 动画的基类

`autoReverse` 是一个 `Boolean` 属性, 表示下一周期中动画是否要倒转方向。`cycleCount` 表示了该动画的循环次数。你可以使用常量 `Timeline.INDEFINITE` 来表示无限循环。`rate` 定义了动画的速度。一个负的 `rate` 值表示动画的相反方向。`status` 是只读属性, 表明了动画的状态 (`Animation.Status.PAUSED`、`Animation.Status.RUNNING` 和 `Animation.Status.STOPPED`)。方法 `pause()`、`play()` 和 `stop()` 暂停、播放和终止动画。

15.11.1 PathTransition

`PathTransition` 类制作一个在给定时间内, 节点沿着一条路径从一个端点到另外一个端点的移动动画, `PathTransition` 是 `Animation` 的子类型。它的 UML 类图如 15-16 所示。

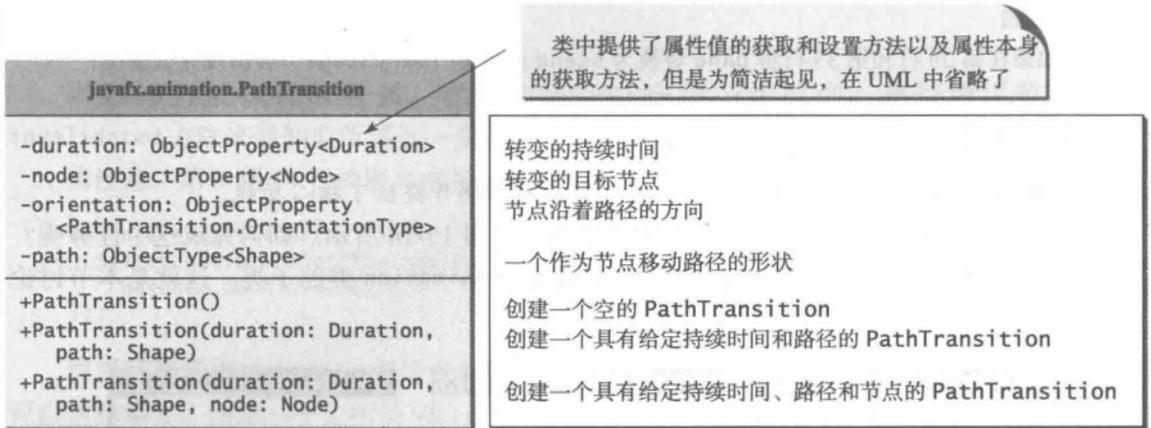


图 15-16 PathTransition 类定义了一个节点沿着一条路径的移动动画

Duration 类定义了持续事件。它是一个不可更改的类。这个类定义了常量 INDEFINITE、ONE、UNKNOWN 和 ZERO 来代表一个无限循环、1 毫秒、未知以及 0 的持续时间。可以使用 new Duration(double millis) 来创建一个 Duration 实例，可以使用 add、subtract、multiply 和 divide 方法来执行算术操作，还可以使用 toHours()、toMinutes()、toSeconds() 和 toMillis() 来返回持续时间值中的小时数、分钟数、秒钟数以及毫秒数。还可以使用 compareTo 来比较两个持续时间。

常量 NONE 和 ORTHOGONAL_TO_TANGET 在 PathTransition.OrientationType 中定义。后者确定节点在沿着几何路径移动的过程中是否和路径的切线保持垂直。

程序清单 15-12 给出了一个示例，将一个矩形沿着一个圆的轮廓移动，如图 15-17a 所示。

程序清单 15-12 PathTransitionDemo.java

```

1  import javafx.animation.PathTransition;
2  import javafx.animation.Timeline;
3  import javafx.application.Application;
4  import javafx.scene.Scene;
5  import javafx.scene.layout.Pane;
6  import javafx.scene.paint.Color;
7  import javafx.scene.shape.Rectangle;
8  import javafx.scene.shape.Circle;
9  import javafx.stage.Stage;
10 import javafx.util.Duration;
11
12 public class PathTransitionDemo extends Application {
13     @Override // Override the start method in the Application class
14     public void start(Stage primaryStage) {
15         // Create a pane
16         Pane pane = new Pane();
17
18         // Create a rectangle
19         Rectangle rectangle = new Rectangle (0, 0, 25, 50);
20         rectangle.setFill(Color.ORANGE);
21
22         // Create a circle
23         Circle circle = new Circle(125, 100, 50);
24         circle.setFill(Color.WHITE);
25         circle.setStroke(Color.BLACK);
26
27         // Add circle and rectangle to the pane
28         pane.getChildren().add(circle);

```

```

29     pane.getChildren().add(rectangle);
30
31     // Create a path transition
32     PathTransition pt = new PathTransition();
33     pt.setDuration(Duration.millis(4000));
34     pt.setPath(circle);
35     pt.setNode(rectangle);
36     pt.setOrientation(
37         PathTransition.OrientationType.ORTHOGONAL_TO_TANGENT);
38     pt.setCycleCount(Timeline.INDEFINITE);
39     pt.setAutoReverse(true);
40     pt.play(); // Start animation
41
42     circle.setOnMousePressed(e -> pt.pause());
43     circle.setOnMouseReleased(e -> pt.play());
44
45     // Create a scene and place it in the stage
46     Scene scene = new Scene(pane, 250, 200);
47     primaryStage.setTitle("PathTransitionDemo"); // Set the stage title
48     primaryStage.setScene(scene); // Place the scene in the stage
49     primaryStage.show(); // Display the stage
50 }
51 }

```

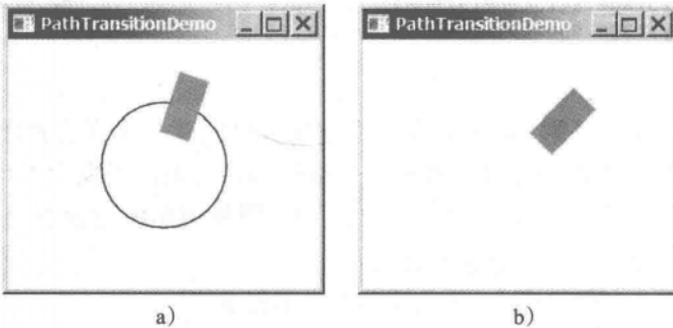


图 15-17 PathTransition 使一个矩形沿着圆移动

程序创建了一个面板（第 16 行）、一个矩形（第 19 行）以及一个圆（第 23 行）。圆和矩形被放置在面板中（第 28 和 29 行）。如果该圆没有放置在面板中，你将看到如图 15-17b 所示的截屏。

程序创建了一个路径移动对象（第 32 行），设置它每个动画周期的持续时间为 4 秒（第 33 行），将圆设置为路径（第 34 行），将矩形设置为节点（第 35 行），并设置方向为垂直于切线（第 36 行）。

循环次数设为无限多次（第 38 行），从而动画将一直持续。自动倒转设置为真（第 39 行），所以每个交替周期中运动方向会倒转。程序通过调用 `play()` 方法启动动画（第 40 行）。

如果第 42 行的 `pause()` 方法被 `stop()` 方法替代，动画将在重新开始的时候从最开始状态启动。

程序清单 15-13 给出了一个升旗的动画的程序，如图 15-14 所示。

程序清单 15-13 FlagRisingAnimation.java

```

1 import javafx.animation.PathTransition;
2 import javafx.application.Application;
3 import javafx.scene.Scene;
4 import javafx.scene.image.ImageView;

```

```

5 import javafx.scene.layout.Pane;
6 import javafx.scene.shape.Line;
7 import javafx.stage.Stage;
8 import javafx.util.Duration;
9
10 public class FlagRisingAnimation extends Application {
11     @Override // Override the start method in the Application class
12     public void start(Stage primaryStage) {
13         // Create a pane
14         Pane pane = new Pane();
15
16         // Add an image view and add it to pane
17         ImageView imageView = new ImageView("image/us.gif");
18         pane.getChildren().add(imageView);
19
20         // Create a path transition
21         PathTransition pt = new PathTransition(Duration.millis(10000),
22             new Line(100, 200, 100, 0), imageView);
23         pt.setCycleCount(5);
24         pt.play(); // Start animation
25
26         // Create a scene and place it in the stage
27         Scene scene = new Scene(pane, 250, 200);
28         primaryStage.setTitle("FlagRisingAnimation"); // Set the stage title
29         primaryStage.setScene(scene); // Place the scene in the stage
30         primaryStage.show(); // Display the stage
31     }
32 }

```

程序创建了一个面板（第 14 行），从一个图像文件创建一个图像视图（第 17 行），并将图像视图放置在面板中（第 18 行）。创建一个路径移动对象，周期为 10 秒，使用一条直线作为路径，图像视图作为节点（第 21 行和 22 行）。图像视图将沿着直线移动。由于直线没有放置在场景中，你不会在窗体中看到直线。

循环数被设置为 5（第 23 行），因此该动画将重复 5 次。

15.11.2 FadeTransition

FadeTransition 类在一个给定的时间内，通过改变一个节点的透明度来产生动画。FadeTransition 是 Animation 的子类型。它的 UML 类图如 15-18 所示。

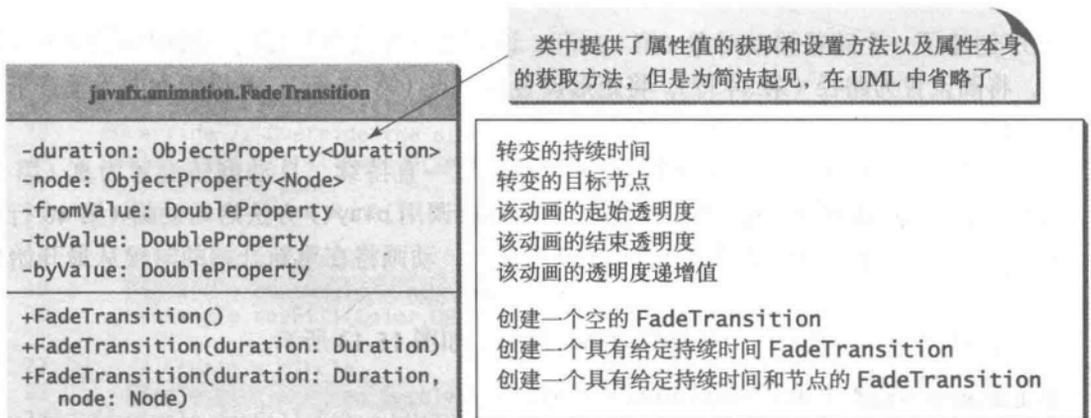


图 15-18 FadeTransition 类定义了一个节点透明度变化的动画

程序清单 15-14 给出了一个示例，将一个褪色变化应用在一个椭圆的填充颜色中，如

图 15-19 所示。

程序清单 15-14 FadeTransitionDemo.java

```
1 import javafx.animation.FadeTransition;
2 import javafx.animation.Timeline;
3 import javafx.application.Application;
4 import javafx.scene.Scene;
5 import javafx.scene.layout.Pane;
6 import javafx.scene.paint.Color;
7 import javafx.scene.shape.Ellipse;
8 import javafx.stage.Stage;
9 import javafx.util.Duration;
10
11 public class FadeTransitionDemo extends Application {
12     @Override // Override the start method in the Application class
13     public void start(Stage primaryStage) {
14         // Place an ellipse to the pane
15         Pane pane = new Pane();
16         Ellipse ellipse = new Ellipse(10, 10, 100, 50);
17         ellipse.setFill(Color.RED);
18         ellipse.setStroke(Color.BLACK);
19         ellipse.centerXProperty().bind(pane.widthProperty().divide(2));
20         ellipse.centerYProperty().bind(pane.heightProperty().divide(2));
21         ellipse.radiusXProperty().bind(
22             pane.widthProperty().multiply(0.4));
23         ellipse.radiusYProperty().bind(
24             pane.heightProperty().multiply(0.4));
25         pane.getChildren().add(ellipse);
26
27         // Apply a fade transition to ellipse
28         FadeTransition ft =
29             new FadeTransition(Duration.millis(3000), ellipse);
30         ft.setFromValue(1.0);
31         ft.setToValue(0.1);
32         ft.setCycleCount(Timeline.INDEFINITE);
33         ft.setAutoReverse(true);
34         ft.play(); // Start animation
35
36         // Control animation
37         ellipse.setOnMousePressed(e -> ft.pause());
38         ellipse.setOnMouseReleased(e -> ft.play());
39
40         // Create a scene and place it in the stage
41         Scene scene = new Scene(pane, 200, 150);
42         primaryStage.setTitle("FadeTransitionDemo"); // Set the stage title
43         primaryStage.setScene(scene); // Place the scene in the stage
44         primaryStage.show(); // Display the stage
45     }
46 }
```

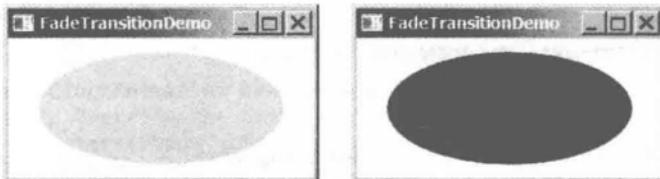


图 15-19 FadeTransition 生成一个椭圆内的透明度变化的动画

程序创建一个面板 (第 15 行) 以及一个椭圆 (第 16 行), 并将椭圆放置在面板中 (第 25 行)。椭圆的 `centerX`、`centerY`、`radiusX` 和 `radiusY` 属性绑定到面板的大小上 (第 19 ~ 24 行)。

针对椭圆创建一个持续时间为 3 秒的褪色转换对象 (第 29 行)。它将开始的透明度设置为 1.0 (第 30 行), 结束透明度设为 0.1 (第 31 行)。循环数设置为无限, 因此动画将无限次数的重复 (第 32 行)。当单击鼠标时, 动画暂停 (第 37 行), 当鼠标释放的时候, 动画从暂停的地方继续 (第 38 行)。

15.11.3 Timeline

PathTransition 和 FadeTransition 定义了特定的动画。Timeline 类可以用于通过使用一个或者更多的 KeyFrame (关键帧) 来编写任意动画。每个 KeyFrame 在一个给定的时间间隔内顺序执行。Timeline 继承自 Animation。你可以通过构造方法 new Timeline(KeyFrame...keyframe) 来构建一个 Timeline。一个 KeyFrame 可以使用以下语句来构建:

```
new KeyFrame(Duration duration, EventHandler<ActionEvent> onFinish)
```

处理器 onFinish 方法当这个关键帧的持续时间结束后被调用。

程序清单 15-15 给出了一个示例, 显示一个闪烁的文本, 如图 15-20 所示。文本交替的显示和消失来产生闪烁动画效果。

程序清单 15-15 TimelineDemo.java

```

1  import javafx.animation.Animation;
2  import javafx.application.Application;
3  import javafx.stage.Stage;
4  import javafx.animation.KeyFrame;
5  import javafx.animation.Timeline;
6  import javafx.event.ActionEvent;
7  import javafx.event.EventHandler;
8  import javafx.scene.Scene;
9  import javafx.scene.layout.StackPane;
10 import javafx.scene.paint.Color;
11 import javafx.scene.text.Text;
12 import javafx.util.Duration;
13
14 public class TimelineDemo extends Application {
15     @Override // Override the start method in the Application class
16     public void start(Stage primaryStage) {
17         StackPane pane = new StackPane();
18         Text text = new Text(20, 50, "Programming is fun");
19         text.setFill(Color.RED);
20         pane.getChildren().add(text); // Place text into the stack pane
21
22         // Create a handler for changing text
23         EventHandler<ActionEvent> eventHandler = e -> {
24             if (text.getText().length() != 0) {
25                 text.setText("");
26             }
27             else {
28                 text.setText("Programming is fun");
29             }
30         };
31
32         // Create an animation for alternating text
33         Timeline animation = new Timeline(
34             new KeyFrame(Duration.millis(500), eventHandler));
35         animation.setCycleCount(Timeline.INDEFINITE);
36         animation.play(); // Start animation
37
38         // Pause and resume animation
39         text.setOnMouseClicked(e -> {

```

```

40     if (animation.getStatus() == Animation.Status.PAUSED) {
41         animation.play();
42     }
43     else {
44         animation.pause();
45     }
46 });
47
48 // Create a scene and place it in the stage
49 Scene scene = new Scene(pane, 250, 250);
50 primaryStage.setTitle("TimelineDemo"); // Set the stage title
51 primaryStage.setScene(scene); // Place the scene in the stage
52 primaryStage.show(); // Display the stage
53 }
54 }

```

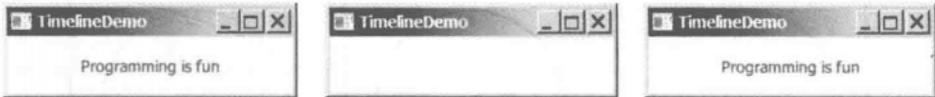


图 15-20 调用处理器方法以交替将文本设置为“Programming is fun”或者空文本

程序创建一个堆栈面板（第 17 行）和一个文本（第 18 行），并将文本放置在面板中（第 20 行）。一个处理器被创建，如果文本非空，则将本文设置为空字符串（第 24 ~ 26 行），如果文本为空，则设置为“Programming is fun”（第 27 ~ 29 行）。一个 KeyFrame 被创建用于每半秒钟运行一个动作事件（第 34 行）。一个 Timeline 动画被创建以获得一个关键帧（第 33 和 34 行）。动画被设置为无限运行（第 35 行）。

程序为文本设置鼠标单击事件（第 39 ~ 46 行）。如果动画暂停了，鼠标在文本上单击一次会继续动画（第 40 ~ 42 行）；如果动画正在执行，那么在文本上的一次鼠标单击将暂停动画（第 43 ~ 45 行）。

在 14.12 节的示例学习的 ClockPane 类中，你绘制了一个时钟用于显示当前事件。显示时钟后并不会走动。如何让钟表每一秒显示一次最新的当前时间呢？使时钟走动的关键是每一秒让它绘制当前的最新时间。你可以使用一个 Timeline 来控制时钟的重绘，代码列在程序清单 15-16 中。程序的一个运行示例显示在图 15-21 中。

程序清单 15-16 ClockAnimation.java

```

1  import javafx.application.Application;
2  import javafx.stage.Stage;
3  import javafx.animation.KeyFrame;
4  import javafx.animation.Timeline;
5  import javafx.event.ActionEvent;
6  import javafx.event.EventHandler;
7  import javafx.scene.Scene;
8  import javafx.util.Duration;
9
10 public class ClockAnimation extends Application {
11     @Override // Override the start method in the Application class
12     public void start(Stage primaryStage) {
13         ClockPane clock = new ClockPane(); // Create a clock
14
15         // Create a handler for animation
16         EventHandler<ActionEvent> eventHandler = e -> {
17             clock.setCurrentTime(); // Set a new clock time
18         };
19

```

```

20 // Create an animation for a running clock
21 Timeline animation = new Timeline(
22     new KeyFrame(Duration.millis(1000), eventHandler));
23 animation.setCycleCount(Timeline.INDEFINITE);
24 animation.play(); // Start animation
25
26 // Create a scene and place it in the stage
27 Scene scene = new Scene(clock, 250, 50);
28 primaryStage.setTitle("ClockAnimation"); // Set the stage title
29 primaryStage.setScene(scene); // Place the scene in the stage
30 primaryStage.show(); // Display the stage
31 }
32 }

```



图 15-21 在窗体中显示一个活动的钟表

程序创建一个 `ClockPane` 的实例 `clock` 用于显示一个时钟 (第 13 行)。`ClockPane` 类在程序清单 14-21 中定义。在第 27 行中时钟被放置在场景中。一个事件处理器被创建用于在时钟中设置当前事件 (第 16 ~ 18 行)。在时间线动画的每个关键帧中, 这个处理器每秒被调用一次 (第 21 ~ 24 行)。所以动画中时钟的时间每秒被更新一次。

复习题

- 15.20 如何将一个动画的循环次数设置为无限次? 如何自动倒转一个动画? 如何开始、暂停以及停止一个动画?
- 15.21 `PathTransition`、`FadeTransition` 和 `Timeline` 是 `Animation` 的一个子类型吗?
- 15.22 如何创建一个 `PathTransition`? 如何创建一个 `FadeTransition`? 如何创建一个 `Timeline`?
- 15.23 如何创建一个关键帧?

15.12 示例学习: 弹球

要点提示: 本节介绍一个动画, 显示一个球在面板中弹动。

程序使用 `Timeline` 来实现弹球的动画, 如图 15-22 所示。

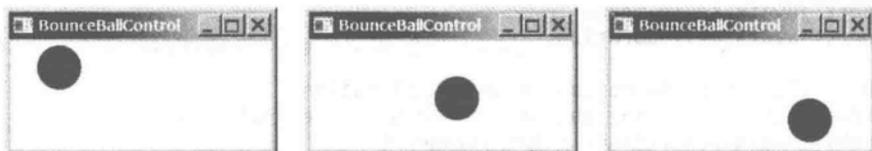


图 15-22 一个球在窗体中弹动

下面是编写这个程序的关键步骤:

- 1) 定义一个名为 `BallPane` 的 `Pane` 类的子类, 用于显示一个弹动的球, 如程序清单 15-17 所示。

2) 定义一个名为 `BounceBallControl` 的 `Application` 的子类, 用来使用鼠标动作控制弹球, 如程序清单 15-18 所示。当鼠标按下的时候动画暂停, 当鼠标释放的时候动画恢复执行。按下 UP / DOWN 方向键可以增加 / 减少动画的速度。

类之间的关系如图 15-23 所示。

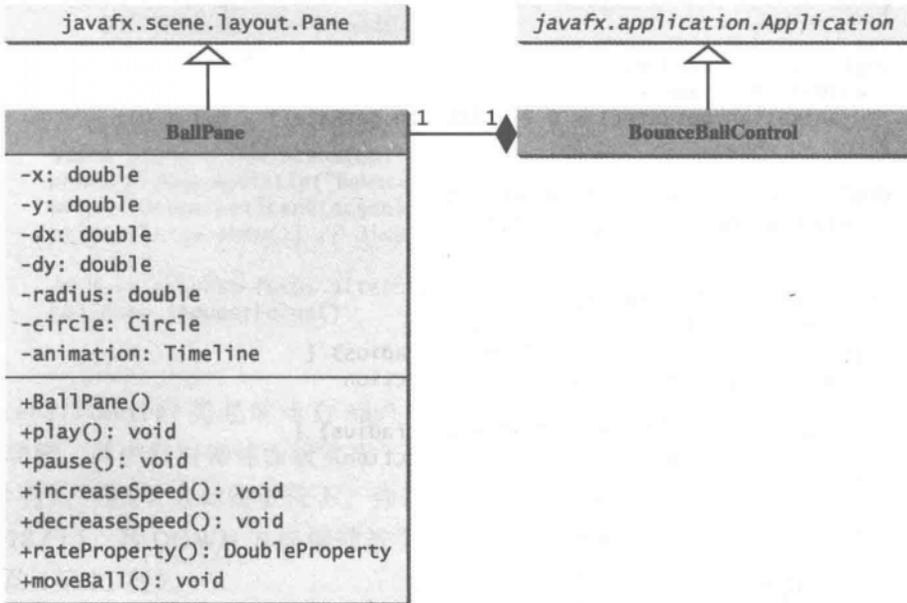


图 15-23 `BounceBallControl` 包含 `BallPane`

程序清单 15-17 `BallPane.java`

```

1 import javafx.animation.KeyFrame;
2 import javafx.animation.Timeline;
3 import javafx.beans.property.DoubleProperty;
4 import javafx.scene.layout.Pane;
5 import javafx.scene.paint.Color;
6 import javafx.scene.shape.Circle;
7 import javafx.util.Duration;
8
9 public class BallPane extends Pane {
10     public final double radius = 20;
11     private double x = radius, y = radius;
12     private double dx = 1, dy = 1;
13     private Circle circle = new Circle(x, y, radius);
14     private Timeline animation;
15
16     public BallPane() {
17         circle.setFill(Color.GREEN); // Set ball color
18         getChildren().add(circle); // Place a ball into this pane
19
20         // Create an animation for moving the ball
21         animation = new Timeline(
22             new KeyFrame(Duration.millis(50), e -> moveBall()));
23         animation.setCycleCount(Timeline.INDEFINITE);
24         animation.play(); // Start animation
25     }
26
27     public void play() {
28         animation.play();
29     }
  
```

```

30
31 public void pause() {
32     animation.pause();
33 }
34
35 public void increaseSpeed() {
36     animation.setRate(animation.getRate() + 0.1);
37 }
38
39 public void decreaseSpeed() {
40     animation.setRate(
41         animation.getRate() > 0 ? animation.getRate() - 0.1 : 0);
42 }
43
44 public DoubleProperty rateProperty() {
45     return animation.rateProperty();
46 }
47
48 protected void moveBall() {
49     // Check boundaries
50     if (x < radius || x > getWidth() - radius) {
51         dx *= -1; // Change ball move direction
52     }
53     if (y < radius || y > getHeight() - radius) {
54         dy *= -1; // Change ball move direction
55     }
56
57     // Adjust ball position
58     x += dx;
59     y += dy;
60     circle.setCenterX(x);
61     circle.setCenterY(y);
62 }
63 }

```

BallPane 继承自 Pane 用来显示一个移动的球 (第 9 行)。一个 Timeline 的实例被创建用于控制动画 (第 21 和 22 行)。该实例包含一个 KeyFrame 对象, 在一个固定的速率上调用 moveBall() 方法。moveBall() 方法移动球以模拟动画。球的中心位于 (x,y), 下一个移动中改变成 (x+dx,y+dy) (第 58 ~ 61 行)。当球超出水平边界时, dx 的符号发生改变 (从正改变为负, 或者相反) (第 50 ~ 52 行)。这使得球改变它水平移动的方向。当球超出垂直边界时, dy 的符号发生改变 (从正改变为负, 或者相反) (第 53 ~ 55 行)。这使得球改变它垂直移动的方向。pause 和 play 方法 (第 27 ~ 33 行) 可以用于暂停和恢复动画。increaseSpeed() 和 decreaseSpeed() 方法 (第 35 ~ 42 行) 用于增加和减小动画速度。rateProperty() 方法 (第 44 ~ 46 行) 返回一个速率的绑定属性。该绑定属性对下一章在应用中绑定速率很有用处。

程序清单 15-18 BounceBallControl.java

```

1 import javafx.application.Application;
2 import javafx.stage.Stage;
3 import javafx.scene.Scene;
4 import javafx.scene.input.KeyCode;
5
6 public class BounceBallControl extends Application {
7     @Override // Override the start method in the Application class
8     public void start(Stage primaryStage) {
9         BallPane ballPane = new BallPane(); // Create a ball pane
10
11         // Pause and resume animation
12         ballPane.setOnMousePressed(e -> ballPane.pause());
13         ballPane.setOnMouseReleased(e -> ballPane.play());

```

```

14
15 // Increase and decrease animation
16 ballPane.setOnKeyPressed(e -> {
17     if (e.getCode() == KeyCode.UP) {
18         ballPane.increaseSpeed();
19     }
20     else if (e.getCode() == KeyCode.DOWN) {
21         ballPane.decreaseSpeed();
22     }
23 });
24
25 // Create a scene and place it in the stage
26 Scene scene = new Scene(ballPane, 250, 150);
27 primaryStage.setTitle("BounceBallControl"); // Set the stage title
28 primaryStage.setScene(scene); // Place the scene in the stage
29 primaryStage.show(); // Display the stage
30
31 // Must request focus after the primary stage is displayed
32 ballPane.requestFocus();
33 }
34 }

```

BounceBallControl 类是继承自 Application 的 JavaFX 主类，用于显示弹球的面板并具有控制功能。其中针对弹球面板实现了鼠标按下和鼠标释放的处理器，以暂停和恢复动画（第 12、13 行）。当 UP 方向键被按下，弹球面板的 increaseSpeed() 方法被调用以增加球的移动（第 18 行）。当 DOWN 方向键被按下，弹球面板的 decreaseSpeed() 方法被调用以减少球的移动（第 21 行）。

第 32 行中调用 ballPane.requestFocus() 将输入焦点设置到 ballPane 上。

复习题

- 15.24 程序是如何使球移动的？
- 15.25 程序清单 15-17 中的代码是如何改变球的移动方向的？
- 15.26 当在弹球面板上按下鼠标时，程序将做什么？当在弹球面板上释放鼠标时，程序将做什么？
- 15.27 在程序清单 15-18 中，如果第 32 行不存在，当你按下 UP 或者 DOWN 方向键的时候，会出现什么情况？
- 15.28 如果程序清单 15-17 中没有第 23 行，会出现什么情况？

关键术语

anonymous inner class (匿名内部类)	functional interface (功能接口)
event (事件)	lambda expression (lambda 表达式)
event-driven programming (事件驱动编程)	inner class (内部类)
event handler (事件处理器)	key code (键的编码)
event-handler interface (事件处理器接口)	observable object (可观察对象)
event object (事件对象)	single abstract method interface (单抽象方法接口)
event source object (事件源对象)	

本章小结

1. JavaFX 事件类的基类是 javafx.event.Event，它是 java.util.EventObject 的子类。Event 的子类处理特殊类型的事件、比如动作事件、窗体事件、鼠标事件以及键盘事件。如果一个节点可以触发一个事件，该节点的任何一个子类都可以触发同类事件。

2. 处理器对象的类必须实现相应的事件处理器接口。JavaFX 为每个事件类 T 提供了一个处理器接口 `EventHandler<T extends Event>`。处理器接口包含 `handle(T e)` 方法用于对事件 e 进行处理。
3. 处理器对象必须通过源对象进行注册。注册的方法取决于事件类型。对于一个动作事件而言，方法是 `setOnAction`。对于一个鼠标按下事件，方法是 `setOnMousePressed`。对于一个按键事件，方法是 `setOnKeyPressed`。
4. 一个内部类，或者称为嵌套类，是定义在另外一个类中的类。一个内部类可以引用它所在的外部类中的数据和类，所以你无须传递外部类的引用到内部类的构造方法中。
5. 一个匿名内部类可以用于减少事件处理的代码。更进一步的，对于功能接口处理器而言，使用 lambda 表达式可以用于极大的简化事件处理代码。
6. 功能接口是指一个只包含一个抽象方法的接口，也被称为单抽象方法 (SAM) 接口。
7. 当在一个节点或者场景上按下、释放、单击、移动、拖动鼠标的时候，一个 `MouseEvent` 事件被触发。`getButton()` 方法可以用于探测这个事件中哪个鼠标按钮被按下。
8. 当在一个节点或者场景上按下、释放或者敲击键盘上的一个按键时，一个 `KeyEvent` 事件被触发。`getCode()` 方法可以用于返回该键的编码值。
9. 一个 `Observable` 的实例称为一个可观察对象，它包含了一个 `addListener(InvalidationListener listener)` 方法用于增加一个监听器。一旦属性中的值被改变，监听器收到通知。监听器类应该实现 `InvalidationListener` 接口，使用其中的 `invalidated` 方法来处理属性值的改变。
10. 抽象类 `Animation` 提供了 JavaFX 中动画制作的核心功能。`PathTransition`、`FadeTransition` 和 `Timeline` 是用于实现动画的特定类。

测试题

回答本章位于 www.cs.armstrong.edu/liang/intro10e/quiz.html 的测试题。

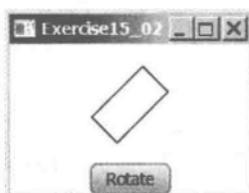
编程练习题

15.2 ~ 15.7 节

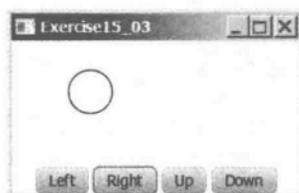
- *15.1 (选取 4 张卡牌) 请写一个程序，可以让用户通过单击 `Refresh` 按钮以显示从一副 52 张卡牌选取的 4 张卡牌，如图 15-24a 所示。(参见编程练习题 14.3 中关于如何获得 4 张随机卡牌中的提示)。



a) 练习题 15.1 显示四张随机卡牌



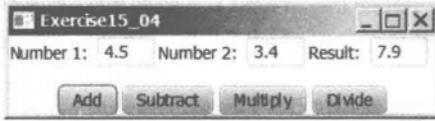
b) 练习题 15.2 旋转四边形



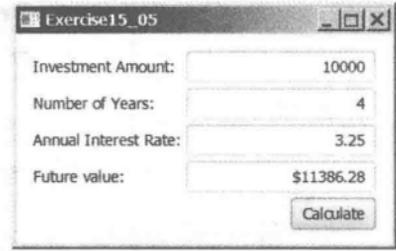
c) 练习题 15.3 使用按钮来移动球

图 15-24

- 15.2 (旋转一个四边形) 请写一个程序，在 `Rotate` 按钮被单击时，将一个四边形向右旋转 15 度，如图 15-24b 所示。
- *15.3 (移动小球) 编写一个程序，在面板上移动小球。应该定义一个面板类来显示小球，并提供向左、向右、向上和向下移动小球的方法，如图 15-24c 所示。请进行边界检查以防止球完全移到视线之外。
- *15.4 (创建一个简单的计算器) 编写一个程序完成加法、减法、乘法和除法操作，参见图 15-25a。



a) 练习题 15.4 完成 double 数值的加法、减法、乘法和除法



b) 用户输入投资总额、年数和利率来计算未来值

图 15-25

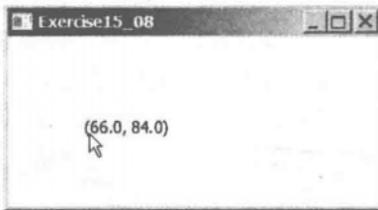
- *15.5 (创建一个投资值计算器) 编写一个程序, 计算投资值在给定利率以及给定年数下的未来值。计算的公式如下所示:

$$\text{未来值} = \text{投资值} \times (1 + \text{月利率})^{\text{年数} \times 12}$$

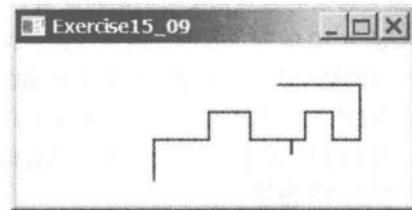
使用文本域显示利率、投资值和年数。当用户单击 Calculate 按钮时在文本域显示未来值, 如图 15-25b 所示。

15.8 ~ 15.9 节

- **15.6 (两个消息交替出现) 编写一个程序, 当单击鼠标时, 面板上交替显示两个文本 “Java is fun” 和 “Java is powerful”。
- *15.7 (使用鼠标改变颜色) 编写一个程序, 显示一个圆的颜色, 当按下鼠标键时颜色为黑色, 释放鼠标时颜色为白色。
- *15.8 (显示鼠标的位置) 编写两个程序, 一个当单击鼠标时显示鼠标的位置 (参见图 15-26a), 而另一个当按下鼠标时显示鼠标的位置, 当释放鼠标时停止显示。



a) 练习题 15.8 显示鼠标的位置



b) 练习题 15.9 使用箭头键绘制直线

图 15-26

- *15.9 (使用箭头键画线) 请编写一个程序, 使用箭头键绘制线段。所画的线从面板的中心开始, 当敲击向右、向上、向左或向下的箭头键时, 相应地向东、向北、向西或向南方向画线, 如图 15-26b 所示。
- **15.10 (输入并显示字符串) 请编写一个程序, 从键盘接收一个字符串并把它显示在面板上。回车键表明字符串结束。任何时候输入一个新字符串时都会将它显示在面板上。
- *15.11 (使用键移动圆) 请编写程序, 可以使用箭头键向上、向下、向左、向右移动一个圆。
- **15.12 (几何问题: 是否在圆内?) 请编写一个程序, 绘制一个圆心在 (100, 60) 而半径为 50 的固定的圆。当鼠标移动时, 显示一条消息表示鼠标点是在圆内还是在圆外, 如图 15-27a 所示。
- **15.13 (几何问题: 是否在矩形内?) 请编写一个程序, 绘制一个中心在 (100, 60) 宽为 100 而高为 40 的固定的矩形。当鼠标移动时, 显示一条消息表示鼠标指针是否在矩形内, 如图 15-27b 所示。为了检查一个点是否在矩形内, 使用定义在 Node 类中的 contains 方法。

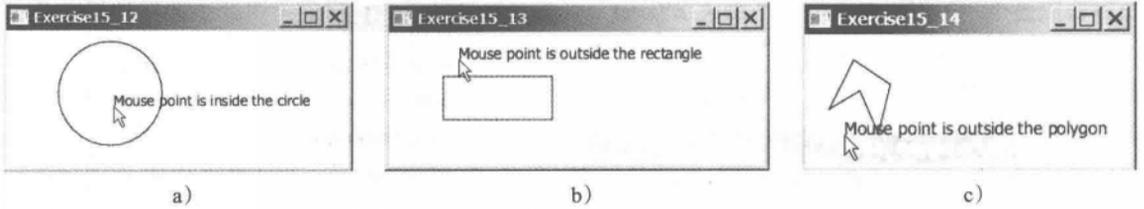
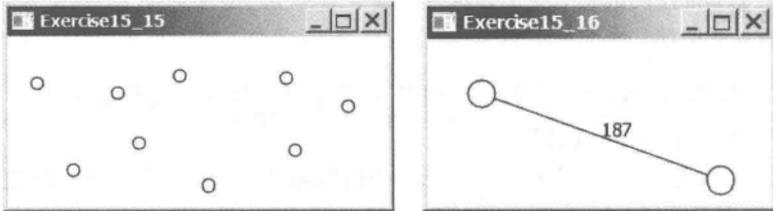


图 15-27 检查一个点是否在圆内，是否在矩形内，是否在多边形内

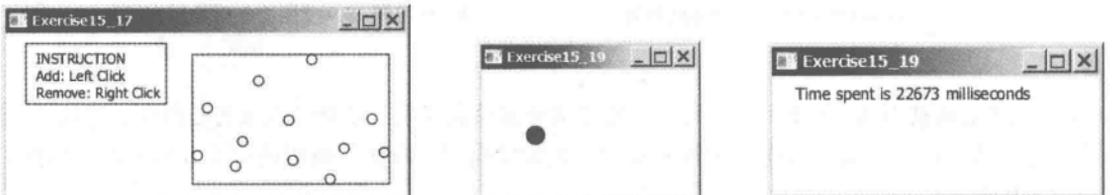
- **15.14** (几何问题: 是否在多边形内?) 请编写一个程序, 绘制端点分别是 (40, 20)、(70, 40)、(60, 80)、(45, 45) 和 (20, 60) 的固定多边形。当鼠标移动时, 显示一条消息表示鼠标点是否在多边形内, 如图 15-27c 所示。为了检查一个点是否在多边形内, 可以使用定义在 `Node` 类中的 `contains` 方法。
- **15.15** (几何问题: 添加或删除点) 请编写一个程序, 让用户在面板上单击以自动创建或移去点 (参见 15-28a)。当用户左击鼠标时 (主按钮), 就创建一个点并且显示在鼠标的位置, 用户还可以将鼠标移到一个点上, 然后右击鼠标 (次按钮) 以移去这个点。



a) 练习题 15.15 允许用户动态地创建 / 移去点 b) 练习题 15.16 显示两个顶点和一条连接的边

图 15-28

- *15.16** (两个可移动的顶点以及它们间的距离) 请编写一个程序, 显示两个分别位于 (40,40) 和 (120,150) 的半径为 10 的圆, 并用一条直线连接两个圆, 如图 15-28b 所示。圆之间的距离显示在直线上。用户可以拖动圆, 圆和它上面的直线会相应移动, 并且两个圆之间的距离会更新。
- **15.17** (几何问题: 寻找边界矩形) 请编写一个程序, 让用户可以在一个二维面板上动态地增加和移除点, 如图 15-29a 所示。当点加入和移除的时候, 一个最小的边界矩形更新显示。假设每个点的半径是 10 像素。



a) 练习题 15.17 允许用户动态增加和移除点, 并显示边界矩形

b) 当单击一个圆时, 一个新的圆显示在一个随机位置上

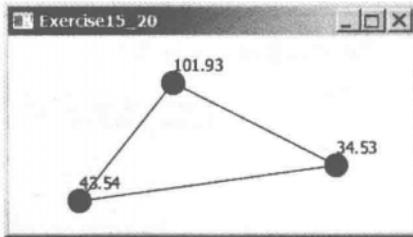
c) 当单击了 20 个圆后, 在面板上显示所用的时间

图 15-29

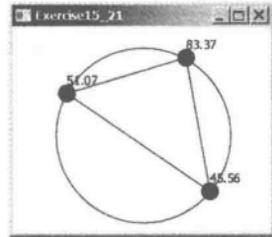
- **15.18** (使用鼠标来移动一个矩形) 请编写一个程序显示一个矩形。可以使用鼠标单击矩形内部并且拖动 (即按住鼠标移动) 矩形到鼠标的位置。鼠标点成为矩形的中央。
- **15.19** (游戏: 手眼协调) 请编写一个程序, 显示一个半径为 10 像素的实心圆, 该圆放置在面板上的

随机位置，并填充随机的颜色，如图 15-29b 所示。单击这个圆时，它会消失，然后在另一个随机的位置显示新的随机颜色的圆。在单击了 20 个圆之后，在面板上显示所用的时间，如图 15-29c 所示。

- **15.20 (几何问题：显示角度) 请编写一个程序，使用户可以拖动一个三角形的顶点，并在三角形改变时动态显示角度，如图 15-30a 所示。计算角度的公式在程序清单 4-1 中给出。



a) 练习题 15.20 允许用户拖动顶点并动态显示角度



b) 练习题 15.21 允许用户沿着圆拖动顶点并动态显示角度

图 15-30

- *15.21 (拖动点) 绘制一个圆，在圆上有三个随机点。连接这些点构成一个三角形。显示三角形中的角度。使用鼠标沿着圆的边拖动点。拖动的时候，三角形以及角度动态地重新显示，如图 15-30b 所示。计算三角形角度的公式参考程序清单 4-1。

15.10 节

- *15.22 (自动改变大小的圆柱) 重写编程练习题 14.10，当窗体改变大小的时候，圆柱的宽度和高度自动改变大小。
- *15.23 (自动改变大小的停止标识) 重写编程练习题 14.15，当窗体改变大小的时候，停止标识的宽度和高度自动改变大小。

15.11 节

- **15.24 (动画：来回摆动) 编写一个程序，用动画完成来回摆动，如图 15-31 所示。单击 / 释放鼠标以暂停 / 恢复动画。



图 15-31 程序用动画实现一个来回摆动

- **15.25 (动画：曲线上的球) 请编写一个程序，用动画实现一个沿着正弦函数曲线移动的球，如图 15-32 所示。当球到达右边界时，它从左边重新开始。用户可以单击鼠标左 / 右按钮来继续 / 暂停动画。

- *15.26 (改变透明度) 重写编程练习题 15.24，当球摆动的时候改变球的透明度。

- *15.27 (控制一个移动的文本) 请编写一个程序，显示一个移动的文本，如图 15-33a 和 15-33b 所示。文本从左到右循环的移动。当它消失在右侧的时候，又会从左侧再次出现。当鼠标按下的时候，文本停滞不动，当按钮释放的时候，将继续移动。

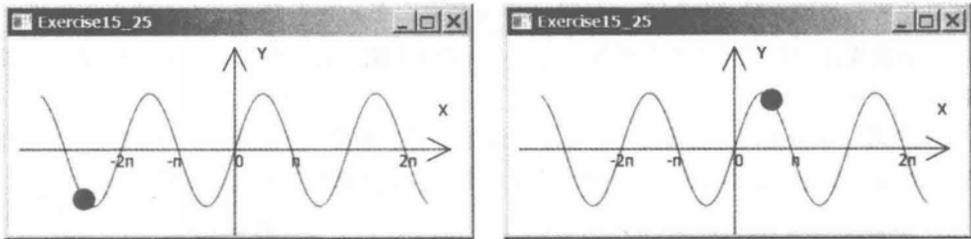
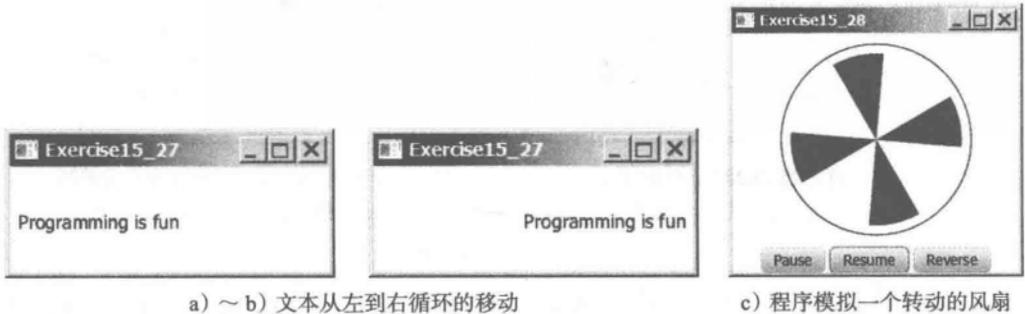


图 15-32 程序用动画实现一个沿着正弦函数曲线旅行的球



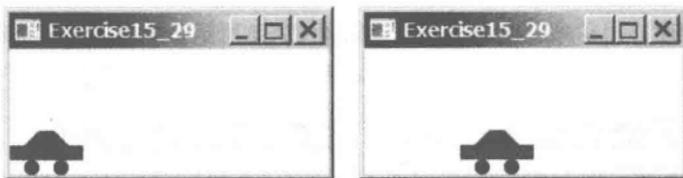
a) ~ b) 文本从左到右循环的移动

c) 程序模拟一个转动的风扇

图 15-33

**15.28 (显示一个转动的风扇) 编写一个程序显示一个转动的风扇, 如图 15-33c 所示。Pause、Resume 和 Reverse 按钮用于暂停、继续和反转风扇的转动。

**15.29 (赛车) 编写一个程序, 模拟汽车比赛, 如图 15-34a 所示。汽车从左向右移动。当它到达右端, 就从左边重新开始, 然后继续同样的过程。可以使用定时器控制动画。使用新的坐标原点 (x, y) 重新绘制汽车, 如图 15-34b 所示。同样让用户通过按钮的按下 / 释放来暂停 / 继续动画, 并且通过按下 UP 和 DOWN 的箭头键来增加 / 降低汽车速度。



a) 程序显示一个移动的汽车

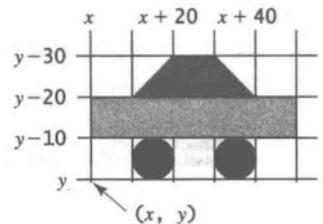
b) 在一个新的坐标原点 (x, y) 重新绘制汽车

图 15-34

**15.30 (播放幻灯片) 25 张幻灯片都以图像文件 (slide0.jpg, slide1.jpg, ..., slide24.jpg) 的形式存储在图像目录中, 可以在本书的源代码中下载。每个图像的大小都是 800×600 像素。编写一个 Java 应用程序, 自动重复显示这些幻灯片。每两秒显示一张幻灯片。幻灯片按顺序显示。当显示完最后一张幻灯片时, 第一张幻灯片重复显示, 依此类推。当动画正在播放的时候可以单击按钮暂停, 如果动画当前是暂停的, 单击恢复。

**15.31 (几何问题: 钟摆) 编写一个程序, 用动画完成钟摆, 如图 15-35 所示。单击向上箭头 UP 键增加速度, 单击向下箭头键 DOWN 降低速度。单击 S 键停止动画, 单击 R 键重新开始。

*15.32 (控制时钟) 修改程序清单 14-21, 在类中加入动画。添加两个方法 start() 和 stop() 以启动和停止时钟。编写一个程序, 让用户使用 Start 和 Stop 按钮来控制时钟, 如图 15-36a 所示。

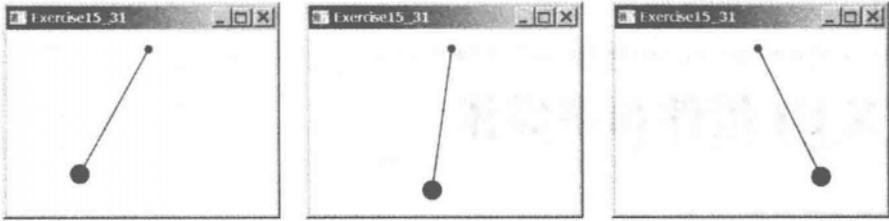
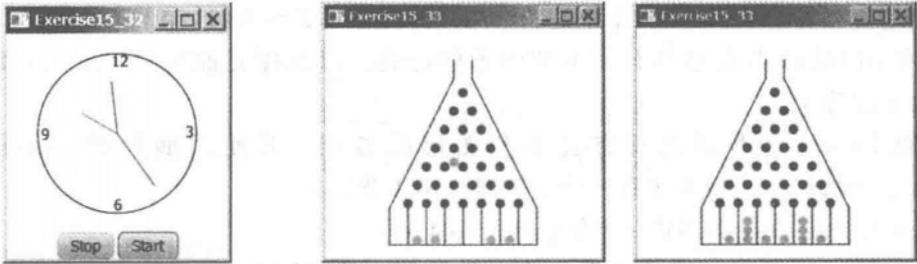


图 15-35 制作钟摆动画



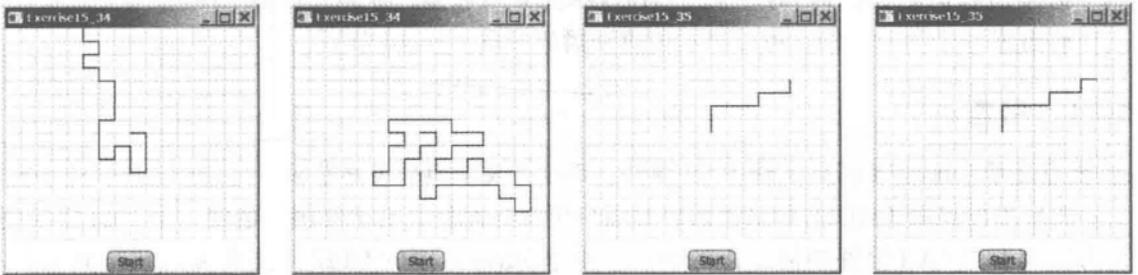
a) 练习题 15.32 让用户可以开始和停止一个时钟

b) ~ c) 球被扔进豆机

图 15-36

***15.33 (游戏: 豆机动画) 编写程序用动画实现编程练习题 7.21 中介绍的豆机。在 10 个球掉下来之后动画结束, 如图 15-36b 和 15-36c 所示。

***15.34 (模拟: 自回避随机漫步) 在一个网格中的自回避漫步是指从一个点到另一点的过程中, 不重复两次访问一个点。自回避漫步已经广泛应用在物理、化学和数学学科中。它们可以用来模拟像溶剂和聚合物这样的链状物。编写一个程序, 显示一个从中心点出发到边界点结束的随机路径, 如图 15-37a 所示, 或者在一个尽头点结束 (即该点被四个已经访问过的点包围), 如图 15-37b 所示。假设网格的大小是 16 × 16。



a) 一条在边界点结束的路径 b) 一条在尽头点结束的路径

c) ~ d) 动画显示逐步构造路径的过程

图 15-37

***15.35 (动画: 自回避随机漫步) 修改上一个练习题, 在一个动画中逐步地显示漫步, 如图 15-37c 和图 15-37d 所示。

**15.36 (模拟: 自回避随机漫步) 编写一个模拟程序, 显示出现尽头点路径的可能性随着格子数量的增加而变大。程序模拟大小从 10 到 80 的网格。对每种大小的网格, 模拟自回避随机漫步 10 000 次, 然后显示出现尽头点路径的概率, 如下面的示例输出所示:

```

For a lattice of size 10, the probability of dead-end paths is 10.6%
For a lattice of size 11, the probability of dead-end paths is 14.0%
...
For a lattice of size 80, the probability of dead-end paths is 99.5%

```

JavaFX UI 组件和多媒体

教学目标

- 使用各种用户界面组件来创建图形用户界面 (16.2 ~ 16.11 节)。
- 使用 Label 类创建具有文本和图形的标签, 以及探究抽象类 Labeled 类中的属性 (16.2 节)。
- 使用 Button 类创建具有文本和图形的按钮, 并使用抽象类 ButtonBase 中的 setOnAction 方法来设置一个处理器 (16.3 节)。
- 使用 CheckBox 类创建一个复选框 (16.4 节)。
- 使用 RadioButton 类来创建一个单选按钮, 并使用 ToggleGroup 来将单选按钮分组 (16.5 节)。
- 使用 TextField 类来输入数据, 以及使用 PasswordField 类来输入密码 (16.6 节)。
- 使用 TextArea 类来多行输入数据 (16.7 节)。
- 使用 ComboBox 来选择单个条目 (16.8 节)。
- 使用 ListView 来选择单个或者多个条目 (16.9 节)。
- 使用 ScrollBar 来选择范围内的值 (16.10 节)。
- 使用 Slider 来选择范围内的值, 并探究 ScrollBar 和 Slider 的区别 (16.11 节)。
- 开发一个 tic-tac-toe 游戏 (16.12 节)。
- 使用 Media、MediaPlayer 和 MediaView 来观看和播放视频和音频 (16.13 节)。
- 开发一个学习示例, 可以显示国旗和播放国歌 (16.14 节)。

16.1 引言

要点提示: JavaFX 提供了许多 UI 组件, 用于开发全面的用户界面。

图形用户界面 (GUI) 可以让系统对用户更友好而且更易于使用。创建一个 GUI 需要创造力以及有关 GUI 组件如何工作的知识。由于 JavaFX 中的 UI 组件非常灵活和功能全面, 你可以为富因特网应用创建类别广泛的实用的用户界面。

Oracle 公司提供了可视化设计和开发 GUI 的工具。这使得程序员可以用最少的编码快速将图形用户界面 (GUI) 元素组装在一起。然而, 任何工具都不是万能的。有时需要修改这些工具生成的程序。因此, 在开始使用可视化工具之前, 必须理解 JavaFX GUI 程序设计的一些基本概念。

前几章使用了一些 GUI 组件, 比如 Button、Label 和 TextField。本章详细地介绍经常会用到的 UI 组件 (参见图 16-1)。

注意: 整本书中, 前缀 lbl、bt、chk、rb、tf、pf、ta、cbo、lv、scb、sld 和 mp 被用于定义 Label、Button、CheckBox、RadioButton、TextField、PasswordField、TextArea、ComboBox、ListView、ScrollBar、Slider 和 MediaPlayer 的引用变量。

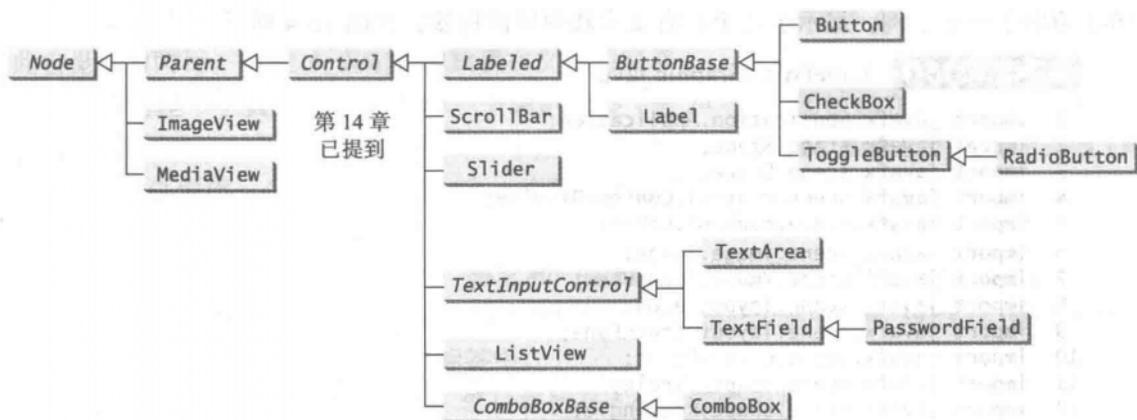
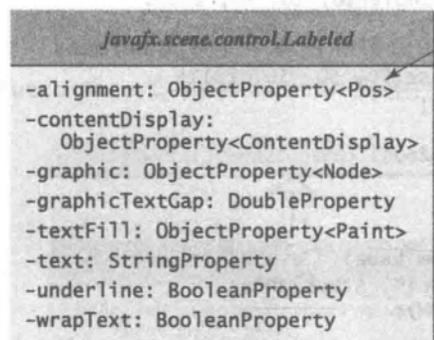


图 16-1 这些 UI 组件在创建用户界面中被经常使用

16.2 Labeled 和 Label

要点提示：JavaFX 提供了许多 UI 组件，用于开发全面的用户界面。

标签 (label) 是一个显示小段文字、一个节点或同时显示两者的区域。它经常用来给其他组件 (通常为文本域) 做标签。标签和按钮共享许多共同的属性。这些共同属性定义在 Labeled 类中，如图 16-2 所示。

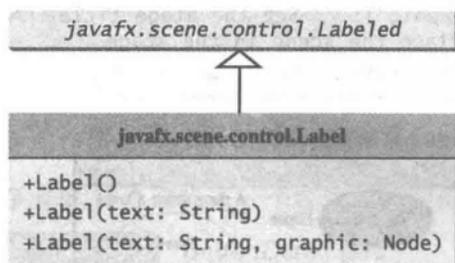


类中提供了属性值的获取和设置方法以及属性本身的获取方法，但是为简洁起见，在 UML 图中省略了

指定 labeled 中文本和节点的对齐方式
使用 ContentDisplay 中定义的常量 TOP、BOTTOM、LEFT 和 RIGHT 指定节点相对于文本的位置
用于 labeled 的图形
图形和文本之间的间隔
用于填充文本的图画
用于标签的文本
文本是否需要加下划线
如果文本超过了宽度，是否要换至下一行

图 16-2 Labeled 类定义了 Label、Button、CheckBox 和 RadioButton 的共同属性

Label 可以用下面三个构造方法的其中之一进行构建，如图 16-3 所示。



创建一个空 Label
创建一个特定文本的标签
创建一个特定文本和图形的标签

图 16-3 创建 Label 以显示文本或者一个节点，或同时显示两者

Graphic 属性可以是任何一个节点，比如一个形状、一个图像或者一个组件。程序清单

16-1 给出了一个示例，显示了几个具有文本和图像的标签，如图 16-4 所示。

程序清单 16-1 LabelWithGraphic.java

```

1  import javafx.application.Application;
2  import javafx.stage.Stage;
3  import javafx.scene.Scene;
4  import javafx.scene.control.ContentDisplay;
5  import javafx.scene.control.Label;
6  import javafx.scene.image.Image;
7  import javafx.scene.image.ImageView;
8  import javafx.scene.layout.HBox;
9  import javafx.scene.layout.StackPane;
10 import javafx.scene.paint.Color;
11 import javafx.scene.shape.Circle;
12 import javafx.scene.shape.Rectangle;
13 import javafx.scene.shape.Ellipse;
14
15 public class LabelWithGraphic extends Application {
16     @Override // Override the start method in the Application class
17     public void start(Stage primaryStage) {
18         ImageView us = new ImageView(new Image("image/us.gif"));
19         Label lb1 = new Label("US\n50 States", us);
20         lb1.setStyle("-fx-border-color: green; -fx-border-width: 2");
21         lb1.setContentDisplay(ContentDisplay.BOTTOM);
22         lb1.setTextFill(Color.RED);
23
24         Label lb2 = new Label("Circle", new Circle(50, 50, 25));
25         lb2.setContentDisplay(ContentDisplay.TOP);
26         lb2.setTextFill(Color.ORANGE);
27
28         Label lb3 = new Label("Rectangle", new Rectangle(10, 10, 50, 25));
29         lb3.setContentDisplay(ContentDisplay.RIGHT);
30
31         Label lb4 = new Label("Ellipse", new Ellipse(50, 50, 50, 25));
32         lb4.setContentDisplay(ContentDisplay.LEFT);
33
34         Ellipse ellipse = new Ellipse(50, 50, 50, 25);
35         ellipse.setStroke(Color.GREEN);
36         ellipse.setFill(Color.WHITE);
37         StackPane stackPane = new StackPane();
38         stackPane.getChildren().addAll(ellipse, new Label("JavaFX"));
39         Label lb5 = new Label("A pane inside a label", stackPane);
40         lb5.setContentDisplay(ContentDisplay.BOTTOM);
41
42         HBox pane = new HBox(20);
43         pane.getChildren().addAll(lb1, lb2, lb3, lb4, lb5);
44
45         // Create a scene and place it in the stage
46         Scene scene = new Scene(pane, 450, 150);
47         primaryStage.setTitle("LabelWithGraphic"); // Set the stage title
48         primaryStage.setScene(scene); // Place the scene in the stage
49         primaryStage.show(); // Display the stage
50     }
51 }
52 }
```

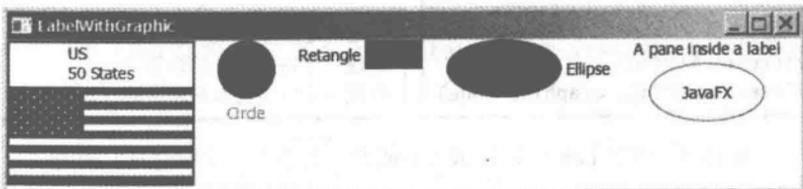


图 16-4 程序显示具有文本和节点的标签

程序创建一个具有一段文本和一个图像的标签 (第 19 行)。文本是 US\n50 States, 因此它显示为两行。第 21 行确定图像放置在文本的底部。

程序创建一个具有一段文本和一个圆的标签 (第 24 行)。圆被放在文本的上方 (第 25 行)。程序创建了一个具有一段文本和一个矩形的标签 (第 28 行)。矩形位于文本的右侧 (第 29 行)。程序创建一个具有一段文本和一个椭圆的标签 (第 31 行)。椭圆放置于文本的左侧 (第 32 行)。

程序创建一个椭圆 (第 34 行), 将它和一个标签一起放到一个堆栈面板中 (第 38 行), 然后创建一个具有一段文本以及将该堆栈面板作为节点的一个标签 (第 39 行)。如这个例子所示, 你可以将任何节点放在一个标签中。

程序创建一个 HBox (第 42 行), 然后将所有 5 个标签置于 HBox 中 (第 43 行)。

复习题

- 16.1 如何创建一个具有一个节点但是没有文本的标签?
- 16.2 如何在一个标签中将文本放在节点的右侧?
- 16.3 如何在一个标签中显示多行文本?
- 16.4 标签中的文本如何加下划线?

16.3 按钮

按钮 (button) 是单击时触发动作事件的组件。JavaFX 提供了常规按钮、开关按钮、复选框按钮和单选按钮。这些按钮的公共特性在 `ButtonBase` 和 `Labeled` 类中定义, 如图 16-5 所示。

`Labeled` 类定义了标签和按钮的共同属性。按钮和标签非常类似, 除了按钮具有定义在 `ButtonBase` 类中的 `onAction` 属性, 该属性设置一个用于处理按钮动作的处理器。

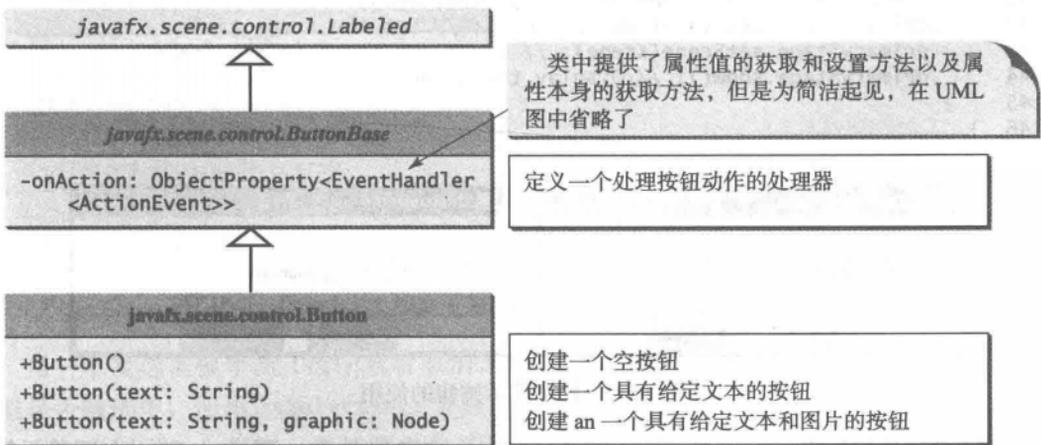


图 16-5 `ButtonBase` 继承自 `Labeled`, 定义了用于所有按钮的共同属性

程序清单 16-2 给出了一个程序, 使用按钮来控制一段文本的移动, 如图 16-6 所示。

程序清单 16-2 ButtonDemo.java

```

1 import javafx.application.Application;
2 import javafx.stage.Stage;
3 import javafx.geometry.Pos;
4 import javafx.scene.Scene;
  
```

```

5 import javafx.scene.control.Button;
6 import javafx.scene.image.ImageView;
7 import javafx.scene.layout.BorderPane;
8 import javafx.scene.layout.HBox;
9 import javafx.scene.layout.Pane;
10 import javafx.scene.text.Text;
11
12 public class ButtonDemo extends Application {
13     protected Text text = new Text(50, 50, "JavaFX Programming");
14
15     protected BorderPane getPane() {
16         HBox paneForButtons = new HBox(20);
17         Button btLeft = new Button("Left",
18             new ImageView("image/left.gif"));
19         Button btRight = new Button("Right",
20             new ImageView("image/right.gif"));
21         paneForButtons.getChildren().addAll(btLeft, btRight);
22         paneForButtons.setAlignment(Pos.CENTER);
23         paneForButtons.setStyle("-fx-border-color: green");
24
25         BorderPane pane = new BorderPane();
26         pane.setBottom(paneForButtons);
27
28         Pane paneForText = new Pane();
29         paneForText.getChildren().add(text);
30         pane.setCenter(paneForText);
31
32         btLeft.setOnAction(e -> text.setX(text.getX() - 10));
33         btRight.setOnAction(e -> text.setX(text.getX() + 10));
34
35         return pane;
36     }
37
38     @Override // Override the start method in the Application class
39     public void start(Stage primaryStage) {
40         // Create a scene and place it in the stage
41         Scene scene = new Scene(getPane(), 450, 200);
42         primaryStage.setTitle("ButtonDemo"); // Set the stage title
43         primaryStage.setScene(scene); // Place the scene in the stage
44         primaryStage.show(); // Display the stage
45     }
46 }

```

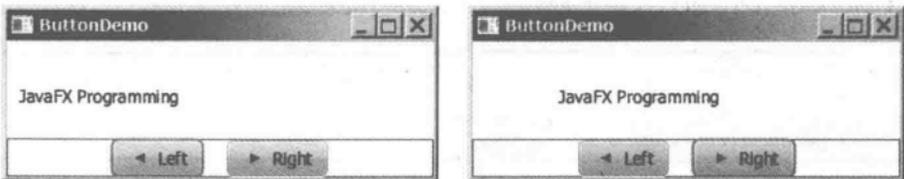


图 16-6 程序演示按钮的使用

程序创建了两个按钮 `btLeft` 和 `btRight`，每个按钮都包含一段文本和一个图像（第 17 到 20 行）。按钮置于一个 `HBox` 中（第 21 行），而 `HBox` 又放在一个 `border` 面板的底部（第 26 行）。第 13 行创建了一段文本并置于一个 `border` 面板中央（第 30 行）。`btLeft` 的动作处理器将文本往左边移动（第 32 行）。`btRight` 的动作处理器将文本往右边移动（第 33 行）。

程序有目的的定义了一个受保护的 `getPane()` 方法以返回一个面板（第 15 行）。该方法将在后面的例子中被子类重写，以在面板中增加更多节点。文本被声明为受保护的，从而可以被子类所访问到（第 13 行）。

复习题

- 16.5 如何创建一个具有一段文本和一个节点的按钮？可以将所有 Labeled 的方法应用于 Button 上吗？
- 16.6 程序清单 16-2 中为何 getPane() 方法是受保护的？为何数据域 text 是受保护的？
- 16.7 如何设置一个处理器用于处理按钮单击的动作？

16.4 复选框

复选框用于提供给用户进行选择。如同 Button、CheckBox 继承了来自 ButtonBase 和 Labeled 的所有属性，比如 onAction、text、graphic、alignment、graphicTextGap、textFill、contentDisplay，如图 16-7 所示。另外，它提供了 selected 属性用于表明一个复选框是否被选中。

下面是一个复选框的例子，该复选框具有文本 US，一个图像，绿色的文本颜色和黑色的边框，并且一开始是被选中状态。

```

CheckBox chkUS = new CheckBox("US");
chkUS.setGraphic(new ImageView("image/usIcon.gif"));
chkUS.setTextFill(Color.GREEN);
chkUS.setContentDisplay(ContentDisplay.LEFT);
chkUS.setStyle("-fx-border-color: black");
chkUS.setSelected(true);
chkUS.setPadding(new Insets(5, 5, 5, 5));
    
```

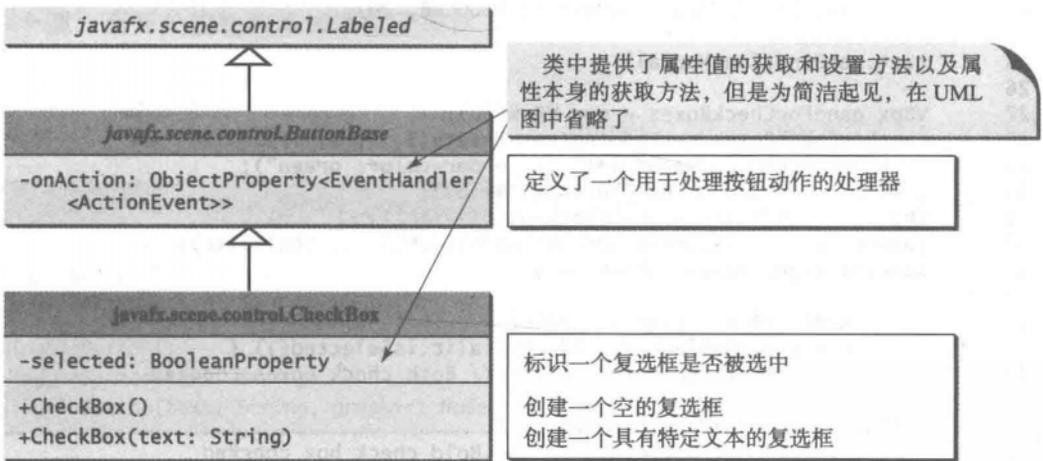


图 16-7 CheckBox 包含了继承自 ButtonBase 和 Labeled 的属性

当一个复选框被单击（选中或者取消选中），都会触发一个 ActionEvent。要判断一个复选框是否被选中，使用 isSelected() 方法。

现在我们写一个程序，增加两个命名为 Bold 和 Italic 的复选框到前面的例子中，让用户可以指定消息是使用黑体还是斜体，如图 16-8 所示。



图 16-8 程序演示复选框

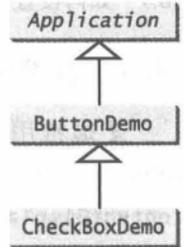
至少有两种途径来写这个程序。方法一是修改前面的 `ButtonDemo` 类来加入代码，用于增加复选框以及处理它们的事件。方法二是定义一个继承自 `ButtonDemo` 的子类。请实现第一种方法作为练习。程序清单 16-3 给出了实现第二种方法的代码。

程序清单 16-3 `CheckBoxDemo.java`

```

1  import javafx.event.ActionEvent;
2  import javafx.event.EventHandler;
3  import javafx.geometry.Insets;
4  import javafx.scene.control.CheckBox;
5  import javafx.scene.layout.BorderPane;
6  import javafx.scene.layout.VBox;
7  import javafx.scene.text.Font;
8  import javafx.scene.text.FontPosture;
9  import javafx.scene.text.FontWeight;
10
11 public class CheckBoxDemo extends ButtonDemo {
12     @Override // Override the getPane() method in the super class
13     protected BorderPane getPane() {
14         BorderPane pane = super.getPane();
15
16         Font fontBoldItalic = Font.font("Times New Roman",
17             FontWeight.BOLD, FontPosture.ITALIC, 20);
18         Font fontBold = Font.font("Times New Roman",
19             FontWeight.BOLD, FontPosture.REGULAR, 20);
20         Font fontItalic = Font.font("Times New Roman",
21             FontWeight.NORMAL, FontPosture.ITALIC, 20);
22         Font fontNormal = Font.font("Times New Roman",
23             FontWeight.NORMAL, FontPosture.REGULAR, 20);
24
25         text.setFont(fontNormal);
26
27         VBox paneForCheckBoxes = new VBox(20);
28         paneForCheckBoxes.setPadding(new Insets(5, 5, 5, 5));
29         paneForCheckBoxes.setStyle("-fx-border-color: green");
30         CheckBox chkBold = new CheckBox("Bold");
31         CheckBox chkItalic = new CheckBox("Italic");
32         paneForCheckBoxes.getChildren().addAll(chkBold, chkItalic);
33         pane.setRight(paneForCheckBoxes);
34
35         EventHandler<ActionEvent> handler = e -> {
36             if (chkBold.isSelected() && chkItalic.isSelected()) {
37                 text.setFont(fontBoldItalic); // Both check boxes checked
38             }
39             else if (chkBold.isSelected()) {
40                 text.setFont(fontBold); // The Bold check box checked
41             }
42             else if (chkItalic.isSelected()) {
43                 text.setFont(fontItalic); // The Italic check box checked
44             }
45             else {
46                 text.setFont(fontNormal); // Both check boxes unchecked
47             }
48         };
49
50         chkBold.setOnAction(handler);
51         chkItalic.setOnAction(handler);
52
53         return pane; // Return a new pane
54     }
55 }

```



`CheckBoxDemo` 继承自 `ButtonDemo` 并且重写了 `getPane()` 方法 (第 13 行)。新的 `getPane()`

方法调用 `ButtonDemo` 类的 `super.getPane()` 方法来获得一个包含了按钮和文本的 `border` 面板 (第 14 行)。复选框被创建并加入到 `paneForCheckBoxes` 中 (第 30 ~ 32 行)。`paneForCheckBoxes` 被加入到 `border` 面板中 (第 33 行)。

用于处理复选框动作事件的处理器在第 35 ~ 48 行被创建。它根据复选框的状态来设置合适的字体。

用于这个 JavaFX 程序的 `start` 方法在 `ButtonDemo` 中定义并在 `CheckBoxDemo` 中被继承。所以当运行 `CheckBoxDemo` 时, `ButtonDemo` 中的 `start` 方法被调用。由于 `getPane()` 方法在 `CheckBoxDemo` 中被重写, 程序清单 16-2 的第 41 行调用的是 `CheckBoxDemo` 中的方法。

复习题

- 16.8 如何检测一个复选框是否被选中?
 16.9 可以将用于 `Labeled` 的所有方法用于 `CheckBox`?
 16.10 可否将一个复选框中 `graphic` 属性设置为一个节点?

16.5 单选按钮

单选按钮 (radio button) 也称为选项按钮 (option button), 它可以让用户从一组选项中选择单一的一个条目。从外观上看, 单选按钮类似于复选框。复选框是方形的, 可以选中或者不选中; 而单选按钮显示一个圆, 或是填充的 (选中时), 或是空白的 (未选中时)。

`RadioButton` 是 `ToggleButton` 的子类。单选按钮和开关按钮的不同之处是, 单选按钮显示一个圆, 而开关按钮渲染成类似于按钮。`ToggleButton` 和 `RadioButton` 的 UML 图显示如图 16-9 所示。

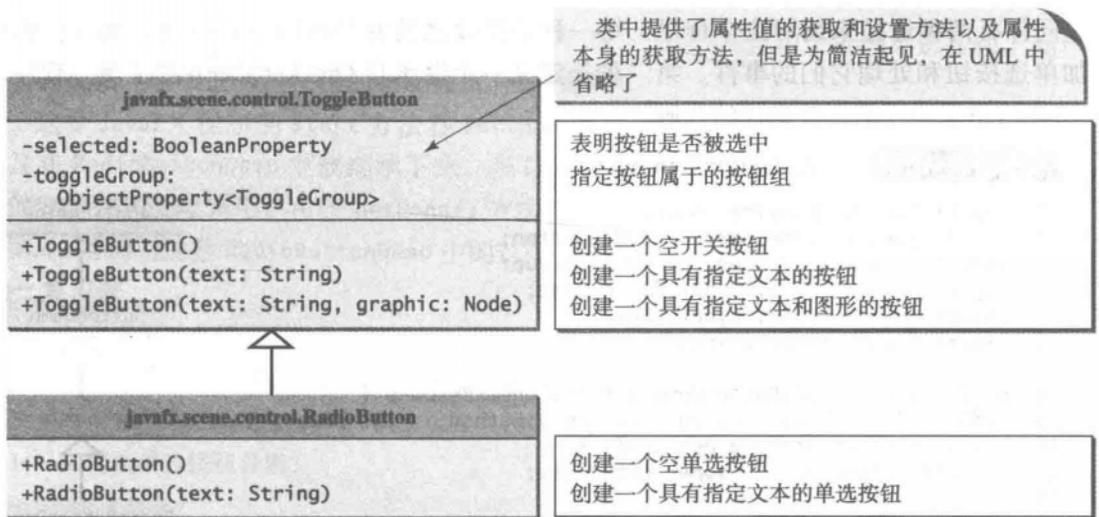


图 16-9 `ToggleButton` 和 `RadioButton` 是用于进行选择的特定按钮

这里是一个单选按钮的示例, 这个单选按钮有一个文本 `US`, 一个图片, 绿色的文本字体, 黑色的边框, 而且初始状态是选中的。

```

RadioButton rbUS = new RadioButton("US");
rbUS.setGraphic(new ImageView("image/usIcon.gif"));
rbUS.setTextFill(Color.GREEN);
rbUS.setContentDisplay(ContentDisplay.LEFT);
  
```

```
rbUS.setStyle("-fx-border-color: black");
rbUS.setSelected(true);
rbUS.setPadding(new Insets(5, 5, 5,));
```



为了将单选按钮分组，需要创建一个 `ToggleGroup` 的实例，并且设置一个单选按钮的 `ToggleGroup` 属性以加入分组，如下所示：

```
ToggleGroup group = new ToggleGroup();
rbRed.setToggleGroup(group);
rbGreen.setToggleGroup(group);
rbBlue.setToggleGroup(group);
```

这段代码为单选按钮 `rbRed`、`rbGreen`、`rbBlue` 创建一个按钮组，从而 `rbRed`、`rbGreen` 和 `rbBlue` 可以互斥地单一选择。没有分组的话，这些按钮将是独立的。

当一个单选按钮被改变时（选中或者取消选中），它触发一个 `ActionEvent`。如果要判断一个单选按钮是否选中，使用 `isSelected()` 方法。

现在我们给出一个程序，将命名为 `Red`、`Green` 和 `Blue` 的三个单选按钮加入之前的例子，让用户可以选择消息的颜色，如图 16-10 所示。

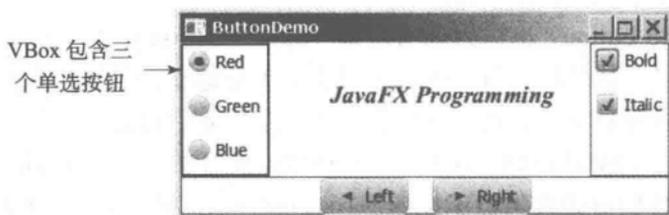
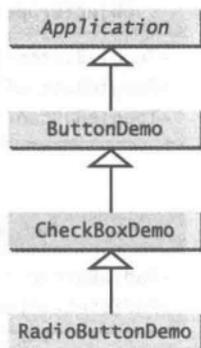


图 16-10 程序演示单选按钮的使用

同样有两种途径来编写这个程序。第一种是修改之前的 `CheckBoxDemo` 类，加入代码以增加单选按钮和处理它们的事件。第二种是定义一个继承自 `CheckBoxDemo` 的子类。程序清单 16-4 给出了实现第二种途径的代码。

程序清单 16-4 RadioButtonDemo.java

```
1 import javafx.geometry.Insets;
2 import javafx.scene.control.RadioButton;
3 import javafx.scene.control.ToggleGroup;
4 import javafx.scene.layout.BorderPane;
5 import javafx.scene.layout.VBox;
6 import javafx.scene.paint.Color;
7
8 public class RadioButtonDemo extends CheckBoxDemo {
9     @Override // Override the getPane() method in the super class
10    protected BorderPane getPane() {
11        BorderPane pane = super.getPane();
12
13        VBox paneForRadioButtons = new VBox(20);
14        paneForRadioButtons.setPadding(new Insets(5, 5, 5, 5));
15        paneForRadioButtons.setStyle("-fx-border-color: green");
16        paneForRadioButtons.setStyle
17            ("-fx-border-width: 2px; -fx-border-color: green");
18        RadioButton rbRed = new RadioButton("Red");
19        RadioButton rbGreen = new RadioButton("Green");
20        RadioButton rbBlue = new RadioButton("Blue");
21        paneForRadioButtons.getChildren().addAll(rbRed, rbGreen, rbBlue);
22        pane.setLeft(paneForRadioButtons);
23
24        ToggleGroup group = new ToggleGroup();
```



```
25     rbRed.setToggleGroup(group);
26     rbGreen.setToggleGroup(group);
27     rbBlue.setToggleGroup(group);
28
29     rbRed.setOnAction(e -> {
30         if (rbRed.isSelected()) {
31             text.setFill(Color.RED);
32         }
33     });
34
35     rbGreen.setOnAction(e -> {
36         if (rbGreen.isSelected()) {
37             text.setFill(Color.GREEN);
38         }
39     });
40
41     rbBlue.setOnAction(e -> {
42         if (rbBlue.isSelected()) {
43             text.setFill(Color.BLUE);
44         }
45     });
46
47     return pane;
48 }
49 }
```

RadioButtonDemo 继承自 CheckBoxDemo，重写 getPane() 方法（第 10 行）。新的 getPane() 方法调用来自 CheckBoxDemo 的 getPane() 方法，创建一个包含了复选按钮、按钮和一段文本的边框面板（第 11 行）。这个边框面板是通过调用 super.getPane() 返回的。创建单选按钮并将其加入到 paneForRadioButtons 中（第 18 ~ 21 行）。paneForRadioButtons 加入到边框面板中（第 22 行）。

代码 24 ~ 27 行将单选按钮组在一起。代码 29 ~ 45 行创建用于处理单选按钮上动作事件的处理器。它根据单选按钮的状态设置合适的颜色。

这个 JavaFX 程序的 start 方法在 ButtonDemo 中定义，在 CheckBoxDemo 中被继承，然后又在 RadioButtonDemo 中被继承下来。所以，当你运行 RadioButtonDemo 时，ButtonDemo 中的 start 方法被调用。由于 getPane() 方法在 RadioButtonDemo 中被重写，程序清单 16-2 中第 41 行调用的是 RadioButtonDemo 中的这个方法。

复习题

- 16.11 如何判断一个单选按钮是否被选中？
- 16.12 可以将 Labeled 中所有的方法应用于 RadioButton 吗？
- 16.13 可以将单选按钮的 graphic 属性设置为任何节点吗？
- 16.14 如何将单选按钮分组？

16.6 文本域

文本域 (text field) 可用于输入或显示一个字符串。TextField 是 TextInputControl 的子类。图 16-11 列举了 TextField 中的属性和构造方法。

这里是一个例子，创建一个不可编辑的文本域，具有红色的文本颜色，指定的字体以及水平右对齐的排版。

```
TextField tfMessage = new TextField("T-Storm");
tfMessage.setEditable(false);
```

```
tfMessage.setStyle("-fx-text-fill: red");
tfMessage.setFont(Font.font("Times", 20));
tfMessage.setAlignment(Pos.BASELINE_RIGHT);
```

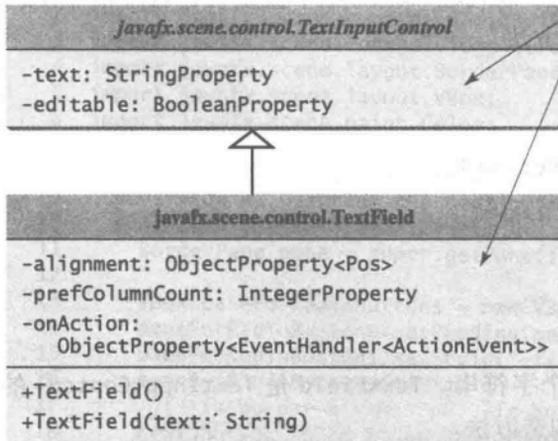
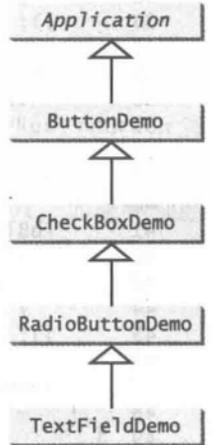
T-Strom

当你将光标移至文本域并按下回车键时，它将触发一个 `ActionEvent` 事件。

程序清单 16-5 给出了一个程序，在前面的例子中增加了一条文本域，让用户可以创建一个新的消息，如图 16-12 所示。

程序清单 16-5 TextFieldDemo.java

```
1 import javafx.geometry.Insets;
2 import javafx.geometry.Pos;
3 import javafx.scene.control.Label;
4 import javafx.scene.control.TextField;
5 import javafx.scene.layout.BorderPane;
6
7 public class TextFieldDemo extends RadioButtonDemo {
8     @Override // Override the getPane() method in the super class
9     protected BorderPane getPane() {
10         BorderPane pane = super.getPane();
11
12         BorderPane paneForTextField = new BorderPane();
13         paneForTextField.setPadding(new Insets(5, 5, 5, 5));
14         paneForTextField.setStyle("-fx-border-color: green");
15         paneForTextField.setLeft(new Label("Enter a new message: "));
16
17         TextField tf = new TextField();
18         tf.setAlignment(Pos.BOTTOM_RIGHT);
19         paneForTextField.setCenter(tf);
20         pane.setTop(paneForTextField);
21
22         tf.setOnAction(e -> text.setText(tf.getText()));
23
24         return pane;
25     }
26 }
```



类中提供了属性值的获取和设置方法以及属性本身的获取方法，但是为简洁起见，在 UML 图中省略了

该组件中的文本内容
表明文本是否可以被用户编辑

指定文本在文本域中如何对齐
指定文本域优先列数
指定文本域上动作事件的处理器

创建一个空的文本域
创建一个具有指定文本的文本域

图 16-11 TextField 让用户可以输入或者显示一个字符串

`TextFieldDemo` 继承自 `RadioButtonDemo` (第 7 行)，增加了一个标签和文本域让用户输入新的文本 (第 12 ~ 19 行)。当在文本域中设定了一个新的文本并且按回车键后，一条新

的消息将被显示（第 22 行）。在文本域中按回车键将触发一个动作事件。

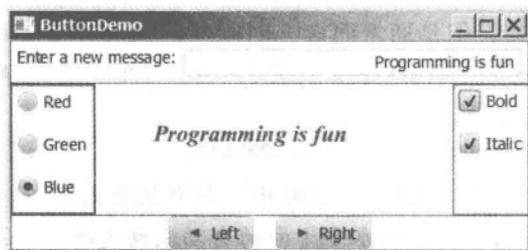


图 16-12 程序演示文本域的使用

注意：如果一个文本域用于输入密码，使用 `PasswordField` 来替代 `TextField`。`PasswordField` 继承自 `TextField`，将输入文本隐藏为回显字符 `*****`。

复习题

- 16.15 可以禁用文本域的编辑功能吗？
- 16.16 可以将 `TextInputControl` 的所有方法应用于 `TextField` 之上吗？
- 16.17 可以将文本域的 `graphic` 属性设置为一个节点吗？
- 16.18 如何将文本域里面的文本设置为右对齐？

16.7 文本区域

要点提示：`TextArea` 允许用户输入多行文本。

如果你希望让用户输入多行文本，你可以创建多个 `TextField` 的实例。然而，一个更好的选择是使用 `TextArea`，它允许用户输入多行文本。图 16-13 列出 `TextArea` 的属性和构造方法。

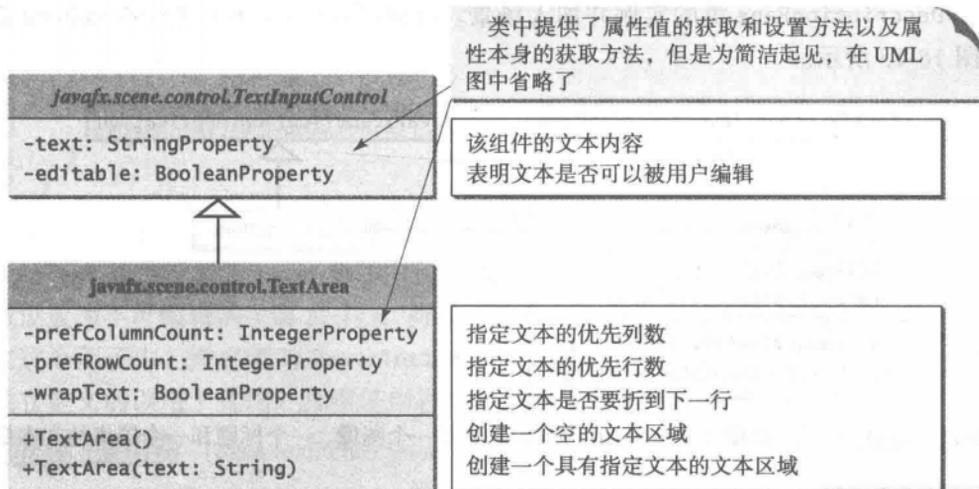


图 16-13 `TextArea` 使得用户可以输入和显示多行字符

这里是一个示例，创建一个具有 5 行 20 列的文本区域，可以折到下一行，红色的文本颜色，字体是 Courier 以及 20 像素。

```

TextArea taNote = new TextArea("This is a text area");
taNote.setPrefColumnCount(20);
taNote.setPrefRowCount(5);
taNote.setText("This is a text area");
taNote.setStyle("-fx-text-fill: red; -fx-font-family: Courier; -fx-font-size: 20px;");
  
```

```
taNote.setStyle("-fx-text-fill: red");
taNote.setFont(Font.font("Times", 20));
```

TextArea 提供滚动支持，但是通常而言，创建一个 ScrollPane 对象来包含一个 TextArea 的实例，并且让 ScrollPane 处理 TextArea 的滚动会更加方便，如下面代码所示：

```
// Create a scroll pane to hold text area
ScrollPane scrollPane = new ScrollPane(taNote);
```

提示：可以将任何节点放置在 ScrollPane 中。如果控件太大以致于不能在显示区域内完整显示，ScrollPane 提供了垂直和水平方向的自动滚动支持。

现在给出一个程序，在一个标签上显示图像和一段短文本，在一个文本区域内显示一段长文本，如图 16-14 所示。

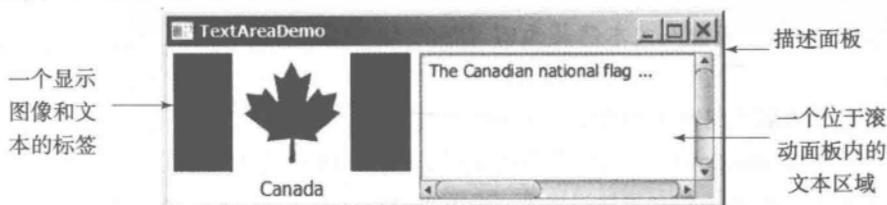


图 16-14 程序显示一个标签内图像、一个标签内标题和一个文本区域内的文本

下面是程序中的几个主要步骤：

1) 定义一个继承自 `BorderPane` 的命名为 `DescriptionPane` 的类，如程序清单 16-6 所示。这个类包含了一个滚动面板内的文本区域，以及一个显示一个图像图标和一个标题的标签。类 `DescriptionPane` 在后面的例子中将被重用。

2) 定义一个继承自 `Application` 的命名为 `TextAreaDemo` 的类，如程序清单 16-7 所示。创建一个 `DescriptionPane` 类的实例并加入场景。`DescriptionPane` 和 `TextAreaDemo` 之间的关系如图 16-15 所示。

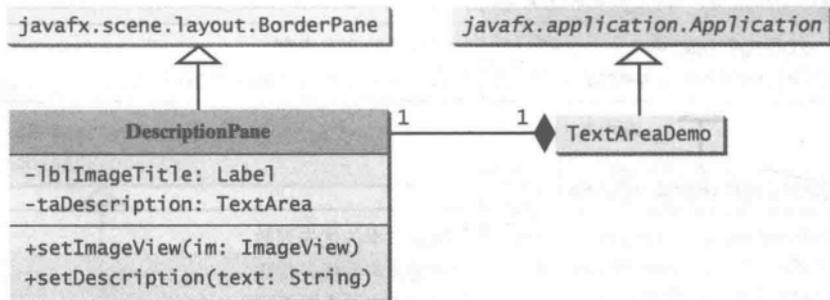


图 16-15 TextAreaDemo 使用 DescriptionPane 显示一个图像、一个标题和一个国旗的文本描述

程序清单 16-6 DescriptionPane.java

```
1 import javafx.geometry.Insets;
2 import javafx.scene.control.Label;
3 import javafx.scene.control.ContentDisplay;
4 import javafx.scene.control.ScrollPane;
5 import javafx.scene.control.TextArea;
6 import javafx.scene.image.ImageView;
7 import javafx.scene.layout.BorderPane;
8 import javafx.scene.text.Font;
9
```

```
10 public class DescriptionPane extends BorderPane {
11     /** Label for displaying an image and a title */
12     private Label lblImageTitle = new Label();
13
14     /** Text area for displaying text */
15     private TextArea taDescription = new TextArea();
16
17     public DescriptionPane() {
18         // Center the icon and text and place the text under the icon
19         lblImageTitle.setContentDisplay(ContentDisplay.TOP);
20         lblImageTitle.setPrefSize(200, 100);
21
22         // Set the font in the label and the text field
23         lblImageTitle.setFont(new Font("SansSerif", 16));
24         taDescription.setFont(new Font("Serif", 14));
25
26         taDescription.setWrapText(true);
27         taDescription.setEditable(false);
28
29         // Create a scroll pane to hold the text area
30         ScrollPane scrollPane = new ScrollPane(taDescription);
31
32         // Place label and scroll pane in the border pane
33         setLeft(lblImageTitle);
34         setCenter(scrollPane);
35         setPadding(new Insets(5, 5, 5, 5));
36     }
37
38     /** Set the title */
39     public void setTitle(String title) {
40         lblImageTitle.setText(title);
41     }
42
43     /** Set the image view */
44     public void setImageView(ImageView icon) {
45         lblImageTitle.setGraphic(icon);
46     }
47
48     /** Set the text description */
49     public void setDescription(String text) {
50         taDescription.setText(text);
51     }
52 }
```

本文区域位于一个 `ScrollPane` 中 (第 30 行), `ScrollPane` 为文本区域提供滚动功能。

`wrapText` 属性设置为 `true` (第 26 行), 因此当文本不能一行内显示的时候自动换行。文本区域设置为不可编辑的 (第 27 行), 所以不能在文本区域中编辑描述文字。

在这个例子中, 没必要为 `DescriptionPane` 创建一个独立的类。然而, 在下一节中这个类是独立定义的以用于重用, 且将使用它为多个图像显示描述面板。

程序清单 16-7 TextAreaDemo.java

```
1 import javafx.application.Application;
2 import javafx.stage.Stage;
3 import javafx.scene.Scene;
4 import javafx.scene.image.ImageView;
5
6 public class TextAreaDemo extends Application {
7     @Override // Override the start method in the Application class
8     public void start(Stage primaryStage) {
9         // Declare and create a description pane
10        DescriptionPane descriptionPane = new DescriptionPane();
```

```

11
12 // Set title, text, and image in the description pane
13 descriptionPane.setTitle("Canada");
14 String description = "The Canadian national flag ...";
15 descriptionPane.setImageView(new ImageView("image/ca.gif"));
16 descriptionPane.setDescription(description);
17
18 // Create a scene and place it in the stage
19 Scene scene = new Scene(descriptionPane, 450, 200);
20 primaryStage.setTitle("TextAreaDemo"); // Set the stage title
21 primaryStage.setScene(scene); // Place the scene in the stage
22 primaryStage.show(); // Display the stage
23 }
24 }

```

程序创建了一个 `DescriptionPane` 类的实例（第 10 行），然后在描述面板内设置了标题（第 13 行）、图像（第 15 行）以及文本（第 16 行）。`DescriptionPane` 是 `Pane` 的子类，它包含一个显示图像和标题的标签，以及显示关于图像的描述的一个文本区域。

复习题

- 16.19 如何创建一个具有 10 行、20 列的文本区域？
 16.20 如何获得文本区域里面的文本？
 16.21 如何禁用一个文本区域里面的编辑功能？
 16.22 在文本区域里面可以使用什么方法来将一行文本进行折行显示？

16.8 组合框

要点提示：组合框（combo box）也称为选择列表（choice list）或下拉式列表（drop-down list），它包含一个条目列表，用户能够从中进行选择。

使用组合框可以限制用户的选择范围，并避免对输入数据有效性进行繁琐的检查。图 16-16 列出在 `ComboBox` 类中一些常用的属性和构造方法。`ComboBox` 定义为一个泛型类。泛型 `T` 为保存在一个组合框中的元素指定元素类型。

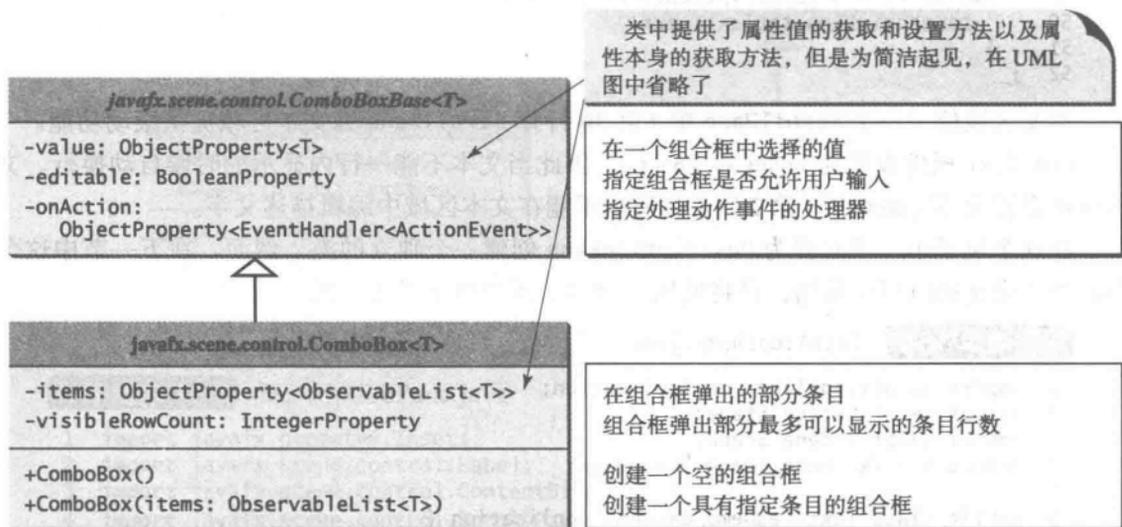


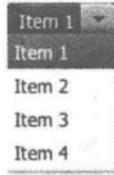
图 16-16 `ComboBox` 使得用户可以从条目列表中选择一个条目

下面的语句创建一个有四个条目的红色组合框，然后选中第一个条目：

```

ComboBox<String> cbo = new ComboBox<>();
cbo.getItems().addAll("Item 1", "Item 2",
    "Item 3", "Item 4");
cbo.setStyle("-fx-color: red");
cbo.setValue("Item 1");

```



ComboBox 继承自 ComboBoxBase。ComboBox 可以触发一个 ActionEvent 事件。当一个条目被选中时，一个 ActionEvent 事件被触发。ObservableList 是 java.util.List 的子接口。因此你可以将定义在 List 中的所有方法应用于 ObservableList。为了方便，JavaFX 提供了一个静态方法 FXCollections.observableArrayList(arrayOfElements) 来从一个元素数组中创建一个 ObservableList。

程序清单 16-8 使用户可以通过组合框选择国家，从而查看该国家国旗的图像及其描述，如图 16-17 所示。

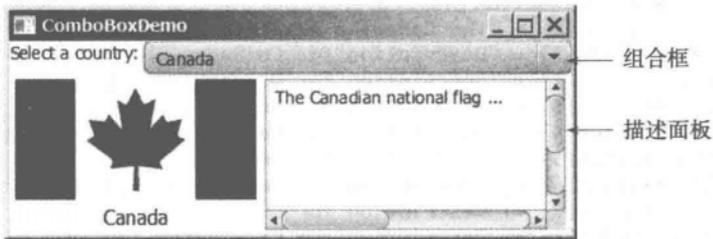


图 16-17 当组合框中的国家名被选定时将显示关于这个国家的信息，包含它的国旗的图像和描述

以下是程序的几个主要步骤：

1) 创建用户界面。

创建一个组合框，将国家名作为选择值。创建一个 DescriptionPane 对象 (DescriptionPane 类在前一节中介绍过)。将组合框放置在边框面板的上部，而将描述面板放置在边框面板的中央。

2) 处理事件。

创建一个处理器以处理来自组合框的动作事件、用于设置选定国家名字的国旗的标题、图像以及描述面板中的文本。

程序清单 16-8 ComboBoxDemo.java

```

1 import javafx.application.Application;
2 import javafx.stage.Stage;
3 import javafx.collections.FXCollections;
4 import javafx.collections.ObservableList;
5 import javafx.scene.Scene;
6 import javafx.scene.control.ComboBox;
7 import javafx.scene.control.Label;
8 import javafx.scene.image.ImageView;
9 import javafx.scene.layout.BorderPane;
10
11 public class ComboBoxDemo extends Application {
12     // Declare an array of Strings for flag titles
13     private String[] flagTitles = {"Canada", "China", "Denmark",
14         "France", "Germany", "India", "Norway", "United Kingdom",
15         "United States of America"};
16
17     // Declare an ImageView array for the national flags of 9 countries
18     private ImageView[] flagImage = {new ImageView("image/ca.gif"),

```

```

19     new ImageView("image/china.gif"),
20     new ImageView("image/denmark.gif"),
21     new ImageView("image/fr.gif"),
22     new ImageView("image/germany.gif"),
23     new ImageView("image/india.gif"),
24     new ImageView("image/norway.gif"),
25     new ImageView("image/uk.gif"), new ImageView("image/us.gif"));
26
27 // Declare an array of strings for flag descriptions
28 private String[] flagDescription = new String[9];
29
30 // Declare and create a description pane
31 private DescriptionPane descriptionPane = new DescriptionPane();
32
33 // Create a combo box for selecting countries
34 private ComboBox<String> cbo = new ComboBox<>(); // flagTitles;
35
36 @Override // Override the start method in the Application class
37 public void start(Stage primaryStage) {
38     // Set text description
39     flagDescription[0] = "The Canadian national flag ...";
40     flagDescription[1] = "Description for China ... ";
41     flagDescription[2] = "Description for Denmark ... ";
42     flagDescription[3] = "Description for France ... ";
43     flagDescription[4] = "Description for Germany ... ";
44     flagDescription[5] = "Description for India ... ";
45     flagDescription[6] = "Description for Norway ... ";
46     flagDescription[7] = "Description for UK ... ";
47     flagDescription[8] = "Description for US ... ";
48
49     // Set the first country (Canada) for display
50     setDisplay(0);
51
52     // Add combo box and description pane to the border pane
53     BorderPane pane = new BorderPane();
54
55     BorderPane paneForComboBox = new BorderPane();
56     paneForComboBox.setLeft(new Label("Select a country: "));
57     paneForComboBox.setCenter(cbo);
58     pane.setTop(paneForComboBox);
59     cbo.setPrefWidth(400);
60     cbo.setValue("Canada");
61
62     ObservableList<String> items =
63         FXCollections.observableArrayList(flagTitles);
64     cbo.getItems().addAll(items);
65     pane.setCenter(descriptionPane);
66
67     // Display the selected country
68     cbo.setOnAction(e -> setDisplay(items.indexOf(cbo.getValue())));
69
70     // Create a scene and place it in the stage
71     Scene scene = new Scene(pane, 450, 170);
72     primaryStage.setTitle("ComboBoxDemo"); // Set the stage title
73     primaryStage.setScene(scene); // Place the scene in the stage
74     primaryStage.show(); // Display the stage
75 }
76
77 /** Set display information on the description pane */
78 public void setDisplay(int index) {
79     descriptionPane.setTitle(flagTitles[index]);
80     descriptionPane.setImageView(flagImage[index]);
81     descriptionPane.setDescription(flagDescription[index]);
82 }
83 }

```

程序将国旗信息存储在三个数组：flagTitles、flagImage 和 flagDescription（第 13 ~ 28 行）中。数组 flagTitles 存放九个国家的名称，数组 flagImage 存放九个国家国旗的图像，数组 flagDescription 存放对这些国旗的描述。

程序创建 DescriptionPane 类的一个实例（第 31 行），该类在程序清单 16-6 中给出。程序以数组 flagTitles 中的值创建一个组合框（第 62 ~ 63 行）。getItem() 方法从组合框返回一个列表（第 64 行），addAll 方法将多个条目加入到列表中。

当用户选择组合框中的一项后，动作事件触发处理器的执行。处理器确定选中项的索引值（第 68 行），并调用 setDisplay(int index) 方法在面板上设置相应的国旗名、国旗图像及国旗描述（第 78 ~ 82 行）。

复习题

- 16.23 如何创建一个组合框并加入三个条目？
- 16.24 如何从一个组合框中获取一个条目？如何从一个组合框中获取一个选中条目？
- 16.25 如何得到一个组合框中的条目数？如何获得组合框中一个指定索引号的条目？
- 16.26 当选择一个新的条目时，ComboBox 将触发什么事件？

16.9 列表视图

要点提示：列表视图是一个组件，它完成的功能与组合框基本相同，但它允许用户选择一个或多个值。

图 16-18 列出了 ListView 中一些常用的属性和构造方法。ListView 定义为一个泛型类。泛型 T 为存储在一个列表视图中的元素指定了元素类型。

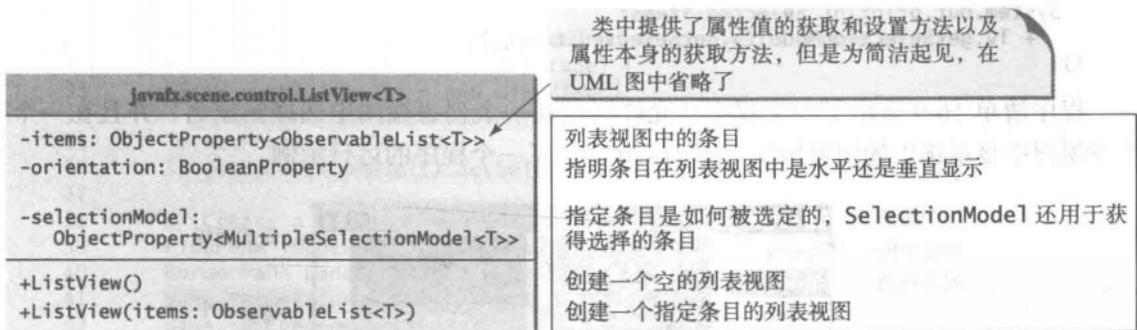


图 16-18 ListView 让用户可以从一个条目列表选择一个或者多个条目

getSelectionMode() 方法返回一个 SelectionModel 实例，该实例包含了设置选择模式以及获得被选中的索引值和条目的方法。选择模式由以下两个常量之一定义，SelectionMode.MULTIPLE 和 SelectionMode.SINGLE。这两个值指明可以选择单个还是多个条目。默认值是 SelectionMode.SINGLE。图 16-19a 显示了一个单选示例，图 16-19b ~ c 显示了多项选择。

以下语句创建了一个具有六个选择项的列表视图，允许多项选择。

```
ObservableList<String> items =
    FXCollections.observableArrayList("Item 1", "Item 2",
        "Item 3", "Item 4", "Item 5", "Item 6");
ListView<String> lv = new ListView<>(items);
lv.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
```

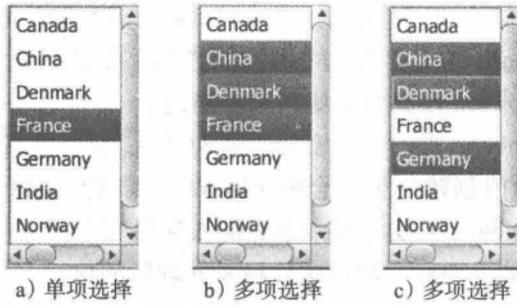


图 16-19 SelectionModel 有两种选择模式：单选和多项 - 间隔选择

列表视图的选择模式具有 `selectedItemProperty` 属性，该属性是一个 `Observable` 的实例。如 15.10 节中所讨论的，可以在这个属性上加一个监听器用以处理属性的变化，如下所示：

```
lv.getSelectionModel().selectedItemProperty().addListener(
    new InvalidationListener() {
        public void invalidated(Observable ov) {
            System.out.println("Selected indices: "
                + lv.getSelectionModel().getSelectedIndices());
            System.out.println("Selected items: "
                + lv.getSelectionModel().getSelectedItems());
        }
    });
```

这个匿名内部类可以使用 `lambda` 表达式简化如下：

```
lv.getSelectionModel().selectedItemProperty().addListener(ov -> {
    System.out.println("Selected indices: "
        + lv.getSelectionModel().getSelectedIndices());
    System.out.println("Selected items: "
        + lv.getSelectionModel().getSelectedItems());
});
```

程序清单 16-9 给出了一个程序，允许用户在一个列表视图中选择国家名，并且在一个图像视图中显示选中的国家国旗。图 16-20 显示了一个程序的运行示例。

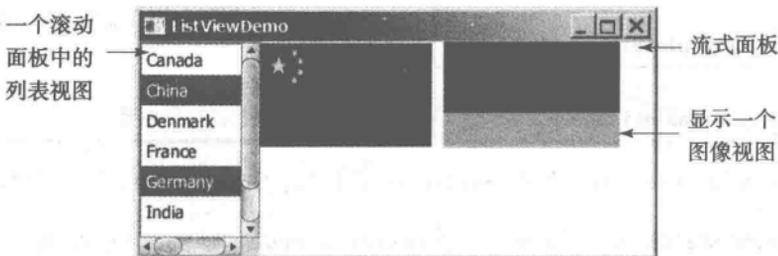


图 16-20 当列表视图中的国家被选中时，相应的国旗图像在图像视图中显示

这里是程序的几个主要步骤：

1) 创建用户界面。

创建有九个国家名的列表作为选择值，然后将这个列表框放到一个滚动面板中。将滚动面板放到边框面板的左边。创建九个图像视图用来显示这九个国家的国旗图像。创建一个流式面板来包含图像视图，并且将面板放在在边框面板的中央。

2) 处理事件。

创建一个监听器，实现 `InvalidationListener` 接口中的 `invalidated` 方法，在面板中

放置选定国家的国旗图像视图。

程序清单 16-9 ListViewDemo.java

```
1 import javafx.application.Application;
2 import javafx.stage.Stage;
3 import javafx.collections.FXCollections;
4 import javafx.scene.Scene;
5 import javafx.scene.control.ListView;
6 import javafx.scene.control.ScrollPane;
7 import javafx.scene.control.SelectionMode;
8 import javafx.scene.image.ImageView;
9 import javafx.scene.layout.BorderPane;
10 import javafx.scene.layout.FlowPane;
11
12 public class ListViewDemo extends Application {
13     // Declare an array of Strings for flag titles
14     private String[] flagTitles = {"Canada", "China", "Denmark",
15         "France", "Germany", "India", "Norway", "United Kingdom",
16         "United States of America"};
17
18     // Declare an ImageView array for the national flags of 9 countries
19     private ImageView[] ImageViews = {
20         new ImageView("image/ca.gif"),
21         new ImageView("image/china.gif"),
22         new ImageView("image/denmark.gif"),
23         new ImageView("image/fr.gif"),
24         new ImageView("image/germany.gif"),
25         new ImageView("image/india.gif"),
26         new ImageView("image/norway.gif"),
27         new ImageView("image/uk.gif"),
28         new ImageView("image/us.gif")
29     };
30
31     @Override // Override the start method in the Application class
32     public void start(Stage primaryStage) {
33         ListView<String> lv = new ListView<
34             >(FXCollections.observableArrayList(flagTitles));
35         lv.setPrefSize(400, 400);
36         lv.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
37
38         // Create a pane to hold image views
39         FlowPane imagePane = new FlowPane(10, 10);
40         BorderPane pane = new BorderPane();
41         pane.setLeft(new ScrollPane(lv));
42         pane.setCenter(imagePane);
43
44         lv.getSelectionModel().selectedItemProperty().addListener(
45             ov -> {
46                 imagePane.getChildren().clear();
47                 for (Integer i: lv.getSelectionModel().getSelectedIndices()) {
48                     imagePane.getChildren().add(ImageViews[i]);
49                 }
50             });
51
52         // Create a scene and place it in the stage
53         Scene scene = new Scene(pane, 450, 170);
54         primaryStage.setTitle("ListViewDemo"); // Set the stage title
55         primaryStage.setScene(scene); // Place the scene in the stage
56         primaryStage.show(); // Display the stage
57     }
58 }
```

程序创建一个代表国家的字符串数组 (第 14 ~ 16 行), 以及一个包含 9 个图像视图的

数组，用于显示代表 9 个国家的国旗图像（第 19 ~ 29 行），和前面代表国家的数组保持顺序一致。列表视图中的条目来自代表国家的数组（第 34 行）。因此，图像视图数组中的序号 0 对应于列表视图数组中的第一个国家。

列表视图置于一个滚动面板中（第 41 行），这样当列表中的条目数超过显示区域的时候可以滚动。

默认的，列表视图的选择模式是单选。列表视图的选择模式被设为多选（第 36 行），从而允许用户在列表视图中选择多项。当用户在列表视图中选择了国家，监听器的处理器（第 44 ~ 50 行）被执行，从而得到被选中条目的序号，并且将它们相应的图像视图加入到流式面板中。

复习题

- 16.27 如何创建一个具有一个字符串数组的可观察的列表？
 16.28 如何设置一个列表视图的方向？
 16.29 列表视图有什么可用的选择模式？什么是默认的选择模式？如何设置一个选择模式？
 16.30 如何获得选中的条目以及选中的下标？

16.10 滚动条

要点提示：滚动条 (ScrollBar) 是一个允许用户从一个范围内的值中进行选择的组件。

图 16-21 显示了一个滚动条。通常，用户通过鼠标操作改变滚动条的值。例如，用户可以上下拖动滚动块，或者单击滚动条轨道，或者单击滚动条的左按钮或者右按钮。

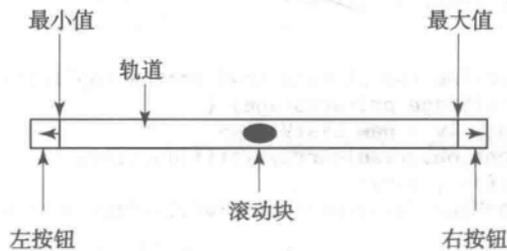


图 16-21 一个滚动条图形化的代表了一个范围内的值

ScrollBar 有以下属性，如图 16-22 所示。

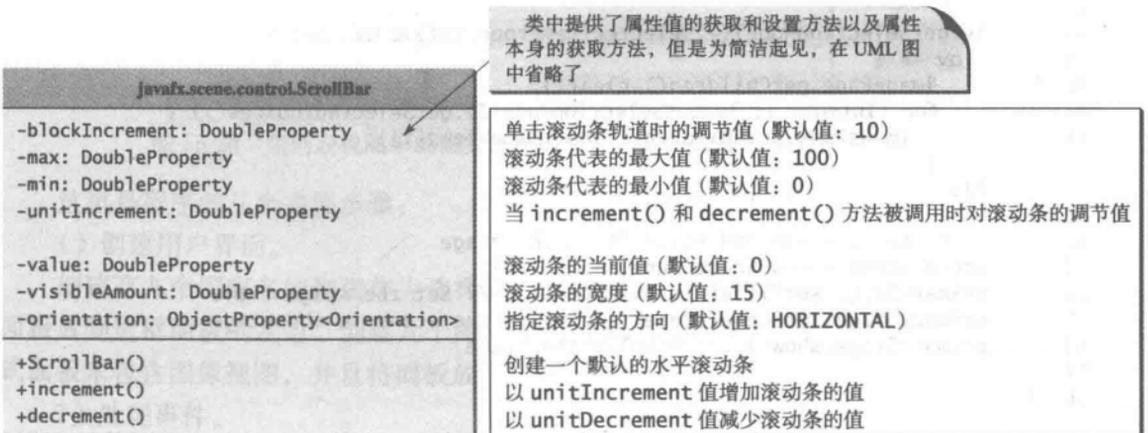


图 16-22 ScrollBar 使得用户可以从一个范围内的值中进行选择

注意：滚动条的轨道宽度对应于 `max + visibleAmount`。当一个滚动条设置为它的最大值时，块的左侧位于 `max`，右侧位于 `max + visibleAmount`。

当用户改变滚动条的值时，它通知监听器这个改变。可以在滚动条的 `valueProperty` 上面注册一个监听器来对这个改变进行反应，如下所示：

```
ScrollBar sb = new ScrollBar();
sb.valueProperty().addListener(ov -> {
    System.out.println("old value: " + oldVal);
    System.out.println("new value: " + newVal);
});
```

程序清单 16-10 给出一个程序，使用水平滚动条和垂直滚动条来控制面板显示的一个文本。水平滚动条用于左右移动消息，而垂直滚动条用于上下移动消息。程序的运行示例如图 16-23 所示。

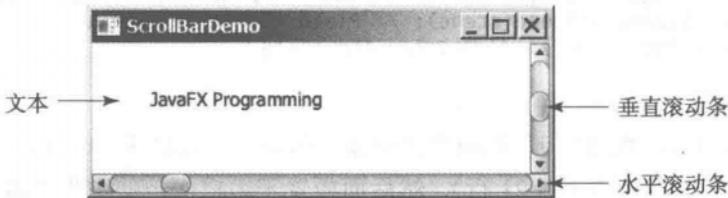


图 16-23 滚动条在面板上水平和垂直移动文本

下面是程序的主要步骤：

1) 创建用户界面。

创建一个 `Text` 对象，将它放置于边框面板的中央。创建一个垂直滚动条，将它放到边框面板的右边。创建一个水平滚动条，将它放到边框面板的底部。

2) 处理事件。

创建一个监听器，当滚动条中的滑块由于 `value` 属性的改变而产生移动时，监听器方法相应移动文本。

程序清单 16-10 ScrollBarDemo.java

```
1 import javafx.application.Application;
2 import javafx.stage.Stage;
3 import javafx.geometry.Orientation;
4 import javafx.scene.Scene;
5 import javafx.scene.control.ScrollBar;
6 import javafx.scene.layout.BorderPane;
7 import javafx.scene.layout.Pane;
8 import javafx.scene.text.Text;
9
10 public class ScrollBarDemo extends Application {
11     @Override // Override the start method in the Application class
12     public void start(Stage primaryStage) {
13         Text text = new Text(20, 20, "JavaFX Programming");
14
15         ScrollBar sbHorizontal = new ScrollBar();
16         ScrollBar sbVertical = new ScrollBar();
17         sbVertical.setOrientation(Orientation.VERTICAL);
18
19         // Create a text in a pane
20         Pane paneForText = new Pane();
21         paneForText.getChildren().add(text);
22
23         // Create a border pane to hold text and scroll bars
```

```

24     BorderPane pane = new BorderPane();
25     pane.setCenter(paneForText);
26     pane.setBottom(sbHorizontal);
27     pane.setRight(sbVertical);
28
29     // Listener for horizontal scroll bar value change
30     sbHorizontal.valueProperty().addListener(ov ->
31         text.setX(sbHorizontal.getValue() * paneForText.getWidth() /
32             sbHorizontal.getMax()));
33
34     // Listener for vertical scroll bar value change
35     sbVertical.valueProperty().addListener(ov ->
36         text.setY(sbVertical.getValue() * paneForText.getHeight() /
37             sbVertical.getMax()));
38
39     // Create a scene and place it in the stage
40     Scene scene = new Scene(pane, 450, 170);
41     primaryStage.setTitle("ScrollBarDemo"); // Set the stage title
42     primaryStage.setScene(scene); // Place the scene in the stage
43     primaryStage.show(); // Display the stage
44 }
45 }

```

程序创建一段文本（第 13 行）和两个滚动条（sbHorizontal 和 sbVertical）（第 15 ~ 16 行）。将文本放在一个面板中（第 21 行），然后面板置于边框面板的中央（第 25 行）。如果文本直接放在边框面板中央，不能通过重设它的 x 和 y 属性改变文本的位置。将 sbHorizontal 和 sbVertical 分别放置在边框面板的右侧和底部（第 26 ~ 27 行）。

可以指定滚动条的属性值。默认的，max 的属性值是 100，min 是 0，blockIncrement 是 10，visibleAmount 是 15。

注册一个监听器用于监听 sbHorizontal value 属性的改变（第 30 ~ 32 行）。当滚动条的值改变时，监听器得到通知，调用处理器根据 sbHorizontal 的当前值为文本设置新的 x 值（第 31 ~ 32 行）。

注册一个监听器用于监听 sbVertical value 属性的改变（第 35 ~ 37 行）。当滚动条的值改变时，监听器得到通知，调用处理器根据 sbVertical 的当前值为文本设置新的 y 值（第 36 ~ 37 行）。

作为一种选择，可以使用绑定属性将 30 ~ 37 行代码替换成如下所示：

```

text.xProperty().bind(sbHorizontal.valueProperty().
    multiply(paneForText.widthProperty()).
    divide(sbHorizontal.maxProperty()));

text.yProperty().bind(sbVertical.valueProperty().multiply(
    paneForText.heightProperty().divide(
    sbVertical.maxProperty())));

```

🔪 复习题

- 16.31 如何创建一个水平滚动条？如何创建一个垂直滚动条？
- 16.32 如何编写代码，用以响应滚动条的 value 属性的改变？
- 16.33 如何从滚动条获得值？如何从滚动条获得最大值？

16.11 滑动条

🔪 要点提示：Slider 与 ScrollBar 类似，但是 Slider 具有更多的属性，并且可以以多种形式显示。

图 16-24 显示两个滑动条。Slider 允许用户通过在一个有界的区间中滑动滑块，从而以图形方式选择一个值。滑动条可以显示区间中的主刻度以及次刻度。刻度之间的像素值是由 majorTickUnit 和 minorTickUnit 属性指定的。滑块可以水平显示也可以垂直显示，可以带刻度也可以不带刻度，可以有标签也可以没有。

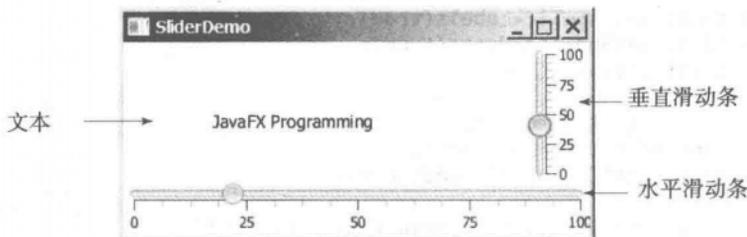


图 16-24 滑动条在面板上水平和垂直地移动文本

Slider 中经常使用的构造方法和属性如图 16-25 所示。

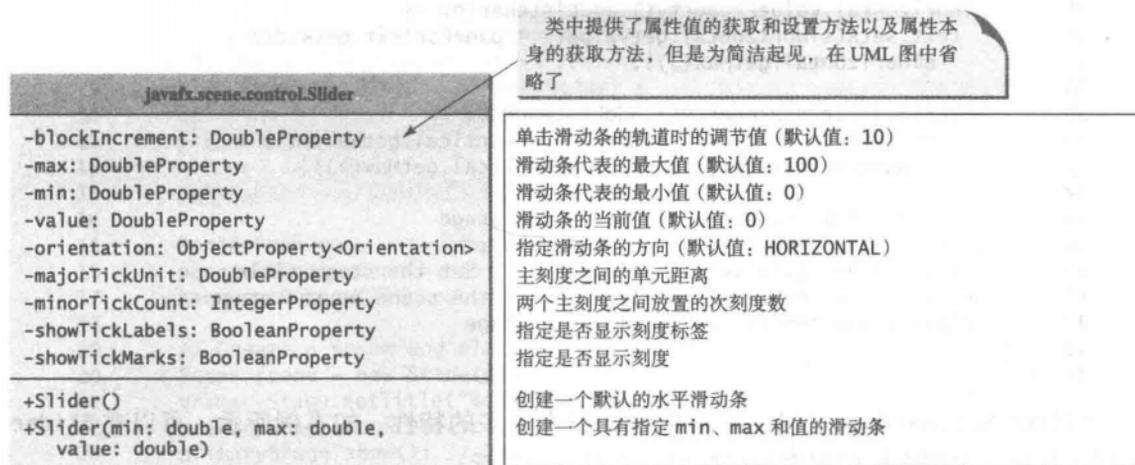


图 16-25 Slider 让用户可以在一个范围内的值中进行选择

注意：垂直滚动条的值从上向下是增加的，但垂直滑动条的值从上向下是减少的。

可以为滑动条中 value 属性值的改变添加一个监听器，与在滚动条中采用同样的方式。现在我们使用滑动条重写前面一节的程序，用于移动显示在面板中的一段文本，如程序清单 16-11 所示。程序的一个运行示例如图 16-24 所示。

程序清单 16-11 SliderDemo.java

```

1 import javafx.application.Application;
2 import javafx.stage.Stage;
3 import javafx.geometry.Orientation;
4 import javafx.scene.Scene;
5 import javafx.scene.control.Slider;
6 import javafx.scene.layout.BorderPane;
7 import javafx.scene.layout.Pane;
8 import javafx.scene.text.Text;
9
10 public class SliderDemo extends Application {
11     @Override // Override the start method in the Application class
12     public void start(Stage primaryStage) {
13         Text text = new Text(20, 20, "JavaFX Programming");

```

```

14
15     Slider s1Horizontal = new Slider();
16     s1Horizontal.setShowTickLabels(true);
17     s1Horizontal.setShowTickMarks(true);
18
19     Slider s1Vertical = new Slider();
20     s1Vertical.setOrientation(Orientation.VERTICAL);
21     s1Vertical.setShowTickLabels(true);
22     s1Vertical.setShowTickMarks(true);
23     s1Vertical.setValue(100);
24
25     // Create a text in a pane
26     Pane paneForText = new Pane();
27     paneForText.getChildren().add(text);
28
29     // Create a border pane to hold text and scroll bars
30     BorderPane pane = new BorderPane();
31     pane.setCenter(paneForText);
32     pane.setBottom(s1Horizontal);
33     pane.setRight(s1Vertical);
34
35     s1Horizontal.valueProperty().addListener(ov ->
36         text.setX(s1Horizontal.getValue() * paneForText.getWidth() /
37             s1Horizontal.getMax()));
38
39     s1Vertical.valueProperty().addListener(ov ->
40         text.setY((s1Vertical.getMax() - s1Vertical.getValue())
41             * paneForText.getHeight() / s1Vertical.getMax()));
42
43     // Create a scene and place it in the stage
44     Scene scene = new Scene(pane, 450, 170);
45     primaryStage.setTitle("SliderDemo"); // Set the stage title
46     primaryStage.setScene(scene); // Place the scene in the stage
47     primaryStage.show(); // Display the stage
48 }
49 }

```

Slider 与 ScrollBar 类似，但是 Slider 具有更多的特性。如本例所示，可以在 Slider 上指定标签、主刻度标记和次刻度标记（第 16 ~ 17 行）。

注册一个监听器用于监听 s1Horizontal value 属性的改变（第 35 ~ 37 行），注册另外一个用于监听 s1Vertical value 属性的改变（第 39 ~ 41 行）。当滑动条的值改变时，监听器得到通知，调用处理器为文本设置一个新的位置（第 36 ~ 37，40 ~ 41 行）。请注意，由于一个垂直滑动条的值从上到下是递减的，文本的对应 y 值做相应调整。

可以使用绑定属性将 35 ~ 41 行代码替换成如下所示：

```

text.xProperty().bind(s1Horizontal.valueProperty().
    multiply(paneForText.widthProperty()).
    divide(s1Horizontal.maxProperty()));

text.yProperty().bind((s1Vertical.maxProperty().subtract(
    s1Vertical.valueProperty()).multiply(
    paneForText.heightProperty().divide(
    s1Vertical.maxProperty())));

```

程序清单 15-17 给出一个显示弹球的程序。可以加入一个滑动条以控制球的移动速度，如图 16-26 所示。新的程序在程序清单 16-12 中给出。

程序清单 15-17 中定义的 BallPane 类生成一个球在面板中弹动的动画。BallPane 的 rateProperty() 方法返回一个动画速度的属性值。如果速度为 0，动画停止；如果速度高于 20，动画将过快。所以，我们特意将速度设置为一个 0 和 20 之间的值。这个值绑定到滑动

条值上 (第 13 行)。因此滑动条的最大值设置为 20 (第 12 行)。

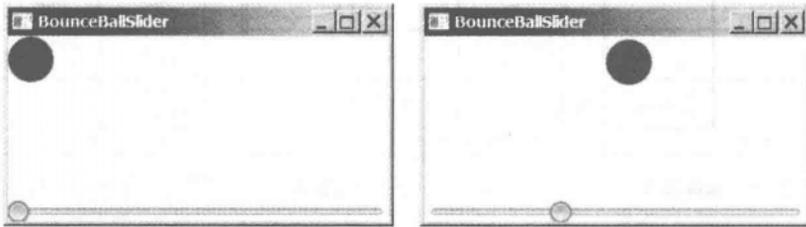


图 16-26 可以使用滑动条来增加或者降低球的速度

程序清单 16-12 BounceBallSlider.java

```

1  import javafx.application.Application;
2  import javafx.stage.Stage;
3  import javafx.scene.Scene;
4  import javafx.scene.control.Slider;
5  import javafx.scene.layout.BorderPane;
6
7  public class BounceBallSlider extends Application {
8      @Override // Override the start method in the Application class
9      public void start(Stage primaryStage) {
10         BallPane ballPane = new BallPane();
11         Slider s1Speed = new Slider();
12         s1Speed.setMax(20);
13         ballPane.rateProperty().bind(s1Speed.valueProperty());
14
15         BorderPane pane = new BorderPane();
16         pane.setCenter(ballPane);
17         pane.setBottom(s1Speed);
18
19         // Create a scene and place it in the stage
20         Scene scene = new Scene(pane, 250, 250);
21         primaryStage.setTitle("BounceBallSlider"); // Set the stage title
22         primaryStage.setScene(scene); // Place the scene in the stage
23         primaryStage.show(); // Display the stage
24     }
25 }

```

复习题

- 16.34 如何创建一个水平滑动条？如何创建一个垂直滑动条？
- 16.35 如何添加一个监听器用于处理滑动条的属性值改变？
- 16.36 如何获得滑动条上的值？如何获得滑动条上的最大值？

16.12 示例学习：开发一个井字游戏

要点提示：本节开发一个程序用于玩井字游戏 (Tic-Tac-Toe)。

从本章和前面各章的例子中我们已经学习了对象、类、数组、类的继承、GUI、事件驱动编程。现在到了应用所学知识开发综合项目的时候了。本节将开发一个流行的井字游戏的 JavaFX 程序。

在井字游戏中，两个玩家在一个 3×3 的网格中轮流将各自的标记填在空格中（一个人用 X，另一个人用 O）。如果一个玩家在网格的水平方向、垂直方向或对角线方向上放了三个连续标记，游戏就以这个玩家得胜而告终。若网格的所有单元格都填满了标记还没有产生胜者，就会出现平局（没有胜者）。图 16-27 是这个例子的典型运行示例。

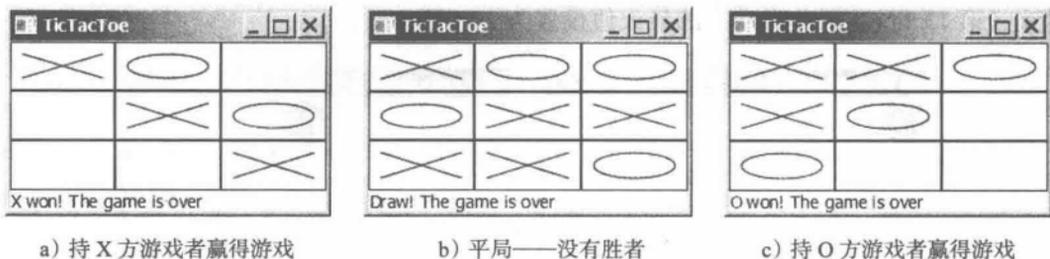
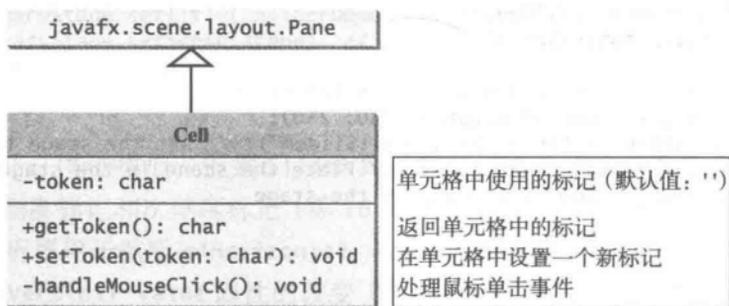


图 16-27 两个玩家玩井字游戏

至今为止，我们见过的所有例子的行为都很简单，容易用类来建模。但是井字游戏的行为有些复杂。为了定义对游戏的行为建模的类，需要研究和了解这个游戏。

假设开始时所有的单元格都是空的，并且第一个玩家用 X 标记，第二个玩家用 O 标记。要在单元格上做标记，玩家应该将鼠标指针放在这个单元格上，然后单击它。如果这个单元格为空，就显示标记 (X 或 O)。如果这个单元格已经被填充，则忽略这个玩家的动作。

从前面的描述可以知道，单元格显然是处理鼠标单击事件和显示标记的 GUI 对象。对于构建这个对象而言，有许多选择。我们将使用一个面板来对单元格建模并显示一个标记 (X 或者 O)。如何获知单元格的状态 (空、X 或 O) 呢？可以使用单元格类 `Cell` 中命名为 `token` 的 `char` 类型属性来解决这个问题。`Cell` 类负责空单元格被单击时绘制标记。因此，需要编写代码来监听鼠标单击动作，以及绘制标记 X 和 O 的形状。`Cell` 类可以定义为如图 16-28 所示。

图 16-28 `Cell` 类在单元格中显示标记

井字棋盘由 9 个单元格组成，使用 `new Cell[3][3]` 创建。为了判断轮到哪个玩家出棋，可以引入名为 `whoseTurn` 的 `char` 型变量，该变量的初始值为 'X'，然后变为 'O'；接下来，每当填充新单元格，它就在 'X' 和 'O' 之间依次转换。当游戏结束时，`whoseTurn` 设置为 ' ' (空)。

如何才能知道这场游戏是否结束，是否产生了胜者？如果有胜者，那么谁是胜者？可以创建一个名为 `isWon(char token)` 的方法来判断指定标记是否是胜者，并且创建一个名为 `isFull()` 的方法来判断是否所有的单元格都被占满。

显然，前面的分析中出现两个类。一个是处理单个单元格上操作的 `Cell` 类；另一个是玩整个游戏并处理所有单元格的 `TicTacToe` 类。这两个类之间的关系如图 16-29 所示。

因为 `Cell` 类只用于支持 `TicTacToe` 类，所以，它可以定义为 `TicTacToe` 类的一个内部类。完整的程序在程序清单 16-13 中给出。

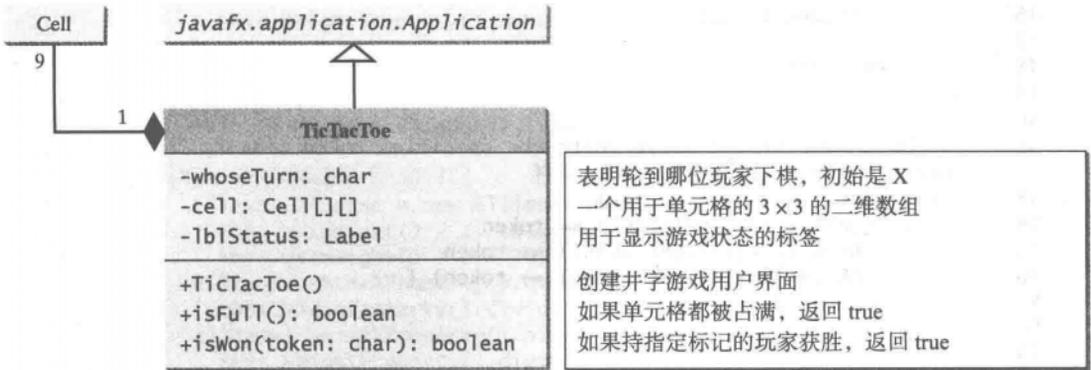


图 16-29 TicTacToe 类包含 9 个单元格

程序清单 16-13 TicTacToe.java

```

1  import javafx.application.Application;
2  import javafx.stage.Stage;
3  import javafx.scene.Scene;
4  import javafx.scene.control.Label;
5  import javafx.scene.layout.BorderPane;
6  import javafx.scene.layout.GridPane;
7  import javafx.scene.layout.Pane;
8  import javafx.scene.paint.Color;
9  import javafx.scene.shape.Line;
10 import javafx.scene.shape.Ellipse;
11
12 public class TicTacToe extends Application {
13     // Indicate which player has a turn, initially it is the X player
14     private char whoseTurn = 'X';
15
16     // Create and initialize cell
17     private Cell[][] cell = new Cell[3][3];
18
19     // Create and initialize a status label
20     private Label lblStatus = new Label("X's turn to play");
21
22     @Override // Override the start method in the Application class
23     public void start(Stage primaryStage) {
24         // Pane to hold cell
25         GridPane pane = new GridPane();
26         for (int i = 0; i < 3; i++)
27             for (int j = 0; j < 3; j++)
28                 pane.add(cell[i][j] = new Cell(), j, i);
29
30         BorderPane borderPane = new BorderPane();
31         borderPane.setCenter(pane);
32         borderPane.setBottom(lblStatus);
33
34         // Create a scene and place it in the stage
35         Scene scene = new Scene(borderPane, 450, 170);
36         primaryStage.setTitle("TicTacToe"); // Set the stage title
37         primaryStage.setScene(scene); // Place the scene in the stage
38         primaryStage.show(); // Display the stage
39     }
40
41     /** Determine if the cell are all occupied */
42     public boolean isFull() {
43         for (int i = 0; i < 3; i++)
44             for (int j = 0; j < 3; j++)
45                 if (cell[i][j].getToken() == ' ')
  
```

```

46         return false;
47
48     return true;
49 }
50
51 /** Determine if the player with the specified token wins */
52 public boolean isWon(char token) {
53     for (int i = 0; i < 3; i++)
54         if (cell[i][0].getToken() == token
55             && cell[i][1].getToken() == token
56             && cell[i][2].getToken() == token) {
57             return true;
58         }
59
60     for (int j = 0; j < 3; j++)
61         if (cell[0][j].getToken() == token
62             && cell[1][j].getToken() == token
63             && cell[2][j].getToken() == token) {
64             return true;
65         }
66
67     if (cell[0][0].getToken() == token
68         && cell[1][1].getToken() == token
69         && cell[2][2].getToken() == token) {
70         return true;
71     }
72
73     if (cell[0][2].getToken() == token
74         && cell[1][1].getToken() == token
75         && cell[2][0].getToken() == token) {
76         return true;
77     }
78
79     return false;
80 }
81
82 // An inner class for a cell
83 public class Cell extends Pane {
84     // Token used for this cell
85     private char token = ' ';
86
87     public Cell() {
88         setStyle("-fx-border-color: black");
89         this.setPrefSize(2000, 2000);
90         this.setOnMouseClicked(e -> handleMouseClicked());
91     }
92
93     /** Return token */
94     public char getToken() {
95         return token;
96     }
97
98     /** Set a new token */
99     public void setToken(char c) {
100         token = c;
101
102         if (token == 'X') {
103             Line line1 = new Line(10, 10,
104                 this.getWidth() - 10, this.getHeight() - 10);
105             line1.endXProperty().bind(this.widthProperty().subtract(10));
106             line1.endYProperty().bind(this.heightProperty().subtract(10));
107             Line line2 = new Line(10, this.getHeight() - 10,
108                 this.getWidth() - 10, 10);
109             line2.startYProperty().bind(

```

```

110     this.heightProperty().subtract(10));
111     line2.endXProperty().bind(this.widthProperty().subtract(10));
112
113     // Add the lines to the pane
114     this.getChildren().addAll(line1, line2);
115 }
116 else if (token == 'O') {
117     Ellipse ellipse = new Ellipse(this.getWidth() / 2,
118     this.getHeight() / 2, this.getWidth() / 2 - 10,
119     this.getHeight() / 2 - 10);
120     ellipse.centerXProperty().bind(
121     this.widthProperty().divide(2));
122     ellipse.centerYProperty().bind(
123     this.heightProperty().divide(2));
124     ellipse.radiusXProperty().bind(
125     this.widthProperty().divide(2).subtract(10));
126     ellipse.radiusYProperty().bind(
127     this.heightProperty().divide(2).subtract(10));
128     ellipse.setStroke(Color.BLACK);
129     ellipse.setFill(Color.WHITE);
130
131     getChildren().add(ellipse); // Add the ellipse to the pane
132 }
133 }
134
135 /* Handle a mouse click event */
136 private void handleMouseClicked() {
137     // If cell is empty and game is not over
138     if (token == ' ' && whoseTurn != ' ') {
139         setToken(whoseTurn); // Set token in the cell
140
141         // Check game status
142         if (isWon(whoseTurn)) {
143             lblStatus.setText(whoseTurn + " won! The game is over");
144             whoseTurn = ' '; // Game is over
145         }
146         else if (isFull()) {
147             lblStatus.setText("Draw! The game is over");
148             whoseTurn = ' '; // Game is over
149         }
150         else {
151             // Change the turn
152             whoseTurn = (whoseTurn == 'X') ? 'O' : 'X';
153             // Display whose turn
154             lblStatus.setText(whoseTurn + "'s turn");
155         }
156     }
157 }
158 }
159 }

```

TicTacToe 类通过将 9 个单元格放置在一个网格面板上来初始化用户界面 (第 25 ~ 28 行)。一个命名为 `lblStatus` 的标签用来显示游戏的状态 (第 20 行)。变量 `whoseTurn` (第 14 行) 用来跟踪下一个要放在单元格中的标记的类型。`isFull` 方法 (第 42 ~ 49 行) 和 `isWon` 方法 (第 52 ~ 80 行) 用来判断这个游戏的状态。

由于 `Cell` 类是 `TicTacToe` 类中的内部类, 所以, 可以在 `Cell` 类中引用 `TicTacToe` 类中定义的变量 (`whoseTurn`) 和方法 (`isFull` 和 `isWon`)。内部类可以使程序简洁明了。如果没有把 `Cell` 类定义为 `TicTacToe` 的内部类, 为了可以在 `Cell` 中使用 `TicTacToe` 中的变量和方法, 就必须给 `Cell` 传递一个 `TicTacToe` 对象。

为单元格注册用于监听鼠标单击动作的监听器 (第 90 行)。如果游戏没有结束时单击空

单元格，那么在单元格中会设置一个标记（第 138 行）。如果游戏结束，whoseTurn 设置为 ''（空）（第 144 行和第 148 行）。否则，whoseTurn 被轮流设置为新的下棋方（第 152 行）。

提示：采用渐进的方法开发和测试这一类 Java 项目。例如，这个程序可以分解为五个步骤：

- 1) 对用户界面布局，然后在单元格中显示一个固定标记 X。
- 2) 使单元格能够响应鼠标单击以显示固定标记 X。
- 3) 在两个玩家间协调，以便交替地显示标记 X 和 O。
- 4) 判断是否有玩家获胜，或者是否所有的单元格都被占满且仍无获胜者。
- 5) 对于一个玩家下的每一步棋，实现在标签上显示一条消息。

复习题

- 16.37 游戏开始时，whoseTurn 中的值是什么？游戏结束时候，whoseTurn 中的值是什么？
- 16.38 如果游戏尚未结束，当用户在一个空单元格上单击时，将发生什么？如果游戏已经结束，当用户在一个空单元格上单击时，又将发生什么？
- 16.39 程序如何判断是否已经有玩家获胜？程序如何判断是否所有的单元格都被填充？

16.13 视频和音频

要点提示：可以使用 Media 类来获得媒体源，使用 MediaPlayer 类来播放和控制媒体，使用 MediaView 来显示视频。

媒体（视频和音频）对于开发富因特网应用是必要的。JavaFX 提供了 Media、MediaPlayer 和 MediaView 类用于媒体编程。目前，JavaFX 支持 MP3、AIFF、WAV 以及 MPEG-4 音频格式，以及 FLV 和 MPEG-4 视频格式。

Media 类代表了一个媒体源，具有属性 duration、width 以及 height，如图 16-30 所示。可以从一个 Internet URL 字符串中构建一个 Media 对象。

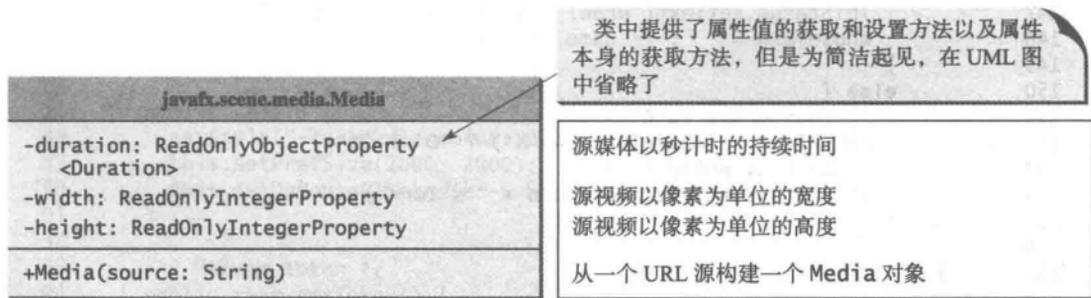
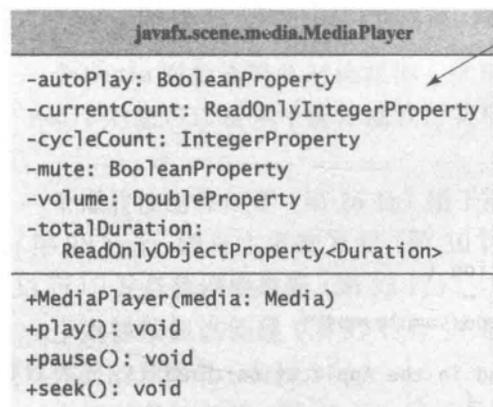


图 16-30 Media 代表了一个媒体源，如一段视频或者音频

MediaPlayer 类播放媒体，并通过一些属性来控制媒体播放，比如 autoPlay、currentCount、cycleCount、mute、volume 和 totalDuration，如图 16-31 所示。可以从一个媒体对象来构建一个 MediaPlayer 对象，并使用 pause() 和 play() 方法来暂停和继续播放。

MediaView 类是 Node 的子类，提供了 MediaPlayer 播放的 Media 的视图。MediaView 类提供了一些属性用于观看媒体，如图 16-32 所示。

程序清单 16-14 给出了一个示例，在一个视图中播放一个视频，如图 16-33 所示。可以通过使用播放 / 暂停按钮来播放 / 暂停视频，使用重播按钮来重新播放视频，使用滑动条来控制音量。

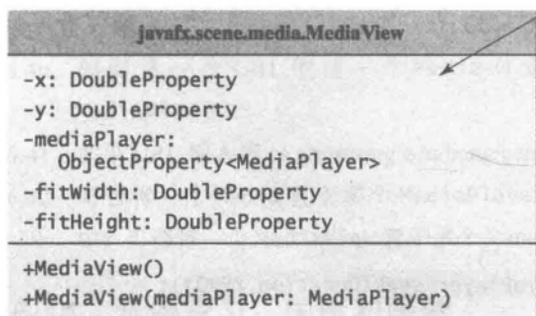


类中提供了属性值的获取和设置方法以及属性本身的获取方法，但是为简洁起见，在 UML 图中省略了

指定一个播放是否应该自动开始
已经完成的循环重放数
指定媒体播放的次数
指定音频是否静音
音频的音量
从开始到结束播放媒体的持续时间

为指定媒体创建一个播放器
播放媒体
暂停媒体播放
将播放器定位到一个新的重新播放时间点

图 16-31 MediaPlayer 播放和控制一个媒体



类中提供了属性值的获取和设置方法以及属性本身的获取方法，但是为简洁起见，在 UML 图中省略了

指定媒体视图的当前 x 坐标
指定媒体视图的当前 y 坐标
为媒体视图指定一个媒体播放器

为媒体指定一个适合的视图宽度
为媒体指定一个适合的视图高度

构建一个空的媒体视图
构建一个具有指定媒体播放器的媒体视图

图 16-32 MediaView 提供观看媒体的属性



图 16-33 程序控制和播放一个视频

程序清单 16-14 MediaDemo.java

```

1 import javafx.application.Application;
2 import javafx.stage.Stage;
3 import javafx.geometry.Pos;
  
```

```

4 import javafx.scene.Scene;
5 import javafx.scene.control.Button;
6 import javafx.scene.control.Label;
7 import javafx.scene.control.Slider;
8 import javafx.scene.layout.BorderPane;
9 import javafx.scene.layout.HBox;
10 import javafx.scene.layout.Region;
11 import javafx.scene.media.Media;
12 import javafx.scene.media.MediaPlayer;
13 import javafx.scene.media.MediaView;
14 import javafx.util.Duration;
15
16 public class MediaDemo extends Application {
17     private static final String MEDIA_URL =
18         "http://cs.armstrong.edu/liang/common/sample.mp4";
19
20     @Override // Override the start method in the Application class
21     public void start(Stage primaryStage) {
22         Media media = new Media(MEDIA_URL);
23         MediaPlayer mediaPlayer = new MediaPlayer(media);
24         MediaView mediaView = new MediaView(mediaPlayer);
25
26         Button playButton = new Button(">");
27         playButton.setOnAction(e -> {
28             if (playButton.getText().equals(">")) {
29                 mediaPlayer.play();
30                 playButton.setText("||");
31             } else {
32                 mediaPlayer.pause();
33                 playButton.setText(">");
34             }
35         });
36
37         Button rewindButton = new Button("<<");
38         rewindButton.setOnAction(e -> mediaPlayer.seek(Duration.ZERO));
39
40         Slider s1Volume = new Slider();
41         s1Volume.setPrefWidth(150);
42         s1Volume.setMaxWidth(Region.USE_PREF_SIZE);
43         s1Volume.setMinWidth(30);
44         s1Volume.setValue(50);
45         mediaPlayer.volumeProperty().bind(
46             s1Volume.valueProperty().divide(100));
47
48         HBox hBox = new HBox(10);
49         hBox.setAlignment(Pos.CENTER);
50         hBox.getChildren().addAll(playButton, rewindButton,
51             new Label("Volume"), s1Volume);
52
53         BorderPane pane = new BorderPane();
54         pane.setCenter(mediaView);
55         pane.setBottom(hBox);
56
57         // Create a scene and place it in the stage
58         Scene scene = new Scene(pane, 650, 500);
59         primaryStage.setTitle("MediaDemo"); // Set the stage title
60         primaryStage.setScene(scene); // Place the scene in the stage
61         primaryStage.show(); // Display the stage
62     }
63 }

```

媒体源是一个 URL 字符串，在 17 和 18 行定义。程序从这个 URL 创建一个 Media 对象（第 22 行），从 Media 对象创建一个 MediaPlayer 对象（第 23 行），并从 MediaPlayer 对象创建一个 MediaView（第 24 行）。这三个对象之间的关系如图 16-34 所示。

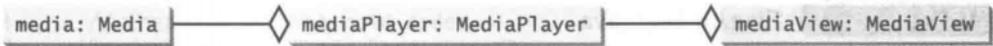


图 16-34 媒体代表了播放源，媒体播放器控制播放，媒体视图显示视频

一个 `Media` 对象支持实时流媒体。你现在可以下载一个大的媒体文件并且同时播放它。一个 `Media` 对象可以被多个媒体播放器共享，并且不同的视图可以使用同一个 `MediaPlayer` 对象。

一个播放按钮被创建（第 26 行）用于播放 / 暂停媒体（第 29 行）。如果按钮当前的文字是 `>`（第 28 行），则将文字改为 `||`（第 30 行）。如果按钮当前的文字是 `||`，则将文字改为 `>`（第 33 行），并且暂停播放器（第 32 行）。

一个重播按钮被创建（第 37 行），并通过调用 `seek(Duration.ZERO)` 以重设再次播放时间到媒体流的开始处（38 行）。

一个滑动条被创建（第 40 行）用于设置音量。媒体播放器的音量属性绑定到滑动条上（第 45 和 46 行）。

将按钮和滑动条置于一个 `HBox` 中（第 48 ~ 51 行），将媒体视图置于边框面板中央（第 54 行），并将 `HBox` 放置在边框面板的底部（第 55 行）。

复习题

- 16.40 如何从一个 URL 创建一个 `Media` 对象？如何创建一个 `MediaPlayer`？如何创建一个 `MediaView`？
- 16.41 如果 URL 输入成 `cs.armstrong.edu/liang/common/sample.mp4`，前面不包含 `http://`，可以运行吗？
- 16.42 可否将一个 `Media` 置于多个 `MediaPlayer` 中？可否将一个 `MediaPlayer` 置于多个 `MediaView` 中？可否将一个 `MediaView` 置于多个 `Pane` 中？

16.14 示例学习：国旗和国歌

要点提示：本示例学习给出一个程序，用来显示一个国家的国旗以及播放国歌。

七个名为 `flag0.gif`，...，`flag6.gif` 的图像分别是丹麦、德国、中国、印度、挪威、英国和美国七个国家的国旗。它们保存在 `www.cs.armstrong.edu/liang/common/image` 下面。音频包括了这七个国家的国歌 `anthem0.mp3`，`anthem1.mp3`，...，`anthem6.mp3`。它们保存在 `www.cs.armstrong.edu/liang/common/audio` 下面。

程序可以让用户从组合框中选择一个国家，从而显示它的国旗并播放它的国歌。用户可以通过单击 `||` 按钮暂停音频，单击 `>` 按钮继续播放动画，如图 16-35 所示。



图 16-35 程序显示国旗并播放国歌

程序在程序清单 16-15 中给出。

程序清单 16-15 FlagAnthem.java

```

1  import javafx.application.Application;
2  import javafx.collections.FXCollections;
3  import javafx.collections.ObservableList;
4  import javafx.stage.Stage;
5  import javafx.geometry.Pos;
6  import javafx.scene.Scene;
7  import javafx.scene.control.Button;
8  import javafx.scene.control.ComboBox;
9  import javafx.scene.control.Label;
10 import javafx.scene.image.Image;
11 import javafx.scene.image.ImageView;
12 import javafx.scene.layout.BorderPane;
13 import javafx.scene.layout.HBox;
14 import javafx.scene.media.Media;
15 import javafx.scene.media.MediaPlayer;
16
17 public class FlagAnthem extends Application {
18     private final static int NUMBER_OF_NATIONS = 7;
19     private final static String URLBase =
20         "http://cs.armstrong.edu/liang/common";
21     private int currentIndex = 0;
22
23     @Override // Override the start method in the Application class
24     public void start(Stage primaryStage) {
25         Image[] images = new Image[NUMBER_OF_NATIONS];
26         MediaPlayer[] mp = new MediaPlayer[NUMBER_OF_NATIONS];
27
28         // Load images and audio
29         for (int i = 0; i < NUMBER_OF_NATIONS; i++) {
30             images[i] = new Image(URLBase + "/image/flag" + i + ".gif");
31             mp[i] = new MediaPlayer(new Media(
32                 URLBase + "/audio/anthem/anthem" + i + ".mp3"));
33         }
34
35         Button btPlayPause = new Button(">");
36         btPlayPause.setOnAction(e -> {
37             if (btPlayPause.getText().equals(">")) {
38                 btPlayPause.setText("||");
39                 mp[currentIndex].pause();
40             } else {
41                 btPlayPause.setText(">");
42                 mp[currentIndex].play();
43             }
44         });
45
46         ImageView imageView = new ImageView(images[currentIndex]);
47         ComboBox<String> cboNation = new ComboBox<>();
48         ObservableList<String> items = FXCollections.observableArrayList
49             ("Denmark", "Germany", "China", "India", "Norway", "UK", "US");
50         cboNation.getItems().addAll(items);
51         cboNation.setValue(items.get(0));
52         cboNation.setOnAction(e -> {
53             mp[currentIndex].stop();
54             currentIndex = items.indexOf(cboNation.getValue());
55             imageView.setImage(images[currentIndex]);
56             mp[currentIndex].play();
57         });
58
59         HBox hBox = new HBox(10);
60         hBox.getChildren().addAll(btPlayPause,
61             new Label("Select a nation: "), cboNation);
62         hBox.setAlignment(Pos.CENTER);
63

```

```
64 // Create a pane to hold nodes
65 BorderPane pane = new BorderPane();
66 pane.setCenter(imageView);
67 pane.setBottom(hBox);
68
69 // Create a scene and place it in the stage
70 Scene scene = new Scene(pane, 350, 270);
71 primaryStage.setTitle("FlagAnthem"); // Set the stage title
72 primaryStage.setScene(scene); // Place the scene in the stage
73 primaryStage.show(); // Display the stage
74 }
75 }
```

程序从 Internet 上载入图像和声音文件 (第 29 ~ 33 行)。创建一个播放 / 暂停按钮来控制音频的播放 (第 35 行)。当按钮被单击时, 如果按钮当前的文字是 >(第 37 行), 它的文字被改为 || (第 38 行) 并且暂停播放器 (第 39 行); 如果按钮的当前文字是 ||, 则改为 >(第 41 行) 并继续播放 (第 42 行)。

创建一个图像视图用于显示一个国旗图像 (第 46 行)。创建一个组合框用于选择一个国家 (第 47 ~ 49 行)。当组合框中一个新的国家名字被选择时, 终止当前的音频 (第 53 行), 显示最新选择国家的国旗图像 (第 55 行), 并且播放新的国歌 (第 56 行)。

JavaFX 也提供了 `AudioClip` 类用于创建音频片段。可以使用 `new AudioClip(URL)` 创建一个 `AudioClip` 对象。一个音频片段将音频保存在内存中。对于在程序中播放小段音频而言, `AudioClip` 比使用 `MediaPlayer` 更加高效。`AudioClip` 拥有和 `MediaPlayer` 类相似的方法。

☛ 复习题

- 16.43 程序清单 16-15 中, 哪行代码设置了初始图像图标, 哪行代码播放音频?
16.44 程序清单 16-15 中, 当组合框中的新的国家被选择时, 程序会做什么?

本章小结

1. 抽象类 `Labeled` 是 `Label`、`Button`、`CheckBox` 和 `RadioButton` 的基类。它定义了属性 `alignment`、`contentDisplay`、`text`、`graphic`、`graphicTextGap`、`textFill`、`underline` 和 `wrapText`。
2. 抽象类 `ButtonBase` 是 `Button`、`CheckBox` 和 `RadioButton` 的基类。它定义了 `onAction` 属性用于为动作事件指定一个处理器。
3. 抽象类 `TextInputControl` 是 `TextField` 和 `TextArea` 的基类。它定义了 `text` 和 `editable` 属性。
4. 在一个获得焦点的文本域上按回车键时, `TextField` 将触发一个动作事件。`TextArea` 通常用于编辑多行文本。
5. `ComboBox<T>` 和 `ListView<T>` 是用于保存类型 `T` 的元素的泛型类。组合框或者列表视图中的元素保存在一个可观察的列表中。
6. 当一个新的条目被选中时, `ComboBox` 触发一个动作事件。
7. 可以为 `ListView` 设置单选或者多项选择, 并添加一个监听器用于处理选中的条目。
8. 可以使用 `ScrollBar` 或者 `Slider` 用于选择一个范围内的值, 并给 `value` 属性添加一个监听器, 用于响应值的改变。
9. JavaFX 提供 `Media` 类用于载入一个媒体, 提供 `MediaPlayer` 类用于控制一个媒体, 提供 `MediaView` 用于显示一个媒体。

测试题

在线回答本章的测试题，网址是 www.cs.armstrong.edu/liang/intro10e/quiz.html。

编程练习题

16.2 ~ 16.5 节

- *16.1 (使用单选按钮) 编写一个 GUI 程序如图 16-36a 所示。可以使用按钮将消息进行左右移动，并且使用单选按钮来修改消息显示的颜色。

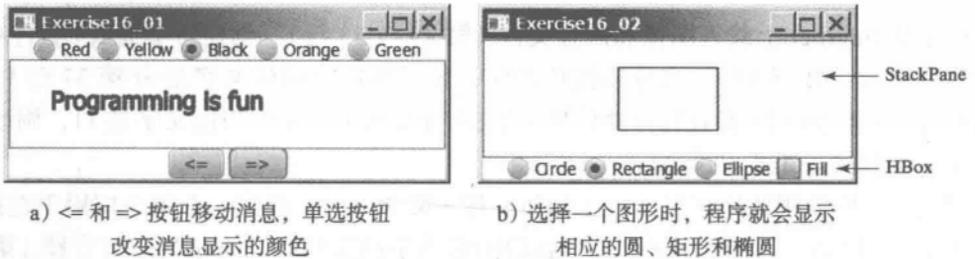


图 16-36

- *16.2 (选择几何图形) 编写一个绘制各种几何图形的程序，如图 16-36b 所示。用户从单选按钮中选择一个几何图形，并且使用复选框指定是否被填充。
- **16.3 (交通信号灯) 编写一个程序来模拟交通信号灯。程序可以让用户从红、黄、绿三种颜色灯中选择一种。当选择一个单选按钮后，相应的灯被打开，并且一次只能亮一种灯(如图 16-37a 所示)。程序开始时所有的灯都是不亮的。

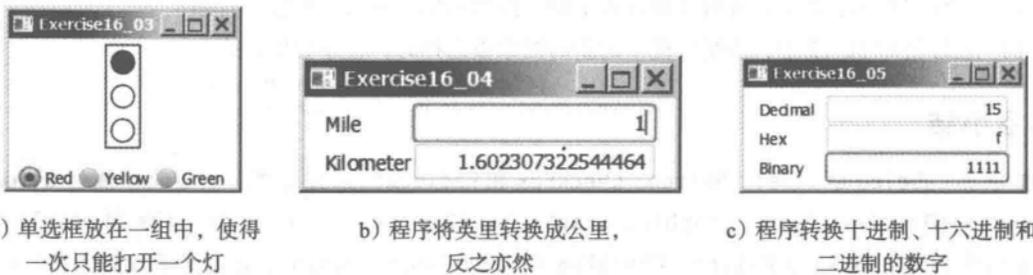
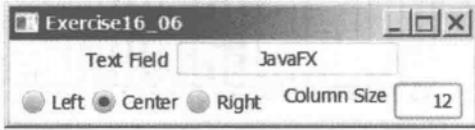


图 16-37

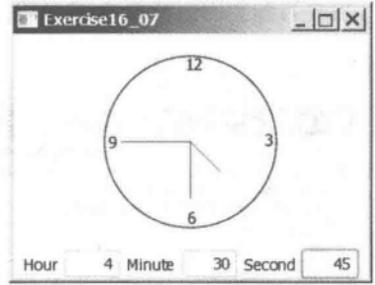
- *16.4 (创建一个英里/公里的转换器) 编写一个程序来转换英里和公里，如图 16-37b 所示。如果在英里文本域 Mile 中输入一个值之后按下回车键，就会在公里文本域 Kilometer 中显示对应的公里值。同样的，在公里文本域 Kilometer 中输入一个值之后按下回车键，就会在英里文本域 Mile 中显示对应的英里值。
- *16.5 (转换数字) 编写一个程序，在十进制、十六进制和二进制间转换数字，如图 16-37c 所示。当在十进制值的文本域中输入一个十进制值并且按回车键，会在其他两个文本域中显示相应的十六进制和二进制数字。同理，也可以在其他文本域中输入值，然后进行相应转换。

提示：使用 `Integer.parseInt(s,radix)` 方法将字符串解析成十进制数，使用 `Integer.toHexString(decimal)` 和 `Integer.toBinaryString(decimal)` 从一个十进制数字得到一个十六进制数和二进制数。

- *16.6 (演示 TextField 的属性) 编写一个程序，动态地设置文本域的水平对齐属性和列宽属性，如图 16-38a 所示。



a) 可以动态地设置文本域的水平对齐属性和列宽属性

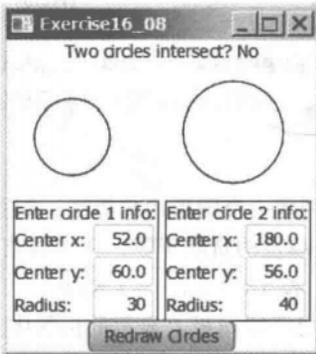


b) 程序显示文本域指定的时间

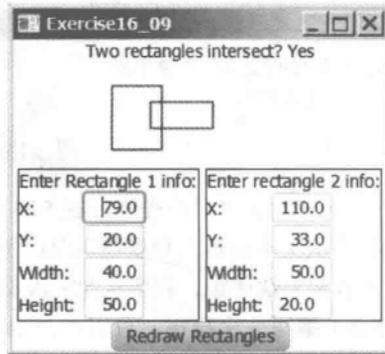
图 16-38

*16.7 (设置时钟的时间) 编写一个程序, 显示一个时钟, 并通过在三个文本域中输入小时、分钟和秒钟来设置时钟的时间, 如图 16-38b 所示。使用程序清单 14-21 中的 ClockPane 改变时钟大小使其居于面板中央。

**16.8 (几何: 两个圆相交吗?) 编写一个程序, 让用户指定两个圆的位置和大小, 并且显示两个圆是否相交, 如图 16-39a 所示。用户可以通过鼠标单击圆内部区域并且拖动圆。圆被拖动时, 文本域中的圆心坐标被更新。



a)



b)

图 16-39 检测两个圆和两个矩形是否重叠

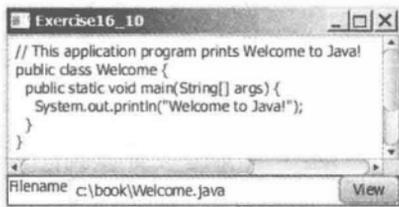
**16.9 (几何: 两个矩形相交吗?) 编写一个程序, 让用户指定两个矩形的位置和大小, 并且显示两个矩形是否相交, 如图 16-39b 所示。用户可以通过鼠标单击矩形内部区域并且拖动矩形。矩形被拖动时, 文本域中的矩形中心坐标被更新。

16.6 ~ 16.8 节

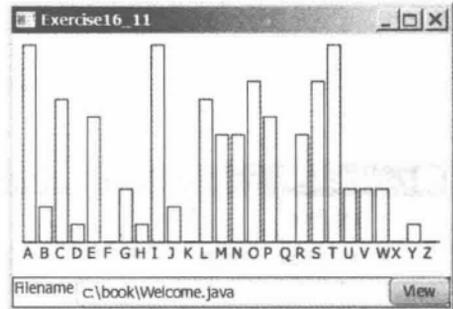
**16.10 (文本浏览器) 编写一个程序在文本区域中显示一个文本文件, 如图 16-40a 所示。用户在文本域中输入一个文件名, 然后单击 View 按钮; 在文本区域中会显示这个文件。

**16.11 (创建表示字母出现次数的直方图) 编写一个程序, 从文件中读取内容并显示一个直方图, 表示文件中每个字母出现的次数, 如图 16-40b 所示。从文本域中输入文件名。在文本域上按回车键从而程序开始读取并处理文件, 并且显示直方图。直方图在窗体中央显示。定义一个继承自 Pane 的名为 Histogram 的类。该类包含 counts 属性, 该属性是一个包含 26 个元素的数组。Counts[0] 存储 A 的出现次数, counts[1] 存储 B 的出现次数, 依此类推。类还包含一个设置方法, 用于设置一个新的 counts 并且为新的 counts 显示直方图。

*16.12 (演示 TextArea 的属性) 编写一个程序, 演示文本域的属性。程序使用复选框表明文本是否换行, 如图 16-41a 所示。

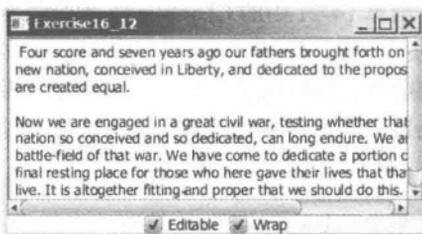


a) 程序在文本区域中显示文件文本内容



b) 程序显示一个直方图来表示文件中每个字母出现的次数

图 16-40



a) 可以设置选项以使得文本可以被编辑以及文本换行

Interest Rate	Monthly Payment	Total Payment
5.0	188.71	11322.74
5.125	189.28	11357.13
5.25	189.85	11391.59
5.375	190.43	11426.11
5.5	191.01	11460.69
5.625	191.58	11495.34
5.75	192.16	11530.06

b) 程序显示一个表格，显示给定贷款时按不同利率计算的月偿还款和总偿还款

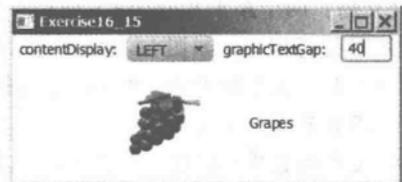
图 16-41

*16.13 (比较不同利率的贷款) 改写编程练习题 5.21, 创建一个图形用户界面, 如图 16-41b 所示。程序应该允许用户从文本域输入贷款额以及以年为单位的贷款年限, 在文本域中会显示关于每种利率的月偿还款和总偿还款, 利率从 5% 到 8%, 按 1/8 (0.125%) 递增。

**16.14 (选择一种字体) 编写一个程序, 可以动态地改变堆栈面板上显示的标签中文本的字体。这个消息可以同时以粗体和斜体显示。可以从组合框中选择字体名和字体大小, 如图 16-42a 所示。使用 `Font.getFamilies()` 可以得到可用的字体名。字体大小的组合框初始化为从 1 到 100 之间的数字。



a) 可以动态设置消息的字体



b) 可以动态地设置标签的对齐方式以及文本的位置属性

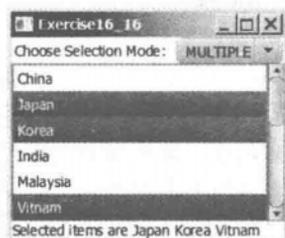
图 16-42

**16.15 (演示 Label 的属性) 编写一个程序, 允许用户动态地设置属性 `contentDisplay` 和 `graphicTextGap`, 如图 16-42b 所示。

*16.16 (使用 ComboBox 和 ListView) 编写一个程序, 演示在列表中选择的项目。程序用组合框指定选择方式, 如图 16-43a 所示。当选择项目后, 列表下方的标签中就会显示选定项。

**16.17 (使用 ScrollBar 和 Slider) 编写一个程序, 使用滚动条或者滑动条选择文本的颜色, 如图

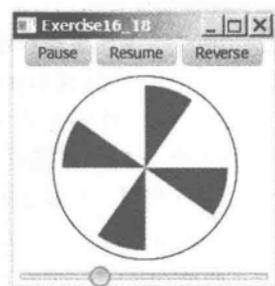
16-43b 所示。使用四个水平滚动条选择颜色（红色、绿色和蓝色），以及透明度的百分比。



a) 可以在列表中选择单项选择或者多项选择



b) 调节滚动条时改变文本的颜色



c) 程序模拟一个转动的风扇

图 16-43

**16.18 (模拟: 一个转动的风扇) 重写编程练习题 15.28, 增加一个滑动条控制风扇的速度, 如图 16-43c 所示。

**16.19 (控制一组风扇) 编写一个程序, 在一组中显示三个风扇, 用控制按钮来启动和停止整组风扇, 如图 16-44 所示。

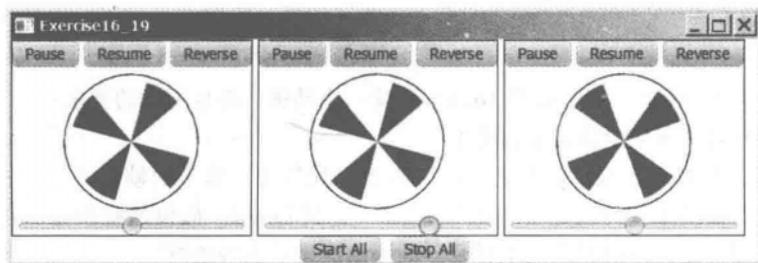
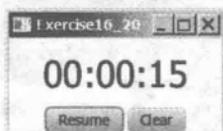
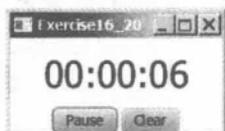
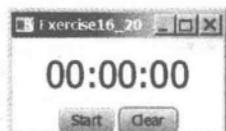
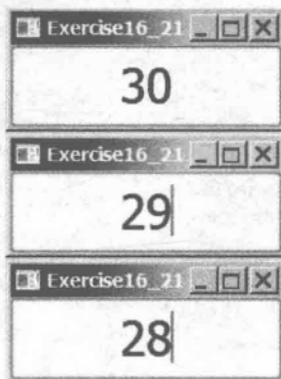


图 16-44 程序转动和控制一组风扇

*16.20 (累计秒表) 编写一个程序, 模拟一个秒表, 如图 16-45a 所示。当用户单击 Start 按钮时, 按钮的标签变为 Pause, 如图 16-45b 所示。当用户单击 Pause 按钮时, 按钮的标签变为 Resume, 如图 16-45c 所示。Clear 按钮重设计数为 0 并且重设按钮的标签为 Start。



a) ~ c) 程序累计时间



d) 程序进行时间倒计时

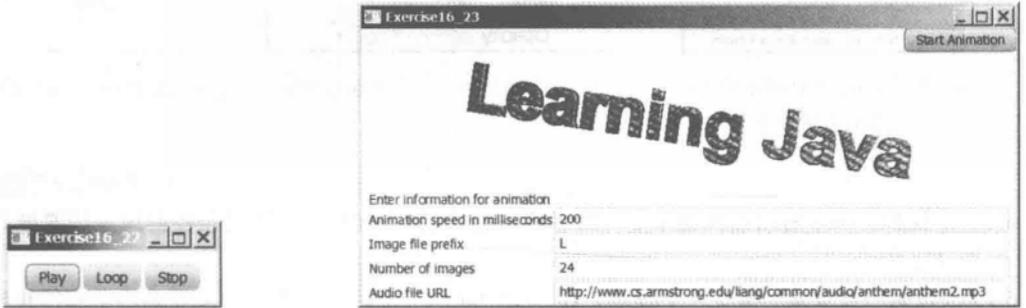
图 16-45

*16.21 (秒表倒计时) 编写一个程序, 允许用户在文本域中以秒为单位输入时间, 然后按下 Enter 键来

进行倒计时,如图 16-45d 所示。余下的秒数每秒重新显示一次。当倒计时结束时,程序开始连续播放音乐。

16.22 (播放、循环播放和停止播放一个音频剪辑)编写一个满足下面要求的程序:

- 1) 使用 `AudioClip` 获取一个音频文件,该文件存放在类目录下。
- 2) 放置三个标记为 Play、Loop 和 Stop 的按钮,如图 16-46a 所示。
- 3) 单击 Play 按钮时,会播放音频文件一次。单击 Loop 按钮时,会循环播放音频。单击 Stop 按钮时,停止播放该音频。



a) 单击 Play 播放音频剪辑一次,单击 Loop 会重复播放音频,而单击 Stop 会终止播放

b) 允许用户选择图像文件、音频文件和动画速度

图 16-46

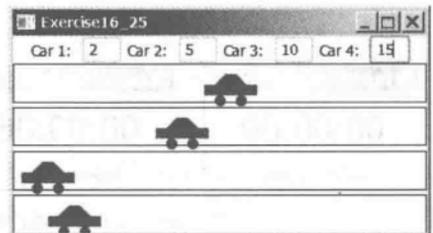
**16.23 (创建一个有声的图像动画)如图 16-46b 创建一个动画,满足下面的要求:

- 1) 允许用户在文本域中指定动画速度。
- 2) 用户输入帧数和图像文件名的前缀。例如,如果用户输入的帧数为 n ,图像文件名的前缀为 L,那么图像文件就是 L1.gif, L2.gif 一直到 Ln.gif。假设这些图像都存储在 image 目录下,该目录是程序类目录的子目录。动画依次显示这些图像。
- 3) 允许用户指定音频文件 URL,动画开始时播放这个音频。

**16.24 (修改程序清单 16-14)增加一个滑动条让用户可以为视频设置当前时间,增加一个标签显示当前时间和视频的整体时间,如图 16-47a 所示。整个时间是 5 分钟 03 秒,当前时间是 3 分 58 秒。当播放视频时,滑条值和当前时间持续更新。



a) 增加一个滑条为视频设置当前时间,增加一个标签显示当前时间和视频的整体时间



b) 设置每个汽车的速度

图 16-47

- **16.25 (赛车)** 编写一个程序，模拟四辆赛车，如图 16-47b 所示。可以对每辆赛车设置速度，用 100 表示最高速。
- **16.26 (模拟：升旗并播放国歌)** 创建一个显示升国旗的程序，如图 15-14 所示。随着国旗的升起，播放国歌（可以使用程序清单 16-15 中的国旗图像和国歌音频文件）。

综合部分

- **16.27 (显示国旗和国旗描述)** 程序清单 16-8 中给出了一个程序，让用户可以从一个组合框中选择国家，从而查看一个国家的国旗以及描述。其中描述是一个写在程序中的字符串。重写这个程序，从文件中来读取文本描述，假设这些描述保存在 text 目录下的文件 description0.txt, ..., description8.txt 中，按照顺序分别表示 9 个国家：加拿大、中国、丹麦、法国、德国、印度、挪威、英国和美国。
- **16.28 (显示幻灯片)** 编程练习题 15.30 使用图像开发了一个幻灯片显示程序。使用文本文件重写编程练习题 15.30 来开发一个幻灯片显示程序。假设十个名为 slide0.txt, slide1.txt, ..., slide9.txt 的文本文件都存储在 text 目录下。每张幻灯片显示一个文件的文本，每张幻灯片持续显示一秒。幻灯片依次显示。当显示完最后一张幻灯片时，重新显示第一张，依此类推。使用一个文本区域显示幻灯片。

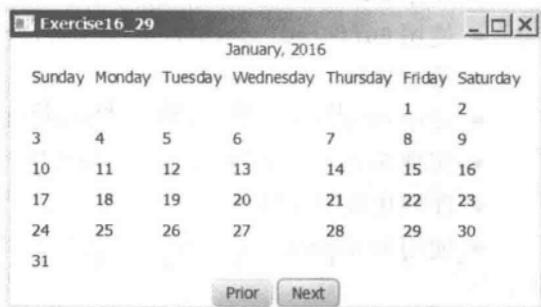
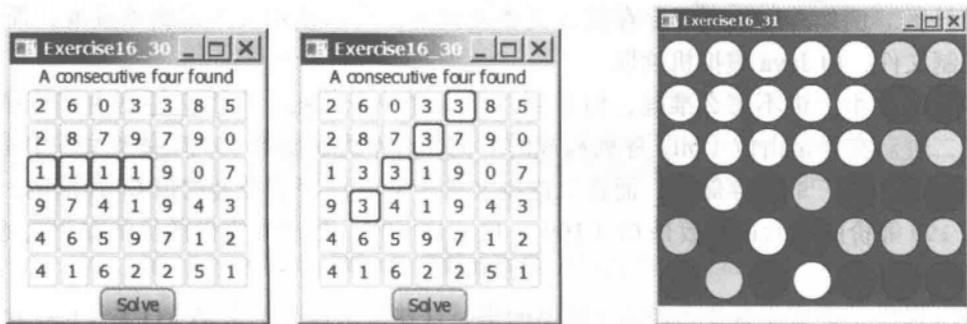


图 16-48 程序显示当月的日历

- ***16.29 (显示一个日历)** 编写一个程序，显示当前月的日历。可以使用 Prior 和 Next 按钮来显示前一个月和后一个月的日历。使用黑色字体显示当月日历中的日期，而使用灰色字体来显示前一个月和后一个月日历中的日期，如图 16-48 所示。
- **16.30 (模式识别：连续四个相同的数)** 为编程练习题 8.19 编写一个 GUI 程序，如图 16-49a ~ b 所示。让用户在 6 行 7 列的网格的文本域中输入数字。如果存在一串四个相等的数字，用户单击 Solve 按钮后，可以高亮显示它们。初始的，文本域中的值随机填充了 0 到 9 的数字。



a) ~ b) 单击 Solve 按钮高亮显示在一行、
一列或对角线上四个连续的数字

c) 程序让两个玩家玩四子连的游戏

图 16-49

- ***16.31 (游戏：四子连)** 编程练习题 8.20 让两个玩家在控制台上可以玩四子连的游戏。为这个程序重写一个 GUI 版本，如图 16-49c 所示。这个程序让两个玩家轮流放置红色和黄色棋子。为了放置棋子，玩家需要在可用的格子单击。可用的格子 (available cell) 是指不被占用的格子，而其下方临接的格子是被占用的格子。如果一个玩家胜了，这个程序就闪烁这四个赢的格子，如果所有格子都被占用但还没有胜者，就报告无胜者。

二进制 I/O

教学目标

- 了解在 Java 中如何处理 I/O (17.2 节)。
- 区分文本 I/O 与二进制 I/O 的不同 (17.3 节)。
- 使用 `FileInputStream` 和 `FileOutputStream` 来读写字节 (17.4.1 节)。
- 使用基类 `FilterInputStream` 和 `FilterOutputStream` 来过滤数据 (17.4.2 节)。
- 使用 `DataInputStream` 或 `DataOutputStream` 来读写基本类型值和字符串 (17.4.3 节)。
- 使用 `BufferedInputStream` 和 `BufferedOutputStream` 来提高 I/O 的性能 (17.4.4 节)。
- 编写复制一个文件的程序 (17.5 节)。
- 使用 `ObjectOutputStream` 和 `ObjectInputStream` 实现对象的存储与恢复 (17.6 节)。
- 实现 `Serializable` 接口使对象可序列化 (17.6.1 节)。
- 序列化数组 (17.6.2 节)。
- 使用 `RandomAccessFile` 对文件进行读写 (17.7 节)。

17.1 引言

 **要点提示:** Java 提供了许多类用于实现文本 I/O 和二进制 I/O。

文件可以分为文本或者二进制的。可以使用文本编辑器,比如 Windows 下的记事本或者 UNIX 下的 vi 编辑器,进行处理(读取、创建或者修改)的文件称为文本文件。所有的文件称为二进制文件。不能使用文本编辑器来读取二进制文件——它们是为让程序来读取而设计的。例如,Java 源程序存储在文本文件中,可以使用文本编辑器读取,而 Java 类是二进制文件,由 Java 虚拟机读取。

尽管从技术上讲不怎么准确,但是可以做这样一个比喻,文本文件是由字符序列构成的,而二进制文件是由位(bit)序列构成的。例如,十进制整数 199 在文本文件中是以三个字符序列 '1'、'9'、'9' 来存储的,而在二进制文件中它是以字节类型的值 C7 存储的,因为十进制数 199 等价的十六进制数是 C7 ($199 = 12 \times 16^1 + 7$)。二进制文件的优势在于它的处理效率比文本文件高。

Java 提供了许多实现文件输入/输出的类。这些类可以分为文本 I/O 类(text I/O class)和二进制 I/O 类(binary I/O class)。在 12.11 节中已经介绍过使用 `Scanner` 和 `PrintWriter` 如何从/向文本文件读/写字符串和数字值。本节介绍执行二进制 I/O 的类。

17.2 在 Java 中如何处理文本 I/O

 **要点提示:** 使用 `Scanner` 类读取文本数据,使用 `PrintWriter` 类写文本数据。

回顾一下,`File` 对象封装了文件或路径属性,但是不包含从/向文件读/写数据的方法。为了进行 I/O 操作,需要使用正确的 Java I/O 类创建对象。这些对象包含从/向文件中读/写数据的方法。例如,为了将文本写入一个名为 `temp.txt` 的文件中,可以使用 `PrintWriter`

类按如下方式创建一个对象：

```
PrintWriter output = new PrintWriter("temp.txt");
```

现在，可以调用该对象的 `print` 方法向文件写入一个字符串。例如，下面的语句将 Java 101 写入这个文件中。

```
output.print("Java 101");
```

下面的语句关闭这个文件。

```
output.close();
```

Java 有许多用于各种目的的 I/O 类。通常，可以将它们分为输入类和输出类。输入类包含读数据的方法，而输出类包含写数据的方法。`PrintWriter` 是一个输出类的例子，而 `Scanner` 是一个输入类的例子。下面的代码为文件 `temp.txt` 创建一个输入对象，并从该文件中读取数据：

```
Scanner input = new Scanner(new File("temp.txt"));
System.out.println(input.nextLine());
```

如果文件 `temp.txt` 中包含 Java 101，那么 `input.nextLine()` 方法就会返回字符串 "Java 101"。

图 17-1 描述了 Java I/O 程序设计。输入对象从文件中读取数据流，输出对象将数据流写入文件。输入对象也称作输入流（input stream）。同样，输出对象也称作输出流（output stream）。

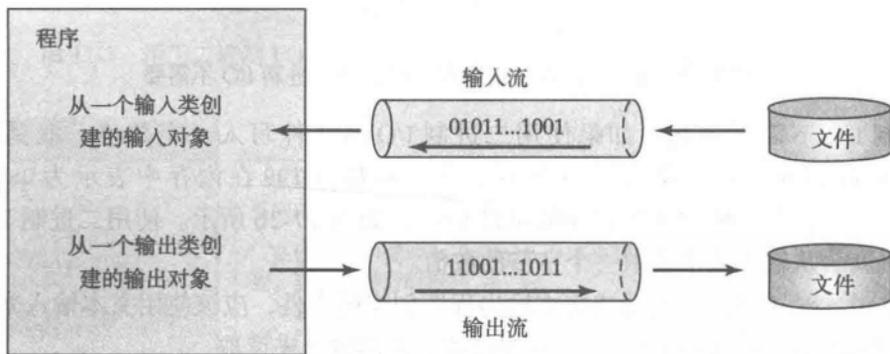


图 17-1 程序通过输入对象接收数据，通过输出对象发送数据

复习题

- 17.1 什么是文本文件，什么是二进制文件？可以使用文本编辑器来查看文本文件或者二进制文件吗？
- 17.2 在 Java 中如何读取和写入文本数据？什么是流？

17.3 文本 I/O 与二进制 I/O

要点提示：二进制 I/O 不涉及编码和解码，因此比文本 I/O 更加高效。

计算机并不区分二进制文件与文本文件。所有的文件都是以二进制形式来存储的，因此，从本质上说，所有的文件都是二进制文件。文本 I/O 建立在二进制 I/O 的基础之上，它

能提供一层抽象，用于字符层次的编码和解码，如图 17-2a 所示。对于文本 I/O 而言，编码和解码是自动进行的。

在写入一个字符时，Java 虚拟机会将统一码转化为文件指定的编码，而在读取字符时，将文件指定的编码转化为统一码。例如，假设使用文本 I/O 将字符串 "199" 写入文件，那么每个字符都会写入到文件中。由于字符 '1' 的统一码为 0x0031，所以，会根据文件的编码方案将统一码 0x0031 转化成一个代码。（注意，前缀 0x 表示十六进制数。）在美国，Windows 系统中文本文件的默认编码方案是 ASCII 码。字符 '1' 的 ASCII 码是 49（十六进制数是 0x31），而字符 '9' 是 57（十六进制数是 0x39）。所以，为了写入字符 "199"，就应该将三个字节 0x31、0x39 和 0x39 发送到输出，如图 17-2a 所示。

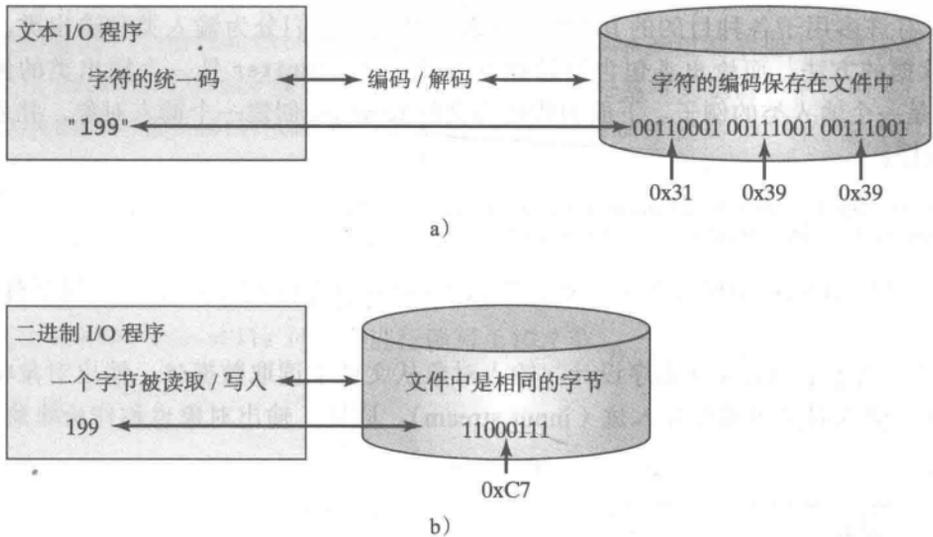


图 17-2 文本 I/O 需要编码和解码，而二进制 I/O 不需要

二进制 I/O 不需要转化。如果使用二进制 I/O 向文件写入一个数值，就是将内存中的那个值复制到文件中。例如，一个字节类型的数值 199 在内存中表示为 0xC7 ($199 = 12 \times 16^1 + 7$)，并且在文件中实际出现的也是 0xC7，如图 17-2b 所示。使用二进制 I/O 读取一个字节时，就会从输入流中读取一个字节的数值。

一般来说，对于文本编辑器或文本输出程序创建的文件，应该使用文本输入来读取，对于 Java 二进制输出程序创建的文件，应该使用二进制输入来读取。

由于二进制 I/O 不需要编码和解码，所以，它比文本 I/O 效率高。二进制文件与主机的编码方案无关，因此，它是可移植的。在任何机器上的 Java 程序可以读取 Java 程序所创建的二进制文件。这就是为什么 Java 的类文件存储为二进制文件的原因。Java 类文件可以在任何具有 Java 虚拟机的机器上运行。

注意：为了保持一致性，本书使用扩展名 .txt 来命名文本文件，使用 .dat 来命名二进制文件。

复习题

17.3 文本 I/O 与二进制 I/O 的区别是什么？

17.4 在 Java 中，字符在内存中是如何表示的，在文本文件中是如何表示的？

17.5 如果在一个 ASCII 码文本文件中写入字符串 "ABC"，那么在文件中存储的是什么值？

- 17.6 如果在一个 ASCII 码文本文件中写入字符串 "100", 那么在文件中存储的是什么值? 如果使用二进制 I/O 写入字节类型数值 100, 那么文件中存储的又是什么值?
- 17.7 在 Java 程序中, 表示一个字符使用的编码方案是什么? 在默认情况下, Windows 中文本文件的编码方案是什么?

17.4 二进制 I/O 类

要点提示: 抽象类 `InputStream` 是读取二进制数据的根类, 抽象类 `OutputStream` 是写入二进制数据的根类。

Java I/O 类的设计是一个很好的应用继承的例子, 它们的公共操作是由父类生成的, 而子类提供特定的操作。图 17-3 列出一些实现二进制 I/O 的类。`InputStream` 类是二进制输入类的根类, 而 `OutputStream` 类是二进制输出类的根类。图 17-4 和图 17-5 列出了 `InputStream` 类和 `OutputStream` 类的所有方法。

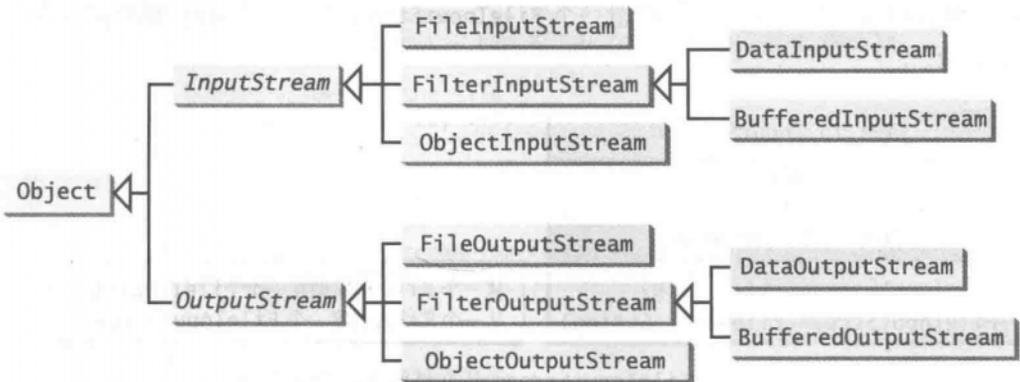


图 17-3 用于二进制 I/O 的 `InputStream` 类、`OutputStream` 类及其子类

<code>java.io.InputStream</code>	
<code>+read(): int</code>	从输入流中读取下一个字节数据。字节值以 0 到 255 取值范围的 <code>int</code> 值返回。如果因为已经达到流的最后而没有可读的字节, 则返回值 -1
<code>+read(b: byte[]): int</code>	从输入流中读取 <code>b.length</code> 个字节到数组 <code>b</code> 中, 并且返回实际读取的字节数。到流的最后时返回 -1
<code>+read(b: byte[], off: int, len: int): int</code>	从输入流中读取字节并且将它们保存在 <code>b[off]</code> , <code>b[off + 1]</code> , ..., <code>b[off + len - 1]</code> 中。返回实际读取的字节数。到流的最后时返回 -1
<code>+available(): int</code>	返回可以从输入流中读取的字节数的估计值
<code>+close(): void</code>	关闭输入流, 释放其占用的任何系统资源
<code>+skip(n: long): long</code>	从输入流中跳过并且丢弃 <code>n</code> 字节的数据。返回实际跳过的字节数
<code>+markSupported(): boolean</code>	测试该输入流是否支持 <code>mark</code> 和 <code>reset</code> 方法
<code>+mark(readlimit: int): void</code>	在该输入流中标记当前位置
<code>+reset(): void</code>	将该流重新定位到最后一次调用 <code>mark</code> 方法时的位置

图 17-4 抽象的 `InputStream` 类定义字节输入流的方法

注意: 二进制 I/O 类中的所有方法都声明为抛出 `java.io.IOException` 或 `java.io.IOException` 的子类。

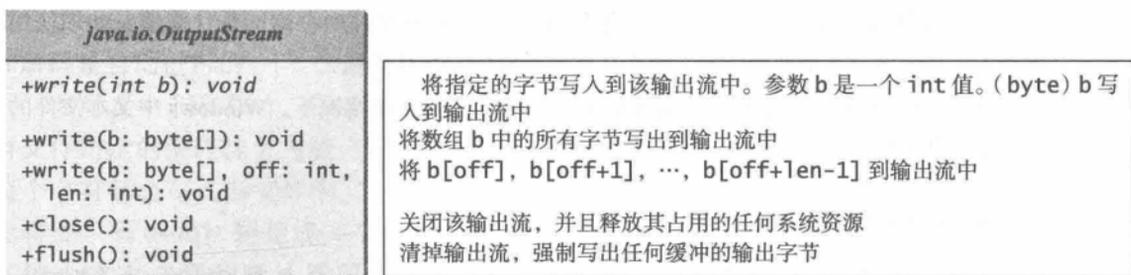


图 17-5 抽象的 OutputStream 类定义字节输出流的方法

17.4.1 FileInputStream 和 FileOutputStream

FileInputStream 类和 FileOutputStream 类用于从 / 向文件读取 / 写入字节。它们的所有方法都是从 InputStream 类和 OutputStream 类继承的。FileInputStream 类和 FileOutputStream 类没有引入新的方法。为了构造一个 FileInputStream 对象, 使用下面的构造方法, 如图 17-6 所示。

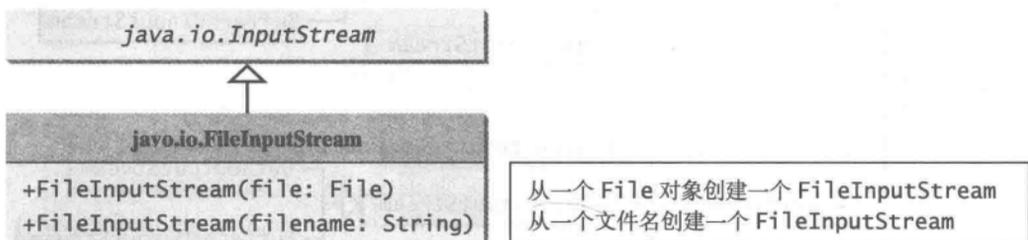


图 17-6 FileInputStream 从文件输入一个字节流

如果试图为一个不存在的文件创建 FileInputStream 对象, 将会发生 `java.io.FileNotFoundException` 异常。

要构造一个 FileOutputStream 对象, 使用如图 17-7 所示的构造方法。

如果这个文件不存在, 就会创建一个新文件。如果这个文件已经存在, 前两个构造方法将会删除文件的当前内容。为了既保留文件现有的内容又可以给文件追加新数据, 将最后两个构造方法中的参数 `append` 设置为 `true`。

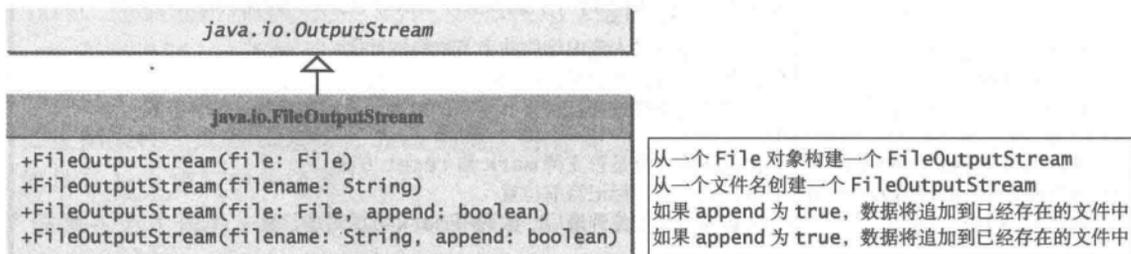


图 17-7 FileOutputStream 将一个字节流输出到文件中

几乎所有的 I/O 类中的方法都会抛出异常 `java.io.IOException`。因此, 必须在方法中声明会抛出 `java.io.IOException` 异常, 或者将代码放到 `try-catch` 块中, 如下所示:

在方法中声明异常

```
public static void main(String[] args)
    throws IOException {
    // Perform I/O operations
}
```

使用 try-catch 块

```
public static void main(String[] args) {
    try {
        // Perform I/O operations
    }
    catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

程序清单 17-1 使用二进制 I/O 将从 1 到 10 的 10 个字节值写入一个名为 temp.dat 的文件，再把它们从文件中读出来。

程序清单 17-1 TestFileStream.java

```
1 import java.io.*;
2
3 public class TestFileStream {
4     public static void main(String[] args) throws IOException {
5         try (
6             // Create an output stream to the file
7             FileOutputStream output = new FileOutputStream("temp.dat");
8         ) {
9             // Output values to the file
10            for (int i = 1; i <= 10; i++)
11                output.write(i);
12        }
13
14        try (
15            // Create an input stream for the file
16            FileInputStream input = new FileInputStream("temp.dat");
17        ) {
18            // Read values from the file
19            int value;
20            while ((value = input.read()) != -1)
21                System.out.print(value + " ");
22        }
23    }
24 }
```

1 2 3 4 5 6 7 8 9 10

程序使用了 try-with-resources 来声明和创建输入输出流，从而在使用后可以自动关闭。java.io.InputStream 和 java.io.OutputStream 实现了 AutoClosable 接口。AutoClosable 接口定义了 close() 方法，用于关闭资源。任何 AutoClosable 类型的对象可以用于 try-with-resources 语法中，实现自动关闭。

第 7 行为文件 temp.dat 创建了一个 FileOutputStream 对象。for 循环将 10 个字节值写入文件（第 10 ~ 11 行）。调用 write(i) 方法与调用 write((byte)i) 具有相同的功能。第 16 行为文件 temp.dat 创建一个 FileInputStream 对象。第 19 ~ 21 行从文件读取字节值并在控制台上显示出来。表达式 ((value = input.read()) != -1)（第 20 行）从 input.read() 中读取一个字节，然后将它赋值给 value，并且检验它是否为 -1。输入值为 -1 意味着文件的结束。

在这个例子中创建的文件 temp.dat 是一个二进制文件。可以从 Java 程序中读取它，但不能用文本编辑器阅读它，如图 17-8 所示。

提示：当流不再需要使用时，记得使用 close() 方法将其关闭，或者使用 try-with-resource 语句自动关闭。不关闭流可能会在输出文件中造成数据受损，或导致其他的程序设计错误。

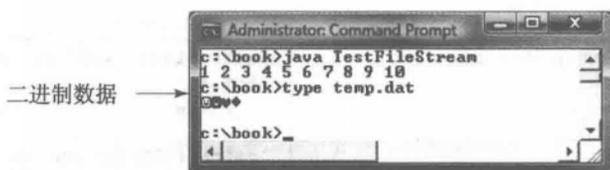


图 17-8 二进制文件不能以文本模式显示

注意：这些文件的根目录是类路径的目录。对于本书的例子，根目录是 `c:\book`。因此，文件 `temp.dat` 放在 `c:\book` 中。如果希望将 `temp.dat` 放在特定的目录下，使用下面的语句替换第 7 行：

```
FileOutputStream output =
    new FileOutputStream ("directory/temp.dat");
```

注意：`FileInputStream` 类的实例可以作为参数去构造一个 `Scanner` 对象，而 `FileOutputStream` 类的实例可以作为参数构造一个 `PrintWriter` 对象。可以创建一个 `PrintWriter` 对象来向文件中追加文本。如果 `temp.txt` 不存在，就会创建这个文件。如果 `temp.txt` 文件已经存在，就将新数据追加到该文件中。

```
new PrintWriter(new FileOutputStream("temp.txt", true));
```

17.4.2 FilterInputStream 和 FilterOutputStream

过滤器数据流 (filter stream) 是为某种目的过滤字节的数据流。基本字节输入流提供的读取方法 `read` 只能用来读取字节。如果要读取整数值、双精度值或字符串，那就需要一个过滤器类来包装字节输入流。使用过滤器类就可以读取整数值、双精度值和字符串，而不是字节或字符。`FilterInputStream` 类和 `FilterOutputStream` 类是过滤数据的基本类。需要处理基本数值类型时，就使用 `DataInputStream` 类和 `DataOutputStream` 类来过滤字节。

17.4.3 DataInputStream 和 DataOutputStream

`DataInputStream` 从数据流读取字节，并且将它们转换为合适的基本类型值或字符串。`DataOutputStream` 将基本类型的值或字符串转换为字节，并且将字节输出到数据流。

`DataInputStream` 类扩展 `FilterInputStream` 类，并实现 `DataInput` 接口，如图 17-9 所示。`DataOutputStream` 类扩展 `FilterOutputStream` 类，并实现 `DataOutput` 接口，如图 17-10 所示。

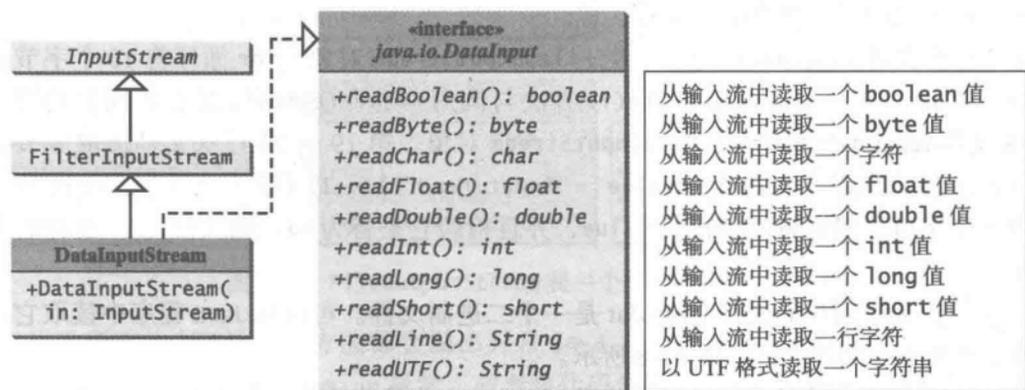


图 17-9 DataInputStream 过滤字节输入流并将其转化为基本类型值和字符串



图 17-10 DataOutputStream 可以将基本数据类型的值和字符串写入输出流

DataInputStream 实现了定义在 DataInput 接口中的方法来读取基本数据类型值和字符串。DataOutputStream 实现了定义在 DataOutput 接口中的方法来写入基本数据类型值和字符串。基本类型的值不需要做任何转化就可以从内存复制到输出数据流。字符串中的字符可以写成多种形式, 这将在下面介绍。

1. 二进制 I/O 中的字符与字符串

一个统一码由两个字节构成。writeChar(char c) 方法将字符 c 的统一码写入输出流。writeChars(String s) 方法将字符串 s 中所有字符的统一码写到输出流中。writeBytes(String s) 方法将字符串 s 中每个字符统一码的低字节写到输出流。统一码的高字节被丢弃。writeBytes 方法适用于由 ASCII 码字符构成的字符串, 因为 ASCII 码仅存储统一码的低字节。如果一个字符串包含非 ASCII 码的字符, 必须使用 writeChars 方法实现写入这个字符串。

writeUTF(String s) 方法将两个字节的长度信息写入输出流, 后面紧跟的是字符串 s 中每个字符的改进版 UTF-8 的形式。UTF-8 是一种编码方案, 它允许系统可以同时操作统一码及 ASCII 码。大多数操作系统使用 ASCII 码, Java 使用统一码。ASCII 码字符集是统一码字符集的子集。由于许多应用程序只需要 ASCII 码字符集, 所以将 8 位的 ASCII 码表示为 16 位的统一码是很浪费的。UTF-8 的修改版方案分别使用 1 字节、2 字节或 3 字节来存储字符。如果字符的编码值小于或等于 0x7F 就将该字符编码为一个字节, 如果字符的编码值大于 0x7F 而小于或等于 0x7FF 就将该字符编码为两个字节, 如果该字符的编码值大于 0x7FF 就将该字符编码为三个字节。

UTF-8 字符起始的几位表明这个字符是存储在一个字节、两个字节还是三个字节中。如果首位是 0, 那它就是一个字节的字符。如果前三位是 110, 那它就是两字节序列的第一个字节。如果前四位是 1110, 那它就是三字节序列的第一个字节。UTF-8 字符之前的两个字节用来存储表明字符串中的字符个数的信息。例如, 实际上, writeUTF("ABCDEF") 写入文件的是 8 个字节 (即 00 06 41 42 43 44 45 46), 因为头两个字节存储的是字符串中的字符个数。

writeUTF(String s) 方法将字符串转化成 UTF-8 格式的一串字节, 然后将它们写入一个输出流。readUTF() 方法读取一个使用 writeUTF 方法写入的字符串。

UTF-8 格式具有存储每个 ASCII 码就节省一个字节的优势, 因为一个统一码字符的存

储需要两个字节，而在 UTF-8 格式中 ASCII 字符仅占一个字节。如果一个长字符串的大多数字符都是普通的 ASCII 字符，采用 UTF-8 格式存储更加高效。

2. 创建 DataInputStream 类和 DataOutputStream 类

使用下面的构造方法来创建 DataInputStream 类和 DataOutputStream (参见图 17-9 和图 17-10):

```
public DataInputStream(InputStream instream)
public DataOutputStream(OutputStream outstream)
```

以下语句会创建数据流。第一条语句为文件 in.dat 创建一个输入流；而第二条语句为文件 out.dat 创建一个输出流：

```
DataInputStream input =
    new DataInputStream(new FileInputStream("in.dat"));
DataOutputStream output =
    new DataOutputStream(new FileOutputStream("out.dat"));
```

程序清单 17-2 将学生的名字和分数写入名为 temp.dat 的文件中，然后又将数据从这个文件中读出来。

程序清单 17-2 TestDataStream.java

```
1 import java.io.*;
2
3 public class TestDataStream {
4     public static void main(String[] args) throws IOException {
5         try ( // Create an output stream for file temp.dat
6             DataOutputStream output =
7                 new DataOutputStream(new FileOutputStream("temp.dat"));
8         ) {
9             // Write student test scores to the file
10            output.writeUTF("John");
11            output.writeDouble(85.5);
12            output.writeUTF("Jim");
13            output.writeDouble(185.5);
14            output.writeUTF("George");
15            output.writeDouble(105.25);
16        }
17
18        try ( // Create an input stream for file temp.dat
19            DataInputStream input =
20                new DataInputStream(new FileInputStream("temp.dat"));
21        ) {
22            // Read student test scores from the file
23            System.out.println(input.readUTF() + " " + input.readDouble());
24            System.out.println(input.readUTF() + " " + input.readDouble());
25            System.out.println(input.readUTF() + " " + input.readDouble());
26        }
27    }
28 }
```

John 85.5 Susan 185.5 Kim 105.25
--

第 6 行和第 7 行程序为文件 temp.dat 创建一个 DataOutputStream 对象。第 10 ~ 15 行将学生的名字和分数写入文件中。第 19 ~ 20 行为同一个文件创建 DataInputStream。第 23 ~ 25 行将这个文件中的学生名字和分数读回，并显示在控制台上。

DataInputStream 类和 DataOutputStream 类以同机器平台无关的方式读写 Java 基本类

型值和字符串，因此，如果在一台机器上写好一个数据文件，可以在另一台具有不同操作系统或文件结构的机器上读取该文件。应用程序可以利用数据输出流写入数据，之后某个程序可以利用数据输入流读取这个数据。

`DataInputStream` 将一个输入流的数据过滤成合适的基本类型值或者字符串。`DataOutputStream` 将基本类型值或者字符串转换成字节并且输出字节到输出流中。可以将 `DataInputStream/FileInputStream` 和 `DataOutputStream/FileOutputStream` 看作工作在一个管道线中，如图 17-11 所示。

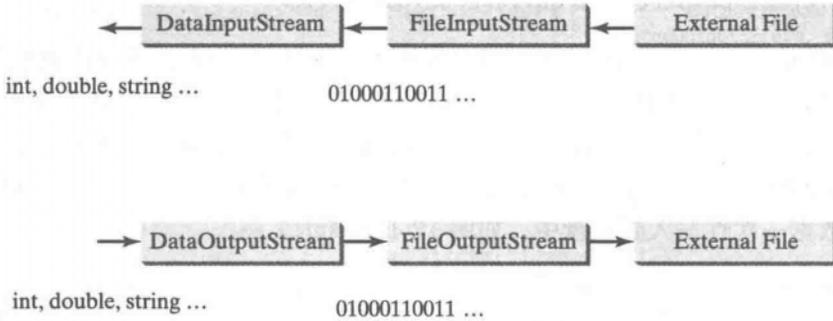


图 17-11 `DataInputStream` 将一个字节输入流过滤成数据，`DataOutputStream` 将数据转换成字节流

警告：应该按存储的顺序和格式读取文件中的数据。例如，学生的姓名是用 `writeUTF` 方法以 UTF-8 格式写入的，所以，读取时必须使用 `readUTF` 方法。

3. 检测文件的末尾

如果到达 `InputStream` 的末尾之后还继续从中读取数据，就会发生 `EOFException` 异常。这个异常可以用来检查是否已经到达文件末尾，如程序清单 17-3 所示。

程序清单 17-3 DetectEndOfFile.java

```

1 import java.io.*;
2
3 public class DetectEndOfFile {
4     public static void main(String[] args) {
5         try {
6             try (DataOutputStream output =
7                 new DataOutputStream(new FileOutputStream("test.dat"))) {
8                 output.writeDouble(4.5);
9                 output.writeDouble(43.25);
10                output.writeDouble(3.2);
11            }
12
13            try (DataInputStream input =
14                new DataInputStream(new FileInputStream("test.dat"))) {
15                while (true)
16                    System.out.println(input.readDouble());
17            }
18        }
19        catch (EOFException ex) {
20            System.out.println("All data were read");
21        }
22        catch (IOException ex) {
23            ex.printStackTrace();
24        }
25    }
26 }

```

```

4.5
43.25
3.2
All data were read

```

程序使用 `DataOutputStream` 向文件写入三个双精度值 (第 6 ~ 11 行), 然后使用 `DataInputStream` 读取这些数据 (第 13 ~ 17 行)。当读取文件超过了文件末尾, 就会抛出一个 `EOFException` 异常。该异常在第 19 行捕获。

17.4.4 BufferedInputStream 和 BufferedOutputStream

`BufferedInputStream` 类和 `BufferedOutputStream` 类可以通过减少磁盘读写次数来提高输入和输出的速度。使用 `BufferedInputStream` 时, 磁盘上的整块数据一次性地读入到内存中的缓冲区中。然后从缓冲区中将个别的数据传递到程序中, 如图 17-12a 所示。使用 `BufferedOutputStream`, 个别的数据首先写入到内存中的缓冲区中。当缓冲区已满时, 缓冲区中的所有数据一次性写入到磁盘中, 如图 17-12b 所示。

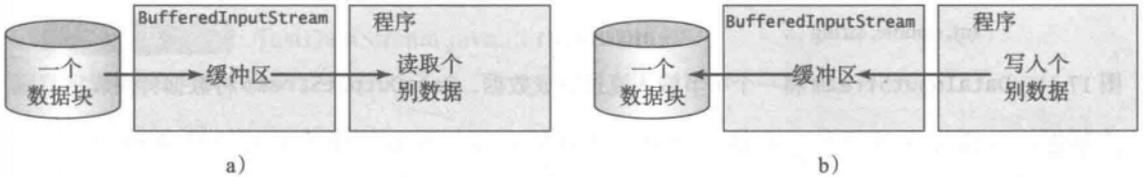


图 17-12 缓冲 I/O 将数据置于一个缓冲区中, 从而快速处理

`BufferedInputStream` 类和 `BufferedOutputStream` 类没有包含新的方法。`BufferedInputStream` 类和 `BufferedOutputStream` 中的所有方法都是从 `InputStream` 类和 `OutputStream` 类继承而来的。`BufferedInputStream` 类和 `BufferedOutputStream` 类在后台管理了一个缓冲区, 根据要求自动从磁盘中读取数据和写入数据。

可以使用如图 17-13 和 17-14 所示的构造方法包装在任何一个 `InputStream` 类和 `OutputStream` 类上的 `BufferedInputStream` 类和 `BufferedOutputStream` 类。

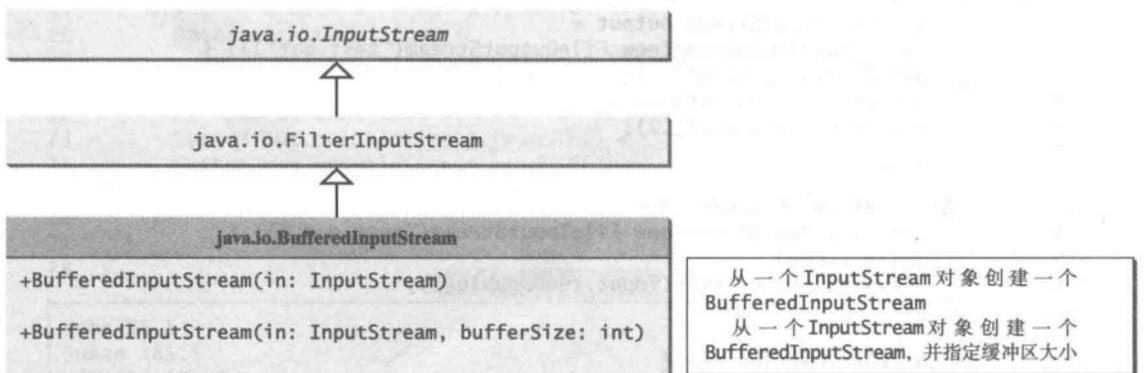


图 17-13 BufferedInputStream 缓冲一个输入流

如果没有指定缓冲区大小, 默认的大小是 512 个字节。通过在第 6 ~ 7 行与第 19 ~ 20 行给流添加缓冲区, 可以提高前面程序清单 17-2 中 `TestDataStream` 程序的效率, 如下所示:

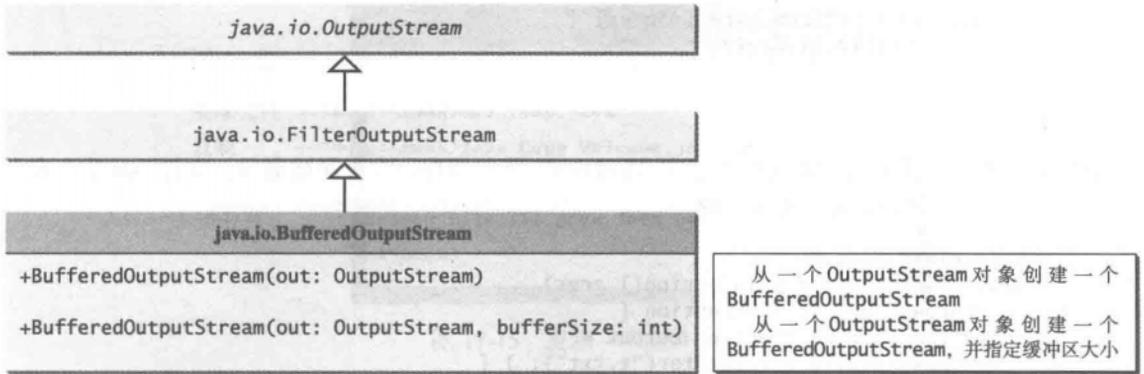


图 17-14 BufferedOutputStream 缓冲一个输出流

```

DataOutputStream output = new DataOutputStream(
    new BufferedOutputStream(new FileOutputStream("temp.dat")));
DataInputStream input = new DataInputStream(
    new BufferedInputStream(new FileInputStream("temp.dat")));
  
```

提示：应该总是使用缓冲区 I/O 来加速输入和输出。对于小文件，我们可能注意不到性能的提升。但是，对于超过 100MB 的大文件，我们将会看到使用缓冲的 I/O 带来的实质性的性能提升。

复习题

- 17.8 在 Java I/O 程序中，为什么必须在方法中声明抛出异常 IOException 或者在 try-catch 块中处理该异常？
- 17.9 为什么总是要求关闭流？如何关闭流？
- 17.10 InputStream 的 read() 方法读取字节。为什么 read() 方法返回 int 值而不是字节？找出 InputStream 和 OutputStream 中的抽象方法。
- 17.11 FileInputStream 类和 FileOutputStream 类是否相对于继承自的 InputStream/OutputStream 引入了新方法？如何创建 FileInputStream 和 FileOutputStream 对象？
- 17.12 如果试图为一个不存在的文件创建输入流，会发生什么？如果试图为一个已经存在的文件创建输出流，会发生什么？能够将数据追加到一个已存在的文件中吗？
- 17.13 如何使用 java.io.PrintWriter 向一个已存在的文本文件中追加数据？
- 17.14 假如一个文件包含了未指定个数的 double 值，使用 DataOutputStream 的 writeDouble 方法将这些值写入文件。如何编写程序读取所有这些值？如何检测是否到达这个文件的末尾？
- 17.15 在 FileOutputStream 上使用 writeByte(91) 方法后，写入文件的是什么？
- 17.16 在输入流 (FileInputStream 和 DataInputStream) 中如何判断是否已经到达文件末尾？
- 17.17 下面的代码有什么错误？

```

import java.io.*;

public class Test {
    public static void main(String[] args) {
        try (
            FileInputStream fis = new FileInputStream("test.dat"); ) {
        }
        catch (IOException ex) {
            ex.printStackTrace();
        }
    }
  
```

```

        catch (FileNotFoundException ex) {
            ex.printStackTrace();
        }
    }
}

```

- 17.18 假设使用默认的 ASCII 编码方案在 Windows 上运行程序，在程序结束后，文件 t.txt 中会有多少个字节？给出每个字节的内容。

```

public class Test {
    public static void main(String[] args)
        throws java.io.IOException {
        try (java.io.PrintWriter output =
            new java.io.PrintWriter("t.txt"); ) {
            output.printf("%s", "1234");
            output.printf("%s", "5678");
            output.close();
        }
    }
}

```

- 17.19 下面的程序运行完成后，文件 t.dat 中会有多少个字节？给出每个字节的内容。

```

import java.io.*;

public class Test {
    public static void main(String[] args) throws IOException {
        try (DataOutputStream output = new DataOutputStream(
            new FileOutputStream("t.dat")); ) {
            output.writeInt(1234);
            output.writeInt(5678);
            output.close();
        }
    }
}

```

- 17.20 对于下面这些关于 `DataOutputStream` 对象 `output` 上的语句，会有多少个字节发送到输出？

```

output.writeChar('A');
output.writeChars("BC");
output.writeUTF("DEF");

```

- 17.21 使用缓冲流有什么好处？下面的语句是否正确？

```

BufferedInputStream input1 =
    new BufferedInputStream(new FileInputStream("t.dat"));

DataInputStream input2 = new DataInputStream(
    new BufferedInputStream(new FileInputStream("t.dat")));

DataOutputStream output = new DataOutputStream(
    new BufferedOutputStream(new FileOutputStream("t.dat")));

```

17.5 示例学习：复制文件

 **要点提示：**本节开发一个有用的功能，用于复制文件。

本节中，将学习如何编写一个让用户复制文件的程序。用户需要提供一个源文件与一个目标文件作为命令行参数，所使用的命令如下：

```
java Copy source target
```

该程序将源文件复制到目标文件，然后显示这个文件中的字节数。如果源文件不存在，或者目标文件已经存在，程序应该给用户相应的提示。这个程序的一个运行示例如图 17-15 所示。

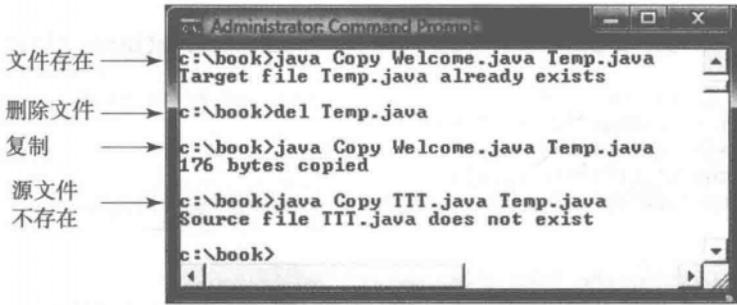


图 17-15 复制一个文件

要把源文件的内容复制到目标文件，不管文件的内容如何，使用输入流从源文件读出字节，并且使用输出流将字节写入目标文件比较合适。源文件和目标文件都是从命令行指定的。为源文件创建一个 `InputStream` 对象，为目标文件创建一个 `OutputStream` 对象。使用 `read()` 方法从输入流中读取一个字节，使用 `write(b)` 方法将一个字节写入输出流。使用 `BufferedInputStream` 类和 `BufferedOutputStream` 类来提高执行效率。程序清单 17-4 给出这个问题的解决方案。

程序清单 17-4 Copy.java

```

1  import java.io.*;
2
3  public class Copy {
4      /** Main method
5          @param args[0] for sourcefile
6          @param args[1] for target file
7      */
8      public static void main(String[] args) throws IOException {
9          // Check command-line parameter usage
10         if (args.length != 2) {
11             System.out.println(
12                 "Usage: java Copy sourceFile targetfile");
13             System.exit(1);
14         }
15
16         // Check if source file exists
17         File sourceFile = new File(args[0]);
18         if (!sourceFile.exists()) {
19             System.out.println("Source file " + args[0]
20                 + " does not exist");
21             System.exit(2);
22         }
23
24         // Check if target file exists
25         File targetFile = new File(args[1]);
26         if (targetFile.exists()) {
27             System.out.println("Target file " + args[1]
28                 + " already exists");
29             System.exit(3);
30         }
31
32         try {
33             // Create an input stream
34             BufferedInputStream input =
35                 new BufferedInputStream(new FileInputStream(sourceFile));
36
37             // Create an output stream

```

```

38     BufferedOutputStream output =
39         new BufferedOutputStream(new FileOutputStream(targetFile));
40     ) {
41         // Continuously read a byte from input and write it to output
42         int r, numberOfBytesCopied = 0;
43         while ((r = input.read()) != -1) {
44             output.write((byte)r);
45             numberOfBytesCopied++;
46         }
47
48         // Display the file size
49         System.out.println(numberOfBytesCopied + " bytes copied");
50     }
51 }
52 }

```

程序首先在第 10 ~ 14 行检查用户是否在命令行中传递了两个所需的参数。

程序使用 `File` 类检查源文件和目标文件是否存在。如果源文件不存在 (第 18 ~ 22 行), 或者目标文件已经存在 (第 25 ~ 30 行), 则程序退出。

在第 34 和 35 行, 使用包装在 `FileInputStream` 类上的 `BufferedInputStream` 类来创建一个输入流, 在第 38 和 39 行, 使用包装在 `FileOutputStream` 类上的 `BufferedOutputStream` 类来创建一个输出流。

表达式 `((r = input.read()) != -1)` (第 43 行) 从 `input.read()` 读取一个字节, 将该字节赋值给 `r`, 然后检查它是否为 `-1`。输入值 `-1` 表示一个文件的结束。程序不断地从输入流读取字节, 然后将它们写入输出流, 直到读取完所有的字节为止。

☛ 复习题

- 17.22 程序如何检测一个文件是否已经存在?
 17.23 程序如何在读取数据的时候检测是否已经到达文件末尾?
 17.24 程序如何计算从文件读取的字节数?

17.6 对象 I/O

🔑 **要点提示:** `ObjectInputStream` 类和 `ObjectOutputStream` 类可以用于读 / 写可序列化的对象。

`DataInputStream` 类和 `DataOutputStream` 类可以实现基本数据类型与字符串的输入和输出。而 `ObjectInputStream` 类和 `ObjectOutputStream` 类除了可以实现基本数据类型与字符串的输入和输出之外, 还可以实现对象的输入和输出。由于 `ObjectInputStream` 类和 `ObjectOutputStream` 类包含 `DataInputStream` 类和 `DataOutputStream` 类的所有功能, 所以, 完全可以用 `ObjectInputStream` 类和 `ObjectOutputStream` 类代替 `DataInputStream` 类和 `DataOutputStream` 类。

`ObjectInputStream` 扩展 `InputStream` 类, 并实现接口 `ObjectInput` 和 `ObjectStreamConstants`, 如图 17-16 所示。`ObjectInput` 是 `DataInput` 的子接口。`DataInput` 如图 17-9 所示。`ObjectStreamConstants` 包含支持 `ObjectInputStream` 类和 `ObjectOutputStream` 类所用的常量。

`ObjectOutputStream` 扩展 `OutputStream` 类, 并实现接口 `ObjectOutput` 与 `ObjectStreamConstants`, 如图 17-17 所示。`ObjectOutput` 是 `DataOutput` 的子接口 (`DataOutput` 如图 17-10 所示)。

可以使用下面的构造方法包装任何一个 `InputStream` 和 `OutputStream`, 构建 `ObjectInputStream` 和 `ObjectOutputStream`:

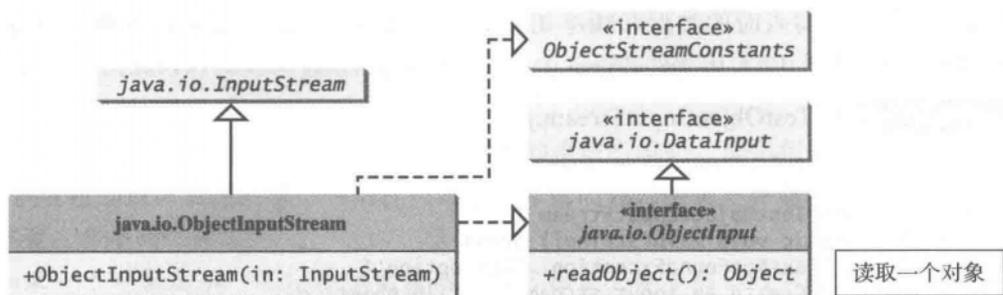


图 17-16 ObjectInputStream 可以读取对象、基本数据类型值和字符串

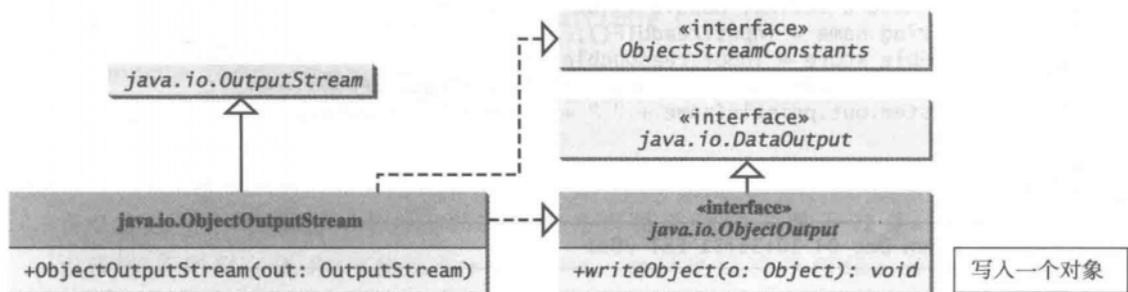


图 17-17 ObjectOutputStream 可以写入对象、基本数据类型值和字符串

```

// Create an ObjectInputStream
public ObjectInputStream(InputStream in)

// Create an ObjectOutputStream
public ObjectOutputStream(OutputStream out)

```

程序清单 17-5 将学生的姓名、分数和当前日期写入名为 object.dat 的文件中。

程序清单 17-5 TestObjectOutputStream.java

```

1 import java.io.*;
2
3 public class TestObjectOutputStream {
4     public static void main(String[] args) throws IOException {
5         try ( // Create an output stream for file object.dat
6             ObjectOutputStream output =
7                 new ObjectOutputStream(new FileOutputStream("object.dat"));
8         ) {
9             // Write a string, double value, and object to the file
10            output.writeUTF("John");
11            output.writeDouble(85.5);
12            output.writeObject(new java.util.Date());
13        }
14    }
15 }

```

在第 6 和第 7 行，创建一个 ObjectOutputStream 对象来将数据写入文件 object.dat 中。在第 10 ~ 12 行，将一个字符串、一个双精度值和一个对象写入这个文件。为了提高程序的性能，可以使用下面的语句替换第 6 和第 7 行，以完成在流中添加一个缓冲区：

```

ObjectOutputStream output = new ObjectOutputStream(
    new BufferedOutputStream(new FileOutputStream("object.dat")));

```

可以向数据流中写入多个对象或基本类型数据。从对应的 ObjectInputStream 中读回这

些对象时，必须与其写入时的类型和顺序相同。为了得到所需的类型，必须使用 Java 安全的类型转换。程序清单 17-6 从文件 object.dat 中读回数据。

程序清单 17-6 TestObjectInputStream.java

```
1 import java.io.*;
2
3 public class TestObjectInputStream {
4     public static void main(String[] args)
5         throws ClassNotFoundException, IOException {
6         try ( // Create an input stream for file object.dat
7             ObjectInputStream input =
8                 new ObjectInputStream(new FileInputStream("object.dat"));
9         ) {
10            // Read a string, double value, and object from the file
11            String name = input.readUTF();
12            double score = input.readDouble();
13            java.util.Date date = (java.util.Date)(input.readObject());
14            System.out.println(name + " " + score + " " + date);
15        }
16    }
17 }
```

John 85.5 Sun Dec 04 10:35:31 EST 2011

readObject() 方法可能会抛出异常 java.lang.ClassNotFoundException。这是因为 Java 虚拟机恢复一个对象时，如果没有加载该对象所在的类，就应该先加载这个类。因为 ClassNotFoundException 异常是一个必检异常，所以，在 main 方法中第 5 行声明抛出它。第 7 和 8 行创建了一个 ObjectInputStream 对象以从文件 object.dat 中读取输入。必须以数据写入文件时的顺序和格式从文件中读取这些数据。第 11 ~ 13 行会读取一个字符串、一个双精度值和一个对象。由于 readObject() 方法返回一个 Object 对象，所以，在第 13 行将它转换为 Date 类型并且赋给一个 Date 型变量。

17.6.1 Serializable 接口

并不是每一个对象都可以写到输出流。可以写入输出流中的对象称为可序列化的 (serializable)。因为可序列化的对象是 java.io.Serializable 接口的实例，所以，可序列化对象的类必须实现 Serializable 接口。

Serializable 接口是一种标记接口。因为它没有方法，所以，不需要在类中为实现 Serializable 接口增加额外的代码。实现这个接口可以启动 Java 的序列化机制，自动完成存储对象和数组的过程。

为了体会这个自动功能和理解对象是如何存储的，考虑一下不使用这一功能，储存一个对象需要做哪些工作。假设要存储一个 ArrayList 对象。为了完成这个任务，需要存储列表中的每个元素。每个元素是一个可能包含其他对象的对象。如你所见，这是一个非常繁琐冗长的过程。幸运的是，不必手工完成这个过程。Java 提供一个内在机制自动完成写对象的过程。这个过程称为对象序列化 (object serialization)，它是在 ObjectOutputStream 中实现的。与此相反，读取对象的过程称作对象反序列化 (object deserialization)，它是在 ObjectInputStream 类中实现的。

许多 Java API 中的类都实现了 Serializable 接口。所有针对基本类型值的包装类，java.math.BigInteger、java.math.BigDecimal、java.lang.String、java.lang.

StringBuilder、java.lang.StringBuffer、java.util.Date 以及 java.util.ArrayList 都实现了 java.io.Serializable 接口。试图存储一个不支持 Serializable 接口的对象会引起一个 NotSerializableException 异常。

当存储一个可序列化对象时，会对该对象的类进行编码。编码包括类名、类的签名、对象实例变量的值以及该对象引用的任何其他对象的闭包，但是不存储对象静态变量的值。

🔑 注意：非序列化的数据域

如果一个对象是 Serializable 的实例，但它包含了非序列化的实例数据域，那么可以序列化这个对象吗？答案是否定的。为了使该对象是可序列化的，需要给这些数据域加上关键字 transient，告诉 Java 虚拟机将对象写入对象流时忽略这些数据域。思考下面的类：

```
public class C implements java.io.Serializable {
    private int v1;
    private static double v2;
    private transient A v3 = new A();
}
```

```
class A { } // A is not serializable
```

当 C 类的一个对象进行序列化时，只需序列化变量 v1。因为 v2 是一个静态变量，所以没有序列化。因为 v3 标记为 transient，所以也没有序列化。如果 v3 没有标记为 transient，将会发生异常 java.io.NotSerializableException。

🔑 注意：重复的对象

如果一个对象不止一次写入对象流，会存储对象的多份副本吗？答案是不会。第一次写入一个对象时，就会为它创建一个序列号。Java 虚拟机将对象的所有内容和序列号一起写入对象流。以后每次存储时，如果再写入相同的对象，就只存储序列号。读出这些对象时，它们的引用相同，因为在内存中实际上存储的只是一个对象。

17.6.2 序列化数组

如果数组中的所有元素都是可序列化的，这个数组就是可序列化的。一个完整的数组可以用 writeObject 方法存入文件，随后用 readObject 方法恢复。程序清单 17-7 存储由五个 int 元素构成的数组和由三个字符串构成的数组，然后将它们从文件中读取出来显示在控制台上。

程序清单 17-7 TestObjectStreamForArray.java

```
1 import java.io.*;
2
3 public class TestObjectStreamForArray {
4     public static void main(String[] args)
5         throws ClassNotFoundException, IOException {
6         int[] numbers = {1, 2, 3, 4, 5};
7         String[] strings = {"John", "Susan", "Kim"};
8
9         try ( // Create an output stream for file array.dat
10             ObjectOutputStream output = new ObjectOutputStream(new
11                 FileOutputStream("array.dat", true));
12         ) {
13             // Write arrays to the object output stream
14             output.writeObject(numbers);
15             output.writeObject(strings);
16         }
17     }
```

```

18     try ( // Create an input stream for file array.dat
19         ObjectInputStream input =
20             new ObjectInputStream(new FileInputStream("array.dat"));
21     ) {
22         int[] newNumbers = (int[])(input.readObject());
23         String[] newStrings = (String[])(input.readObject());
24
25         // Display arrays
26         for (int i = 0; i < newNumbers.length; i++)
27             System.out.print(newNumbers[i] + " ");
28         System.out.println();
29
30         for (int i = 0; i < newStrings.length; i++)
31             System.out.print(newStrings[i] + " ");
32     }
33 }
34 }

```

```

1 2 3 4 5
John Susan Kim

```

第 14 和 15 行将两个数组写入文件 array.dat 中，第 22 和 23 行将这两个数组以存入时的顺序从文件中读取出来。由于 readObject() 方法返回 Object 对象，所以应该使用类型转换将其分别转换成 int[] 和 String[]。

复习题

- 17.25 使用 ObjectOutputStream 可以存储什么类型的对象？什么方法可以写入对象？什么方法可以读取对象？从 ObjectInputStream 读取对象的方法的返回值类型是什么？
- 17.26 如果序列化两个同样类型的对象，它们占用的空间相同吗？如果不同，举一个例子。
- 17.27 是否 java.io.Serializable 中的任何实例都可以成功地实现序列化？对象的静态变量是否可序列化？如何标记才能避免一个实例变量序列化？
- 17.28 可以向 ObjectOutputStream 中写入一个数组吗？
- 17.29 在任何情况下，DataInputStream 和 DataOutputStream 都可以用 ObjectInputStream 和 ObjectOutputStream 替换吗？
- 17.30 运行下面的代码时，会发生什么？

```

import java.io.*;

public class Test {
    public static void main(String[] args) throws IOException {
        try ( ObjectOutputStream output =
            new ObjectOutputStream(new FileOutputStream("object.dat")); ) {
            output.writeObject(new A());
        }
    }
}

class A implements Serializable {
    B b = new B();
}

class B {
}

```

17.7 随机访问文件

 **要点提示：**Java 提供了 RandomAccessFile 类，允许从文件的任何位置进行数据的读写。

到现在为止，所使用的所有流都是只读的 (read.only) 或只写的 (write.only)。这些流称为顺序 (sequential) 流。使用顺序流打开的文件称为顺序访问文件。顺序访问文件的内容不能更新。然而，经常需要修改文件。Java 提供了 `RandomAccessFile` 类，允许在文件的任意位置上进行读写。使用 `RandomAccessFile` 类打开的文件称为随机访问文件。

`RandomAccessFile` 类实现了 `DataInput` 和 `DataOutput` 接口，如图 17-18 所示。`DataInput` 接口 (参见图 17-9) 定义了读取基本数据类型和字符串的方法 (例如，`readInt`、`readDouble`、`readChar`、`readBoolean` 和 `readUTF`)。`DataOutput` 接口 (参见图 17-10) 定义了输出基本数据类型和字符串的方法 (例如，`writeInt`、`writeDouble`、`writeChar`、`writeBoolean` 和 `writeUTF`)。

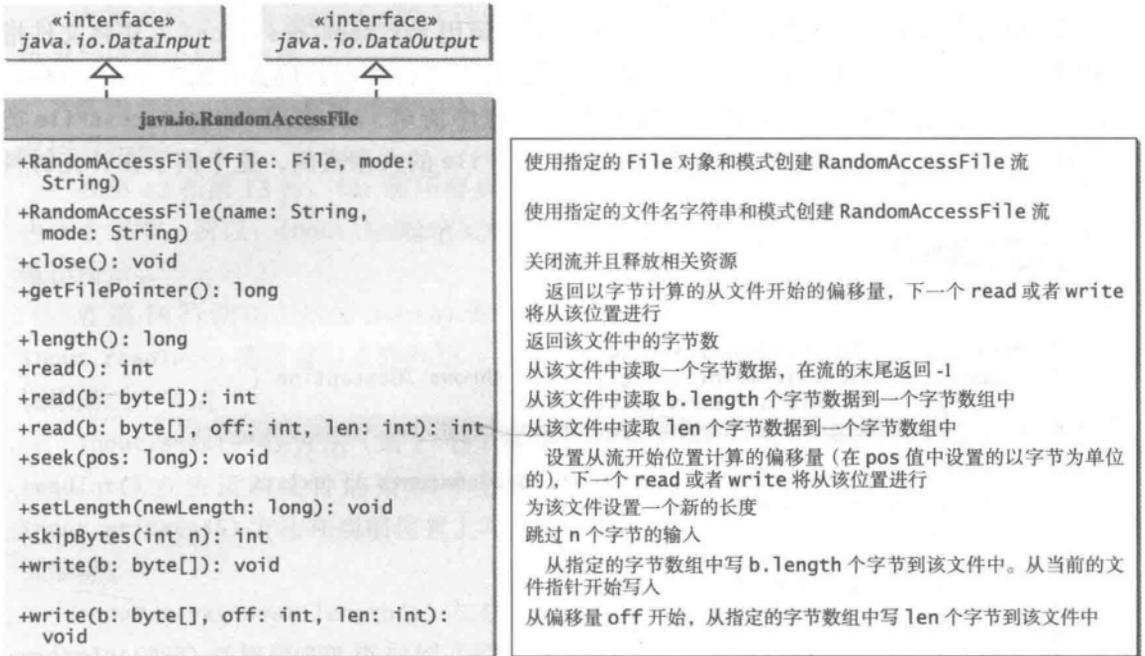


图 17-18 `RandomAccessFile` 类实现 `DataInput` 和 `DataOutput` 接口，并且增加了支持随机访问的方法

当创建一个 `RandomAccessFile` 时，可以指定两种模式 ("r" 或 "rw") 之一。模式 "r" 表明这个数据流是只读的，模式 "rw" 表明这个数据流既允许读也允许写。例如，下面的语句创建一个新的数据流 `raf`，它允许程序对文件 `test.dat` 进行读取和写入：

```
RandomAccessFile raf = new RandomAccessFile("test.dat", "rw");
```

如果文件 `test.dat` 已经存在，则创建 `raf` 以便访问这个文件；如果 `test.dat` 不存在，则创建一个名为 `test.dat` 的新文件，再创建 `raf` 来访问这个新文件。`raf.length()` 方法返回在给定时刻文件 `test.dat` 中的字节数。如果向文件中追加新数据，`raf.length()` 就会增加。

提示： 如果不想改动文件，就将文件以 "r" 模式打开。这样做可以防止不经意中改动文件。

随机访问文件是由字节序列组成的。一个称为文件指针 (file pointer) 的特殊标记定位这些字节中的某个字节的位置。文件的读写操作就是在文件指针所指的位置上进行的。打开文件时，文件指针置于文件的起始位置。在文件中进行读写数据后，文件指针就会向前移到下一个数据项。例如，如果使用 `readInt()` 方法读取一个 `int` 数据，Java 虚拟机就会从文件指针处读取 4 个字节，现在，文件指针就会从它之前的位置向前移动 4 个字节，如图 17-19 所示。

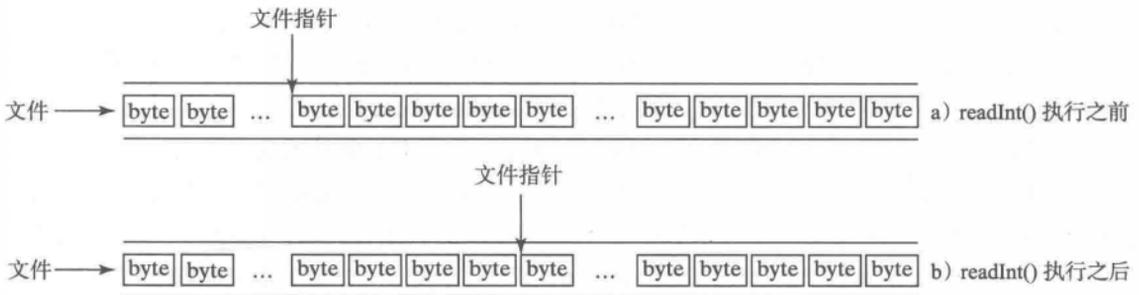


图 17-19 一个 int 值被读取后, 文件指针往前移动 4 个字节

设 `raf` 是 `RandomAccessFile` 的一个对象, 可以调用 `raf.seek(position)` 方法将文件指针移到指定的位置。`raf.seek(0)` 方法将文件指针移到文件的起始位置, 而 `raf.seek(raf.length())` 方法则将文件指针移到文件的末尾。程序清单 17-8 演示 `RandomAccessFile` 类的使用。管理地址簿是一个学习使用 `RandomAccessFile` 的大型实例, 这个例子在补充材料 VI.D 中给出。

程序清单 17-8 TestRandomAccessFile.java

```

1  import java.io.*;
2
3  public class TestRandomAccessFile {
4      public static void main(String[] args) throws IOException {
5          try ( // Create a random access file
6              RandomAccessFile inout = new RandomAccessFile("inout.dat", "rw");
7          ) {
8              // Clear the file to destroy the old contents if exists
9              inout.setLength(0);
10
11             // Write new integers to the file
12             for (int i = 0; i < 200; i++)
13                 inout.writeInt(i);
14
15             // Display the current length of the file
16             System.out.println("Current file length is " + inout.length());
17
18             // Retrieve the first number
19             inout.seek(0); // Move the file pointer to the beginning
20             System.out.println("The first number is " + inout.readInt());
21
22             // Retrieve the second number
23             inout.seek(1 * 4); // Move the file pointer to the second number
24             System.out.println("The second number is " + inout.readInt());
25
26             // Retrieve the tenth number
27             inout.seek(9 * 4); // Move the file pointer to the tenth number
28             System.out.println("The tenth number is " + inout.readInt());
29
30             // Modify the eleventh number
31             inout.writeInt(555);
32
33             // Append a new number
34             inout.seek(inout.length()); // Move the file pointer to the end
35             inout.writeInt(999);
36
37             // Display the new length
38             System.out.println("The new length is " + inout.length());
39
40             // Retrieve the new eleventh number

```

```
41     inout.seek(10 * 4); // Move the file pointer to the eleventh number
42     System.out.println("The eleventh number is " + inout.readInt());
43 }
44 }
45 }
```

```
Current file length is 800
The first number is 0
The second number is 1
The tenth number is 9
The new length is 804
The eleventh number is 555
```

在第 6 行为名为 `inout.dat` 的文件创建了一个模式为 "rw" 的 `RandomAccessFile` 对象，允许进行读取和写入操作。

在第 9 行，`inout.setLength(0)` 方法将文件长度设置为 0。这样做的效果是将文件的原有内容删除。

在第 12 和第 13 行，`for` 循环将从 0 到 199 的 200 个 `int` 值存入文件。由于每个 `int` 值占 4 个字节，所以，`inout.length()` 方法返回文件的现有总长度为 800（第 16 行），如示例输出所示。

在第 19 行调用 `inout.seek(0)` 方法将文件指针设置到文件的起始位置。第 20 行中 `inout.readInt()` 方法读取文件的第一个数值，然后将文件指针移动到下一个数值。第 24 行读取第二个数值。

`inout.seek(9*4)` 方法（第 27 行）将文件指针指向第 10 个数值。第 28 行中的 `inout.readInt()` 方法读取文件的第 10 个数值，然后将文件指针移动到文件的第 11 个数值。`inout.write(555)` 方法在当前位置上写入新的第 11 个数值（第 31 行），原来的第 11 个数据被删除。

`inout.seek(inout.length())` 方法将文件指针指向文件末尾（第 34 行），`inout.writeInt(999)` 将数值 999 添加到文件中（第 35 行）。现在文件的长度又增加了 4，所以，`inout.length()` 方法返回 804（第 38 行）。

在第 41 行，`inout.seek(10*4)` 方法将文件指针指向第 11 个数值。第 42 行显示新的第 11 个数值为 555。

☛ 复习题

- 17.31 `RandomAccessFile` 流是否可以读写由 `DataOutputStream` 创建的数据文件？`RandomAccessFile` 流是否可以读写对象？
- 17.32 为文件 `address.dat` 创建一个 `RandomAccessFile` 流，以便更新文件中的学生信息。为文件 `address.dat` 创建一个 `DataOutputStream` 流。解释这两条语句之间的差别。
- 17.33 如果文件 `test.dat` 不存在，那么试图编译运行下面的代码会出现什么情况？

```
import java.io.*;

public class Test {
    public static void main(String[] args) {
        try ( RandomAccessFile raf =
            new RandomAccessFile("test.dat", "r"); ) {
            int i = raf.readInt();
        }
        catch (IOException ex) {
```

```

        System.out.println("IO exception");
    }
}
}

```

关键术语

binary I/O (二进制输入 / 输出)	sequential-access file (顺序访问文件)
deserialization (反序列化)	serialization (序列化)
file pointer (文件指针)	stream (流)
random-access file (随机访问文件)	text I/O (文本输入 / 输出)

本章小结

1. I/O 类可以分为文本 I/O 和二进制 I/O。文本 I/O 将数据解释成字符序列，二进制 I/O 将数据解释成原始的二进制数值。文本在文件中如何存储依赖于文件的编码方式。Java 自动完成对文本 I/O 的编码和解码。
2. `InputStream` 类和 `OutputStream` 类是所有二进制 I/O 类的根类。`FileInputStream` 类和 `FileOutputStream` 类关联一个文件用于输入 / 输出。`BufferedInputStream` 类和 `BufferedOutputStream` 类可以包装任何一个二进制输入 / 输出流以提高其性能。`DataInputStream` 类和 `DataOutputStream` 类可以用来读写基本类型数据和字符串。
3. `ObjectInputStream` 类和 `ObjectOutputStream` 类除了可以读写基本类型数据值和字符串，还可以读写对象。为实现对象的可序列化，对象的定义类必须实现 `java.io.Serializable` 标记接口。
4. `RandomAccessFile` 类允许对文件读写数据。可以打开一个模式为 "r" 的文件，这个模式表示文件是只读的，也可以打开一个模式为 "rw" 的文件，这个模式表示文件是可更新的。由于 `RandomAccessFile` 类实现了 `DataInput` 和 `DataOutput` 接口，所以，`RandomAccessFile` 中的许多方法都与 `DataInputStream` 和 `DataOutputStream` 中的方法一样。

测试题

回答本章的在线测试题，地址为 www.cs.armstrong.edu/liang/intro10e/quiz.html。

编程练习题

17.3 节

- *17.1 (创建一个文本文件) 编写一个程序，如果文件 `Exercise17_01.txt` 不存在，就创建一个名为 `Exercise17_01.txt` 的文件。向这个文件追加新数据。使用文本 I/O 将 100 个随机生成的整数写入这个文件。文件中的整数用空格分隔。

17.4 节

- *17.2 (创建二进制数据文件) 编写一个程序，如果文件 `Exercise17_02.dat` 不存在，就创建一个名为 `Exercise17_02.dat` 的文件。向这个文件追加新数据。使用二进制 I/O 将 100 个随机生成的整数写入这个文件中。
- *17.3 (对二进制数据文件中的所有整数求和) 假设已经使用 `DataOutputStream` 中的 `writeInt(int)` 方法创建了一个名为 `Exercise17_03.dat` 的二进制数据文件，文件包含数目不确定的整数，编写一个程序来计算这些整数的总和。
- *17.4 (将文本文件转换为 UTF 格式) 编写一个程序，每次从文本文件中读取多行字符，并将这些行字符以 UTF-8 字符串格式写入一个二进制文件中。显示文本文件和二进制文件的大小。使用下面的命令运行这个程序：

```
java Exercise17_04 Welcome.java Welcome.utf
```

17.6 节

- *17.5 (将对象和数组存储在文件中) 编写一个程序, 向一个名为 Exercise17_05.dat 的文件中存储一个含 5 个 int 值 1, 2, 3, 4, 5 的数组, 一个表示当前时间的 Date 对象, 以及一个 double 值 5.5。
- *17.6 (存储 Loan 对象) 在程序清单 10-2 中的类 Loan 没有实现 Serializable, 改写类 Loan 使之实现 Serializable。编写程序创建 5 个 Loan 对象, 并且将它们存储在一个名为 Exercise17_06.dat 的文件中。
- *17.7 (从文件中恢复对象) 假设已经用 ObjectOutputStream 创建了一个名为 Exercise17_07.dat 的文件。这个文件包含 Loan 对象。在程序清单 10-2 中的 Loan 类没有实现 Serializable。改写 Loan 类实现 Serializable。编写程序, 从文件中读取 Loan 对象, 并且计算总的贷款额。假定文件中 Loan 对象的个数未知。使用 EOFException 来结束这个循环。

17.7 节

- *17.8 (更新计数器) 假设要追踪一个程序的运行次数。可以存储一个 int 值来对文件计数。程序每执行一次, 计数器就加 1。将程序命名为 Exercise17_08, 并且将计数器存储在文件 Exercise17_08.dat 中。
- ***17.9 (地址簿) 编写程序用于存储、返回、增加, 以及更新如图 17-20 所示的地址簿。使用固定长度的字符串来存储地址中的每个属性。使用随机访问文件来读取和写入一个地址。假设姓名、街道、城市、州以及邮政编码的长度分别是 32、32、20、2、5 字节。

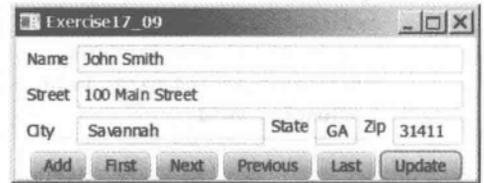


图 17-20 这个应用程序可以从 / 向一个文件中存储、返回以及更新地址簿

综合

- *17.10 (分割文件) 假设希望在 CD-R 上备份一个大文件 (例如, 一个 10GB 的 AVI 文件)。可以将该文件分割为几个小一些的片段, 然后独立备份这些小片段。编写一个工具程序, 使用下面的命令将一个大文件分割为小一些的文件:

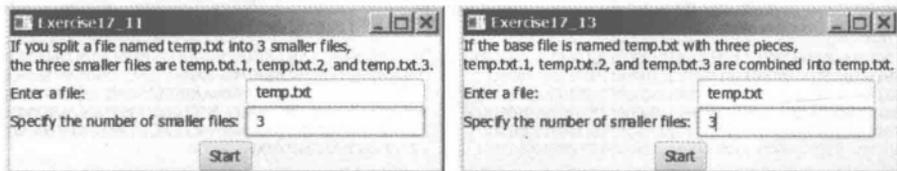
```
java Exercise17_10 SourceFile numberOfPieces
```

这个命令创建文件 SourceFile.1, SourceFile.2, ..., SourceFile.n, 这里的 n 是 numberOfPieces 而输出文件的大小基本相同。

- **17.11 (带 GUI 的分割文件工具) 改写练习题 17.10 使之带有 GUI, 如图 17-21a 所示。
- *17.12 (组合文件) 编写一个工具程序, 它能够用下面的命令, 将文件组合在一起构成一个新文件:

```
java Exercise17_12 SourceFile1 . . . SourceFileN TargetFile
```

这个命令将 SourceFile1, ..., SourceFileN 合并为 TargetFile。



a) 程序分割一个文件

b) 程序将文件组合成一个新文件

图 17-21

- *17.13 (带 GUI 的组合文件工具) 改写编程练习题 17.12 使之带有 GUI, 如图 17-21b 所示。
- 17.14 (加密文件) 通过给文件中的每个字节加 5 来对文件编码。编写一个程序, 提示用户输入一个输入文件名和一个输出文件名, 然后将输入文件的加密版本存入输出文件。

- 17.15 (解密文件) 假设文件是用编程练习题 17.14 中的编码方案加密的。编写一个程序, 解码这个加密文件。程序应该提示用户输入一个输入文件名和一个输出文件名, 然后将输入文件的解密版本存入输出文件。
- 17.16 (字符的频率) 编写一个程序, 提示用户输入一个 ASCII 文本文件名, 然后显示文件中每个字符出现的频率。
- **17.17 (BitOutputStream) 实现一个名为 BitOutputStream 的类, 如图 17-22 所示, 将比特写入一个输出流。方法 writeBit(char bit) 存储一个字节变量形式的比特。创建一个 BitOutputStream 时, 该字节是空的。在调用 writeBit('1') 之后, 这个字节就变成 00000001。在调用 writeBit("0101") 之后, 这个字节就变成 00010101。前三个字节还没有填充。当字节填满后, 就发送到输出流。现在, 字节重置为空。必须调用 close() 方法关闭这个流。如果这个字节非空也不满, close() 方法就会先填充 0 以使字节的 8 个比特都被填满, 然后输出字节并关闭这个流。可以参见编辑练习题 5.44 得到提示。编写一个测试程序, 将比特 0100001001000010011101 发送给一个名为 Exercise17_17.dat 的文件。

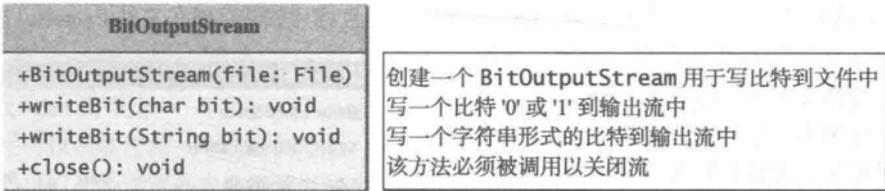


图 17-22 BitOutputStream 输出比特流到文件中

- *17.18 (查看比特) 编写下面的方法, 用于显示一个整数的最后一个字节的比特表示:

```
public static String getBits(int value)
```

可以参见编程练习题 5.44 获得提示。编写一个程序, 提示用户输入一个文件名, 从文件读取字节, 然后显示每个字节的二进制表示形式。

- *17.19 (查看十六进制) 编写一个程序, 提示用户输入文件名, 从文件读取字节, 然后显示每个字节的十六进制表示形式。

 提示: 可以先将字节值转换为一个 8 比特的字符串, 然后再将比特字符串转换为一个两位的十六进制字符串。

- **17.20 (二进制编辑器) 编写一个 GUI 应用程序, 让用户在文本域输入一个文件名, 然后单击回车键, 在文本区域显示它的二进制表示形式。用户也可以修改这个二进制代码, 然后将它回存到这个文件中, 如图 17-23a 所示。

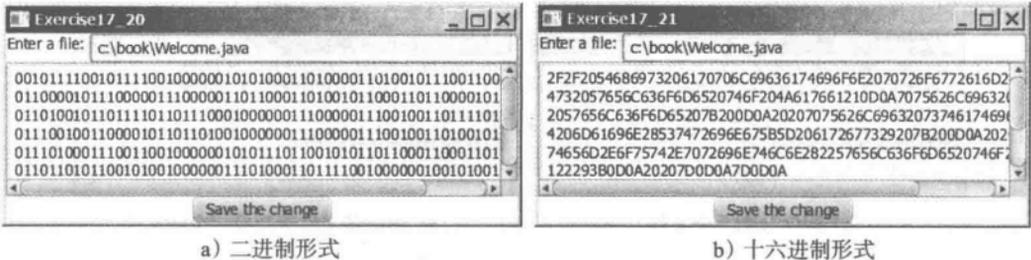


图 17-23

- **17.21 (十六进制编辑器) 编写一个 GUI 应用程序, 让用户在文本域输入一个文件名, 然后按回车键, 在文本域显示它的十六进制表达形式。用户也可以修改十六进制代码, 然后将它回存到这个文件中, 如图 17-23b 所示。

递 归

教学目标

- 描述什么是递归方法以及使用递归方法的好处 (18.1 节)。
- 为递归数学函数开发递归方法 (18.2 ~ 18.3 节)。
- 解释在调用栈中如何处理递归方法的调用 (18.2 ~ 18.3 节)。
- 使用递归进行问题求解 (18.4 节)。
- 使用一个重载的辅助方法设计一个递归方法 (18.5 节)。
- 使用递归实现选择排序 (18.5.1 节)。
- 使用递归实现二分查找 (18.5.2 节)。
- 使用递归获取一个目录的大小 (18.6 节)。
- 使用递归解决汉诺塔问题 (18.7 节)。
- 使用递归绘制分形 (18.8 节)。
- 了解递归和迭代之间的联系与区别 (18.9 节)。
- 了解尾递归方法以及为什么需要它 (18.10 节)。

18.1 引言

 **要点提示:** 递归是一种针对使用简单的循环难以编程实现的问题, 提供优雅解决方案的技术。

假设希望找出某目录下所有包含某个特定单词的文件, 该如何解决这个问题呢? 有几种方式可以解决这个问题。一个直观且有效的解决方法是使用递归在子目录下递归地搜索所有的文件。

H- 树, 如图 18-1 所示, 在超大规模集成电路 (Very Large-Scale Integration, VLSI) 设计中作为时钟线分布网使用, 用于将记时信号以同等的时延路由到芯片的所有部分。如何编写程序显示 H- 树呢? 一个好的方法是使用递归。

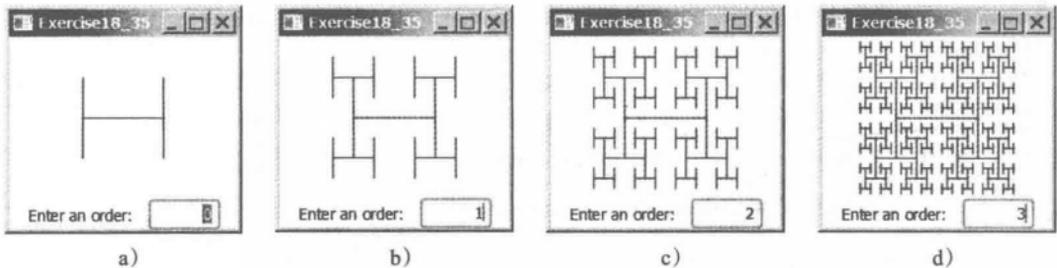


图 18-1 H-树可以采用递归来显示

使用递归就是使用递归方法 (recursive method) 编程, 递归方法就是直接或间接调用自身的方法。递归是一个很有用的程序设计技术。在某些情况下, 对于用其他方法很难解决的

问题，使用递归就能给出一个直观、直接的简单解法。本章介绍递归程序设计的概念和技术，并用例子来演示如何进行“递归思考”。

18.2 示例学习：计算阶乘

 **要点提示：**递归方法是调用自身的方法。

许多数学函数都是使用递归来定义的。我们从一个简单的例子开始。数字 n 的阶乘可以递归地定义如下：

```
0! = 1;
n! = n × (n - 1)!; n > 0
```

对给定的 n 如何求 $n!$ 呢？由于已经知道 $0!=1$ ，而 $1!=1 \times 0!$ ，因此很容易求得 $1!$ 。假设已知知道 $(n-1)!$ ，使用 $n!=n \times (n-1)!$ 就可以立即得到 $n!$ 。这样，计算 $n!$ 的问题就简化为计算 $(n-1)!$ 。当计算 $(n-1)!$ 时，可以递归地应用这个思路直到 n 递减为 0。

假定计算 $n!$ 的方法是 `factorial(n)`。如果用 $n=0$ 调用这个方法，立即就能返回它的结果。这个方法知道如何处理最简单的情况，这种最简单的情况称为基础情况（base case）或终止条件（stopping condition）。如果用 $n>0$ 调用这个方法，就应该把这个问题简化为计算 $n-1$ 的阶乘的子问题。子问题在实质上和原始问题是一样的，但是它比原始问题更简单也更小。因为子问题和原始问题具有相同的性质，所以可以用不同的参数调用这个方法，这称作递归调用（recursive call）。

计算 `factorial(n)` 的递归算法可以简单地描述如下：

```
if (n == 0)
    return 1;
else
    return n * factorial(n - 1);
```

一个递归调用可以导致更多的递归调用，因为这个方法继续把每个子问题分解成新的子问题。要终止一个递归方法，问题最后必须达到一个终止条件。当问题达到这个终止条件时，就将结果返回给调用者。然后调用者进行计算并将结果返回给它自己的调用者。这个过程持续进行，直到结果传回原始的调用者为止。现在，原始问题就可以将 `factorial(n-1)` 的结果乘以 n 得到。

程序清单 18-1 给出一个完整的程序，提示用户输入一个非负整数，然后显示这个数的阶乘。

程序清单 18-1 ComputeFactorial.java

```
1 import java.util.Scanner;
2
3 public class ComputeFactorial {
4     /** Main method */
5     public static void main(String[] args) {
6         // Create a Scanner
7         Scanner input = new Scanner(System.in);
8         System.out.print("Enter a nonnegative integer: ");
9         int n = input.nextInt();
10
11         // Display factorial
12         System.out.println("Factorial of " + n + " is " + factorial(n));
13     }
14
15     /** Return the factorial for the specified number */
```

```

16 public static long factorial(int n) {
17     if (n == 0) // Base case
18         return 1;
19     else
20         return n * factorial(n - 1); // Recursive call
21 }
22 }

```

Enter a nonnegative integer: 4 Enter
Factorial of 4 is 24

Enter a nonnegative integer: 10 Enter
Factorial of 10 is 3628800

本质上讲，factorial 方法（第 16 ~ 21 行）是把阶乘在数学上的递归定义直接转换为 Java 代码。因为对 factorial 的调用是调用它自己，所以这个调用是递归的。传递到 factorial 的参数一直递减，直到达到它的基础情况 0。

现在，你看到了如何编写一个递归方法。那么递归在后台是如何工作的呢？图 18-2 展示了一个递归调用的执行过程，从 $n=4$ 开始。针对递归调用的堆栈空间的使用如图 18-3 所示。

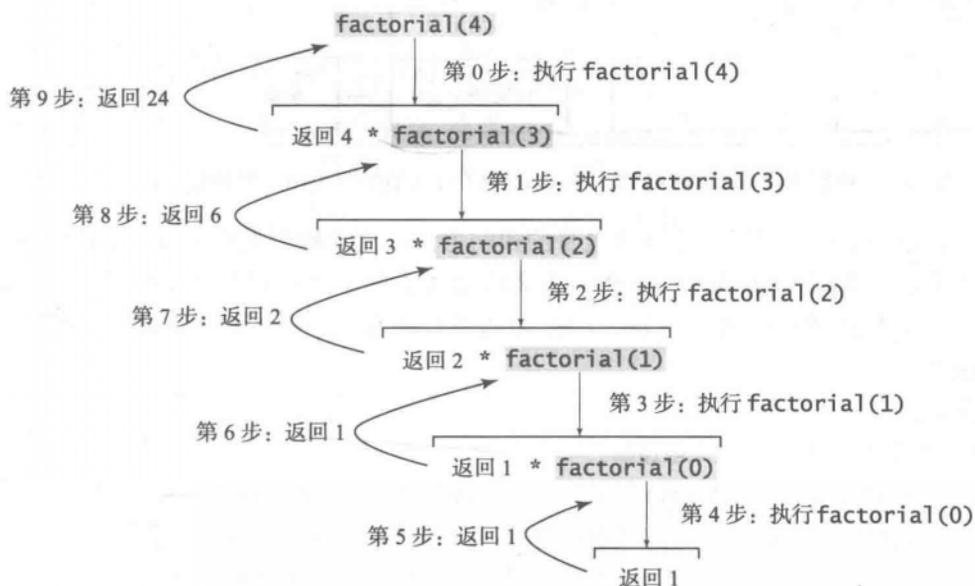


图 18-2 调用 factorial(4) 会引起对 factorial 的递归调用

教学注意: 使用循环来实现 factorial 方法是比较简单且更加高效的。然而，这里使用递归 factorial 方法演示递归的概念。在本章后续内容中还将给出一些问题，其内在逻辑是递归的，不使用递归很难解决。

如果递归不能使问题简化并最终收敛到基础情况，就有可能出现无限递归。例如，假设将 factorial 方法错误地写成如下所示：

```

public static long factorial(int n) {
    return n * factorial(n - 1);
}

```

那么这个方法会无限地运行下去，并且会导致一个 StackOverflowError。

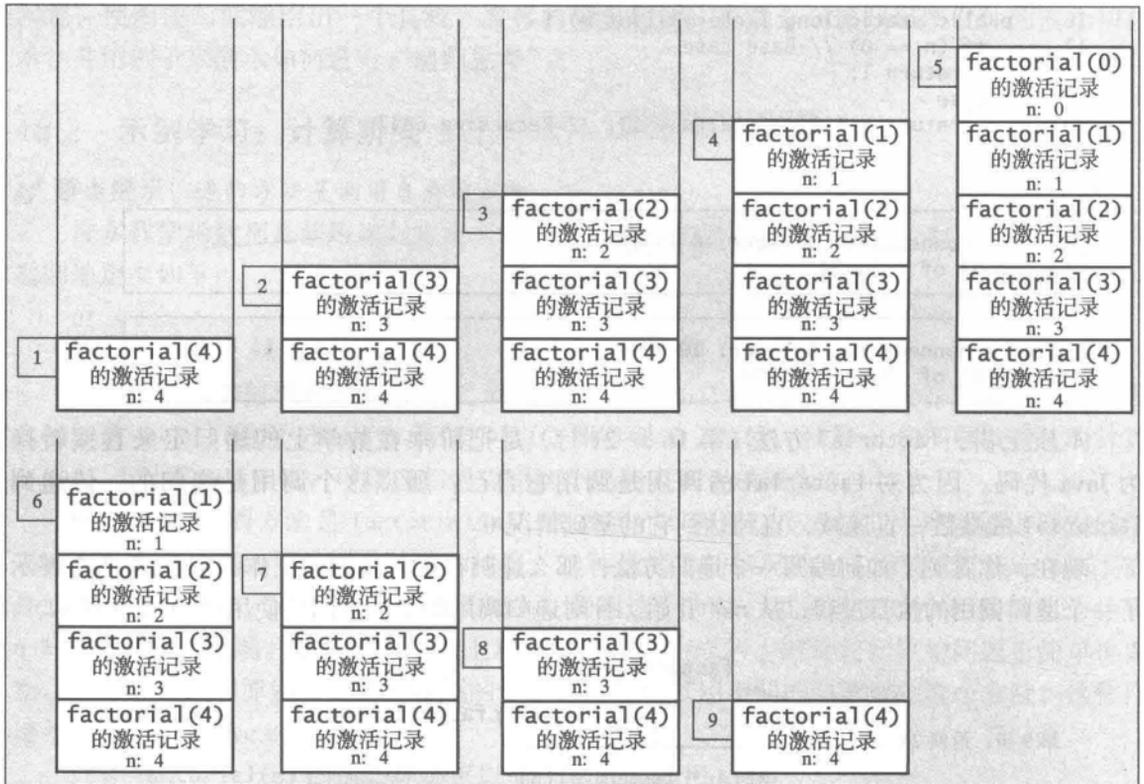


图 18-3 执行 factorial(4) 时, factorial 方法被递归调用, 导致栈空间动态变化

本节讨论的示例演示了一个调用自身的递归方法。这被称为直接递归。也可能创建间接递归。当方法 A 调用方法 B, 接着 B 方法又调用 A 方法, 间接递归就发生了。甚至可以有更多的方法参与到递归中来。例如, 方法 A 调用方法 B, 方法 B 调用方法 C, 而方法 C 又调用方法 A。

复习题

- 18.1 什么是递归方法? 什么是无限递归?
- 18.2 程序清单 18-1 中, 对于 factorial(6) 而言, 涉及多少次的 factorial 方法调用?
- 18.3 给出下面程序的输出, 指出基础情况以及递归调用。

```
public class Test {
    public static void main(String[] args) {
        System.out.println(
            "Sum is " + xMethod(5));
    }

    public static int xMethod(int n) {
        if (n == 1)
            return 1;
        else
            return n + xMethod(n - 1);
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        xMethod(1234567);
    }

    public static void xMethod(int n) {
        if (n > 0) {
            System.out.print(n % 10);
            xMethod(n / 10);
        }
    }
}
```

- 18.4 编写一个递归的数学定义来计算 2^n , 其中 n 为正整数。
- 18.5 编写一个递归的数学定义来计算 x^n , 其中 n 为正整数, x 为实数。
- 18.6 编写一个递归的数学定义来计算 $1+2+3+\dots+n$, 其中 n 为正整数。

18.3 示例学习：计算斐波那契数

🔑 要点提示：某些情况下，递归调用可以帮助你给出一个问题直观、直接、简单的解决方法。

前一节中的 `factorial` 方法可以很容易地不使用递归改写。但是，在某些情况下，用其他方法不容易解决的问题可以利用递归给出一个直观、直接、简单的解法。考虑众所周知的斐波那契 (Fibonacci) 数列问题：

数列：0 1 1 2 3 5 8 13 21 34 55 89 ...

下标：0 1 2 3 4 5 6 7 8 9 10 11

斐波那契数列从 0 和 1 开始，之后的每个数都是序列中前两个数的和。数列可以递归定义为：

```
fib(0) = 0;
fib(1) = 1;
fib(index) = fib(index - 2) + fib(index - 1); index >= 2
```

斐波那契数列是以中世纪数学家 Leonardo Fibonacci 的名字命名的，他为建立兔子繁殖数量的增长模型而构造出这个数列。这个数列可用于数值优化和其他很多领域。

对给定的 `index`，怎样求 `fib(index)` 呢？因为已知 `fib(0)` 和 `fib(1)`，所以很容易求得 `fib(2)`。假设已知 `fib(index-2)` 和 `fib(index-1)`，就可以立即得到 `fib(index)`。这样，计算 `fib(index)` 的问题就简化为计算 `fib(index-2)` 和 `fib(index-1)` 的问题。以这种方式求解，就可以递归地运用这个思路直到 `index` 递减为 0 或 1。

基础情况是 `index=0` 或 `index=1`。若用 `index=0` 或 `index=1` 调用这个方法，它会立即返回结果。若用 `index>=2` 调用这个方法，则通过使用递归调用把问题分解成计算 `fib(index-2)` 和 `fib(index-1)` 两个子问题。计算 `fib(index)` 的递归算法可以简单地描述如下：

```
if (index == 0)
    return 0;
else if (index == 1)
    return 1;
else
    return fib(index - 1) + fib(index - 2);
```

程序清单 18-2 给出一个完整的程序，提示用户输入一个下标，然后计算这个下标值相应的斐波那契数。

程序清单 18-2 ComputeFibonacci.java

```
1 import java.util.Scanner;
2
3 public class ComputeFibonacci {
4     /** Main method */
5     public static void main(String[] args) {
6         // Create a Scanner
7         Scanner input = new Scanner(System.in);
8         System.out.print("Enter an index for a Fibonacci number: ");
9         int index = input.nextInt();
10
11         // Find and display the Fibonacci number
12         System.out.println("The Fibonacci number at index "
13             + index + " is " + fib(index));
14     }
15
16     /** The method for finding the Fibonacci number */
17     public static long fib(long index) {
18         if (index == 0) // Base case
```

```

19     return 0;
20     else if (index == 1) // Base case
21         return 1;
22     else // Reduction and recursive calls
23         return fib(index - 1) + fib(index - 2);
24 }
25 }

```

Enter an index for a Fibonacci number: 1
The Fibonacci number at index 1 is 1

Enter an index for a Fibonacci number: 6
The Fibonacci number at index 6 is 8

Enter an index for a Fibonacci number: 7
The Fibonacci number at index 7 is 13

程序并没有显示计算机在后台所做的大量工作。然而图 18-4 给出了计算 `fib(4)` 所进行的连续递归调用。原始方法 `fib(4)` 产生两个递归调用 `fib(3)` 和 `fib(2)`，然后返回 `fib(3)+fib(2)` 的值。但是，按怎样的顺序调用这些方法呢？在 Java 中，操作数是从左到右计算的，所以在完全计算完 `fib(3)` 之后才会调用 `fib(2)`。图 18-4 中的箭头表示方法调用的顺序。

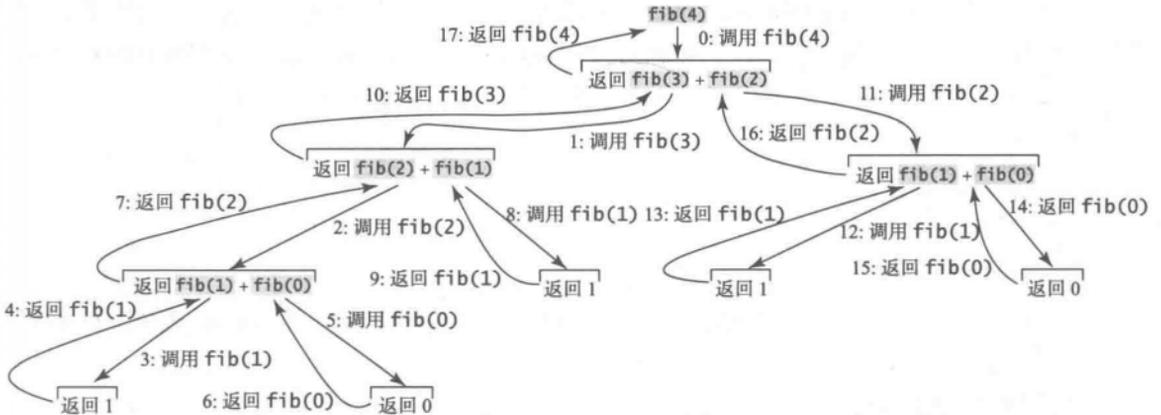


图 18-4 调用 `fib(4)` 会引起对 `fib` 的递归调用

如图 18-4 所示，会出现很多重复的递归调用。例如，`fib(2)` 调用了 2 次，`fib(1)` 调用了 3 次，`fib(0)` 也调用了 2 次。通常，计算 `fib(index)` 所需的递归调用次数大致是计算 `fib(index-1)` 所需次数的 2 倍。如果尝试更大的下标值，那么相应的调用次数会急剧增加，如表 18-1 所示。

表 18-1 `fib(index)` 的递归调用次数

下标	2	3	4	10	20	30	40	50
调用次数	3	5	9	177	21 891	2 692 537	331 160 281	2 075 316 483

教学注意：`fib` 方法的递归实现非常简单、直接，但是并不高效，因为它要求更多的时间和内存来运行递归方法。参见编程练习题 18.2 中使用循环的高效方案。虽然递归的 `fib` 方法并不实用，但是它是一个演示如何编写递归方法的很好的例子。

复习题

18.7 给出以下两个程序的输出:

```
public class Test {
    public static void main(String[] args) {
        xMethod(5);
    }

    public static void xMethod(int n) {
        if (n > 0) {
            System.out.print(n + " ");
            xMethod(n - 1);
        }
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        xMethod(5);
    }

    public static void xMethod(int n) {
        if (n > 0) {
            xMethod(n - 1);
            System.out.print(n + " ");
        }
    }
}
```

18.8 下面方法中的错误是什么?

```
public class Test {
    public static void main(String[] args) {
        xMethod(1234567);
    }

    public static void xMethod(double n) {
        if (n != 0) {
            System.out.print(n);
            xMethod(n / 10);
        }
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        Test test = new Test();
        System.out.println(test.toString());
    }

    public Test() {
        Test test = new Test();
    }
}
```

18.9 程序清单 18-2 中, 对 fib(6) 进行了多少次 fib 方法的调用?

18.4 使用递归解决问题

 **要点提示:** 采用递归的思考方式可以解决许多问题。

前几节给出了两个经典的递归例子。所有的递归方法都具有以下特点:

- 这些方法使用 if-else 或 switch 语句来引导不同的情况。
- 一个或多个基础情况 (最简单的情况) 用来停止递归。
- 每次递归调用都会简化原始问题, 让它不断地接近基础情况, 直到它变成这种基础情况为止。

通常, 要使用递归解决问题, 就要将这个问题分解为子问题。每个子问题几乎与原始问题是一样的, 只是规模小一些。可以应用相同的方法来递归解决子问题。

许多地方存在着递归。使用递归进行思考非常有趣。考虑喝咖啡这一事情, 你可以如下描述过程:

```
public static void drinkCoffee(Cup cup) {
    if (!cup.isEmpty()) {
        cup.takeOneSip(); // Take one sip
        drinkCoffee(cup);
    }
}
```

假设 cup 是描述一杯咖啡的实例对象, 具有 isEmpty() 和 takeOneSip() 方法。可以将问题转换为两个子问题: 一个是喝一小口咖啡, 另外一个为喝杯中剩下的咖啡。第二个问题和原问题是一样的, 只是规模上更小。而问题的基础情形是杯子空了。

考虑打印一条消息 n 次的简单问题。可以将这个问题分解为两个子问题：一个是打印消息一次，另一个是打印消息 $n-1$ 次。第二个问题与原始问题是一样的，只是规模小一些。这个问题的基础情况是 $n=0$ 。可以使用递归来解决这个问题，如下所示：

```
public static void nPrintln(String message, int times) {
    if (times >= 1) {
        System.out.println(message);
        nPrintln(message, times - 1);
    } // The base case is times == 0
}
```

需要注意的是，前面例子中的 `fib` 方法向其调用者返回一个数值，但是 `drinkCoffee` 和 `nPrintln` 方法的返回类型是 `void`，并不向其调用者返回一个数值。

如果以递归的思路进行思考 (think recursively)，那么，本书前面章节中的许多问题都可以用递归来解决。考虑程序清单 5-14 中的回文问题。回想一下，如果一个字符串从左读和从右读是一样的，那么它就是一个回文串。例如，`mom` 和 `dad` 都是回文串，但是 `uncle` 和 `aunt` 不是回文串。检查一个字符串是否是回文串的问题可以分解为两个子问题：

- 检查字符串中的第一个字符和最后一个字符是否相等。
- 忽略两端的字符之后检查子串的其余部分是否是回文串。

第二个子问题与原始问题是一样的，但是规模小一些。基本状态有两个：1) 两端的字符不同；2) 字符串大小是 0 或 1。在第一种情况下，字符串不是回文串；而在第二种情况下，字符串是回文串。这个问题的递归方法可以如程序清单 18-3 实现。

程序清单 18-3 RecursivePalindromeUsingSubstring.java

```
1 public class RecursivePalindromeUsingSubstring {
2     public static boolean isPalindrome(String s) {
3         if (s.length() <= 1) // Base case
4             return true;
5         else if (s.charAt(0) != s.charAt(s.length() - 1)) // Base case
6             return false;
7         else
8             return isPalindrome(s.substring(1, s.length() - 1));
9     }
10
11     public static void main(String[] args) {
12         System.out.println("Is moon a palindrome? "
13             + isPalindrome("moon"));
14         System.out.println("Is noon a palindrome? "
15             + isPalindrome("noon"));
16         System.out.println("Is a a palindrome? " + isPalindrome("a"));
17         System.out.println("Is aba a palindrome? " +
18             isPalindrome("aba"));
19         System.out.println("Is ab a palindrome? " + isPalindrome("ab"));
20     }
21 }
```

```
Is moon a palindrome? false
Is noon a palindrome? true
Is a a palindrome? true
Is aba a palindrome? true
Is ab a palindrome? false
```

第 8 行的 `substring` 方法创建了一个新字符串，它除了没有原始字符串中的第一个和最后一个字符，其余都是和原始字符串一样的。如果原始字符串中的两端字符相同，那么检查

一个字符串是否是回文串等价于检查子串是否是回文串。

复习题

- 18.10 描述递归方法的特点。
- 18.11 对于程序清单 18-3 中的 `isPalindrome` 方法，什么是基础情况？当调用 `isPalindrome("abdxcxdba")` 时，该方法被调用多少次？
- 18.12 使用程序清单 18-3 中定义的方法，给出 `isPalindrome("abcba")` 的调用栈。

18.5 递归辅助方法

要点提示：有时候可以通过针对要解决的初始问题的类似问题定义一个递归方法，来找到初始问题的解决方法。这个新的方法称为递归辅助方法。初始问题可以通过调用递归辅助方法来得到解决。

因为程序清单 18-3 中的 `isPalindrome` 方法要为每次递归调用创建一个新字符串，因此它不够高效。为避免创建新字符串，可以使用 `low` 和 `high` 下标来表明子串的范围。这两个下标必须传递给递归方法。由于原始方法是 `isPalindrome(String s)`，因此，必须产生一个新方法 `isPalindrome(String s,int low,int high)` 来接收关于字符串的额外信息，如程序清单 18-4 所示。

程序清单 18-4 RecursivePalindrome.java

```
1 public class RecursivePalindrome {
2     public static boolean isPalindrome(String s) {
3         return isPalindrome(s, 0, s.length() - 1);
4     }
5
6     private static boolean isPalindrome(String s, int low, int high) {
7         if (high <= low) // Base case
8             return true;
9         else if (s.charAt(low) != s.charAt(high)) // Base case
10            return false;
11        else
12            return isPalindrome(s, low + 1, high - 1);
13    }
14
15    public static void main(String[] args) {
16        System.out.println("Is moon a palindrome? "
17            + isPalindrome("moon"));
18        System.out.println("Is noon a palindrome? "
19            + isPalindrome("noon"));
20        System.out.println("Is a a palindrome? " + isPalindrome("a"));
21        System.out.println("Is aba a palindrome? " + isPalindrome("aba"));
22        System.out.println("Is ab a palindrome? " + isPalindrome("ab"));
23    }
24 }
```

程序中定义了两个重载的 `isPalindrome` 方法。第一个方法 `isPalindrome(String s)` 检查一个字符串是否是回文串，而第二个方法 `isPalindrome(String s,int low,int high)` 检查一个子串 `s(low..high)` 是否是回文串。第一个方法将 `low=0` 和 `high=s.length()-1` 的字符串 `s` 传递给第二个方法。第二个方法采用递归调用，检查不断缩减的子串是否是回文串。在递归程序设计中定义第二个方法来接收附加的参数是一个常用的设计技巧，这样的方法称为递归辅助方法 (recursive helper method)。

辅助方法在设计关于字符串和数组问题的递归方案上是非常有用的。下面将给出另外两

个例子。

18.5.1 递归选择排序

在 7.11 节中已经介绍过选择排序。回顾一下，选择排序法是先找到列表的最小数，并和第一个元素交换。然后，在剩余的数中找到最小数，再将它和剩余列表中的第一个元素交换，这样的过程一直进行下去，直到列表中仅剩一个数为止。这个问题可以分解为两个子问题：

- 找出列表中的最小数，然后将它与第一个数进行交换。
- 忽略第一个数，对余下的较小一些的列表进行递归排序。

基础情况是该列表只包含一个数。程序清单 18-5 给出了递归的排序方法。

程序清单 18-5 RecursiveSelectionSort.java

```
1 public class RecursiveSelectionSort {
2     public static void sort(double[] list) {
3         sort(list, 0, list.length - 1); // Sort the entire list
4     }
5
6     private static void sort(double[] list, int low, int high) {
7         if (low < high) {
8             // Find the smallest number and its index in list[low .. high]
9             int indexOfMin = low;
10            double min = list[low];
11            for (int i = low + 1; i <= high; i++) {
12                if (list[i] < min) {
13                    min = list[i];
14                    indexOfMin = i;
15                }
16            }
17
18            // Swap the smallest in list[low .. high] with list[low]
19            list[indexOfMin] = list[low];
20            list[low] = min;
21
22            // Sort the remaining list[low+1 .. high]
23            sort(list, low + 1, high);
24        }
25    }
26 }
```

程序中定义了两个重载的 sort 方法。第一个方法 sort(double[] list) 对数组 list[0.. list.length-1] 进行排序，而第二个方法 sort(double[] list, int low, int high) 对数组 list[low..high] 进行排序。第二个方法采用递归调用，对不断变小的子数组进行排序。

18.5.2 递归二分查找

在 7.10.2 节中介绍过二分查找。使用二分查找的前提条件是数组元素必须已经排好序。二分查找法首先将关键字与数组的中间元素进行比较，考虑下面三种情况。

- 情况 1：如果关键字比中间元素小，那么只需在前一半数组元素中进行递归查找。
- 情况 2：如果关键字和中间元素相等，则匹配成功，查找结束。
- 情况 3：如果关键字比中间元素大，那么只需在后一半数组元素中进行递归查找。

情况 1 和情况 3 都将查找范围降为一个更小的数列。而当匹配成功时，情况 2 就是一个基础情况。另一个基础情况是查找完毕而没有一个成功的匹配。程序清单 18-6 使用递归给

二分查找问题提供了一个清晰、简单的解决方案。

程序清单 18-6 Recursive Binary Search Method

```

1 public class RecursiveBinarySearch {
2     public static int recursiveBinarySearch(int[] list, int key) {
3         int low = 0;
4         int high = list.length - 1;
5         return recursiveBinarySearch(list, key, low, high);
6     }
7
8     private static int recursiveBinarySearch(int[] list, int key,
9         int low, int high) {
10        if (low > high) // The list has been exhausted without a match
11            return -low - 1;
12
13        int mid = (low + high) / 2;
14        if (key < list[mid])
15            return recursiveBinarySearch(list, key, low, mid - 1);
16        else if (key == list[mid])
17            return mid;
18        else
19            return recursiveBinarySearch(list, key, mid + 1, high);
20    }
21 }

```

第一个方法在整个数列中查找关键字。第二个方法是在数列下标从 low 到 high 的数列中查找关键字。

第一个 binarySearch 方法是将 low=0 和 high=list.length-1 的初始数组传递给第二个 binarySearch 方法。第二个方法采用递归调用，在不断变小的子数组中查找关键字。

复习题

- 18.13 使用程序清单 18-4 中定义的方法，给出 isPalindrome("abcba") 的调用栈。
- 18.14 使用程序清单 18-5 中定义的方法，给出 selectionSort(new double[]{2,3,5,1}) 的调用栈。
- 18.15 什么是递归辅助方法？

18.6 示例学习：得到目录的大小

要点提示：对于具有递归结构的问题，采用递归方法求解更高效。

前面的例子不用递归也很容易解决。本节提出一个不使用递归很难解决的问题，即求出一个目录的大小。一个目录的大小是指该目录下所有文件大小之和。目录 d 可能会包含子目录。假设一个目录包含文件 f_1, f_2, \dots, f_m 以及子目录 d_1, d_2, \dots, d_n ，如图 18-5 所示。

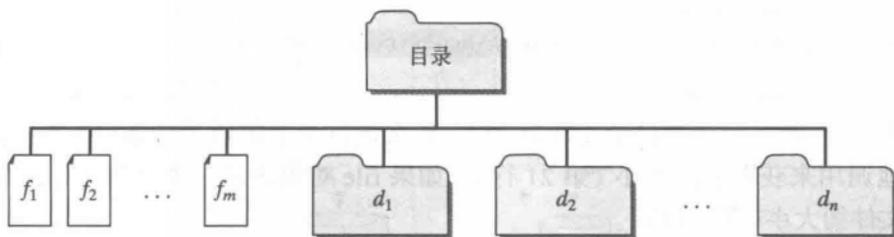


图 18-5 一个目录包含文件和子目录

目录的大小可以如下递归地定义：

$$size(d) = size(f_1) + size(f_2) + \dots + size(f_m) + size(d_1) + size(d_2) + \dots + size(f_n)$$

12.10 节介绍的 File 类可以用来表示一个文件或一个目录，并且获取文件和目录的属性。File 类中的两个方法对这个问题是很有用的：

- length() 方法返回一个文件的大小。
- listFiles() 方法返回一个目录下的 File 对象构成的数组。

程序清单 18-7 给出一个程序，提示用户输入一个目录或一个文件，然后显示它的大小。

程序清单 18-7 DirectorySize.java

```

1  import java.io.File;
2  import java.util.Scanner;
3
4  public class DirectorySize {
5      public static void main(String[] args) {
6          // Prompt the user to enter a directory or a file
7          System.out.print("Enter a directory or a file: ");
8          Scanner input = new Scanner(System.in);
9          String directory = input.nextLine();
10
11         // Display the size
12         System.out.println(getSize(new File(directory)) + " bytes");
13     }
14
15     public static long getSize(File file) {
16         long size = 0; // Store the total size of all files
17
18         if (file.isDirectory()) {
19             File[] files = file.listFiles(); // All files and subdirectories
20             for (int i = 0; files != null && i < files.length; i++) {
21                 size += getSize(files[i]); // Recursive call
22             }
23         }
24         else { // Base case
25             size += file.length();
26         }
27
28         return size;
29     }
30 }

```

```

Enter a directory or a file: c:\book 
48619631 bytes

```

```

Enter a directory or a file: c:\book\Welcome.java 
172 bytes

```

```

Enter a directory or a file: c:\book\NonExistentFile 
0 bytes

```

如果 file 对象表示一个目录（第 18 行），那么该目录下的每个子条目（文件或子目录）都被递归地调用来获取它的大小（第 21 行）。如果 file 对象表示一个文件（第 24 行），获取的就是该文件的大小（第 25 行）。

如果输入的是一个错误的目录或者不存在的目录，会发生什么情况呢？该程序将会发现它不是一个目录，并且调用 file.length()（第 25 行），它会返回 0。因此，在这种情况下，getSize 方法将返回 0。

 **提示：**为了避免错误，测试基本状态是一个很好的尝试。例如，应该输入一个文件、一个空目录、一个不存在的目录以及一个不存在的文件来测试这个程序。

复习题

18.16 `getSize` 方法的基础情况是什么？

18.17 程序是如何得到一个给定目录下所有的文件和目录的？

18.18 如果一个目录具有三个子目录，每个子目录具有四个文件，`getSize` 方法将调用多少次？

18.19 如果目录为空的话（即，不包含任何文件），程序可以工作吗？

18.20 如果第 20 行替换成以下代码，程序可以工作吗？

```
for (int i = 0; i < files.length; i++)
```

18.21 如果第 20 ~ 21 行替换成以下代码的话，程序可以工作吗？

```
for (File file: files)
    size += getSize(file); // Recursive call
```

18.7 示例学习：汉诺塔

 **要点提示：**汉诺塔问题是一个经典的递归例子。用递归可以很容易地解决这个问题，但是，不使用递归则非常难解决。

这个问题是将指定个数而大小互不相同的盘子从一个塔移到另一个塔上，移动要遵从下面的规则：

- n 个盘子标记为 1, 2, 3, ..., n ，而三个塔标记为 A、B 和 C。
- 任何时候盘子都不能放在比它小的盘子的上方。
- 初始状态时，所有的盘子都放在塔 A 上。
- 每次只能移动一个盘子，并且这个盘子必须在塔顶位置。

这个问题的目标是借助塔 C 把所有的盘子从塔 A 移到塔 B。例如，如果有三个盘子，将所有的盘子从 A 移到 B 的步骤如图 18-6 所示。

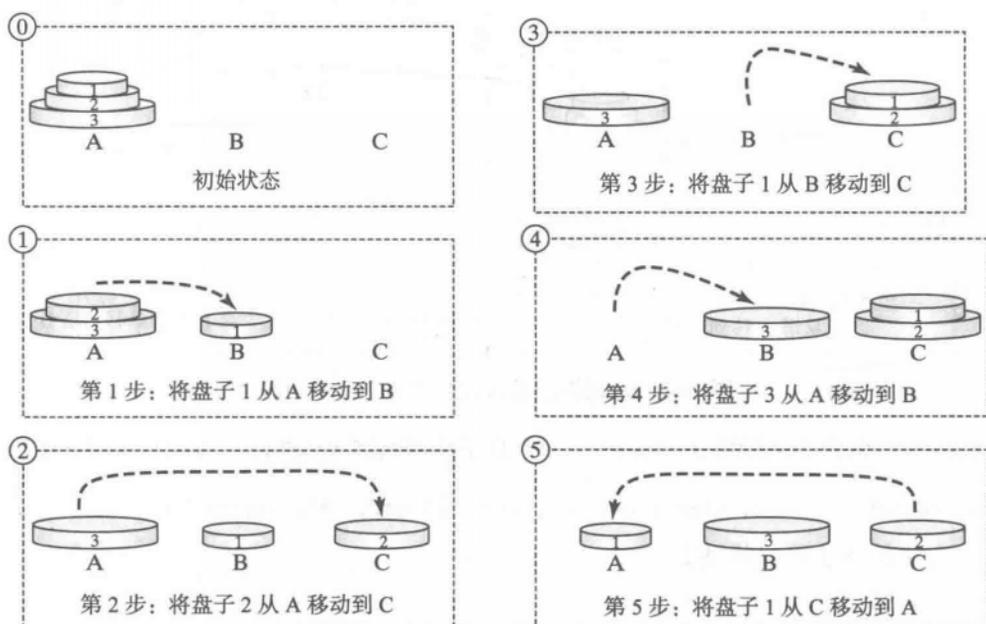


图 18-6 汉诺塔问题的目的是在遵从规则的条件下把盘子从塔 A 移到塔 B

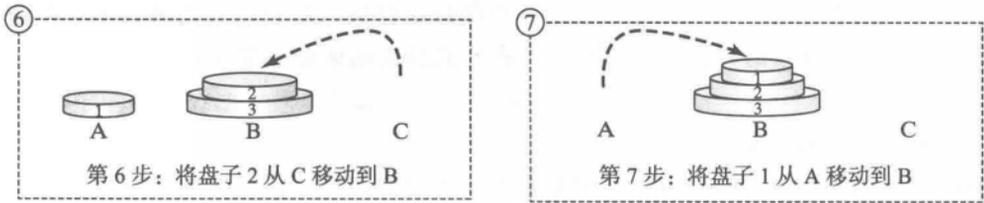


图 18-6 (续)

注意：汉诺塔是一个经典的计算机科学问题。许多网站都有关于该问题的解法。其中很值得参考的网站是 www.cut-the-knot.com/recurrence/hanoi.shtml。

在三个盘子的情况下，可以手动地找出解决方案。然而，当盘子数量较大时，即使是四个，这个问题还是非常复杂的。幸运的是，这个问题本身就具有递归性质，可以得到直观的递归解法。

问题的基础情况是 $n=1$ 。若 $n==1$ ，就可以简单地把盘子从 A 移到 B。当 $n>1$ 时，可以将原始问题拆成下面三个子问题，然后依次解决。

- 1) 借助塔 B 将前 $n-1$ 个盘子从 A 移到 C，如图 18-7 中的步骤 1 所示。
- 2) 将盘子 n 从 A 移到 B，如图 18-7 中的步骤 2 所示。
- 3) 借助塔 A 将 $n-1$ 个盘子从 C 移到 B，如图 18-7 中的步骤 3 所示。

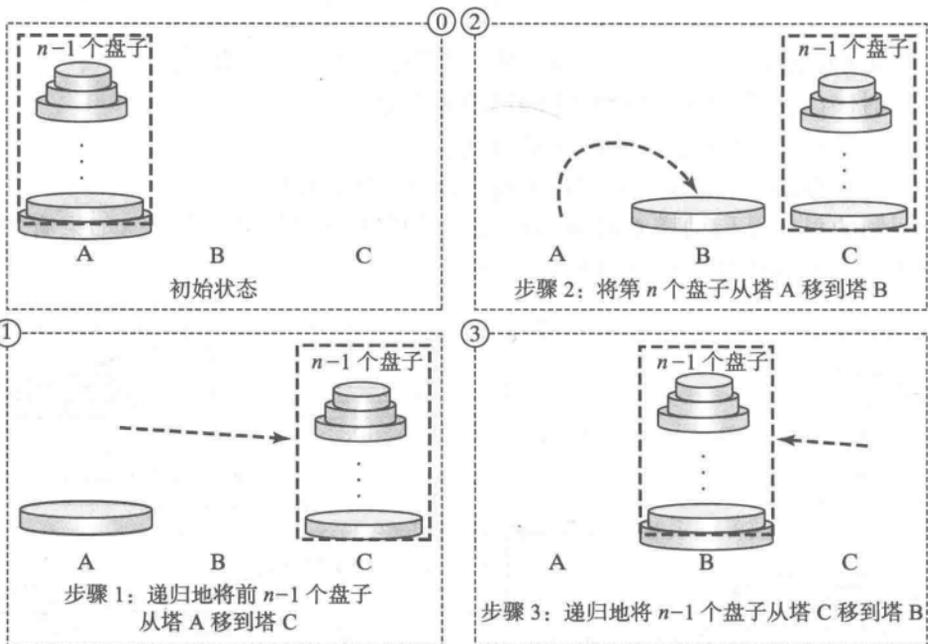


图 18-7 汉诺塔问题可以分解成三个子问题

下面的方法借助于辅助塔 `auxTower` 将 n 个盘子从原始塔 `fromTower` 移到目标塔 `toTower` 上：

```
void moveDisks(int n, char fromTower, char toTower, char auxTower)
```

这个方法的算法可以描述如下：

```
if (n == 1) // Stopping condition
    Move disk 1 from the fromTower to the toTower;
else {
```

```
    moveDisks(n - 1, fromTower, auxTower, toTower);
    Move disk n from the fromTower to the toTower;
    moveDisks(n - 1, auxTower, toTower, fromTower);
}
```

程序清单 18-8 给出一个程序，提示用户输入盘子个数，然后调用递归的方法 `moveDisks` 来显示移动盘子的解决方案。

程序清单 18-8 TowerOfHanoi.java

```
1 import java.util.Scanner;
2
3 public class TowerOfHanoi {
4     /** Main method */
5     public static void main(String[] args) {
6         // Create a Scanner
7         Scanner input = new Scanner(System.in);
8         System.out.print("Enter number of disks: ");
9         int n = input.nextInt();
10
11         // Find the solution recursively
12         System.out.println("The moves are:");
13         moveDisks(n, 'A', 'B', 'C');
14     }
15
16     /** The method for finding the solution to move n disks
17      from fromTower to toTower with auxTower */
18     public static void moveDisks(int n, char fromTower,
19     char toTower, char auxTower) {
20         if (n == 1) // Stopping condition
21             System.out.println("Move disk " + n + " from " +
22             fromTower + " to " + toTower);
23         else {
24             moveDisks(n - 1, fromTower, auxTower, toTower);
25             System.out.println("Move disk " + n + " from " +
26             fromTower + " to " + toTower);
27             moveDisks(n - 1, auxTower, toTower, fromTower);
28         }
29     }
30 }
```

```
Enter number of disks: 4  Enter
The moves are:
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
Move disk 4 from A to B
Move disk 1 from C to B
Move disk 2 from C to A
Move disk 1 from B to A
Move disk 3 from C to B
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
```

这个问题本质上是递归的。利用递归就能够找到一个自然、简单的解决方案。如果不使

用递归，解决这个问题将会很困难。

考虑跟踪 $n=3$ 的程序。连续的递归调用如图 18-8 所示。可见，编写这个程序比跟踪这个递归调用要容易些。系统使用栈来管理后台的调用。从某种程度上讲，递归提供了某种层次的抽象，这种抽象对用户隐藏迭代和其他细节。

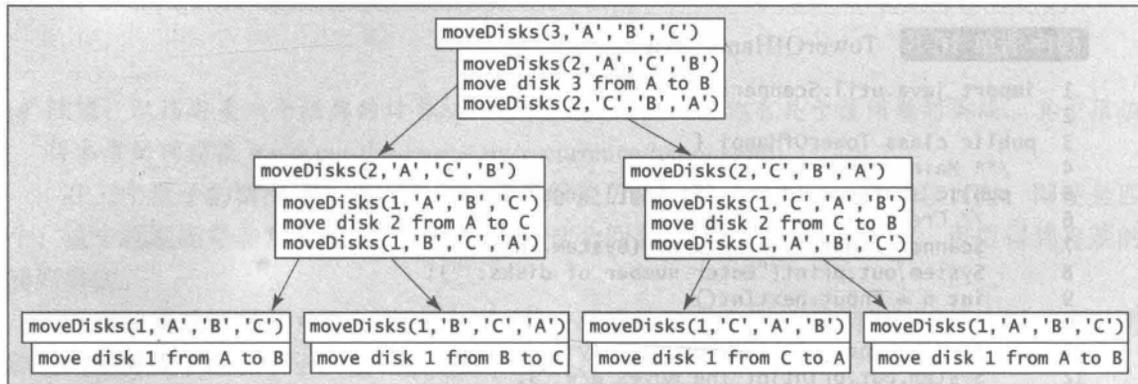


图 18-8 调用 `moveDisks(3, 'A', 'B', 'C')` 会引起对 `moveDisks` 的递归调用

复习题

18.22 程序清单 18-8 中调用 `moveDisks(5, 'A', 'B', 'C')` 的话，将会调用 `moveDisks` 方法多少次？

18.8 示例学习：分形

要点提示：递归是显示分形的理想方法，因为分形本身就具有递归特性。

分形是一个几何图形，但是它不像三角形、圆形和矩形。分形可以分成几个部分，每部分都是整体的一个缩小的副本。分形有许多有趣的例子。本节介绍一个称为思瑞平斯基三角形 (Sierpinski triangle) 的简单分形，它是以一位著名的波兰数学家的名字来命名的。

思瑞平斯基三角形是如下创建的：

- 1) 从一个等边三角形开始，将它作为 0 阶 (或 0 级) 的思瑞平斯基分形，如图 18-9a 所示。
- 2) 将 0 阶三角形的各边中点连接起来产生 1 阶思瑞平斯基三角形 (图 18-9b)。
- 3) 保持中间的三角形不变，将另外三个三角形各边的中点连接起来产生 2 阶思瑞平斯基分形 (图 18-9c)。
- 4) 可以递归地重复同样的步骤产生 3 阶，4 阶， \dots ， n 阶的思瑞平斯基三角形 (图 18-9d)。

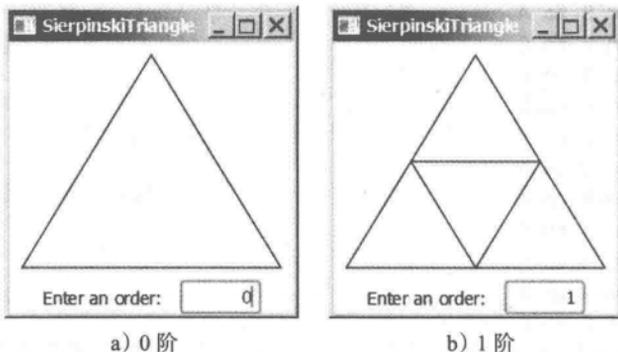


图 18-9 思瑞平斯基三角形是一种递归三角形的图形

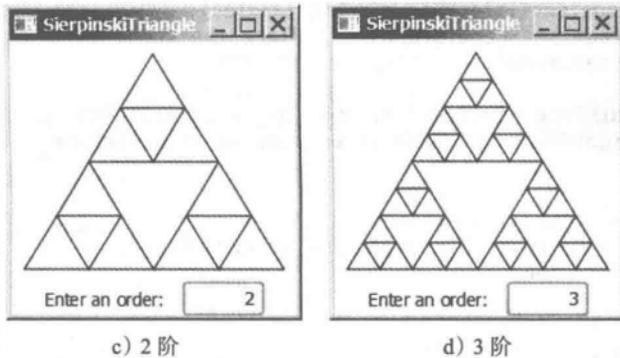


图 18-9 (续)

这个问题本质上是递归的。那么，该如何解答这个递归问题呢？考虑阶数为 0 的基础情况。这时，能够很容易地绘制出 0 阶思瑞平斯基三角形。如何绘制出 1 阶思瑞平斯基三角形呢？这个问题可以简化为绘制三个 0 阶思瑞平斯基三角形。如何绘制 2 阶思瑞平斯基三角形呢？这个问题可以简化为绘制三个 1 阶思瑞平斯基三角形。因此，绘制 n 阶思瑞平斯基三角形可以简化为绘制三个 $n-1$ 阶思瑞平斯基三角形。

程序清单 18-9 给出显示任意阶的思瑞平斯基三角形的程序，如图 18-9 所示。用户可以在文本域输入阶数，然后显示这个指定阶数的思瑞平斯基三角形。

程序清单 18-9 SierpinskiTriangle.java

```

1  import javafx.application.Application;
2  import javafx.geometry.Point2D;
3  import javafx.geometry.Pos;
4  import javafx.scene.Scene;
5  import javafx.scene.control.Label;
6  import javafx.scene.control.TextField;
7  import javafx.scene.layout.BorderPane;
8  import javafx.scene.layout.HBox;
9  import javafx.scene.layout.Pane;
10 import javafx.scene.paint.Color;
11 import javafx.scene.shape.Polygon;
12 import javafx.stage.Stage;
13
14 public class SierpinskiTriangle extends Application {
15     @Override // Override the start method in the Application class
16     public void start(Stage primaryStage) {
17         SierpinskiTrianglePane trianglePane = new SierpinskiTrianglePane();
18         TextField tfOrder = new TextField();
19         tfOrder.setOnAction(
20             e -> trianglePane.setOrder(Integer.parseInt(tfOrder.getText())));
21         tfOrder.setPrefColumnCount(4);
22         tfOrder.setAlignment(Pos.BOTTOM_RIGHT);
23
24         // Pane to hold label, text field, and a button
25         HBox hBox = new HBox(10);
26         hBox.getChildren().addAll(new Label("Enter an order: "), tfOrder);
27         hBox.setAlignment(Pos.CENTER);
28
29         BorderPane borderPane = new BorderPane();
30         borderPane.setCenter(trianglePane);
31         borderPane.setBottom(hBox);
32
33         // Create a scene and place it in the stage
34         Scene scene = new Scene(borderPane, 200, 210);

```

```

35 primaryStage.setTitle("SierpinskiTriangle"); // Set the stage title
36 primaryStage.setScene(scene); // Place the scene in the stage
37 primaryStage.show(); // Display the stage
38
39 scene.widthProperty().addListener(ov -> trianglePane.paint());
40 scene.heightProperty().addListener(ov -> trianglePane.paint());
41 }
42
43 /** Pane for displaying triangles */
44 static class SierpinskiTrianglePane extends Pane {
45     private int order = 0;
46
47     /** Set a new order */
48     public void setOrder(int order) {
49         this.order = order;
50         paint();
51     }
52
53     SierpinskiTrianglePane() {
54     }
55
56     protected void paint() {
57         // Select three points in proportion to the pane size
58         Point2D p1 = new Point2D(getWidth() / 2, 10);
59         Point2D p2 = new Point2D(10, getHeight() - 10);
60         Point2D p3 = new Point2D(getWidth() - 10, getHeight() - 10);
61
62         this.getChildren().clear(); // Clear the pane before redisplay
63
64         displayTriangles(order, p1, p2, p3);
65     }
66
67     private void displayTriangles(int order, Point2D p1,
68         Point2D p2, Point2D p3) {
69         if (order == 0) {
70             // Draw a triangle to connect three points
71             Polygon triangle = new Polygon();
72             triangle.getPoints().addAll(p1.getX(), p1.getY(), p2.getX(),
73                 p2.getY(), p3.getX(), p3.getY());
74             triangle.setStroke(Color.BLACK);
75             triangle.setFill(Color.WHITE);
76
77             this.getChildren().add(triangle);
78         }
79         else {
80             // Get the midpoint on each edge in the triangle
81             Point2D p12 = p1.midpoint(p2);
82             Point2D p23 = p2.midpoint(p3);
83             Point2D p31 = p3.midpoint(p1);
84
85             // Recursively display three triangles
86             displayTriangles(order - 1, p1, p12, p31);
87             displayTriangles(order - 1, p12, p2, p23);
88             displayTriangles(order - 1, p31, p23, p3);
89         }
90     }
91 }
92 }

```

初始三角形有三个与面板大小成比例的点集(第 58 ~ 60 行)。如果 `order==0`, `displayTriangle(order,p1,p2,p3)` 方法显示一个链接三个点 `p1`、`p2` 和 `p3` 的三角形, 见代码第 71 ~ 77 行, 如图 18-10a 所示。否则, 程序执行下列任务:

1) 获取 $p1$ 和 $p2$ 的中点 (第 81 行), $p2$ 和 $p3$ 的中点 (第 82 行), 以及 $p3$ 和 $p1$ 的中点 (第 83 行), 如图 18-10b 所示。

2) 使用递减的阶数来递归地调用 `displayTriangles`, 以显示三个更小的思瑞平斯基三角形 (第 86 ~ 88 行)。注意, 每个小的思瑞平斯基三角形除了阶数会少一个之外, 其结构和原始的大思瑞平斯基三角形是一样的, 如图 18-10b 所示。

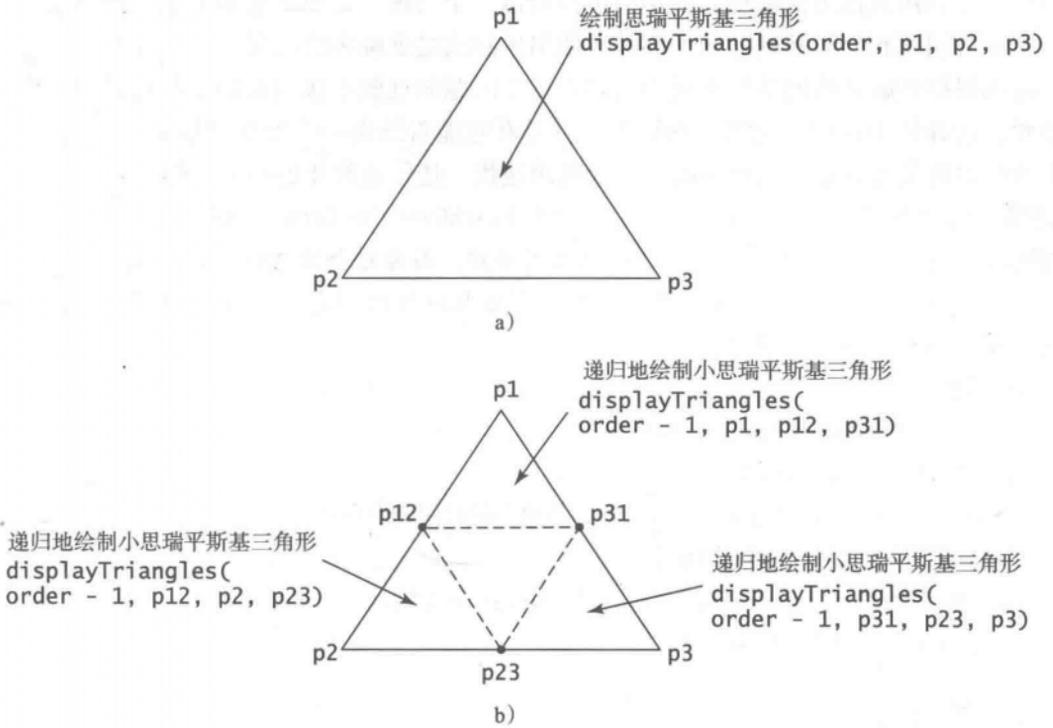


图 18-10 绘制一个思瑞平斯基三角形会发起对绘制三个小的思瑞平斯基三角形的调用

在 `SierpinskiTrianglePanel` 中显示思瑞平斯基三角形。内部类 `SierpinskiTrianglePanel` 中的 `order` 属性表明思瑞平斯基三角形的阶数。9.8 节中介绍过的 `Point2D` 类表示一个具有 x 和 y 坐标值的点。`p1.midpoint(p2)` 方法返回 $p1$ 和 $p2$ 的中点 `Point2D` 对象 (第 81 ~ 83 行)。

复习题

- 18.23 如何得到两个点的中点?
- 18.24 `displayTriangles` 方法的基础情况是什么?
- 18.25 对于 0 阶, 1 阶, 2 阶, 以及 n 阶的思瑞平斯基三角形, 将分别调用多少次 `displayTriangles` 方法?
- 18.26 如果输入一个负数阶值, 将发生什么? 如何修正代码中的这个问题?
- 18.27 重写代码第 71 ~ 77 行, 绘制三条连接点的线段来绘制三角形, 取代原来使用绘制多边形的方法来绘制的方法。

18.9 递归与迭代

要点提示: 递归是程序控制的另一种形式, 实质上就是不用循环控制的重复。

使用循环时, 可以指定一个循环体。循环控制结构控制循环体的重复。在递归中, 方法

重复地调用自己。必须使用一条选择语句来控制是否继续递归调用该方法。

递归会产生相当大的系统开销。程序每调用一个方法，系统就要给方法中所有的局部变量和参数分配空间。这就要占用大量的内存，还需要额外的时间来管理内存。

任何用递归解决的问题都可以用非递归的迭代解决。递归有很多副作用：它耗费了太多时间并占用了太多内存。那么，为什么还要用它呢？因为在某些情况下，本质上有递归特性的问题很难用其他方法解决，而递归可以给出一个清晰、简单的解决方案。像目录大小问题、汉诺塔问题和分形问题的例子都是不使用递归就很难解决的问题。

应该根据要解决的问题的本质和我们对这个问题的理解来决定是用递归还是用迭代。根据经验，选择使用递归还是迭代的原则，就是看它能否给出一个反映问题本质的直观解法。如果迭代的解决方案是显而易见的，那就使用迭代。迭代通常比递归效率更高。

注意：递归程序可能会用完内存，引起一个 `StackOverflowError` 错误。

提示：如果关注程序的性能，就要避免使用递归，因为它会比迭代占用更多的时间且浪费更多的内存。通常，递归用于解决本质上有递归特性的问题，例如汉诺塔问题、递归目录，以及思瑞平斯基三角形。

复习题

18.28 下面的语句中哪些是正确的：

- 任何递归方法都可以转换为非递归方法。
- 执行递归方法比执行非递归方法要占用更多的时间和内存。
- 递归方法总是比非递归方法简单一些。
- 递归方法中总是有一个选择语句检查是否达到基础情况。

18.29 引起栈溢出异常的原因是什么？

18.10 尾递归

要点提示：尾递归对于减少栈的大小比较有效。

如果在从递归调用返回时没有继续的操作要完成，那么这个递归方法就称为尾递归 (tail recursive)，如图 18-11a 所示。然而，图 18-11b 中的方法 B 就不是尾递归，因为方法调用返回后还有继续要执行的操作。

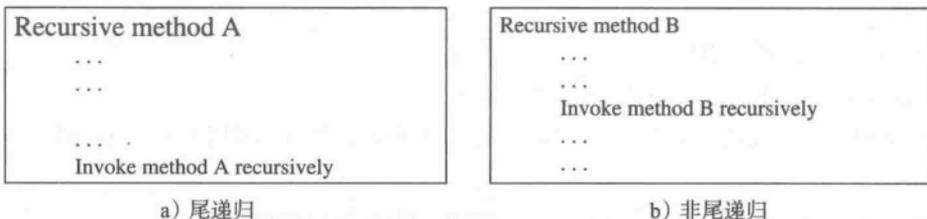


图 18-11 尾递归方法在递归调用后没有继续要执行的操作

例如，因为在程序清单 18-4 中的第 12 行递归调用 `isPalindrome` 之后没有后续的操作，所以，递归的 `isPalindrome` 方法（第 6 ~ 13 行）就是尾递归的。但是，在程序清单 18-1 中，因为从每个递归调用返回时都有一个名为 `multiplication` 的后续操作要完成，所以，递归的 `factorial` 方法（第 16 ~ 21 行）就不是尾递归的。

尾递归更可取：因为当最后一个递归调用结束时，方法也结束，因此，无须将中间的调

用存储在栈中。编译器可以优化尾递归以减小栈空间。

通常，可以使用辅助参数将非尾递归方法转换为尾递归方法。这些参数被用于保存结果。思路是将后续的操作以一种方式结合到辅助参数中，这样递归调用中将不再有后续操作。可以定义一个带辅助参数的新的辅助递归方法，这个方法可以重载原始方法，具有相同的名字而签名不同。例如，程序清单 18-1 中的 `factorial` 方法可以写成尾递归形式，如代码清单 18-10 所示。

程序清单 18-10 ComputeFactorialTailRecursion.java

```

1 public class ComputeFactorialTailRecursion {
2     /** Return the factorial for a specified number */
3     public static long factorial(int n) {
4         return factorial(n, 1); // Call auxiliary method
5     }
6
7     /** Auxiliary tail-recursive method for factorial */
8     private static long factorial(int n, int result) {
9         if (n == 0)
10            return result;
11        else
12            return factorial(n - 1, n * result); // Recursive call
13    }
14 }

```

第一个 `factorial` 方法（第 3 行）只是简单调用了第二个辅助方法（第 4 行）。第二个方法包括了一个辅助参数 `result`，它存储了 `n` 的阶乘的结果。这个方法在第 12 行被递归地调用。在调用返回之后，就没有了后续的操作。最终的结果在第 10 行返回，它也是在第 4 行调用 `factorial(n, 1)` 的返回值。

复习题

- 18.30 指出本章中的尾递归方法。
 18.31 使用尾递归重写程序清单 18-2 中的 `fib` 方法。

关键术语

base case (基础情况)	recursive helper method (递归辅助方法)
direct recursion(直接递归)	recursive method (递归方法)
indirect recursion(间接递归)	stopping condition (终止条件)
infinite recursion (无限递归)	tail recursion (尾递归)

本章小结

1. 递归方法是一个直接或间接调用自己的方法。要终止一个递归方法，必须有一个或多个基础情况。
2. 递归是程序控制的另外一种形式。本质上它是没有循环控制的重复。对于用其他方法很难解决而本质上是递归的问题，使用递归可以给出简单、清楚的解决方案。
3. 为了进行递归调用，有时候需要修改原始方法使其接收附加的参数。为达到这个目的，可以定义递归辅助方法。
4. 递归需要相当大的系统开销。程序每调用一个方法一次，系统必须给方法中所有的局部变量和参数分配空间。这就要消耗大量的内存，并且需要额外的时间来管理这些内存。
5. 如果从递归调用返回时没有后续的操作要完成，这个递归的方法就称为尾递归 (tail recursive)。某些编译器会优化尾递归以减少栈空间。

测试题

回答本章位于 www.cs.armstrong.edu/liang/intro10e/quiz.html 的测试题。

编程练习题

18.2 ~ 18.3 节

*18.1 (计算阶乘) 使用 10.9 节介绍的 `BigInteger` 类, 求得大数字的阶乘 (例如, $100!$)。使用递归实现 `factorial` 方法。编写一个程序, 提示用户输入一个整数, 然后显示它的阶乘。

*18.2 (斐波那契数) 使用迭代改写程序清单 18-2 中的 `fib` 方法。

提示: 不使用递归来计算 `fib(n)`, 首先要得到 `fib(n-2)` 和 `fib(n-1)`。设 `f0` 和 `f1` 表示前面的两个斐波那契数, 那么当前的斐波那契数就是 `f0+f1`。这个算法可以描述为如下所示:

```
f0 = 0; // For fib(0)
f1 = 1; // For fib(1)

for (int i = 1; i <= n; i++) {
    currentFib = f0 + f1;
    f0 = f1;
    f1 = currentFib;
}
// After the loop, currentFib is fib(n)
```

编写一个测试程序, 提示用户输入一个索引, 然后显示它的斐波那契数。

*18.3 (使用递归求最大公约数) 求最大公约数的 `gcd(m,n)` 方法也可以如下递归地定义:

- 如果 $m \% n$ 为 0, 那么 `gcd(m,n)` 的值为 `n`。
- 否则, `gcd(m,n)` 就是 `gcd(n,m%n)`。

编写一个递归的方法来求最大公约数。编写一个测试程序, 提示用户输入两个整数, 显示它们的最大公约数。

18.4 (对数列表求和) 编写一个递归方法来计算下面的级数:

$$m(i) = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{i}$$

编写一个测试程序, 为 $i=1,2,\dots,10$ 显示 `m(i)`。

18.5 (对数列表求和) 编写一个递归的方法来计算下面的级数:

$$m(i) = \frac{1}{3} + \frac{2}{5} + \frac{3}{7} + \frac{4}{9} + \frac{5}{11} + \frac{6}{13} + \dots + \frac{i}{2i+1}$$

编写一个测试程序, 为 $i=1,2,\dots,10$ 显示 `m(i)`。

*18.6 (对数列表求和) 编写一个递归的方法来计算下面的级数:

$$m(i) = \frac{1}{2} + \frac{2}{3} + \dots + \frac{i}{i+1}$$

编写一个测试程序, 为 $i=1,2,\dots,10$ 显示 `m(i)`。

*18.7 (斐波那契数列) 修改程序清单 18-2, 使程序可以找出调用 `fib` 方法的次数。

提示: 使用一个静态变量, 每当调用这个方法时, 该变量就加 1。

18.4 节

*18.8 (以逆序输出一个整数中的数字) 编写一个递归方法, 使用下面的方法头在控制台上以逆序显示一个 `int` 型的值:

```
public static void reverseDisplay(int value)
```

例如, `reverseDisplay(12345)` 显示的是 54321。编写一个测试程序, 提示用户输入一个

整数，然后显示它的逆序数字。

- *18.9 (以逆序输出一个字符串中的字符) 编写一个递归方法，使用下面的方法头在控制台上以逆序显示一个字符串：

```
public static void reverseDisplay(String value)
```

例如，`reverseDisplay("abcd")` 显示的是 `dcba`。编写一个测试程序，提示用户输入一个字符串，然后显示它的逆序字符串。

- *18.10 (字符串中某个指定字符出现的次数) 编写一个递归方法，使用下面的方法头给出一个指定字符在字符串中出现的次数。

```
public static int count(String str, char a)
```

例如，`count("Welcome", 'e')` 会返回 2。编写一个测试程序，提示用户输入一个字符串和一个字符，显示该字符在字符串中出现的次数。

- *18.11 (使用递归求一个整数各位数之和) 编写一个递归方法，使用下面的方法头计算一个整数中各位数之和：

```
public static int sumDigits(long n)
```

例如，`sumDigits(234)` 返回的是 $2+3+4=9$ 。编写一个测试程序，提示用户输入一个整数，然后显示各位数字之和。

18.5 节

- **18.12 (以逆序打印字符串中的字符) 使用辅助方法改写编程练习题 18.9，将子串的 `high` 下标传递给这个方法。辅助方法头为：

```
public static void reverseDisplay(String value, int high)
```

- *18.13 (找出数组中的最大数) 编写一个递归方法，返回一个数组中的最大整数。编写一个测试程序，提示用户输入一个包含 8 个整数的列表，然后显示最大的元素。
- *18.14 (求字符串中大写字母的个数) 编写一个递归方法，返回一个字符串中大写字母的个数。编写一个测试程序，提示用户输入一个字符串，然后显示该字符串中大写字母的数目。
- *18.15 (字符串中某个指定字符出现的次数) 使用辅助方法改写编程练习题 18.10，将子串的 `high` 下标传递给这个方法。辅助方法头为：

```
public static int count(String str, char a, int high)
```

- *18.16 (求数组中大写字母的个数) 编写一个递归的方法，返回一个字符数组中大写字母的个数。需要定义下面两个方法。第二个方法是一个递归辅助方法。

```
public static int count(char[] chars)
public static int count(char[] chars, int high)
```

编写一个测试程序，提示用户在一行中输入一个字符列表，然后显示该列表中大写字母的个数。

- *18.17 (数组中某个指定字符出现的次数) 编写一个递归的方法，求出数组中一个指定字符出现的次数。需要定义下面两个方法，第二个方法是一个递归的辅助方法。

```
public static int count(char[] chars, char ch)
public static int count(char[] chars, char ch, int high)
```

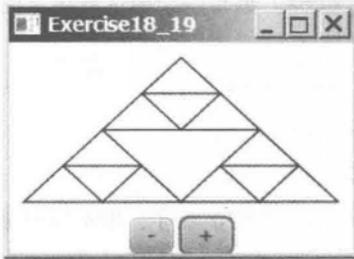
编写一个测试程序，提示用户在一行中输入一个字符列表以及一个字符，然后显示该字符在列表中出现的次数。

18.6 ~ 18.10 节

- *18.18 (汉诺塔) 修改程序清单 18-8，使程序可以计算将 n 个盘子从塔 A 移到塔 B 所需的移动次数。

 提示：使用一个静态变量，每当调用方法一次，该变量就加 1。

- *18.19 (思瑞平斯基三角形) 修改程序清单 18-9，开发一个程序，让用户使用“+”和“-”按钮将当前阶数增 1 或减 1，如图 18-12a 所示。初始阶数为 0。如果当前阶数为 0，就忽略“-”按钮。



a) 编程练习题 18.19 使用“+”和“-”按钮将当前阶数增加 1 或减小 1



b) 练习题 18.20 使用递归方法绘制多个圆

图 18-12

- *18.20 (显示多个圆) 编写一个 Java 程序显示多个圆，如图 18-12b 所示。这些圆都处于面板的中心位置。两个相邻圆之间相距 10 像素，面板和最大圆之间也相距 10 像素。

- *18.21 (将十进制数转换为二进制数) 编写一个递归方法，将一个十进制数转换为一个二进制数的字符串。方法头如下：

```
public static String dec2Bin(int value)
```

编写一个测试程序，提示用户输入一个十进制数，然后显示等价的二进制数。

- *18.22 (将十进制数转换为十六进制数) 编写一个递归方法，将一个十进制数转换为一个十六进制数的字符串。方法头如下：

```
public static String dec2Hex(int value)
```

编写一个测试程序，提示用户输入一个十进制数，然后显示等价的十六进制数。

- *18.23 (将二进制数转换为十进制数) 编写一个递归方法，将一个字符串形式的二进制数转换为一个十进制数。方法头如下：

```
public static int bin2Dec(String binaryString)
```

编写一个测试程序，提示用户输入一个二进制字符串，然后显示等价的十进制数。

- *18.24 (将十六进制数转换为十进制数) 编写一个递归方法，将一个字符串形式的十六进制数转换为一个十进制数。方法头如下：

```
public static int hex2Dec(String hexString)
```

编写一个测试程序，提示用户输入一个十六进制字符串，然后显示等价的十进制数。

- **18.25 (字符串排列) 编写一个递归方法，输出一个字符串的所有排列。例如，对于字符串 abc，输出为：

```
abc
acb
bac
bca
cab
cba
```

 提示：定义下面两个方法，第二个方法是一个辅助方法。

```
public static void displayPermutation(String s)
public static void displayPermutation(String s1, String s2)
```

第一个方法简单地调用 `displayPermutation("",s)`。第二个方法使用循环，将一个字符从 `s2` 移到 `s1`，并使用新的 `s1` 和 `s2` 递归地调用该方法。基础情况是 `s2` 为空，将 `s1` 打印到控制台。

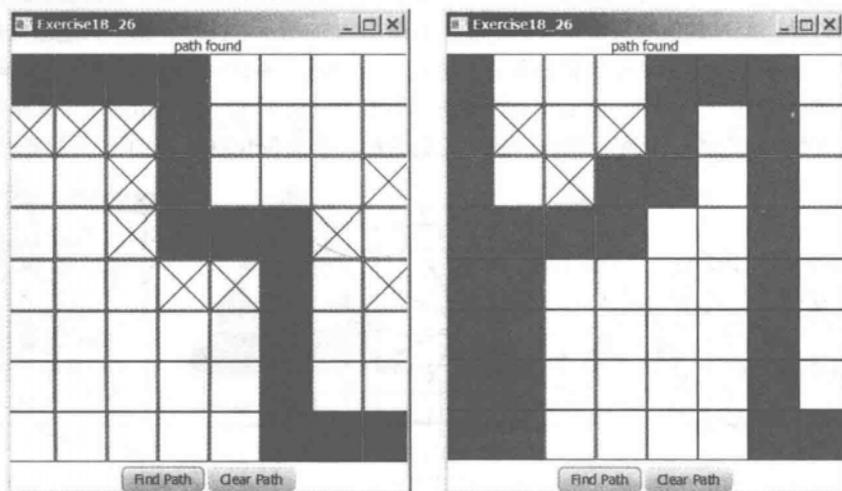
编写一个测试程序，提示用户输入一个字符串，然后显示其所有排列。

****18.26 (创建一个迷宫)** 编写一个程序，在迷宫中寻找一条路径，如图 18-13a 所示。该迷宫由一个 8×8 的棋盘表示。路径必须满足下列条件：

路径在迷宫的左上角单元和右下角单元之间。

程序允许用户在一个单元格中放入或移走一个标志。路径由相邻的未放标志的单元格组成。如果两个单元格在水平方向或垂直方向相邻，但在对角线方向上不相邻，那么就称它们是相邻的。

路径不包含能形成一个正方形的单元格。例如，在图 18-13b 中的路径就不满足这个条件。(这个条件使得面板上的路径很容易识别。)



a) 合法路径

b) 非法路径

图 18-13 程序求出从左上角到右下角的路径

****18.27 (科赫雪花分形)** 本章给出了思瑞平斯基三角形分形。本练习要编写一个程序，显示另一个称为科赫雪花 (Koch snowflake) 的分形，这是根据一位著名的瑞典数学家的名字命名的。科赫雪花按如下方式产生：

1) 从一个等边三角形开始，将其作为 0 阶 (或 0 级) 科赫分形，如图 18-14a 所示。

2) 三角形中的每条边分成三个相等的线段，以中间的线段作为底边向外画一个等边三角形，产生 1 阶科赫分形，如图 18-14b 所示。

3) 重复步骤 2 产生 2 阶科赫分形，3 阶科赫分形，...，如图 18-14c ~ d 所示。

***18.28 (非递归目录大小)** 不使用递归改写程序清单 18-7。

***18.29 (某个目录下的文件数目)** 编写一个程序，提示用户输入一个目录，然后显示该目录下的文件数。

****18.30 (找出单词)** 编写一个程序，递归地找出某个目录下的所有文件中某个单词出现的次数。从命令行如下传递参数：

```
java Exercise18_30 dirName word
```

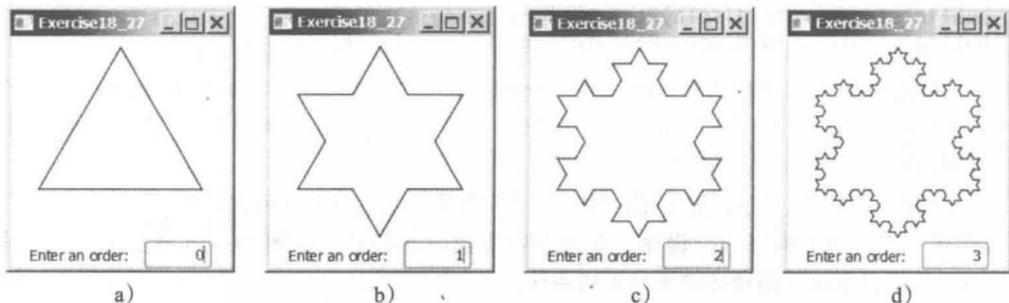
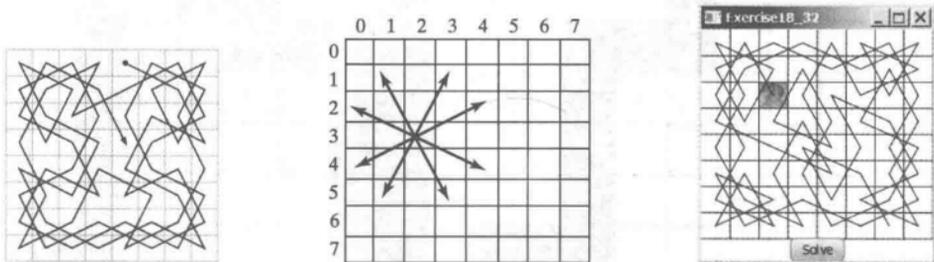


图 18-14 科赫雪花是一个从三角形开始的分形

****18.31 (替换单词)** 编写一个程序，递归地用一个新单词替换某个目录下的所有文件中出现的某个单词。从命令行如下传递参数：

```
java Exercisel8_31 dirName oldWord newWord
```

*****18.32 (游戏：骑士的旅途)** 骑士的旅途是一个古老的谜题。它的目的是使骑士从棋盘上的任意一个正方形开始移动，经过其他的每个正方形一次，如图 18-15a 所示。注意，骑士只能做 L 形的移动（两个空格在一个方向上而一个空格在垂直的方向上）。如图 18-15b 所示，骑士可以移动到八个正方形的位置。编写一个程序，显示骑士的移动，如图 18-15c 所示。当单击一个单元格的时候，骑士被放置在该单元格中。该单元格作为骑士的起始点。单击 Solve 按钮显示作为解答的路径。



a) 骑士遍历所有的正方形一次

b) 骑士作 L 形的移动

c) 程序显示骑士的旅途路径

图 18-15

提示：这个问题的穷举方法是将骑士从一个正方形随意地移动到另一个可用的正方形。使用这样的方法，程序将需要很多时间来完成。比较好的方法是采用一些启发式方法。依据骑士目前的位置，它可以有两个、三个、四个、六个或八个可能的移动线路。直觉上讲，应该首先尝试将骑士移动到可访问性最小的正方形，将那些更多的可访问的正方形保留为开放的，这样，在查找的结尾就会有更好的成功机会。

*****18.33 (游戏：骑士旅途的动画)** 为骑士旅途的问题编写一个程序，该程序应该允许用户将骑士放到任何一个起始正方形，并单击 Solve 按钮，用动画展示骑士沿着路径的移动，如图 18-16 所示。

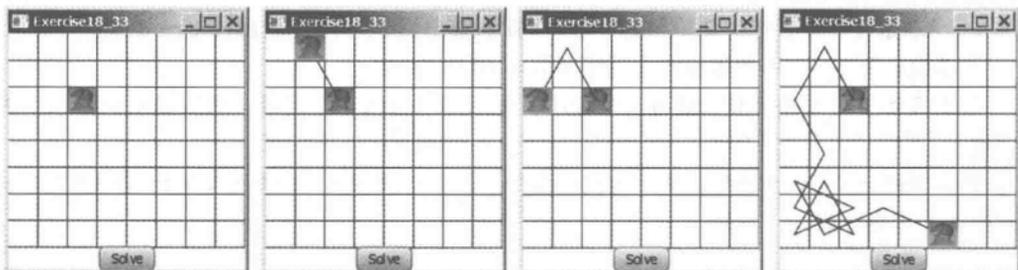


图 18-16 骑士沿着路径遍历

****18.34 (游戏：八皇后问题)** 八皇后问题是要找到一个解决方案，将一个皇后棋子放到棋盘上的每行中，并且两个皇后棋子之间不能相互攻击。编写一个程序，使用递归来解决八皇后问题，并如图 18-17 显示结果。



图 18-17 程序显示八皇后问题的求解

****18.35 (H-树分形)** 一个 H-树分形 (本章开始部分介绍过，如图 18-1) 如下定义：

1) 从字母 H 开始。H 的三条线长度一样，如图 18-1a 所示。

2) 字母 H (以它的 sans-serif 字体形式，H) 有四个端点。以这四个端点为中心位置绘制一个 1 阶 H 树，如图 18-1b 所示。这些 H 的大小是包括这四个端点的 H 的一半。

3) 重复步骤 2 来创建 2 阶，3 阶，...，n 阶的 H 树，如图 18-1c-d 所示。

编写程序，绘制如图 18-1 所示的 H-树。

18.36 (思瑞平斯基三角形) 编写一个程序，让用户输入一个阶数，然后显示填充的思瑞平斯基三角形，如图 18-18 所示。

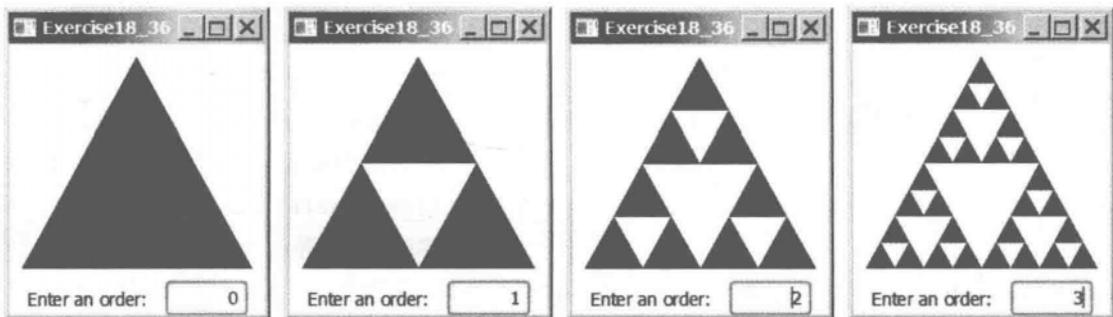


图 18-18 显示一个填充的思瑞平斯基三角形

****18.37 (希尔伯特曲线)** 希尔伯特曲线，由德国数学家希尔伯特于 1891 年第一个给出描述，是一种空间填充曲线，以 2×2 ， 4×4 ， 8×8 ， 16×16 ，或者任何其他 2 的幂的大小来访问一个方格网的每个点。编写程序，以给定的阶数显示希尔伯特曲线，如图 18-19 所示。

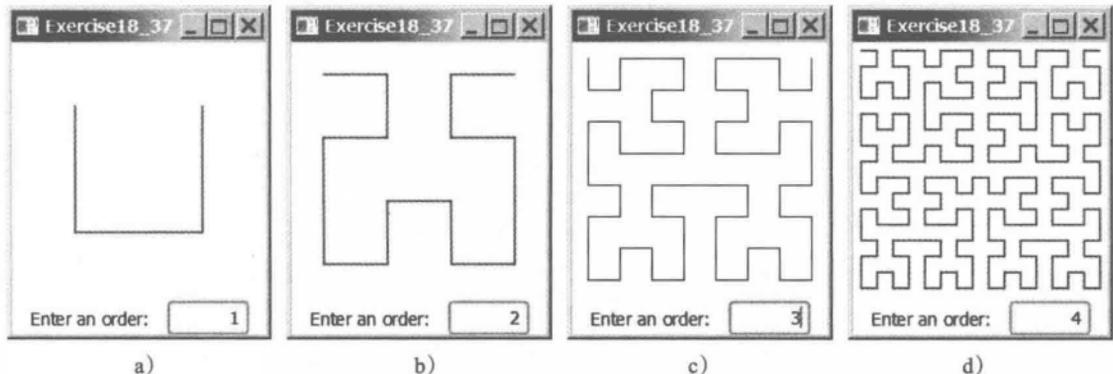


图 18-19 绘制给定阶数的希尔伯特曲线

****18.38 (递归树)** 编写一个程序来显示一个递归树，如图 18-20 所示。

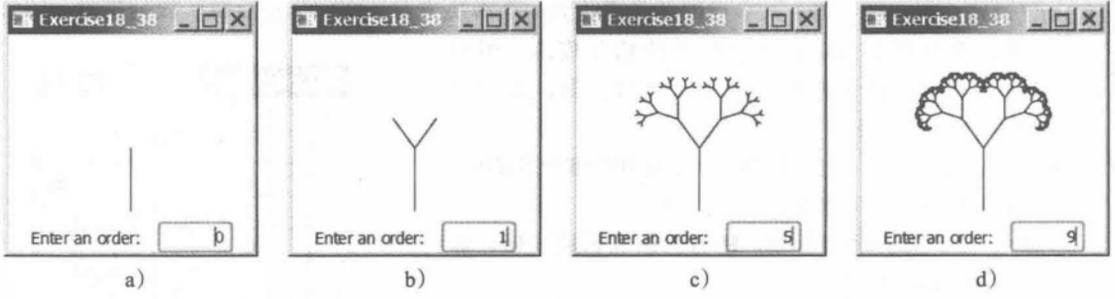


图 18-20 绘制一个带特定深度的递归树

**18.39 (拖动树) 修改编程练习题 18.38, 将树移动到鼠标所拖动到的位置。

Java 关键字

下面是 Java 语言保留使用的 50 个关键字：

<code>abstract</code>	<code>double</code>	<code>int</code>	<code>super</code>
<code>assert</code>	<code>else</code>	<code>interface</code>	<code>switch</code>
<code>boolean</code>	<code>enum</code>	<code>long</code>	<code>synchronized</code>
<code>break</code>	<code>extends</code>	<code>native</code>	<code>this</code>
<code>byte</code>	<code>final</code>	<code>new</code>	<code>throw</code>
<code>case</code>	<code>finally</code>	<code>package</code>	<code>throws</code>
<code>catch</code>	<code>float</code>	<code>private</code>	<code>transient</code>
<code>char</code>	<code>for</code>	<code>protected</code>	<code>try</code>
<code>class</code>	<code>goto</code>	<code>public</code>	<code>void</code>
<code>const</code>	<code>if</code>	<code>return</code>	<code>volatile</code>
<code>continue</code>	<code>implements</code>	<code>short</code>	<code>while</code>
<code>default</code>	<code>import</code>	<code>static</code>	
<code>do</code>	<code>instanceof</code>	<code>strictfp</code> [⊖]	

关键字 `goto` 和 `const` 是 C++ 保留的关键字，目前并没有在 Java 中使用到。如果它们出现在 Java 程序中，Java 编译器能够识别它们，并产生错误信息。

字面常量 `true`、`false` 和 `null` 如同字面值 `100` 一样，不是关键字。但是它们也不能用作标识符，就像 `100` 不能用作标识符一样。

在代码清单中，我们对 `true`、`false` 和 `null` 使用了关键字的颜色，以和 Java IDE 中它们的颜色保持一致。

-
- ⊖ `strictfp` 关键字是用于修饰方法或者类的，使其使用严格的浮点计算。浮点计算可以使用以下两种模式：严格的和非严格的。严格模式可以保证计算结果在所有的虚拟机实现中都是一样的。非严格模式允许计算的中间结果以一种扩展的格式存储，该格式不同于标准的 IEEE 浮点数格式。扩展格式是依赖于机器的，可以使代码执行更快。然而，当在不同的虚拟机上使用非严格模式执行代码时，可能不会总能精确地得到同样结果。默认情况下，非严格模式被用于浮点数的计算。若在方法和类中使用严格模式，需要在方法或者类的声明前面增加 `strictfp` 关键字。严格的浮点数可能会比非严格浮点数具有略好的精确度，但这种区别仅影响部分应用。严格模式不会被继承，即，在类或者接口的声明中使用 `strictfp` 不会使得继承的子类或接口也是严格模式。

ASCII 字符集

表 B-1 和表 B-2 分别列出了 ASCII 字符与它们相应的十进制和十六进制编码。字符的十进制或十六进制编码是字符行下标和列下标的组合。例如，在表 B-1 中，字母 A 在第 6 行第 5 列，所以它的十进制代码为 65；在表 B-2 中，字母 A 在第 4 行第 1 列，所以它的十六进制代码为 41。

表 B-1 十进制编码的 ASCII 字符集

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	nl	vt	ff	cr	so	si	dle	dcl	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	"	#	\$	%	&	'
4	()	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[\]	^	-	'	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{		}	~	del		

表 B-2 十六进制编码的 ASCII 字符集

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht	nl	vt	ff	cr	so	si
1	dle	dcl	dc2	dc3	dc4	nak	syn	etb	can	em	sub	esc	fs	gs	rs	us
2	sp	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	-
6	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	del

操作符优先级表

操作符按照优先级递减的顺序从上到下列出。同一栏中的操作符优先级相同，它们的结合方向如表中所示。

操作符	名称	结合方向	操作符	名称	结合方向
()	圆括号	从左向右	>>>	用零扩展的右移	从左向右
()	函数调用	从左向右	<	小于	从左向右
[]	数组下标	从左向右	<=	小于等于	从左向右
.	对象成员访问	从左向右	>	大于	从左向右
++	后置增量	从右向左	>=	大于等于	从左向右
--	后置减量	从右向左	instanceof	检测对象类型	从左向右
++	前置增量	从右向左	==	相等	从左向右
--	前置减量	从右向左	!=	不等	从左向右
+	一元加	从右向左	&	(无条件与)	从左向右
-	一元减	从右向左	^	(异或)	从左向右
!	一元逻辑非	从右向左		(无条件或)	从左向右
(type)	一元类型转换	从右向左	&&	条件与	从左向右
new	创建对象	从右向左		条件或	从左向右
*	乘法	从左向右	?:	三元条件	从右向左
/	除法	从左向右	=	赋值	从右向左
%	求余	从左向右	+=	加法赋值	从右向左
+	加法	从左向右	-=	减法赋值	从右向左
-	减法	从左向右	*=	乘法赋值	从右向左
<<	左移	从左向右	/=	除法赋值	从右向左
>>	用符号位扩展的右移	从左向右	%=	求余赋值	从右向左

Java 修饰符

修饰符用于类和类的成员（构造方法、方法、数据和类一级的块），但 `final` 修饰符也可以用在方法中的局部变量上。可以用在类上的修饰符称为类修饰符（class modifier）。可以用在方法上的修饰符称为方法修饰符（method modifier）。可以用在数据域上的修饰符称为数据修饰符（data modifier）。可以用在类一级块上的修饰符称为块修饰符（block modifier）。下表给出 Java 修饰符的一个总结。

修饰符	类	构造方法	方法	数据	块	解释
(default) [⊖]	√	√	√	√	√	类、构造方法、方法或数据域在所在的包中可见
<code>public</code>	√	√	√	√		类、构造方法、方法或数据域在任何包任何程序中都可可见
<code>private</code>		√	√	√		构造方法、方法或数据域只在所在的类中可见
<code>protected</code>		√	√	√		构造方法、方法或数据域在所属包中可见，或者在任何包中该类的子类中可见
<code>static</code>			√	√	√	定义类方法、类数据域或静态初始化模块
<code>final</code>	√		√	√		终极类不能扩展。终极方法不能在子类中修改。终极数据域是常量
<code>abstract</code>	√		√			抽象类必须被扩展。抽象方法必须在具体的子类中实现
<code>native</code>			√			用 <code>native</code> 修饰的方法表明它是用 Java 以外的语言实现的
<code>synchronized</code>			√		√	同一时间只有一个线程可以执行这个方法
<code>strictfp</code>	√		√			使用精确浮点数计算模式，保证在所有的 Java 虚拟机中计算结果都相同
<code>transient</code>				√		标记实例数据域，使其不进行序列化

默认（没有修饰符）、`public`、`private` 以及 `protected` 等修饰符称为可见或者可访问性修饰符，因为它们给定了类，以及类的成员是如何被访问的。

`public`、`private`、`protected`、`static`、`final` 以及 `abstract` 也可以用于内部类。

⊖ 默认访问没有任何修饰符与之关联。例如：`class Test{}`

特殊浮点值

整数除以零是非法的，会抛出异常 `ArithmeticException`，但是浮点值除以零不会引起异常。在浮点运算中，如果运算结果对 `double` 型或 `float` 型来说数值太大，则向上溢出为无穷大；如果运算结果对 `double` 型或 `float` 型来说数值太小，则向下溢出为零。Java 用特殊的浮点值 `POSITIVE_INFINITY`、`NEGATIVE_INFINITY` 和 `NaN` (Not a Number, 非数值) 来表示这些结果。这些值被定义为 `Float` 类和 `Double` 类中的特殊常量。

如果正浮点数除以零，结果为 `POSITIVE_INFINITY`。如果负浮点数除以零，结果为 `NEGATIVE_INFINITY`。如果浮点数零除以零，结果为 `NaN`，表示这个结果在数学上是无定义的。这三个值的字符串表示为 `Infinity`、`-Infinity` 和 `NaN`。例如，

```
System.out.print(1.0 / 0); // Print Infinity
System.out.print(-1.0 / 0); // Print -Infinity
System.out.print(0.0 / 0); // Print NaN
```

这些特殊值也可以在运算中用作操作数。例如，一个数除以 `POSITIVE_INFINITY` 得到零。表 E-1 总结了运算符 `/`、`*`、`%`、`+` 和 `-` 的各种组合。

表 E-1 特殊的浮点值

x	y	x/y	x*y	x%y	x+y	x-y
Finite	± 0.0	$\pm \text{infinity}$	± 0.0	NaN	Finite	Finite
Finite	$\pm \text{infinity}$	± 0.0	± 0.0	x	$\pm \text{infinity}$	infinity
± 0.0	± 0.0	NaN	± 0.0	NaN	± 0.0	± 0.0
$\pm \text{infinity}$	Finite	$\pm \text{infinity}$	± 0.0	NaN	$\pm \text{infinity}$	$\pm \text{infinity}$
$\pm \text{infinity}$	$\pm \text{infinity}$	NaN	± 0.0	NaN	$\pm \text{infinity}$	infinity
± 0.0	$\pm \text{infinity}$	± 0.0	NaN	± 0.0	$\pm \text{infinity}$	± 0.0
NaN	Any	NaN	NaN	NaN	NaN	NaN
Any	NaN	NaN	NaN	NaN	NaN	NaN

 注意：如果一个操作数是 NaN，则结果一定是 NaN。

数 系

F.1 引言

因为计算机本身只能存储和处理 0 和 1，所以其内部使用的是二进制数。二进制数系只有两个数：0 和 1。在计算机中，数字或字符是以由 0 和 1 组成的序列来存储的。每个 0 或 1 都称为一个比特（二进制数字）。

我们在日常生活中使用十进制数。当我们在程序中编写一个数字，如 20，它被假定为一个十进制数。在计算机内部，通常会用软件将十进制数转换成二进制数，反之亦然。

我们使用十进制数编写程序。然而，如果要与操作系统打交道，需要使用二进制数以达到“机器级”。二进制数冗长烦琐，所以经常使用十六进制数简化二进制数，每个十六进制数可以确切表示四个二进制数。十六进制数系有十六个数：0 ~ 9、A ~ F，其中字母 A、B、C、D、E 和 F 对应十进制数 10、11、12、13、14 和 15。

十进制数系中的数是 0、1、2、3、4、5、6、7、8 和 9。一个十进制数是用一个或多个这些数所构成的一个序列来表示的。这个序列中每个数所表示的值和它的位置有关，序列中数的位置决定了 10 的幂次。例如，十进制数 7423 中的数 7、4、2 和 3 分别表示 7000、400、20 和 3，如下所示：

$$\boxed{7} \boxed{4} \boxed{2} \boxed{3} = 7 \times 10^3 + 4 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$$

$$10^3 10^2 10^1 10^0 = 7000 + 400 + 20 + 3 = 7423$$

十进制数系有十个数，它们的位置值都是 10 的整数次幂。10 是十进制数系的基数。类似地，由于二进制数系有两个数，所以它的基数为 2；而十六进制数系有 16 个数，所以它的基数为 16。

如果 1101 是一个二进制数，那么数 1、1、0 和 1 分别表示：

$$\boxed{1} \boxed{1} \boxed{0} \boxed{1} = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$2^3 2^2 2^1 2^0 = 8 + 4 + 0 + 1 = 13$$

如果 7423 是一个十六进制数，那么数字 7、4、2 和 3 分别表示：

$$\boxed{7} \boxed{4} \boxed{2} \boxed{3} = 7 \times 16^3 + 4 \times 16^2 + 2 \times 16^1 + 3 \times 16^0$$

$$16^3 16^2 16^1 16^0 = 28672 + 1024 + 32 + 3 = 29731$$

F.2 二进制数与十进制数之间的转换

给定二进制数 $b_n b_{n-1} b_{n-2} \cdots b_2 b_1 b_0$ ，等价的十进制数为

$$b_n \times 2^n + b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + \cdots + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$$

下面是二进制数转换为十进制数的例子：

二进制	转换公式	十进制
10	$1 \times 2^1 + 0 \times 2^0$	2
1000	$1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$	8
10101011	$1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$	171

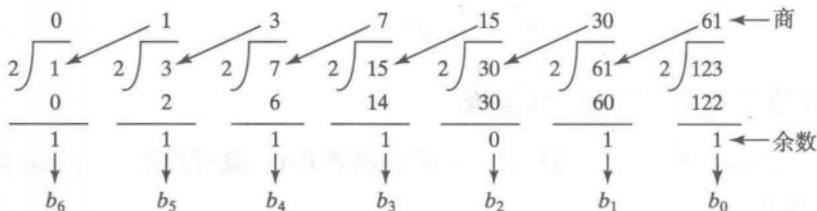
把一个十进制数 d 转换为二进制数，就是求满足

$$d = b_n \times 2^n + b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + \dots + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$$

的位 $b_n, b_{n-1}, b_{n-2}, \dots, b_2, b_1$ 和 b_0 。

用 2 不断地除 d ，直到商为 0 为止，余数即为所求的位 $b_0, b_1, \dots, b_{n-2}, b_{n-1}, b_n$ 。

例如，十进制数 123 用二进制数 1111011 表示，所做的转换如下：



提示：Windows 操作系统所带的计算器是进行数制转换的一个有效工具，如图 F-1 所示。要运行它，从 Start 按钮搜索 Calculator 并运行 Calculator，然后在 View 菜单下面选择 Scientific。



图 F-1 使用 Windows 的计算器进行数制转换

F.3 十六进制数与十进制数的转换

给定十六进制数 $h_n h_{n-1} h_{n-2} \dots h_2 h_1 h_0$ ，其等价的十进制数为

$$h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + \dots + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0$$

下面是十六进制数转换为十进制数的例子：

十六进制	转换公式	十进制
7F	$7 \times 16^1 + 15 \times 16^0$	127
FFFF	$15 \times 16^3 + 15 \times 16^2 + 15 \times 16^1 + 15 \times 16^0$	65535
431	$4 \times 16^2 + 3 \times 16^1 + 1 \times 16^0$	1073

将一个十进制数 d 转换为十六进制数，就是求满足

$$d = h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + \dots + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0$$

的位 $h_n, h_{n-1}, h_{n-2}, \dots, h_2, h_1$ 和 h_0 。用 16 不断地除 d ，直到商为 0 为止。余数即为所求的位 $h_0, h_1, \dots, h_{n-2}, h_{n-1}, h_n$ 。

例如，十进制数 123 用十六进制表示为 7B，所做的转换如下：

$$\begin{array}{r}
 \begin{array}{r} 0 \\ 16 \overline{) 7} \\ \underline{0} \\ 7 \end{array} \quad \begin{array}{r} 7 \\ 16 \overline{) 123} \\ \underline{112} \\ 11 \end{array} \\
 \hline
 \begin{array}{r} 7 \\ \downarrow \\ h_1 \end{array} \quad \begin{array}{r} 11 \\ \downarrow \\ h_0 \end{array}
 \end{array}$$

← 商

← 余数

F.4 二进制数与十六进制数的转换

将一个十六进制数转换为二进制数，只需利用表 F-1，就可以把十六进制数的每一位转换为四位二进制数。

例如，十六进制数 7B 转换为二进制是 1111011，其中 7 的二进制表示为 111，B 的二进制表示为 1011。

要将一个二进制数转换为十六进制数，从右向左将每四位二进制数转换为位十六进制数。

例如，二进制数 1110001101 的十六进制表示是 38D，因为 1101 是 D，1000 是 8，11 是 3，如下所示：

$$\begin{array}{cccccccc}
 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
 & \downarrow \\
 & 3 & & 8 & & & & & & & D
 \end{array}$$

表 F-1 十六进制数转换为二进制数

十六进制	二进制	十进制	十六进制	二进制	十进制
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	A	1010	10
3	0011	3	B	1011	11
4	0100	4	C	1100	12
5	0101	5	D	1101	13
6	0110	6	E	1110	14
7	0111	7	F	1111	15

注意：八进制数也很有用。八进制数系有 0 到 7 共八个数。十进制数 8 在八进制数系中的作用就和十进制数系中的 10 一样。

这里有一些好的在线资源，用于练习数值转换：

- http://forums.cisco.com/CertCom/game/binary_game_page.htm
- <http://people.sinclair.edu/nickreeder/Flash/binDec.htm>
- <http://people.sinclair.edu/nickreeder/Flash/binHex.htm>

 复习题

F.1 将下列十进制数转换为十六进制数和二进制数。

100; 4340; 2000

F.2 将下列二进制数转换为十六进制数和十进制数。

1000011001; 100000000; 100111

F.3 将下列十六进制数转换为二进制数和十进制数。

FEFA9; 93; 2000

位 操 作

用机器语言编写程序，经常要直接处理二进制数值，并在位级别上执行操作。Java 提供了位操作符和移位操作符，如表 G-1 所示。

表 G-1

操作符	名称	示例 (例中使用字节)	描述
&	位与	10101110 & 10010010 得到 10000010	两个相应位上的比特如果都为 1，则执行与操作会得到 1
	位或	10101110 10010010 得到 10111110	两个相应位上的比特如果其中有一个为 1，则执行或操作会得到 1
^	位与或	10101110 ^ 10010010 得到 00111100	两个相应位上的比特如果相异，则执行与或操作会得到 1
~	求反	~10101110 得到 01010001	操作符将每个比特从 0 到 1 或者从 1 到 0 进行转换
<<	左移位	10101110 << 2 得到 10111000	操作符将其左边的操作数按照第二个操作数指定的位移数进行左移位，右边空出来的补 0
>>	带符号位右移位	10101110 >> 2 得到 11101011 00101110 >> 2 得到 00001011	操作符将其第一个操作数按照第二个操作数指定的位移数进行右移位，最高位补上符号位
>>>	无符号位右移位	10101110 >>> 2 得到 00101011 00101110 >>> 2 得到 00001011	操作符将其第一个操作数按照第二个操作数指定的位移数进行右移位，左边空出来的补 0

位操作符仅适用于整数类型 (byte、short、int 和 long)。位操作涉及的字符将转换为整数。所有的位操作符可以构成位赋值操作符，例如 =, |=, <<=, >>=, 以及 >>>=。

正则表达式

经常会需要编写代码来验证用户输入，比如验证输入是否是一个数字，是否是一个全部小写的字符串，或者社会安全号。如何编写这种类型的代码呢？一个简单而有效的做法是使用正则表达式来完成这个任务。

正则表达式 (regular expression, 简称为 regex) 是一个字符串，用来描述匹配一个字符串集合的模式。对于字符串处理来说，正则表达式是一个强大的工具。可以使用正则表达式来匹配、替换和分割字符串。

H.1 匹配字符串

让我们从 String 类中的 matches 方法开始。乍一看，matches 方法很类似 equals 方法。例如，以下两个语句结果都为 true。

```
"Java".matches("Java");
"Java".equals("Java");
```

然而，matches 方法更强大。它不仅可以匹配固定字符串，还可以匹配一个模式的字符串集。例如，以下语句结果都为 true。

```
"Java is fun".matches("Java.*")
"Java is cool".matches("Java.*")
"Java is powerful".matches("Java.*")
```

前面语句中的 "Java.*" 是一个正则表达式。它描述了一个字符串模式，以 Java 开始，后面跟 0 个或者多个字符串。这里，子字符串 .* 匹配任何 0 个或者多个字符。

H.2 正则表达式语法

正则表达式由字面值字符和特殊符号组成。表 H-1 列出了正则表达式常用的语法。

🔧 注意：反斜杠是一个特殊的字符，在字符串中开始转义序列。因此 Java 中需要使用 \\d 来表示 \d。

🔧 注意：回顾下，空白字符是 ' ', '\t', '\n', '\r', 或者 '\f'。因此，\s 和 [\t\n\r\f] 等同，\S 和 [^\t\n\r\f] 等同。

表 H-1 常用的正则表达式

正则表达式	匹配	示例
x	指定字符 x	Java 匹配 Java
.	任意单个字符	Java 匹配 J..a
(ab cd)	ab 或者 cd	ten 匹配 t(en im)
[abc]	a、b 或者 c	Java 匹配 Ja[uvw]a
[^abc]	除开 a、b 或者 c 外的任意字符	Java 匹配 Ja[^ars]a
[a-z]	a 到 z	Java 匹配 [A-M]av[a-d]
[^a-z]	除开 a 到 z 的任意字符	Java 匹配 Jav[^b-d]

(续)

正则表达式	匹配	示例
[a-e[m-p]]	a 到 e 或 m 到 p	Java 匹配 [A-G[I-M]]av[a-d]
[a-e&&[c-p]]	a 到 e 与 c 到 p 的交集	Java 匹配 [A-P&&[I-M]]av[a-d]
\d	个位数, 等同于 [0-9]	Java2 匹配 "Java[\d]"
\D	一位非数字	\$Java 匹配 "[\D][\D]ava"
\w	单词字符	Java1 匹配 "[\w]ava[\w]"
\W	非单词字符	\$Java 匹配 "[\W][\w]ava"
\s	空白字符	"Java 2" 匹配 "Java\s2"
\S	非空白字符	Java 匹配 "[\S]ava"
p*	模式 p 的 0 或者多次出现	aaaabb 匹配 "a*bb" ababab 匹配 "(ab)*"
p+	模式 p 的 1 或者多次出现	a 匹配 "a+b*" able 匹配 "(ab)+.*"
p?	模式 p 的 0 或者 1 次出现	Java 匹配 "J?Java" Java 匹配 "J?ava"
p{n}	模式 p 的正好 n 次出现	Java 匹配 "Ja{1}.*" Java 不匹配 ".{2}"
p{n,}	模式 p 的至少 n 次出现	aaaa 匹配 "a{1,}" a 不匹配 "a{2,}"
p{n,m}	模式 p 出现次数位于 n 和 m 间 (不包含)	aaaa 匹配 "a{1,9}" abb 不匹配 "a{2,9}bb"

注意: 单词字符是任何的字母, 数字或者下划线字符。因此 \w 等同于 [a-zA-Z][0-9_] 或者简化为 [a-Za-z0-9_]。 \W 等同于 [^a-zA-z0-9_]。

注意: 表 H-1 中最后六个实体 *、+、?、{n}、{n,}、以及 {n,m} 称为量词符, 用于确定量词符前面的模式会重复多少次。例如, A* 匹配 0 或者多个 A, A+ 匹配 1 或者多个 A, A? 匹配 0 或者 1 个 A, A{3} 精确匹配 AAA, A{3,} 匹配至少 3 个 A, A{3,6} 匹配 3 到 6 之间个 A。 * 等同于 {0,}, + 等同于 {1,}, ? 等同于 {0,1}。

警告: 不要在重复量词符中使用空白。例如, A{3,6} 不能写成逗号后面有一个空白符的 A{3, 6}。

注意: 可以使用括号来将模式进行分组。例如, (ab){3} 匹配 ababab, 但是 ab{3} 匹配 abbb。

让我们用一些示例来演示如何构建正则表达式。

1. 示例 1

社会安全号的模式是 xxx-xx-xxx, 其中 x 是一位数字。社会安全号的正则表达式可以描述为

```
[\\d]{3}-[\\d]{2}-[\\d]{4}
```

例如

```
"111-22-3333".匹配("[\\d]{3}-[\\d]{2}-[\\d]{4}")返回 true.  
"11-22-3333".匹配("[\\d]{3}-[\\d]{2}-[\\d]{4}")返回 false.
```

2. 示例 2

偶数以数字 0、2、4、6 或者 8 结尾。偶数的模式可以描述为

```
[\d]*[02468]
```

例如,

```
"123".matches("[\d]*[02468]") 返回 false.
"122".matches("[\d]*[02468]") 返回 true.
```

3. 示例 3

电话号码的模式是 (xxx)xxx-xxxx, 这里 x 是一位数字, 并且第一位数字不能为 0。电话号码的正则表达式可以描述为

```
\\([1-9][\d]{2}\\\\) [\d]{3}-[\d]{4}
```

注意: 括号 (和) 在正则表达式中是特殊字符, 用于对模式分组。为了在正则表达式中表示字面值 (或者), 必须使用 \\(和 \\)。

例如

```
"(912) 921-2728".matches("\\([1-9][\d]{2}\\\\) [\d]{3}-[\d]{4}")
返回 true.
"921-2728".matches("\\([1-9][\d]{2}\\\\) [\d]{3}-[\d]{4}")
返回 false.
```

4. 示例 4

假定姓由最多 25 个字母组成, 并且第一个字母为大写形式。则姓的模式可以描述为

```
[A-Z][a-zA-Z]{1,24}
```

注意: 不能任意放空白符到正则表达式中。如 [A-Z][a-zA-Z]{1,24} 将报错。例如:

```
"Smith".matches("[A-Z][a-zA-Z]{1,24}") 返回 true.
"Jones123".matches("[A-Z][a-zA-Z]{1,24}") 返回 false.
```

5. 示例 5

Java 标识符在第 2.4 节中定义

- 标识符必须以字母、下划线 (_), 或者美元符号 (\$) 开始。不能以数字开头。
- 标识符是一个由字母、数字、下划线 (_) 和美元符号组成的字符序列。

标识符的模式可以描述为

```
[a-zA-Z_$][\w$]*
```

6. 示例 6

什么字符串匹配正则表达式 "Welcome to (Java|HTML)"? 答案是 Welcome to Java 或者 Welcome to HTML。

7. 示例 7

什么字符串匹配正则表达式 "A.*"? 答案是任何以字母 A 开头的字符串。

H.3 替换和分割字符串

如果字符串匹配正则表达式, String 类的 matches 方法返回 true。String 类也包含 replaceAll、replaceFirst 和 split 方法, 用于替换和分割字符串, 如图 H-1 所示。

replaceAll 方法替换所有匹配的子字符串, replaceFirst 方法替换第一个匹配的子字符串。例如, 下面代码

```
System.out.println("Java Java Java".replaceAll("v\\w", "wi"));
```

java.lang.String	
<pre>+matches(regex: String): boolean +replaceAll(regex: String, replacement: String): String +replaceFirst(regex: String, replacement: String): String +split(regex: String): String[] +split(regex: String, limit: int): String[]</pre>	<p>如果字符串匹配模式，则返回 true 将匹配的子字符串替换为 replacement 变量中的字符串，并返回新的字符串</p> <p>将匹配的的第一个子字符串替换为 replacement 变量中的字符串，并返回新的字符串</p> <p>返回一个字符串数组，包含被匹配模式的分割符分割的子字符串</p> <p>与前面的分割方法等同，除开 limit 参数控制了模式应用的次数</p>

图 H-1 String 类包含使用正则表达式来匹配、替换和分割字符串的方法

显示

```
Jawi Jawi Jawi
```

下面代码

```
System.out.println("Java Java Java".replaceFirst("\\w", "wi"));
```

显示

```
Jawi Java Java
```

有两个重载的 split 方法。split(regex) 方法使用匹配的分割符将一个字符串分割为子字符串。例如，以下语句

```
String[] tokens = "JavaHTML2Perl".split("\\d");
```

将字符串 "JavaHTML2Perl" 分割为 Java、HTML 以及 Perl 并且保存在 tokens[0]，tokens[1] 以及 tokens[2] 中。

在 split(regex, limit) 方法中，limit 参数确定模式匹配多少次。如果 limit <= 0，split(regex, limit) 等同于 split(regex)。如果 limit > 0，模式最多匹配 limit - 1 次。

下面是一些示例：

```
"JavaHTML2Perl".split("\\d", 0); 分割为 Java, HTML, Perl
"JavaHTML2Perl".split("\\d", 1); 分割为 JavaHTML2Perl
"JavaHTML2Perl".split("\\d", 2); 分割为 Java, HTML2Perl
"JavaHTML2Perl".split("\\d", 3); 分割为 Java, HTML, Perl
"JavaHTML2Perl".split("\\d", 4); 分割为 Java, HTML, Perl
"JavaHTML2Perl".split("\\d", 5); 分割为 Java, HTML, Perl
```

注意：默认的，所有的量词符都是“贪婪”的。这意味着它们会尽量匹配可能的最多次。比如，下面语句显示 JRvaa。因为第一个匹配成功的是 aaa。

```
System.out.println("Jaaavaa".replaceFirst("a+", "R"));
```

可以通过在后面添加问号符号来改变量词符的默认行为。量词符变为“不情愿”的，这意味着它将匹配尽可能少的次数。例如，下面的语句显示 JRaavaa，因为第一个匹配成功的是 a。

```
System.out.println("Jaaavaa".replaceFirst("a+?", "R"));
```

枚举类型

1.1 简单枚举类型

枚举类型定义了一个枚举值的列表。每个值是一个标识符。例如，下面的语句声明了一个枚举类型，命名为 `MyFavoriteColor`，具有 `RED`、`BLUE`、`GREEN`、`YELLOW` 值。

```
enum MyFavoriteColor {RED, BLUE, GREEN, YELLOW};
```

枚举类型的值类似于一个常量，因此，按惯例拼写都是使用大写字母。因此，前面的声明采用 `RED`，而不是 `red`。按惯例，枚举类型命名类似于一个类，每个单词的第一个字母大写。

一旦定义了类型，就可以声明这个类型的变量了：

```
MyFavoriteColor color;
```

变量 `color` 可以具有定义在枚举类型 `MyFavoriteColor` 中的一个值，或者 `null`，但是不能具有其他值。Java 的枚举类型是类型安全的，这意味着试图赋一个枚举类型所列出的值或者 `null` 之外的一个值，都将导致编译错误。

枚举值可以使用下面的语法进行访问：

```
EnumeratedTypeName.valueName
```

例如，下面的语句将枚举值 `BLUE` 赋值给变量 `color`：

```
color = MyFavoriteColor.BLUE;
```

注意：必须使用枚举类型名称作为限定词来引用一个值，比如 `BLUE`。

如同其他类型一样，可以在一行语句中来声明和初始化一个变量：

```
MyFavoriteColor color = MyFavoriteColor.BLUE;
```

枚举类型被作为一个特殊的类来对待。因此，枚举类型的变量是引用变量。一个枚举类型是 `Object` 类和 `Comparable` 接口的子类。因此，枚举类型继承了 `Object` 类中的所有方法，以及 `Comparable` 接口中的 `compareTo` 方法。另外，可以在一个枚举类型的对象上面使用下面的方法：

- `public String name();`

为对象返回名字值。

- `public int ordinal();`

返回和枚举值关联的序号值。枚举类型中的第一个值具有序号数 0，第二个值具有序号值 1，第三个为 2，依次类推。

程序清单 I-1 给出了一个程序，展示了枚举类型的使用。

程序清单 I-1 EnumeratedTypeDemo.java

```

1 public class EnumeratedTypeDemo {
2     static enum Day {SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
3         FRIDAY, SATURDAY};
4
5     public static void main(String[] args) {
6         Day day1 = Day.FRIDAY;
7         Day day2 = Day.THURSDAY;
8
9         System.out.println("day1's name is " + day1.name());
10        System.out.println("day2's name is " + day2.name());
11        System.out.println("day1's ordinal is " + day1.ordinal());
12        System.out.println("day2's ordinal is " + day2.ordinal());
13
14        System.out.println("day1.equals(day2) returns " +
15            day1.equals(day2));
16        System.out.println("day1.toString() returns " +
17            day1.toString());
18        System.out.println("day1.compareTo(day2) returns " +
19            day1.compareTo(day2));
20    }
21 }

```

```

day1's name is FRIDAY
day2's name is THURSDAY
day1's ordinal is 5
day2's ordinal is 4
day1.equals(day2) returns false
day1.toString() returns FRIDAY
day1.compareTo(day2) returns 1

```

在第 2 ~ 3 行定义了枚举类型 Day。变量 day1 和 day2 声明为 Day 类型，在第 6 ~ 7 行赋枚举值。由于 day1 的值为 FRIDAY，它的序号值为 5（第 11 行）。由于 day2 的值为 THURSDAY，它的序号值为 4（第 12 行）。

由于一个枚举类型是 Object 类和 Comparable 接口的子类。可以从一个枚举对象引用变量调用 equals，toString 以及 compareTo 方法（第 14 ~ 19 行）。如果 day1 和 day2 具有同样的序号数，day1.equals(day2) 返回真。day1.compareTo(day2) 返回 day1 的序号数到 day2 的序号数之间的差距。

作为另外一种选择，可以将程序清单 I-1 中的代码重新写为程序清单 I-2。

程序清单 I-2 StandaloneEnumTypeDemo.java

```

1 public class StandaloneEnumTypeDemo {
2     public static void main(String[] args) {
3         Day day1 = Day.FRIDAY;
4         Day day2 = Day.THURSDAY;
5
6         System.out.println("day1's name is " + day1.name());
7         System.out.println("day2's name is " + day2.name());
8         System.out.println("day1's ordinal is " + day1.ordinal());
9         System.out.println("day2's ordinal is " + day2.ordinal());
10
11        System.out.println("day1.equals(day2) returns " +
12            day1.equals(day2));
13        System.out.println("day1.toString() returns " +
14            day1.toString());
15        System.out.println("day1.compareTo(day2) returns " +
16            day1.compareTo(day2));
17    }

```

```

18 }
19
20 enum Day {SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
21          FRIDAY, SATURDAY}

```

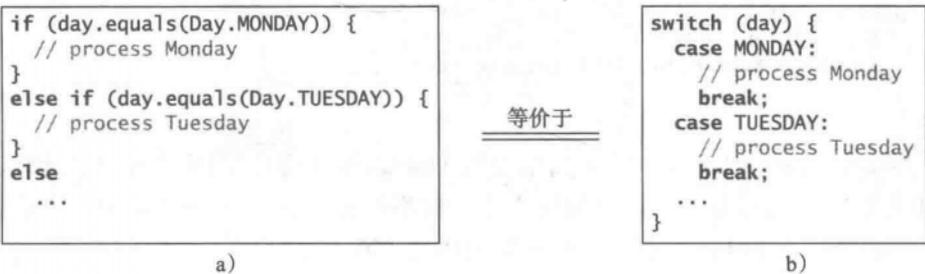
枚举类型可以在一个类中定义，如程序清单 I-1 中的第 2 ~ 3 行所示；或者单独定义，如程序清单 I-2 的第 20 ~ 21 行所示。在前一种情况下，枚举类型被作为内部类对待。程序编译后，将创建一个名为 `EnumeratedTypeDemo$Day` 的类。在后一种情况下，枚举类型作为一个独立的类来对待。程序编译后，将创建一个名为 `Day.class` 的类。

注意：当一个枚举类型在一个类中声明时，类型必须声明为类的一个成员，而不能在一个方法中声明。而且，类型总是 `static` 的。由于这个原因，程序清单 I-1 第 2 行的 `static` 关键字可以省略。可以用于内部类的可见性修饰符也可以应用到在一个类中定义的枚举类型中。

技巧：使用枚举值（例如，`Day.MONDAY`，`Day.TUESDAY`，等等）而不是字面量整数值（例如，0，1，等等）可以让程序更加易于阅读和维护。

1.2 通过枚举变量使用 if 或者 switch 语句

枚举变量具有一个值。程序经常需要根据取值来执行特定的动作。例如，如果值为 `Day.MONDAY`，则踢足球；如果值为 `Day.TUESDAY`，则学习钢琴课，等等。可以使用 `if` 语句或者 `switch` 语句来测试变量的值，如图 a) 和 b) 所示。



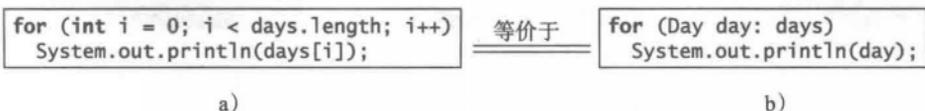
在 b 图的 `switch` 语句中，`case` 标签是一个无限定词的枚举值（即，`MONDAY`，而不是 `Day.MONDAY`）。

1.3 使用 foreach 循环处理枚举值

每个枚举类型有一个静态方法 `value()`，可以返回这个类型中所有的枚举值到一个数组中。例如，

```
Day[] days = Day.values();
```

可以使用通常的循环如图 a 中所示，或者图 b 中的 `foreach` 循环来处理数组中的所有值。



1.4 具有数据域，构造方法和方法的枚举类型

前面介绍的简单枚举类型定义了一个类型，具有一个枚举值的列表。也可以定义一个具

有数据域，构造方法和方法的枚举类型，如程序清单 I-3 所示。

程序清单 I-3 TrafficLight.java

```
1 public enum TrafficLight {
2     RED ("Please stop"), GREEN ("Please go"),
3     YELLOW ("Please caution");
4
5     private String description;
6
7     private TrafficLight(String description) {
8         this.description = description;
9     }
10
11     public String getDescription() {
12         return description;
13     }
14 }
```

第 2~3 行定义了枚举值。值的声明必须是类型声明的第一条语句。一个名为 `description` 的数据域在第 5 行声明，描述了一个枚举值。构造方法 `TrafficLight` 在第 7~9 行声明。当访问枚举值的时候，构造方法将被调用。枚举值的参数将传递给构造方法，在构造方法中赋值给 `description`。

程序清单 I-4 给出了一个使用 `TrafficLight` 的测试程序。

程序清单 I-4 TestTrafficLight.java

```
1 public class TestTrafficLight {
2     public static void main(String[] args) {
3         TrafficLight light = TrafficLight.RED;
4         System.out.println(light.getDescription());
5     }
6 }
```

一个枚举值 `TrafficLight.RED` 赋值给变量 `light` (第 3 行)。访问 `TrafficLight.RED` 引起 JVM 使用参数 “please stop” 调用构造方法。枚举类型中的方法是和类中的方法调用一样的。`light.getDescription()` 返回对枚举值的描述 (第 4 行)。

 **注意：**Java 语法要求枚举类型的构造方法是私有的，避免被直接调用。私有修饰符可以省略。在这种情况下，被默认为是私有的。

推荐阅读



中文版
第6版

作者: Abraham Silberschatz 著
中文翻译版: 978-7-111-37529-6, 99.00元
本科教学版: 978-7-111-40085-1, 59.00元



中文版
第3版

作者: Jiawei Han 等著
中文版: 978-7-111-39140-1, 79.00元



中文版
第3版

作者: Ian H.Witten 等著
中文版: 978-7-111-45381-9, 79.00元



中文版
第2版

作者: Randal E. Bryant 等著
书号: 978-7-111-32133-0, 99.00元



中文版
第4版

作者: David A. Patterson John L. Hennessy
中文版: 978-7-111-35305-8, 99.00元



中文版
第6版

作者: James F. Kurose 著
书号: 978-7-111-45378-9, 79.00元



中文版
第3版

作者: Thomas H. Cormen 等著
书号: 978-7-111-40701-0, 128.00元



中文版
第2版

作者: Brian W. Kernighan 等著
书号: 978-7-111-12806-0, 30.00元



中文版
第7版

作者: Roger S. Pressman 著
书号: 978-7-111-33581-8, 79.00元

推荐阅读



Java编程思想 (第4版)

作者: Bruce Eckel 译者: 陈昊鹏 ISBN: 7-111-21382-6 定价: 108.00元



Java程序设计教程 (原书第3版)

作者: Stuart Reges 等 译者: 陈志 等 ISBN: 978-7-111-48990-0 定价: 119.00元



Java语言导学 (原书第5版)

作者: Sharon Biocca Zakhour 等 译者: 董笑菊 等 ISBN: 978-7-111-50392-7 定价: 79.00元



Java核心技术 卷I 基础知识 (原书第9版)

作者: Cay S. Horstmann 等 译者: 周立新 等 ISBN: 978-7-111-44514-2 定价: 119.00元



Java核心技术 卷II 高级特性 (原书第9版)

作者: Cay S. Horstmann 等 译者: 陈昊鹏 等 ISBN: 978-7-111-44250-9 定价: 139.00元