

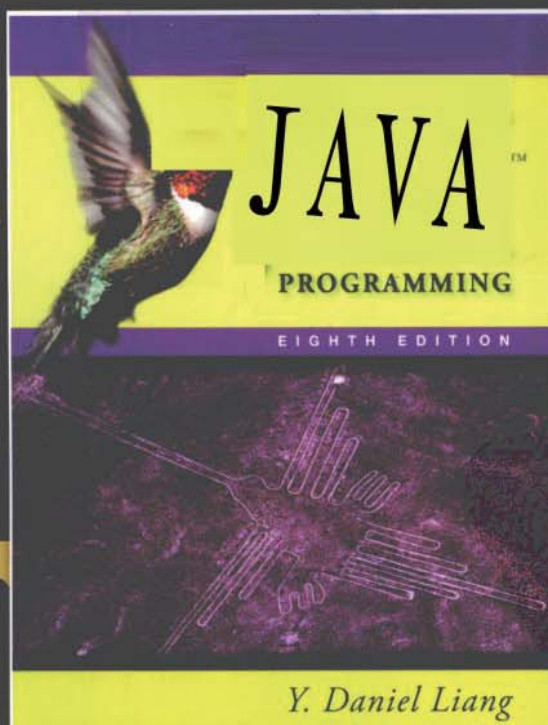
原书第8版

PEARSON

Java语言程序设计 基础篇

(美) Y. Daniel Liang 著 李娜 译
阿姆斯特朗亚特兰大州立大学 西安电子科技大学

Introduction to Java Programming
Eighth Edition



机械工业出版社
China Machine Press

Java语言程序设计 基础篇 (原书第8版)

Introduction to Java Programming Eighth 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 6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1~20章,进阶篇对应原书的第21~37章。

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 6的特性,对全书的内容进行了修订和更新,以反映Java程序设计方面的最新技术发展。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实例。
- 提供的大量实例中都包括了对问题求解的详细步骤,很多实例都是随着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。
- 较上一版增加了大量难易程度不同的习题,同时在作者的网站<http://www.cs.armstrong.edu/liang/intro8e/index.html>中还提供了很多自测题。
- 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38~48章的内容,以提供更多的相关信息。

作者简介

Y. Daniel Liang 普度大学终身教授,阿姆斯特朗亚特兰大州立大学计算机科学系教授。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。



Java语言程序设计 进阶篇 (原书第8版)
书号: 978-7-111-34236-6
定价: 79.00元



客服热线: (010) 88378991, 88361066
购书热线: (010) 68326294, 88379649, 68995259
投稿热线: (010) 88379604
读者信箱: hzjsj@hzbook.com

华章网站 <http://www.hzbook.com>

www.pearsonhighered.com

网上购书: www.china-pub.com

封面设计: 李锡 林

上架指导: 计算机/程序设计

ISBN 978-7-111-34081-2



9 787111 340812

定价: 75.00元

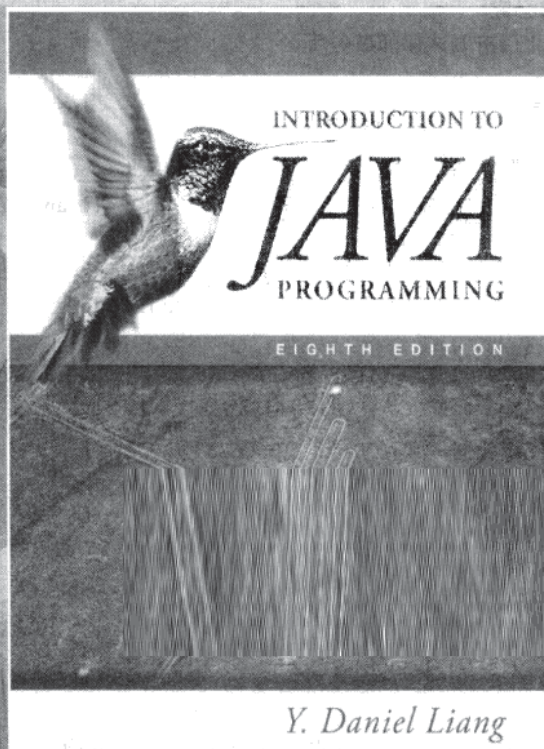
计 算 机 科 学 丛 书

原书第8版

Java语言程序设计 基础篇

(美) Y. Daniel Liang 著 李娜 译
阿姆斯特朗亚特兰大州立大学 西安电子科技大学

Introduction to Java Programming
Eighth Edition



机械工业出版社
China Machine Press

本书是Java语言的经典教材，中文版分为《Java语言程序设计 基础篇》和《Java语言程序设计 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、算法和数据结构、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和相应的提示，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，并应用所学的技术解决实际应用程序开发中遇到的问题。

基础篇从Java语言的特点入手，介绍了语法结构、面向对象程序设计基础知识到面向对象程序设计、图形用户界面设计、异常处理、applet和多媒体、二进制I/O、递归等内容。

本书可作为高等院校相关专业程序设计课程的教材，对软件开发人员也有很高的参考价值。

Simplified Chinese edition copyright © 2011 by Pearson Education Asia Limited and China Machine Press.

Original English language title: *Introduction to Java Programming, Eighth Edition* (ISBN 978-0-13-213080-6) by Y. Daniel Liang, Copyright © 2011, 2009, 2007, 2004.

All rights reserved.

Published by arrangement with the original publisher, Pearson Education, Inc., publishing as Prentice Hall.

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

封底无防伪标均为盗版

版权所有，侵权必究

本书法律顾问 北京市展达律师事务所

本书版权登记号：图字：01-2011-1518

图书在版编目（CIP）数据

Java语言程序设计 基础篇（原书第8版）／（美）梁勇（Liang, Y. D.）著；李娜译．
—北京：机械工业出版社，2011.5

（计算机科学丛书）

书名原文：Introduction to Java Programming, Eighth Edition

ISBN 978-7-111-34081-2

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

中国版本图书馆CIP数据核字（2011）第060683号

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 荣

三河市杨庄长鸣印刷装订厂印刷

2011年6月第1版第1次印刷

185mm×260mm · 37.75印张

标准书号：ISBN 978-7-111-34081-2

定价：75.00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

购书热线：(010) 68326294；88379649；68995259

投稿热线：(010) 88379604

读者信箱：hzjsj@hzbook.com

数字资源
PDG

文艺复兴以降，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；也正是这样的传统，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，不仅擘划了研究的范畴，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, Brian W. Kernighan, Dennis Ritchie, Jim Gray, Alfred 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襄助，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，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近两百个品种，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



华章教育

华章科技图书出版中心

译者序

Introduction to Java Programming, 8E

很荣幸成为这本书第8版的译者。在辛苦工作了数月之后，整本书终于翻译完毕。就在翻译这个版本的过程中，我还在使用本书的上一版本进行Java程序设计课程的教学，所以在译完新版之后，想谈谈自己的一些感想。

其实，市场上有很多关于Java的书籍，为什么我会选择这本书作为教学所用的教材呢？看了一些关于Java的教材，有些书假定读者已经有了程序设计的基础，如果初学者想从Java开始学习程序设计会很吃力；有些书虽然从程序设计基础讲起，但它的着眼点放在代码上，就事论事，并未将程序设计的思想引入其中，这样，学生学习之后，只能解决课本上所出现的问题，不能从书中学到内容延伸到所遇到的新问题上。

本书采用基础优先的方法，并且以问题驱动的方式教授程序设计的概念和技术。这样就在很大程度上克服了以上缺点，不仅涵盖的内容全面，而且自始至终都渗透着程序设计的思想，引导读者从宏观上把握程序设计。另外，本版对前一版又进行了精简与修正，略去了一些过时的知识，调整了内容的先后次序，并补充了许多新的内容，使新版对最新技术的介绍更为及时。整本书脉络清晰、可读性很强，便于查阅，既适合作为初学者的入门读物，也适合教师和专业人员参考。

这里也想提醒读者，如果可以的话，最好能按照书上所指示的，到本书配套网站上下载可用的相关资料。我自己这两年的教学过程中借鉴的就是作者在网站上的课件，再根据自己课程的需求做了一些调整，教学效果很好，所以推荐同行也去申请相关资料。对学生来讲，网站上的测试题是对课本内容的一个检验，希望你们能充分利用这些好的资源。祝福大家都能从这本书中受益！

在整个翻译工作结束之时，衷心感谢机械工业出版社华章公司的编辑所做的大量细致工作，特别是负责和我联系的王春华编辑。翻译过程中，教学工作的繁忙和不断的生病耽误了进度，给王编辑的工作带来了困扰，她心里再急也总是体谅我，对我给予了充分的信任和支持，很庆幸在翻译这本书的时候能遇到这么耐心细致的编辑。也想借此机会感谢家人、朋友和同事在翻译过程中对我的支持，没有你们的支持，我可能没有时间和精力来完成这本书的翻译工作。

由于时间仓促，译者水平有限，译文中难免存在欠妥和纰漏之处，恳请广大读者不吝赐教和指正。

译者

2011年2月

PDG

本书采用基础优先的方法，并且以问题驱动的方式教授程序设计的概念和技术。

基础优先的方法是指在学习对象和类之前，首先介绍基本程序设计的概念和技术。经验证明，学习基本逻辑以及循环和逐步求精这样的基本程序设计技术，对于初学编程的人员是非常重要的。像循环、方法和数组这样的基本概念和技术都是程序设计的基础，它们为学生进一步学习面向对象程序设计、GUI、数据库和Web程序设计做好准备。

问题驱动意味着将重点放在问题的解决而不是语法上。我们通过使用一些有趣的问题使得程序设计的介绍也变得更加有趣。前些章的主线放在问题的解决上，介绍正确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兴趣。为了吸引各个专业的学生来学习，这些问题涉及了很多应用领域，例如，数学、科学、商业、金融、游戏、动画以及多媒体。

两个版本

本书有两个版本：完全版（包括第1~37章）和基础版（包括第1~20章）。完全版^①包括程序设计基础、面向对象程序设计、GUI程序设计、算法和数据结构、并发、网络、国际化、高级GUI、数据库和Web程序设计。设计这个版本是为了培养专家级Java程序员。基础版可用于程序设计的第一门课程（通常称为CS1）。

本版新增内容

与第7版相比，本版主要的改动如下：

- 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。
- 例子和练习题都是为了激发学生对程序设计的兴趣，其中五分之一的问题都是新的。
- 在前一版中，控制台输入是在第2章的末尾介绍的。新版本在第2章之初就介绍控制台输入，这样，学生就可以更早地开始编写可交互的程序。
- 许多程序都增加了手动跟踪框，这样有助于初学者读程序和跟踪程序。
- 一维数组和多维数组分别在两章中介绍，这样可以给教师提供一定的灵活性，便于教师确定是否随后介绍多维数组。
- 将九宫格问题的实例学习移到了本书配套网站（www.cs.armstrong.edu/liang/intro8e或www.pearsonhighered.com/liang）上。本书中给出的是有利于教学的简单版本的九宫格问题。
- 为Java GUI程序设计所做的API设计是一个如何应用面向对象准则的非常好的例子。使用具体的、

^① 本书中文版将完全版分成《Java语言程序设计 基础篇》和《Java语言程序设计 进阶篇》。基础篇对应原书的第1~20章，进阶篇对应原书的第21~37章。

可视化的例子，学生可以学得更好。因此，基本GUI现在放在抽象类和接口的介绍之前，当然，教师还是可以选择在GUI之前介绍抽象类和接口。

- 异常处理是在抽象类和接口之前介绍的，因此，学生可以更早地编写健壮的程序。当然，教师也可以选择在后面教授异常处理。
- 前一版的第12章“面向对象设计和模式”被替换成将设计指南和模式分成几章，这样，就可以在合适的上下文中介绍这些主题。
- 关于排序的一章刚好放在关于算法效率的一章之后，这样，学生可以立即将算法效率应用在排序算法上。
- 全新的第44章^①介绍Java 2D。
- 关于数据结构的内容扩展为关于AVL树、splay树、2-4树、B树、红黑树以及散列的几章，所以本书也可以作为数据结构课程的完整教材。

学习策略

程序设计课程与其他课程有很大的区别。在程序设计课程中，学生要从例子中学习、从实践中学习、从错误中学习，需要花费大量的时间来编写程序、调试程序并修改错误。

对于刚接触程序设计的新手来说，学习Java与学习其他高级程序设计语言一样。学习程序设计的基本目的就是培养描述实际问题的程序化解决方案的关键技能，并通过条件语句、循环、方法和数组将方案转变成程序。

一旦掌握了使用循环、方法和数组编写程序的基本技能，就可以开始学习如何使用面向对象的方法开发大型程序和GUI程序。

一旦知道了如何编程并理解了面向对象程序设计的概念，那么，学习Java就变成了学习Java API。Java API为程序员搭建了使用Java开发应用程序的框架。必须使用API中的类和接口，并遵循它们的惯例和规则来创建应用程序。学习Java API最好的办法是模仿例子并进行练习。

教学特点

本书采用下列要素组织素材：

- **学习目标：**列出学生学习本章应该掌握的内容，有助于他们学完各章后判断自己是否达到了目标。
- **引言：**由一个典型的问题开始，讨论本章所能学到的内容。
- **问题：**以容易理解的方式仔细地挑选和描述问题，教授问题解决方案和程序设计概念。本书使用多个小的、简单的、令人兴奋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，有助于巩固本章所学的关键概念。
- **复习题：**按节组织，帮助学生评估学习状况。
- **编程练习题：**按节组织，给学生提供独立应用所学技能的机会。练习题的难度分为容易（没有星号）、适度（*）、难（**）和非常难（***）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。
- **LiveLab：**课程评估和管理系统。学生可以在线提交程序，系统会自动地给程序/多选题打分，并给出一个快速的反馈。教师可以自己定制程序设计练习题和测验题，并使用这个系统预建练习题和测验题。

① 第38~48章放在本书配套网站上，需要付费申请。——编辑注

• **注意、提示和警告：**贯穿全书，对程序开发的重要方面提供有价值的建议和深刻的认识。

注意 提供主题的附加信息，巩固重要概念。

提示 讲解好的程序设计风格和经验。

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集合架构。下页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。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，最后是开发复杂的Java应用程序。

第一部分 程序设计基础（第1~7章）

第一部分是基石，让你开始Java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、表达式以及运算符这样的基本程序设计技术（第2章），控制语句（第3~4章），方法（第5章），数组（第6~7章）。在第6章之后，可以跳到第20章去学习如何编写递归的方法来解决本质递归的问题。

第二部分 面向对象程序设计（第8~11、13~14和19章）

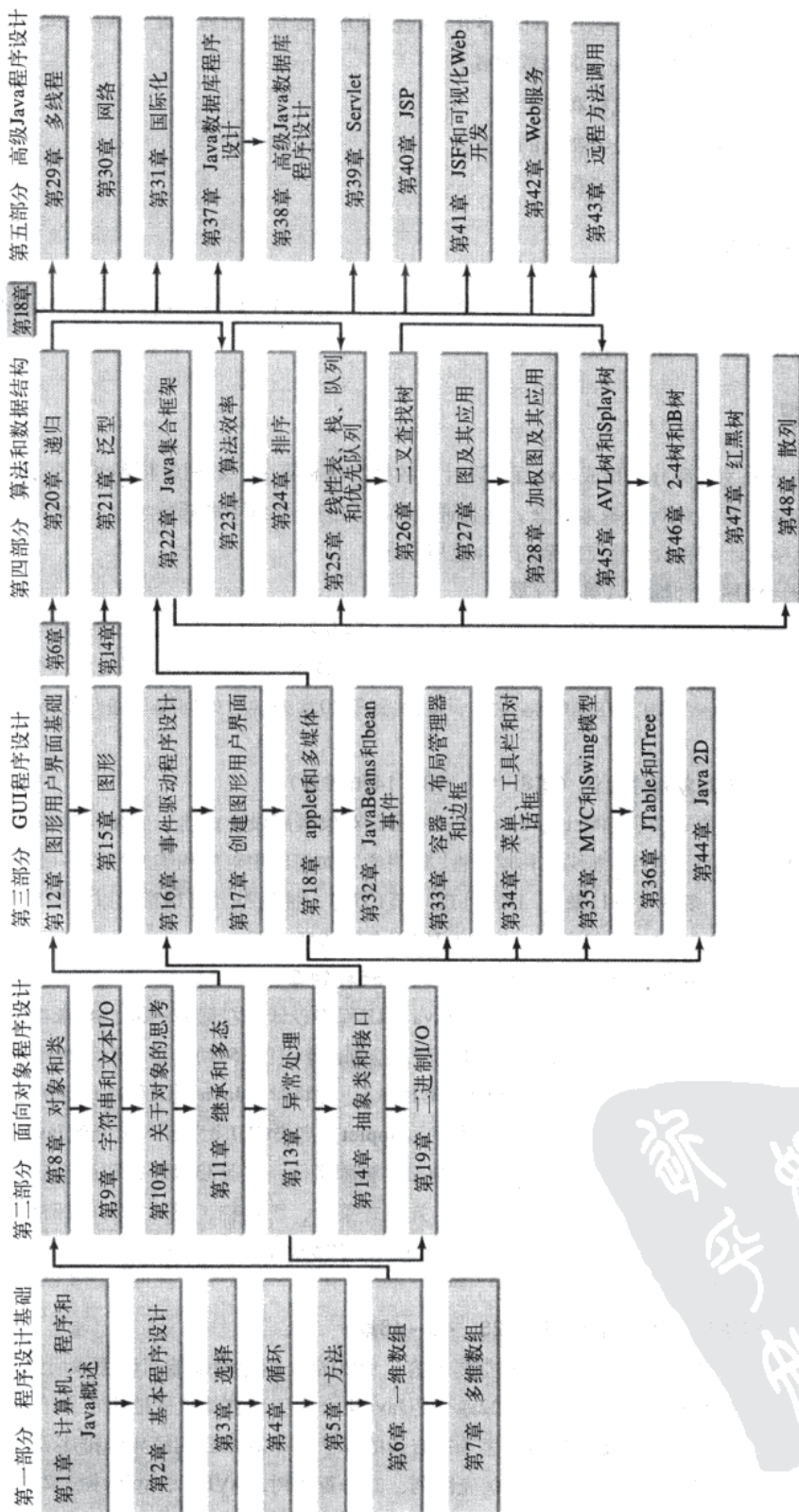
这一部分介绍面向对象程序设计。Java是一种面向对象程序设计语言，它使用抽象、封装、继承和多态来提供开发软件的灵活性、模块化和复用性。你将学习如何使用对象和类进行程序设计（第8~10章）、类的继承（第11章）、多态（第11章）、异常处理（第13章）、抽象类（第14章）以及接口（第14章）。处理字符串将在第9章和文本I/O一起介绍。二进制I/O将在第19章介绍。

第三部分 GUI程序设计（第12、15~18、32~36和44章）

这一部分在第12、15~18章中介绍基本的Java GUI程序设计，在第32~36、44章中介绍高级的Java GUI程序设计。主要的主题包括GUI基础（第12章）、绘制图形（第15章）、事件驱动程序设计（第16章）、创建图形用户界面（第17章）以及编写applet（第18章）。你将学习Java GUI程序设计的基础架构，并且使用来自基本GUI章节的GUI组件来开发应用程序和applet。高级GUI章节深入地介绍Java GUI程序设计。你将在第32章学习JavaBeans以及如何开发自定义事件和源组件，在第33章回顾和讨论新的容器、布局管理器以及边框，在第34章学习如何创建带菜单、弹出式菜单、工具栏、对话框和内部框架的GUI，在第35和36章使用MVC方法开发组件并讨论高级Swing组件JSpinner、JList、JComboBox、JTable和JTree。第44章介绍Java 2D。

第四部分 算法和数据结构（第20~28、45~48章）

这一部分介绍经典数据结构课程中的主要内容。第20章介绍递归来编写解决本质递归问题的方法。第21章介绍泛型来提高软件的复用性。第22章介绍Java集合框架，它为数据结构定义了一套有用的API。第23章介绍算法效率的度量以便给应用程序选择合适的算法。第24章介绍经典的排序算法。第25~26章和第45~47章介绍如何实现列表、队列、优先队列、二分查找树、AVL树、splay树、2-4树、B树以及红黑树的经典数据结构。第27和28章介绍图像应用程序。第48章介绍散列。



第五部分 高级Java程序设计（第29～31、37～43章）

这一部分是高级Java程序设计。第29章用多线程使程序具有更好的响应和交互。第30章介绍如何编写程序使得Internet上的不同主机能够相互通信。第31章介绍利用国际化支持来开发国际客户的项目。第37章介绍使用Java来开发数据库项目，第38章介绍高级Java数据库程序设计，而第39和40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41章介绍使用Java Server Faces进行快速Web应用程序开发。第42章介绍Web服务。第43章介绍远程方法调用。

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，TextPad、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。TextPad是一个基本的集成开发环境工具。NetBeans和Eclipse更加复杂，如果遵照指南，可以很容易地使用这些工具。关于TextPad、NetBeans和Eclipse的使用指南，参见本书配套网站上的补充材料。

LiveLab

本书有一个配套的基于Web的课程评估和管理系统。这个系统有以下三个主要组件：

- **自动打分系统：**它可以自动给书中的程序或教师自己创建的程序打分。
- **测验题的创建/提交/打分系统：**它可以让教师创建/修改学生所用的测验题，并且进行自动打分。
- **跟踪分数、考勤等：**它可以让教师跟踪分数，同时教师也可以看到所有学生的成绩以及跟踪他们的考勤。

自动打分系统的主要特征如下：

- 允许学生编译、运行和提交练习题。（系统检查他们的程序是否能正常运行——学生可以在到期日之前继续运行和重新提交程序。）
- 允许教师评阅提交的作业；用教师的测试用例来运行程序；更正程序；提交反馈给学生。
- 允许教师创建/修改自定义的练习题，创建公共的和秘密的测试用例，布置练习题，为整个班级或个人设置到期日。
- 可以将所有的练习题布置给学生。除此之外，LiveLab还提供了书中没有的附加练习题。
- 允许教师排序和过滤所有的练习题，并且检查成绩（依时间框、学生或练习题）。
- 允许教师从系统中删除学生。
- 允许学生和教师跟踪练习题的分数。

测试题系统的主要特征如下：

- 允许教师创建/修改来自试题库或文本文件的测验题，或者创建在线的全新测试题。
- 允许教师给全班学生或某个学生布置测验题，设置到期日和测试时间限制。
- 允许学生和教师查看已提交的测验题。
- 允许学生和教师跟踪测验成绩。

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（依章节组织）。
- LiveLab。
- 资源链接。
- 勘误表。

补充材料

本书的正文讲解基本主题。补充材料是正文的延伸，介绍读者可能感兴趣的附加主题。本书配套网站上可以访问的补充材料如下表所示。

配套网站上的补充材料

第I部分 常用的补充材料

- A 词汇表
- B 安装和配置JDK
- C 从命令窗口编译和运行Java
- D Java编码风格指南
- E 在Windows中为Java应用程序创建桌面快捷方式
- F 使用包组织教材中的类

第II部分 IDE补充材料

- A TextPad教程
- B NetBeans 6教程|一页启动指令
- C 使用NetBeans高效学习Java
- D Eclipse教程|一页启动指令
- E 使用Eclipse高效学习Java
- F JBuilder X教程|JBuilder 2005教程|一页启动指令

第III部分 Java补充材料

- G 使用JBuilder高效学习Java
- H JBuilder 2007教程
- 第III部分 Java补充材料
- A Java特征
- B 关于操作符和操作数的讨论
- C &和操作符
- D 按位操作
- E 带break和continue的语句标签
- F 枚举类型
- G 包
- H 正则表达式
- I 格式化字符串

J Object类中的方法

- K 隐藏数据域和静态方法
- L 初始化块
- M 关于覆盖方法的扩展讨论
- N 设计模式
- O JDK 1.5之前的文本I/O（Reader类和Writer类）
- P 断言
- Q 打包和部署Java工程
- R Java的Web启动
- S 签名的Java applet
- T GridBagLayout|OverlayLayout|SpringLayout
- U 使用数据报协议的网络
- V 创建内部的帧
- W 可插拔的Look和Feel
- X UML图形符号
- Y 使用JUnit测试类
- Z JNI
- AA StringTokenizer类
- AB SwingWorker和JProgressBar

第IV部分 数据库补充材料

- A 本书中用来创建和初始化表格的SQL语句
- B MySQL教程
- C Oracle教程
- D Microsoft Access教程
- E 数据库系统简介
- F 关系数据库概念
- G 数据库设计

H SQL基础	第VI部分 实例学习
I 高级SQL	A 完整的九宫格问题的解决方案（第7章）
第V部分 Web程序设计补充材料	B 地址簿（第19章）
A HTML和XHTML教程	C 骑士旅行问题（第27章） Knight Tour Applet
B CSS教程	第VII部分 常见错误
C XML	第VIII部分 有用链接
D Java和XML	A Java API
E Tomcat教程	B 排序算法视频
F 更多关于JSF和可视化Web开发的例子	

教师资源

教师资源可以从本书的配套网站下载。这些资源包括：

- 带源代码和运行程序的PowerPoint教学幻灯片。
- 教师解答手册。
- 计算机化测试题产生器。
- 使用多选题和简答题的样本测验，编写和跟踪程序，并且纠正程序设计的错误。
- LiveLab。
- 勘误表。

此外，教学幻灯片和教师解答手册也可从华章网站（www.hzbook.com）下载。

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写出来。教学是继续改进本书的灵感之源，感谢提出批评、建议、纠错报告和赞扬的教师和学生。

由于有了对本版和以前版本的富有见解的评审，本书得到很大的改进。感谢以下评审人员：Elizabeth Adams (James Madison University), Syed Ahmed (North Georgia College and State University), Omar Aldawud (Illinois Institute of Technology), Yang Ang (University of Wollongong, Australia), Kevin Bierre (Rochester Institute of Technology), David Champion (DeVry Institute), James Chegwiddden (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), Olac Fuentes (University of Texas at El Paso), Harold Grossman (Clemson University), Barbara Guillot (Louisiana State 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), 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), Shyamal Mitra (University of Texas at Austin), Michel Mitri (James Madison University), Kenrick Mock (University of Alaska Anchorage), 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), Ronald F. Taylor (Wright State University), Carolyn Schauble (Colorado State University), David Scuse (University of Manitoba), Ashraf Shirani (San Jose State University), Daniel Spiegel (Kutztown University), Amr Sabry (Indiana University), Lixin Tao (Pace University), Russ Tront (Simon Fraser University), Deborah Trytten (University of Oklahoma), 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 Dunkelberger和她的同事Marcia Horton、Margaret Waples、Erin Davis、Michael Hirsh、Matt Goldstein、Jake Warde、Melinda Haggerty、Allison Michael、Scott Disanno、Irwin Zucker，感谢他们组织、开展和积极促进本项目，同时感谢Robert Lentz的编辑工作。

一如既往，感谢我妻子Samantha的爱、支持和鼓励。

Y. Daniel Liang

y.daniel.liang@gmail.com

www.cs.armstrong.edu/liang

www.pearsonhighered.com/liang



出版者的话

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1

1.2.1 中央处理器2

1.2.2 内存2

1.2.3 存储设备3

1.2.4 输入和输出设备3

1.2.5 通信设备4

1.3 程序4

1.4 操作系统5

1.4.1 控制和监视系统的活动6

1.4.2 分配和调遣系统资源6

1.4.3 安排操作的顺序6

1.5 Java、万维网以及其他6

1.6 Java语言规范、API、JDK和IDE8

1.7 一个简单的Java程序8

1.8 创建、编译和执行Java程序10

1.9 (GUI) 在消息对话框中显示文本12

关键术语14

本章小结15

复习题15

编程练习题17

第2章 基本程序设计18

2.1 引言18

2.2 编写简单的程序18

2.3 从控制台读取输入21

2.4 标识符22

2.5 变量23

2.6 赋值语句和赋值表达式24

2.7 定名常量24

2.8 数值数据类型及其运算25

2.8.1 数值运算符26

2.8.2 数值直接量27

2.8.3 计算Java表达式28

2.9 问题：显示当前时间29

2.10 简捷运算符31

2.11 数值类型转换32

2.12 问题：计算贷款支付额33

2.13 字符数据类型及运算35

2.13.1 统一码和ASCII码35

2.13.2 特殊字符的转义序列36

2.13.3 字符型char数据与数值型数据
之间的转换37

2.14 问题：整钱兑零37

2.15 String类型39

2.16 程序设计风格和文档41

2.16.1 适当的注释和注释风格41

2.16.2 命名习惯41

2.16.3 适当的缩进和空白41

2.16.4 块的风格42

2.17 程序设计错误42

2.17.1 语法错误42

2.17.2 运行错误43

2.17.3 逻辑错误43

2.17.4 调试43

2.18 (GUI) 从输入对话框获取输入44

2.18.1 将字符串转换为数字44

2.18.2 使用输入对话框44

关键术语46

本章小结46

复习题	47	4.7 最小化数值误差	102
编程练习题	50	4.8 实例学习	103
第3章 选择	55	4.8.1 举例: 求最大公约数	104
3.1 引言	55	4.8.2 问题: 预测未来学费	105
3.2 boolean数据类型	55	4.8.3 问题: 蒙特卡罗模拟	105
3.3 问题: 一个简单的数学学习工具	56	4.9 关键字break和continue	106
3.4 if语句	57	4.10 (GUI) 使用确认对话框控制循环	110
3.5 问题: 猜生日	58	关键术语	111
3.6 双向if语句	61	本章小结	111
3.7 嵌套的if语句	62	复习题	112
3.8 选择语句中的常见错误	63	编程练习题	116
3.9 问题: 一个改进的数学学习工具	64	第5章 方法	123
3.10 问题: 计算身体质量指数	66	5.1 引言	123
3.11 问题: 计算税款	67	5.2 定义方法	124
3.12 逻辑运算符	69	5.3 调用方法	125
3.13 问题: 判定闰年	71	5.4 void方法举例	127
3.14 问题: 彩票	72	5.5 参数的值传递	129
3.15 switch语句	74	5.6 模块化代码	130
3.16 条件表达式	75	5.7 问题: 将十进制数转换为十六进制数	132
3.17 格式化控制台输出	76	5.8 重载方法	134
3.18 运算符的优先级和结合方向	77	5.9 变量的作用域	136
3.19 (GUI) 确认对话框	78	5.10 Math数学类	136
关键术语	80	5.10.1 三角函数方法	137
本章小结	80	5.10.2 指数函数方法	137
复习题	80	5.10.3 取整方法	138
编程练习题	84	5.10.4 min、max和abs方法	138
第4章 循环	90	5.10.5 random方法	139
4.1 引言	90	5.11 实例学习: 生成随机字符	139
4.2 while循环	91	5.12 方法抽象和逐步求精	141
4.2.1 举例: 猜数字	92	5.12.1 自顶向下的设计	141
4.2.2 循环设计策略	94	5.12.2 自顶向下和自底向上的实现	143
4.2.3 问题: 高级数学学习工具	94	5.12.3 实现细节	144
4.2.4 使用标志值控制循环	96	关键术语	146
4.2.5 输入和输出重定向	97	本章小结	147
4.3 do-while循环	98	复习题	147
4.4 for循环	99	编程练习题	150
4.5 采用哪种循环	100	第6章 一维数组	157
4.6 嵌套循环	101	6.1 引言	157

6.2 数组的基本知识	158
6.2.1 声明数组变量	158
6.2.2 创建数组	158
6.2.3 数组大小和默认值	159
6.2.4 数组下标变量	159
6.2.5 数组初始化语法	160
6.2.6 处理数组	160
6.2.7 for-each循环	161
6.3 问题：乐透号码	162
6.4 问题：一副牌	164
6.5 数组的复制	166
6.6 给方法传递数组	167
6.7 从方法中返回数组	169
6.8 可变长参数列表	172
6.9 数组的查找	172
6.9.1 线性查找法	173
6.9.2 二分查找法	173
6.10 数组的排序	175
6.10.1 选择排序	175
6.10.2 插入排序	177
6.11 Arrays类	178
关键术语	179
本章小结	179
复习题	180
编程练习题	182
第7章 多维数组	188
7.1 引言	188
7.2 二维数组的基础知识	188
7.2.1 声明二维数组变量并创建 二维数组	188
7.2.2 获取二维数组的长度	189
7.2.3 锯齿数组	190
7.3 处理二维数组	190
7.4 给方法传递二维数组	192
7.5 问题：多选题测验评分	192
7.6 问题：找出距离最近的点对	194
7.7 问题：九宫格	195
7.8 多维数组	198
7.8.1 问题：每日温度和湿度	199
7.8.2 问题：猜生日	200

本章小结	201
复习题	201
编程练习题	202
第8章 对象和类	210
8.1 引言	210
8.2 定义对象的类	210
8.3 举例：定义类和创建对象	212
8.4 使用构造方法构造对象	216
8.5 通过引用变量访问对象	216
8.5.1 引用变量和引用类型	216
8.5.2 访问对象的数据和方法	217
8.5.3 引用数据域和null值	217
8.5.4 基本类型变量和引用类型 变量的区别	218
8.6 使用Java库中的类	219
8.6.1 Date类	219
8.6.2 Random类	219
8.6.3 显示GUI组件	220
8.7 静态变量、常量和方法	222
8.8 可见性修饰符	225
8.9 数据域封装	227
8.10 给方法传递对象参数	229
8.11 对象数组	230
关键术语	232
本章小结	232
复习题	233
编程练习题	237
第9章 字符串和文本I/O	241
9.1 引言	241
9.2 字符串类String	241
9.2.1 构造一个字符串	241
9.2.2 不可变字符串与限定字符串	242
9.2.3 字符串的比较	242
9.2.4 字符串长度、字符以及组合 字符串	243
9.2.5 获取子串	244
9.2.6 字符串的转换、替换和分隔	245
9.2.7 依照模式匹配、替换和分隔	246

9.2.8 找出字符串中的某个字符或者 某个子串	246	10.11 类的设计原则	292
9.2.9 字符串与数组之间的转换	247	10.11.1 内聚性	292
9.2.10 将字符和数值转换成字符串	247	10.11.2 一致性	292
9.2.11 格式化字符串	248	10.11.3 封装性	292
9.2.12 问题: 检测回文串	248	10.11.4 清晰性	292
9.2.13 问题: 将十六进制转换为十进制	249	10.11.5 完整性	293
9.3 字符类Character	250	10.11.6 实例和静态	293
9.4 StringBuilder/StringBuffer类	252	关键术语	293
9.4.1 修改StringBuilder中的字符串	253	本章小结	294
9.4.2 toString、capacity、length、 setLength和charAt方法	254	复习题	294
9.4.3 问题: 忽略既非字母又非数字的 字符, 判断回文串	255	编程练习题	295
9.5 命令行参数	256	第11章 继承和多态	301
9.5.1 向main方法传递字符串	257	11.1 引言	301
9.5.2 问题: 计算器	257	11.2 父类和子类	301
9.6 文件类File	258	11.3 使用super关键字	306
9.7 文件输入和输出	260	11.3.1 调用父类的构造方法	306
9.7.1 使用PrintWriter写数据	261	11.3.2 构造方法链	307
9.7.2 使用Scanner读数据	262	11.3.3 调用父类的方法	308
9.7.3 Scanner如何工作	263	11.4 覆盖方法	309
9.7.4 问题: 替换文本	263	11.5 覆盖和重载	309
9.8 (GUI) 文件对话框	264	11.6 对象类Object和它的toString() 方法	310
本章小结	265	11.7 多态	310
复习题	266	11.8 动态绑定	311
编程练习题	270	11.9 对象转换和instanceof运算符	312
第10章 关于对象的思考	276	11.10 Object的equals方法	314
10.1 引言	276	11.11 数组线性表ArrayList类	315
10.2 不可变对象和类	276	11.12 自定义栈类	318
10.3 变量的作用域	277	11.13 protected数据和方法	319
10.4 this引用	278	11.14 防止扩展和覆盖	320
10.5 类的抽象和封装	279	关键术语	320
10.6 面向对象的思考	282	本章小结	321
10.7 对象的组合	284	复习题	321
10.8 设计类Course	286	编程练习题	326
10.9 设计堆栈类	287	第12章 图形用户界面基础	328
10.10 设计GuessDate类	290	12.1 引言	328
		12.2 Swing和AWT	328
		12.3 Java GUI API	329

12.3.1 组件类	329	复习题	364
12.3.2 容器类	329	编程练习题	368
12.3.3 GUI辅助类	330		
12.4 框架	330	第14章 抽象类和接口	370
12.4.1 创建一个框架	330	14.1 引言	370
12.4.2 向框架中添加组件	331	14.2 抽象类	370
12.5 布局管理器	332	14.2.1 为什么要用抽象方法	372
12.5.1 FlowLayout	333	14.2.2 关于抽象类的几个关注点	374
12.5.2 GridLayout	334	14.3 举例: 日历类Calendar和公历类 GregorianCalendar	374
12.5.3 BorderLayout	336	14.4 接口	376
12.5.4 布局管理器的属性	337	14.5 举例: Comparable接口	378
12.6 使用面板作为子容器	337	14.6 举例: ActionListener接口	380
12.7 Color类	339	14.7 举例: Cloneable接口	381
12.8 Font类	339	14.8 接口与抽象类	384
12.9 Swing GUI组件的公共特性	340	14.9 将基本数据类型值作为对象处理	386
12.10 图像图标	342	14.10 举例: 对一个对象数组排序	388
关键术语	343	14.11 基本类型和包装类类型之间的 自动转换	390
本章小结	343	14.12 BigInteger和BigDecimal类	390
复习题	344	14.13 实例学习: Rational类	391
编程练习题	346	关键术语	395
第13章 异常处理	348	本章小结	395
13.1 引言	348	复习题	396
13.2 异常处理概述	348	编程练习题	399
13.3 异常处理的优势	351		
13.4 异常类型	353	第15章 图形	402
13.5 关于异常处理的更多知识	354	15.1 引言	402
13.5.1 声明异常	355	15.2 图形坐标系	402
13.5.2 抛出异常	355	15.3 Graphics类	403
13.5.3 捕获异常	355	15.4 绘制字符串、直线、矩形和椭圆	405
13.5.4 从异常中获取信息	357	15.5 实例学习: FigurePanel类	406
13.5.5 举例: 声明、抛出和捕获异常	358	15.6 绘制弧形	409
13.6 finally子句	360	15.7 绘制多边形和折线段	410
13.7 何时使用异常	361	15.8 使用FontMetrics类居中显示 字符串	413
13.8 重新抛出异常	361	15.9 实例学习: MessagePanel类	414
13.9 链式异常	362	15.10 实例学习: StillClock类	418
13.10 创建定制异常类	362	15.11 显示图像	422
关键术语	364	15.12 实例学习: ImageViewer类	423
本章小结	364	本章小结	425

复习题	426	本章小结	492
编程练习题	427	复习题	492
第16章 事件驱动程序设计	432	编程练习题	493
16.1 引言	432	第18章 applet和多媒体	498
16.2 事件和事件源	432	18.1 引言	498
16.3 监听器、注册以及处理事件	434	18.2 开发applet	499
16.4 内部类	438	18.3 HTML文件和<applet>标记	499
16.5 匿名类监听器	439	18.3.1 从Web浏览器查看applet	501
16.6 定义监听器类的另一种方式	441	18.3.2 使用applet查看器工具查看applet	501
16.7 问题: 贷款计算器	443	18.4 applet安全限制	501
16.8 窗口事件	444	18.5 让applet像应用程序一样运行	502
16.9 监听器接口适配器	446	18.6 applet生命周期方法	503
16.10 鼠标事件	447	18.6.1 init方法	503
16.11 按键事件	449	18.6.2 start方法	503
16.12 使用Timer类的动画	451	18.6.3 stop方法	503
关键技术	454	18.6.4 destroy方法	503
本章小结	454	18.7 给applet传递字符串	504
复习题	455	18.8 实例学习: 弹跳的小球	507
编程练习题	456	18.9 实例学习: 井字游戏	510
第17章 创建图形用户界面	462	18.10 使用URL类定位资源	513
17.1 引言	462	18.11 在任意Java程序中播放音频	515
17.2 按钮	463	18.12 实例学习: 多媒体动画	516
17.2.1 图标、按下图标和翻转图标	463	关键技术	518
17.2.2 对齐方式	464	本章小结	518
17.2.3 文本位置	465	复习题	519
17.2.4 使用按钮	465	编程练习题	520
17.3 复选框	467	第19章 二进制I/O	527
17.4 单选按钮	470	19.1 引言	527
17.5 标签	472	19.2 在Java中如何处理输入/输出	527
17.6 文本域	473	19.3 文本I/O与二进制I/O	528
17.7 文本区域	475	19.4 二进制I/O类	529
17.8 组合框	478	19.4.1 FileInputStream类和 FileOutputStream类	530
17.9 列表框	481	19.4.2 FilterInputStream类和 FilterOutputStream类	532
17.10 滚动条	484	19.4.3 DataInputStream类和 DataOutputStream类	532
17.11 滑块	486	19.4.4 BufferedInputStream类和 BufferedOutputStream类	535
17.12 创建多个窗口	489		

19.5 问题：复制文件	536	20.6 问题：求出目录的大小	558
19.6 对象的输入/输出	537	20.7 问题：汉诺塔	559
19.6.1 可序列化接口 <code>Serializable</code>	539	20.8 问题：分形	562
19.6.2 序列化数组	540	20.9 问题：八皇后	564
19.7 随机访问文件、.....	541	20.10 递归与迭代	567
关键术语	544	20.11 尾递归	567
本章小结	544	关键术语	568
复习题	544	本章小结	568
编程练习题	547	复习题	568
编程练习题	547	编程练习题	569
第20章 递归	550	附录A Java关键字	575
20.1 引言	550	附录B ASCII码字符集	576
20.2 问题：计算阶乘	550	附录C 运算符优先级表	577
20.3 问题：计算斐波那契数	553	附录D Java修饰符	578
20.4 使用递归解决问题	554	附录E 特殊浮点值	579
20.5 递归的辅助方法	556	附录F 数系	580
20.5.1 选择排序	556		
20.5.2 二分查找	557		



计算机、程序和Java概述

学习目标

- 回顾计算机的基本组成、程序和操作系统 (1.2~1.4节)。
- 探究Java与万维网 (World Wide Web) 之间的关系 (1.5节)。
- 区分术语API、IDE和JDK (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)。
- (GUI) 使用JOptionPane输出对话框显示输出结果 (1.9节)。

1.1 引言

人们使用字处理程序编写文档,使用Web浏览器在互联网中探索,使用电子邮件程序发送电子邮件。这些字处理程序、Web浏览器以及电子邮件程序都是在计算机上运行的软件实例。软件是用程序设计语言开发出来的。程序设计语言有很多种——那么为什么要选择Java这种程序设计语言呢?答案就是Java能够让用户开发和部署可用于Internet上的服务器、台式电脑和小型手持设备的应用程序。互联网将会对计算技术的未来产生深远的影响,而Java肯定会在未来计算技术中占很大的比重。所以说,Java是一种真正的互联网程序设计语言。

我们即将开始一段激动人心的旅行,学习一门功能强大的程序设计语言。在开始这一奇幻之旅之前,很有必要回顾一下计算机的基本组成、程序和操作系统,熟悉一下数字系统。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,那么你可以跳过1.2~1.4节中对这些内容的回顾。

1.2 什么是计算机

计算机是存储和处理数据的电子设备。它包括硬件 (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要求硬件完成特定的任务。编写指令让计算机来完成的过程就称为计算机程序设计。要学习一门程序设计语言,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 (如图1-1所示):

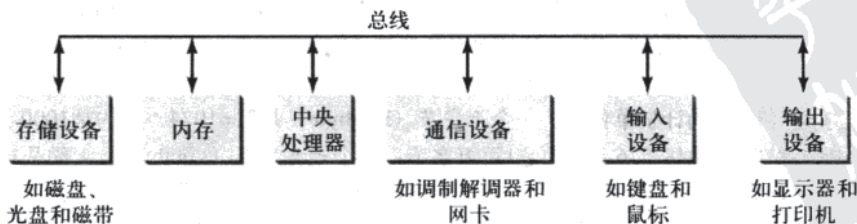


图1-1 计算机由中央处理器、内存、存储设备、输入设备、输出设备和通信设备组成

2 · 第 1 章 计算机、程序和Java概述

- 中央处理器 (CPU)
- 内存 (主存)
- 存储设备 (例如, 磁盘、光盘、磁带)
- 输入和输出设备 (例如, 显示器、键盘、鼠标、打印机)
- 通信设备 (例如, 调制解调器和网络接口卡 (NIC, 简称网卡))

这些组件通过一个称为总线 (bus) 的子系统连接, 总线负责在这些组件之间传输数据和电信号。

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除CPU之外其他组件的动作。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都是内嵌在一块小小的硅半导体芯片里, 这块芯片上有数百万个晶体管。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, 它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。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)。CPU的速度在不断地提高。Intel公司的奔腾3处理器的运行速度是500MHz左右, 而奔腾4处理器的运行速度大约是3GHz (1GHz是1000MHz)。

1.2.2 内存

为了存储和处理信息, 计算机利用电的两种状态: 关 (off) 和开 (on), 习惯上认为它们分别是0和1。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)。各种类型的数据, 例如, 数字、字符和字符串, 都被编码为比特序列。CPU要执行的数据和程序指令都以一组比特或字节的形式存储在计算机内存 (memory) 中, 每个字节由8比特构成。内存单元是由字节 (byte) 构成的有序序列, 如图1-2所示。

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表来自动完成的。例如, 在流行的ASCII编码表中, 字符 'J' 是用一个字节01001010来表示的。

字节是最小的存储单元。像3这样的小数字就可以存储在单个字节中。为了存储一个单个字节放不下的大数字, 计算机需要使用几个相邻的字节。任何两个数据都不能共享或分割同一个字节。

内存中字节的内容永远非空, 但是它的原始内容可能对于你的程序来说是毫无意义的。一旦新的信息被放入内存字节, 该字节的当前内容就会丢失。

程序及其所需数据必须在它们被执行前放入内存。

每个字节都有一个唯一的地址。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)。现在的个人电脑通常至少有1GB的RAM。计算机的存储大小是以字节、千字节 (KB)、兆字节 (MB)、千兆字节 (GB) 和万亿字节 (TB) 为计量单位的。一个千字节 (kilobyte) 是 $2^{10}=1024$, 大约是1000字节; 一个兆字节 (megabyte) 是 $2^{20}=1\,048\,576$, 大约是1百万字节; 一个千兆字节 (gigabyte) 大约是10亿字节; 而一个万亿字节 (terabyte) 大约是1000千兆字节。同CPU一样, 内存也是内置在一个表面上嵌有成千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。

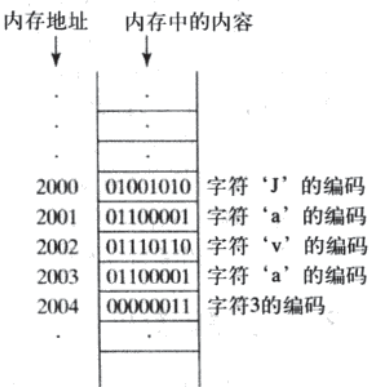


图1-2 内存存储数据和程序指令

1.2.3 存储设备

内存是不能长久保存数据的，因为断电时信息就会丢失。程序和数据都会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四种类型：

- 磁盘驱动器
- 光盘驱动器（只读光盘（CD-R）、可擦写光盘（CD-RW）和数字化视频磁盘（DVD））
- 磁带驱动器
- 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、光盘和磁带等都是存储介质。

1. 磁盘

每台计算机至少有一个硬盘驱动器。硬盘（hard disk）是用来永久地存储数据和程序的。在最新的个人电脑上，硬盘容量一般为80GB~250GB。磁盘驱动器通常都安装在计算机内。此外，还可以用移动硬盘。

2. 光盘和数字化视频磁盘

CD的全称是致密的盘片。光盘驱动器的类型有两种：只读光盘（CD-R）和可擦写光盘（CD-RW）。只读光盘只能用于读取那些永久存储在光盘上的信息，内容一旦记录到光盘上，用户是不能修改它们的。可擦写光盘可以像硬盘一样使用，对这类光盘可以进行读取和重写操作。一张光盘的容量可以达到700MB。多数软件都是存储在只读光盘上发布的。大多数新型的个人电脑都安装了可擦写光驱，它既支持只读光盘也支持可擦写光盘。

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，可以使用任何一种来存储数据。一张DVD上可以保存的信息要比一张CD保存的信息多。一张标准DVD的存储容量是4.7GB。

3. 磁带

磁带（tape）主要用于备份数据和程序。不同于磁盘和光盘，磁带是顺序存储信息的。计算机必须按照信息被存储的顺序来获取信息。磁带速度非常慢。备份1GB的硬盘需要一两个小时。新的趋势是使用闪存或者外挂硬盘来备份数据。

4. USB闪存驱动器

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32GB。

1.2.4 输入和输出设备

用户是通过输入设备和输出设备与计算机进行通信的。通常，输入设备是指键盘（keyboard）和鼠标（mouse），而输出设备是指显示器（monitor）和打印机（printer）。

1. 键盘

计算机键盘（keyboard）很像一个附带完成某些特定功能的特殊键的打字机键盘。

功能键（function key）位于键盘的最上边，而且都是以F为前缀。它们的功能取决于软件的设置。

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，是为了快速输入数字而独立出来的一套按键集合。

方向键（arrow key）位于主键盘和数字小键盘之间，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）都位于方向键的上方，分别用来在字处理过程中完成插入、删除、向上翻页和向下翻页的功能。

4 • 第 1 章 计算机、程序和Java概述

2. 鼠标

鼠标 (mouse) 是定点设备, 是用来在屏幕上移动光标的电子指针, 或者用于点击屏幕上的对象来触发它以响应这个动作。

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) 是指每平方英寸的像素数。像素 (“图像元素” 的简称) 就是在屏幕上构成图像的小点。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。

1.2.5 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、网络接口卡以及无线通信设备。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) 使用的也是电话线, 但是传输数据的速度比拨号调制解调器快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, 速度与DSL一样快。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10BaseT的典型NIC能够以10Mbps (Mbps表示每秒百万比特) 的速度传输数据。无线通信正在变得越来越流行。现在, 每台手提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Internet上。

1.3 程序

计算机程序 (program) 通常称为软件 (software), 是发给计算机的指令, 告诉计算机该做什么。因为计算机不理解人类的语言, 所以, 需要在计算机程序中使用计算机语言。程序设计 (programming) 就是创建一个可以让计算机执行并完成所需任务的程序。

计算机本身的语言因计算机类型的不同而有差异, 计算机本身的语言就是它的机器语言 (machine language) ——最初植入计算机的一套原始指令集。因为这些指令都是以二进制代码的形式存在, 所以, 为了告诉机器该做什么, 必须输入二进制代码。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。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
```

汇编语言 (assembly language) 是一种低级的程序设计语言, 它用助记符表示每一条机器语言指令。例如, 为进行两数相加, 用汇编代码所编写的指令形式如下:

```
ADD3 R1, R2, R3
```

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, 所以需要使用时一种称为汇编器 (assembler) 的程序将汇编语言程序转换为机器代码, 如图1-3所示。

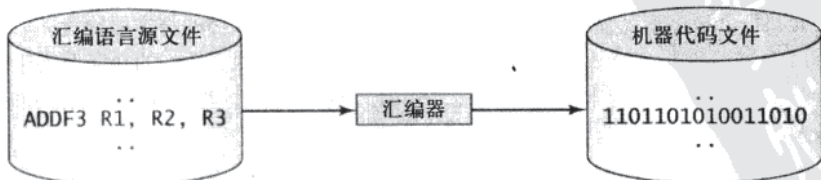


图1-3 汇编器将汇编语言指令翻译为机器码

汇编程序是用易于记忆的助记符形式的机器指令编写的。因为汇编语言具有机器依赖性, 所以汇编程序只能在某种特定的机器上执行。为了克服平台依赖性问题以及降低程序设计难度, 开发了高级语言。

高级语言 (high-level language) 很像英语, 易于学习和编写程序。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 * 3.1415;
```

在一百多种高级语言中, 以下几种是很著名的:

- COBOL (面向商业的通用语言)
- FORTRAN (公式翻译)
- BASIC (初学者通用符号指令代码)
- Pascal (以Blaise Pascal命名)
- Ada (以Ada Lovelace命名)
- C (由B的设计者开发)
- Visual Basic (Microsoft公司开发的类似Basic的可视化语言)
- Delphi (Borland公司开发的类似Pascal的可视化语言)
- C++ (基于C语言的一种面向对象程序设计语言)
- C# (Microsoft公司开发的类似Java的语言)
- Java

这里的每一种语言都是为了特定目的而设计的。COBOL是为商业应用而设计的, 现在主要用于商业数据处理。FORTRAN为数学运算而设计的, 主要用于数值计算。BASIC是为了易学易用而设计的。Ada是为美国国防部开发的, 主要用于国防项目。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。Visual Basic和Delphi用于开发图形用户界面, 还可以进行快速应用开发。C++非常适合开发系统软件项目, 例如, 编写编译器和操作系统。Microsoft公司的Windows操作系统就是用C++编写的。C# (读作: C Sharp) 是由微软开发出来的新语言, 用来开发基于微软.NET平台的应用程序。Java是由Sun公司开发的, 广泛用于开发一些独立于平台的互联网应用程序。

用高级语言编写的程序称为源程序 (source program) 或源代码 (source code)。由于计算机不能理解源程序, 所以, 要使用称为编译器 (compiler) 的程序将源程序翻译成机器语言程序。然后, 这个机器语言程序再与其他辅助的库代码进行链接, 构成可执行文件, 该文件就可以在机器上运行, 如图1-4所示。在Windows平台上, 可执行文件的扩展名是.exe。

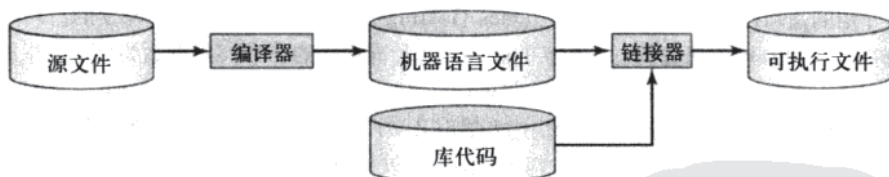


图1-4 源程序被编译为机器语言文件, 然后和系统库链接形成可执行文件

1.4 操作系统

操作系统 (Operating System, OS) 是运行在计算机上最重要的程序, 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

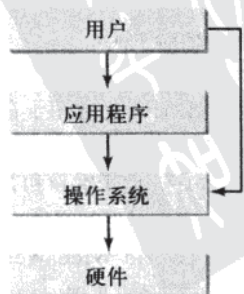


图1-5 操作系统是控制和管理系统的软件

6 • 第1章 计算机、程序和Java概述

- 分配和调遣系统资源
- 安排操作的顺序

1.4.1 控制和监视系统的活动

操作系统完成基本的功能，例如，识别来自键盘的输入，向显示器发送输出结果，保存磁盘中文件和目录的路径，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 and 用户同时使用计算机时不会相互干扰，而且要对安全负责，以确保未经授权的用户无权访问系统。

1.4.2 分配和调遣系统资源

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遣以运行程序。

1.4.3 安排操作的顺序

操作系统负责确定程序的执行顺序，以便有效地利用系统资源。为了提高系统的性能，目前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。

多线程允许在一个程序内部进行并发处理，这样，程序的子任务就可以同时运行。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。这两个任务是并发运行的独立线程。

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完成同一个任务。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。

1.5 Java、万维网以及其他

本书介绍的是Java程序设计。Java是James Gosling在Sun公司领导的小组开发的。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Internet应用程序。关于Java的历史，参见www.java.com/en/javahistory/index.jsp。

近年来，Java变得非常流行。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：一旦编写了一个程序，在任何地方都可以运行。就像Sun公司声称的，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）、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，可以用来开发可靠的、要求严格的应用程序。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三十多年。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。

万维网上的主要创作语言是超文本标记语言 (Hypertext Markup Language, HTML)。HTML是一种简单的语言：能够部署Internet上的文档、链接互联网上的文档，能够在万维网上提供生动的图像、声音和视频。但是，除了简单的表单外，它不能和用户进行交互。用HTML制作的Web网页基本上都是静态的，也是很单调的。

Java一开始就富有吸引力，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使用流行的图形用户界面与网络用户进行交互，处理用户的要求，这些界面中包括了按钮、文本字段、文本域、单选按钮等。applet使得网页更加灵活、生动和易于交互。图1-6显示了一个在Web浏览器上运行井字游戏的applet。

在Web浏览器中输入这个URL

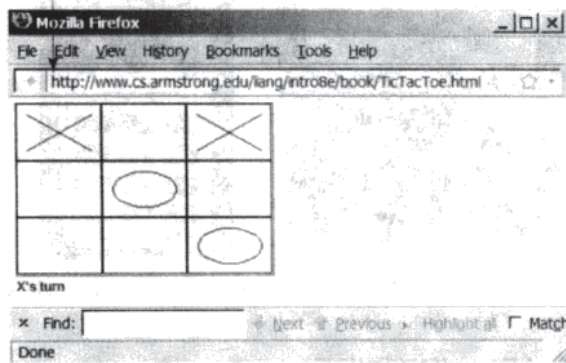


图1-6 井字游戏的Java applet内嵌在一个HTML页面中

提示 要查看Java applet的演示，请访问网站java.sun.com/applets。该网站提供了丰富的Java资源，还有很多其他很酷的演示applet的站点链接。java.sun.com是Sun公司的Java官方网站。

Java还可以用来开发服务器端的应用程序。这些应用程序可以在Web服务器上运行，生成动态网页。如图1-7所示，这是本书的自动打分系统，该系统就是用Java开发出来的。

在Web浏览器中输入这个URL

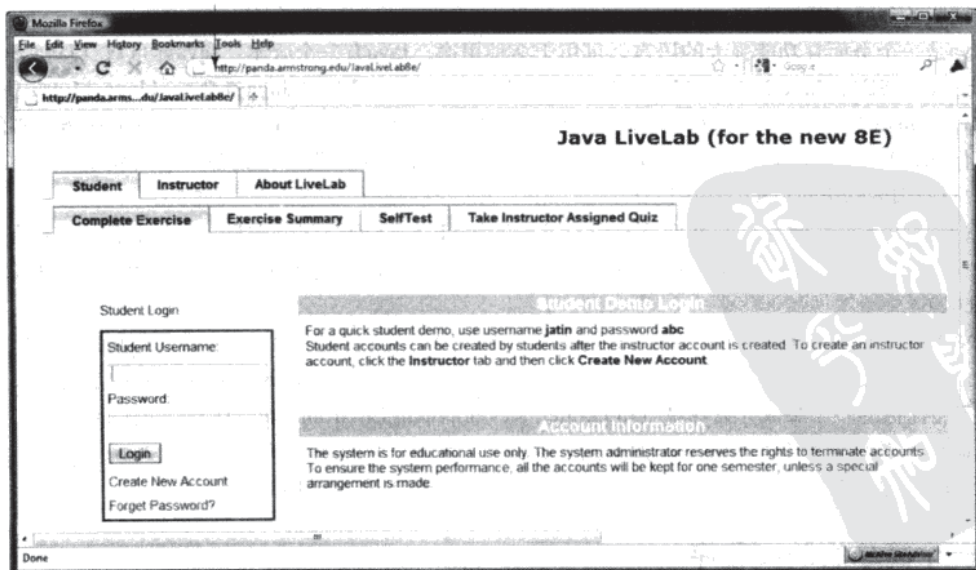


图1-7 用Java开发和本书配套的自动打分系统

8 • 第1章 计算机、程序和Java概述

Java是一个功能强大的程序设计语言，可以用它来开发台式机和服务器上的应用程序，也可以用它来开发小型手持设备上的应用程序。图1-8演示了用Java编写的在个人数字助理BlackBerry和手机上显示的日历。



图1-8 可以使用Java来开发手持设备和无线设备上的应用程序，如BlackBerry（左）和手机（右）

1.6 Java语言规范、API、JDK和IDE

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则，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，它包括Java程序设计语言的语法和语义。完整的Java语言规范可以在java.sun.com/docs/books/jls上找到。

应用程序接口（Application Program Interface, API）包括为开发Java程序而预定义的类和接口。Java语言的规范是稳定的，但是API一直在扩展。在Sun公司的Java网站上（java.sun.com），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）、Java企业版（Java Enterprise Edition, Java EE）、Java微型版（Java Micro Edition, Java ME）。Java SE可以用来开发客户端独立的应用程序或applet。Java EE可以用来开发服务器端的应用程序，例如，Java servlet和JavaServer Pages。Java ME可以用来开发移动设备的应用程序，例如，手机等。本书使用Java SE介绍Java程序设计。

Java SE也有很多版本，本书采用最新的版本Java SE 6。Sun公司发布Java的各个版本都带有Java开发工具包（Java Development Toolkit, JDK），Java SE 6对应的Java开发工具包称为JDK 1.6（也称为Java 6或者JDK 6）。

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）——它们是为了快速开发Java程序而提供的一个集成开发环境（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，然后点击按钮、菜单选项或者使用功能键就可以编译和运行源代码。

1.7 一个简单的Java程序

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) 是指计算机的文本输入和显示设备。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! to the console
4         System.out.println("Welcome to Java!");
5     }
6 }
```

Welcome to Java!



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)。为了运行某个类, 该类必须包含名为main的方法。程序是从main方法开始执行的。

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)。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, 它能知道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它们的。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 prints Welcome to Java!
/* This application program prints Welcome to Java! */
/* This application program
   prints Welcome to Java! */
```

程序中的一对花括号将程序的一些组件组合起来, 形成一个块 (block)。在Java中, 每个块以左括号 ({) 开始, 以右括号 (}) 结束。每个类都有一个将该类的数据和方法放在一起的类块 (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!");
    }
}
```

类块

方法块

提示 一个左括号必须匹配一个右括号。任何时候, 当你输入一个左括号时, 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。

注意 你可能想知道为什么main方法要以这样的方式声明, 为什么使用System.out.println(...) 就可以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。这一问题将在后续的章节中得到完整的回答。

警告 Java源程序是区分大小写的。例如, 在该程序中用Main代替main是错误的。

注意 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，或者拼错了关键字String，Java编译器会报告语法错误。尝试去编译带有这些错误的程序，看看编译器会报告些什么。

程序清单1-1中的程序会显示一条消息。一旦你理解了这个程序，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消息，如程序清单1-2所示。

程序清单1-2 Welcome1.java

```
1 public class Welcome1 {
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
```



还可以进一步完成科学计算，并将结果显示到控制台上。程序清单1-3给出计算 $\frac{10.5 + 2 \times 3}{45 - 3.5}$ 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中的乘法运算符是*。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。

1.8 创建、编译和执行Java程序

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1-9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（editor）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。如果希望使用像Eclipse、NetBeans或者TextPad这样的IDE，请参考补充材料II。从命令窗口，可以使用记事本（NotePad）来创建Java源代码文件，如图1-10所示。

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

```
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

(NotePad) 和写字板 (WordPad) 来创建和编辑文件。所有补充材料都可以在本书配套的网站上访问到。

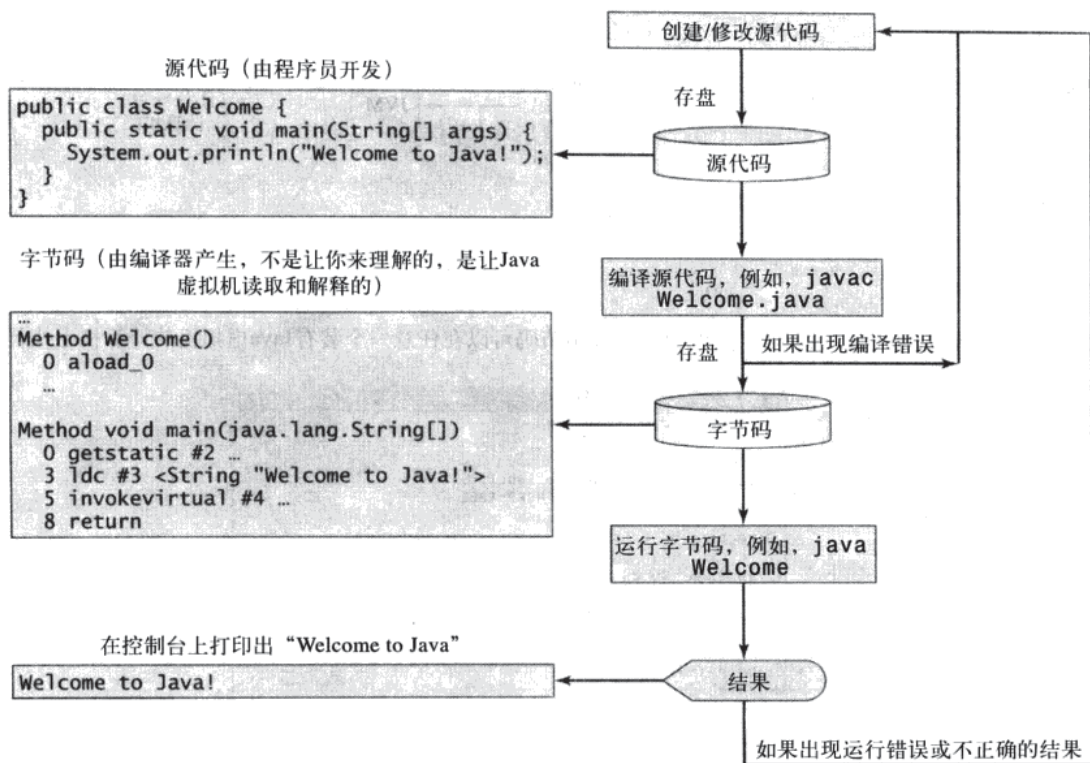


图1-9 Java程序开发过程就是重复地创建/修改源代码、编译和执行程序

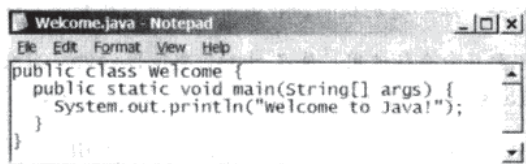


图1-10 可以使用Windows记事本创建Java源程序文件

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1-11a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, 但它是体系结构中的, 是可以在任何带Java虚拟机 (JVM) 的平台上运行的, 如图1-11b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。

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。

下述命令用来运行字节码:

```
java Welcome
```

图1-12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。

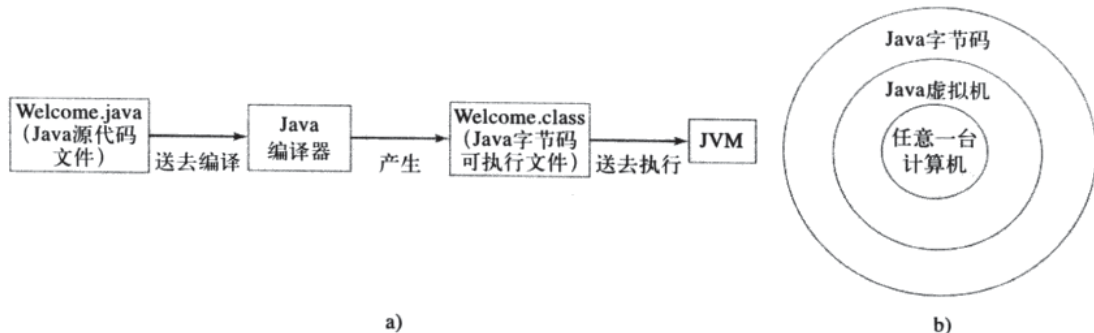


图1-11 a) Java源代码被翻译为字节码；b) Java字节码可以在任意一个装有Java虚拟机的计算机上执行

```

编译 → C:\book>javac Welcome.java
显示文件 → C:\book>dir Welcome.*
              Volume in drive C has no label.
              Volume Serial Number is 48F3-188E

              Directory of C:\book

              12/14/2006  05:24 PM                424 Welcome.class
              12/14/2006  05:23 PM                176 Welcome.java
                        2 File(s)                  600 bytes
                        0 Dir(s)  30,250,360,832 bytes free

运行 → C:\book>java Welcome
        Welcome to Java!

C:\book>
  
```

图1-12 程序清单1-1的输出显示消息“Welcome to Java!”

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。

警告 在执行程序时，不要在命令行使用扩展名.class。要使用java **ClassName**来运行程序。如果在命令行使用java **ClassName.class**，系统就会尝试去读取**ClassName.class.class**。

提示 如果要执行一个不存在的类，就会出现NoClassDefFoundError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程序不会危害到你的计算机。

教学注意 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，请参考补充材料I.F。

1.9 （GUI）在消息对话框中显示文本

如图1-12所示，程序清单1-1中的程序是在控制台上显示文本。可以改写该程序，在消息对话框中显示

文本。要实现这个目标，需要使用JOptionPane类中的showMessageDialog方法。JOptionPane是Java系统中众多预定义的类之一，这些类可以反复使用，而不必每次“重新编写”。可以使用showMessageDialog方法在消息对话框中显示任意文本，如图1-13所示。程序清单1-4给出这个新的程序。

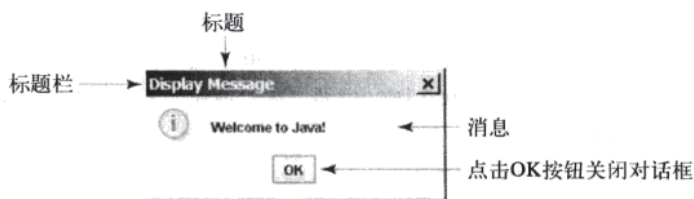


图1-13 在消息对话框中显示字符串“Welcome to Java!”

程序清单1-4 WelcomeInMessageDialogBox.java

```
1 /* This application program displays Welcome to Java!
2  * in a message dialog box.
3  */
4 import javax.swing.JOptionPane;
5
6 public class WelcomeInMessageDialogBox {
7     public static void main(String[] args) {
8         // Display Welcome to Java! in a message dialog box
9         JOptionPane.showMessageDialog(null, "Welcome to Java!");
10    }
11 }
```

这个程序使用一个Java类JOptionPane（第9行）。Java中预定义的分类组放在包中。JOptionPane放在包javax.swing中。使用第4行的import语句将JOptionPane导入，这样，编译器无须全称javax.swing.JOptionPane就可以找到类JOptionPane的位置。

注意 如果将第9行的JOptionPane用javax.swing.JOptionPane来替换，就无须在第4行导入它。javax.swing.JOptionPane是JOptionPane类的全称。

showMessageDialog方法是一个静态方法。这样的方法是要通过类名后面紧跟一个圆点运算符（.）以及带参数的方法名来调用的。方法将在第5章中介绍。调用showMessageDialog方法需要两个参数，如下所示。



第一个参数总是null。null是Java的关键字，将在第8章中详细介绍。第二个参数是要显示出来的文本字符串。

还有几种使用showMessageDialog方法的途径，目前你只需要知道两种方式就可以了。一种方式是使用如下面例子所示的语句：

```
JOptionPane.showMessageDialog(null, x);
```

这里的x就是要显示的文本字符串。

另一种方式是使用下面的语句：

```
JOptionPane.showMessageDialog(null, x,
    y, JOptionPane.INFORMATION_MESSAGE);
```

这里的x也是要显示的文本字符串，而y是表示消息对话框标题的字符串。第四个参数可以是JOptionPane.INFORMATION_MESSAGE，它是为了让消息框能够显示图标（i），如下面这个例子所示。



注意 `import`语句有两种类型：明确导入（specific import）和通配符导入（wildcard import）。明确导入是指将某个单独的类放在`import`语句中。例如，下面的语句就是从包`javax.swing`中导入`JOptionPane`。

```
import javax.swing.JOptionPane;
```

通配符导入是指导入一个包中所有的类。例如，下面的语句就是从包`javax.swing`中导入所有类。

```
import javax.swing.*;
```

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取的。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。

注意 回顾程序清单1-1中的语句`System.out.println("Welcome to Java");`中用到的`System`类。这个`System`类并没有被导入，那是因为它在`java.lang`包内。`java.lang`包内的所有类在每个Java程序中都将被隐式导入。

关键术语

.class file（.class文件）

.java file（.java文件）

assembly language（汇编语言）

bit（比特）

block（块）

block comment（块注释）

bus（总线）

byte（字节）

bytecode（字节码）

bytecode verifier（字节码验证器）

cable modem（电缆调制解调器）

central processing unit（CPU，中央处理器）

class loader（类加载器）

comment（注释）

compiler（编译器）

console（控制台）

dot pitch（点距）

DSL（Digital Subscriber Line，数字用户线）

hardware（硬件）

high-level language（高级语言）

Integrated Development Environment

（IDE，集成开发环境）

java command（java命令）

javac command（javac命令）

Java Development Toolkit（JDK，Java开发工具包）

Java Virtual Machine（JVM，Java虚拟机）

keyword or reserved word（关键字或保留字）

line comment（行注释）

machine language（机器语言）

main method（main方法）

memory（内存）

modem（调制解调器）

Network Interface Card（NIC，网络接口卡）

Operation System（OS，操作系统）

pixel（像素）

program（程序）

programming（程序设计）

resolution（分辨率）

software（软件）

source code（源代码）

source file（源文件）

specific import（明确导入）

storage devices（存储设备）

statement（语句）

wildcard import（通配符导入）

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

- 计算机是存储和处理数据的电子设备。
- 计算机包括硬件和软件两部分。
- 硬件是计算机中可以看得见的物理部分。
- 计算机程序，也就是通常所说的软件，是一些看不见的指令，它们控制硬件完成任务。
- 计算机程序设计就是编写计算机执行的指令（即代码）。
- 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。
- 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。
- 一个比特是指二进制数0或1。
- 一个字节是指8比特的序列。
- 一个千字节大约是1000字节，一个兆字节大约是100万字节，一个千兆字节大约是10亿字节，而一个万亿字节大约是1000千兆字节。
- 内存存储CPU要执行的数据和程序指令。
- 内存单元是字节的有序序列。
- 内存是不能长久保存数据的，因为断电时信息就会丢失。
- 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。
- 机器语言是一套植入每台计算机的原始指令集。
- 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。
- 高级语言很像英语，易于学习和编写程序。
- 用高级语言编写的程序称为源程序。
- 编译器是将源程序翻译成机器语言的软件程序。
- 操作系统（OS）是管理和控制计算机活动的程序。
- Java是平台无关的，这意味着只需编写一次程序，就可以在任何地方运行。
- 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。
- Java源程序文件以扩展名.java结束。
- 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。
- 使用javac命令可以从命令行编译Java源代码文件。
- 使用java命令可以从命令行运行Java类。
- 每个Java程序都是一套类定义的集合。关键字class引入类的定义，类的内容包含在块内。
- 一个块以左花括号（{）开始，以右花括号（}）结束。方法包含在类中。
- 每个Java程序必须有一个main方法。main方法是程序开始执行的入口。
- 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。
- 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。
- 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*/括住注释，称为块注释。
- Java源程序是区分大小写的。
- import语句有两种类型：明确导入和通配符导入。明确导入是指将某个单独类放在import语句中。通配符导入是指导入一个包中所有的类。

复习题

注意 复习题的答案在本书配套网站上。

16 • 第1章 计算机、程序和Java概述

1.2~1.4节

- 1.1 给出硬件和软件的定义。
- 1.2 列举计算机的主要组件。
- 1.3 给出机器语言、汇编语言和高级程序设计语言的定义。
- 1.4 什么是源程序？什么是编译器？
- 1.5 什么是JVM？

- 1.6 什么是操作系统？

1.5~1.6节

- 1.7 描述Java的历史。Java可以在任何机器上运行吗？在计算机上运行Java时需要什么？
- 1.8 Java编译器的输入和输出是什么？
- 1.9 列举一些Java的开发工具。像NetBeans、Eclipse之类的工具是与Java不同的语言，还是Java的替代品或是对Java的扩展？
- 1.10 Java与HTML之间的关系是什么？

1.7~1.9节

- 1.11 解释Java关键字的含义，罗列你在本章中学到的一些关键字。
- 1.12 Java区分大小写吗？Java关键字是大写还是小写？
- 1.13 Java源文件的扩展名是什么？Java字节码文件的扩展名是什么？
- 1.14 什么是注释？注释能被编译器忽略吗？该如何表示一个注释行和一个注释段？
- 1.15 在控制台上显示字符串的语句是什么？在消息对话框中显示消息“Hello world”的语句是什么？
- 1.16 下面的程序是错的。重新排行使程序在morning之后显示afternoon。

```
public static void main(String[] args) {  
}
```

```
public class Welcome {  
    System.out.println("afternoon");  
    System.out.println("morning");  
}
```

- 1.17 找出并修改下面代码的错误：

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

- 1.18 编译Java程序的命令是什么？运行Java程序的命令是什么？
- 1.19 如果在运行程序时出现NoClassDefFoundError，产生这个错误的原因是什么？
- 1.20 如果在运行程序时出现NoSuchMethodError，产生这个错误的原因是什么？
- 1.21 为什么System类无须被导入？
- 1.22 以下两个import语句在性能上有没有不同之处？

```
import javax.swing.JOptionPane;
```

```
import javax.swing.*;
```

- 1.23 显示以下代码的输出结果：

```
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);  
    }  
}
```



编程练习题

注意 偶数号练习的答案在本书配套的网站上。所有练习的答案都放在教师资源网站上。难度等级分为容易（没有星号）、适度（*）、难（**）和非常难（***）。

- 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      AAAAA      V V      AAAAA
      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 $\frac{9.5 \times 4.5 - 2.5 \times 3}{45.5 - 3.5}$ 的结果。
- 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 ：

$$\pi = 4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} + \cdots \right)$$

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

资源
程序
学习
PDG

第2章 |

Introduction to Java Programming, 8E

基本程序设计

学习目标

- 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)。
- 声明Java基本数据类型: `byte`、`short`、`int`、`long`、`float`、`double`和`char` (2.8.1节)。
- 使用Java运算符书写数学表达式 (2.8.2~2.8.3节)。
- 显示当前时间 (2.9节)。
- 使用简捷运算符 (2.10节)。
- 将一种类型的值强制转换为另一种类型 (2.11节)。
- 计算贷款支付额 (2.12节)。
- 使用`char`类型表示字符 (2.13节)。
- 计算整钱兑零 (2.14节)。
- 使用`String`类型表示字符串 (2.15节)。
- 熟悉Java的文档管理、程序设计风格和命名习惯 (2.16节)。
- 区分语法错误、运行时错误、逻辑错误和调试错误 (2.17节)。
- (GUI) 使用JOptionPane输入对话框获取输入 (2.18节)。

2.1 引言

在第1章里,学习了如何创建、编译和运行一个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、运算符、表达式以及输入/输出来进行基本的程序设计。

2.2 编写简单的程序

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代码两部分内容。算法 (algorithm) 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。

在学习程序设计引论课程时,遇到的绝大多数问题都可以用简单、直观的算法描述。

当你编写代码时，就是要把算法翻译成程序。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。这个程序的概貌就如下所示：

```
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：

- 1) 读取半径。
- 2) 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 (variable) 的符号。变量指定在程序中用于存储数据和计算结果的内存位置。每个变量都有自己的名字，可以用来访问它在内存的位置。

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。Java提供简单数据类型来表示整数、浮点数（即带小数点的数）、字符以及布尔类型。这些类型称为原始数据类型 (primitive data type) 或基本类型 (fundamental type)。

将radius和area声明为双精度型浮点数。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是以双精度型浮点数形式存储在计算机中的。

第一步是读取半径radius。从键盘读取数字并不是一件简单的事情。在当前情况下，我们就先给程序中的radius赋一个固定值。

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。

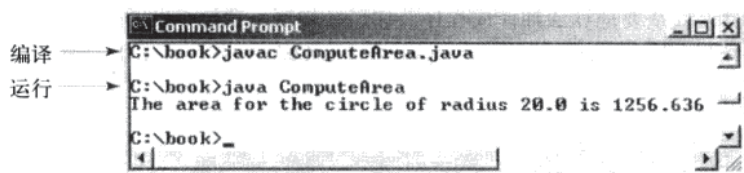


图2-1 程序显示圆面积

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; // New value is radius
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 }

```

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。手动跟踪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。第13~14行中的加号(+)称为字符串连接符(string concatenation operator)。如果两个操作数都是字符串，字符串连接符就把两个字符串连接起来。如果其中一个操作数非字符串(例如，一个数字)，这个非字符串值就会先被转换为一个字符串，然后再与另一个字符串相连。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2.15节中讨论。

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");
```

提示 这个例子包括三个步骤。通过一次添加一个步骤，逐步开发和测试每一个步骤是一种很好的方法。

2.3 从控制台读取输入

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。可以调用表2-1中的方法以读取各种不同类型的输入。


表2-1 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
next()	读取一个字符串，该字符在一个空白符之前结束
nextLine()	读取一行文本（即以按下回车键为结束标志）

现在，我们将会看到如何通过调用nextDouble()方法来读取带小数点的数。其他的方法将在用到时再讨论。程序清单2-2改写了程序清单2-1，提示用户来输入一个半径值。

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 result
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



Scanner类在包java.util里。它在第1行被导入。第6行创建一个Scanner对象。

第9行的语句显示一个消息，提示用户输入。

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

print方法和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10行的语句是从键盘读入一个输入。

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

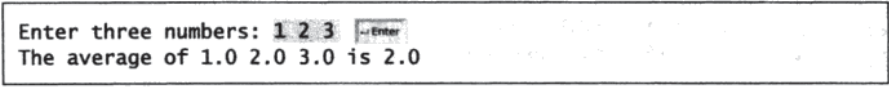
在用户键入一个数值然后点击回车键之后，该数值就被读入并赋值给radius。

更多关于对象的细节将在第8章中介绍。目前，只要知道这就是如何从控制台获取输入的方式就可以了。

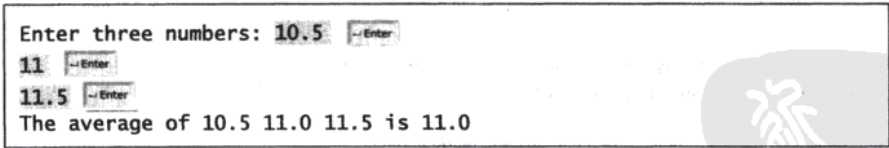
程序清单2-3给出从键盘读取输入的另一个例子。这个例子读取三个数值，然后显示它们的平均值。

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 result
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，这两行也都是同样的。

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。

2.4 标识符

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。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showMessageDialog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是完全不同的标识符。

提示 标识符是为了命名变量、常量、方法、类和包。描述性的标识符可提高程序的可读性。

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。

2.5 变量

正如在前几节的程序中看到的, 使用变量来存储将在程序中用到的数据。它们被称为变量是因为它们的值可能会被改变。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, 这样, 就可以对它们重新赋值。例如, 可以使用如下所示的代码计算不同半径的面积:

```
// Compute the first area
radius = 1.0;
area = radius * radius * 3.14159;
System.out.println("The area is " + area + " for radius " + radius);

// Compute the second area
radius = 2.0;
area = radius * radius * 3.14159;
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他可以存储的数据类型来声明该变量。变量声明 (variable declaration) 告知编译器根据数据类型为变量分配合适的存储空间。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
```

注意 按照习惯, 变量名使用小写字母表示。如果一个名字由多个词组成, 那么将所有的词连接起来, 而且除了第一个词以外, 其他词的第一个字母都要大写, 例如radius和interestRate。

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。

2.6 赋值语句和赋值表达式

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（=）作为赋值运算符（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、变量和运算符的一个运算，它们组合在一起计算出一个新值。例如，考虑下面的代码：

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

变量也可用在表达式中。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，变量名必须在赋值运算符的左边。因此， $1=x$ 是错误的。

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运算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);
```

下面的语句也是正确的：

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

该语句等价于：

```
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2.11节介绍。

2.7 定名常量

一个变量的值在程序执行过程中可能会发生变化，但是定名常量（named constant）或简称常量则表示从不改变的永久数据。在`ComputeArea`程序中， 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改写如下：

```
// ComputeArea.java: Compute the area of a circle
public class ComputeArea {
    public static void main(String[] args) {
        final double PI = 3.14159; // Declare a constant

        // Assign a radius
        double radius = 20;

        // Compute area
        double area = radius * radius * PI;

        // Display results
        System.out.println("The area for the circle of radius " +
            radius + " is " + area);
    }
}
```

警告 按照习惯，常量用大写字母命名：用PI，而不是pi或Pi。

注意 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 数值数据类型及其运算

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。

表2-2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2-2 数值数据类型

类型名	范 围	存 储 大 小
byte	-2^7 (-128) \sim 2^7-1 (127)	8位带符号数
short	-2^{15} (-32 768) \sim $2^{15}-1$ (32 767)	16位带符号数
int	-2^{31} (-2 147 483 648) \sim $2^{31}-1$ (2 147 483 647)	32位带符号数
long	$-2^{63} \sim 2^{63}-1$ (即-9 223 372 036 854 775 808 \sim 9 223 372 036 854 775 807)	64位带符号数
float	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
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值，这些值都在附录E中给出。

Java使用四种类型的整数：byte、short、int和long。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，就应该将该变量声明为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。

警告 当被赋值的变量的值太大（在大小方面）以至于无法存储时，就会造成上溢（overflow）。例如，执行以下语句就会造成上溢，因为可以存储在int类型变量中的最大值是2147483647。2147483648对int型而言就太大了。

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

类似地，执行下面的语句也会造成上溢，因为可以存储在int类型变量中的最小值是-2147483648，而值-2147483649太大，不能存储在一个int变量中。

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

Java不会报关于上溢的警告或错误。所以，当使用接近给定类型最大范围或最小范围的数时要小心。当浮点数太小（即特别接近0时）而不能被存储时，会造成下溢（underflow）。Java将它近似认为是0。所以通常无须考虑下溢问题。

2.8.1 数值运算符

数值数据类型的运算符包括标准的算术运算符：加号（+）、减号（-）、乘号（*）、除号（/）和求余号（%），如表2-3所示。

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。为了实现通常意义的算术除法，其中一个操作数必须是浮点数。例如：5.0/2的结果是2.5。

运算符%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1，12%4的结果是0，26%8的结果是2，20%13的结果是7。

表2-3 算术运算符

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

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

1 ← 商
 20 ← 被除数
 7 ← 余数

运算符%经常用在正整数上，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天之后就又是星期六。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：

$$\begin{array}{c}
 \text{一周的第6天是星期六} \\
 \downarrow \\
 (6 + 10) \% 7 = 2 \\
 \uparrow \\
 \text{10天后}
 \end{array}$$

一周有7天
 一周的第2天是星期二
 注意：第0天是指星期天

程序清单2-4给出计算一个以秒为单位的时间量所包含的分钟数和剩余秒数的程序。例如，500秒就是8分钟20秒。

程序清单2-4 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）读取整数 `seconds`。第4行使用 `seconds/60` 获取分钟数。第10行（`seconds%60`）获取在减去分钟数之后得到的剩余秒数。

运算符 `+` 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` 取反，而在表达式 `4-5` 中，负号（`-`）是二元操作符，是从 `4` 中减去 `5`。

注意 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。

2.8.2 数值直接量

直接量（literal）是指在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;
```

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 `-128~127`）。

整型直接量默认为 `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`）。推荐使用 `L`，因为 `l`（`L` 的小写）很容易与 `1`（数字 `1`）混淆。为了在 Java 程序中写整数 `2147483648`，必须将它写成 `2147483648L`，因为 `2147483648` 超出了 `int` 型的范围。

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，下面的代码将十六进制数 `FFFF` 显示为十进制数 `65535`：

```
System.out.println(0xFFFF);
```

十六进制数、二进制数和八进制数都将在附录F中介绍。

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。

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

显示结果为：1.0F/3.0F is 0.33333334。

3. 科学记数法

浮点型直接量也可以用科学记数法表示，例如，1.23456e+2也可以写成1.23456e2，相当于 $1.23456 \times 10^2 = 123.456$ ，而1.23456e-2等于 $1.23456 \times 10^{-2} = 0.0123456$ 。E（或e）表示指数，既可以是写大的也可以是写小的。

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。

2.8.3 计算Java表达式

用Java编写数值表达式就是使用Java运算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。接下来，执行乘法、除法和求余运算。如果表达式中包含若干个乘法、除法和求余运算符，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运算符，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：

$$\begin{array}{rcl}
 3 + 4 * 4 + 5 * (4 + 3) - 1 & & \\
 \uparrow & \text{————— (1) 圆括号里的先计算} & \\
 3 + 4 * 4 + 5 * 7 - 1 & & \\
 \uparrow & \text{————— (2) 乘法} & \\
 3 + 16 + 5 * 7 - 1 & & \\
 \uparrow & \text{————— (3) 乘法} & \\
 3 + 16 + 35 - 1 & & \\
 \uparrow & \text{————— (4) 加法} & \\
 19 + 35 - 1 & & \\
 \uparrow & \text{————— (5) 加法} & \\
 54 - 1 & & \\
 \uparrow & \text{————— (6) 减法} & \\
 53 & &
 \end{array}$$


程序清单2-5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2-5 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 $\left(\frac{5}{9}\right)$ 转换为5.0/9而不是5/9，因为在Java中5/9的结果是0。

2.9 问题：显示当前时间

本节的问题是开发一个显示当前GMT（格林威治标准时间）的程序，格林威治时间的格式为小时：分钟：秒（hour:minute:second），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。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）。

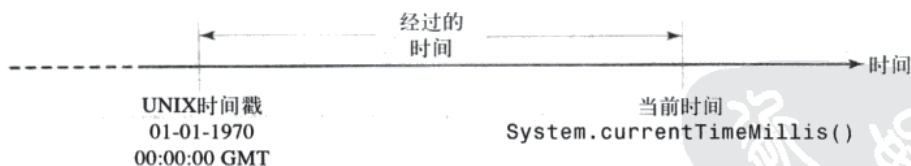


图2-2 System.currentTimeMillis()返回自UNIX时间戳以来的毫秒数

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()方法获取存放在变量totalMilliseconds中从1970年1月1日午夜到现在的毫秒数（例如：1203183086328毫秒）。
- 2) 通过将总毫秒数totalMilliseconds除以1000得到总秒数totalSeconds（例如：1203183086328毫秒/1000=1203183086秒）。
- 3) 通过totalSeconds%60得到当前的秒数（例如：1203183086秒%60=26，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：1203183086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

30 • 第2章 基本程序设计

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2-6给出完整的程序。

程序清单2-6 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 17:31:26 GMT



line#	4	7	10	13	16	19	22
variables							
totalMilliseconds	1203183086328						
totalSeconds		1203183086					
currentSecond			26				
totalMinutes				20053051			
currentMinute					31		
totalHours						334217	
currentHour							17

当调用`System.currentTimeMillis()` (第4行) 时, 它返回GMT当前时间与1970年1月1日0点之间单位为毫秒的差值。这个方法返回一个`long`型的毫秒值。因此, 在该程序中的所有变量都被声明为`long`型。

2.10 简捷运算符

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, 下面的语句将就是给`i`的当前值加8, 再将结果值赋值给`i`:

```
i = i + 8;
```

Java允许使用简捷赋值运算符合并赋值符号和加号的功能。例如, 上面的语句可以改写成:

```
i += 8;
```

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

表2-4 简捷赋值运算符

运 算 符	名 称	举 例	等 价 于
<code>+=</code>	加法赋值运算符	<code>i += 8</code>	<code>i = i + 8</code>
<code>-=</code>	减法赋值运算符	<code>i -= 8</code>	<code>i = i - 8</code>
<code>*=</code>	乘法赋值运算符	<code>i *= 8</code>	<code>i = i * 8</code>
<code>/=</code>	除法赋值运算符	<code>i /= 8</code>	<code>i = i / 8</code>
<code>%=</code>	求余赋值运算符	<code>i %= 8</code>	<code>i = i % 8</code>

警告 在简捷运算符中是没有空格的。例如: `+=`应该是`+=`。

注意 就像赋值运算符 (`=`) 一样, 运算符 (`+=`, `-=`, `*=`, `/=`, `%=`) 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`是一个语句, 而在第2行中它就是一个表达式:

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

这里还有两个更简捷的赋值运算符, 它们是对变量进行自增1和自减1的操作。由于经常需要多次改变某个值, 所以这两个运算符使用起来很方便。这两个运算符分别是`++`和`--`。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
```

运算符`++`和`--`都有前置和后置两种方式, 如表2-5所示。

表2-5 自增和自减运算符

运算符	名 称	说 明	举例 (假设i=1)
<code>++var</code>	前置自增运算符	变量var的值加1且使用var增加后的新值	<code>int j=++i;</code>
<code>var++</code>	后置自增运算符	变量var的值加1但使用var原来的值	<code>int j=i++;</code>
<code>--var</code>	前置自减运算符	变量var的值减1且使用var减少后的新值	<code>int j=--i;</code>
<code>var--</code>	后置自减运算符	变量var的值减1但使用var原来的值	<code>int j=i--;</code>

若运算符是在变量的前面 (前置于变量), 则该变量自增1或自减1, 然后返回的是变量的新值。若运算符是在变量的后面 (后置于变量), 则变量也会自增1或自减1, 但是返回的是变量原来的旧值。因此, 前置的`++x`和`--x`分别称为前置自增运算符 (preincrement operator) 和前置自减运算符

(predecrement operator)，而后置 $x++$ 和 $x--$ 分别称为后置自增运算符 (postincrement operator) 和后置自减运算符 (postdecrement operator)。如果单独使用某种前置形式 $++$ (或 $--$) 和后置形式 $++$ (或 $--$)，那么不管前置还是后置其效果是一样的，但是用在表达式中它们就会产生不同的效果。下面的代码就解释了这一点：

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

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);    等效于    i = i + 1;
                                int newNum = 10 * i;
```

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。

自增运算符 $++$ 和自减运算符 $--$ 可以用于所有的整型和浮点型。这些运算符经常用在循环语句中。循环语句是控制一个操作或一系列操作连续执行多少次的结构体。这种结构以及循环语句相关的主题都将在第4章中介绍。

提示 使用自增运算符和自减运算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运算符修改多个变量或多次修改同一个变量，如：

```
int k=++i + i.
```

2.11 数值类型转换

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)。拓宽类型不需要显式转换，可以自动执行转换。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);
```

显式结果为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);
```

显式结果为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);
```

显式结果为0，因为1和2都是整数，那么对它们做除法的结果也必须是整数。

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 not changed, still 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内,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2.8.2节。

程序清单2-7给出保留营业税小数点后两位的程序。

程序清单2-7 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

```
tax * 100 is 1185.3
(int)(tax * 100) is 1185
(int)(tax * 100) / 100.0 is 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。

2.12 问题: 计算贷款支付额

本节的问题是编写程序计算贷款支付额。这里的贷款可以是购车贷款、学生贷款或者房屋抵押贷款。程序要求用户输入利率、年数以及贷款总额,并要求显示月支付金额和总偿还金额。

计算月支付额的公式如下:

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

不必知道这个公式是如何推导出来的。但是, 给定了月利率、年数和贷款总额, 就可以利用这个公式计算出月支付额。

在这个公式中, 必须计算 $(1 + \text{月利率})^{\text{年数} \times 12}$ 。可以使用Math类中的方法pow(a,b)来计算 a^b 。Java API中的Math类适用于所有的Java程序。例如:

```
System.out.println(Math.pow(2, 3)); // Display 8.0
System.out.println(Math.pow(4, 0.5)); // Display 2.0
```

使用Math.pow(1+monthlyInterestRate, numberOfYears*12)可以计算 $(1 + \text{月利率})^{\text{年数} \times 12}$ 。

下面是开发该程序的步骤:

- 1) 提示用户输入年利率、年数和贷款总额。
- 2) 利用年利率获取月利率。
- 3) 使用前面的公式计算月支付额。
- 4) 计算总支付额, 它是月支付额乘以12再乘以年数。
- 5) 显示月支付额和总支付额。

程序清单2-8给出完整的程序。

程序清单2-8 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 yearly interest rate
9         System.out.print("Enter yearly interest rate, for example 8.25: ");
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, for example 5: ");
18        int numberOfYears = input.nextInt();
19
20        // Enter loan amount
21        System.out.print("Enter loan amount, for example 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 yearly interest rate, for example 8.25: 5.75 [Enter]
Enter number of years as an integer, for example 5: 15 [Enter]
Enter loan amount, for example 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			0.0047916666666				
numberOfYears				15			
loanAmount					250000		
monthlyPayment						2076.0252175	
totalPayment							373684.539

第10行读取年利率，在第13行将它转为月利率。如果输入的不是一个数值，就会出现运行错误。

需要为变量选择最合适的数据类型。例如，尽管可以将numberOfYears声明为long、float或double型，但是最好还是将其声明为int型（第18行）。注意，可能最适合于numberOfYears的类型是byte。但是为了简便起见，本书中的例子都是用int来表示整数，而用double来表示浮点值。

计算月支付额的公式被翻译成第25~27行的Java代码。

第31行和33行使用类型转换得到新的monthlyPayment和totalPayment，两个变量都保留到小数点后两位。

程序使用在第一行导入的Scanner类。程序还要用到Math类。为什么不导入Math类呢？这是因为Math类在java.lang包中。java.lang包中所有的类都被隐式导入，所以，是无须显式导入Math类的。

2.13 字符数据类型及运算

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。

2.13.1 统一码和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。将字符映射为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。

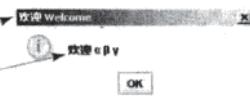
Java支持统一码（Unicode），统一码是由统一码协会（Unicode Consortium）建立的一种编码方案，它支持世界不同语言的可书写文本的交换、处理和显示。统一码一开始被设计为16比特的字符编码。基本数据类型char本身就能够利用这个设计思路来提供一种能够存放任意字符的简单数据类型。但是，这样的16比特的编码所能产生的字符只有65536个，它是不足以表示全世界所有字符的。因此，统一码标准被扩展为1112064个字符。这些字符都远远超过了原来16位的限制，它们称为补充字符（supplementary character）。Java支持这些补充字符。对补充字符的处理和表示都超过了本书的范围。为了简化问题，本书只考虑原来的16位统一码字符。这些字符都可以存储在一个char型变量中。

一个16比特位统一码占两个字节，用以\u开头的4位十六进制数表示，范围从'\u0000'到'\uFFFF'。例如：单词“welcome”被翻译成中文需要两个字符“欢迎”。这两个字符的统一码为“\u6B22\u8FCE”。

程序清单2-9给出显示两个中文字符和三个希腊字母的程序。

程序清单2-9 DisplayUnicode.java

```
1 import javax.swing.JOptionPane;
2
3 public class DisplayUnicode {
4     public static void main(String[] args) {
5         JOptionPane.showMessageDialog(null,
6             "\u06B2\u08FCE \u03b1 \u03b2 \u03b3",
7             "\u06B2\u08FCE Welcome",
8             JOptionPane.INFORMATION_MESSAGE);
9     }
10 }
```



如果系统中没有安装中文字体，是无法看到中文字符的。希腊字母的统一码是 \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7位编码表。Unicode码包括ASCII码，从 '\u0000' 到 '\u007F' 对应128个ASCII字符（参见附录B中的ASCII字符列表以及它们的十进制和十六进制编码）。在Java程序中，可以使用像 'X'、'1' 和 '\$' 等这样的ASCII字符，也可以使用统一码。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 自增和自减运算符也可用在char型变量上，这会得到该字符之前或之后的统一码字符。

例如，下面的语句显示字符b：

```
char ch = 'a';
System.out.println(++ch);
```

2.13.2 特殊字符的转义序列

假如你想在输出时打印如下带引号的信息，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定义了转义序列来表示特殊的字符，如表2-6所示。一个转义序列以反斜杠（\）开始，后面跟一个对编译器而言具有特殊意义的字符。

表2-6 Java转义字符序列

转义序列	名 称	Unicode
\b	退格键	\u0008
\t	Tab键	\u0009
\n	换行符号	\u000A
\f	进纸	\u000C
\r	回车键	\u000D
\\	反斜杠	\u005C
\'	单引号	\u0027
\"	双引号	\u0022

所以，现在可以使用下面的语句打印带引号的消息：

```
System.out.println("He said \"Java is fun\"");
```

它的输出才是：

```
He said "Java is fun"
```

2.13.3 字符型char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。将一个整数转换成一个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，然后就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，这个字符的统一码就被转换成某个特定的数值。

```
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'的统一码是97，它是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，因为统一码\uFFFF4不适用于一个字节范围内，下面的转换就是不正确的：

```
byte b = '\uFFFF4';
```

为了强制赋值，就必须使用显式转换方式，如下所示：

```
byte b = (byte)\uFFFF4';
```

在0到FFFF之间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。而任何不在此范围内的其他数值都必须显式地转换为char型。

注意 所有数值运算符都可以用在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);

System.out.println("Chapter " + '2');
```

显示结果

```
i is 101
j is 99
99 is the Unicode for character c
Chapter 2
```

注意 小写字母的统一码是从'a'的统一码开始，然后是'b'、'c'、...、'z'的统一码所构成的连续整数。大写字母的情况也是一样的。此外，'a'的统一码比'A'的统一码大，所以，'a'-'A'与'b'-'B'相等。因此，对应于小写字母ch的大写字母是(char)('A'+(ch-'a'))。

2.14 问题：整钱兑零

假如你希望开发一个程序，将给定的钱数分类成较小的货币单位。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dollar（1美元）、quarter（2角5分）、dime（1角）、nickel（5分）和penny（1分）的数目，如运行示例所示。

38 • 第2章 基本程序设计

该程序应该能够列出1美元的最大数, 其次是2角5分的最大数等, 依此类推。

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 \n" +
36             "\t" + numberOfOneDollars + " dollars\n" +
37             "\t" + numberOfQuarters + " quarters\n" +
38             "\t" + numberOfDimes + " dimes\n" +
39             "\t" + numberOfNickels + " nickels\n" +
40             "\t" + numberOfPennies + " pennies");
41     }

```


42 }

```

Enter an amount in double, for example 11.56: 11.56 --Enter
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				
numberOfNickles										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型的以美元和美分形式表示的十进制数。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。

程序从总钱数中去掉了1美元币的最大数，得到的余额存储在变量remainingAmount中（第16~17行）。接着从remainingAmount中去掉2角5分币的最大数，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角的最多个数、5分的最多个数和1分的最多个数。

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（参见练习题2.9）。

结果如运行示例所示，显示0个1角、1个5美分和1个1美分。最好不要显示0个1角，也最好能用字的单数形式显示1个5美分和1个1美分。下一章将学习如何使用选择语句修改这个程序（参见练习题3.7）。

2.15 String类型

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，下述代码将消息声明为一个字符串，其值为“Welcome to Java”：

```
String message = "Welcome to Java";
```

String实际上与System类、JOptionPane类和Scanner类一样，都是一个Java库中预定义的类。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。引用数据类型将在第8章中详细讨论。目前，只需要知道如何声明一个String类型的变量，如何将字符串赋值给该变量以及如何连接字符串就可以了。

如前面的程序清单2-1中所示，可以进行两个字符串的连接。如果操作数之一是字符串，加号（+）就是连接运算符。如果操作数之一不是字符串（例如是一个数字），非字符串值先转换为字符串，再与另一个字符串连接起来。下面是几个例子：

```
// 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，那么加号（+）就是将两个数值加起来的加法运算符。

简捷运算符+=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”追加到message中的字符串“Welcome to Java”之后：

```
message += " and Java is fun";
```

这样，新的message就成为“Welcome to Java and Java is fun”。

假设i=1且j=2，下列语句的输出结果是什么？


```
System.out.println("i + j is " + i + j);
```

由于“i + j is”首先与i的值连接，所以，输出结果是“i + j is 12”。要强制先执行i + j，就需要将i + j用括号括起来，如下所示：

```
System.out.println("i + j is " + (i + j));
```

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.println("Enter three strings: ");
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 a string: 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 string: ");
String s = input.nextLine();
System.out.println("The string entered is " + s);
```

```
Enter a string: Welcome to Java 
The string 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9.7.3节中解释。

2.16 程序设计风格和文档

编程风格 (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，但是这样是非常不好的程序设计风格，因为这会使程序的可读性很差。文档 (documentation) 是嵌在程序中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几率，并且提高程序的可读性。到现在为止，已经学习了一些好的编程风格。本节把它们总结出来，并给出几条指导原则。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ID中找到。

2.16.1 适当的注释和注释风格

在程序的开头写一个摘要，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。在较大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`//`和块注释`/*`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。它们能被JDK的javadoc命令提取出来，放入一个HTML文件。为了得到更多信息，参见java.sun.com/j2se/javadoc。

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的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，建议使用行注释 (`//`)。

2.16.2 命名习惯

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。名字是区分大小写的。下面列出变量、常量、方法和类的命名习惯。

- 1) 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`showInputDialog`。
- 2) 类名中的每个单词的首字母大写，例如，类名`ComputeArea`、`Math`和`JOptionPane`。
- 3) 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`。

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`Math`类，就不要用`Math`来命名自己的类。

提示 应避免对标识符使用缩写。使用完整的单词更具描述性。例如，`numberOfStudents`比`numStuds`、`numOfStuds`或`numOfStudents`好。

2.16.3 适当的缩进和空白

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) 用于描述程序中组件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，对Java而言，读懂这样的程序也是没问题的，但是人们发现正确的对齐能够使我们更易读懂和维护代码。在嵌套结构中，每个内层的组件或语句应该比外层缩进两格，这会比仅仅只做嵌套好得多。

二元运算符的两边应该各加一个空格，如下面语句所示：

```
int i = 3+4 * 4; ← 不好的风格
```

```
int i = 3 + 4 * 4; ← 良好的风格
```

应该使用一个空行把代码分段，以使程序更易阅读。

2.16.4 块的风格

块是由大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喜好。应该持续采用一种风格，建议最好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。

2.17 程序设计错误

即使是非常有经验的程序员，也不能避免程序设计错误。错误可以分为三类：语法错误、运行错误和逻辑错误。

2.17.1 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）。语法错误是由代码结构体中的错误引起的，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：编译下面的程序会出现语法错误，如图2-3所示。

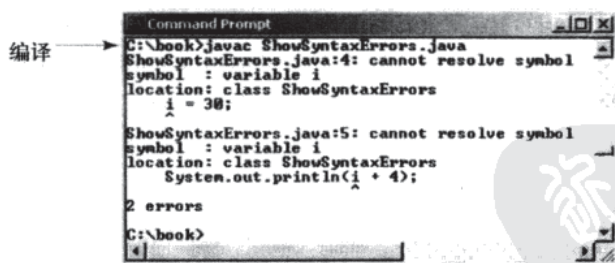


图2-3 编译器报告语法错误

```
1 // ShowSyntaxErrors.java: The program contains syntax errors
2 public class ShowSyntaxErrors {
3     public static void main(String[] args) {
4         i = 30;
5         System.out.println(i + 4);
6     }
7 }
```

检测出的错误有两个。它们都是因为没有声明变量*i*造成的。因为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调试是一个很好的习惯。解决程序前面出现的错误，可能就改正了程序中后面出现的其他错误。

2.17.2 运行错误

运行错误 (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, 就会出现运行错误。输入错误就是典型的运行错误。

当用户输入一个程序不期望的输入, 导致程序不能处理该值时, 就会发生输入错误。例如: 如果程序希望读入的是一个数, 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。为了防止输入错误, 程序应该提醒用户输入正确类型的值。在从键盘读入整数之前, 最好能显示 “Please enter an integer” (请输入一个整数) 之类的提示信息。

另一个常见的运行错误是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: 下面的程序将会导致运行错误, 如图2-4所示。

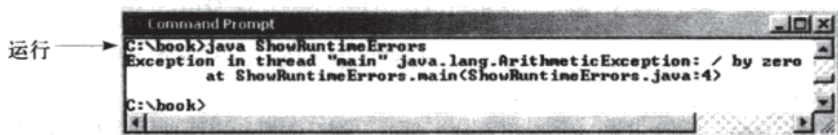


图2-4 运行错误导致程序非正常中断

```
1 // ShowRuntimeErrors.java: Program contains runtime errors
2 public class ShowRuntimeErrors {
3     public static void main(String[] args) {
4         int i = 1 / 0;
5     }
6 }
```

2.17.3 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多种多样。例如: 假设你编写下面的程序将number1和number2相加。

```
// ShowLogicErrors.java: The program contains a logic error
public class ShowLogicErrors {
    public static void main(String[] args) {
        // Add number1 to number2
        int number1 = 3;
        int number2 = 3;
        number2 += number1 + number2;
        System.out.println("number2 is " + number2);
    }
}
```

该程序既没有语法错误也没有运行错误, 但是它不能打印出number2的正确结果。看看你能否找到错误。

2.17.4 调试

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。

逻辑错误也称为小虫子 (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。

教学注意 IDE不仅有助于调试错误, 而且也是一个有效的教学工具。补充材料II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。

2.18 (GUI) 从输入对话框获取输入

可以从控制台获取输入。还有一种可以选择的方法，那就是通过调用 `JOptionPane.showInputDialog` 方法从一个输入对话框中获取输入，如图2-5所示。

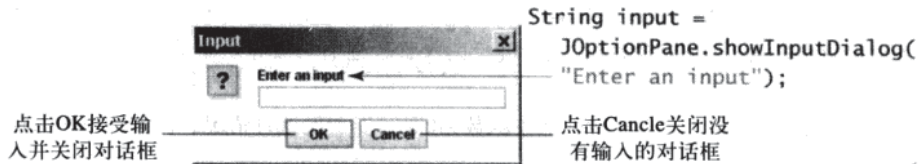


图2-5 输入对话框可以让用户输入一个字符串

当执行这个方法时，就会显示一个对话框，让你键入输入值。输入一个字符串后，单击OK接收输入并关闭该对话框。从该方法中返回的输入是一个字符串。

使用 `showInputDialog` 方法的途径有很多种。目前，你只需要知道两种调用方式即可。

一种是使用像下面例子中的语句：

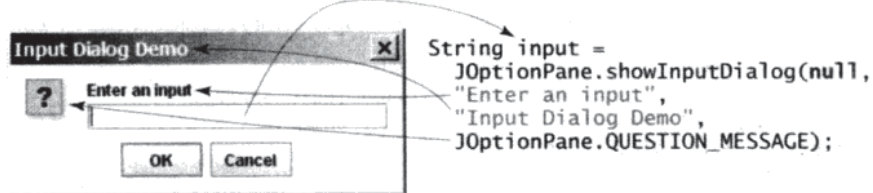
```
JOptionPane.showInputDialog(x);
```

其中 `x` 是表示提示信息的字符串。

而另一种是使用如下所示的语句：

```
String string = JOptionPane.showInputDialog(null, x,
    y, JOptionPane.QUESTION_MESSAGE);
```

其中 `x` 是表示提示信息的字符串，`y` 是表示输入对话框标题的字符串，如下图所示。



2.18.1 将字符串转换为数字

输入对话框返回的输入是一个字符串。如果你输入的是一个数字值123，它会返回"123"。必须把字符串转化为数字值以得到数字型的输入。

要把一个字符串转换为一个 `int` 型值，使用 `Integer` 类中的 `parseInt` 方法，如下所示：

```
int intValue = Integer.parseInt(intString);
```

这里的 `intString` 是一个数值字符串，例如："123"。

要将一个字符串转换为一个 `double` 型的值，使用 `Double` 类中的 `parseDouble` 方法，如下所示：

```
double doubleValue = Double.parseDouble(doubleString);
```

这里的 `doubleString` 是一个数值字符串，例如："123.45"。

`Integer` 类和 `Double` 类都包含在包 `java.lang` 中，因此，它们都是自动导入的。

2.18.2 使用输入对话框

程序清单2-8是从控制台读取输入。还可以使用输入对话框。

程序清单2-11给出完整的程序。图2-6给出程序的示例运行。

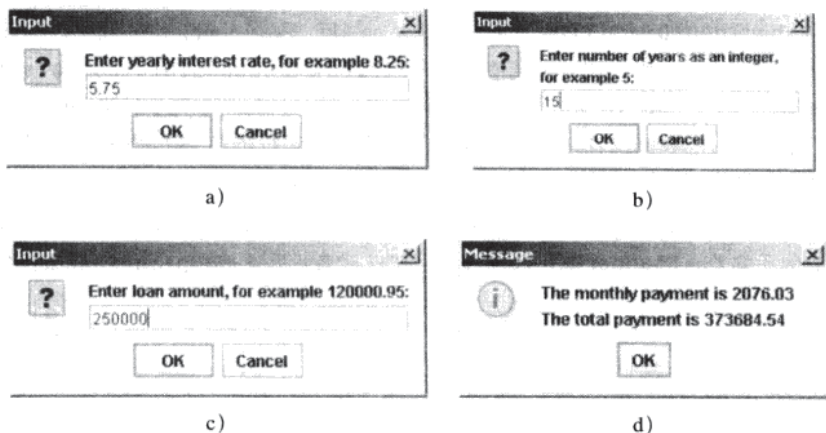


图2-6 程序接受年利率a、年数b、贷款总额c，然后显示月支付额和总支付额d

程序清单2-11 ComputeLoanUsingInputDialog.java

```

1 import javax.swing.JOptionPane;
2
3 public class ComputeLoanUsingInputDialog {
4     public static void main(String[] args) {
5         // Enter yearly interest rate
6         String annualInterestRateString = JOptionPane.showInputDialog(
7             "Enter yearly interest rate, for example 8.25:");
8
9         // Convert string to double
10        double annualInterestRate =
11            Double.parseDouble(annualInterestRateString);
12
13        // Obtain monthly interest rate
14        double monthlyInterestRate = annualInterestRate / 1200;
15
16        // Enter number of years
17        String numberOfYearsString = JOptionPane.showInputDialog(
18            "Enter number of years as an integer, \nfor example 5:");
19
20        // Convert string to int
21        int numberOfYears = Integer.parseInt(numberOfYearsString);
22
23        // Enter loan amount
24        String loanString = JOptionPane.showInputDialog(
25            "Enter loan amount, for example 120000.95:");
26
27        // Convert string to double
28        double loanAmount = Double.parseDouble(loanString);
29
30        // Calculate payment
31        double monthlyPayment = loanAmount * monthlyInterestRate / (1
32            - 1 / Math.pow(1 + monthlyInterestRate, numberOfYears * 12));
33        double totalPayment = monthlyPayment * numberOfYears * 12;
34
35        // Format to keep two digits after the decimal point
36        monthlyPayment = (int)(monthlyPayment * 100) / 100.0;
37        totalPayment = (int)(totalPayment * 100) / 100.0;
38
39        // Display results
40        String output = "The monthly payment is " + monthlyPayment +
41            "\nThe total payment is " + totalPayment;
42        JOptionPane.showMessageDialog(null, output);
43    }
44 }

```

第6~7行的showInputDialog方法显示一个输入对话框。输入double值的利率，然后点击OK接受这个输入。返回值是一个字符串，该值被赋值给一个String变量annualInterestRateString。使用Double.parseDouble(annualInterestRateString)（第11行）将字符型转换为double型值。如果在输入对话框中输入的不是数值或者点击了Cancel，都会出现运行错误。在第13章中，将会学习如何处理这些异常，以使程序能够继续运行。

教学注意 可以使用JOptionPane或Scanner来获取输入，这是很方便的。为保持一致性，本书中的很多例子都是用Scanner来获取输入的。可以很容易地使用JOptionPane来改写这些例子以获取输入。

关键术语

algorithm (算法)	incremental development and testing (递进式开发和测试)
assignment operator (=) (赋值运算符)	indentation (缩进)
assignment statement (赋值语句)	int type (整数类型)
backslash (\) (反斜杠)	literal (直接量)
byte type (字节类型)	logic error (逻辑错误)
casting (类型转换)	long type (长整型类型)
char type (字符类型)	narrowing (of types) ((类型的) 缩窄)
constant (常量)	operator (操作符)
data type (数据类型)	overflow (上溢)
debugger (调试器)	pseudocode (伪代码)
debugging (调试)	primitive data type (基本数据类型)
declaration (声明)	runtime error (运行错误)
decrement operator (--) (自减运算符)	short type (短整型类型)
double type (双精度类型)	syntax error (语法错误)
encoding (编码)	supplementary Unicode (补充统一码)
final (final关键字)	underflow (下溢)
float type (浮点类型)	Unicode (统一码)
floating-point number (浮点数)	UNIX epoch (UNIX时间戳)
expression (表达式)	variable (变量)
identifier (标识符)	widening (of types) ((类型的) 拓宽)
increment operator (++) (自增运算符)	whitespace (空白符)

本章小结

- 标识符是程序中的事物的名称。
- 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。
- 选择描述性的标识符可提高程序的可读性。
- 使用变量存储在程序中使用的数据。
- 声明变量就是告诉编译器何种数据类型可以存储变量。
- 按照习惯，变量名是小写字母。

- 在Java中，等号(=)被用作赋值运算符。
- 方法中声明的变量必须在使用前被赋值。
- 定名常量(或简称为常量)表示从不改变的永久数据。
- 用关键字final声明定名常量。
- 按照习惯，常量用大写字母命名。
- 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提供运算符完成数值运算：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。
- Java表达式中的数值运算符和算术表达式中应用的数值运算符的使用方法是完全一致的。
- Java提供简捷运算符：+= (加法赋值)、-= (减法赋值)、*= (乘法赋值)、/= (除法赋值)以及%= (求余赋值)。
- 自增运算符(++)和自减运算符(--)分别对变量加1或减1。
- 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正确类型。
- 可以使用(type)exp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。
- 将一个小范围类型的变量转换为大范围类型的过程称为拓宽类型。
- 将一个大范围类型的变量转换为小范围类型的过程称为缩窄类型。
- 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。
- 字符类型(char)表示单个字符。
- 字符\称为转义字符。
- Java允许使用转义序列来表示特殊字符，例如：'\t'和'\n'。
- 字符' '、'\t'、'\f'、'\r'和'\n'都称为空白字符。
- 在计算机科学中，1970年1月1日午夜零点称为UNIX时间戳。
- 程序设计错误可归纳为三类：语法错误、运行错误和逻辑错误。
- 编译过程中出现的错误为语法错误或编译错误。
- 运行错误是指引起程序非正常中断的错误。
- 逻辑错误是指程序没有按预期的方式完成。

复习题

2.2~2.7节

2.1 下列哪些标识符是合法的？哪些是Java的关键字？

**applet, Applet, a++, --a, 4#R, \$4, #44, apps
class, public, int, x, y, radius**

2.2 将下面的算法翻译成Java代码：

- 第一步：声明一个名为miles初始值为100的double型变量。
- 第二步：声明一个名为KILOMETERS_PER_MILE值为1.609的double型常量。
- 第三步：声明一个名为kilometers的double型变量，将miles和KILOMETERS_PER_MILE相乘的结果赋值给kilometers。
- 第四步：将kilometers显示在控制台上。

在第四步之后，kilometers的值是多少？

2.3 使用常量的好处是什么？声明一个值为20的int型常量SIZE。

2.8~2.10节

2.4 假设int a=1和double d=1.0，并且每个表达式都是独立的，那么下面表达式的结果是什么？

48 • 第2章 基本程序设计

```

a = 46 / 9;
a = 46 % 9 + 4 * 4 - 2;
a = 45 + 43 % 5 * (23 * 3 % 2);
a %= 3 / a + 3;
d = 4 + d * d + 4;
d += 1.5 * 3 + (++a);
d -= 1.5 * 3 + a++;

```

2.5 给出下面求余运算的结果。

```

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

```

2.6 如果今天是星期二，那么100天后是星期几？

2.7 分别找出byte、short、int、long、float和double中的最大数和最小数，哪种数据类型所需的存储空间最小？

2.8 25/4的结果是什么？如果希望得到的结果是浮点数，应该怎样改写这个表达式？

2.9 下列语句正确吗？如果正确，写出其输出值。

```

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);

```

2.10 如何用Java书写下面的算术表达式？

$$\frac{4}{3(r+34)} - 9(a+bc) + \frac{3+d(2+a)}{a+bd}$$

2.11 假设m和r都是整数。编写mr²的Java表达式，得到一个浮点数。

2.12 下列说法哪些是正确的？

- (1) 任何表达式都可以用作语句。
- (2) 表达式x++可以用作语句。
- (3) 语句x=x+5也是一个表达式。
- (4) 语句x=y=x=0是非法的。

2.13 下列哪些是正确的浮点数直接量？

12.3, 12.3e+2, 23.4e-2, -334.4, 20, 39F, 40D

2.14 找出并修改下列代码中的错误：

```

1 public class Test {
2     public void main(string[] args) {
3         int i;
4         int k = 100.0;
5         int j = i + 1;
6
7         System.out.println("j is " + j + " and
8             k is " + k);
9     }
10 }

```

2.15 如何使用System.currentTimeMillis()方法获取当前分钟数？

2.11节

2.16 不同类型的数值能在一起计算吗？

2.17 如果显式地将double型转换为int型，那么对double型值的小数部分是如何处理的？类型转换是否改变被转换变量的值？

2.18 写出下面语句段的输出：



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

2.13节

2.19 使用打印语句求出 '1'、'A'、'B'、'a'、'b' 的ASCII码。使用打印语句求出ASCII码为十进制数 40、59、79、85、90 的字符。使用打印语句求出ASCII码为十六进制数 40、5A、71、72、7A 的字符。

2.20 下列哪些是正确的字符直接量?

'1', '\u345dE', '\u3fFa', '\b', \t

2.21 如何显示字符 \ 和 " ?

2.22 执行下述代码:

```
int i = '1';
int j = '1' + '2';
int k = 'a';
char c = 90;
```

2.23 下面哪些类型转换是允许的? 如果允许, 写出转换后的结果。

```
char c = 'A';
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;
```

2.24 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);
    }
}
```

2.15节

2.25 给出下面语句的输出结果 (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);
```

2.26 计算下面表达式的结果 (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
```

2.16~2.17节

2.27 类名、方法名、常量和变量的命名习惯是什么? 根据Java的命名习惯, 下面哪些是常量、方法、变量或类?



MAX_VALUE、Test、read、readInt

2.28 根据编程风格和文档指南，使用次行花括号方式，重新布局下列程序的格式。

```
public class Test
{
    // Main method
    public static void main(String[] args) {
        /** Print a line */
        System.out.println("2 % 3 = "+2%3);
    }
}
```

2.29 描述何谓语法错误、运行错误和逻辑错误。

2.18节

2.30 为什么必须导入JOptionPane类而无须导入Math类？

2.31 如何使用对话框提示用户输入一个整数？

2.32 如何将字符串转换为整数？如何将字符串转换为double型？

编程练习题

注意 学生可以从www.cs.armstrong.edu/liang/intro8e/exercise8e.zip上下载 exercise8e.zip运行所有的练习，可以使用命令 `java -cp exercise8e.zip Exercisei_j` 来运行练习Exercisei_j。例如：要运行练习Exercise2_1，就使用

```
java -cp exercise8e.zip Exercise2_1
```

这也给你一种如何运行程序的思路。

调试提示 编译器经常会给出语法错误出现的原因。如果不知道如何改正它，就在文本里将程序同与其相类似的例子逐字地进行比较。

2.2~2.9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：

$$\text{fahrenheit} = (9/5) * \text{celsius} + 32 \quad (\text{华氏度} = (9/5) * \text{摄氏度} + 32)$$

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 Enter
16 feet is 4.88 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 Enter
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: 15.69 15 Enter
The gratuity is 2.35 and total is 18.04
```



**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: 利用运算符%分解数字, 然后使用运算符/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.
```



2.13节

*2.8 (求ASCII码对应的字符) 编写程序接收一个ASCII码 (从0到128的整数), 然后显示它所代表的字符。例如, 如果用户输入的是97, 程序显示的是字符a。下面是一个运行示例:

```
Enter an ASCII code: 69 Enter
The character for ASCII code 69 is E
```



*2.9 (财务应用程序: 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分币值。例如: 1156就表示的是11美元56美分。

2.18节

*2.10 (使用图形用户界面输入) 改写程序清单2-10, 使用图形用户界面进行输入和输出。

综合题

*2.11 (财务应用程序: 工资单) 编写程序, 读入下列信息并打印工资单:

- 雇员的名字 (例如Smith)
- 每周工作小时数 (例如10)
- 每小时工资 (例如6.75)
- 联邦所得税税率 (例如20%)

州所得税税率（例如9%）

编写两种版本的程序：（1）使用对话框获取输入并显示输出；（2）使用控制台进行输入和输出。从控制台进行输入和输出的示例运行如下所示：

```

Enter employee's name: Smith
Enter number of hours worked in a week: 10
Enter hourly pay rate: 6.75
Enter federal tax withholding rate: 0.20
Enter state tax withholding rate: 0.09
Employee Name: Smith
Hours Worked: 10.0
Pay Rate: $6.75
Gross Pay: $67.5
Deductions:
    Federal Withholding (20.0%): $13.5
    State Withholding (9.0%): $6.07
    Total Deduction: $19.57
Net Pay: $47.92
  
```



*2.12（财务应用程序：计算利息）如果你知道收支余额和年利率的百分比，你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：（1）使用对话框获取输入并显示输出；（2）使用控制台进行输入和输出。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
  
```



*2.13（财务应用程序：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：

$$\text{futureInvestmentValue} = \text{investmentAmount} \times (1 + \text{annuallyInterestRate})^{\text{numberOfYears} \times 12}$$

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。

提示 使用方法Math.pow(a, b)来计算a的b次幂。

下面是一个运行示例：

```

Enter investment amount: 1000
Enter monthly interest rate: 4.25
Enter number of years: 1
Accumulated value is 1043.34
  
```



*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公斤而一英寸是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 (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。(在练习题4.30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。)

- 2.16 (科学方面: 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 the initial temperature: 3.5
Enter the final temperature: 10.5
The energy needed is 1.62548e+06
```



- *2.17 (科学方面: 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, 使用新的风寒温度来测量寒冷程度。计算公式如下所示:

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

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 以下或 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 $\text{Math.pow}(a,b)$ 来计算 $v^{0.16}$ 。下面是一个运行示例:

```
Enter the temperature in Fahrenheit: 5.3
Enter the wind speed 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:

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

- 2.19 (随机字符)** 编写程序, 使用`System.currentTimeMillis()`显示任意一个大写字母。

- 2.20 (几何方面: 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。下面是一个运行示例:

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



- *2.21 (几何方面: 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:

$$s = (side1 + side2 + side3) / 2;$$

$$area = \sqrt{s(s - side1)(s - side2)(s - side3)}$$

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 Enter
The area of the triangle is 33.6
```



- 2.22 (几何方面: 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:

$$area = \frac{3\sqrt{3}}{2} s^2$$

这里的 s 就是边长。下面是一个运行示例:

```
Enter the side: 5.5 Enter
The area of the hexagon is 78.5895
```



- 2.23 (物理方面: 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, 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
```



- 2.24 (物理方面: 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:

$$length = \frac{v^2}{2a}$$

编写程序, 提示用户输入以米/秒为单位的速度 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 v and a: 60 3.5 Enter
The minimum runway length for this airplane is 514.286
```



- *2.25 (当前时间) 程序清单2-6给出了显示当前格林威治时间的程序。修改这个程序, 使之能够做到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
```



选 择

学习目标

- 声明boolean类型以及使用比较运算符书写布尔表达式 (3.2节)。
- 使用布尔表达式编写程序AdditionQuiz (3.3节)。
- 使用单向if语句实现选择控制 (3.4节)。
- 使用单向if语句编写游戏GuessBirthday的程序 (3.5节)。
- 使用双向if语句实现选择控制 (3.6节)。
- 使用嵌套的if语句实现选择控制 (3.7节)。
- 避免if语句中的常见错误 (3.8节)。
- 使用选择语句编程的不同种类的例子 (SubstractionQuiz、BMI、ComputeTax) (3.9~3.11节)。
- 使用Math.random()方法产生随机数 (3.9节)。
- 使用逻辑运算符(&&、||和!)对条件进行组合 (3.12节)。
- 使用带组合条件的选择语句进行编程 (LeapYear、Lottery) (3.13~3.14节)。
- 使用switch语句实现选择控制 (3.15节)。
- 使用条件运算符书写表达式 (3.16节)。
- 使用System.out.printf方法格式化输出 (3.17节)。
- 检查控制运算符优先级和结合方向的规则 (3.18节)。
- (GUI) 使用确定对话框获取用户的确认信息 (3.19节)。

3.1 引言

如果给程序清单2-2中的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?

Java和所有高级程序设计语言一样也提供选择语句,以便在两个或更多可选择的流程中做出选择。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);
}
```

选择语句要用到条件。条件就是布尔表达式。本章首先介绍布尔类型、布尔值、布尔比较运算符和布尔表达式。

3.2 boolean数据类型

该如何比较两个值呢?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提供六种比较运算符 (comparison operator) (也称为关系运算符 (relational operator)),用于两个值的比较(假设表中的半径值为5)。

注意 也可以对字符进行比较。对字符进行比较和对字符统一码进行比较是一样的。例如:因为'a'的统一码大于'A'的统一码,所以'a'大于'A'。参见附录B就可以找到字符的顺序。

表3-1 比较运算符

运 算 符	名 称	举 例	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 相等的比较运算符是两个等号 (==)，而不是一个等号 (=)，后者是指赋值运算符。

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型变量只可能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。它们都是保留字，不能用做程序中的标识符。




3.3 问题：一个简单的数学学习工具

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，然后显示给学生“What is 7 + 9?”，如示例运行所示。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 <input type="button" value="Enter"/> 1 + 7 = 8 is true					
What is 4 + 8? 9 <input type="button" value="Enter"/> 4 + 8 = 9 is false					
line#	number1	number2	answer	output	
5	4				
6		8			
14			9		
16				4 + 8 = 9 is false	

3.4 if语句

前面的程序会显示像“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细微的改变。

本节介绍选择语句。Java有几种类型的选择语句：单向if语句、双向if语句、嵌套if语句、switch语句和条件表达式。

单向if语句

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

```
if (布尔表达式) {
    语句 (组);
}
```

执行的流程图如图3-1a所示。

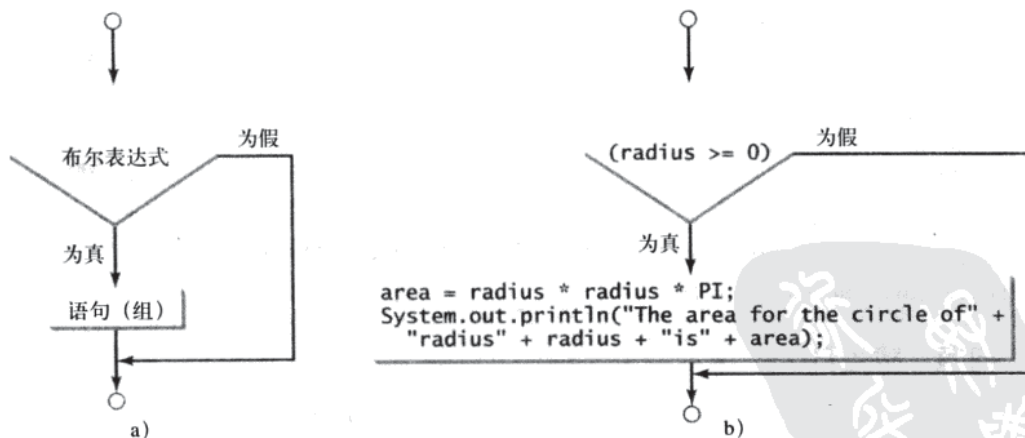


图3-1 if语句在布尔表达式计算结果为true的情况下执行语句组

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);
}
```

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)

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
HiEven



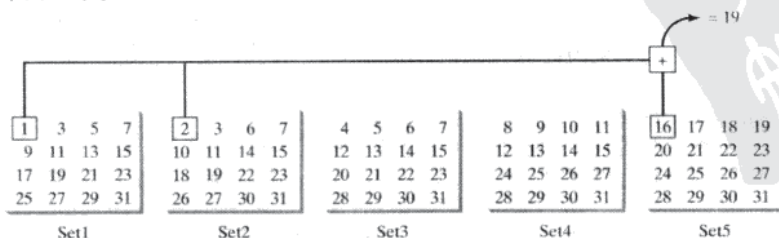
Enter an integer: 30
HiFive
HiEven



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

3.5 问题：猜生日

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。每个问题都是在询问他的出生日是否是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3-3给出程序，提示用户回答该天是否在集合1中（第41~47行），是否在集合2中（第50~56行），是否在集合3中（第59~65行），是否在集合4中（第68~74行），是否在集合5中（第77~83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3-3 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;
```

60 • 第3章 选 择

```

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

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

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

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

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		Your birthday is 19



这个游戏是很容易编程的。你可能很好奇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。它们放在五个集合中的方式是经过深思熟虑的。这五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。从1到31的十进制数最多用五个二进制数就可以表示，如图3-2a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3-2b所示。如果某天的二进制数在 b_i 位为整数1，那么该数就该出现在Set*k*中。例如：数字19的二进制是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 $1+10+100+1000+10000=11111$ ，或是十进制数 $1+2+4+8+16=31$ 。

Decimal	Binary			
1	00001	b_5 0 0 0 0		10000
2	00010	b_4 0 0 0		1000
3	00011	b_3 0 0	10000	100
...		b_2 0	10	10
19	10011	b_1	$+ \frac{1}{10011}$	$+ \frac{1}{11111}$
...		$b_5 b_4 b_3 b_2 b_1$	19	31

a)

b)

图3-2 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

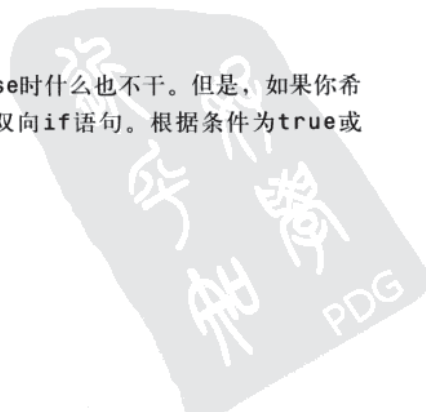
3.6 双向if语句

当指定条件为true时单向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？你可以使用双向if语句。根据条件为true或false，双向if语句可以指定不同的操作。

下面是双向if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3-3所示。



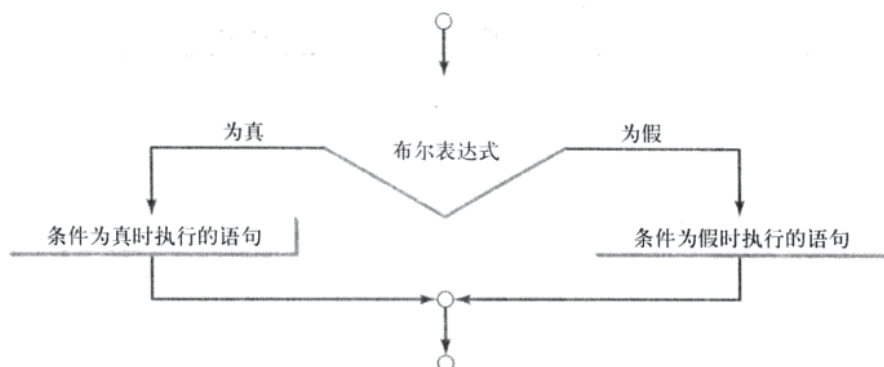


图3-3 若布尔表达式计算结果为true, if...else语句运行true情况下的语句组;
否则, 运行false情况下的语句组

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>=0为true, 则计算并显示area; 如果radius>=0为false, 则打印信息"Negative input"。

通常, 如果花括号中只有一条语句, 那就可以省略花括号。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.");
  
```

3.7 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。外层if语句里的内层if语句称为是嵌套的(nested)。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3-4a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。

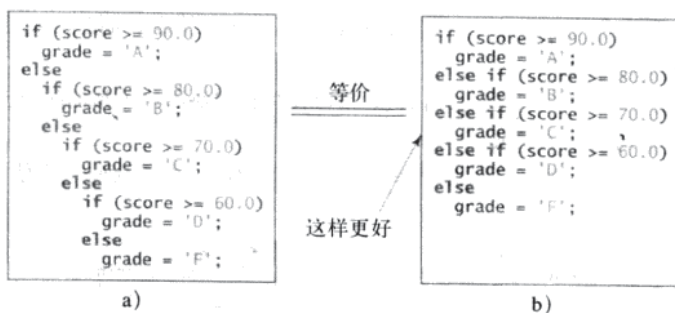
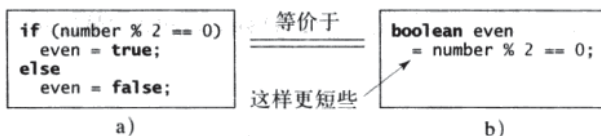


图3-4 推荐使用如图3-4b所示的多选if语句格式

这个if语句的执行过程如下。测试第一个条件 ($\text{score} \geq 90.0$)。如果它为true, 级别就变成 'A'。如果它为false, 就测试第二个条件 ($\text{score} \geq 80.0$)。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3-4a中的if语句等价于图3-4b中的if语句。事实上, 图3-4b是推荐使用的多重选择if语句的书写风格。这种风格可以避免深度缩进, 并使程序容易阅读。

提示 编程新手通常会用下面图a中的代码, 将一个测试条件赋值给一个布尔变量。



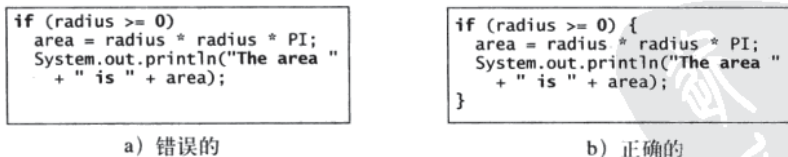
可以通过将这个测试值直接赋值给该变量来简化代码, 如图b中所示。

3.8 选择语句中的常见错误

以下错误是编程新手经常会犯的错误。

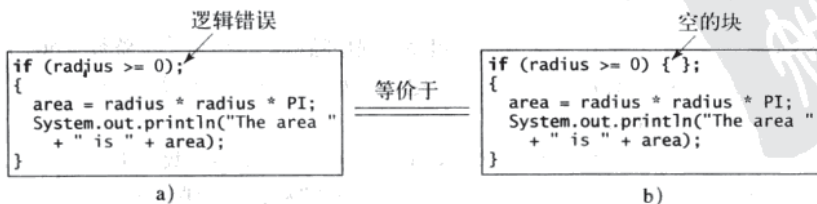
常见错误1: 忘记必要的括号

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。



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。

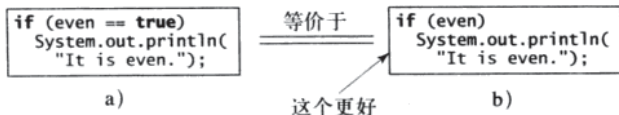


这个错误是很难发现的，因为它既不是编译错误也不是运行错误，而是一个逻辑错误。图a中的代码等价于一个带空块的图b中的代码。

当使用下行块风格时，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运算符是多余的：



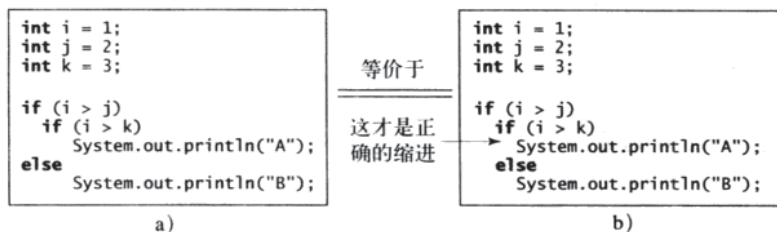
比较好的替代方法就是直接测试布尔变量，如图b所示。这么做的另一个原因就是避免出现难以发现的错误。使用=运算符而不是==运算符去比较测试条件中的两项是否相等是一个常见错误。它可能会导致出现下面的错误语句：

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

这条语句没有语法错误。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。

3.9 问题：一个改进的数学学习工具

假设你想开发一个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 $\text{number1} \geq \text{number2}$ 。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)。

程序可能如下工作:

- 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3-4所示。

程序清单3-4 **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\n" + number1 + " - "
27                               + number2 + " should be " + (number1 - number2));
28    }
29 }

```

What is 6 - 6? 0 
You are correct!



What is 9 - 2? 5 
Your answer is wrong
9 - 2 should be 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)。

3.10 问题: 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。将以千克为单位的体重除以以米为单位身高的平方, 就得到BMI的值。针对16岁及以上年龄的人群, 他们的BMI值的说明如右表所示:

编写程序, 提示用户输入以英镑为单位的体重, 以及以英尺为单位的身高, 然后显示BMI。注意: 一磅是0.45359237千克, 而一英尺是0.0254米。程序清单3-5给出这个程序。



BMI	说 明
16以下	严重偏轻
16~18	偏轻
18~24	正常体重
24~29	超重
20~35	严重超重
35以上	非常严重超重

程序清单3-5 ComputeBMI.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("Your BMI is " + bmi);
26        if (bmi < 16)
27            System.out.println("You are seriously underweight");
28        else if (bmi < 18)
29            System.out.println("You are underweight");
30        else if (bmi < 24)
31            System.out.println("You are normal weight");
32        else if (bmi < 29)
33            System.out.println("You are overweight");
34        else if (bmi < 35)
35            System.out.println("You are seriously overweight");
36        else
37            System.out.println("You are gravely overweight");
38    }
39 }

```

Enter weight in pounds: 146 
Enter height in inches: 70 
Your BMI is 20.948603801493316
You are normal weight



line#	weight	height	WeightInKilograms	heightInMeters	bmi	output
9	146					
13		70				
19			66.22448602			
20				1.778		
21					20.9486	
25						Your BMI is 20.95
31						You are normal weight

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。

3.11 问题：计算税款

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
}
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3-6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3-6 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(
10             "(0-single filer, 1-married jointly,\n" +
11             "2-married separately, 3-head of household)\n" +
12             "Enter the filing status: ");
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) { // Compute tax for married file jointly
43             // Left as exercise
44         }
45         else if (status == 2) { // Compute tax for married separately
46             // Left as exercise
47         }
48         else if (status == 3) { // Compute tax for head of household
49             // Left as exercise
50         }
51         else {
52             System.out.println("Error: invalid status");
53             System.exit(0);
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,
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。多重选择if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(0)`（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0表明程序正常结束。

赋初始值0给tax（第20行）。因为所有其他给tax赋值的语句都在if语句中，所以，如果tax没有初值，就会出现一个语法错误。编译器认为这些语句不可能执行，因此会报告一个语法错误。

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小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）。这个方法使得调试变得更加容易，因为错误很可能就你刚刚添加进去的新代码中。

3.12 逻辑运算符

有时候，是否执行一条语句是由几个条件的组合来决定的。可以使用逻辑运算符组合这些条件。逻辑运算符（logical operator）也称为布尔运算符（boolean operator），是对布尔值进行的运算，它会创建新的布尔值。表3-3列出了布尔运算符清单。表3-4定义了非运算符（!）。非运算符（!）对true取反是false，而false取反之后则是true。表3-5定义了与运算符（&&）。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定义了或运算符（||），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定义了异或运算符（^）。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。

表3-3 布尔运算符

运 算 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 运算符!的真值表

操作数p	非的结果!p	举例 (假设age=24, gender='F')
true	false	!(age>18)为false, 因为 (age>18) 为true
false	true	!(gender=='M')为true, 因为 (gender=='M') 为false

表3-5 运算符&&的真值表

操作数p1	操作数p2	与的结果p1&p2	举例 (假设age=24, gender='F')
false	false	false	(age>18) && (gender=='F') 为true, 因为 (age>18) 和 (gender=='F') 都为true
false	true	false	
true	false	false	(age>18) && (gender!='F') 为false, 因为 (gender!='F') 为false
true	true	true	

表3-6 或运算符||的真值表

操作数p1	操作数p2	或的结果p1 p2	举例 (假设age=24, gender='F')
false	false	false	(age>34) (gender=='F') 为true, 因为 (gender=='F') 为true
false	true	true	
true	false	true	(age>34) (gender=='M') 为false, 因为 (age>34) 和 (gender=='M') 都是false
true	true	true	

表3-7 异或运算符^的真值表

操作数p1	操作数p2	异或的结果p1^p2	举例 (假设age=24, gender='F')
false	false	false	(age>34) ^ (gender=='F') 为true, 因为 (age>34) 为false而 (gender=='F') 为true
false	true	true	
true	false	true	(age>34) ^ (gender=='M') 为false, 因为 (age>34) 和 (gender=='M') 都为false
true	true	false	

程序清单3-7给出的程序检验一个数是否能被2和3整除, 是被2还是3整除, 是否只能被2或3两者中的一个整除。

程序清单3-7 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 System.out.println("Is " + number +
13     "\n\tdivisible by 2 and 3? " +
14     (number % 2 == 0 && number % 3 == 0)
15     + "\n\tdivisible by 2 or 3? " +
16     (number % 2 == 0 || number % 3 == 0) +
17     "\n\tdivisible by 2 or 3, but not both? "
18     + (number % 2 == 0 ^ number % 3 == 0));
19 }
20 }

```

```

Enter an integer: 18
Is 18
    divisible by 2 and 3? true
    divisible by 2 or 3? true
    divisible by 2 or 3, but not both? false

```



在第12~18行对子串进行链接形成一个长的字符串。三个\n字符将字符串显示在四行中。(number%2==0&&number%3==0) (第14行) 检验一个数是否能被2和3整除。(number%2==0||number%3==0) (第16行) 检验一个数是否能被2或3整除。(number%2==0^ number%3==0) (第20行) 检验一个数是否能被2或3整除但又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。下面的两个操作数 (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 如前一章所示, 一个char型值可以转换为int型值, 反之亦然。但是, 一个布尔型值不能转换为其他类型的值, 其他类型的值也不能转换为布尔类型值。

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是一样的。

!(condition1 || condition2)和!condition1 && ! condition2是一样的。

例如:

!(n == 2 || n==3)和n!= 2 && n!= 3是一样的。

!(n % 2 == 0 && n % 3 == 0)和n % 2 != 0 || n % 3 != 0是一样的。

如果运算符&&的操作数之一为false, 那么表达式就是false; 如果运算符||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运算符的效率。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。因此, &&又称为条件与 (conditional AND) 运算符或短路与 (short-circuit AND) 运算符, 而||称为条件或 (conditional OR) 运算符或短路或 (short-circuit OR) 运算符。

3.13 问题: 判定闰年

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 (leap year)。所以, 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3-8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3-8 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: 2002 
 2002 is a leap year? false



3.14 问题：彩票

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美金。
- 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美金。
- 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美金。

完整的程序如程序清单3-9所示。

程序清单3-9 Lottery.java


```
1 import java.util.Scanner;
2
3 public class Lottery {
4     public static void main(String[] args) {
5         // Generate a lottery
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): 45 
 The lottery number is 12
 Sorry, no match



Enter your lottery pick: 23 
 The lottery number is 34
 Match one digit: you win \$1,000



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。

3.15 switch语句

程序清单3-6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3-6中的嵌套if语句:

```
switch (status) {
    case 0: compute taxes for single filers;
            break;
    case 1: compute taxes for married filing jointly;
            break;
    case 2: compute taxes for married filing separately;
            break;
    case 3: compute taxes for head of household;
            break;
    default: System.out.println("Errors: invalid status");
            System.exit(0);
}
```

上面的switch语句的流程图如图3-5所示。

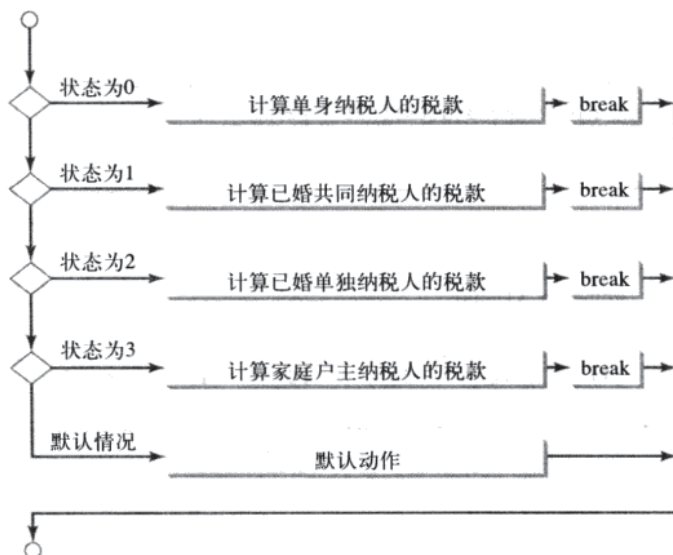


图3-5 switch语句检验所有的情况并执行匹配条件时的语句

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:

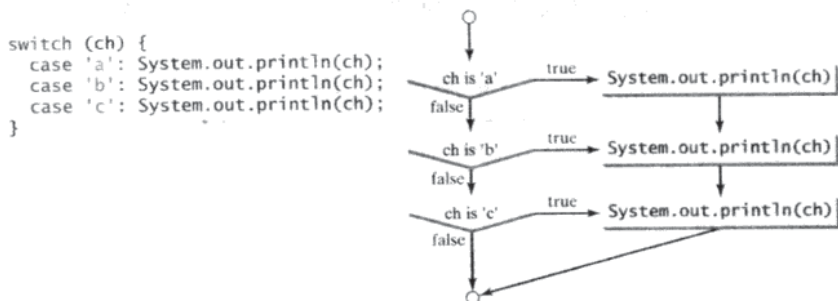
```
switch (switch表达式) {
    case 值1: 语句 (组) 1;
            break;
    case 值2: 语句 (组) 2;
            break;
    ...
    case 值N: 语句 (组) N;
            break;
    default: 默认情况下执行的语句 (组)
}

```

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或int型值，并且必须总是要用括号括住。
- 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语句的末端。
- 4) 关键字break是可选的。break语句会立即终止整个switch语句。
- 5) 默认情况（default）是可选的，用来指定没有一个指定的case与switch表达式匹配时应该采取的操作。
- 6) case语句是顺序检测的，但是，这些case的出现顺序（包括默认情况）是不重要的。然而，良好的编程风格是将情况按照逻辑顺序排列，并把默认情况放在最后。

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语句的末端。这种现象称为向下贯通行为（fall-through behavior）。例如，如果ch是'a'，下面的代码将字母a打印三次：



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。

3.16 条件表达式

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");
```

注意 符号?和:在条件表达式中同时出现。它们构成一种条件运算符，因为操作数有三个，所以称为三目运算符 (ternary operator)。它是Java中唯一的三目运算符。

3.17 格式化控制台输出

如果希望显示浮点值小数点后两位，那么可以如下编写代码：

```
double x = 2.0 / 3;
System.out.println("x is " + (int)(x * 100) / 100.0);
```

x is 0.66



但是，完成这个任务更好的方法是使用printf方法格式化输出。调用这个方法的话语是：

```
System.out.printf(format, item1, item2, ..., itemk)
```

这里的format是指一个子串和格式标识符构成的字符串。

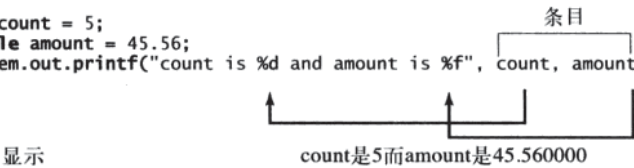
格式标识符指定每个条目应该如何显示。这里的条目可以是数值、字符、布尔值或字符串。一个标识符是以百分号(%)开头的转换码。表3-8列出了一些常用的简单标识符。

表3-8 常用的标识符

标 识 符	输 出	举 例
%b	布尔值	true或false
%c	字符	'a'
%d	十进制整数	200
%f	浮点数	45.460000
%e	标准科学记数法形式的数	4.556000e+01
%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);
```



条目与标识符必须在次序、数量和类型上匹配。例如：count的标识符应该是%d，而amount的标识符应该是%f。默认情况下，浮点值显示小数点后6位。可以在标识符中指定宽度和精度，如表3-9中的例子所示。

表3-9 指定宽度和精度的例子

举 例	输 出
%5c	输出字符并在这个字符条目前面加4个空格
%6b	输出布尔值，在false前加一个空格，在true前加两个空格
%5d	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，就在前面加空格。如果该条目的位数大于5，则自动增加宽度
%10.2f	输出的浮点数宽度至少为10，包括小数点和小数点后两位。这样，给小数点前分配了7位。如果该项小数点前的位数小于7，就在数字前面加空格。如果该项小数点前的位数大于7，则自动增加宽度
%10.2e	输出的浮点条目的宽度至少为10，包括小数点、小数点后两位和指数部分。如果按科学记数法显示的数字小于10，就给数前加空格
%12s	输出的字符串宽度至少为12。如果该字符串条目小于12个字符，就在该条目前加空格。如果该字符串条目多于12个字符，则自动增加宽度

可以使用printf方法改写在本节开始部分出现的，用于显示浮点值小数点后两位的代码，如下所示：

```
double x = 2.0 / 3;
System.out.printf("x is %4.2f", x);
display      x is 0.67
```

%4.2f

← 格式描述符

域宽度

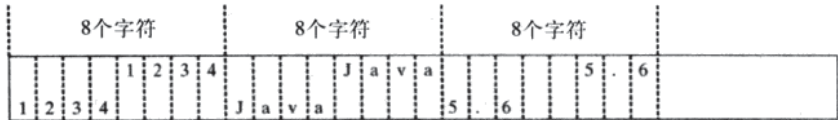
转换码

精度

默认情况下，输出是右对齐的。可以在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", 5.6);
System.out.printf("%-8d%-8s%-8.1f\n", 1234, "Java", 5.6);
```

显示



警告 条目与标识符必须在类型上严格匹配。对应于标识符%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%来标记标识符，要在格式字符串里输出直接量%，就要使用%%。

3.18 运算符的优先级和结合方向

运算符的优先级和结合方向决定了运算符的计算顺序。假设有这样一个表达式：

```
3 + 4 * 4 > 5 * (4 + 3) - 1
```

它的值是多少呢？这些运算符的执行顺序是什么呢？

从数学角度看，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，运算符会依照优先级规则和结合规则进行运算。

优先级规则定义了运算符的先后次序，如表3-10所示，它包含了目前所学的所有运算符。它们从上到下按优先级递减的方式排列。优先级相同的运算符排在同一行。（Java运算符及其优先级的完整列表参见附录C。）

如果相邻运算符的优先级相同，则结合方向（associativity）决定它们的执行顺序。除了赋值运算符之外，所有的二元运算符都是左结合的（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 等价于 ((a - b) + c) - d

表3-10 运算符优先级表

优 先 级	运 算 符
最高级	var++和var--（后置运算符） +、-（一元加号和一元减号）、++var、--var（前置运算符） (type)（类型转换） ！（非） *、/、%（乘法、除法和求余运算） +、-（二元加法和减法） <、<=、>、>=（比较运算符）

(续)

优 先 级	运 算 符
	<code>==</code> 、 <code>!=</code> (相等运算符)
	<code>^</code> (异或)
	<code>&&</code> (条件与)
	<code> </code> (条件或)
最低级	<code>=</code> 、 <code>+=</code> 、 <code>-=</code> 、 <code>*=</code> 、 <code>/=</code> 、 <code>%=</code> (赋值运算符)

赋值运算符是右结合的 (right-associative)。因此, 表达式:

$$a = b += c = 5 \quad \text{等价于} \quad a = (b += (c = 5))$$

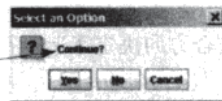
假设赋值前`a`, `b`和`c`都是1, 在计算表达式之后, `a`变成6, `b`变成6, 而`c`变成5。注意: 左结合对赋值运算符而言是没有什么意义的。

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。

3.19 (GUI) 确认对话框

你已经使用过`showMessageDialog`来显示一个消息对话框, 用`showInputDialog`来显示一个输入对话框。有时候, 使用确认对话框来回答问题是非常有用的。可以使用下面的语句创建一个确认对话框:

```
int option =
JOptionPane.showConfirmDialog
(null, "Continue");
```



当点击按钮时, 这个方法会返回一个选择值。点击Yes按钮返回的值是`JOptionPane.YES_OPTION(0)`, 点击No按钮返回的值是`JOptionPane.NO_OPTION(1)`, 点击Cancel按钮返回的值是`JOptionPane.CANCEL_OPTION(2)`。

可以使用确认对话框改写程序清单3-3中猜生日的程序, 如程序清单3-10所示。图3-6给出要猜出的生日是19时, 程序的一个运行示例。

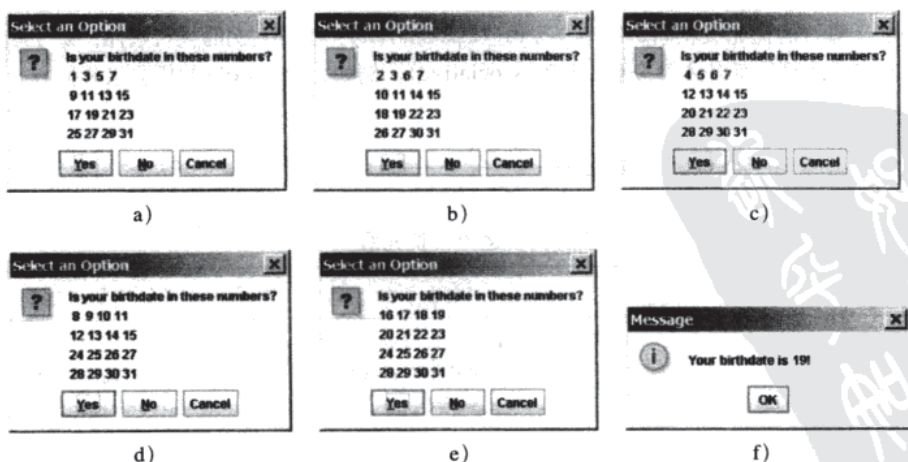


图3-6 在a中点击Yes, 在b中点击Yes, 在c中点击No, 在d中点击No, 在e中点击Yes

程序清单3-10 GuessBirthdayUsingConfirmationDialog.java

```
1 import javax.swing.JOptionPane;
2
3 public class GuessBirthdayUsingConfirmationDialog {
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         // Prompt the user to answer questions
38         int answer = JOptionPane.showConfirmDialog(null,
39             "Is your birthday in these numbers?\n" + set1);
40
41         if (answer == JOptionPane.YES_OPTION)
42             day += 1;
43
44         answer = JOptionPane.showConfirmDialog(null,
45             "Is your birthday in these numbers?\n" + set2);
46
47         if (answer == JOptionPane.YES_OPTION)
48             day += 2;
49
50         answer = JOptionPane.showConfirmDialog(null,
51             "Is your birthday in these numbers?\n" + set3);
52
53         if (answer == JOptionPane.YES_OPTION)
54             day += 4;
55
56         answer = JOptionPane.showConfirmDialog(null,
57             "Is your birthday in these numbers?\n" + set4);
58
59         if (answer == JOptionPane.YES_OPTION)
60             day += 8;
61
62         answer = JOptionPane.showConfirmDialog(null,
63             "Is your birthday in these numbers?\n" + set5);
```

数字图书馆
PDG

```

64
65     if (answer == JOptionPane.YES_OPTION)
66         day += 16;
67
68     JOptionPane.showMessageDialog(null, "Your birthday is " +
69         day + "!");
70 }
71 }

```

该程序显示确认对话框，提示用户回答某个数字是否在集合1中（第38行），是否在集合2中（第44行），是否在集合3中（第50行），是否在集合4中（第56行）以及是否在集合5中（第62行）。如果得到的答案是Yes，就将该集合中的第一个数加到day中（第42、48、54、60和66行）。

关键术语

Boolean expression（布尔表达式）	fall-through behavior（向下贯通行为）
Boolean value（布尔值）	operator associativity（运算符结合方向）
boolean type（boolean类型）	operator precedence（运算符优先级）
break statement（break语句）	selection statement（选择语句）
conditional operator（条件运算符）	short-circuit evaluation（短路求值）
dangling-else ambiguity（悬空else歧义）	

本章小结

- 一个boolean变量可以存储值true或false。
- 关系运算符（<、<=、==、!=、>、>=）和数值及字符一起运算，然后产生一个布尔值。
- 布尔运算符&&、||、|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，就不在对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，就不在对p2求值。因此，&&也称为条件与运算符或短路与运算符，而||也称为条件或运算符或短路或运算符。
- 使用选择语句可以对有可选择路径的情况进行程序设计。选择语句有以下几种类型：if语句、if...else语句、嵌套if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，选择两种可能路径中的一种。
- switch语句根据switch表达式的类型char、byte、short或int来控制决定。
- 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。

复习题

3.2节

3.1 列出六个比较运算符。

3.2 是否允许做以下的类型转换？如果允许，给出转换结果。

```

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

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

```

3.3~3.11节

3.3 如果number分别是30和35,那么图a中的代码和图b中的代码会打印出什么结果?

```
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)

3.4 假设 $x=3$ 而 $y=2$,若有输出,给出下面代码的输出结果。当 $x=3$ 而 $y=4$ 时,输出结果是什么?当 $x=2$ 而 $y=2$ 时输出又是什么?画出下列代码的流程图:

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

3.5 下面哪些语句是等价的?哪些语句缩进正确?

```
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)

3.6 假设 $x=2$ 而 $y=3$,若有输出,给出下列代码的输出结果。如果 $x=3$ 而 $y=2$,输出是什么?当 $x=3$ 而 $y=3$ 时,输出又是什么?

提示 首先正确缩进这些语句。

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

3.7 下面这两个语句等价吗?

```
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;
```

3.8 下面哪些是调用Math.random()后可能出现的输出?

323.4, 0.5, 34, 1.0, 0.0, 0.234

3.9 如何产生一个满足条件 $0 \leq i < 20$ 的随机整数 i ?如何产生一个满足条件 $10 \leq i < 20$ 的随机整数 i ?如何产生一个满足条件 $10 \leq i \leq 50$ 的随机整数 i ?

3.10 编写一个if语句实现:如果 y 大于0就给 x 赋值为1。

3.11 (1) 编写一个if语句实现:如果score大于90就给pay增加3%。(2) 编写一个if语句实现:如果score大于90就给pay增加3%,若不满足条件就给pay增加1%。

3.12 下面代码中的错误是什么?

82 • 第3章 选 择

```

if (score >= 60.0)
    grade = 'D';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 90.0)
    grade = 'A';
else
    grade = 'F';

```

3.13 使用布尔表达式改写下面的语句:

```

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

```

3.12~3.14节

3.14 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)

```

3.15 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。

3.16 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。

3.17 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)

```

3.18 假定x是1, 经过下面的表达式运算后, x的值是多少?

```

(x >= 1) && (x++ > 1)
(x > 1) && (x++ > 1)

```

3.19 如果ch是'A'、'p'、'E'或'5', 那么表达式ch>='A'&&ch<='Z'的值是什么?

3.20 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));
    }
}

```

3.21 编写当年龄age大于13且小于18时结果为true的布尔表达式。

3.22 编写当体重weight大于50或身高大于160时结果为true的布尔表达式。

3.23 编写当体重weight大于50且身高height大于160时结果为true的布尔表达式。

3.24 编写当体重weight大于50或身高height大于160, 但不能同时满足这两个条件时, 结果为true的布尔表达式。

3.15节

3.25 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? 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?

3.26 执行下列switch语句之后, y是多少?

```
x = 3; y = 3;
switch (x + 3) {
    case 6: y = 1;
    default: y += 1;
}
```

3.27 使用switch语句改写下面的if语句, 并画出switch语句的流程图:

```
if (a == 1)
    x += 5;
else if (a == 2)
    x += 10;
else if (a == 3)
    x += 16;
else if (a == 4)
    x += 34;
```

3.28 编写switch语句, 如果day是0、1、2、3、4、5、6, 那么String变量dayName被依次赋值为Sunday、Monday、Tuesday、Wednesday、Thursday、Friday、Saturday。

3.16节

3.29 使用条件运算符改写下面的语句:

```
if (count % 10 == 0)
    System.out.print(count + "\n");
else
    System.out.print(count + " ");
```

3.30 使用条件表达式改写下面的语句:

```
if (temperature > 90)
    pay = pay * 1.5;
else
    pay = pay * 1.1;
```

3.17节

3.31 输出布尔值、字符、十进制整数、浮点数和字符串的格式描述符分别是什么?

3.32 下面的语句错在哪里?

- (a) System.out.printf("%5d %d", 1, 2, 3);
- (b) System.out.printf("%5d %f", 1);
- (c) System.out.printf("%5d %f", 1, 2);

3.33 给出下面语句的输出。

- (a) System.out.printf("amount is %f %e\n", 32.32, 32.32);
- (b) System.out.printf("amount is %5.4f %5.4e\n", 32.32, 32.32);
- (c) System.out.printf("%6b\n", (1 > 2));
- (d) System.out.printf("%6s\n", "Java");
- (e) System.out.printf("%-6b%s\n", (1 > 2), "Java");
- (f) System.out.printf("%6b%-8s\n", (1 > 2), "Java");

3.34 如何创建一个格式化字符串?

3.18节

3.35 列出布尔运算符的优先级顺序。计算下面的表达式:

```
true || true && false
true && true || false
```

3.36 除=之外的所有二元运算符都是左结合的说法是真还是假?

3.37 计算下面的表达式:

```
2 * 2 - 3 > 2 && 4 - 2 > 5
2 * 2 - 3 > 2 || 4 - 2 > 5
```

3.38 $(x>0 \ \&\& \ x<10)$ 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?

3.19节

3.39 如何显示一个确认对话框? 当调用 `JOptionPane.showConfirmDialog` 时会返回什么值?

编程练习题

教学注意 对于每一个练习题, 学生都应在编码之前仔细地分析解决该问题的需求及其策略设计。

教学注意 教师可以要求学生针对布置的练习题进行文档分析和设计。学生应该用自己的话来分析问题, 包括输入、输出以及需要计算什么, 并用伪代码描述如何解决该问题。

调试提示 在寻求帮助之前, 自己先阅读和解释一下程序, 然后手动使用几个具有代表性的输入跟踪程序, 或者使用某个IDE调试器跟踪程序。可以通过调试自己的错误来学习如何编程。

3.2节


*3.1 (代数方面: 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:

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \text{ 和 } r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$


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正根。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 roots are -0.381966 and -2.61803




Enter a, b, c: 1 2.0 1 
The root is -1



Enter a, b, c: 1 2 3 
The equation has no real roots



3.2 (检查一个数字是否是偶数) 编写程序, 读入一个整数并检查它是不是偶数。这个程序的运行示例如下所示:

Enter an integer: 25 
Is 25 an even number? false



Enter an integer: 2000 
Is 2000 an even number? true



3.3~3.8节

*3.3 (代数方面: 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:

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

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 Enter
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 Enter
The equation has no solution



****3.4（游戏：学习加法）** 编写程序，产生两个100以下的整数，然后提示用户输入这两个整数的和。如果答案正确，程序报告结果true；否则，报告false。该程序类似于程序清单3-1。

****3.5（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。

***3.6（医疗应用程序：BMI）** 修改程序清单3-5，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而英寸值为10。

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）。（使用输入值23.67来测试该程序。）

***3.8（对三个整数排序）** 编写程序对三个整数排序。这些整数都由输入对话框输入，并分别存储在变量num1、num2和num3中。程序对这些数进行排序，使之满足 $num1 \leq num2 \leq num3$ 。

3.9（商业方面：检查ISBN） ISBN（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。例如：输入的是013601267，那么程序就该显示0136012671。

***3.10（游戏：加法测验）** 程序清单3-4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。

3.9~3.19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2000，那么程序应该显示“February 2000 has 29 days”（2000年2月有29天）。如果用户输入的月份为3而年份为2005，那么程序就应该显示“March 2005 has 31 days”（2005年3月有31天）。

3.12（检测数字） 编写程序，提示用户输入一个整数，然后判断这个整数是否能被5和6都整除，或者不能被它们中的任何一个整除，或者只能被其中一个整除。下面是分别输入10、30和23时的一些运行示例。

```
10 is divisible by 5 or 6, but not both
30 is divisible by both 5 and 6
23 is not divisible by either 5 or 6
```

3.13（财务应用程序：计算税款） 程序清单3-6给出计算单身登记人税款的源代码。整个程序清单3-6给出的就是完整的源代码。

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3-9, 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美金。
- (2) 如果用户输入的所有数匹配彩票的所有数字, 奖金是3000美金。
- (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1000美金。

3.16 (任意字符) 使用Math.random()编写程序, 显示任意的一个大写字母。

*3.17 (游戏: 剪刀、石头、布) 编写可以玩最流行的剪刀-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, 然后显示一个消息, 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 (使用输入对话框) 使用输入对话框改写程序清单3-8。

3.19 (验证三角形的有效性) 编写程序, 读取三角形的三条边, 并确定输入是否有效。如果任意两条边的和大于第三条边则输入有效。以下是该程序的运行示例:

```
Enter three edges: 1 2.5 1
Can edges 1, 2.5, and 1 form a triangle? false
```



```
Enter three edges: 2.5 2 1
Can edges 2.5, 2, and 1 form a triangle? true
```



*3.20 (科学方面: 风寒温度) 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 (科学方面: 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:

$$h = \left(q + \left\lfloor \frac{26(m+1)}{10} \right\rfloor + k + \left\lfloor \frac{k}{4} \right\rfloor + \left\lfloor \frac{j}{4} \right\rfloor + 5j \right) \% 7$$

其中:

- h是一个星期中的每一天 (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是某月的天数。
- 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 是世纪的年数（即 $\text{year} \% 100$ ）。

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., 2008): 2002
Enter month: 1-12: 3
Enter the day of the month: 1-31: 26
Day of the week is Tuesday
```



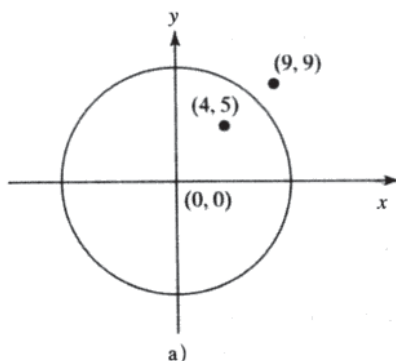
```
Enter year: (e.g., 2008): 2011
Enter month: 1-12: 5
Enter the day of the month: 1-31: 2
Day of the week is monday
```



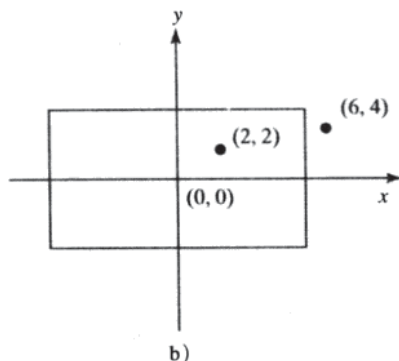
提示 对正数 n 而言 $[n] = (\text{int})n$ 。一月和二月在这个公式里是用13和14表示的。所以需要将由用户输入的月份1转换为13，将用户输入的月份2转换为14，同时将年份改为前一年。

****3.22**（几何方面：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 $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ 。以下是两个运行示例。



a)



b)

图3-7 a) 圆内和圆外的点；b) 矩形外和矩形内的点

```
Enter a point with two coordinates: 4 5
Point (4.0, 5.0) is in the circle
```



```
Enter a point with two coordinates: 9 9
Point (9.0, 9.0) is not in the circle
```



****3.23**（几何方面：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5/2，该点就在矩形内。这里有两个运行示例。这两个运行示例如下所示。

```
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)、Diamond (红方块)、Heart (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
```



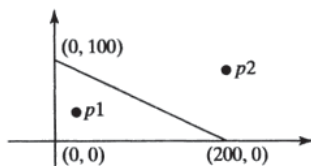
****3.25** (计算一个三角形的周长) 编写程序, 读取三角形的三条边, 如果输入值合法就计算这个三角形的周长; 否则, 显示这些输入值不合法。如果任意两条边的和大于第三边, 那么输入值都是合法的。

3.26 (使用运算符&&、||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 Enter
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** (几何方面: 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 Enter
The point is in the triangle
```



```
Enter a point's x- and y-coordinates: 100.5 50.5 Enter
The point is not in the triangle
```



****3.28** (几何方面: 两个三角形) 编写一个程序, 提示用户输入两个三角形中点的x坐标和y坐标以及它们的宽度和高度, 然后判定第二个三角形是在第一个三角形内, 还是和第一个三角形重叠, 如图3-8所示。

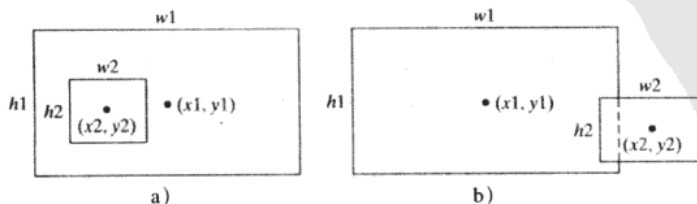


图3-8 a) 一个三角形在另一个三角形里; b) 一个三角形和另一个三角形重叠

下面是运行示例：

```
Enter r1's center x-, y-coordinates, width, and height:
2.5 4 2.5 43 --Enter
Enter r2's center x-, y-coordinates, width, and height:
1.5 5 0.5 3 --Enter
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**（几何方面：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否在第一个圆内，还是和第一个圆重叠，如图3-9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。

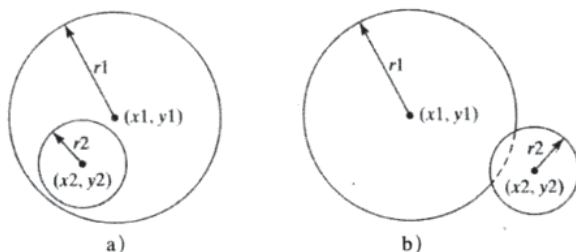


图3-9 a) 一个圆在另一个圆内；b) 一个圆和另一个圆重叠

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
```



第4章

Introduction to Java Programming, 8E

循 环

学习目标

- 使用while循环编写重复执行某些语句的程序 (4.2节)。
- 开发程序GuessNumber (4.2.1节)。
- 遵循循环设计策略来开发循环 (4.2.2节)。
- 开发程序SubtractionQuizLoop (4.2.3节)。
- 使用标志值控制循环 (4.2.4节)。
- 使用输入重定向而不是从键盘输入以获取大量输入 (4.2.4节)。
- 使用do-while语句编写循环 (4.3节)。
- 使用for语句编写循环 (4.4节)。
- 了解三种类型循环语句的相似处和不同点 (4.5节)。
- 编写嵌套循环 (4.6节)。
- 学习最小化数值误差的技术 (4.7节)。
- 从多种多样的例子 (GCD、FutureTution、MonteCarloSimulation) 中学习循环 (4.8节)。
- 使用break和continue来实现程序的控制 (4.9节)。
- (GUI) 使用确定对话框控制循环 (4.10节)。

4.1 引言

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。循环检查 (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。

4.2 while循环

while循环的语法如下：

```
while (循环继续条件) {
    //循环体
    语句 (组);
}
```

while循环的流程图如图4-1a所示。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 (iteration of the loop)。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。

4.1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4-1b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

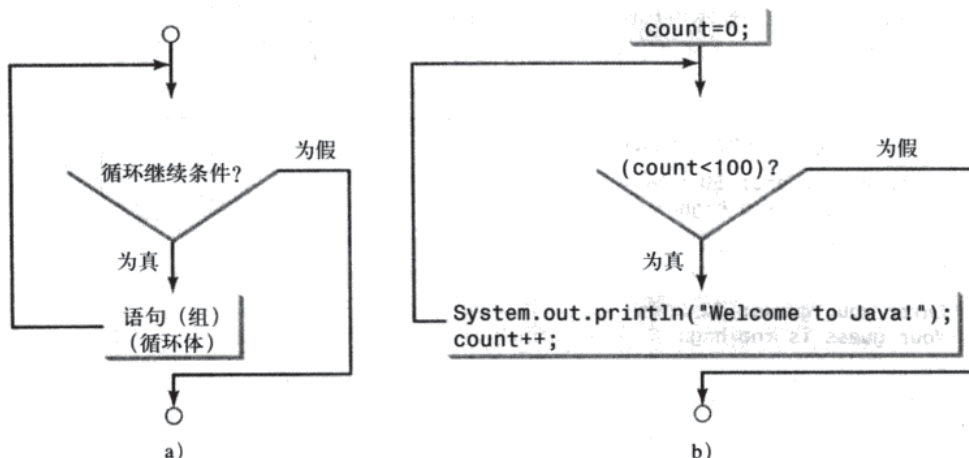


图4-1 当循环继续条件为true时，while循环重复执行循环体中的语句

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`。

警告 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, 由于循环继续条件出错而使程序不能结束。


程序员经常会犯的错误就是使循环多执行一次或少执行一次。这种情况通常称为出一错误 (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 <= 100`。

```
int count = 0;
while (count <= 100) {
    System.out.println("Welcome to Java!");
    count++;
}
```


4.2.1 举例: 猜数字

本节的问题是猜测计算机“脑子”里想的是什么数。可以编写一个程序, 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太大了, 还是太小了, 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 high

Enter your guess: 12 


Your guess is too high

Enter your guess: 6 

Your guess is too low

Enter your guess: 9 

Yes, the number is 9



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 (think before coding) 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逐步编码 (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4-1所示。

程序清单4-1 `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4-2给出完整的代码。

程序清单4-2 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	9		
iteration 1	11		-1	
	15		50	
	20			Your guess is too high
iteration 2	15		25	
	20			Your guess is too high
iteration 3	15		12	
	20			Your guess is too high
iteration 4	15		6	
	22			Your guess is too low
iteration 5	15		9	
	18			Yes, the number is 9

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，是太高还是太低了（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。

4.2.2 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;
}

```

4.2.3 问题：高级数学学习工具

每次运行程序清单3-4中的数学减法学习工具的程序只能产生一道题目。可以使用一个循环重复产生

题目。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4-3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，并列出的所有的题目。

程序清单4-3 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++;
33            }
34            else
35                System.out.println("Your answer is wrong.\n" + number1
36                    + " - " + number2 + " should be " + (number1 - number2));
37
38            // Increase the 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 Enter
You are correct!

What is 3 - 0? 3 Enter
You are correct!

What is 3 - 2? 1 Enter
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。

4.2.4 使用标志值控制循环

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志值（sentinel value），用以表明循环的结束。如果一个循环使用标志值来控制它的执行，它就称为标志位控制的循环（sentinel-controlled loop）。

程序清单4-4编写程序，用来读取和计算个数不确定的整数之和。输入0则表示输入结束。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4-4 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 int value (the program exits if the input 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 int value (the program exits if the input is 0): ";

```

```

22     data = input.nextInt();
23 }
24
25     System.out.println("The sum is " + sum);
26 }
27 }

```

Enter an int value (the program exits if the input is 0): 2 
Enter an int value (the program exits if the input is 0): 3 
Enter an int value (the program exits if the input is 0): 4 
Enter an int value (the program exits if the input 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while循环。输入值0是该循环的标志值。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。

4.2.5 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乏味的事。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。

4.3 do-while循环

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4-2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。while循环和do-while循环具有相同的表达能力。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4-4中的while循环，如程序清单4-5所示。

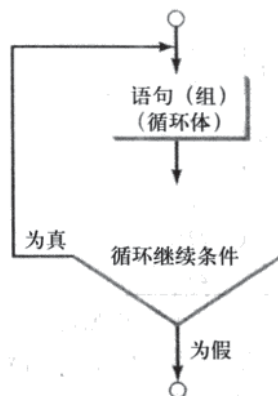


图4-2 do-while循环首先执行循环体，然后检查循环继续条件，以确定继续执行循环还是终止循环

程序清单4-5 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 int value (the program exits if the input 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 int value (the program exits if the input is 0): 3 Enter
Enter an int value (the program exits if the input is 0): 5 Enter
Enter an int value (the program exits if the input is 0): 6 Enter
Enter an int value (the program exits if the input is 0): 0 Enter
The sum is 14
```



提示 同前面程序TestDoWhile中do-while循环的情形一样，如果循环中的语句至少需要执行一次，建议使用do-while循环。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。

4.4 for循环

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4-3a所示。

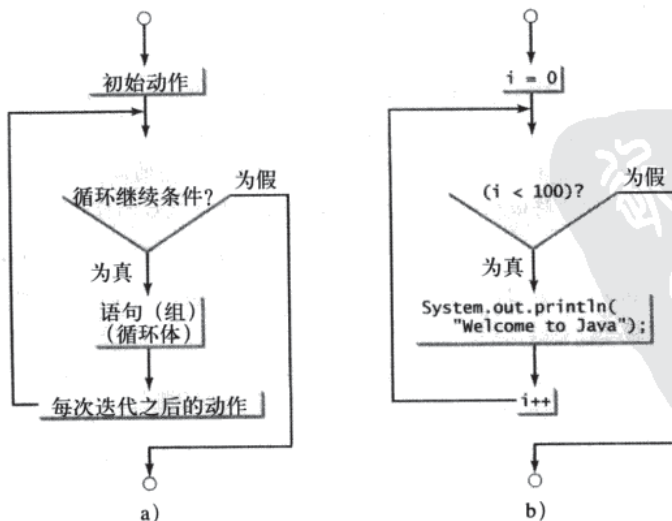


图4-3 for循环只执行初始动作一次，当循环继续条件为真时，重复执行循环体中的语句，然后完成每次迭代后的操作

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4-3b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:

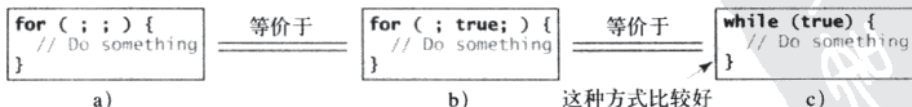
```
for (int i = 0, j = 0; (i + j < 10); i++, j++) {
    // Do something
}
```

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:



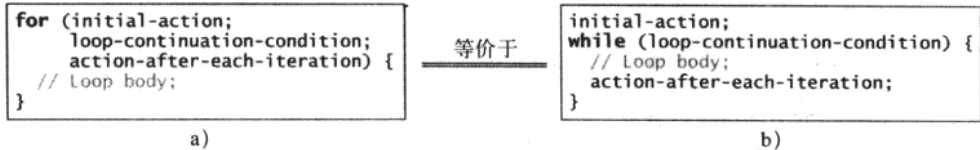
4.5 采用哪种循环

while循环和**for**循环都称为预测式循环(pretest loop),因为继续条件是在循环体执行之前检测的,

do-while循环称为后测试循环 (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：

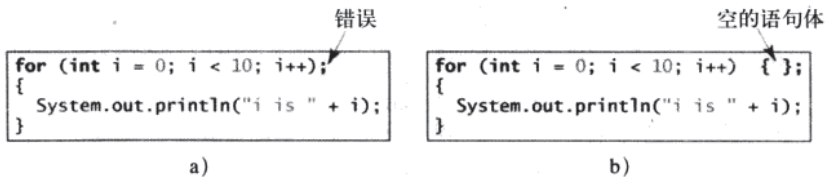


除了某些特殊情况外（参见复习题4.17中的情况），下面图a中的for循环通常都能转化为图b中的while循环：

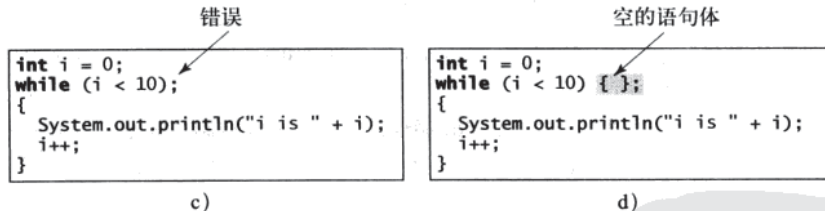


建议使用自己觉得最自然、最舒服的一种循环语句。通常，如果已经提前知道重复次数，那就采用for循环，例如，需要打印一条信息100次时，如果无法确定重复次数，就采用while循环，就像读入一些数值直到读入0为止的这种情况。如果在检验继续条件前需要执行循环体，就用do-while循环替代while循环。

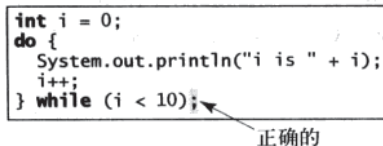
警告 在for子句的末尾和循环体之间多写分号是一个常见的错误，如下面的图a中所示。图a中分号过早地表明循环的结束。循环体实际上都是为空的，如图b所示。图a和图b是等价的。



类似地，图c中的循环也是错的，图c与图d等价。



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。



4.6 嵌套循环

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再次进入内部循环，然后重新开始。

程序清单4-6是使用嵌套for循环打印一个乘法表的程序。

程序清单4-6 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        // Print 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。

4.7 最小化数值误差

涉及浮点数的数值误差是不可避免的。本节将通过实例讨论如何最小化这种误差。

程序清单4-7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4-7 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 }

```



```

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。如果结果值需要比变量所能存储的容量值更高的精度，那么添加一个非常小的数给一个非常大的数可能没有什么影响。例如，100000000.0+0.000000001的精确结果是100000000.0。为了得到更精确的结果，仔细选择计算的顺序。在大数之前先增加小数是减小误差的一种方法。

4.8 实例学习

循环语句是程序设计的基础，编写循环语句的能力在学习Java程序设计时是非常必要的。如果能够使用循环编写程序，你就懂得如何编程了。正是由于这个原因，本节提供三个附加的例子，学习如何使用循环来解决问题。

4.8.1 举例：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。如何求最大公约数呢？设输入的两个整数为n1和n2。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...) 是否为n1和n2的最大公约数，直到k大于n1或n2。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n1或n2之间所有可能的公约数后，变量gcd的值就是最大公约数。这个思路可以翻译成下面的循环：

```
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4-8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4-8 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



如何编写这个程序呢？应该立即开始写代码吗？不，输代码之前先思考一下是很重要的。思考能够使你在不涉及怎样编写代码的情况下，得到问题的逻辑方案。一旦有了逻辑方案，再输入代码，将解决方案翻译成Java语言。翻译方式不是唯一的。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。练习题4.15就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。参考网站<http://www.cut-the-knot.org/blue/Euclid.shtml>可以得到相关的更多信息。

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4.14找出答案。

4.8.2 问题：预测未来学费

假设某个大学今年的学费是10000美金，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 = 1   // Year 1
tuition = tuition * 1.07; year++;         // Year 2
tuition = tuition * 1.07; year++;         // Year 3
tuition = tuition * 1.07; year++;         // Year 4
...
```

不断地计算新一年的学费，直到学费至少是20000美金为止。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 1
int year = 1;
while (tuition < 20000) {
    tuition = tuition * 1.07;
    year++;
}
```

完整的程序如程序清单4-9所示。

程序清单4-9 FutureTuition.java

```
1 public class FutureTuition {
2     public static void main(String[] args) {
3         double tuition = 10000;   // Year 1
4         int year = 1;
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    }
13 }
```

Tuition will be doubled in 12 years

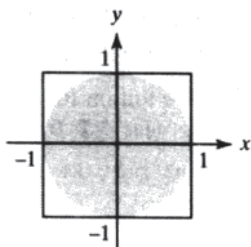


使用while循环（第5~8行）重复计算新一年的学费。当学费大于或等于20000美金时，循环结束。

4.8.3 问题：蒙特卡罗模拟

蒙特卡罗模拟使用随机数和概率来解决问题。这个方法在计算数学、物理、化学和财经方面有很广泛的应用。本节给出使用蒙特卡罗模拟来估算 π 值的例子。

为了使用蒙特卡罗方法来估算 π ，画出一个圆的外接正方形，如下所示：



假设这个圆的半径是1。那么圆面积就是 π 而外接正方形的面积是4。随便产生正方形中的一个点。该点落在这个圆内的概率是 $\text{circleArea}/\text{squareArea}$ (圆面积/正方形面积) $=\pi/4$ 。

编写程序，在正方形内随机产生1000000个点，用`numberOfHits`表示落在圆内的点。因此，`numberOfHits`大约是 $1000000 * (\pi/4)$ 。可以近似估算 π 为 $4 * \text{numberOfHits} / 1000000$ 。完整的程序如程序清单4-10所示。

程序清单4-10 MonteCarloSimulation.java

```
1 public class MonteCarloSimulation {
2     public static void main(String[] args) {
3         final int NUMBER_OF_TRIALS = 10000000;
4         int numberOfHits = 0;
5
6         for (int i = 0; i < NUMBER_OF_TRIALS; i++) {
7             double x = Math.random() * 2.0 - 1;
8             double y = Math.random() * 2.0 - 1;
9             if (x * x + y * y <= 1)
10                numberOfHits++;
11        }
12
13        double pi = 4.0 * numberOfHits / NUMBER_OF_TRIALS;
14        System.out.println("PI is " + pi);
15    }
16 }
```

PI is 3.14124



程序的第7~8行重复地产生落在正方形内的随机点 (x, y) :

```
double x = Math.random() * 2.0 - 1;
double y = Math.random() * 2.0 - 1;
```

如果 $x^2 + y^2 \leq 1$ ，那么这个点就在圆内，并给`numberOfHits`自增1。 π 就近似为 $4 * \text{numberOfHits} / \text{NUMBER_OF_TRIALS}$ (第13行)。

4.9 关键字break和continue

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4-11给出的程序演示在循环中使用break的效果。

程序清单4-11 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4-11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4-12给出的程序演示在循环中使用continue的效果。

程序清单4-12 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4-12中的程序，将1到20中除去10和11外的整数都加到sum中。程序中有了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4.18。通常，只有在能够简化代码并使程序更容易阅读的情况下，才可以适当地使用`break`和`continue`。

程序清单4-2给出猜数字的程序，可以使用`break`语句改写它，如程序清单4-13所示。

程序清单4-13 `GuessNumberUsingBreak.java`

```

1 import java.util.Scanner;
2
3 public class GuessNumberUsingBreak {
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         while (true) {
12             // Prompt the user to guess the number
13             System.out.print("\nEnter your guess: ");
14             int guess = input.nextInt();
15
16             if (guess == number) {
17                 System.out.println("Yes, the number is " + number);
18                 break;
19             }
20             else if (guess > number)
21                 System.out.println("Your guess is too high");
22             else
23                 System.out.println("Your guess is too low");
24         } // End of loop
25     }
26 }

```

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。

问题：显示素数

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, …`，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 $\text{number}/2$ 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4-14中给出。

程序清单4-14 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            // Print 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                    // Print 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。开发程序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，就该检验该数是否能被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。

4.10 (GUI) 使用确认对话框控制循环

使用确认对话框可以实现一个标志值控制的循环。答案Yes或者No决定是继续还是终止这个循环。这个循环的模板可能看起来如下所示：

```

int option = JOptionPane.YES_OPTION;
while (option == JOptionPane.YES_OPTION) {
    System.out.println("continue loop");
    option = JOptionPane.showConfirmDialog(null, "Continue?");
}

```

程序清单4-15使用一个确认对话框改写程序清单4-14。运行示例如图4-4所示。

程序清单4-15 SentinelValueUsingConfirmationDialog.java

```

1 import javax.swing.JOptionPane;
2
3 public class SentinelValueUsingConfirmationDialog {
4     public static void main(String[] args) {
5         int sum = 0;
6
7         // Keep reading data until the user answers No
8         int option = JOptionPane.YES_OPTION;
9         while (option == JOptionPane.YES_OPTION) {
10             // Read the next data
11             String dataString = JOptionPane.showInputDialog(

```



```

12     "Enter an int value: ");
13     int data = Integer.parseInt(dataString);
14
15     sum += data;
16
17     option = JOptionPane.showConfirmDialog(null, "Continue?");
18 }
19
20 JOptionPane.showMessageDialog(null, "The sum is " + sum);
21 }
22 }

```

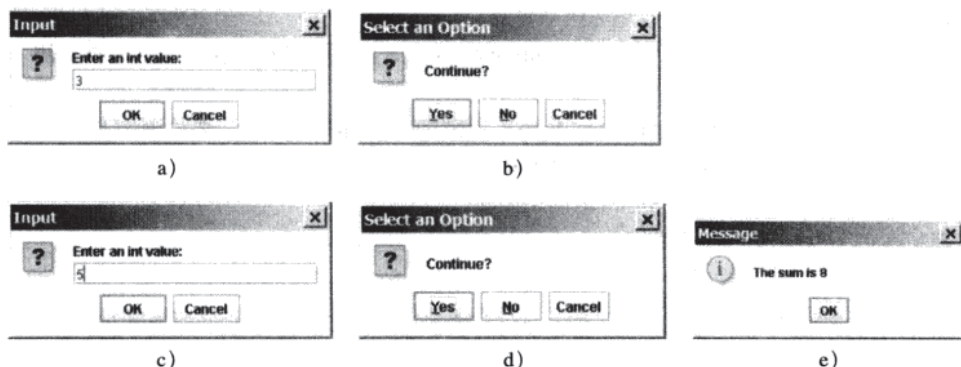


图4-4 用户在a中输入3，在b中点击Yes，在c中输入5，在d中点击Yes，结果显示在e中

程序显示一个输入对话框，提示用户输入一个整数（第11行），并将它加到sum中（第15行）。第17行显示一个确认对话框，让用户来决定是否继续输入。如果用户点击Yes，那么循环继续；否则循环终止。最后程序会在消息对话框中显示结果（第20行）。

关键术语

break statement (break语句)	loop (循环)
continue statement (continue语句)	loop-condition condition (循环继续条件)
do-while loop (do-while循环)	loop body (循环体)
for loop (for循环)	nested loop (嵌套循环)
loop control structure (循环控制结构)	off-by-one error (出一错误)
infinite loop (无限循环、死循环)	output redirection (输出重定向)
input redirection (输入重定向)	sentinel value (标志值)
iteration (迭代)	while loop (while循环)
labeled continue statement (带标号的continue语句)	

本章小结

- 循环语句有三类：while循环、do-while循环和for循环。
- 循环中需要重复执行的语句所构成的整体称为循环体。
- 循环体执行一次称为循环的一次迭代。
- 无限循环是指循环语句被无限次执行。
- 在设计循环时，既需要考虑循环控制结构体，还需要考虑循环体。
- 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。

- **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。
- 由于**while**循环和**do-while**循环都包含循环继续条件，这个条件是和循环体相关的，重复的次数就是由循环体决定的。因此，**while**和**do-while**循环常用于循环次数不确定的情况。
- 标志值是一个特殊的值，用来标志循环的结束。
- **for**循环一般用在循环体执行次数预知的情况，这个执行次数不是由循环体确定的。
- **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。
- **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。
- **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。
- 在循环中可以使用关键字**break**和**continue**。
- 关键字**break**立即终止包含**break**的最内层循环。
- 关键字**continue**只是终止当前迭代。

复习题

4.2~4.4节

4.1 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**，还是有时是**true**有时是**false**？

```
int count = 0;
while (count < 100) {
    // Point A
    System.out.println("Welcome to Java!\n");
    count++;
    // Point B
}
// Point C
```

4.2 在程序清单4-2中的第11行中，如果**guess**初始化为0，会出现什么错误？

4.3 下面的循环体要重复多少次？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)

4.4 **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。

```
int sum = 0;
int number = input.nextInt();
while (number != 0) {
    sum += number;
    number = input.nextInt();
}
```

4.5 做完下列两个循环之后，**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)

4.6 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。

4.7 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);
    }
}
```

4.8 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);
    }
}
```

4.9 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);
    }
}
```



114 • 第4章 循 环

4.10 下面的语句做什么？

```
for ( ; ; ) {
    do something;
}
```

4.11 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？

4.12 可以将for循环转换为while循环吗？列出使用for循环的好处？

4.13 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;
```

4.14 如果将程序清单4-8中第17行的n1和n2用n1/2和n2/2来替换，程序还会工作吗？

4.9节

4.15 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 = 1000;
while (true) {
    if (balance < 9)
        break;
    balance = balance - 9;
}

System.out.println("Balance is "
    + balance);
```

a)

```
int balance = 1000;
while (true) {
    if (balance < 9)
        continue;
    balance = balance - 9;
}

System.out.println("Balance is "
    + balance);
```

b)

4.16 是否总能将while循环转换为for循环？将下列while循环转换为for循环。

```
int i = 1;
int sum = 0;
while (sum < 10000) {
    sum = sum + i;
    i++;
}
```

4.17 将下面左边的for循环转换成右边的while循环，有什么错误？改正该错误。

```
for (int i = 0; i < 4; i++) {
    if (i % 3 == 0) continue;
    sum += i;
}
```

转换为

错误转换

```
int i = 0;
while (i < 4) {
    if(i % 3 == 0) continue;
    sum += i;
    i++;
}
```

4.18 不使用关键字break和continue，改写程序清单4-11和程序清单4-12的程序TestBreak和TestContinue。

4.19 在下面的循环中，执行完break语句之后，执行哪条语句？显示输出。

```
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);
}
```

4.20 在下面的循环中，执行continue语句之后，执行哪条语句？显示输出。

```
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);
}

```

综合题

4.21 查找并修改下面代码中的错误：

```

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 }

```

4.22 下面程序的错误在哪里？

```

1 public class ShowErrors {
2     public static void main(String[] args) {
3         int i;
4         int j = 5;
5
6         if (j > 3)
7             System.out.println(i + 4);
8     }
9 }

```

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)

4.23 给出下面程序的输出结果（提示：画一个表格，将变量放在某一列中，跟踪这些程序）。

```

public class Test {
    /** Main method */
    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 {
    /** Main method */
    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)

4.24 下面程序的输出是什么？解释原因。

```
int x = 80000000;
while (x > 0)
    x++;
System.out.println("x is " + x);
```

4.25 统计下面循环的迭代次数。

```
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)

编程练习题

教学注意 对每个问题，都应该多读几次，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。

一个问题经常是有多种解决方案的。鼓励学生探索多种解决方案。

4.2~4.7节

*4.1 (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 int value, the program exits if the input is 0:
1 2 -1 3 0
The number of positives is 3
The number of negatives is 1
The total is 5
The average is 1.25
```



PDG

4.2 (重复加法) 程序清单4-3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。

- 4.3 (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

- 4.4 (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

- 4.5 (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 (注意: 1千克为2.2磅):

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

- 4.6 (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 (注意: 1英里为1.609千米):

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

- **4.7 (财务应用程序: 计算将来的学费) 假设今年某大学的学费为10000美元, 学费的年增长率为5%。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?

- 4.8 (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。

- *4.9 (找出两个最高分) 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, 最后显示获得最高分的学生和第二高分的学生。

- 4.10 (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。

- 4.11 (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。

- 4.12 (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。

- 4.13 (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。

- *4.14 (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。

4.8节

- *4.15 (计算最大公约数) 下面是求两个整数n1和n2的最大公约数的程序清单4-8的另一种解法: 首先找出n1和n2的最小值d, 然后依次检验d, d-1, d-2, ..., 2, 1是否是n1和n2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。

- **4.16 (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。


- **4.17 (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

图案，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
    
```



*4.18 (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

**4.19 (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
    
```

*4.20 (打印2到1000之间的素数) 修改程序清单4-14，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。


综合题

**4.21 (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，每月的支付额和总偿还额。下面是一个运行示例：

Loan Amount: 10000

Number of Years: 5

Interest Rate	Monthly Payment	Total Payment
5%	188.71	11322.74
5.125%	189.28	11357.13
5.25%	189.85	11391.59
...		
7.875%	202.17	12129.97
8.0%	202.76	12165.83



计算月支付额的公式，请参见程序清单2-8。

**4.22 (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
 Number of Years: 1
 Annual Interest Rate: 7%

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);
}
```

*4.23 (获取更精确的结果) 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：

$$1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

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 $n=50000$ 。

*4.24 (数列求和) 编写程序，计算下面数列的和：

$$\frac{1}{3} + \frac{3}{5} + \frac{5}{7} + \frac{7}{9} + \frac{9}{11} + \frac{11}{13} + \cdots + \frac{95}{97} + \frac{97}{99}$$

**4.25 (计算 π) 使用下面的数列可以近似计算 π ：

$$\pi = 4 \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \cdots + \frac{1}{2i-1} - \frac{1}{2i+1} \right)$$

编写程序，显示当 $i=10000, 20000, \dots, 100000$ 时 π 的值。

**4.26 (计算 e) 使用下面的数列可以近似计算 e ：

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdots + \frac{1}{i!}$$

编写程序，显示当 $i=10000, 20000, \dots, 100000$ 时 e 的值。

提示 由于 $i! = i \times (i-1) \times \cdots \times 2 \times 1$ ，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 $i=2, 3, 4, \dots$ 。

**4.27 (显示闰年) 编写程序，显示21世纪(2001年~2100年)中所有的闰年，每行显示10个。

**4.28 (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2005和代表2005年1月1日为星期六的6，程序应该显示如下输出(注意：0表示星期天)：

January 1, 2005 is Saturday

...

December 1, 2005 is Thursday

****4.29** (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2005和代表2005年1月1日为星期六的6, 程序应该显示该年每个月的日历, 如下所示:

January 2005						
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 2005						
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

***4.30** (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。

***4.31** (财务应用程序: 计算CD价值) 假设你投资10 000美元买一张CD, 年获利率为5.75%。一个月后, 这张CD价值为

$$10000 + 10000 * 5.75 / 1200 = 10047.91$$

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

$$10096.06 + 10096.06 * 5.75 / 1200 = 10144.43$$

依此类推。

编写程序, 提示用户输入一个总数 (例如: 10000)、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 annual percentage yield: 5.75
Enter maturity period (number of months): 18

Month      CD Value
1          10047.91
2          10096.06
...
17         10846.56
18         10898.54

```



****4.32 (游戏: 彩票)** 修改程序清单3-9, 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。

****4.33 (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 $28=1+2+4+7+14$ 。10000以下的完全数有四个。编写程序, 找出这四个完全数。

*****4.34 (游戏: 石头、剪刀、布)** 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连续赢两次以上为止。

***4.35 (加法)** 编写程序, 计算下面的和。

$$\frac{1}{1+\sqrt{2}} + \frac{1}{\sqrt{2}+\sqrt{3}} + \frac{1}{\sqrt{3}+\sqrt{4}} + \cdots + \frac{1}{\sqrt{624}+\sqrt{625}}$$

****4.36 (商业应用程序: 检测ISBN)** 使用循环简化练习题3.19。

****4.37 (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.toBinaryString(int)`方法。

****4.38 (十进制到十六进制)** 编写程序, 提示用户输入一个十进制整数, 然后显示对应的十六进制值。在这个程序中不要使用Java的`Integer.toHexString(int)`方法。

***4.39 (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美金。使用下面的方案确定你的提成率。

销售额	提成率
0.01 ~ 5000美金	8%
5000.01 ~ 10 000美金	10%
10 000.01及以上	12%

你的目标是一年挣30 000美金。编写程序找出为挣到30 000美金, 你所必须完成的最小销售额。

4.40 (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。

****4.41 (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

```



*4.42 (财务应用程序: 求出销售额) 如下改写练习题4.39:

(1) 使用for循环替代do-while循环。

(2) 允许用户自己输入COMMISSION_SOUGHT而不是将它固定为一个常量。

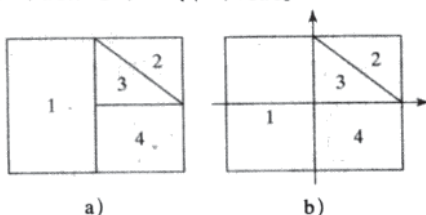
*4.43 (模拟: 表的倒计时) 编写程序, 提示用户输入秒数, 每一秒都显示一条消息, 直到时间用完时结束程序。下面是一个运行示例:

```
Enter the number of second: 3
2 seconds remaining
1 second remaining
Stopped
```



**4.44 (蒙特卡罗模拟) 一个正方形被分为更小的四部分, 如下面的图a中所示。如果将一个飞镖掷入这个正方形1 000 000次, 那么这个飞镖掷入奇数标记的区域的概率有多大? 编写程序模拟这个过程, 然后显示结果。

提示 将这个正方形的中点放在直角坐标系的原点, 如图b所示。随机产生一个正方形中的点, 然后计算这个点落在奇数标记的区域中的次数。



*4.45 (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, 还要显示所有组合的总数。

```
1 2
1 3
...
...
```



*4.46 (计算机体系结构方面: 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 需要使用按位右移运算符(>>)以及按位AND运算符(&), 它们都放在配套网站上的补充材料III.D中。

方 法

学习目标

- 定义方法 (5.2节)。
- 调用带返回值的方法 (5.3节)。
- 调用无返回值的方法 (5.4节)。
- 按值传参 (5.5节)。
- 开发模块化的、易读、易调试和易维护的可重用代码 (5.6节)。
- 编写方法将十进制数转换为十六进制数 (5.7节)。
- 使用方法重载, 理解歧义重载 (5.8节)。
- 确定变量的作用域 (5.9节)。
- 使用Math类中的方法解决数学问题 (5.10~5.11节)。
- 在软件开发中应用方法抽象的概念 (5.12节)。
- 使用逐步求精的办法设计和实现方法 (5.12节)。

5.1 引言

假如需要分别求出从1到10、从20到30以及从35到45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 <= 30; i++)
    sum += i;
System.out.println("Sum from 20 to 30 is " + sum);

sum = 0;
for (int i = 35; i <= 45; i++)
    sum += i;
System.out.println("Sum from 35 to 45 is " + sum);
```

你会发现计算从1到10、从20到30以及从35到45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? 可以通过定义方法实现该功能。方法就是用来创建可重用的代码。

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 sum = 0;
3     for (int i = i1; i <= i2; i++)
4         sum += i;
5
6     return sum;
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 30 is " + sum(20, 30));
12    System.out.println("Sum from 35 to 45 is " + sum(35, 45));
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(20,30)计算从20到30的整数和，而sum(35,45)计算从35到45的整数和。

方法是完成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、JOptionPane.showMessageDialog、JOptionPane.showInputDialog、Integer.parseInt、Double.parseDouble、System.exit、Math.pow和Math.random，这些方法都在Java库中定义。在本章里，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。

5.2 定义方法

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创建的方法，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5-1解释了这个方法的组成。

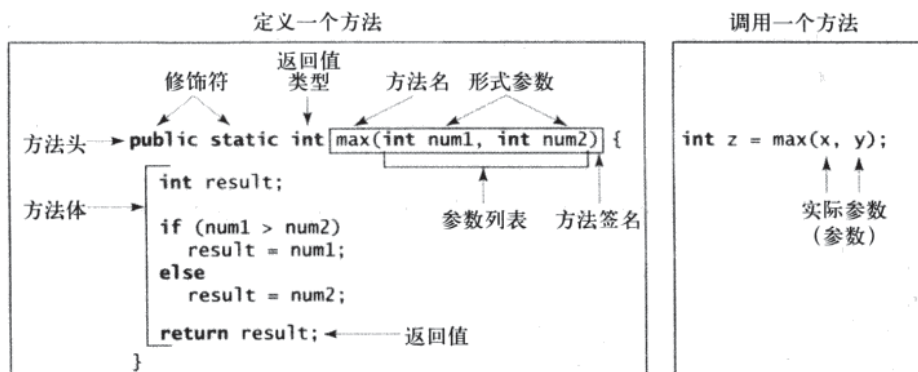


图5-1 方法定义包括方法头和方法体

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，使用它的理由将在第8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和JOptionPane.showMessageDialog方法中返回值类型也是如此。如果方法有返回值，则称之为带返回值的方法（value-returning method），否则就称这个方法为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。

方法体中包含一个定义方法做什么的语句集合。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语句意味着方法终止。

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。

5.3 调用方法

在创建方法时，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还给调用者。

程序清单5-1给出了测试`max`方法的完整程序。

程序清单5-1 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 between " + i +
8             " and " + j + " is " + k);
9     }
10
11     /** Return the max between 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 between 5 and 2 is 5



数字图书馆
PDG

	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方法与其他方法的唯一区别在于它是由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6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方法调用与main方法在同一个类中定义的方法max(i,j)。

当调用max方法时（第6行），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5-2所示。

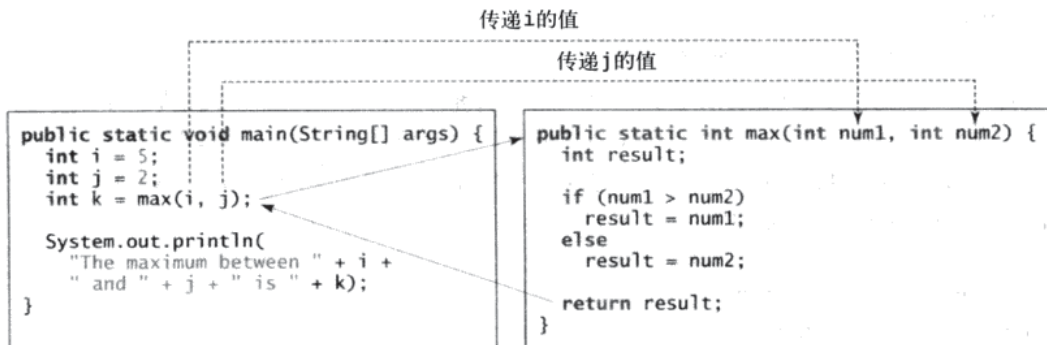
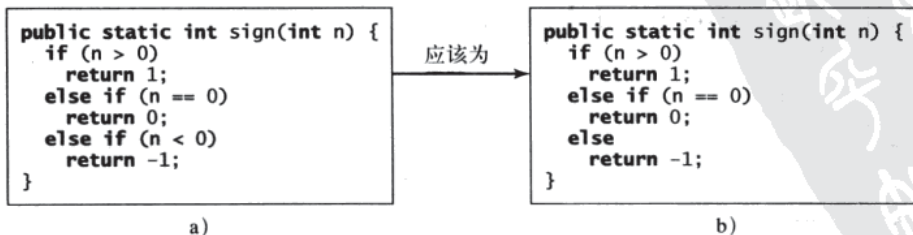


图5-2 当调用max方法时，控制流程转向max方法。一旦max方法结束，将控制返还给调用者

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。

为了解决这个问题，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。

调用堆栈

每当调用一个方法时，系统都会将参数、局部变量存储在一个称为堆栈（stack）的内存区域中，它用后进先出的方式存储数据。当一个方法调用另一个方法时，调用者的堆栈空间保持不动，新开辟的空间处理新方法的调用。一个方法结束返回到调用者时，其相应的空间也被释放。

理解调用堆栈有助于理解方法是如何调用的。`main`方法中定义的变量是*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*都是方法的参数。它们的值通过方法调用进行传递。图5-3描述了堆栈中的变量。

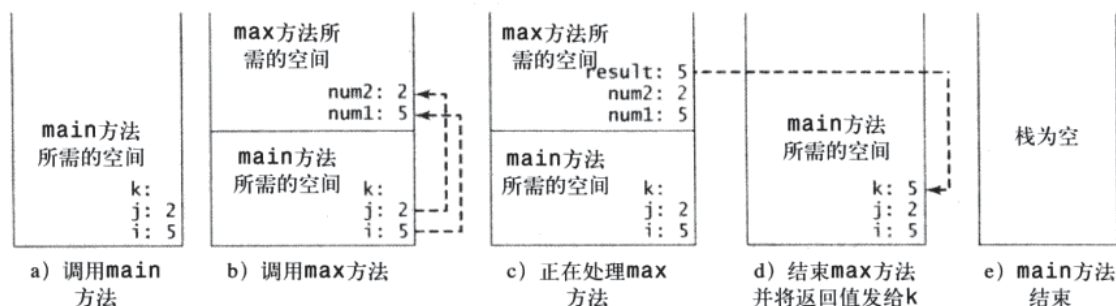


图5-3 调用`max`方法时，程序控制流转到`max`方法。一旦`max`方法结束，就将程序控制还给调用者

5.4 void方法举例

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5-2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5-2 TestVoidMethod.java

```

1 public class TestVoidMethod {
2     public static void main(String[] args) {
3         System.out.print("The grade is ");
4         getGrade(78.5);
5
6         System.out.print("The grade is ");
7         getGrade(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



PDF

`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`，返回等级，如程序清单5-3所示。

程序清单5-3 `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，它也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`语句结束函数。

```

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');
    }
}

```

数字图书馆
PDG

5.5 参数的值传递

方法的威力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")就是错的。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,这个过程称为通过值传递 (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5-4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的值是不论方法做了什么都保持不变。

程序清单5-4 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5-5给出另一个演示值传递效果的程序。程序创建了一个能实现两个变量互换的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5-5 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         + " 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         + " 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 n2 is 2
 After swapping n1 is 2 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5-4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。

另一个转变是把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。

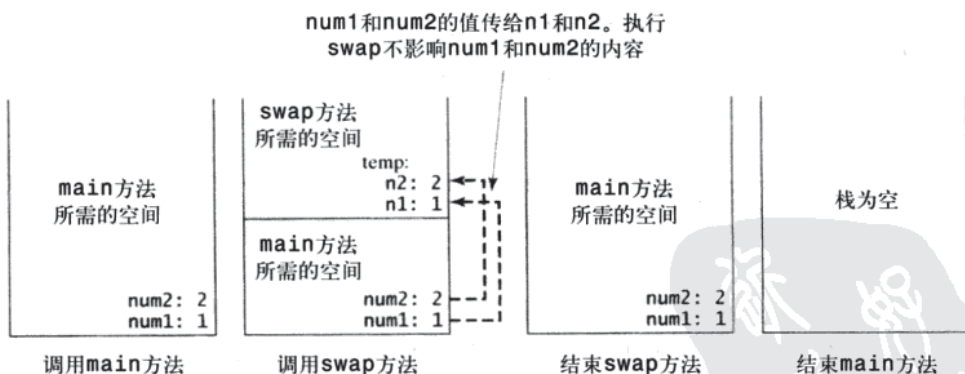


图5-4 变量的值传递给方法中的形参

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。

5.6 模块化代码

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4-8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。可以使用一种方法改写这个程序，如程序清单5-6所示。

程序清单5-6 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 second integer: 75
 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5-7应用了代码模块化的概念来对程序清单4-14进行改进。

程序清单5-7 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) {
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，这样，新的程序会更易读，也更易于调试。而且，其他程序也可以使用方法 `printPrimeNumbers` 和 `isPrime`。

5.7 问题：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。附录F介绍数字系统。本节介绍将十进制数转换为十六进制数的程序。

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 。

例如：十进制数123被转换为十六进制数7B。这个转换过程如下所示：

$$\begin{array}{r}
 \begin{array}{r}
 0 \\
 16 \overline{) 123} \\
 \underline{0} \\
 7 \\
 \underline{0} \\
 7
 \end{array}
 \qquad
 \begin{array}{r}
 7 \\
 16 \overline{) 123} \\
 \underline{112} \\
 11
 \end{array}
 \end{array}$$

\downarrow 商
 \downarrow 余数

h_1
 h_0


程序清单5-8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5-8 **Decimal2HexConversion.java**

```

1 import java.util.Scanner;
2
3 public class Decimal2HexConversion {
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        System.out.println("The hex number for decimal " +
14            decimal + " is " + decimalToHex(decimal) );
15    }
16
17    /** Convert a decimal to a hex as a string */
18    public static String decimalToHex(int decimal) {
19        String hex = "";
20
21        while (decimal != 0) {
22            int hexValue = decimal % 16;
23            hex = toHexChar(hexValue) + hex;
24            decimal = decimal / 16;
25        }
26
27        return hex;
28    }
29
30    /** Convert an integer to a single hex digit in a character */
31    public static char toHexChar(int hexValue) {
32        if (hexValue <= 9 && hexValue >= 0)
33            return (char)(hexValue + '0');
34        else // hexValue <= 15 && hexValue >= 10
35            return (char)(hexValue - 10 + 'A');
36    }
37 }

```

Enter a decimal number: 1234 
The hex number for decimal 1234 is 4D2



	line#	decimal	hex	hexValue	toHexChar(hexValue)
	19	1234	""		
iteration 1 {	22			2	
	23		"2"		2
	24	77			
iteration 2 {	22			13	
	23		"D2"		D
	24	4			
iteration 3 {	22			4	
	23		"4D2"		4
	24	0			

程序使用`decimalToHex`方法（第18~28行）将一个十进制整数转换为一个字符串形式的十六进制数。该方法得到这个十进制整数整除16之后的余数（第22行）。通过调用`toHexChar`方法将得到的余数转换为字符串（第23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19行）。对这个十进制数除以16，就从该数中去掉一个十六进制数字（第24行）。方法在一个循环中重复执行这些操作，直到商是0为止（第21~25行）。

方法`toHexChar`（第31~36行）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33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统一码。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+'A')`（第35行）。例如，如果`hexValue`是11，那么`(char)(hexValue+'A')`返回'B'。

5.8 重载方法

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5-9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。所有这三个方法都被命名为`max`。

程序清单5-9 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 between 3 and 4 is "
6             + max(3, 4));
7
8         // Invoke the max method with the double parameters
9         System.out.println("The maximum between 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 between 3.0, 5.4, and 10.14 is "
14            + max(3.0, 5.4, 10.14));
15    }
16
17    /** Return the max between 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 between 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 among 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 between 3 and 4 is 4
 The maximum between 3.0 and 5.4 is 5.4
 The maximum between 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)`。

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。

PDF

5.9 变量的作用域

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。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5-5所示。

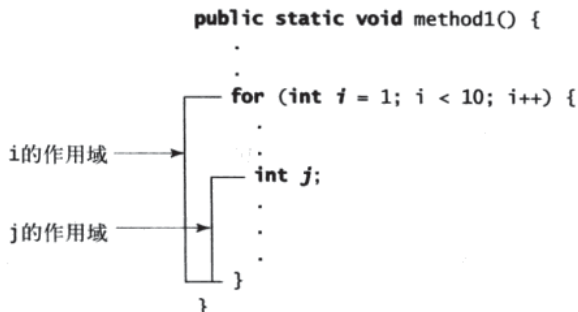


图5-5 在for循环头中初始动作部分声明的变量, 其作用域是整个for循环

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5-6所示。

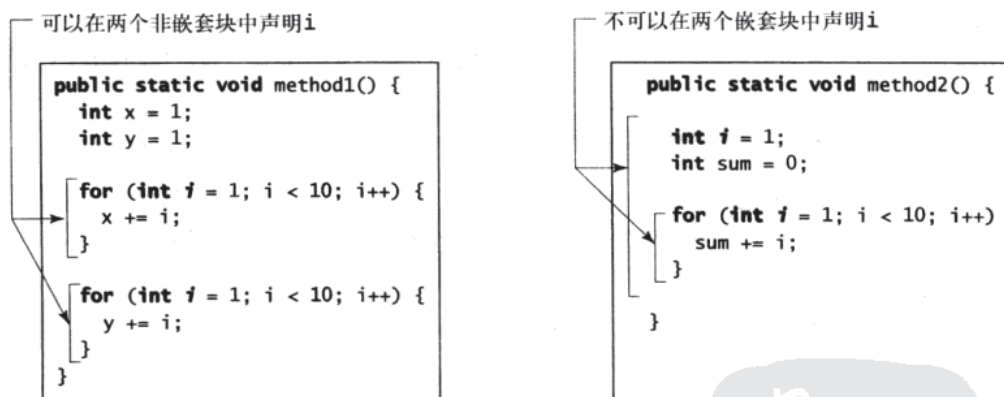


图5-6 一个变量可以在非嵌套的块中多次声明, 而在嵌套块中只能声明一次

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。

5.10 Math数学类

Math类包含完成基本数学函数所需的方法。在程序清单2-8中已经使用过方法`pow(a,b)`计算 a^b , 在程序清单3-4中也使用过方法`Math.random()`。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)。

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和E（自然对数的底）。可以在任意程序中用Math.PI和Math.E的形式来使用这两个常量。

5.10.1 三角函数方法

Math类包含下面的三角函数方法：

```
/** Return the trigonometric sine of an angle in radians */  
public static double sin(double radians)  
  
/** Return the trigonometric cosine of an angle in radians */  
public static double cos(double radians)  
  
/** Return the trigonometric tangent of an angle in radians */  
public static double tan(double radians)  
  
/** Convert the angle in degrees to an angle in radians */  
public static double toRadians(double degree)  
  
/** Convert the angle in radians to an angle in degrees */  
public static double toDegrees(double radians)  
  
/** Return the angle in radians for the inverse of sin */  
public static double asin(double a)  
  
/** Return the angle in radians for the inverse of cos */  
public static double acos(double a)  
  
/** Return the angle in radians for the inverse of tan */  
public static double atan(double a)
```

sin、cos和tan的参数都是以弧度为单位的角。asin、acos和atan的返回值是在 $-\pi/2$ 到 $\pi/2$ 之间的一个弧度值。1度相当于 $\pi/180$ 弧度，90度相当于 $\pi/2$ 弧度，而30度相当于 $\pi/6$ 弧度。

例如：

```
Math.toDegrees(Math.PI / 2) returns 90.0  
Math.toRadians(30) returns  $\pi/6$   
Math.sin(0) returns 0.0  
Math.sin(Math.toRadians(270)) returns -1.0  
Math.sin(Math.PI / 6) returns 0.5  
Math.sin(Math.PI / 2) returns 1.0  
Math.cos(0) returns 1.0  
Math.cos(Math.PI / 6) returns 0.866  
Math.cos(Math.PI / 2) returns 0  
Math.asin(0.5) returns  $\pi/6$ 
```

5.10.2 指数函数方法

Math类中有五个与指数函数有关的方法：

```
/** Return e raised to the power of x ( $e^x$ ) */  
public static double exp(double x)  
  
/** Return the natural logarithm of x ( $\ln(x) = \log_e(x)$ ) */  
public static double log(double x)  
  
/** Return the base 10 logarithm of x ( $\log_{10}(x)$ ) */  
public static double log10(double x)  
  
/** Return a raised to the power of b ( $a^b$ ) */  
public static double pow(double a, double b)  
  
/** Return the square root of x ( $\sqrt{x}$ ) for  $x \geq 0$  */  
public static double sqrt(double x)
```

例如：



```

Math.exp(1) returns 2.71828
Math.log(Math.E) returns 1.0
Math.log10(10) returns 1.0
Math.pow(2, 3) returns 8.0
Math.pow(3, 2) returns 9.0
Math.pow(3.5, 2.5) returns 22.91765
Math.sqrt(4) returns 2.0
Math.sqrt(10.5) returns 3.24

```

5.10.3 取整方法

Math类包括五个取整方法:

```

/** x is rounded up to its nearest integer. This integer is
 * returned as a double value. */
public static double ceil(double x)

/** x is rounded down to its nearest integer. This integer is
 * returned as a double value. */
public static double floor(double x)

/** x is rounded to its nearest integer. If x is equally close
 * to two integers, the even one is returned as a double. */
public static double rint(double x)

/** Return (int)Math.floor(x + 0.5). */
public static int round(float x)

/** Return (long)Math.floor(x + 0.5). */
public static long round(double x)

```

例如:

```

Math.ceil(2.1) returns 3.0
Math.ceil(2.0) returns 2.0
Math.ceil(-2.0) returns -2.0
Math.ceil(-2.1) returns -2.0
Math.floor(2.1) returns 2.0
Math.floor(2.0) returns 2.0
Math.floor(-2.0) returns -2.0
Math.floor(-2.1) returns -3.0
Math.rint(2.1) returns 2.0
Math.rint(-2.0) returns -2.0
Math.rint(-2.1) returns -2.0
Math.rint(2.5) returns 2.0
Math.rint(3.5) returns 4.0
Math.rint(-2.5) returns -2.0
Math.round(2.6f) returns 3 // Returns int
Math.round(2.0) returns 2 // Returns long
Math.round(-2.0f) returns -2
Math.round(-2.6) returns -3

```

5.10.4 min、max和abs方法

重载min和max方法以返回两个数(int、long、float或double型)的最小值和最大值。例如, max(3.4,5.0)返回5.0, 而min(3,2)返回2。

重载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) returns 3
Math.max(2.5, 3) returns 3.0
Math.min(2.5, 3.6) returns 2.5
Math.abs(-2) returns 2
Math.abs(-2.1) returns 2.1

```


5.10.5 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 ($0.0 \leq \text{Math.random()} < 1.0$)。这个方法十分有用。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到9之间的一个随机整数

`50 + (int) (Math.random() * 50)` → 返回50到99之间的一个随机整数

通常,

`a + Math.random() * b` → 返回一个在a到a+b之间但不包括a+b的随机数

提示 可以在网站<http://java.sun.com/javase/6/docs/api/index.html>上在线查阅Math类的完整文档,如图5-7所示。

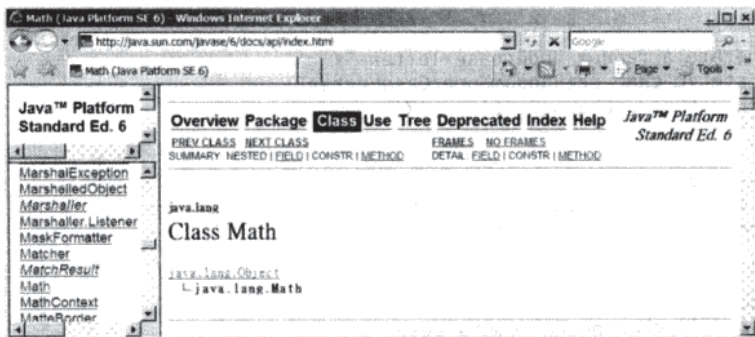


图5-7 可以在线查看Java API文档

注意 并非所有的类都需要`main`方法。`Math`类和`JOptionPane`类都没有`main`方法。这些类中所包含的方法主要是为了供其他类使用。

5.11 实例学习：生成随机字符

计算机程序处理的是数值数据和字符。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2.13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65 535)之间的统一码。生成一个随机字符就是使用下面的表达式,生成从0到65 535之间的一个随机整数(注意:因为 $0 \leq \text{Math.random()} < 1.0$,必须给65 535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。小写字母的统一码是一串连续的整数,从小写字母'a'的统一码开始,然后是'b'、'c'、...和'z'的统一码。'a'的统一码是:

`(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2.13.3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))`

推广前面的讨论,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5-10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5-10 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5-11给出一个测试程序,显示175个随机的小写字母。

程序清单5-11 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 }
```

```
gmjssohezfkg tazqgmswfc lrao
pnrunu lnw mazt l fjedmpchcif
la lqdgiv xkxpbzu lrmqbhikr
lbnrjlsopfxahssqh wuuljvbe
xbhdotzhpehbqmuwsfktwsoli
cbuwkzgxpm tzi h gatds l vwbwz
bfesok lwbhnooygi gzxuqni
```



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。

5.12 方法抽象和逐步求精

开发软件的关键在于应用抽象的概念。从本书中将学习到多种层次的抽象。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“黑匣子”中,如图5-8所示。

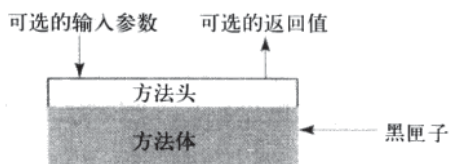


图5-8 方法体可以看做是一个包括该方法实现细节的黑匣子

前面已经使用过方法System.out.print来显示一个字符串,使用方法JOptionPane.showInputDialog从对话框中读入一个字符串,然后用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”(divide-and-conquer)策略,也称之为逐步求精(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., 2001):

Enter month in number between 1 and 12:

June 2006

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。

5.12.1 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,只在最后才实现这些细节。

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。在这一步,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5-9a)。

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5-9b所示。日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5-10a）。

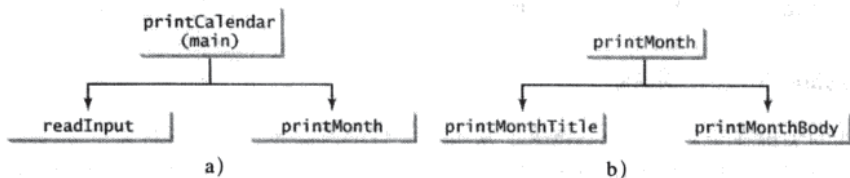


图5-9 结构图显示将打印日历printCalendar问题分解成两个子问题——读取输入readInput和打印日历printMonth，而将printMonth分解成两个更小的问题——打印日历头printMonthTitle和打印日历体printMonthBody

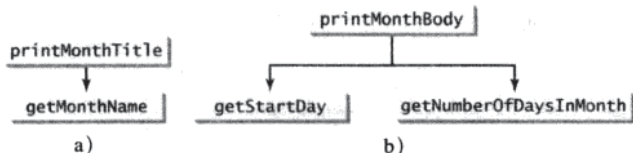


图5-10 a) 需要getMonthName才能完成printMonthTitle,
b) printMonthBody被细化成几个更小的问题

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5-10b所示。例如：2005年12月有31天，2005年12月1号是星期四。

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（startDay 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 $(totalNumberOfDays + startDay 1800) \% 7$ 。这样getStartDay问题就可以进一步细化为getTotalNumberOfDays，如图5-11a所示。

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5-11b所示。完整的结构图如图5-12所示。

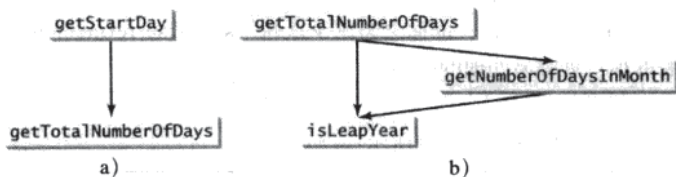


图5-11 a) 需要getTotalNumberOfDays才能得到getStartDay,
b) 问题getTotalNumberOfDays可以细化成两个更小的问题

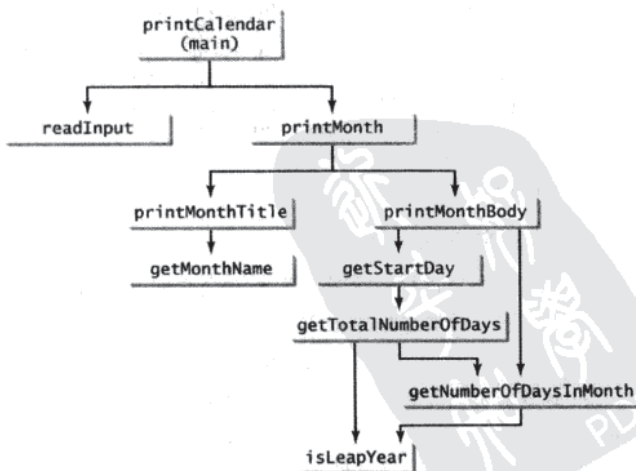


图5-12 结构图显示程序中子问题之间的层次关系

5.12.2 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，而哪些模块要结合在另一个方法中。这种决策应该基于所做的选择，整个程序是否更易读做出的。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., 2001): ");
        int year = input.nextInt();

        // Prompt the user to enter month
        System.out.print("Enter month as 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 getMonthName 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 getTotalNumberOfDays 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。

5.12.3 实现细节

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，然后求出在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5-12。

程序清单5-12 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., 2001): ");
10        int year = input.nextInt();
11
12        // Prompt the user to enter month
13        System.out.print("Enter month in 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., May, 1999 */
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");

```

数字图书馆
PDG

```

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，以便能够追溯到1800年之前的月份。

注意 方法抽象将程序模块化为整齐、层次明显的形式。将程序写成由简洁的方法构成的集合，比其他方式更容易编写、调试、维护和修改。这样的编写风格也会提高方法的可重用性。

提示 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。

关键术语

actual parameter (实际参数即实参)

argument (实参)

ambiguous invocation (歧义调用)

divide and conquer (分治)

formal parameter (ie.parameter) (形式参数即形参)

information hiding (信息隐藏)

method (方法)

method abstraction (方法抽象)

method overloading (方法重载)

method signature (方法签名)

modifier (修饰符)

pass-by-value (按值传递)

parameter (参数)

return type (返回值类型)

return value (返回值)

scope of variable (变量的作用域)

stepwise refinement (逐步求精)

stub (待完善方法)

本章小结

- 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。
- 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。
- 方法可以返回一个值。返回值类型returnValueType是方法要返回的值的数据类型。如果方法不返回值，则返回值类型就是关键字void。
- 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。
- 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，用于偶尔改变正常流程控制是很有用的。
- 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。
- 当程序调用一个方法时，程序控制就转移到被调用的方法。当执行到该方法的return语句或到达方法结束的右括号时，被调用的方法将程序控制还给调用者。
- 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。
- 每次调用一个方法时，系统都会将参数和局部变量存储在一个称为堆栈（stack）的区域中。当一个方法调用另一个方法时，调用者的堆栈空间保持不动，开辟新的空间处理新方法的调用。一个方法完成它的工作之后返回到它的调用者时，就释放其相应的空间。
- 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。
- 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。
- 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。这就称为信息隐藏或封装。
- 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。
- 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。

复习题

5.2~5.4节

- 5.1 使用方法的优点有哪些？如何定义一个方法？如何调用一个方法？
- 5.2 main方法的返回类型是什么？
- 5.3 可以使用条件运算符简化程序清单5-1中的max方法吗？
- 5.4 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总是表达式的一部分。
- 5.5 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;  
}
```

- 5.6 给出术语形参、实参和方法签名的定义。
- 5.7 写出下列方法的方法头：

- 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。
- 按指定次数打印一条信息。
- 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。

5.8 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 }

```

5.9 根据2.16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 method1(double i, double j)
    {
        while (i < j) {
            j--;
        }

        return j;
    }
}

```

5.5~5.7节

5.10 实参是如何传递给方法的？实参可以和形参同名吗？

5.11 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)

5.12 在前面问题的图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。

5.8节

5.13 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？

5.14 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;
    }
}

```

5.9节

5.15 指出并改正下面程序中的错误：

```

1 public class Test {
2     public static void main(String[] args) {
3         nPrintln("Welcome to Java!", 5);
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 }

```

5.10节

5.16 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。

5.17 编写一个表达式，返回34到55之间的一个随机整数。编写一个表达式，返回0到999之间的一个随机整数。编写一个表达式，返回5.5到55.5之间的一个随机数。编写一个表达式，返回任意一个小写字母。

5.18 计算调用下列方法后的结果：

- (1) `Math.sqrt(4)`
- (2) `Math.sin(2*Math.PI)`
- (3) `Math.cos(2*Math.PI)`
- (4) `Math.pow(2,2)`
- (5) `Math.log(Math.E)`
- (6) `Math.exp(1)`
- (7) `Math.max(2,Math.min(3,4))`
- (8) `Math rint(-2.5)`
- (9) `Math.ceil(-2.5)`
- (10) `Math.floor(-2.5)`
- (11) `Math.round(-2.5F)`
- (12) `Math.round(-2.5)`
- (13) `Math.rint(2.5)`
- (14) `Math.ceil(2.5)`
- (15) `Math.floor(2.5)`
- (16) `Math.round(2.5F)`
- (17) `Math.round(2.5)`
- (18) `Math.round(Math.abs(-2.5))`

编程练习题

5.2~5.9节

5.1 (数学方面: 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。

*5.2 (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:

```
public static int sumDigits(long n)
```

例如: `sumDigits(234)` 返回`9(2+3+4)`。

提示 使用求余运算符`%`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。

**5.3 (回文整数) 编写下面两个方法:

```
// Return the reversal of an integer, i.e. reverse(456) returns 654
public static int reverse(int number)
```

```
// Return true if number is 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。

*5.4 (反向显示一个整数) 编写下面的方法, 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。

*5.5 (对三个数排序) 编写下面的方法,按升序显示三个数:

```
public static void displaySortedNumbers(  
    double num1, double num2, double num3)
```

*5.6 (显示模式) 编写方法显示如下模式:

```
      1  
     2 1  
    3 2 1  
    ...  
n n-1 ... 3 2 1
```

方法头为:

```
public static void displayPattern(int n)
```

*5.7 (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值,未来投资是用练习题2.13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
```

5.8 (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:

```
/** Converts from Celsius to Fahrenheit */  
public static double celsiusToFahrenheit(double celsius)  
  
/** Converts from Fahrenheit to Celsius */  
public static double fahrenheitToCelsius(double fahrenheit)
```

转换公式如下:

$$\text{华氏度} = (9.0 / 5) * \text{摄氏度} + 32$$

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

5.9 (英尺和米之间的转换) 编写一个类,包含如下两个方法:

```
/** Converts from feet to meters */  
public static double footToMeter(double foot)  
  
/** Converts from meters to feet */  
public static double meterToFoot(double meter)
```

转换公式如下:



$$1\text{米} = 0.305 \times \text{英尺数}$$

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	0.61	25.0	81.967
...			
9.0	2.745	60.0	196.721
10.0	3.05	65.0	213.115

5.10 (使用isPrime方法) 程序清单5-7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。

5.11 (财务应用程序：计算佣金) 编写一个方法，利用练习4.39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	佣金
10000	900.0
15000	1500.0
...	
95000	11100.0
100000	11700.0

5.12 (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。

*5.13 (数列求和) 编写一个方法计算下列级数：

$$m(i) = \frac{1}{2} + \frac{2}{3} + \cdots + \frac{i}{i+1}$$

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

*5.14 (计算数列) 编写一个方法计算下面的数列：

$$m(i) = 4 \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \cdots + \frac{1}{2i-1} - \frac{1}{2i+1} \right)$$

编写一个测试程序显示下面的表格：

i	m(i)
10	3.04184
20	3.09162
...	
90	3.13048
100	3.13159

*5.15 (财务应用程序：打印税表) 程序清单3-6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美元的所有四种纳税人的纳税表，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

*5.16 (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2010年每年的天数。

5.10~5.11节

*5.17 (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, 打印如下所示的 3×3 矩阵:

```
0 1 0
0 0 0
1 1 1
```

5.18 (使用Math.sqrt方法) 使用Math类中的sqrt方法编写程序, 打印如下表格:

数字	平方根
0	0.0000
2	1.4142
...	
18	4.2426
20	4.4721

*5.19 (MyTriangle类) 创建一个名为MyTriangle的类, 它包含如下两个方法:

```
/** Returns true if the sum of any two sides is
 * greater than the third side. */
public static boolean isValid(
    double side1, double side2, double side3)
```

```
/** Returns 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练习题2.21中给出。

5.20 (使用三角函数方法) 打印下面的表格, 显示 0° 到 360° 之间, 角度间隔为 10° 的正弦和余弦值, 保留小数点后4位。

角度	正弦值	余弦值
0	0.0000	1.0000
10	0.1736	0.9848
...		
350	-0.1736	0.9848
360	0.0000	1.0000

**5.21 (统计学方面: 计算均值和标准差) 在商业应用中, 经常需要计算数据的均值和标准差。均值就是这些数据的平均值。标准差是一个统计数据, 它会告诉你数据集之中的数据距离平均值的远近程度。例如: 一个班学生的平均年龄是多少? 这些年龄的集中程度如何? 如果所有学生的年龄相同, 则标准差为0。编写一个程序, 提示用户输入十个数字, 然后使用下面的公式显示它们的均值和标准差:

$$\text{均值} = \frac{\sum_{i=1}^n x_i}{n} = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

$$\text{标准差} = \sqrt{\frac{\sum_{i=1}^n x_i^2 - \frac{\left(\sum_{i=1}^n x_i\right)^2}{n}}{n-1}}$$

下面是一个运行示例的结果：

```
Enter ten numbers: 1 2 3 4.5 5.6 6 7 8 9 10 Enter
The mean is 5.61
The standard deviation is 2.99794
```



****5.22** (数学方面：平方根的近似求法) 实现sqrt方法。数num的平方根可以通过对下面公式的反复计算近似地得到：

```
nextGuess = (lastGuess + (num / 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num平方根的近似值；否则，nextGuess就成为lastGuess，近似过程持续执行。

5.10~5.11节

***5.23** (生成随机字符) 使用程序清单5-10中RandomCharacter里的方法，打印100个大写字母和100个一位数，每行打印10个。

****5.24** (显示当前日期和时间) 程序清单2-9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5-12中的日历例子应该会给你提供一些如何求年、月和日的思路。

****5.25** (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

****5.26** (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，如下所示：

```
  2    3    5    7   11   101   131   151   181   191
313  353  373  383  727   757   787   797   919   929
...
```

****5.27** (反素数) 反素数(逆向拼写的素数)是指一个将其逆向之后也是一个素数的非回文素数。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个素数。每行显示10个，并且准确对齐，如下所示：

```
 13  17  31  37  71  73  79  97 107 113
149 157 167 179 199 311 337 347 359 389
...
```

****5.28** (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      2p - 1
 2         3
 3         7
 5        31
...
```

****5.29** (游戏：掷骰子游戏) 掷骰子游戏是赌场中非常流行的骰子游戏。编写程序，玩这个游戏的另一种玩法，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), 你就输了; 如果和是7或者11 (称作自然), 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



****5.30 (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)

...

****5.31 (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, 那么就将这两位加在一起得到一位数。

$$2*2=4 \qquad 6*2=12 \ (1+2=3)$$

$$2*2=4 \qquad 5*2=10 \ (1+0=1)$$

$$4*2=8 \qquad 8*2=16 \ (1+6=7)$$

$$1*2=2 \qquad 4*2=8$$

(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)
```

**5.32 (游戏: 赢取双骰子赌博游戏的机会) 修改练习题5.29使该程序运行10000次, 然后显示赢得游戏的次数。

***5.33 (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, 2009 10:34:23



**5.34 (打印日历) 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5-12以获得每月开始的第一天是星期几。

5.35 (几何问题: 五边形的面积) 使用下面的公式计算五边形的面积:

$$\text{面积} = \frac{5 \times s^2}{4 \times \tan\left(\frac{\pi}{5}\right)}$$

编写程序, 提示用户输入五边形的边, 然后显示它的面积。

*5.36 (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, 它的每个边的长度都相等, 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。

一维数组

学习目标

- 描述数组在程序设计中的必要性 (6.1节)。
- 声明数组引用变量、创建数组 (6.2.1~6.2.2节)。
- 初始化数组中的值 (6.2.3节)。
- 使用下标变量访问数组元素 (6.2.4节)。
- 利用一条数组初始化语法声明、创建和初始化数组 (6.2.5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, 随意打乱, 移动元素) (6.2.6节)。
- 使用for-each循环简化程序设计 (6.2.7节)。
- 在LottoNumbers和DeckOfCards问题中应用数组 (6.3~6.4节)。
- 将一个数组的内容复制到另一个数组 (6.5节)。
- 开发和调用带数组参数和数组返回值的方法 (6.6~6.7节)。
- 定义带变长参数列表的方法 (6.8节)。
- 使用线性查找算法 (6.9.1节) 或二分查找算法 (6.9.2节) 查找数组的元素。
- 使用选择排序法对数组排序 (6.10.1节)。
- 使用插入排序法对数组排序 (6.10.2节)。
- 使用Arrays类中的方法 (6.11节)。

6.1 引言

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, 然后用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, 必须将这些数全部存储到变量中。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, 通过一个一维数组变量访问它。解决方案看起来就像这样:

```
1 public class AnalyzeNumbers {
2     public static void main(String[] args) {
3         final int NUMBER_OF_ELEMENTS = 100;
4         double[] numbers = new double[NUMBER_OF_ELEMENTS];
5         double sum = 0;
6
7         java.util.Scanner input = new java.util.Scanner(System.in);
8         for (int i = 0; i < NUMBER_OF_ELEMENTS; i++) {
9             System.out.print("Enter a new number: ");
10            numbers[i] = input.nextDouble();
11            sum += numbers[i];
12        }
13
14        double average = sum / NUMBER_OF_ELEMENTS;
15    }
```

```

16    int count = 0; // The number of elements above average
17    for (int i = 0; i < NUMBER_OF_ELEMENTS; i++)
18        if (numbers[i] > average)
19            count++;
20
21    System.out.println("Average is " + average);
22    System.out.println("Number of elements above the average "
23        + count);
24 }
25 }

```

程序的第4行创建了一个有100个元素的数组，第10行将这些数存储在数组中，第11行将每个数加到sum中，并在第14行获得其平均值。然后，将数组中的每个数和平均值进行比较，计算比平均值大的数的个数（第16~19行）。

本章介绍一维数组。第7章将介绍二维数组和多维数组。

6.2 数组的基本知识

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做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。

6.2.1 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语言的风格被Java采纳以适用于C程序员。推荐使用elementType[] arrayRefVar（元素类型[] 数组引用变量）风格。

6.2.2 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：

```
arrayRefVar = new elementType[arraySize]; (数组引用变量=new元素类型[数组大小];)
```

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条语句里，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，下面的代码可以初始化数组：

```
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6-1给出这个数组的全貌。

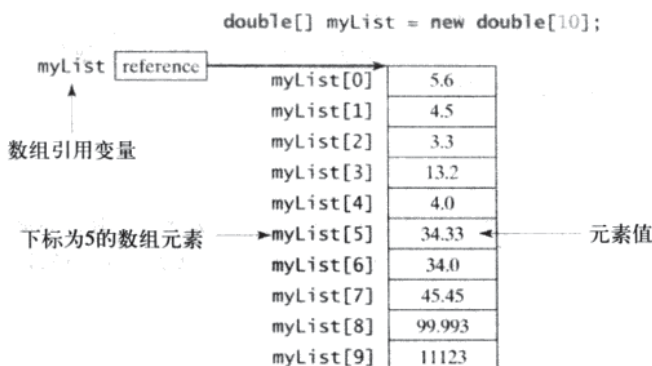


图6-1 数组myList包含10个double型元素并且下标从0到9为int型

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。

6.2.3 数组大小和默认值

当给数组分配空间时，必须通过指定该数组能够存储的元素个数来确定数组大小。创建数组之后就不能再修改它的大小。可以使用arrayRefVar.length求得数组的大小。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。

6.2.4 数组下标变量

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6-1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;
}
```

6.2.5 数组初始化语法

Java有一个简捷的记法，称作数组初始化语法（array initializer），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运算符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};
```

6.2.6 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;
    }
}
```

如果用 (myList[i] >= max) 替换 (myList[i] > max), 那么结果会是什么?

7) (随意打乱) 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 = 0; i < myList.length; i++) {
    // Generate an index j randomly
    int index = (int) (Math.random()
        * myList.length);

    // Swap myList[i] with myList[j]
    double temp = myList[i];
    myList[i] = myList[index];
    myList[index] = temp;
}
```

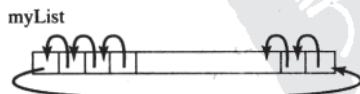


8) (移动元素) 有时候, 需要向左或向右移动元素。这里的例子就是将元素向左移动一个位置并且将第一个元素放在最后一个元素的位置:

```
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;
```



6.2.7 for-each循环

Java支持一个简便的for循环, 称之为for-each循环 (for-each loop) 或增强型for循环 (enhanced for

loop), 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 u: myList) {
    System.out.println(u);
}
```

此代码可以读作“对myList中每个元素u进行以下操作”。注意, 变量u必须声明为与myList中元素相同的数据类型。

通常, for-each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<=时会犯的的错误。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<=。

6.3 问题: 乐透号码

选10乐透的每张筹码都有10个取值范围在1到99的独特数字。假设你买了很多筹码, 希望它们涵盖从1到99的所有数字。编写一个程序, 从文件中读取筹码上的数字, 并且检查是否涵盖所有的数。将文件中最后一个数设定为0。假设文件包含如下数字:

```
80 3 87 62 30 90 10 21 46 27
12 40 83 9 39 88 95 59 20 37
80 40 87 67 31 90 11 24 56 77
11 48 51 42 8 74 1 41 36 53
52 82 16 72 19 70 44 56 29 33
54 64 99 14 23 22 94 79 55 2
60 86 34 4 31 63 84 89 7 78
43 93 97 45 25 38 28 26 85 49
47 65 57 67 73 69 32 71 24 66
92 98 96 77 6 75 17 61 58 13
35 81 18 15 5 68 91 50 76
0
```

程序应该显示:

The tickets cover all numbers

假设文件包含如下数字:

```
11 48 51 42 8 74 1 41 36 53
52 82 16 72 19 70 44 56 29 33
0
```

程序应该显示:

The tickets don't cover all numbers

该如何标定一个数字被涵盖呢? 可以创建一个由99个布尔元素构成的数组。每个数组中的元素都可以用来标定是否涵盖了一个数字。假定这个数组是isCovered。初始状态时, 每个元素都是false, 如



图6-2a所示。当读取一个数时，它对应的元素设置为true。假如输入的数字是1、2、3、99、0。当读取数字1时，isCovered[0]就设置为true（参见图6-2b）。当读取数字2时，isCovered[2-1]就设置为true（参见图6-2c）。当读取数字3时，isCovered[3-1]就设置为true（参见图6-2d）。当读取数字99时，isCovered[99-1]就设置为true（参见图6-2e）。

isCovered	isCovered	isCovered	isCovered	isCovered
[0] false	[0] true	[0] true	[0] true	[0] true
[1] false	[1] false	[1] true	[1] true	[1] true
[2] false	[2] false	[2] false	[2] true	[2] true
[3] false	[3] false	[3] false	[3] false	[3] false
...
[97] false	[97] false	[97] false	[97] false	[97] false
[98] false	[98] false	[98] false	[98] false	[98] true
a)	b)	c)	d)	e)

图6-2 如果在乐透筹码中出现数字*i*，就将isCovered[i-1]设置为true

这个程序的算法可以描述为如下所示：

对于从文件中读出的每个数字*k*，

通过将isCovered[k-1]设置为true将数字*k*标记为涵盖的；

if 每一个isCovered[i]都为true

那么该筹码涵盖所有的数字

else

该筹码并未涵盖所有数字

程序清单6-1给出完整的程序。

程序清单6-1 LottoNumbers.java

```

1 import java.util.Scanner;
2
3 public class LottoNumbers {
4     public static void main(String args[]) {
5         Scanner input = new Scanner(System.in);
6         boolean[] isCovered = new boolean[99]; // Default is false
7
8         // Read each number and mark its corresponding element covered
9         int number = input.nextInt();
10        while (number != 0) {
11            isCovered[number - 1] = true;
12            number = input.nextInt();
13        }
14
15        // Check whether all covered
16        boolean allCovered = true; // Assume all covered initially
17        for (int i = 0; i < 99; i++)
18            if (!isCovered[i]) {
19                allCovered = false; // Find one number not covered
20                break;
21            }
22
23        // Display result
24        if (allCovered)
25            System.out.println("The tickets cover all numbers");
26        else
27            System.out.println("The tickets don't cover all numbers");
28    }
29 }

```

假设已经创建了一个名为LottoNumbers.txt的文本文件，该文件包括输入数值2 5 6 5 4 3 23 43 2 0。可以使用下面的命令运行该程序：

```
java LottoNumbers < LottoNumbers.txt
```

可以如下跟踪这个程序：

这个程序创建了一个由99个布尔元素构成的数组，并且将每个元素初始化为false（第6行）。它从文件中读取第一个数字（第9行）。然后程序在循环中重复下面的操作：

- 如果数字非0，就将数组isCovered中的对应值设置为true（第11行）。
- 读取下一个数字（第12行）。

line	Representative elements in array isCovered						number	allCovered
	[1]	[2]	[3]	[4]	[5]	[22]	[42]	
6	false	false	false	false	false	false	false	
9							2	
11	true							
12							5	
11				true				
12							6	
11					true			
12							5	
11				true				
12							4	
11			true					
12							3	
11		true						
12							23	
11						true		
12							43	
11							true	
12							2	
11	true							
12							0	
16								true
18(i=0)								false

当输入为0时，输入结束。程序第16~21行检测它是否涵盖所有数字，第24~27行显示结果。

6.4 问题：一副牌

这里的问题是要编写一个程序，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6-3所示。在打乱数组deck之后，从deck中选出前四张牌。cardNumber/13决定牌的花色而cardNumver%13决定是具体花色中的哪张牌。

程序清单6-2给出这个问题的解决方案。

程序清单6-2 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 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

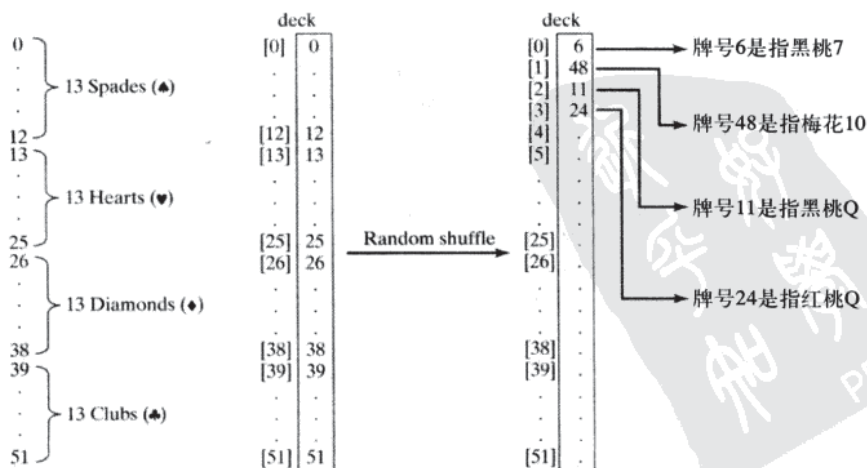


图6-3 52张牌存储在一个名为deck的数组中

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。

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）。

6.5 数组的复制

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，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6-4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。

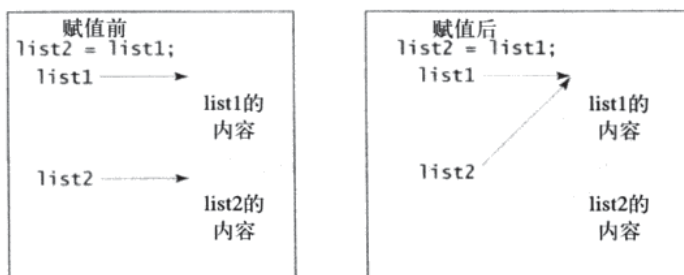


图6-4 赋值语句执行前，list1和list2指向各自的内存地址。

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14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, src_pos, targetArray, tar_pos, length);
```

其中，参数src_pos和tar_pos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)。

6.6 给方法传递数组

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}; (new数据类型[] {值0, 值1, ..., 直接量k};)
```

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会有很大的不同。

- 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
```

将会看到, 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6-5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。



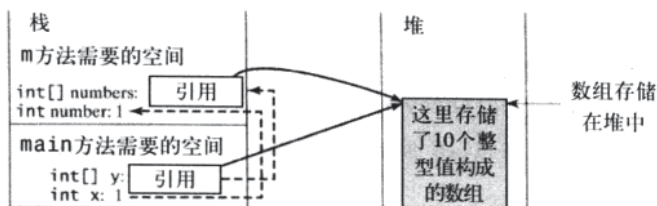


图6-5 x中的基本数据类型值被传递给number，而y中的引用值被传递给numbers

注意 JVM将数组存储在一个称作堆 (heap) 的内存区域中，堆用于动态内存分配，在堆中内存块可以按随意的顺序分配和释放。

传递数组参数

程序清单6-3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对换。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6-3 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6-6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。

swapFirstTwoInArray方法的参数是一个数组。如图6-6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。

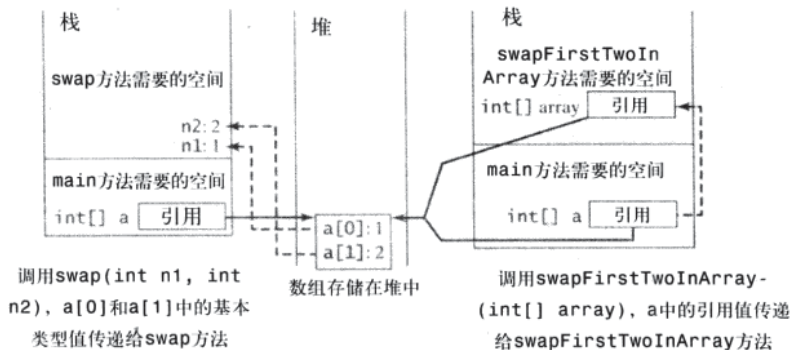
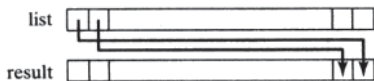


图6-6 将数组传给方法时,传给方法的是数组的引用

6.7 从方法中返回数组

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);
```

案例研究:统计每个字母出现的次数

程序清单6-4中给出一个程序,统计字符数组中每个字母出现的次数。该程序完成下述任务:

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6-7a所示。可以使用程序清单5-10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6-7b所示。也就是说,counts[0]记录a出现的次数,counts[1]记录b出现的次数,依此类推。

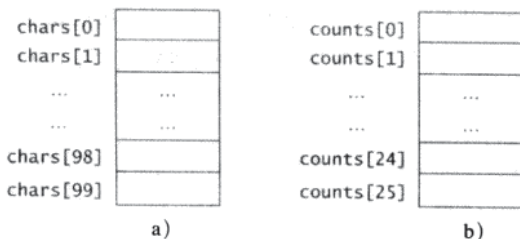


图6-7 数组char存储100个字符，数组counts存储26个计数器，每个计数器对一个字母进行计数

程序清单6-4 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();

```

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5-10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能会如下所示:

```

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'的统一码比'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'的统一码比'a'的大25, 所以它对应的计数器为counts['z' - 'a'] (即counts[25])。

图6-8显示在执行createArray方法的过程中和执行之后调用栈和堆的情况。参见复习题6.14会显示程序中其他方法对栈和堆的调用。

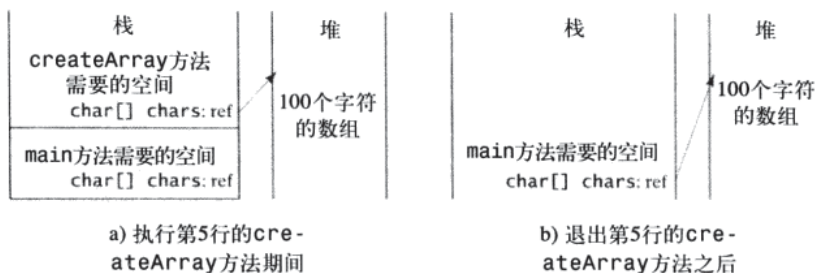


图6-8 a) 执行createArray方法创建100个字符的数组;
b) 在main方法中返回这个数组并赋值给变量chars

6.8 可变长参数列表

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，指定类型后紧跟着省略号 (...)。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。可以将一个数组或可变的参数个数传递给可变长参数。当用可变的参数个数调用方法时，Java会创建一个数组并把参数传给它。程序清单6-5包括了打印出个数不定的数列中最大值的方法。

程序清单6-5 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行将一个可变长参数列表传给数组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行调用一个数组调用printMax方法。

6.9 数组的查找

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）。

6.9.1 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6-6中的linearSearch方法给出解决方案：

程序清单6-6 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 }
```

[0] [1] [2] ...

list

--	--	--	--	--	--

key Compare key with list[i] for i = 0, 1, ...

为了更好地理解这个方法，用下面的语句跟踪这个方法：

```
int[] list = {1, 4, 4, 2, 5, -3, 6, 2};
int i = linearSearch(list, 4); // Returns 1
int j = linearSearch(list, -4); // Returns -1
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。

6.9.2 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6-9显示怎样使用二分法从数列{2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79}中找出关键字11。

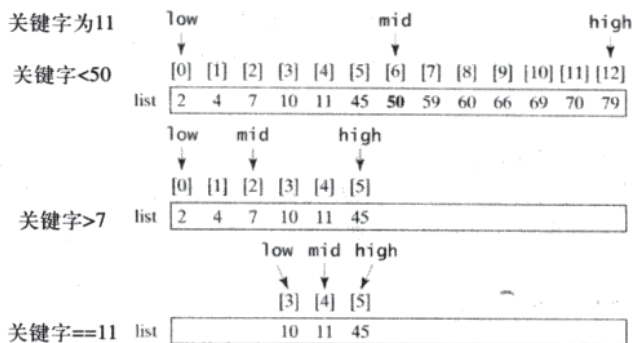


图6-9 二分查找法在每次比较后，数列中需要进一步考虑的元素减少一半

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6-10a所示。它将关键字key和低下标low为0、高下标high为list.length-1的数列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。

```

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6-10 逐步实现二分查找法

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6-10b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。

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6-7中给出。

程序清单6-7 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 $-1 - \text{low}$ （第17行）。

如果用 $(\text{high} > \text{low})$ 替换第7行的 $(\text{high} \geq \text{low})$ ，会出现什么现象呢？这个查找也许会漏掉可能的匹配元素。假如数列只有一个元素，这个查找就会漏掉这个元素。

如果数列中有重复的元素，这个方法还能使用吗？回答是肯定的，只要数列中的元素是按递增顺序排列的。如果查找的元素在数列中，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和值从调用方法返回时， low 和 high 的值。

方法	Low	High	返回值
binarySearch(list,2)	0	1	0
binarySearch(list,11)	3	5	4
binarySearch(list,12)	5	4	-6
binarySearch(list,1)	0	-1	-1
binarySearch(list,3)	1	0	-2

注意 线性查找法适用于在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。

6.10 数组的排序

像查找一样，排序是计算机程序设计中的一个常用任务。已经开发出了很多不同的排序算法。本节将介绍两种简单、直观的排序算法：选择排序法（selection sort）和插入排序法（insertion sort）。

6.10.1 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，然后将它放在数列的最前面。接下来，在剩下的数中找到最小数，将它放到第一个数的后面，依此类推，直到数列中仅剩一个数为止。图6-11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。

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。

解决方案如下所述：

```

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 apply on list[i+1..list.length-1]
}

```

程序清单6-8实现这个解决方案。

程序清单6-8 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 }

```

第一步：找出最小值1，并且将它和2（数列中的第一个数字）互换

互换

2 9 5 4 8 1 6

现在，数字1在正确的位置上，接下来就无须再考虑它

互换

1 9 5 4 8 2 6

选择数字2（最小值）和数字9（剩余数列中的第一个数字）互换

现在，数字2在正确的位置上，接下来就无须再考虑它

互换

1 2 5 4 8 9 6

选择数字4（最小值）和数字5（剩余数列中的第一个数字）互换

现在，数字4在正确的位置上，接下来就无须再考虑它

1 2 4 5 8 9 6

数字5是最小的且放在正确的位置上，无须进行交换

现在，数字5在正确的位置上，接下来就无须再考虑它

互换

1 2 4 5 8 9 6

选择数字6（最小值）和数字8（剩余数列中的第一个数字）互换

现在，数字6在正确的位置上，接下来就无须再考虑它

互换

1 2 4 5 6 9 8

选择数字8（最小值）和数字9（剩余数列中的第一个数字）互换

现在，数字8在正确的位置上，接下来就无须再考虑它

1 2 4 5 6 8 9

由于剩余数列中只剩一个数字，排序结束

图6-11 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

方法`selectionSort (double[] list)`可以对任意一个`double`型元素数组进行排序。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[1]`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);
```

6.10.2 插入排序

假设希望对一个数列进行递增排序。插入排序法的算法是在已排好序的子数列中反复插入一个新元素来对数列值进行排序的，直到整个数列全部排好序。图6-12描述如何用插入排序法对数列{2, 9, 5, 4, 8, 1, 6}进行排序。

这个算法可以描述如下：

```
for (int i=1; i<list.length; i++) {
    将list[i]插入已排好序的子数列中，这样list[0..i]也是排好序的。
}
```

第一步：初始状态时，排好序的子数列包含数列的第一个元素。将9插入这个子数列

2 9 5 4 8 1 6

第二步：排好序的子数列是[2, 9]。将5插入子数列

2 9 5 4 8 1 6

第三步：排好序的子数列是[2, 5, 9]。将4插入子数列

2 5 9 4 8 1 6

第四步：排好序的子数列是[2, 4, 5, 9]。将8插入子数列

2 4 5 9 8 1 6

第五步：排好序的子数列是[2, 4, 5, 8, 9]。将1插入子数列

2 4 5 8 9 1 6

第六步：排好序的子数列是[1, 2, 4, 5, 8, 9]。将6插入子数列

1 2 4 5 8 9 6

第七步：整个数列已排好序

1 2 4 5 6 8 9

图6-12 插入排序将新元素重复插入已排好序的子数列中

为了将`list[i]`插入`list[0..i-1]`，需要将`list[i]`存储在一个名为`currentElement`的临时变量中。如果`list[i-1]>currentElement`，就将`list[i-1]`移到`list[i]`；如果`list[i-2]>currentElement`，就将`list[i-2]`移到`list[i-1]`，依此类推，直到`list[i-k]<=currentElement`或者`k>i`（传递的是排好序的数列的第一个元素）。将`currentElement`赋值给`list[i-k+1]`。例如：为了在图6-13的步骤3中将4插入{2, 5, 9}中，由于`9>4`，所以把`list[2]`（9）移到`list[3]`，又因为`5>4`，所以把`list[1]`（5）移到`list[2]`。最后，把`currentElement`（4）移到`list[1]`。

算法可以扩展和执行，如程序清单6-9所示。

程序清单6-9 InsertionSort.java

```
1 public class InsertionSort {
2     /** The method for sorting the numbers */
3     public static void insertionSort(double[] list) {
```

```

4   for (int i = 1; i < list.length; i++) {
5       /** insert list[i] into a sorted sublist list[0..i-1] so that
6           list[0..i] is sorted. */
7       double currentElement = list[i];
8       int k;
9       for (k = i - 1; k >= 0 && list[k] > currentElement; k--) {
10          list[k + 1] = list[k];
11      }
12      // Insert the current element into list[k + 1]
13      list[k + 1] = currentElement;
14  }
15  }
16  }
17  }

```



图6-13 新元素插入到已排好序的子数列中

`insertionSort(double[] list)`方法是对任意一个`double`类型元素构成的数组进行排序。该方法是用嵌套的`for`循环实现的。外层循环（循环控制变量`i`）（第4行）的迭代是为了获取已排好序的子数列，其范围从`list[0]`到`list[i]`。内层循环（循环控制变量`k`）将`list[i]`插入到从`list[0]`到`list[i-1]`的子数列中。

为了更好地理解这个方法，使用下面的语句跟踪这个方法：

```

double[] list = {1, 9, 4.5, 6.6, 5.7, -4.5};
InsertionSort.insertionSort(list);

```

6.11 Arrays类

为实现数组的排序和查找、数组的比较和对数组填充元素，`java.util.Arrays`类包括各种各样的静态方法。这些方法都有对所有基本类型的重载方法。

可以使用`sort`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

char[] chars = {'a', 'A', '4', 'F', 'D', 'P'};
java.util.Arrays.sort(chars, 1, 3); // Sort part of the array

```

调用`sort(numbers)`对整个数组`numbers`排序。调用`sort(chars, 1, 3)`对从`chars[1]`到`chars[3-1]`的部分数组排序。

可以采用二分查找法（`binarySearch`方法）在数组中查找关键字。数组必须提前按增序排列好。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]和list2[3-1]中。

```

int[] list1 = {2, 4, 7, 10};
int[] list2 = {2, 4, 7, 10};
java.util.Arrays.fill(list1, 5); // Fill 5 to the whole array
java.util.Arrays.fill(list2, 1, 3, 8); // Fill 8 to a partial array

```

关键术语

anonymous array (匿名数组)	index (下标)
array (数组)	indexed variable (下标变量)
array initializer (数组初始化语法)	insertion sort (插入排序)
binary search (二分查找)	linear search (线性查找)
garbage collection (垃圾回收)	selection sort (选择排序)

本章小结

- 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。
- 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。
- 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])。
- 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。
- 创建数组之后,它的大小就不能改变,使用 `arrayRefVar.length` 就可以得到数组的大小。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。

- 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）。
- 当创建一个数组时，若它的元素是基本数据类型的数值，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。
- 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}
- 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。

复习题

6.2节

6.1 如何声明和创建一个数组？

6.2 如何访问数组的元素？

6.3 声明数组时给数组分配内存吗？什么时候为数组分配内存？下面代码的输出结果是什么？

```
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);
```

6.4 指出下列语句是对还是错：

- (1) 数组中的每个元素都有相同的类型。
- (2) 一旦数组被声明，大小就不能改变。
- (3) 一旦数组被创建，大小就不能改变。
- (4) 数组中的元素必须是基本数据类型。

6.5 下面哪些语句是合法的数组声明？

```
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();
```

6.6 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？

6.7 编写语句完成：

- (1) 创建一个含10个double值的数组。
- (2) 将5.5赋值给数组中最后一个元素。
- (3) 显示数组前两个元素的和。
- (4) 编写循环计算数组中所有元素的和。
- (5) 编写循环找出数组的最小值。
- (6) 随机产生一个下标，然后显示该下标所对应的数组元素。
- (7) 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。

6.8 当程序尝试访问下标不合法的数组元素，会发生什么？

6.9 找出并修改下面的代码：

```
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 }

```

6.3节

6.10 使用arraycopy()方法将下述数组复制到目标数组t:

```
int[] source = {3, 4, 5};
```

6.11 一旦数组被创建,其大小就不能再改变。下列代码可以修改数组大小吗?

```

int[] myList;
myList = new int[10];
// Some time later you want to assign a new array to myList
myList = new int[20];

```

6.4~6.7节

6.12 当给方法传递数组时,就会创建一个新数组然后传递给方法。这种说法正确吗?

6.13 给出下面两个程序的输出:

```

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)

6.14 在程序执行过程中,数组存储在哪里?写出程序清单6-4中的createArray、displayArray、countLetters和displayCounts方法在执行中和执行后栈和堆的内容。

6.8节

6.15 下面的方法声明错在哪里?

```

public static void print(String... strings, double... numbers)
public static void print(double... numbers, String name)
public static double... print(double d1, double d2)

```

6.16 能用下列语句调用程序清单6-5中的printMax方法吗?

```

printMax(1, 2, 2, 1, 4);
printMax(new double[]{1, 2, 3});
printMax(new int[]{1, 2, 3});

```

6.9~6.10节

6.17 以图6-9为例,显示如何应用二分查找法在数列{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。

6.18 以图6-11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。

6.19 以图6-12为例,显示如何应用插入排序方法对{3.4, 5, 3, 3.5, 2.2, 1.9, 2}进行排序。

6.20 应该如何修改程序清单6-8中的selectionSort方法,实现数字按递减顺序排序?

6.21 应该如何修改程序清单6-9中的insertionSort方法,实现数字按递减顺序排序?

6.11节

6.22 什么类型的数组能够用 `java.util.Arrays.sort` 方法进行排序？这个 `sort` 方法会创建一个新数组吗？

6.23 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？

6.24 显示执行完每一行之后数组的内容。

```
int[] list = {2, 4, 7, 10};
java.util.Arrays.fill(list, 7);
java.util.Arrays.fill(list, 1, 3, 8);
System.out.print(java.util.Arrays.equals(list, list));
```

编程练习题

6.2节

*6.1（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 $\geq \text{best} - 10$ ，等级为A
- 如果分数 $\geq \text{best} - 20$ ，等级为B
- 如果分数 $\geq \text{best} - 30$ ，等级为C
- 如果分数 $\geq \text{best} - 40$ ，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
```



6.2（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。

**6.3（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。下面就是这个程序的运行示例：

```
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”。

6.4（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。假设成绩的最高分为10分。

**6.5（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 distinct numbers are: 1 2 3 6 4 5
```



*6.6 (修改程序清单4-14) 程序清单4-14通过检验2, 3, 4, 5, 6, ..., $n/2$ 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4-11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。

*6.7 (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。

6.4~6.7节

6.8 (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。

6.9 (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
```



6.10 (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。

*6.11 (统计学方面: 计算标准差) 练习题5.21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。

$$\text{均值} = \frac{\sum_{i=1}^n x_i}{n} = \frac{x_1 + x_2 + \cdots + x_n}{n} \quad \text{标准差} = \sqrt{\frac{\sum_{i=1}^n (x_i - \text{均值})^2}{n-1}}$$

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
```



- *6.12 (倒置数组) 6.7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。

6.8节

- *6.13 (随机数选择器) 编写一个方法, 返回1到54之间的随机数, 不包括传递到实参中的数。如下指定这个方法头:

```
public static int getRandom(int... numbers)
```

- 6.14 (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。

6.9~6.10节

- 6.15 (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[] numbers)
```

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
The distinct numbers are: 1 2 3 6 4 5
```



- 6.16 (执行时间) 编写程序, 随机产生100000个整数值和一个关键字。估算一下调用程序清单6-6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6-7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;
```

- *6.17 (修改选择排序法) 在6.10.1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。
- **6.18 (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)。使用{6.0, 4.4, 1.9, 2.9, 3.4, 2.9, 3.5}测试这个方法。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。
- **6.19 (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。
- ***6.20 (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。一个示例输出如下所示:

```
|Q| | | | | |
| | | |Q| | |
| | | | |Q| |
| |Q| | | | |
| | | | |Q| |
|Q| | | | | |
| | |Q| | | |
| | | | | |
```

***6.21 (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。它是一个三角形状的均匀放置钉子 (或钩子) 的直立板子, 如图6-14所示。

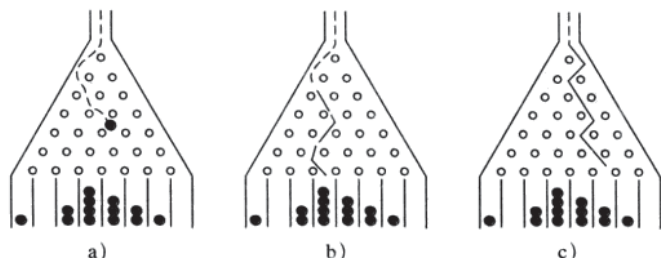


图6-14 每个球都选取一个随机路径, 然后掉入一个槽中

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6-14b中球的路径是LLRRLLR, 而在图6-14c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
Enter the number of slots in the bean machine: 7  -- Enter

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。

***6.22 (游戏: 多种八皇后问题的解决方案) 练习题6.20找出八皇后问题的一种解决方案。编写一个程序, 计算八皇后问题所有可能的解决方案, 然后显示所有的解决方案。

**6.23 (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, 从第二个柜子开始, 第二个柜子用L2表示, 然后关闭其他的柜子。学生S3从第三个柜子开始, 然后改变每个第三个柜子 (如果它是开的就关上, 如果它是关的就打开)。学生S4从柜子L4开始, 然后改变每个第四个柜子。学生S5从L5开始, 然后改变每个第五个柜子, 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。

**6.24 (模拟: 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
```



6.25（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，然后显示根的个数以及所有非复数的根。

6.26（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 equal(int[] list1, int[] list2)
```

编写一个测试程序，提示用户输入两个整数数列，然后显示这两个数列是否完全相同。下面是运行示例。注意，输入的第二个数字表明数列中元素的个数。

```
Enter list1: 5 2 5 6 1 6
Enter list2: 5 2 5 6 1 6
Two lists are strictly identical
```



```
Enter list1: 5 2 5 6 6 1
Enter list2: 5 2 5 6 1 6
Two lists are not strictly identical
```



6.27（相同的数组）如果两个数组list1和list2的内容相同，那么就说它们是相同的。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 equal(int[] list1, int[] list2)
```

编写一个测试程序，提示用户输入两个整数数列，然后显示它们两个是否相同。下面是运行示例。注意，输入的第二个数字表示数列中元素的个数。

```
Enter list1: 5 2 5 6 6 1
Enter list2: 5 5 2 6 1 6
Two lists are identical
```



```
Enter list1: 5 5 5 6 6 1
Enter list2: 5 2 5 6 1 6
Two lists are not identical
```



- 6.28 (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。
- 6.29 (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。程序应该显示若得到的和为24的选牌次数。
- 6.30 (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数列, 如果这个数列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, 即数列中值的个数。



第7章 |

Introduction to Java Programming, 8E

多维数组

学习目标

- 给出使用二维数组表示数据的例子 (7.1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 (7.2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) (7.3节)。
- 给方法传递二维数组 (7.4节)。
- 使用二维数组编写多选题评分程序 (7.5节)。
- 使用二维数组解决距离最近的点对问题 (7.6节)。
- 使用二维数组检测一种九宫格的解决方案 (7.7节)。
- 使用多维数组 (7.8节)。

7.1 引言

第6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

7.2 二维数组的基础知识

该如何声明一个二维数组变量呢？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。

7.2.1 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7-1a所示。

如图7-1b所示,要将7赋值给第2行第1列的特定元素,可以使用下面的语句:

`matrix[2][1] = 7;`

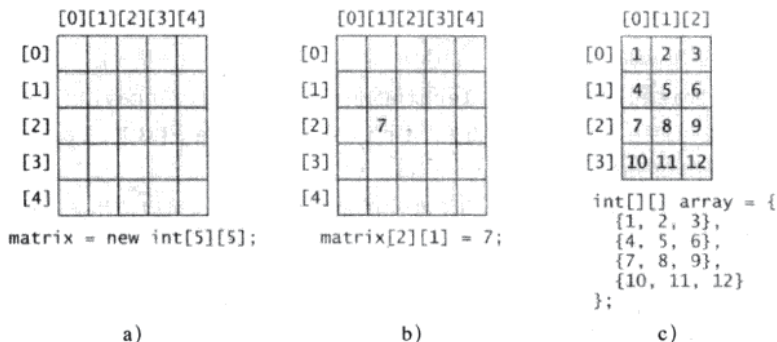
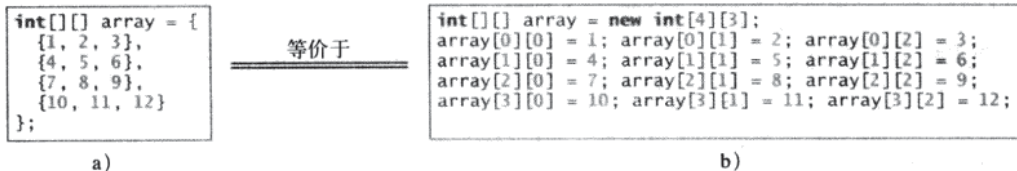


图7-1 二维数组的每个下标索引值都是从0开始的`int`值

警告 使用`matrix[2,1]`访问第2行第1列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:图a中的代码创建一个具有特定初值的数组,如图7-1c所示。它和图b中的代码是等价的。



7.2.2 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 = 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7-2所示。`x.length`为3, `x[0].length`、`x[1].length`和`x[2].length`都是4。

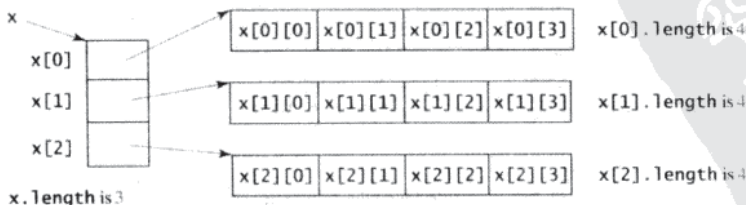
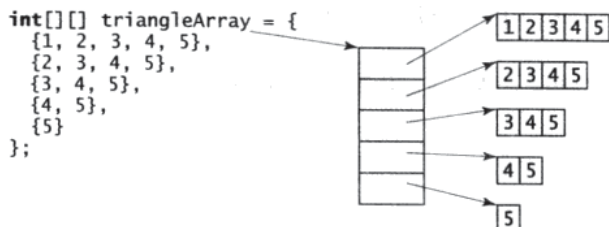


图7-2 二维数组是一个一维数组,它的每个元素是另一个一维数组

7.2.3 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：



从上图可以看到，`triangleArray[0].length`的值为5，`triangleArray[1].length`的值为4，`triangleArray[2].length`的值为3，`triangleArray[3].length`的值为2，`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`new int[5][]`创建数组时，必须指定第一个下标。语法`new int[][]`是错误的。

7.3 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6.2.6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;
    }
}

```

7.4 给方法传递二维数组

可以像传递一维数组一样，给方法传递二维数组。程序清单7-1给出一个返回矩阵中所有元素之和的方法。

程序清单7-1 PassTwoDimensionalArray.java

```

1 import java.util.Scanner;
2
3 public class PassTwoDimensionalArray {
4     public static void main(String[] args) {
5         // Create a Scanner
6         Scanner input = new Scanner(System.in);
7
8         // Enter array values
9         int[][] m = new int[3][4];
10        System.out.println("Enter " + m.length + " rows and "
11            + m[0].length + " columns: ");
12        for (int i = 0; i < m.length; i++)
13            for (int j = 0; j < m[i].length; j++)
14                m[i][j] = input.nextInt();
15
16        // Display result
17        System.out.println("\nSum of all elements is " + sum(m));
18    }
19
20    public static int sum(int[][] m) {
21        int total = 0;
22        for (int row = 0; row < m.length; row++) {
23            for (int column = 0; column < m[row].length; column++) {
24                total += m[row][column];
25            }
26        }
27
28        return total;
29    }
30 }

```

Enter 3 rows and 4 columns:

1 2 3 4

5 6 7 8

9 10 11 12

Sum of all elements is 78



方法sum（第20行）有一个二维数组参数。可以使用m.length获取行数（第22行），而使用m[row].column得到特定行的列数（第23行）。

7.5 问题：多选题测验评分

这里的问题是编写一个程序，对多选题测验进行打分。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：

学生给出的题目答案：

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

PDFG

正确答案存储在一个一维数组中:

题目的正确答案:
 0 1 2 3 4 5 6 7 8 9
 Key D B D C C D A E A D

程序给测验打分并显示结果。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7-2给出该程序。

程序清单7-2 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打分。给一个学生打完分数后就立刻将结果显示出来。

7.6 问题：找出距离最近的点对

GPS导航系统正在变得越来越流行。这个系统使用图形和几何算法计算距离，并且绘制出一个线路。本节介绍一个找出最接近点对的几何问题。

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7-3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7-3所示。

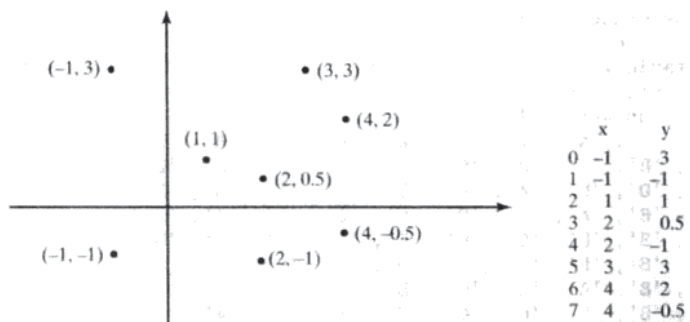


图7-3 使用二维数组表示点

程序清单7-3 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7.8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
```

7.7 问题：九宫格

本书教授如何为各种不同难度的问题来编程，使用简单、短小和刺激的例子来介绍程序设计以及解决问题的技术，并且使用有趣的和有挑战性的例子来激发学生的兴趣。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，通常称为九宫格（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，本节给出九宫格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。解决九宫格问题的完整方案放在补充材料VII.A中。

九宫格是一个 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7-4a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，以便达到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7-4b所示。

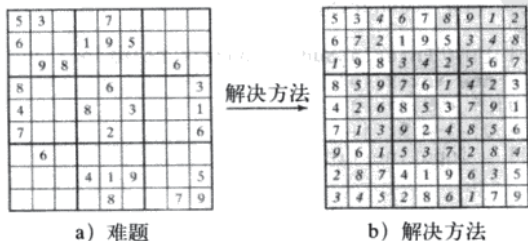


图7-4 图a中的难题在图b中解决

为了方便起见,使用值0表示自由方格,如图7-5a所示。很自然就会使用二维数组表示网格,如图7-5b所示。

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7-5 使用一个二维数组表示网格

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7-4b中的解决方案,网格应该如图7-6所示。

```
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7-6 解决方案存储在网格grid中

九宫格问题的简化版本用来检测解决方案的有效性。程序清单7-4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。

程序清单7-4 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 // Check whether each row has numbers 1 to 9
29 for (int i = 0; i < 9; i++)
30     if (!is1To9(grid[i])) // If grid[i] does not contain 1 to 9
31         return false;
32
33 // Check whether each column has numbers 1 to 9
34 for (int j = 0; j < 9; j++) {
35     // Obtain a column in the one-dimensional array
36     int[] column = new int[9];
37     for (int i = 0; i < 9; i++) {
38         column[i] = grid[i][j];
39     }
40
41     if (!is1To9(column)) // If column does not contain 1 to 9
42         return false;
43 }
44
45 // Check whether each 3-by-3 box has numbers 1 to 9
46 for (int i = 0; i < 3; i++) {
47     for (int j = 0; j < 3; j++) {
48         // The starting element in a small 3-by-3 box
49         int k = 0;
50         int[] list = new int[9]; // Get all numbers in the box to list
51         for (int row = i * 3; row < i * 3 + 3; row++)
52             for (int column = j * 3; column < j * 3 + 3; column++)
53                 list[k++] = grid[row][column];
54
55         if (!is1To9(list)) // If list does not contain 1 to 9
56             return false;
57     }
58 }
59
60 return true; // The fixed cells are valid
61 }
62
63 /** Check whether the one-dimensional array contains 1 to 9 */
64 public static boolean is1To9(int[] list) {
65     // Make a copy of the array
66     int[] temp = new int[list.length];
67     System.arraycopy(list, 0, temp, 0, list.length);
68
69     // Sort the array
70     java.util.Arrays.sort(temp);
71
72     // Check whether the list contains 1, 2, 3, ..., 9
73     for (int i = 0; i < 9; i++)
74         if (temp[i] != i + 1)
75             return false;
76
77     return true; // The list contains exactly 1 to 9
78 }
79 }

```

Enter a Sudoku puzzle solution:

9	6	3	1	7	4	2	5	8
1	7	8	3	2	5	6	4	9
2	5	4	6	8	9	7	3	1
8	2	1	4	3	7	5	9	6
4	9	6	8	5	2	3	1	7
7	3	5	9	6	1	8	2	4
5	8	9	7	1	3	4	6	2
3	1	7	2	4	6	9	8	5
6	4	2	5	9	8	1	7	3

Valid solution



PDFG

程序调用`readASolution()`方法(第6行)来读取一个九宫格的解决方案,并且返回一个表示九宫格网格的二维数组。

`isValid(grid)`方法(第27~61行)检查每行是否都包含从1到9的数字(第29~31行)。`grid`是一个二维数组。`grid[i]`是第*i*行的一维数组。如果`grid[i]`行的确包含从1到9的数,那么调用`is1To9(grid[i])`方法就会返回`true`(第30行)。

为了检测`grid`中每一列是否都包含从1到9的数字,将一列放到一个一维数组中(第36~39行),然后调用`is1To9`方法来检测它是否包括从1到9的数字(第41行)。

为了检测`grid`中每个小的 3×3 的盒子中是否包含从1到9的数字,将一个盒子放到一个一维数组中(第49~53行),然后调用`is1To9`方法来检测它是否包含从1到9的数字(第55行)。

如何将所有的方格都放置在同一个盒子里呢?首先,定位 3×3 盒子的起始方格。它们的位置在 $(3i, 3j)$ (其中 $i=0, 1, 2$; $j=0, 1, 2$),如图7-7所示。

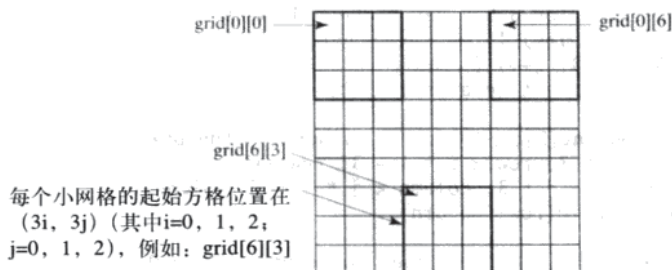


图7-7 在 3×3 盒子中的第一个方格的位置决定了盒子中其他格子位置

随着这种观测的深入,可以很容易地确定盒子中的所有元素。假设`grid[r][c]`是 3×3 盒子的起始方格。这个盒子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 column = c; column < c + 3; column++)
        // grid[row][column] is in the box
```

所有小盒子中的数字都集中在一个一维数组`list`中(第53行),然后调用`is1To9(list)`检测`list`是否包含从1到9的数字(第55行)。

`is1To9(list)`方法(第64~78行)检测数组`list`是否的确包含从1到9的数字。它首先将`list`复制给一个新数组`temp`,然后对`temp`排序。注意:如果对`list`排序,那么`grid`中的内容将改变。在对`temp`排序后,如果`temp`的确包含从1到9的数字,那么`temp`中的数字就应该是1,2,...,9。第73~75行的循环会检测它是否是这种情况。

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 CheckSudokuSoluton < CheckSudokuSoluton.txt
```

7.8 多维数组

在7.7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*维数组变量和创建 $n \geq 3$ 的*n*维数组。例如,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[10][24][2];
```


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。

7.8.1 问题：每日温度和湿度

假设气象站每天每小时都会报告温度和湿度，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。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：

```
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)

```
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 文件的行不一定要顺序排列的。例如：文件可以如图b所示。

你的任务是编写程序，计算这10天的平均日温度和平均日湿度。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，表示10天；第二个下标范围从0到23，表示24小时；而第三个下标范围从0到1，分别表示温度和湿度。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个小时的温度，而`data[9][23][1]`存储的是第10天第24小时的湿度。

该程序在程序清单7-5中给出。

程序清单7-5 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 }
35 }

```

```

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行显示平均日温度和日湿度。

7.8.2 问题：猜生日

程序清单3-3给出一个猜生日的程序。可以通过用三维数组来存储5个集合的数字来简化程序，然后使用循环提示用户回答，如程序清单7-6所示。该程序的运行示例和程序清单3-3所显示的是一样。

程序清单7-6 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 birth day is " + day);
49 }
50 }

```

在第8~28行创建三维数组**dates**。这个数组存储5个集合的数字。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，然后提示用户回答这个生日是否在该集合中（第41~42行）。如果该天是在某个集合中，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）。

本章小结

- 使用二维数组来存储表格。
- 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。

- 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]。

- 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]。

- 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} }。

- 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。

复习题

- 7.1 声明并创建一个 4×5 的整型矩阵。
- 7.2 二维数组的行可以有不同的长度吗？
- 7.3 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]);

```

- 7.4 以下语句中，哪些是合法的数组声明？

```

int[][] r = new int[2];
int[] x = new int[];
int[][] y = new int[3][];

```

- 7.5 为什么is1To9方法需要将list复制到temp？如果用下面的代码替换程序清单7-4中第66~70行的代



码,会发生什么?

```
java.util.Arrays.sort(list);
```

7.6 声明并创建一个 $4 \times 6 \times 5$ 的整型矩阵。


编程练习题

*7.1 (求矩阵中所有数的和) 编写一个方法,求整数矩阵中所有整数的和,使用下面的方法头:

```
public static double sumMatrix(int[][] m)
```

编写一个测试程序,读取一个 4×4 的矩阵,然后显示所有元素的和。下面是一个运行示例:

```
Enter a 4-by-4 matrix row by row:
1 2 3 4 --Enter
5 6 7 8 --Enter
9 10 11 12 --Enter
13 14 15 16 --Enter
Sum of the matrix is 136
```




*7.2 (求矩阵主对角线元素的和) 编写一个方法,求 $n \times n$ 的整数矩阵中主对角线上所有整数的和,使用下面的方法头:

```
public static int sumMajorDiagonal(int[][] 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 --Enter
5 6 7 8 --Enter
9 10 11 12 --Enter
13 14 15 16 --Enter
Sum of the elements in the major diagonal is 34
```



*7.3 (按考分对学生排序) 重写程序清单7-2,以正确答案个数的升序显示学生。

**7.4 (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:下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	H	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

7.5 (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
Enter matrix2: 0 2 4 1 4.5 2.2 1.1 4.3 5.2 --Enter
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

```



****7.6**（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 能够与矩阵 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
Enter matrix2: 0 2 4 1 4.5 2.2 1.1 4.3 5.2 --Enter
The matrices are multiplied as follows:
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

```



***7.7**（距离最近的两个点）程序清单7-3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 $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$ 。

****7.8**（所有最近的点对）修改程序清单7-3，找出所有具有相同最小距离的点对。

*****7.9**（游戏：玩井字游戏）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，表明游戏结束，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 (1, 2, or 3) for player X: 1 Enter
Enter a column (1, 2, or 3) for player X: 1 Enter

| | | |
| | X | |
| | | |

Enter a row (1, 2, or 3) for player O: 1 Enter
Enter a column (1, 2, or 3) for player O: 2 Enter

| | | |
| | X | O |
| | | |

```

```

Enter a row (1, 2, or 3) for player X:
...
| X | | |
| O | X | O |
| | | X |

X player won

```

*7.10 (游戏: 井字游戏棋盘) 编写程序, 随机地在井字游戏棋盘上填入0或者1, 打印该棋盘, 然后找出全是0或者全是1的行、列或对角线。使用一个二维数组表示一个井字游戏的棋盘。下面是这个程序的一个运行示例:

```

001
001
111
All 1s on row 2
All 1s on column 2

```

**7.11 (游戏: 九个正面和背面) 一个 3×3 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 3×3 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  Enter
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。

****7.12** (财务应用程序：计算税款) 使用数组重写程序清单3-6。针对每个登记者的身份，都有六种税率。每个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%，82 250~372 550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
    {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]
```

***7.13** (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
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 at (1, 2)
```



****7.14** (探究矩阵) 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  Enter
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
```



*7.15 (几何方面: 在同一行吗?) 假设给出一个点的集合, 编写程序, 检测是否所有的点都在同一行上。使用下面的集合测试程序:

```
double[][] set1 = {{1, 1}, {2, 2}, {3, 3}, {4, 4}};
double[][] set2 = {{0, 1}, {1, 2}, {4, 5}, {5, 6}};
double[][] set3 = {{0, 1}, {1, 2}, {4, 5}, {4.5, 4}};
```

*7.16 (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}}排序后就成了{{1, 1}, {1, 2}, {1, 7}, {4, 1}, {4, 2}, {4, 5}}。

***7.17 (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7-8就是五个银行的情况图。每个银行的当前余额分别是2500万美金、1亿2500万美金、1亿7500万美金、7500万美金和1亿8100万美金。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4千万美金。

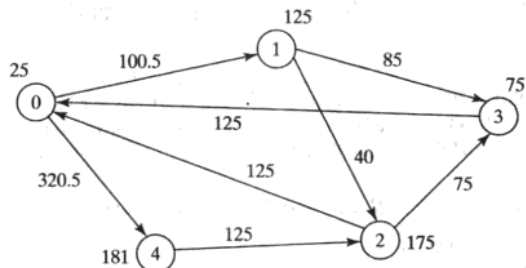


图7-8 银行之间互相借款

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, 程序会输出描述 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7-8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。

*7.18 (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}};
```

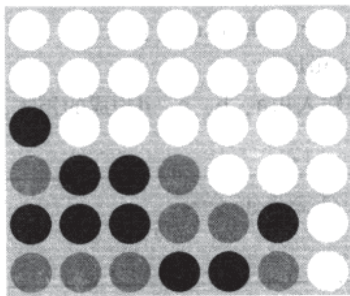

****7.19 (模式识别：连续四个相等的数)** 编写下面的方法，测试一个二维数组是否有四个连续的数字具有相同的值，这四个数可以是水平方向的、垂直方向的或者对角线方向的。

public static boolean isConsecutiveFour(int[][] values)

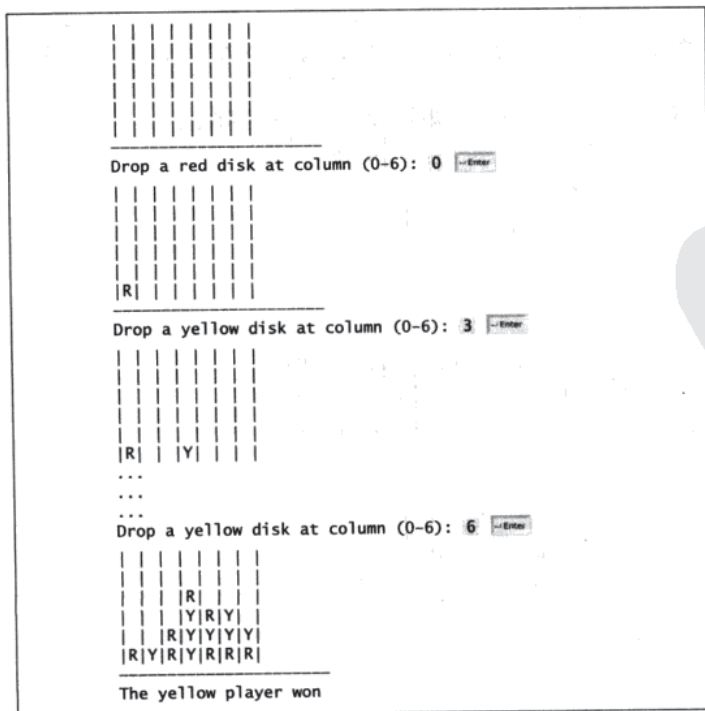
编写一个测试程序，提示用户输入一个二维数组的行数、列数以及数组中的值。如果这个数组有四个连续的数字具有相同的值，就显示true；否则，显示false。下面是结果为true的一些例子：

0 1 0 3 1 6 1	0 1 0 3 1 6 1	0 1 0 3 1 6 1	0 1 0 3 1 6 1
0 1 6 8 6 0 1	0 1 6 8 6 0 1	0 1 6 8 6 0 1	0 1 6 8 6 0 1
5 6 2 1 8 2 9	5 5 2 1 8 2 9	5 6 2 1 6 2 9	9 6 2 1 8 2 9
6 5 6 1 1 9 1	6 5 6 1 1 9 1	6 5 6 6 1 9 1	6 9 6 1 1 9 1
1 3 6 1 4 0 7	1 5 6 1 4 0 7	1 3 6 1 4 0 7	1 3 9 1 4 0 7
3 3 3 3 4 0 7	3 5 3 3 4 0 7	3 6 3 3 4 0 7	3 3 3 9 4 0 7

*****7.20 (游戏：连接四子)** 连接四子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反方出现一行、一列或者一条对角线上有四个相同颜色的棋子之前，正方能先做到。程序提示两个玩家交替地下红子Red或黄子Yellow。当丢下一子时，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：



数字资源
PDG

***7.21 (游戏: 多种九宫格解决方案) 完整的九宫格问题的解决方案在补充材料VII.A中给出。一个九宫格问题可以有多种解决方案。修改补充材料VII.A中的Sudoku.java文件, 显示所有解决方案的总个数。如果存在多种解决方案, 显示两个解决方案。

*7.22 (代数问题: 2×2 矩阵的逆阵) 方阵A的逆阵用 A^{-1} 来表示, 满足 $A \times A^{-1} = I$, 这里的I是指一个单位矩阵, 它的对角线上的值都为1, 而其他位置的值都为0。例如: 矩阵 $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ 的逆阵是 $\begin{bmatrix} -0.5 & 1 \\ 1.5 & 0 \end{bmatrix}$, 即

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} -0.5 & 1 \\ 1.5 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

可以使用下面的公式得到一个 2×2 的矩阵A的逆阵:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad A^{-1} = \frac{1}{ad-bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

实现下面的方法可以获取该矩阵的逆阵:

```
public static double[][] inverse(double[][] A)
```

如果 $ad-bc$ 为0, 则该方法返回null。

编写一个测试程序, 提示用户输入由a、b、c、d构成的矩阵, 然后显示该矩阵的逆阵。下面是一个运行示例:

```
Enter a, b, c, d: 1 2 3 4
-2.0 1.0
1.5 -0.5
```



```
Enter a, b, c, d: 0.5 2 1.5 4.5
-6.0 2.6666666666666665
2.0 -0.6666666666666666
```



*7.23 (代数问题: 3×3 矩阵的逆阵) 方阵A的逆阵表示为 A^{-1} , 满足 $A \times A^{-1} = I$, 其中I是所有对角线

上的值为1而其他所有值为0的单位矩阵。例如, 矩阵 $\begin{bmatrix} 1 & 2 & 1 \\ 2 & 3 & 1 \\ 4 & 5 & 3 \end{bmatrix}$ 的逆阵是:

$$\begin{bmatrix} -2 & 0.5 & 0.5 \\ 1 & 0.5 & -0.5 \\ 1 & -1.5 & 0.5 \end{bmatrix}$$

也就是:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 3 & 1 \\ 4 & 5 & 3 \end{bmatrix} \times \begin{bmatrix} -2 & 0.5 & 0.5 \\ 1 & 0.5 & -0.5 \\ 1 & -1.5 & 0.5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

对于一个 3×3 的矩阵:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

使用下面的公式可以得到其逆阵, 其中 $|A| \neq 0$:

数字资源

PDG

$$A^{-1} = \frac{1}{|A|} \begin{bmatrix} a_{22}a_{33} - a_{23}a_{32} & a_{13}a_{32} - a_{12}a_{33} & a_{12}a_{23} - a_{13}a_{22} \\ a_{23}a_{31} - a_{21}a_{33} & a_{11}a_{33} - a_{13}a_{31} & a_{13}a_{21} - a_{11}a_{23} \\ a_{21}a_{32} - a_{22}a_{31} & a_{12}a_{31} - a_{11}a_{32} & a_{11}a_{22} - a_{12}a_{21} \end{bmatrix}$$

$$|A| = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}a_{22}a_{33} + a_{31}a_{12}a_{23} + a_{13}a_{21}a_{32} - a_{13}a_{22}a_{31} - a_{11}a_{23}a_{32} - a_{33}a_{21}a_{12}$$

实现下面的方法可得到矩阵的逆阵：

```
public static double[][] inverse(double[][] A)
```

如果 $|A|$ 为0，那么方法返回null。

编写一个测试程序，提示用户输入一个矩阵的 a_{11} ， a_{12} ， a_{13} ， a_{21} ， a_{22} ， a_{23} ， a_{31} ， a_{32} ， a_{33} ，然后显示这个矩阵的逆阵。下面是一个运行示例：

```
Enter a11, a12, a13, a21, a22, a23, a31, a32, a33:
1 2 1 2 3 1 4 5 3
-2 0.5 0.5
1 0.5 -0.5
1 -1.5 0.5
```



```
Enter a11, a12, a13, a21, a22, a23, a31, a32, a33:
1 4 2 2 5 8 2 1 8
2.0 -1.875 1.375
0.0 0.25 -0.25
-0.5 0.4375 -0.1875
```



资源分享

PDG

第8章

Introduction to Java Programming, 8E

对象和类

学习目标

- 描述对象和类，使用类来建模对象（8.2节）。
- 使用UML图形符号来描述类和对象（8.2节）。
- 演示如何定义类以及如何创建对象（8.3节）。
- 使用构造方法创建对象（8.4节）。
- 通过对象引用变量访问对象（8.5节）。
- 使用引用类型定义引用变量（8.5.1节）。
- 使用对象成员访问操作符（.）来访问对象的数据和方法（8.5.2节）。
- 定义引用类型的数据域并给对象的数据域赋默认值（8.5.3节）。
- 区分对象引用变量和基本类型变量的不同（8.5.4节）。
- 使用Java类库中的Data类、Random类和JFrame类（8.6节）。
- 区分实例变量与静态变量、实例方法与静态方法的不同（8.7节）。
- 定义有恰当的get方法和set方法的私有数据域（8.8节）。
- 封装数据域以便于类的维护（8.9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（8.10节）。
- 在数组中存储和处理对象（8.11节）。

8.1 引言

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8-1所示，该如何用程序实现它呢？



图8-1 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。

8.2 定义对象的类

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，也称之为特征（property）或属性（attribute））是指那些具有它们当前值的数据域。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，它们都是标识矩形的属性。
- 一个对象的行为（behavior，也称之为动作（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()`的方法。圆对象可以调用`getArea()`返回圆的面积。

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）。术语对象（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8-2显示名为`Circle`的类和它的三个对象。

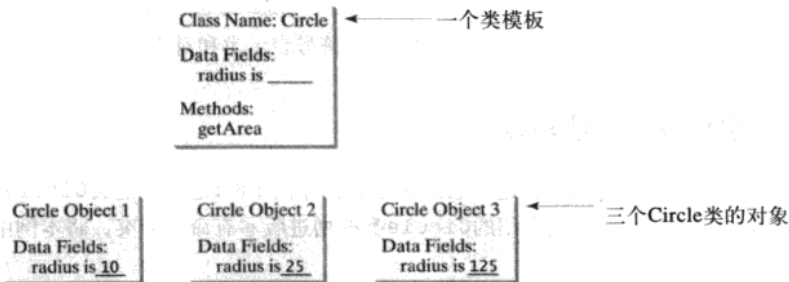


图8-2 类是创建对象的模板

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，但是设计构造方法的初衷还是为了完成初始化动作，例如：初始化对象的数据域。图8-3显示定义圆对象的类的例子。

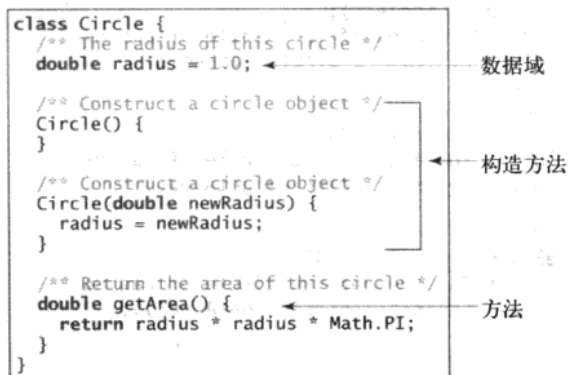


图8-3 类是一个定义相同类型对象的结构体

`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8-2中类的模板和对象的图解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8-4所示，这种符号称为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` 数据域名:数据域类型

构造方法可以表示为：

`ClassName(parameterName: parameterType)` 类名（参数名:参数类型）

方法可以表示为：

`methodName(parameterName: parameterType): returnType` 方法名（参数名:参数类型）:返回类型

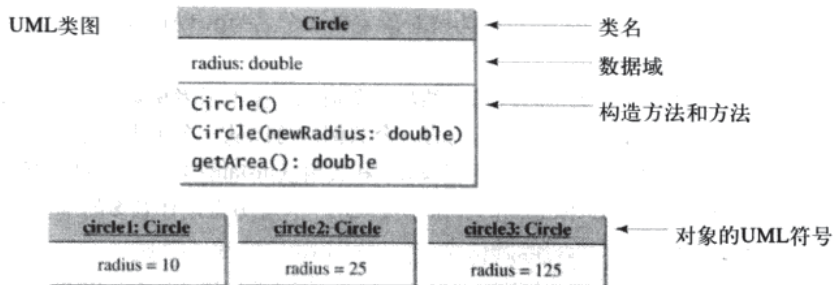


图8-4 可以使用UML符号表示类和对象

8.3 举例：定义类和创建对象

本节给出两个定义类和使用类创建对象的例子。程序清单8-1是一个定义Circle类并使用该类创建对象的程序。为了避免与本章后续介绍的Circle类的增进版本有命名冲突，将本例中的Circle类命名为Circle1。

程序构造了三个圆对象，其半径分别为1.0、25和125，然后显示这三个圆的半径和面积。将第二个对象的半径改为100，然后显示它的新半径和面积。

程序清单8-1 TestCircle1.java

```

1 public class TestCircle1 {
2     /** Main method */
3     public static void main(String[] args) {
4         // Create a circle with radius 1.0
5         Circle1 circle1 = new Circle1();
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        Circle1 circle2 = new Circle1(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        Circle1 circle3 = new Circle1(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
26 // Define the circle class with two constructors
27 class Circle1 {
28     double radius;
29
30     /** Construct a circle with radius 1 */
31     Circle1() {
32         radius = 1.0;
33     }
34
35     /** Construct a circle with a specified radius */
36     Circle1(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 }

```

```

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TestCircle1是主类。它的唯一目的就是测试第二个类Circle1。使用这样的类的程序通常称为是该类的客户 (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的。此外, 公共类必须与文件名。因此, 文件名就应该是TestCircle1.java, 因为TestCircle1是公共的。

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 Circle1()创建一个半径为1.0的对象 (第5行), new Circle1(25)创建一个半径为25的对象 (第10行), 而new Circle1(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。

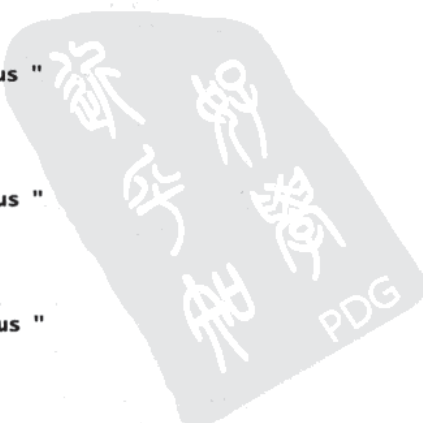
编写Java程序的方法有很多种。例如: 可以将例子中的两个类组合成一个, 如程序清单8-2所示。

程序清单8-2 Circle1.java

```

1 public class Circle1 {
2     /** Main method */
3     public static void main(String[] args) {
4         // Create a circle with radius 1.0
5         Circle1 circle1 = new Circle1();
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        Circle1 circle2 = new Circle1(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        Circle1 circle3 = new Circle1(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 Circle1() {
29     radius = 1.0;
30 }
31
32 /** Construct a circle with a specified radius */
33 Circle1(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 }

```

由于组合的类中有一个main方法，所以它可以由Java解释器来执行。main方法和程序清单8-1中的是一样的。它演示如何通过在一个类中只要加入main方法，就能测试这个类。

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8-5所示。

程序清单8-3给出定义TV类的程序。

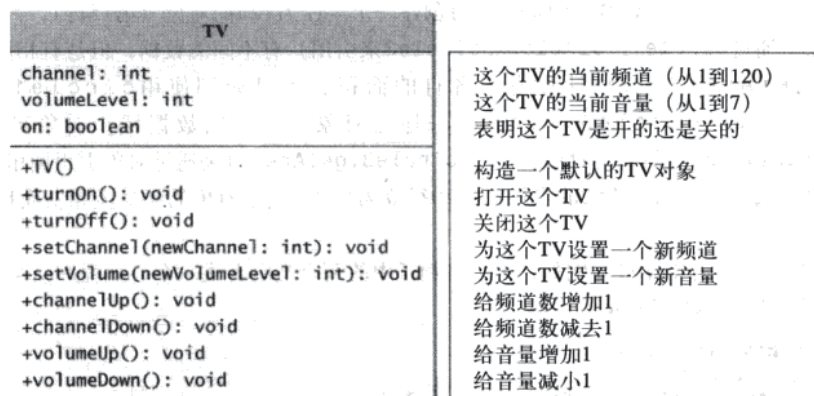


图8-5 TV类是对电视机的建模

程序清单8-3 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; // By default 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     }

```

新华书店
PDG


```

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 }

```

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8-4给出使用TV类创建两个对象的程序。

程序清单8-4 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, 本节随后将会详细讨论这些话题。

8.4 使用构造方法构造对象

构造方法是一种特殊的方法。它们有以下三个特殊性：

- 1) 构造方法必须具备和所在类相同的名字。
- 2) 构造方法没有返回类型，甚至连void也没有。
- 3) 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); new 类名(参数);
```

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。

8.5 通过引用变量访问对象

要给新创建的对象在内存中分配空间。它们可以通过引用变量来访问。

8.5.1 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; 类名 对象引用变量;
```

一个类基本上等同于一个程序员定义的类型。一个类就是一种引用类型（reference type），这意味着任何类型为类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();
```

利用如下所示的语法，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(); 类名 对象引用变量 = new 类名();
```

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，它只是包括了对该对象的引用。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，可以简单地说明myCircle是一个Circle对象，而不用冗长地描述说，myCircle是一个包含对Circle对象的引用变量。

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。

8.5.2 访问对象的数据和方法

在创建一个对象之后，它的数据和方法可以使用圆点运算符（.）来访问和调用，该运算符也称为对象成员访问运算符（object member access operator）：

- objectRefVar.dataField引用对象的数据域。
- objectRefVar.method(参数)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8.7节中给出。

注意 大多数时候，我们创建一个对象，然后会将它赋值给一个变量。接下来，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）。

8.5.3 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 default value null
    int age; // age has 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。

8.5.4 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8-6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。

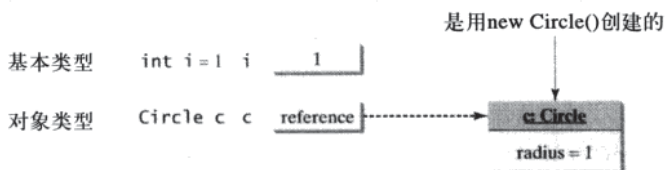


图8-6 基本类型变量在内存中存储的是一个基本类型值，而引用类型变量存储的是一个引用，它指向对象在内存中的位置

将一个变量赋值给另一个变量时，另一个变量就被赋了同样的值。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8-7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8-8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。

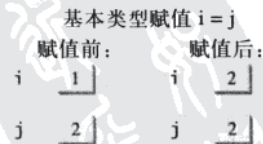


图8-7 基本类型变量*j*复制到变量*i*中

注意 如图8-8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了，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，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。

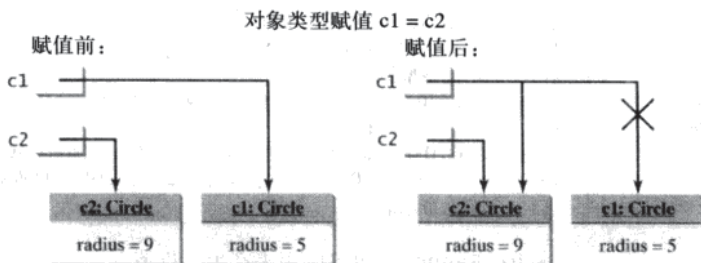


图8-8 引用变量c2复制到变量c1中

8.6 使用Java库中的类

程序清单8-1声明了Circle1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。

8.6.1 Date类

在程序清单2-6中，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8-9所示。

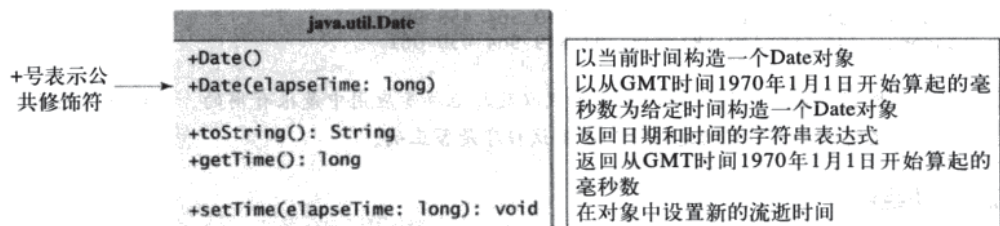


图8-9 Date对象表示特定的日期和时间

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日算起至今逝去的时间，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 1100547210284 milliseconds
Mon Nov 15 14:33:30 EST 2004
```

Date类还有另外一个构造方法：Date(long elapseTime)，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日算起至今逝去的以毫秒为单位的给定时间。

8.6.2 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8-10所示的java.util.Random类，它可以产生一个int、long、double、float和boolean型值。

<code>java.util.Random</code>
<code>+Random()</code>
<code>+Random(seed: long)</code>
<code>+nextInt(): int</code>
<code>+nextInt(n: int): int</code>
<code>+nextLong(): long</code>
<code>+nextDouble(): double</code>
<code>+nextFloat(): float</code>
<code>+nextBoolean(): boolean</code>

以当前时间作为种子构造Random对象
 以特定种子构造Random对象
 返回一个随机整型值
 返回一个在0到n之间（不包括0和n）的随机整型值
 返回一个随机的long型值
 返回一个在0.0到1.0之间（不包括0.0和1.0）的随机double型值
 返回一个在0.0F到1.0F之间（不包括0.0F和1.0F）的随机float型值
 返回一个随机布尔值

图8-10 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。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。可以在使用不同随机数序列之前，使用固定的随机序列以测试程序是否正确。

8.6.3 显示GUI组件

教学注意 图形用户界面（GUI）组件是讲授OOP的很好的例子。为了教授OOP，先介绍一些简单的GUI的例子。对GUI程序设计的完整介绍从第12章开始。

开发程序创建图形用户界面时，将会用到像JFrame、JButton、JRadioButton、JComboBox和JList这样的Java类来创建框架、单选按钮、组合框、列表等。程序清单8-5是一个使用JFrame类创建两个窗口的例子。这个程序的输出如图8-11所示。

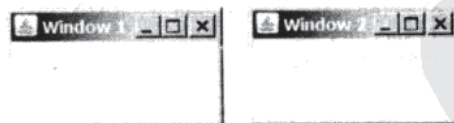


图8-11 这个程序使用JFrame类创建两个窗口

程序清单8-5 TestFrame.java

```
1 import javax.swing.JFrame;
2
3 public class TestFrame {
4     public static void main(String[] args) {
5         JFrame frame1 = new JFrame();
6         frame1.setTitle("Window 1");
7         frame1.setSize(200, 150);
8         frame1.setLocation(200, 100);
9         frame1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```

10     frame1.setVisible(true);
11
12     JFrame frame2 = new JFrame();
13     frame2.setTitle("Window 2");
14     frame2.setSize(200, 150);
15     frame2.setLocation(410, 100);
16     frame2.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17     frame2.setVisible(true);
18 }
19 }

```

这个程序创建两个JFrame类的对象（第5、12行），然后使用方法setTitle、setSize、setLocation、setDefaultCloseOperation和setVisible来设置这两个对象的属性。方法setTitle设置窗口的标题（第6、13行）。方法setSize设置窗口的宽度和高度（第7、14行）。方法setLocation指明窗口左上角的位置（第8、15行）。方法setDefaultCloseOperation在关闭框架时终止该程序（第9、16行）。方法setVisible表示是否显示这个窗口。

可以给窗口添加像按钮、标签、文本域、复选框和组合框这样的组件。组件是使用类来定义的。程序清单8-6给出一个创建图形用户界面的例子，如图8-1所示。

程序清单8-6 GUIComponents.java

```

1  import javax.swing.*;
2
3  public class GUIComponents {
4      public static void main(String[] args) {
5          // Create a button with text OK
6          JButton jbtOK = new JButton("OK");
7
8          // Create a button with text Cancel
9          JButton jbtCancel = new JButton("Cancel");
10
11         // Create a label with text "Enter your name: "
12         JLabel jlblName = new JLabel("Enter your name: ");
13
14         // Create a text field with text "Type Name Here"
15         JTextField jtfnName = new JTextField("Type Name Here");
16
17         // Create a check box with text bold
18         JCheckBox jchkBold = new JCheckBox("Bold");
19
20         // Create a check box with text italic
21         JCheckBox jchkItalic = new JCheckBox("Italic");
22
23         // Create a radio button with text red
24         JRadioButton jrbRed = new JRadioButton("Red");
25
26         // Create a radio button with text yellow
27         JRadioButton jrbYellow = new JRadioButton("Yellow");
28
29         // Create a combo box with several choices
30         JComboBox jcbColor = new JComboBox(new String[]{"Freshman",
31             "Sophomore", "Junior", "Senior"});
32
33         // Create a panel to group components
34         JPanel panel = new JPanel();
35         panel.add(jbtOK); // Add the OK button to the panel
36         panel.add(jbtCancel); // Add the Cancel button to the panel
37         panel.add(jlblName); // Add the label to the panel
38         panel.add(jtfnName); // Add the text field to the panel
39         panel.add(jchkBold); // Add the check box to the panel
40         panel.add(jchkItalic); // Add the check box to the panel
41         panel.add(jrbRed); // Add the radio button to the panel
42         panel.add(jrbYellow); // Add the radio button to the panel

```

```

43    panel.add(jcboColor); // Add the combo box to the panel
44
45    JFrame frame = new JFrame(); // Create a frame
46    frame.add(panel); // Add the panel to the frame
47    frame.setTitle("Show GUI Components");
48    frame.setSize(450, 100);
49    frame.setLocation(200, 100);
50    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
51    frame.setVisible(true);
52 }
53 }

```

这个程序使用类JButton、JLabel、JTextField、JCheckBox、JRadioButton和JComboBox来创建GUI对象（第6~31行）。然后，它使用JPanel类创建一个面板对象（第34行），并将按钮、标签、文本域、复选框、单选框以及组合框添加到这个面板中（第35~43行）。接着，程序创建一个框架，将刚才的面板添加到这个框架中（第45行）。第51行显示这个框架。

8.7 静态变量、常量和方法

程序清单8-1中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），也称之为类变量（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8-12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，静态变量和方法都是以下划线标注的。）

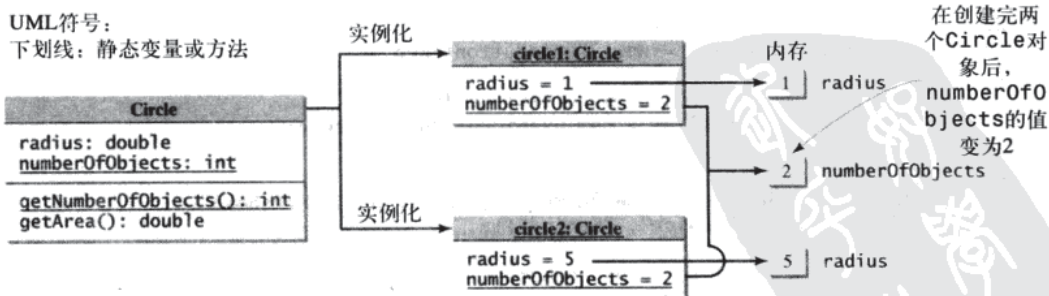


图8-12 属于实例的实例变量存储在互不相关的内存中。静态变量是被同一个类的所有实例所共享的

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Circle2的圆类的声明如程序清单8-7所示。

程序清单8-7 Circle2.java

```

1 public class Circle2 {
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     Circle2() {
10         radius = 1.0;
11         numberOfObjects++;
12     }
13
14     /** Construct a circle with a specified radius */
15     Circle2(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 }

```

Circle2类中的getNumberOfObjects()方法是一个静态方法。静态方法的其他例子有JOptionPane类中的showMessageDialog方法和showInputDialog方法，以及Math类中的所有的方法。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8-8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8-8 TestCircle2.java

```

1 public class TestCircle2 {
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             Circle2.numberOfObjects);
7
8         // Create c1
9         Circle2 c1 = new Circle2();
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     Circle2 c2 = new Circle2(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TestCircle2.java时，如果最后一次修改了Circle2.java之后还没有编译它，Java编译器就会自动编译Circle2.java。

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，使用Math.PI来访问这个常量。可以用Circle2.numberOfObjects来代替c.numberOfObjects。这样可以提高可读性，因为读者可以很容易地识别静态变量。也可以用Circle2.getNumberOfObjects()替换Circle2.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，因为读者可以很容易地识别出类中的静态方法和数据。

静态变量和静态方法既可以在类的实例方法中使用，也可以在类的静态方法中使用。但是，实例变量和实例方法只能在实例方法中使用，不能在静态方法中使用，因为静态变量和静态方法不属于某个特定的对象。因此，下面给出的代码就是错误的。

```

1 public class Foo {
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 如果用下面的代码替换第5~8行的代码, 程序就是正确的, 因为实例数据域*i*和方法*m1*是通过对象*foo*访问的 (第6~7行):

```

1 public class Foo {
2     int i = 5;
3     static int k = 2;
4
5     public static void main(String[] args) {
6         Foo foo = new Foo();
7         int j = foo.i; // OK, foo.i accesses the object's instance variable
8         foo.m1(); // OK. Foo.m1() invokes 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

8.8 可见性修饰符

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, 那么默认为类、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.G。

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11.13节中介绍。

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8-13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。

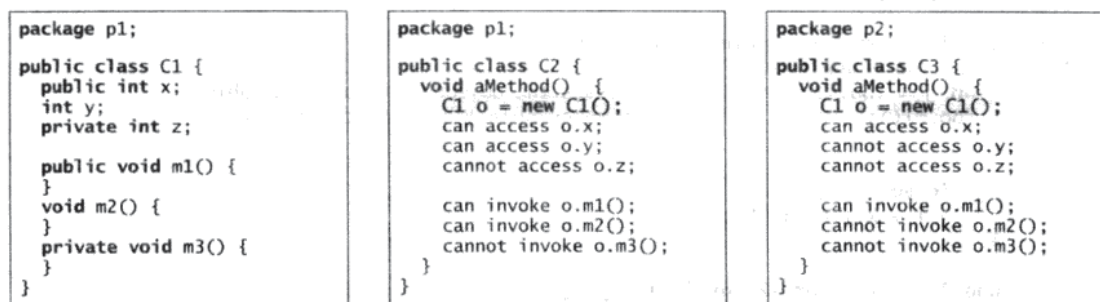


图8-13 私有的修饰符将访问权限限定在它自己的类内，

默认修饰符将访问权限限定在包内，而公共的修饰符没有限定权限

如果一个类没有被定义为公共的，那么它只能在同一个包内被访问。如图8-14所示，C2可以访问C1，而C3不能访问C1。

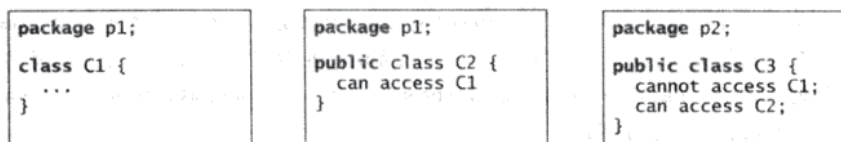
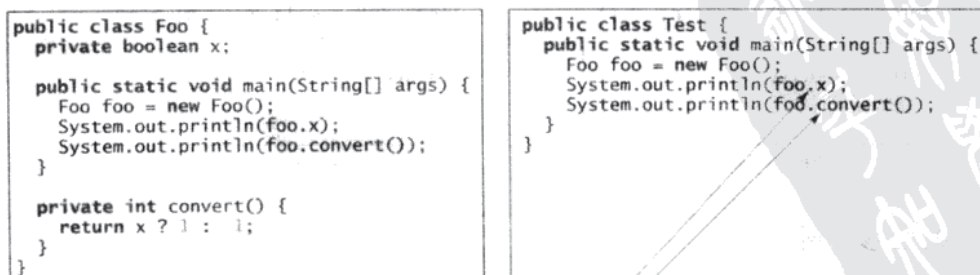


图8-14 一个非公共类具有包访问性

可见性修饰符指明类中的数据域和方法是否能从该类之外被访问。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8-15b所示，Foo类的对象foo不能引用它的私有成员，因为foo在Test类中。如图8-15a所示，Foo类的对象foo可以访问它的私有成员，因为foo在自己的类内定义。



a) 因为对象foo是在Foo类中使用的，
所以这样做是可以的

b) 因为x和convert在Foo中是私有的，
所以这样做是错误的

图8-15 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

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() {
}
```

8.9 数据域封装

在程序清单8-7中，`Circle2`类的数据域`radius`和`numberOfObjects`可以直接修改（例如：`myCircle.radius = 5`或`Circle2.numberOfObjects = 10`）。这不是一个好的实例，原因有两点：

第一，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：`Circle2.numberOfObjects = 10`）。

第二，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`Circle2`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`Circle2`类，而且还要修改使用`Circle2`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。这就称为数据域封装（data field encapsulation）。

在定义私有数据域的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`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`set`方法给数据域设置新值。

注意 通俗地讲，`get`方法称为读取器（getter）（或访问器（accessor）），而`set`方法称为设置器（setter）（或修改器（mutator））。

`get`方法有如下签名：

```
public returnType getPropertyName()
```

如果返回类型是`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)
```

我们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8-16所示。程序清单8-9中定义一个名为`Circle3`的新圆的类。

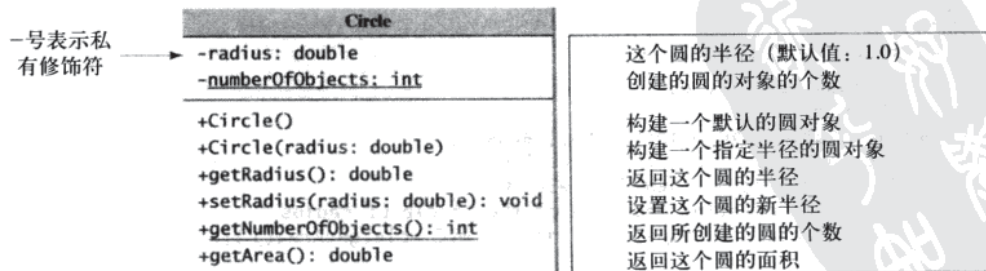


图8-16 `Circle`类封装了圆的属性并提供了`get/set`方法以及其他方法

程序清单8-9 Circle3.java

```

1 public class Circle3 {
2     /** The radius of the circle */
3     private double radius = 1;
4
5     /** The number of the objects created */
6     private static int numberOfObjects = 0;
7
8     /** Construct a circle with radius 1 */
9     public Circle3() {
10         numberOfObjects++;
11     }
12
13     /** Construct a circle with a specified radius */
14     public Circle3(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8-10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8-10 TestCircle3.java

```

1 public class TestCircle3 {
2     /** Main method */
3     public static void main(String[] args) {
4         // Create a Circle with radius 5.0
5         Circle3 myCircle = new Circle3(5.0);
6         System.out.println("The area of the circle of radius "
7             + myCircle.getRadius() + " is " + myCircle.getArea());
8
9         // Increase myCircle's radius by 10%
10        myCircle.setRadius(myCircle.getRadius() * 1.1);
11        System.out.println("The area of the circle of radius "
12            + myCircle.getRadius() + " is " + myCircle.getArea());
13
14        System.out.println("The number of objects created is "

```

```

15         + Circle3.getNumberOfObjects() );
16     }
17 }

```

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。如果企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TestCircle类中的main方法移到Circle类中，实现将TestCircle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8.15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。

8.10 给方法传递对象参数

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         // Circle3 is defined in Listing 8.9
4         Circle3 myCircle = new Circle3(5.0);
5         printCircle(myCircle);
6     }
7
8     public static void printCircle(Circle3 c) {
9         System.out.println("The area of the circle of radius "
10            + c.getRadius() + " is " + c.getArea());
11     }
12 }

```

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。

通过程序清单8-11中的程序，让我们来看一看传递基本类型值和传递引用值的差异。

程序清单8-11 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         Circle3 myCircle = new Circle3(1);
6
7         // Print areas for radius 1, 2, 3, 4, and 5.
8         int n = 5;
9         printAreas(myCircle, n);
10
11         // See myCircle.radius and times
12         System.out.println("\n" + "Radius is " + myCircle.getRadius());
13         System.out.println("n is " + n);
14     }
15
16     /** Print a table of areas for radius */
17     public static void printAreas(Circle3 c, int times) {
18         System.out.println("Radius \t\tArea");
19         while (times >= 1) {
20             System.out.println(c.getRadius() + "\t\t" + c.getArea());
21             c.setRadius(c.getRadius() + 1);
22             times--;
23         }
24     }
25 }

```

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	



Circle3类是在程序清单8-9中定义的。这个程序使用Circle3类的一个对象myCircle和n的整数值调用printAreas(myCircle,n)方法（第9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8-17展示执行程序的这个方法的过程中对栈的调用。注意，对象是存储在堆中的。

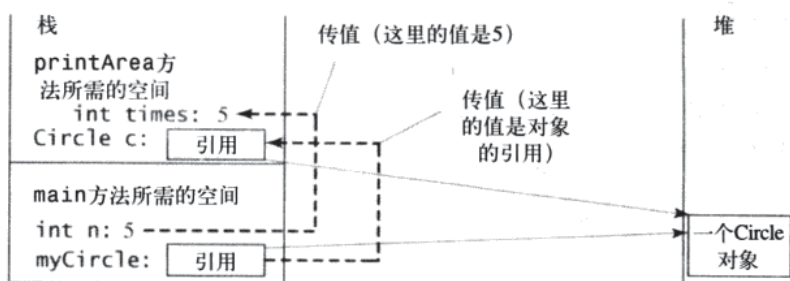


图8-17 n的值被传递给times，而myCircle的引用值被传递给printAreas方法中的c

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。

8.11 对象数组

在第6章中创建的是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8-18所示。circleArray引用了整个数组，circleArray[1]引用了一个Circle对象。

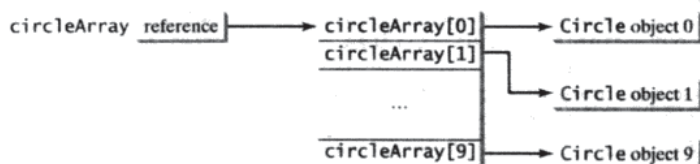


图8-18 在对象数组中，数组的每个元素都包含一个对象的引用

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。

程序清单8-12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8-12 TotalArea.java

```
1 public class TotalArea {
2     /** Main method */
3     public static void main(String[] args) {
4         // Declare circleArray
5         Circle3[] 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 Circle3[] createCircleArray() {
16        Circle3[] circleArray = new Circle3[5];
17
18        for (int i = 0; i < circleArray.length; i++) {
19            circleArray[i] = new Circle3(Math.random() * 100);
20        }
21
22        // Return Circle array
23        return circleArray;
24    }
25
26    /** Print an array of circles and their total area */
27    public static void printCircleArray(Circle3[] circleArray) {
28        System.out.printf("%-30s%-15s\n", "Radius", "Area");
29        for (int i = 0; i < circleArray.length; i++) {
30            System.out.printf("%-30f%-15f\n", circleArray[i].getRadius(),
31                               circleArray[i].getArea());
32        }
33
34        System.out.println("-----");
35
36        // Compute and display the result
37        System.out.printf("%-30s%-15f\n", "The total area of circles is",
38                            sum(circleArray));
39    }
40
41    /** Add circle areas */
42    public static double sum(Circle3[] circleArray) {
43        // Initialize sum
44        double sum = 0;
45
46        // Add areas to sum
47        for (int i = 0; i < circleArray.length; i++)
48            sum += circleArray[i].getArea();
49
50        return sum;
51    }
52 }
```

Radius	Area
70.577708	15648.941866
44.152266	6124.291736
24.867853	1942.792644
5.680718	101.380949
36.734246	4239.280350
<hr/>	
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`Circle`对象组成的数组。本章介绍了几个`Circle`类。本例使用的是8.9节中介绍的`Circle`类。

圆的半径是使用`Math.random()`方法随机生成的(第19行)。`createCircleArray`方法返回`Circle`对象构成的数组(第23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38行),该方法以`Circle`对象的数组为参数,返回的是`double`型的总面积值。

关键术语

accessor method(getter)

(读取器方法(访问器))

action(动作)

attribute(属性)

behavior(行为)

class(类)

client(客户)

constructor(构造方法)

data field(数据域)

data-field encapsulation(数据域封装)

default constructor(默认构造方法)

dot operator(·)(圆点运算符)

instance(实例)

instance method(实例方法)

instance variable(实例变量)

instantiation(实例化)

mutator method(setter)(设置器方法(修改器))

null(空值)

no-arg constructor(无参构造方法)

object-oriented programming(OOP)(面向对象程序设计)

Unified Modeling Language(UML)(统一建模语言)

package-private(or package-access)(包私有(或包访问))

private(私有的)

property(特征)

public(公共的)

reference variable(引用变量)

reference type(引用类型)

state(状态)

static method(静态方法)

static variable(静态变量)

本章小结

- 类是对象的模板。它定义对象的属性,并提供创建对象的构造方法以及对对象进行操作的方法。
- 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。
- 对象是类的实例。可以使用`new`操作符创建对象,使用点运算符(`·`)通过对象的引用变量来访问该对象的成员。
- 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。
- 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。

- 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) 方法或数据只可能在类内被访问。
- 可以提供 get 方法或者 set 方法, 使客户能够看到或修改数据。通俗点讲, get 方法称为读取器 (或访问器), set 方法称为设置器 (或修改器)。
- get 方法具有签名 `public returnType getPropertyNames()`。如果返回类型 (returnType) 是 boolean 型, 则 get 方法应该定义为 `public boolean isPropertyName()`。set 方法具有签名 `public void setPropertyName(dataType propertyValue)`。
- 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。
- Java 数组是一个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。

复习题

8.2~8.5节

8.1 描述对象和它的定义类之间的关系。如何定义一个类? 如何声明一个对象引用变量? 如何创建一个对象? 如何在一条语句中完成对象的声明和创建?

8.2 构造方法和普通方法之间的区别是什么?

8.3 数组是一个对象或一个基本类型值吗? 数组可以包含对象类型和基本类型的元素吗? 描述数组元素的默认值。

8.4 下面的程序有什么错误?

```
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)

8.5 下面的代码有什么错误?

```
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 s) {
12         this.s = s;
```

234 • 第8章 对象和类

```

13 }
14
15 public void print() {
16     System.out.print(s);
17 }
18 }

```

8.6 下面代码的输出是什么?

```

public class Foo {
    private boolean x;

    public static void main(String[] args) {
        Foo foo = new Foo();
        System.out.println(foo.x);
    }
}

```

8.6节

8.7 如何为当前时间创建一个Date? 如何显示当前时间?

8.8 如何创建一个JFrame, 如何在框架中设置一个标题以及显示这个框架?

8.9 哪些包中包含类Date、JFrame、JOptionPane、System和Math?

8.7节

8.10 假设Foo类在图a中定义, f是Foo的一个实例, 那么图b中的哪些语句是正确的?

```

public class Foo {
    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(Foo.i);
System.out.println(Foo.s);
Foo.imethod();
Foo.smethod();

```

b)

8.11 如果合适的话, 在出现?的位置添加static关键字。

```

public class Test {
    private 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;
    }
}

```

8.12 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? 下面代码的错误在哪里?

```

1 public class Foo {
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 }

```

8.8~8.9节

8.13 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？

8.14 数据域封装的优点是什么？

8.15 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，下面强调的代码会导致什么问题吗？解释为什么。

```

public class Circle {
    private double radius = 1.0;

    /** 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);
    }
}

```

8.10节

8.16 描述传递基本类型参数和传递引用类型参数的区别。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;
    }
}

```

8.17 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 = " +

```

数字水印

PDG

```

        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;
    }
}

```

8.18 显示下面代码的输出:

```

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)

8.19 下面程序的输出是什么?

```
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)

8.11节

8.20 下面代码的错误在哪里?

```
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 }
```

编程练习题

教学注意 第8~14章的练习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。

8.2~8.5节

8.1 (矩形类Rectangle) 遵从8.2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。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。依照每个矩形的宽、高、面积和周长的顺序显示。

8.2 (股票类`Stock`) 遵循8.2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。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`Java`，股票名字为`Sun Microsystems Inc`，前一日收盘价是4.5。设置新的当前值为4.35，然后显示市值变化的百分比。

8.6节

*8.3 (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，然后使用`toString()`方法显示日期。

*8.4 (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。

*8.5 (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。

8.7~8.9节

**8.6 (显示日历) 改写程序清单5-12中的`PrintCalendar`类，在消息对话框中显示日历。因为输出是由该类中的几个静态方法产生的，所以可以定义一个静态的`String`型变量`output`，用以存储输出结果，并且在消息对话框中显示它。

8.7 (账户类`Account`) 设计一个名为`Account`的类，它包括：

- 一个名为`id`的`int`类型私有账户数据域（默认值为0）。
- 一个名为`balance`的`double`类型私有账户数据域（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`类型私有数据域存储账户的开户日期。
- 一个能创建默认账户的无参构造方法。
- 一个能创建带特定`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。实现这个类。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。

8.8 (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**而值是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。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。

8.9 (几何方面：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，它定义多边形中点的x坐标，默认值为0。
- 一个名为**y**的**double**型私有数据域，它定义多边形中点的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：

$$\text{面积} = \frac{n \times s^2}{4 \times \tan\left(\frac{p}{n}\right)}$$

画出该类的UML图。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。

*8.10 (代数方面：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 。
- 一个名为**getRoot1()**和**getRoot2()**的方法返回等式的两个根：

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{and} \quad r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

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。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）。参见练习题3.1的运行示例。

*8.11（代数方面： 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：

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

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。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.”。参见练习题3.3的运行示例。

**8.12（几何方面：交点）假设两个线段相交。第一个线段的两个端点是 (x_1, y_1) 和 (x_2, y_2) ，第二个线段的两个端点是 (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。

提示 使用前面练习题中的LinearEquation类。

```
Enter the endpoints of the first line segment: 2.0 2.0 0.0
Enter the endpoints of the second line segment: 0.2 0.2 0.0
The intersecting point is: (1.0, 1.0)
```



**8.13（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的最大元素。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 45 at (1, 2)
```



字符串和文本I/O

学习目标

- 使用String类处理定长的字符串 (9.2节)。
- 使用Character类处理单个字符 (9.3节)。
- 使用StringBuilder/StringBuffer类处理可变长字符串 (9.4节)。
- 区别String、StringBuilder和StringBuffer类 (9.2~9.4节)。
- 学习如何从命令行传递参数给main方法 (9.5节)。
- 使用File类获取文件的属性、删除和重命名文件 (9.6节)。
- 使用PrintWriter类向文件写数据 (9.7.1节)。
- 使用Scanner类从文件读取数据 (9.7.2节)。
- (GUI) 使用对话框打开文件 (9.8节)。

9.1 引言

我们经常会遇到涉及字符串处理和文件输入/输出的问题。假如需要编写一个程序，该程序用一个新字替换文件中所有出现某个字的地方。该如何实现这个功能呢？本章介绍字符串和文本文件，它们可以帮助解决此类问题。（因为这里没有引入什么新的概念，所以教师可以将这章布置给学生自学。）

9.2 字符串类String

字符串 (string) 是由字符构成的一个序列。在很多语言中，字符串都被当做字符数组来处理，但是在Java中，字符串是一个对象。String类中有11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。

9.2.1 构造一个字符串

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，从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将字符串直接量看做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，术语字符串通常用来指String变量、String对象和字符串的值。

9.2.2 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9-1所示。

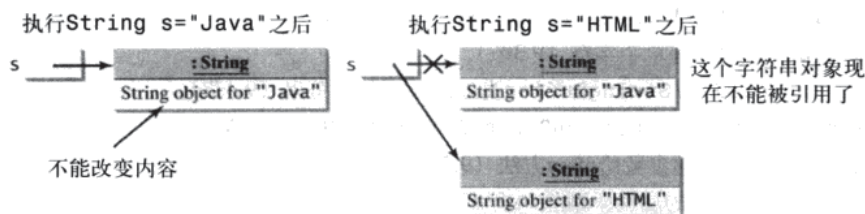
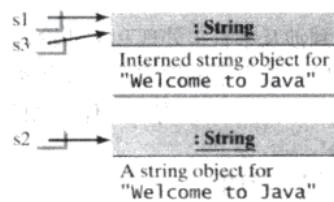


图9-1 字符串是不可变的；一旦创建，它们的内容是不能修改的

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，对具有相同字符串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）。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。

9.2.3 字符串的比较

String类提供了多种对字符串进行比较的方法，如图9-2所示。

java.lang.String
+equals(s1: String): boolean
+equalsIgnoreCase(s1: String): boolean
+compareTo(s1: String): int
+compareToIgnoreCase(s1: String): int
+regionMatches(index: int, s1: String, s1Index: int, len: int): boolean
+regionMatches(ignoreCase: boolean, index: int, s1: String, s1Index: int, len: int): boolean
+startsWith(prefix: String): boolean
+endsWith(suffix: String): boolean

如果这个字符串等于字符串s1则返回true
如果不区分大小写这个字符串等于字符串s1则返回true
返回一个大于0、等于0或小于0的整数以表明这个字符串是大于、等于还是小于s1

除了不区分大小写之外，其他都和compareTo是一样的

如果这个字符串指定的子域精确匹配字符串s1中指定的子域则返回true

除了可以指定匹配是否是区分大小写的，其他都和前一个方法一样

如果这个字符串以指定前缀开始则返回true
如果这个字符串以指定后缀结束则返回true

图9-2 String类包含多种比较字符串的方法

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，运算符==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==运算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，可以使用下面给出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 = new String("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；如果按字典序（即以统一码的顺序）s1小于s2，那么方法返回值小于0；如果按字典序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<=这样的比较运算符比较两个字符串，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、compareToIgnoreCase和regionMatches。当比较两个字符串是否相等时，方法equalsIgnoreCase和compareToIgnoreCase忽略字母的大小写。方法regionMatches比较两个字符串是否部分相等。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。

9.2.4 字符串长度、字符以及组合字符串

String类提供获取字符串长度、获取单个字符和连接字符串的方法，如图9-3所示。

java.lang.String	
+length(): int	返回这个字符串的字符个数
+charAt(index: int): char	返回这个字符串的指定下标处的字符
+concat(s1: String): String	返回连接这个字符串和字符串s1所构成的新字符串

图9-3 String类包括了获取字符串长度、获取单个字符和组合字符串的方法

可以调用字符串的length()方法获取它的长度。例如，message.length()返回字符串message的长度。

警告 `length`是String类的一个方法，但它是数组对象的一个属性。所以，要获取字符串s中的字符个数，必须使用`s.length()`，而要获取数组a中的元素个数，就必须使用`a.length`。

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到`s.length()-1`之间。例如，`message.charAt(0)`返回字符W，如图9-4所示。

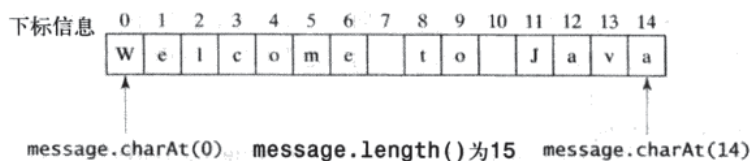


图9-4 String对象在计算机内部是用数组表示的

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".charAt(0)`是正确的，它返回W。

注意 在计算机内部，字符串的值是用私有数组变量表示的。数组是不能从String类的外部访问的。String类提供了许多像`length()`和`charAt(index)`这样的公共方法以获取该数组的信息。这是一个关于封装的很好的例子：类的数据域通过私有修饰符对用户隐藏，这样，用户就不能直接操作数据域。如果数组不是私有的，用户就能通过修改数组而改变字符串的内容，这就会违反String类不可变的原则。

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。

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(+)连接两个或更多的字符串。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，若要加号实现连接功能，至少要有一个操作数必须为字符串。

9.2.5 获取子串

可以使用`charAt`方法从字符串中获取单个字符，如图9-3所示。也可以使用String类中的`substring`方法从字符串中提取子串，如图9-5所示。

<pre>java.lang.String +substring(beginIndex: int): String +substring(beginIndex: int, endIndex: int): String</pre>	<p>返回这个字符串中以指定的<code>beginIndex</code>开始并延续到这个字符串末尾的子串，如图9-6所示</p> <p>返回这个字符串中以指定的<code>beginIndex</code>开始并延续到下标为<code>endIndex-1</code>的字符串的子串，如图9-6所示。注意，在<code>endIndex</code>处的字符不是子串的一部分</p>
--	--

图9-5 String类包含的获取子串的方法

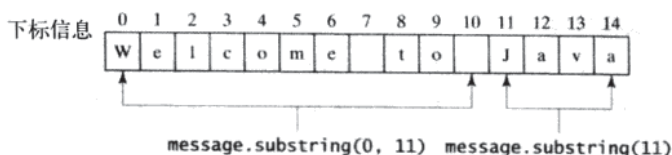


图9-6 substring方法从字符串获取子串

例如：

```
String message = "Welcome to Java".substring(0, 11) + "HTML";
```

现在，字符串message变成"Welcome to HTML"。

注意 如果beginIndex就是endIndex，那么substring(beginIndex,endIndex)就会返回一个长度为0的空字符串。如果beginIndex>endIndex，就会出现运行错误。

9.2.6 字符串的转换、替换和分隔

String类提供转换、替换和分隔字符串的方法，如图9-7所示。

java.lang.String	
+toLowerCase(): String	返回将所有字符都转换为小写之后的新字符串
+toUpperCase(): String	返回将所有字符都转换为大写之后的新字符串
+trim(): String	返回去掉两端的空白字符之后的新字符串
+replace(oldChar: char, newChar: char): String	返回用一个新字符替换这个字符串中所有和它匹配的字符的新字符串
+replaceFirst(oldString: String, newString: String): String	返回用一个新子串替换这个字符串中第一个和它匹配的子串之后的新字符串
+replaceAll(oldString: String, newString: String): String	返回用一个新子串替换这个字符串中所有和它匹配的子串之后的新字符串
+split(delimiter: String): String[]	返回用定界符分隔的子串所构成的一个字符串数组

图9-7 String类包含转换、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toLowerCase、toUpperCase、trim、repalce、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toLowerCase和toUpperCase通过将字符串中所有字符都转换成小写或大写获取一个新字符串。方法trim通过删除字符串两端的空格字符返回一个新字符串。方法replace的版本有好几个，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".toLowerCase() 返回一个新字符串welcome。

"Welcome".toUpperCase() 返回一个新字符串WELCOME。

"Welcome".trim() 返回一个新字符串Welcome。

"Welcome".replace('e','A') 返回一个新字符串WAlcoma。

"Welcome".replaceFirst("e","AB") 返回一个新字符串WABlcome。

"Welcome".replace("e","AB") 返回一个新字符串WABlcomAB。

"Welcome".replace("el","AB") 返回一个新字符串WABcome。

split方法可以从一个带特定限定符的字符串中提取标志。例如，下面的代码：

```
String[] tokens = "Java#HTML#Perl".split("#", 0);
for (int i = 0; i < tokens.length; i++)
    System.out.print(tokens[i] + " ");
```

显示

Java HTML Perl

9.2.7 依照模式匹配、替换和分隔

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，通常称为正则表达式 (regular expression)。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III.H。

以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。

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。

9.2.8 找出字符串中的某个字符或者某个子串

String类提供了几个重载的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，如图9-8所示。

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.
```



java.lang.String	
+indexOf(ch: char): int	返回字符串中第一次出现ch的下标。如果不匹配则返回-1
+indexOf(ch: char, fromIndex: int): int	返回字符串中fromIndex之后第一次出现ch的下标。如果不匹配则返回-1
+indexOf(s: String): int	返回字符串中第一次出现字符串s的下标。如果不匹配则返回-1
+indexOf(s: String, fromIndex: int): int	返回字符串中fromIndex之后第一次出现字符串s的下标。如果不匹配则返回-1
+lastIndexOf(ch: int): int	返回字符串中最后一次出现ch的下标。如果不匹配则返回-1
+lastIndexOf(ch: int, fromIndex: int): int	返回字符串中fromIndex之前最后一次出现ch的下标。如果不匹配则返回-1
+lastIndexOf(s: String): int	返回字符串中最后一次出现字符串s的下标。如果不匹配则返回-1
+lastIndexOf(s: String, fromIndex: int): int	返回字符串中fromIndex之前最后一次出现字符串s的下标。如果不匹配则返回-1

图9-8 String类包含匹配子串的方法

9.2.9 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'});
```

9.2.10 将字符和数值转换成字符串

使用静态的valueOf方法能够将字符数组转换成字符串。可以使用valueOf方法的几种重载版本将字符和数值转换成字符串，这些重载的方法的参数类型可以是char、double、long、int和float型，如图9-9所示。

java.lang.String	
+valueOf(c: char): String	返回包含字符c的字符串
+valueOf(data: char[]): String	返回包含数组中字符的字符串
+valueOf(d: double): String	返回表示double值的字符串表述
+valueOf(f: float): String	返回表示float值的字符串表述
+valueOf(i: int): String	返回表示int值的字符串表述
+valueOf(l: long): String	返回表示long值的字符串表述
+valueOf(b: boolean): String	返回表示boolean值的字符串表述

图9-9 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。

注意 可以使用Double.parseDouble(str)或者Integer.parseInt(str)将一个字符串转换成double型值或int型值。

9.2.11 格式化字符串

String类包含在String类中的静态format方法，它可以创建一个格式化的字符串。调用该方法的方法是：

```
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("%5.2f", 45.556);
```

创建一个格式化的字符串"45.56"。

9.2.12 问题：检测回文串

对于一个字符串，如果从前向后读和从后向前读都是同一个字符串，则称之为回文串。例如，单词“mom”、“dad”和“noon”都是回文串。

这里的问题是编写一个程序，提示用户输入一个字符串，然后报告该字符串是否是回文串。一种解决方案是：先判断该字符串的第一个字符和最后一个字符是否相等。如果相等，检查第二个字符和倒数第二个字符是否相等。这个过程持续进行，直到出现不匹配的情况或者串中所有的字符都检验完毕，当字符串有奇数个字符时，不用检查中间字符。

要实现这个想法，使用两个名为low和high的变量来表示字符串s开始位置和结尾位置的两个字符，如程序清单9-1所示（第22、25行）。初始状态时，low的值为0，high的值为s.length()-1。如果处在这两个位置的字符匹配，就给low加1，同时给high减1（第31~32行）。这个过程一直持续到low>=high或没有出现匹配的情况。

程序清单9-1 CheckPalindrome.java

```
1 import java.util.Scanner;
2
3 public class CheckPalindrome {
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        if (isPalindrome(s))
14            System.out.println(s + " is a palindrome");
15        else
16            System.out.println(s + " is not a palindrome");
17    }
18
19    /** Check if a string is a palindrome */
20    public static boolean isPalindrome(String s) {
21        // The index of the first character in the string
22        int low = 0;
23
24        // The index of the last character in the string
25        int high = s.length() - 1;
26    }
```

新华书店
PDG

```

27 while (low < high) {
28     if (s.charAt(low) != s.charAt(high))
29         return false; // Not a palindrome
30
31     low++;
32     high--;
33 }
34
35 return true; // The string is a palindrome
36 }
37 }

```

Enter a string: noon Enter
noon is a palindrome

Enter a string: moon Enter
moon is not a palindrome



Scanner类中的nextLine()方法(第11行)读取一行给s.isPalindrome(s),以检查s是否是一个回文串(第13行)。

9.2.13 问题: 将十六进制转换为十进制

5.7节给出了一个将一个十进制数转换为十六进制数的程序。本节给出将十六进制数转换为十进制数的程序。

假定一个十六进制数是 $h_n h_{n-1} h_{n-2} \dots h_2 h_1 h_0$, 那么等价的十进制数的值为:

$$h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + \dots + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0$$

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,

$$\begin{aligned} & h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + \dots + h_1 \times 16^1 + h_0 \times 16^0 \\ &= ((\dots((h_n \times 16 + h_{n-1}) \times 16 + h_{n-2}) \times 16 + \dots + h_1) \times 16 + h_0 \end{aligned}$$

这个发现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	hexChar	hexCharToDecimal(hexChar)	decimalValue
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9-2给出完整的程序。

程序清单9-2 HexToDecimalConversion.java


```

1 import java.util.Scanner;
2
3 public class HexToDecimalConversion {
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的。

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统一码做减法。例如，'5'-'0'是5。

9.3 字符类Character

Java为每一种基本数据类型都提供了一个包装类。这些类是Character、Boolean、Byte、Short、Integer、Long、Float和Double，它们分别对应基本类型char、boolean、byte、short、int、

long、float和double。所有这些类都在java.lang包中。它们把基本类型数据值当作对象处理。它们还包含一些有用的处理基本数据值的方法。本节介绍Character类。其他包装类将在第14章中介绍。

Character类有一个构造方法和多个确定字符类别（大写字母、小写字母、数字等）的方法，以及将大写字母转换成小写字母或者将小写字母转换成大写字母的方法，如图9-10所示。

可以用一个char值创建一个Character对象。例如，下面的语句为字符'a'创建一个Character对象。

```
Character character = new Character('a');
```

charValue方法返回包装在Character对象中的字符值。方法compareTo将该字符与另一个字符进行比较，返回一个整数值，这个整数值是该字符与另一个字符统一码的差值。当且仅当两个字符相同时，equals方法才返回true。例如，假设charObject为new Character('b')，则：

```
charObject.compareTo(new Character('a'))  返回1
charObject.compareTo(new Character('b'))  返回0
charObject.compareTo(new Character('c'))  返回-1
charObject.compareTo(new Character('d'))  返回-2
charObject.equals(new Character('b'))     返回true
charObject.equals(new Character('d'))     返回false
```

java.lang.Character	
<pre>+Character(value: char) +charValue(): char +compareTo(anotherCharacter: Character): int +equals(anotherCharacter: Character): boolean +isDigit(ch: char): boolean +isLetter(ch: char): boolean +isLetterOrDigit(ch: char): boolean +isLowerCase(ch: char): boolean +isUpperCase(ch: char): boolean +toLowerCase(ch: char): char +toUpperCase(ch: char): char</pre>	<p>使用char值构建一个字符对象 从这个对象返回char值 将这个字符和其他字符进行比较 如果这个字符等于另一个字符则返回true 如果指定字符是一个数字则返回true 如果指定字符是一个字母则返回true 如果指定字符是一个字母或者数字则返回true 如果指定字符是一个小写字母则返回true 如果指定字符是一个大写字母则返回true 返回指定字母的小写 返回指定字母的大写</p>

图9-10 Character类提供处理字符的方法

Character类中的大多数方法都是静态方法。如果字符是一个数字，方法isDigit(char ch)返回true。如果字符是一个字母，方法isLetter(char ch)返回true。如果字符是一个字母或数字，方法isLetterOrDigit(char ch)返回true。如果字符是一个小写字母，方法isLowerCase(char ch)返回true。如果字符是一个大写字母时，方法isUpperCase(char ch)返回true。方法toLowerCase(char ch)返回这个字符的小写字母，方法toUpperCase(char ch)返回这个字符的大写字母。

问题：统计字符串中的每个字母

这里的问题是编写一个程序，提示用户输入一个字符串，然后统计在忽略字母大小写的情况下，字符串中每个字母出现的次数。

下面是解决这个问题的步骤：

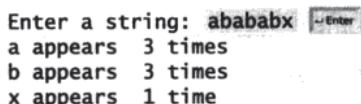
- 1) 使用String类中的toLowerCase方法将字符串中的大写字母转换成小写形式。
 - 2) 创建一个由26个int值构成的数组counts，数组的每个元素记录一个字母出现的次数。也就是说，counts[0]记录a出现的次数，counts[1]记录b出现的次数，依此类推。
 - 3) 对字符串中的每一个字符，判断其是否为小写字母。如果是，则给数组中对应的计数器加1。
- 程序清单9-3给出完整的程序。

程序清单9-3 CountEachLetter.java

```

1 import java.util.Scanner;
2
3 public class CountEachLetter {
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        // Invoke the countLetters method to count each letter
14        int[] counts = countLetters(s.toLowerCase());
15
16        // Display results
17        for (int i = 0; i < counts.length; i++) {
18            if (counts[i] != 0)
19                System.out.println((char)('a' + i) + " appears " +
20                    counts[i] + ((counts[i] == 1) ? " time" : " times"));
21        }
22    }
23
24    /** Count each letter in the string */
25    public static int[] countLetters(String s) {
26        int[] counts = new int[26];
27
28        for (int i = 0; i < s.length(); i++) {
29            if (Character.isLetter(s.charAt(i)))
30                counts[s.charAt(i) - 'a']++;
31        }
32
33        return counts;
34    }
35 }

```



```

Enter a string: abababx
a appears 3 times
b appears 3 times
x appears 1 time

```



main方法读取一行（第11行），然后调用countLetters方法（第14行）统计每个字母在该字符串中出现的次数。由于字母的大小写是忽略的，因此，程序使用toLowerCase方法将字符串转换成所有字母都是小写字母，然后将这个新串传递给countLetters方法。

countLetters方法（第25~34行）返回一个26个元素构成的数组，每个元素统计字符串s中一个字母出现的次数。该方法处理字符串中的每个字符。如果字符为字母，其对应的计数器增加1。例如，如果字符（s.charAt(i)）为'a'，则对应的计数器为counts['a'-'a']（也就是counts[0]）。如果字符为'b'，则对应的计数器为counts['b'-'a']（也就是counts[1]），因为'b'的统一码值比'a'的统一码大1。如果字符为'z'，则对应的计数器为counts['z'-'a']（也就是counts[25]），因为'z'的统一码值比'a'的统一码大25。

9.4 StringBuilder/StringBuffer类

StringBuilder/StringBuffer类是可以替代String类的另一种处理字符串的解决方案。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`；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`。可以用`StringBuffer`替换`StringBuilder`。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多个用于管理生成器或修改生成器内字符串的方法。可以使用构造方法创建一个空的字符串生成器或从一个字符串创建一个字符串生成器，如图9-11所示。

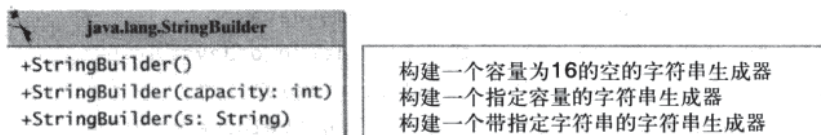


图9-11 `StringBuilder`类包含创建`StringBuilder`实例的构造方法

9.4.1 修改`StringBuilder`中的字符串

可以使用图9-12中列出的方法，在字符串生成器的末尾追加新内容，在字符串生成器的特定位置插入新的内容，还可以删除或替换字符串生成器中的字符：

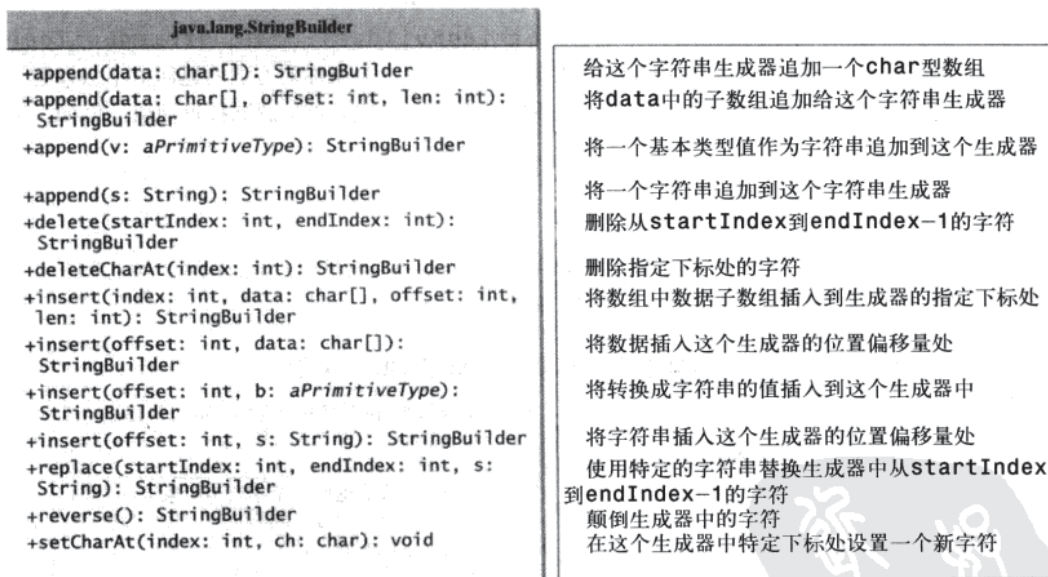


图9-12 `StringBuilder`类包含修改字符串生成器的方法

`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生成器。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生成器。考虑下面的代码:

```
stringBuilder.insert(11, "HTML and ");
```

insert

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`方法将字符从生成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)`将生成器变为Welcome Java。

`stringBuilder.deleteCharAt(8)`将生成器变为Welcome o Java。

`stringBuilder.reverse()`将生成器变为javaJ ot emocleW。

`stringBuilder.replace(11,15,"HTML")`将生成器变为Welcome to HTML。

`stringBuilder.setCharAt(0,'w')`将生成器变为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:

- 改变字符串生成器的内容。
- 返回字符串生成器的引用。

例如,下面的语句:

```
StringBuilder stringBuilder1 = stringBuilder.reverse();
```

将生成器中的字符倒置并把生成器的引用赋值给`stringBuilder1`。这样, `stringBuilder`和`stringBuilder1`都指向同一个`StringBuffer`对象。回顾一下,如果你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。

9.4.2 toString、capacity、length、setLength和charAt方法

`StringBuffer`类提供了许多其他处理字符串生成器和获取它的属性的方法,如图9-13所示。

java.lang.StringBuilder	
<pre>+toString(): String +capacity(): int +charAt(index: int): char +length(): int +setLength(newLength: int): void +substring(startIndex: int): String +substring(startIndex: int, endIndex: int): String +trimToSize(): void</pre>	<p>从字符串生成器返回一个字符串对象 返回这个字符串生成器的容量 返回指定下标处的字符 返回这个生成器中的字符个数 在这个生成器中设置新的长度 返回从startIndex开始的子串 返回从startIndex到endIndex-1的子串 减少字符串生成器所使用的存储大小</p>

图9-13 `StringBuilder`类包括修改字符串生成器的方法

`capacity()`方法返回字符串生成器当前的容量。容量是指在不增加生成器大小的情况下,能够存储的字符数量。

`length()`方法返回字符串生成器中实际存储的字符数量。`setLength(newLength)`方法设置字符

串生成器的长度。如果参数newLength小于字符串生成器的当前长度,则字符串生成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,则给字符串生成器追加足够多的null字符('    '),使其长度length变成新参数newLength。参数newLength必须大于等于0。

charAt(index)方法返回字符串生成器中下标为某个特定下标index的字符。下标是基于0的,字符串生成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,并且小于字符串生成器的长度。

注意 字符串的长度总是小于或等于生成器的容量。长度是存储在生成器中的字符串的实际大小,而容量是生成器的当前大小。如果有更多的字符添加到字符串生成器,超出它的容量,则生成器的容量就会自动增加。在计算机内部,字符串生成器是一个字符数组,因此,生成器的容量就是数组的大小。如果超出生成器的容量,就用新的数组替换现有数组。新数组的大小为 $2 \times (\text{前一个数组的长度} + 1)$ 。

提示 可以使用new StringBuilder(initialCapacity)创建指定初始容量的StringBuilder。仔细选择初始容量,能够使程序更有效。如果容量总是超过生成器的实际长度,JVM将永远不需要为生成器重新分配内存。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。

9.4.3 问题:忽略既非字母又非数字的字符,判断回文串

程序清单9-1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掉非字母和非数字的字符时,它是否是一个回文串。

下面是解决这个问题的步骤:

1) 通过删除非字母和非数字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生成器,将字符串中每一个字母或数字字符添加到字符串生成器中,然后从这个生成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9-4所示。

程序清单9-4 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    // Compare 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<>cb?a 
 Ignoring nonalphanumeric characters,
 is ab<>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生成器。该方法返回生成器中的字符串。`reverse(String s)`方法(第47~52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9-1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9-4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。

9.5 命令行参数

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`TestMain`类中的`main`方法是被`A`中的方法调用的,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。

9.5.1 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，用双引号括住它们是没有必要的。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
```

它用三个字符串"First num"、alpha和53启动这个程序，其中53是一个数值字符串。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。

9.5.2 问题：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、紧随其后的一个运算符以及另一个整数。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9-14显示这个程序的示例运行。

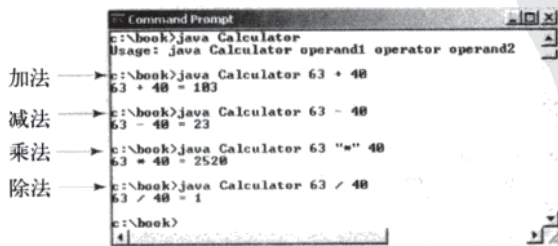


图9-14 程序从命令行获取三个参数（操作数1 运算符 操作数2），然后显示这个算术运算表达式以及算术运算的结果

传递给main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(0)`结束程序。

2) 运用`args[1]`中指定的运算符完成对操作数`args[0]`和`args[2]`的二元运算。

这个程序如程序清单9-5所示。

程序清单9-5 **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。

注意 在运行示例中，必须在命令行中用`"*"`代替`*`：

```
java Calculator 63 "*" 40
```

符号`*`用于命令行时，表示当前目录下的所有文件。所以，为了明确说明乘法运算符，在命令行中必须用双引号括住符号`*`。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]);
    }
}

```

9.6 文件类File

存储在变量、数组和对象中的数据是暂时的，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光盘上的文件中。这些文件可以传送，也可以随后被其他程序使用。由于数据存储在文件中，所以本节就介绍如何使用File类获取文件的属性以及删除和重命名文件。9.7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。

File类特意提供了一种抽象，这种抽象是指以不依赖机器的方式来处理很多文件和路径名依赖机器的复杂问题。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的方法，如图9-15所示。但是，File类不包含读写文件内容的方法。

java.io.File	
+File(pathname: String)	为特定路径名创建一个File对象。这里的路径名可以是一个目录也可以是一个文件
+File(parent: String, child: String)	为目录parent下的child创建一个File对象。这个child可以是一个文件名也可以是一个子目录
+File(parent: File, child: String)	为目录parent下的child创建一个File对象。这个parent是一个File对象。
+exists(): boolean	在前面的构造方法中，parent是一个字符串
+canRead(): boolean	如果File对象表示的文件或目录存在则返回true
+canWrite(): boolean	如果File对象表示的文件存在且可读则返回true
+isDirectory(): boolean	如果File对象表示的文件存在且可写则返回true
+isFile(): boolean	如果File对象表示一个目录则返回true
+isAbsolute(): boolean	如果File对象表示一个文件则返回true
+isHidden(): boolean	如果使用绝对路径名创建一个File对象则返回true
+getAbsolutePath(): String	如果File对象中表示的文件被隐藏则返回true。被隐藏的确切定义是依赖系统的。在Windows系统中，可以在文件属性对话框中标记该文件为隐藏的。在UNIX系统中，如果文件名以句号字符"."开始则隐藏该文件
+getCanonicalPath(): String	返回File对象表示的完整的路径名或目录名
+getName(): String	除了它能从路径名中删除像"."和".."这样的冗余名，能解析符号链接（在UNIX平台上），能将驱动器号转换为标准的大写字母（在Windows平台上），返回的都是和getAbsolutePath()一样的
+getPath(): String	返回File对象表示的完整的路径名和文件名的最后一个名字。例如：new File("c:\\book\\test.dat").getName()返回test.dat
+getParent(): String	返回File对象表示的完整的路径名和文件名。例如：new File("c:\\book\\test.dat").getPath()返回c:\\book\\test.dat
+lastModified(): long	返回File对象表示的当前路径名和文件名的完整父目录。例如：new File("c:\\book\\test.dat").getParent()返回c:\\book
+length(): long	返回文件最后一次修改的时间
+listFile(): File[]	返回文件的大小，如果文件不存在则返回0，如果是目录则返回目录大小
+delete(): boolean	返回一个File对象目录下的文件
+renameTo(dest: File): boolean	删除这个文件。如果删除成功则该方法返回true
	重命名这个文件。如果操作成功则该方法返回true

图9-15 File类可以用来获取文件和目录属性以及删除和重命名文件

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2-6）。

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。

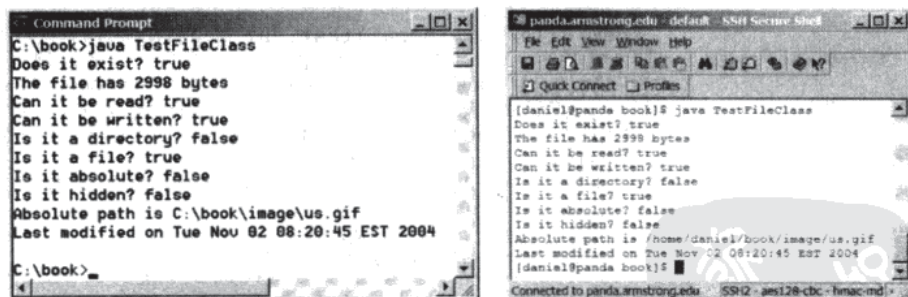
程序清单9-6演示如何创建一个`File`对象，以及如何使用`File`类中的方法获取它的属性。这个程序为文件`us.gif`创建了一个`File`对象。这个文件存储在当前目录的`image`目录下。

程序清单9-6 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9-16a显示程序在Windows平台上的运行示例，而图9-16b显示程序在UNIX平台上的运行示例。如图9-16所示，Windows平台上和UNIX平台的路径命名习惯是不一样的。



a) 在Windows平台上

b) 在UNIX平台上

图9-16 程序创建一个`File`对象然后显示文件属性

9.7 文件输入和输出

`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，也不包括从（向）文件读（写）数据的方法。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。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。

9.7.1 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9-17总结了PrintWriter中的常用方法。

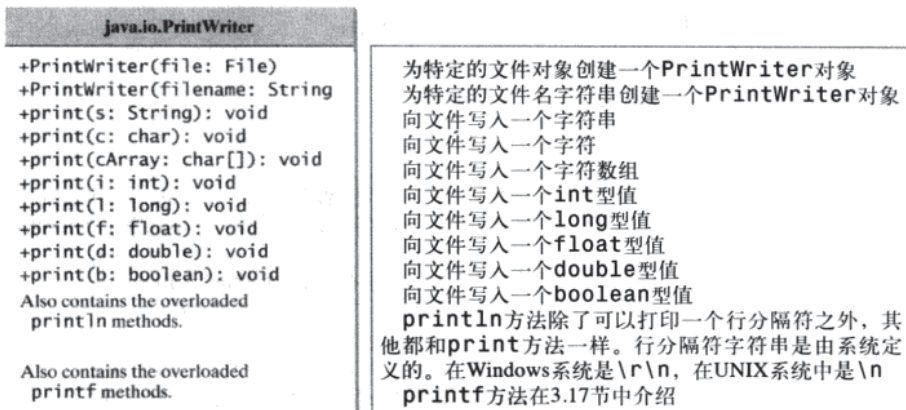


图9-17 PrintWriter类包括将数据写入文本文件的方法

程序清单9-7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9-7 WriteData.java

```
1 public class WriteData {
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         // 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3~7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。目前，只要在方法头声明中声明throws Exception（第2行）即可。

你已经使用过System.out.print和System.out.println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。如果没有调用该方法,数据就不能正确地保存在文件中。

9.7.2 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有用信息。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9-18总结出`Scanner`中的常用方法。

java.util.Scanner	
+Scanner(source: File)	创建一个所产生的值都是从特定文件扫描而来的扫描器
+Scanner(source: String)	创建一个所产生的值都是从特定字符串扫描而来的扫描器
+close()	关闭这个扫描器
+hasNext(): boolean	如果这个扫描器还有可读的数据则返回true
+next(): String	从这个扫描器返回下一个标志作为字符串
+nextLine(): String	使用行分隔符从这个扫描器返回一个行结束
+nextByte(): byte	从这个扫描器返回下一个标志作为一个byte值
+nextShort(): short	从这个扫描器返回下一个标志作为一个short值
+nextInt(): int	从这个扫描器返回下一个标志作为一个int值
+nextLong(): long	从这个扫描器返回下一个标志作为一个long值
+nextFloat(): float	从这个扫描器返回下一个标志作为一个float值
+nextDouble(): double	从这个扫描器返回下一个标志作为一个double值
+useDelimiter(pattern: String): Scanner	设置这个扫描器的分隔模式并返回这个扫描器

图9-18 Scanner类包含扫描数据的方法

程序清单9-8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9-8 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 }
```

scores.txt

John	T	Smith	90
Eric	K	Jones	85

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。

9.7.3 Scanner如何工作

方法`nextByte()`、`nextShort()`、`nextInt()`、`nextLong()`、`nextFloat()`、`nextDouble()`和`next()`等都称为令牌读取方法 (token-reading method), 因为它们会读取用分隔符分隔开的令牌。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? 一个令牌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令牌。然后, 对应于`nextByte()`、`nextShort()`、`nextInt()`、`nextLong()`、`nextFloat()`和`nextDouble()`, 这个令牌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。如果令牌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()`读取一个以行分隔符结束的行。

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。

令牌读取方法不能读取令牌后面的分隔符。如果在令牌读取方法之后调用`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。令牌读取方法`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。

9.7.4 问题: 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9-9给出这个问题的解决方案。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28行), 为目标文件创建一个PrintWriter, 然后重复从源文件读入一行 (第32行), 替换文本 (第33行), 向目标文件中写入新的一行 (第34行)。必须关闭输出文件 (第38行), 以确保数据正确地保存在文件中。

程序清单9-9 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(0);
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(0);
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(0);
25        }
26
27        // Create a Scanner for input and a PrintWriter for output
28        Scanner input = new Scanner(sourceFile);
29        PrintWriter output = new PrintWriter(targetFile);
30
31        while (input.hasNext()) {
32            String s1 = input.nextLine();
33            String s2 = s1.replaceAll(args[2], args[3]);
34            output.println(s2);
35        }
36
37        input.close();
38        output.close();
39    }
40 }
```

9.8 (GUI) 文件对话框

Java提供javax.swing.JFileChooser类来显示文件对话框, 如图9-19所示。在这个对话框中, 用户可以选择一个文件。

程序清单9-10给出的程序提示用户选择一个文件, 然后在控制台上显示它的内容。

程序清单9-10 ReadFileUsingJFileChooser.java

```

1 import java.util.Scanner;
2 import javax.swing.JFileChooser;
3
4 public class ReadFileUsingJFileChooser {
5     public static void main(String[] args) throws Exception {
6         JFileChooser fileChooser = new JFileChooser();
7         if (fileChooser.showOpenDialog(null)
8             == JFileChooser.APPROVE_OPTION) {
9             // Get the selected file
10            java.io.File file = fileChooser.getSelectedFile();
11
12            // Create a Scanner for the file
13            Scanner input = new Scanner(file);
14
15            // Read text from the file
16            while (input.hasNext()) {
17                System.out.println(input.nextLine());
18            }
19
20            // Close the file
21            input.close();
22        }
23        else {
24            System.out.println("No file selected");
25        }
26    }
27 }

```

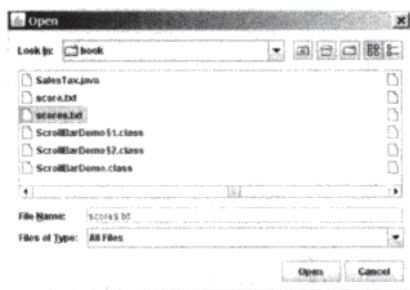


图9-19 JFileChooser可以用来显示一个打开某文件的文件对话框

程序在第6行创建一个JFileChooser。showOpenDialog(null)方法显示一个对话框，如图9-19所示。这个方法返回一个int型值，或者是APPROVE_OPTION或者是CANCEL_OPTION，它们分别表明点击Open按钮或者点击Cancel按钮。

getSelectedFile()方法（第10行）返回从文件对话框中选中的文件。第13行创建该文件的一个scanner。程序不断地从文件读取一行，然后将它们显示在控制台（第16~18行）。

本章小结

- 字符串是封装在String类中的对象。要创建一个字符串，可以使用11种构造方法之一，也可以使用字符串直接量进行简洁初始化。
- 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。
- 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。

- 可以使用concat方法连接两个字符串，或者使用加号(+)连接两个或两个以上的字符串。
- 可以使用substring方法从字符串中提取子串。
- 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。
- Character类是单个字符的包装类。Character类提供很多实用的静态方法，用于判断一个字符是否是字母(isLetter(char))、数字(isDigit(char))、大写字母(isUpperCase(char))或小写字母(isLowerCase(char))。
- StringBuilder/StringBuffer类可以用来替代String类。String对象是不可改变的，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；如果需要改变，则使用StringBuilder/StringBuffer类。
- 可以从命令行向main方法传递字符串。传递给main程序的字符串存储在一个字符串数组args中。第一个字符串用args[0]表示，而arg.length表示传入的字符串的个数。
- File类用来获取文件的属性和对文件进行操作。它不包括创建文件，以及从(向)文件读(写)数据的方法。
- 可以使用Scanner从文本文件中读取字符串和基本类型数据值，使用PrintWriter创建一个文件并向文本文件中写入数据。
- 可以使用JFileChooser类以图形的形式显示文件。

复习题

9.2节

9.1 假设s1、s2、s3和s4是四个字符串，如下给出其值：

```
String s1 = "Welcome to Java";
String s2 = s1;
String s3 = new String("Welcome to Java");
String s4 = "Welcome to Java";
```

那么下面表达式的结果是什么？

- | | |
|------------------------------|--------------------------------|
| (1) s1 == s2 | (13) s1.length() |
| (2) s2 == s3 | (14) s1.substring(5) |
| (3) s1.equals(s2) | (15) s1.substring(5, 11) |
| (4) s2.equals(s3) | (16) s1.startsWith("Wel") |
| (5) s1.compareTo(s2) | (17) s1.endsWith("Java") |
| (6) s2.compareTo(s3) | (18) s1.toLowerCase() |
| (7) s1 == s4 | (19) s1.toUpperCase() |
| (8) s1.charAt(0) | (20) " Welcome ".trim() |
| (9) s1.indexOf('j') | (21) s1.replace('o', 'T') |
| (10) s1.indexOf("to") | (22) s1.replaceAll("o", "T") |
| (11) s1.lastIndexOf('a') | (23) s1.replaceFirst("o", "T") |
| (12) s1.lastIndexOf("o", 15) | (24) s1.toCharArray() |

为了创建一个字符串“Welcome to Java”，可能会用到如下所示的语句：

```
String s = "Welcome to Java";
```

或者

```
String s = new String("Welcome to Java");
```

哪个语句更好？为什么？

9.2 假设s1和s2是两个字符串，下面哪些语句或表达式是错误的？

```
String s = new String("new string");
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());

```

9.3 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);

```

9.4 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。
- 检查在忽略大小写的情况下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值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值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。
- 创建一个新字符串s3,它是s1和s2的组合。
- 创建一个s1的子串,下标从1开始。
- 创建一个s1的子串,下标从1到4。
- 创建一个新字符串s3,它将s1转换为小写。
- 创建一个新字符串s3,它将s1转换为大写。
- 创建一个新字符串s3,它将s1的两端的空格去掉。
- 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s3。
- 将"Welcome to Java and HTML"按空格分隔为一个数组tokens。
- 将s1中字符e第一次出现的下标赋值给一个int型变量x。
- 将s1中字符串abc最后一次出现的下标赋值给一个int型变量x。

9.5 String类中是否有可以改变字符串内容的方法?

9.6 假设字符串s是用new String()创建的,那么s.length()是多少?

9.7 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?

9.8 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 }

```

9.9 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     this.text = s;
6 }
7
8 public static void main(String[] args) {
9     Test test = new Test("ABC");
10    System.out.println(test);
11 }
12 }

```

9.3节

9.10 怎样判断一个字母是大写还是小写?

9.11 怎样判断一个字符是否为字母或数字?

9.4节

9.12 `StringBuilder`和`StringBuffer`之间的区别是什么?

9.13 如何为一个字符串创建字符串生成器? 如何从一个字符串生成器获取字符串?

9.14 使用`StringBuilder`类中的`reverse`方法编写三条语句, 倒置字符串`s`。

9.15 编写一条语句, 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。

9.16 字符串和字符串生成器的内部结构是什么?

9.17 假设给出如下所示的`s1`和`s2`:

```

StringBuilder s1 = new StringBuilder("Java");
StringBuilder s2 = new StringBuilder("HTML");

```

显示执行每条语句之后`s1`的结果。假定这些表达式都是相互独立的。

(1) <code>s1.append(" is fun");</code>	(7) <code>s1.deleteCharAt(3);</code>
(2) <code>s1.append(s2);</code>	(8) <code>s1.delete(1, 3);</code>
(3) <code>s1.insert(2, "is fun");</code>	(9) <code>s1.reverse();</code>
(4) <code>s1.insert(1, s2);</code>	(10) <code>s1.replace(1, 3, "Computer");</code>
(5) <code>s1.charAt(2);</code>	(11) <code>s1.substring(1, 3);</code>
(6) <code>s1.length();</code>	(12) <code>s1.substring(2);</code>

9.18 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");
    }
}

```

9.5节

9.19 本书声明`main`方法为:

```
public static void main(String[] args)
```

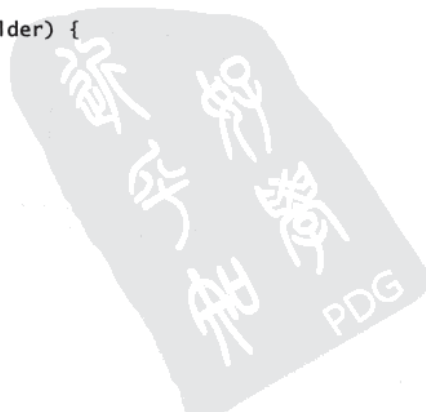
能将它替换成下面的某条语句吗?

```
public static void main(String args[])
```

```
public static void main(String[] x)
```

```
public static void main(String x[])
```

```
static void main(String x[])
```



9.20 显示当使用命令

- (1) `java Test I have a dream`
- (2) `java Test "1 2 3"`
- (3) `java Test`
- (4) `java Test "*"`
- (5) `java Test *`

调用下面的程序时，程序的输出结果。

```
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]);
    }
}
```

9.6节

9.21 使用下面的语句创建File对象时，错在哪里？

```
new File("c:\book\test.dat");
```

9.22 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？

9.23 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？

9.7节

9.24 如何创建一个PrintWriter向文件写数据？在程序清单9-7中，为什么要在main方法中声明throws Exception？在程序清单9-7中，如果不调用close()方法，将会发生什么？

9.25 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();
    }
}
```

9.26 如何创建一个Scanner从文件读数据？在程序清单9-8中，为什么要在main方法中声明throws Exception？在程序清单9-8中，如果不调用close()方法，将会发生什么？

9.27 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？

9.28 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？

9.29 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();
```

9.30 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();
```

编程练习题

9.2~9.3节

*9.1 (检查SSN) 编写程序, 提示用户输入一个社会保险号码, 它的格式是DDD-DD-DDDD, 其中D是一个数字。程序会为正确的社保号显示"Valid SSN"; 否则, 显示"Invalid SSN"。

**9.2 (检测子串) 可以使用String类中的indexOf方法检测一个字符串是否是另一个字符串的子串。编写自己的方法实现这个功能。编写一个程序, 提示用户输入两个字符串, 检测第一个字符串是否是第二个字符串的子串。

**9.3 (检验密码) 一些网站设定了一些制定密码的规则。编写一个方法, 检验一个字符串是否是合法的密码。假设密码规则如下:

- 密码必须至少有8个字符。
- 密码只能包括字母和数字。
- 密码必须至少有2个数字。

编写一个程序, 提示用户输入密码, 如果该密码符合规则就显示"Valid Password", 否则显示"Invalid Password"。

9.4 (求某个指定字符的出现次数) 使用下面的方法头编写一个方法, 找出某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') 返回2。编写一个测试程序, 提示用户输入一个字符串, 在该字符串后紧跟着一个字符, 然后显示这个字符在字符串中出现的次数。

**9.5 (在字符串中每个数字出现的次数) 使用下面的方法头编写一个方法, 统计每个数字在字符串中出现的次数:

```
public static int[] count(String s)
```

这个方法统计每个数字在字符串中出现的次数。返回值是10个元素构成的数组, 每个元素存储的是一个数字出现的次数。例如, 在执行完int[] counts = count("12203AB3")之后, counts[0]为1, counts[1]为1, counts[2]为2, counts[3]为2。

编写一个测试程序, 提示用户输入一个字符串, 然后显示每个数字在字符串中出现的次数。

*9.6 (统计字符串中字母的个数) 使用下面的方法头编写一个方法, 统计字母在字符串中出现的个数。

```
public static int countLetters(String s)
```

编写一个测试程序, 提示用户输入一个字符串, 然后显示这个字符串中字母的个数。

*9.7 (电话按键盘) 国际标准的字母/数字匹配图可以在电话上找到, 如下所示:

1	2	3
	ABC	DEF
4	5	6
GHI	JKL	MNO
7	8	9
PQRS	TUV	WXYZ
	0	

编写一个方法返回给定大写字母的数字, 如下所示:

```
public static 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)翻译成数字, 然后保持其他字符不动。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



***9.8 (二进制转换为十进制)** 编写一个方法，将一个二进制数字作为一个字符串转换为一个十进制整数。这个方法头如下所示：

```
public static int binaryToDecimal(String binaryString)
```

例如，二进制字符串10001表示17 ($1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 = 17$)。所以，`binaryToDecimal("10001")`返回的是17。注意，`Integer.parseInt("10001", 2)`也可以将一个二进制字符串转变成十进制的数值。在这个练习中不要使用该方法。

编写一个测试程序，提示用户输入一个二进制数，然后显示对应的十进制整数值。

9.4节

****9.9 (将二进制数转换为十六进制数)** 编写一个方法，将一个二进制数转换为一个十六进制数。方法头如下所示：

```
public static String binaryToHex(String binaryValue)
```

编写一个测试程序，提示用户输入一个二进制数，然后显示对应的十六进制数。

****9.10 (将十进制数转换为二进制数)** 编写一个方法，将一个十进制数转换为一个二进制数。方法头如下所示：

```
public static String decimalToBinary(int value)
```

编写一个测试程序，提示用户输入一个十进制整数，然后显示对应的二进制数。

****9.11 (对字符串中的字符排序)** 使用下面的方法头编写一个方法，返回排好序的字符串

```
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，然后显示排好序的字符串。

****9.12 (变位词)** 编写一个方法，检测两个单词是否互为变位词。如果在不计顺序的情况下两个单词包含完全相同的字母，则称这两个单词互为变位词 (anagram)。例如，"silent"和"listen"互为变位词。该方法的方法头如下所示：

```
public static boolean isAnagram(String s1, String s2)
```

编写一个测试程序，提示用户输入两个字符串，如果它们是变位词，则显示"anagram"，否则显示"not anagram"。

9.5节

***9.13 (传递字符串检测回文串)** 改写程序清单9-1，将被检测的字符串以命令行参数的方式传入。

***9.14 (求整数的和)** 编写两个程序。第一个程序给main方法传入个数不定的整数，每个整数都是一个独立的字符串，然后显示它们的和。第二个程序给main方法传入个数不定的整数构成的同一个字符串，数字之间被一个空格分隔，然后显示它们的和。将这两个程序分别命名为Exercise9_14a和Exercise9_14b，如图9-20所示。

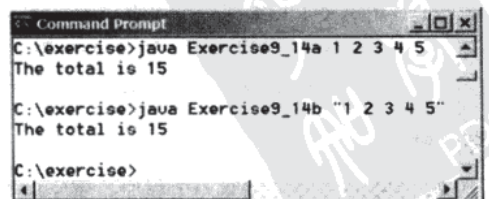


图9-20 程序对从命令行传来的所有数字求和

*9.15 (求字符串中大写字母的个数) 编写一个程序, 传给main方法一个字符串, 显示该字符串中大写字母的个数。

9.7~9.8节

**9.16 (改变Java源代码的格式) 编写一个程序, 将次行块风格的Java源代码转换成行尾块风格。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 Exercise9_16 Test.java
```

*9.17 (统计一个文件中的字符数、单词数和行数) 编写程序统计一个文件中的字符数(控制字符'\r'和'\n'除外)、单词数以及行数。单词由空格、制表符、回车键或换行符分隔, 文件名应该作为命令行参数被传递, 如图9-21所示。

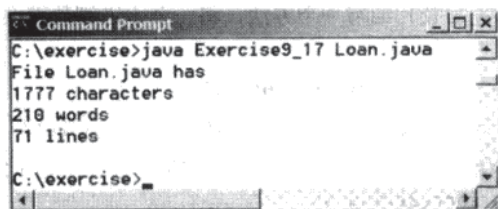


图9-21 程序显示给定文件中的字符数、单词数和行数

*9.18 (处理文本文件中的分数) 假定一个名为Exercise9_18.txt的文本文件中包含未指定个数个分数。编写一个程序, 从文件中读入分数, 显示它们的和以及平均值。分数之间用空格分开。

*9.19 (写/读数据) 编写一个程序, 如果名为Exercise9_19.txt的文件不存在, 则创建该文件。使用文本I/O编写随机产生100个整数给文件。文件中的整数由空格分开。从文件中读回数据然后显示排好序的数据。

**9.20 (替换文本) 程序清单9-9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 Exercise9_20 file oldString newString
```

用newString代替源文件中的oldString。

**9.21 (删除文本) 编写一个程序, 从一个文本文件中删掉所有出现某个指定字符串的地方。例如, 调用

```
java Exercise9_21 John filename
```

从指定文件中删掉字符串John。

综合题

9.22 (猜测首府) 编写一个程序, 重复地提示用户输入一个州的首府, 如图9-22a所示。一旦接收到用户的输入, 程序就会报告这个答案是否正确, 如图9-22b所示。假设有一个二维的数组存储了55个州和它们的首府, 如图9-23所示。程序提示用户回答所有10个州的首府, 然后显示答对的总次数。

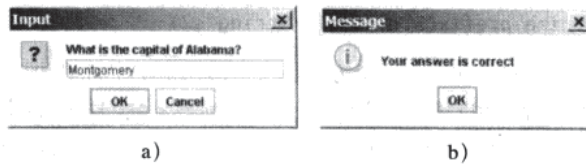


图9-22 程序提示用户在图a中输入首府，然后报告答对的次数

Alabama	Montgomery
Alaska	Juneau
Arizona	Phoenix
...	...
...	...

图9-23 二维数组存储州名和它们的首府

****9.23 (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);
```

****9.24 (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);
```

****9.25 (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。

****9.26 (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();
```

****9.27 (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();
```

***9.28 (相同的前缀)** 编写一个方法，返回两个字符串共有的前缀。例如，"distance"和"disinfection"的共同前缀是"dis"。方法头如下所示:

```
public static String prefix(String s1, String s2)
```

如果两个字符串没有共同的前缀，这个方法就会返回一个空字符串。

编写一个main方法，提示用户输入两个字符串，然后显示它们共同的前缀。

****9.29**（新字符串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、?、b、gf、#和e构成的字符串数组。

****9.30**（财务问题：信用卡号的合法性）使用信用卡号码作为字符串输入改写练习题5.31。使用下面的方法重新设计这个程序：

```
/** Return true if the card number is valid */
public static boolean isValid(String cardNumber)

/** Get the result from Step 2 */
public static int sumOfDoubleEvenPlace(String cardNumber)

/** 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(String cardNumber)
```

*****9.31**（游戏：刽子手）编写一个刽子手游戏，随机产生一个单词，然后提示用户一次猜测一个字母，如运行示例所示。单词中的每个字母都以星号显示。当用户的猜测准确时，就显示实际的字母。当用户完成了一个单词，就显示猜错的次数，然后询问用户是否继续猜测另一个单词。声明一个数组来存储这些单词，如下所示：

```
// Use any words you wish
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*** > p
      p is already in the word
(Guess) Enter a letter in word pr*** > 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 for another word? Enter y or n>
```



****9.32**（检测ISBN）使用字符串操作来简化练习题3.9。以字符串形式输入一个ISBN的前9个数字。

*****9.33**（游戏：刽子手）改写练习题9.31。程序读取存储在一个名为Exercise9_33.txt的文本文件中的单词。这些单词用空格分隔。

****9.34**（替换文本）修改练习题Exercise9_20，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 Exercise9_34 dir oldString newString
```


*9.35 (生物信息方面: 找出基因) 生物学家使用字母A、C、T和G构成的序列来对基因组进行建模。基因是基因组的一个子串, 基因组在三字符ATG之后开始, 在三字符TAG、TAA和TGA之前结束。因此, 基因字符串的长度是3的倍数, 而基因不包含任何ATG、TAG、TAA和TGA这样的三字符。

编写一个程序, 提示用户输入一个基因组, 然后显示基因组中所有的基因。如果在输入序列中没有找到任何基因, 就显示无基因。下面是一些运行示例:

```
Enter a genome string: TTATGTTTTAAGGATGGGCGTTAGTT
TTT
GGGCGT
```



```
Enter a genome string: TGTGTGTATAT
no gene is found
```



歡迎
光臨
PDG

第10章

Introduction to Java Programming, 8E

关于对象的思考

学习目标

- 从不可变类创建不可变对象以保护对象的内容（10.2节）。
- 判断类中出现的变量的作用域（10.3节）。
- 使用关键字 `this` 指代调用对象自己（10.4节）。
- 应用类的抽象来开发软件（10.5节）。
- 探索面向过程范式和面向对象范式的不同之处（10.6节）。
- 开发用于建模对象之间组合关系的类（10.7节）。
- 使用面向对象范式设计程序（10.8~10.10节）。
- 按照类的设计原则来设计类（10.11节）。

10.1 引言

前面两章已经介绍了对象和类。你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：`Date`、`Random`、`String`、`StringBuilder`、`File`、`Scanner`、`PrintWriter`）。本书的方法是在教授面向对象程序设计之前，先讲述解决问题的技术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以及如何使用这些类。下面首先介绍一些支持这些例子的语言特征。

10.2 不可变对象和类

通常，创建一个对象后，它的内容是允许随后改变的。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8-9中`Circle`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，它必须满足下面的需求：

- 所有数据域都是私有的。
- 没有修改器方法。
- 没有一个访问器方法，它会返回一个指向可变数据域的引用。

10.3 变量的作用域

在第5章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。不管在何处声明，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10-1a所示。当一个数据域是基于对另一个数据域的引用来进行初始化的情况例外。在这种情况下，必须首先声明另一个数据域，如图10-1b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;
}

```

```

public class Foo {
    private int i;
    private int j = i + 1;
}

```

a) 可以按任意顺序声明变量`radius`和方法`findArea()`

b) `i`必须在`j`之前声明，因为`j`的初始值依赖于`i`

图10-1 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 Foo {
    private int x = 0; // Instance variable
    private int y = 0;

    public Foo() {
    }

    public void p() {
        int x = 1; // Local variable
    }
}

```

```

        System.out.println("x = " + x);
        System.out.println("y = " + y);
    }
}

```

假设f是Foo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。

10.4 this引用

关键字this是指向调用对象本身的引用名。一种常见的用法就是引用类的隐藏数据域 (hidden data field)。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10-2a所示。

```

public class Foo {
    int i = 5;
    static double k = 0;

    void setI(int i) {
        this.i = i;
    }

    static void setK(double k) {
        Foo.k = k;
    }
}

```

a)

Suppose that f1 and f2 are two objects of Foo.

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

b)

图10-2 关键字this是指调用方法的对象

关键字this给出一种指代对象的方法，这样，可以在实例方法代码中调用实例方法。this.i=i这一行的意思是“将参数i的值赋给调用对象的数据域i”。关键字this是指调用实例方法setI的对象，如图10-2b所示。Foo.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。

关键字this的另一个常用方法是让构造方法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;
    }
    // 显式使用this来引用被创建对象的数据域radius

    public Circle() {
        this(1.0);
    }
    // 使用this调用另一个构造方法

    public double getArea() {
        return this.radius * this.radius * Math.PI;
    }
}

```

每个实例变量都属于一个this表示的实例，通常这个this是被忽略的

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。

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。

注意 Java要求在构造方法中，语句`this`（参数列表）应在任何其他语句之前出现。

10.5 类的抽象和封装

在第5章中，已经学习了方法的抽象，以及如何在程序开发中使用它。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。类的创建者提供类的描述，让使用者明白如何才能使用类。从类外可以访问的全部方法和数据域，以及期望这些成员如何行动的描述，合称为类的合约（class's contract）。如图10-3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。

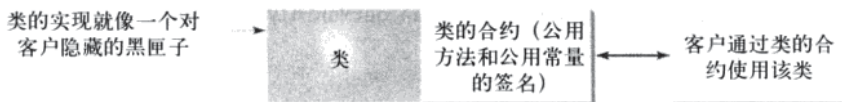


图10-3 类抽象将类的实现与类的使用分离

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。每个组件都可以看做是一个有属性和方法的对象。要使各个组件一起工作，只需要知道每个部件是怎么用的，以及如何与其他部件进行交互的。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，对你隐藏。所以，你可以组装一台计算机，而不需要了解每个组件是如何实现功能的。

对计算机系统的模拟准确地反映了面向对象方法。每个部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做贷款类`Loan`的一个对象，利率、贷款额以及还贷时间都是它的数据属性，计算每月偿还金额和总偿还金额是它的方法。当你购买一辆汽车时，就用贷款利率、贷款额和还贷时间实例化这个类，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2-8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。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，这个解决方案是有限的。假设希望将一个数据和这个贷款联系起来，最理想的方法是创建一个对象，将这个数据和关于贷款信息的属性绑定在一起。图10-4给出`Loan`类的UML类图。

在图10-4中的UML图被当做是`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。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。我们从编写一个使用`Loan`类的测试类开始。

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 yearly interest rate
10  System.out.print(
11      "Enter yearly 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 yearly interest rate, for example, 8.25: 2.5

Enter number of years as an integer: 5

Enter loan amount, for example, 120000.95: 1000

The loan was created on Sat Jun 10 21:12:50 EDT 2006

The monthly payment is 17.74

The total payment is 1064.84



Loan	
-annualInterestRate: double	
-numberOfYears: int	
-loanAmount: double	
-loanDate: java.util.Date	
+Loan()	
+Loan(annualInterestRate: double, numberOfYears: int, loanAmount: double)	
+getAnnualInterestRate(): double	
+getNumberOfYears(): int	
+getLoanAmount(): double	
+getLoanDate(): java.util.Date	
+setAnnualInterestRate(annualInterestRate: double): void	
+setNumberOfYears(numberOfYears: int): void	
+setLoanAmount(loanAmount: double): void	
+getMonthlyPayment(): double	
+getTotalPayment(): double	

贷款的年利率（默认值：2.5）
 贷款的年数（默认值：1）
 贷款额（默认值：1000）
 产生贷款的日期

构建一个默认的Loan对象
 构建一笔带指定利率、年数和贷款额的贷款

返回这笔贷款的年利率
 返回这笔贷款的年数
 返回这笔贷款的数额
 返回产生这笔贷款的日期
 设置这笔贷款新的年利率

设置这笔贷款新的年数
 设置这笔贷款新的数额
 返回这笔贷款的月支付额
 返回这笔贷款的总支付额

图10-4 Loan类对贷款的性质和行为建模

main方法读取利率、还贷时间（以年为单位）、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额（第29行）和总偿还额（第30行）。

Loan类可以在程序清单10-2中实现。

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            (Math.pow(1 / (1 + monthlyInterestRate), numberOfYears * 12)));
58        return monthlyPayment;
59    }
60 }
```

数字水印
PDG

```

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、属性和方法提供不同方式的定制。

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。三个get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10-4所示。学生应该从编写使用Loan类的测试程序开始,即使它们不知道这个Loan类是如何执行的。它有三个优点:

- 1) 它演示开发类和使用类是两个不同的任务。
- 2) 它能使你跳过某个类的复杂实现,而不干扰整本书的顺序。
- 3) 如果你通过使用它熟悉了类,那么你更容易学会如何实现一个类。

从现在开始的所有例子,你都可以先从这个类创建一个对象,然后尝试使用它的方法,之后就将注意力放在它的实现上。

10.6 面向对象的思考

第1~7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,可以看到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3-5给出计算身体质量指数的程序。它的代码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。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10-5所示。

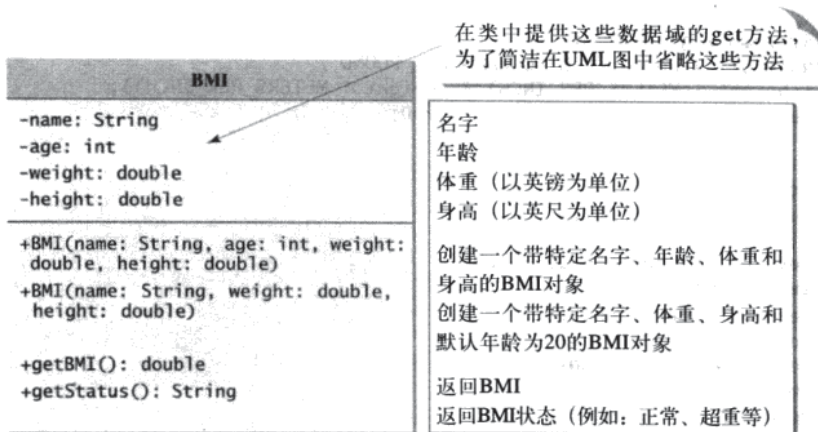


图10-5 BMI类封装BMI信息

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("John Doe", 18, 145, 70);
4         System.out.println("The BMI for " + bmi1.getName() + " is "
5             + bmi1.getBMI() + " " + bmi1.getStatus());
6
7         BMI bmi2 = new BMI("Peter King", 215, 70);
8         System.out.println("The BMI for " + bmi2.getName() + " is "
9             + bmi2.getBMI() + " " + bmi2.getStatus());
10    }
11 }

```

```

The BMI for John Doe is 20.81 normal weight
The BMI for Peter King is 30.85 seriously overweight

```



第3行为John Doe创建一个对象bmi1，而第7行为Peter 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类可以在程序清单10-4中实现。

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 < 16)
29         return "seriously underweight";
30     else if (bmi < 18)
31         return "underweight";
32     else if (bmi < 24)
33         return "normal weight";
34     else if (bmi < 29)
35         return "overweight";
36     else if (bmi < 35)
37         return "seriously overweight";
38     else
39         return "gravely overweight";
40 }
41
42 public String getName() {
43     return name;
44 }
45
46 public int getAge() {
47     return age;
48 }
49
50 public double getWeight() {
51     return weight;
52 }
53
54 public double getHeight() {
55     return height;
56 }
57 }

```

使用体重和身高来计算BMI的数学公式在3.10节中给出了。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。

实例方法**getStatus()**返回解释BMI的字符串。这个解释也在3.10节中给出了。

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中上的操作。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，而且这里的方法论要求给方法发送数据。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中的问题。面向对象程序设计方法按照镜像到真实世界的方式组织程序，在真实世界中，所有的对象都是和属性及动作相关联的。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程序设计涉及的是对对象的思考，一个Java程序可以看做是一个相互操作的对象集合。

10.7 对象的组合

一个对象可以包含另一个对象。这两个对象之间的关系称为组合（composition）。在程序清单10-4中，定义的BMI类包含了一个String数据域。BMI和String之间的关系就是组合。

组合实际上是聚集关系的一种特殊形式。聚集模拟了具有（has-a）关系，表示两个对象之间的归属

关系。归属关系中的所有者对象称为聚集对象 (aggregating object)，而它的类称为聚集类 (aggregating class)。归属关系中的从属对象称为被聚集对象 (aggregated object)，而它的类称为被聚集类 (aggregated class)。

一个对象可以被几个其他聚集对象所拥有。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。

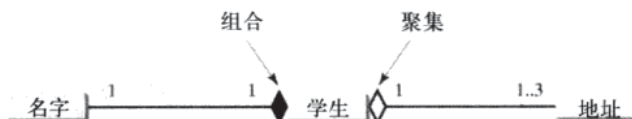
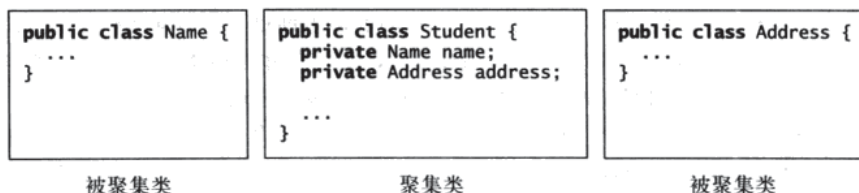


图10-6 一个学生有一个名字和一个地址

在一个关系中的每个类可以指定一个多重性 (multiplicity)。这个多重性可能是一个数字或者一个数字范围，它表明关系中要涉及的类的对象个数。字符*意味着无限多个对象，而数字范围m..n意味着对象的个数应该在m和n之间，包括m和n。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中的关系可以如下表示：



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10-7所示。

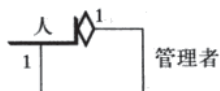


图10-7 一个人可能有一个管理者

在图10-7所示的关系“一个人可能有一个管理者”中，管理者可以如下表示为Person类的一个数据域：

```
public class Person {
    // The type for the data is the class itself
    private Person supervisor;
    ...
}
```

如果一个人有几个管理者，如图10-8a所示，可以用一个数组存储管理者，如图10-8b所示。

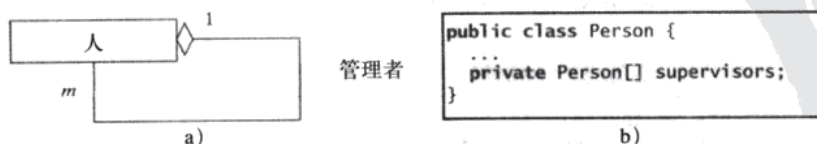


图10-8 一个人可能会有几个管理者

注意 由于聚集和组合关系都以相似方式用类来表示，许多教材都不区分它们，将两者都称为组合。

10.8 设计类Course

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)。本书提供了多种类的例子来演示面向对象程序设计。10.8~10.10节将给出设计类的附加例子。

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10-9所示。

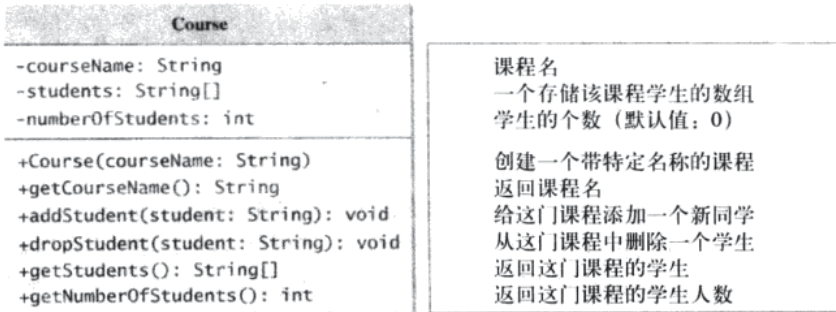


图10-9 Course类对课程建模

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。你可以使用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掉一个学生，而使用getStudents()方法可以返回选这门课程的所有学生。假设该类是可用的。程序清单10-5就给出一个测试类，这个测试类创建了两门课程，并向课程中加入学生。

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("Brian 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, Brian Smith, Anne Kennedy,
Number of students in course2: 2
```



Course类在程序清单10-6中执行。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 Exercise 10.9
29    }
30 }

```

在程序清单10-6中,数组的大小固定为100(第3行)。可以在练习题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。简单地说,这个Course对象包含了这个数组。

用户可以创建一个Course,然后通过公共方法addStudent、dropStudent、getNumberOfStudents和getStudents来处理它。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。该例使用一个数组存储学生。也可以使用不同的数据结构存储学生。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。

10.9 设计堆栈类

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10-10所示。

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10-11所示。

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)。

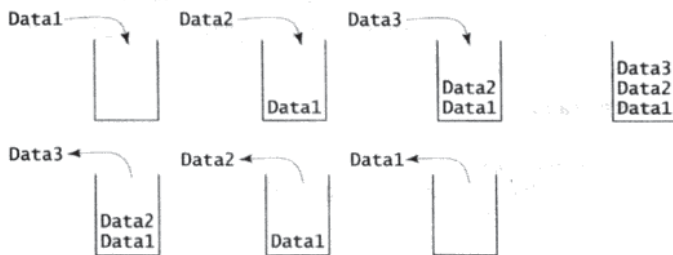


图10-10 栈是用后进先出的方式存放数据的

StackOfIntegers	
-elements: int[] -size: int	一个存储栈中整数的数组 栈中整数的个数
+StackOfIntegers() +StackOfIntegers(capacity: int) +empty(): boolean +peek(): int	构建一个默认容量为16的空栈 构建一个指定容量的空栈 如果栈为空则返回true 返回栈顶的整数而不从栈中删除该数
+push(value: int): void +pop(): int +getSize(): int	将一个整数存储到栈顶 删除栈顶整数并返回它 返回栈中元素的个数

图10-11 StackOfIntegers类封装栈的存储, 也提供处理栈的操作

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10-12所示。对空栈来说, size为0。

StackOfIntegers类是在程序清单10-8中实现的。方法empty()、peek()、pop()和getSize()都很容易实现。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, 可以给这个数组中添加新值 (第24行)。

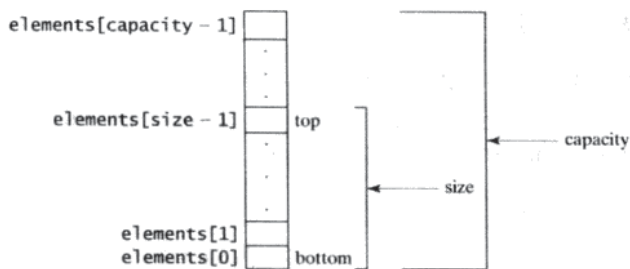


图10-12 StackOfIntegers类封装栈的存储, 也提供处理栈的操作

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 into 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        elements[size++] = value;
24    }
25
26    /** Return and remove the top element from the stack */
27    public int pop() {
28        return elements[--size];
29    }
30
31    /** Return the top element from the stack */
32    public int peek() {
33        return elements[size - 1];
34    }
35
36    /** Test whether the stack is empty */
37    public boolean empty() {
38        return size == 0;
39    }
40
41    /** Return the number of elements in the stack */
42    public int getSize() {
43        return size;
44    }
45 }
46

```

10.10 设计GuessDate类

程序清单3-3和程序清单7-6是两个猜测生日的程序。两个程序都使用相同的数据，并且都用面向对象范式开发。这两个程序的绝大多数代码就是定义五个数据集合。不能重用这两个程序的代码。为使代码可重用，需要设计一个如图10-13所定义的类型来封装数据。

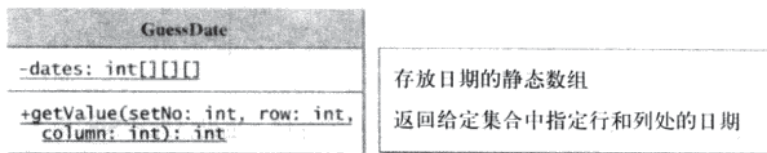


图10-13 GuessDate类定义猜测生日的数据

注意，getValue定义为一个静态方法，因为它不依赖于GuessDate类的某个特定对象。GuessDate类将dates封装为一个私有成员。这个类的使用者无须知道dates是如何实现的，甚至无须知道类中是否存在dates数据域。用户所需要知道的就是如何使用这些方法来访问数据。假设该类是可用的，如3.5节所示，这里有五个数据集合。调用getValue(setNo,row,column)会返回在给定集合中指定行和列的数据。例如：getValue(1,0,0)返回2。

假设GuessDate类是可用的。程序清单10-9是一个使用该类的测试程序。

程序清单10-9 UseGuessDateClass.java

```

1 import java.util.Scanner;
2
3 public class UseGuessDateClass {
4     public static void main(String[] args) {
5         int date = 0; // Date to be determined
6         int answer;
7
8         // Create a Scanner
9         Scanner input = new Scanner(System.in);
10
11         for (int i = 0; i < 5; i++) {
12             System.out.println("Is your birthday in Set" + (i + 1) + "?");
13             for (int j = 0; j < 4; j++) {
14                 for (int k = 0; k < 4; k++)
15                     System.out.print(GuessDate.getValue(i, j, k) + " ");
16                 System.out.println();
17             }
18
19             System.out.print("\nEnter 0 for No and 1 for Yes: ");
20             answer = input.nextInt();
21
22             if (answer == 1)
23                 date += GuessDate.getValue(i, 0, 0);
24         }
25
26         System.out.println("Your birthday is " + date);
27     }
28 }

```

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: 0



PDG


```

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 

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 

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: 1 

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 

Your birthday is 26

```

因为`getValue`是一个静态方法，所以为了调用它，无须创建一个对象。`GuessDate.getValue(i,j,k)` (第15行) 返回集合`i`中第`i`行第`k`列的数据。

`GuessDate`类可以在程序清单10-10中实现。

程序清单10-10 `GuessDate.java`

```

1 public class GuessDate {
2     private final static int[][][] dates = {
3         {{ 1, 3, 5, 7},
4          { 9, 11, 13, 15},
5          {17, 19, 21, 23},
6          {25, 27, 29, 31}},
7         {{ 2, 3, 6, 7},
8          {10, 11, 14, 15},
9          {18, 19, 22, 23},
10         {26, 27, 30, 31}},
11         {{ 4, 5, 6, 7},
12          {12, 13, 14, 15},
13          {20, 21, 22, 23},
14          {28, 29, 30, 31}},
15         {{ 8, 9, 10, 11},
16          {12, 13, 14, 15},
17          {24, 25, 26, 27},
18          {28, 29, 30, 31}},
19         {{16, 17, 18, 19},
20          {20, 21, 22, 23},
21          {24, 25, 26, 27},
22          {28, 29, 30, 31}}};
23
24     /** Prevent the user from creating objects from GuessDate */
25     private GuessDate() {
26     }
27

```



```

28  /** Return a date at the specified row and column in a given set */
29  public static int getValue(int setNo, int k, int j) {
30      return dates[setNo][k][j];
31  }
32 }

```

这个类使用一个三维数组存储数据（第2~22行）。可以使用不同的数据结构（即五个二维数组表示五个数字集合）。`getValue`方法的实现将被改变，但是使用`GuessDate`的程序无须改动，只要公共方法`getValue`的合约保持不变。这一点显示出数据封装的优点。

类定义了一个私有的无参构造方法（第25行），防止用户创建这个类的对象。由于这个类中的所有方法都是静态的，因此，无须创建这个类的对象。

10.11 类的设计原则

从前面两个例子以及前面几章中的其他许多例子中，已经学习了如何设计类。下面是一些设计原则。

10.11.1 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，支持一个连贯性的目标。例如：可以专门为学生使用一个类，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。

10.11.2 一致性

遵循标准Java程序设计风格和命名习惯。给类、数据域和方法选择有信息量的名字。流行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好的习惯。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，应该一律提供一个为构造默认实例的公共无参构造方法。如果一个类不支持无参构造方法，要用文档写出原因。如果没有显式定义构造方法，就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，就像`Math`类和`GuessDate`类的情况一样。

10.11.3 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。

如果想让数据域可读，只需提供`get`方法。如果想让数据域可更新，应该提供`set`方法。例如：`Course`类为`courseName`提供了`get`方法，但是没有提供`set`方法，这是因为一旦创建了课程类的对象，是不允许用户修改课程名字的。

10.11.4 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，易于解释和理解。

用户可以以多种不同组合和顺序，在许多不同环境中结合多个类。因此，应该设计一个类，这个类应该没有对用户使用目的及使用时间的限制，设计属性以容许用户按值的任何顺序和任何组合来设置，所设计的方法实现的功能应该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凭直觉来定义。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。

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;

    ...
}
```

10.11.5 完整性

类经常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。

10.11.6 实例和静态

依赖于类的具体实例的变量或方法应该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8-9中，Circle3中的变量numberOfObjects被SimpleCircle1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：Circle3中的getNumberOfObjects方法都未绑定到任何具体实例，因此，它被声明为静态方法。

经常用类名（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。数据域或方法要不是实例的就是静态的。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。

关键术语

class abstraction（类抽象）
class's contract（类的合约）

class encapsulation（类封装）
class's variable（类变量）

immutable class (不可变类)

stack (栈)

immutable object (不可变对象)

this keyword (this关键字)

本章小结

- 不可变对象一旦创建, 就不能改变。为防止用户修改对象, 可以定义不可变类。
- 实例变量和静态变量的作用域是整个类, 与变量在何处声明无关。实例变量和静态变量可以在类中的任意位置声明。为保持一致, 最好在类的起始位置声明它们。
- 关键字 **this** 可用于表明调用对象。关键字 **this** 也可以用在构造方法中, 调用同一个类的另一个构造方法。
- 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。

复习题

10.2节

10.1 如果一个类只有私有数据域而没有 **set** 方法, 那么该类是不可变的吗?

10.2 如果一个类中所有的数据域都是私有的且都是基本类型, 而且这个类也不包含 **set** 方法, 那么该类是否是不可变的?

10.3 下面的类是不可变的吗?

```
public class A {
    private int[] values;

    public int[] getValues() {
        return values;
    }
}
```

10.4 如果无须 **set** 方法重新定义程序清单 10-2 中的 **Loan** 类, 这个类是不可变的吗?

10.3节

10.5 下面程序的输出是什么?

```
public class Foo {
    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);
    }
}
```



10.4节

10.6 描述this关键字的作用。下面代码有什么错误?

```

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 }

```

编程练习题

*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。

画出该类的UML图。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。例如, 如果逝去的时间为555550秒, 那么转换为小时10、分钟19、秒数9。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2-6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isEven(int)、isOdd(int)和isPrime(int)方法都会返回true。

- 如果指定值分别为偶数、奇数或素数，那么`isEven(MyInteger)`、`isOdd(MyInteger)`和`isPrime(MyInteger)`方法都会返回`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。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。
- 两个数据域x和y各自的`get`方法。
- 一个名为`distance`的方法，返回**MyPoint**类型的两个点之间的距离。
- 一个名为`distance`的方法，返回指定x和y坐标的两个点之间的距离。

画出该类的UML图。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。

*10.5 (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) 使用练习题8.7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



PDF

```

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 1 Enter
The balance is 97.0

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 3 Enter
Enter an amount to deposit: 10 Enter

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 1 Enter
The balance is 107.0

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 4 Enter

Enter an id:

```

****10.8 (财务问题: 税款类Tax)** 练习题7.12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、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(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。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	已婚共缴纳税人或合法寡妇	已婚单缴纳税人	户主纳税人
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, 删掉一个学生, 然后显示这门课程的学生。

***10.10 (游戏: GuessDate类)** 修改程序清单10-10中的GuessDate类。不使用三维数组表示数据, 而是用五个二维数组来表示五个集合的数字。所以, 需要声明:

```
private static int[][] set1 = {{1, 3, 5, 7}, ... };
private static int[][] set2 = {{2, 3, 6, 7}, ... };
private static int[][] set3 = {{4, 5, 6, 7}, ... };
private static int[][] set4 = {{8, 9, 10, 11}, ... };
private static int[][] set5 = {{16, 17, 18, 19}, ... };
```

***10.11 (几何方面: 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10-14a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10-14b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10-14c所示。

画出该类的UML图。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))。

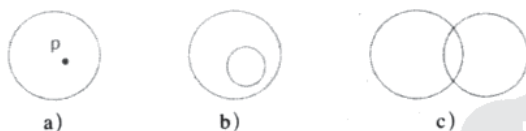


图10-14 a) 点在圆内; b) 一个圆在另一个圆内; c) 一个圆和另一个圆重叠

***10.12 (几何问题: 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10-15a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10-15b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10-15c所示。

画出该类的UML图。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。

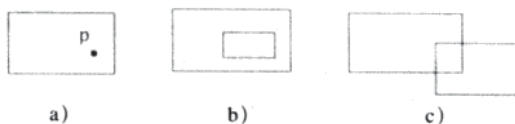


图10-15 a) 点在矩形内；b) 一个矩形在另一个矩形内；c) 一个矩形和另一个矩形重叠

***10.13 (几何问题: `Triangle2D`类) 定义`Triangle2D`类, 包含:

- 三个名为`p1`、`p2`和`p3`的`MyPoint`型数据域，这三个数据域都带有`get`和`set`方法。`MyPoint`在练习题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10-16a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10-16b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10-16c所示。

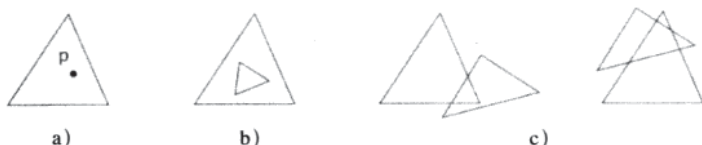


图10-16 a) 点在三角形内；b) 一个三角形在另一个三角形内；c) 一个三角形和另一个三角形重叠

画出该类的UML图。实现这个类。编写测试程序，创建一个`Triangle2D`对象`t1`(`new Triangle2D(new MyPoint(2.5,2), new MyPoint(4.2,3), new MyPoint(5,3.5))`)，显示它的面积和周长，还要显示`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，请参见练习题5.19。使用Java API中的`java.awt.geo.Line2D`类来实现`contains`和`overlaps`方法。`Line2D`类包括检查两条线段是否相交以及检查一条线是否包含一个点等方法。请参见Java API获取关于`Line2D`更多的信息。为了检测一个点是否在三角形中，画三条虚线，如图10-17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。

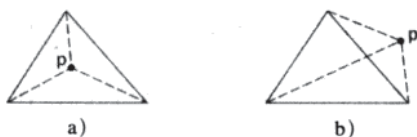


图10-17 a) 点在三角形内；b) 一个点在三角形外

*10.14 (MyDate类) 设计一个名为MyDate类。该类包含：

- 表示日期的数据域year、month和day。
- 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各自的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。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(34355555133101L))，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561555550000毫秒，那么年就是1987，月就是10，而天是17。对于无参构造方法，当前时间可以使用System.currentTimeMillis()获取，如程序清单2-6所示。



继承和多态

学习目标

- 通过继承由父类创建子类 (11.2节)。
- 使用关键字`super`调用父类的构造方法 and 方法 (11.3节)。
- 在子类中覆盖方法 (11.4节)。
- 区分覆盖和重载的不同 (11.5节)。
- 探究`Object`类中的`toString()`方法 (11.6节)。
- 理解多态性和动态绑定 (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)。
- 使用`ArrayList`实现`Stack`类 (11.12节)。
- 使用可见性修饰符`protected`使父类中的数据和 method 可以被子类访问 (11.13节)。
- 使用修饰符`final`防止类的扩展以及方法的覆盖 (11.14节)。

11.1 引言

在面向对象程序设计中，可以从已有的类派生出新类。这称做继承 (inheritance)。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的方式是什么？答案就是使用继承。

11.2 父类和子类

使用类来对同一类型的对象建模。不同的类也可能会有有一些共同的特征和动作，这些共同的特征和行动都统一放在一个类中，它是可以被其他类所共享的。继承可以让你定义一个通用类，随后将它扩展为更多特定的类。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，通过扩展`GeometricObject`类来定义`Circle`类是很有意义的。同理，`Rectangle`也可以声明为`GeometricObject`的子类。图11-1显示这些类之间的关系。指向父类的箭头用来表示相关的两个类之间的继承关系。

在Java的术语中，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。

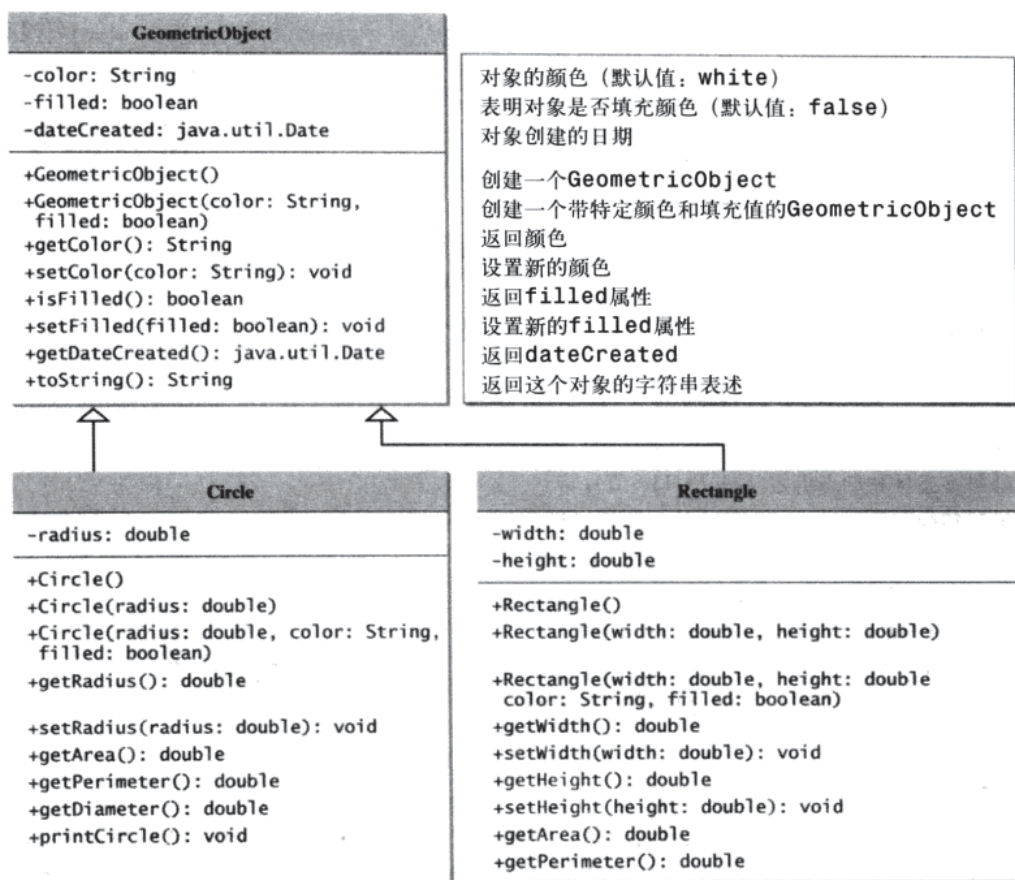


图11-1 GeometricObject类是Circle类和Rectangle类的父类

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。除此之外，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类分别显示在程序清单11-1、程序清单11-2和程序清单11-3中。

注意 为了避免与第12章介绍的改进版的GeometricObject类、Circle类和Rectangle类发生命名冲突，就将本章的类命名为GeometricObject1、Circle4和Rectangle1。为方便起见，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。然而，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 GeometricObject1.java

```

1 public class GeometricObject1 {
2     private String color = "white";
3     private boolean filled;
4     private java.util.Date dateCreated;
5

```



```

6  /** Construct a default geometric object */
7  public GeometricObject1() {
8      dateCreated = new java.util.Date();
9  }
10
11  /** Construct a geometric object with the specified color
12   * and filled value */
13  public GeometricObject1(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 get 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 + "\n color: " + color +
48          " and filled: " + filled;
49  }
50 }

```

程序清单11-2 Circle4.java

```

1  public class Circle4 extends GeometricObject1 {
2      private double radius;
3
4      public Circle4() {
5      }
6
7      public Circle4(double radius) {
8          this.radius = radius;
9      }
10
11     public Circle4(double radius, String color, boolean filled) {
12         this.radius = radius;
13         setColor(color);
14         setFilled(filled);
15     }
16
17     /** Return radius */

```




```

18 public double getRadius() {
19     return radius;
20 }
21
22 /** Set a new radius */
23 public void setRadius(double radius) {
24     this.radius = radius;
25 }
26
27 /** Return area */
28 public double getArea() {
29     return radius * radius * Math.PI;
30 }
31
32 /** Return diameter */
33 public double getDiameter() {
34     return 2 * radius;
35 }
36
37 /** Return perimeter */
38 public double getPerimeter() {
39     return 2 * radius * Math.PI;
40 }
41
42 /* Print the circle info */
43 public void printCircle() {
44     System.out.println("The circle is created " + getDateCreated() +
45         " and the radius is " + radius);
46 }
47 }

```

Circle类使用下面的语法扩展GeometricObject类 (程序清单11-2):

子类
父类

public class Circle extends GeometricObject

关键字**extends** (第1行) 告诉编译器, Circle类扩展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getColor和setFilled方法设置color和filled属性来执行的 (第11~15行)。这两个公共方法是在基类GeometricObject中定义的, 在Circle中继承。因此, 可以在派生类中使用它们。

可以尝试在构造方法中使用数据域color和filled, 如下所示:

```

public Circle4(double radius, String color, boolean filled) {
    this.radius = radius;
    this.color = color; // Illegal
    this.filled = filled; // Illegal
}

```

这是错的, 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) 使用下面的语法扩展GeometricObject类 (程序清单11-2):

子类
父类

public class Rectangle extends GeometricObject

关键字**extends** (第1行) 告诉编译器**Rectangle**类扩展**GeometricObject**类, 这样, 它就继承了**getColor**、**setColor**、**isFilled**、**setFilled**和**toString**方法。

程序清单11-3 **Rectangle1.java**

```

1 public class Rectangle1 extends GeometricObject1 {
2     private double width;
3     private double height;
4
5     public Rectangle1() {
6     }
7
8     public Rectangle1(double width, double height) {
9         this.width = width;
10        this.height = height;
11    }
12
13    public Rectangle1(double width, double height, String color,
14        boolean filled) {
15        this.width = width;
16        this.height = height;
17        setColor(color);
18        setFilled(filled);
19    }
20
21    /** Return width */
22    public double getWidth() {
23        return width;
24    }
25
26    /** Set a new width */
27    public void setWidth(double width) {
28        this.width = width;
29    }
30
31    /** Return height */
32    public double getHeight() {
33        return height;
34    }
35
36    /** Set a new height */
37    public void setHeight(double height) {
38        this.height = height;
39    }
40
41    /** Return area */
42    public double getArea() {
43        return width * height;
44    }
45
46    /** Return perimeter */
47    public double getPerimeter() {
48        return 2 * (width + height);
49    }
50 }

```

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, 并且从**Circle**对象 (第4行) 和**Rectangle**对象 (第10行) 调用。

程序清单11-4 **TestCircleRectangle.java**

```

1 public class TestCircleRectangle {
2     public static void main(String[] args) {
3         Circle4 circle = new Circle4(1);
4         System.out.println("A circle " + circle.toString());
5         System.out.println("The radius is " + circle.getRadius());

```

```

6      System.out.println("The area is " + circle.getArea());
7      System.out.println("The diameter is " + circle.getDiameter());
8
9      Rectangle1 rectangle = new Rectangle1(2, 4);
10     System.out.println("\nA rectangle " + rectangle.toString());
11     System.out.println("The area is " + rectangle.getArea());
12     System.out.println("The perimeter is " +
13         rectangle.getPerimeter());
14 }
15 }

```

A circle created on Thu Sep 24 20:31:02 EDT 2009

color: white and filled: false

The radius is 1.0

The area is 3.141592653589793

The diameter is 2.0

A rectangle created on Thu Sep 24 20:31:02 EDT 2009

color: white and filled: false

The area is 8.0

The perimeter is 12.0



下面是在考虑继承时很值得注意的几点:

- 和传统的理解相反, 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它们。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。
- 不是所有的是关系 (is-a) 都该用继承来建模。例如: 一个正方形是一个矩形, 但是不应该定义一个Square类来扩展Rectangle类, 因为没有任何东西可以从矩形扩展 (或者补充) 到正方形。所以, 应该定义一个扩展自GeometricObject类的Square类。如果要用类B去扩展类A, 那么A应该要比B包含更多的信息。
- 继承是用来为是关系 (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14.4节中介绍。

11.3 使用super关键字

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同它们相关的问题。

10.4节中介绍了关键字this的作用, 它是一个调用对象的引用。关键字super是指这个super出现的类的父类。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

调用父类构造方法的语法是:

super(), or **super(parameters)**;

语句**super()**调用它的父类的无参构造方法,而语句**super(参数)**调用与参数匹配的父类的构造方法。语句**super()**和**super(参数)**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11~15行的构造方法可以用下面的代码替换:

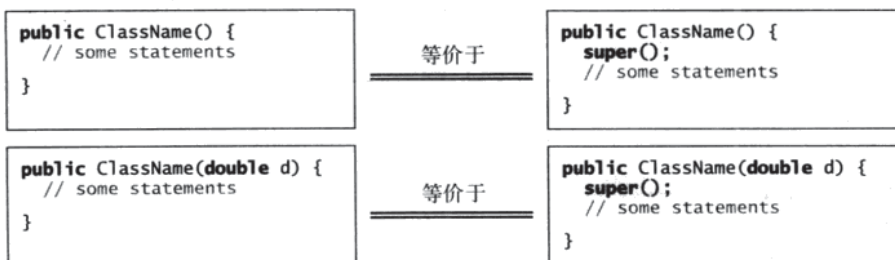
```
public Circle4(double radius, String color, boolean filled) {
    super(color, filled);
    this.radius = radius;
}
```

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。

注意 构造方法可用来构造一个类的实例。不像属性和方法,父类的构造方法是不被子类继承的。它们只能从子类的构造方法中用关键字**super**调用。

11.3.2 构造方法链

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。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)。

考虑下面的代码:

```
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() {
```

新华书店
PDG

```

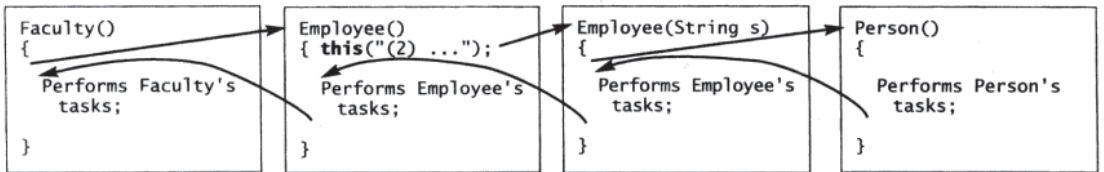
24      System.out.println("(1) Performs Person's tasks");
25  }
26  }

```

(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12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 如果一个类要设计为扩展的，最好提供一个无参构造方法以避免程序设计错误。考虑下面的代码：

```

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，程序不能被编译。

设计指南 最好能为每个类提供一个无参构造方法（如果需要的话），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);
}

```

然而，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，可以被`Circle`类继承。然而，在有些情况下，如11.4节所示，关键字`super`是必不可少的。

11.4 覆盖方法

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，这称做方法覆盖（method overriding）。

GeometricObject类中的toString方法返回表示几何对象的字符串。这个方法可以被覆盖，返回表示圆的字符串。为了覆盖它，在程序清单11-2中加入下面的新方法：

```
1 public class Circle4 extends GeometricObject1 {
2     // Other methods are omitted
3
4     /** Override the toString method defined in GeometricObject */
5     public String toString() {
6         return super.toString() + "\nradius is " + radius;
7     }
8 }
```

toString()方法在GeometricObject类中定义，在Circle类中修改。在这两个类中定义的该方法都可以在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6行）。

Circle的子类能用语法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。这样，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，那么定义在父类中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。

11.5 覆盖和重载

在5.8节中已经学过关于重载方法的内容。重载方法意味着可以定义多个同名的方法，但这些方法具有不同的签名。覆盖方法意味着为子类中的方法提供一个全新的实现。该方法已经在父类中定义。

为了覆盖一个方法，这个方法必须使用相同的签名以及相同的返回值类型在子类中进行定义。

用一个例子来显示覆盖和重载的不同。在图a中，类A中的方法p(double i)覆盖了在类B中定义的同名方法。但是，在图b中，类B中有两个重载的方法p(double i)和p(int i)。类B的p(double i)方法被继承。

```
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)

运行图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。运行图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。

11.6 对象类Object和它的toString()方法

Java中的每个类都源于`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：

```
public class ClassName {
    ...
}
```

等价于

```
public class ClassName extends Object {
    ...
}
```

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、`@`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，考虑下面这些在程序清单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，应该覆盖这个`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`类中覆盖，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，应该使用`System.out.println(loan)`来替换`System.out.println(loan.toString())`。

11.7 多态

面向对象程序设计的三个特点是封装、继承和多态。前面已经学习了前两个特点，本节将介绍多态性。

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 Circle4(1, "red", false));
6         displayObject(new Rectangle1(1, 1, "black", true));
7     }
8
9     /** Display geometric object properties */
10    public static void displayObject(GeometricObject1 object) {
11        System.out.println("Created on " + object.getDateCreated() +
12            ". Color is " + object.getColor());
13    }
14 }

```

Created on Mon Mar 09 19:25:20 EDT 2009. Color is white
 Created on Mon Mar 09 19:25:20 EDT 2009. Color is black



方法displayObject (第10行) 采用GeometricObject类型的参数。可以通过传递任何一个GeometricObject的实例 (例如: 在第5~6行的new Circle4(1, "red", false)和new Rectangle1(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”)。用简单的术语来描述, 多态就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

一个方法可以在父类中定义而在它的子类中覆盖。例如: toString()方法是在Object类中定义的, 而该方法在GeometricObject1类中覆盖。考虑下面的代码:

```

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调用的到底是哪个toString()方法是由o的实际类型所决定的。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 $C_1, C_2, \dots, C_{n-1}, C_n$ 的实例, 其中 C_1 是 C_2 的子类, C_2 是 C_3 的子类, \dots, C_{n-1} 是 C_n 的子类, 如图11-2所示。也就是说, C_n 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Java虚拟机会依次在类 $C_1, C_2, \dots, C_{n-1}, C_n$ 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第一次找到的实现。

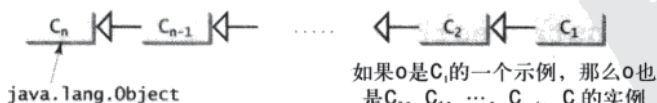


图11-2 将调用的方法是在运行时动态决定的

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     public String toString() {
19         return "Student";
20     }
21 }
22
23 class Person extends Object {
24     public String toString() {
25         return "Person";
26     }
27 }

```

```

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）会导致定义在Student类中的toString方法被调用。

调用m(new Person())（第5行）会导致定义在Person类中的toString方法被调用。调用m(new Object())（第6行）会导致定义在Object类中的toString方法被调用。

匹配方法的签名和绑定方法的实现是两个独立的事情。引用变量的声明类型决定了编译时匹配哪个方法。编译器会在编译时，根据参数类型、参数个数和参数顺序找到匹配的方法。一个方法可能在几个子类中都被实现。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。

11.9 对象转换和instanceof运算符

你已经使用过转换运算符把一种基本类型变量转换为另一种基本类型。转换也可以用来把一种类类型的对象转换为继承体系结构中的另一种类类型的对象。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的实例自动地就是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懂得这一点。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，用圆括弧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（称之为向上转换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（称之为向下转换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实例。如果父类对象不是子类的实例，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 Circle4(1);
6         Object object2 = new Rectangle1(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 Circle4) {
16         System.out.println("The circle area is " +
17             ((Circle4)object).getArea());
18         System.out.println("The circle diameter is " +
19             ((Circle4)object).getDiameter());
20     }
21     else if (object instanceof Rectangle1) {
22         System.out.println("The rectangle area is " +
23             ((Rectangle1)object).getArea());
24     }
25 }
26 }

```

```

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14~25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。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23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();
```

11.10 Object的equals方法

在`Object`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`equals`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覆盖这个方法, 以测试两个不同的对象是否具有相同的内容。



在9.2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类中被覆盖，使之能够检验两个字符串的内容是否相等。可以覆盖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 false;
}
```

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，覆盖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)) 覆盖equals方法是一个常见错误。应该使用equals(Object obj)。参见复习题11.15。

11.11 数组线性表ArrayList类

现在，我们准备介绍一个很有用的存储对象的类。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ArrayList中的一些方法。

java.util.ArrayList	
<pre>+ArrayList() +add(o: Object): void +add(index: int, o: Object): void +clear(): void +contains(o: Object): boolean +get(index: int): Object +indexOf(o: Object): int +isEmpty(): boolean +lastIndexOf(o: Object): int +remove(o: Object): boolean +size(): int +remove(index: int): boolean +set(index: int, o: Object): Object</pre>	<p>创建一个空的线性表</p> <p>在这个线性表的末尾追加一个新元素0</p> <p>在这个线性表的特定下标处增加一个新元素0</p> <p>从这个线性表中删除所有的元素</p> <p>如果这个线性表包含元素0则返回true</p> <p>返回这个线性表在特定下标处的元素</p> <p>返回这个线性表中第一个匹配元素的下标</p> <p>如果这个线性表不包含元素则返回true</p> <p>返回这个线性表中最后一个匹配元素的下标</p> <p>从这个线性表中删除元素0</p> <p>返回这个线性表中元素的个数</p> <p>删除指定下标处的元素</p> <p>设置在特定下标处的元素</p>

图11-3 ArrayList中存储不限定个数的对象

程序清单11-8是使用ArrayList存储对象的一个例子。

程序清单11-8 TestArrayList.java

```
1 public class TestArrayList {
2     public static void main(String[] args) {
3         // Create a list to store cities
4         java.util.ArrayList cityList = new java.util.ArrayList();
5
6         // Add some cities in the list
7         cityList.add("London");
8
9         // cityList now contains [London]
10        cityList.add("Denver");
11        // cityList now contains [London, Denver]
```

```

11  cityList.add("Paris");
12  // cityList now contains [London, Denver, Paris]
13  cityList.add("Miami");
14  // cityList now contains [London, Denver, Paris, Miami]
15  cityList.add("Seoul");
16  // contains [London, Denver, Paris, Miami, Seoul]
17  cityList.add("Tokyo");
18  // contains [London, Denver, Paris, Miami, Seoul, Tokyo]
19
20  System.out.println("List size? " + cityList.size());
21  System.out.println("Is Miami in the list? " +
22      cityList.contains("Miami"));
23  System.out.println("The location of Denver in the list? " +
24      + cityList.indexOf("Denver"));
25  System.out.println("Is the list empty? " +
26      cityList.isEmpty()); // Print false
27
28  // Insert a new city at index 2
29  cityList.add(2, "Xian");
30  // contains [London, Denver, Xian, Paris, Miami, Seoul, Tokyo]
31
32  // Remove a city from the list
33  cityList.remove("Miami");
34  // contains [London, Denver, Xian, Paris, Seoul, Tokyo]
35
36  // Remove a city at index 1
37  cityList.remove(1);
38  // contains [London, Xian, Paris, Seoul, Tokyo]
39
40  // Display the contents in the list
41  System.out.println(cityList.toString());
42
43  // Display the contents in the list in reverse order
44  for (int i = cityList.size() - 1; i >= 0; i--)
45      System.out.print(cityList.get(i) + " ");
46  System.out.println();
47
48  // Create a list to store two circles
49  java.util.ArrayList list = new java.util.ArrayList();
50
51  // Add two circles
52  list.add(new Circle4(2));
53  list.add(new Circle4(3));
54
55  // Display the area of the first circle in the list
56  System.out.println("The area of the circle? " +
57      ((Circle4)list.get(0)).getArea());
58 }
59 }

```

```

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

```

程序使用无参构造方法创建一个ArrayList (第4行), add方法将Object的任一实例加入线性表中。由于String是Object的一个子类, 所以字符串可以加入到线性表中。add方法 (第7~17行) 将一个对象加入到线性表的末尾。所以, 在执行完cityList.add("London") (第7行) 之后, 这个线性表包含

[London]

执行完cityList.add("New Denver") (第9行) 后, 这个线性表包含

```
[London, Denver]
```

在加入Paris、Miami、Seoul和Tokyo之后（第11~17行），这个线性表包含

```
[London,Denver,Paris,Miami,Seoul,Tokyo]
```

调用size()（第20行）返回这个线性表的大小，线性表的当前大小为6。调用contains("Miami")（第22行）检测这个对象是否在这个线性表中。在这种情况下，它返回true，因为Miami在这个线性表中。调用indexOf("Denver")（第24行）返回该对象在线性表中的索引值，这里它的值为1。如果对象不在这个线性表中，它返回-1。isEmpty()方法（第26行）检测这个线性表是否为空。因为当前列表不为空，所以它返回false。

语句cityList.add(2,"Xian")（第29行）在这个线性表的指定下标位置插入一个对象。该语句执行完之后，线性表变成

```
[London,Denver,Xian,Paris,Miami,Seoul,Tokyo]
```

语句cityList.remove("Miami")（第33行）从线性表中删除该对象。该语句执行后，线性表就变成

```
[London,Denver,Xian,Paris,Seoul,Tokyo]
```

语句cityList.remove(1)语句（第37行）从线性表中删除指定下标位置的元素，该语句执行后，线性表变成

```
[London,Xian,Paris,Seoul,Tokyo]
```

第41行的语句相当于

```
System.out.println(cityList);
```

方法toString()返回表示线性表的字符串，其形式为[e0.toString(),e1.toString(),...,ek.toString()]，这里的e0, e1, ..., ek都是线性表中的元素。...

方法get(index)（第45行）返回指定下标位置处的对象。

注意 从命令行编译这个程序时，会产生下面的警告：

```
Note: TestArrayList.java uses unchecked or unsafe operations.
```

```
Note: Recompile with -Xlint:unchecked for details.
```

这个警告可以使用第21章中介绍的泛型类型来消除。目前就先忽略它。尽管会有这个警告，但是程序还是会编译，产生一个.class文件。

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。

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。如果元素不在线性表中，就不能使用get和set方法。向线性表中添加、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。

表11-1 数组和ArrayList之间的异同

操 作	数 组	ArrayList
创建数组/ArrayList	Object[] a = new Object[10]	ArrayList list = new ArrayList()
引用元素	a [index]	list.get(index)
更新元素	a [index] = "London";	list.set(index, "London");
返回大小	a length	list.size()
添加一个新元素		list.add("London")
插入一个新元素		list.add(index, "London")
删除一个元素		list.remove(index)
删除一个元素		list.remove(Object)
删除所有元素		list.clear()

注意 `java.util.Vector`也是一个存储对象的类，它与`ArrayList`类非常相似。`Vector`也有`ArrayList`中的所有方法。`Vector`类是在JDK 1.1中介绍的。在JDK 1.2中介绍`ArrayList`类是为了代替`Vector`类。

11.12 自定义栈类

10.9节给出了一个栈类存放`int`值。本节介绍存储对象的栈类。可以使用`ArrayList`实现`Stack`，如程序清单11-9所示。该类的UML图如图11-4所示。

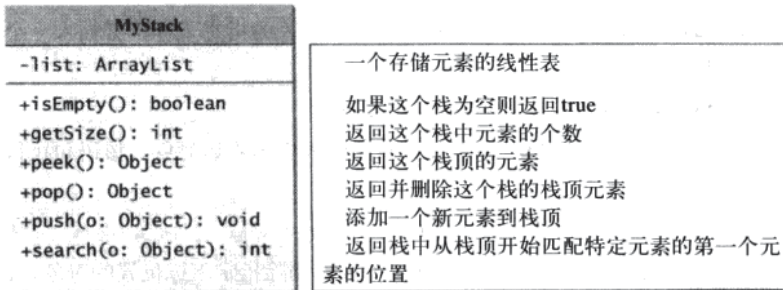


图11-4 MyStack类封装栈的存储并提供处理这个栈的操作

程序清单11-9 MyStack.java

```

1 public class MyStack {
2     private java.util.ArrayList list = new java.util.ArrayList();
3
4     public boolean isEmpty() {
5         return list.isEmpty();
6     }
7
8     public int getSize() {
9         return list.size();
10    }
11
12    public Object peek() {
13        return list.get(getSize() - 1);
14    }
15
16    public Object pop() {
17        Object o = list.get(getSize() - 1);
18        list.remove(getSize() - 1);
19        return o;
20    }
21
22    public void push(Object o) {
23        list.add(o);
24    }
25
26    public int search(Object o) {
27        return list.lastIndexOf(o);
28    }
29
30    /** Override the toString in the Object class */
31    public String toString() {
32        return "stack: " + list.toString();
33    }
34 }

```

创建数组线性表来存储栈中的元素（第2行）。`isEmpty()`方法（第4~6行）返回`list.isEmpty()`。

`getSize()`方法（第8~10行）返回`list.size()`。`peek()`方法（第12~14行）可以获取栈顶元素而不删除它，线性表末尾的元素正是栈顶的元素。`pop()`方法（第16~20行）删除栈顶元素并返回该元素。`push(Object element)`方法（第22~24行）将指定元素添加到这个栈中。`search(Object element)`方法检测指定元素是否在栈中，并通过调用`list.lastIndexOf(o)`返回从栈顶开始第一个匹配元素的索引。通过调用`list.toString()`覆盖`Object`类中定义的`toString()`方法（第31~33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11-9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，组合是对“是一部分”（has-a）关系建模。可以将`MyStack`实现为`ArrayList`的一个子类（参见练习题11.4）。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。

11.13 protected数据和方法

目前，已经使用过关键字`private`和`public`来表示是否可以从类外访问这些数据域。私有成员只能在类内访问，而公共成员可以被其他类访问。

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：

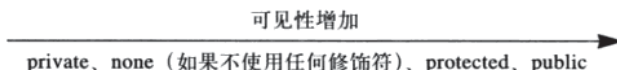


表11-2总结了类中成员的可访问性。图11-5描述了C1类中的`public`、`protected`、默认的和`private`数据或方法是如何被C2、C3、C4和C5类访问的，其中，C2类与C1类在同一个包中、C3类是C1类在同一个包中的子类、C4类是C1类在不同包中的子类、C5类与C1类在不同包中。

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：一种是为创建该类的实例；另一种是通过扩展该类创建它的子类。如果不想从类外使用类的成员，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，同样也可以用于类。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。

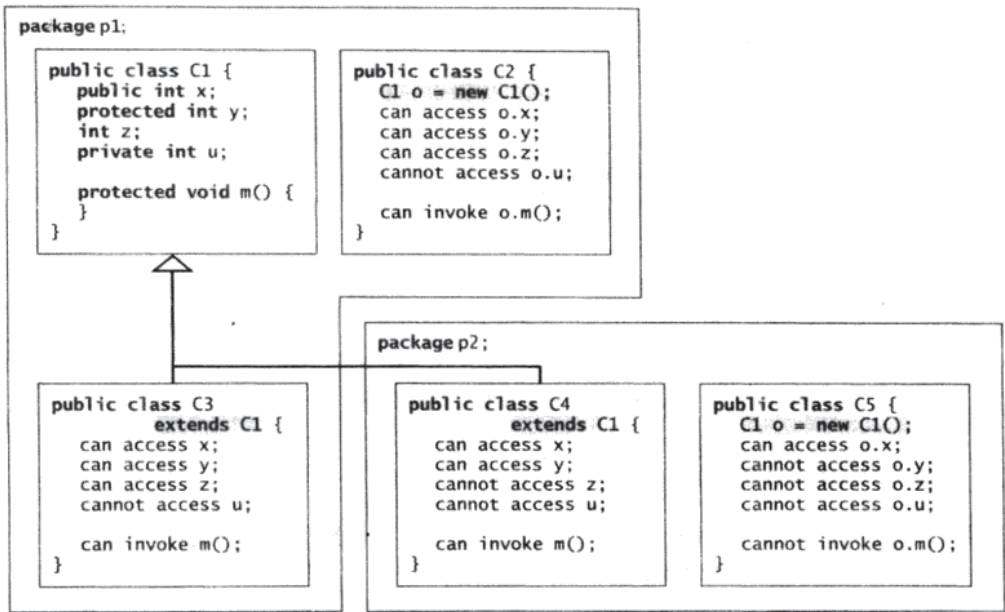


图11-5 使用可见性修饰符控制如何访问数据和方法

注意 子类可以覆盖它的父类的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。

11.14 防止扩展和覆盖

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终极的，是不能作为父类的。Math类就是一个终极类。String、StringBuilder和StringBuffer类也可以是终极类。例如，下面的类就是终极的，是不能扩展的：

```
public final class C {
    // Data fields, constructors, and methods omitted
}
```

也可以定义一个方式为终极的，一个终极方法不能被它的子类覆盖。

例如，下面的方法是终极的，是不能覆盖的：

```
public class Test {
    // Data fields, constructors, and methods omitted

    public final void m() {
        // Do something
    }
}
```

注意 修饰符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。方法内的终极局部变量就是常量。

关键术语

actual type（实际类型）

array list（数组线性表）

casting object（转换对象）

composition（组合）

constructor chaining (构造方法链)	override (覆盖)
declared type (声明类型)	polymorphism (多态)
dynamic binding (动态绑定)	protected (保护修饰符)
final (终极修饰符)	subclass (子类)
has-a relationship (是一部分的关系)	subtype (子类型)
inheritance (继承)	superclass (父类)
instanceof (运算符instanceof)	supertype (父类型)
is-a relationship (是一种的关系)	vector (矢量)

本章小结

- 可以从现有的类派生出新类。这称为类的继承。新类称为次类、子类或派生类。现有的类称为超类、父类或基类。
- 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。
- 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。
- 为了覆盖一个方法，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。
- 实例方法只有在是可访问的时候才能覆盖。这样，私有方法是不能覆盖的，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。
- 静态方法与实例方法一样可以继承。但是，静态方法不能覆盖，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。
- Java中的每个类都源于**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**。
- 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。当在方法中使用一个对象（例如：**Circle**对象或**String**对象）时，动态地决定调用该对象方法（例如：**toString**）的某个特定的实现。
- 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。
- 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。
- 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。当访问数据域或静态方法时，引用变量的声明类型在编译时决定使用哪个方法。
- 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。
- 可以使用**protected**修饰符来防止方法和数据被不同包的非子类访问。
- 可以用**final**修饰符来表明一个类是终极的，是不能成为父类的；并且用它来表明一个方法是终极的，是不能覆盖的。

复习题

11.2~11.5节

11.1 运行图a中的类C时，输出结果是什么？图b中的程序在编译时会出现什么问题？

```

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)

11.2 下面的说法是对还是错?

- (1) 子类是父类的一个子集。
- (2) 当从子类调用一个构造方法时，它的父类的无参构造方法总是被调用。
- (3) 可以覆盖定义在父类中的私有方法。
- (4) 可以覆盖定义在父类中的静态方法。

11.3 指出下面类中的问题:

```

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 getArea() */
26     public double getArea() {
27         return getArea() * length;
28     }
29 }

```

11.4 如何从子类显式调用父类的构造方法?

11.5 如何从子类调用被覆盖的父类方法?

11.6 解释方法覆盖和方法重载之间的区别。

11.7 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头，且返回值类型也相同，那么这是方法的覆盖还是重载呢?

11.8 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头，但返回值类型不相同，这会是一个问题吗?

11.9 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，那么这是方法的覆盖还是重载呢?

11.6~11.9节

11.10 是否每个类都有toString方法和equals方法？它们是从哪儿来的？如何使用它们？覆盖这些方法是否合适？

11.11 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，会调用Object的无参构造方法吗？

11.12 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：

(1) 下面的布尔表达式的值是true还是false？

```

Circle circle = new Circle(1);
GeometricObject object1 = new GeometricObject();
(circle instanceof GeometricObject)
(object1 instanceof GeometricObject)
(circle instanceof Circle)
(object1 instanceof Circle)

```

(2) 下面的语句能够编译吗？

```

Circle circle = new Circle(5);
GeometricObject object = circle;

```

(3) 下面的语句能够编译吗？

```

GeometricObject object = new GeometricObject();
Circle circle = (Circle)object;

```

11.13 假设Fruit、Apple、Orange、GoldenDelicious和Macintosh声明为如图11-6所示。

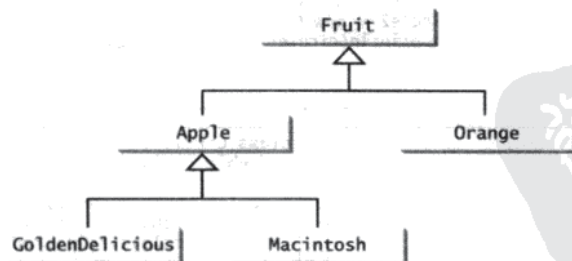


图11-6 GoldenDelicious和Macintosh是Apple的子类；Apple和Orange是Fruit的子类

假设给出下面的声明：

```

Fruit fruit = new GoldenDelicious();
Orange orange = new Orange();

```

回答下面的问题：

(1) fruit instanceof Fruit的值为true吗？

- (2) `fruit instanceof Orange`的值为true吗?
- (3) `fruit instanceof Apple`的值为true吗?
- (4) `fruit instanceof GoldenDelicious`的值为true吗?
- (5) `fruit instanceof Macintosh`的值为true吗?
- (6) `orange instanceof Orange`的值为true吗?
- (7) `orange instanceof Fruit`的值为true吗?
- (8) `orange instanceof Apple`的值为true吗?
- (9) 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?
- (10) 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?
- (11) 语句`Orange p=new Apple()`是否合法?
- (12) 语句`Macintosh p=new Apple()`是否合法?
- (13) 语句`Apple p=new Macintosh()`是否合法?

11.14 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节

11.15 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)`,如下面图a中的代码所示;应该使用如图b中所示的`equals(Object circle)`替换它。分别给出运行图a和图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)

11.11~11.12节

- 11.16 如何创建一个`ArrayList`? 如何向线性表中追加一个对象? 如何在线性表的开始位置插入一个对象? 如何找出线性表中所包含对象的个数? 如何从线性表中删除给定对象? 如何从线性表中删除最后的对象? 如何检测一个给定的对象是否在线性表中? 如何从线性表中获取指定下标位置的对象?
- 11.17 下面代码中有三处错误,请找出它们:

```

ArrayList 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));

```

11.13~11.14节

11.18 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？

11.19 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？

11.20 在下面的代码中，类A和类B在同一个包中。如果问号被空白代替，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)

11.21 在下面的代码中，类A和类B在不同的包中。如果问号被空白代替，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)

11.22 如何防止一个类被扩展？如何防止一个方法被覆盖？

综合题

11.23 定义下列术语：继承、父类、子类、关键字 `super` 和 `this`、转换对象、修饰符 `protected` 和 `final`。

11.24 确定下面语句是对还是错：

- (1) 被保护的数据或方法可以被同一包中的任何类访问。
- (2) 被保护的数据或方法可以被不同包中的任何类访问。
- (3) 被保护的数据或方法可以被任意包中它的子类访问。
- (4) 终极类可以有实例。
- (5) 终极类可以被扩展。
- (6) 终极方法可以被覆盖。
- (7) 总可以成功地将子类的实例转换为父类。
- (8) 总可以成功地将父类的实例转换为子类。

11.25 描述方法匹配和方法绑定之间的区别。

11.26 什么是多态？什么是动态绑定？



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:

- (1) 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。
- (2) 一个无参构造方法创建默认的三角形。
- (3) 一个能创建带指定side1、side2和side3的三角形的构造方法。
- (4) 所有三个数据域的访问器方法。
- (5) 一个名为getArea()的方法返回这个三角形的面积。
- (6) 一个名为getPerimeter()的方法返回这个三角形的周长。
- (7) 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练习题2.21。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。实现这些类。编写一个测试程序, 创建边长为1、1.5和1, 颜色为yellow, filled为true的Triangle对象, 然后显示它的面积、周长、颜色以及是否被填充。

11.5~11.11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。一个雇员有办公室、工资和受聘日期。定义一个名为MyDate的类, 包含数据域: year(年)、month(月)和day(日)。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, 显示相应的类名和人名。

画出这些类的UML图。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) 在练习题8.7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。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 (利用继承实现MyStack) 在程序清单11-9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。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。

11.5 (课程类Course) 改写程序清单10-6中的Course类。使用ArrayList代替数组来存储学生。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)。

11.6 (使用ArrayList) 编写程序, 创建一个ArrayList, 然后向这个线性表中添加一个Loan对象、一个Date对象、一个字符串、一个JFrame对象和一个Circle对象, 然后使用循环调用对象的toString()方法, 来显示线性表中所有的元素。

***11.7 (实现ArrayList) 在Java API中实现ArrayList。实现ArrayList以及图11-3中定义的方法。

提示 使用一个数组存储ArrayList中的元素。如果ArrayList的大小超过当前数组的容量，那就创建一个大小是当前数组两倍的新数组，然后将当前数组的内容复制给新数组。

**11.8 (新的Account类) 练习题8.7中给出一个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，该方法创建一个带指定名字、id和收支额的账户。
- 添加一个名为transactions的新数据域，它的类型是ArrayList，可以为账户存储交易。每笔交易都是一个Transaction类的实例。Transaction类的定义如图11-7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练习题8.7中的一样。

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。

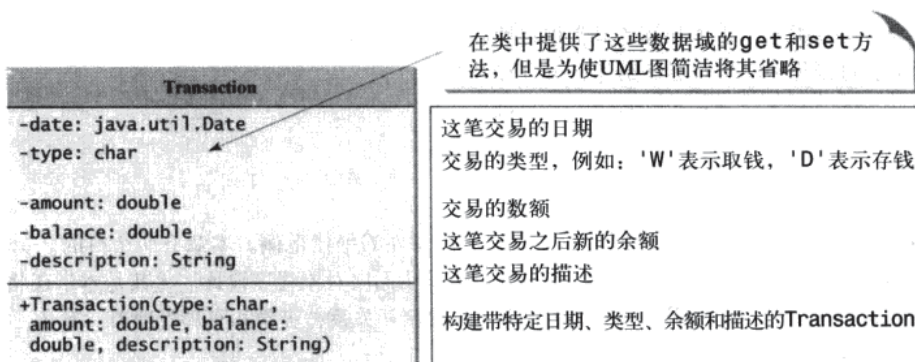


图11-7 Transaction类描述银行账户的一笔交易

第12章

Introduction to Java Programming, 8E

图形用户界面基础

学习目标

- 区分Swing和AWT的不同 (12.2节)。
- 描述Java GUI API的层次体系结构 (12.3节)。
- 使用框架、面板和简单GUI组件创建用户界面 (12.4节)。
- 理解布局管理器的作用 (12.5节)。
- 使用FlowLayout、GridLayout和BorderLayout管理器在一个容器中布局组件 (12.5节)。
- 使用JPanel类将面板作为一个子容器 (12.6节)。
- 使用Color类和Font类指定颜色和字体 (12.7~12.8节)。
- 将边界、工具提示、字体和颜色等常用特性应用在Swing组件上 (12.9节)。
- 使用边界可视化地将用户界面组件分组 (12.9节)。
- 使用ImageIcon类创建图形图标 (12.10节)。

12.1 引言

为Java GUI程序设计而设计的API是应用面向对象原理的绝佳范例。本章有两个目的。第一，它介绍了Java GUI程序设计的基础知识。第二，它使用GUI演示面向对象程序设计。尤其是本章还将介绍Java GUI API的框架结构，还要讨论GUI组件以及组件之间的相互关系、容器和布局管理器、颜色、字体、边界、图像图标以及工具提示。

12.2 Swing和AWT

在8.6.3节中使用了简单的GUI例子演示OOP。我们已经使用过像JButton、JLabel、JTextField、JRadioButton和JComboBox这样的GUI组件。为什么GUI组件的类名都有前缀J呢？为什么不是简单地将它命名为Button，而是使用JButton来命名呢？事实上，在包java.awt中已经有一个名为Button的类。

介绍Java的时候，将图形用户界面相关的类捆绑在一起，放在一个称为抽象窗口工具箱（Abstract Window Toolkit, AWT）的库中。AWT适合开发简单的图形用户界面，但并不适合开发复杂的GUI项目。除此之外，AWT更容易发生与特定平台相关的故障。AWT的用户界面组件就被一种更稳定、更通用和更灵活的库取代，这种库称为Swing组件（Swing component）库。大多数Swing组件都是直接用Java代码在画布上绘图的，而java.awt.Window或java.awt.Panel的子类的组件例外，它们必须使用特定平台上自己的GUI来绘图。Swing组件更少地依赖于目标平台并且更少地使用自己的GUI资源。因此，不依赖于自己GUI的Swing组件称为轻量级组件（lightweight component），而AWT组件称为重量级组件（heavyweight component）。

为了区别新的Swing组件类与与它对应的AWT组件类，Swing GUI组件类都以字母J为前缀来命名。尽管在Java中仍然支持AWT组件，但是最好学习如何使用Swing组件编程，因为AWT用户界面组件终究是要退出历史舞台的。本书只使用Swing GUI组件。

12.3 Java GUI API

GUI API包含的类可以分成三个组：组件类（component class）、容器类（container class）和辅助类（helper class）。它们的层次体系结构关系如图12-1所示。

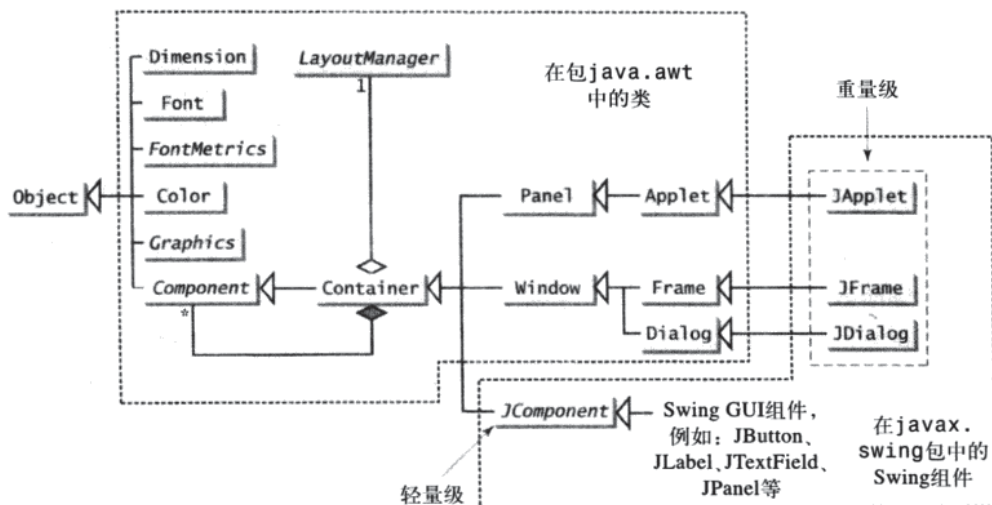


图12-1 Java GUI程序设计使用的是这个体系结构图中所展示的类

组件类是用来创建用户界面的，例如，JButton、JLabel和JTextField。容器类是用来包含其他组件的，例如，JFrame、JPanel和JApplet。辅助类是用来支持GUI组件的，例如，Graphics、Color、Font、FontMetrics和Dimension。

注意 JFrame、JApplet、JDialog和JComponent类及其子类一起放置在javax.swing包中。

图12-1中的其他类均一起放置在java.awt包中。

12.3.1 组件类

Component类的实例可以显示在屏幕上。Component类是包括容器类的所有用户界面类的根类，而JComponent类是所有轻量级Swing组件类的根类。Component和JComponent都是抽象类。抽象类将在第14章中介绍。当目前为止，只需要知道抽象类除了不能使用new操作符创建它的实例之外，其他地方和其他类是一样的。例如，不能使用new JComponent()创建一个JComponent的实例。但是，可以使用JComponent的具体子类的构造方法来创建JComponent的实例。熟悉这些类的继承层次结构是非常必要的。例如，下面语句的结果都显示true：

```
JButton jbtOK = new JButton("OK");
System.out.println(jbtOK instanceof JButton);
System.out.println(jbtOK instanceof JComponent);
System.out.println(jbtOK instanceof Container);
System.out.println(jbtOK instanceof Component);
System.out.println(jbtOK instanceof Object);
```

12.3.2 容器类

一个Container的实例可以包含Component实例。容器类是用于盛装其他GUI组件的GUI组件。Window、Panel、Applet、Frame和Dialog都是AWT组件的容器类。要使用Swing组件作容器，可以使用表12-1中所描述的Container、JFrame、JDialog、JApplet和JPanel。

表12-1 GUI容器类

容器类	说 明
<code>java.awt.Container</code>	用于对组件分组。框架Frame、面板Panel和applet都是它的子类
<code>javax.swing.JFrame</code>	一个不能包含在另一个窗口中的窗口。在Java GUI应用程序中，它用于存放其他Swing用户界面组件
<code>java.swing.JPanel</code>	一个存放用户界面组件的不可见的容器。面板可以嵌套。可以将面板放在包含面板的容器中。JPanel也可用作画图的画布
<code>java.swing.JApplet</code>	Applet的一个子类。必须扩展JApplet才能创建基于Swing的Java applet
<code>java.swing.JDialog</code>	一个弹出式窗口或消息框，一般用作接收来自用户的附加信息或通知事件发生的临时窗口

12.3.3 GUI辅助类

辅助类都不是Component的子类，例如，Graphics、Color、Font、FontMetrics、Dimension和LayoutManager等。它们用来描述GUI组件的属性，例如，图形的内容、颜色、字体以及大小尺寸等，如表12-2所示。

表12-2 GUI辅助类

辅助类	说 明
<code>java.awt.Graphics</code>	一个抽象类，提供绘制字符串、线和简单几何图形的方法
<code>java.awt.Color</code>	处理GUI组件的颜色。例如，可以在像JFrame和JPanel这样的组件中指定背景色或前景色，或者指定绘制的线条、几何图形和字符串的颜色
<code>java.awt.Font</code>	指定GUI组件上文本和图形的字体。例如，可以指定按钮上文本的字型（例如，SansSerif）、风格（例如，粗体）以及大小（例如，24号）
<code>java.awt.FontMetrics</code>	一个获取字体属性的抽象类
<code>java.awt.Dimension</code>	将组件的宽度和高度（以整数为精度）封装在单个对象中
<code>java.awt.LayoutManager</code>	指定组件在容器中如何放置

注意 辅助类是在包java.awt中的。Swing组件不能取代AWT中的全部类，只能替代AWT GUI的组件类（例如，Button、TextField、TextArea）。AWT辅助类在GUI程序设计中仍然很有用。

12.4 框架

创建一个用户界面需要创建一个框架或一个applet来存放用户界面组件。在第18章中将介绍Java applet的创建。本节先介绍框架。

12.4.1 创建一个框架

使用JFrame类创建一个框架，如图12-2所示。

程序清单12-1创建了一个框架。

程序清单12-1 MyFrame.java

```

1 import javax.swing.JFrame;
2
3 public class MyFrame {
4     public static void main(String[] args) {

```




```

5   JFrame frame = new JFrame("MyFrame"); // Create a frame
6   frame.setSize(400, 300); // Set the frame size
7   frame.setLocationRelativeTo(null); // Center a frame
8   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9   frame.setVisible(true); // Display the frame
10  }
11  }

```

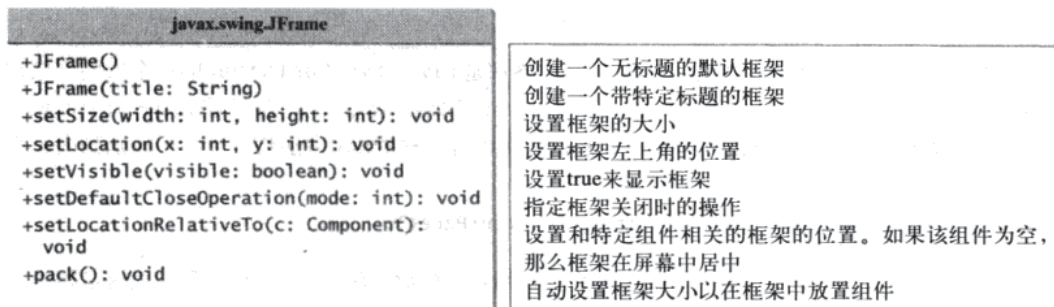
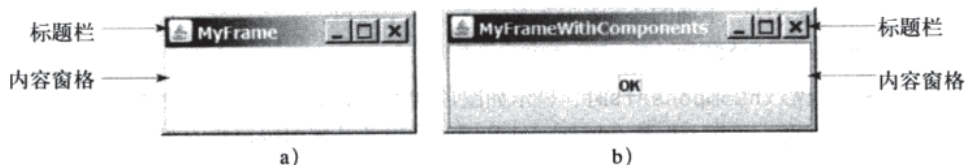


图12-2 JFrame是存放GUI组件的最高层的容器

直到调用`frame.setVisible(true)`方法之后才会显示框架。`frame.setSize(400,300)`指定框架的宽度为400像素，高度为300像素。如果没有使用`setSize`方法，那么框架的大小就只够显示标题栏。由于`setSize`和`setVisible`方法都被定义在`Component`类中，所以，它们都可以被`JFrame`类所继承。随后还将看到，这些方法在`Component`的很多其他子类中也是非常有用的。

运行程序`MyFrame`时，屏幕上就会显示一个窗口（参见图12-3a）。

图12-3 a) 程序创建并显示一个标题为`MyFrame`的框架；b) 给这个框架添加一个OK按钮

调用`setLocationRelativeTo(null)`（第7行）方法可以在屏幕居中显示框架。调用`setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`（第8行）方法来告诉程序，当框架关闭时结束程序。如果不使用这条语句，就会造成关闭框架后程序并不会结束。这种情况下，就必须在Windows系统中的DOS提示符窗口中按下`Ctrl+C`组合键或者在UNIX系统中使用`kill`命令来结束这个进程。

注意 回顾一下，像素是在屏幕上绘图的最小空间单位。可以将像素想象成一个小长方形，将屏幕认为是由像素铺砌而成的。分辨率表示每平方英寸的像素数。屏幕像素越多，屏幕的分辨率就越高。分辨率越高，可以看到的细节越多。

注意 应该在调用`setLocationRelativeTo(null)`之前调用`setSize(w,h)`将框架居中。

12.4.2 向框架中添加组件

图12-3a显示的框架是空的。可以使用`add`方法在框架中添加组件，如程序清单12-2所示。

程序清单12-2 `MyFrameWithComponents.java`

```

1  import javax.swing.*;
2
3  public class MyFrameWithComponents {
4      public static void main(String[] args) {
5          JFrame frame = new JFrame("MyFrameWithComponents");
6

```

```

7      // Add a button into the frame
8      JButton jbtOK = new JButton("OK");
9      frame.add(jbtOK);
10
11     frame.setSize(400, 300);
12     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13     frame.setLocationRelativeTo(null); // Center the frame
14     frame.setVisible(true);
15 }
16 }

```

每个JFrame都包含一个内容窗格。每个内容窗格都是java.awt.Container的一个实例。像按钮之类的GUI组件放置在框架的内容窗格中。在Java早期的版本中，必须使用JFrame类中的getContentPane方法返回框架的内容窗格，然后调用内容窗格的add方法将一个组件添加到内容窗格中，如下所示：

```

java.awt.Container container = frame.getContentPane();
container.add(jbtOK);

```

这是很麻烦的。所以，Java 5之后的Java新版本允许调用框架的add方法，将组件放置在内容窗格中，如下所示：

```
frame.add(jbtOK);
```

这种新特性称为内容窗格授权 (content-pane delegation)。严格地讲，就是将一个组件添加到框架的内容窗格中。为简便起见，我们就说将组件添加到框架中。

使用new JButton("OK")创建一个JButton对象，并把该对象添加到框架的内容窗格中（第9行）。

定义在Container类中的add(Component comp)方法给容器中添加一个Component实例。由于JButton是Component的一个子类，所以，JButton的实例也是Component的实例。为了从容器中删除组件，可以使用remove方法。下面的语句是从容器中删除一个按钮：

```
container.remove(jbtOK);
```

运行程序MyFrameWithComponents时，显示如图12-3b所示的窗口。不管如何调整窗口的大小，按钮都会显示在框架的中央，并且占据整个框架。这是因为，组件是被内容窗格的布局管理器放置在框架上的，而内容窗格的默认布局管理器就是将按钮放到中央。在下一节中，将会使用几种不同的布局管理器将组件放置在需要的位置上。

12.5 布局管理器

在许多其他窗口系统中，用户界面组件是通过使用硬编码 (hard-code) 的像素度量管理的。例如，将一个按钮放在窗口的 (10, 10) 位置处。使用硬编码的像素度量，这个用户界面可能在一个系统中看上去很好，但在另一个系统上就不正常。Java的布局管理器提供了一种层面的抽象，自动将用户界面映射到所有的窗口系统。

Java的GUI组件都放置在容器中，它们的位置是由容器的布局管理器来管理的。在前面的程序中，并没有指定将OK按钮放置在框架中的什么位置，但是，Java知道应该把它放在哪里，因为在后台工作的布局管理器能够将组件放到正确的位置。布局管理器是使用布局管理器类创建的。

使用setLayout(aLayoutManager)方法在容器中设置布局管理器。例如，可以使用下面的语句创建一个XLayout的实例，并将它置于一个容器内：

```

LayoutManager layoutManager = new XLayout();
container.setLayout(layoutManager);

```

本节介绍三种基本的布局管理器：FlowLayout、GridLayout和BorderLayout。

12.5.1 FlowLayout

FlowLayout是最简单的布局管理器。按照组件添加的顺序，从左到右地将组件排列在容器中。当放满一行时，就开始新的一行。可以使用三个常量FlowLayout.RIGHT、FlowLayout.CENTER和FlowLayout.LEFT之一来指定组件的对齐方式。还可以指定组件之间以像素为单位的间隔。FlowLayout的类图如图12-4所示。

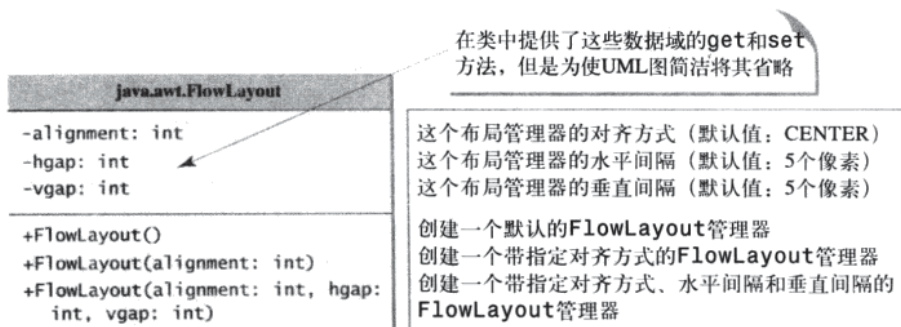


图12-4 FlowLayout逐行放置组件

程序清单12-3给出一个演示流布局的程序。这个程序使用FlowLayout管理器向这个框架添加三个标签和文本域，如图12-5所示。

程序清单12-3 ShowFlowLayout.java

```
1 import javax.swing.JLabel;
2 import javax.swing.JTextField;
3 import javax.swing.JFrame;
4 import java.awt.FlowLayout;
5
6 public class ShowFlowLayout extends JFrame {
7     public ShowFlowLayout() {
8         // Set FlowLayout, aligned left with horizontal gap 10
9         // and vertical gap 20 between components
10        setLayout(new FlowLayout(FlowLayout.LEFT, 10, 20));
11
12        // Add labels and text fields to the frame
13        add(new JLabel("First Name"));
14        add(new JTextField(8));
15        add(new JLabel("MI"));
16        add(new JTextField(1));
17        add(new JLabel("Last Name"));
18        add(new JTextField(8));
19    }
20
21    /** Main method */
22    public static void main(String[] args) {
23        ShowFlowLayout frame = new ShowFlowLayout();
24        frame.setTitle("ShowFlowLayout");
25        frame.setSize(200, 200);
26        frame.setLocationRelativeTo(null); // Center the frame
27        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
28        frame.setVisible(true);
29    }
30 }
```

这个例子使用和12.4节程序不同的风格创建一个程序，12.4节程序的框架是用JFrame类创建的。这个例子扩展了JFrame类，创建一个名为ShowFlowLayout的类（第6行）。这个程序中的main方法创建了一个ShowFlowLayout的实例（第23行）。ShowFlowLayout的构造方法在框架中创建并放置组件。这是创建GUI应用程序时推崇的风格，原因有以下三点：

- 创建一个GUI应用程序意味着创建一个框架，所以，会很自然地扩展JFrame类来定义一个框架。
- 这个框架可能会进一步扩展以添加新的组件或者功能。
- 这个类可以很容易地重用。例如，可以通过创建该类的多个实例来创建多个框架。

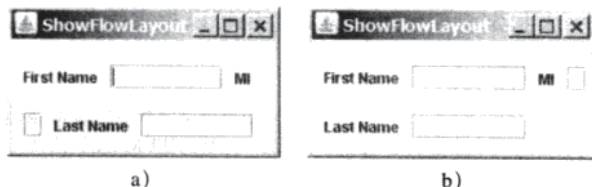


图12-5 组件被FlowLayout管理器逐个地添加到容器的每一行

持续使用一种风格可以使得程序易于阅读。从现在开始，大多数GUI主类都将扩展JFrame类。这个主类的构造方法创建用户界面。main方法创建这个主类的实例，然后显示这个框架。

在这个例子中，使用FlowLayout管理器在框架中放置组件。如果改变框架的大小，组件会自动地重新排列以适合框架。在图12-5a中，第一行有三个组件，但是，在图12-5b中，第一行有四个组件，这是因为宽度增加了。

如果使用setLayout(new FlowLayout(FlowLayout.RIGHT,0,0))代替setLayout语句（第10行），那么所有的按钮行都将右对齐且没有间隔。

在下面的语句中创建了一个匿名的FlowLayout对象（第10行）：

```
setLayout(new FlowLayout(FlowLayout.LEFT, 10, 20));
```

它等价于语句：

```
FlowLayout layout = new FlowLayout(FlowLayout.LEFT, 10, 20);
setLayout(layout);
```

这个代码创建了一个对FlowLayout类的对象layout的显式引用。这个显式引用是没有必要的，因为该对象在ShowFlowLayout类中并不是直接引用的。

假设将同一个按钮在框架中添加10次，那么框架中会出现10个按钮吗？答案是不会，像按钮这样的GUI组件只可以添加到一个容器中，且只能在一个容器中出现一次。将一个按钮向容器添加多次和添加一次是一样的。

警告 在设置布局风格时，不要忘记在布局管理器类之前使用new操作符。例如，setLayout(new FlowLayout())。

注意 构造方法ShowFlowLayout()没有显式地调用构造方法JFrame()，但是构造方法JFrame()被隐式调用。参见11.3.2节。

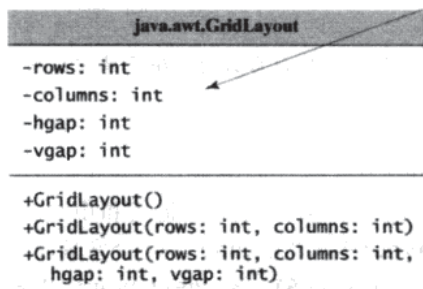
12.5.2 GridLayout

GridLayout管理器以网格（矩阵）的形式管理组件。组件按照它们添加的顺序从左到右排列，先是第一行，接着是第二行，依此类推。布局管理器GridLayout的类图如图12-6所示。

可以指定网格中的行数和列数。基本规则如下：

1) 行数或列数可以为零，但不能两者都为零。如果一个为零另一个不为零，那么不为零的行或列的大小已经固定，而为零的行或列的大小由布局管理器动态地决定。例如，如果指定一个网格有0行3列10个组件，GridLayout就会创建3个固定的列和4个行，最后1行只包含1个组件。如果指定一个网格有3行0列10个组件，GridLayout就会创建3个固定的行和4个列，最后1行包含2个组件。

2) 如果行数和列数都不为零，那么行数就是主导参数；也就是说，行数是固定的，布局管理器会动态地计算列数。例如，如果指定一个网格有3行3列10个组件，GridLayout就会创建3个固定的行和4个列，最后1行包含2个组件。



在类中提供了这些数据域的get和set方法，但是为使UML图简洁将其省略

这个布局管理器中的行数（默认值：1）
 这个布局管理器中的列数（默认值：1）
 这个布局管理器的水平间隔（默认值：0）
 这个布局管理器的垂直间隔（默认值：0）

创建一个默认GridLayout管理器
 创建一个带指定行数和列数的GridLayout
 创建一个带指定行数和列数、水平间隔、垂直间隔的GridLayout管理器

图12-6 GridLayout将组件放在网格上大小相同的单元中

程序清单12-4给出一个演示网格布局的程序。该程序类似于程序清单12-3中的程序。它添加三个标签和三个文本域到GridLayout的框架而不是FlowLayout的框架，如图12-7所示。

程序清单12-4 ShowGridLayout.java

```

1 import javax.swing.JLabel;
2 import javax.swing.JTextField;
3 import javax.swing.JFrame;
4 import java.awt.GridLayout;
5
6 public class ShowGridLayout extends JFrame {
7     public ShowGridLayout() {
8         // Set GridLayout, 3 rows, 2 columns, and gaps 5 between
9         // components horizontally and vertically
10        setLayout(new GridLayout(3, 2, 5, 5));
11
12        // Add labels and text fields to the frame
13        add(new JLabel("First Name"));
14        add(new JTextField(8));
15        add(new JLabel("MI"));
16        add(new JTextField(1));
17        add(new JLabel("Last Name"));
18        add(new JTextField(8));
19    }
20
21    /** Main method */
22    public static void main(String[] args) {
23        ShowGridLayout frame = new ShowGridLayout();
24        frame.setTitle("ShowGridLayout");
25        frame.setSize(200, 125);
26        frame.setLocationRelativeTo(null); // Center the frame
27        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
28        frame.setVisible(true);
29    }
30 }
  
```

如果改变这个框架的大小，那么按钮的布局保持不变（也就是说，行数和列数不变，间隔也不变）。在GridLayout的容器中，所有组件的大小都被认为是一样的。

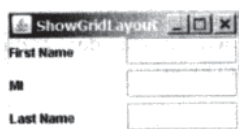


图12-7 GridLayout管理器将容器分为网格，然后添加组件逐行地填充每个格子

使用setLayout(new GridLayout(3,10))代替setLayout语句（第10行）还是会得到3行2列。

因为行的参数非零，所以列的参数被忽略。列的实际数量是由布局管理器计算出来的。

如果用`setLayout(new GridLayout(4,2))`或者用`setLayout(new GridLayout(2,2))`代替`setLayout`语句（第10行），会发生什么情况？请自己试一试。

注意 在`FlowLayout`和`GridLayout`两个布局管理器中，组件添加到容器的顺序是很重要的。它决定了组件在容器中的位置。

12.5.3 BorderLayout

`BorderLayout`管理器将容器分成五个区域：东区、南区、西区、北区和中央。使用`add(Component, index)`方法可以将组件添加到`BorderLayout`中，其中`index`是一个常量，取值为`BorderLayout.EAST`、`BorderLayout.SOUTH`、`BorderLayout.WEST`、`BorderLayout.NORTH`或`BorderLayout.CENTER`。`BorderLayout`的类图如图12-8所示。

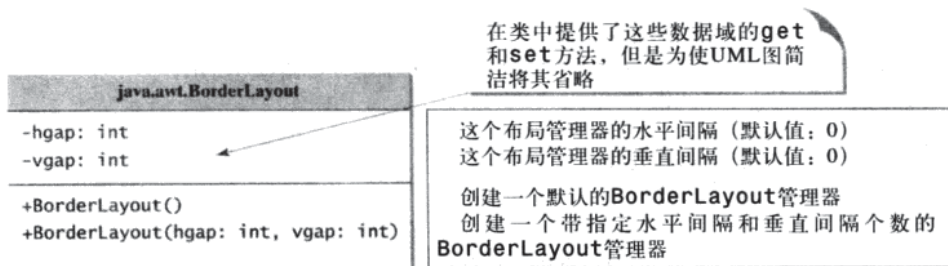


图12-8 BorderLayout将组件放置在五个区域中

组件根据它们最合适的尺寸和它们在容器中的位置来放置。南、北组件可以水平拉伸；东、西组件可以垂直拉伸；中央组件既可以水平拉伸也可以竖直拉伸以填充空白空间。

程序清单12-5给出一个演示边界布局的程序。程序将5个标有`East`、`South`、`West`、`North`和`Center`的按钮添加到一个`BorderLayout`管理器的框架中，如图12-9所示。

程序清单12-5 ShowBorderLayout.java

```
1 import javax.swing.JButton;
2 import javax.swing.JFrame;
3 import java.awt.BorderLayout;
4
5 public class ShowBorderLayout extends JFrame {
6     public ShowBorderLayout() {
7         // Set BorderLayout with horizontal gap 5 and vertical gap 10
8         setLayout(new BorderLayout(5, 10));
9
10        // Add buttons to the frame
11        add(new JButton("East"), BorderLayout.EAST);
12        add(new JButton("South"), BorderLayout.SOUTH);
13        add(new JButton("West"), BorderLayout.WEST);
14        add(new JButton("North"), BorderLayout.NORTH);
15        add(new JButton("Center"), BorderLayout.CENTER);
16    }
17
18    /** Main method */
19    public static void main(String[] args) {
20        ShowBorderLayout frame = new ShowBorderLayout();
21        frame.setTitle("ShowBorderLayout");
22        frame.setSize(300, 200);
23        frame.setLocationRelativeTo(null); // Center the frame
24        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
25        frame.setVisible(true);
26    }
27 }
```

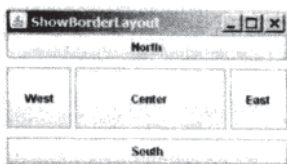


图12-9 BorderLayout将容器分为五个区域，每个区域都可以包含一个组件

将按钮添加到框架中（第11~15行）。注意，BorderLayout的add方法与FlowLayout和GridLayout的add方法不同，使用BorderLayout管理器指定组件放置的位置。

没有必要将组件放得占满整个区域。如果在程序中删掉东区按钮并重新运行它，将会发现中央按钮向右延伸占据东区。

注意 BorderLayout将省略的下标说明解释为BorderLayout.CENTER。例如，add(component)和add(Component, BorderLayout.CENTER)是一样的。如果要在BorderLayout的容器中添加两个组件，如下所示：

```
container.add(component1);
container.add(component2);
```

只会显示最后一个组件。

12.5.4 布局管理器的属性

可以动态地改变布局管理器的属性。FlowLayout具有属性alignment、hgap和vgap。可以使用setAlignment、setHgap和setVgap方法来表明对齐方式、水平间隔和垂直间隔。GridLayout具有属性rows、columns、hgap和vgap。可以使用setRows、setColumns、setHgap和setVgap方法来指定行数、列数以及水平间隔和垂直间隔。BorderLayout具有属性hgap和vgap。可以使用setHgap和setVgap方法来指定水平间隔和垂直间隔。

在前几节中，因为一旦创建了布局管理器，它的属性就不能改变，所以使用的都是匿名布局管理器。如果需要动态地改变布局管理器的属性，布局管理器必须用一个变量显式地引用。然后，可以通过这个变量来改变布局管理器的属性。例如，下面的代码创建一个布局管理器并且设置它的属性：

```
// Create a layout manager
FlowLayout flowLayout = new FlowLayout();

// Set layout properties
flowLayout.setAlignment(FlowLayout.RIGHT);
flowLayout.setHgap(10);
flowLayout.setVgap(20);
```

12.6 使用面板作为子容器

假设要在框架中放置十个按钮和一个文本域。按钮以网格形式放置，文本域单独占一行。如果将所有这些组件放在一个单独的容器中，是很难达到要求的视觉效果。使用Java图形用户界面进行程序设计，可以将一个窗口分成几个面板。面板的作用就是分组放置用户界面组件的子容器。可以将这些按钮添加到一个面板中，然后再将这个面板添加到框架中。

面板的Swing版本是JPanel。可以使用new JPanel()创建一个带默认FlowLayout管理器的面板，也可以使用new JPanel(LayoutManager)创建一个带特定布局管理器的面板。使用add(Component)方法可以向面板添加一个组件。例如，下面的代码创建了一个面板并且给它添加一个按钮：

```
JPanel p = new JPanel();
p.add(new JButton("OK"));
```

面板可以放到一个框架中或者放在另一个面板中。下面的语句将面板p放到框架f中：

```
f.add(p);
```

程序清单12-6是一个演示使用面板作为子容器的例子。程序创建了一个微波炉的用户界面，如图12-10所示。

程序清单12-6 TestPanels.java

```
1 import java.awt.*;
2 import javax.swing.*;
3
4 public class TestPanels extends JFrame {
5     public TestPanels() {
6         // Create panel p1 for the buttons and set GridLayout
7         JPanel p1 = new JPanel();
8         p1.setLayout(new GridLayout(4, 3));
9
10        // Add buttons to the panel
11        for (int i = 1; i <= 9; i++) {
12            p1.add(new JButton(" " + i));
13        }
14
15        p1.add(new JButton(" " + 0));
16        p1.add(new JButton("Start"));
17        p1.add(new JButton("Stop"));
18
19        // Create panel p2 to hold a text field and p1
20        JPanel p2 = new JPanel(new BorderLayout());
21        p2.add(new JTextField("Time to be displayed here"),
22            BorderLayout.NORTH);
23        p2.add(p1, BorderLayout.CENTER);
24
25        // add contents into the frame
26        add(p2, BorderLayout.EAST);
27        add(new JButton("Food to be placed here"),
28            BorderLayout.CENTER);
29    }
30
31    /** Main method */
32    public static void main(String[] args) {
33        TestPanels frame = new TestPanels();
34        frame.setTitle("The Front View of a Microwave Oven");
35        frame.setSize(400, 250);
36        frame.setLocationRelativeTo(null); // Center the frame
37        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
38        frame.setVisible(true);
39    }
40 }
```

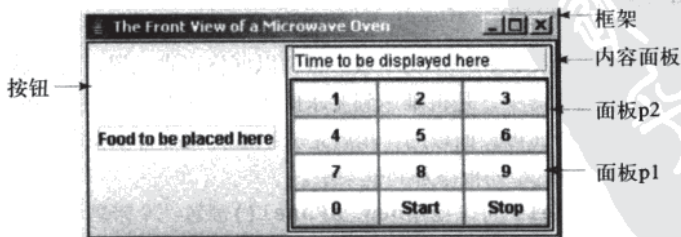


图12-10 程序使用面板组织组件

setLayout方法是在java.awt.Container中定义的。由于JPanel是Container的子类，所以，可以使用setLayout在面板中设置一个新的布局管理器（第8行）。第7~8行可以用语句JPanel p1 =

`new JPanel(new GridLayout(4,3))`代替。

为了得到所需的布局，程序使用`GridLayout`的面板`p1`将数字按钮、`Stop`按钮和`Start`按钮放在一组，使用`BorderLayout`的面板`p2`将文本域放在北区，将`p1`放在中央。表示食物的按钮放在框架的中央，而`p2`放在框架的东区。

语句（第21~22行）

```
p2.add(new JTextField("Time to be displayed here"),
        BorderLayout.NORTH);
```

创建了一个`JTextField`的实例，然后将它添加到`p2`。`JTextField`是一个GUI组件，可以被用户用作输入和显示值。

注意 `Container`类是像`JButton`这样的GUI组件类的超类，这点是很值得注意的。每个GUI组件都是一个容器。理论上讲，可以使用`setLayout`方法在按钮中设置布局，并且将组件添加到按钮中，因为`Container`类中所有的公共方法都被继承到`JButton`，但是，如果不是实际情况不得已，是不应该将按钮作为容器使用的。

12.7 Color类

可以使用`java.awt.Color`类为GUI组件设置颜色。颜色是由红、绿、蓝这三原色构成的，每种原色都用一个`int`值表示它的深度，取值范围从0（最暗度）到255（最亮度）。这就是通常所说的RGB模式（RGB model）。

可以使用下面的构造方法创建一个`color`对象：

```
public Color(int r, int g, int b);
```

其中`r`、`g`和`b`指定某个颜色的红、绿、蓝成分。例如，

```
Color color = new Color(128, 100, 100);
```

注意 参数`r`、`g`和`b`的取值都在0到255之间。如果传递给参数的值超过这个范围，就会导致一个`IllegalArgumentException`异常。

可以使用定义在`java.awt.Component`类中的`setBackground(Color c)`和`setForeground(Color c)`方法来设置一个组件的背景色和前景色。下面是设置一个按钮背景色和前景色的例子：

```
JButton jbtOK = new JButton("OK");
jbtOK.setBackground(color);
jbtOK.setForeground(new Color(100, 1, 1));
```

还可以选择使用`java.awt.Color`中定义为常量的13种标准颜色（`BLACK`黑色、`BLUE`蓝色、`CYAN`青色、`DARK_GRAY`深灰、`GRAY`灰色、`GREEN`绿色、`LIGHT_GRAY`淡灰、`MAGENTA`洋红、`ORANGE`橘色、`PINK`粉红、`RED`大红、`WHITE`白色和`YELLOW`黄色）之一。例如，下面的代码可以将按钮的前景色设置成红色：

```
jbtOK.setForeground(Color.RED);
```

12.8 Font类

可以使用`java.awt.Font`类创建一种字体，然后使用`Component`类中的`SetFont`方法设置组件的字体。

`Font`的构造方法是：

```
public Font(String name, int style, int size);
```

可以从`SansSerif`、`Serif`、`Monospaced`、`Dialog`或`DialogInput`中选择一种字体名，可以从`Font.PLAIN(0)`、`Font.BOLD(1)`、`Font.ITALIC(2)`和`Font.BOLD + Font.ITALIC(3)`中选择风格，然后指定正整数的字体大小。例如，下面的语句创建两种字体，并且给按钮设置一种字体：

```
Font font1 = new Font("SansSerif", Font.BOLD, 16);
Font font2 = new Font("Serif", Font.BOLD + Font.ITALIC, 12);

JButton jbtOK = new JButton("OK");
jbtOK.setFont(font1);
```

提示 如果系统支持其他字体,例如,“Times New Roman”,那就可以使用它创建一个Font对象。为了找出系统上可用的字体,需要使用java.awt.GraphicsEnvironment类的静态方法getLocalGraphicsEnvironment()创建这个类的一个实例。GraphicsEnvironment是描述特定系统上图形环境的一个抽象类。可以使用它的getAllFonts()方法来获取系统中所有可用的字体,也可以使用它的getAvailableFontFamilyNames()方法来获取所有可用字体的名字。例如,下面的语句打印系统中所有可用字体的名字:

```
GraphicsEnvironment e =
    GraphicsEnvironment.getLocalGraphicsEnvironment();
String[] fontnames = e.getAvailableFontFamilyNames();

for (int i = 0; i < fontnames.length; i++)
    System.out.println(fontnames[i]);
```

12.9 Swing GUI组件的公共特性

在本章中,已经使用了一些GUI组件(例如,JFrame、Container、JPanel、JButton、JLabel和JTextField)。本书还将介绍更多的GUI组件。理解这些Swing GUI组件的一般特性是很重要的。Component类是所有GUI组件和容器的根。所有Swing GUI组件(除了JFrame、JApplet和JDialog)都是JComponent的子类,如图12-1所示。图12-11列出了Component、Container和JComponent中对像字体、颜色、大小、工具提示文本及其边界这样的属性的常用操作方法。

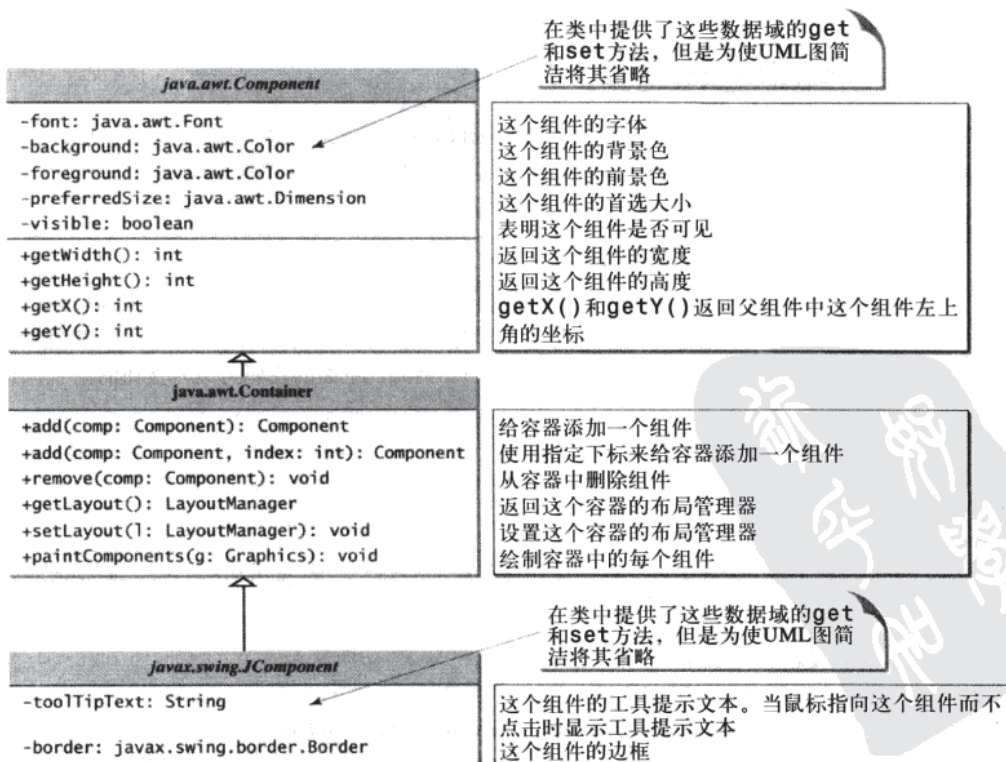


图12-11 所有Swing GUI组件都继承Component、Container和JComponent的公共方法

工具提示 (tool tip) 是将鼠标移动到组件上时, 这个组件上显示的文本。经常用它来描述一个组件的功能。

可以给JComponent类的任何对象设置边界。Swing具有各种类型的边界。为了创建一个带标题的边界, 使用`new TitleBorder(String title)`。为了创建一个线边界, 使用`new LineBorder (Color color,int width)`这里的width表明线的粗细。

程序清单12-7是演示Swing一般特性的例子。该例创建一个面板p1放置三个按钮 (第8行), 创建p2放置两个标签 (第25行), 如图12-12所示。按钮jbtLeft的背景色设置为白色 (第12行), 按钮jbtCenter的前景色设置为绿色 (第13行)。按钮jbtRight的工具提示在第14行设置。在面板p1和p2上设置标题边界 (第18、36行), 而在标签上设置线边界 (第32~33行)。

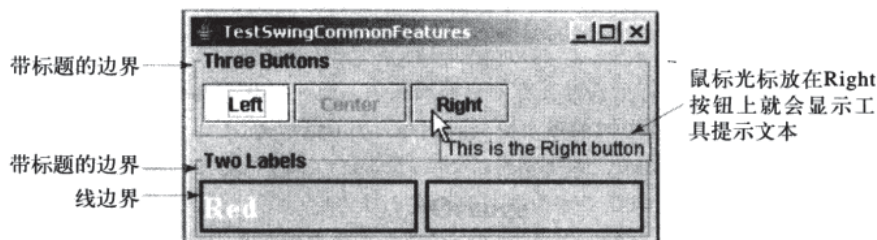


图12-12 在消息面板中设置字体、颜色、边界和工具提示文本

程序清单12-7 TestSwingCommonFeatures.java

```
1 import java.awt.*;
2 import javax.swing.*;
3 import javax.swing.border.*;
4
5 public class TestSwingCommonFeatures extends JFrame {
6     public TestSwingCommonFeatures() {
7         // Create a panel to group three buttons
8         JPanel p1 = new JPanel(new FlowLayout(FlowLayout.LEFT, 2, 2));
9         JButton jbtLeft = new JButton("Left");
10        JButton jbtCenter = new JButton("Center");
11        JButton jbtRight = new JButton("Right");
12        jbtLeft.setBackground(Color.WHITE);
13        jbtCenter.setForeground(Color.GREEN);
14        jbtRight.setTooltipText("This is the Right button");
15        p1.add(jbtLeft);
16        p1.add(jbtCenter);
17        p1.add(jbtRight);
18        p1.setBorder(new TitledBorder("Three Buttons"));
19
20        // Create a font and a line border
21        Font largeFont = new Font("TimesRoman", Font.BOLD, 20);
22        Border lineBorder = new LineBorder(Color.BLACK, 2);
23
24        // Create a panel to group two labels
25        JPanel p2 = new JPanel(new GridLayout(1, 2, 5, 5));
26        JLabel jlblRed = new JLabel("Red");
27        JLabel jlblOrange = new JLabel("Orange");
28        jlblRed.setForeground(Color.RED);
29        jlblOrange.setForeground(Color.ORANGE);
30        jlblRed.setFont(largeFont);
31        jlblOrange.setFont(largeFont);
32        jlblRed.setBorder(lineBorder);
33        jlblOrange.setBorder(lineBorder);
34        p2.add(jlblRed);
35        p2.add(jlblOrange);
36        p2.setBorder(new TitledBorder("Two Labels"));
37    }
}
```



```

38     // Add two panels to the frame
39     setLayout(new GridLayout(2, 1, 5, 5));
40     add(p1);
41     add(p2);
42 }
43
44 public static void main(String[] args) {
45     // Create a frame and set its properties
46     JFrame frame = new TestSwingCommonFeatures();
47     frame.setTitle("TestSwingCommonFeatures");
48     frame.setSize(300, 150);
49     frame.setLocationRelativeTo(null); // Center the frame
50     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
51     frame.setVisible(true);
52 }
53 }

```

注意 在不同的组件中同样的属性会有不同的默认值。例如，JFrame中的visible属性默认值为false，但是在JComponent的每个实例（例如，JButton和JLabel）中该属性默认值都为true。为了显示一个JFrame，必须调用setVisible(true)将属性visible设置为true，但是不必为JButton或JLabel设置该属性，因为它已经为true。为使JButton或JLabel不可见，可以调用setVisible(false)。请运行这个程序，然后看看在第37行插入下面两条语句之后的效果：

```

jbtLeft.setVisible(false);
jlblRed.setVisible(false);

```

12.10 图像图标

图标是一个大小固定的图片；通常情况下，它都比较小，用来装饰组件。图像通常存储在图像文件中。Java目前支持三种图像格式：GIF（图像交换格式）、JPEG（联合图像专家组）以及PNG（便携网络图片）。这些类型的图像文件名分别以.gif、.jpg和.png结尾。如果有一个其他格式的位图文件或图像文件，可以使用图像处理工具将它们转为GIF、JPEG或PNG格式，以便于在Java中使用。

为了显示一个图像图标，首先使用new java.swing.ImageIcon(filename)创建一个ImageIcon对象。例如，下面的语句使用当前类路径下的image目录的图像文件us.gif来创建一个图标：

```
ImageIcon icon = new ImageIcon("image/us.gif");
```

“image/us.gif”放置在“c:\book\image\us.gif”下。反斜杠（\）是Windows系统的文件路径符号。在UNIX系统中，应该使用的是斜杠（/）。在Java中，斜杠（/）用来表示在Java的classpath下的相对文件路径（例如，本例中的image/us.gif）。

提示 在Windows系统中，文件名是不区分大小写的，但是在UNIX系统中是区分大小写的。为了使程序可以在所有平台上运行，就将所有的图像文件统一使用小写命名。

使用new JLabel(imageIcon)或new JButton(imageIcon)在标签或按钮上显示图像图标。程序清单12-8演示如何在标签和按钮上显示图标。这个例子创建两个带图标的标签和两个带图标的按钮，如图12-13所示。

程序清单12-8 TestImageIcon.java

```

1 import javax.swing.*;
2 import java.awt.*;
3
4 public class TestImageIcon extends JFrame {
5     private ImageIcon usIcon = new ImageIcon("image/us.gif");
6     private ImageIcon myIcon = new ImageIcon("image/my.jpg");
7     private ImageIcon frIcon = new ImageIcon("image/fr.gif");
8     private ImageIcon ukIcon = new ImageIcon("image/uk.gif");

```



```

9
10 public TestImageIcon() {
11     setLayout(new GridLayout(1, 4, 5, 5));
12     add(new JLabel(usIcon));
13     add(new JLabel(myIcon));
14     add(new JButton(frIcon));
15     add(new JButton(ukIcon));
16 }
17
18 /** Main method */
19 public static void main(String[] args) {
20     TestImageIcon frame = new TestImageIcon();
21     frame.setTitle("TestImageIcon");
22     frame.setSize(200, 200);
23     frame.setLocationRelativeTo(null); // Center the frame
24     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
25     frame.setVisible(true);
26 }
27 }

```

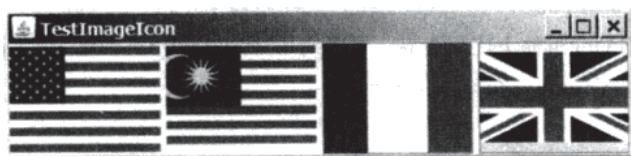


图12-13 在标签和按钮上显示图像图标

注意 GUI组件不能被多个容器共享，因为一个GUI组件只能在一个容器中出现一次。因此，组件和容器之间的关系是用实心菱形表示的组合关系，如图12-1所示。

注意 边界和图标是可以共享的。这样，可以创建一个边界或图标，然后使用它来设置任意一个GUI组件的border或icon属性。例如，下面的语句就是将两个面板p1和p2设置为边界b：

```
p1.setBorder(b);
p2.setBorder(b);
```

下面的语句是在两个按钮jbt1和jbt2中设置一个图标：

```
jbt1.setIcon(icon);
jbt2.setIcon(icon);
```

提示 启动画面是应用程序开始启动时显示的图像。如果程序要占用很多时间去加载，就要显示一个启动画面来警示用户。例如，下面的命令：

```
java -splash:image/us.gif TestImageIcon
```

就会实现加载程序TestImageIcon时显示一个图像。

关键术语

AWT (抽象窗口工具箱)

heavyweight component (重量级组件)

lightweight component (轻量级组件)

layout manager (布局管理器)

Swing (Swing组件)

splash screen (启动画面)

本章小结

- 每个容器都有一个布局管理器，它按照所需的位置在容器中定位和放置组件。三个简单且常用的布局管理器是FlowLayout、GridLayout和BorderLayout。

- 可以将JPanel作为子容器来将组件分组以得到所需的布局。
- 使用add方法将组件放到JFrame和JPanel。默认情况下, 框架的布局是BorderLayout, 而JPanel的布局是FlowLayout。
- 可以使用java.awt.Color类设置GUI组件的颜色。颜色是由红、绿和蓝三原色组成的, 每种颜色都是用一个无符号的字节值表示它的深度, 它的取值范围从0 (最暗度) 到255 (最亮度)。这就是通常所说的RGB模式。
- 为了创建一个Color对象, 应该使用new Color(r,g,b), 这里的r、g和b表示该颜色的红色、绿色和蓝色的成分。还可以使用在java.awt.Color中定义的13种标准色 (BLACK黑色、BLUE蓝色、CYAN青色、DARK_GRAY深灰、GRAY灰色、GREEN绿色、LIGHT_GRAY淡灰、MAGENTA洋红、ORANGE橘色、PINK粉红、RED大红、WHITE白色和YELLOW黄色)。
- 每个Swing GUI组件都是javax.swing.JComponent的子类, 而JComponent则是java.awt.Component的子类。Component中的属性font、background、foreground、height、width和preferredSize都被它们的子类继承, JComponent中的toolTipText和border属性也是如此。
- 可以在任何一个Swing组件上使用边界。可以使用ImageIcon类创建一个图像图标, 然后将它显示在标签和按钮上。按钮和边界是可以共享的。

复习题

12.3~12.4节

12.1 哪个类是Java GUI组件类的根? 容器类是Component的子类吗? 哪个类是Swing GUI组件的根?

12.2 解释像java.awt.Button这样的AWT GUI组件和像javax.swing.JButton这样的Swing GUI组件的不同。

12.3 如何创建一个框架? 如何设置框架的大小? 如何获取框架的大小? 怎样给框架中添加组件? 如果将程序清单12-2中的语句frame.setSize(400,300)和语句frame.setVisible(true)互换位置, 将会发生什么?

12.4 判断下面语句是真还是假:

- (1) 可以给框架添加一个按钮。
- (2) 可以给面板添加一个框架。
- (3) 可以给框架添加一个面板。
- (4) 可以给面板或框架添加任意数量的组件。
- (5) 可以从JButton、JPanel或者JFrame派生一个类。

12.5 下面的程序是要在框架中显示一个按钮, 但是什么也没有显示出来。这个程序有什么问题?

```
1 public class Test extends javax.swing.JFrame {
2     public Test() {
3         add(new javax.swing.JButton("OK"));
4     }
5
6     public static void main(String[] args) {
7         javax.swing.JFrame frame = new javax.swing.JFrame();
8         frame.setSize(100, 200);
9         frame.setVisible(true);
10    }
11 }
```

12.6 下面的哪条语句有语法错误?

```
Component c1 = new Component();
JComponent c2 = new JComponent();
Component c3 = new JButton();
```

```

JComponent c4 = new JButton();
Container c5 = new JButton();
c5.add(c4);
Object c6 = new JButton();
c5.add(c6);

```

12.5节

12.7 为什么需要使用布局管理器？框架的默认布局管理器是什么？如何将一个组件添加到框架中？

12.8 描述FlowLayout。如何创建一个FlowLayout管理器？如何将一个组件添加到FlowLayout容器中？添加到FlowLayout容器中的组件数量有限制吗？

12.9 描述GridLayout。如何创建一个GridLayout管理器？如何将一个组件添加到GridLayout容器中？添加到GridLayout容器中的组件数量有限制吗？

12.10 描述BorderLayout。如何创建一个BorderLayout管理器？如何将一个组件添加到BorderLayout容器中？

12.6节

12.11 如何创建一个带特定布局管理器的面板？

12.12 JPanel的默认布局管理器是什么？如何向JPanel添加一个组件？

12.13 可以在面板中使用setTitle方法吗？使用面板的目的是什么？

12.14 由于像JButton这样的GUI组件类是Container的子类，那么是否能够将组件添加到按钮中？

12.7~12.8节

12.15 如何创建一种颜色？使用new Color(400,200,300)创建Color时会有什么问题？下面两种颜色哪种比较深：new Color(10,0,0)、new Color(200,0,0)？

12.16 如何创建一种字体？如何找出系统中所有可用的字体？

12.9~12.10节

12.17 如何设置Swing GUI组件的背景色、前景色、字体和工具提示文本？为什么在下面的代码中没有显示工具提示文本？

```

1 import javax.swing.*;
2
3 public class Test extends JFrame {
4     private JButton jbtOK = new JButton("OK");
5
6     public static void main(String[] args) {
7         // Create a frame and set its properties
8         JFrame frame = new Test();
9         frame.setTitle("Logic Error");
10        frame.setSize(200, 100);
11        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12        frame.setVisible(true);
13    }
14
15    public Test() {
16        jbtOK.setToolTipText("This is a button");
17        add(new JButton("OK"));
18    }
19 }

```

12.18 显示下面代码的输出结果：

```

import javax.swing.*;

public class Test {
    public static void main(String[] args) {
        JButton jbtOK = new JButton("OK");
        System.out.println(jbtOK.isVisible());

        JFrame frame = new JFrame();

```

数字水印

PDG

```

        System.out.println(frame.isVisible());
    }
}

```

12.19 如何从类目录下的image/us.gif文件创建一个ImageIcon。

12.20 如果要将一个按钮如下所示地多次添加到容器中会发生什么情况？它是否会引起语法错误？它是否会引起运行错误？

```

JButton jbt = new JButton();
JPanel panel = new JPanel();
panel.add(jbt);
panel.add(jbt);
panel.add(jbt);

```

12.21 下面的代码会显示三个按钮吗？按钮会显示同样的图标吗？

```

1 import javax.swing.*;
2 import java.awt.*;
3
4 public class Test extends JFrame {
5     public static void main(String[] args) {
6         // Create a frame and set its properties
7         JFrame frame = new Test();
8         frame.setTitle("ButtonIcons");
9         frame.setSize(200, 100);
10        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11        frame.setVisible(true);
12    }
13
14    public Test() {
15        ImageIcon usIcon = new ImageIcon("image/us.gif");
16        JButton jbt1 = new JButton(usIcon);
17        JButton jbt2 = new JButton(usIcon);
18
19        JPanel p1 = new JPanel();
20        p1.add(jbt1);
21
22        JPanel p2 = new JPanel();
23        p2.add(jbt2);
24
25        JPanel p3 = new JPanel();
26        p2.add(jbt1);
27
28        add(p1, BorderLayout.NORTH);
29        add(p2, BorderLayout.SOUTH);
30        add(p3, BorderLayout.CENTER);
31    }
32 }

```

12.22 GUI组件可以共享边框或图标吗？

编程练习题

12.5~12.6节

12.1（使用FlowLayout管理器）编写一个满足下面需求的程序（参见图12-14）：

- 创建一个框架，并且将它的布局设置为FlowLayout。
- 创建两个面板，然后将它们添加到这个框架。
- 每个面板包含三个按钮。面板使用FlowLayout布局管理器。

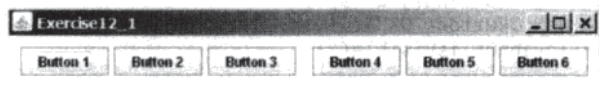


图12-14 练习题12.1将前三个按钮放到一个面板上而将另外三个按钮放到另一个面板上

- 12.2 (使用BorderLayout管理器) 改写前面的程序, 创建同样的用户界面, 但是在框架上不使用FlowLayout, 而是采用BorderLayout。将一个面板放在框架的南区, 而将另一个放在中央。
- 12.3 (使用GridLayout管理器) 改写前面的程序, 创建同样的用户界面。在这些面板中不用FlowLayout, 而是采用两行三列的GridLayout。
- 12.4 (使用JPanel对按钮分组) 改写前面的程序, 创建同样的用户界面。不是分别创建面板和按钮, 而是定义一个类扩展JPanel类。在面板类中放置三个按钮, 然后从这个用户定义的面板类创建两个面板。
- 12.5 (显示标签) 编写一个程序, 实现在四个标签上显示四行文本, 如图12-15a所示。在每个标签上添加一条线边界。

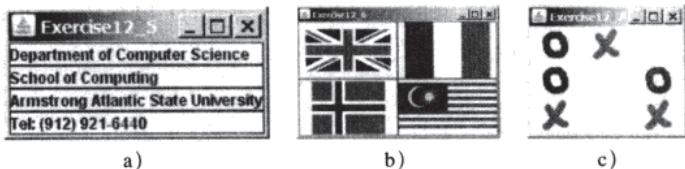


图12-15 a) 练习题12.5显示四个标签; b) 练习题12.6显示四个图标;
c) 练习题12.7显示图像图标在标签中的井字游戏的棋盘

12.7~12.10节

- 12.6 (显示图标) 编写一个程序, 实现在四个标签上显示四个图标, 如图12-15b所示。在每个标签上添加一个线边界 (使用你选择的任意图像或者从本书中源代码中获得的任意图像)。
- **12.7 (游戏: 显示井字游戏的棋盘)** 显示一个包含九个标签的框架。标签可以显示为一个X字形的图像图标、一个O字形的图像图标, 或者什么都不显示, 如图12-15c所示。显示什么是随机决定的。使用Math.random()方法产生一个整数0、1或2, 相应地显示十字形图像图标、O形图像图标或者什么都不显示。十字形的图像和O形的图像放在www.cs.armstrong.edu/liang/intro8e/book.zip中的图像目录下的x.gif和o.gif文件中。
- *12.8 (Swing通用特性)** 显示包含六个标签的框架。将标签背景色设置为白色。将标签前景色分别设置为黑色、蓝色、青色、绿色、洋红色和橙色, 如图12-16a所示。设置每个标签的边界为黄色的线边界。设置每个标签的字体为TimesRoman、加粗、20像素。将每个标签的文本和工具提示文本都设置为它的前景色的名字。

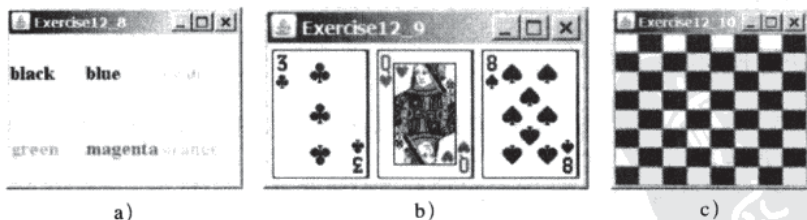


图12-16 a) 框架中放置六个标签; b) 随机选择三张牌; c) 使用按钮显示一个棋盘

- *12.9 (游戏: 显示三张牌)** 显示包含三个标签的框架。每个标签显示一张牌, 如图12-16b所示。牌的图像文件命名为1.png, 2.png, ..., 54.png, 并且存储在image/card目录中。这三张牌是不同的并且是随机选取的。这些图像文件都可以从www.cs.armstrong.edu/liang/intro8e/book.zip上获取。
- *12.10 (游戏: 显示一个棋盘)** 编写一个程序, 显示一个棋盘, 棋盘中的每一个白色格和黑色格都是将背景色设置为黑色或者白色的JButton, 如图12-16c所示。

第13章

Introduction to Java Programming, 8E

异常处理

学习目标

- 了解异常和异常处理的概况 (13.2节)。
- 探究使用异常处理的优点 (13.3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 (13.4节)。
- 在方法头中声明异常 (13.5.1节)。
- 在方法中抛出异常 (13.5.2节)。
- 编写 **try-catch** 块处理异常 (13.5.3节)。
- 解释异常是如何传播的 (13.5.3节)。
- 在 **try-catch** 块中使用 **finally** 子句 (13.6节)。
- 只为非预期错误使用异常 (13.7节)。
- 在 **catch** 块中重新抛出异常 (13.8节)。
- 创建链式异常 (13.9节)。
- 定义自定义的异常类 (13.10节)。

13.1 引言

在程序运行过程中, 如果环境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。为了从文件中读取数据, 需要使用 `new Scanner(new File(filename))` 创建一个 **Scanner** 对象 (参见程序清单9-6)。如果该文件不存在, 程序将会出现一个 **FileNotFoundException** 的运行时错误。

在Java中, 异常会导致运行时错误。异常就是一个表示阻止执行正常进行的错误或者情况。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平稳终止呢? 这就是本章要介绍的主题。

13.2 异常处理概述

为了演示异常处理, 包括一个异常是如何创建的以及如何抛出的, 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13-1) 开始。

程序清单13-1 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0除一个整数（回顾一下，一个浮点数除以0是会产生异常的）。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13-2所示。

程序清单13-2 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



为了演示异常处理的概念，包括如何创建、抛出、捕获以及处理异常，我们改写程序清单13-2为如程序清单13-3所示。

程序清单13-3 QuotientWithException.java


```
1 import java.util.Scanner;
2
3 public class QuotientWithException {
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        try {
13            if (number2 == 0)
```

新华书店
PDFG


```

14      throw new ArithmeticException("Divisor cannot be zero");
15
16      System.out.println(number1 + " / " + number2 + " is " +
17          (number1 / number2));
18  }
19  catch (ArithmeticException ex) {
20      System.out.println("Exception: an integer " +
21          "cannot be divided by zero ");
22  }
23
24      System.out.println("Execution continues ...");
25  }
26 }

```

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 ...



程序包含一个try块和一个catch块。try块（第12~18行）包含的是在正常情况下执行的代码。catch块（第19~22行）包含的是在number2为0时执行的代码。在这种情况下，程序会通过执行下面的语句来抛出一个异常：

```
throw new ArithmeticException("Divisor cannot be zero");
```

在这种情况下new ArithmeticException("Divisor cannot be zero")下抛出的值，可以称为一个异常（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。

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。异常被catch块捕获。执行catch块中的代码以处理这个异常（handle the exception）。然后，执行catch块后的语句（第24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，它在执行完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块的头部的标识符ex

```
catch (ArithmeticException 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 try;
    Throw an exception with a throw statement or
    from method if necessary;
    More code to try;
}
catch (type ex) {
    Code to process the exception;
}

```


可以使用try块中的throw语句直接抛出一个异常，或者调用一个可能会抛出异常的方法。

13.3 异常处理的优势

从程序清单13-3中已经看出了一个异常是如何创建、抛出、捕获和处理的。你可能会奇怪这样做有什么好处。为了能看出这些优势，我们改写程序清单13-3，使用方法来计算商，如程序清单13-4所示。

程序清单13-4 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             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         } catch (ArithmeticException ex) {
24             System.out.println("Exception: an integer " +
25                 "cannot be divided by zero ");
26         }
27
28         System.out.println("Execution continues ...");
29     }
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 ...



方法quotient（第4~9行）返回两个整数的商。如果number2为0，它不会返回值。所以，在第6行会抛出一个异常。

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。这个调用者可以处理该异常。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（throw exception）。程序清单13-5给出一个处理调用Scanner(File file)构造方法时出现异常FileNotFoundException的例子。

程序清单13-5 FileNotFoundExceptionDemo.java

```

1 import java.util.Scanner;
2 import java.io.*;
3
4 public class FileNotFoundExceptionDemo {
5     public static void main(String[] args) {
6         Scanner inputFromConsole = new Scanner(System.in);
7         // Prompt the user to enter a file name
8         System.out.print("Enter a file name: ");
9         String filename = inputFromConsole.nextLine();
10
11         try {
12             Scanner inputFromFile = new Scanner(new File(filename));
13             System.out.println("File " + filename + " exists ");
14             // Processing file ...
15         }
16         catch (FileNotFoundException ex) {
17             System.out.println("Exception: " + filename + " not found");
18         }
19     }
20 }

```

Enter a file name: c:\book\Welcome.java Enter
File c:\book\Welcome.java exists



Enter a file name: c:\book\Test10.java Enter
Exception: c:\book\Test10.java not found



该程序为文件创建一个Scanner（第12行）。如果这个文件不存在，那么它的构造方法就会抛出一个异常FileNotFoundException，该异常在catch块中被捕获。

程序清单13-6给出一个处理异常InputMismatchException的例子。

程序清单13-6 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                 // Display the result
13                 System.out.println(
14                     "The number entered is " + number);
15                 continueInput = false;
16             }
17             catch (InputMismatchException ex) {
18                 System.out.println("Try again. (" +
19                     "Incorrect input: an integer is required)");
20                 input.nextLine(); // Discard input
21             }
22         }
23     }

```

新华书店
PDG

```

24     } while (continueInput);
25 }
26 }

```

```

Enter an integer: 3.5
Try again. (Incorrect input: an integer is required)
Enter an integer: 4
The number entered is 4

```



当执行() (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)。

13.4 异常类型

13.3节使用了三个异常ArithmeticException、FileNotFoundException和InputMismatchException。是否还有可以使用的其他类型的异常? 回答是肯定的。在Java API中有很多预定义的异常类。图13-1给出它们中的一部分。

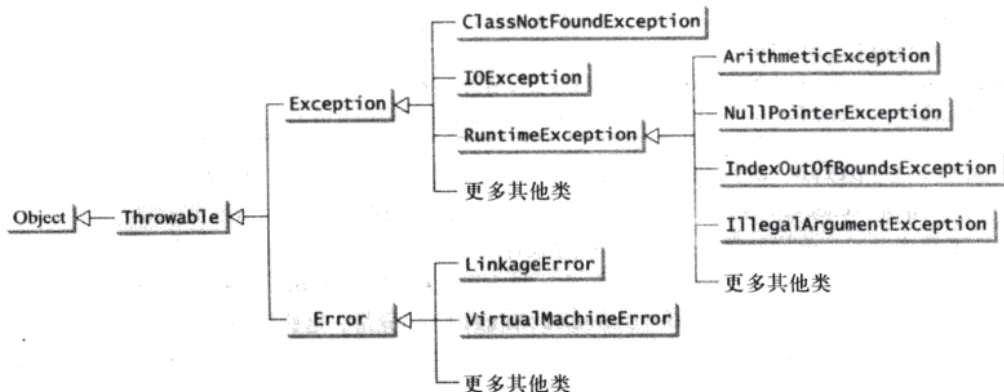


图13-1 抛出的异常都是这个图中给出的类的实例, 或者是这些类的子类的实例

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。可以通过扩展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。

1) 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13-1列出Error子类的一些例子。

2) 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13-2列出Exception类的子类的一些例子。

3) 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13-3列出RuntimeException的子类的一些例子。

表13-1 Error类的子类的例子

类	可能引起异常的原因
LinkageError	一个类对另一个类有某种依赖性, 编译前者后, 后者变得不兼容 Java虚拟机中断或者没有继续运行所必需的资源可用
VirtualMachineError	

表13-2 Exception类的子类的例子

类	可能引起异常的原因
ClassNotFoundException	企图使用一个不存在的类。例如, 如果试图使用命令java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
IOException	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。IOException的子类的例子有InterruptedException、EOFException (EOF是End Of File的缩写) 和FileNotFoundException

表13-3 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	企图通过一个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处理它们。

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语言不允许编写代码捕获或声明免检异常。

警告 目前, Java还没有抛出整数上溢或下溢异常。下面的语句给最大整数加1。

```
int number = Integer.MAX_VALUE + 1;
System.out.println(number);
```

它会显示-2147483648, 这在逻辑上是错误的。引起该问题的原因是上溢; 也就是说, 结果超出了int型值的最大范围。

Java今后的版本可能会通过抛出一个上溢异常来解决这个问题。

13.5 关于异常处理的更多知识

13.4节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13-2所示。

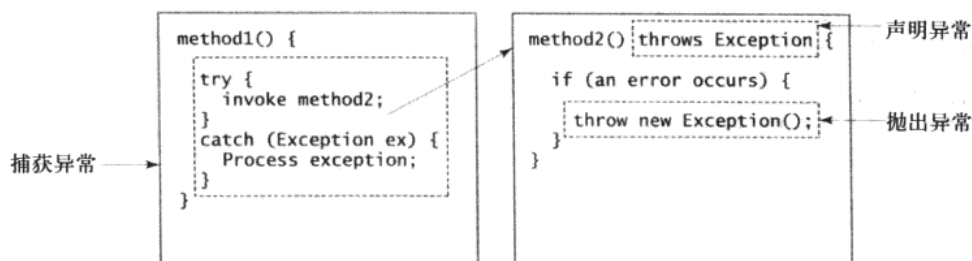


图13-2 Java中的异常处理包括声明异常、抛出异常以及捕获和处理异常

13.5.1 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，如下例所示：

```
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覆盖来声明异常。

13.5.2 抛出异常

检测一个错误的程序可以创建一个正确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，如果你愿意，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。

13.5.3 捕获异常

现在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 exVar3) {
    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13-3所示。考虑下面的情形：

- 1) 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。
- 2) 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。
- 3) 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。
- 4) 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。

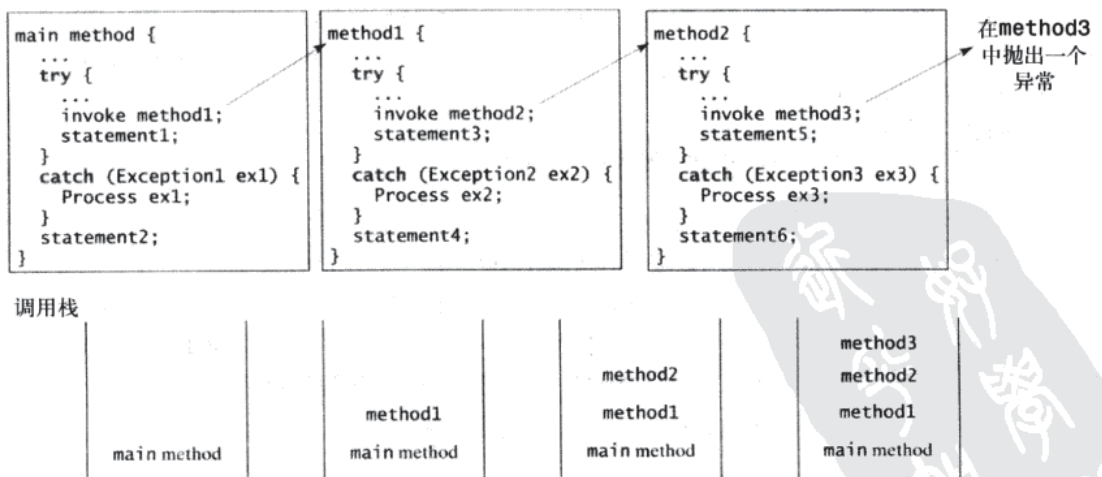
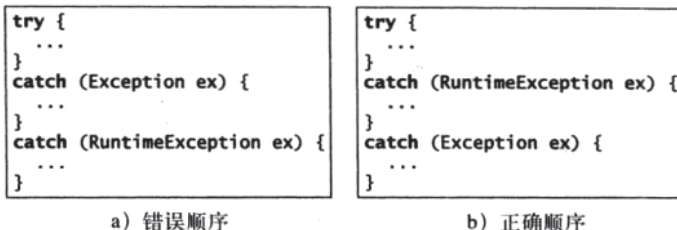


图13-3 如果异常没有在当前的方法中被捕获，就被传给该方法的调用者。

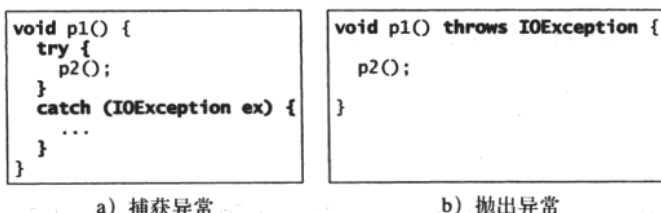
这个过程一直重复，直到异常被捕获或被传给main方法

注意 从一个通用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,图a中的顺序是错误的,因为RuntimeException是Exception的一个子类。正确的顺序应该如图b中所示。



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),就必须如图a和图b所示编写代码。



13.5.4 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13-4所示。printStackTrace()方法在控制台上输出栈跟踪信息。getStackTrace()方法提供方案,来访问由printStackTrace()打印输出的栈跟踪信息。

java.lang.Throwable	
+getMessage(): String	返回这个对象的消息
+toString(): String	返回以下三个字符串的连接: (1) 异常类的全名; (2) ":" (冒号或空格); (3) getMessage()方法
+printStackTrace(): void	在控制台上打印Throwable对象以及它的调用栈的跟踪信息
+getStackTrace(): StackTraceElement[]	返回栈跟踪元素构成的数组来表示这个可抛出的栈跟踪信息

图13-4 Throwable是所有异常类的根类

程序清单13-7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13-5所示。第10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12行)、类名(第13行)和异常行号(第14行)。

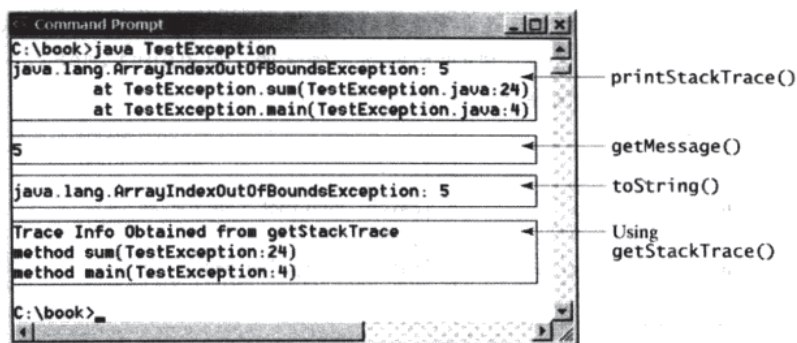


图13-5 可以使用printStackTrace()、getMessage()、toString()和getStackTrace()方法从异常对象获取信息

程序清单13-7 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 }

```

13.5.5 举例：声明、抛出和捕获异常

本例改写程序清单8-9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。

将程序清单13-8中给出的circle类重命名为CircleWithException，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Circle3类是一样的。

程序清单13-8 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13-9给出使用新Circle类的测试程序。

程序清单13-9 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方法声明了一个异

常并抛出它。

`setRadius`方法在方法头中声明为抛出`IllegalArgumentException`异常(`CircleWithException.java`中的第25~32行)。即使在方法声明中删除`throws IllegalArgumentException`子句, `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13-9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信息`ex.toString()`(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。

13.6 finally子句

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:

- 1) 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。
- 2) 如果`try`块中有一条语句会引起异常, 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。
- 3) 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`子句时可能会忽略`catch`块。

`finally`子句的一个常见用法是用于I/O程序设计中。为确保一个文件在所有情况下均被关闭, 可能要在`finally`块中放置一条文件关闭语句, 如程序清单13-10所示。

程序清单13-10 `FinallyDemo.java`

```
1 public class FinallyDemo {
2     public static void main(String[] args) {
3         java.io.PrintWriter output = null;
4
5         try {
6             // Create a file
7             output = new java.io.PrintWriter("text.txt");
```

```

8
9    // Write formatted output to the file
10   output.println("Welcome to Java");
11   }
12   catch (java.io.IOException ex) {
13       ex.printStackTrace();
14   }
15   finally {
16       // Close the file
17       if (output != null) output.close();
18   }
19
20   System.out.println("End of program");
21 }
22 }

```

第7行和第10行的语句可能会抛出异常IOException，因此，将它们放在try块中。在finally块中，output.close()语句关闭PrintWriter对象输出。不管try块中是否出现异常或者是否捕获异常，该语句都会执行。

13.7 何时使用异常

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，可以使程序更易读和更易修改。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，什么时候应该使用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，有时是很难判断的。但有一点要把握住，不要把异常处理用作简单的逻辑测试。

13.8 重新抛出异常

如果异常处理器没有处理某异常，或者处理器只是希望它的调用者注意到该异常，Java就允许异常处理器重新抛出该异常。这个语法如下所示：

```

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。

13.9 链式异常

在前一节中，`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13-11解释了如何产生和抛出链式异常。

程序清单13-11 **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` 方法中的第4行的 `catch` 块中被捕获。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。

13.10 创建自定义异常类

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13-8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，就必须创建自定义的异常类，如程序清单13-12所示。

程序清单13-12 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 }

```

这个自定义异常类扩展 `java.lang.Exception` (第1行)。而 `Exception` 类扩展 `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() +Exception(message: String)	构造一个无消息的异常 构造一个带有特定消息的异常

第6行调用父类的带有一条消息的构造方法。这条信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13-8中的 `setRadius` 方法必须修改如下:

```

/** Set a new radius */
public void setRadius(double newRadius)
    throws InvalidRadiusException {
    if (newRadius >= 0)
        radius = newRadius;
    else
        throw new InvalidRadiusException(newRadius);
}

```

下面代码创建一个圆对象, 并且将它的半径设置为-5。

```

try {
    CircleWithException1 c = new CircleWithException1(4);
    c.setRadius(-5);
}
catch (InvalidRadiusException ex) {
    System.out.println("The invalid radius is " + ex.getRadius());
}

```

调用 `setRadius(-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的。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。

关键术语

chained exception (链式异常)

checked exception (必检异常)

declare exception (声明异常)

exception (异常)

exception propagation (异常传播)

throw exception (抛出异常)

unchecked exception (免检异常)

本章小结

- 异常处理能够使一个方法给它的调用者抛出一个异常。
- Java异常是派生自`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。
- 异常发生在一个方法的执行过程中。`RuntimeException`和`Error`都是免检异常，其他所有的异常都是必检的。
- 当声明一个方法时，如果这个方法可能抛出一个必检异常，则必须声明为必检异常，告诉编译器可能会出现什么错误。
- 声明异常的关键字是`throws`，而抛出异常的关键字是`throw`。
- 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。
- 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。
- 可以从一个通用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。
- 在`catch`块中，异常被指定顺序是非常重要的。如果在一个类的父类的异常对象之前没有指定这个类的一个异常对象，就会导致一个编译错误。
- 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。如果方法想在退出前执行一些任务，可以在方法中捕获这个异常，然后再重新抛给真正的处理器。
- 任何情况下都会执行`finally`块中的代码，不管`try`块中是否出现或者捕获了异常。
- 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。
- 不应该使用异常处理代替简单的测试。应该尽可能地测试简单异常，将异常处理保留为处理那些无法用`if`语句处理的异常。

复习题

13.1~13.3节

13.1 描述Java的`Throwable`类、子类以及异常的类型。如果有异常的话，下面的程序会抛出什么`RuntimeException`异常？

```
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)

13.2 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)

13.3 指出下面代码的问题。这些代码会抛出什么异常吗？

```
long value = Long.MAX_VALUE + 1;
System.out.println(value);
```

13.4 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？

13.5 什么是必检异常？什么是免检异常？

13.6 如何抛出一个异常？可以在一个throw语句中抛出多个异常吗？

13.7 关键字throw的作用是什么？关键字throws的作用是什么？

13.8 发生异常后，Java虚拟机会做什么？如何捕获一个异常？

13.9 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");
    }
}
```

如果将int value=30; 这一行替换为int value=50;，输出结果会是什么？

13.10 假设下面的try-catch块中的statement2引起一个异常：

```
try {
    statement1;
    statement2;
    statement3;
}
catch (Exception1 ex1) {
```

新华书店
PDG

```

}
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?
- 如果异常被传递给调用者, 会执行statement4吗?

13.11 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");
        }
    }
}

```

13.12 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);
    }
}

```

13.13 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()");
    }
}
}

```

13.14 方法getMessage()可以做什么?

13.15 方法printStackTrace可以做什么?

13.16 没有异常发生时, try-catch块的存在会引起额外的系统开销吗?

13.17 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");
}

```

13.4~13.10节

13.18 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) {
}
catch (Exception3 ex3) {
    throw ex3;
}
finally {
    statement4;
};
statement5;

```

回答以下问题:

- 如果异常没有被捕获, 会执行语句statement5吗?
- 如果异常类型是Exception3, 那么会执行statement4吗? 会执行statement5吗?

13.19 假定setRadius方法抛出13.7节中定义的RadiusException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;
    }
}
}

```

13.20 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。

编程练习题

13.2~13.10节

*13.1 (NumberFormatException异常) 程序清单9-5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信息, 通知用户发生了操作数类型错误 (参见图13-6)。

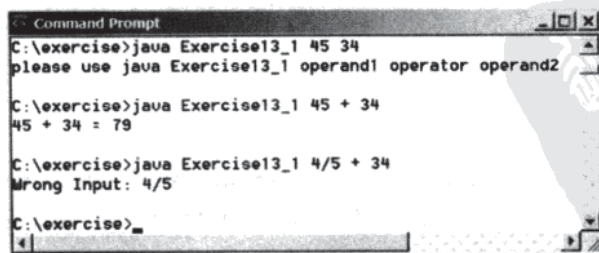


图13-6 程序完成算术运算并检查输入错误

*13.2 (NumberFormatException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。

*13.3 (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。

*13.4 (IllegalArgumentException异常) 修改程序清单10-2中的Loan类，如果贷款总额、利率或年数小于或等于零，则抛出IllegalArgumentException异常。

*13.5 (IllegalTriangleException异常) 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  
}
```

*13.6 (NumberFormatException异常) 程序清单9-2实现了hexToDecimal(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hexToDecimal方法，在字符串不是一个十六进制字符串时抛出NumberFormatException异常。

*13.7 (NumberFormatException异常) 练习题9.8指出binaryToDecimal(String binaryString)方法是将一个二进制字符串转换为一个十进制数。实现binaryToDecimal方法，在字符串不是一个二进制字符串时抛出NumberFormatException异常。

*13.8 (HexFormatException异常) 练习题13.6实现hexToDecimal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hexToDecimal方法，在字符串不是一个十六进制字符串时抛出HexFormatException异常。

*13.9 (BinaryFormatException异常) 练习题13.7实现binaryToDecimal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binaryToDecimal方法，在字符串不是一个二进制字符串时抛出BinaryFormatException异常。

*13.10 (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。



第14章

Introduction to Java Programming, 8E

抽象类和接口

学习目标

- 设计和使用抽象类 (14.2节)。
- 使用Calendar类和GregorianCalendar类处理日历 (14.3节)。
- 使用接口指定对象共有的行动 (14.4节)。
- 定义接口以及实现接口的类 (14.4节)。
- 使用Comparable接口定义自然顺序 (14.5节)。
- 使用ActionListener接口实现对象对动作事件的监听 (14.6节)。
- 使用Cloneable接口使对象成为可克隆的 (14.7节)。
- 探究抽象类和接口的相同点和不同点 (14.8节)。
- 使用包装类 (Byte、Short、Integer、Long、Float、Double、Character和Boolean) 创建基本类型值的对象 (14.9节)。
- 创建一个通用的排序方法 (14.10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 (14.11节)。
- 使用BigInteger和BigDecimal类计算任意精度的大数字 (14.12节)。
- 设计Rational类来定义Rational类型 (14.13节)。

14.1 引言

你已经学习了如何编写简单的程序来创建和显示GUI组件。你能编写代码以响应像点击一个按钮这样的用户动作吗？如图14-1所示，当点击一个按钮时，控制台上就会显示一条消息。

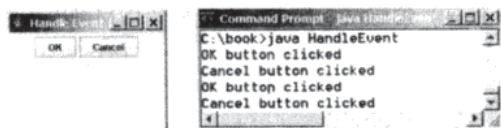


图14-1 程序响应点击按钮的行动事件

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（特别是不相关的类）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。

14.2 抽象类

在继承的层次结构中，随着每个新子类的出现，类会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类都包含分别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，因为它们的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14-2所示。程序清单14-1给出了新GeometricObject类的源代码。

程序清单14-1 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    /** Return a string representation of this object */
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 }

```

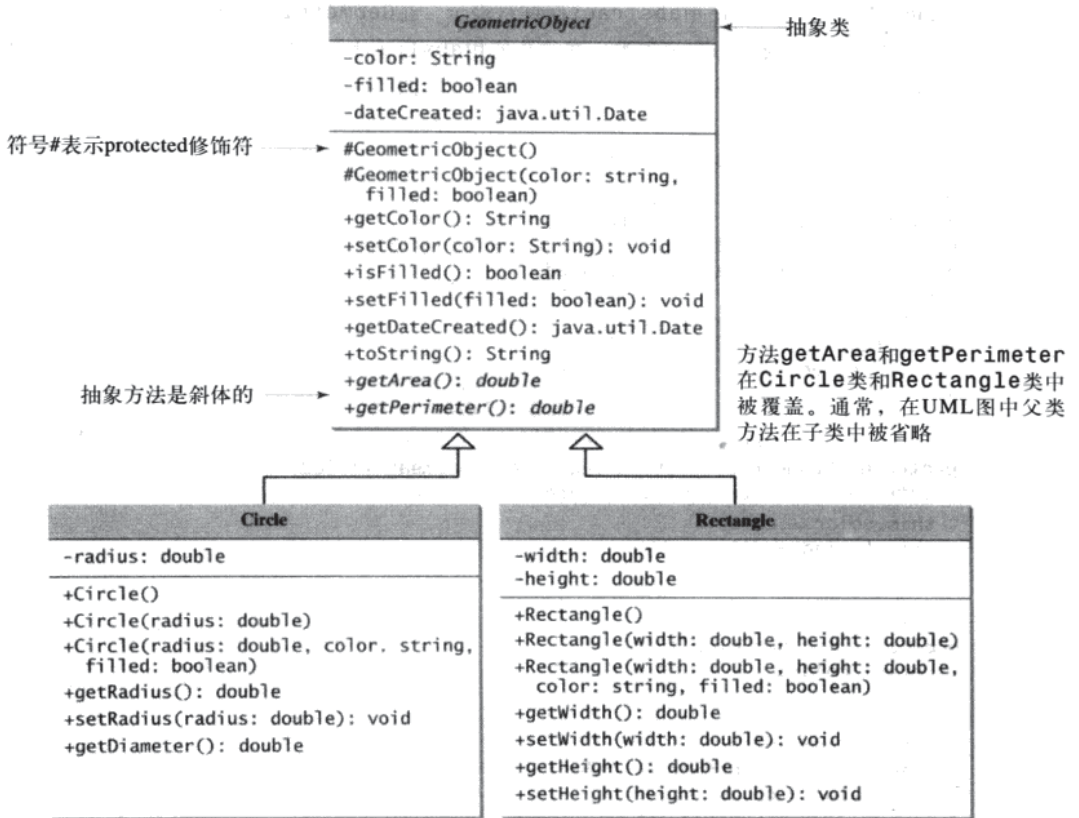


图14-2 新GeometricObject类包含抽象方法

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），并且提供了正确的构造方法。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，其他的都是同程序清单11-2和程序清单11-3一样的。

程序清单14-2 Circle.java

```

1 public class Circle extends GeometricObject {
2     // Same as lines 2-46 in Listing 11.2, so omitted
3 }
  
```

程序清单14-3 Rectangle.java

```

1 public class Rectangle extends GeometricObject {
2     // Same as lines 2-49 in Listing 11.3, so omitted
3 }
  
```

14.2.1 为什么要用抽象方法

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。下面的例子就能看出在GeometricObject中定义它们的好处。

程序清单14-4中的例子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14-4 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 14.0

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。

14.2.2 关于抽象类的几个关注点

下面是关于抽象类的值得注意的几点：

- 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。
- 包含抽象对象的类必须是抽象的。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。
- 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。
- 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不合法时是很有用的。在这种情况下，子类必须定义为abstract。
- 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，下面的语句是创建一个元素是GeometricObject类型的数组，这个语句是正确的：

```
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();
```

14.3 举例：日历类Calendar和公历类GregorianCalendar

一个java.util.Date的实例表示以毫秒为单位的特定时间段。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）、阴历和犹太历。目前，Java支持公历类java.util.GregorianCalendar，如图14-3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。

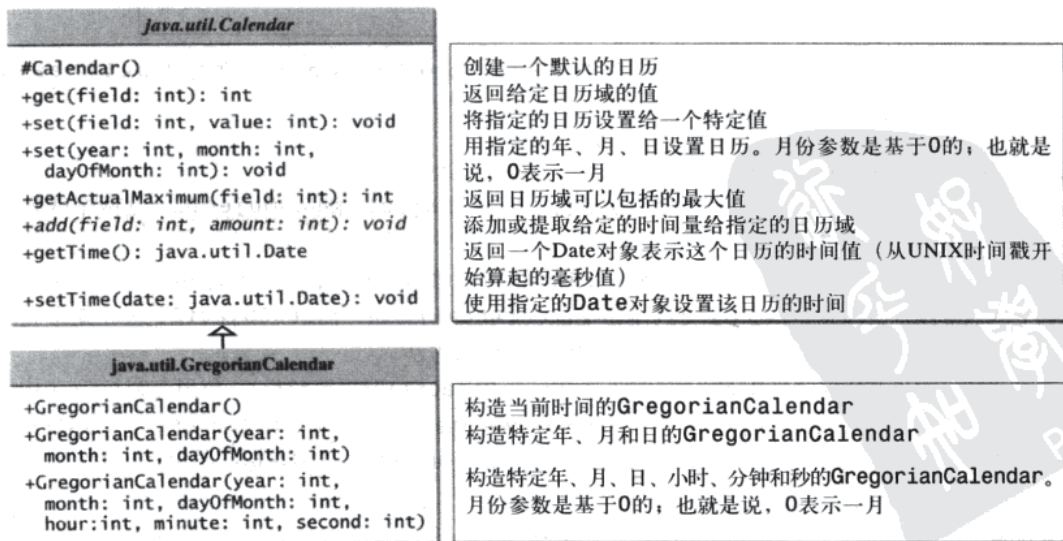


图14-3 抽象的Calendar类定义了各种日历的共同特点

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)` 利用指定的年 `year`、月 `month` 和日 `date` 构造一个 `GregorianCalendar` 对象。参数 `month` 是基于0的, 即0就是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14-1所示。

表14-1 `Calendar`类的域常量

常 量	说 明
<code>YEAR</code>	日历的年份
<code>MONTH</code>	日历的月份, 0表示一月
<code>DATE</code>	日历的天
<code>HOURL</code>	日历的小时 (12小时制)
<code>HOURL_OF_DAY</code>	日历的小时 (24小时制)
<code>MINUTE</code>	日历的分钟
<code>SECOND</code>	日历的秒
<code>DAY_OF_WEEK</code>	一周的天数, 1是星期日
<code>DAY_OF_MONTH</code>	和 <code>DATE</code> 一样
<code>DAY_OF_YEAR</code>	当前年的天数, 1是一年的第一天
<code>WEEK_OF_MONTH</code>	当前月内的星期数, 1是该月的第一个星期
<code>WEEK_OF_YEAR</code>	当前年内的星期数, 1是该年的第一个星期
<code>AM_PM</code>	表明是上午还是下午 (0表示上午, 1表示下午)

程序清单14-5给出的例子显示当前时间的日期和时间信息。

程序清单14-5 `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:\t" + calendar.get(Calendar.YEAR));
9         System.out.println("MONTH:\t" + calendar.get(Calendar.MONTH));
10        System.out.println("DATE:\t" + calendar.get(Calendar.DATE));
11        System.out.println("HOUR:\t" + calendar.get(Calendar.HOUR));
12        System.out.println("HOUR_OF_DAY:\t" +
13            calendar.get(Calendar.HOUR_OF_DAY));
14        System.out.println("MINUTE:\t" + calendar.get(Calendar.MINUTE));
15        System.out.println("SECOND:\t" + calendar.get(Calendar.SECOND));
16        System.out.println("DAY_OF_WEEK:\t" +
17            calendar.get(Calendar.DAY_OF_WEEK));
18        System.out.println("DAY_OF_MONTH:\t" +
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        System.out.println("September 11, 2001 is a " +
31            dayNameOfWeek(calendar1.get(Calendar.DAY_OF_WEEK)));
32    }

```

```

33
34 public static String dayNameOfWeek(int dayOfWeek) {
35     switch (dayOfWeek) {
36         case 1: return "Sunday";
37         case 2: return "Monday";
38         case 3: return "Tuesday";
39         case 4: return "Wednesday";
40         case 5: return "Thursday";
41         case 6: return "Friday";
42         case 7: return "Saturday";
43         default: return null;
44     }
45 }
46 }

```

Current time is Sun Sep 09 21:23:59 EDT 2007

YEAR: 2007
 MONTH: 8
 DATE: 9
 HOUR: 9
 HOUR_OF_DAY: 21
 MINUTE: 23
 SECOND: 59
 DAY_OF_WEEK: 1
 DAY_OF_MONTH: 9
 DAY_OF_YEAR: 252
 WEEK_OF_MONTH: 3
 WEEK_OF_YEAR: 37
 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。

14.4 接口

接口(interface)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、可食用的或可克隆的。

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:

```

public interface Edible {
    /** Describe how to eat */
    public abstract String howToEat();
}

```

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。与抽象类相似，不能使用new操作符创建接口的实例，但是大多数情况下，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。

现在，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14-6中的Chicken类和Fruit类（第14、23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14-6 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 }
9
10 class Animal {
11     // Data fields, constructors, and methods omitted here
12 }
13
14 class Chicken extends Animal implements Edible {
15     public String howToEat() {
16         return "Chicken: Fry it";
17     }
18 }
19
20 class Tiger extends Animal {
21 }
22
23 abstract class Fruit implements Edible {
24     // Data fields, constructors, and methods omitted here
25 }
26
27 class Apple extends Fruit {
28     public String howToEat() {
29         return "Apple: Make apple cider";
30     }
31 }
32
33 class Orange extends Fruit {
34     public String howToEat() {
35         return "Orange: Make orange juice";
36     }
37 }

```

```

Chicken: Fry it
Apple: Make apple cider

```



Chicken类扩展自Animal类，并实现Edible以表明小鸡是可食用的。当一个类实现接口时，该类实现了定义在接口中的所有带确切签名和返回类型的方法。Chicken类实现了howToEat方法（第15~17行）。

Fruit类实现Edible。因为它不实现howToEat方法，所以Fruit必须表示为abstract（第23行）。Fruit的具体子类必须实现howToEat方法。Apple类和Orange类实现howToEat方法（第28、34行）。

main方法创建由Tiger、Chicken和Apple的三个对象构成的数组（第3行），如果这个元素是可食用的，调用howToEat方法（第6行）。

注意 由于接口中所有的数据域都是`public final static`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:

```
public interface T {
    public static final int K = 1;

    public abstract void p();
}
```

等价于

```
public interface T {
    int K = 1;

    void p();
}
```

提示 接口内定义的常量可以使用语法“接口名.常量名”(例如, `T.K`) 来访问。

14.5 举例: Comparable接口

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, 这两个对象都该有共同的方法就是`comparable` (可比较的)。Java为此目的提供了`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 {
    public int compareTo(Object 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。

Java类库中的许多类 (例如, `String`和`Date`) 实现了`Comparable`接口以定义对象的自然顺序。如果你检查这些类的源代码, 就会发现这些类中使用的关键字`implements`, 如下所示:

```
public class String extends Object
    implements Comparable {
    // class body omitted
}
```

```
public class Date extends Object
    implements Comparable {
    // class body omitted
}
```

这样, 字符串就是可比较的, 日期也是如此。假设`s`为`String`对象, `d`为`Date`对象, 那么下面所有的表达式都为`true`:

```
s instanceof String
s instanceof Object
s instanceof Comparable
```

```
d instanceof java.util.Date
d instanceof Object
d instanceof Comparable
```

从两个对象中找出最大者的通用方法`max`可以定义为如图a或图b所示:

```
// Max.java: Find a maximum object
public class Max {
    /** Return the maximum of two objects */
    public static Comparable max
        (Comparable o1, Comparable o2) {
        if (o1.compareTo(o2) > 0)
            return o1;
        else
            return o2;
        }
}
```

a)

```
// Max.java: Find a maximum object
public class Max {
    /** Return the maximum of two objects */
    public static Object max
        (Object o1, Object o2) {
        if (((Comparable)o1).compareTo(o2) > 0)
            return o1;
        else
            return o2;
        }
}
```

b)

图a中的`max`方法比图b中的简单。在图b中的`Max`类中, `o1`被声明为`Object`, 而 `(Comparable) o1`告诉编译器将`o1`转换成`Comparable`, 因此, 可以从`o1`中调用`compareTo`方法。但是, 图a中的`Max`类无须转换, 因为`o1`被声明为`Comparable`。

图a中的max方法比图b中的更鲁棒。必须用两个可比较的对象调用max方法。假设用两个不可比较的对象调用方法max:

```
Max.max(anyObject1,anyObject2);
```

因为anyObject1不是Comparable的实例,所以编译器会发现使用图a中max方法时的错误。使用图b中的max方法,该行代码将顺利编译,但在运行时会出现ClassCastException异常,因为anyObject1不是Comparable的实例,并且不能转换成Comparable。

从现在起,假设使用的是图a中的max方法。因为字符串和日期都是可比较的,所以,可以使用max方法找出String类或Date类的两个实例中的较大者。考虑下面的例子:

```
String s1 = "abcdef";
String s2 = "abcdee";
String s3 = (String)Max.max(s1, s2);
```

```
Date d1 = new Date();
Date d2 = new Date();
Date d3 = (Date)Max.max(d1, d2);
```

从max方法返回的值是Comparable类型。所以,需要将它显式地转换为String或Date类型。

不能使用max方法得到两个Rectangle实例中较大的一个,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14-7所示。

程序清单14-7 ComparableRectangle.java

```
1 public class ComparableRectangle extends Rectangle
2     implements Comparable {
3     /** Construct a ComparableRectangle with specified properties */
4     public ComparableRectangle(double width, double height) {
5         super(width, height);
6     }
7
8     /** Implement the compareTo method defined in Comparable */
9     public int compareTo(Object o) {
10        if (getArea() > ((ComparableRectangle)o).getArea())
11            return 1;
12        else if (getArea() < ((ComparableRectangle)o).getArea())
13            return -1;
14        else
15            return 0;
16    }
17 }
```

ComparableRectangle类扩展自Rectangle类并实现Comparable方法,如图14-4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。

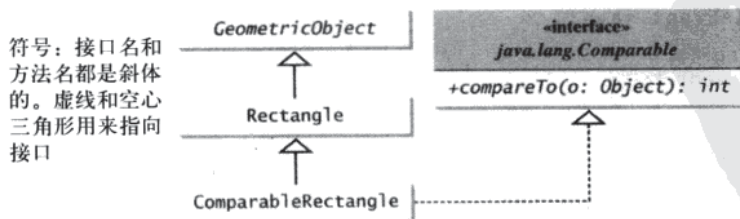


图14-4 ComparableRectangle类扩展Rectangle类并实现Comparable接口

现在,可以使用max方法找出两个ComparableRectangle对象中较大的一个。请看下面的例子:

```
ComparableRectangle rectangle1 = new ComparableRectangle(4, 5);
ComparableRectangle rectangle2 = new ComparableRectangle(3, 6);
System.out.println(Max.max(rectangle1, rectangle2));
```

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max方法去求对象的最大值，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，那么新的max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。

14.6 举例：ActionListener接口

现在，你已经准备好编写一个小程序解决本章前言中提出的问题。程序在框架中显示两个按钮，如图14-1所示。为了响应对一个按钮的点击，需要编写代码来处理点击按钮动作。按钮就是动作来源的源对象（source object）。需要创建一个对象能够处理按钮上的动作事件。这个对象称为监听器（listener），如图14-5所示。

不是所有的对象都能成为动作事件的监听器。一个对象要成为源对象上动作事件的监听器，需要满足两个条件：

- 这个对象必须是ActionListener（事件监听器）接口的一个实例。该接口定义了所有动作监听器共有的动作。
- ActionListener对象listener必须使用方法source.addActionListener(listener)注册给源对象。

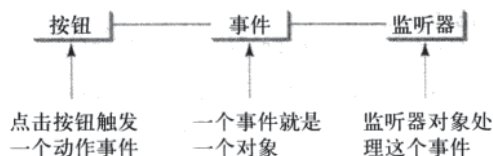


图14-5 监听器对象处理源对象触发的事件

ActionListener接口包含处理事件的actionPerformed方法。监听器必须覆盖该方法来响应事件。程序清单14-8给出在两个按钮上处理ActionEvent的代码。点击OK按钮时，就会显示消息“OK button clicked”（OK按钮被点击）。点击Cancel按钮时，就会显示消息“Cancel button clicked”（Cancel按钮被点击），如图14-1所示。

程序清单14-8 HandleEvent.java

```

1 import javax.swing.*;
2 import java.awt.event.*;
3
4 public class HandleEvent extends JFrame {
5     public HandleEvent() {
6         // Create two buttons
7         JButton jbtOK = new JButton("OK");
8         JButton jbtCancel = new JButton("Cancel");
9
10        // Create a panel to hold buttons
11        JPanel panel = new JPanel();
12        panel.add(jbtOK);
13        panel.add(jbtCancel);
14
15        add(panel); // Add panel to the frame
16    }
17 }

```

PDF

```

17 // Register listeners
18 OKListenerClass listener1 = new OKListenerClass();
19 CancelListenerClass listener2 = new CancelListenerClass();
20 jbtOK.addActionListener(listener1);
21 jbtCancel.addActionListener(listener2);
22 }
23
24 public static void main(String[] args) {
25     JFrame frame = new HandleEvent();
26     frame.setTitle("Handle Event");
27     frame.setSize(200, 150);
28     frame.setLocation(200, 100);
29     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
30     frame.setVisible(true);
31 }
32 }
33
34 class OKListenerClass implements ActionListener {
35     public void actionPerformed(ActionEvent e) {
36         System.out.println("OK button clicked");
37     }
38 }
39
40 class CancelListenerClass implements ActionListener {
41     public void actionPerformed(ActionEvent e) {
42         System.out.println("Cancel button clicked");
43     }
44 }

```

两个监听器对象在第34~44行定义。每个监听器类实现ActionListener来处理ActionEvent。对象listener1是OKListenerClass的一个实例（第18行），它注册给按钮jbtOK（第20行）。当点击OK按钮时，OKListenerClass中的actionPerformed(ActionEvent)方法（第36行）会被调用来处理该事件。对象listener2是CancelListenerClass的一个实例（第19行），将它在第21行注册给按钮jbtCancel。当点击Cancel按钮时，CancelListenerClass中的actionPerformed(ActionEvent)方法（第42行）会被调用来处理该事件。处理事件的方式将在第16章中进一步讨论。

14.7 举例：Cloneable接口

经常会出现需要创建一个对象拷贝的情况。需要使用clone方法并理解Cloneable接口，这些都是本节的主题。

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(2003, 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。

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; list1[1] = 8;
4
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, 8
list2 is 1, 2

```

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14-9定义一个实现Cloneable和Comparable的名为House的类。

程序清单14-9 House.java

```

1 public class House implements Cloneable, Comparable {
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 the protected clone method defined in the Object
25     class, and strengthen its accessibility */
26    public Object clone() throws CloneNotSupportedException {
27        return super.clone();
28    }
29
30    /** Implement the compareTo method defined in Comparable */
31    public int compareTo(Object o) {
32        if (area > ((House)o).area)
33            return 1;
34        else if (area < ((House)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视性修饰符改为public，这样，该方法就可以在任何包中使用。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14-6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。

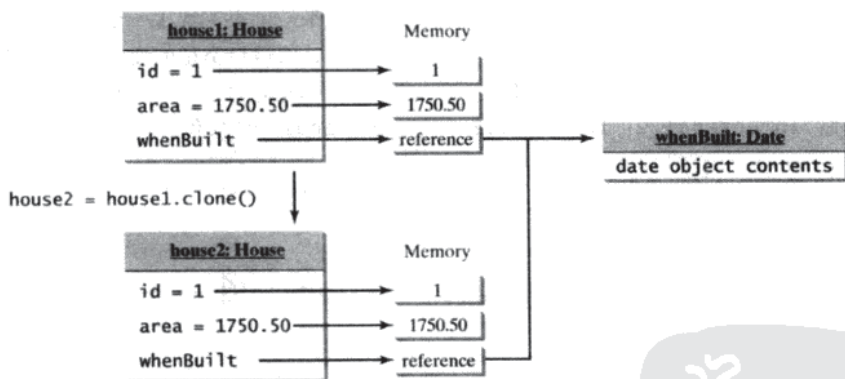


图14-6 clone方法复制基本类型域的值以及对象类型域的引用

如果希望完成深复制，可以在调用super.clone()之后用自定义的克隆操作来覆盖clone方法。参见练习题14.4。

警告 如果House没有覆盖clone()方法，程序就会收到一个语法错误，因为在java.lang.Object中clone()是被保护的。如果House不实现java.lang.Cloneable，调用House.java中的super.clone()会导致一个CloneNotSupportedException异常。这样，为了克隆一个对象，该对象的类必须覆盖clone()方法并实现Cloneable。

14.8 接口与抽象类

或多或少可以使用和抽象类一样的方式来使用接口，但是，定义一个接口与定义一个抽象类有所不同。表14-2总结出这些不同点。

表14-2 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new操作符实例化	无限制
接口	所有的变量必须是 public static final	没有构造方法。接口不能用new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
}
```

一个类实现NewInterface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14-7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。

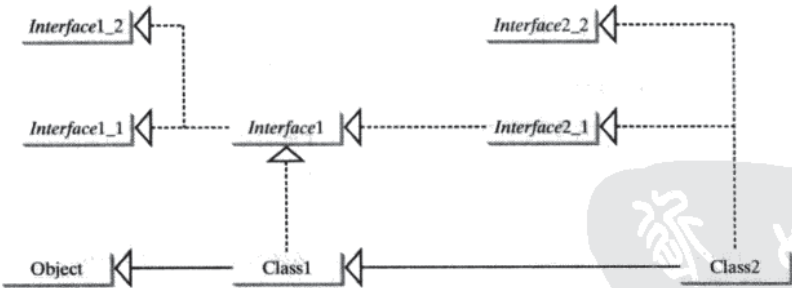


图14-7 Class1实现接口Interface1，Interface1扩展接口Interface1_1和Interface1_2。Class2扩展Class1并实现接口Interface2_1和Interface2_2

注意 类名是一个名词。接口名可以是形容词或名词。例如，java.lang.Comparable和java.awt.event.ActionListener都是接口。Comparable是一个形容词，而ActionListener是一个名词。

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，详细描述父子关系的强是关系（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。弱是关系 (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。弱是关系可以用接口来建模。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是因为接口可以定义不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 {
    public String howToEat() {
        return "Fry it";
    }
}
```

```
class Duck extends Animal {
    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。接口比类带来更多的灵活性, 因为不用使每件东西都适用于同一类型的类。可以定义接口中的 `howToEat()` 方法, 然后把它当做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 {
    public String howToEat() {
        return "Fry it";
    }
}

class Duck implements Edible {
    public String howToEat() {
        return "Roast it";
    }
}

class Broccoli implements Edible {
    public String howToEat() {
        return "Stir-fry it";
    }
}

```

为了定义表示可食用对象的一个类，只需让该类实现**Edible**接口即可。现在，这个类就成为**Edible**类型的子类型。任何**Edible**对象都可以被传递来调用**eat**方法。

14.9 将基本数据类型值作为对象处理

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做对象，就会给语言性能带来负面影响。然而，许多Java中的方法需要将对象作为参数。例如，在**ArrayList**类中的**add(object)**方法向**ArrayList**中添加一个对象。Java提供了一个方便的办法，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）。对应的类称为包装类（wrapper class）。通过使用包装对象而不是基本数据类型变量，就可以利用通用程序设计。

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。它们的继承层次体系结构如图14-8所示。

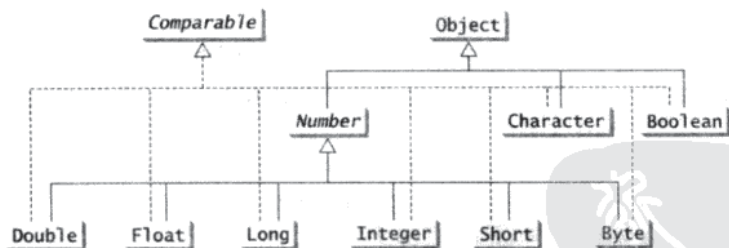


图14-8 **Number**类是**Double**、**Float**、**Long**、**Integer**、**Short**和**Byte**类的抽象父类

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，但第一个字母要大写。**Integer**和**Character**例外。

每一个数值包装类都是从抽象类**Number**扩展而来的，类**Number**包含**doubleValue()**、**floatValue()**、**intValue()**、**longValue()**、**shortValue()**和**byteValue()**方法。这些方法将对象“转换”为基本类型值。每一个包装类覆盖了在**Object**类中定义的**toString**和**equals**方法。因为所有的包装类都实现**Comparable**接口，所以这些类中都会实现**compareTo**方法。

包装类相互之间都非常相似。在第9章中已经介绍了**Character**类。**Boolean**类包装了布尔值**true**

或false。本节使用Integer和Double类为例介绍数值包装类。Integer类和Double类的主要特征如图14-9所示。

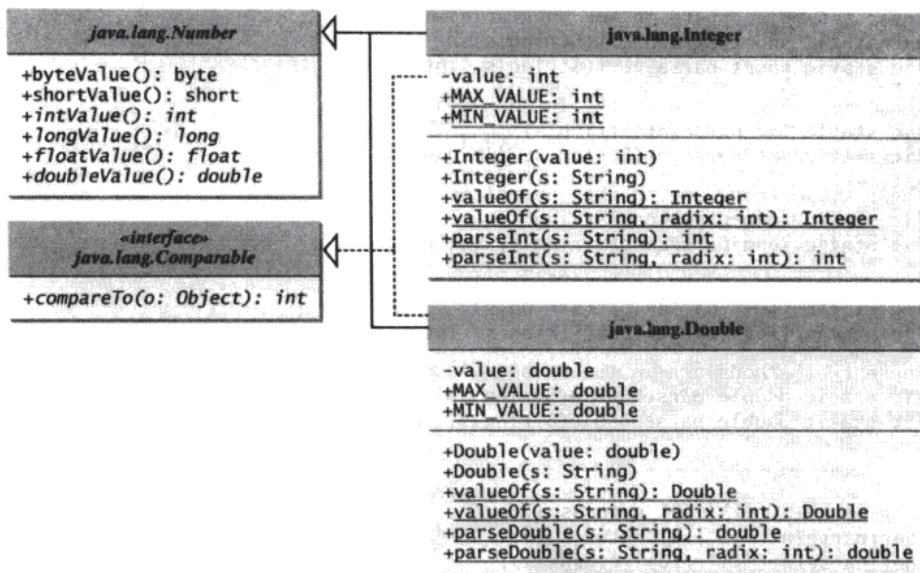


图14-9 包装类提供构造方法、常量和处理各种数据类型的转换方法

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")`。

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，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实现在Number类中定义的抽象方法doubleValue()、floatValue()、intValue()、longValue()和shortValue()。这些方法返回包装对象的double、float、int、long或short值。

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");

```

你已经使用过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。这些方法如下所示：

```
// 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。

14.10 举例: 对一个对象数组排序

这个例子给出一个静态通用方法, 对一个可比较的对象数组进行排序。这些对象都是`Comparable`接口的实例, 并且可以使用`compareTo`方法进行比较。对于任何对象, 只要其所在的类都实现了`Comparable`接口, 这个方法就能对这些对象构成的数组进行排序。

为了测试这个方法, 程序对整数数组、双精度数数组、字符数组和字符串数组进行排序。程序如程序清单14-10所示。

程序清单14-10 `GenericSort.java`

```
1 public class GenericSort {
2     public static void main(String[] args) {
3         // Create an Integer array
4         Integer[] intArray = {new Integer(2), new Integer(4),
5             new Integer(3)};
6
7         // Create a Double array
8         Double[] doubleArray = {new Double(3.4), new Double(1.3),
9             new Double(-22.1)};
10
11        // Create a Character array
12        Character[] charArray = {new Character('a'),
13            new Character('J'), new Character('r')};
14
15        // Create a String array
16        String[] stringArray = {"Tom", "John", "Fred"};
17
18        // Sort the arrays
19        sort(intArray);
20        sort(doubleArray);
21        sort(charArray);
22        sort(stringArray);
23    }
```

```

24 // Display the sorted arrays
25 System.out.print("Sorted Integer objects: ");
26 printList(intArray);
27 System.out.print("Sorted Double objects: ");
28 printList(doubleArray);
29 System.out.print("Sorted Character objects: ");
30 printList(charArray);
31 System.out.print("Sorted String objects: ");
32 printList(stringArray);
33 }
34
35 /** Sort an array of comparable objects */
36 public static void sort(Comparable[] list) {
37     Comparable currentMin;
38     int currentMinIndex;
39
40     for (int i = 0; i < list.length - 1; i++) {
41         // Find the maximum in the list[0..i]
42         currentMin = list[i];
43         currentMinIndex = i;
44
45         for (int j = i + 1; j < list.length; j++) {
46             if (currentMin.compareTo(list[j]) > 0) {
47                 currentMin = list[j];
48                 currentMinIndex = j;
49             }
50         }
51
52         // Swap list[i] with list[currentMinIndex] if necessary;
53         if (currentMinIndex != i) {
54             list[currentMinIndex] = list[i];
55             list[i] = currentMin;
56         }
57     }
58 }
59
60 /** Print an array of objects */
61 public static void printList(Object[] list) {
62     for (int i = 0; i < list.length; i++)
63         System.out.print(list[i] + " ");
64     System.out.println();
65 }
66 }

```

```

Sorted Integer objects: 2 3 4
Sorted Double objects: -22.1 1.3 3.4
Sorted Character objects: J a r
Sorted String objects: Fred John Tom

```



sort方法的算法和6.10.1节中的算法是一样的。在6.10.1节中的sort方法对double值构成的数组进行排序。假定这些对象也是Comparable接口的实例，那么这个例子中的sort方法可以对任何对象类型的数组排序。这是通用程序设计（generic programming）的另一个例子。因为通用程序设计使一个方法可以对通用类型参数进行操作，所以这种方法可以在多种类型上重复使用。

Integer、Double、Character和String都实现了Comparable接口，所以这些类的对象都可以使用compareTo方法进行比较。sort方法使用compareTo方法判断对象在数组中的顺序。

提示 在java.util.Array类中，Java提供一个对任意对象类型的数组进行排序的静态方法sort，假定数组中的元素都是可比较的。这样，就能使用下面的代码对这个例子中的数组进行排序：


```
java.util.Arrays.sort(intArray);
java.util.Arrays.sort(doubleArray);
java.util.Arrays.sort(charArray);
java.util.Arrays.sort(stringArray);
```

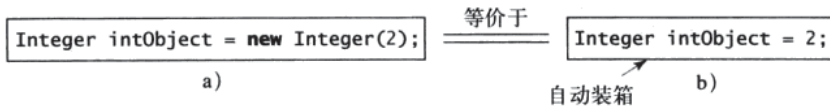
注意 数组是对象，一个数组是Object类的一个实例。此外，如果A是B的子类型，那么A[]的每一个实例都是B[]的实例。所以，下面语句的值都是true：

```
new int[10] instanceof Object
new Integer[10] instanceof Object
new Integer[10] instanceof Comparable[]
new Integer[10] instanceof Number[]
new Number[10] instanceof Object[]
```

警告 尽管int型的值可以赋给double型的变量，但是int[]和double[]是两个不兼容的类型。所以，不能把一个int[]数组赋值给double[]型变量或Object[]型变量。

14.11 基本类型和包装类类型之间的自动转换

Java允许基本类型和包装类类型之间进行自动转换。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：



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)。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，编译器将对象进行自动开箱。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。

14.12 BigInteger和BigDecimal类

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。它们都扩展Number类且实现Comparable接口。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、multiple、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14-11给出可以返回任意整数阶乘的方法。

程序清单14-11 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。

调用`multiply`方法 (第11行) 获得一个新结果。

14.13 实例学习: Rational类

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中, 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14-10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, $1/3$ 称为最低形式。

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4-8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低形式。

通常, 首先编写一个测试程序来创建两个`Rational`对象, 然后测试它的方法。程序清单14-12是一个测试程序。

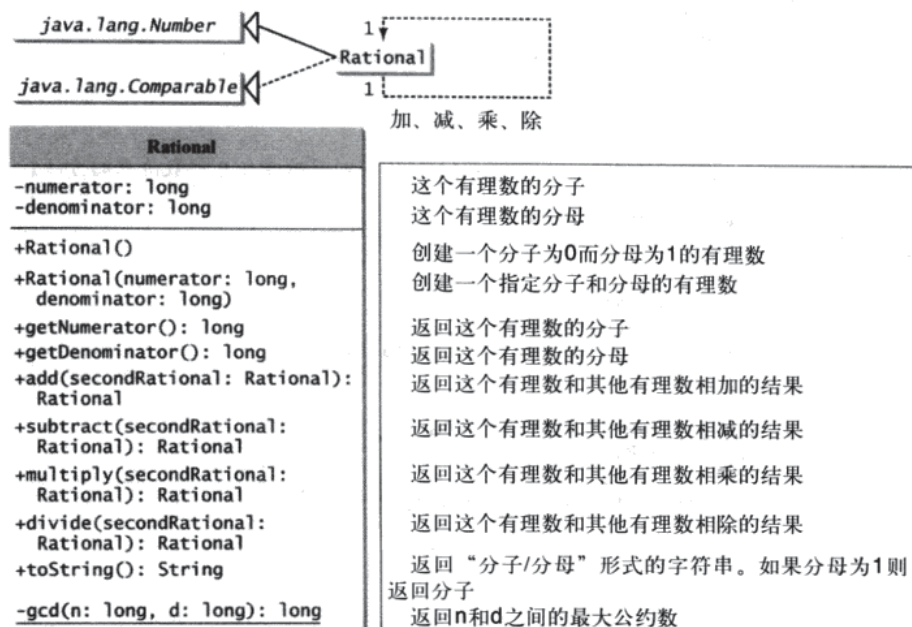


图14-10 Rational类的属性、构造方法和方法在UML中的图解

程序清单14-12 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方法创建两个有理数：r1和r2（第5~6行），然后显示 $r1+r2$ 、 $r1-r2$ 、 $r1 \times r2$ 和 $r1/r2$ 的结果（第9~12行）。为了完成 $r1+r2$ ，调用`r1.add(r2)`返回一个新的Rational对象。同样地，`r1.subtract(r2)`用以完成 $r1-r2$ ，`r1.multiply(r2)`用以完成 $r1 \times r2$ ，而`r1.divide(r2)`用以完成 $r1/r2$ 。

`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14-13中实现。

程序清单14-13 Rational.java

```

1 public class Rational extends Number implements Comparable {
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 to 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 from 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 the toString() method */
73 public String toString() {
74     if (denominator == 1)
75         return numerator + "";
76     else
77         return numerator + "/" + denominator;
78 }
79
80 /** Override the equals method in the Object class */
81 public boolean equals(Object parm1) {
82     if ((this.subtract((Rational)(parm1))).getNumerator() == 0)
83         return true;
84     else
85         return false;
86 }
87
88 /** Implement the abstract intValue method in java.lang.Number */
89 public int intValue() {
90     return (int)doubleValue();
91 }
92
93 /** Implement the abstract floatValue method in java.lang.Number */
94 public float floatValue() {
95     return (float)doubleValue();
96 }
97
98 /** Implement the doubleValue method in java.lang.Number */
99 public double doubleValue() {
100     return numerator * 1.0 / denominator;
101 }
102
103 /** Implement the abstract longValue method in java.lang.Number */
104 public long longValue() {
105     return (long)doubleValue();
106 }
107
108 /** Implement the compareTo method in java.lang.Comparable */
109 public int compareTo(Object o) {
110     if ((this.subtract((Rational)o)).getNumerator() > 0)
111         return 1;
112     else if ((this.subtract((Rational)o)).getNumerator() < 0)
113         return -1;
114     else
115         return 0;
116 }
117 }

```

有理数封装在Rational对象中。在机器内部，一个有理数表示为它的最低形式（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）。

方法gcd()（Rational类中的第19~30行）是私有的，它是不能被用户使用的。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。

方法abs(x)（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73~91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同，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类中实现（第88~106行）。这些方法为有理数返回int、float和double值。

Comparable接口中的compareTo(Object other)方法在Rational类中实现（第109~116行），对这个有理数和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。参见练习题14.18。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。这是一个解释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类有很严格的限定。这很容易溢出。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
```



为了解决这个问题，可以使用BigInteger表示分子和分母实现Rational类（参见练习题14.19）。

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）

multiple inheritance（多重继承）

subinterface（子接口）

shallow copy（浅复制）

single inheritance（单一继承）

wrapper class（包装类）

本章小结

- 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。
- 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，就必须将子类也定义为抽象类。
- 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。
- 即使父类是具体的，子类也可以是抽象的。

- 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。
- 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。
- 接口 `java.lang.Comparable` 定义了 `compareTo` 方法。Java类库中的许多类都实现了 `Comparable`。
- 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。
- 一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。
- 一个接口可以扩展一个或多个接口。
- 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）。对应的类称作包装类。使用包装对象而不是基本数据类型的变量，将有助于通用程序设计。
- 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。
- `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。

复习题

14.2节

14.1 在下面类的定义中，哪个定义了一个合法的抽象类？

```
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)

14.2 `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？

14.3 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。

14.4~14.6节

14.4 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)

14.5 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。

14.6 在14.5节中定义了两个max方法。解释一下，为什么签名为max(Comparable, Comparable)的max优于签名为max(Object, Object)的max。

14.7 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？

14.8 下面说法为真还是为假？如果一个类实现java.awt.event.ActionListener，那么类的对象可以调用actionPerformed方法。

14.7~14.8节

14.9 如果一个对象的类没有实现java.lang.Cloneable，那么可以调用clone()方法来克隆这个对象吗？Date类实现Cloneable吗？

14.10 如果House类（在程序清单14-9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？

14.11 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));
```

14.12 给出下面代码的输出结果：

```
java.util.ArrayList list = new java.util.ArrayList();
list.add("New York");
java.util.ArrayList list1 = list;
java.util.ArrayList list2 = (java.util.ArrayList)(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());
```

14.13 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);
    }
}
```

14.14 给出一个例子显示接口比抽象类有优势。

14.9节

14.15 描述基本类型包装类。为什么需要这些包装类？

14.16 下面的每条语句都能编译吗？

```
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();
String s = (Double.valueOf("23.4")).toString();
```

14.17 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数转换为一个字符串？如何将一个数值字符串转换为一个double数？



398 • 第14章 抽象类和接口

14.18 为什么下面两行代码可以编译，但会导致运行时错误？

```
Number numberRef = new Integer(0);
Double doubleRef = (Double)numberRef;
```

14.19 为什么下面两行代码可以编译，但会导致运行时错误？

```
Number[] numberArray = new Integer[2];
numberArray[0] = new Double(1.5);
```

14.20 下面的代码有什么错误？

```
public class Test {
    public static void main(String[] args) {
        Number x = new Integer(3);
        System.out.println(x.intValue());
        System.out.println(x.compareTo(new Integer(4)));
    }
}
```

14.21 下面的代码有什么错误？

```
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)));
    }
}
```

14.22 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));
    }
}
```

14.10~14.12节

14.23 什么是自动装箱和自动开箱？下面的语句正确吗？

```
Number x = 3;
Integer x = 3;
Double x = 3;
Double x = 3.0;
int x = new Integer(3);
int x = new Integer(3) + new Integer(4);
double y = 3.4;
y.intValue();
```

```
JOptionPane.showMessageDialog(null, 45.5);
```

14.24 可以将new int[10]、new String[100]、new Object[50]或者new Calendar[20]赋值给一个Object[]类型的变量吗？

14.25 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");
        x.add(y);
        System.out.println(x);
    }
}
```

数字图书馆
PDG

综合题

14.26 定义下面的术语：抽象类、接口。抽象类和接口的相同之处和不同之处是什么？

14.27 判断下面语句的对和错：

- (1) 抽象类可以有使用该抽象类的构造方法创建的实例。
- (2) 抽象类可以扩展。
- (3) 接口被编译为独立的字节码文件。
- (4) 非抽象父类的子类不能是抽象类。
- (5) 子类不能覆盖父类中的具体方法将其声明成抽象方法。
- (6) 抽象方法必须是非静态的。
- (7) 接口可以有静态方法。
- (8) 接口可以扩展一个或多个接口。
- (9) 接口可以扩展抽象类。
- (10) 抽象类可以扩展接口。

编程练习题

14.1~14.7节

*14.1 (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。

*14.2 (ComparableCircle类) 创建名为ComparableCircle的类, 它扩展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。

*14.3 (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。设计一个扩展GeometricObject类并实现Colorable接口的名为Square的类。实现howToColor方法, 显示消息"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。

*14.4 (修改House类) 改写程序清单14-9中的House类, 对数据域whenBuilt进行深复制。

*14.5 (将Circle类变成可比较的) 改写程序清单14-2中的Circle类, 它扩展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。

*14.6 (将Rectangle类变成可比较的) 改写程序清单14-3的Rectangle类, 它扩展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。

*14.7 (八边形Octagon类) 编写一个名为Octagon的类, 它扩展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。

*14.8 (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。

*14.9 (找出最大的对象) 编写一个方法, 返回对象数组中最大的对象。方法签名如下:

```
public static Object max(Comparable[] a)
```

所有对象都是Comparable接口的实例。对象在数组中的顺序是由compareTo方法决定的。

编写测试程序, 创建一个由10个字符串构成的数组、一个由10个整数构成的数组和一个由10个日期构成的数组, 找出数组中最大的字符串、整数和日期。

**14.10 (显示日历) 改写程序清单5-12中的PrintCalendar类, 使用Calendar类和GregorianCalendar类显示指定月份的日历。程序从命令行接收月份和年份。例如,

```
java Exercise14_10 1 2010
```

这将显示如图14-11所示的日历。

也可以不输入年份运行这个程序。在这种情况下, 年份为当前年。如果运行程序时没有指定月份和年份, 月份就是当前月。

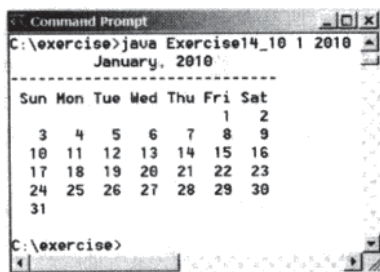


图14-11 程序显示2010年1月的日历

14.12节

**14.11 (被5或6整除) 找出能被5或6整除的前10个数字 (大于long.MAX_VALUE)。

**14.12 (被2或3整除) 找出能被2或3整除的前10个数字, 这些数字都有十进制数50。

**14.13 (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。

**14.14 (大素数) 编写程序找出五个大于long.MAX_VALUE的素数。

**14.15 (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。程序可能得花几个小时来完成。)

p	$2^p - 1$
2	3
3	7
5	31
...	

14.13节

**14.16 (近似 e) 练习题4.26使用下面数列近似计算 e :

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdots + \frac{1}{i!}$$

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。

14.17 (使用Rational类) 编写程序, 使用Rational类计算下面的和数列:

$$\frac{1}{2} + \frac{2}{3} + \frac{3}{4} + \cdots + \frac{98}{99} + \frac{99}{100}$$

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, 参见练习题14.19。

*14.18 (演示封装的好处) 使用新的分子分母的内部表达改写14.13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。

**14.19 (在Rational类中使用BigInteger) 使用BigInteger表示分子和分母, 重新设计和实现14.13节的Rational类。

*14.20 (创建一个有理数的计算器) 编写一个类似于程序清单9-5的程序。这里不使用整数, 而是使用有理数, 如图14-12所示。

需要使用在9.2.6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。

*14.21 (数学方面: 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 -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:

$$|a+bi|=\sqrt{a^2+b^2}$$

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 。

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 + -15.75i
3.5 + 5.5i / -3.5 + 1.0i = -0.5094 + -1.7i
|3.5 + 5.5i| = 6.519202405202649
```

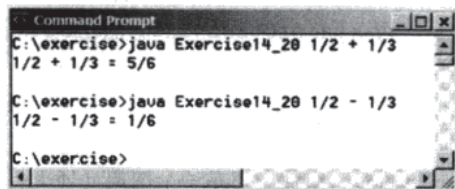


图14-12 程序从命令行读取三个参数 (操作数1、运算符、操作数2), 然后显示这个表达式以及这个算术运算的结果



PDG

第15章

Introduction to Java Programming, 8E

图 形

学习目标

- 描述GUI组件中的Java坐标系 (15.2节)。
- 使用Graphics类中的方法画图 (15.3节)。
- 覆盖paintComponent方法在GUI组件上绘画 (15.3节)。
- 使用面板作画布来绘画 (15.3节)。
- 绘制字符串、直线、矩形、椭圆、弧形和多边形等 (15.4、15.6~15.7节)。
- 使用FontMetrics获取字体属性并且了解如何将消息居中 (15.8节)。
- 在GUI组件中显示一个图像 (15.11节)。
- 开发可重用的GUI组件FigurePanel、MessagePanel、StillClock和ImageViewer (15.5、15.9、15.10、15.12节)

15.1 引言

假设希望画出如图15-1所示的像条形图、时钟或停止符号这样的图形，如何才能做到呢？

本章描述如何使用Graphics类中的方法绘制字符串、直线、矩形、椭圆、弧形、多边形和图像，以及如何开发可重用的GUI组件。

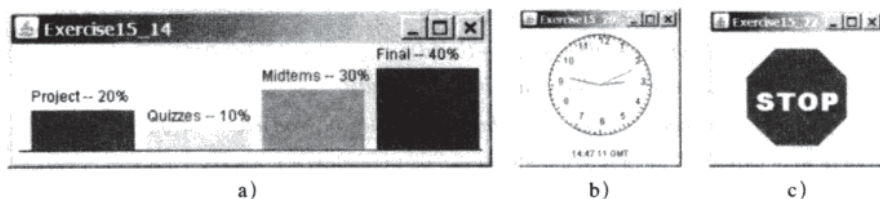


图15-1 可以使用Graphics类中的绘图方法绘制图形

15.2 图形坐标系

要绘图，首先需要确定在哪里画。每个组件都有自己的坐标系，原点 $(0, 0)$ 在组件的左上角。 x 坐标向右增加， y 坐标向下增加。注意，Java的坐标系不同于传统的坐标系，如图15-2所示。

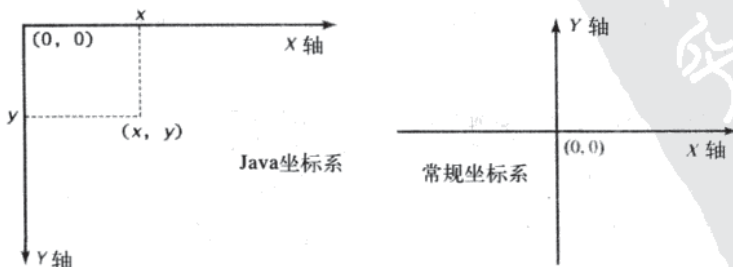


图15-2 Java图形坐标系的计量单位是像素，点 $(0, 0)$ 在它的左上角

在父组件c2（例如面板）中的组件c1（例如按钮）的左上角位置可以使用c1.getX()和c1.getY()进行定位。如图15-3所示， $(x_1, y_1) = (c1.getX(), c1.getY())$ ， $(x_2, y_2) = (c2.getX(), c2.getY())$ 以及 $(x_3, y_3) = (c3.getX(), c3.getY())$ 。

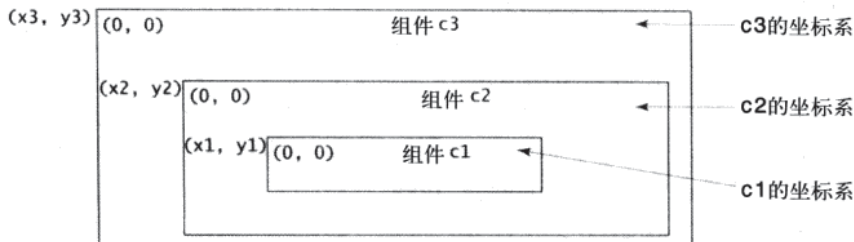


图15-3 每个GUI组件都有自己的坐标系

15.3 Graphics类

Graphics类中提供了绘制字符串、直线、矩形、椭圆、弧形、多边形和折线段的方法，如图15-4所示。

java.awt.Graphics	
<pre> +setColor(color: Color): void +setFont(font: Font): void +drawString(s: String, x: int, y: int): void +drawLine(x1: int, y1: int, x2: int, y2: int): void +drawRect(x: int, y: int, w: int, h: int): void +fillRect(x: int, y: int, w: int, h: int): void +drawRoundRect(x: int, y: int, w: int, h: int, arcAngle: int): void +fillRoundRect(x: int, y: int, w: int, h: int, arcAngle: int): void +draw3DRect(x: int, y: int, w: int, h: int, raised: boolean): void +fill3DRect(x: int, y: int, w: int, h: int, raised: boolean): void +drawOval(x: int, y: int, w: int, h: int): void +fillOval(x: int, y: int, w: int, h: int): void +drawArc(x: int, y: int, w: int, h: int, startAngle: int, arcAngle: int): void +fillArc(x: int, y: int, w: int, h: int, startAngle: int, arcAngle: int): void +drawPolygon(xPoints: int[], yPoints: int[], nPoints: int): void +fillPolygon(xPoints: int[], yPoints: int[], nPoints: int): void +drawPolygon(g: Polygon): void +fillPolygon(g: Polygon): void +drawPolyline(xPoints: int[], yPoints: int[], nPoints: int): void </pre>	<p>为后续绘图设置新的颜色</p> <p>为后续绘图设置新的字体</p> <p>绘制从点 (x, y) 开始的字符串</p> <p>绘制从 (x1, y1) 到 (x2, y2) 的一条直线</p> <p>绘制一个左上角在点 (x, y) 处、宽为w、高为h的矩形</p> <p>绘制一个左上角在点 (x, y) 处、宽为w、高为h的填充矩形</p> <p>绘制一个圆弧宽度为aw、高度为ah的圆角矩形</p> <p>绘制一个圆弧宽度为aw、高度为ah的填充圆角矩形</p> <p>绘制一个从表面凸起或者凹进的3D矩形</p> <p>绘制一个从表面凸起或者凹进的填充3D矩形</p> <p>绘制一个椭圆，椭圆的外接矩形由参数x、y、w和h确定</p> <p>绘制一个填充椭圆，填充椭圆的外接矩形由参数x、y、w和h确定</p> <p>绘制一个圆弧，该圆弧是外接矩形由参数x、y、w和h确定的椭圆的一部分</p> <p>绘制一个x坐标和y坐标构成的数组所定义的闭合多边形。每一对坐标 (x[i], y[i]) 表示一个点</p> <p>绘制一个x坐标和y坐标构成的数组所定义的填充多边形。每一对坐标 (x[i], y[i]) 表示一个点</p> <p>绘制一个由Polygon对象定义的闭合多边形</p> <p>绘制一个由Polygon对象定义的填充多边形</p> <p>绘制一个x坐标和y坐标构成的数组所定义的折线。每一对坐标 (x[i], y[i]) 表示一个点</p>

图15-4 Graphics类包含用于绘制字符串和图形的方法

可以将GUI组件看作一张纸，而将Graphics看作铅笔或画刷。可以应用Graphics类中的方法在GUI组件上进行绘画。

Graphics类是一个提供与设备无关的图形界面的抽象类，它可以在不同平台的屏幕上显示图形和图像。任何时候需要显示组件（例如，按钮、标签和面板）时，JVM都会自动在本地平台上为该组件创建一个Graphics对象，然后传递这个对象来调用paintComponent方法来显示图画。

paintComponent方法的签名如下所示：

```
protected void paintComponent(Graphics g)
```

这个定义在JComponent类中的方法是在第一次显示组件或重新显示组件的时候调用的。

为了在组件上绘图，需要定义一个扩展JPanel的类，并且覆盖它的paintComponent方法来表明绘制什么。程序清单15-1给出在面板上绘制一条线和一个字符串的例子，如图15-5所示。

程序清单15-1 TestPaintComponent.java

```
1 import javax.swing.*;
2 import java.awt.Graphics;
3
4 public class TestPaintComponent extends JFrame {
5     public TestPaintComponent() {
6         add(new NewPanel());
7     }
8
9     public static void main(String[] args) {
10         TestPaintComponent frame = new TestPaintComponent();
11         frame.setTitle("TestPaintComponent");
12         frame.setSize(200, 100);
13         frame.setLocationRelativeTo(null); // Center the frame
14         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15         frame.setVisible(true);
16     }
17 }
18
19 class NewPanel extends JPanel {
20     protected void paintComponent(Graphics g) {
21         super.paintComponent(g);
22         g.drawLine(0, 0, 50, 50);
23         g.drawString("Banner", 0, 40);
24     }
25 }
```

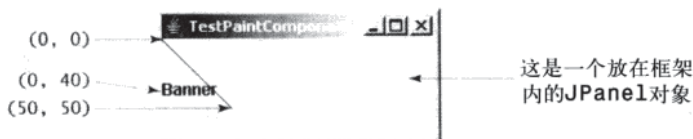


图15-5 在面板上绘制一条线和一个字符串

当第一次显示组件或者任何时候需要重新显示组件时，都会自动调用方法paintComponent来绘制图像。调用super.paintComponent(g)（第21行）来调用父类中定义的paintComponent方法。必须确保在显示新的图画之前清空视图区域。第22行调用drawLine方法来绘制一条从（0，0）到（50，50）的直线。第23行调用drawString方法来绘制一个字符串。

所有的绘图方法都有表明在什么地方绘制对象的参数。在Java中所有的计量单位都是像素。字符串“Banner”绘制在位置（0，40）。

JVM调用paintComponent在组件上进行绘画。用户永远都不要直接调用paintComponent。因为这个原因，将paintComponent的可见性设置为protected就足够了。

面板是不可见的，它们用作一个小型的容器，这个容器将组件进行分组获得所需的布局。JPanel的另一个重要应用是绘画。可以在任何一个Swing GUI组件上绘画，但是，通常应该使用JPanel作为画布在其上绘图。如果如下所示在第19行使用JLabel替换JPanel，那会发生什么呢？

```
class JPanel extends JLabel {
```

程序仍可以工作，但是不推荐这样做，因为JLabel设计为创建一个标签，而不是为了绘画。为保持统一性，本书将通过JPanel的子类来定义画布类。

提示 一些教科书通过JComponent的子类来定义画布类。这里的问题是如果要在画布上设置背景色，就必须写绘制背景色的代码。一个简单的setBackground(Color color)方法是不能在JComponent中设置背景色的。

15.4 绘制字符串、直线、矩形和椭圆

方法drawString(String s,int x,int y)绘制以点(x,y)为起点的字符串，如图15-6a所示。

方法drawLine(int x1,int y1,int x2,int y2)绘制从点(x1,y1)到点(x2,y2)的直线段，如图15-6b所示。

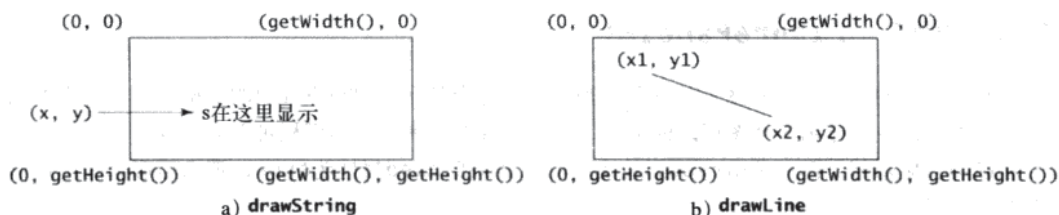


图15-6 a) 方法drawString(s, x, y) 绘制一个从点(x, y)开始的字符串；

b) 方法drawString(x1, y1, x2, y2) 绘制在两个定点之间的线段

Java提供六种绘制空心矩形或填充颜色的矩形的方法。可以绘制或填充直角矩形、圆角矩形或三维矩形。

drawRect(int x,int y,int w,int h)方法绘制一个直角矩形，而fillRect(int x, int y, int w,int h)方法绘制一个填充颜色的矩形。参数x和y表示这个矩形的左上角，w和h表示这个矩形的宽和高（参见图15-7）。

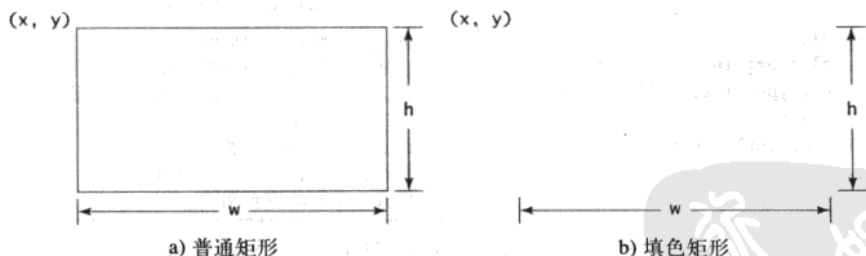


图15-7 a) 方法drawRect(x, y, w, h) 绘制一个矩形；

b) 方法fillRect(x, y, w, h) 绘制填充颜色的矩形

方法drawRoundRect(int x,int y,int w,int h,int aw,int ah)绘制一个圆角矩形，而方法fillRoundRect(int x,int y,int w,int h,int aw,int ah)绘制一个填充颜色的圆角矩形。参数x、y、w和h的含义与drawRect方法中的参数含义一样，参数aw指角上圆弧的水平直径，ah指角上圆弧的垂直直径（参见图15-8a）。换句话说，aw和ah是每个角上四分之一椭圆的宽和高。

方法draw3DRect(int x,int y,int w,int h,boolean raised)绘制一个三维矩形，方法fill3DRect(int x,int y,int w,int h,boolean raised)绘制一个填充颜色的三维矩形。参数x、y、w和h的含义与drawRect方法的参数含义相同。最后一个参数是布尔值，表示矩形是从表面凸起还是凹进。

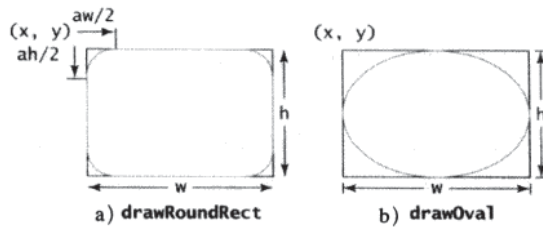


图15-8 a) 方法drawRoundRect(x,y,w,h,aw,ah)绘制一个圆角矩形;
b) 方法drawOval(x,y,w,h)绘制一个基于它的外接矩形的椭圆

根据要绘制的是空心椭圆还是填充椭圆, 可以使用drawOval(int x,int y,int w,int h)方法或者fillOval(int x,int y,int w,int h)方法。椭圆是根据它的外接矩形绘制的。参数x和y表示外接矩形左上角的坐标, w和h分别表示外接矩形的宽和高, 如图15-8b所示。

15.5 实例学习: FigurePanel类

这个例子开发可以显示各种图形的非常有用的类。这个类允许用户设置图的类型、确定是否填充该图形以及是否在面板上显示这个图形。该类的UML图如图15-9所示。该面板可以显示直线段、矩形、圆角矩形和椭圆。显示哪个图形是由type属性决定的。如果filled属性为true, 那么矩形、圆角矩形和椭圆在面板上都是填充颜色的。

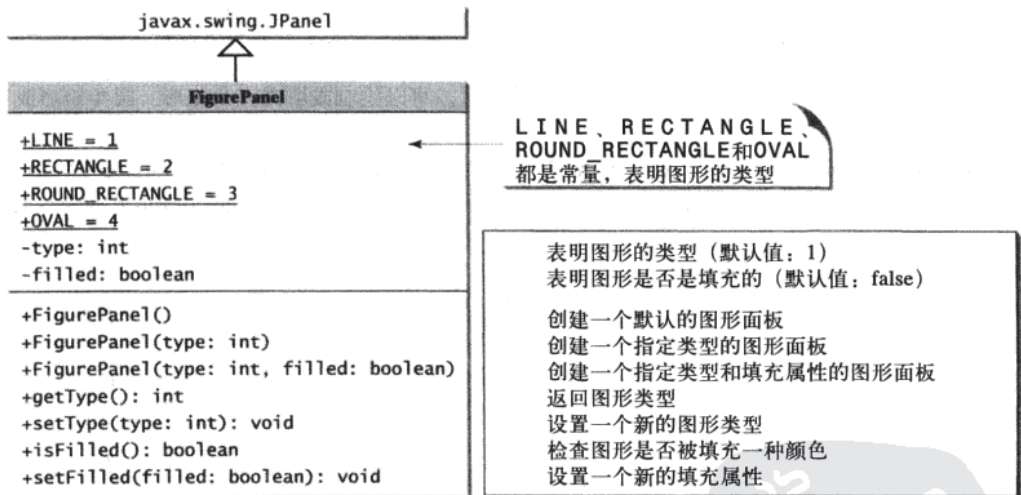


图15-9 FigurePanel在面板上显示各种类型的图形

这个UML图作为FigurePanel类的合约。使用者可以在不知道这个类是如何实现的情况下使用它。我们从编写程序清单15-2中的程序开始介绍, 该程序使用这个类显示6个图形面板, 如图15-10所示。

程序清单15-3实现FigurePanel类。四个常量——LINE、RECTANGLE、ROUND_RECTANGLE和OVAL, 都是在第6~9行声明的。这四种图形都是依照type属性绘制的 (第37行)。setColor方法 (第39、44、53、62行) 为图设置新的颜色。

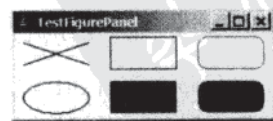


图15-10 创建六个FigurePanel对象显示六个图形

程序清单15-2 TestFigurePanel.java

```

1 import java.awt.*;
2 import javax.swing.*;
3
4 public class TestFigurePanel extends JFrame {
5     public TestFigurePanel() {
6         setLayout(new GridLayout(2, 3, 5, 5));
7         add(new FigurePanel(FigurePanel.LINE));
8         add(new FigurePanel(FigurePanel.RECTANGLE));
9         add(new FigurePanel(FigurePanel.ROUND_RECTANGLE));
10        add(new FigurePanel(FigurePanel.OVAL));
11        add(new FigurePanel(FigurePanel.RECTANGLE, true));
12        add(new FigurePanel(FigurePanel.ROUND_RECTANGLE, true));
13    }
14
15    public static void main(String[] args) {
16        TestFigurePanel frame = new TestFigurePanel();
17        frame.setSize(400, 200);
18        frame.setTitle("TestFigurePanel");
19        frame.setLocationRelativeTo(null); // Center the frame
20        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21        frame.setVisible(true);
22    }
23 }

```

程序清单15-3 FigurePanel.java

```

1 import java.awt.*;
2 import javax.swing.JPanel;
3
4 public class FigurePanel extends JPanel {
5     // Define constants
6     public static final int LINE = 1;
7     public static final int RECTANGLE = 2;
8     public static final int ROUND_RECTANGLE = 3;
9     public static final int OVAL = 4;
10
11     private int type = 1;
12     private boolean filled = false;
13
14     /** Construct a default FigurePanel */
15     public FigurePanel() {
16     }
17
18     /** Construct a FigurePanel with the specified type */
19     public FigurePanel(int type) {
20         this.type = type;
21     }
22
23     /** Construct a FigurePanel with the specified type and filled */
24     public FigurePanel(int type, boolean filled) {
25         this.type = type;
26         this.filled = filled;
27     }
28
29     /** Draw a figure on the panel */
30     protected void paintComponent(Graphics g) {
31         super.paintComponent(g);
32
33         // Get the appropriate size for the figure
34         int width = getWidth();
35         int height = getHeight();
36
37         switch (type) {
38             case LINE: // Display two cross lines

```

```

39         g.setColor(Color.BLACK);
40         g.drawLine(10, 10, width - 10, height - 10);
41         g.drawLine(width - 10, 10, 10, height - 10);
42         break;
43     case RECTANGLE: // Display a rectangle
44         g.setColor(Color.BLUE);
45         if (filled)
46             g.fillRect((int)(0.1 * width), (int)(0.1 * height),
47                         (int)(0.8 * width), (int)(0.8 * height));
48         else
49             g.drawRect((int)(0.1 * width), (int)(0.1 * height),
50                        (int)(0.8 * width), (int)(0.8 * height));
51         break;
52     case ROUND_RECTANGLE: // Display a round-cornered rectangle
53         g.setColor(Color.RED);
54         if (filled)
55             g.fillRoundRect((int)(0.1 * width), (int)(0.1 * height),
56                             (int)(0.8 * width), (int)(0.8 * height), 20, 20);
57         else
58             g.drawRoundRect((int)(0.1 * width), (int)(0.1 * height),
59                              (int)(0.8 * width), (int)(0.8 * height), 20, 20);
60         break;
61     case OVAL: // Display an oval
62         g.setColor(Color.BLACK);
63         if (filled)
64             g.fillOval((int)(0.1 * width), (int)(0.1 * height),
65                       (int)(0.8 * width), (int)(0.8 * height));
66         else
67             g.drawOval((int)(0.1 * width), (int)(0.1 * height),
68                       (int)(0.8 * width), (int)(0.8 * height));
69     }
70 }
71
72 /** Set a new figure type */
73 public void setType(int type) {
74     this.type = type;
75     repaint();
76 }
77
78 /** Return figure type */
79 public int getType() {
80     return type;
81 }
82
83 /** Set a new filled property */
84 public void setFilled(boolean filled) {
85     this.filled = filled;
86     repaint();
87 }
88
89 /** Check if the figure is filled */
90 public boolean isFilled() {
91     return filled;
92 }
93
94 /** Specify preferred size */
95 public Dimension getPreferredSize() {
96     return new Dimension(80, 80);
97 }
98 }

```

repaint方法(第75、86行)是在Component类中定义的。调用repaint方法会导致paintComponent方法被调用。调用repaint方法以刷新视图区域。一般情况下,如果要显示新的东西,就应该调用这个方法。

警告 永远都不要直接调用`paintComponent`方法。它应该在视图区域改变时由JVM调用或者由`repaint`方法调用。应该覆盖`paintComponent`方法告诉系统如何绘制视图区域，但永远都不要覆盖`repaint`方法。

注意 `repaint`方法提出更新视图区域的请求并且立即返回。它的效果是异步的，这意味着，它由JVM决定在独立的线程上执行`paintComponent`方法。

在`FigurePanel`类中覆盖定义在`Component`类中的`getPreferredSize()`方法（第95~97行），指定合适的尺寸，以便布局管理器考虑什么时候放置一个`FigurePanel`对象。根据布局管理器的规则，布局管理器可能考虑也可能不考虑这一属性。例如，在`FlowLayout`管理器的容器中，组件可以使用希望的尺寸，但是如果放在`GridLayout`管理器的容器中，组件希望的尺寸可能会被忽略。最好的方法就是覆盖在`JPanel`子类中的`getPreferredSize()`方法以确定希望的尺寸，因为默认情况下`JPanel`的尺寸是 0×0 。

15.6 绘制弧形

弧形可以认为是以矩形为边界的椭圆的一部分。绘制或者填充弧形的方法如下：

```
drawArc(int x, int y, int w, int h, int startAngle, int arcAngle);
fillArc(int x, int y, int w, int h, int startAngle, int arcAngle);
```

参数`x`、`y`、`w`和`h`的含义与`drawOval`方法中参数的含义是一样的；参数`startAngle`是起始角，`arcAngle`是跨度角（即弧线覆盖的角）。角的单位是度，遵循通常的数学习惯（即0度指向东边，并且从东边开始沿逆时针方向旋转的角度为正角），参见图15-11。

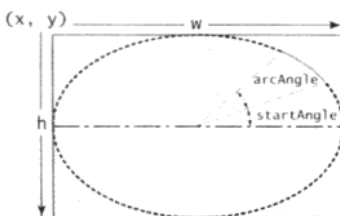


图15-11 `drawArc`方法绘制一个基于特定角的椭圆的弧形

程序清单15-4是一个如何绘制弧形的例子，它的输出如图15-12所示。

程序清单15-4 DrawArcs.java

```
1 import javax.swing.JFrame;
2 import javax.swing.JPanel;
3 import java.awt.Graphics;
4
5 public class DrawArcs extends JFrame {
6     public DrawArcs() {
7         setTitle("DrawArcs");
8         add(new ArcsPanel());
9     }
10
11     /** Main method */
12     public static void main(String[] args) {
13         DrawArcs frame = new DrawArcs();
14         frame.setSize(250, 300);
15         frame.setLocationRelativeTo(null); // Center the frame
16         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17         frame.setVisible(true);
18     }
19 }
20
```



```

21 // The class for drawing arcs on a panel
22 class ArcsPanel extends JPanel {
23     // Draw four blades of a fan
24     protected void paintComponent(Graphics g) {
25         super.paintComponent(g);
26
27         int xCenter = getWidth() / 2;
28         int yCenter = getHeight() / 2;
29         int radius = (int)(Math.min(getWidth(), getHeight()) * 0.4);
30
31         int x = xCenter - radius;
32         int y = yCenter - radius;
33
34         g.fillArc(x, y, 2 * radius, 2 * radius, 0, 30);
35         g.fillArc(x, y, 2 * radius, 2 * radius, 90, 30);
36         g.fillArc(x, y, 2 * radius, 2 * radius, 180, 30);
37         g.fillArc(x, y, 2 * radius, 2 * radius, 270, 30);
38     }
39 }

```

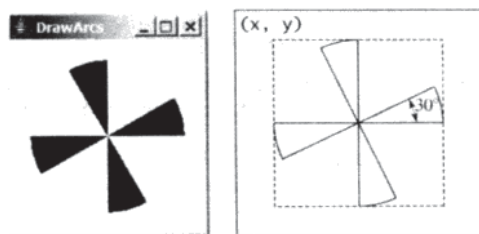


图15-12 程序绘制四个填充颜色的弧形

角度可以是负的。负起始角是从东边开始沿顺时针方向旋转，如图15-13所示。负跨度角是从起始角开始顺时针转动的角度。下面的两条语句绘制同一段弧形：

```

g.fillArc(x, y, 2 * radius, 2 * radius, -30, -20);
g.fillArc(x, y, 2 * radius, 2 * radius, -50, 20);

```

- 语句使用负的起始角-30和负的跨度角-20，如图15-13a所示。第二条语句使用负的起始角-50和正的跨度角20，如图15-13b所示。

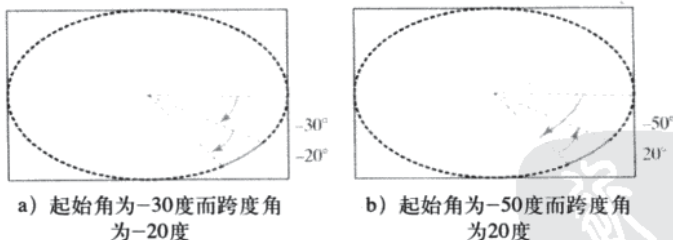


图15-13 角度可以是负的

15.7 绘制多边形和折线段

为了绘制一个多边形，首先需要使用多边形类Polygon来创建一个Polygon对象，如图15-14所示。

多边形是一个封闭的二维区域。这个区域由任意多条线段围成，每条线段都是多边形的一条边。在内部，多边形由坐标对 (x, y) 的序列组成，每对坐标定义多边形的一个顶点，两对相邻坐标对就是多边形一条边的两个端点。第一对点和最后一对点由封闭多边形的线段连接。

下面是用于创建Polygon对象并给多边形增加一个点的例子：


```

Polygon polygon = new Polygon();
polygon.addPoint(40, 20);
polygon.addPoint(70, 40);
polygon.addPoint(60, 80);
polygon.addPoint(45, 45);
polygon.addPoint(20, 60);

```

java.awt.Polygon	
+xpoints: int[]	所有多边形的点的x坐标
+ypoints: int[]	所有多边形的点的y坐标
+npoints: int	多边形中点的个数
<hr/>	
+Polygon()	创建一个空的多边形
+Polygon(xpoints: int[], ypoints: int[], npoints: int)	创建一个指定点构成的多边形
+addPoint(x: int, y: int)	给多边形追加一个点

图15-14 Polygon类对多边形建模

在添加完这些点之后, xpoints是 {40, 70, 60, 45, 20}, ypoint是{20, 40, 80, 45, 60}, npoints是5。xpoint、ypoint和npoint都是Polygon中的公共数据域, 这是一个不好的设计。原则上讲, 所有的数据域都应该是私有的。

为了绘制或填充一个多边形, 使用下面的Graphics类中的方法之一:

```

drawPolygon(Polygon polygon);
fillPolygon(Polygon polygon);
drawPolygon(int[] xpoints, int[] ypoints, int npoints);
fillPolygon(int[] xpoints, int[] ypoints, int npoints);

```

例如,

```

int x[] = {40, 70, 60, 45, 20};
int y[] = {20, 40, 80, 45, 60};
g.drawPolygon(x, y, x.length);

```

该绘制方法通过绘制点 (x[i], y[i]) 与点 (x[i+1], y[i+1]) 之间的线段打开多边形, 其中i=0, 1, 2, ..., x.length-1; 通过绘制第一个点和最后一个点之间的连线封闭多边形 (参见图15-15a)。

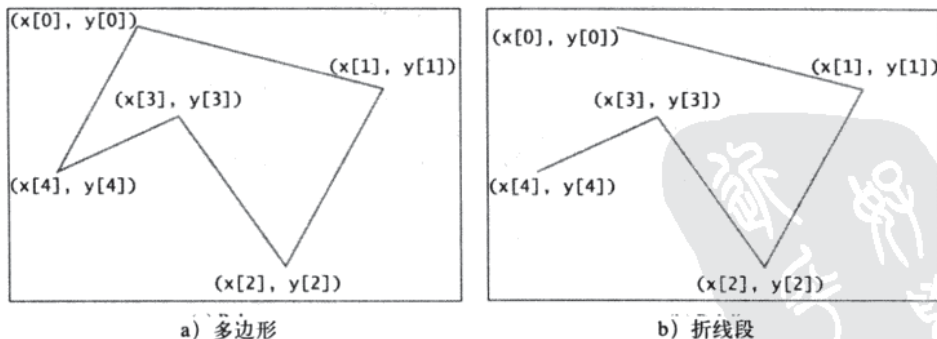


图15-15 drawPolygon方法绘制一个多边形, drawPolyline绘制一条折线段

要绘制一条折线段, 可以使用drawPolyline(int[] x, int[] y, int nPoints)方法, 它绘制出由x坐标和y坐标数组定义的相互连接的一系列线段。例如, 下面的代码画出了图15-15b所示的折线段。

```

int x[] = {40, 70, 60, 45, 20};
int y[] = {20, 40, 80, 45, 60};
g.drawPolyline(x, y, x.length);

```

程序清单15-5是一个如何绘制六边形的例子，输出结果如图15-16所示。

程序清单15-5 DrawPolygon.java

```
1 import javax.swing.JFrame;
2 import javax.swing.JPanel;
3 import java.awt.Graphics;
4 import java.awt.Polygon;
5
6 public class DrawPolygon extends JFrame {
7     public DrawPolygon() {
8         setTitle("DrawPolygon");
9         add(new PolygonsPanel());
10    }
11
12    /** Main method */
13    public static void main(String[] args) {
14        DrawPolygon frame = new DrawPolygon();
15        frame.setSize(200, 250);
16        frame.setLocationRelativeTo(null); // Center the frame
17        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
18        frame.setVisible(true);
19    }
20 }
21
22 // Draw a polygon in the panel
23 class PolygonsPanel extends JPanel {
24     protected void paintComponent(Graphics g) {
25         super.paintComponent(g);
26
27         int xCenter = getWidth() / 2;
28         int yCenter = getHeight() / 2;
29         int radius = (int)(Math.min(getWidth(), getHeight()) * 0.4);
30
31         // Create a Polygon object
32         Polygon polygon = new Polygon();
33
34         // Add points to the polygon in this order
35         polygon.addPoint(xCenter + radius, yCenter);
36         polygon.addPoint((int)(xCenter + radius *
37             Math.cos(2 * Math.PI / 6)), (int)(yCenter - radius *
38             Math.sin(2 * Math.PI / 6)));
39         polygon.addPoint((int)(xCenter + radius *
40             Math.cos(2 * 2 * Math.PI / 6)), (int)(yCenter - radius *
41             Math.sin(2 * 2 * Math.PI / 6)));
42         polygon.addPoint((int)(xCenter + radius *
43             Math.cos(3 * 2 * Math.PI / 6)), (int)(yCenter - radius *
44             Math.sin(3 * 2 * Math.PI / 6)));
45         polygon.addPoint((int)(xCenter + radius *
46             Math.cos(4 * 2 * Math.PI / 6)), (int)(yCenter - radius *
47             Math.sin(4 * 2 * Math.PI / 6)));
48         polygon.addPoint((int)(xCenter + radius *
49             Math.cos(5 * 2 * Math.PI / 6)), (int)(yCenter - radius *
50             Math.sin(5 * 2 * Math.PI / 6)));
51
52         // Draw the polygon
53         g.drawPolygon(polygon);
54     }
55 }
```

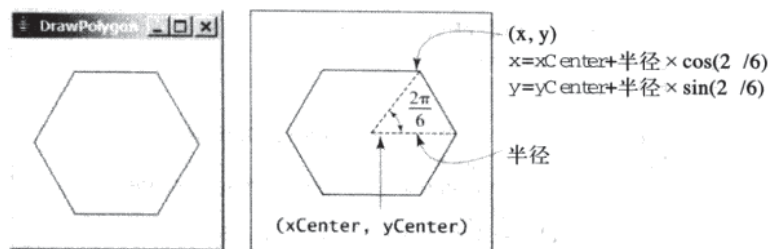


图15-16 程序使用drawPolygon方法绘制一个多边形

15.8 使用FontMetrics类居中显示字符串

可以在一个面板的任何位置显示字符串。可以居中显示它吗？要想做到这一点，需要使用FontMetrics类，对特定字体的字符串测量出确切的宽度和高度。FontMetrics可以测量给定字体的以下属性（如图15-17所示）：

- **Leading**（发音为ledding）是文本行之间的距离。
- **Ascent**表示字符从基线到上升线的距离。字体中多数字符的顶点都在上升线之下，但是也有一些可能会延伸到上升线以上。
- **Descent**是从基线到下沉线的距离。字体中大多数降字（例如，j、y和g）都在下沉线之上，但是也有一些可能会延伸到下沉线以下。
- **Height**是leading、ascent和descent的和。

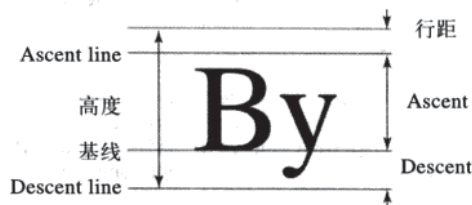


图15-17 FontMetrics类可以被用来确定给定字体的字符的字体属性

FontMetrics是一个抽象类。要得到给定字体的FontMetrics对象，可以使用定义在Graphics类中的以下getFontMetrics方法：

- **public FontMetrics getFontMetrics (Font font)**

返回指定字体的字体尺寸。

- **public FontMetrics getFontMetrics ()**

返回当前字体的字体尺寸。

可以使用下面的FontMetrics类中的实例方法得到字体属性，以及使用这种字体绘制的字符串的宽度：

```
public int getAscent() // Return the ascent
public int getDescent() // Return the descent
public int getLeading() // Return the leading
public int getHeight() // Return the height
public int stringWidth(String str) // Return the width of the string
```

程序清单15-6给出在面板中央显示消息的例子，如图15-18所示。



图15-18 程序使用FontMetrics类测量字符串的宽度和高度，并在框架中央显示该字符串

程序清单15-6 TestCenterMessage.java

```

1 import javax.swing.*;
2 import java.awt.*;
3
4 public class TestCenterMessage extends JFrame{
5     public TestCenterMessage() {
6         CenterMessage messagePanel = new CenterMessage();
7         add(messagePanel);
8         messagePanel.setBackground(Color.WHITE);
9         messagePanel.setFont(new Font("Californian FB", Font.BOLD, 30));
10    }
11
12    /** Main method */
13    public static void main(String[] args) {
14        TestCenterMessage frame = new TestCenterMessage();
15        frame.setSize(300, 150);
16        frame.setLocationRelativeTo(null); // Center the frame
17        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
18        frame.setVisible(true);
19    }
20 }
21
22 class CenterMessage extends JPanel {
23     /** Paint the message */
24     protected void paintComponent(Graphics g) {
25         super.paintComponent(g);
26
27         // Get font metrics for the current font
28         FontMetrics fm = g.getFontMetrics();
29
30         // Find the center location to display
31         int stringWidth = fm.stringWidth("Welcome to Java");
32         int stringAscent = fm.getAscent();
33
34         // Get the position of the leftmost character in the baseline
35         int xCoordinate = getWidth() / 2 - stringWidth / 2;
36         int yCoordinate = getHeight() / 2 + stringAscent / 2;
37
38         g.drawString("Welcome to Java", xCoordinate, yCoordinate);
39     }
40 }

```

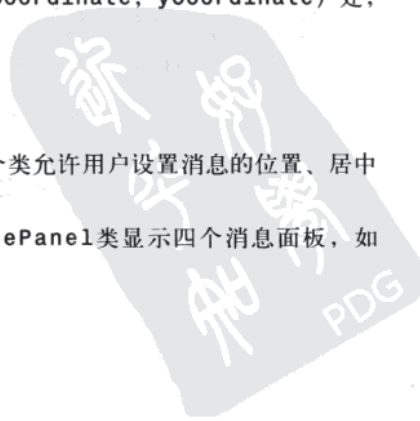
在Component类中定义的方法getWidth()和getHeight() (第35~36行), 分别返回组件的宽度和长度。

由于消息是centered, 所以字符串的第一个字符应该放在 (xCoordinate, yCoordinate) 处, 如图15-18所示。

15.9 实例学习: MessagePanel类

本实例开发一个有用的类, 它可以在面板中显示一条消息。这个类允许用户设置消息的位置、居中放置消息、使用指定间距移动消息。该类的合约如图15-19所示。

我们从编写程序清单15-7中的测试程序开始, 它使用MessagePanel类显示四个消息面板, 如图15-20所示。



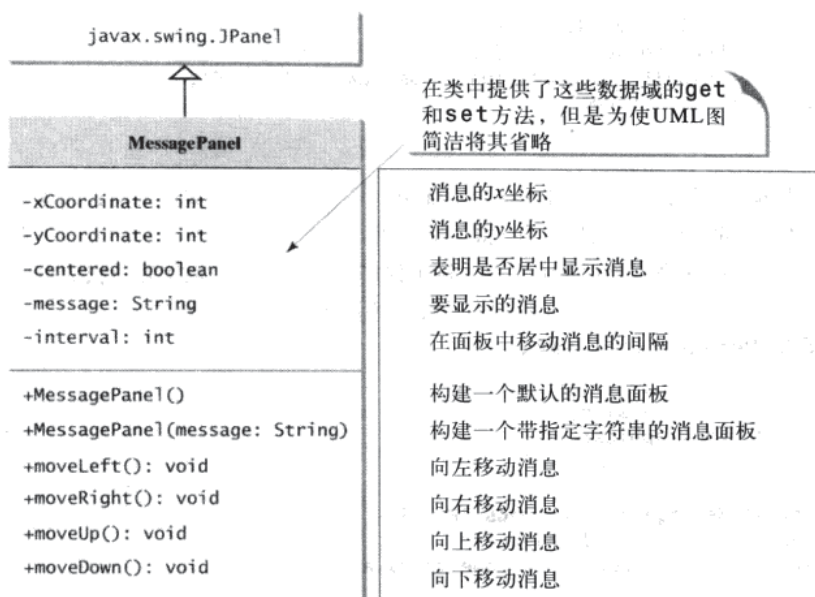


图15-19 MessagePanel类在面板上显示消息

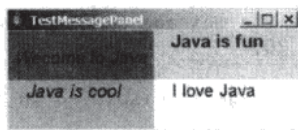


图15-20 TestMessagePanel使用MessagePanel显示四个消息面板

程序清单15-7 TestMessagePanel.java

```

1 import java.awt.*;
2 import javax.swing.*;
3
4 public class TestMessagePanel extends JFrame {
5     public TestMessagePanel() {
6         MessagePanel messagePanel1 = new MessagePanel("Welcome to Java");
7         MessagePanel messagePanel2 = new MessagePanel("Java is fun");
8         MessagePanel messagePanel3 = new MessagePanel("Java is cool");
9         MessagePanel messagePanel4 = new MessagePanel("I love Java");
10        messagePanel1.setFont(new Font("SansSerif", Font.ITALIC, 20));
11        messagePanel2.setFont(new Font("Courier", Font.BOLD, 20));
12        messagePanel3.setFont(new Font("Times", Font.ITALIC, 20));
13        messagePanel4.setFont(new Font("Californian FB", Font.PLAIN, 20));
14        messagePanel1.setBackground(Color.RED);
15        messagePanel2.setBackground(Color.CYAN);
16        messagePanel3.setBackground(Color.GREEN);
17        messagePanel4.setBackground(Color.WHITE);
18        messagePanel1.setCentered(true);
19
20        setLayout(new GridLayout(2, 2));
21        add(messagePanel1);
22        add(messagePanel2);
23        add(messagePanel3);
24        add(messagePanel4);
25    }
26
27    public static void main(String[] args) {
28        TestMessagePanel frame = new TestMessagePanel();
29        frame.setSize(300, 200);
  
```

```

30     frame.setTitle("TestMessagePanel");
31     frame.setLocationRelativeTo(null); // Center the frame
32     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
33     frame.setVisible(true);
34 }
35 }

```

本节的剩余内容就是解释如何实现MessagePanel类。因为可以使用这个类，而无须知道它是如何实现的，所以，如果你希望的话，可以跳过它的实现过程。

MessagePanel类在程序清单15-8中实现。这个程序看起来很长，但实际上很简单，因为大多数方法都是get方法和set方法，并且每个方法相对来说都很短，而且易于阅读。

程序清单15-8 MessagePanel.java

```

1  import java.awt.FontMetrics;
2  import java.awt.Dimension;
3  import java.awt.Graphics;
4  import javax.swing.JPanel;
5
6  public class MessagePanel extends JPanel {
7      /** The message to be displayed */
8      private String message = "Welcome to Java";
9
10     /** The x-coordinate where the message is displayed */
11     private int xCoordinate = 20;
12
13     /** The y-coordinate where the message is displayed */
14     private int yCoordinate = 20;
15
16     /** Indicate whether the message is displayed in the center */
17     private boolean centered;
18
19     /** The interval for moving the message horizontally and
20      *   vertically */
21     private int interval = 10;
22
23     /** Construct with default properties */
24     public MessagePanel() {
25     }
26
27     /** Construct a message panel with a specified message */
28     public MessagePanel(String message) {
29         this.message = message;
30     }
31
32     /** Return message */
33     public String getMessage() {
34         return message;
35     }
36
37     /** Set a new message */
38     public void setMessage(String message) {
39         this.message = message;
40         repaint();
41     }
42
43     /** Return xCoordinator */
44     public int getXCoordinate() {
45         return xCoordinate;
46     }
47
48     /** Set a new xCoordinator */
49     public void setXCoordinate(int x) {
50         this.xCoordinate = x;
51         repaint();

```

```
52 }
53
54 /** Return yCoordinator */
55 public int getYCoordinate() {
56     return yCoordinate;
57 }
58
59 /** Set a new yCoordinator */
60 public void setYCoordinate(int y) {
61     this.yCoordinate = y;
62     repaint();
63 }
64
65 /** Return centered */
66 public boolean isCentered() {
67     return centered;
68 }
69
70 /** Set a new centered */
71 public void setCentered(boolean centered) {
72     this.centered = centered;
73     repaint();
74 }
75
76 /** Return interval */
77 public int getInterval() {
78     return interval;
79 }
80
81 /** Set a new interval */
82 public void setInterval(int interval) {
83     this.interval = interval;
84     repaint();
85 }
86
87 /** Paint the message */
88 protected void paintComponent(Graphics g) {
89     super.paintComponent(g);
90
91     if (centered) {
92         // Get font metrics for the current font
93         FontMetrics fm = g.getFontMetrics();
94
95         // Find the center location to display
96         int stringWidth = fm.stringWidth(message);
97         int stringAscent = fm.getAscent();
98         // Get the position of the leftmost character in the baseline
99         xCoordinate = getWidth() / 2 - stringWidth / 2;
100         yCoordinate = getHeight() / 2 + stringAscent / 2;
101     }
102
103     g.drawString(message, xCoordinate, yCoordinate);
104 }
105
106 /** Move the message left */
107 public void moveLeft() {
108     xCoordinate -= interval;
109     repaint();
110 }
111
112 /** Move the message right */
113 public void moveRight() {
114     xCoordinate += interval;
115     repaint();
116 }
```

```

117
118  /** Move the message up */
119  public void moveUp() {
120      yCoordinate -= interval;
121      repaint();
122  }
123
124  /** Move the message down */
125  public void moveDown() {
126      yCoordinate += interval;
127      repaint();
128  }
129
130  /** Override get method for preferredSize */
131  public Dimension getPreferredSize() {
132      return new Dimension(200, 30);
133  }
134 }

```

如果属性centered为true (第91行), 那么paintComponent方法居中显示这条消息。message在第8行初始化为“Welcome to Java”。如果它没有初始化, 当使用无参构造方法创建MessagePanel时就会导致一个NullPointerException的运行时错误, 因为message在第103行会是null。

警告 MessagePanel类使用属性xCoordinate和yCoordinate指定消息在面板上显示的位置。不要使用属性名x和y, 因为它们已经在Component类中定义, 使用方法getX()和getY()返回上一级坐标系中组件的位置。

注意 Component类具有setBackground、setForeground和setFont方法。这些方法用来设置整个组件的颜色和字体。如果想用不同的颜色和字体在同一个面板中绘制几条消息, 必须使用Graphics类中的setColor和setFont方法为当前的图形设定颜色和字体。

注意 Java程序设计的一个主要特征就是类的复用。贯穿本书, 我们将开发可重用的类, 然后在后面重用它们。MessagePanel就是一个这样的例子, 程序清单10-2中的Loan和程序清单15-3中的FigurePanel都是这样的例子。无论什么时候需要在面板中显示一条消息, MessagePanel都可以重复使用。要使类在一个广泛的应用范围内可复用, 应该提供多种使用它的方式。MessagePanel类提供了许多将在本书的很多例子中使用的属性和方法。下一节给出在面板上显示一个钟表图像的有用的且可重用的类。

15.10 实例学习: StillClock类

这个实例会开发一个在面板上显示时钟的类。该类的合约如图15-21所示。

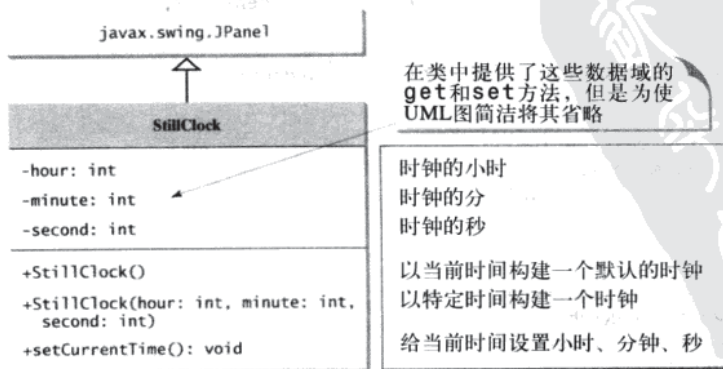


图15-21 StillClock类显示一个模拟时钟

我们首先编写程序清单15-9中的测试程序，程序使用StillClock类显示一个模拟时钟，然后使用MessagePanel类在面板中显示小时、分钟和秒，如图15-22a所示。

程序清单15-9 DisplayClock.java

```

1 import java.awt.*;
2 import javax.swing.*;
3
4 public class DisplayClock extends JFrame {
5     public DisplayClock() {
6         // Create an analog clock for the current time
7         StillClock clock = new StillClock();
8
9         // Display hour, minute, and second in the message panel
10        MessagePanel messagePanel = new MessagePanel(clock.getHour() +
11        ":" + clock.getMinute() + ":" + clock.getSecond());
12        messagePanel.setCentered(true);
13        messagePanel.setForeground(Color.blue);
14        messagePanel.setFont(new Font("Courier", Font.BOLD, 16));
15
16        // Add the clock and message panel to the frame
17        add(clock);
18        add(messagePanel, BorderLayout.SOUTH);
19    }
20
21    public static void main(String[] args) {
22        DisplayClock frame = new DisplayClock();
23        frame.setTitle("DisplayClock");
24        frame.setSize(300, 350);
25        frame.setLocationRelativeTo(null); // Center the frame
26        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
27        frame.setVisible(true);
28    }
29 }

```

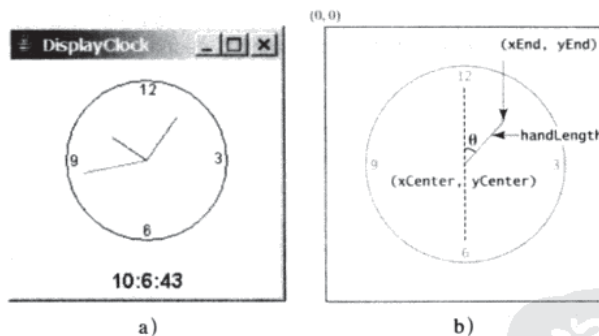


图15-22 a) 程序DisplayClock显示一个给出当前时间的时钟；
b) 时钟表针的端点由给定的跨度角、表针长度和中心点确定

本节的剩余内容就来解释如何实现StillClock类。因为可以使用这个类，而无须知道它是如何实现的，所以，如果你希望的话是可以跳过它的实现过程的。

为了画一个时钟，需要先画一个圆和三个表示秒、分、时的表针。画表针需要指定一条线段的两个端点。如图15-22b所示，一个端点是时钟的中心 ($xCenter$, $yCenter$)，而另一个端点 ($xEnd$, $yEnd$) 由下面的公式决定：

$$\begin{aligned}
 xEnd &= xCenter + handLength \times \sin(\theta) \\
 yEnd &= yCenter - handLength \times \cos(\theta)
 \end{aligned}$$

由于一分钟有60秒，所以秒针的角度是：

$$second \times (2\pi/60)$$

分针的位置由分钟数和秒数决定。和秒一起计算的精确分钟值是 $\text{minute} + \text{second}/60$ 。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，所以分针的角度是

$$(\text{minute} + \text{second}/60) \times (2\pi/60)$$

由于一圈被分成12小时，所以时针的角度是

$$(\text{hour} + \text{minute}/60 + \text{second}/(60 \times 60)) \times (2\pi/12)$$

为了简单起见，在计算分针角度和时针角度时可以忽略秒数，因为它们小到可以忽略不计。因此，秒针、分针和时针的端点可以根据下面的公式计算：

```
xSecond = xCenter + secondHandLength × sin(second × (2π/60))
ySecond = yCenter - secondHandLength × cos(second × (2π/60))
xMinute = xCenter + minuteHandLength × sin(minute × (2π/60))
yMinute = yCenter - minuteHandLength × cos(minute × (2π/60))
xHour = xCenter + hourHandLength × sin((hour + minute/60) × (2π/60))
yHour = yCenter - hourHandLength × cos((hour + minute/60) × (2π/60))
```

StillClock类在程序清单15-10中实现。

程序清单15-10 StillClock.java

```
1 import java.awt.*;
2 import javax.swing.*;
3 import java.util.*;
4
5 public class StillClock extends JPanel {
6     private int hour;
7     private int minute;
8     private int second;
9
10    /** Construct a default clock with the current time */
11    public StillClock() {
12        setCurrentTime();
13    }
14
15    /** Construct a clock with specified hour, minute, and second */
16    public StillClock(int hour, int minute, int second) {
17        this.hour = hour;
18        this.minute = minute;
19        this.second = second;
20    }
21
22    /** Return hour */
23    public int getHour() {
24        return hour;
25    }
26
27    /** Set a new hour */
28    public void setHour(int hour) {
29        this.hour = hour;
30        repaint();
31    }
32
33    /** Return minute */
34    public int getMinute() {
35        return minute;
36    }
37
38    /** Set a new minute */
39    public void setMinute(int minute) {
40        this.minute = minute;
41        repaint();
42    }
43
44    /** Return second */
```

```

45 public int getSecond() {
46     return second;
47 }
48
49 /** Set a new second */
50 public void setSecond(int second) {
51     this.second = second;
52     repaint();
53 }
54
55 /** Draw the clock */
56 protected void paintComponent(Graphics g) {
57     super.paintComponent(g);
58
59     // Initialize clock parameters
60     int clockRadius =
61         (int)(Math.min(getWidth(), getHeight()) * 0.8 * 0.5);
62     int xCenter = getWidth() / 2;
63     int yCenter = getHeight() / 2;
64
65     // Draw circle
66     g.setColor(Color.BLACK);
67     g.drawOval(xCenter - clockRadius, yCenter - clockRadius,
68         2 * clockRadius, 2 * clockRadius);
69     g.drawString("12", xCenter - 5, yCenter - clockRadius + 12);
70     g.drawString("9", xCenter - clockRadius + 3, yCenter + 5);
71     g.drawString("3", xCenter + clockRadius - 10, yCenter + 3);
72     g.drawString("6", xCenter - 3, yCenter + clockRadius - 3);
73
74     // Draw second hand
75     int sLength = (int)(clockRadius * 0.8);
76     int xSecond = (int)(xCenter + sLength *
77         Math.sin(second * (2 * Math.PI / 60)));
78     int ySecond = (int)(yCenter - sLength *
79         Math.cos(second * (2 * Math.PI / 60)));
80     g.setColor(Color.red);
81     g.drawLine(xCenter, yCenter, xSecond, ySecond);
82
83     // Draw minute hand
84     int mLength = (int)(clockRadius * 0.65);
85     int xMinute = (int)(xCenter + mLength *
86         Math.sin(minute * (2 * Math.PI / 60)));
87     int yMinute = (int)(yCenter - mLength *
88         Math.cos(minute * (2 * Math.PI / 60)));
89     g.setColor(Color.blue);
90     g.drawLine(xCenter, yCenter, xMinute, yMinute);
91
92     // Draw hour hand
93     int hLength = (int)(clockRadius * 0.5);
94     int xHour = (int)(xCenter + hLength *
95         Math.sin((hour % 12 + minute / 60.0) * (2 * Math.PI / 12)));
96     int yHour = (int)(yCenter - hLength *
97         Math.cos((hour % 12 + minute / 60.0) * (2 * Math.PI / 12)));
98     g.setColor(Color.green);
99     g.drawLine(xCenter, yCenter, xHour, yHour);
100 }
101
102 public void setCurrentTime() {
103     // Construct a calendar for the current date and time
104     Calendar calendar = new GregorianCalendar();
105
106     // Set current hour, minute and second
107     this.hour = calendar.get(Calendar.HOUR_OF_DAY);
108     this.minute = calendar.get(Calendar.MINUTE);
109     this.second = calendar.get(Calendar.SECOND);

```

```

110 }
111
112 public Dimension getPreferredSize() {
113     return new Dimension(200, 200);
114 }
115 }

```

程序可以使时钟的大小随着框架大小的改变而调整。每次改变框架的大小时，系统都会自动调用 `paintComponent` 方法来绘制一个新框架。`paintComponent` 方法按照面板宽度 (`getWidth()`) 和高度 (`getHeight()`) 的比例显示时钟 (Still Clock类中的第60~63行)。

15.11 显示图像

在12.10节中已经学习了如何创建图像图标，以及在标签和按钮上如何显示它们。例如，下面的语句创建一个图像图标，然后在标签上显示它：

```

ImageIcon ImageIcon = new ImageIcon("image/us.gif");
JLabel lblImage = new JLabel(ImageIcon);

```

图像图标显示一个尺寸固定的图像。为了显示大小灵活的图像，需要使用 `java.awt.Image` 类。可以使用 `getImage()` 方法从一个图像图标中创建一个图像，如下所示：

```
Image image = ImageIcon.getImage();
```

使用标签作为显示图像的区域比较简单、方便，但是你对如何显示图像没有多少控制权。更加灵活的显示图像的方式就是在面板上使用 `Graphics` 类的 `drawImage` 方法。`drawImage` 方法的四个版本如图15-23所示。

<i>java.awt.Graphics</i>
<code>+drawImage(image: Image, x: int, y: int, bgcolor: Color, observer: ImageObserver): void</code>
<code>+drawImage(image: Image, x: int, y: int, observer: ImageObserver): void</code>
<code>+drawImage(image: Image, x: int, y: int, width: int, height: int, observer: ImageObserver): void</code>
<code>+drawImage(image: Image, x: int, y: int, width: int, height: int, bgcolor: Color, observer: ImageObserver): void</code>

在特定位置绘制图像。图像的左上角是在图像环境的坐标空间中的点 (x, y) 处。图像中的透明像素可以用特定颜色 `bgcolor` 来绘制。`observer` 是指在其上显示图像的对象。如果图像大于要绘制它的面积则对它进行截取

除了不要指定背景色之外，其他都和前一个方法一样

绘制图像的一个可度量版本以将它填满指定矩形中所有的可用空间

它除了提供要绘制的图像后台的一个单色背景色之外，其他都和前一个方法一样

图15-23 可以应用 `Graphics` 对象上的 `drawImage` 方法以显示 GUI 组件中的图像

`ImageObserver` 表明在创建一个图像时，会指定一个 GUI 组件接收图像信息的通知。为了使用像 `JPanel` 的 Swing 组件中的 `drawImage` 方法绘制图像，需要覆盖 `paintComponent` 方法以告诉组件如何在面板内显示图像。

程序清单15-11给出显示来自 `image/us.gif` 的图像的代码。文件 `image/us.gif` (第20行) 在类目录下。在第21行获取一个 `Image` 对象。`drawImage` 方法显示填充整个面板的图像，如图15-24所示。

程序清单15-11 DisplayImage.java

```

1 import java.awt.*;
2 import javax.swing.*;
3
4 public class DisplayImage extends JFrame {
5     public DisplayImage() {
6         add(new JPanel());
7     }
8 }

```



图15-24 在面板内显示一个图像


```

9  public static void main(String[] args) {
10     JFrame frame = new DisplayImage();
11     frame.setTitle("DisplayImage");
12     frame.setSize(300, 300);
13     frame.setLocationRelativeTo(null); // Center the frame
14     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15     frame.setVisible(true);
16 }
17 }
18
19 class ImagePanel extends JPanel {
20     private ImageIcon imageIcon = new ImageIcon("image/us.gif");
21     private Image image = imageIcon.getImage();
22
23     /** Draw image on the panel */
24     protected void paintComponent(Graphics g) {
25         super.paintComponent(g);
26
27         if (image != null)
28             g.drawImage(image, 0, 0, getWidth(), getHeight(), this);
29     }
30 }

```

15.12 实例学习: ImageViewer类

显示图像是Java程序设计的一个常用任务。这个实例学习会开发一个名为ImageViewer的可重用组件,它在面板上显示一个图像。该类包括属性image、stretched、xCoordinate和yCoordinate,它们都有相关的访问器和修改器方法,如图15-25所示。

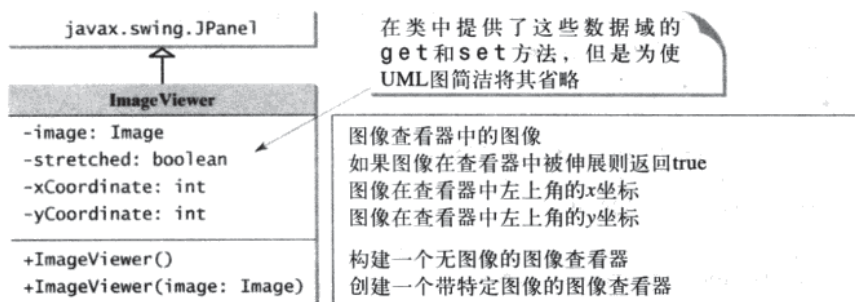


图15-25 ImageViewer类在面板上显示图像

可以使用像JLabel和JButton这样的Swing组件中的图像,但是这些图像是不可伸展的。在ImageViewer中的图像是可伸展的。

我们编写程序清单15-12中的测试程序,使用ImageViewer类显示六个图像。图15-26显示该程序的运行示例。



图15-26 在六个ImageViewer组件中显示六个图像

程序清单15-12 SixFlags.java

```

1 import javax.swing.*;
2 import java.awt.*;
3

```

```

4 public class SixFlags extends JFrame {
5     public SixFlags() {
6         Image image1 = new ImageIcon("image/us.gif").getImage();
7         Image image2 = new ImageIcon("image/ca.gif").getImage();
8         Image image3 = new ImageIcon("image/india.gif").getImage();
9         Image image4 = new ImageIcon("image/uk.gif").getImage();
10        Image image5 = new ImageIcon("image/china.gif").getImage();
11        Image image6 = new ImageIcon("image/norway.gif").getImage();
12
13        setLayout(new GridLayout(2, 0, 5, 5));
14        add(new ImageViewer(image1));
15        add(new ImageViewer(image2));
16        add(new ImageViewer(image3));
17        add(new ImageViewer(image4));
18        add(new ImageViewer(image5));
19        add(new ImageViewer(image6));
20    }
21
22    public static void main(String[] args) {
23        SixFlags frame = new SixFlags();
24        frame.setTitle("SixFlags");
25        frame.setSize(400, 320);
26        frame.setLocationRelativeTo(null); // Center the frame
27        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
28        frame.setVisible(true);
29    }
30 }

```

程序清单15-13实现ImageViewer类。(注意,可以跳过这个实现)。很容易实现属性image、stretched、xCoordinate和yCoordinate的访问器和修改器方法。paintComponent方法(第26~35行)在面板上显示图像。第29行在显示图像之前确保图像不为null。第30行检查图像是否是可伸展的。

程序清单15-13 ImageViewer.java

```

1 import java.awt.*;
2 import javax.swing.*;
3
4 public class ImageViewer extends JPanel {
5     /** Hold value of property image. */
6     private java.awt.Image image;
7
8     /** Hold value of property stretched. */
9     private boolean stretched = true;
10
11    /** Hold value of property xCoordinate. */
12    private int xCoordinate;
13
14    /** Hold value of property yCoordinate. */
15    private int yCoordinate;
16
17    /** Construct an empty image viewer */
18    public ImageViewer() {
19    }
20
21    /** Construct an image viewer for a specified Image object */
22    public ImageViewer(Image image) {
23        this.image = image;
24    }
25
26    protected void paintComponent(Graphics g) {
27        super.paintComponent(g);
28
29        if (image != null)
30            if (isStretched())
31                g.drawImage(image, xCoordinate, yCoordinate,
32                    getWidth(), getHeight(), this);

```

```
33     else
34         g.drawImage(image, xCoordinate, yCoordinate, this);
35     }
36
37     /** Return value of property image */
38     public java.awt.Image getImage() {
39         return image;
40     }
41
42     /** Set a new value for property image */
43     public void setImage(java.awt.Image image) {
44         this.image = image;
45         repaint();
46     }
47
48     /** Return value of property stretched */
49     public boolean isStretched() {
50         return stretched;
51     }
52
53     /** Set a new value for property stretched */
54     public void setStretched(boolean stretched) {
55         this.stretched = stretched;
56         repaint();
57     }
58
59     /** Return value of property xCoordinate */
60     public int getXCoordinate() {
61         return xCoordinate;
62     }
63
64     /** Set a new value for property xCoordinate */
65     public void setXCoordinate(int xCoordinate) {
66         this.xCoordinate = xCoordinate;
67         repaint();
68     }
69
70     /** Return value of property yCoordinate */
71     public int getYCoordinate() {
72         return yCoordinate;
73     }
74
75     /** Set a new value for property yCoordinate */
76     public void setYCoordinate(int yCoordinate) {
77         this.yCoordinate = yCoordinate;
78         repaint();
79     }
80 }
```

本章小结

- 每个组件都有自己的坐标系，原点 (0, 0) 在窗口的左上角，x坐标向右增加，y坐标向下增加。
- Graphics类是在不同平台的屏幕上显示图形和图像的抽象类。Graphics类在JVM本地平台上实现。使用paintComponent(g)方法在GUI组件上绘图时，这个g是特定平台的抽象Graphics类的具体子类的实例。Graphics类封装了平台细节并且可以在不考虑特定平台的情况下统一绘画。
- 为确保视图域在显示新图之前被清除，必须调用super.paintComponent(g)。用户可以通过调用定义在Component类中的repaint()方法请求再次显示组件。调用repaint()会引起JVM调用paintComponent。用户应该永远都不要直接调用paintComponent。因为这个原因，将paintComponent可见性定义为protected就足够了。

- 通常使用JPanel作为画布。为了在JPanel上绘图，创建新类扩展JPanel并且覆盖paintComponent方法以告诉面板如何绘画。
- 可以设置组件或所画物体的字体，使用字体尺寸测量字体的大小。字体和字体尺寸封装在Font类和FontMetrics类中。FontMetrics可以用来计算字符串的准确长度和宽度，这有助于测量字符串的大小，以便在正确的位置显示它。
- Component类中有setBackground、setForeground和setFont方法。这些方法都用来为整个组件设置颜色和字体。假如想用不同的颜色和字体在同一个面板中显示几条消息，必须使用Graphics类中的setColor和setFont方法设定当前绘画对象的颜色和字体。
- 为了显示一幅图像，首先创建一个图像图标，然后使用ImageIcon的getImage()方法来获取图像的一个Image对象，再使用java.awt.Graphics类中的drawImage方法绘制图像。

复习题

15.2~15.3节

- 15.1 假设在现有的消息下面绘制一条新消息，x和y的坐标应该增加还是减少？
- 15.2 为什么Graphics类是抽象的？如何创建一个Graphics对象？
- 15.3 描述paintComponent方法。它在哪里定义？它是如何调用的？它可以直接调用吗？程序如何调用这个方法？
- 15.4 为什么paintComponent方法是protected的？如果在子类中将它改为public或private，会发生什么？为什么在程序清单15-1中第21行调用super.paintComponent(g)，在程序清单15-3中第31行调用super.paintComponent(g)？
- 15.5 可以在任意Swing GUI组件上进行绘画吗？为什么绘画时要将面板作为画布而不是标签或按钮？

15.4~15.7节

- 15.6 描述绘制字符串、直线、矩形、圆角矩形、3D矩形、椭圆、弧形、多边形和折线段的方法。
- 15.7 描述填充矩形、圆角矩形、椭圆、弧线和多边形的方法。
- 15.8 如何在Graphics对象中获取和设置颜色和字体？
- 15.9 写出绘制下面图形的语句：
- 绘制一条从点(10, 10)到(70, 30)的粗直线。可以画几条相互紧挨着的线以创造出一条粗线的效果。
 - 绘制/填充一个左上角在点(10, 10)、宽为100而高为50的矩形。
 - 绘制/填充一个宽为100而高为200的圆角矩形。角的水平直径为40，垂直直径为20。
 - 绘制/填充一个半径为30的圆。
 - 绘制/填充一个宽为50而高为100的椭圆。
 - 绘制一个半径为50的上半圆。
 - 绘制/填充一个由下面几个点构成的多边形：(20, 40)、(30, 50)、(40, 90)、(90, 10)、(10, 30)。

15.8~15.10节

- 15.10 如何找出字体的行距、上升线、下沉线以及高？如何找出Graphics对象中以像素为单位的字符串的准确长度？
- 15.11 如果在程序清单15-8中的第8行没有初始化message，那么当使用它的无参构造方法创建MessagePanel时会发生什么？
- 15.12 下面的程序是要在面板上显示一条消息，但是却什么都没有显示。在第2行和第14行都有问题，纠正这些错误。


```

1 public class TestDrawMessage extends javax.swing.JFrame {
2     public void TestDrawMessage() {
3         add(new DrawMessage());
4     }
5
6     public static void main(String[] args) {
7         javax.swing.JFrame frame = new TestDrawMessage();
8         frame.setSize(100, 200);
9         frame.setVisible(true);
10    }
11 }
12
13 class DrawMessage extends javax.swing.JPanel {
14     protected void PaintComponent(java.awt.Graphics g) {
15         super.paintComponent(g);
16         g.drawString("Welcome to Java", 20, 20);
17     }
18 }

```

15.11~15.12节

15.13 如何由ImageIcon对象创建一个Image对象?

15.14 如何由Image对象创建一个ImageIcon对象?

15.15 描述Graphics类中的drawImage方法。

15.16 解释在JLabel和JPanel中显示图像的不同之处。

15.17 哪个包会包括ImageIcon, 哪个包会包括Image?

编程练习题

15.2~15.7节

*15.1 (显示一个3×3的网格) 编写程序, 显示一个3×3的网格, 如图15-27a所示。使用红色画垂直线, 使用蓝色画水平线。

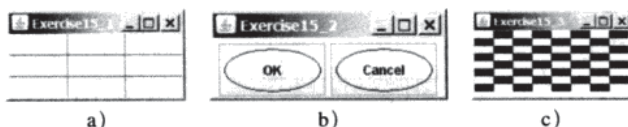


图15-27 a) 练习题15.1显示一个网格; b) 练习题15.2显示两个OvalButton对象;
c) 练习题15.3显示一个棋盘

**15.2 (创建一个自定义的按钮类) 扩展JButton类, 开发一个名为OvalButton的自定义按钮类, 将按钮上的文本显示在椭圆中。图15-27b显示使用OvalButton类创建的两个按钮。

*15.3 (显示一个棋盘) 练习题12.10显示一个棋盘, 每一个黑格和白格都是一个JButton。改写程序, 使用Graphics类中的绘图方法, 在JPanel上绘制一个棋盘, 如图15-27c所示。

*15.4 (显示一个乘法表) 编写程序, 使用绘图方法在面板中显示一个乘法表, 如图15-28a所示。

	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

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10 11

图15-28 a) 练习题15.4显示一个乘法表; b) 练习题15.5将数字显示为一个三角形形式

****15.5** (用三角形显示数字) 编写程序, 将数字显示成三角形形式, 如图15-28b所示。改变窗口大小时, 为了适应窗口, 行数会随窗口的大小而变化。

****15.6** (改进FigurePanel) 程序清单15-3中的FigurePanel类可以显示直线、矩形、圆角矩形和椭圆。在类中添加合适的新代码来显示弧形和多边形。使用新的FigurePanel类编写测试程序, 显示如图15-29a所示的图形。

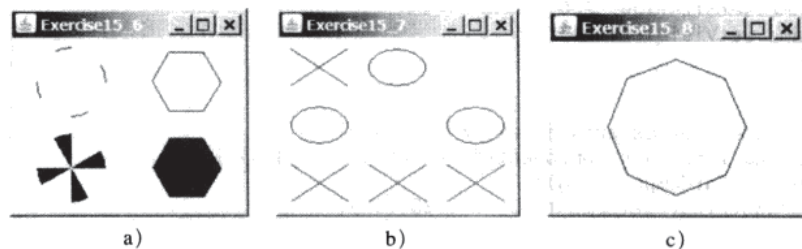


图15-29 a) 在GridLayout框架中显示四种几何图案的面板;

b) 在井字游戏的棋格随机显示X、O或者空白; c) 练习题15.8绘制一个八边形

****15.7** (显示一个井字游戏的棋盘) 创建一个自定义面板, 它可以显示X、O或者空白。显示什么是重画面板时随机决定的。使用Math.random()方法产生整数0、1或2, 对应于面板上显示X、O或者空白。创建一个包含九个自定义面板的框架, 如图15-29b所示。

****15.8** (绘制一个八边形) 编写一个绘制八边形的程序, 如图15-29c所示。

***15.9** (创建四个扇形图) 编写程序, 使用两行两列的GridLayout框架放置四个扇形图, 如图15-30a所示。

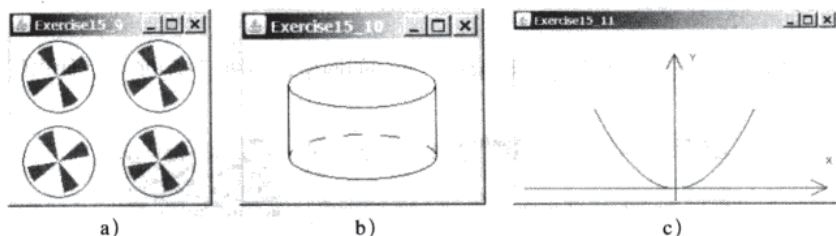


图15-30 a) 练习题15.9绘制四个扇形; b) 练习题15.10绘制一个圆柱体;

c) 练习题15.11绘制函数 $f(x) = x^2$ 的图形

***15.10** (创建一个圆柱体) 编写程序, 绘制一个圆柱体, 如图15-30b所示。

****15.11** (绘图表示平方的函数) 编写程序, 绘制函数 $f(x) = x^2$ 的图形 (参见图15-30c)。

提示 使用下面的循环在多边形p中加入点:

```
double scaleFactor = 0.1;
```

```
for (int x = -100; x <= 100; x++) {
    p.addPoint(x + 200, 200 - (int)(scaleFactor * x * x));
}
```

对Graphics对象g使用g.drawPolyline(p.xpoints, p.ypoints, p.npoints)来连接点。p.xpoints返回x坐标的数组, p.ypoints返回y坐标的数组, p.npoints返回Polygon对象p的点的个数。

****15.12** (画出正弦函数的图形) 编写一个程序, 画出正弦函数的图形, 如图15-31a所示。

提示 π 的统一码是\u03c0。要显示 -2π , 使用g.drawString("-2\u03c0", x, y)。对于像 $\sin(x)$ 的三角函数, x是弧度。使用下面的循环将点添加到多边形p中:

```

for (int x = -100; x <= 100; x++) {
    p.addPoint(x + 200,
        100 - (int)(50 * Math.sin((x / 100.0) * 2 * Math.PI)));
}

```

-2π 位于点(100, 100)处, 坐标轴的中心位于点(200, 100)处, 而 2π 位于点(300, 100)处。使用Graphics类中的drawPolyline方法来连接这些点。

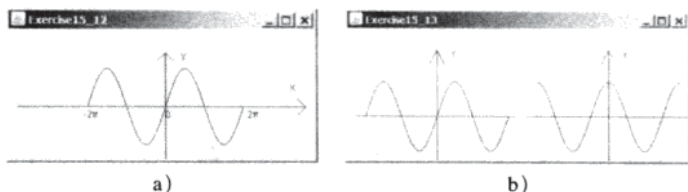


图15-31 a) 练习题15.12绘制函数 $f(x) = \sin(x)$ 的图形; b) 练习题15.13绘制正弦和余弦函数

****15.13** (使用抽象方法绘制函数图形) 编写一个抽象类, 绘制函数图形。这个类的定义如下:

```

public abstract class AbstractDrawFunction extends JPanel {
    /** Polygon to hold the points */
    private Polygon p = new Polygon();

    protected AbstractDrawFunction () {
        drawFunction();
    }

    /** Return the y-coordinate */
    abstract double f(double x);

    /** Obtain points for x-coordinates 100, 101, ..., 300 */
    public void drawFunction() {
        for (int x = -100; x <= 100; x++) {
            p.addPoint(x + 200, 200 - (int)f(x));
        }
    }

    /** Implement paintComponent to draw axes, labels, and
     *  connecting points
     */
    protected void paintComponent(Graphics g) {
        // To be completed by you
    }
}

```

使用下面的函数测试这个类:

```

f(x) = x2;
f(x) = sin(x);
f(x) = cos(x);
f(x) = tan(x);
f(x) = cos(x) + 5sin(x);
f(x) = 5cos(x) + sin(x);
f(x) = log(x) + x2;

```

对每个函数, 扩展AbstractDrawFunction类来创建一个新类, 并实现f方法。图15-31b显示所绘出的正弦函数和余弦函数的图形。

****15.14** (显示一个条形图) 编写程序, 使用条形图显示作业、平时测验、期中考试和期末考试占总成绩的百分比, 如图15-1a所示。假设作业占20%用红色显示, 平时测验占10%用蓝色显示, 期中考试占30%用绿色显示, 期末考试占40%用橙色显示。

****15.15** (显示一个饼图) 编写程序, 使用饼图显示作业、平时测验、期中考试和期末考试占总成绩的

百分比,如图15-32a所示。假设作业占20%用红色显示,平时测验占10%用蓝色显示,期中考试占30%用绿色显示,期末考试占40%用橙色显示。

15.16 (获得字体信息) 编写程序,在面板上显示消息“Java is fun”。将面板的字体设为TimesRoman、粗体、20像素。将字体的行距、上升线、下沉线、高度和字符串宽度显示为面板的工具提示文本,如图15-32b所示。

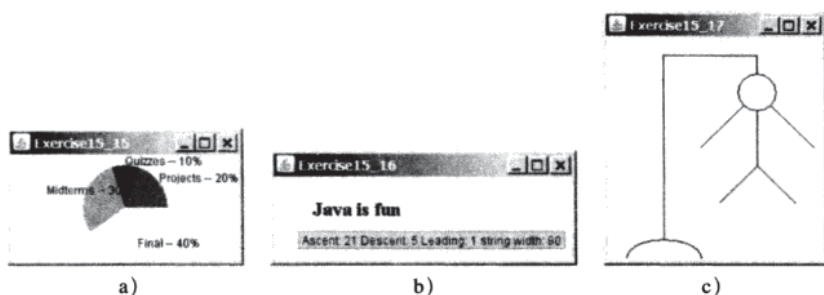


图15-32 a) 练习题15.15使用饼图显示作业、平时测验、期中考试和期末考试占总成绩的百分比;

b) 练习题15.16在工具提示文本中显示字体属性; c) 练习题15.17绘制出剑子手游戏的框架

15.17 (游戏: 剑子手) 编写程序显示流行的剑子手游戏的图像,如图15-32c所示。

15.18 (使用StillClock类) 编写程序显示两个时钟。第一个时钟的小时、分钟和秒值为4、20、45,而第二个时钟的小时、分钟和秒值为22、46、15,如图15-33a所示。

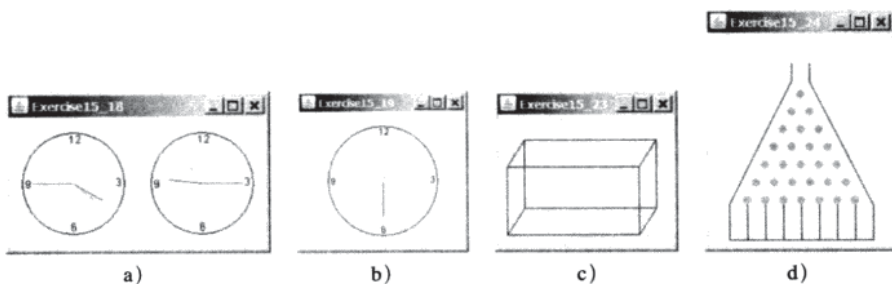


图15-33 a) 练习题15.18显示两个时钟; b) 练习题15.19显示一个任意小时和分钟值的时钟;

c) 练习题15.23显示一个长方体; d) 练习题15.24模拟一个豆机

*15.19 (任意时间) 用三个新的Boolean属性hourHandVisible、minuteHandVisible和secondHandVisible以及和它们相关的访问器和修改器方法修改StillClock类。可以使用set方法指定指针为可见的或者不可见的。编写测试程序只显示小时指针和分钟指针。小时和分钟的值是随机产生的。小时值在0到11之间,而分钟值是0或者30,如图15-33b所示。

**15.20 (画一个精细的时钟) 修改15.12节中的StillClock类,绘制一个将小时和分钟分得更精细的时钟,如图15-1b所示。

**15.21 (显示一个带图像的井字游戏棋盘) 改写练习题12.7,在JPanel上显示图像而不是在JLabel上显示图像图标。

**15.22 (显示STOP信号) 编写程序显示STOP信号,如图15-1c所示。八边形是红色的而信号是白色的。

提示 参见程序清单15-5以及程序清单15-6。

15.23 (显示一个长方体) 编写一个程序,显示一个长方体,如图15-33c所示。这个立方体可以随着框架的增加和减小而扩张或者收缩。

**15.24 (游戏: 豆机) 编写一个程序,显示练习题6.21中介绍过的豆机。豆机应该放在可改变大小的面

板的中央,如图15-33d所示。

****15.25** (几何方面:显示一个正 n 边形) 创建一个名为`RegularPolygonPanel`的`JPanel`的子类,来绘制一个 n 条边的正多边形。这个类包括了名为`numberOfSides`的属性,它表示多边形边的个数。多边形放在面板的中心位置。多边形大小和面板的大小成比例。从`RegularPolygonPanel`创建一个五边形、六边形、七边形、八边形、九边形和十边形,然后在框架中显示它们,如图15-34a所示。

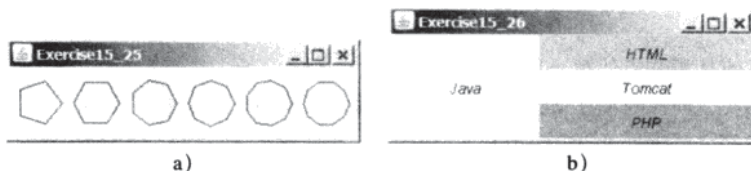


图15-34 a) 练习题15.25显示几个 n 边形; b) 练习题15.26使用`MessagePanel`显示4个字符串

15.26 (使用`MessagePanel`类) 编写程序显示四条消息,如图15-35b所示。

****15.27** (显示一个图形) 一个图形包括一些端点以及连接这些点的线。编写程序从文件中读取一个图形,然后在面板上显示它。文件的第一行包括表明端点个数的数字(n)。这些点都被标上 $0, 1, \dots, n-1$ 。每一个连接线的形式都是 $u \ x \ y \ v_1, v_2, \dots$, 这种形式描述点 u 的位置在 (x, y) 处并且有和它相连的两条线 (u, v_1) 、 (u, v_2) 等。图15-35a给出一个图形的文件的例子,然后在面板上显示这个图形,如图15-35b所示。

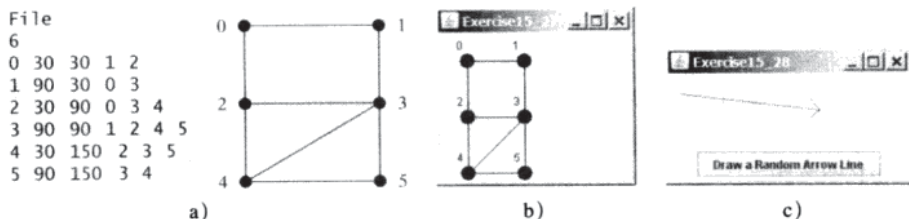


图15-35 a)~b) 程序读取关于图形的信息,然后显示它; c) 程序显示一条带箭头的直线

****15.28** (绘制一条带箭头的直线) 编写一个静态方法,使用下面的方法头绘制一条从起点到终点的带箭头的直线:

```
public static void drawArrowLine(int x1, int y1,
    int x2, int y2, Graphics g)
```

编写一个测试程序,当点击Draw Random Arrow Line按钮时,程序会随机绘制一条带箭头的直线,如图15-35c所示。

资源
程序
代码
PDG

第16章

Introduction to Java Programming, 8E

事件驱动程序设计

学习目标

- 描述事件、事件源和事件类（16.2节）。
- 定义监听器类、向源对象注册监听器对象，然后编写代码来处理事件（16.3节）。
- 使用内部类定义监听器类（16.4节）。
- 使用匿名内部类定义监听器类（16.5节）。
- 探究创建和注册监听器的各种编码风格（16.6节）。
- 点击按钮从文本域获取输入（16.7节）。
- 编写程序处理WindowEvent（16.8节）。
- 使用监听器接口适配器简化监听器类的代码（16.9节）。
- 编写程序处理鼠标事件MouseEvent（16.10节）。
- 编写程序处理键盘事件KeyEvent（16.11节）。
- 使用`javax.swing.Timer`类控制动画（16.12节）。

16.1 引言

假如希望编写一个GUI程序，提示用户输入贷款总额、年利率和年数，然后点击Compute Loan按钮获取月偿还额和总偿还额，如图16-1a所示。如何完成这个任务呢？必须使用事件驱动程序设计来编写代码以响应点击按钮事件。

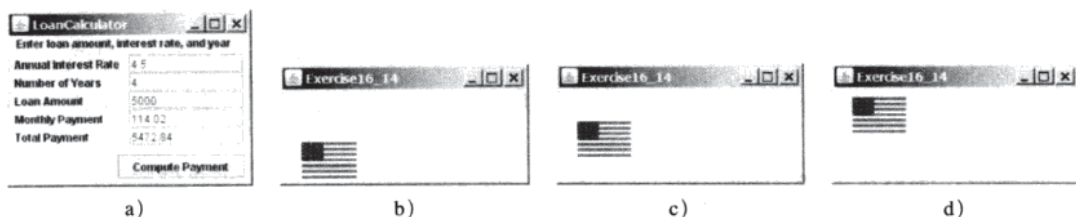


图16-1 a) 程序计算贷款偿还额；b)~d) 一面旗向上升起

假设希望编写程序用动画实现一面旗上升，如图16-1b~图16-1d所示。该如何完成这个任务呢？解决这个问题的方式有好几种。一种有效的方式就是在事件驱动程序设计中使用一个定时器，这也是本章的主题。

在事件驱动程序设计中，当事件发生时（例如，点击按钮、移动鼠标或定时器）执行代码。14.6节中尝试了一下事件驱动程序设计。你可能会很多问题，例如，为什么一个定义监听器类来实现ActionListener接口。本章将给你所有问题的答案。

16.2 事件和事件源

运行Java图形用户界面程序时，程序与用户进行交互，事件驱动程序的执行。事件（event）可以定义为程序发生了某些事情的信号。外部用户动作和内部程序动作都可以触发事件，外部用户动作的例子有移动鼠标、点击按钮和敲击键盘等，而内部程序动作的例子有定时器。程序可以选择响应事件或忽略

事件。

能创建一个事件并触发该事件的组件称为源对象 (source object) 或源组件 (source component)。例如，按钮是按钮点击动作事件的源对象。一个事件是事件类的实例。事件类的根类是 `java.util.EventObject`。一些事件类的层次关系如图16-2所示。

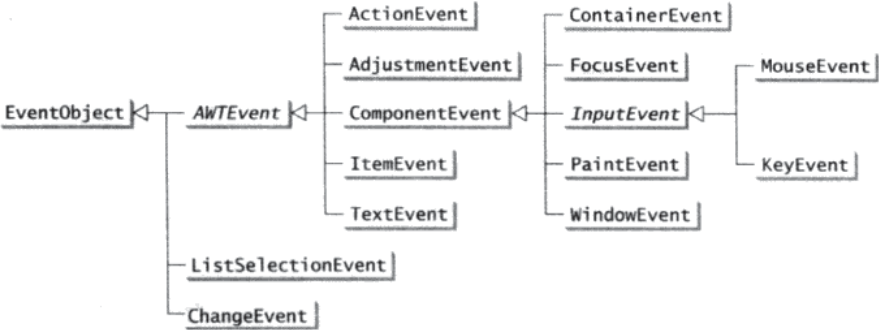


图16-2 事件是EventObject类的对象

事件对象包含与事件相关的一切属性。可以使用EventObject类中的实例方法getSource()获得事件的源对象。EventObject类的子类处理特定类型的事件，例如，动作事件、窗口事件、组件事件、鼠标事件以及按键事件。表16-1列出了外部用户动作、源对象和触发的事件类型。

表16-1 用户动作、源对象和事件类型

用户动作	源对象	触发的事件类型
点击按钮	JButton	ActionEvent
在文本域按回车键	JTextField	ActionEvent
选定一个新项	JComboBox	ItemEvent、ActionEvent
选定(多)项	JList	ListSelectionEvent
点击复选框	JCheckBox	ItemEvent、ActionEvent
点击单选按钮	JRadioButton	ItemEvent、ActionEvent
选定菜单项	JMenuItem	ActionEvent
移动滚动条	JScrollBar	AdjustmentEvent
移动滚动条	JSlider	ChangeEvent
窗口打开、关闭、最小化、还原或关闭中	Window	WindowEvent
按住、释放、点击、回车或退出鼠标	Component	MouseEvent
移动或拖动鼠标	Component	MouseEvent
释放或按下键盘上的键回车或退出	Component	KeyEvent
从容器中添加或删除组件	Container	ContainerEvent
组件移动、改变大小、隐藏或显示	Component	ComponentEvent
组件获取或失去焦点	Component	FocusEvent

注意 如果一个组件可以触发某个事件，那么这个组件的任意子类都可以触发同类型的事件。例如，每个GUI组件都可以触发MouseEvent、KeyEvent、FocusEvent和ComponentEvent，因为Component是所有GUI组件的父类。

注意 图16-2中除了ListSelectionEvent和ChangeEvent之外的所有事件类都包括在java.awt.event包中，ListSelectionEvent和ChangeEvent在javax.swing.event包中。AWT事件本来是为AWT组件设计的，但是许多Swing组件都会触发它们。

16.3 监听器、注册以及处理事件

Java使用一种基于委托的模型来处理事件：源对象触发一个事件，对此事件感兴趣的对象会处理它。将对此事件感兴趣的对象称为监听器（listener）。一个对象要成为源对象上的事件监听器，需要具备两个条件，如图16-3所示。

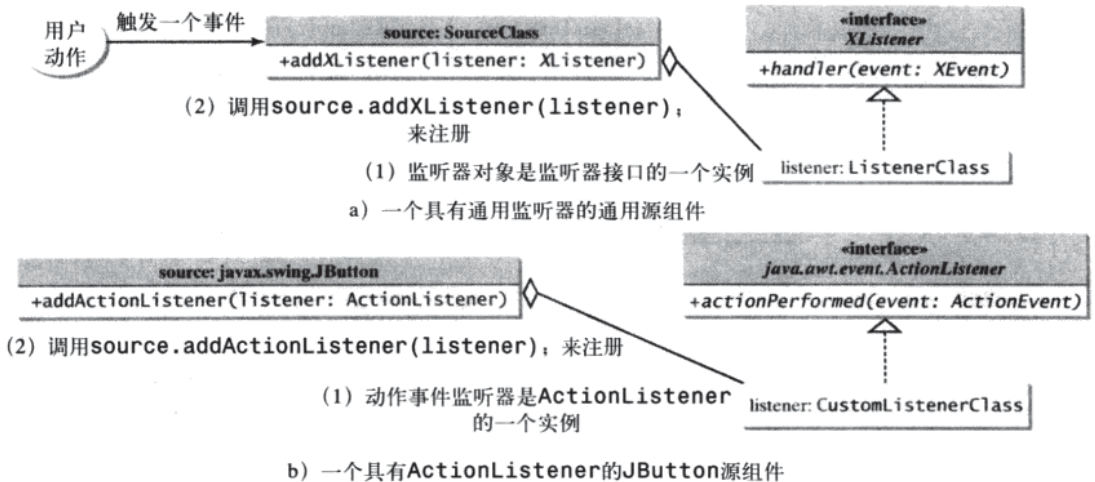


图16-3 监听器必须是一个监听器接口的实例，并且必须注册到源组件

1) 监听器对象的类必须是相应的事件监听器接口的实例，以确保监听器有处理这个事件的正确方法。Java为每一种类型的事件都提供了监听器接口。通常，事件XEvent的监听器接口命名为XListener，监听器MouseEvent例外。例如，事件ActionEvent对应的监听器接口是ActionListener，ActionEvent的每个监听器都应该实现ActionListener接口。表16-2列出事件类型、对应的监听器接口以及定义在监听器接口中的方法。包含处理事件的监听器接口称为处理器（handler）的方法。

2) 监听器对象必须由源对象注册。注册方法依赖于事件的类型。如果事件类型是ActionEvent，那么对应的注册方法是addActionListener。一般来说，XEvent的注册方法命名为addXListener。一个源对象可以触发几种类型的事件。一个源对象针对每个事件都维护着一个注册的监听器列表，通过调用监听器对象上的处理器来通知所有注册的监听器响应这个事件，如图16-4所示（图16-4显示源类的内部实现。为了使用它，不需要知道像JButton这样的源类是如何实现的。然而，这些知识将有助于理解Java事件驱动程序设计的框架结构）。

我们重新看看程序清单14-8。因为JButton对象触发ActionEvent，ActionEvent的监听器对象必须是ActionListener的实例，所以监听器类在第34行实现ActionListener。源对象调用addActionListener(listener)来注册一个监听器，如下所示：

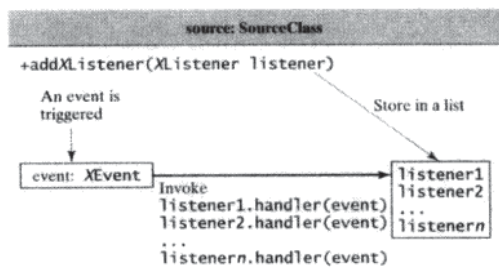
```
JButton jbtOK = new JButton("OK"); // Line 7 in Listing 14.8
ActionListener listener1
    = new OKListenerClass(); // Line 18 in Listing 14.8
jbtOK.addActionListener(listener1); // Line 20 in Listing 14.8
```

点击该按钮时，JButton对象触发一个ActionEvent，然后将它作为参数传递以调用监听器的actionPerformed方法来处理这个事件。

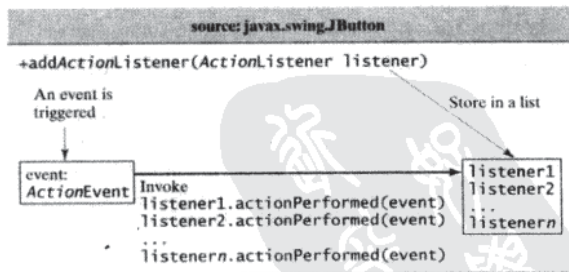
事件对象包含与事件相关的信息，这些可以使用如图16-5所示的方法获得。例如，为判断一个对象是按钮、复选框还是单选按钮，都可以使用e.getSource()方法获得源对象。对于一个动作事件，可以通过使用e.getWhen()方法获得该事件的发生时间。

表16-2 事件、事件监听器和监听器方法

事件类（处理器）	监听器接口	监听器方法
ActionEvent	ActionListener	actionPerformed(ActionEvent)
ItemEvent	ItemListener	itemStateChanged(ItemEvent)
MouseEvent	MouseListener	mousePressed(MouseEvent) mouseReleased(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mouseClicked(MouseEvent)
	MouseMotionListener	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
KeyEvent	KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
WindowEvent	WindowListener	windowClosing(WindowEvent) windowOpened(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent)
ContainerEvent	ContainerListener	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
ComponentEvent	ComponentListener	componentMoved(ComponentEvent) componentHidden(ComponentEvent) componentResized(ComponentEvent) componentShown(ComponentEvent)
FocusEvent	FocusListener	focusGained(FocusEvent) focusLost(FocusEvent)
AdjustmentEvent	AdjustmentListener	adjustmentValueChanged(AdjustmentEvent)
ChangeEvent	ChangeListener	stateChanged(ChangeEvent)
ListSelectionEvent	ListSelectionListener	valueChanged(ListSelectionEvent)



a) 通用源对象的内部函数



b) JButton对象的内部函数

图16-4 源对象通过调用监听器对象的处理器通知事件监听器

现在，我们可以编写一个程序，使用两个按钮控制一个圆的大小，如图16-6所示。

我们将逐步开发这个程序。首先，编写程序清单16-1中的程序，显示一个圆在中央（第14行）以及两个按钮在底部（第15行）的用户接口。

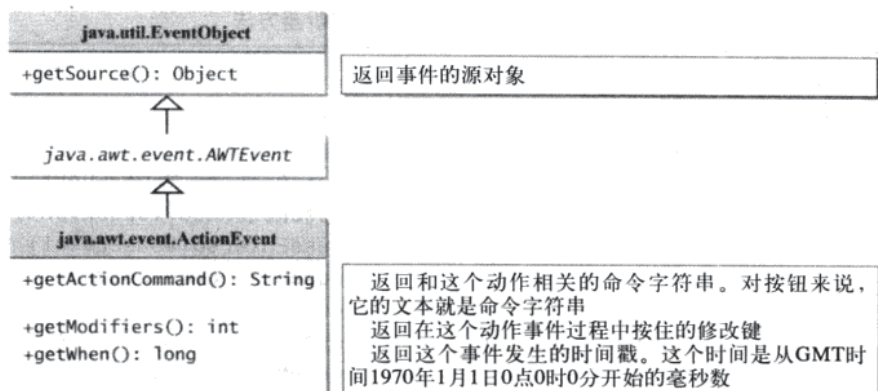


图16-5 可以从一个事件对象获取有用的信息

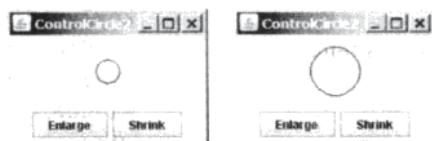


图16-6 用户点击Enlarge和Shrink按钮来放大和缩小圆的尺寸

程序清单16-1 ControlCircle1.java

```

1 import javax.swing.*;
2 import java.awt.*;
3
4 public class ControlCircle1 extends JFrame {
5     private JButton jbtEnlarge = new JButton("Enlarge");
6     private JButton jbtShrink = new JButton("Shrink");
7     private CirclePanel canvas = new CirclePanel();
8
9     public ControlCircle1() {
10         JPanel panel = new JPanel(); // Use the panel to group buttons
11         panel.add(jbtEnlarge);
12         panel.add(jbtShrink);
13
14         this.add(canvas, BorderLayout.CENTER); // Add canvas to center
15         this.add(panel, BorderLayout.SOUTH); // Add buttons to the frame
16     }
17
18     /** Main method */
19     public static void main(String[] args) {
20         JFrame frame = new ControlCircle1();
21         frame.setTitle("ControlCircle1");
22         frame.setLocationRelativeTo(null); // Center the frame
23         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24         frame.setSize(200, 200);
25         frame.setVisible(true);
26     }
27 }
28
29 class CirclePanel extends JPanel {
30     private int radius = 5; // Default circle radius
31
32     /** Repaint the circle */
33     protected void paintComponent(Graphics g) {
34         super.paintComponent(g);
35         g.drawOval(getWidth() / 2 - radius, getHeight() / 2 - radius,
36                 2 * radius, 2 * radius);
37     }
38 }
  
```

如何使用按钮放大和缩小这个圆呢？当点击Enlarge按钮时，希望能用一个比较大的半径来重新绘制这个圆。该如何完成它呢？可以用下面的特征来将程序清单16-1中的程序扩展到程序清单16-2中：

- 1) 定义一个名为EnlargeListener的监听器类，实现ActionListener（第31~35行）。
- 2) 创建一个监听器，并且将它注册到jbtEnlarge（第18行）。
- 3) 在CirclePanel中添加一个名为enlarge()的方法来增加半径，然后重新绘制面板（第41~44行）。
- 4) 实现EnlargeListener中的actionPerformed方法来调用canvas.enlarge()（第33行）。
- 5) 为了让actionPerformed方法可以访问引用变量canvas，将EnlargeListener定义为ControlCircle2类的内部类（第31~35行）。内部类定义在另一个类中。我们将在16.4节介绍内部类。
- 6) 为了避免编译错误，CirclePanel类（第37~52行）现在也定义为ControlCircle2的一个内部类，因为旧的CirclePanel类已经在程序清单16-1中定义。

程序清单16-2 ControlCircle2.java

```

1 import javax.swing.*;
2 import java.awt.*;
3 import java.awt.event.*;
4
5 public class ControlCircle2 extends JFrame {
6     private JButton jbtEnlarge = new JButton("Enlarge");
7     private JButton jbtShrink = new JButton("Shrink");
8     private CirclePanel canvas = new CirclePanel();
9
10    public ControlCircle2() {
11        JPanel panel = new JPanel(); // Use the panel to group buttons
12        panel.add(jbtEnlarge);
13        panel.add(jbtShrink);
14
15        this.add(canvas, BorderLayout.CENTER); // Add canvas to center
16        this.add(panel, BorderLayout.SOUTH); // Add buttons to the frame
17
18        jbtEnlarge.addActionListener(new EnlargeListener());
19    }
20
21    /** Main method */
22    public static void main(String[] args) {
23        JFrame frame = new ControlCircle2();
24        frame.setTitle("ControlCircle2");
25        frame.setLocationRelativeTo(null); // Center the frame
26        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
27        frame.setSize(200, 200);
28        frame.setVisible(true);
29    }
30
31    class EnlargeListener implements ActionListener { // Inner class
32        public void actionPerformed(ActionEvent e) {
33            canvas.enlarge();
34        }
35    }
36
37    class CirclePanel extends JPanel { // Inner class
38        private int radius = 5; // Default circle radius
39
40        /** Enlarge the circle */
41        public void enlarge() {
42            radius++;
43            repaint();
44        }
45
46        /** Repaint the circle */
47        protected void paintComponent(Graphics g) {
48            super.paintComponent(g);

```

```

49      g.drawOval(getWidth() / 2 - radius, getHeight() / 2 - radius,
50                2 * radius, 2 * radius);
51    }
52  }
53 }

```

同样地，可以添加Shrink按钮的代码，当点击Shrink按钮时，显示比较小的圆。

16.4 内部类

内部类 (inner class) 或者嵌套类 (nested class) 是定义在另一个类的范围内的类。图16-7a中的代码定义了两个独立的类：Test和A。图16-7b中的代码定义A为Test的一个内部类。

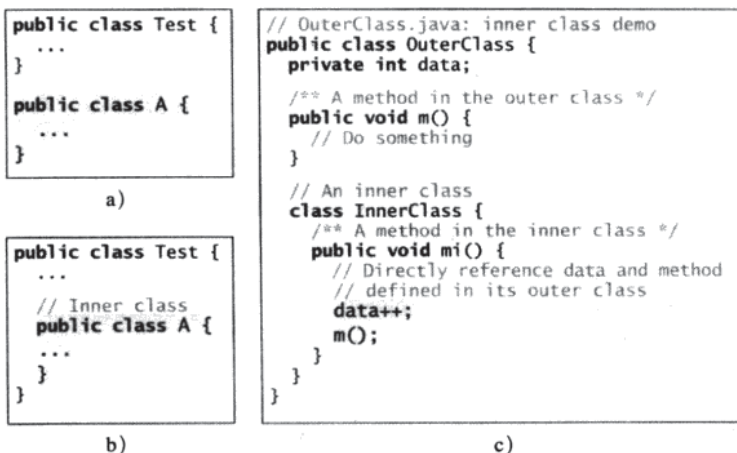


图16-7 内部类结合相关类成为主类

图16-7c中定义在OuterClass中的InnerClass类是内部类的另一个例子。内部类可以像常规类一样使用。通常，如果内部类只是被外部类使用，那就将该类定义为内部类。一个内部类有如下特征：

- 一个内部类被编译成一个名为OuterClassName\$InnerClassName.class的类。例如，在图16-7b中Test中的内部类A被编译为Test\$A.class。
- 内部类可以引用定义在它嵌套的外部类中的数据和方法，所以，不需要将外部类对象的引用传递给内部类的构造方法。因为这个原因，内部类可以使程序更加简单和简洁。
- 使用可见性修饰符定义内部类时，遵从和应用与在类成员上一样的可见性规则。
- 可以将内部类定义为static。一个static内部类可以使用外部类的名字访问。一个static类是不能访问外部类的非静态成员的。
- 内部类的对象经常在外类中创建。但是，也可以从另一个类中创建一个内部类的对象。如果该内部类是非静态的，就必须先创建一个外部类的实例，然后使用下面的语法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，那么使用下面的语法为它创建一个对象：

```
OuterClass.InnerClass innerObject = new OuterClass.InnerClass();
```

内部类的一个简单应用就是将附属类合并到主类中。这可以减少源文件的数量。它也可以使类文件更易于组织，因为所有的类文件都是以主类作为前缀命名的。例如，不是创建图16-7a的两个源文件Test.java和A.java，而是将类A合并到类Test中，只创建一个源文件Test.java，如图16-7b所示。最终的类文件是Test.class和Test\$A.class。

内部类的另一个实际应用就是为了避免类命名冲突。在程序清单16-1和程序清单16-2中定义了两个版本的CirclePanel。可以将它们定义为内部类，以避免冲突。

16.5 匿名类监听器

监听器类是特意为创建一个GUI组件（例如，一个按钮）而设计的监听器对象。监听器类不被其他应用程序所共享，因此，正确的做法是将其作为一个内部类定义在框架类中。

可以使用匿名内部类简化内部类监听器。匿名内部类（anonymous inner class）是没有名字的内部类。它一步完成定义内部类和创建一个该类的实例。程序清单16-2的内部类可以用匿名内部类替换，如下所示：

```
public ControlCircle2() {
    // Omitted

    jbtEnlarge.addActionListener(
        new EnlargeListener());
}

class EnlargeListener
    implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        canvas.enlarge();
    }
}
```

a) 内部类EnlargeListener

```
public ControlCircle2() {
    // Omitted

    jbtEnlarge.addActionListener(
        new class EnlargeListener
            implements ActionListener() {
        public void
            actionPerformed(ActionEvent e) {
                canvas.enlarge();
            }
        });
}
```

b) 匿名内部类

匿名内部类的语法如下：

```
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的内部类，所以，可以将它看做有以下特征的内部类：

- 匿名内部类必须总是扩展父类或实现接口，但是它不能有显式的extends或者implements子句。
- 匿名内部类必须实现父类或接口中所有的抽象方法。
- 匿名内部类总是使用它的父类的无参数构造方法来创建实例。如果匿名内部类实现了接口，构造方法就是Object()。
- 匿名内部类被编译为一个名为OuterClassName\$.class的类。例如，如果外部类Test有两个匿名内部类，那么它们就被编译为Test\$1.class和Test\$2.class。

程序清单16-3给出处理来自四个按钮的事件的例子，如图16-8所示。

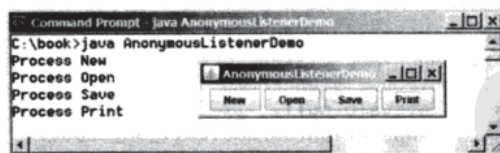


图16-8 程序处理来自四个按钮的事件

程序清单16-3 AnonymousListenerDemo.java

```
1 import javax.swing.*;
2 import java.awt.event.*;
3
4 public class AnonymousListenerDemo extends JFrame {
5     public AnonymousListenerDemo() {
6         // Create four buttons
7         JButton jbtNew = new JButton("New");
8         JButton jbtOpen = new JButton("Open");
9         JButton jbtSave = new JButton("Save");
10        JButton jbtPrint = new JButton("Print");
11    }
```

```

12 // Create a panel to hold buttons
13 JPanel panel = new JPanel();
14 panel.add(jbtNew);
15 panel.add(jbtOpen);
16 panel.add(jbtSave);
17 panel.add(jbtPrint);
18
19 add(panel);
20
21 // Create and register anonymous inner-class listener
22 jbtNew.addActionListener(
23     new ActionListener() {
24         public void actionPerformed(ActionEvent e) {
25             System.out.println("Process New");
26         }
27     }
28 );
29
30 jbtOpen.addActionListener(
31     new ActionListener() {
32         public void actionPerformed(ActionEvent e) {
33             System.out.println("Process Open");
34         }
35     }
36 );
37
38 jbtSave.addActionListener(
39     new ActionListener() {
40         public void actionPerformed(ActionEvent e) {
41             System.out.println("Process Save");
42         }
43     }
44 );
45
46 jbtPrint.addActionListener(
47     new ActionListener() {
48         public void actionPerformed(ActionEvent e) {
49             System.out.println("Process Print");
50         }
51     }
52 );
53 }
54
55 /** Main method */
56 public static void main(String[] args) {
57     JFrame frame = new AnonymousListenerDemo();
58     frame.setTitle("AnonymousListenerDemo");
59     frame.setLocationRelativeTo(null); // Center the frame
60     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
61     frame.pack();
62     frame.setVisible(true);
63 }
64 }

```

程序使用匿名内部类创建四个监听器（第22~52行）。不使用匿名内部类，就必须创建四个独立的类。匿名监听器和内部类监听器的工作方式是一样的。使用匿名内部类压缩这个程序。

匿名内部类被编译为OuterClassName\$#.class，这里的#是从1开始的，每次编译器遇到匿名类它就会自增1。在这个例子中，匿名内部类就被编译为AnonymousListenerDemo\$1.class、AnonymousListenerDemo\$2.class、AnonymousListenerDemo\$3.class和AnonymousListenerDemo\$4.class。

程序没有使用setSize方法来设置框架的大小，而是使用pack()方法（第61行），该方法会依据放在框架中组件的大小来自动调整框架的大小。

16.6 定义监听器类的另一种方式

定义监听器类有很多其他方式。例如，可以改写程序清单16-3，新程序只创建一个监听器，将这个监听器注册给按钮，然后让监听器检测出事件源，即哪个按钮触发了这个事件，如程序清单16-4所示。

程序清单16-4 DetectSourceDemo.java

```
1 import javax.swing.*;
2 import java.awt.event.*;
3
4 public class DetectSourceDemo extends JFrame {
5     // Create four buttons
6     private JButton jbtNew = new JButton("New");
7     private JButton jbtOpen = new JButton("Open");
8     private JButton jbtSave = new JButton("Save");
9     private JButton jbtPrint = new JButton("Print");
10
11     public DetectSourceDemo() {
12         // Create a panel to hold buttons
13         JPanel panel = new JPanel();
14         panel.add(jbtNew);
15         panel.add(jbtOpen);
16         panel.add(jbtSave);
17         panel.add(jbtPrint);
18
19         add(panel);
20
21         // Create a listener
22         ButtonListener listener = new ButtonListener();
23
24         // Register listener with buttons
25         jbtNew.addActionListener(listener);
26         jbtOpen.addActionListener(listener);
27         jbtSave.addActionListener(listener);
28         jbtPrint.addActionListener(listener);
29     }
30
31     class ButtonListener implements ActionListener {
32         public void actionPerformed(ActionEvent e) {
33             if (e.getSource() == jbtNew)
34                 System.out.println("Process New");
35             else if (e.getSource() == jbtOpen)
36                 System.out.println("Process Open");
37             else if (e.getSource() == jbtSave)
38                 System.out.println("Process Save");
39             else if (e.getSource() == jbtPrint)
40                 System.out.println("Process Print");
41         }
42     }
43
44     /** Main method */
45     public static void main(String[] args) {
46         JFrame frame = new DetectSourceDemo();
47         frame.setTitle("DetectSourceDemo");
48         frame.setLocationRelativeTo(null); // Center the frame
49         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
50         frame.pack();
51         frame.setVisible(true);
52     }
53 }
```

这个程序只定义了一个内部监听器类（第31~42行），从该类中创建一个监听器（第22行），然后将它注册给四个按钮（第25~28行）。当点击一个按钮时，该按钮就触发一个ActionEvent，然后调用监听器的actionPerformed方法。actionPerformed方法使用事件的getSource()方法来检查事件源

(第33、35、37、39行), 并且确定哪个按钮触发了这个事件。

也可以定义一个自定义框架类实现ActionListener, 以改写程序清单16-3, 如程序清单16-5所示。

程序清单16-5 FrameAsListenerDemo.java

```

1 import javax.swing.*;
2 import java.awt.event.*;
3
4 public class FrameAsListenerDemo extends JFrame
5     implements ActionListener {
6     // Create four buttons
7     private JButton jbtNew = new JButton("New");
8     private JButton jbtOpen = new JButton("Open");
9     private JButton jbtSave = new JButton("Save");
10    private JButton jbtPrint = new JButton("Print");
11
12    public FrameAsListenerDemo() {
13        // Create a panel to hold buttons
14        JPanel panel = new JPanel();
15        panel.add(jbtNew);
16        panel.add(jbtOpen);
17        panel.add(jbtSave);
18        panel.add(jbtPrint);
19
20        add(panel);
21
22        // Register listener with buttons
23        jbtNew.addActionListener(this);
24        jbtOpen.addActionListener(this);
25        jbtSave.addActionListener(this);
26        jbtPrint.addActionListener(this);
27    }
28
29    /** Implement actionPerformed */
30    public void actionPerformed(ActionEvent e) {
31        if (e.getSource() == jbtNew)
32            System.out.println("Process New");
33        else if (e.getSource() == jbtOpen)
34            System.out.println("Process Open");
35        else if (e.getSource() == jbtSave)
36            System.out.println("Process Save");
37        else if (e.getSource() == jbtPrint)
38            System.out.println("Process Print");
39    }
40
41    /** Main method */
42    public static void main(String[] args) {
43        JFrame frame = new FrameAsListenerDemo();
44        frame.setTitle("FrameAsListenerDemo");
45        frame.setLocationRelativeTo(null); // Center the frame
46        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
47        frame.pack();
48        frame.setVisible(true);
49    }
50 }

```

框架类扩展JFrame并实现ActionListener (第5行)。因此, 该类是一个动作事件的监听器类。该监听器注册给四个按钮 (第23~26行)。点击按钮时, 该按钮触发一个ActionEvent, 然后调用监听器的actionPerformed方法。actionPerformed方法使用事件的getSource()方法来检查事件的源 (第31、33、35、37行), 并确定哪个按钮触发了这个事件。

这种设计不是很好, 因为它将太多的责任放在了一个类中。最好能设计一个监听器类, 它只负责处理事件。这种设计会让代码易于阅读和维护。

可以用多种方式定义监听器类。哪种方式比较好呢？使用内部类或者匿名内部类定义监听器类已经成为事件处理程序设计的标准，因为它通常都能提供清晰、干净和简洁的代码。因此，我们将在这本书里持续使用它。

16.7 问题：贷款计算器

现在，我们可以编写解决本章的前言部分给出的贷款计算器问题的代码。下面是这个程序的主要步骤：

(1) 创建一个如图16-9所示的用户接口

1) 创建一个5行2列的GridLayout面板。将标签和文本域添加到面板中。将面板标题设置为“Enter loan amount, interest rate, and years”（请输入贷款总额、利率和年份）。

2) 创建另一个带FlowLayout(FlowLayout.RIGHT)的面板，并将一个按钮添加到这个面板。

3) 将第一个面板添加到框架的中央，将第二个面板添加到框架的南边。

(2) 处理事件

创建和注册用以处理点击按钮动作事件的监听器。处理器获取关于贷款总额、利率和年数的输入，计算月偿还额和总偿还额，然后在文本域显示这些值。

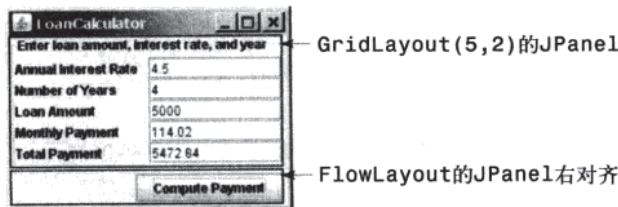


图16-9 程序计算贷款偿还额

程序清单16-6给出完整的程序。

程序清单16-6 LoanCalculator.java

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4 import javax.swing.border.TitledBorder;
5
6 public class LoanCalculator extends JFrame {
7     // Create text fields for interest rate,
8     // year, loan amount, monthly payment, and total payment
9     private JTextField jtfAnnualInterestRate = new JTextField();
10    private JTextField jtfNumberOfYears = new JTextField();
11    private JTextField jtfLoanAmount = new JTextField();
12    private JTextField jtfMonthlyPayment = new JTextField();
13    private JTextField jtfTotalPayment = new JTextField();
14
15    // Create a Compute Payment button
16    private JButton jbtComputeLoan = new JButton("Compute Payment");
17
18    public LoanCalculator() {
19        // Panel p1 to hold labels and text fields
20        JPanel p1 = new JPanel(new GridLayout(5, 2));
21        p1.add(new JLabel("Annual Interest Rate"));
22        p1.add(jtfAnnualInterestRate);
23        p1.add(new JLabel("Number of Years"));
24        p1.add(jtfNumberOfYears);
25        p1.add(new JLabel("Loan Amount"));
26        p1.add(jtfLoanAmount);
27        p1.add(new JLabel("Monthly Payment"));
28        p1.add(jtfMonthlyPayment);
```

```

29     p1.add(new JLabel("Total Payment"));
30     p1.add(jtfTotalPayment);
31     p1.setBorder(new
32         TitledBorder("Enter loan amount, interest rate, and year"));
33
34     // Panel p2 to hold the button
35     JPanel p2 = new JPanel(new FlowLayout(FlowLayout.RIGHT));
36     p2.add(jbtComputeLoan);
37
38     // Add the panels to the frame
39     add(p1, BorderLayout.CENTER);
40     add(p2, BorderLayout.SOUTH);
41
42     // Register listener
43     jbtComputeLoan.addActionListener(new ButtonListener());
44 }
45
46 /** Handle the Compute Payment button */
47 private class ButtonListener implements ActionListener {
48     public void actionPerformed(ActionEvent e) {
49         // Get values from text fields
50         double interest =
51             Double.parseDouble(jtfAnnualInterestRate.getText());
52         int year =
53             Integer.parseInt(jtfNumberOfYears.getText());
54         double loanAmount =
55             Double.parseDouble(jtfLoanAmount.getText());
56
57         // Create a loan object
58         Loan loan = new Loan(interest, year, loanAmount);
59
60         // Display monthly payment and total payment
61         jtfMonthlyPayment.setText(String.format("%.2f",
62             loan.getMonthlyPayment()));
63         jtfTotalPayment.setText(String.format("%.2f",
64             loan.getTotalPayment()));
65     }
66 }
67
68 public static void main(String[] args) {
69     LoanCalculator frame = new LoanCalculator();
70     frame.pack();
71     frame.setTitle("LoanCalculator");
72     frame.setLocationRelativeTo(null); // Center the frame
73     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
74     frame.setVisible(true);
75 }
76 }

```

构造方法创建用户界面（第18~44行）。按钮是事件的源。创建一个监视器并注册到按钮（第43行）。

监听器类（第47~66行）实现actionPerformed方法。点击按钮时，调用actionPerformed方法获取利率（第51行）、年数（第53行）以及贷款总额（第55行）。调用jtfAnnualInterestRate.getText()返回jtfAnnualInterestRate文本域中的字符串文本。贷款用来计算贷款偿还额。这个类已经在程序清单10-2中介绍过。调用loan.getMonthlyPayment()会返回这笔贷款的月偿还额。String.format方法使用类似printf语法将一个数字格式化为希望的格式。在文本域上调用setText方法可以在文本域中设置一个字符串值（第61行）。

16.8 窗口事件

前面几节使用的都是动作事件。也可以用相同的方式处理其他事件。本节给出一个处理

WindowEvent的例子。Window类的任何一个子类都可能触发下面的窗口事件：打开窗口、正在关闭窗口、关闭窗口、激活窗口、变成非活动窗口、最小化窗口和还原窗口。程序清单16-7中的程序创建一个框架，监听窗口事件，然后显示一条信息表明当前发生的事件。图16-10显示这个程序的一个运行示例。

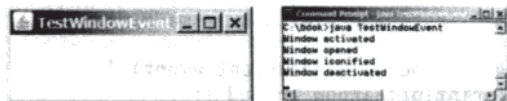


图16-10 从命令行提示符运行程序时，窗口事件就显示在控制台上

程序清单16-7 TestWindowEvent.java

```

1 import java.awt.event.*;
2 import javax.swing.JFrame;
3
4 public class TestWindowEvent extends JFrame {
5     public static void main(String[] args) {
6         TestWindowEvent frame = new TestWindowEvent();
7         frame.setSize(220, 80);
8         frame.setLocationRelativeTo(null); // Center the frame
9         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10        frame.setTitle("TestWindowEvent");
11        frame.setVisible(true);
12    }
13
14    public TestWindowEvent() {
15        addWindowListener(new WindowListener() {
16            /**
17             * Handler for window-deiconified event
18             * Invoked when a window is changed from a minimized
19             * to a normal state.
20             */
21            public void windowDeiconified(WindowEvent event) {
22                System.out.println("Window deiconified");
23            }
24
25            /**
26             * Handler for window-iconified event
27             * Invoked when a window is changed from a normal to a
28             * minimized state. For many platforms, a minimized window
29             * is displayed as the icon specified in the window's
30             * iconImage property.
31             */
32            public void windowIconified(WindowEvent event) {
33                System.out.println("Window iconified");
34            }
35
36            /**
37             * Handler for window-activated event
38             * Invoked when the window is set to be the user's
39             * active window, which means the window (or one of its
40             * subcomponents) will receive keyboard events.
41             */
42            public void windowActivated(WindowEvent event) {
43                System.out.println("Window activated");
44            }
45
46            /**
47             * Handler for window-deactivated event
48             * Invoked when a window is no longer the user's active
49             * window, which means that keyboard events will no longer
50             * be delivered to the window or its subcomponents.
51             */
52            public void windowDeactivated(WindowEvent event) {

```

```

53         System.out.println("Window deactivated");
54     }
55
56     /**
57      * Handler for window-opened event
58      * Invoked the first time a window is made visible.
59      */
60     public void windowOpened(WindowEvent event) {
61         System.out.println("Window opened");
62     }
63
64     /**
65      * Handler for window-closing event
66      * Invoked when the user attempts to close the window
67      * from the window's system menu. If the program does not
68      * explicitly hide or dispose the window while processing
69      * this event, the window-closing operation will be cancelled.
70      */
71     public void windowClosing(WindowEvent event) {
72         System.out.println("Window closing");
73     }
74
75     /**
76      * Handler for window-closed event
77      * Invoked when a window has been closed as the result
78      * of calling dispose on the window.
79      */
80     public void windowClosed(WindowEvent event) {
81         System.out.println("Window closed");
82     }
83 }
84 }
85 }

```

Window类或者Window的任何子类都可能会触发WindowEvent。因为JFrame是Window的子类，所以它可以触发WindowEvent。

TestWindowEvent扩展JFrame，并且实现WindowListener。WindowListener接口定义了几个抽象方法（windowActivated、windowClosed、windowClosing、windowDeactivated、windowDeiconified、windowIconified、WindowOpened）来处理窗口事件：激活窗口、关闭窗口、正在关闭窗口、变成非活动的窗口、最小化窗口、还原窗口或者打开窗口。

当像激活窗口这样的窗口事件发生时，windowActivated方法就被触发。如果希望处理事件，就应该用一个具体响应实现windowActivated方法。

16.9 监听器接口适配器

因为WindowListener接口中的方法都是抽象的，所以即使程序并不关注某些事件，还是必须实现所有的方法。为了方便起见，Java提供称作方便适配器（convenience adapter）的支持类，它提供监听器接口中所有方法的默认实现。默认的实现只是一个空的程序体。Java为每个AWT监听器接口提供有多个处理器的方便监听器适配器。每个XListener的方便监听器适配器命名为XAdapter。例如，WindowAdapter是WindowListener的方便监听器适配器。表16-3列出方便适配器。

表16-3 方便适配器

适配器	接口
WindowAdapter	WindowListener
MouseAdapter	MouseListener
MouseMotionAdppter	MouseMotionListener
KdyAdppter	KeyListener
ContainerAdppter	ContainerListener
ComponentAdppter	ComponentListener
FocusAdppter	FocusListener

如果你只对激活窗口的事件感兴趣，那么使用WindowAdapter就可以将前面的例子简化为如程序清单16-8所示。WindowAdapter类用来创建一个匿名监听器而不是WindowListener类（第15行）。windowActivated处理器在第16行实现。

程序清单16-8 AdapterDemo.java

```

1 import java.awt.event.*;
2 import javax.swing.JFrame;
3
4 public class AdapterDemo extends JFrame {
5     public static void main(String[] args) {
6         AdapterDemo frame = new AdapterDemo();
7         frame.setSize(220, 80);
8         frame.setLocationRelativeTo(null); // Center the frame
9         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10        frame.setTitle("AdapterDemo");
11        frame.setVisible(true);
12    }
13
14    public AdapterDemo() {
15        addWindowListener(new WindowAdapter() {
16            public void windowActivated(WindowEvent event) {
17                System.out.println("Window activated");
18            }
19        });
20    }
21 }

```

16.10 鼠标事件

当在一个组件上按下、释放、点击、移动或拖动鼠标时就会产生鼠标事件。MouseEvent对象捕获这个事件，例如，获取相应的点击次数或鼠标的位置（x和y坐标）等，如图16-11所示。

因为MouseEvent类继承InputEvent类，所以可以在MouseEvent对象上使用InputEvent类中定义的方法。

java.awt.Point类表示组件上的一个点。该类包含两个用来表示坐标的公共变量x和y。为了创建一个Point，使用下面的构造方法：

Point(int x, int y)

它利用指定的x坐标和y坐标构造一个Point对象。通常，类中的数据域应该是私有的，但是，这个类具有两个公共数据域。

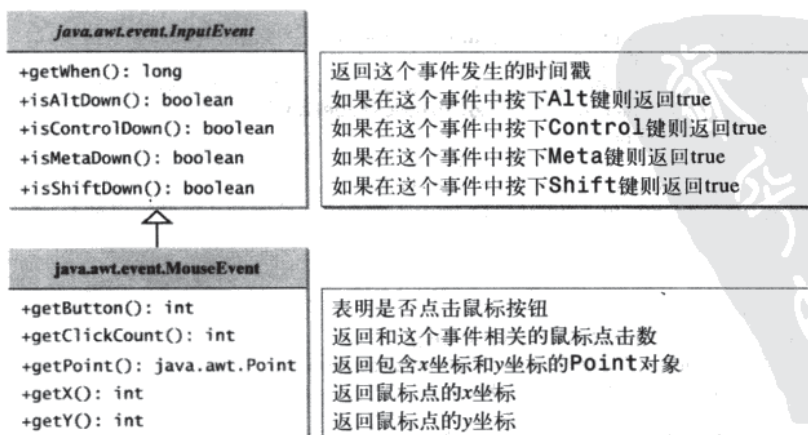


图16-11 MouseEvent类封装了鼠标事件的信息

Java提供了两个处理鼠标事件的监听器接口MouseListener和MouseMotionListener, 如图16-12所示。实现MouseListener接口可以监听如鼠标的按下、释放、输入、退出或点击的动作, 然后实现MouseMotionListener接口以监听如拖动鼠标或移动鼠标的动作。

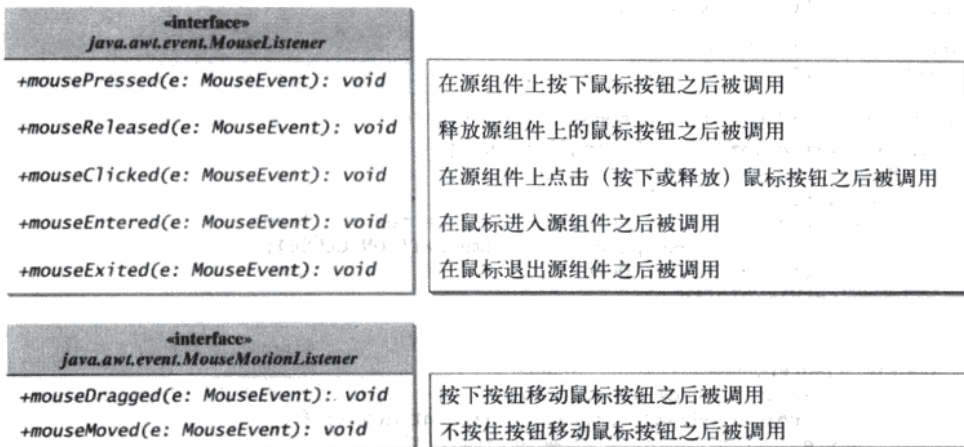


图16-12 MouseListener接口处理鼠标键按下、释放、点击、输入和退出事件。

MouseMotionListener接口处理拖动和移动鼠标的事件

举例：使用鼠标在面板上移动消息

本例编写一个程序，在面板中显示一条信息，如清序清单16-9所示。可以用鼠标移动这条消息。信息会随着鼠标的拖动而移动，信息总是显示在鼠标指针处。图16-13显示了程序的一个运行示例。



图16-13 可以拖动鼠标移动消息

程序清单16-9 MoveMessageDemo.java

```

1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4
5 public class MoveMessageDemo extends JFrame {
6     public MoveMessageDemo() {
7         // Create a MovableMessagePanel instance for moving a message
8         MovableMessagePanel p = new MovableMessagePanel
9             ("Welcome to Java");
10
11         // Place the message panel in the frame
12         setLayout(new BorderLayout());
13         add(p);
14     }
15
16     /** Main method */
17     public static void main(String[] args) {
18         MoveMessageDemo frame = new MoveMessageDemo();
19         frame.setTitle("MoveMessageDemo");
20         frame.setSize(200, 100);
21         frame.setLocationRelativeTo(null); // Center the frame
22         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

```

```

23     frame.setVisible(true);
24 }
25
26 // Inner class: MovableMessagePanel draws a message
27 static class MovableMessagePanel extends JPanel {
28     private String message = "Welcome to Java";
29     private int x = 20;
30     private int y = 20;
31
32     /** Construct a panel to draw string s */
33     public MovableMessagePanel(String s) {
34         message = s;
35         addMouseListener(new MouseMotionAdapter() {
36             /** Handle mouse-dragged event */
37             public void mouseDragged(MouseEvent e) {
38                 // Get the new location and repaint the screen
39                 x = e.getX();
40                 y = e.getY();
41                 repaint();
42             }
43         });
44     }
45
46     /** Paint the component */
47     protected void paintComponent(Graphics g) {
48         super.paintComponent(g);
49         g.drawString(message, x, y);
50     }
51 }
52 }

```

MovableMessagePanel类扩展JPanel类来绘制消息（第27行）。除此之外，拖动鼠标时它还处理重新显示消息。这个类被定义为主类中的内部类，因为它只在这个类中使用。因此，这个内部类被定义为静态的，因为它不引用主类的任何实例成员。

MouseListener接口包含两个处理器：mouseMoved和mouseDragged，它们用于处理鼠标动作事件。移动鼠标按下按钮时，调用mouseDragged方法，用于重画视图区域并在鼠标点处显示消息。不按按钮而移动鼠标时，就会调用mouseMoved方法。

因为监听器只关心鼠标的拖动事件，所以匿名内部类监听器扩展MouseMotionAdapter覆盖mouseDragged方法。如果内部类实现了MouseListener接口，即使监听器并不关心某些事件，也必须实现所有的处理器。

当按下按钮移动鼠标时，就会调用mouseDragged方法。该方法利用MouseEvent类中的getX方法和getY方法（第39~40行）来获取鼠标的位置。它就成为消息的新位置。调用repaint()方法（第41行）会导致paintComponent被调用（第47行），这样，就将在新位置显示这个消息。

16.11 按键事件

按键事件可以利用键盘来控制 and 执行一些动作，或者从键盘上获取输入。只要按下、释放一个键或者在一个组件上敲击，就会触发按键事件。KeyEvent对象描述事件的特性（即按下、放开或者敲击一个键）和对应的键值，如图16-14所示。Java提供KeyListener接口来处理按键事件，如图16-15所示。

当按下下一个键时，就会调用keyPressed处理器，当松开一个键时，就会调用keyReleased处理器，当输入一个统一码字符时，就会调用keyTyped处理器。如果这个键没有统一码（例如，功能键、修改键、动作键和控制键），则不会调用keyTyped处理器。

每个按键事件都有一个相关的按键字符或按键代码，它们分别由KeyEvent中的getKeyChar()和getKeyCode()方法返回。按键编码是定义在表16-4中的常量。对于统一码字符的按键，按键代码和统一码的值相同。

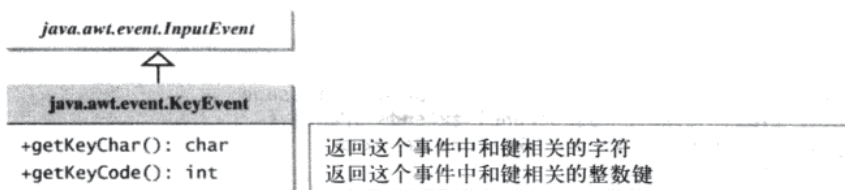


图16-14 KeyEvent类封装了关于按键事件的信息

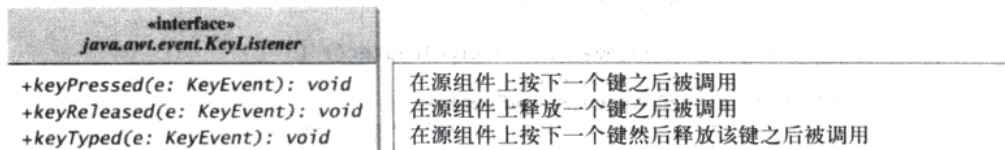


图16-15 按下、释放和敲击KeyListener接口处理键的事件

表16-4 按键常量

常 量	描 述	常 量	描 述
VK_HOME	Home键	VK_CONTROL	控制键
VK_End	End键	VK_SHIFT	Shift键
VK_PGUP	Page Up键	VK_BACK_SPACE	退格键
VK_PGDN	Page Down键	VK_CAPS_LOCK	大写字母锁定键
VK_UP	上箭头	VK_NUM_LOCK	数字键盘锁定键
VK_DOWN	下箭头	VK_ENTER	回车键
VK_LEFT	左箭头	VK_UNDEFINED	按键代码未知
VK_RIGHT	右箭头	VK_F1到VK_F12	从F1到F12的功能键
VK_ESCAPE	Esc键	VK_0到VK_9	从0到9的数字键
VK_TAB	Tab键	VK_A到VK_Z	从A到Z的字母键

对于按下按键和释放按键事件，`getKeyCode()`返回定义在表中的值。对于按键敲击事件，`getKeyCode()`返回VK_UNDEFINED，而`getKeyChar()`返回输入的字符。

程序清单16-10中的程序显示用户输入的字符。用户可以使用箭头键VK_UP、VK_DOWN、VK_LEFT和VK_RIGHT向上、向下、向左、向右移动字符，程序的运行示例如图16-16所示。

程序清单16-10 KeyEventDemo.java

```

1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4
5 public class KeyEventDemo extends JFrame {
6     private KeyboardPanel keyboardPanel = new KeyboardPanel();
7
8     /** Initialize UI */
9     public KeyEventDemo() {
10         // Add the keyboard panel to accept and display user input
11         add(keyboardPanel);
12
13         // Set focus
14         keyboardPanel.setFocusable(true);
15     }
  
```

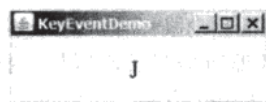
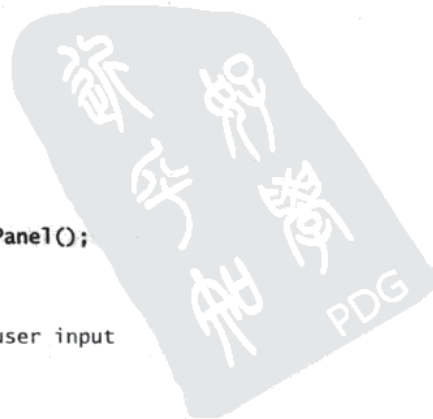


图16-16 程序通过显示一个字符以及向上、向下、向左或向右响应按键事件




```

16
17 /** Main method */
18 public static void main(String[] args) {
19     KeyEventDemo frame = new KeyEventDemo();
20     frame.setTitle("KeyEventDemo");
21     frame.setSize(300, 300);
22     frame.setLocationRelativeTo(null); // Center the frame
23     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24     frame.setVisible(true);
25 }
26
27 // Inner class: KeyboardPanel for receiving key input
28 static class KeyboardPanel extends JPanel {
29     private int x = 100;
30     private int y = 100;
31     private char keyChar = 'A'; // Default key
32
33     public KeyboardPanel() {
34         addKeyListener(new KeyAdapter() {
35             public void keyPressed(KeyEvent e) {
36                 switch (e.getKeyCode()) {
37                     case KeyEvent.VK_DOWN: y += 10; break;
38                     case KeyEvent.VK_UP: y -= 10; break;
39                     case KeyEvent.VK_LEFT: x -= 10; break;
40                     case KeyEvent.VK_RIGHT: x += 10; break;
41                     default: keyChar = e.getKeyChar();
42                 }
43
44                 repaint();
45             }
46         });
47     }
48
49     /** Draw the character */
50     protected void paintComponent(Graphics g) {
51         super.paintComponent(g);
52
53         g.setFont(new Font("TimesRoman", Font.PLAIN, 24));
54         g.drawString(String.valueOf(keyChar), x, y);
55     }
56 }
57 }

```

`KeyboardPanel`类扩展`JPanel`显示一个字符（第28行）。这个类定义为主类中的内部类，因为它只在这个类中。而且，内部类要定义为静态的，因为它不引用任何主类中的实例成员。

由于程序从键盘上获取输入，因此，它监听键盘事件`KeyEvent`，并且扩展`KeyAdapter`来处理按键输入（第34行）。

当按下一个键时，就会调用`keyPressed`处理器。程序使用`e.getKeyCode()`方法获取按键代码，使用`e.getKeyChar()`方法获取按键的对应字符。当按下一个非箭头键时，显示它的字符（第41行）。当按下箭头键时，字符就按照箭头键指示的方向移动（第37~40行）。

仅有一个焦点的组件能够接收`KeyEvent`。要使一个组件成为聚焦的，需要将属性`isFocusable`设置为`true`（第14行）。

每次重新绘制组件时，都会在第53行为`Graphics`对象创建一种新字体。这是不够的。最好一次就将字体创建一个数据域。

16.12 使用`Timer`类的动画

并非所有的源对象都是GUI组件。类`javax.swing.Timer`就是一个按照预定频率触发

ActionEvent事件的源组件。图16-17列出了这个类中的一些方法。

javax.swing.Timer	
+Timer(delay: int, listener: ActionListener)	创建一个带特定的毫秒延时和ActionListener的Timer对象
+addActionListener(listener: ActionListener): void	给定时器增加一个ActionListener
+start(): void	启动一个定时器
+stop(): void	终止一个定时器
+setDelay(delay: int): void	为这个定时器设置一个新的延时值

图16-17 Timer对象以固定频率触发ActionEvent事件

一个Timer对象可以作为ActionEvent事件的源。监听器必须是ActionListener的实例并且要注册到Timer对象。利用延时和监听器，可以使用Timer类的唯一一个构造方法创建一个Timer对象，其中延时是指两个动作事件之间间隔的毫秒数。可以使用addActionListener方法添加其他的监听器，用setDelay方法调整延时。要启动定时器，调用start()方法。要终止定时器时，调用stop()方法。

Timer类可以用于控制动画。例如，可以利用它显示一条移动的消息，如图16-18所示，其代码在程序清单16-11中。

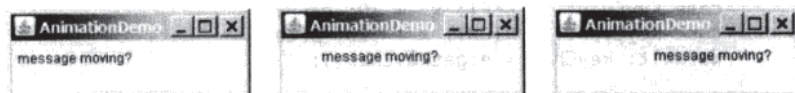


图16-18 消息在面板上移动

程序清单16-11 AnimationDemo.java

```

1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4
5 public class AnimationDemo extends JFrame {
6     public AnimationDemo() {
7         // Create a MovingMessagePanel for displaying a moving message
8         add(new MovingMessagePanel("message moving?"));
9     }
10
11     /** Main method */
12     public static void main(String[] args) {
13         AnimationDemo frame = new AnimationDemo();
14         frame.setTitle("AnimationDemo");
15         frame.setSize(280, 100);
16         frame.setLocationRelativeTo(null); // Center the frame
17         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
18         frame.setVisible(true);
19     }
20
21     // Inner class: Displaying a moving message
22     static class MovingMessagePanel extends JPanel {
23         private String message = "Welcome to Java";
24         private int xCoordinate = 0;
25         private int yCoordinate = 20;
26
27         public MovingMessagePanel(String message) {
28             this.message = message;
29
30             // Create a timer
31             Timer timer = new Timer(1000, new TimerListener());
32             timer.start();
33         }
34     }

```

```

35  /** Paint message */
36  protected void paintComponent(Graphics g) {
37      super.paintComponent(g);
38
39      if (xCoordinate > getWidth()) {
40          xCoordinate = -20;
41      }
42      xCoordinate += 5;
43      g.drawString(message, xCoordinate, yCoordinate);
44  }
45
46  class TimerListener implements ActionListener {
47      /** Handle ActionEvent */
48      public void actionPerformed(ActionEvent e) {
49          repaint();
50      }
51  }
52 }
53 }

```

MovingMessagePanel类扩展JPanel以显示一条消息（第22行）。这个类定义为主类中的一个内部类，因为只在这个类中使用它。而且，因为这个内部类不能引用主类的任意实例成员，所以，要把这个内部类定义为静态的。

一个内部类监听器在第46行定义为监听ActionEvent。第31行为监听器创建一个Timer类。这个定时器从第32行开始。每秒钟定时器都会触发一个ActionEvent，而监听器会在第49行响应来重新绘制面板。当绘制好了一个面板，它的x坐标就会增加（第42行），因此，消息会向右显示。当x坐标超过了面板的范围，它会被重置为-20（第40行），所以，消息继续从左向右移动。

在15.12节中，绘制一个显示当前时间的StillClock。但是，时钟显示过后就不再走针了。如何才能使时钟每秒钟都能显示新的当前时间呢？使时钟走针的关键是每秒都用新的当前时间来重新绘制这个时钟。可以用程序清单16-12中的代码使用定时器控制时钟的重新绘制。

程序清单16-12 ClockAnimation.java

```

1  import java.awt.event.*;
2  import javax.swing.*;
3
4  public class ClockAnimation extends JFrame {
5      private StillClock clock = new StillClock();
6
7      public ClockAnimation() {
8          add(clock);
9
10         // Create a timer with delay 1000 ms
11         Timer timer = new Timer(1000, new TimerListener());
12         timer.start();
13     }
14
15     private class TimerListener implements ActionListener {
16         /** Handle the action event */
17         public void actionPerformed(ActionEvent e) {
18             // Set new time and repaint the clock to display current time
19             clock.setCurrentTime();
20             clock.repaint();
21         }
22     }
23
24     /** Main method */
25     public static void main(String[] args) {
26         JFrame frame = new ClockAnimation();
27         frame.setTitle("ClockAnimation");
28         frame.setSize(200, 200);
29         frame.setLocationRelativeTo(null); // Center the frame

```

```

30     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
31     frame.setVisible(true);
32 }
33 }

```

程序显示一个正在运转的时钟，如图16-19所示。ClockAnimation创建一个StillClock（第5行）。第11行为ClockAnimation创建一个Timer。这个定时器从第12行开始。定时器每秒钟触发一个ActionEvent，而监听器响应设置新时间（第19行）并重新绘制时钟（第20行）。定义在StillClock中的setCurrentTime()方法设置时钟的当前时间。



图16-19 在面板上显示一个运转的时钟

关键术语

anonymous inner class（匿名内部类）	event-listener interface（事件监听器接口）
convenience listener adapter（方便监听器适配器）	event object（事件对象）
event（事件）	event registration（事件注册）
event delegation（事件委托）	event source（source object）（事件源（源对象））
event handler（事件处理器）	event-driven programming（事件驱动的程序设计）
event listener（事件监听器）	inner class（内部类）

本章小结

- 事件类的根类是java.util.EventObject。EventObject的子类处理各种特殊类型的事件，例如，动作事件、窗口事件、组件事件、鼠标事件和按键事件。可以使用EventObject类中的getSource()实例方法判断事件的源对象。如果一个组件能够触发某个事件，那么它的所有子类都能触发同类型的事件。
- 监听器对象的类必须实现相应的事件监听器接口。Java语言为每种事件类提供监听器接口。XEvent的监听器接口通常命名为XListener，但是，MouseMotionListener除外。例如，ActionEvent对应的监听器接口是ActionListener；每个ActionEvent的监听器都应该实现ActionListener接口。监听器接口包含称为处理器的处理事件的方法。
- 监听器对象必须由源对象注册。注册方法依赖于事件的类型。对ActionEvent来讲，注册方法是addActionListener。一般来说，XEvent的方法命名为addXListener。
- 内部类或者嵌套类是定义在另一个类中的类。内部类可以引用定义在它嵌套的外部类中的数据和他方法，所以，不需要将外部类的引用传递给内部类的构造方法。
- 方便适配器是能够提供监听器接口中所有方法默认实现的支持类。Java为每个AWT监听器接口提供带多个处理器的方便监听器适配器。XListener的方便监听器适配器命名为XAdapter。
- 一个源对象可以触发几种类型的事件。对每种事件，源对象维护一个注册的监听器列表，通过调用监听器对象的处理器，通知所有已经注册的监听器去处理事件。
- 在一个组件上点击、释放、移动或拖动鼠标时就会触发鼠标事件。鼠标事件对象捕获事件，例如，

和事件相关的点击次数或鼠标点的位置 (x和y坐标)。

- Java提供两个处理鼠标事件的监听器接口：`MouseListener`和`MouseMotionListener`来处理事件，实现`MouseListener`接口来监听诸如按下、释放、点击、输入或退出鼠标等动作，实现`MouseMotionListener`接口来监听诸如移动或拖动鼠标的动作。
- `KeyEvent`对象描述事件的性质（即按下、释放或敲击一个键）以及相对应的键值。
- 当按下按键时就会调用`keyPressed`处理器，当释放按键时就会调用`keyReleased`处理器，而当敲入一个统一码字符键时，就会调用`keyTyped`处理器。如果某个按键没有统一码（例如，功能键、修改键、动作键和控制键），则不会调用`keyTyped`处理器。
- 可以使用`Timer`类控制Java的动画。定时器以固定频率触发`ActionEvent`。监听器通过更新画面来模拟动画。

复习题

16.2~16.3节

- 16.1 一个按钮能够触发`WindowEvent`吗？一个按钮能够触发`MouseEvent`吗？一个按钮能够触发`ActionEvent`吗？
- 16.2 为什么监听器必须是一个合适的监听器接口的实例？解释如何注册一个监听器对象以及如何实现一个监听器接口。
- 16.3 一个源可以有多个监听器吗？一个监听器可以监听多个源吗？一个源可以是自身的监听器吗？
- 16.4 如何实现定义在监听器接口中的方法？要实现所有定义在监听器接口中的方法吗？

16.4~16.9节

- 16.5 内部类可以用在不是嵌套它的其他类中吗？
- 16.6 修饰符`public`、`private`和`static`可以用在内部类吗？
- 16.7 如果类A是类B的内部类，那么A的.class文件是什么？如果类B包括两个匿名内部类，那么这两个类的.class文件名是什么？
- 16.8 下面代码有何错误？

```
import java.swing.*;
import java.awt.*;

public class Test extends JFrame {
    public Test() {
        JButton jbtOK = new JButton("OK");
        add(jbtOK);
    }

    private class Listener
        implements ActionListener {
        public void actionPerformed
            (ActionEvent e) {
            System.out.println
                (jbtOK.getActionCommand());
        }
    }

    /** Main method omitted */
}
```

a)

```
import java.awt.event.*;
import javax.swing.*;

public class Test extends JFrame {
    public Test() {
        JButton jbtOK = new JButton("OK");
        add(jbtOK);
        jbtOK.addActionListener(
            new ActionListener() {
                public void actionPerformed
                    (ActionEvent e) {
                    System.out.println
                        (jbtOK.getActionCommand());
                }
            } // Something missing here
        )

    /** Main method omitted */
}
```

b)

- 16.9 `JFrame`中的方法`setSize(width,height)`和方法`pack()`有什么不同？

16.10~16.11节

- 16.10 使用什么方法可以获取事件的源？使用什么方法获取动作事件、鼠标事件或者按键事件的时间戳？使用什么方法获取鼠标事件的鼠标点位置？使用什么方法获取按键事件的按键字符？

16.11 鼠标的按下、释放、点击、进入和退出的监听器接口是什么？鼠标移动和拖动的监听器接口是什么？

16.12 键盘上的每个键是否都有统一码？KeyEvent类中的按键编码是否等价于统一码？

16.13 按下一个键之后是否调用keyPressed处理器？释放一个键之后是否调用keyReleased处理器？在敲击任意一个键之后是否调用keyTyped处理器？

16.12节

16.14 如何创建一个定时器？如何启动定时器？如何停止定时器？

16.15 Timer类是否有无参构造方法？能将多个监听器添加到定时器吗？

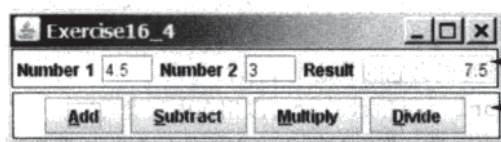
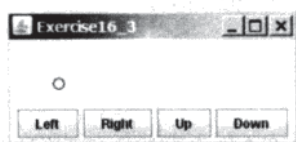
编程练习题

16.2~16.9节

16.1 (在控制台上找出点击了哪个按钮) 通过添加代码给练习题12.1, 在控制台上显示消息, 指出点击了哪个按钮。

16.2 (使用ComponentEvent) 任何GUI组件都可以触发一个ComponentEvent。ComponentListener定义componentMoved、componentResized、componentShown和componentHidden方法来处理组件事件。编写测试程序来演示ComponentEvent。

*16.3 (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16-20a所示。



使用FlowLayout的面板
使用FlowLayout的面板

图16-20 a) 练习题16.3使用按钮来移动小球; b) 程序完成double数字上加法、减法、乘法和除法

*16.4 (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16-20b)。

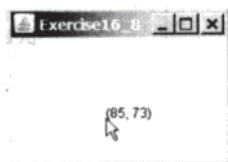
*16.5 (创建一个投资值计算器) 编写一个程序, 计算投资值在给定利率以及给定年数下的未来值。计算的公式如下所示:

$$\text{futureValue} = \text{investmentAmount} * (1 + \text{monthlyInterestRate})^{\text{years} * 12} \quad (\text{未来值} = \text{投资值} \times (1 + \text{月利率})^{\text{年数} * 12})$$

使用文本域显示利率、投资总额和年数。当用户点击Calculate按钮时在文本域显示未来的总额, 如图16-21a所示。



a)



b)



c)

图16-21 a) 用户输入投资总额、年数和利率来计算未来值; b) 练习题16.8显示鼠标的位置; c) 练习题16.9使用箭头键绘制直线

16.10节

**16.6 (两个消息交替出现) 编写一个程序, 当点击鼠标时, 面板上交替显示两个消息“Java is fun”和“Java is powerful”。

*16.7 (使用鼠标设置背景色) 编写一个程序, 显示一个面板的背景色, 当按下鼠标键时背景色为黑色, 释放鼠标时背景色为白色。

- *16.8 (显示鼠标的位置) 编写两个程序, 一个显示点击鼠标时鼠标的位置 (参见图16-21a), 而另一个显示当按下鼠标时显示鼠标的位置, 当释放鼠标时停止显示。

16.11节

- *16.9 (使用箭头键画线) 编写一个程序, 使用箭头键绘制线段。所画的线从框架的中心开始, 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16-21c所示。
- **16.10 (输入并显示字符串) 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。回车键表明字符串的结束。当输入一个新字符串时, 会将它显示在面板上。
- *16.11 (显示一个字符) 编写一个程序, 从键盘获得一个字符输入, 然后将这个字符显示在鼠标点所在的位置。

16.12节

- **16.12 (显示一个转动的风扇) 程序清单15-4显示了一个静止的风扇。编写程序显示一个转动的风扇。
- **16.13 (播放幻灯) 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, 在本书的源代码中可以下载。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。

提示 在框架中放置一个标签, 并且在标签中将幻灯片设置为图像图标。

- **16.14 (升旗) 编写一个Java程序, 用动画实现升旗, 如图16-1所示 (参见15.11节中关于如何显示图像的内容)。
- **16.15 (赛车) 编写一个Java程序, 模拟汽车比赛, 如图16-22a所示。汽车从左向右移动。当它到达右终点, 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。使用新的基坐标 (x, y) 重新绘制汽车, 如图16-22b所示。

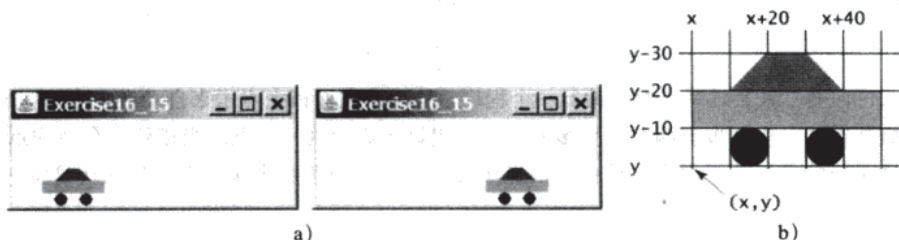


图16-22 a) 练习题16.15显示一个移动的汽车; b) 可以使用一个新的基点重画汽车

- *16.16 (显示一个闪烁的标签) 编写一个显示闪烁标签的程序。

提示 为了使标签闪烁, 需要以固定频率交替地重画带标签的面板和不带标签的面板 (黑屏)。使用一个boolean变量来控制交替显示。

- *16.17 (控制一个移动的标签) 修改程序清单16-11, 使用鼠标控制一个移动的标签。按住鼠标键时, 标签停止移动, 释放按钮则继续移动。

综合题

- *16.18 (使用键移动圆) 编写程序使用箭头键向上、向下、向左、向右移动一个圆。
- **16.19 (几何方面: 是否在圆内?) 编写一个程序, 绘制一个圆心在 (100, 60) 而半径为50的圆。当鼠标移动时, 显示一个消息表示鼠标点是否在圆内, 如图16-23a所示。

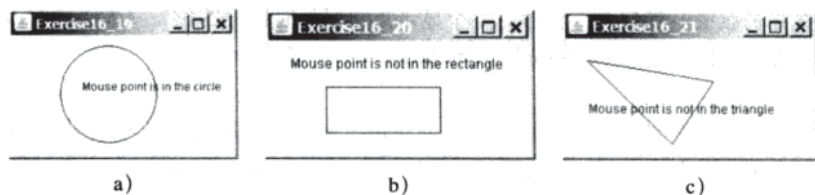


图16-23 检查一个点是否在圆内, 是否在矩形内, 是否在三角形内

****16.20 (几何问题: 是否在矩形内?)** 编写一个程序, 绘制一个中心在 (100, 60) 宽为100而高为40的矩形。当鼠标移动时, 显示一个消息表示鼠标点是否在矩形内, 如图16-23b所示。为了检查一个点是否在矩形内, 需要使用定义在练习题10.12中的MyRectangle2D类。

****16.21 (几何问题: 是否在三角形内?)** 编写一个程序, 绘制三个端点分别是 (20, 20)、(100, 100) 和 (140, 40) 的三角形。当鼠标移动时, 显示一个消息表示鼠标点是否在三角形内, 如图16-23c所示。为了检查一个点是否在三角形内, 需要使用定义在练习题10.13中的Triangle2D类。

*****16.22 (游戏: 豆机动画)** 编写程序用动画实现练习题15.24介绍的豆机。在10个球掉下来之后动画结束, 如图16-24所示。

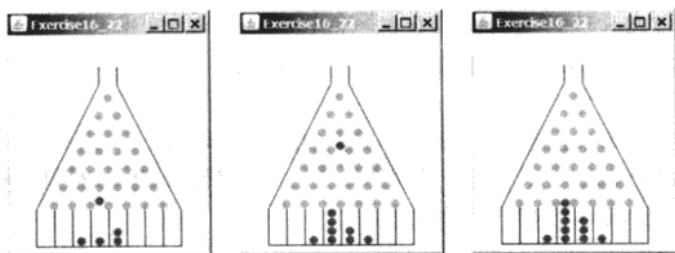


图16-24 球落入豆机

*****16.23 (几何问题: 最接近的点对)** 编写一个程序, 让用户在面板上点击来自动产生点。初始状态时, 面板是空的。当面板有两个或更多的点, 突出显示最近的一对点。当创建一个新的点时, 就突出显示新的最近的一对点。使用小圆显示点, 使用实心圆突出显示点, 如图16-25a~图16-25c所示。

提示 将点存储在ArrayList中。

***16.24 (控制时钟)** 修改程序清单16-12来添加start()和stop()方法以启动和停止时钟。编写一个程序, 让用户控制带Start和Stop按钮的时钟, 如图16-25d所示。

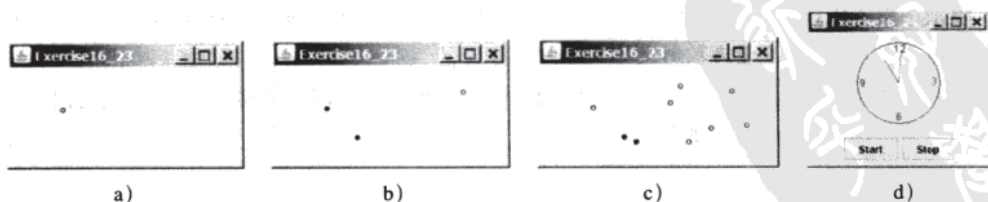


图16-25 练习题16.23允许用户随着鼠标的点击而创建新的点, 并凸显最近的点对。

练习题16.24允许用户启动和停止一个时钟

*****16.25 (游戏: 打气球)** 编写一个程序, 在面板内的任意位置显示一个气球 (如图16-26a所示)。使用向左和向右箭头键指定枪向左或向右瞄准气球 (如图16-26b)。点击向上箭头键可以从枪中发出一个小球 (如图16-26c所示)。一旦球击中该气球, 就显示气球碎片 (如图16-26e所示), 然后

在任意位置显示一个新的气球（如图16-26f所示）。如果球没有击中气球，一旦它碰到面板的边界，该球就消失了。然后点击向上的箭头键就可以发出另一个球。当点击向左或向右箭头键，枪就会向左或向右转5度。（教师可以如下修改这个游戏：（1）显示被打破的气球的个数；（2）显示一个倒计时的计时器（例如，60秒），一旦时间到就终止这个游戏；（3）允许气球动态地上升。）

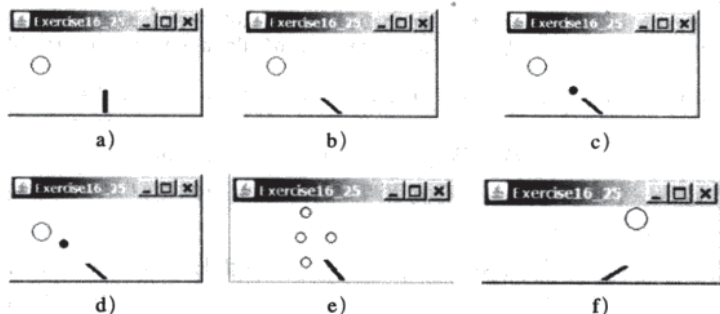


图16-26 a) 在任意一个位置显示气球；b) 点击向左/向右箭头键来瞄准气球；c) 点击向上键发出一个球；d) 球直接向气球移动；e) 球击中气球；f) 在任意位置显示一个新的气球

****16.26**（使用鼠标移动一个圆）编写一个程序，显示半径为10像素的圆。可以将鼠标在圆内点一下，然后随着鼠标的移动来拖动（即按住鼠标移动）圆，如图16-27a~图16-27b所示。

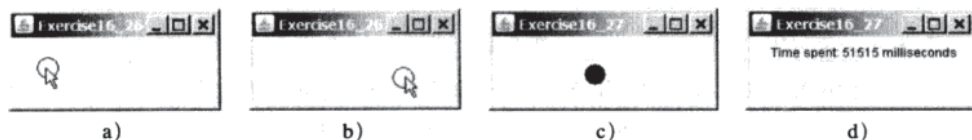


图16-27 a) ~b) 可以点击、拖动和移动圆；c) 点击一个圆时，一个新的圆会出现在任意位置；d) 点击20个圆后，面板中会显示所用的时间

*****16.27**（游戏：手眼协调）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任意位置，并填充任意颜色，如图16-27c所示。点击这个圆时，它会消失，然后在另一个任意的位置显示新的任意颜色的圆。在点击了20个圆之后，在面板上显示所用的时间，如图16-27d所示。

*****16.28**（模拟：自避免随机漫步）栅栏中的自避免漫步是从一个点到另一点的过程中，不重复两次访问一个点的路线。自避免漫步已经应用在物理、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16-28a所示，或者在一个死端点结束（即该点被四个已经访问过的点包围），如图16-28b所示。假设栅栏的大小是16×16。

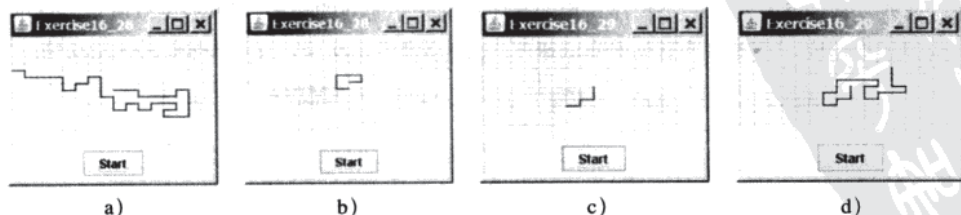


图16-28 a) 一个在边界点结束的路径；b) 一个在死端点结束的路径；c) ~d) 动画显示逐步构造路径的过程

***16.29 (动画: 自避免随机漫步) 修改上一个练习题, 在动画中一步一步地显示路线, 如图16-28c图16-28d所示。

**16.30 (模拟: 自避免随机漫步) 编写一个模拟程序, 显示出现死端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的栅栏。对每种大小的栅栏, 模拟自避免随机漫步10 000次, 然后显示出现死端点路径的概率, 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%
```



*16.31 (几何问题: 显示正 n 边形) 练习题15.25创建RegularPolygonPanel显示一个正 n 边形。编写一个程序, 显示一个正多边形, 然后使用两个名为+1和-1的按钮来增加或减少多边形的边数, 如图16-29a~图16-29b所示。

**16.32 (几何问题: 添加或删除点) 编写一个程序, 让用户在面板上点击以自动创建或移去点。当用户右击鼠标时, 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, 然后左击鼠标以移去这个点, 如图16-29c所示。

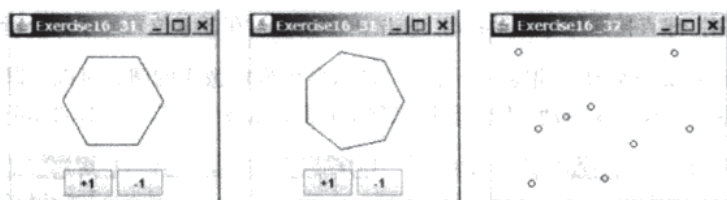


图16-29 点击+1或-1按钮增加或减少练习题16.31中正多边形的边数。

练习题16.32允许用户动态地创建/移去点

**16.33 (几何问题: 回文) 编写一个程序, 用动画完成回文摆动, 如图16-30所示。点击向上箭头键增加速度, 点击向下箭头键降低速度。点击S键停止动画, 点击R键重新开始。



图16-30 练习题16.33用动画实现一个回文摆动

**16.34 (游戏: 刽子手) 编写一个程序, 用动画实现刽子手游戏的摆动, 如图16-31所示。点击向上箭头键增加速度, 点击向下箭头键降低速度。点击S键停止动画, 点击R键重新开始动画。

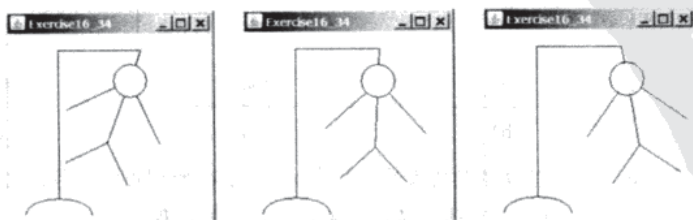


图16-31 练习题16.34用动画实现一个刽子手游戏

*****16.35 (游戏：刽子手)** 练习题9.31给出了流行的刽子手游戏的控制台版本。编写一个GUI程序让用户来玩这个游戏。用户通过一次输入一个字母来猜单词，如图16-32a所示。如果用户七次都没猜对，被吊的人就摆动起来，如图16-32b~图16-32c所示。一旦完成一个单词，用户就可以点击Enter键继续猜另一个单词。

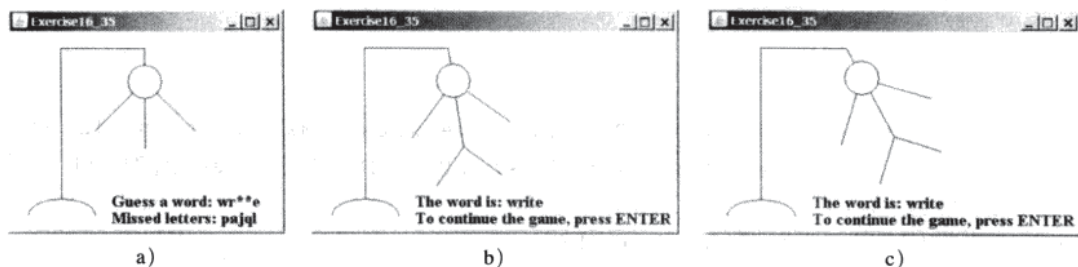


图16-32 练习题16.35开发一个完整的刽子手游戏

***16.36 (翻硬币)** 编写一个程序，显示九个硬币的正面(H)或反面(T)，如图16-33所示。当点击一个格子时，硬币就被翻面。一个格子就是一个JLabel。编写一个自定义的格子类，该类扩展JLabel且使用鼠标监听器处理这些点击。当程序开始时，所有的格子都被初始化为H。

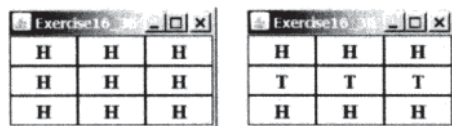


图16-33 练习题16.36可以让用户点击格子来翻硬币

资源
分享
PDG

第17章

Introduction to Java Programming, 8E

创建图形用户界面

学习目标

- 使用不同的用户界面组件，例如，JButton、JCheckBox、JRadioButton、JLabel、JTextField、JTextArea、JComboBox、JList、JScrollBar和JSlider创建图形用户界面（17.2~17.11节）。
- 为不同类型的事件创建监听器（17.2~17.11节）。
- 探究JButton（17.2节）。
- 探究JCheckBox（17.3节）。
- 探究JRadioButton（17.4节）。
- 探究JLabel（17.5节）。
- 探究JTextField（17.6节）。
- 探究JTextArea（17.7节）。
- 探究JComboBox（17.8节）。
- 探究JList（17.9节）。
- 探究JScrollBar（17.10节）。
- 探究JSlider（17.11节）。
- 在一个应用程序中显示多个窗口（17.12节）。

17.1 引言

图形用户界面（GUI）可以使系统对用户更友好且更易于使用。创建一个GUI需要创造性以及有关GUI组件如何工作的知识。在Java中，GUI组件非常灵活且功能多样，因而可以创建各种各样有用的用户界面。

许多Java集成开发环境都提供了可视化设计和开发GUI接口的工具。这样，可以用最少的编码快速为Java应用程序或applet将用户界面（UI）元素集合在一起。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Java GUI程序设计的一些基本概念。

前几章简要介绍了一些GUI组件。本章详细地介绍经常会用到的GUI组件（参见图17-1）。（因为本章没有介绍什么新概念，所以教师可以将本章布置给学生自学。）

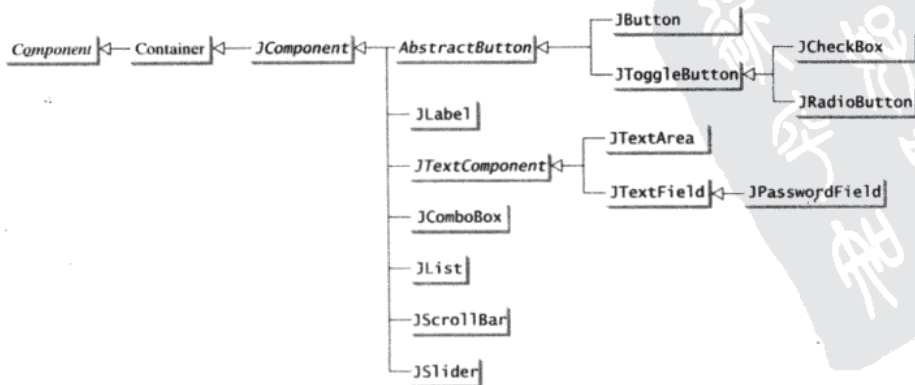


图17-1 这些Swing组件经常用来创建用户接口

注意 本书整篇都使用前缀jbt、jchk、jrb、jlbl、jtf、jpf、jta、jcbo、jlst、jscb和jsld来命名JButton、JCheckBox、JRadioButton、JLabel、JTextField、JPasswordField、JTextArea、JComboBox、JList、JScrollbar和JSlider的引用变量。

17.2 按钮

按钮(button)是点击时触发动作事件的组件。Swing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javax.swing.AbstractButton中定义,如图17-2所示。

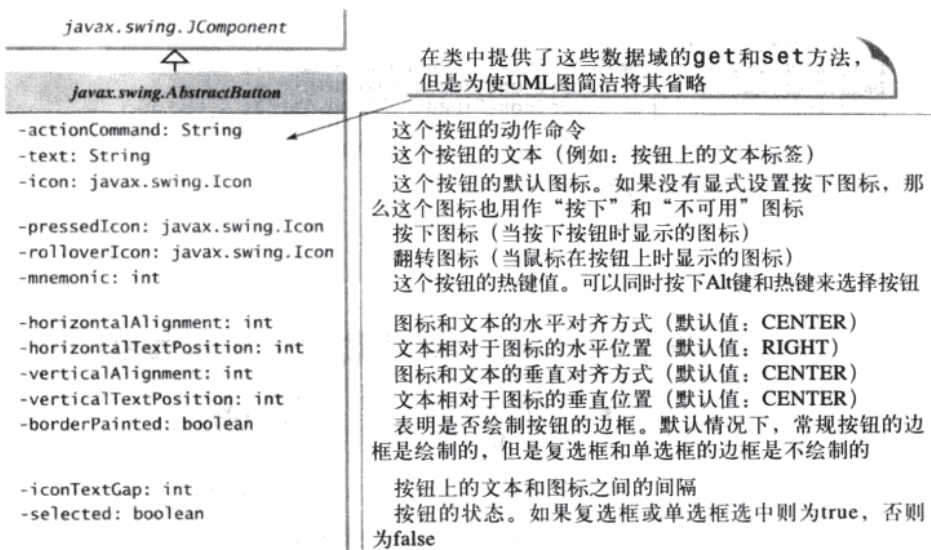


图17-2 AbstractButton定义了不同类型按钮的公共特性

本节介绍定义在JButton类中的常规按钮。JButton继承AbstractButton并且提供创建按钮的几个构造方法,如图17-3所示。

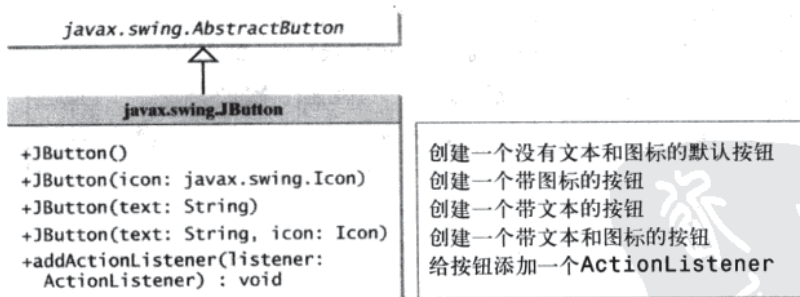


图17-3 JButton定义常规的下压按钮

17.2.1 图标、按下图标和翻转图标

每个常规按钮都有一个默认图标、一个按下图标和一个翻转图标。通常使用的是默认图标。其他的图标是为了显示特殊效果。当按下按钮的时候,显示按下图标,当鼠标移到按钮上而不按下时,显示翻转图标。例如,程序清单17-1将美国国旗作为常规图标显示,将加拿大国旗作为按下图标,将英国国旗作为翻转图标,如图17-4所示。

程序清单17-1 TestButtonIcons.java

```

1 import javax.swing.*;
2
3 public class TestButtonIcons extends JFrame {
4     public static void main(String[] args) {
5         // Create a frame and set its properties
6         JFrame frame = new TestButtonIcons();
7         frame.setTitle("ButtonIcons");
8         frame.setSize(200, 100);
9         frame.setLocationRelativeTo(null); // Center the frame
10        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11        frame.setVisible(true);
12    }
13
14    public TestButtonIcons() {
15        ImageIcon usIcon = new ImageIcon("image/usIcon.gif");
16        ImageIcon caIcon = new ImageIcon("image/caIcon.gif");
17        ImageIcon ukIcon = new ImageIcon("image/ukIcon.gif");
18
19        JButton jbt = new JButton("Click it", usIcon);
20        jbt.setPressedIcon(caIcon);
21        jbt.setRolloverIcon(ukIcon);
22
23        add(jbt);
24    }
25 }

```



图17-4 一个按钮可以有多种类型的图标

17.2.2 对齐方式

水平对齐 (horizontal alignment) 指定以什么样的水平方式在按钮上放置图标和文本。可以使用五个常量 LEADING、LEFT、CENTER、RIGHT 和 TRAILING 之一作为参数调用 `setHorizontalAlignment(int)` 方法来设置水平对齐方式，如图17-5所示。目前，LEADING和LEFT一样，TRAILING和RIGHT一样。今后可能会对它们加以区分。默认的水平对齐方式是 `SwingConstants.CENTER`。



图17-5 可以指定如何在按钮的水平方向上放置图标和文本

垂直对齐 (vertical alignment) 指定以什么样的垂直方式在按钮上放置文本和图标。可以使用三个常量 TOP、CENTER 和 BOTTOM 之一作为参数调用 `setVerticalAlignment(int)` 方法设置垂直对齐方式，如图17-6所示。默认的垂直对齐方式是 `SwingConstants.CENTER`。



图17-6 可以指定如何在按钮的垂直方向上放置图标和文本

17.2.3 文本位置

水平文本位置 (horizontal text position) 指定文本相对于图标的水平位置。可以使用五个常量 LEADING、LEFT、CENTER、RIGHT 和 TRAILING 之一作为参数调用 `setHorizontalTextPosition(int)` 方法设置文本的水平位置, 如图17-7所示。目前, LEADING 和 LEFT 一样, TRAILING 和 RIGHT 一样。Java 今后可能会区分它们。默认的水平文本位置是 `SwingConstants.RIGHT`。



图17-7 可以指定文本相对于图标的水平位置

垂直文本位置 (vertical text position) 指定文本相对于图标的垂直位置。可以使用三个常量 TOP、CENTER 和 BOTTOM 之一作为参数调用 `setVerticalTextPosition(int)` 方法设置文本的垂直位置, 如图17-8所示。默认的垂直文本位置是 `SwingConstants.CENTER`。

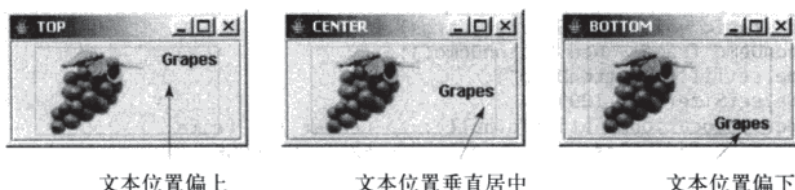


图17-8 可以指定文本相对于图标的垂直位置

注意 `AbstractButton` 类中使用的常量 LEFT、CENTER、RIGHT、LEADING、TRAILING、TOP 和 BOTTOM 也可以被许多其他的 Swing 组件使用。这些常量集中定义在 `javax.swing.SwingConstants` 接口中。由于所有的 Swing GUI 组件都实现了 `SwingConstants`, 所以, 可以通过 `SwingConstants` 或 GUI 组件引用这些常量。例如, `SwingConstants.CENTER` 与 `JButton.CENTER` 是一样的。

`JButton` 可以触发许多类型的事件, 但是, 通常需要添加监听器以响应 `ActionEvent` 事件。当下一个按钮时, 它就会触发一个 `ActionEvent` 事件。

17.2.4 使用按钮

本节给出一个程序, 如程序清单17-2所示, 它在面板上显示一条消息, 然后使用两个按钮: `<=>` 和 `=>`, 在面板上向左或向右移动这个消息。用户界面的布局如图17-9所示。

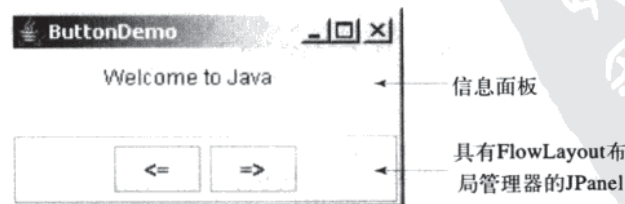


图17-9 点击按钮 `<=>` 和 `=>` 使面板上的消息分别向左和向右移动

下面是程序的主要步骤:

- 1) 创建用户界面。

创建一个显示消息的MessagePanel对象。MessagePanel类是在程序清单15-8中创建的。将MessagePanel对象放置在框架中央。在一个面板上创建两个按钮<=和=>。然后将这个面板放在框架的南边。

2) 处理事件。

创建并注册监听器，根据是否点击向左按钮或向右按钮处理动作事件，以便向左或向右移动消息。

程序清单17-2 ButtonDemo.java

```

1 import java.awt.*;
2 import java.awt.event.ActionListener;
3 import java.awt.event.ActionEvent;
4 import javax.swing.*;
5
6 public class ButtonDemo extends JFrame {
7     // Create a panel for displaying message
8     protected MessagePanel messagePanel
9         = new MessagePanel("Welcome to Java");
10
11     // Declare two buttons to move the message left and right
12     private JButton jbtLeft = new JButton("<=");
13     private JButton jbtRight = new JButton(">=");
14
15     public static void main(String[] args) {
16         ButtonDemo frame = new ButtonDemo();
17         frame.setTitle("ButtonDemo");
18         frame.setSize(250, 100);
19         frame.setLocationRelativeTo(null); // Center the frame
20         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21         frame.setVisible(true);
22     }
23
24     public ButtonDemo() {
25         // Set the background color of messagePanel
26         messagePanel.setBackground(Color.white);
27
28         // Create Panel jpButtons to hold two Buttons "<=" and ">="
29         JPanel jpButtons = new JPanel();
30         jpButtons.add(jbtLeft);
31         jpButtons.add(jbtRight);
32
33         // Set keyboard mnemonics
34         jbtLeft.setMnemonic('L');
35         jbtRight.setMnemonic('R');
36
37         // Set icons and remove text
38         // jbtLeft.setIcon(new ImageIcon("image/left.gif"));
39         // jbtRight.setIcon(new ImageIcon("image/right.gif"));
40         // jbtLeft.setText(null);
41         // jbtRight.setText(null);
42
43         // Set tool tip text on the buttons
44         jbtLeft.setToolTipText("Move message to left");
45         jbtRight.setToolTipText("Move message to right");
46
47         // Place panels in the frame
48         setLayout(new BorderLayout());
49         add(messagePanel, BorderLayout.CENTER);
50         add(jpButtons, BorderLayout.SOUTH);
51
52         // Register listeners with the buttons
53         jbtLeft.addActionListener(new ActionListener() {
54             public void actionPerformed(ActionEvent e) {
55                 messagePanel.moveLeft();

```



```

56     }
57   });
58   jbtRight.addActionListener(new ActionListener() {
59     public void actionPerformed(ActionEvent e) {
60       messagePanel.moveRight();
61     }
62   });
63 }
64 }

```

messagePanel (第8行) 被刻意声明为protected, 这样, 它就可以被后面例子中的子类引用。可以使用setIcon方法在按钮上设置一个图标图像。如果把第38~41行的以下代码中的注释去掉:

```

// jbtLeft.setIcon(new ImageIcon("image/left.gif"));
// jbtRight.setIcon(new ImageIcon("image/right.gif"));
// jbtLeft.setText(null);
// jbtRight.setText(null);

```

按钮上的文本就被图标替换, 如图17-10a所示。"image/left.gif"放置在"c:\book\image\left.gif"中。注意, 反斜杠是Windows文件路径的符号。而Java中应该使用斜杠。

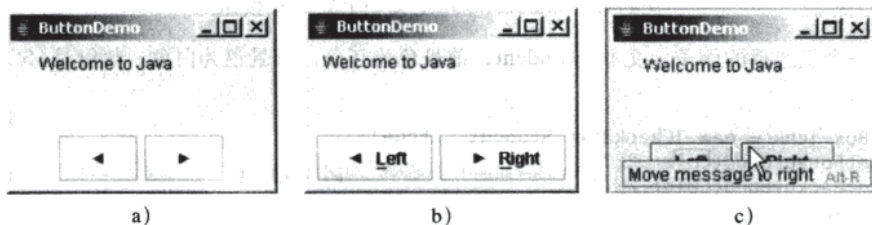


图17-10 可以在JButton上设置一个图标并且使用按钮的热键访问按钮

如果愿意, 就可以在按钮上同时设置本文和图标, 如图17-10b所示。默认情况下, 文本和图标在水平方向和垂直方向都是居中的。

按钮也可以通过键盘上的热键来访问。由于把左边按钮的热键属性设置为'L' (第34行), 所以, 按下Alt+L键相当于点击<=按钮。如果将左边按钮的文本改为"Left", 而将右边按钮的文本改为"Right", 那么这两个按钮上文字中的L和R都会有下划线, 如图17-10b所示。

每个按钮都有一个工具提示文本 (第44~45行), 它在鼠标放在按钮上但不被按下时出现, 如图17-10c所示。

注意 由于MessagePanel没有在Java API中, 所以应该将MessagePanel.java放在和ButtonDemo.java相同的目录中。

17.3 复选框

一个开关按钮 (toggle button) 就像是电灯开关一样会有两种状态。JToggleButton类继承AbstractButton并实现了一个开关按钮。通常, 使用JToggleButton的子类JCheckBox和JRadioButton让用户在开或关这两种状态之间进行选择。本节介绍复选框按钮类JCheckBox, 单选按钮类JRadioButton将在17.4节介绍。

JCheckBox继承AbstractButton类的所有属性, 例如, text、icon、mnemonic、verticalAlignment、horizontalAlignment、horizontalTextPosition、verticalTextPosition和selected, 而且还提供了创建复选框的几种构造方法, 如图17-11所示。

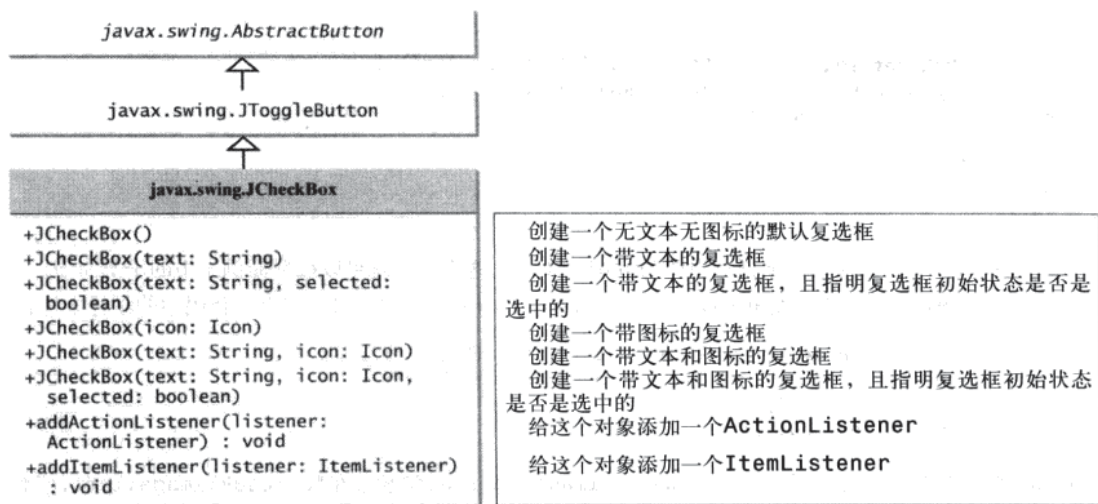


图17-11 JCheckBox定义一个复选框按钮

下面是一个复选框的例子，文本是Student，前景色为红色，背景色为白色，热键是'S'，而初始状态为选中的。

```

JCheckBox jchk = new JCheckBox("Student", true);
jchk.setForeground(Color.RED);
jchk.setBackground(Color.WHITE);
jchk.setMnemonic('S');
  
```



当点击（选中或取消）一个复选框时，它会触发一个ItemEvent事件，然后触发一个ActionEvent事件。要了解复选框是否被选中，使用isSelected()方法。

程序清单17-3给出的程序将三个名为Centered、Bold和Italic的复选框添加到前面的例子中。用户使用它们指定信息是否居中、是否为粗体或斜体，如图17-12所示。

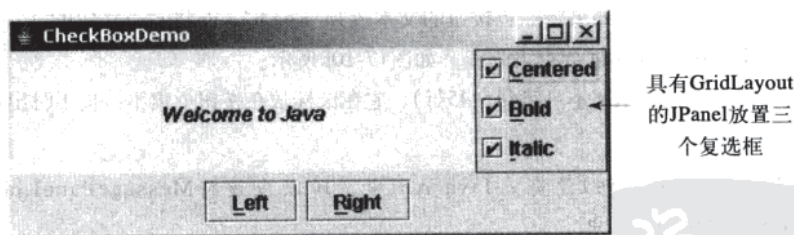


图17-12 添加三个复选框以指定如何显示消息

编写这个程序至少有两种方法。第一种是修改上面的ButtonDemo类，插入代码来添加复选框并处理它们的事件。第二种是创建一个扩展自ButtonDemo类的子类。第一种方法的实现留作练习题。程序清单17-3给出实现第二种方法的代码。

CheckBoxDemo类扩展ButtonDemo类，并且增加三个复选框来控制如何显示消息。当构造一个CheckBoxDemo（第12行）时，调用其父类的无参数构造方法，因此，不必改写已经在ButtonDemo构造方法中的代码。

程序清单17-3 CheckBoxDemo.java

```

1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4
5 public class CheckBoxDemo extends ButtonDemo {
6     // Create three check boxes to control the display of message
7     private JCheckBox jchkCentered = new JCheckBox("Centered");
8     private JCheckBox jchkBold = new JCheckBox("Bold");
9     private JCheckBox jchkItalic = new JCheckBox("Italic");
10
11     public static void main(String[] args) {
12         CheckBoxDemo frame = new CheckBoxDemo();
13         frame.setTitle("CheckBoxDemo");
14         frame.setSize(500, 200);
15         frame.setLocationRelativeTo(null); // Center the frame
16         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17         frame.setVisible(true);
18     }
19
20     public CheckBoxDemo() {
21         // Set mnemonic keys
22         jchkCentered.setMnemonic('C');
23         jchkBold.setMnemonic('B');
24         jchkItalic.setMnemonic('I');
25
26         // Create a new panel to hold check boxes
27         JPanel jpCheckBoxes = new JPanel();
28         jpCheckBoxes.setLayout(new GridLayout(3, 1));
29         jpCheckBoxes.add(jchkCentered);
30         jpCheckBoxes.add(jchkBold);
31         jpCheckBoxes.add(jchkItalic);
32         add(jpCheckBoxes, BorderLayout.EAST);
33
34         // Register listeners with the check boxes
35         jchkCentered.addActionListener(new ActionListener() {
36             public void actionPerformed(ActionEvent e) {
37                 messagePanel.setCentered(jchkCentered.isSelected());
38             }
39         });
40         jchkBold.addActionListener(new ActionListener() {
41             public void actionPerformed(ActionEvent e) {
42                 setNewFont();
43             }
44         });
45         jchkItalic.addActionListener(new ActionListener() {
46             public void actionPerformed(ActionEvent e) {
47                 setNewFont();
48             }
49         });
50     }
51
52     private void setNewFont() {
53         // Determine a font style
54         int fontStyle = Font.PLAIN;
55         fontStyle += (jchkBold.isSelected() ? Font.BOLD : Font.PLAIN);
56         fontStyle += (jchkItalic.isSelected() ? Font.ITALIC : Font.PLAIN);
57
58         // Set font for the message
59         Font font = messagePanel.getFont();
60         messagePanel.setFont(
61             new Font(font.getName(), fontStyle, font.getSize()));
62     }
63 }

```

当选中或取消选中复选框时，调用监听器的`actionPerformed`方法处理这个事件。选中或取消选中`Centered`复选框时，`MessagePanel`类的`centered`属性被设置为`true`或`false`。

使用`getName()`和`getSize()`方法，从`messagePanel.getFont()`中获取`MessagePanel`当前使用的字体名和字体大小。字体风格(`Font.BOLD`和`Font.ITALIC`)在复选框中指定。如果没有选择字体风格，那么字体风格就是`Font.PLAIN`。可以通过将选择的表示字体的整数加在一起组合成字体风格。

分别将C、B、I设置为复选框`Centered`、`Bold`和`Italic`的热键(第22~24行)。可以使用鼠标或快捷键来选择一个复选框。

`MessagePanel`类中继承了定义在`Component`类中的`setFont`方法(第60行)。这个方法自动调用`repaint`方法。调用`MessagePanel`中的`setFont`方法可以自动重新绘制这条消息。

点击复选框时，会触发`ActionEvent`事件和`ItemEvent`事件。处理`ActionEvent`事件或`ItemEvent`事件都可以重新显示这个消息。这个例子处理`ActionEvent`事件。如果希望处理`ItemEvent`事件，可以为`ItemEvent`创建一个监听器并且将其注册到一个复选框。这个监听器必须实现`itemStateChanged`处理器来处理一个`ItemEvent`。例如，下面的代码向`jchkCentered`注册一个`ItemListener`：

```
// To listen for ItemEvent
jchkCentered.addItemListener(new ItemListener() {
    /** Handle ItemEvent */
    public void itemStateChanged(ItemEvent e) {
        messagePanel.setCentered(jchkCentered.isSelected());
    }
});
```

17.4 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一个单一的条目。从外观上看，单选按钮类似于复选框。不过不管选中与否，复选框都是方形的，而单选按钮都是圆的，不论它是填充的(选中时)还是空白的(未选中时)。

`JRadioButton`类继承`AbstractButton`类，同时提供一些创建单选按钮的构造方法，如图17-13所示。这些构造方法类似于`JCheckBox`的构造方法。

下面是一个单选按钮的例子，文本为`Student`，前景色为红色，背景色为白色，热键为`S`，而初始状态为选中。

```
JRadioButton jrb = new JRadioButton("Student", true);
jrb.setForeground(Color.RED);
jrb.setBackground(Color.WHITE);
jrb.setMnemonic('S');
```

为了将单选按钮放在一组，需要创建`java.swing.ButtonGroup`的一个实例，并且使用`add`方法把这些单选按钮添加到这个实例中，如下所示：

```
ButtonGroup group = new ButtonGroup();
group.add(jrb1);
group.add(jrb2);
```

这个代码给单选按钮`jrb1`和`jrb2`创建了一个单选按钮组，这样，单选按钮`jrb1`和`jrb2`在选择时就是互斥的。如果没有创建这个组，`jrb1`与`jrb2`就是相互独立的。

注意 `ButtonGroup`不是`java.awt.Component`的子类，所以，`ButtonGroup`对象不能添加到容器中。

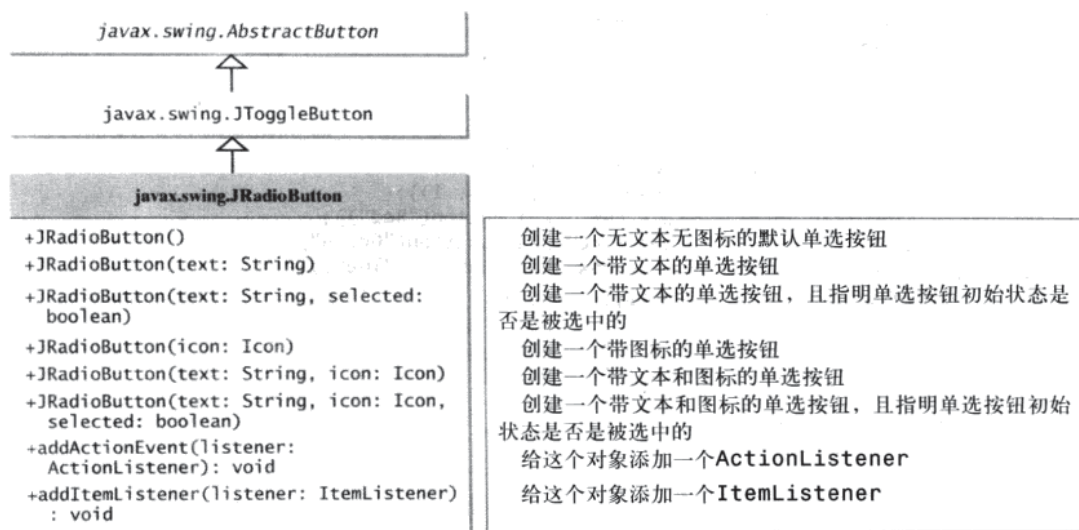


图17-13 JRadioButton定义一个单选按钮

当改变一个单选按钮（选中或者取消选中）时，它会触发一个ItemEvent事件以及一个ActionEvent事件。要判断一个单选按钮是否被选中，使用isSelected()方法。

程序清单17-4给出的程序是给前面的例子中添加三个名为Red、Green和Blue的单选按钮，让用户选择消息的颜色，如图17-14所示。

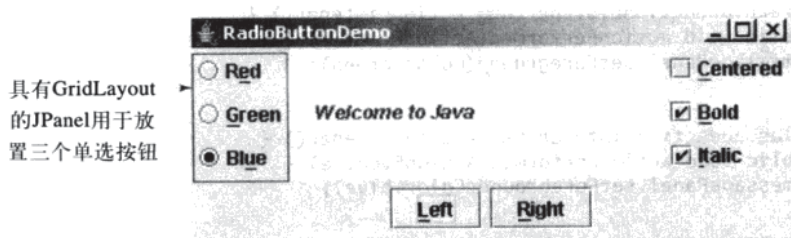


图17-14 添加三个单选按钮以指定消息的颜色

至少有两种方法编写这个程序。第一种是修改前面的CheckBoxDemo类，插入添加单选按钮以及处理它们的事件的代码。第二种是创建一个扩展CheckBoxDemo的子类。程序清单17-4给出实现第二种方法的代码。

程序清单17-4 RadioButtonDemo.java

```

1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4
5 public class RadioButtonDemo extends CheckBoxDemo {
6     // Declare radio buttons
7     private JRadioButton jrbRed, jrbGreen, jrbBlue;
8
9     public static void main(String[] args) {
10         RadioButtonDemo frame = new RadioButtonDemo();
11         frame.setSize(500, 200);
12         frame.setLocationRelativeTo(null); // Center the frame
13         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  
```

```

14     frame.setTitle("RadioButtonDemo");
15     frame.setVisible(true);
16 }
17
18 public RadioButtonDemo() {
19     // Create a new panel to hold check boxes
20     JPanel jpRadioButtons = new JPanel();
21     jpRadioButtons.setLayout(new GridLayout(3, 1));
22     jpRadioButtons.add(jrbRed = new JRadioButton("Red"));
23     jpRadioButtons.add(jrbGreen = new JRadioButton("Green"));
24     jpRadioButtons.add(jrbBlue = new JRadioButton("Blue"));
25     add(jpRadioButtons, BorderLayout.WEST);
26
27     // Create a radio-button group to group three buttons
28     ButtonGroup group = new ButtonGroup();
29     group.add(jrbRed);
30     group.add(jrbGreen);
31     group.add(jrbBlue);
32
33     // Set keyboard mnemonics
34     jrbRed.setMnemonic('E');
35     jrbGreen.setMnemonic('G');
36     jrbBlue.setMnemonic('U');
37
38     // Register listeners for radio buttons
39     jrbRed.addActionListener(new ActionListener() {
40         public void actionPerformed(ActionEvent e) {
41             messagePanel.setForeground(Color.red);
42         }
43     });
44     jrbGreen.addActionListener(new ActionListener() {
45         public void actionPerformed(ActionEvent e) {
46             messagePanel.setForeground(Color.green);
47         }
48     });
49     jrbBlue.addActionListener(new ActionListener() {
50         public void actionPerformed(ActionEvent e) {
51             messagePanel.setForeground(Color.blue);
52         }
53     });
54
55     // Set initial message color to blue
56     jrbBlue.setSelected(true);
57     messagePanel.setForeground(Color.blue);
58 }
59 }

```

RadioButtonDemo类扩展**CheckBoxDemo**类，并且增加三个指定消息颜色的单选按钮。当点击一个单选按钮时，它的动作事件监听器设置**messagePanel**中对应的前景色。

R和**B**已经被设置为**Right**按钮和**Bold**复选框的热键。为了避免冲突，将**E**、**G**和**U**分别设置为单选按钮**Red**、**Green**和**Blue**的热键（第34~36行）。

程序创建一个**ButtonGroup**，并且将三个**JRadioButton**的实例（**jrbRed**、**jrbGreen**和**jrbBlue**）放到这个组中（第28~31行）。

当选中或取消选中一个单选按钮时，它会触发**ActionEvent**事件和**ItemEvent**事件。可以处理**ActionEvent**或**ItemEvent**来选择一种颜色。这个例子处理**ActionEvent**事件。请使用**ItemEvent**事件改写这些代码作为练习题。

17.5 标签

标签（label）是一个显示小段文字、一幅图像或同时显示两者的区域。它经常用来给其他组件（通

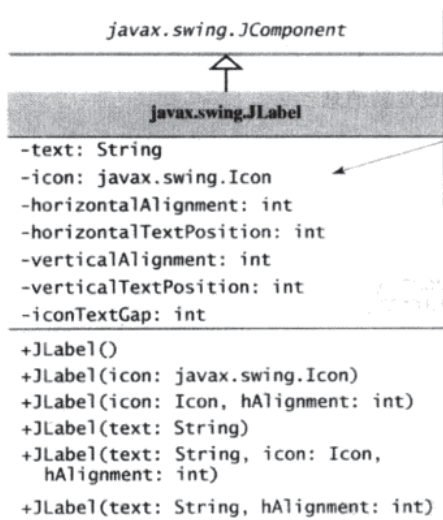
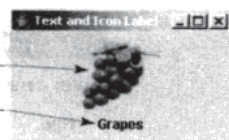
常为文本域)做标记。图17-15列出了JLabel中的构造方法和方法。

JLabel继承了JComponent类的所有属性,并具有与JButton类相似的许多属性,例如, text、icon、horizontalAlignment、verticalAlignment、horizontalTextPosition、VerticalTextPosition和iconTextGap。例如,下述代码显示了一个带文本和图标

```
// Create an image icon from an image file
ImageIcon icon = new ImageIcon("image/grapes.gif");

// Create a label with a text, an icon,
// with centered horizontal alignment
JLabel jlb1 = new JLabel("Grapes", icon, SwingConstants.CENTER);

//Set label's text alignment and gap between text and icon
jlb1.setHorizontalTextPosition(SwingConstants.CENTER);
jlb1.setVerticalTextPosition(SwingConstants.BOTTOM);
jlb1.setIconTextGap(5);
```



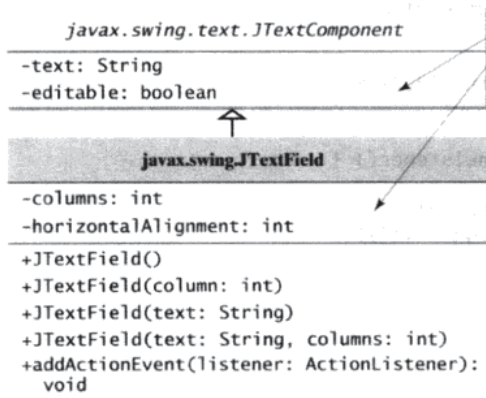
在类中提供了这些数据域的get和set方法,但是为使UML图简洁将其省略

标签的文本
 标签的图像图标
 标签上文本和图标的水平对齐方式
 标签上相对于图标的文本水平位置
 标签上文本和图标的垂直对齐方式
 标签上相对于图标的文本垂直位置
 标签上文本和图标之间的间隔
 创建一个无文本无图标的默认标签
 创建一个带图标的标签
 创建一个带图标以及指定的水平对齐方式的标签
 创建一个带文本的标签
 创建一个带文本、图标和指定水平对齐方式的标签

图17-15 JLabel显示文本或图标或两者都显示

17.6 文本域

文本域(text field)可用于输入或显示一个字符串。JTextField类是JTextComponent类的一个子类。图17-16列出JTextField的构造方法和方法。



在类中提供了这些数据域的get和set方法,但是为使UML图简洁将其省略

这个文本组件所包含的文本
 表明这个文本组件是否是可编辑的(默认值: true)

这个文本域中的列数
 这个文本域的水平对齐方式(默认值: LEFT)
 创建一个列数设置为0的默认空文本域
 创建一个指定列数的空文本域
 创建一个用指定文本初始化的文本域
 创建一个用指定文本和列初始化的文本域
 给这个对象添加一个ActionListener

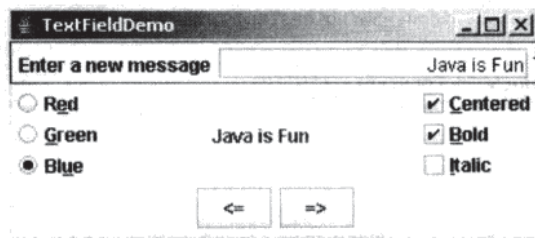
图17-16 JTextField允许输入或显示一个字符串

JTextField类继承JTextComponent类，而JTextComponent类继承JComponent类。下面是创建一个文本域的例子，它的前景色为红色，对齐方式为水平向右对齐方式：

```
JTextField jtfMessage = new JTextField("T-Strom");
jtfMessage.setForeground(Color.RED);
jtfMessage.setHorizontalAlignment(SwingConstants.RIGHT);
```

把光标移动到文本域，然后按下回车键时，会触发一个ActionEvent事件。

程序清单17-5给前一个例子添加一个文本域，允许用户设置一条新消息，如图17-17所示。



具有BorderLayout的JPanel用于标签和文本域

图17-17 添加标签和文本域以设置新消息

程序清单17-5 TextFieldDemo.java

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4
5 public class TextFieldDemo extends RadioButtonDemo {
6     private JTextField jtfMessage = new JTextField(10);
7
8     /** Main method */
9     public static void main(String[] args) {
10         TextFieldDemo frame = new TextFieldDemo();
11         frame.pack();
12         frame.setTitle("TextFieldDemo");
13         frame.setLocationRelativeTo(null); // Center the frame
14         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15         frame.setVisible(true);
16     }
17
18     public TextFieldDemo() {
19         // Create a new panel to hold label and text field
20         JPanel jpTextField = new JPanel();
21         jpTextField.setLayout(new BorderLayout(5, 0));
22         jpTextField.add(
23             new JLabel("Enter a new message"), BorderLayout.WEST);
24         jpTextField.add(jtfMessage, BorderLayout.CENTER);
25         add(jpTextField, BorderLayout.NORTH);
26
27         jtfMessage.setHorizontalAlignment(JTextField.RIGHT);
28
29         // Register listener
30         jtfMessage.addActionListener(new ActionListener() {
31             /** Handle ActionEvent */
32             public void actionPerformed(ActionEvent e) {
33                 messagePanel.setMessage(jtfMessage.getText());
34                 jtfMessage.requestFocusInWindow();
35             }
36         });
37     }
38 }
```

TextFieldDemo类扩展RadioButtonDemo类，并且添加一个标签和一个文本域以允许用户输入新

消息。在文本域内设置一条新消息，然后按下回车键，就会显示一条新消息。在文本域上按下回车键就会触发一个动作事件。监听器在messagePanel中设置一条新消息（第33行）。

方法pack()可以按照放在框架里的组件的大小自动调整框架的尺寸。

Component类中定义的requestFocusInWindow()方法（第34行）请求组件接收输入焦点。这样，jtfMessage.requestFocusInWindow()方法（第34行）请求将输入聚焦于jtfMessage上。所以，在调用actionPerformed方法后，将会看到光标在jtfMessage上。

注意 如果使用一个文本域输入密码，那么可以使用JPasswordField替换JTextField。

JPasswordField扩展JTextField并且用回显字符（例如，*****）隐藏输入的文本。默认情况下，回显字符是*。可以使用setEchoChar(char)方法指定一个新的回显字符。

17.7 文本区域

如果想让用户输入多行文本，必须创建JTextArea的几个实例。一个更好的可选择的办法是使用JTextArea，它允许用户输入多行文本。图17-18列出JTextArea的构造方法和方法。

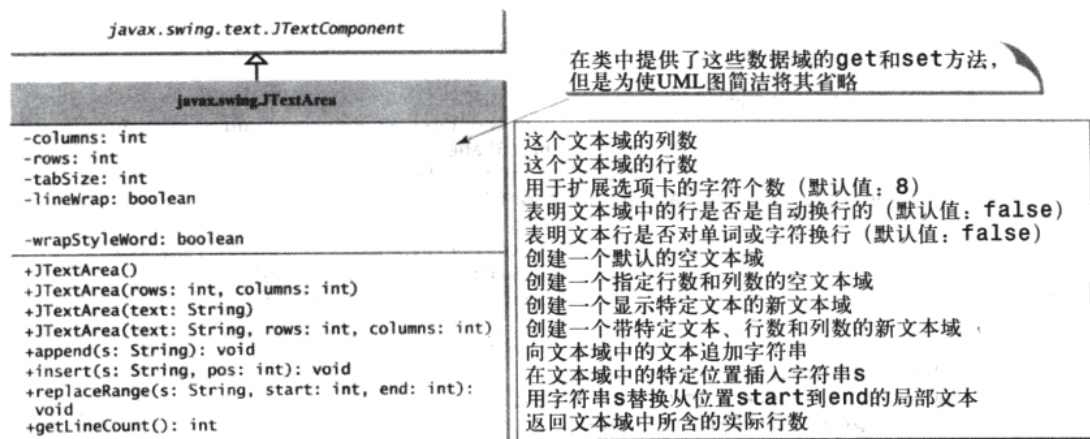


图17-18 JTextArea允许输入或显示多行字符

同JTextField类一样，JTextArea类也继承自JTextComponent类，并且包含方法getText、setText、isEditable以及setEditable。这里是一个创建5行20列文本域的例子，按单词自动换行，前景色为红色，字体为Courier，字型为粗体，字号大小为20像素。

```

JTextArea jtaNote = new JTextArea("This is a text area", 5, 20);
jtaNote.setLineWrap(true);
jtaNote.setWrapStyleWord(true);
jtaNote.setForeground(Color.red);
jtaNote.setFont(new Font("Courier", Font.BOLD, 20));
  
```

JTextArea不处理滚动，但是，可以创建一个JScrollPane的对象来保存JTextArea的一个实例，让JScrollPane处理JTextArea的滚动，如下所示：

```

// Create a scroll pane to hold text area
JScrollPane scrollPane = new JScrollPane(jta = new JTextArea());
add(scrollPane, BorderLayout.CENTER);
  
```

程序清单17-7给出的程序是在一个标签上显示图像和文本，在一个文本区域中显示文本，如图17-19所示。

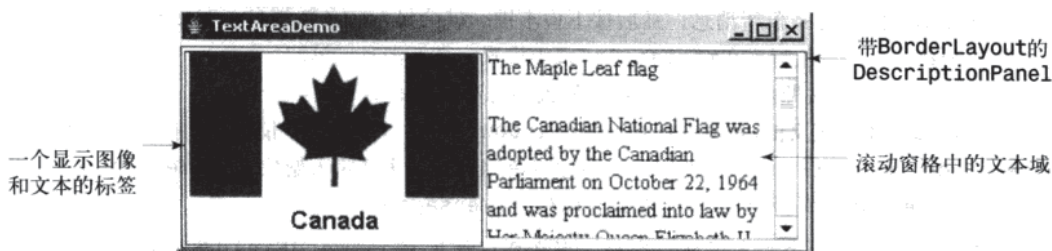


图17-19 程序在标签中显示图像和标题，且在文本区域中显示文本

程序清单17-6 DescriptionPanel.java

```

1 import javax.swing.*;
2 import java.awt.*;
3
4 public class DescriptionPanel extends JPanel {
5     /** Label for displaying an image icon and a text */
6     private JLabel jlblImageTitle = new JLabel();
7
8     /** Text area for displaying text */
9     private JTextArea jtaDescription = new JTextArea();
10
11     public DescriptionPanel() {
12         // Center the icon and text and place the text under the icon
13         jlblImageTitle.setHorizontalAlignment(JLabel.CENTER);
14         jlblImageTitle.setHorizontalTextPosition(JLabel.CENTER);
15         jlblImageTitle.setVerticalTextPosition(JLabel.BOTTOM);
16
17         // Set the font in the label and the text field
18         jlblImageTitle.setFont(new Font("SansSerif", Font.BOLD, 16));
19         jtaDescription.setFont(new Font("Serif", Font.PLAIN, 14));
20
21         // Set lineWrap and wrapStyleWord true for the text area
22         jtaDescription.setLineWrap(true);
23         jtaDescription.setWrapStyleWord(true);
24         jtaDescription.setEditable(false);
25
26         // Create a scroll pane to hold the text area
27         JScrollPane scrollPane = new JScrollPane(jtaDescription);
28
29         // Set BorderLayout for the panel, add label and scrollpane
30         setLayout(new BorderLayout(5, 5));
31         add(scrollPane, BorderLayout.CENTER);
32         add(jlblImageTitle, BorderLayout.WEST);
33     }
34
35     /** Set the title */
36     public void setTitle(String title) {
37         jlblImageTitle.setText(title);
38     }
39
40     /** Set the image icon */
41     public void setImageIcon(ImageIcon icon) {
42         jlblImageTitle.setIcon(icon);
43     }
44
45     /** Set the text description */
46     public void setDescription(String text) {
47         jtaDescription.setText(text);
48     }
49 }

```

PDF
PDG

这个程序有以下几个主要的步骤：

1) 扩展JPanel，创建一个名为DescriptionPanel的类，如程序清单17-6所示。这个类中包含一个在滚动窗格中的文本域，还包含一个显示图像图标和标题的标签。现在的例子使用这个类，在后面的例子中还会复用这个类。

2) 扩展JFrame，创建一个名为TextAreaDemo的类，如程序清单17-7所示。创建DescriptionPanel的一个实例，并且将它添加到框架的中心位置。DescriptionPanel和TextAreaDemo之间的关系如图17-20所示。

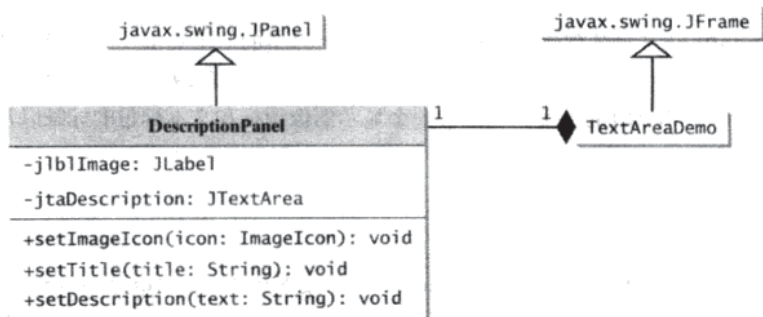


图17-20 TextAreaDemo类用DescriptionPanel类显示国旗的图像、标题及其文本描述

文本域在JScrollPane（第27行）内，它为此文本域提供滚动功能。如果文本的尺寸大于文本域的物理尺寸，就会自动出现滚动条；如果文本被删除之后剩余的文本尺寸小于文本域尺寸，那么滚动条就会消失。

属性lineWrap设置为true（第22行），这样，当文本超出一行时就会自动换行。属性wrapStyleWord设置为true（第23行），这样，就按单词而不是按字符换行。文本域设置为不可编辑的（第24行），所以不能在文本域中编辑描述文字。

在这个例子中，没必要为DescriptionPanel创建一个独立的类。而且，这个创建出来的类可以在17.8节中复用，17.8节将使用它为多种图像显示描述面板。

程序清单17-7 TextAreaDemo.java

```

1 import java.awt.*;
2 import javax.swing.*;
3
4 public class TextAreaDemo extends JFrame {
5     // Declare and create a description panel
6     private DescriptionPanel descriptionPanel = new DescriptionPanel();
7
8     public static void main(String[] args) {
9         TextAreaDemo frame = new TextAreaDemo();
10        frame.pack();
11        frame.setLocationRelativeTo(null); // Center the frame
12        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13        frame.setTitle("TextAreaDemo");
14        frame.setVisible(true);
15    }
16
17    public TextAreaDemo() {
18        // Set title, text and image in the description panel
19        descriptionPanel.setTitle("Canada");
20        String description = "The Maple Leaf flag \n\n" +
21            "The Canadian National Flag was adopted by the Canadian " +
22            "Parliament on October 22, 1964 and was proclaimed into law " +
23            "by Her Majesty Queen Elizabeth II (the Queen of Canada) on " +
24            "February 15, 1965. The Canadian Flag (colloquially known " +
25            "as The Maple Leaf Flag) is a red flag of the proportions " +

```



```

26     "two by length and one by width, containing in its center a " +
27     "white square, with a single red stylized eleven-point " +
28     "maple leaf centered in the white square.";
29     descriptionPanel.setDescription(description);
30     descriptionPanel.setIcon(new ImageIcon("image/ca.gif"));
31
32     // Add the description panel to the frame
33     setLayout(new BorderLayout());
34     add(descriptionPanel, BorderLayout.CENTER);
35 }
36 }

```

程序TextAreaDemo类只是创建了DescriptionPanel类的一个实例（第6行），然后在描述面板内设置描述面板的标题（第19行）、图像（第30行）以及文本（第29行）。DescriptionPanel是JPanel.DescriptionPanel的子类，它包含一个显示图像图标和文本标题的标签以及显示关于图像的描述的一个文本区域。

17.8 组合框

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。使用它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17-21列出在JComboBox类中一些常用的构造方法和方法。

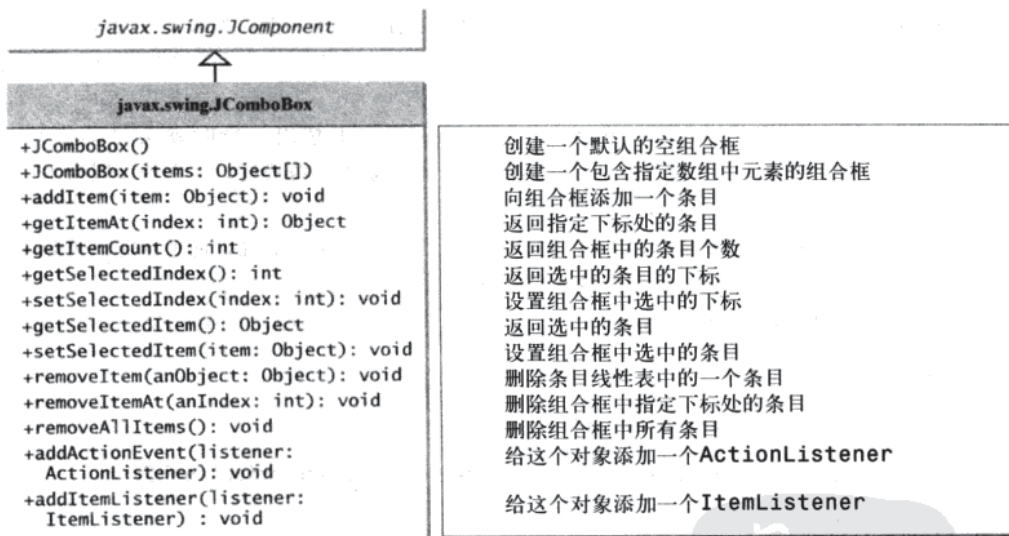


图17-21 JComboBox从条目集中选择一个条目

下面的语句创建一个有四个条目的组合框，前景色为红色，背景色为白色，然后选中第二个条目：

```

JComboBox jcb = new JComboBox(new Object[]
{ "Item 1", "Item 2", "Item 3", "Item 4" });
jcb.setForeground(Color.red);
jcb.setBackground(Color.white);
jcb.setSelectedItem("Item 2");

```

JComboBox可以在很多其他事件中激发ActionEvent和ItemEvent事件。每当选一个条目时，就会触发一个ActionEvent。每当选一个新条目时，JComboBox就会激发ItemEvent事件两次，一次是取消前一个选择的条目，另一次是选中当前选择的条目。注意，如果当前条目被重新选择，那么就没有ItemEvent触发。要响应ItemEvent事件，需要实现itemStateChanged(ItemEvent e)处理器来

处理选择。要从一个JComboBox菜单中获取数据，可以使用getSelectedItem()方法返回当前已选定的条目，或者使用e.getItem()方法从itemStateChanged(ItemEvent e)处理器中获取条目。

程序清单17-8给出的程序使用户通过选择组合框中的国家来查看某个国家国旗的图像及其描述，如图17-22所示。

程序清单17-8 ComboBoxDemo.java

```

1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4
5 public class ComboBoxDemo extends JFrame {
6     // Declare an array of Strings for flag titles
7     private String[] flagTitles = {"Canada", "China", "Denmark",
8         "France", "Germany", "India", "Norway", "United Kingdom",
9         "United States of America"};
10
11     // Declare an ImageIcon array for the national flags of 9 countries
12     private ImageIcon[] flagImage = {
13         new ImageIcon("image/ca.gif"),
14         new ImageIcon("image/china.gif"),
15         new ImageIcon("image/denmark.gif"),
16         new ImageIcon("image/fr.gif"),
17         new ImageIcon("image/germany.gif"),
18         new ImageIcon("image/india.gif"),
19         new ImageIcon("image/norway.gif"),
20         new ImageIcon("image/uk.gif"),
21         new ImageIcon("image/us.gif")
22     };
23
24     // Declare an array of strings for flag descriptions
25     private String[] flagDescription = new String[9];
26
27     // Declare and create a description panel
28     private DescriptionPanel descriptionPanel = new DescriptionPanel();
29
30     // Create a combo box for selecting countries
31     private JComboBox jcbo = new JComboBox(flagTitles);
32
33     public static void main(String[] args) {
34         ComboBoxDemo frame = new ComboBoxDemo();
35         frame.pack();
36         frame.setTitle("ComboBoxDemo");
37         frame.setLocationRelativeTo(null); // Center the frame
38         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
39         frame.setVisible(true);
40     }
41
42     public ComboBoxDemo() {
43         // Set text description
44         flagDescription[0] = "The Maple Leaf flag \n\n" +
45             "The Canadian National Flag was adopted by the Canadian " +
46             "Parliament on October 22, 1964 and was proclaimed into law " +
47             "by Her Majesty Queen Elizabeth II (the Queen of Canada) on " +
48             "February 15, 1965. The Canadian Flag (colloquially known " +
49             "as The Maple Leaf Flag) is a red flag of the proportions " +
50             "two by length and one by width, containing in its center a " +
51             "white square, with a single red stylized eleven-point " +
52             "maple leaf centered in the white square.";
53         flagDescription[1] = "Description for China ... ";
54         flagDescription[2] = "Description for Denmark ... ";

```

```

55  flagDescription[3] = "Description for France ... ";
56  flagDescription[4] = "Description for Germany ... ";
57  flagDescription[5] = "Description for India ... ";
58  flagDescription[6] = "Description for Norway ... ";
59  flagDescription[7] = "Description for UK ... ";
60  flagDescription[8] = "Description for US ... ";
61
62  // Set the first country (Canada) for display
63  setDisplay(0);
64
65  // Add combo box and description panel to the list
66  add(jcbo, BorderLayout.NORTH);
67  add(descriptionPanel, BorderLayout.CENTER);
68
69  // Register listener
70  jcbo.addItemListener(new ItemListener() {
71      /** Handle item selection */
72      public void itemStateChanged(ItemEvent e) {
73          setDisplay(jcbo.getSelectedIndex());
74      }
75  });
76  }
77
78  /** Set display information on the description panel */
79  public void setDisplay(int index) {
80      descriptionPanel.setTitle(flagTitles[index]);
81      descriptionPanel.setIcon(flagImage[index]);
82      descriptionPanel.setDescription(flagDescription[index]);
83  }
84  }

```

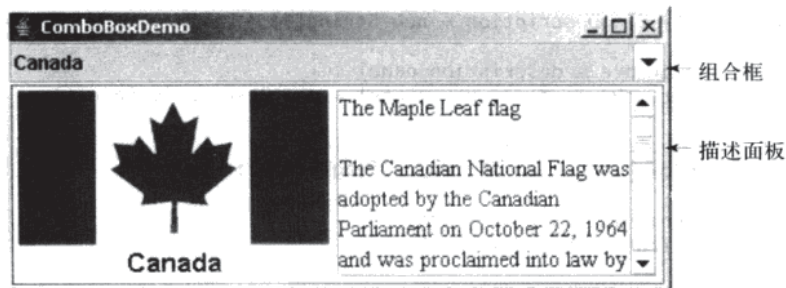


图17-22 选中组合框中的一个国家后，就显示该国的信息，包括国旗的图像及其描述

下面是这个程序的主要步骤：

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DescriptionPanel对象。DescriptionPanel类在前一个例子中介绍过。将组合框放置在框架的北区，而将描述面板放置在框架的中央。

2) 处理事件。

创建一个监听器来实现itemStateChanged处理器，在描述面板中为所选择的国家名设置国旗名、国旗图像及其描述。

监听器监听组合框触发的ItemEvent事件，并实现ItemListener（第70~75行）。可以不用ItemEvent事件，而是使用ActionEvent事件来改写这个程序来处理组合框条目的选择。

程序将国旗信息存储在三个数组：flagTitles、flagImage和flagDescription（第7~25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DescriptionPanel类的一个实例（第28行），该类在程序清单17-6中。程序以数组flagTitles中的值为初值创建一个组合框（第31行）。当用户选择组合框中的一项后，执行itemStateChanged处理器，确定选项的序号并在面板上设置相应的国旗名、国旗图像及国旗描述。

17.9 列表框

列表框（list）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。Swing组件JList的功能非常丰富。图17-23列出了JList中一些常用的构造方法和方法。

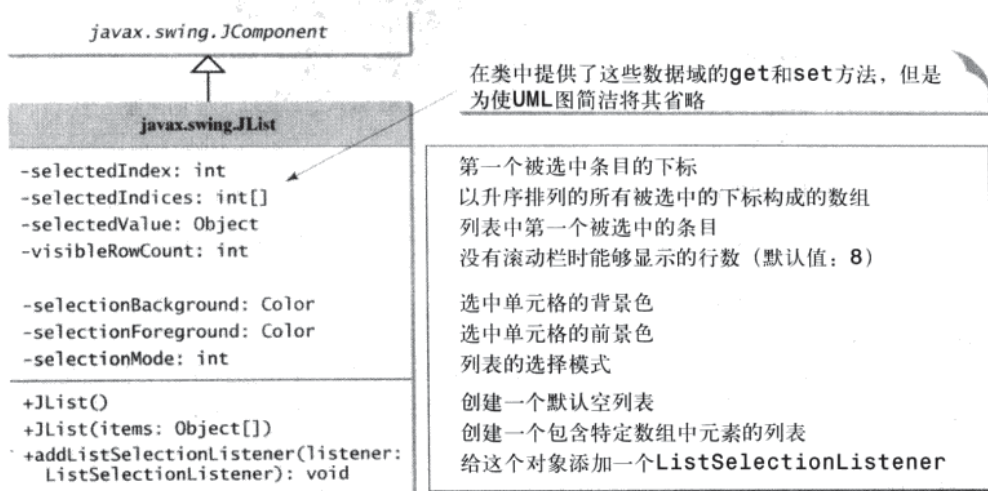


图17-23 JList允许从一个条目集合中选择多个条目

选择模式属性selectionMode是三个在javax.swing.ListSelectionModel类中定义的数值之一（SINGLE_SELECTION、SINGLE_INTERVAL_SELECTION、MULTIPLE_INTERVAL_SELECTION），它们分别表示选择的是单项选择、单区间选择还是多区间选择。单项选择只允许选择一项。单区间选择允许选择多项，但是选定的项必须是连续的。多区间选择允许选择多种连续项，没有限制，如图17-24所示。此属性的默认值为MULTIPLE_INTERVAL_SELECTION。



图17-24 JList有三种选择模式：单项选择、单区间选择和多区间选择

下面的语句创建有6个条目的列表框，前景色为红色，背景色为白色，选定单元格的前景色为粉色，选定单元格的背景色为黑色，可见行数为4行。

```

JList jlst = new JList(new Object[]
{ "Item 1", "Item 2", "Item 3", "Item 4", "Item 5", "Item 6" });
jlst.setForeground(Color.RED);
jlst.setBackground(Color.WHITE);
jlst.setSelectionForeground(Color.PINK);
jlst.setSelectionBackground(Color.BLACK);
jlst.setVisibleRowCount(4);
  
```

列表不能自动滚动。为了使一个列表能够滚动，创建一个滚动窗格并将该列表框添加到这个窗格中。

JList触发`javax.swing.event.ListSelectionEvent`事件，通知用于处理选择的监听器。监听器必须实现`javax.swing.event.ListSelectionListener`接口中的`valueChanged`处理器来处理这个事件。

程序清单17-9给出的程序让用户在列表中选择国家，并且在标签中显示选中国家的国旗。图17-25显示该程序的一个运行示例。

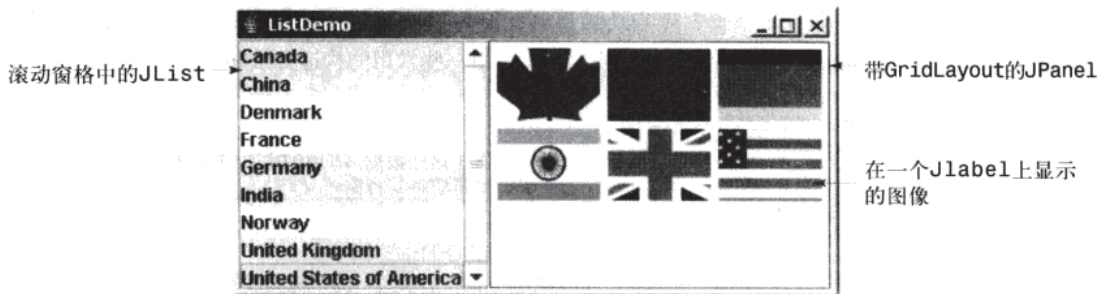


图17-25 选中列表框中的国家后，在标签中显示对应国旗的图像

下面是这个程序的主要步骤：

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窗格中。将滚动窗格放到框架的西边。创建九个标签用来显示这九个国家的国旗图像。将标签放在面板中，再将面板放在框架的中央。

2) 处理事件。

创建一个监听器，实现`ListSelectionListener`接口中的`valueChanged`方法，在标签中放置选定国家的国旗图像。

程序清单17-9 ListDemo.java

```

1 import java.awt.*;
2 import javax.swing.*;
3 import javax.swing.event.*;
4
5 public class ListDemo extends JFrame {
6     final int NUMBER_OF_FLAGS = 9;
7
8     // Declare an array of Strings for flag titles
9     private String[] flagTitles = {"Canada", "China", "Denmark",
10         "France", "Germany", "India", "Norway", "United Kingdom",
11         "United States of America"};
12
13     // The list for selecting countries
14     private JList jlst = new JList(flagTitles);
15
16     // Declare an ImageIcon array for the national flags of 9 countries
17     private ImageIcon[] imageIcons = {
18         new ImageIcon("image/ca.gif"),
19         new ImageIcon("image/china.gif"),
20         new ImageIcon("image/denmark.gif"),
21         new ImageIcon("image/fr.gif"),
22         new ImageIcon("image/germany.gif"),
23         new ImageIcon("image/india.gif"),
24         new ImageIcon("image/norway.gif"),
25         new ImageIcon("image/uk.gif"),
26         new ImageIcon("image/us.gif")
27     };
28

```



```

29 // Arrays of labels for displaying images
30 private JLabel[] jlblImageViewer = new JLabel[NUMBER_OF_FLAGS];
31
32 public static void main(String[] args) {
33     ListDemo frame = new ListDemo();
34     frame.setSize(650, 500);
35     frame.setTitle("ListDemo");
36     frame.setLocationRelativeTo(null); // Center the frame
37     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
38     frame.setVisible(true);
39 }
40
41 public ListDemo() {
42     // Create a panel to hold nine labels
43     JPanel p = new JPanel(new GridLayout(3, 3, 5, 5));
44
45     for (int i = 0; i < NUMBER_OF_FLAGS; i++) {
46         p.add(jlblImageViewer[i] = new JLabel());
47         jlblImageViewer[i].setHorizontalAlignment
48             (SwingConstants.CENTER);
49     }
50
51     // Add p and the list to the frame
52     add(p, BorderLayout.CENTER);
53     add(new JScrollPane(jlst), BorderLayout.WEST);
54
55     // Register listeners
56     jlst.addListSelectionListener(new ListSelectionListener() {
57         /** Handle list selection */
58         public void valueChanged(ListSelectionEvent e) {
59             // Get selected indices
60             int[] indices = jlst.getSelectedIndices();
61
62             int i;
63             // Set icons in the labels
64             for (i = 0; i < indices.length; i++) {
65                 jlblImageViewer[i].setIcon(imageIcons[indices[i]]);
66             }
67
68             // Remove icons from the rest of the labels
69             for (; i < NUMBER_OF_FLAGS; i++) {
70                 jlblImageViewer[i].setIcon(null);
71             }
72         }
73     });
74 }
75 }

```

为处理列表框中选定的国家名，匿名内部类监听器监听ListSelectionEvent（第56~73行）。ListSelectionEvent和ListSelectionListener都定义在javax.swing.event包中，因此，程序需要导入这个包（第3行）。

这个程序创建包含九个标签的数组，用来显示九个国家的国旗图像。程序将九个国家国旗的图像装入一个图像数组中（第17~27行），并创建一个与图像数组顺序相同的九国国名列表框（第9~11行）。这样，图像数组的下标0就对应于列表框的第一个国家。

列表框放置在一个滚动窗格中（第53行），这样，当列表框中的项数超出视图域时，列表框是可以滚动的。

默认情况下，列表框的选择模式是多区间选择，它允许用户在列表框中从不同块中选择多个选项。当用户在列表框中选择国家时，执行valueChanged处理器（第58~73行），它获取选定项的下标并给标签设置相应的图像图标以显示国旗。

17.10 滚动条

滚动条 (JScrollBar) 是一个允许用户从一个值的范围中进行选择的组件, 如图17-26所示。

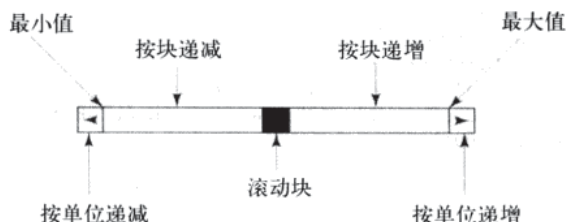
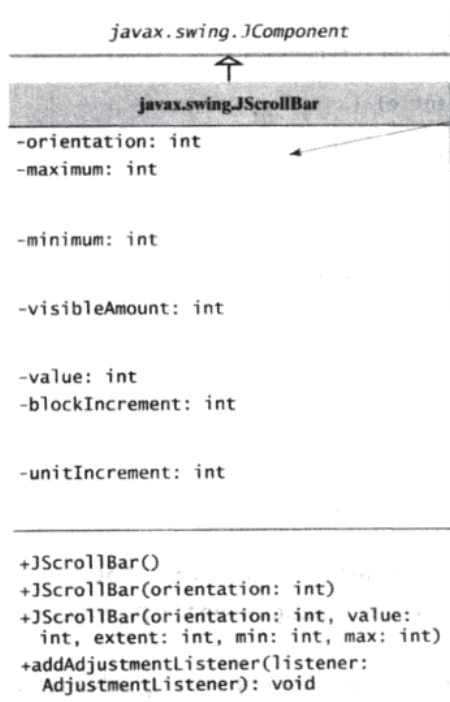


图17-26 滚动条可以用图形表示一个值的范围

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, 或者点击滚动条的单位递增区或块递增区。键盘操作也可改变滚动条的值。习惯上, Page Up键和Page Down键相当于点击块增加区和块减少区。

注意 滚动条轨迹的宽度为`maximum+visibleAmount`。当滚动条设置为它的最大值时, 滚动块的左侧就在`maximum`处, 而右侧在`maximum+visibleAmount`处。

JScrollBar有以下属性, 如图17-27所示。



在类中提供了这些数据域的get和set方法, 但是为使UML图简洁将其省略

- 指定水平和垂直方式, 默认值是水平的
- 指定滚动条的最大值, 水平风格时它表示滚动块达到滚动条的最右端, 垂直风格时它表示滚动块达到滚动条的最底端
- 指定滚动条的最小值, 水平风格时它表示滚动块达到滚动条的最左端, 垂直风格时它表示滚动块达到滚动条的最顶端
- 指定滚动条中滚动块的相对宽度。屏幕上出现的实际宽度是由最大值和`visibleAmount`值确定的
- 表示滚动条的当前值
- 指定当用户激活滚动条的按块递增 (递减) 域时所增加 (减小) 的值, 如图17-26所示
- 指定当用户激活滚动条的按单位递增 (递减) 域时所增加 (减小) 的值, 如图17-26所示
- 创建一个默认的垂直滚动条
- 创建一个带指定方向的滚动条
- 创建一个带指定方向、指定值、指定范围、指定最小值和最大值的滚动条
- 给这个对象添加一个AdjustmentListener

图17-27 JScrollBar允许在一个值的范围中进行选择

当用户改变滚动条的值时, 滚动条会触发一个AdjustmentEvent的实例, 传递给每一个已经注册的监听器。如果一个对象希望滚动条的值发生变化时能够通知它, 就必须实现`java.awt.event.AdjustmentListener`监听器接口中的`adjustmentValueChanged`方法。

程序清单17-10给出使用水平滚动条和垂直滚动条来控制面板上显示一条消息的程序。水平滚动条用以左右移动消息, 而垂直滚动条用以上下移动消息。程序的运行示例如图17-28所示。

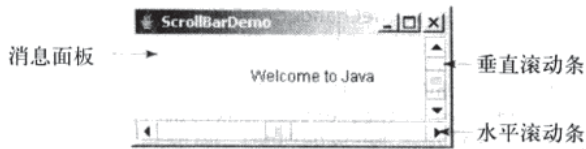


图17-28 滚动条在面板上水平和垂直移动消息

下面是程序的主要步骤：

1) 创建用户界面。

创建一个MessagePanel对象，将它放置于框架的中央。创建一个垂直滚动条，将它放到框架的东边。创建一个水平滚动条，将它放到框架的南边。

2) 处理事件。

创建一个监听器实现adjustmentValueChanged处理器，根据滚动块在滚动条中的移动来移动消息。

程序清单17-10 ScrollBarDemo.java

```

1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4
5 public class ScrollBarDemo extends JFrame {
6     // Create horizontal and vertical scroll bars
7     private JScrollBar jscbHort =
8         new JScrollBar(JScrollBar.HORIZONTAL);
9     private JScrollBar jscbVert =
10        new JScrollBar(JScrollBar.VERTICAL);
11
12    // Create a MessagePanel
13    private MessagePanel messagePanel =
14        new MessagePanel("Welcome to Java");
15
16    public static void main(String[] args) {
17        ScrollBarDemo frame = new ScrollBarDemo();
18        frame.setTitle("ScrollBarDemo");
19        frame.setLocationRelativeTo(null); // Center the frame
20        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21        frame.pack();
22        frame.setVisible(true);
23    }
24
25    public ScrollBarDemo() {
26        // Add scroll bars and message panel to the frame
27        setLayout(new BorderLayout());
28        add(messagePanel, BorderLayout.CENTER);
29        add(jscbVert, BorderLayout.EAST);
30        add(jscbHort, BorderLayout.SOUTH);
31
32        // Register listener for the scroll bars
33        jscbHort.addAdjustmentListener(new AdjustmentListener() {
34            public void adjustmentValueChanged(AdjustmentEvent e) {
35                // getValue() and getMaximumValue() return int, but for better
36                // precision, use double
37                double value = jscbHort.getValue();
38                double maximumValue = jscbHort.getMaximum();
39                double newX = (value * messagePanel.getWidth() /
40                    maximumValue);
41                messagePanel.setXCoordinate((int)newX);
42            }
43        });
44        jscbVert.addAdjustmentListener(new AdjustmentListener() {

```

```

45     public void adjustmentValueChanged(AdjustmentEvent e) {
46         // getValue() and getMaximumValue() return int, but for better
47         // precision, use double
48         double value = jscbVert.getValue();
49         double maximumValue = jscbVert.getMaximum();
50         double newY = (value * messagePanel.getHeight() /
51             maximumValue);
52         messagePanel.setYCoordinate((int)newY);
53     }
54 }
55 }
56 }

```

程序创建两个滚动条（`jscbVert`和`jscbHort`）（第7~10行）和一个`MessagePanel`实例（`messagePanel`）（第13~14行）。将`messagePanel`放置在框架的中央，将`jscbVert`和`jscbHort`分别放置在框架的东区和南区（第29~30行）。

可以在构造方法中指定滚动条的方向，也可以使用`setOrientation`方法来指定。默认情况下，属性`maximum`的值为100，`minimum`的值为0，`blockIncrement`的值为10，而`visibleAmount`的值为10。

当用户拖动滚动块或点击单位增加区、单位减少区时，滚动条的值将改变。`AdjustmentEvent`事件的一个实例被触发，并通过调用`adjustmentValueChanged`处理器传递给监听器。垂直滚动条的监听器向上、向下移动消息（第33~43行），水平滚动条的监听器向左、向右移动消息（第44~54行）。

垂直滚动条的最大值对应为面板的高度，而水平滚动条的最大值对应为面板的宽度。水平滚动条的当前值与最大值之间的比率等于`x`的值与消息面板的宽度的比值。类似地，垂直滚动条的当前值与最大值之间的比率等于`y`的值与消息面板的高度的比值。随着滚动条的调整，相应地设置消息面板的`x`坐标和`y`坐标（第39和第50行）。

17.11 滑块

`JSlider`与`JScrollBar`类似，但是`JSlider`具有更多的属性，并且可以以多种形式显示。图17-29显示两个滑块。

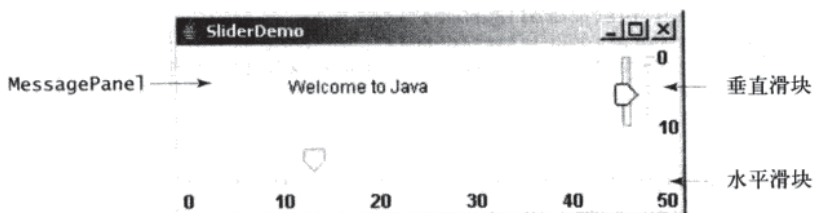


图17-29 滑块在面板上沿水平方向和垂直方向移动消息

`JSlider`允许用户以图形方式通过滑动按钮，在一个有界的区间中选择一个数值。滑块可以在主要刻度以及次要刻度之间滑动。刻度之间的像素值是由`majorTickSpacing`和`minorTickSpacing`属性控制的。滑块可以水平显示也可以垂直显示，可以带也可以不带刻度，可以有标签也可以没有。`JSlider`中经常使用的构造方法和属性如图17-30所示。

注意 垂直滚动条的值从上向下是增加的，但是默认情况下，垂直滚动条的值从上向下是减少的。

注意 图17-30中列出的所有属性都有相关的`get`和`set`方法，为了简洁可以忽略它们。习惯上，布尔属性的`get`方法命名为`is<PropertyName>()`。但是在`JSlider`类中，`paintLabels`、`paintTicks`、`paintTrack`和`inverted`的`get`方法分别命名为`getPaintLabels()`、`getPaintTicks()`、`getPaintTrack()`和`getInverted()`，这违反了Java的命名习惯。

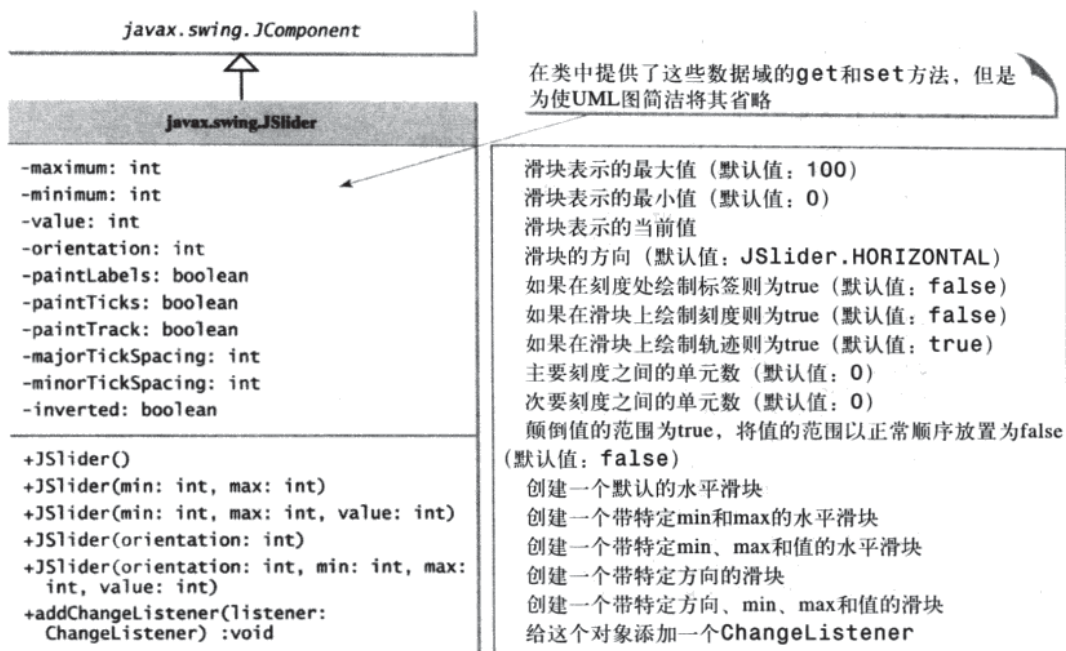


图17-30 JSlider允许从某个范围的值中进行选择

当用户改变滑块的值时，滑块就会触发javax.swing.event.ChangeEvent的一个实例，可以把它传递给任何已经注册的监听器。如果一个对象希望滑块的值发生变化时能够通知它，那就必须实现定义在包java.swing.event中的ChangeListener接口中的stateChanged方法。

程序清单17-11编写程序，使用滑块来控制面板上显示一条消息，如图17-29所示。下面是程序的主要步骤：

1) 创建用户界面。

创建一个MessagePanel对象，然后将它放置在框架的中央。创建一个垂直滑块，将它放置在框架的东边。创建一个水平滑块，将它放置在框架的南边。

2) 处理事件。

创建一个监听器来实现ChangeListener接口中的stateChanged处理器，根据滑块在滑动区中的移动来移动这条消息。

程序清单17-11 SliderDemo.java

```

1 import java.awt.*;
2 import javax.swing.*;
3 import javax.swing.event.*;
4
5 public class SliderDemo extends JFrame {
6     // Create horizontal and vertical sliders
7     private JSlider jsldHort = new JSlider(JSlider.HORIZONTAL);
8     private JSlider jsldVert = new JSlider(JSlider.VERTICAL);
9
10    // Create a MessagePanel
11    private MessagePanel messagePanel =
12        new MessagePanel("Welcome to Java");
13
14    public static void main(String[] args) {
15        SliderDemo frame = new SliderDemo();
16        frame.setTitle("SliderDemo");
17        frame.setLocationRelativeTo(null); // Center the frame
18        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

```

```

19     frame.pack();
20     frame.setVisible(true);
21 }
22
23 public SliderDemo() {
24     // Add sliders and message panel to the frame
25     setLayout(new BorderLayout(5, 5));
26     add(messagePanel, BorderLayout.CENTER);
27     add(jsldVert, BorderLayout.EAST);
28     add(jsldHort, BorderLayout.SOUTH);
29
30     // Set properties for sliders
31     jsldHort.setMaximum(50);
32     jsldHort.setPaintLabels(true);
33     jsldHort.setPaintTicks(true);
34     jsldHort.setMajorTickSpacing(10);
35     jsldHort.setMinorTickSpacing(1);
36     jsldHort.setPaintTrack(false);
37     jsldVert.setInverted(true);
38     jsldVert.setMaximum(10);
39     jsldVert.setPaintLabels(true);
40     jsldVert.setPaintTicks(true);
41     jsldVert.setMajorTickSpacing(10);
42     jsldVert.setMinorTickSpacing(1);
43
44     // Register listener for the sliders
45     jsldHort.addChangeListener(new ChangeListener() {
46         /** Handle scroll-bar adjustment actions */
47         public void stateChanged(ChangeEvent e) {
48             // getValue() and getMaximumValue() return int, but for better
49             // precision, use double
50             double value = jsldHort.getValue();
51             double maximumValue = jsldHort.getMaximum();
52             double newX = (value * messagePanel.getWidth() /
53                 maximumValue);
54             messagePanel.setXCoordinate((int)newX);
55         }
56     });
57     jsldVert.addChangeListener(new ChangeListener() {
58         /** Handle scroll-bar adjustment actions */
59         public void stateChanged(ChangeEvent e) {
60             // getValue() and getMaximumValue() return int, but for better
61             // precision, use double
62             double value = jsldVert.getValue();
63             double maximumValue = jsldVert.getMaximum();
64             double newY = (value * messagePanel.getHeight() /
65                 maximumValue);
66             messagePanel.setYCoordinate((int) newY);
67         }
68     });
69 }
70 }

```

JSlider与JScrollBar类似，但是JSlider具有更多的特性。如本例所示，可以在JSlider上指定最大值、标签、主要标记和次要标记（第31~35行）。也可以选择隐藏轨迹（第36行）。由于垂直滑块的值从上向下是递减的，所以，可以使用setInverted将其顺序颠倒过来（第37行）。

改变滑块的值时，JSlider会触发ChangeEvent事件。监听器需要实现ChangeListener接口中的stateChanged处理器（第45~68行）。注意，当调整滚动条时，JScrollBar会触发AdjustmentEvent事件。

17.12 创建多个窗口

有时候,可能希望在一个应用程序中创建多个窗口。这个程序会新开一个窗口执行某个指定的任务。新开的窗口称为子窗口 (subwindow), 而主框架称作主窗口 (main window)。

为了从应用程序中创建一个子窗口, 需要定义一个指明任务的JFrame的子类, 并且告诉新窗口做什么。然后, 可以在应用程序中创建该子类的一个实例, 通过把它设为可见的框架实例即可出现一个新窗口。

程序清单17-12给出的程序创建一个主窗口, 它有一个在滚动窗格中的文本域和一个名为Show Histogram (显示直方图) 的按钮。当用户点击这个按钮时, 就会出现一个显示一个直方图的新窗口, 这个直方图显示文本域中字母出现的次数。图17-31中包含这个程序的一个运行示例。

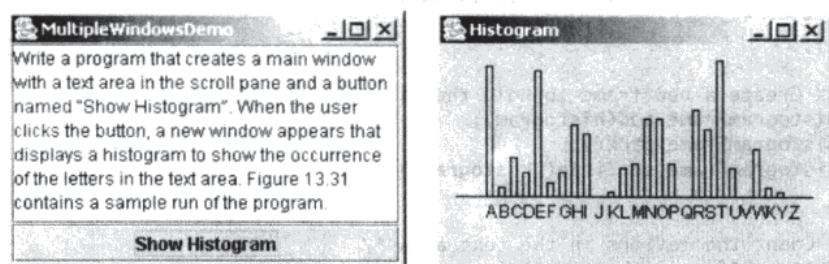


图17-31 在一个独立的框架中显示直方图

下面是这个程序的主要步骤:

1) 在程序清单17-12中, 为一个名为MultipleWindowsDemo的框架创建一个主类。在滚动窗格中添加一个文本域, 然后将这个滚动窗格放置在框架的中央。创建一个按钮Show Histogram, 然后将它放到框架的南边。

2) 在程序清单17-13中创建一个名为Histogram的JPanel类的子类。这个类包含一个名为count的int[]类型的数据域, 它是用来统计26个字母的出现次数的。在直方图中显示count的值。

3) 实现MultipleWindowsDemo中的actionPerformed处理器, 如下所示:

① 创建一个Histogram的实例, 统计文本域中字母的出现次数, 然后将统计结果传递给Histogram对象。

② 创建一个新框架, 然后将Histogram对象放到框架的中央。显示这个框架。

程序清单17-12 MultipleWindowsDemo.java

```

1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4
5 public class MultipleWindowsDemo extends JFrame {
6     private JTextArea jta;
7     private JButton jbtShowHistogram = new JButton("Show Histogram");
8     private Histogram histogram = new Histogram();
9
10    // Create a new frame to hold the histogram panel
11    private JFrame histogramFrame = new JFrame();
12
13    public MultipleWindowsDemo() {
14        // Store text area in a scroll pane
15        JScrollPane scrollPane = new JScrollPane(jta = new JTextArea());
16        scrollPane.setPreferredSize(new Dimension(300, 200));
17        jta.setWrapStyleWord(true);
18        jta.setLineWrap(true);
19
20        // Place scroll pane and button in the frame

```

```

21 add(scrollPane, BorderLayout.CENTER);
22 add(jbtShowHistogram, BorderLayout.SOUTH);
23
24 // Register listener
25 jbtShowHistogram.addActionListener(new ActionListener() {
26     /** Handle the button action */
27     public void actionPerformed(ActionEvent e) {
28         // Count the letters in the text area
29         int[] count = countLetters();
30
31         // Set the letter count to histogram for display
32         histogram.showHistogram(count);
33
34         // Show the frame
35         histogramFrame.setVisible(true);
36     }
37 });
38
39 // Create a new frame to hold the histogram panel
40 histogramFrame.add(histogram);
41 histogramFrame.pack();
42 histogramFrame.setTitle("Histogram");
43 }
44
45 /** Count the letters in the text area */
46 private int[] countLetters() {
47     // Count for 26 letters
48     int[] count = new int[26];
49
50     // Get contents from the text area
51     String text = jta.getText();
52
53     // Count occurrences of each letter (case insensitive)
54     for (int i = 0; i < text.length(); i++) {
55         char character = text.charAt(i);
56
57         if ((character >= 'A') && (character <= 'Z')) {
58             count[character - 'A']++;
59         }
60         else if ((character >= 'a') && (character <= 'z')) {
61             count[character - 'a']++;
62         }
63     }
64
65     return count; // Return the count array
66 }
67
68 public static void main(String[] args) {
69     MultipleWindowsDemo frame = new MultipleWindowsDemo();
70     frame.setLocationRelativeTo(null); // Center the frame
71     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
72     frame.setTitle("MultipleWindowsDemo");
73     frame.pack();
74     frame.setVisible(true);
75 }
76 }

```

程序清单17-13 Histogram.java

```

1 import javax.swing.*;
2 import java.awt.*;
3
4 public class Histogram extends JPanel {
5     // Count the occurrences of 26 letters
6     private int[] count;
7

```



```

8  /** Set the count and display histogram */
9  public void showHistogram(int[] count) {
10     this.count = count;
11     repaint();
12 }
13
14 /** Paint the histogram */
15 protected void paintComponent(Graphics g) {
16     if (count == null) return; // No display if count is null
17
18     super.paintComponent(g);
19
20     // Find the panel size and bar width and interval dynamically
21     int width = getWidth();
22     int height = getHeight();
23     int interval = (width - 40) / count.length;
24     int individualWidth = (int)((width - 40) / 24) * 0.60);
25
26     // Find the maximum count. The maximum count has the highest bar
27     int maxCount = 0;
28     for (int i = 0; i < count.length; i++) {
29         if (maxCount < count[i])
30             maxCount = count[i];
31     }
32
33     // x is the start position for the first bar in the histogram
34     int x = 30;
35
36     // Draw a horizontal base line
37     g.drawLine(10, height - 45, width - 10, height - 45);
38     for (int i = 0; i < count.length; i++) {
39         // Find the bar height
40         int barHeight =
41             (int)((double)count[i] / (double)maxCount * (height - 55));
42
43         // Display a bar (i.e. rectangle)
44         g.drawRect(x, height - 45 - barHeight, individualWidth,
45             barHeight);
46
47         // Display a letter under the base line
48         g.drawString((char)(65 + i) + "", x, height - 30);
49
50         // Move x for displaying the next character
51         x += interval;
52     }
53 }
54
55 /** Override getPreferredSize */
56 public Dimension getPreferredSize() {
57     return new Dimension(300, 300);
58 }
59 }

```

程序包含两个类MultipleWindowsDemo和Histogram，它们的关系如图17-32所示。

MultipleWindowsDemo是一个框架，它包括一个在滚动窗格中的文本域以及一个按钮。Histogram是JPanel的子类，用于显示文本域中字母出现次数的直方图。

当用户点击按钮Show Histogram后，处理器会统计文本域中每个字母出现的次数。统计字母时不区分大小写。不统计非字母的字符。统计结果存储在一个26个元素组成的int型数组中。数组中的第一个元素存放的是字母'a'或'A'的个数，最后一个元素存放的是'z'或'Z'的个数。数组count的值传递给显示字母统计情况的直方图。

MultipleWindowsDemo类包含一个main方法。这个main方法创建MultipleWindowsDemo的一个

实例并显示框架。MultipleWindowsDemo类还包含一个名为histogramFrame的JFrame实例，而histogramFrame又包含了一个Histogram的实例。当用户点击按钮Show Histogram之后，histogramFrame被设置为可见的，用以显示直方图。

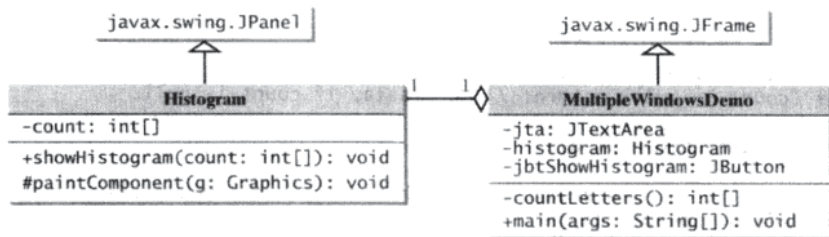


图17-32 MultipleWindowsDemo使用Histogram在框架中显示一个直方图，
该图表示文本域中字母出现的次数

直方图中块的高度和宽度是根据直方图窗口的大小动态决定的。

不能把JFrame的实例添加到容器中。例如，将histogramFrame添加到主框架会导致一个运行时异常。不过，创建一个框架实例并将它设置为可见，就可以出现一个新窗口。

本章小结

- 学习如何使用像JButton、JCheckBox、JRadioButton、JLabel、JTextField、JTextArea、JComboBox、JList、JScrollBar和JSlider这样的Swing GUI组件来创建图形用户界面。学习如何处理这些组件上的事件。
- 可以在按钮（JButton、JCheckBox和JRadioButton）和标签（JLabel）上显示文本和图标。

复习题

17.2~17.4节

- 17.1 如何创建一个标有OK的按钮？如何改变按钮上的文本？如何在按钮上设置一个图标、按下图标和翻转图标？
- 17.2 假定一个JButton对象jbtOK，编写语句设置按钮的前景色为红色、背景色为黄色、热键为'K'、工具提示文本为"Click OK to proceed"、水平对齐方式为RIGHT、垂直对齐方式为BOTTOM、水平文本位置为LEFT、垂直文本位置为TOP，而图标的文本间隔为5。
- 17.3 如何创建一个复选框？如何创建一个初始状态是选中状态的复选框？如何确定复选框是否选中？
- 17.4 如何创建一个单选按钮？如何创建一个初始状态是选中状态的单选按钮？如何把单选按钮组织在一起？如何确定单选按钮是否选中？
- 17.5 列出定义在AbstractButton类中定义的至少五个属性。

17.5~17.9节

- 17.6 如何创建一个命名为"Address"的标签？如何改变标签上的名字？如何在标签上设置一个图标？
- 17.7 假设有一个JLabel对象jlblMap，编写语句设置标签的前景色为红色、背景色为黄色、热键为'K'、工具提示文本为"Map image"、水平对齐方式为RIGHT、垂直对齐方式为BOTTOM、水平文本位置为LEFT、垂直文本位置为TOP，而图标的文本间隔为5。
- 17.8 如何创建一个10列的文本域，并且将默认文本设置为"Welcome to Java"？如何在文本域中添加代码以检验文本域是否为空？
- 17.9 如何创建一个10行20列的文本域？如何在文本域中插入三行代码？如何创建一个可滚动的文本域？
- 17.10 如何创建一个组合框，向它添加三个条目，并再取一个选定项？

17.11 如何用一个字符串数组创建列表框?

17.10~17.12节

17.12 如何创建一个水平的滚动条? 滚动条可以触发什么事件?

17.13 如何创建一个垂直的滑块? 滑块可以触发什么事件?

17.14 解释如何在一个应用程序中创建和显示多个框架。

编程练习题

教学注意 教师可以将第18章的练习题作为本章的练习题布置给学生, 要求学生编写Java应用程序, 而不是编写Java applet。

17.2~17.5节

*17.1 (修改程序清单17-2) 改写程序清单17-2, 添加一组选择背景颜色的单选按钮。供选择的颜色有红色、黄色、白色、灰色和绿色 (参见图17-33)。

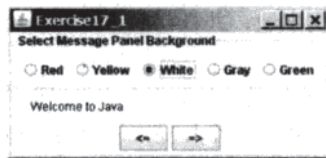


图17-33 <=和>=按钮在面板上移动消息, 也可以为消息设置背景色

*17.2 (选择几何图形) 编写一个绘制各种几何图形的程序, 如图17-34所示。用户从单选按钮中选择一个几何图形, 并且在复选框中确认是否被填充 (提示, 使用程序清单15-3中介绍的FigurePanel类来显示一个图形)。

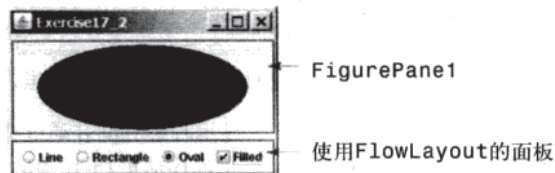


图17-34 选择一个图形时, 程序就会显示直线、矩形和椭圆

**17.3 (交通信号灯) 编写一个程序, 模拟交通信号灯。程序让用户从红、黄、绿三色灯中选择一种。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17-35所示)。程序开始时所有的灯都不亮。



图17-35 单选框放在一组中, 在组中只能选择一种颜色来控制交通信号灯

17.6~17.10节

**17.4 (文本浏览器) 编写一个程序在文本域中显示一个文本文件, 如图17-36所示。用户在文本域中输入一个文件名, 然后点击View按钮; 随后在文本域中会显示这个文件。

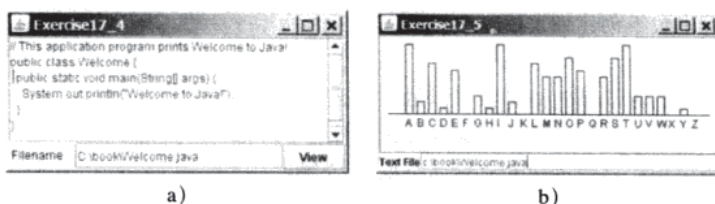


图17-36 a) 程序将文件中的文本显示到文本域;

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

****17.5 (创建表示字母出现次数的直方图)** 在程序清单17-12中, 开发的程序显示一个直方图, 它显示的是文本区域中每个字母出现的次数。复用程序清单17-12中的Histogram类来编写一个在面板上显示直方图的程序。直方图应该显示文本文件中每个字母出现的次数, 如图17-36b所示。假设字母是不区分大小写的。

(1) 将显示直方图的面板放置在框架的中央位置。

(2) 在面板中放置一个标签和文本域, 并且将面板放在框架的南边。文本文件将从这个文本域输入。

(3) 在文本域中点击回车键后, 程序就会计算每个字母出现的次数, 并且用直方图显示。

***17.6 (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17-37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。

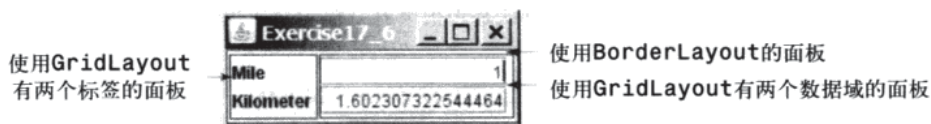


图17-37 程序可以将英里转换成公里, 反之亦然

***17.7 (设置时钟的时间)** 编写一个程序, 显示时钟时间并通过在三个文本域中输入小时、分钟和秒来设置时钟的时间, 如图17-38所示。使用程序清单15-10中的StillClock类。

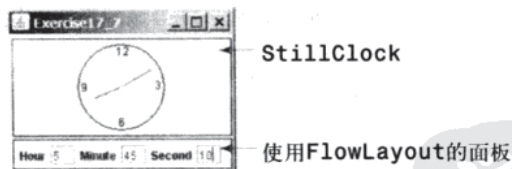


图17-38 程序显示字体域确定的时间

****17.8 (选择一种字体)** 编写一个程序, 可以动态地改变面板上显示的消息的字体。这个消息可以同时以粗体和斜体显示, 也可以在面板上居中显示。可以从组合框中选择字体名和字体大小, 如图17-39所示。使用GraphicsEnvironment类中的getAvailableFontFamilyNames()方法可以得到可用的字体名(12.8节)。字体大小的组合框初始化为从1到100之间的数字。

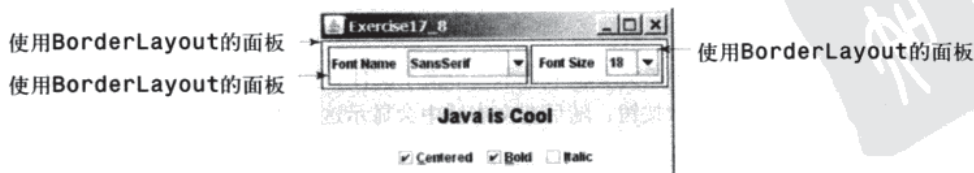


图17-39 可以动态设置消息的字体

****17.9** (演示JLabel的属性) 编写一个程序, 允许用户动态地设置属性: 水平对齐方式horizontalAlignment、垂直对齐方式verticalAlignment、水平文本对齐方式horizontalTextAlignment和垂直文本对齐方式verticalTextAlignment, 如图17-40所示。

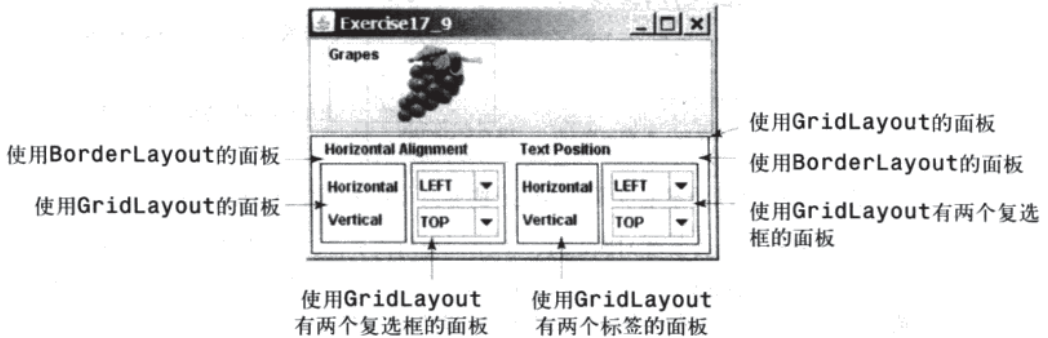


图17-40 可以动态地设置按钮的对齐方式以及按钮文本的位置属性

***17.10** (逐步给程序清单17-2增加新特性) 如下所示, 逐步改进程序清单17-2 (如图17-41):

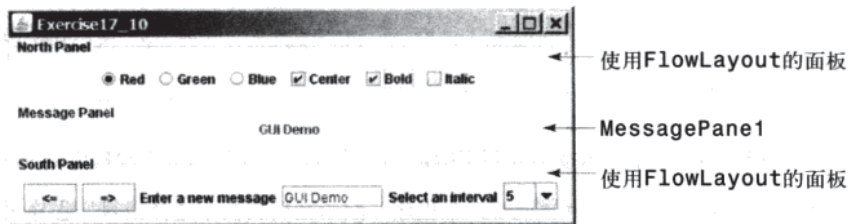


图17-41 程序使用按钮、标签、文本域、组合框、单选按钮、复选框和边框

(1) 在放置按钮的同一个面板上添加标签为"Enter a new message"的文本域。在该文本域中输入新消息之后按下回车键, 新消息就会显示在消息面板中。

(2) 在放置按钮的同一个面板上添加标签为"Select an interval"的组合框。这个组合框允许用户选择移动消息的新间隔。该值的选择范围从5到100, 间隔为5。用户也可以在组合框中输入新的间隔值。

(3) 添加三个单选按钮, 用户可以使用它们选择消息的前景色为红色、绿色和蓝色。这些单选按钮组成一组放在一个面板中, 这个面板放置在框架的内容窗格的北边。

(4) 添加三个复选框, 用户可以使用它们来设置消息居中, 并且用粗体或斜体显示消息。将这些复选框和单选按钮放在同一个面板中。

(5) 给消息面板添加标题为"Message Panel"的边框, 给按钮的面板添加标题为"South Panel"的边框, 给单选按钮和复选框所在的面板添加标题为"North Panel"的边框。

***17.11** (演示JTextField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17-42所示。

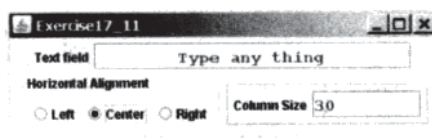


图17-42 可以动态地设置文本域的水平对齐属性和列宽属性

*17.12 (演示JTextArea的属性) 编写一个程序, 演示文本域的换行方式。程序用复选框选择文本域是否自动换行。在文本区域可以换行的情况下, 需要指定按单词还是按字符换行, 如图17-43所示。

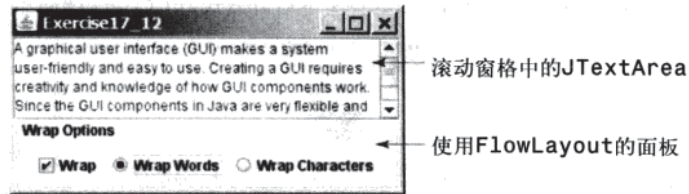


图17-43 可以动态地设置文本域是按单词还是按字符换行的选项

*17.13 (比较不同利率的贷款) 改写练习题4.21, 创建一个用户界面, 如图17-44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, 然后, 在文本域中显示针对每种利率的月偿还额和总偿还额, 利率从5%到8%, 按1/8 (0.125%) 递增。

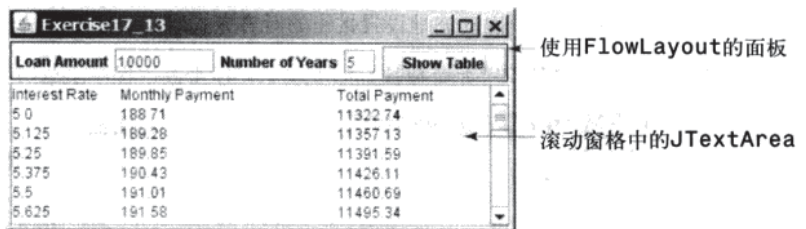


图17-44 程序显示按不同利率给定贷款的月偿还额和总偿还额构成的表格

*17.14 (使用JComboBox和JList) 编写一个程序, 演示在列表框中选择条目。程序用组合框指定选择方式, 如图17-45所示。当选择条目后, 列表框下方的标签中就会显示选定项。

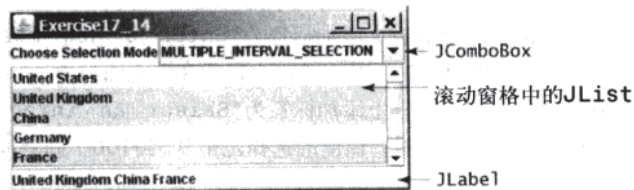


图17-45 可以在列表中选择单项选择、单区间选择或者多区间选择

17.11~17.13节

**17.15 (使用JScrollBar) 编写一个程序, 使用滚动条选择标签的前景色, 如图17-46所示。使用三个水平滚动条选择一种颜色的红色、绿色和蓝色的成分。给放置滚动条的面板上加一个带标题的边框。

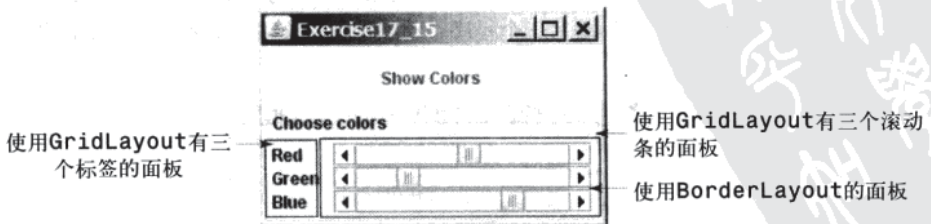


图17-46 标签中的前景色随着滚动条的调整而改变

**17.16 (使用JSlider) 使用滑块改写前一个练习题。

***17.17 (显示一个日历) 编写一个程序, 显示当前月的日历, 如图17-47所示。使用标签并且在标签上设置文本以显示日历。使用14.3节中的GregorianCalendar类获取月份、年份、该月第一天和该月的天数等信息。

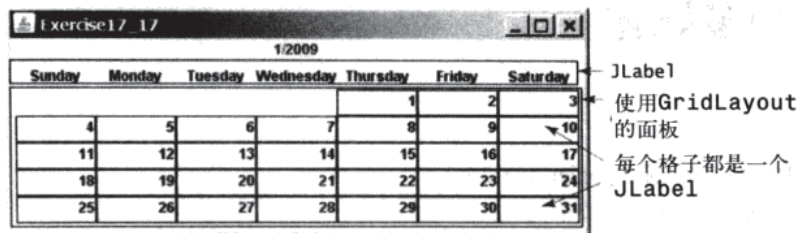


图17-47 程序显示当月的日历

*17.18 (修改程序清单17-12) 不使用程序清单17-12中的Histogram组件显示字母出现的次数, 而是采用条形图显示, 这样, 显示结果如图17-48所示。

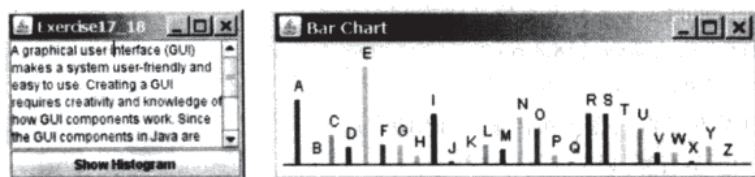


图17-48 使用条形图显示每个字母出现的次数

**17.19 (显示国旗和国旗的描述) 程序清单17-8给出的程序允许用户通过在组合框中选择国家来查看该国的国旗图像及对它的描述。对它的描述在程序中是用字符串表示的。改写这个程序从一个文件中读取它的文本描述。假设相关的描述存储在text目录下的description0.txt, ..., description8.txt文件中, 这九个文件依次对应于九个国家: 加拿大、中国、丹麦、法国、德国、印度、挪威、英国和美国。

17.20 (被幻灯片) 练习题16.13使用图像开发了一个幻灯片展示。使用文本文件改写练习题16.13来开发一个幻灯片展示。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, 重新显示第一张, 依此类推。使用一个文本域显示幻灯片。



第18章

Introduction to Java Programming, 8E

applet和多媒体

学习目标

- 将GUI应用程序转换为applet (18.2节)。
- 在网页中嵌入applet (18.3节)。
- 在Web浏览器及applet浏览器中运行applet (18.3.1 ~ 18.3.2节)。
- 理解applet安全沙盒模型 (18.4节)。
- 编写既可以作为应用程序运行也可以作为applet运行的Java程序 (18.5节)。
- 覆盖applet生命周期方法init、start、stop和destroy (18.6节)。
- 从HTML中向applet传递字符串参数 (18.7节)。
- 开发一个弹跳小球的动画 (18.8节)。
- 开发一个井字游戏的applet (18.9节)。
- 使用URL类来定位资源 (图像和音频) (18.10节)。
- 在Java程序中播放音频 (18.11节)。

18.1 引言

浏览网页时，经常会看到使用Java开发的图形用户界面和动画。这些程序称作Java applet。假设希望开发一个九宫格游戏的Java applet，如图18-1所示，该如何编写这个程序呢？

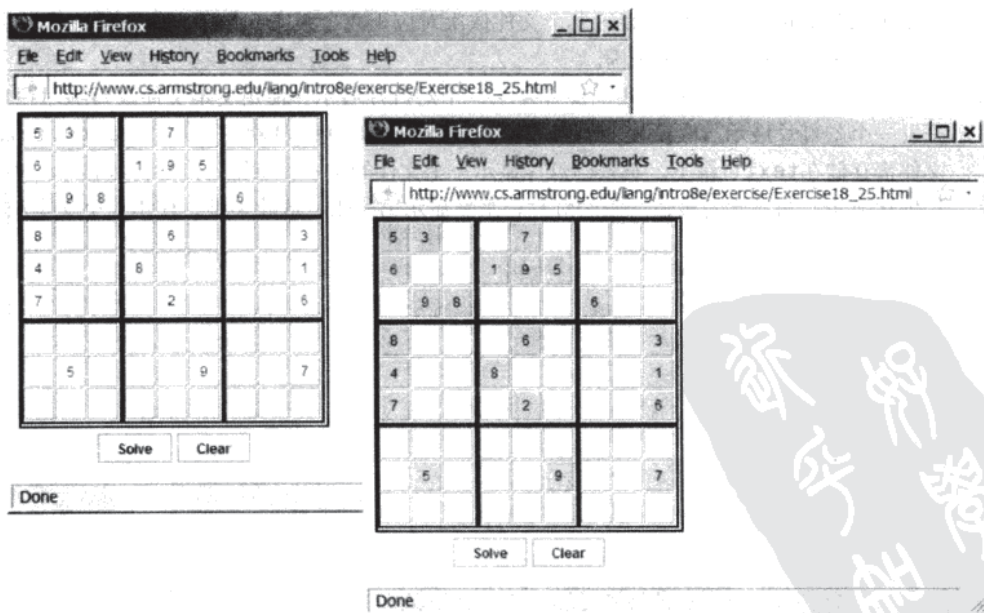


图18-1 在Web浏览器上显示九宫格游戏

在本章中，将学会如何编写Java applet，考察applet和Web浏览器之间的关系，探讨应用程序和applet之间的异同点，还将学习如何创建带图像和音频的多媒体Java应用程序和Java applet。

18.2 开发applet

到目前为止，所使用的只是Java应用程序。但是，所学到的关于如何编写应用程序的知识都可以用来编写applet。应用程序和applet共享许多通用的程序设计特性，尽管它们在某些方面有些小的不同。例如，每个应用程序都必须有main方法，该方法被Java解释器调用。另一方面，Java applet是不需要main方法的，它们可以在Web浏览器环境下运行。因为applet是从Web页面调用的，所以Java提供了能让applet在Web浏览器中运行的特殊特性。

Applet类提供了一个基本的框架结构，使得applet可以在Web浏览器中运行。每个Java应用程序都有一个在应用程序开始时执行的main方法，而applet没有main方法，它们依靠浏览器来运行。每个applet都是java.applet.Applet的子类。Applet类是一个AWT类，它不能和Swing组件一起工作。要在Java applet中使用Swing组件，需要通过扩展javax.swing.JApplet来创建一个Java applet，这里的javax.swing.JApplet是java.applet.Applet的一个子类。

开发的每一个Java GUI程序都可以通过将JFrame替换为JApplet同时删除main方法转换为一个applet。图18-2a显示了一个Java GUI应用程序，它可以转换为一个如图18-2b所示的Java applet。

```
import javax.swing.*;

public class DisplayLabel extends JFrame {
    public DisplayLabel() {
        add(new JLabel("Great!", JLabel.CENTER));
    }

    public static void main(String[] args) {
        JFrame frame = new DisplayLabel();
        frame.setTitle("DisplayLabel");
        frame.setSize(200, 100);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

a) GUI应用程序

```
import javax.swing.*;

public class DisplayLabel extends JApplet {
    public DisplayLabel() {
        add(new JLabel("Great!", JLabel.CENTER));
    }

    public static void main(String[] args) {
        JFrame frame = new DisplayLabel();
        frame.setTitle("DisplayLabel");
        frame.setSize(200, 100);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
    →
}
```

b) applet

图18-2 可以将一个GUI应用程序转换为一个applet

程序清单18-1给出这个applet的完整代码。

程序清单18-1 DisplayLabel.java

```
1 import javax.swing.*;
2
3 public class DisplayLabel extends JApplet {
4     public DisplayLabel() {
5         add(new JLabel("Great!", JLabel.CENTER));
6     }
7 }
```

就像JFrame一样，JApplet是一个可以包含其他GUI组件的容器（参见图14-1中的GUI类图）。

18.3 HTML文件和<applet>标记

为了从浏览器运行一个applet，需要创建带<applet>标记的HTML文件。

HTML是一种在网页上展示静态文档的标识语言。HTML使用标记指示Web浏览器如何绘制Web页面，HTML包含一个称为<applet>的标记，该标记将applet嵌入Web页面。

程序清单18-2中的HTML文件调用DisplayLabel.class：

程序清单18-2 DisplayLabel.html

```

<html>
<head>
  <title>Java Applet Demo</title>
</head>
<body>
  <applet
    code = "DisplayLabel.class"
    width = 250
    height = 50>
  </applet>
</body>
</html>

```

对Web浏览器来说,标记(tag)是一种指令。浏览器会解释标记,然后决定如何显示这个HTML文档中后续的内容,或者决定如何处理后续的内容。这些标记用尖括号括起来。标记中的第一个单词称为标记名(tag name),它描述标记的功能。标记还可以有一些附加的属性,有时候在等号后会有一值,它会进一步定义标记的行为。例如,在前面的HTML文件中,<applet>是标记名,而code、width和height都是属性。属性width和height表明applet的矩形视图区域的属性。

大多数标记都有一个开始标记(start tag)和对应的结束标记(end tag)。标记对开始标记和结束标记之间的内容会产生特定的效果。例如,<applet...>...</applet>会告诉浏览器显示一个applet。结束标记总是在开始标记的名字之前加一条斜杠。

HTML文档以标记<html>开始,声明这个文档是用HTML语言编写的。每个文档都有两部分:头(head)和体(body),分别用<head>标记和<body>标记来定义。头部分包含包括在<title>中的文档标题,以及绘制文档时浏览器可能使用的其他信息,而体部分包含文档的实际内容。头是可选的。

<applet>标记的完整语法如下:

```

<applet
  [codebase = applet_url]
  code = classfilename.class
  width = applet_viewing_width_in_pixels
  height = applet_viewing_height_in_pixels
  [archive = archivefile]
  [vspace = vertical_margin]
  [hspace = horizontal_margin]
  [align = applet_alignment]
  [alt = alternative_text]
>
<param name = param_name1 value = param_value1>
<param name = param_name2 value = param_value2>
...
<param name = param_name3 value = param_value3>
</applet>

```

属性code、width和height是必需的,其余属性是可选的。<param>标记将在18.7节中介绍,其他属性解释如下:

1) codebase指定类该加载到哪里。如果不使用这个属性,Web浏览器会从HTML页面所在的目录加载applet。如果applet与HTML网页不在同一目录下,则必须为浏览器指定装载applet的applet_url。利用这一属性,可以从Internet的任何地方加载类。必要时,applet动态地加载所使用的类。

2) archive指示浏览器加载一个存档文件,该文件包含运行applet所需要的所有类文件。存档允许Web浏览器从一个压缩文件一次性加载所有的类,这样可以减少加载时间同时提高性能。要想建立存档文件,请参见补充材料Ⅲ.Q。

3) vspace和hspace指定applet周围的垂直方向和水平方向空白边界的大小,它们的单位是像素。

4) align指定applet在浏览器中是如何对齐的。使用下面的九个值之一: left、right、top、texttop、middle、absmiddle、baseline、bottom或absbottom。

5) alt指定浏览器不能运行Java时显示的文本。

18.3.1 从Web浏览器查看applet

要从一个Web浏览器中显示applet，打开applet的HTML文件（例如，DisplayLabel.html）即可。它的输出结果如图18-3a所示。

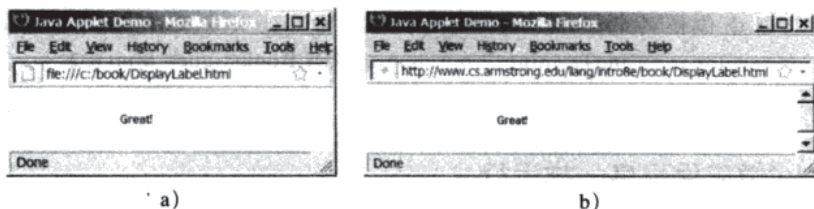


图18-3 a) 中的DisplayLabel程序是从本地主机加载的，而b) 中的DisplayLabel是从Web服务器加载的

要想在网上可访问applet，需要将DisplayLabel.class和DisplayLabel.html都存在一个Web服务器上，如图18-4所示。可以从一个恰当的URL中浏览一个applet。例如，可以向Web服务器www.cs.armstrong.edu/上传这两个文件。如图18-3b所示，可以从www.cs.armstrong.edu/liang/intro8e/book/DisplayLabel.html访问这个applet。

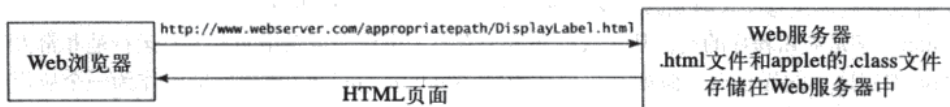


图18-4 Web浏览器请求来自Web服务器的一个HTML网页

18.3.2 使用applet查看器工具查看applet

可以使用applet查看器工具测试applet。在DOS命令行中，从目录c:\book中使用appletviewer命令就可以调用这个工具，如图18-5a所示。它的输出结果如图18-5b所示。

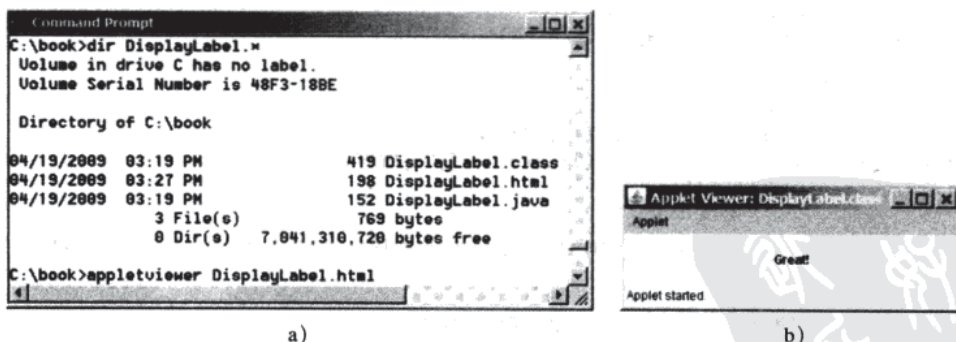


图18-5 使用appletviewer命令在applet查看器工具中运行一个Java applet

applet查看器工具的功能就相当于一个Web浏览器。在没有启动Web浏览器的开发过程中，这是测试applet的一个很简便的方法。

18.4 applet安全限制

Java使用所谓的“沙盒安全模型”来执行applet，以防止毁灭性的程序危害到运行浏览器的系统。不允许applet使用“沙盒”之外的资源。沙盒会特别地限制下面的动作：

- 不允许applet对计算机的文件系统进行读取或写入。否则，它们会损坏这些文件，还会传播病毒。
- 不允许applet运行浏览器所在的计算机上的任何程序。否则，它们可能会调用有破坏力的本地程序，并且破坏用户计算机的本地系统。
- 除了在服务器上存储applet之外，不允许applet同用户的计算机和其他任何计算机之间建立任何连接。这个限制是为了防止在用户不知道的情况下，applet将用户的计算机连接到另一台计算机。

注意 可以创建信任的applet (signed applet) 规避这些安全限制。可以参见网页 java.sun.com/developer/onlineTraining/Programming/JDCBook/signed.html 获取如何创建信任applet的详细指示。

18.5 让applet像应用程序一样运行

尽管JFrame类和JApplet类有一些差别，但是它们有很多共同之处。由于它们都是Container类的子类，所以，它们所有的用户界面组件、布局管理器以及事件处理特征都是一样的。但是，应用程序是Java解释器从静态main方法调用的，而applet是由Web浏览器运行的。Web浏览器使用applet的无参数构造方法创建applet的一个实例，然后控制和执行这个applet。

一般来说，applet都能转换为应用程序而不损失任何功能。只要不违反applet上的强制性安全限制，应用程序也可以转换为applet。可以在applet中实现一个main方法，这样，applet就能像应用程序一样运行。这个特点既有理论意义又有实践意义。理论上讲，它模糊了applet和应用程序之间的区别。可以编写一个既是applet又是应用程序的类。从实践的角度来看，一个程序可以用两种方式运行是非常方便的。

该如何编写这样的程序呢？假如有一个名为MyApplet的applet，要使它能够作为一个应用程序来运行，就只需要在applet中添加并实现一个main方法，如下所示：

```
public static void main(String[] args) {
    // Create a frame
    JFrame frame = new JFrame("Applet is in the frame");

    // Create an instance of the applet
    MyApplet applet = new MyApplet();

    // Add the applet to the frame
    frame.add(applet, BorderLayout.CENTER);

    // Display the frame
    frame.setSize(300, 300);
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
}
```

可以通过在程序清单18-3中添加一个main方法来修改程序清单18-1中的DisplayLabel类，以使它可以独立运行。

程序清单18-3 有main方法的新DisplayLabel.java

```
1 import javax.swing.*;
2
3 public class DisplayLabel extends JApplet {
4     public DisplayLabel() {
5         add(new JLabel("Great!", JLabel.CENTER));
6     }
7
8     public static void main(String[] args) {
9         // Create a frame
10        JFrame frame = new JFrame("Applet is in the frame");
11
12        // Create an instance of the applet
13        DisplayLabel applet = new DisplayLabel();
```



```

14
15 // Add the applet to the frame
16 frame.add(applet);
17
18 // Display the frame
19 frame.setSize(300, 100);
20 frame.setLocationRelativeTo(null); // Center the frame
21 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22 frame.setVisible(true);
23 }
24 }

```

当从Web浏览器运行applet时，浏览器会创建一个applet的实例，然后显示它。当独立运行applet时，就会调用main方法来创建一个存放applet的框架（第10行）。创建applet（第13行）并且将它添加到这个框架中（第16行）。这个框架在第22行显示。

18.6 applet生命周期方法

实际上，applet是从applet容器（applet container）运行的，这个容器是Web浏览器的一个插件。类Applet包括init()、start()、stop()和destroy()方法，这些方法都称为生命周期方法（life-cycle method）。这些方法都被applet容器调用以控制一个applet的执行。它们在Applet类中用空体来实现。所以，默认情况下，它们什么都不做。可以在Applet的子类中覆盖它们以完成所需的操作。图18-6显示applet容器是如何调用这些方法的。

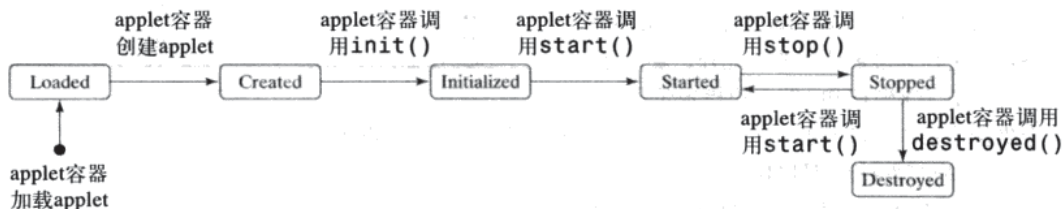


图18-6 applet容器使用init、start、stop和destroy方法来控制applet

18.6.1 init方法

创建applet之后就会调用init方法。如果Applet的子类有一个初始化过程要完成，那么该子类就应覆盖init方法。通常，这个方法实现的功能包括从HTML网页的<applet>标记中获取字符串参数值。

18.6.2 start方法

在init方法之后就会调用start方法。当用户浏览过其他页面之后返回到包含这个applet的Web页面时，该方法也会被调用。

每当访问包含applet的Web网页时，如果还有任何需要完成的操作，Applet的子类就会覆盖这个方法。例如，一个带动画的applet可以启动定时器来重新开始动画。

18.6.3 stop方法

stop方法与start方法恰好相反，start方法是在用户返回包含applet的网页时调用的，而stop方法是在用户离开这个网页时调用的。

每当包含applet的Web页面不再可见时，如果还有其他需要执行的操作，Applet的子类就会覆盖这个方法。例如，一个带动画的applet可能会停止定时器来暂停动画。

18.6.4 destroy方法

当浏览器正常退出时就会调用destroy方法，以通知applet不再需要它并且应该释放它所占有的资

源。stop方法总是在destroy方法之前调用。

如果Applet子类在销毁之前还有需要完成的操作，这个Applet的子类就会覆盖destroy方法。通常情况下是不需要覆盖这个方法的，除非希望释放创建applet所占有的特定资源。

18.7 给applet传递字符串

在9.5节中，已经学习了如何从命令行向Java应用程序传递字符串。字符串是作为字符串数组传递给main方法的。当应用程序开始时，main方法就可以使用这些字符串。但是，在applet中没有main方法，所以不能通过Java解释器从命令行运行。

那么，applet是如何接收参数的呢？在本节中，将学习如何向Java applet传递字符串。要将参数传递给applet，必须在HTML文件中声明它，而且必须在applet初始化时由applet读取。参数是使用<param>标记声明的。<param>标记必须嵌入到<applet>标记中且没有对应的结束标记。它的语法如下所示：

```
<param name = parametername value = stringvalue />
```

这个标记表明一个参数和它对应的字符串值。

注意 在HTML代码中，参数名与参数值之间没有逗号分隔。HTML的参数名是不区分大小写的。

假设要编写一个applet来显示一条消息。这条消息是作为参数传递的。此外，要让这条消息显示在有x坐标和y坐标的指定位置，这两个坐标值也作为参数进行传递。表18-1中列出了参数以及它们的值。

程序清单18-4给出HTML源文件。

程序清单18-4 DisplayMessage.html

```
<html>
<head>
  <title>Passing Strings to Java Applets</title>
</head>
<body>
  <p>This applet gets a message from the HTML
    page and displays it.</p>
  <applet
    code = "DisplayMessage.class"
    width = 200
    height = 50
    alt = "You must have a Java 2-enabled browser to view the applet"
  >
    <param name = MESSAGE value = "Welcome to Java" />
    <param name = X value = 20 />
    <param name = Y value = 30 />
  </applet>
</body>
</html>
```

为了从applet中读取参数，使用在Applet类中定义的下述方法：

```
public String getParameter(String parametername);
```

它会返回指定参数的值。

程序清单18-5给出这个applet。这个applet的运行示例如图18-7所示。

表18-1 DisplayMessage applet的参数名和参数值

参数名	参数值
MESSAGE	"Welcome to Java"
X	20
Y	30

PDF

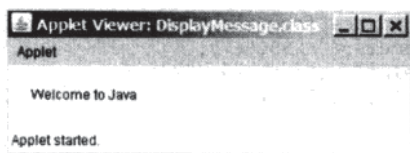


图18-7 这个applet显示从HTML页面传递来的消息Welcome to Java

程序清单18-5 DisplayMessage.java

```

1 import javax.swing.*;
2
3 public class DisplayMessage extends JApplet {
4     /** Initialize the applet */
5     public void init() {
6         // Get parameter values from the HTML file
7         String message = getParameter("MESSAGE");
8         int x = Integer.parseInt(getParameter("X"));
9         int y = Integer.parseInt(getParameter("Y"));
10
11         // Create a message panel
12         MessagePanel messagePanel = new MessagePanel(message);
13         messagePanel.setXCoordinate(x);
14         messagePanel.setYCoordinate(y);
15
16         // Add the message panel to the applet
17         add(messagePanel);
18     }
19 }

```

这个程序在init方法中获取来自HTML文件的参数值。这些值是使用getParameter方法获取的字符串（第7~9行）。因为x和y都是int型的，所以，程序使用Integer.parseInt(string)方法将一个数值字符串转化为一个int值。

如果将HTML文件中的Welcome to Java改为Welcome to HTML，然后在Web浏览器上重新加载这个HTML文件，就会看到显示Welcome to HTML。同样，也可以改变x和y的值，在所需的位置显示这条信息。

警告 Applet类的getParameter方法只能在创建一个applet的实例之后调用。因此，这个方法不能在applet类的构造方法中调用。应该从init方法中调用它。

可以添加一个main方法，让这个applet能够独立运行。当这个applet作为一个applet运行时会从HTML文件中读取参数，而当它独立运行时会从命令行读取参数。除了有一个新的main方法以及一个表明这个程序是作为一个applet还是一个应用程序来运行的名为isStandalone的变量值之外，程序清单18-6所示的程序和DisplayMessage是一样的。

程序清单18-6 DisplayMessageApp.java

```

1 import javax.swing.*;
2 import java.awt.Font;
3 import java.awt.BorderLayout;
4
5 public class DisplayMessageApp extends JApplet {
6     private String message = "A default message"; // Message to display
7     private int x = 20; // Default x-coordinate
8     private int y = 20; // Default y-coordinate
9
10    /** Determine whether it is an application */
11    private boolean isStandalone = false;
12
13    /** Initialize the applet */
14    public void init() {
15        if (!isStandalone) {

```

```

16 // Get parameter values from the HTML file
17 message = getParameter("MESSAGE");
18 x = Integer.parseInt(getParameter("X"));
19 y = Integer.parseInt(getParameter("Y"));
20 }
21
22 // Create a message panel
23 MessagePanel messagePanel = new MessagePanel(message);
24 messagePanel.setFont(new Font("SansSerif", Font.BOLD, 20));
25 messagePanel.setXCoordinate(x);
26 messagePanel.setYCoordinate(y);
27
28 // Add the message panel to the applet
29 add(messagePanel);
30 }
31
32 /** Main method to display a message
33     @param args[0] x-coordinate
34     @param args[1] y-coordinate
35     @param args[2] message
36 */
37 public static void main(String[] args) {
38     // Create a frame
39     JFrame frame = new JFrame("DisplayMessageApp");
40
41     // Create an instance of the applet
42     DisplayMessageApp applet = new DisplayMessageApp();
43
44     // It runs as an application
45     applet.isStandalone = true;
46
47     // Get parameters from the command line
48     applet.getCommandLineParameters(args);
49
50     // Add the applet instance to the frame
51     frame.add(applet, BorderLayout.CENTER);
52
53     // Invoke applet's init method
54     applet.init();
55     applet.start();
56
57     // Display the frame
58     frame.setSize(300, 300);
59     frame.setLocationRelativeTo(null); // Center the frame
60     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
61     frame.setVisible(true);
62 }
63
64 /** Get command-line parameters */
65 private void getCommandLineParameters(String[] args) {
66     // Check usage and get x, y and message
67     if (args.length != 3) {
68         System.out.println(
69             "Usage: java DisplayMessageApp x y message");
70         System.exit(0);
71     }
72     else {
73         x = Integer.parseInt(args[0]);
74         y = Integer.parseInt(args[1]);
75         message = args[2];
76     }
77 }
78 }

```

将这个程序作为一个applet运行时，忽略main方法。若将这个程序作为应用程序运行，调用main方

法。这个程序作为应用程序和作为applet的运行示例如图18-8所示。

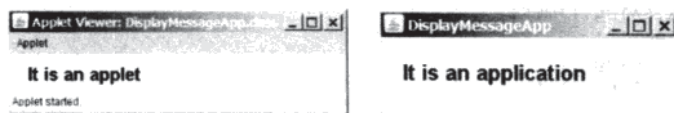


图18-8 DisplayMessageApp类既可以作为一个应用程序运行，也可以作为一个applet运行

main方法创建了一个JFrame的对象frame，还创建了一个JApplet的对象applet，然后，将这个applet放到框架frame中，接着调用它的init方法，应用程序就会像applet一样运行。

main方法将isStandalone设置为true（第45行），所以不要在调用init方法时尝试重新获取HTML参数。

当组件添加到applet之后，就会调用setVisible(true)方法（第61行），然后将applet添加到框架中以确保这个组件是可见的。否则，当框架开始时不显示组件。

重要的教学注意 从现在开始，所有的GUI示例都将创建为带main方法的applet。这样，既可以将程序当作applet运行，也可以当作应用程序运行。为了简洁，文中不列出main方法。

18.8 实例学习：弹跳的小球

本节给出一个applet，显示一个小球在面板上弹跳。使用两个按钮暂停和重新开始小球的运动，使用一个滚动条控制小球弹跳的速度，如图18-9所示。

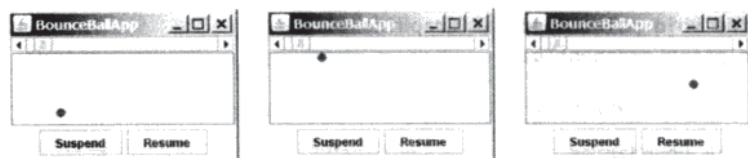


图18-9 用Suspend按钮、Resume按钮以及滚动条控制小球的运动

下面是完成这个例子的主要步骤：

- 1) 创建一个JPanel的子类，将它命名为Ball，用来显示小球的弹跳，如程序清单18-7所示。
- 2) 创建一个JPanel的子类，将它命名为BallControl，包含一个小球、一个滚动条、两个控制按钮Suspend和Resume，如程序清单18-8所示。
- 3) 创建一个名为BounceBallApp的applet，它包含BallControl的一个实例，使得applet能够独立运行，如程序清单18-9所示。

这些类之间的关系如图18-10所示。

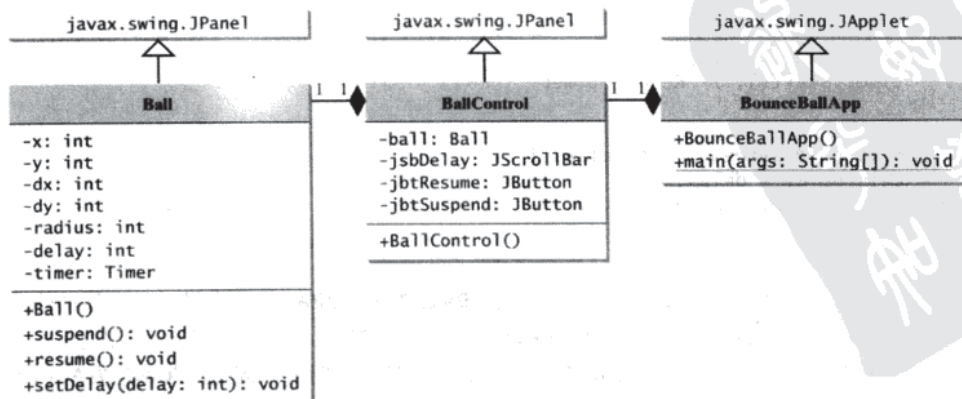


图18-10 BounceBallApp包含BallControl，而BallControl包含Ball

程序清单18-7 Ball.java

```

1 import javax.swing.Timer;
2 import java.awt.*;
3 import javax.swing.*;
4 import java.awt.event.*;
5
6 public class Ball extends JPanel {
7     private int delay = 10;
8
9     // Create a timer with delay 1000 ms
10    private Timer timer = new Timer(delay, new TimerListener());
11
12    private int x = 0; private int y = 0; // Current ball position
13    private int radius = 5; // Ball radius
14    private int dx = 2; // Increment on ball's x-coordinate
15    private int dy = 2; // Increment on ball's y-coordinate
16
17    public Ball() {
18        timer.start();
19    }
20
21    private class TimerListener implements ActionListener {
22        /** Handle the action event */
23        public void actionPerformed(ActionEvent e) {
24            repaint();
25        }
26    }
27
28    protected void paintComponent(Graphics g) {
29        super.paintComponent(g);
30
31        g.setColor(Color.red);
32
33        // Check boundaries
34        if (x < radius) dx = Math.abs(dx);
35        if (x > getWidth() - radius) dx = -Math.abs(dx);
36        if (y < radius) dy = Math.abs(dy);
37        if (y > getHeight() - radius) dy = -Math.abs(dy);
38
39        // Adjust ball position
40        x += dx;
41        y += dy;
42        g.fillOval(x - radius, y - radius, radius * 2, radius * 2);
43    }
44
45    public void suspend() {
46        timer.stop(); // Suspend timer
47    }
48
49    public void resume() {
50        timer.start(); // Resume timer
51    }
52
53    public void setDelay(int delay) {
54        this.delay = delay;
55        timer.setDelay(delay);
56    }
57 }

```

在16.12节中介绍了如何使用Timer来控制动画。Ball类扩展JPanel类来显示一个移动的球。定时器监听器实现ActionListener来监听ActionEvent事件（第21行）。第10行为Ball创建一个Timer对象。当构造一个Ball的实例时，就会启动第18行的定时器。这个定时器以固定的频率触发ActionEvent事件。监听器在第24行做出响应重画小球，以得到小球运动的动画。球的中心坐标在(x, y)处，它在

下一次显示时就会变为 (x+dx, y+dy) (第40~41行)。suspend方法和resume方法 (第45~51行) 可以用来停止和启动定时器。setDelay(int)方法 (第53~56行) 设置一个新延时。

程序清单18-8 BallControl.java

```

1 import javax.swing.*;
2 import java.awt.event.*;
3 import java.awt.*;
4
5 public class BallControl extends JPanel {
6     private Ball ball = new Ball();
7     private JButton jbtSuspend = new JButton("Suspend");
8     private JButton jbtResume = new JButton("Resume");
9     private JScrollBar jsbDelay = new JScrollBar();
10
11     public BallControl() {
12         // Group buttons in a panel
13         JPanel panel = new JPanel();
14         panel.add(jbtSuspend);
15         panel.add(jbtResume);
16
17         // Add ball and buttons to the panel
18         ball.setBorder(new javax.swing.border.LineBorder(Color.red));
19         jsbDelay.setOrientation(JScrollBar.HORIZONTAL);
20         ball.setDelay(jsbDelay.getMaximum());
21         setLayout(new BorderLayout());
22         add(jsbDelay, BorderLayout.NORTH);
23         add(ball, BorderLayout.CENTER);
24         add(panel, BorderLayout.SOUTH);
25
26         // Register listeners
27         jbtSuspend.addActionListener(new ActionListener() {
28             public void actionPerformed(ActionEvent e) {
29                 ball.suspend();
30             }
31         });
32         jbtResume.addActionListener(new ActionListener() {
33             public void actionPerformed(ActionEvent e) {
34                 ball.resume();
35             }
36         });
37         jsbDelay.addAdjustmentListener(new AdjustmentListener() {
38             public void adjustmentValueChanged(AdjustmentEvent e) {
39                 ball.setDelay(jsbDelay.getMaximum() - e.getValue());
40             }
41         });
42     }
43 }

```

BallControl类扩展JPanel类, 显示一个球、一个滚动条和两个控制按钮。当点击Suspend按钮时, 就会调用球的suspend()方法使球暂停运动 (第29行)。当点击Resume按钮时, 就会调用球的resume()方法使球重新开始运动 (第34行)。可以使用滚动条改变小球弹跳的速度。

程序清单18-9 BounceBallApp.java

```

1 import java.awt.*;
2 import javax.swing.*;
3
4 public class BounceBallApp extends JApplet {
5     public BounceBallApp() {
6         add(new BallControl());
7     }
8 }

```

BounceBallApp类只在applet中放置了一个BallControl的实例。该applet还提供了main方法 (为

了简明，没有在清单中列出来)，以便可以独立运行它。

18.9 实例学习：井字游戏

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、事件驱动程序设计以及applet。现在到了应用所学知识开发一些复杂项目的时候了。本节将开发一个玩流行的井字游戏(TicTacToe)的Java applet。

在井字游戏中，两个玩家在 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标满了标记还没有产生优胜者，就会出现平局（没有胜者）。图18-11显示了这个例子的典型运行示例。

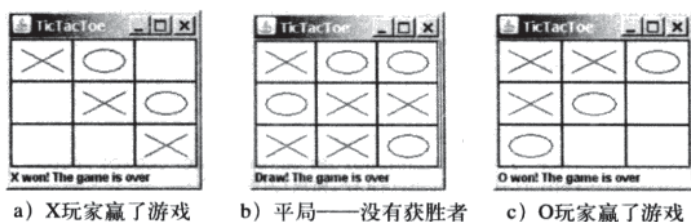


图18-11 两个玩家玩井字游戏

到现在为止，我们见过的所有例子的动作都很简单，都很容易用类来模拟。但是井字游戏的动作有些复杂。为了创建模拟游戏的行为的类，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。要在单元格上作标记，玩家应该将鼠标指针放在这个单元格上，然后点击它。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点击事件和显示标记的GUI对象。这样的对象可以是按钮，也可以是面板。在面板上画图比在按钮上画图具有更高的灵活性，因为在面板上可以画任意大小的标记(X或O)，但是，按钮上的标记只能显示为文本标签。因此，应该选用面板建模单元格。怎样才能知道单元格的状态（空、X或O）呢？可以使用单元格类Cell中名为token的char型属性来解决这个问题。Cell类负责点击空单元格时画出标记。因此，需要编写代码来监听MouseEvent事件，以及绘制标记X和O形状的代码。Cell类可以定义为如图18-12所示。

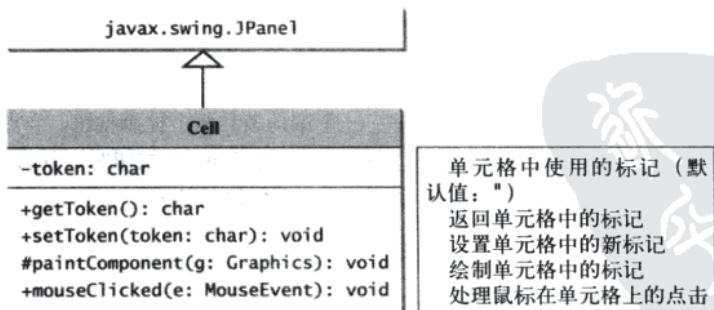


图18-12 Cell类在单元格上绘制标记

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，是否产生了优胜者？如果有优胜者，那么谁是优胜者？可以创建

一个名为isWon(char token)的方法来判断指定标记是否是优胜者；也可以创建一个名为isFull()的方法来判断是否所有的单元格都被占满。

很显然，前面的分析出现了两个类。一个类是处理单个单元格上操作的Cell类；另一个类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18-13所示。

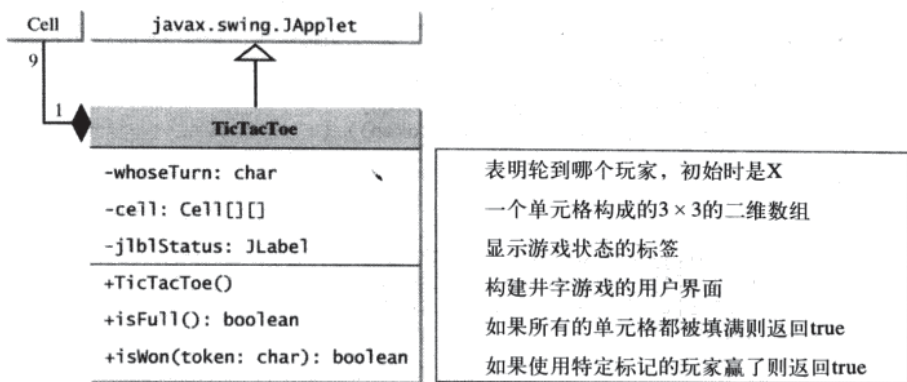


图18-13 TicTacToe类包含9个单元格

因为Cell类只是为了支持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18-10中给出。

程序清单18-10 TicTacToe.java

```

1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4 import javax.swing.border.LineBorder;
5
6 public class TicTacToe extends JApplet {
7     // Indicate which player has a turn; initially it is the X player
8     private char whoseTurn = 'X';
9
10    // Create and initialize cells
11    private Cell[][] cells = new Cell[3][3];
12
13    // Create and initialize a status label
14    private JLabel jlblStatus = new JLabel("X's turn to play");
15
16    /** Initialize UI */
17    public TicTacToe() {
18        // Panel p to hold cells
19        JPanel p = new JPanel(new GridLayout(3, 3, 0, 0));
20        for (int i = 0; i < 3; i++)
21            for (int j = 0; j < 3; j++)
22                p.add(cells[i][j] = new Cell());
23
24        // Set line borders on the cells panel and the status label
25        p.setBorder(new LineBorder(Color.red, 1));
26        jlblStatus.setBorder(new LineBorder(Color.yellow, 1));
27
28        // Place the panel and the label to the applet
29        add(p, BorderLayout.CENTER);
30        add(jlblStatus, BorderLayout.SOUTH);
31    }
32
33    /** Determine whether the cells are all occupied */
34    public boolean isFull() {
35        for (int i = 0; i < 3; i++)
36            for (int j = 0; j < 3; j++)
  
```

```

37         if (cells[i][j].getToken() == ' ')
38             return false;
39
40     return true;
41 }
42
43 /** Determine whether the player with the specified token wins */
44 public boolean isWon(char token) {
45     for (int i = 0; i < 3; i++)
46         if ((cells[i][0].getToken() == token)
47             && (cells[i][1].getToken() == token)
48             && (cells[i][2].getToken() == token)) {
49             return true;
50         }
51
52     for (int j = 0; j < 3; j++)
53         if ((cells[0][j].getToken() == token)
54             && (cells[1][j].getToken() == token)
55             && (cells[2][j].getToken() == token)) {
56             return true;
57         }
58
59     if ((cells[0][0].getToken() == token)
60         && (cells[1][1].getToken() == token)
61         && (cells[2][2].getToken() == token)) {
62         return true;
63     }
64
65     if ((cells[0][2].getToken() == token)
66         && (cells[1][1].getToken() == token)
67         && (cells[2][0].getToken() == token)) {
68         return true;
69     }
70
71     return false;
72 }
73
74 // An inner class for a cell
75 public class Cell extends JPanel {
76     // Token used for this cell
77     private char token = ' ';
78
79     public Cell() {
80         setBorder(new LineBorder(Color.black, 1)); // Set cell's border
81         addMouseListener(new MyMouseListener()); // Register listener
82     }
83
84     /** Return token */
85     public char getToken() {
86         return token;
87     }
88
89     /** Set a new token */
90     public void setToken(char c) {
91         token = c;
92         repaint();
93     }
94
95     /** Paint the cell */
96     protected void paintComponent(Graphics g) {
97         super.paintComponent(g);
98
99         if (token == 'X') {
100             g.drawLine(10, 10, getWidth() - 10, getHeight() - 10);
101             g.drawLine(getWidth() - 10, 10, 10, getHeight() - 10);

```

```

102     }
103     else if (token == 'O') {
104         g.drawOval(10, 10, getWidth() - 20, getHeight() - 20);
105     }
106 }
107
108 private class MyMouseListener extends MouseAdapter {
109     /** Handle mouse click on a cell */
110     public void mouseClicked(MouseEvent e) {
111         // If cell is empty and game is not over
112         if (token == ' ' && whoseTurn != ' ') {
113             setToken(whoseTurn); // Set token in the cell
114
115             // Check game status
116             if (isWon(whoseTurn)) {
117                 jlblStatus.setText(whoseTurn + " won! The game is over");
118                 whoseTurn = ' '; // Game is over
119             }
120             else if (isFull()) {
121                 jlblStatus.setText("Draw! The game is over");
122                 whoseTurn = ' '; // Game is over
123             }
124             else {
125                 // Change the turn
126                 whoseTurn = (whoseTurn == 'X') ? 'O' : 'X';
127                 // Display whose turn
128                 jlblStatus.setText(whoseTurn + "'s turn");
129             }
130         }
131     }
132 }
133 }
134 }

```

TicTacToe类使用放置在GridLayout的面板上的9个单元格来初始化用户界面（第19~22行）。名为jlblStatus的标签用来显示游戏的状态（第14行）。变量whoseTurn（第8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34~41行）和isWon方法（第44~72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，可以从类Cell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类声明为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。编程练习题18.6要求不使用内部类来改写这个程序。

MouseEvent的监听器是为单元格而注册的（第81行）。如果游戏没有结束时点击空单元格，那么在单元格中会设置一个标记（第113行）。如果游戏结束，whoseTurn设置为' '（空）（第118行和第122行）。否则，whoseTurn就会切换到新一轮（第126行）。

提示 采用逐步扩充的方法开发和测试这一类Java项目。前面的程序可以分为五个步骤：

- 1) 部署用户界面，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点击以显示固定标记X。
- 3) 协调两个玩家，以便交替地显示标记X和O。
- 4) 判断是否有玩家获胜，或者所有的单元格都被占满且仍无获胜者。
- 5) 一旦一个玩家移动一步，就在标签上显示一条消息。

18.10 使用URL类定位资源

前面已经使用ImageIcon类从一个图像文件中创建一个图标，并使用setIcon方法或构造方法将图

标放在类似按钮或标签的GUI组件中。例如，下面的语句创建一个ImageIcon，并且将它设置在JLabel对象jlbl1上：

```
ImageIcon imageIcon = new ImageIcon("c:\\book\\image\\us.gif");
jlbl1.setIcon(imageIcon);
```

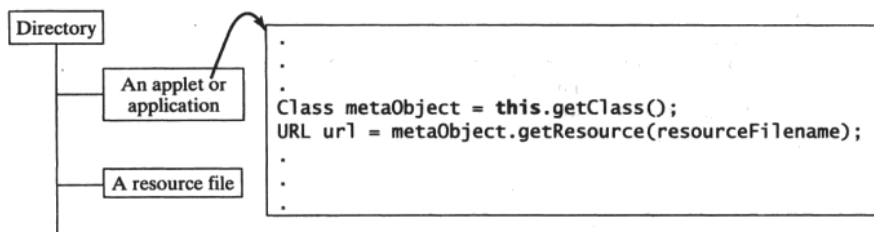
该方法会带来一个问题。文件的位置是固定的，因为它使用的是Windows平台的绝对文件路径。结果就造成它不能在其他平台上运行而且也不能作为applet运行。假设image/us.gif在类目录下。可以使用如下所示的相对路径来规避这个问题：

```
ImageIcon imageIcon = new ImageIcon("image/us.gif");
```

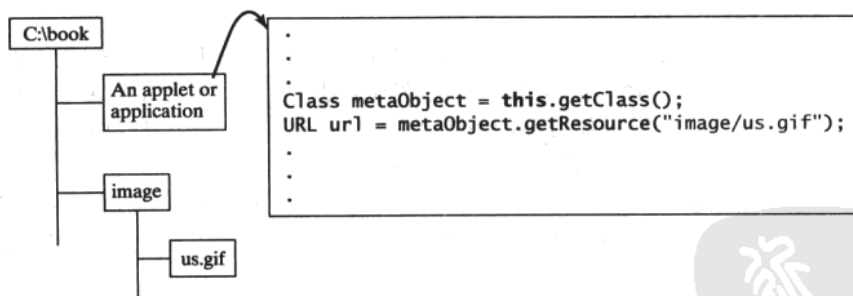
这样，Java应用程序就能在所有平台上正常工作，但是，这种方法对Java applet不起作用，因为applet不能加载本地文件。为了让它在应用程序和applet中都能工作，需要定位这个文件的URL。

java.net.URL类能够用来确定互联网上的文件（图像、音频和文本等）。笼统地说，URL（Uniform Resource Locator，统一资源定位符）是一个指向本地机器或远程主机上某个“资源”的指示器。这里的资源可以只是一个文件或一个目录。

如果文件放在类目录下，那么该文件的URL也可以以独立于文件位置的方式从一个类来访问。回顾一下，类目录就是存放类的位置。为了获取资源文件的URL，在applet或应用程序中使用下面的语句：



getClass()方法返回java.lang.Class类的一个实例。当每个类被载入内存时，Java虚拟机就会为该类创建一个实例。这个实例也称为元对象（meta object），它包括了关于类文件的信息，例如，类名、构造方法和方法。可以调用元对象上的getResource(filename)方法以获取类目录中一个文件的URL。例如，如果类文件在c:\book目录下，那么下面的语句就可以获取c:\book\image\us.gif的一个URL。



现在，可以使用下面的语句创建一个ImageIcon：

```
ImageIcon imageIcon = new ImageIcon(url);
```

程序清单18-11给出显示类目录中来自image/us.gif的图像的代码。文件image/us.gif在类目录下，而它的URL是使用getResource方法获取的（第5行）。一个有图像图标的标签在第6行创建。这个图像图标是从URL获取的。

程序清单18-11 DisplayImageWithURL.java

```

1 import javax.swing.*;
2
3 public class DisplayImageWithURL extends JApplet {
4     public DisplayImageWithURL() {
5         java.net.URL url = this.getClass().getResource("image/us.gif");

```



```

6    add(new JLabel(new ImageIcon(url)));
7  }
8  }

```

如果使用下面的代码替换第5~6行的代码:

```
add(new JLabel(new ImageIcon("image/us.gif")));
```

仍然可以独立地运行这个程序,但是不能在浏览器上运行它。

18.11 在任意Java程序中播放音频

音频文件有多种格式。Java程序能够播放WAV、AIFF、MIDI、AU和RMF格式的声音文件。

要在Java(应用程序或applet)中播放音频文件,应该先为声音文件创建一个音频剪辑对象(audio clip object)。一旦创建了音频剪辑,不需要重新加载文件就能重复播放声音。为了创建一个音频剪辑,使用java.applet.Applet类中的静态方法newAudioClip():

```
AudioClip audioClip = Applet.newAudioClip(url);
```

声音原本是只能在Java applet中播放的。因为这个原因,AudioClip接口位于java.applet包中。从JDK 1.2开始,音频就能在任何一个程序中播放。

例如,下面的语句为类目录下的声音文件beep.au创建一个音频剪辑AudioClip对象:

```

Class metaObject = this.getClass();
URL url = metaObject.getResource("beep.au");
AudioClip audioClip = Applet.newAudioClip(url);

```

使用java.applet.AudioClip中的play()、loop()和stop()方法可以操控音频剪辑中的声音,如图18-14所示。

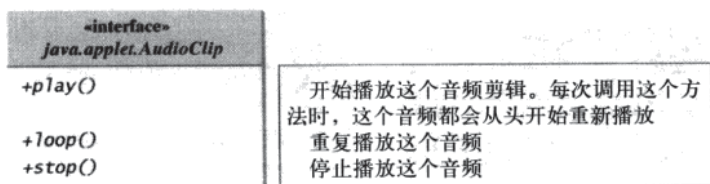


图18-14 AudioClip接口提供播放声音的方法

程序清单18-12给出显示丹麦国旗并重复播放丹麦国歌的代码。图像文件image/denmark.gif和音频文件audio/denmark.mid都存放在类目录下。第12行获取音频文件的URL。第13行为该文件创建一个音频剪辑对象。第14行重复播放这个音频文件。

程序清单18-12 DisplayImagePlayAudio.java

```

1  import javax.swing.*;
2  import java.net.URL;
3  import java.applet.*;
4
5  public class DisplayImagePlayAudio extends JApplet {
6      private AudioClip audioClip;
7
8      public DisplayImagePlayAudio() {
9          URL urlForImage = getClass().getResource("image/denmark.gif");
10         add(new JLabel(new ImageIcon(urlForImage)));
11
12         URL urlForAudio = getClass().getResource("audio/denmark.mid");
13         audioClip = Applet.newAudioClip(urlForAudio);
14         audioClip.loop();
15     }
16
17     public void start() {
18         if (audioClip != null) audioClip.loop();

```

```

19 }
20
21 public void stop() {
22     if (audioClip != null) audioClip.stop();
23 }
24 }

```

当不显示applet时，stop方法（第21~23行）就会终止这个音频，当重新显示applet时，start方法（第17~19行）会重新启动这个音频。请尝试从浏览器运行该applet，并且观察一下没有stop和start方法的效果。

从主方法并在Web浏览器独立运行这个程序来测试它。回顾以前所讲的，为了简洁，所有applet中的main方法都没有在程序清单中打印出来。

18.12 实例学习：多媒体动画

本实例学习给出一个带图像和音频的多媒体动画。七个名为flag0.gif, flag1.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。它们都存储在类路径的image目录下。音频包括了这七个国家的国歌anthem0.mid, anthem1.mid, ..., anthem6.mid。它们都存储在类路径的audio目录下。

程序从第一个国家开始表示这些国家。对每个国家，程序显示它的国旗并播放它的国歌。当一个国家的音频结束时，就会出现下一个国家，依此类推。当显示完最后一个国家之后，程序会重新显示所有的国家。可以通过点击Suspend按钮暂停动画，点击Resume按钮重新开始播放动画，如图18-15所示。也可以直接从组合框中选择一个国家。

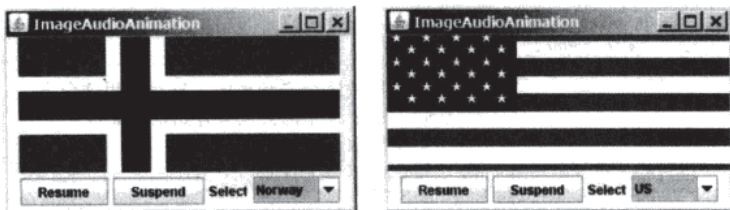


图18-15 applet显示一系列的图像并且播放音频

程序清单18-13给出这个程序。创建一个定时器来控制动画（第15行）。每次表示国旗的定时器延迟就是它的国歌的播放时间。可以使用RealPlayer或者Windows Media找到音频文件的播放时间。延迟时间存储在一个名为delays的数组中（第13~14行）。第一个音频文件（丹麦国歌）的延迟时间是48秒。

程序清单18-13 ImageAudioAnimation.java

```

1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4 import java.applet.*;
5
6 public class ImageAudioAnimation extends JApplet {
7     private final static int NUMBER_OF_NATIONS = 7;
8     private int current = 0;
9     private ImageIcon[] icons = new ImageIcon[NUMBER_OF_NATIONS];
10    private AudioClip[] audioClips = new AudioClip[NUMBER_OF_NATIONS];
11    private AudioClip currentAudioClip;
12
13    private int[] delays =
14        {48000, 54000, 59000, 54000, 59000, 31000, 68000};
15    private Timer timer = new Timer(delays[0], new TimerListener());
16
17    private JLabel jlblImageLabel = new JLabel();
18    private JButton jbtResume = new JButton("Resume");

```

```

19 private JButton jbtSuspend = new JButton("Suspend");
20 private JComboBox jcbnations = new JComboBox(new Object[]
21     {"Denmark", "Germany", "China", "India", "Norway", "UK", "US"});
22
23 public ImageAudioAnimation() {
24     // Load image icons and audio clips
25     for (int i = 0; i < NUMBER_OF_NATIONS; i++) {
26         icons[i] = new ImageIcon(getClass().getResource(
27             "image/flag" + i + ".gif"));
28         audioClips[i] = Applet.newAudioClip(
29             getClass().getResource("audio/anthem" + i + ".mid"));
30     }
31
32     JPanel panel = new JPanel();
33     panel.add(jbtResume);
34     panel.add(jbtSuspend);
35     panel.add(new JLabel("Select"));
36     panel.add(jcbnations);
37     add(jlblImageLabel, BorderLayout.CENTER);
38     add(panel, BorderLayout.SOUTH);
39
40     jbtResume.addActionListener(new ActionListener() {
41         public void actionPerformed(ActionEvent e) {
42             start();
43         }
44     });
45     jbtSuspend.addActionListener(new ActionListener() {
46         public void actionPerformed(ActionEvent e) {
47             stop();
48         }
49     });
50     jcbnations.addActionListener(new ActionListener() {
51         public void actionPerformed(ActionEvent e) {
52             stop();
53             current = jcbnations.getSelectedIndex();
54             presentNation(current);
55             timer.start();
56         }
57     });
58
59     timer.start();
60     lblImageLabel.setIcon(icons[0]);
61     lblImageLabel.setHorizontalAlignment(JLabel.CENTER);
62     currentAudioClip = audioClips[0];
63     currentAudioClip.play();
64 }
65
66 private class TimerListener implements ActionListener {
67     public void actionPerformed(ActionEvent e) {
68         current = (current + 1) % NUMBER_OF_NATIONS;
69         presentNation(current);
70     }
71 }
72
73 private void presentNation(int index) {
74     lblImageLabel.setIcon(icons[index]);
75     jcbnations.setSelectedIndex(index);
76     currentAudioClip = audioClips[index];
77     currentAudioClip.play();
78     timer.setDelay(delays[index]);
79 }
80
81 public void start() {
82     timer.start();
83     currentAudioClip.play();

```

```

84  }
85
86  public void stop() {
87      timer.stop();
88      currentAudioClip.stop();
89  }
90 }

```

第17行创建一个标签来显示国旗图像。第26~27行创建包含七个国旗图像的数组。第28~29行创建一个音频剪辑的数组。通过当前类的URL为每个音频文件创建一个音频剪辑。这些音频文件都存储在和applet类文件相同的目录中。

在第20~21行创建选择国家名的组合框。当在组合框中选择一个新国家名时，当前国家的表示停止，显示新选中的国家（第52~55行）。

presentNation(index)方法（第73~79行）表示一个带指定下标的国家。它在标签里设置一个新图像（第74行），和设置了选中下标的组合框来进行同步（第75行），播放新音频，并且设置新的延迟时间（第78行）。

applet的start和stop方法被覆盖以重新开始和暂停动画（第81~89行）。

关键术语

applet (Java小程序)	HTML (超文本标记语言)
applet container (applet容器)	tag (标记)
archive (存档)	signed applet (信任的applet)

本章小结

- JApplet是Applet的子类。它用于开发带Swing组件的Java applet。
- applet的字节码必须指定，使用HTML文件中的<applet>标记来告诉Web浏览器在何处能找到这个applet。applet可以使用<param>标记接收HTML文件中的字符串参数。
- applet容器通过Applet类中的init、start、stop和destroy方法控制和执行applet。
- 当加载applet时，applet容器调用一个applet的无参构造方法，创建一个applet的实例。创建applet后调用init方法。调用init方法后调用start方法。当再一次访问包含applet的页面再次激活applet时，也会调用start方法。当用户离开applet页面时，调用stop方法。
- 当浏览器正常退出时，就会调用destroy方法以通知applet不再需要它，并且应该释放它所占据的所有资源。stop方法总是在destroy方法之前调用。
- 编写应用程序和编写applet的过程非常相似。applet很容易转换成应用程序，反之亦然。此外，编写一个带主方法的applet可以使之独立运行。
- 在HTML中，可以使用applet标记中的param属性给applet传递参数。调用getParameter(paramName)方法可以获取参数值。
- 只有在创建了applet的一个实例之后，才可以调用Applet的getParameter方法。因此，这个方法不能被applet类的构造方法调用。只能从init方法中调用这个方法。
- 我们学习了如何将一个音频和一个图像放到applet和应用程序中。为了在applet和应用程序中加载音频和图像，必须创建音频和图像的URL。可以为类目录下的文件创建一个URL。
- 要播放音频，要用音频资源的URL创建一个音频剪辑。可以使用AudioClip类的play()方法播放一次，使用loop()方法重复播放，使用stop()方法停止播放。

复习题

18.2~18.6节

- 18.1 每个applet都是java.applet.Applet的一个实例吗？每个applet都是javax.swing.JApplet的一个实例吗？
- 18.2 描述Applet类中的init()、start()、stop()和destroy()方法。
- 18.3 怎样把组件添加到JApplet中？JApplet内容窗格的默认布局管理器是什么？
- 18.4 为什么图a中的applet什么都不显示？为什么图b中的applet在突出显示的行会抛出一个运行异常NullPointerException？

```
import javax.swing.*;

public class WelcomeApplet extends JApplet {
    public void WelcomeApplet() {
        JLabel lblMessage =
            new JLabel("It is Java");
    }
}
```

a)

```
import javax.swing.*;

public class WelcomeApplet extends JApplet {
    private JLabel lblMessage;

    public WelcomeApplet() {
        JLabel lblMessage =
            new JLabel("It is Java");
    }

    public void init() {
        add(lblMessage);
    }
}
```

b)

- 18.5 描述HTML的<applet>标记。如何向applet传递参数？
- 18.6 getParameter方法是在哪里定义的？
- 18.7 按如下修改DisplayMessage的applet，会发生什么错误？

```
public class DisplayMessage extends JApplet {
    /** Initialize the applet */
    public DisplayMessage() {
        // Get parameter values from the HTML file
        String message = getParameter("MESSAGE");
        int x =
            Integer.parseInt(getParameter("X"));
        int y =
            Integer.parseInt(getParameter("Y"));

        // Create a message panel
        MessagePanel messagePanel =
            new MessagePanel(message);
        messagePanel.setXCoordinate(x);
        messagePanel.setYCoordinate(y);

        // Add the message panel to the applet
        add(messagePanel);
    }
}
```

a) 修改1

```
public class DisplayMessage extends JApplet {
    private String message;
    private int x;
    private int y;

    /** Initialize the applet */
    public void init() {
        // Get parameter values from the HTML file
        message = getParameter("MESSAGE");
        x = Integer.parseInt(getParameter("X"));
        y = Integer.parseInt(getParameter("Y"));
    }

    public DisplayMessage() {
        // Create a message panel
        MessagePanel messagePanel =
            new MessagePanel(message);
        messagePanel.setXCoordinate(x);
        messagePanel.setYCoordinate(y);

        // Add the message panel to the applet
        add(messagePanel);
    }
}
```

b) 修改2

- 18.8 应用程序和applet之间有何不同？如何运行一个应用程序？如何运行一个applet？应用程序和applet的编译过程是否相同？列出关于applet的一些安全限制。
- 18.9 能将一个框架放在一个applet中吗？
- 18.10 能将一个applet放在一个框架中吗？
- 18.11 在程序清单18-5的TicTacToe.java中，删除第97行的super.paintComponent(g)之后再运行程序，会发生什么现象？

18.9~18.10节

18.12 如何为类目录下的文件image/us.gif创建一个URL对象?

18.13 如何为类目录下的文件image/us.gif创建一个ImageIcon?

18.11节

18.14 Java中使用哪些类型的音频文件?

18.15 如何用类目录下的文件anthem/us.mid创建一个音频剪辑?

18.16 如何播放、重复播放以及停止播放音频剪辑?

编程练习题

教学注意 对练习题中的每个applet, 都可以添加一个主方法以使它能够独立运行。

18.2~18.6节

18.1 (将应用程序转换成applet) 将程序清单17-2转换成一个applet。

*18.2 (向applet传递字符串) 改写程序清单18-5, 显示带标准颜色、字体和字号的一条消息。

message、x、y、color、fontname和fontsize都是<applet>标记的参数, 如下所示:

```
<applet
  code = "Exercise18_2.class"
  width = 200
  height = 50
  alt = "You must have a Java-enabled browser to view the applet"
>
  <param name = MESSAGE value = "Welcome to Java" />
  <param name = X value = 40 />
  <param name = Y value = 50 />
  <param name = COLOR value = "red" />
  <param name = FONTNAME value = "Monospaced" />
  <param name = FONTSIZE value = 20 />
</applet>
```

18.3 (贷款计算器) 编写一个applet计算贷款偿还额, 如图18-16所示。用户可以输入利率、年数和贷款总额, 然后点击Compute Payment按钮来显示月偿还额和总偿还额。

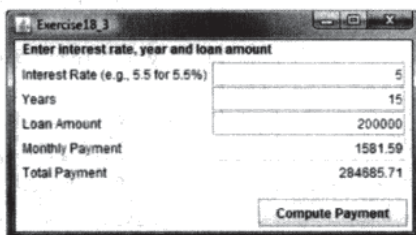


图18-16 applet计算贷款偿还额

*18.4 (将应用程序转换成applet) 将程序清单16-12中的ClockAnimation改写为一个applet并使之能够独立运行。

**18.5 (游戏: 时钟学习工具) 开发一个时钟applet, 演示一年级学生如何读时钟。修改练习题15.19, 在applet中显示一个带小时和分钟指针的详细时钟, 如图18-17a所示。小时和分钟的值是随机产生的。小时的取值范围在0到11之间, 分钟的取值是0、15、30或45。在时钟上点击鼠标就会显示新的随机时间。

**18.6 (游戏: 井字游戏) 做以下的修改来改写18.9节:

(1) 将Cell声明为一个独立的类, 而不是内部类。

(2) 添加一个名为New Game的按钮, 如图18-17b所示。这个New Game按钮会启动一个新游戏。

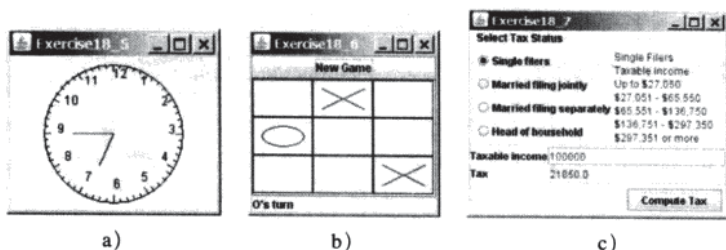


图18-17 a) 在时钟上点击鼠标，就会随机显示一个时钟时间；b) 按钮New Game启动一个新游戏；
c) 税款计算器根据指定的可征税收入和纳税人状况来计算税款

****18.7 (财务应用：税款计算器)** 创建一个计算税款的applet，如图18-17c所示。这个applet允许用户选择纳税人状况，并且输入可征税收入，然后根据2001年美国联邦税率计算税款，如练习题10.8所示。

*****18.8 (创建一个计算器)** 使用FlowLayout、GridLayout和BorderLayout的面板设置下面的计算器，并实现加法(+)、减法(-)、除法(/)、开平方(sqrt)和求余(%)的功能(参见图18-18a)。

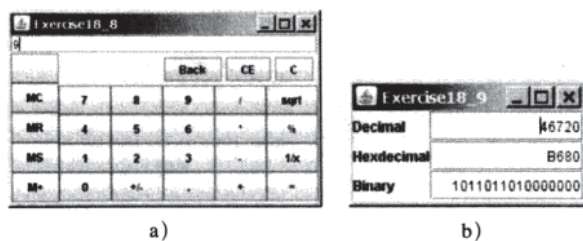


图18-18 a) 练习题18.8是用Java实现一个流行的计算器；

b) 练习题18.9实现十进制数、十六进制数和二进制数之间的转换

***18.9 (数的进制转换)** 编写一个applet，实现十进制数、十六进制数和二进制数之间的转换，如图18-18b所示。在十进制数的文本域中输入一个十进制数之后点击回车键之后，就会在另外两个文本域中显示它对应的十六进制数和二进制数。类似地，也可以在其他文本域中输入值，然后进行相应的转换。

****18.10 (重绘部分区域)** 重画面板的整个视图区域时，有时视图区域中只有一小部分区域发生了改变。只重画面板中受影响的部分是可以提高效率的。因为super.paintComponent(g)方法会导致视图区域全部清空，所以重绘面板时不能调用这个方法。采用部分重画的方式编写一个applet，在直方图中显示最近24小时中每个小时的温度。假设温度是在50~90华氏温度之间的随机温度，而且每小时更新一次。当前小时的温度需要重新显示，而其他各小时的温度图像保持不变。使用唯一的颜色突出显示当前小时的温度(参见图18-19a)。

****18.11 (模拟：一个转动的风扇)** 编写一个Java applet模拟转动的风扇，如图18-19b所示。按钮Start、Stop和Reverse控制风扇。滚动条控制风扇的速度。创建JPanel的一个子类Fan来显示这个风扇。该类还包含暂停和重新启动风扇、设置风扇速度和颠倒转动方向的方法。创建一个名为FanControl的类，它包含一个风扇、三个按钮和一个控制风扇的滚动条。创建一个包含FanControl实例的Java applet。

****18.12 (控制一组风扇)** 编写一个Java applet，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18-20所示。

*****18.13 (创建一个电梯模拟器)** 编写一个applet，模拟电梯的上升和下降(如图18-21所示)。左边的按钮表示乘客现在所在的楼层。乘客必须先点击左边的按钮，要求电梯到达他(她)所处的楼层。

进入电梯后，乘客点击右边的按钮指定想要到达的楼层。

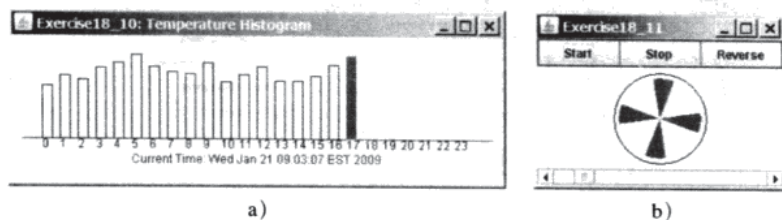


图18-19 a) 直方图显示最近24小时内每小时的平均温度；b) 程序模拟一个转动的风扇

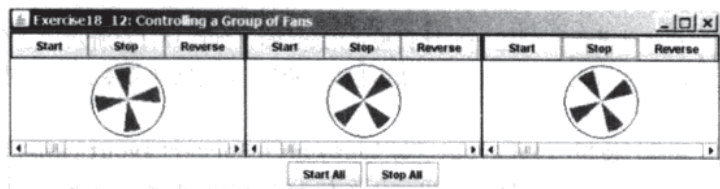


图18-20 程序运行和控制一组风扇

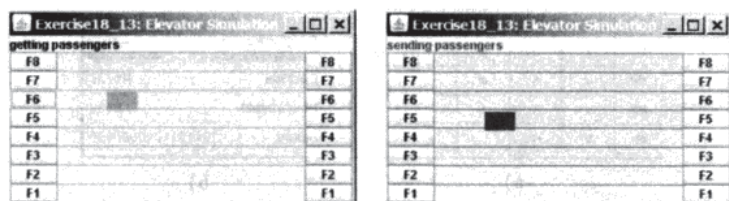


图18-21 程序模拟电梯的运转

*18.14 (控制一组时钟) 编写一个Java applet，显示一组中的三个时钟，通过控制按钮来开始或停止它们，如图18-22所示。

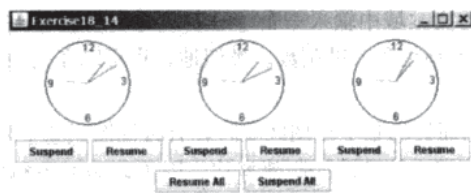


图18-22 三个独立运行的时钟，既可以单独控制又可以统一控制

18.10~18.12节

*18.15 (放大或缩小一个图像) 编写一个applet，用同一个图像文件显示一系列大小不同的图像。初始状态时，这个图像的视图区域宽为300、高为300。程序应该使视图区域持续缩小到宽为50、高为50，每次宽和高都缩小1。达到这一状态后，视图区域应该持续扩大到宽为300、高为300，每次宽和高都增加1。这样，这个视图区域应该缩小和放大（交替地）一个图像以产生一个动画。

***18.16 (模拟股票行情) 编写一个Java applet，显示股票指数行情（参见图18-23）。股票指数信息是从HTML文件中的<param>标记传递而来的。每个指数都有4个参数：指数名（例如，S&P 500）、当前时间（例如，15:54）、前一天的指数（例如，919.01）和变化量（例如，4.54）。至少使用5种指数，例如，道琼斯指数、标准普尔500指数、纳斯达克指数、日经指数和黄金与白银指数。用绿色表示上涨，红色表示下跌。这些指数在applet的视图区域中从右向左移动。当按下鼠标按钮时，这个applet就停在这个股票处；当放开按钮后，它又开始移动。

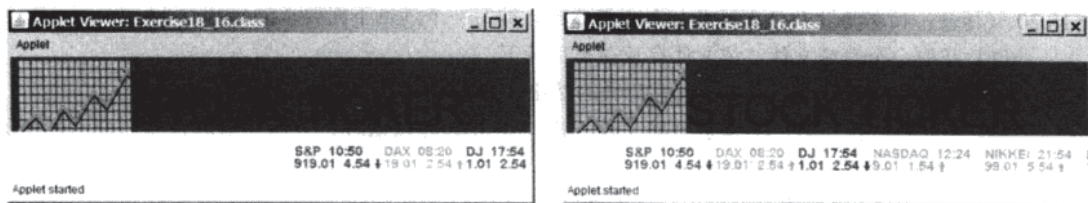


图18-23 程序显示股票指数行情

****18.17 (赛车)** 编写applet模拟四辆赛车, 如图18-24a所示。可以对每辆赛车设置速度, 用1表示最高速。

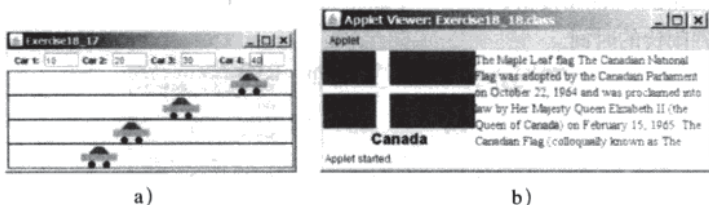


图18-24 a) 可以为每辆车设置速度; b) 这个applet一个接一个地显示每个国家的国旗、名字以及简介, 并且朗读当前显示的国家简介

****18.18 (显示国旗)** 编写一个applet, 通过展示每个国旗的图像、国名和简介一个接一个地介绍国旗 (参见图18-24b), 并且播放朗读简介的音频。

假设applet显示8个国家的国旗。这些照片图像文件的文件名为flag0.gif, flag1.gif, ..., flag7.gif, 都存储在applet目录下一个名为image的子目录中。每个音频的长度不超过10秒。假设每个国家的国名和国旗简介都是从HTML的参数传递得到的, 参数分别为name0, name1, ..., name7和description0, description1, ..., description7。使用numberOfCountries将国家的个数作为HTML的参数来传递。下面是一个例子:

```
<param name = "numberOfCountries" value = 8>
<param name = "name0" value = "Canada">
<param name = "description0" value = "The Maple Leaf flag
The Canadian National Flag was adopted by the Canadian
Parliament on October 22, 1964 and was proclaimed into law
by Her Majesty Queen Elizabeth II (the Queen of Canada) on
February 15, 1965. The Canadian Flag (colloquially known
as The Maple Leaf Flag) is a red flag of the proportions
two by length and one by width, containing in its center a
white square, with a single red stylized eleven-point
maple leaf centered in the white square.">
```

提示 使用DescriptionPanel类显示图像、国名和文本。DescriptionPanel类在程序清单17-6中介绍。

*****18.19 (弹跳的球)** 18.8节的例子模拟一个弹跳的球。扩展这个例子允许程序产生多个球, 如图18-25a所示。可以使用+1或者-1按钮来增加或减少球的个数, 并且使用Suspend和Resume按钮来暂停或重新开始球的跳动。对每个球随机地指定一种颜色。

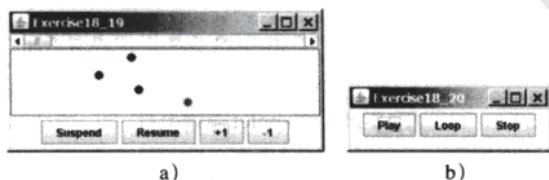


图18-25 a) applet允许添加或删除一个弹跳的球; b) 点击Play播放音频剪辑一次, 点击Loop会重复播放音频, 而点击Stop会终止播放

*18.20 (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applet:

- (1) 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18-25b所示。
- (3) 点击Play按钮时, 会播放音频文件一次。点击Loop按钮时, 会循环播放音频。点击Stop按钮时, 停止播放该音频。
- (4) 该applet能够作为一个应用程序运行。

**18.21 (创建一个闹钟) 编写一个applet, 用一个较大的显示面板来显示小时、分钟和秒的数字钟。这个时钟允许用户设置闹铃时间。图18-26a给出这种时钟的一个例子。选中Alarm复选框可以启用闹铃。点击Set alarm按钮, 显示一个新框架来指定闹铃的时间, 如图18-26b所示。可以在该框架中设置闹铃的时间。

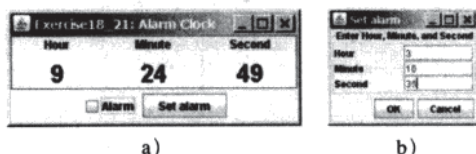


图18-26 程序显示当前的小时、分钟和秒, 并能设置闹铃

**18.22 (创建一个有声的图像动画) 使用applet创建一个动画 (参见图18-27), 满足下面的要求:

- 允许用户指定动画速度。用户可以在文本域中输入速度。
- 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 $L1, L2$ 一直到 Ln 。假设这些图像都存储在image目录下, 该目录是applet所在目录的子目录。
- 允许用户指定音频文件名。音频文件与applet存储在同一个目录下。动画开始时播放这个声音。

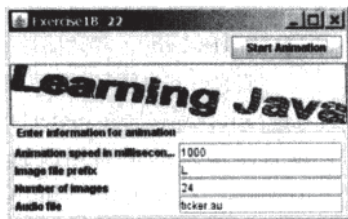


图18-27 这个applet允许用户选择图像文件、音频文件和动画速度

**18.23 (模拟: 升旗并播放国歌) 创建一个显示升国旗的applet, 如图15-1所示。随着国旗的升起, 播放国歌 (可以使用程序清单18-13中的国旗图像和国歌音频文件)。

综合题

**18.24 (游戏: 猜生日) 程序清单3-3给出猜生日的程序。创建一个猜生日的applet, 如图18-28所示。该applet提示用户检查生日是否在五个集合中的任意一个中。在点击Guess Birthday按钮之后文本域显示生日日期。

***18.25 (游戏: 九宫格) 7.7节介绍了九宫格问题。编写一个程序, 让用户在applet的文本域中键入输入, 如图18-1所示。点击Solve按钮显示结果。

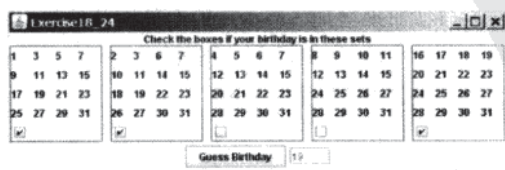


图18-28 该applet猜测生日

***18.26 (游戏: 数学测验) 程序清单3-1和程序清单3-4, 创建数学测验并给其打分。编写一个applet, 允许用户选择题目的类型和难度, 如图18-29a所示。当用户点击Start按钮时, 程序开始生成题目。在用户输入答案和回车键之后, 就会显示新的问题。当用户点击Start按钮时, 就会显示所

用的时间。每秒钟更新时间一次，直到点击Stop按钮。每当给出一个正确答案就更新正确答案的个数。

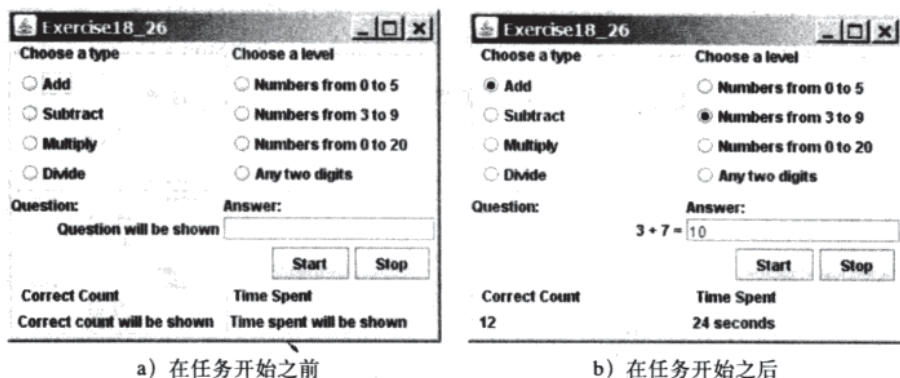


图18-29 这个applet测试数学技巧

***18.27 (模拟: 交通控制) 练习题17.3使用单选按钮改变红绿灯。修改程序模拟十字路口的交通控制, 如图18-30所示。当灯变为红色时, 交通流是垂直的; 当灯变为绿色的, 交通流是水平的。灯每分钟都会自动变化一次。在灯从绿变为红之前, 它首先会变为黄色, 持续大约短短的五秒钟。

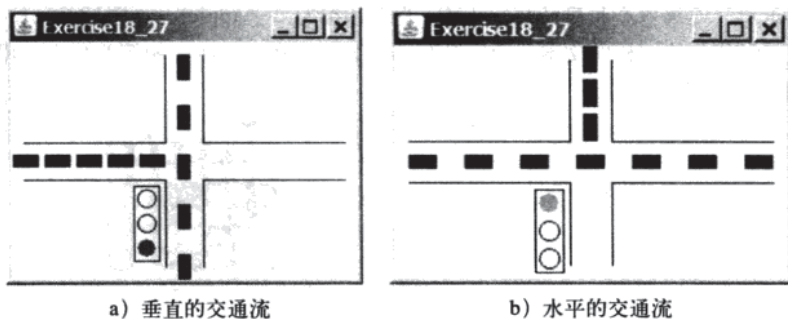


图18-30 这个applet模拟交通控制

**18.28 (几何方面: 两个圆是否相交?) Circle2D类定义在练习题10.11中。编写一个applet, 让用户确定圆的位置和大小, 并且显示这两个圆是否相交, 如图18-31a所示。

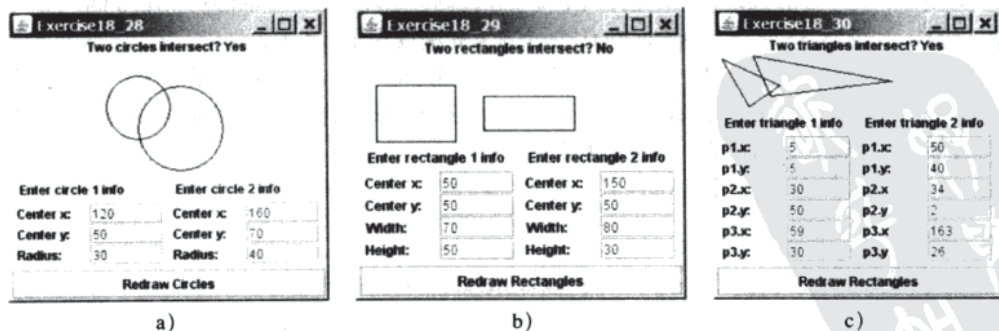


图18-31 检查两个圆、两个矩形和两个三角形是否相交

**18.29 (几何方面: 两个矩形是否相交?) MyRectangle2D类定义在练习题10.12中。编写一个applet, 让用户确定矩形的位置和大小, 并且显示这两个矩形是否相交, 如图18-31b所示。

****18.30** (几何方面: 两个三角形是否相交?) `Triangle2D`类定义在练习题10.13中。编写一个applet, 让用户确定两个三角形的位置和大小, 并且显示这两个三角形是否相交, 如图18-31c所示。

*****18.31** (游戏: 豆机动画) 编写一个applet, 用动画实现练习题16.22所介绍的豆机。这个applet允许设置槽数, 如图18-32所示。点击Start来启动或再次启动动画, 点击Stop来停止这个动画。

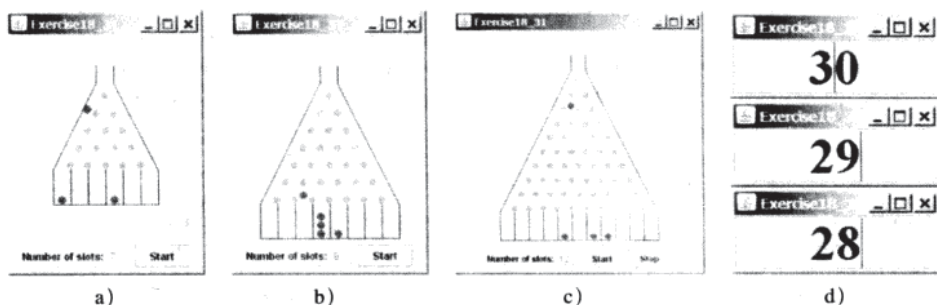


图18-32 a) ~c) applet控制豆机的动画; d) applet进行时间的倒计时

***18.32** (倒计时警告) 编写一个applet, 允许用户在文本域中输入分钟数, 然后点击回车键对分钟数进行倒计时, 如图18-32d所示。每一分钟都会重新显示剩余的分钟数。当倒计时结束时, 程序开始持续播放音乐。

****18.33** (模式识别: 连续四个相同的数) 为练习题7.19编写一个applet, 如图18-33a和图18-33b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, 用户就可以点击Solve按钮突出显示它们。

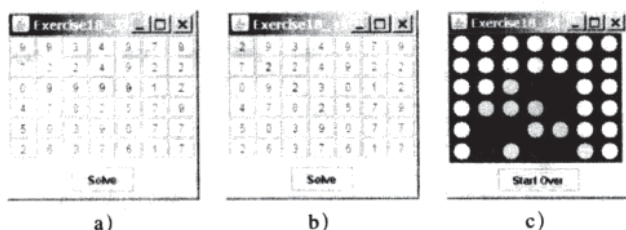


图18-33 a) ~b) 点击Solve按钮突出显示在一行、一列或对角线上四个连续的数字;
c) 该applet是让两个玩家玩连接四子的游戏

*****18.34** (游戏: 连四子) 练习题7.20允许两个玩家在控制台上玩连接四子的游戏。使用applet改写这个程序, 如图18-33c所示。这个applet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上点击。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。如果一个玩家赢了, 这个applet就闪烁这四个赢的格子, 如果所有格子都被占用但还没有赢家, 就报告无赢家。

二进制I/O

学习目标

- 了解在Java中如何处理I/O (19.2节)。
- 区分文本I/O与二进制I/O的不同 (19.3节)。
- 使用`FileInputStream`和`FileOutputStream`来读写字节 (19.4.1节)。
- 使用基类`FilterInputStream`和`FilterOutputStream`来过滤数据 (19.4.2节)。
- 使用`DataInputStream`或`DataOutputStream`来读写基本类型值和字符串 (19.4.3节)。
- 使用`ObjectOutputStream`和`ObjectInputStream`实现对象的存储与恢复, 理解如何序列化对象以及什么样的对象才可以序列化 (19.6节)。
- 实现`Serializable`接口使对象可序列化 (19.6.1节)。
- 序列化数组 (19.6.2节)。
- 使用`RandomAccessFile`对文件进行读写 (19.7节)。

19.1 引言

在文本文件 (text file) 中存储的数据是以我们能读懂的方式表示的。而在二进制文件 (binary file) 中存储的数据是用二进制形式表示的。我们是读不懂二进制文件的, 因为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阅读, 但是, Java类存储在二进制文件中, 可以被Java虚拟机阅读。二进制文件的优势在于它的处理效率比文本文件高。

尽管从技术上讲不怎么准确和正确, 但是可以作这样一个比喻, 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byte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 ($199 = 12 \times 16 + 7$)。

Java提供了许多实现文件输入/输出的类。这些类可以分类为文本I/O类 (text I/O class) 和二进制I/O类 (binary I/O class)。在9.7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。本节介绍实现二进制I/O的类。

19.2 在Java中如何处理输入/输出

回顾一下, `File`对象封装文件或路径属性, 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, 可以将它们分类为输入类和输出类。输入类包括读数据的

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19-1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）。

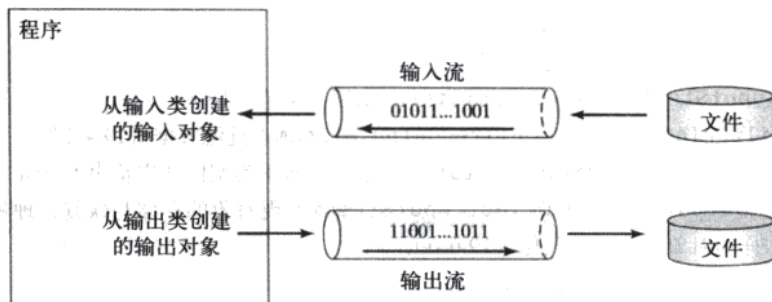


图19-1 程序通过输入对象接收数据，通过输出对象发送数据

19.3 文本I/O与二进制I/O

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，它能提供字符层次的编码和解码的抽象，如图19-2a所示。在文本I/O中自动进行编码和解码。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'的就是57（十六进制数是0x39）。所以，为了写入字符串"199"，就应该将三个字节0x31、0x39和0x39发送到输出，如图19-2a所示。

注意 新版本的Java支持补充的统一码（supplementary Unicode）。但是，为了简单起见，本书只考虑从0到FFFF的原始统一码。

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，就是将内存中的确切值复制到文件中。例如，一个byte类型的数值199在内存中表示为0xC7（ $199 = 12 \times 16 + 7$ ），并且在文件中实际显示的也是0xC7，如图19-2b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。

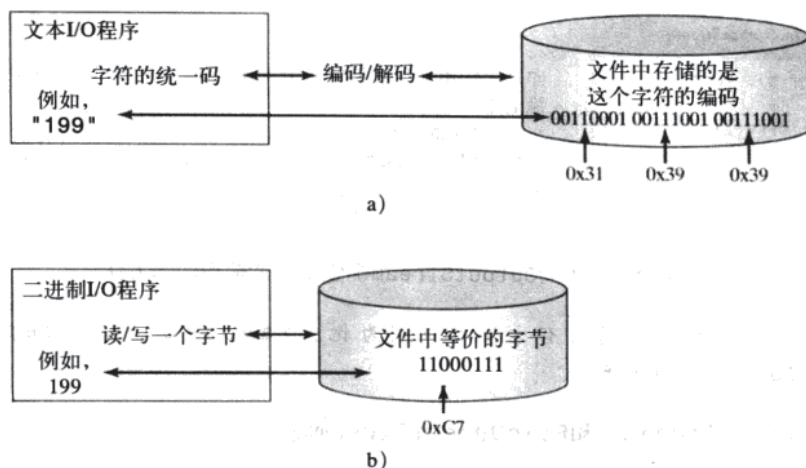


图19-2 文本I/O需要编码和解码，而二进制I/O不需要

19.4 二进制I/O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，而子类提供专门的操作。图19-3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19-4和图19-5列出了`InputStream`类和`OutputStream`类的所有方法。

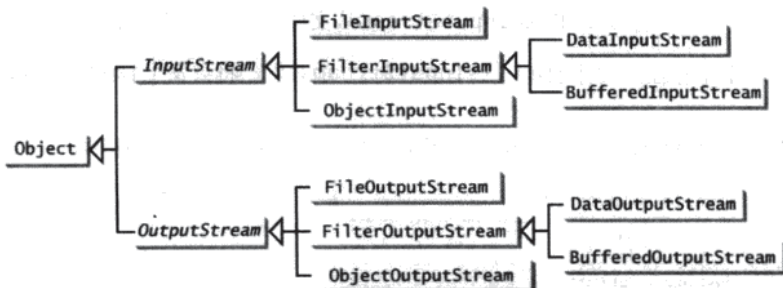


图19-3 用于二进制I/O的InputStream类、OutputStream类及其子类

<i>java.io.InputStream</i>	
<pre> +read(): int +read(b: byte[]): int +read(b: byte[], off: int, len: int): int +available(): int +close(): void +skip(n: long): long +markSupported(): boolean +mark(readlimit: int): void +reset(): void </pre>	<p>从输入流读取数据的下一个字节。返回的字节是一个在0到255之间的整数值。如果因为到达这个流的末尾而无字节可用，则返回-1</p> <p>从输入流读取大到b.length的字节给数组，然后返回读取的实际字节数。在流的末尾返回-1</p> <p>从输入流读取字节，并将它们存储到b[off]、b[off+1]、...b[off+len-1]。返回实际读取的字节数。在流的末尾返回-1</p> <p>返回能从输入流读取的字节数的估计值</p> <p>关闭这个输入流并释放它所占的系统资源</p> <p>跳过和丢弃输入流的n字节数据。返回实际跳过的字节数</p> <p>测试这个输入流是否支持mark和reset方法</p> <p>标记这个输入流的当前位置</p> <p>最后一次调用这个输入流上的mark方法时复位这个流</p>

图19-4 抽象的InputStream类定义字节输入流的方法

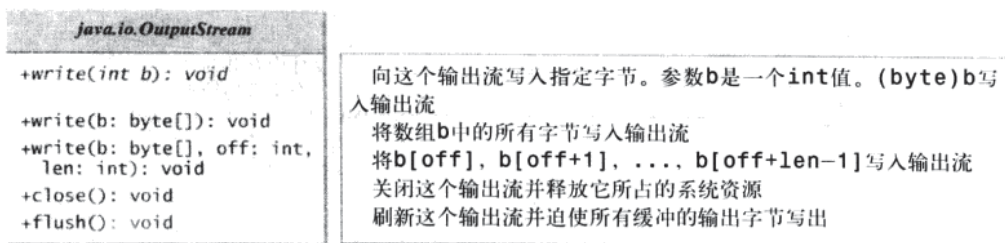


图19-5 抽象的OutputStream类定义字节输出流的方法

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。

19.4.1 FileInputStream类和FileOutputStream类

`FileInputStream`类和`FileOutputStream`类是为了从/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19-6所示。

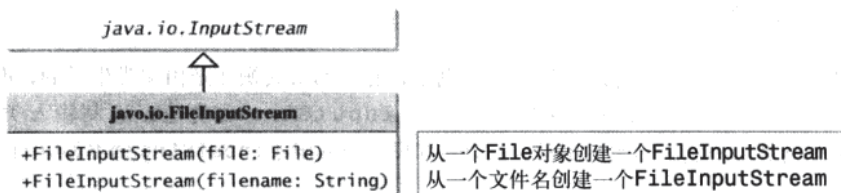


图19-6 FileInputStream从文件输入一个字节流

如果试图为一个不存在的文件创建`FileInputStream`对象，将会发生`java.io.File Not Found-Exception`异常。

要构造一个`FileOutputStream`对象，使用下面的构造方法，如图19-7所示。

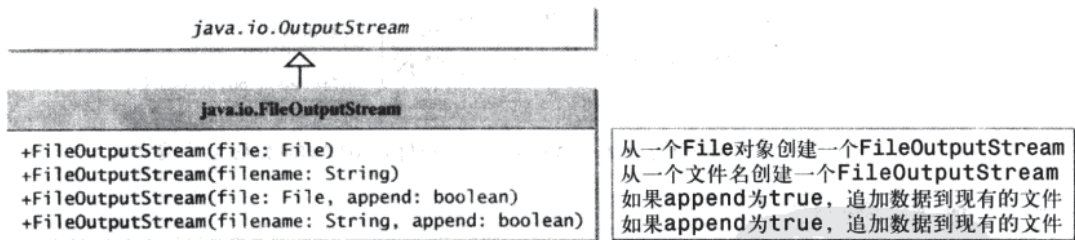
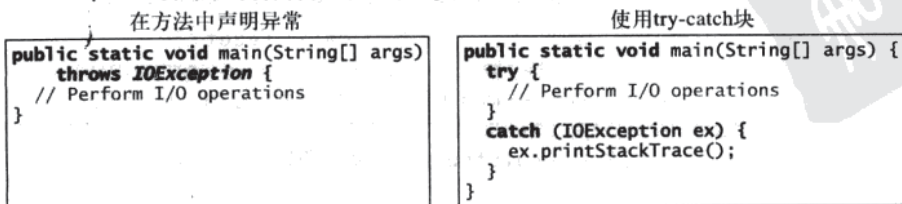


图19-7 FileOutputStream将一个字节流输出到文件中

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`。

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：



程序清单19-1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，再把它从文件中读出来。

程序清单19-1 TestFileStream.java

```

1 import java.io.*;
2
3 public class TestFileStream {
4     public static void main(String[] args) throws IOException {
5         // Create an output stream to the file
6         FileOutputStream output = new FileOutputStream("temp.dat");
7
8         // Output values to the file
9         for (int i = 1; i <= 10; i++)
10             output.write(i);
11
12         // Close the output stream
13         output.close();
14
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         // Close the output stream
24         input.close();
25     }
26 }

```

1 2 3 4 5 6 7 8 9 10



第6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9~10行）。调用write(i)方法与调用write((byte)i)具有相同的功能。第13行关闭输出流。第16行给文件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19-8所示。

二进制数据

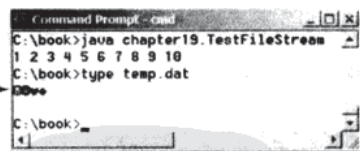


图19-8 二进制文件不能以文本模式显示

提示 当不再需要流时，总是使用close()方法将其关闭。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。

注意 这些文件的根目录是类路径的目录。对于本书中的例子，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8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。可以使用
new PrintWriter(new FileOutputStream("temp.txt", true));

创建一个PrinterWriter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。

19.4.2 FilterInputStream类和FilterOutputStream类

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。

19.4.3 DataInputStream类和DataOutputStream类

DataInputStream从数据流读取字节，并且将它们转换为正确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19-9所示。DataOutputStream类扩展FilterOutputStream类，并实现DataOutput接口，如图19-10所示。

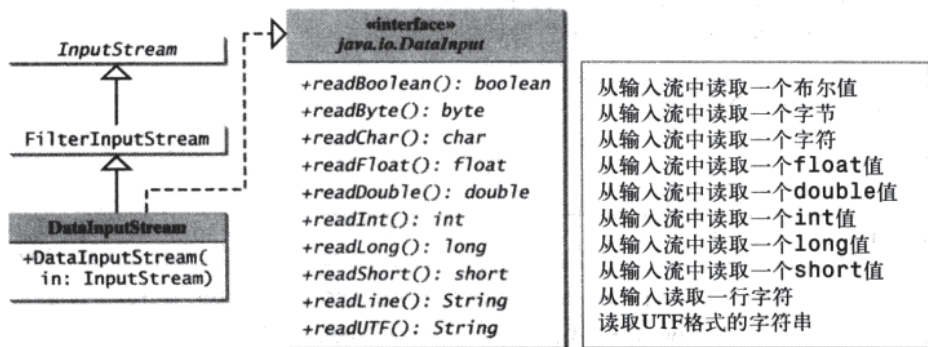


图19-9 DataInputStream过滤字节输入流并将其转化为基本类型值和字符串

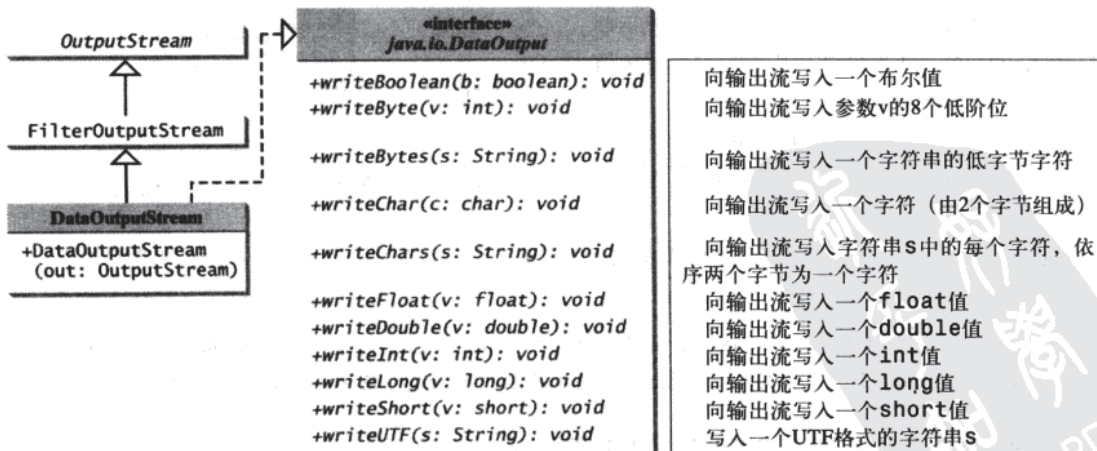


图19-10 DataOutputStream可以将基本数据类型的值和字符串写入输出流

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。`writerChar(char c)`方法将字符`c`的统一码写入输出流。`writerChars(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)`方法将两个字节长度的信息写入输出流，后面紧跟着的是字符串`s`中每个字符的改进版UTF-8的形式。UTF-8是一种编码方案，它允许系统和统一码及ASCII码一起操作的编码方案。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码转化为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二进制数据流。`readUTF()`方法读取一个使用`writeUTF`方法写入的字符串。

UTF-8格式具有以一个字节存储ASCII码的优势，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格式存储的效率是很高的。

2. 使用DataInputStream类和DataOutputStream类

数据流用于对已经存在的输入/输出流进行包装，以便在原始流中过滤数据。可以使用下面的构造方法来创建它们（参见图19-9和图19-10）：

```
public DataInputStream(InputStream instream)
public DataOutputStream(OutputStream outstream)
```

下面给出的语句会创建数据流。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19-2将学生的名字和分数写入名为`temp.dat`的文件中，然后将数据从这个文件中读出来。

程序清单19-2 TestDataStream.java

```
1 import java.io.*;
2
3 public class TestDataStream {
4     public static void main(String[] args) throws IOException {
5         // 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  // Close output stream
18  output.close();
19
20  // Create an input stream for file temp.dat
21  DataInputStream input =
22      new DataInputStream(new FileInputStream("temp.dat"));
23
24  // Read student test scores from the file
25  System.out.println(input.readUTF() + " " + input.readDouble());
26  System.out.println(input.readUTF() + " " + input.readDouble());
27  System.out.println(input.readUTF() + " " + input.readDouble());
28  }
29  }

```

```

John 85.5
Jim 185.5
George 105.25

```



第6~7行为文件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18行关闭输出流。第21~22行为同一个文件创建DataInputStream。第25~27行将这个文件中的学生名字和分数读出，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。

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19-3所示。

程序清单19-3 DetectEndOfFile.java

```

1  import java.io.*;
2
3  public class DetectEndOfFile {
4      public static void main(String[] args) {
5          try {
6              DataOutputStream output = new DataOutputStream
7                  (new FileOutputStream("test.dat"));
8              output.writeDouble(4.5);
9              output.writeDouble(43.25);
10             output.writeDouble(3.2);
11             output.close();
12
13             DataInputStream input = new DataInputStream
14                 (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 read");
21         }
22     }
23 }

```

新华书店
PDF


```

22     catch (IOException ex) {
23         ex.printStackTrace();
24     }
25 }
26 }

```

```

4.5
43.25
3.2
All data read

```



程序使用`DataOutputStream`向文件写入三个双精度值（第6~10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。

19.4.4 `BufferedInputStream`类和`BufferedOutputStream`类

`BufferedInputStream`类和`BufferedOutputStream`类可以通过减少读写次数来提高输入和输出的速度。`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`类为存储字节在流中添加一个缓冲区，以提高处理效率。

可以使用如图19-11和19-12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。

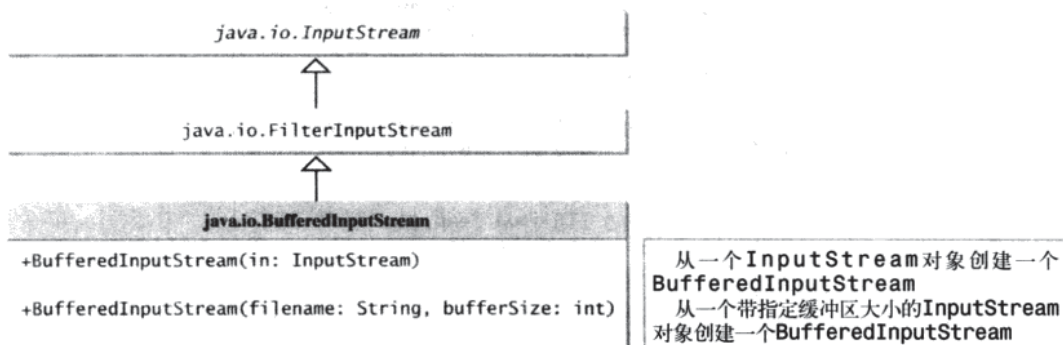


图19-11 `BufferedInputStream`缓冲输入流

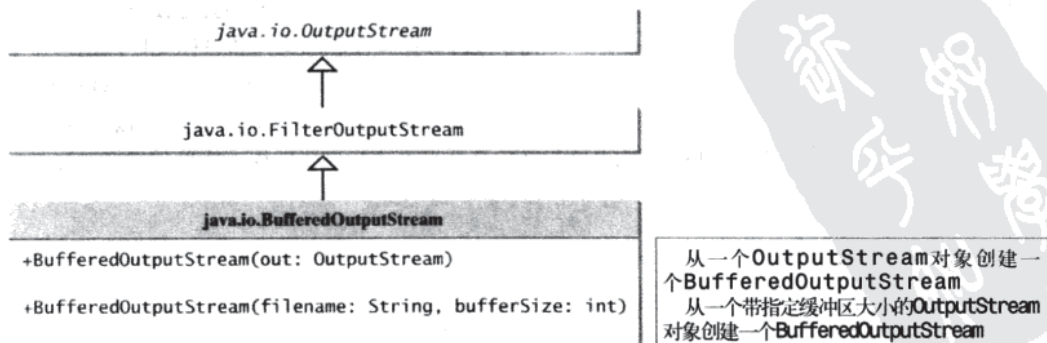


图19-12 `BufferedOutputStream`缓冲输出流

如果没有指定缓冲区大小,默认的大小是512个字节。缓冲区输入流会在每次读取调用中尽可能多地数据读入缓冲区。相反地,只有当缓冲区已满或调用flush()方法时,缓冲输出流才会调用写入方法。

通过在第6~7行与第13~14行给数据流添加缓冲区,可以提高前面例子中TestDataStream程序的效率,如下所示:

```
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IO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IO带来的实质性的性能提升。

19.5 问题:复制文件

本节开发一个复制文件的程序。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 源文件名 目标文件名
```

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,就会告知用户找不到这个文件。如果目标文件已经存在,就告知用户目标文件存在。这个程序的一个运行示例如图19-13所示。

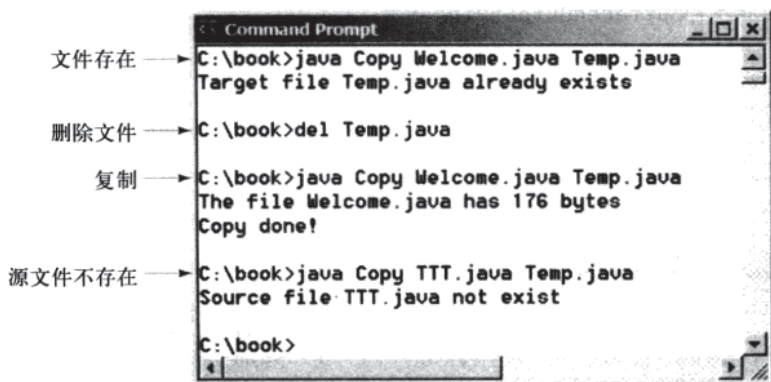


图19-13 程序复制一个文件

要把源文件的内容复制到目标文件,不管文件的内容如何,采用二进制流都会比较合适,使用二进制输入流从源文件读出字节,使用二进制输出流将字节写入目标文件。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19-4给出这个问题的解决方案。

程序清单19-4 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(0);
14 }
15
16 // Check whether source file exists
17 File sourceFile = new File(args[0]);
18 if (!sourceFile.exists()) {
19     System.out.println("Source file " + args[0] + " not exist");
20     System.exit(0);
21 }
22
23 // Check whether target file exists
24 File targetFile = new File(args[1]);
25 if (targetFile.exists()) {
26     System.out.println("Target file " + args[1] + " already
27         exists");
28     System.exit(0);
29 }
30
31 // Create an input stream
32 BufferedInputStream input =
33     new BufferedInputStream(new FileInputStream(sourceFile));
34
35 // Create an output stream
36 BufferedOutputStream output =
37     new BufferedOutputStream(new FileOutputStream(targetFile));
38
39 // Continuously read a byte from input and write it to output
40 int r; int numberOfBytesCopied = 0;
41 while ((r = input.read()) != -1) {
42     output.write((byte)r);
43     numberOfBytesCopied++;
44 }
45
46 // Close streams
47 input.close();
48 output.close();
49
50 // Display the file size
51 System.out.println(numberOfBytesCopied + " bytes copied");
52 }
53 }

```

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18~21行），或者目标文件已经存在，则退出这个程序。

在第32~33行，使用包装在FileInputStream类上的BufferedInputStream类来创建一个输入流，在第36~37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41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，直到读取完所有的字节，然后将它们写入输出流。

19.6 对象的输入/输出

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19-14所示。`ObjectInput`是`DataInput`的子接口。`DataInput`如图19-9所示。`ObjectStreamConstants`包含支持`ObjectInputStream`类和`ObjectOutputStream`类所用的常量。

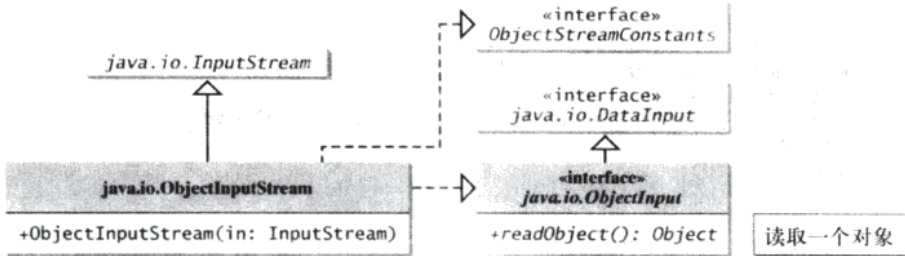


图19-14 `ObjectInputStream`可以读取对象、基本数据类型值和字符串

`ObjectOutputStream`扩展`OutputStream`类，并实现接口`ObjectOutput`与`ObjectStreamConstants`，如图19-15所示。`ObjectOutput`是`DataOutput`的子接口，`DataOutput`如图19-10所示。

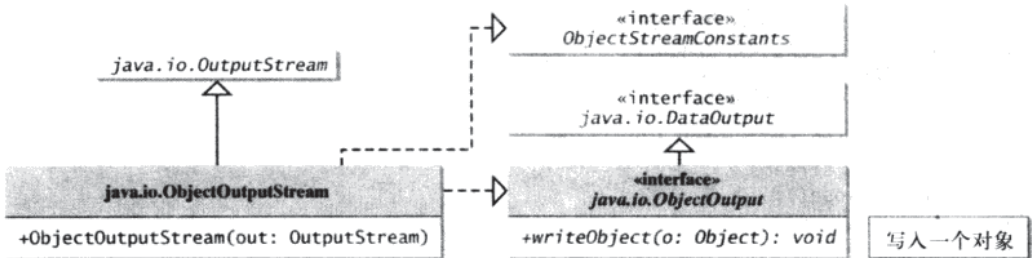


图19-15 `ObjectOutputStream`可以写对象、基本数据类型值和字符串

可以使用下面的构造方法包装任何一个`InputStream`和`OutputStream`上的`ObjectInputStream`和`ObjectOutputStream`：

```
// Create an ObjectInputStream
public ObjectInputStream(InputStream in)
```

```
// Create an ObjectOutputStream
public ObjectOutputStream(OutputStream out)
```

程序清单19-5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19-5 **TestObjectOutputStream.java**

```
1 import java.io.*;
2
3 public class TestObjectOutputStream {
4     public static void main(String[] args) throws IOException {
5         // 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    // Close output stream
15    output.close();
16 }
17 }

```

在第6~7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6~7行, 以完成在数据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19-6从文件object.dat中读回数据。

程序清单19-6 TestObjectInputStream.java

```

1 import java.io.*;
2
3 public class TestObjectInputStream {
4     public static void main(String[] args)
5         throws ClassNotFoundException, IOException {
6         // Create an input stream for file object.dat
7         ObjectInputStream input =
8             new ObjectInputStream(new FileInputStream("object.dat"));
9
10        // Write a string, double value, and object to 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        // Close output stream
17        input.close();
18    }
19 }

```

John 85.5 Mon Jun 26 17:17:29 EDT 2006



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, 在第5行main方法中声明要抛出它。第7~8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行会读出一个字符串、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。

19.6.1 可序列化接口Serializable

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接口是一种标记性接口。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JButton对象,为了完成这个任务,需要存储该对象所有属性(例如,颜色、字体、文本和对齐方式)的当前值。由于JButton是AbstractButton的一个子类,所以,必须把AbstractButton的属性值和AbstractButton所有父类的属性值都存储起来。如果某个属性是对象类型的(例如,background是Color类型的),存储它就要求存储这个对象的所有属性值。如你所见,这是一个非常烦琐冗长的过程。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。工具类如java.util.Date以及所有的Swing GUI组件类都实现了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从初始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,但它包含的都是非序列化的数据域,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。考虑下面的类:

```
public class Foo implements java.io.Serializable {
    private int v1;
    private static double v2;
    private transient A v3 = new A();
}

class A { } // A is not serializable
```

当Foo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 复制对象 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。

19.6.2 序列化数组

只有数组中的元素都是可序列化的,这个数组才是可序列化的。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19-7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19-7 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", "Jim", "Jake"};
8
9         // Create an output stream for file array.dat
10        ObjectOutputStream output =
```

```

11     new ObjectOutputStream(new FileOutputStream
12         ("array.dat", true));
13
14     // Write arrays to the object output stream
15     output.writeObject(numbers);
16     output.writeObject(strings);
17
18     // Close the stream
19     output.close();
20
21     // Create an input stream for file array.dat
22     ObjectInputStream input =
23         new ObjectInputStream(new FileInputStream("array.dat"));
24
25     int[] newNumbers = (int[])(input.readObject());
26     String[] newStrings = (String[])(input.readObject());
27
28     // Display arrays
29     for (int i = 0; i < newNumbers.length; i++)
30         System.out.print(newNumbers[i] + " ");
31     System.out.println();
32
33     for (int i = 0; i < newStrings.length; i++)
34         System.out.print(newStrings[i] + " ");
35 }
36 }

```

```

1 2 3 4 5
John Jim Jake

```



第15~16行将两个数组写入文件array.dat中，第25~26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[]。

19.7 随机访问文件

到现在为止，所使用的所有数据流都是只读的（read.only）或只写的（write.only）。这些数据流的外部文件都是顺序（sequential）文件，如果不创建新文件就不能更新它们。经常需要修改文件或向文件中插入新记录。Java提供了RandomAccessFile类，允许在文件内的随机位置上进行读写。

RandomAccessFile类实现了DataInput和DataOutput接口，如图19-16所示。图19-9中显示的DataInput接口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）。图19-10中的DataOutput接口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）。

当创建一个RandomAccessFile数据流时，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进行读出和写入：

```
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()就会增大。

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四个字节，现在，文件指针就会从它之前的位置向前移动四个字节，如图19-17所示。

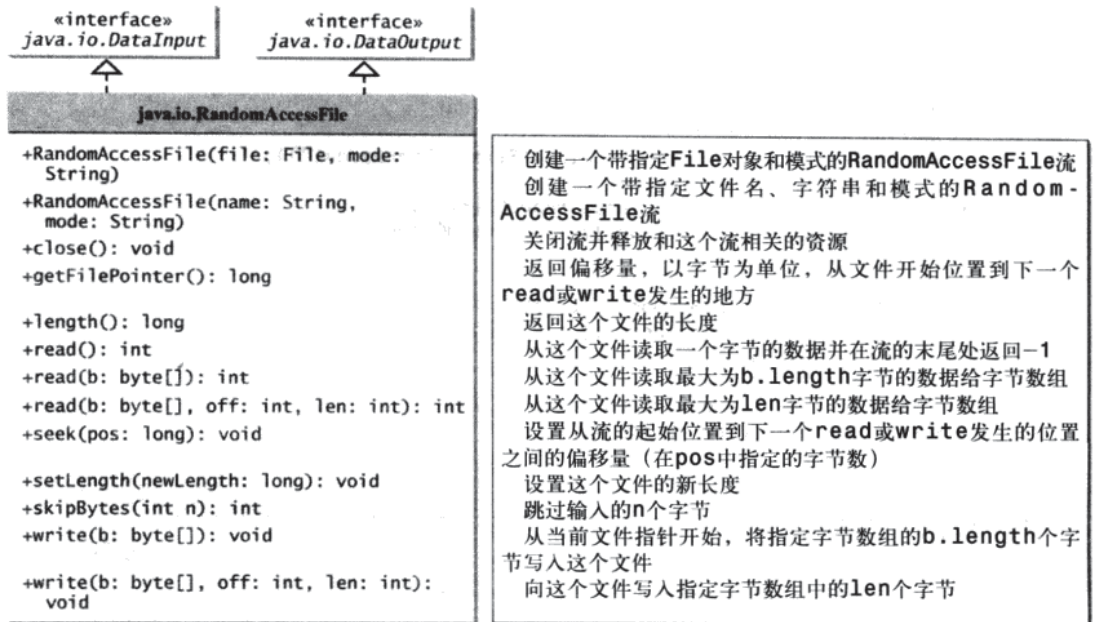


图19-16 RandomAccessFile类实现DataInput和DataOutput接口，并且增加了支持随机访问的方法

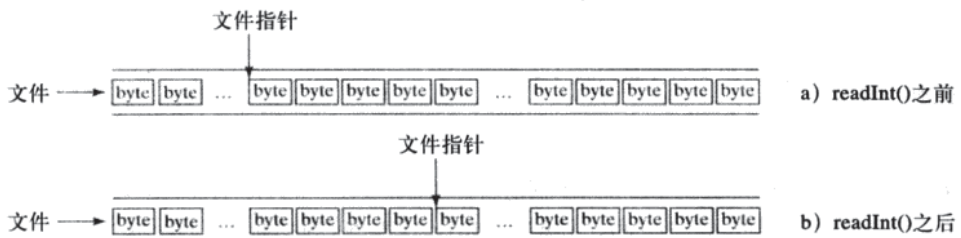


图19-17 读取一个int值之后，文件指针向前移动4个字节

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19-8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VII.B中给出。

程序清单19-8 TestRandomAccessFile.java

```
1 import java.io.*;
2
3 public class TestRandomAccessFile {
4     public static void main(String[] args) throws IOException {
5         // Create a random-access file
6         RandomAccessFile inout = new RandomAccessFile("inout.dat", "rw");
7
8         // Clear the file to destroy the old contents, if any
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 next number
42     System.out.println("The eleventh number is " + inout.readInt());
43
44     inout.close();
45 }
46 }

```

```

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12~13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指向下一个数值。第23行读取第二个数值。

inout.seek(9*4)方法将文件指针指向第10个数值（第27行）。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指向文件的第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。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。

关键术语

binary I/O (二进制输入/输出)

deserialization (反序列化)

file pointer (文件指针)

random-access file (随机访问文件)

sequential-access file (顺序访问文件)

serialization (序列化)

stream (数据流)

text I/O (文本输入/输出)

本章小结

- 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。
- `InputStream`类和`OutputStream`类是所有二进制I/O类的根类。`FileInputStream`类和`FileOutputStream`类用于对文件实现二进制输入和输出。`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。
- `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`标记性接口。
- `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。

复习题

19.1~19.2节

19.1 什么是文本文件，什么是二进制文件？可以使用文本编辑器查看文本文件或二进制文件吗？

19.2 在Java中如何读写数据？什么是数据流？

19.3节

19.3 文本I/O与二进制I/O的区别是什么？

19.4 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？

19.5 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？

19.6 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？

19.7 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？

19.4节

19.8 在Java IO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？

19.9 为什么总是要求关闭数据流？

19.10 `InputStream`可以读取字节。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。

19.11 `FileInputStream`类和`FileOutputStream`类是否引入了新方法？如何创建`FileInputStream`和`FileOutputStream`对象？

19.12 如果视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？

19.13 如何使用 `java.io.PrintWriter` 向一个已存在的文本文件中追加数据？

19.14 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？

19.15 在 `FileOutputStream` 上使用 `writeByte(91)` 方法后，写入文件的是什么？

19.16 在二进制输入流（`FileInputStream` 和 `DataInputStream`）的文件中，如何判断是否已经到达文件末尾？

19.17 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();
        }
    }
}
```

19.18 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 {
        java.io.PrintWriter output =
            new java.io.PrintWriter("t.txt");
        output.printf("%s", "1234");
        output.printf("%s", "5678");
        output.close();
    }
}
```

19.19 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 {
        DataOutputStream output = new DataOutputStream(
            new FileOutputStream("t.dat"));
        output.writeInt(1234);
        output.writeInt(5678);
        output.close();
    }
}
```

19.20 对于下面这些在 `DataOutputStream` 对象 `out` 上的语句，会有多少个字节发送到输出流？

```
output.writeChar('A');
output.writeChars("BC");
output.writeUTF("DEF");
```

19.21 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"))));

ObjectInputStream input3 = new ObjectInputStream(
    new BufferedInputStream(new FileInputStream("t.dat"))));

```

19.6节

- 19.22 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类型是什么？
- 19.23 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。
- 19.24 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？
- 19.25 可以向ObjectOutputStream中写入一个数组吗？
- 19.26 在任何情况下，DataInputStream和DataOutputStream都可以用ObjectInputStream和ObjectOutputStream替换吗？
- 19.27 试图运行下面的代码时，会发生什么？

```

import java.io.*;

public class Test {
    public static void main(String[] args) throws IOException {
        ObjectOutputStream output =
            new ObjectOutputStream(new FileOutputStream("object.dat"));

        output.writeObject(new A());
    }
}

class A implements Serializable {
    B b = new B();
}

class B {
}

```

19.7节

- 19.28 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？
- 19.29 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。
- 19.30 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");
        }
    }
}

```



编程练习题

19.3节

- *19.1 (创建一个文本文件) 编写一个程序, 如果文件Exercise19_1.txt不存在, 就创建一个名为Exercise19_1.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。

19.4节

- *19.2 (创建二进制数据文件) 编写一个程序, 如果文件Exercise19_2.dat不存在, 就创建一个名为Exercise19_2.txt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。
- *19.3 (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)方法创建一个名为Exercise19_3.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。
- *19.4 (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 Exercise19_4 Welcome.java Welcome.utf
```

19.6节

- *19.5 (将对象和数组存储在文件中) 编写一个程序, 向一个名为Exercise19_5.dat的文件中存储一个含5个int值1, 2, 3, 4, 5的数组, 存储一个表示当前时间的Date对象, 存储一个double值5.5。
- *19.6 (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Exercise19_6.dat的文件中。
- *19.7 (从文件中读取对象) 假设已经用ObjectOutputStream创建了一个名为Exercise19_7.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。

19.7节

- *19.8 (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Exercise19_8, 并且将计数器存储在文件Exercise19_8.dat中。
- ***19.9 (地址簿) 补充材料VII.B给出使用随机访问文件来创建和操作一个地址簿的学习实例。通过添加一个如图19-18所示的更新按钮Update来修改这个学习实例, 允许用户修改正在显示的地址。

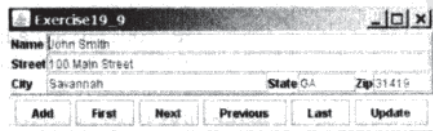


图19-18 这个应用程序可以存储、重新获取以及更新文件的地址

综合题

- *19.10 (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文件分割为小一些的文件:

```
java Exercise19_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。

****19.11 (分割文件的GUI)** 改写练习题19.10使之带有GUI, 如图19-19a所示。

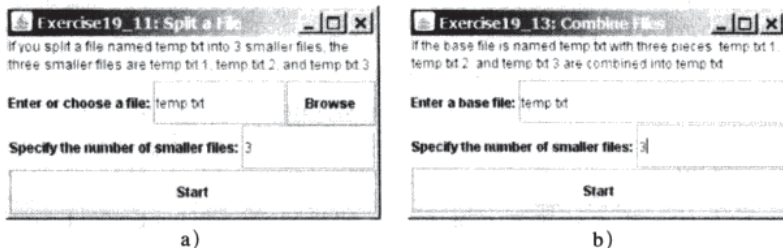


图19-19 a) 程序分割一个文件; b) 程序将文件组合成一个新文件

***19.12 (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 Exercise19_12 SourceFile1 ... SourceFileN TargetFile
```

这个命令将SourceFile1, ..., SourceFileN合并为TargetFile。

***19.13 (组合文件的GUI)** 改写练习题19.12使之带有GUI, 如图19-19b所示。

19.14 (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。

19.15 (解密文件) 假设文件是用练习题19.14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。

19.16 (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, 然后显示文件中的每个字符出现的频率。

****19.17 (BitOutputStream)** 实现一个名为BitOutputStream的类, 如图19-20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非满, 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。参见练习题4.46作为提示。编写一个测试程序, 将比特010000100100001001101发送给一个名为Exercise19_17.dat的文件。

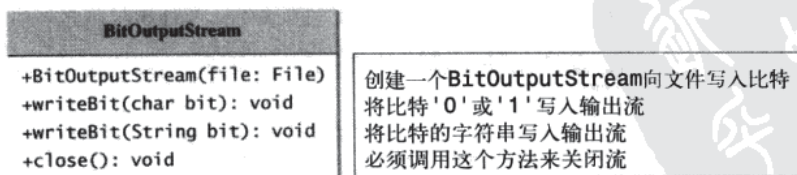


图19-20 BitOutputStream输出比特流给文件

***19.18 (查看比特)** 编写下面的方法, 以显示一个整数的最后一个字节的比特表示:

```
public static String getBits(int value)
```

参见练习题4.46作为提示。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。

*19.19 (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。

**19.20 (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, 然后点击回车键, 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19-21a所示。

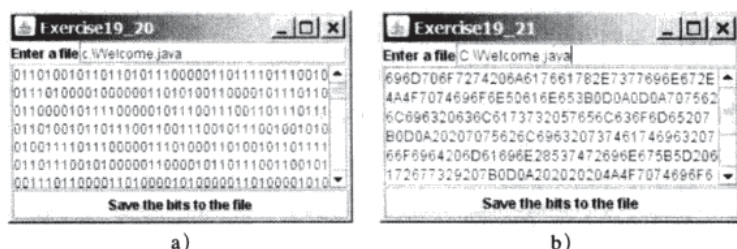


图19-21 该练习题允许用户操作二进制和十六进制形式的文件内容

**19.21 (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19-21b所示。



第20章

Introduction to Java Programming, 8E

递 归

学习目标

- 描述什么是递归方法以及使用递归方法的好处 (20.1节)。
- 开发递归数学函数的递归方法 (20.2~20.3节)。
- 理解在调用栈中如何处理递归方法的调用 (20.2~20.3节)。
- 使用一个重载的辅助方法派生一个递归方法 (20.5节)。
- 使用递归解决选择排序 (20.5.1节)。
- 使用递归解决二分查找 (20.5.2节)。
- 使用递归获取一个目录的大小 (20.6节)。
- 使用递归解决汉诺塔问题 (20.7节)。
- 使用递归绘制分形 (20.8节)。
- 使用递归解决八皇后问题 (20.9节)。
- 了解递归和迭代之间的联系与区别 (20.10节)。
- 了解尾递归方法以及为什么需要它 (20.11节)。

20.1 引言

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。

经典的八皇后难题就是将八个皇后放在棋盘上，而没有任何两个皇后可以互相攻击（即不会出现两个皇后在同一行、同一列或者同一条对角线上的情况），如图20-1所示。该如何编写程序解决这个问题呢？一个好的办法就是使用递归。

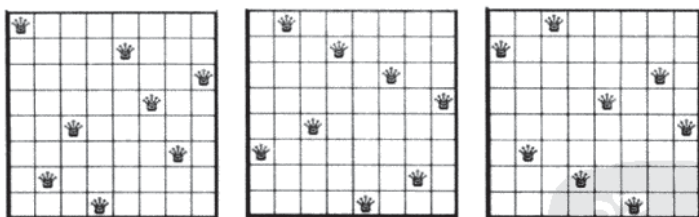


图20-1 可以使用递归解决八皇后问题

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自然、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”。

20.2 问题：计算阶乘

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) × ... × 2 × 1 = 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。子问题在本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，原始问题就可以由`factorial(n-1)`的结果乘以 n 得到。

程序清单20-1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20-1 `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 a 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

到达它的基础情况0。

图20-2描述了从 $n = 4$ 开始执行的递归调用。递归调用对堆栈空间的使用如图20-3所示。

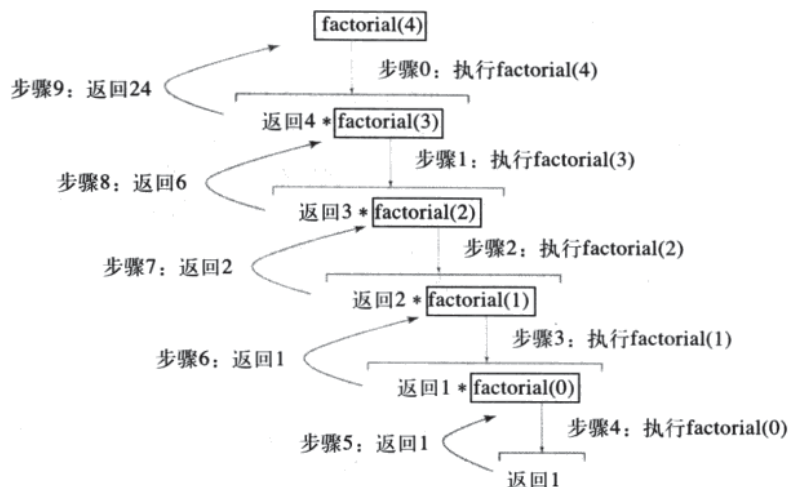


图20-2 调用factorial(4)会引起对factorial的递归调用

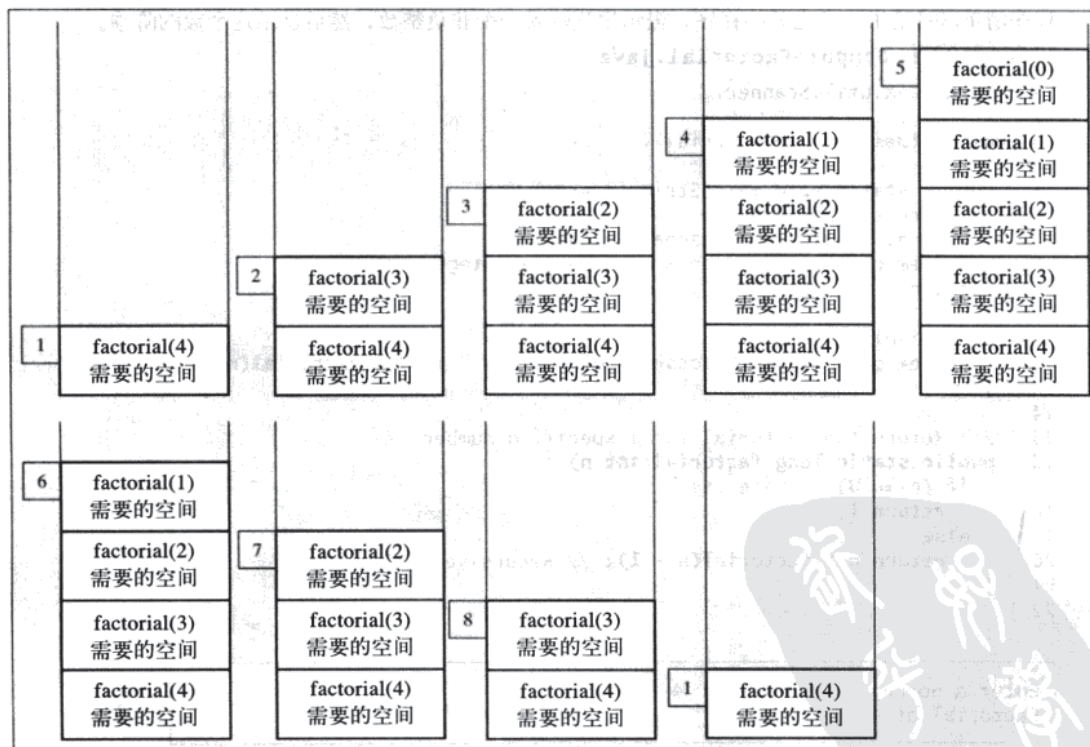


图20-3 执行factorial(4)时，factorial方法被递归调用，导致内存空间动态变化

警告 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。

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，这里使用的递归factorial方法是演示递归概念的一个很好的例子。在本章后续内容中，还将给出一些继承递归以及不使用递归很难解决的问题。

20.3 问题：计算斐波那契数

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自然、直接、简单的解法。考虑众所周知的斐波那契（Fibonacci）数列问题，如下所示：

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。序列可以递归定义为：

```
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。

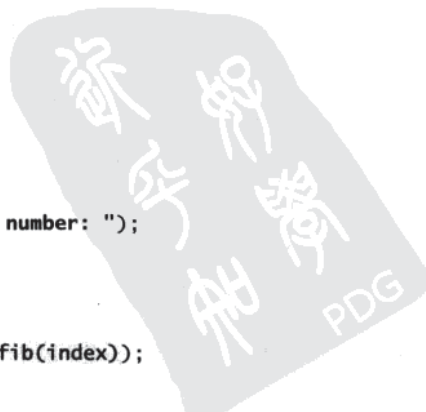
基础情况是index=0或index=1。若用index=0或index=1调用这个方法，它会立即返回结果。若用index>=2调用这个方法，则通过使用递归调用把问题分解成计算fib(index-2)和fib(index-1)的两个子问题。计算fib(index)的递归算法可以简单地描述如下：

```
if (index == 0)
    return 0;
else if (index == 1)
    return 1;
else
    return fib(index - 1) + fib(index - 2);
```

程序清单20-2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20-2 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 the Fibonacci number: ");
9         int index = input.nextInt();
10
11         // Find and display the Fibonacci number
12         System.out.println(
13             "Fibonacci number at index " + 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 the Fibonacci number: 1
Fibonacci number at index 1 is 1



Enter an index for the Fibonacci number: 6
Fibonacci number at index 6 is 8



Enter an index for the Fibonacci number: 7
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。但是，图20-4显示出计算`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20-4中的标签表示方法调用的顺序。

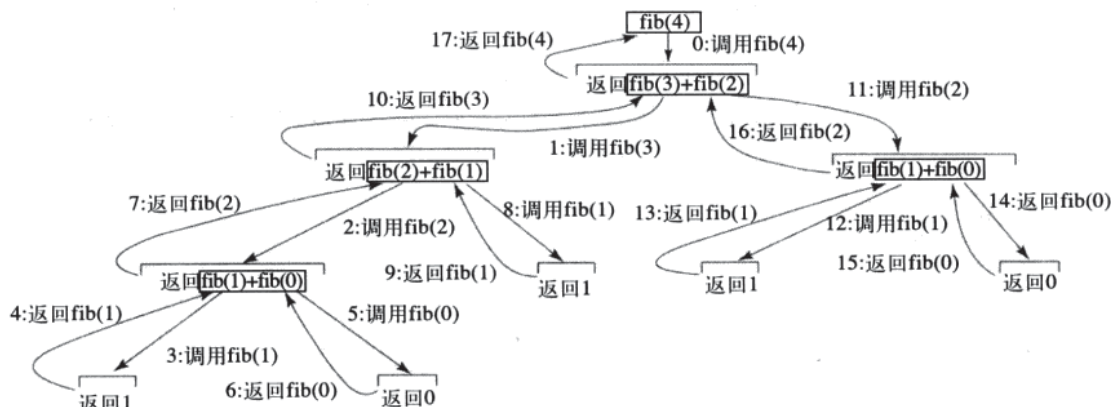


图20-4 调用`fib(4)`会引起对`fib`的递归调用

如图20-4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。

除了大量的递归调用的次数，计算机还需要更多的时间和空间来运行递归的方法。

教学注意 `fib`方法的递归实现非常简单、直接，但是并不高效。参见练习题20.2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。

20.4 使用递归解决问题

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`语句会导致不同的情况。

- 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。

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`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9-1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。

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；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在程序清单20-3中实现。

程序清单20-3 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与检查子串是否是回文的方法是一样的。

20.5 递归的辅助方法

因为前面递归的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)来接受字符串的其余信息，如程序清单20-4所示。

程序清单20-4 RecursivePalindrome.java

```

1 public class RecursivePalindrome {
2     public static boolean isPalindrome(String s) {
3         return isPalindrome(s, 0, s.length() - 1);
4     }
5
6     public 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的辅助方法（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。下面几节将给出两个以上的例子。

20.5.1 选择排序

选择排序在6.10.1节中介绍过。回顾一下，选择排序法是先找到列表的最小数，首先将它放在列表中。然后，在剩余的数中找到最小数，再将它放到第一个数字的后面，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，对剩下的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20-5给出了递归的排序方法。

程序清单20-5 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  public 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，对不断缩小的子数组进行排序。

20.5.2 二分查找

二分查找在6.9.2节中介绍过。使用二分查找法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。当匹配成功时，情况2就是一个基本状态。另一个基本状态是查找完毕而没有一个成功的匹配。程序清单20-6使用递归给二分查找问题一个清晰、简单的解决方案。

程序清单20-6 递归二分查找法

```

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      public 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 }

```

第一个方法是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，在不断缩小的子数组中查找关键字。

20.6 问题：求出目录的大小

前面的例子可以不用递归很容易地解决。对于本节给出的这个问题，要是不使用递归是很难解决的。这里的问题是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20-5所示。

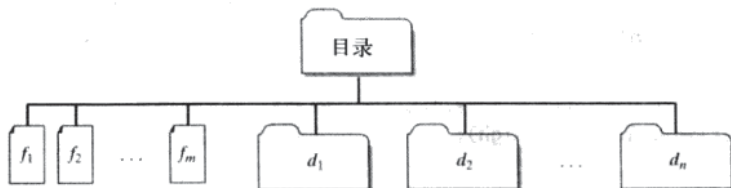


图20-5 一个目录包含多个文件和子目录

目录的大小可以如下递归地定义：

$$\text{size}(d) = \text{size}(f_1) + \text{size}(f_2) + \dots + \text{size}(f_m) + \text{size}(d_1) + \text{size}(d_2) + \dots + \text{size}(d_n)$$

9.6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20-7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20-7 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; 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 }
```




```
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。

20.7 问题：汉诺塔

汉诺塔问题是一个经典的递归例子。这个问题用递归可以很容易地解决，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上移到另一个塔上，移动要遵从下面的规则：

- 1) n 个盘子标记为1, 2, 3, ..., n ，而三个塔标记为A、B和C。
- 2) 任何时候盘子都不能放在比它小的盘子的上方。
- 3) 初始状态时，所有的盘子都放在塔A上。
- 4) 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20-6所示。

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。其中一个很值得一看的网站是www.cut-the-knot.com/recurrence/hanoi.html。

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4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，可以直接得到递归解法。

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的三个子问题，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20-7中的步骤1所示。
- 2) 将盘子 n 从A移到B，如图20-7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20-7中的步骤3所示。

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)
```

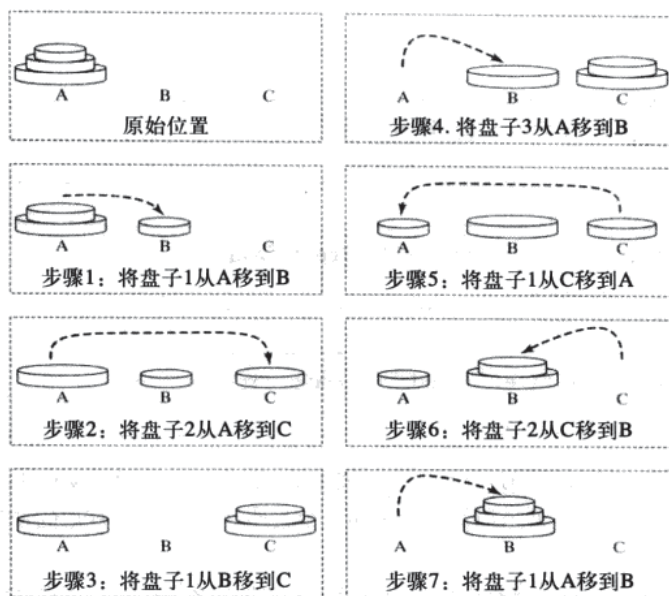


图20-6 汉诺塔问题的目的是在遵从规则的条件下把盘子从塔A移到塔B

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);
}

```

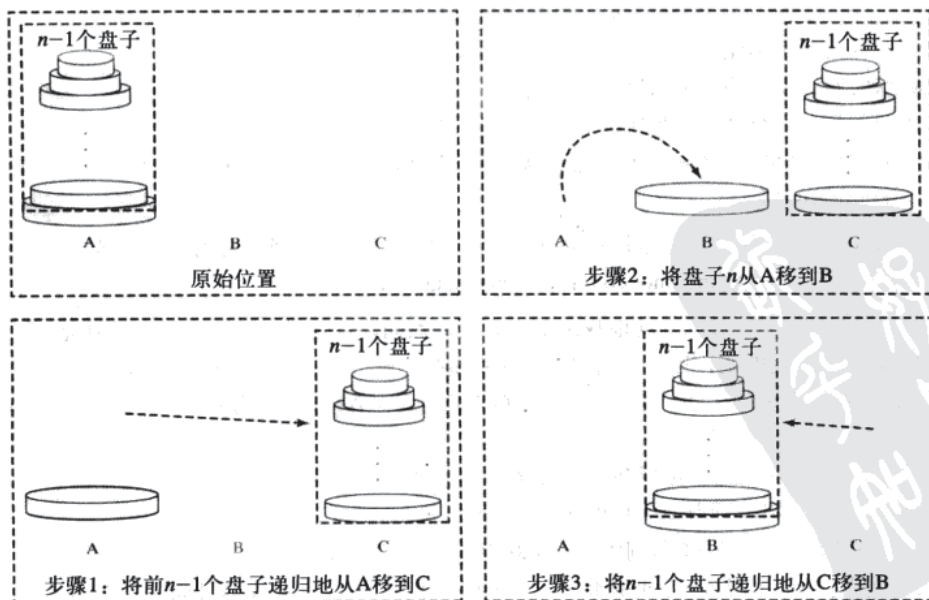


图20-7 汉诺塔问题可以分解成三个子问题

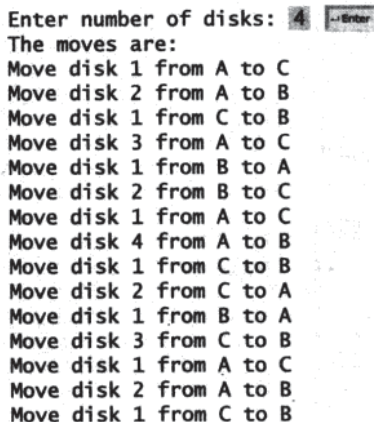
程序清单20-8给出一个程序，提示用户输入盘子个数，然后调用递归的方法moveDisks来显示移动盘子的解决方案。

程序清单20-8 TowerOfHanoi.java

```

1 import java.util.Scanner;
2
3 public class TowersOfHanoi {
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
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，解决这个问题可能会很困难。

考虑跟踪 $n=3$ 的程序。连续的递归调用如图20-8所示。正如所见，编写这个程序比跟踪这个递归调用要容易些。系统使用栈来跟踪后台的调用。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。

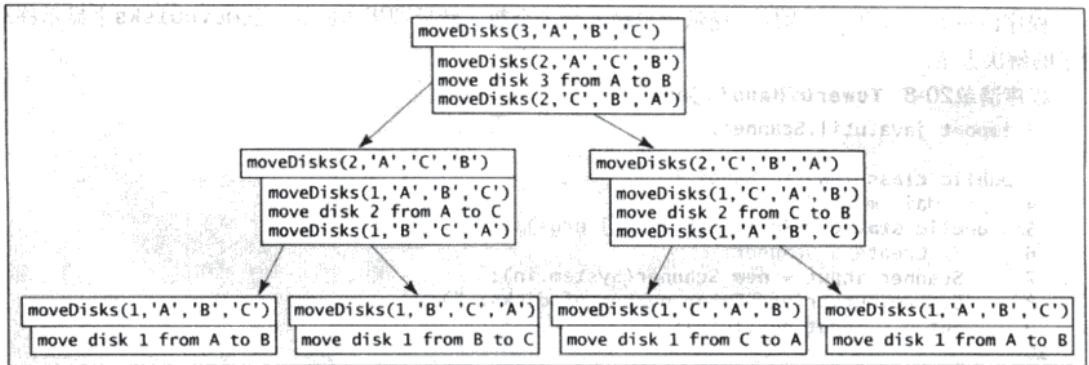


图20-8 调用moveDisks(3, 'A', 'B', 'C')会引起对moveDisks的递归调用

20.8 问题：分形

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20-9a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20-9b）。
- 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20-9c）。
- 4) 可以递归地重复同样的步骤产生3阶，4阶，…的思瑞平斯基三角形（图20-7d）。

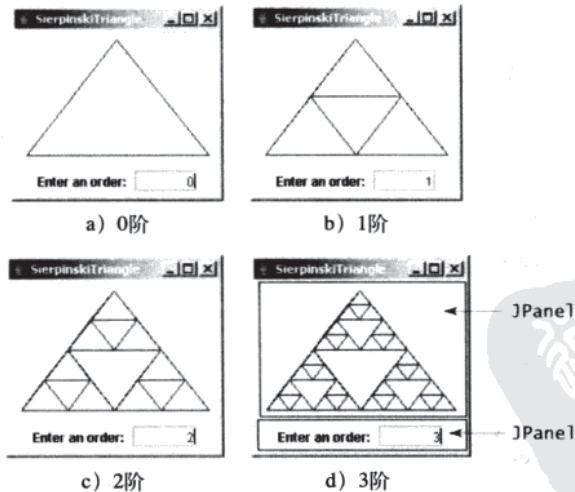


图20-9 思瑞平斯基三角形是一种递归三角形的图形

这个问题本质上是递归的。那么，该如何开发针对该问题的递归方案呢？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20-9给出显示任意阶的思瑞平斯基三角形的Java applet，如图20-9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20-9 SierpinskiTriangle.java

```

1 import javax.swing.*;
2 import java.awt.*;
3 import java.awt.event.*;
4
5 public class SierpinskiTriangle extends JApplet {
6     private JTextField jtfOrder = new JTextField("0", 5); // Order
7     private SierpinskiTrianglePanel trianglePanel =
8         new SierpinskiTrianglePanel(); // To display the pattern
9
10    public SierpinskiTriangle() {
11        // Panel to hold label, text field, and a button
12        JPanel panel = new JPanel();
13        panel.add(new JLabel("Enter an order: "));
14        panel.add(jtfOrder);
15        jtfOrder.setHorizontalAlignment(SwingConstants.RIGHT);
16
17        // Add a Sierpinski triangle panel to the applet
18        add(trianglePanel);
19        add(panel, BorderLayout.SOUTH);
20
21        // Register a listener
22        jtfOrder.addActionListener(new ActionListener() {
23            public void actionPerformed(ActionEvent e) {
24                trianglePanel.setOrder(Integer.parseInt(jtfOrder.getText()));
25            }
26        });
27    }
28
29    static class SierpinskiTrianglePanel extends JPanel {
30        private int order = 0;
31
32        /** Set a new order */
33        public void setOrder(int order) {
34            this.order = order;
35            repaint();
36        }
37
38        protected void paintComponent(Graphics g) {
39            super.paintComponent(g);
40
41            // Select three points in proportion to the panel size
42            Point p1 = new Point(getWidth() / 2, 10);
43            Point p2 = new Point(10, getHeight() - 10);
44            Point p3 = new Point(getWidth() - 10, getHeight() - 10);
45
46            displayTriangles(g, order, p1, p2, p3);
47        }
48
49        private static void displayTriangles(Graphics g, int order,
50            Point p1, Point p2, Point p3) {
51            if (order >= 0) {
52                // Draw a triangle to connect three points
53                g.drawLine(p1.x, p1.y, p2.x, p2.y);
54                g.drawLine(p1.x, p1.y, p3.x, p3.y);
55                g.drawLine(p2.x, p2.y, p3.x, p3.y);
56
57                // Get the midpoint on each edge in the triangle
58                Point p12 = midpoint(p1, p2);
59                Point p23 = midpoint(p2, p3);
60                Point p31 = midpoint(p3, p1);
61
62                // Recursively display three triangles
63                displayTriangles(g, order - 1, p1, p12, p31);

```

```

64         displayTriangles(g, order - 1, p12, p2, p23);
65         displayTriangles(g, order - 1, p31, p23, p3);
66     }
67 }
68
69 private static Point midpoint(Point p1, Point p2) {
70     return new Point((p1.x + p2.x) / 2, (p1.y + p2.y) / 2);
71 }
72 }
73 }

```

初始三角形有三个与面板大小成比例的点集（第42~44行）。如果`order ≥ 0`，那么，`displayTriangles(g, order, p1, p2, p3)`方法完成下面的任务：

- 1) 在第53~55行显示连接三个点`p1`、`p2`、`p3`的三角形，如图20-10a所示。
- 2) 获取`p1`和`p2`的中点（第58行），`p2`和`p3`的中点（第59行），以及`p3`和`p1`的中点（第60行），如图20-10b所示。
- 3) 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63~66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20-10b所示。

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。16.4节中介绍的`Point`类表示组件上的一个点。`midpoint(Point p1, Point p2)`方法返回`p1`和`p2`的中点（第72~74行）。

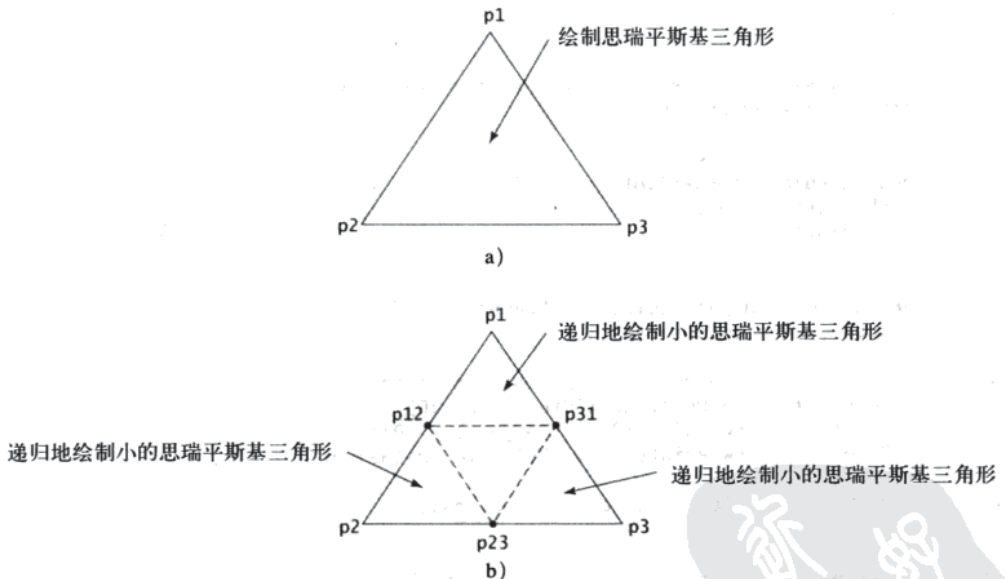


图20-10 绘制一个思瑞平斯基三角形会引起对绘制三个小的思瑞平斯基三角形的调用

20.9 问题：八皇后

本节给出在本章开始时提到的八皇后问题的递归解决方案。它的任务就是找出在棋盘上每行放一个皇后且不出现两个皇后互相攻击的解决方案。可以使用一个二维数组来表示一个棋盘。但是，因为每行都只能有一个皇后，所以，使用一个一维数组来表示皇后在行上的位置就足够了。因此，可以如下定义数组`queens`：

```
int[] queens = new int[8];
```

将j赋值给queens[i]表示皇后放置在i行j列。图20-11a给出如图20-11b所示的棋盘对应的数组queens的内容。

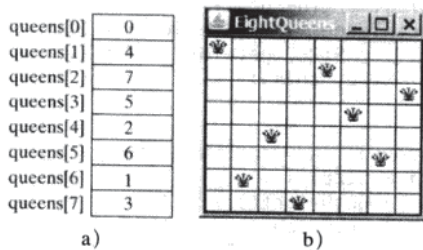


图20-11 queens[i]表示第i行的皇后的位置

程序清单20-10 EightQueens.java

```

1 import java.awt.*;
2 import javax.swing.*;
3
4 public class EightQueens extends JApplet {
5     public static final int SIZE = 8; // The size of the chessboard
6     private int[] queens = new int[SIZE]; // Queen positions
7
8     public EightQueens() {
9         search(0); // Search for a solution from row 0
10        add(new ChessBoard(), BorderLayout.CENTER); // Display solution
11    }
12
13    /** Check if a queen can be placed at row i and column j */
14    private boolean isValid(int row, int column) {
15        for (int i = 1; i <= row; i++)
16            if (queens[row - i] == column // Check column
17                || queens[row - i] == column - i // Check upleft diagonal
18                || queens[row - i] == column + i // Check upright diagonal)
19                return false; // There is a conflict
20        return true; // No conflict
21    }
22
23    /** Search for a solution starting from a specified row */
24    private boolean search(int row) {
25        if (row == SIZE) // Stopping condition
26            return true; // A solution found to place 8 queens in 8 rows
27
28        for (int column = 0; column < SIZE; column++) {
29            queens[row] = column; // Place a queen at (row, column)
30            if (isValid(row, column) && search(row + 1))
31                return true; // Found, thus return true to exit for loop
32        }
33
34        // No solution for a queen placed at any column of this row
35        return false;
36    }
37
38    class ChessBoard extends JPanel {
39        private Image queenImage =
40            new ImageIcon("image/queen.jpg").getImage();
41
42        ChessBoard() {
43            this.setBorder(BorderFactory.createLineBorder(Color.BLACK, 2));
44        }
45
46        protected void paintComponent(Graphics g) {
47            super.paintComponent(g);

```

```

48
49 // Paint the queens
50 for (int i = 0; i < SIZE; i++) {
51     int j = queens[i]; // The position of the queen in row i
52     g.drawImage(queenImage, j * getWidth() / SIZE,
53         i * getHeight() / SIZE, getWidth() / SIZE,
54         getHeight() / SIZE, this);
55 }
56
57 // Draw the horizontal and vertical lines
58 for (int i = 1; i < SIZE; i++) {
59     g.drawLine(0, i * getHeight() / SIZE,
60         getWidth(), i * getHeight() / SIZE);
61     g.drawLine(i * getWidth() / SIZE, 0,
62         i * getWidth() / SIZE, getHeight());
63 }
64 }
65 }
66 }

```

程序调用 `search(0)` (第9行) 开始搜索在第0行的解决方案, 它递归地调用 `search(1)`, `search(2)`, ..., `search(7)` (第30行)。

如果所有的行都被填满, 那么递归的 `search(row)` 方法返回 `true` (第25~26行)。该方法在一个 `for` 循环中检测一个皇后是否放置在第0列, 第1列, 第2列, ..., 第7列中 (第28行)。将一个皇后放置在某一列中 (第29行)。如果这种放法是合法的, 调用 `search(row+1)` 递归地查找下一行 (第30行)。如果查找是成功的, 返回 `true` 以退出这个 `for` 循环 (第31行)。在这种情况下, 无须查找某行的下一列。如果没有将一个皇后放置在这一行的任意一列的解决方案, 这个方法返回 `false` (第35行)。

假设调用行 `row` 为3的 `search(row)`, 如图20-12a所示。这个方法会以第0列, 第1列, 第2列, ... 这样的顺序填充一个皇后。对于每次尝试, 调用 `isValid(row, column)` 方法 (第30行) 来检测是否将一个皇后放在指定的位置会引起和以前放置的皇后的冲突。它确保没有皇后放在同一列 (第16行), 没有皇后放在左上对角线上 (第17行), 没有皇后放在右上对角线上 (第18行), 如图20-12a所示。如果 `isValid(row, column)` 返回 `false`, 那就检查下一列, 如图20-12b所示。如果 `isValid(row, column)` 返回 `true`, 那就递归地调用 `search(row+1)`, 如图20-12d所示。如果 `search(row+1)` 返回 `false`, 那就检查前一行的下一列, 如图20-12c所示。

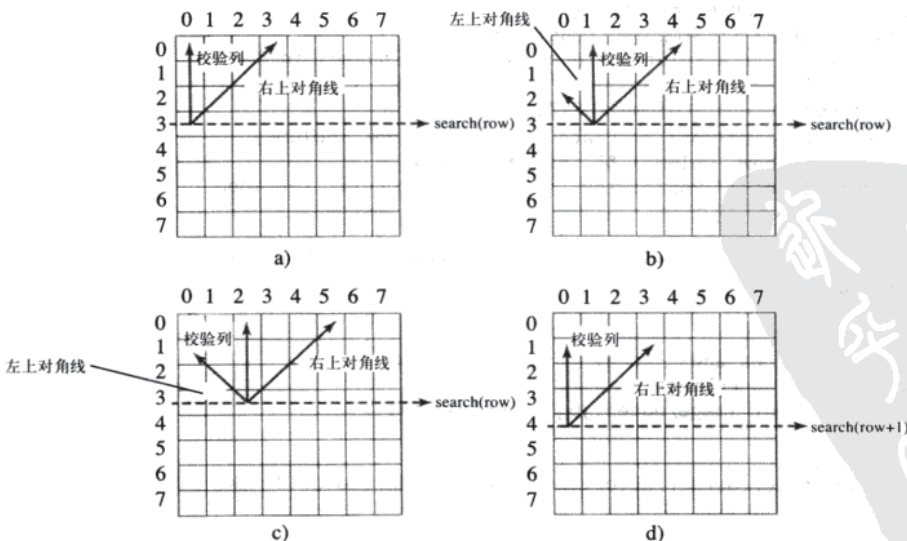


图20-12 调用 `search(row)` 在某行的一列上填充皇后

20.10 递归与迭代

递归是程序控制的一种替代形式，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处理这些附加的空间。

任何用递归解决的问题都可以用迭代非递归地解决。递归有很多副作用：它耗费了太多的时间并占用太多的内存。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。迭代通常都比选择递归效率更高。

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。

20.11 尾递归

如果在从递归调用返回时没有待定的操作要完成，那么这个递归方法就称为尾递归 (tail recursive)。例如，因为在程序清单20-4中的第12行递归调用isPalindrome之后没有待定的操作，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20-1中，因为从每个递归调用返回时都有一个名为multiplication的待定操作要完成，所以，递归的factorial方法（第16~21行）就不是尾递归的。

因为当最后一个递归调用结束时，方法也该结束，所以尾递归是很必要的。因此，无须将中间调用存储在栈中。某些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递归方法。使用这些参数来控制结果，思路是将待定的操作和辅助参数以一种递归调用不再有待定操作的形式相结合。可能会定义一个带辅助参数的新的辅助递归方法，这个方法可以重载名字相同但签名不同的原始方法。例如，程序清单20-1中的factorial方法可以写成尾递归形式，如下所示：

```
1 /** Return the factorial for a specified number */
2 public static long factorial(int n) {
3     return factorial(n, 1); // Call auxiliary method
4 }
5
6 /** Auxiliary tail-recursive method for factorial */
7 private static long factorial(int n, int result) {
8     if (n == 1)
9         return result;
10    else
11        return factorial(n - 1, n * result); // Recursive call
12 }
```

第一个factorial方法只是调用了第二个辅助方法（第3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11行被递归地调用。在返回调用之后，就没有了待定的操作。最终的结果在第9行返回，它也是在第3行调用factorial(n,1)的返回值。

关键术语

base case (基础情况)

infinite recursion (无限递归)

recursive method (递归方法)

recursive helper method (递归辅助方法)

stopping condition (终止条件)

tail recursion (尾递归)

本章小结

- 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。
- 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。
- 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，可以定义递归的辅助方法。
- 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附加的空间。
- 如果从递归调用返回时没有待定的操作要完成，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。

复习题

20.1~20.3节

20.1 什么是递归的方法？描述递归的方法的特点。什么是无限递归？

20.2 编写一个递归的数学定义来计算 2^n ，其中 n 为正整数。

20.3 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。

20.4 编写一个递归的数学定义来计算 $1+2+3+\cdots+n$ ，其中 n 为正整数。

20.5 方法factorial(6)会调用程序清单20-1中的factorial方法多少次？

20.6 方法fib(6)会调用程序清单20-2中的fib方法多少次？

20.4~20.6节

20.7 分别使用程序清单20-3和20.4中定义的方法，给出isPalindrome("abcba")的调用栈。

20.8 使用程序清单20-5中定义的方法，给出selectionSort(new double[] {2,3,5,1})的调用栈。

20.9 什么是递归的辅助方法？

20.7节

20.10 为了调用moveDisks(5, 'A', 'B', 'C')，会调用程序清单20-8中的moveDisks方法多少次？

20.9节

20.11 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。

20.12 引起栈溢出异常的原因是什么？

综合题

20.13 给出下面程序的输出：

```
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);
        }
    }
}
```

20.14 给出下面两个程序的输出：

```
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 + " ");
        }
    }
}
```

20.15 下面的方法有什么错误？

```
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();
    }
}
```

20.16 说明本章中的尾递归方法。

20.17 使用尾递归改写程序清单20-2中的fib方法。

编程练习题

20.2~20.3节

*20.1（计算阶乘）使用14.12节介绍的BigInteger类，求得大数字的阶乘（例如，100!）。编写一个程序，提示用户输入一个整数，然后显示它的阶乘。使用递归实现这个方法。

*20.2（斐波那契数）使用迭代改写程序清单20-2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)

*20.3 (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, 计算gcd(24, 16)和gcd(255, 25)。

20.4 (对数求和) 编写一个递归的方法来求下面的级数:

$$m(i) = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{i}$$

20.5 (对数求和) 编写一个递归的方法来求下面的级数:

$$m(i) = \frac{1}{3} + \frac{2}{5} + \frac{3}{7} + \frac{4}{9} + \frac{5}{11} + \frac{6}{13} + \cdots + \frac{i}{2i+1}$$

*20.6 (对数求和) 编写一个递归的方法来求下面的级数:

$$m(i) = \frac{1}{2} + \frac{2}{3} + \cdots + \frac{i}{i+1}$$

*20.7 (斐波那契数列) 修改程序清单20-2, 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。

20.4节

*20.8 (以逆序输出一个整数中的数字) 编写一个递归的方法, 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。

*20.9 (以逆序输出一个字符串中的字符) 编写一个递归的方法, 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dcba。

*20.10 (字符串中某个指定字符出现的次数) 编写一个递归的方法, 使用下面的方法头求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。

*20.11 (使用递归求一个整数各位数之和) 编写一个递归的方法, 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。

20.5节

**20.12 (以逆序打印字符串中的字符) 使用辅助方法改写练习题20.9, 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)
```

*20.13 (找出数组中的最大数) 编写一个递归的方法, 返回一个数组中的最大整数。

*20.14 (求字符串中大写字母的个数) 编写一个递归的方法, 返回一个字符串中大写字母的个数。

*20.15 (字符串中某个指定字符出现的次数) 使用辅助方法改写练习题20.10, 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)
```

*20.16 (求数组中大写字母的个数) 编写一个递归的方法, 返回一个字符串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的辅助方法。


```
public static int count(char[] chars)
public static int count(char[] chars, int high)
```

*20.17 (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)
```

20.6节

*20.18 (汉诺塔) 修改程序清单20-8, 使程序可以求得将 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。

*20.19 (思瑞平斯基三角形) 修改程序清单20-9, 开发一个applet, 让用户使用Increase和Decrease按钮将当前阶数增1或减1, 如图20-13a所示。初始阶数为0。如果当前阶数为0, 就忽略Decrease按钮。

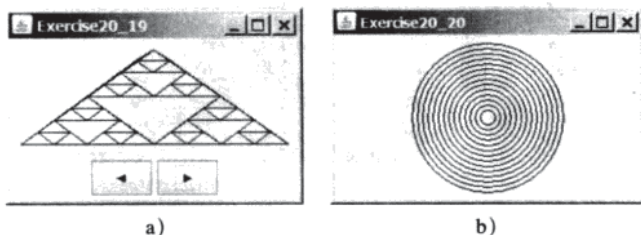


图20-13 a) 编程练习题20.19使用Increase和Decrease按钮将当前阶数增加1或减小1;

b) 练习题20.20使用递归方法绘制多个圆

20.20 (显示多个圆) 编写一个Java applet显示多个圆, 如图20-13b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个像素, 面板和最大圆之间也相距10个像素。

综合题

*20.21 (将十进制数转换为二进制数) 编写一个递归的方法, 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 decimalToBinary(int value)
```

*20.22 (将十进制数转换为十六进制数) 编写一个递归的方法, 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 decimalToHex(int value)
```

*20.23 (将二进制数转换为十进制数) 编写一个递归的方法, 将一个二进制数转换为一个十进制数的字符串。方法头如下:

```
public static int binaryToDecimal(String binaryString)
```

*20.24 (将十六进制数转换为十进制数) 编写一个递归的方法, 将一个十六进制数转换为一个十进制数的字符串。方法头如下:

```
public static int hexToDecimal(String hexString)
```

**20.25 (字符串排列) 编写一个递归的方法, 打印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 定义下面的两个方法, 第二个方法是一个辅助方法。

```
public static void displayPermuation(String s)
public static void displayPermuation(String s1, String s2)
```

第一个方法简单地调用displayPermuation("",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。

****20.26** (创建一个迷宫) 编写一个applet,在迷宫中寻找一条路径,如图20-14a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。
- applet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20-14b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。)

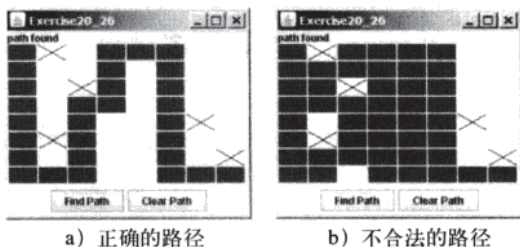


图20-14 程序求出从左上角到右下角的路径

****20.27** (科赫雪花分形) 本章给出了思瑞平斯基三角形分形。在本练习中,要编写一个applet,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20-15a所示。
- (2) 将图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20-15b所示。
- (3) 重复步骤(2)产生2阶科赫分形,3阶科赫分形, ..., 如图20-15c~图20-15d所示。

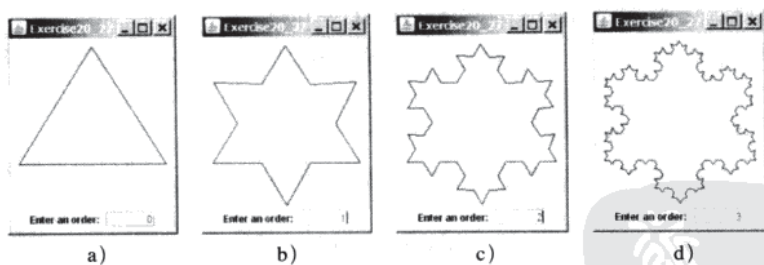


图20-15 科赫雪花是一个从三角形开始的分形

****20.28** (非递归目录大小) 不使用递归改写程序清单20-7。

***20.29** (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。

****20.30** (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 Exercise20_30 dirName word
```

****20.31** (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 Exercise20_31 dirName oldWord newWord
```

***20.32 (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20-16a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20-16b所示, 骑士可以移动到八个正方形的位置。编写一个程序, 在applet中显示骑士的移动, 如图20-16c所示。

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探索技巧。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最小的可访问的正方形, 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。

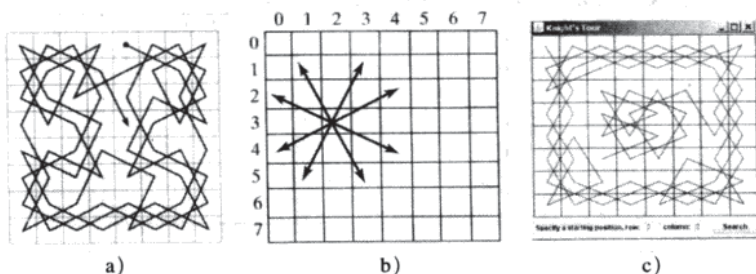


图20-16 a) 骑士遍历所有的正方形一次; b) 骑士作L形的移动; c) applet显示骑士的旅途路径

***20.33 (游戏: 骑士旅途的动画) 编写一个骑士旅途的问题的applet。applet应该允许用户将骑士放到任何一个起始正方形, 并点击Solve按钮, 用动画展示骑士沿着路径的移动, 如图20-17所示。



图20-17 骑士沿着路径遍历

**20.34 (游戏: 九宫格) 编写一个程序, 使用递归来解决九宫格问题。

**20.35 (H树分形) 一个H树分形如下定义:

- (1) 从字母H开始。H的三条线长度一样, 如图20-18a所示。
- (2) 字母H (以它的sans-serif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20-18b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, ...的H树, 如图20-18c~图20-18d所示。

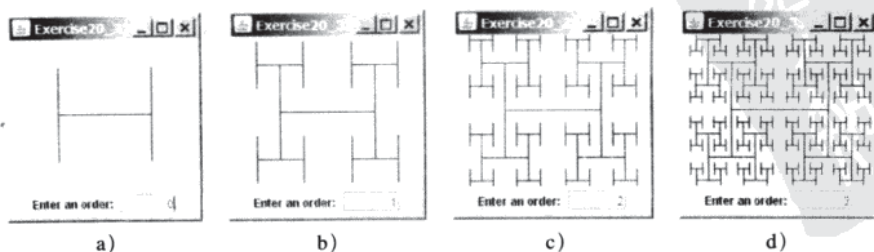


图20-18 H树是一个从H形状的三个等长的线开始的分形

在VLSI中，H树设计为一个时钟分布网络，以相同的传播延时将定时信号路由给芯片的所有部件。编写applet来绘制一个H树，如图20-18所示。

***20.36 (游戏：所有九宫格的解决方案) 改写练习题20.34，找出九宫格问题的所有可能的解决方案。

***20.37 (游戏：多种八皇后的解决方案) 编写一个applet在滚动窗格中显示八皇后谜题的所有可能的解决方案，如图20-19所示。对每个解决方案都放置一个标签来表示解决方案的个数。

提示 将所有的解决方案面板都放到一个面板中，然后将这个面板放入JScrollPane。解决方案面板类应该覆盖getPreferredSize()方法以确保正确地显示一个解决方案面板。参见程序清单15-3了解如何覆盖getPreferredSize()。

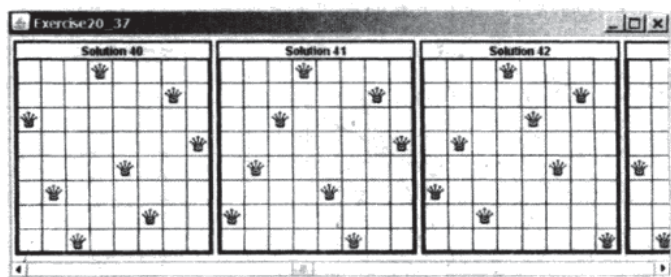


图20-19 所有的解决方案都放置在一个滚动窗格中

**20.38 (递归树) 编写一个applet来显示一个递归树，如图20-20所示。

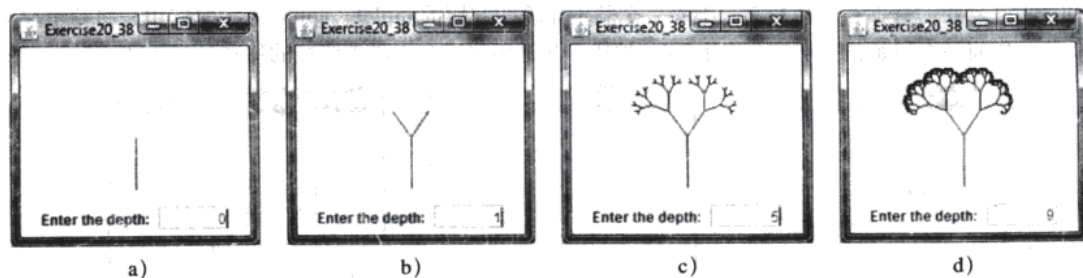


图20-20 一个带特定深度的递归树

**20.39 (拖动树) 修改练习题20.38，将树移动到鼠标指针所在的位置。

Java关键字

下面是Java语言保留专用的50个关键字：

abstract	double	int	super
assert	else	interface	switch
boolean	enum	long	synchronized
break	extends	native	this
byte	for	new	throw
case	final	package	throws
catch	finally	private	transient
char	float	protected	try
class	goto	public	void
const	if	return	volatile
continue	implements	short	while
default	import	static	
do	instanceof	strictfp [⊖]	

关键字**goto**和**const**是C++保留的关键字，目前在Java中不能使用。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。

assert是JDK 1.4增加的关键字，**enum**是JDK 1.5新加的关键字。

⊖ **strictfp**关键字是方法或类的修饰符，用于限制浮点数计算。浮点数运算可以在两种模式（精确或非精确模式）下执行。精确模式保证计算结果在所有Java虚拟机（JVM）实现上保持一致。非精确模式允许中间计算结果以不同于IEEE浮点数格式的扩展格式保存。扩展格式是机器相关的，并且能够使代码执行更快。但是，当使用非精确模式在不同JVM上执行时，可能不会得到相同的结果。默认情况下，非精确模式用于浮点数计算。如果要对方法和类使用精确模式，需要在类或者方法的声明前面加上**strictfp**关键字。精确浮点数比非精确浮点数的精度更好，但这种差异只影响部分应用程序。精确性不能继承，也就是说，出现在类或接口上的**strictfp**不会使扩展的类或接口同样精确。

附录B

Introduction to Java Programming, 8E

ASCII码字符集

表B-1和表B-2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	n1	vt	ff	cr	so	si	dle	dc1	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	n1	vt	ff	cr	so	si
1	dle	dc1	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

运算符优先级表

运算符按照优先级递减的顺序从上到下列出。同一栏中的运算符优先级相同，它们的结合方向如表C-1中所示。

表C-1 运算符优先级

运算符	名称	结合方向
()	圆括号	从左向右
()	函数调用	从左向右
[]	数组下标	从左向右
.	对象成员访问	从左向右
++	后置增量	从右向左
--	后置减量	从右向左
++	前置增量	从右向左
--	前置减量	从右向左
+	一元加	从右向左
-	一元减	从右向左
!	一元逻辑非	从右向左
(type)	一元类型转换	从右向左
new	创建对象	从右向左
*	乘法	从左向右
/	除法	从左向右
%	求余	从左向右
+	加法	从左向右
-	减法	从左向右
<<	左移	从左向右
>>	用符号位扩展的右移	从左向右
>>>	用零扩展的右移	从左向右
<	小于	从左向右
<=	小于等于	从左向右
>	大于	从左向右
>=	大于等于	从左向右
instanceof	检测对象类型	从左向右
==	相等	从左向右
!=	不等	从左向右
&	(无条件与)	从左向右
^	(异或)	从左向右
	(无条件或)	从左向右
&&	条件与	从左向右
	条件或	从左向右
?:	三元条件	从右向左
=	赋值	从右向左
+=	加法赋值	从右向左
-=	减法赋值	从右向左
*=	乘法赋值	从右向左
/=	除法赋值	从右向左
%=	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、数据和类级块），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）。可以用在类级块上的修饰符称为块修饰符（block modifier）。表D-1给出Java修饰符的一个总结。

表D-1 Java修饰符

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
public	✓	✓	✓	✓		类、构造方法、方法或数据域在任何包任何程序中都可见
private		✓	✓	✓		构造方法、方法或数据域只在所在的类中可见
protected		✓	✓	✓		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
static			✓	✓	✓	定义类方法、类数据域或静态初始化模块
final	✓		✓	✓		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
abstract	✓		✓			抽象类必须被扩展。抽象方法必须在具体的子类中实现
native			✓			用native修饰的方法表明它是用Java以外的语言实现的
synchronized			✓		✓	同一时间只有一个线程可以执行这个方法
strictfp	✓	✓				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
transient				✓		标记实例数据域，使其不进行序列化

⊖ 默认访问权限没有与之相关的修饰符。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，则向上溢出到无穷大；如果运算结果对`double`型或`float`型来说数值太小，则向下溢出到零。Java用特殊的浮点值`POSITIVE_INFINITY`、`NEGATIVE_INFINITY`和`NaN`（Not a Number，非数）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`算出零。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	$\pm \infty$	± 0.0	NaN	Finite	Finite
Finite	$\pm \infty$	± 0.0	± 0.0	x	$\pm \infty$	∞
± 0.0	± 0.0	NaN	± 0.0	NaN	± 0.0	± 0.0
$\pm \infty$	Finite	$\pm \infty$	± 0.0	NaN	$\pm \infty$	$\pm \infty$
$\pm \infty$	$\pm \infty$	NaN	± 0.0	NaN	$\pm \infty$	∞
± 0.0	$\pm \infty$	± 0.0	NaN	± 0.0	$\pm \infty$	± 0.0
NaN	Any	NaN	NaN	NaN	NaN	NaN
Any	NaN	NaN	NaN	NaN	NaN	NaN

注 如果一个操作数是NaN，则结果一定是NaN。



附录F

Introduction to Java Programming, 8E

数 系

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：

$$\boxed{7} \boxed{4} \boxed{2} \boxed{3} = 7 \times 16^3 + 4 \times 16^2 + 2 \times 16^1 + 3 \times 16^0$$

$$16^3 \ 16^2 \ 16^1 \ 16^0 = 28 \ 672 + 1024 + 32 + 3 = 29 \ 731$$

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

$$d = b_n \times 2^n + b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + \cdots + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$$

的位 $b_n, b_{n-1}, b_{n-2}, \cdots, b_2, b_1$ 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, \cdots ，

b_{n-2}, b_{n-1}, b_n 。

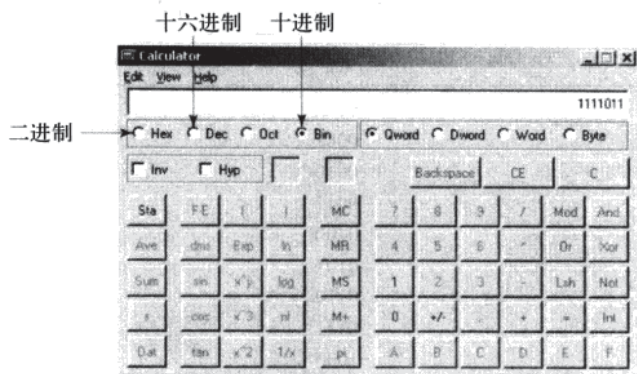
例如，十进制数123用二进制数1111011表示，所做的转换如下：

$$\begin{array}{r}
 \begin{array}{ccccccc}
 0 & 1 & 3 & 7 & 15 & 30 & 61 \\
 2 \overline{) 1} & 2 \overline{) 3} & 2 \overline{) 7} & 2 \overline{) 15} & 2 \overline{) 30} & 2 \overline{) 61} & 2 \overline{) 123} \\
 \hline
 0 & 2 & 6 & 14 & 30 & 60 & 122 \\
 \hline
 1 & 1 & 1 & 1 & 0 & 1 & 1 \\
 \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
 b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0
 \end{array}
 \end{array}$$

商

余数

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，单击“开始”按钮，依次选择“程序”，“附件”，“计算器”命令。



图F-1 使用Windows的计算器进行数制转换

3. 十六进制数与十进制数的转换

给定十六进制数 $h_n h_{n-1} h_{n-2} \cdots h_2 h_1 h_0$ ，其等价的十进制数为

$$h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + \cdots + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0$$

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

$$d = h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + \cdots + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0$$

的位 $h_n, h_{n-1}, h_{n-2}, \cdots, h_2, h_1$ 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 $h_0, h_1, \cdots, h_{n-2}, h_{n-1}$ 和 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：

$$\begin{array}{r}
 \begin{array}{cc}
 0 & 7 \\
 16 \overline{) 123} & 16 \overline{) 112} \\
 \hline
 0 & 112 \\
 \hline
 7 & 11 \\
 \downarrow & \downarrow \\
 h_1 & h_0
 \end{array}
 \end{array}$$

商

余数

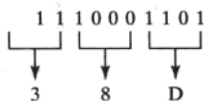
4. 二进制数与十六进制数的转换

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一位十六进制数。

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：



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
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

注意 有时也要用到八进制数, 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。

复习题

1. 将下列十进制数转换为十六进制数和二进制数。
100; 4340; 2000
2. 将下列二进制数转换为十六进制数和十进制数。
1000011001; 100000000; 100111
3. 将下列十六进制数转换为二进制数和十进制数。
FEFA9; 93; 2000