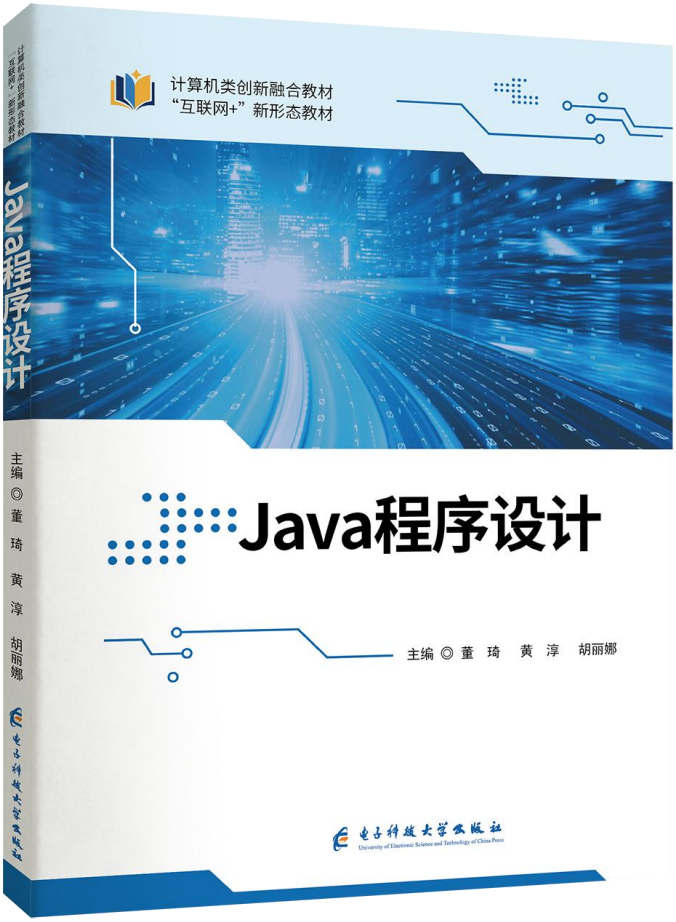
Java 程序设计



类目：计算机类
书名：Java 程序设计
主编：董 琦　黄 淳 胡丽娜
出版社：电子科大出版社
开本：大 16 开
书号：978-7-5770-0500-3
使用层次：通用
出版时间：2023/8/1
定价：59.8
印刷方式：双色
是否有资源：是

计算机类创新融合教材
“互联网+”新形态教材

计算机类创新融合教材
“互联网+”新形态教材

Java程序设计

Java程序设计

主编◎董 琦 黄 淳 胡丽娜

Java程序设计

主编◎董 琦 黄 淳 胡丽娜

电子科技大学出版社
University of Electronic Science and Technology of China Press

策划编辑：万晓桐 李燕芩
责任编辑：李雨纾
封面设计：旗语书装

扫描二维码
关注更多教材信息

ISBN 978-7-5770-0500-3
9 787577 005003 >
定价：59.80元

电子科技大学出版社

计算机类创新融合教材
“互联网 +”新形态教材

Java 程序设计

主　编◎董　琦　黄　淳　胡丽娜
副主编◎王可阳　毛　静　杨　玲
李建基　向知礼　唐　健
邵　嫚

电子科技大学出版社
University of Electronic Science and Technology of China Press
·成都·

图书在版编目（CIP）数据

Java 程序设计 / 董琦，黄淳，胡丽娜主编. — 成都：电子科技大学出版社，2024.8

ISBN 978-7-5770-0500-3

Ⅰ.①J… Ⅱ.①董… ②黄… ③胡… Ⅲ.①JAVA 语言—程序设计 Ⅳ.①TP312.8

中国国家版本馆 CIP 数据核字（2023）第 160234 号

Java 程序设计

Java CHENGXU SHEJI

董 琦 黄 淳 胡丽娜 主编

策划编辑 万晓桐 李燕芩

责任编辑 李雨纾

责任校对 熊晶晶

责任印制 梁 硕

出版发行 电子科技大学出版社

成都市一环路东一段 159 号电子信息产业大厦九楼 邮编 610051

主 页 www.uestcp.com.cn

服务电话 028-83203399

邮购电话 028-83201495

印 刷 涿州汇美亿浓印刷有限公司

成品尺寸 210mm×285mm

印 张 25

字 数 675 千字

版 次 2024 年 8 月第 1 版

印 次 2024 年 8 月第 1 次印刷

书 号 ISBN 978-7-5770-0500-3

定 价 59.80 元

PREFACE 前言

本书作为 Java 程序设计语言的入门教材，将复杂的、难以理解的思想和问题简化，让初学者能够尽快地理解和掌握程序设计的基本思想。本书对 Java 程序设计的整个过程进行梳理，按照程序设计学习过程中接触知识的先后顺序来划分章节，每个章节包含多个必要的知识点，并对每个知识点进行了深入的分析和解释；针对每个知识点精心设计了相应实例，并进行了详细的阐述。读者通过对本书的学习，能够掌握 Java 程序设计的基本语法知识，以及面向过程和面向对象的程序设计思想，具备独立开发 Java 程序的能力。

本书共分 14 章，可划分为三个部分，其具体内容如下。

第一部分：第 1～6 章。

第 1 章 Java 概述。本章介绍 Java 的发展历史与优点，开发 Java 程序必备的工具软件及其安装使用说明，最后以一个入门的 Hello World 程序对 Java 程序的基本结构进行说明。

第 2 章基本程序设计。本章介绍 Java 中的一些基本概念，包括程序的顺序结构、运算符与表达式、数据类型转换以及基于控制台的基本输入和输出。

第 3 章选择结构程序设计。本章介绍比较运算符、逻辑运算符，以及 switch 结构。

第 4 章循环结构程序设计。本章介绍三种循环控制语句 while，do…while…和 for 循环的使用方法，以及循环结构程序分析和设计的基本方法。

第 5 章数组。本章介绍数组的相关知识，包括数组、排序、查找、二维数组等相关内容。

第 6 章方法。本章介绍面向过程的程序设计方法。包括面向过程程序设计、编写方法、方法的定义、调用方法、参数值的传递、方法的嵌套调用、变量的作用域、模块化程序实现等相关内容。

第二部分：第 7～10 章。

第 7 章和第 8 章面向对象。这两章介绍面向对象程序的相关概念，对面向对象的三大特性——封装、继承和多态进行详细的介绍。通过对这两章的学习，读者可以了解到类与对象的概念、继承的概念、重写与覆盖等多态的知识，以及抽象类与接口的相关知识。

第 9 章 Java API。本章介绍 JDK 中提供的一些常用类。

第 10 章集合类。本章介绍 Java 中的集合相关的类和接口。

第三部分：第 11～14 章。

第 11 章图形用户界面（GUI）。本章介绍基于 Swing 的图形界面设计，包括 AWT 概述、AWT 事件处理、常用事件分类、布局管理器、AWT 绘图、Swing 等相关内容。

第 12 章输入输出（IO）。本章首先介绍流的概念，以及 Java 中文件的抽象；然后以文件

操作为线索，介绍基于字符流和字节流的相关基本 API；最后还介绍了其他 IO 流、File 类、RandomAccessFile、字符编码等内容。

第 13 章网络编程。本章介绍 Java 中网络编程的基本知识，主要通过 UDP 和 TCP 协议的简单操作，让读者了解网络相关 API 的使用。

第 14 章多线程。本章介绍线程的相关概念，让读者了解程序中的并发的概念，并对线程的生命周期及状态转换、线程的调度、多线程同步以及线程间的通信等内容做阐述。

上述三个部分中，第一部分是整个 Java 程序设计的基石，涵盖了 Java 中基本的语法知识和基本的程序设计思想。第二部分是现代程序开发设计的主流设计思想，向读者展示如何对复杂的现实问题进行抽象化。第三部分是高阶应用事例，每一章都涉及一个领域。

学习本书时，读者不仅需要掌握原理，更重要的是需要动手进行实践。对于书中的实例，读者都应上机进行试验，在动手练习的过程中发现问题、思考问题、解决问题，并在问题解决后进行总结，如此才能更好地掌握相关的知识和技能。

由于编者水平有限，书中难免会有不妥，欢迎读者给予宝贵的意见，我们将不胜感激。

编　者

2023 年 6 月

目　录

CONTENTS

第 1 章 Java 概述

本章重点

- Java 开发环境搭建
- Java 基本程序结构
- 使用 System. out. println 在控制台中输出字符串

21 世纪，社会进入了信息时代，随着计算机技术的飞速发展，越来越多的计算机软件出现在我们生活中的每个角落。计算机程序开发打破实验室的大门，走向了大众。在众多程序设计语言中，Java 以其优异的表现受到广大软件开发从业者的喜爱，占据了市场较大的份额。本章将介绍 Java 的发展历史及 Java 语言的优点，并一步一步地带领读者完成 Java 开发环境的搭建，编写第一个 Java 程序，解析 Java 程序的基本结构。

1.1 Java 语言发展历史

Java 的历史可以追溯到 1991 年 4 月，Sun 公司的詹姆斯·高斯林（James Gosling）领导的绿色计划（green project）开始着力发展一种分布式系统结构，使其能够在各种消费性电子产品上运行，他们使用了 C/C++/Oak 语言。由于多种原因，绿色计划逐渐陷于停滞状态。

直至 1994 年下半年，由于 Internet 的迅猛发展和环球信息网的快速增长，第一个全球信息网络浏览器 Mosaic 诞生了；此时，工业界对适合在网络异构环境下使用的语言有一种非常急迫的需求；James Gosling 决定改变绿色计划的发展方向，他们对 Oak 进行了小规模的改造，就这样，Java 在 1995 年的 3 月 23 日诞生了！Java 的诞生标志着互联网时代的开始，它能够被应用在全球信息网络的平台上编写互动性及强的 Applet 程序，而 1995 年的 Applet 无疑能带给人们无穷的视觉和脑力震荡。

1996 年 1 月 23 日，JDK 1.0 发布，Java 语言有了第一个正式版本的运行环境。JDK 1.0 提供了一个纯解释执行的 Java 虚拟机实现（Sun Classic VM）。JDK 1.0 版本的代表技术包括：Java 虚拟机、Applet、AWT 等。

1996 年 4 月，10 个最主要的操作系统供应商申明，将在其产品中嵌入 Java 技术。同年 9 月，已有大约 8.3 万个网页应用了 Java 技术来制作。在 1996 年 5 月底，Sun 公司于美国旧金山举行了首届 JavaOne 大会，从此 JavaOne 成为全世界数百万 Java 语言开发者每年一度的技术盛会。

1997 年 2 月 19 日，Sun 公司发布了 JDK 1.1，Java 技术的一些最基础的支撑点（如 JDBC 等）都是在 JDK 1.1 版本中发布的。JDK 1.1 版的技术代表有：JAR 文件格式、JDBC、JavaBeans、RMI。Java 语法也有了一定的发展，如内部类（inner class）和反射（reflection）都是在这个时候出现的。

直到 1999 年 4 月 8 日，JDK 1.1 一共发布了 1.1.0～1.1.8 九个版本。从 1.1.4 之后，每个 JDK 版本都有一个自己的名字（工程代号），分别为：JDK 1.1.4-Sparkler（宝石）、JDK 1.1.5-Pumpkin（南瓜）、JDK 1.1.6-Abigail（阿比盖尔，女子名）、JDK 1.1.7-Brutus（布鲁图，古罗马政治家和将军）和 JDK 1.1.8-Chelsea（切尔西，城市名）。

1998 年 12 月 4 日，JDK 迎来了一个里程碑式的版本 JDK 1.2，工程代号为 Playground（竞技场），Sun 在这个版本中把 Java 技术体系拆分为 3 个方向，分别是面向桌面应用开发的 J2SE（Java 2 platform，standard edition）、面向企业级开发的 J2EE（Java 2 platform，enterprise edition）和面向手机等移动终端开发的 J2ME（Java 2 platform，micro edition）。在这个版本中出现的代表性技术非常多，如 EJB、Java Plug-in、Java IDL、Swing 等，并且这个版本中 Java 虚拟机第一次内置了 JIT（just in time）编译器（JDK 1.2 中曾并存过 3 个虚拟机，即 Classic VM、HotSpot VM 和 Exact VM。其中，Exact VM 只在 Solaris 平台出现过；后面

两个虚拟机都是内置 JIT 编译器的，而之前版本所带的 Classic VM 只能以外挂的形式使用 JIT 编译器）。在语言和 API 级别上，Java 添加了 strictfp 关键字与现在 Java 编码之中极为常用的一系列 Collections 集合类。

1999 年 3 月和 7 月，分别有 JDK 1.2.1 和 JDK 1.2.2 两个小版本发布。

1999 年 4 月 27 日，HotSpot 虚拟机发布，HotSpot 最初由一家名为 Longview Technologies 的小公司开发，因为 HotSpot 的优异表现，这家公司在 1997 年被 Sun 公司收购了。HotSpot 虚拟机发布时是作为 JDK 1.2 的附加程序提供的，后来它成为了 JDK 1.3 及之后所有版本的 Sun JDK 的默认虚拟机。

2000 年 5 月 8 日，工程代号为 Kestrel（美洲红隼）的 JDK 1.3 发布，JDK 1.3 相对于 JDK 1.2 的改进主要表现在一些类库上（如数学运算和新的 Timer API 等），JNDI 服务从 JDK 1.3 开始被作为一项平台级服务提供（以前 JNDI 仅仅是一项扩展），使用 CORBA IIOP 来实现 RMI 的通信协议，等等。这个版本还对 Java 2D 做了很多改进，提供了大量新的 Java 2D API，并且新添加了 JavaSound 类库。JDK 1.3 有 1 个修正版本 JDK 1.3.1，工程代号为 Ladybird（瓢虫），于 2001 年 5 月 17 日发布。

自从 JDK 1.3 开始，Sun 维持了一个习惯：大约每隔两年发布一个 JDK 的主版本，以动物命名，期间发布的各个修正版本则以昆虫作为工程名称。

2002 年 2 月 13 日，JDK 1.4 发布，工程代号为 Merlin（灰背隼）。JDK 1.4 是 Java 真正走向成熟的一个版本，Compaq、Fujitsu、SAS、Symbian、IBM 等著名公司都有参与甚至实现自己独立的 JDK 1.4。哪怕是在 20 年后的今天，仍然有许多主流应用（Spring、Hibernate、Struts 等）能直接运行在 JDK 1.4 之上，或者继续发布能运行在 JDK 1.4 上的版本。JDK 1.4 同样发布了很多新的技术特性，如正则表达式、异常链、NIO、日志类、XML 解析器和 XSLT 转换器等。

JDK 1.4 有两个后续修正版：

2002 年 9 月 16 日发布的工程代号为 Grasshopper（蚱蜢）的 JDK 1.4.1；

2003 年 6 月 26 日发布的工程代号为 Mantis（螳螂）的 JDK 1.4.2。

2002 年前后还发生了一件与 Java 没有直接关系、但事实上对 Java 的发展进程影响很大的事件，那就是微软公司的 .NET Framework 发布了。这个无论是在技术实现上还是目标用户上都与 Java 有很多相近之处的技术平台给 Java 带来了很多讨论、比较和竞争，.NET 平台和 Java 平台之间声势浩大的孰优孰劣的论战到目前为止都在继续。

2004 年 9 月 30 日，JDK 1.5 发布，工程代号 Tiger（老虎）。从 JDK 1.2 以来，Java 在语法层面上的变化一直很小，而 JDK 1.5 在 Java 语法易用性上做出了非常大的改进。例如，自动装箱、泛型、动态注解、枚举、可变长参数、遍历循环（foreach 循环）等语法特性都是在 JDK 1.5 中加入的。在虚拟机和 API 层面上，这个版本改进了 Java 的内存模型（Java memory model，JMM），提供了 Java.util.concurrent 并发包等。另外，JDK 1.5 是官方声明可以支持 Windows 9x 平台的最后一个 JDK 版本。

2006 年 12 月 11 日，JDK 1.6 发布，工程代号 Mustang（野马）。在这个版本中，Sun 终结了从 JDK 1.2 开始已经有 8 年历史的 J2EE、J2SE、J2ME 的命名方式，启用Java SE 6、Java EE 6、Java ME 6 的命名方式。JDK 1.6 的改进包括：提供动态语言支持（通过内置 mozilla Java rhino 引擎实现）、提供编译 API 和微型 HTTP 服务器 API 等。同时，这个版本对 Java 虚拟机内部做了大量改进，包括锁与同步、垃圾收集、类加载等方面的算法都有相当多的改动。

在 2006 年 11 月 13 日的 JavaOne 大会上，Sun 公司宣布最终会将 Java 开源，并在随后的一年多时间内，陆续将 JDK 的各个部分在 GPL v2（GNU general public license v2）协议下公开了源代码，并建立了 OpenJDK 组织对这些源代码进行独立管理。除了极少量的产权源代码（encumbered code，这部分源代码大多是 Sun 本身也无权限进行开源处理的）外，OpenJDK 几乎包括了 Sun JDK 的全部代码，OpenJDK 的质量主管曾经表示，在 JDK 1.7 中，Sun JDK 和 OpenJDK 除了代码文件头的版权注释之外，代码基本完全一样，所以 OpenJDK 7 与 Sun JDK 1.7 本质上就是同一套代码库开发的产品。

JDK 1.6 发布以后，由于源代码复杂性的增加、JDK 开源、开发 JavaFX、经济危机及 Sun 收购案等原因，Sun 在 JDK 发展以外的事情上耗费了很多资源，JDK 的更新没有再维持两年发布一个主版本的发展速度。

2009 年 2 月 19 日，工程代号为 Dolphin（海豚）的 JDK 1.7 完成了其第一个里程碑版本。根据 JDK 1.7 的功能规划，一共设置了 10 个里程碑。最后一个里程碑版本原计划于 2010 年 9 月 9 日结束，但由于各种原因，JDK 1.7 最终无法按计划完成。

从 JDK 1.7 最开始的功能规划来看，它本应是一个包含许多重要改进的 JDK 版本，其中的 Lambda 项目（Lambda 表达式、函数式编程）、Jigsaw 项目（虚拟机模块化支持）、动态语言支持、GarbageFirst 收集器和 Coin 项目（语言细节进化）等子项目对于 Java 业界都会产生深远的影响。在 JDK 1.7 开发期间，Sun 公司由于相继在技术竞争和商业竞争中都陷入泥潭，公司的股票市值跌至仅有高峰时期的 3%，已无力推动 JDK 1.7 的研发工作按正常计划进行。为了尽快结束 JDK 1.7 长期“跳票”的问题，Oracle 公司收购 Sun 公司后不久便宣布将实行“B 计划”，大幅裁剪了 JDK 1.7 预定目标，以便保证 JDK 1.7 的正式版能够于 2011 年 7 月 28 日准时发布。“B 计划”把不能按时完成的 Lambda 项目、Jigsaw 项目和 Coin 项目的部分改进延迟到 JDK 1.8 之中。最终，JDK 1.7 的主要改进包括：提供新的 G1 收集器（G1 在发布时依然处于 Experimental 状态，直至 2012 年 4 月的 Update 4 中才正式“转正”）、加强对非 Java 语言的调用支持（JSR-292，这项特性到目前为止依然没有完全实现定型）、升级类加载架构等。

从 Java SE 7 Update 4 起，Oracle 开始支持 Mac OS X 操作系统，并在 Update 6 中达到完全支持的程度，同时，在 Update 6 中还对 ARM 指令集架构提供了支持。至此，官方提供的 JDK 可以运行于 Windows（不含 Windows 9x）、Linux、Solaris 和 Mac OS 平台上，支持 ARM、x86、x64 和 Sparc 指令集架构类型。

2009 年 4 月 20 日，Oracle 公司宣布正式以 74 亿美元的价格收购 Sun 公司，Java 商标从此正式归 Oracle 所有（Java 语言本身并不属于哪间公司所有，它由 JCP 组织进行管理，JCP 主要由 Sun 公司或者说 Oracle 公司所领导）。

1.2 Java 语言的优点

Java 语言有以下特点 ：简单、面向对象、分布式、解释执行、健壮、安全、体系结构中立、可移植、高性能、多线程以及动态性。

简单：Java 语言的语法与 C 语言和 C++语言很接近，这使得大多数程序员很容易学习和使用 Java。另外，Java 丢弃了 C++中很少使用的、很难理解的、令人迷惑的那些特性，如操作符重载、多继承、自动的强制类型转换。特别地，Java 语言不使用指针，并提供了自

动的废料收集，这使得程序员不必为内存管理而担忧。

面向对象：Java 语言提供类、接口和继承等原语。为了简单起见，只支持类之间的单继承，但支持接口之间的多继承，并支持类与接口之间的实现机制（关键字为 implements）。Java 语言全面支持动态绑定，而 C＋＋语言只对虚函数使用动态绑定。总之，Java 语言是一个纯面向对象的程序设计语言。

分布式：Java 语言支持 Internet 应用的开发，在基本的 Java 应用编程接口中有一个网络应用编程接口（Java net），它提供了用于网络应用编程的类库，包 括 URL、URLConnection、Socket、ServerSocket 等。Java 的远程方法激活（remote method invocation，RMI）机制也是开发分布式应用的重要手段。

解释执行：Java 程序在 Java 平台上被编译为字节码格式，然后可以在实现这个 Java 平台的任何系统中运行。在运行时，Java 平台中的 Java 解释器对这些字节码进行解释执行，执行过程中需要的类在连接阶段被载入到运行环境中。

健壮：Java 的强类型机制、异常处理、废料的自动收集等是 Java 程序健壮性的重要保证。对指针的丢弃是 Java 的明智选择。Java 的安全检查机制使得 Java 更具健壮性。

安全：Java 通常被用在网络环境中，为此，Java 提供了一个安全机制以防恶意代码的攻击。除了 Java 语言具有的许多安全特性以外，Java 对通过网络下载的类具有一个安全防范机制（类 ClassLoader），如分配不同的名字空间以防替代本地的同名类、字节代码检查，并提供安全管理机制（类 SecurityManager）让 Java 应用设置安全哨兵。

体系结构中立：Java 程序（后缀为 Java 的文件）在 Java 平台上被编译为体系结构中立的字节码格式（后缀为 class 的文件），然后可以在实现这个 Java 平台的任何系统中运行。这种途径适合于异构的网络环境和软件的分发。

可移植：这种可移植来源于体系结构中立性。另外，Java 还严格规定了各个基本数据类型的长度。Java 系统本身也具有很强的可移植性，Java 编译器是用 Java 实现的，Java 的运行环境是用 ANSI C 实现的。

高性能：与那些解释型的高级脚本语言相比，Java 的确是高性能的。事实上，Java 的运行速度随着 JIT 编译器技术的发展越来越接近于 C＋＋。

多线程：在系统上运行的程序都是一个进程，而一个进程会存在一到多个线程。线程之间是独立运行的，有自己专用的运行栈，线程之间存在共享资源，如内存、数据、文件。线程是一组指令的集合。Java 中一个程序是可以允许多个线程同时进行，用于支持事物并发和多任务处理。Java 除了内置的多线程技术以外，还定义了一些类和方法用来建立和管理用户定义的线程。

动态性：Java 语言的设计目标之一是适应动态变化的环境。Java 程序需要的类能够动态地或通过网络被载入运行环境。这也有利于软件的升级。另外，Java 中的类有一个运行时刻的表示，能进行运行时刻的类型检查。

1.3 Java 开发环境搭建

要想在自己的计算机中开发 Java 程序，需要在计算机上安装和配置 Java 程序编译和运行的相关环境。在搭建开发环境之前，需要了解一些基本概念。

Java 虚拟机（Java virtual machine，JVM）：Java 虚拟机是在操作系统之上建立的一个抽象层，封装和覆盖了操作系统的实现和运行的差异，使 Java 编写的程序可以在任何安装了 Java 虚拟机的实体机上运行，它是 JRE 的一部分。

Java 运行环境（Java runtime environment，JRE）：Java 运行环境包含了 JVM 的标准实现和 Java 核心类库。普通用户只需要安装 JRE 就可以执行 Java 程序。

Java 开发工具包（Java development kit，JDK）：Java 开发工具包包含了一组用于开发 Java 应用程序的工具，程序开发人员必须安装 JDK 才能编译和调试 Java 程序。JDK 安装包中包含了完整的 JRE。

JDK 一共包含三个版本，分别为 Java SE，Java EE 和 Java ME。

Java 标准版（Java standard edition，Java SE）：用于开发和部署桌面、服务器以及嵌入设备和实时环境中的 Java 应用程序。Java SE 包括用于开发 Java Web 服务的类库，同时，Java SE 为 Java EE 提供了基础，本书使用此版本进行 Java 程序开发。

Java 企业版（Java enterprise edition，Java EE）：在标准版的基础上还提供了对 EJB（enterprise Java beans）、Java Servlet API、JSP（Java server pages）以及 XML 技术的全面支持。其最终目的就是成为一个能够使企业开发者大幅缩短软件投放市场时间的体系结构。

Java ME（Java Micro Edition）：Java 移动版。它为移动设备（包括消费类产品、嵌入式设备、高级移动设备等）提供了基于 Java 环境的开发与应用平台。

了解了基本概念后，我们将在计算机中安装 JDK，并进行相应配置。先打开 Oracle 官方网站 www.oracle.com/cn 首页，如图 1-1 所示。

图 1-1　Oracle 首页

单击“下载”按钮，进入下载页面，如图 1-2 所示。

图 1-2　下载页面

单击“Java 下载”，进入版本选择页面，如图 1-3 所示。

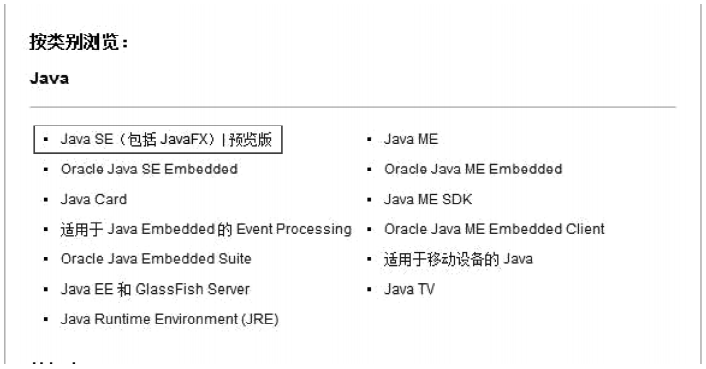


图 1-3　JDK 版本选择页面

单击“Java SE（包括 JavaFX）| 预览版”按钮，进入 Java SE 版本 JDK 下载页面，如图 1-4 所示。

图 1-4　Java SE 下载页面

单击方框处的图标按钮进入开发平台选择页面，如图 1-5 所示。

图 1-5　开发平台选择页面

在开发平台列表上方的方框处选择“接受许可协议”，并在列表中根据自己计算机操作系统选择相应版本下载。下方方框中的两个版本分别为 32 位 Windows 版本和 64 位 Windows 版本。

下载完毕后，将会在下载目录中看到 JDK 安装程序图标，如图 1-6 所示。

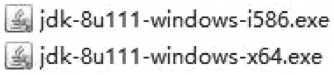


图 1-6　JDK 安装程序图标

双击安装程序图标，启动 JDK 安装程序，进入安装程序初始界面，如图 1-7 所示。

单击“下一步”，进入安装路径选择和功能选择界面，如图 1-8 所示。

图 1-7　JDK 安装程序初始界面

图 1-8　安装路径和功能选择界面

其中，开发工具包括了编译、运行等开发 Java 程序的一套组件，源代码为 Java 核心类库的实现代码，公共 JRE 是 Java 程序运行的必要环境。

单击“下一步”，进入安装过程，如图 1-9 所示。

在 JDK 安装过程中，会弹出 JRE 安装提示框，如图 1-10 所示。

图 1-9　JDK 安装过程界面

图 1-10　JRE 安装提示界面

在此界面中可以设置 JRE 的安装路径。注意，请不要选择和 JDK 完全一致的路径，否则安装程序将覆盖 JDK 安装程序部署的文件，造成开发工具无法使用的后果。

单击“下一步”开始安装 JRE，直至结束，如图 1-11 所示。

图 1-11　JDK 安装结束界面

JDK 安装完毕后，还需要进行一些配置才能开始 Java 程序的开发。打开控制面板，如图 1-12 所示。

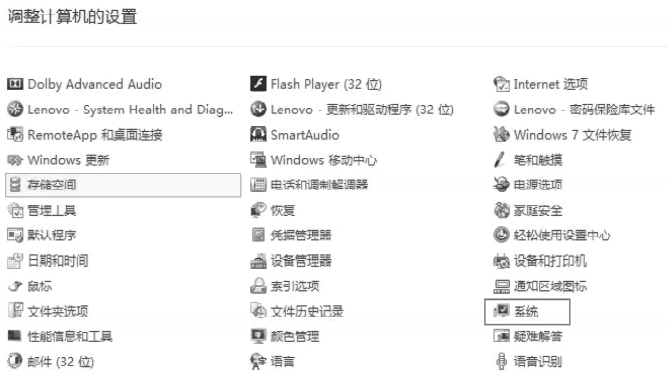


图 1-12　控制面板选项页面

选择“系统”→“高级”，弹出“系统属性”面板，如图 1-13 所示。

选择环境变量属性，弹出“环境变量”面板，如图 1-14 所示。

图 1-13　“系统属性”面板

图 1-14　“环境变量”面板

在系统变量中，选择“Path”，点击“编辑”按钮，在弹出的系统变量编辑窗口中，添加 JDK 安装位置下 bin 文件夹的路径变量值，如图 1-15 所示。

图 1-15　环境变量添加

在键盘上按下 Win＋R 组合键，输入 cmd，打开控制台窗口，在控制台中输入 java-version 指令，如出现版本号提示，则表明环境变量配置成功，如图 1-16 所示。

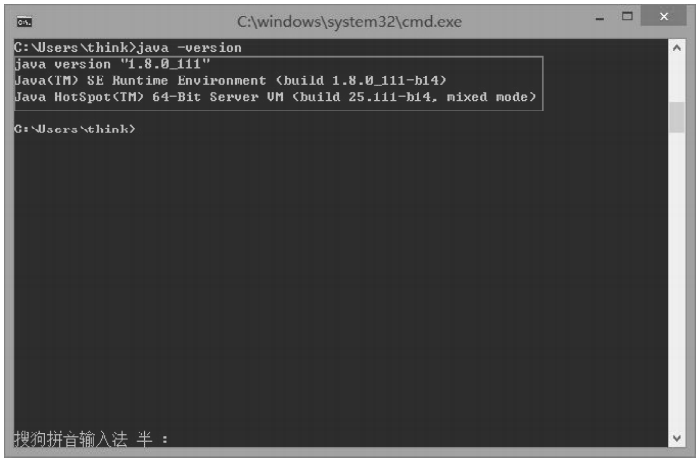


图 1-16　Java 版本提示信息

1.4　Java 程序设计入门

本节将实现一个简单的 Java 程序，该程序在控制台中显示一串字符“Hello Java!”。通过对该程序的分析来了解 Java 程序中的一些基本概念。

在正式编写程序之前，需要了解 Java 程序开发的一般流程。Java 是一种编译型语言，需要先完成程序源代码（Java 语言）的编写，然后通过编译程序将源代码文件编译成字节码文件（二进制），在编译过程中如果出现错误（编译错误，即语法错误），则需要对源代码文件进行修改后重新进行编译，这个过程可能反复出现，直至正确生成字节码文件。字节码文件将在 Java 虚拟机上运行，如果运行错误（运行时错误）或无法得到正确结果（逻辑错误），则必须修改源代码，并重新进行编译，再运行程序，如图 1-17 所示。

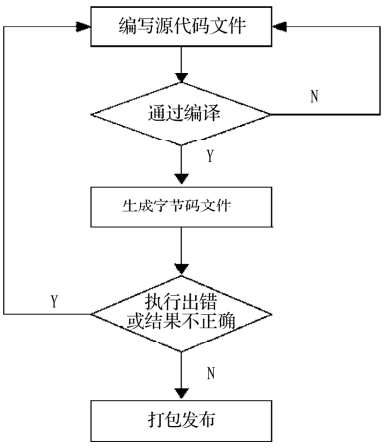


图 1-17　Java 程序一般开发流程

可以使用任意文本编辑器或支持 Java 的集成开发环境来创建和编写 Java 程序源代码文件，本节将演示使用记事本来编写程序源代码文件，并在控制台窗口中对其进行编译、执行的过程。

新建一个记事本文件，将文件名修改为 Hello.Java，Java 源代码文件的后缀名为 .Java。打开文件，在其中录入源代码（书中行号是为了引用方便，并不是源代码中的一部分，请在录入源代码时不要录入行号），如程序清单 1-1 所示。

程序清单 1-1　Hello.Java

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

```
6
7    }
```

保存文件，按下 Win＋R 组合键，输入 cmd，打开控制台窗口，将路径切换到源代码文件所在位置（本书源码在作者计算机中均放在 F:\javaproject 下，后续内容中不再说明），如图 1-18 所示。

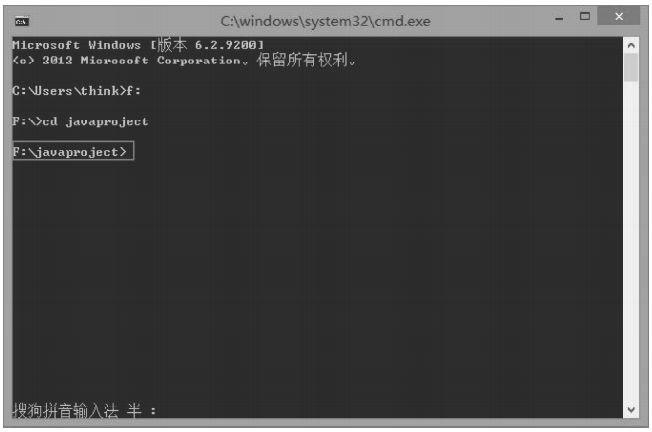


图 1-18　切换路径至源代码所在位置

在控制台中输入 Javac Hello.java，如图 1-19 所示。Javac 指令用于对源代码文件进行编译，如编译过程中未出现错误则生成后缀名为 .class 的字节码文件，如图 1-20 所示。

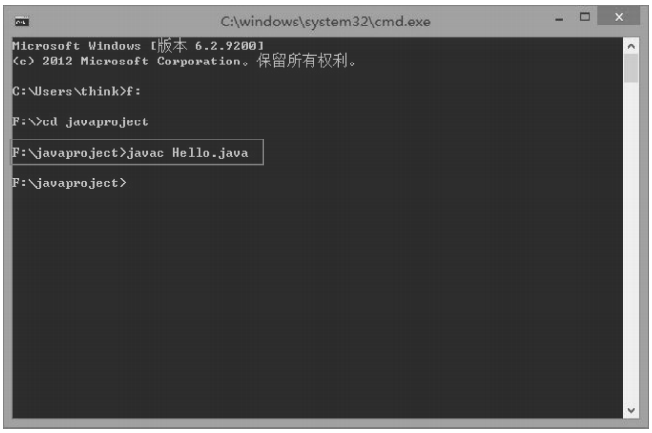


图 1-19　使用 Javac 编译 Hello.java

图 1-20　源代码文件与生成的字节码文件

最后，在控制台中输入 java Hello，Java 指令用于执行 Java 程序，Hello 程序执行结果如图 1-21 所示。

程序清单 1-1 中第 1 行代码 public class Hello 定义了一个类，其中 public 和 class 均为关键字，public 表示该类是公共访问的，class 关键字用于定义一个类，Hello 是自定义的类名。关键字是指在 Java 语言中具有特殊含义的单词，程序员不能使用它们来另作他用。类名 Hello 后的花括号“｛”与第 7 行的花括号“｝”匹配，它们之间的内容定义了该类的实现。类在 Java 中是一个重要的概念，也是面向对象程序设计的基础，我们将在第 7、8 章中对其进行详

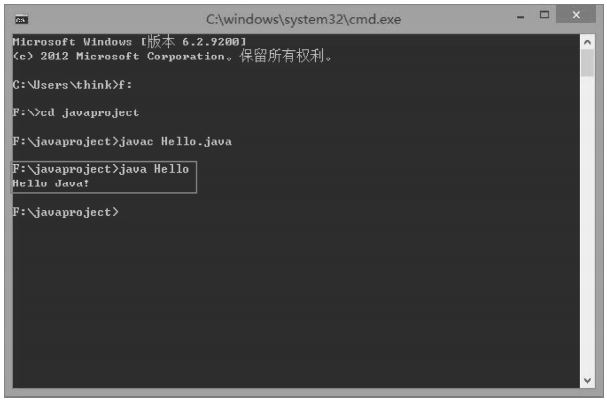


图 1-21　执行 Hello 程序

细介绍。目前大家只需要知道，类是 Java 程序构成的基本单位，Java 程序可以由一个或者多个类组成。第 3 行代码 public static void main（String[] args）定义了程序的主方法。在可执行的 Java 程序中必须定义主方法，且主方法必须由关键字 public、static、void 修饰。计算机如何知道我们的程序从哪里开始执行呢？在 Java 中，程序就是从主方法开始执行的，它是程序执行的起点。主方法必须定义在公共类（即有 public 修饰的类）中。main 方法后的花括号“{”与第 5 行的花括号“}”匹配，它们之间的内容是 main 方法的实现。第 4 行代码 System. out. println("Hello Java!")；是一条可执行语句，System 表示了一个系统类，它使用了内置对象 out 的 println 方法在控制台中输出了一串字符“Hello Java!”。Java 实现了许多类来帮助我们完成各种任务，这些类我们把它们称为系统类。在 Java 程序中每条语句都是以分号结束的。

现在我们修改并保存程序清单 1-1 中的代码，如程序清单 1-2 所示。

程序清单 1-2　Hello. Java

```
1    public class HelloJava {
2
3        public static void main(String[] args){
4        System. out. println("Hello Java!");
5        }
6
7    }
```

重新编译源代码文件，运行结果如图 1-22 所示。

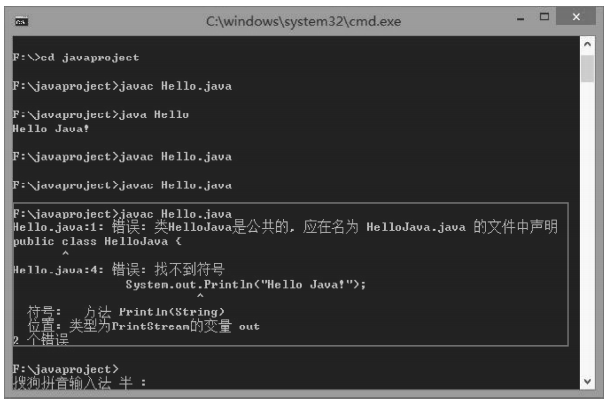


图 1-22　有错误的源代码编译结果

我们看到，在控制台中出现了编译错误的提示。这是因为我们在程序清单 1—2 中修改了源代码。造成错误的原因有两个：一是我们将 Hello 程序中的类名改为了 HelloJava，在 Java 源代码文件中最多只能有一个公共类，如果源代码文件中有公共类，则该源代码文件的名称必须与该类相同；二是我们将 System 类中 out 对象的 println 方法的第一个字母 p 修改为了大写的 P，Java 语言要区分大小写，此处我们将 print 方法的首字改为大写后，编译器在系统类中找不到 Print 方法的定义，不知道这个方法是做什么用，所以无法通过编译。另外，在编写 Java 源代码文件时，对于其中的符号，如括号、引号等均需要使用半角模式进行输入，否则在编译时会出现错误。

1.5 本章小结

本章介绍了 Java 语言的相关基础知识。先介绍了 Java 发展的历史和其优点，接着介绍了 Java 开发环境的搭建方法，最后通过 Hello 程序介绍了 Java 程序的组成结构。

第 2 章 基本程序设计

本章重点

- Java 基本数据类型
- 算数运算符与赋值运算符
- 数据类型转换
- 利用 Scanner 从控制台接收数据

程序设计的目的是使用计算机帮助人们进行科学计算、现实模拟，解决生活中的实际问题。本章将开始学习如何使用 Java 来解决实际问题，并学习使用基本数据结构、运算符、表达式、基于控制台的 IO 等知识来进行基本的程序设计。

2.1 程序的顺序结构

计算机程序分为三种执行结构：顺序、分支、循环。如果将分支结构和循环结构看作一个整体（执行块），从宏观上看，所有的程序都可以看作是顺序结构的。Java 程序从 main 方法开始，依次执行预先编写好的代码块，直至 main 方法执行结束。

计算机程序通过各种渠道，如控制台、文件、数据库、网络等来获取数据，然后根据实际问题的需求对数据进行处理，最后根据用户的需求将处理之后的数据（信息）进行存储，或通过某种可视化手段，将其呈现在用户眼前。因此，我们可以认为计算机程序是这样一种顺序结构：

（1）数据输入，通过某种渠道获取数据；

（2）数据处理，按照特定的逻辑对数据进行处理；

（3）数据呈现（输出），以某种方式存储或显示数据。

其执行结构如图 2-1 所示。

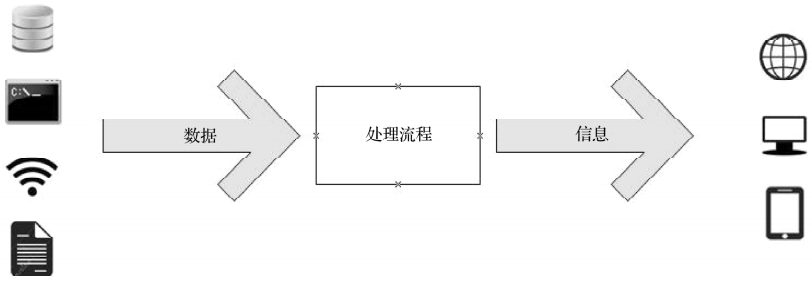


图 2-1　计算机程序宏观流程

2.2 基本数据类型

在进行计算机程序设计时，要面对的第一个问题是如何在计算机中表示实际问题中的数据。Java 提供了一些基本数据类型，我们可以使用它们或它们的组合来表示任何想要表示的事物。这些类型包括整型、浮点型、字符型和布尔类型。为了更好地理解数据类型，我们先要简单地了解内存地址的概念。

2.2.1 计算机内存地址

计算机程序在运行时，将数据保存在内存中，内存就是数据的容器，就像我们在生活中需要使用容器来存放各种物质一样。我们可以思考这样一个场景：假设我们要在图书馆中存放和管理我们的图书，最简单的方式就是将图书直接堆放到图书馆中。但是这样的方法会带来一个严重的问题，当我们想要找到某一本特定的图书，如《Java 程序设计》时，需要在杂

乱无章的书堆中去翻找，会耗费大量的时间和精力。为了解决这个问题，我们对图书馆进行规划，在图书馆中添置一些书架，并且对图书进行分类，把图书分为文学、历史、计算机、小说等类型，并对每类图书进行编号，再依照编号顺序把它们放入书架中。当想要找到《Java 程序设计》这本书时，只需要在计算机类书架中根据该书的编号进行查找，这大大提高了工作的效率。计算机内存也采用了这样的设计方法，我们将内存进行编址，以方便对其进行管理。在计算机中，以字节为单位来对内存进行编号，第一个字节编号为 0，第二个字节编号为 1，以此类推，编号以 16 进制表示，如图 2-2所示。

图 2-2　内存编址示意图

2.2.2　常量、变量与基本数据类型

现在让我们来考虑一个实际问题：假设需要编写一个程序来帮助我们计算矩形的周长和面积。要想让计算机来帮助我们进行数据处理，就必须先告知计算机要处理的数据是什么，即以合适的方式将数据放入计算机的内存中。

数据在计算机的内存中有两种存在形式：常量与变量。顾名思义，常量是指在程序运行过程中为固定值的数据，其值不能改变；而变量在程序运行的过程中其值可以通过赋值改变。在程序中，常量可以直接使用，而变量需要定义并赋值后才能使用。当我们在程序中需要变量时，可以使用如下形式来定义一个变量：

数据类型　变量名

通常使用变量来表示待处理的数据。如前所述，求矩形周长和面积的问题，要解决这个问题，就需要告知计算机待计算矩形的长度和宽度，因此可以定义两个变量 width 和 height 来存储矩形的长度和宽度（假设它们都是整数），如：

```
int width;
int height;
```

其中，int 是 Java 中的关键字。关键字是指在 Java 中具有特定意义的单词，如第 1 章中的 class，它表示要定义一个类。int 在这里表示其后紧跟的变量名表示的是标准整型。数据类型是一个非常重要的概念，它指定了其所代表的类型能够表示的数据的类型和范围。在变量中，存储数据就好像现实生活中使用容器来存储物品，不同类型的物品可以使用不同的容器来存储。例如，使用可以使用密封的容器来存放液体，并且不同容积的容器可以容纳液体的多少也不相同。Java 中的基本数据类型有 4 种，分别是整型、浮点型（实型）、字符型以及布尔型，其能够表示的数据的类型和其使用的内存单元的数量见表 2-1 所列。

表 2-1　Java 基本数据类型

数据类型	关键字	占用内存单元数量	描述
整型	byte	1 字节	用于存储整数，如 100，－25，0xffff 等
	short	2 字节	
	int	4 字节	
	long	8 字节	
浮点型	float	4 字节	用于存储浮点数（即小数），如 3.15，8.0 等
	double	8 字节	
字符型	char	2 字节	用于存储单个字符，如 'a'，'\n' 等
布尔型	boolean	1 位	只能取 true 或 false

数据类型所占用的内存单元数量决定了该类型能够表示的数据的个数。我们都知道，在计算机中一个位（bit）可以表示一个 0 或者一个 1，一个字节（byte）有 8 位，因此一个字节能够表示数据的个数为 2^8 个数据信息。以 int 为例，Java 中的标准整型占用 4 个字节的内存单元（32 位），因此它最多可以表示 2^{32} 个整数，从 -2^{31}～$2^{31}-1$，即－2147483648～2147483647。

接下来我们用一个简单的示例，来演示各种基本数据类型的变量的定义方式，并且对其赋值后在控制台中进行显示，源代码如代码清单 2-1 所示。

代码清单 2-1　BasicDataTypeEx.Java

```
1    public class BasicDataTypeEx{
2
3        public static void main(String[] args){
4            byte    num1=127;
5            short   num2=-32768;
6            int     num3=65535,num4=100;
7            long    num5=10000000000000L;
8            float   num6=32.5f;
9            double  num7=3.1415926;
10           boolean flag=true;
11           char    ch='a';
12           num3=65535;
13           System.out.println("字节类型:"+num1);
14           System.out.println("短整型:"+num2);
15           System.out.println("标准整型:"+num3+","+num4);
16           System.out.println("长整型:"+num5);
17           System.out.println("单精度实型:"+num6);
18           System.out.println("双精度实型:"+num7);
19           System.out.println("布尔类型:"+flag);
20           System.out.println("字符类型:"+ch);
21       }
```

```
22      }
```

运行结果如图 2-3 所示。

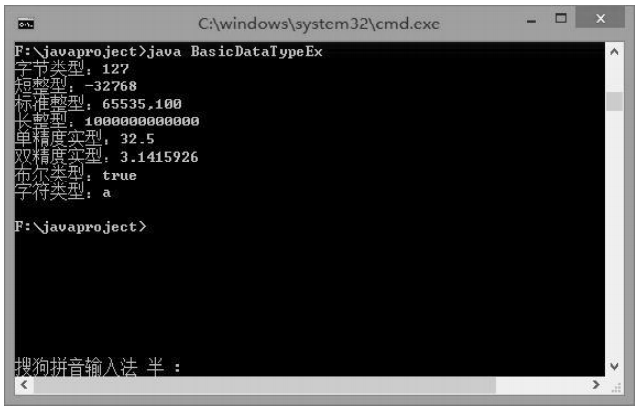


图 2-3　基本数据类型变量的定义

代码清单 2-1 中第 4 行至第 12 行，依次定义了各种基本数据类型的变量并使用相应常量为其赋值。变量 num1 为字节型（byte）变量，其值为 127；变量 num2 为短整型（short）变量，其值为－32768；变量 num3 和 num4 均为标准整型（int）变量，其值分别为 65535 和 100；num5 为长整型（long）变量，其值为 1000000000000L；num6 为单精度实型（float）变量，其值为 32.5f；num7 为双精度实型（double）变量，其值为 3.1415926；flag 为布尔类型（boolean）变量，其值为 true；ch 为字符类型（char）变量，其值为 'a'。第 13 行至第 20 行，使用了 System 类的 out 对象的 println 方法在控制台中输出了这些变量的值。

num1～num7、flag、ch 均为变量名，在 Java 中我们可以使用合法的标识符来作为变量的名称。合法标识符须符合以下规则：

（1）标识符可以由数字、字符、下划线（_）和美元符号（$）组成，首字必须是字母、下划线或美元符号；

（2）Java 关键字不能作为标识符使用；

（3）标识符中不能包含空格；

（4）标识符不能包含除下划线和美元符号之外的其他特殊符号。

由于 Java 使用 Unicode 字符集，因此字母并不仅仅指 26 个英文字母，也包含中文字符、拉丁字母等，但是并不建议使用它们来组建标示符。

使用标识符来命名变量（方法或类）时，应尽量使用有意义的单词或单词缩写，避免使用无意义的单个字符诸如 a、b、c、d 等来作为变量名。当使用多个单词来作为标识符时，首个单词应全部小写，其后的单词首字大写，其他字符均小写。这不是 Java 的语法规则，而是在进行程序设计时，应养成的一种良好的编程习惯。例如，有下面两段代码段，均为求矩形面积。

代码段 1

```
int a,b,c;
a=5;
b=10;
c=a*b;
```

代码段 2

```
int width,height,area;
width=5;
height=10;
area=width * height;
```

代码段 2 的可读性明显高于代码段 1，代码段 1 在没有说明的情况下，只能看出是在做乘法运算，而代码段 2 能够很容易地看出是在求矩形的面积。

在程序清单 2-1 中，除了各种基本数据类型的变量外，我们还使用了各种基本数据类型的常量来为这些变量赋值，如 127、1000000000000L、32.5f、3.1415926、true、'a'等。

整型常量，就是数学中的整数。在 Java 中允许以三种形式来提供整型常量：十进制、八进制与十六进制。在程序清单 2-1 中，所有整型常量均以十进制形式进行表示，如 127。如果需要以八进制表示一个整型常量，则要以数字 0 来作为该常量开头，如 0177，它表示十进制的 127。如果需要以十六进制表示一个常量，则要以 0x 作为常量的开头，如 0x7F，它同样表示十进制 127。下面在代码清单 2-2 中来对此进行演示。

代码清单 2-2　IntegerType.Java

```
1    public class IntegerType{
2        public static void main(String[] args)
3          {
4              int decimalNum=127;
5              int octalNum=0177;
6              int hexNum=0x7F;
7              System.out.println(decimalNum);
8              System.out.println(octalNum);
9              System.out.println(hexNum);
10         }
11   }
```

运行结果如图 2-4 所示。

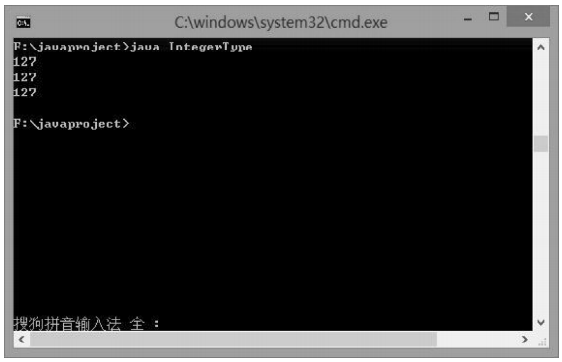


图 2-4　整型常量的表示

在代码清单 2-2 中，我们对三个变量 decimalNum、octalNum 和 hexNum 分别以十进制、八进制和十六进制的形式进行了赋值，最后输出结果均为 127。

另外，我们在程序清单 2-1 中看到，在向长整型变量 num5 赋值时，使用的常量是 1000000000000L。在 Java 中，长整型的常量在尾部必须添加小写字母 l 或大写字母 L，以表示该常量是一个长整型变量。

实型常量，就是数学中的小数。Java 中实型常量的默认类型为双精度实型，其有效位数为 15～16 位。如果希望使用单精度实型的常量，需要在尾部添加小写字母 f 或大写字母 F 来表示该常量是一个单精度实型的常量，单精度实型的有效位数为 6～7 位。实型数据在做运算的过程中可能出现误差，因此适合于做科学计算或工程计算，而不适合于商业计算。我们在代码清单 2-3 中来进行演示。

代码清单 2-3　RealType.Java

```
1  public class RealType{
2      public static void main(String[] args){
3          float op1=2.45f;
4          float op2=4.554271f;
5          float result=op1+op2;
6          System.out.println(op1+"+"+op2+"="+result);
7          double num1=2.45;
8          double num2=3.001;
9          double sum=num1+num2;
10         System.out.println(num1+"+"+num2+"="+sum);
11     }
12 }
```

运行结果如图 2-5 所示。

图 2-5　浮点运算的误差

可以看到，程序运行的结果与预期的结果有差异。造成这种差异的原因是浮点数在计算机中进行表示时采用了近似值，在计算机中 1.0 的表示可能是 0.99999999999999，也可能是 1.000000001，因此在实际运算中可能出现误差。要解决这个问题，可以使用 BigDecimal 类来处理浮点数运算，具体操作过程将在后面的章节中给大家进行讲解。

字符常量是用单引号括起来的单个字符，如 'a'' 谢 ''Φ' 等，Java 使用的是 Unicode 码，因此在 Java 程序中，字符并不仅仅只是 26 个英文符号，也可以是中文或拉丁字母等。有时候在程序中需要输出一些控制字符，或在 Java 中有特殊意义的字符，如单引号，这时可以使用转义字符。

转义字符是一种特殊的字符，以反斜线开始，后面跟上一个对编译器而言具有特殊意义的字符。常用转义字符见表 2-2 所列。

表 2-2　常用转义字符

转义字符	说明	转义字符	说明
\ a	响铃	\ b	退格
\ n	换行	\ r	回车
\ t	制表符	\ \	反斜线
\ '	单引号	\ "	双引号
\ ddd	以三位八进制表示对应 Unicode 码字符	\ xhh	以二位十六进制表示对应 Unicode 码字符

下面通过一个示例来演示字符常量及转义字符的使用，源码如代码清单 2-4 所示。

代码清单 2-4　CharacterType. Java

```
1  public class CharacterType{
2      public static void main(String[] args){
3          char ch='a';
4          char chinese='中';
5          char diamond='\004';
6          char blank='\t';
7          System. out. println(""+ch+blank+chinese+blank+diamond);
8      }
9  }
```

代码清单 2-4 中第 3 行定义了一个字符变量 ch 并将一个英文字符 'a' 赋值给它，第 4 行定义了一个字符变量 chinese 并将一个中文字符 '中' 赋值给它，第 5 行定义了一个字符变量 diamond 并使用转义字符的形式，将 Unicode 码为 004 的字符赋值给它，第 6 行定了一个字符变量 blank 并将制表符赋值给它，最后将它们连接在一起，并输出到控制台。运行结果如图 2-6 所示。

图 2-6　字符变量示例

布尔类型常量最简单，它只有两个值：一个是 true，表示真值；一个是 false，表示假值。注意这两个值均为小写形式。

2.3 运算符与表达式

通过前面的学习，我们知道如何在程序中表示数据，接下来需要对数据进行处理。在计算机中处理数据，最终都是通过各种运算来实现，这些运算主要包括算术运算和逻辑运算。

Java 提供了多种运算符来帮助我们进行数据的处理，包括算术运算符、赋值运算符、比较运算符、逻辑运算符等。

2.3.1 算术运算符

Java 中的算术运算符包括＋（加）、－（减）、＊（乘）、/（除）、%（求余）、＋＋（自增）、－－（自减），共七种。

代码清单 2-5 中的代码演示了这些算术运算符的使用方式。

代码清单 2-5　ArithmeticOperation.Java

```
1  public class ArithmeticOperation{
2      public static void main(String[] args){
3          int num1=12;
4          int num2=5;
5          double op1=4.8;
6          double op2=3.4;
7          System.out.println("整型变量"+num1+"与"+num2+"的算术运算:");
8          System.out.println("num1+num2="+(num1+num2));
9          System.out.println("num1-num2="+(num1-num2));
10         System.out.println("num1*num2="+(num1*num2));
11         System.out.println("num1/num2="+(num1/num2));
12         System.out.println("num1%num2="+(num1%num2));
13         System.out.println("num1="+num1+",num2="+num2);
14
15         System.out.println("双精度实型变量"+op1+"与"+op2+"的算术运算:");
16         System.out.println("op1+op2="+(op1+op2));
17         System.out.println("op1-op2="+(op1-op2));
18         System.out.println("op1*op2="+(op1*op2));
19         System.out.println("op1/op2="+(op1/op2));
20         System.out.println("op1%op2="+(op1%op2));
21         System.out.println("op1="+op1+",op2="+op2);
22
23         System.out.println("整型变量"+num1+"与常量 7 的算术运算:");
24         System.out.println("num1+7="+(num1+7));
25         System.out.println("num1-7="+(num1-7));
26         System.out.println("num1*7="+(num1*7));
```

```
27          System.out.println("num1/7="+(num1/7));
28          System.out.println("num1%7="+(num1%7));
29          System.out.println("num1="+num1);
30      }
31  }
```

运行结果如图 2-7 所示。

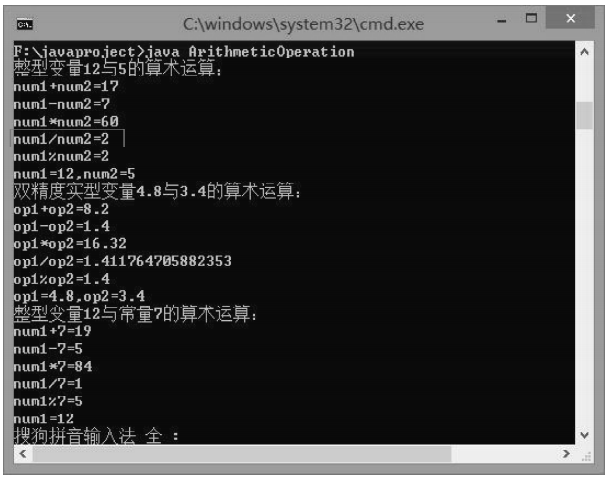


图 2-7　算术运算

代码清单 2-5 中定义了两组变量：整型变量 num1 和 num2、双精度实型变量 op1 和 op2，分别对它们进行了加、减、乘、除和求余运算，并输出了运算结果。在 Java 中，数据在进行运算时会遵循一个原则，运算符两端操作数的数据类型会保持一致，并且得到的结果也是该数据类型。这就是图 2-7 中 num1 除以 num2 的结果为 2，而不是 2.4 的原因。如果运算符两端操作数的数据类型不一致，则在运算之前，需要调整它们的类型，让它们保持一致。另外，在做运算时，这些运算符并没有改变操作数本身的值。

在使用/（除法）运算符时，第二个操作数的值不能为 0，否则会引发算术异常，导致程序运行出错。在使用%（求余）运算符时，如果运算符两端的操作数均为整型，则同样会引发算术异常；如果运算符两端的操作数有实型（浮点型），则计算会得到结果非数：NaN。

算术运算符中有两个特别的运算符：++（自增运算符）与--（自减运算符），均为单目运算符（只有一个操作数）。这两个运算符本身的运算规则十分简单：自增操作符对其操作数做加 1 操作，自减运算符对其操作数做减 1 操作。它们只能对变量进行操作，不能使用自增和自减运算符来操作常量。自增操作符和自减操作符会改变变量本身的值。代码清单 2-6 演示了自增运算符与自减运算符的基本用法。

代码清单 2-6　SelfAddAndSelfSub.Java

```
1  public class SelfAddAndSelfSub{
2      public static void main(String[] args){
3              int num1=10;
4              int num2=15;
5              System.out.println("未做自增前 num1="+num1);
6              System.out.println("未做自减前 num2="+num2);
7              num1++;
8              num2--;
```

```
9                System.out.println("做自增后 num1="+num1);
10               System.out.println("做自减后 num2="+num2);
11       }
12   }
```

运行结果如图 2-8 所示。

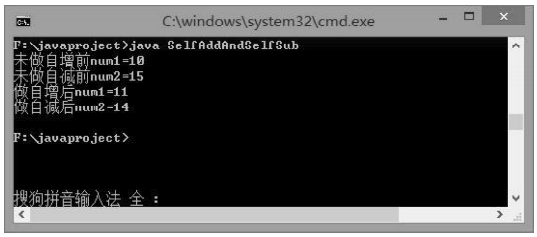


图 2-8　自增、自减运算

自增、自减运算符在变量左端时，会先对变量做加 1 或减 1 运算，然后再使用变量；自增、自减运算符在变量右端时，会先使用变量，然后再做加 1 减 1 运算。我们通过代码清单 2-7 来进行演示。

代码清单 2-7　SelfOperation.Java

```
1   public class SelfOperation{
2       public static void main(String[] args){
3               int num=5;
4               System.out.println(num++);
5               System.out.println(num);
6
7               System.out.println(--num);
8               System.out.println(num);
9
10              int result=++num+5;
11              System.out.println(result);
12              result=12+num--;
13              System.out.println(result);
14      }
15  }
```

运行结果如图 2-9 所示。

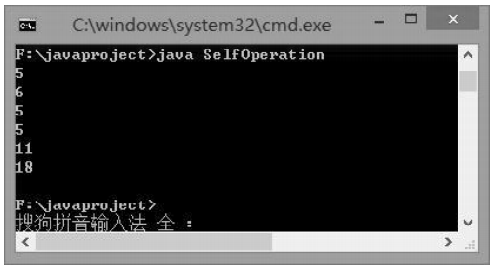


图 2-9　自增、自减运算

在代码清单 2-7 中，第 3 行代码定义了整型变量 num 并赋值为 5；第 4 行代码输出 num

时，由于自增运算符在变量右端，所以先输出了 num 的值，然后再对 num 做自增 1 运算，所以第 4 行代码输出结果为 5；第 5 行代码输出自增后的 num，其值为 6；第 7 行代码输出 num 时，自减运算符在变量左端，所以先对 num 做自减 1 操作，然后再输出 num 的值，所以第 7 行代码和第 8 行代码均输出值 5；同理第 10 行代码中 num 先做自增运算，然后与 5 相加，因此 result 的值为 11；第 12 行代码中 num 先与 12 相加，然后在做自减，所以 result 的值为 18。

在使用自增、自减运算符时，尽量不要和其他运算符连用，避免出现表达式难以理解的情况，如：

```
int a=10;
int b=5;
int result=a+++b;
System.out.println(result+","+a+","+b);
```

上述代码段的输出结果应为多少呢？如果这个表达式理解为(a++)+b，则 result 的值应为 15，a 的值应为 11，b 的值应为 5；如果理解为 a+(++b)，则 result 的值应为 16，a 的值应为 10，b 的值应为 6。因为 Java 运算是从左往右执行，因此该表达式应理解为(a++)+b。对于这种难以阅读的表达式不建议大家使用。为提高代码可读性，可以将其写为

```
int a=10;
int b=5;
int result;
a++;
result=a+b;
System.out.println(result+","+a+","+b);
```

2.3.2　赋值运算符

在 Java 程序中，可以通过使用赋值运算符来改变一个变量的值。与在数学中不同，运算符“=”用于将其右边的值赋给左边的变量。赋值运算符左端必须是变量。除了基本的赋值运算符外，Java 还提供了一些复合的赋值运算符，包括+=、-=、*=和/=。复合赋值运算符将其左端的变量与右端的表达式做指定运算后，再将运算结果赋值给左端的变量。赋值运算符使用示例如代码清单 2-8 所示。

代码清单 2-8　AssignmentEx.Java

```
1  public class Assignment{
2      public static void main(String[] args){
3              int num=5;
4              num+=10;
5              System.out.println(num);
6              num-=2*5;
7              System.out.println(num);
8              num*=2+5;
9              System.out.println(num);
10             num/=5;
11             System.out.println(num);
```

```
12      }
13  }
```

运行结果如图 2-10 所示。

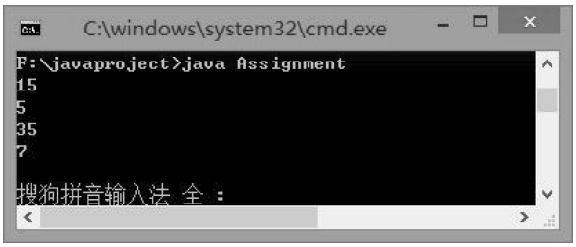


图 2-10　赋值运算符

在代码清单 2-8 中，每次使用赋值运算符都会导致变量 num 的值发生变化。第 3 行使用＝为变量 num 赋初值后，num 的值为 5；第 4 行使用＋＝将 5 与 10 求和后将值赋给 num，其值为 15；第 6 行 num－＝2＊5 等价于 num＝num－2＊5，执行后 num 值为 5；第 8 行 num＊＝2＋5 等价于 num＝num＊（2＋5），执行后 num 值为 35，应注意在使用复合赋值运算符进行指定运算时，将复合赋值运算符右端的表达式作为一个整体来参与运算，因此 num＊＝2＋5 并不等价于 num＝num＊2＋5；第 10 行中 num/＝5，相当于计算 35/5 后将结果赋值给 num，其值为 7。

Java 中除了上述两类常用运算符外，还包括比较运算符和逻辑运算符，我们将在第三章中对这两种运算符进行介绍。

2.4 数据类型转换

在 Java 中进行运算时，需要统一运算符两端数据的数据类型，例如有以下代码段：

```
int a=5;
double b=18;
System.out.println(a+b);
```

输出语句想要输出表达式 a＋b 的值。但在表达式 a＋b 中，变量 a 的类型是 int 类型，而变量 b 的类型是 double 类型，两个操作数的数据类型不一致，想要完成计算，就需要先统一他们的数据类型。Java 中提供了两种数据类型转换的方法：隐式转换和强制转换。

隐式转换是由系统自动完成的。系统可以帮助我们将使用存储单元较少的数据类型转换为使用存储单元较多的数据类型，就好像可以直接把容量较小的容器中盛放的物品放置到容量较大的容器中一样。可以按图 2-11 中箭头指示的方向进行隐式的数据类型转换。

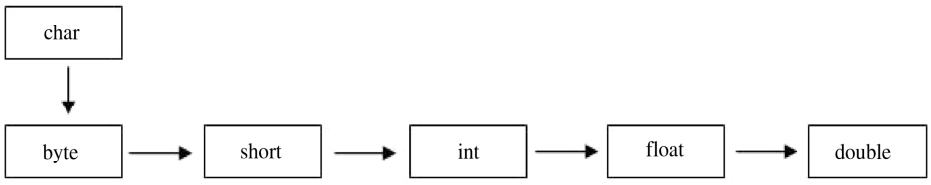


图 2-11　隐式转换示意图

当进行运算时，如果运算符两端操作数数据类型不一致，则系统会尝试进行隐式转换，

如果无法进行隐式转换，会在编译时发生错误。我们在代码清单 2-9 中对隐式转换进行了演示。

代码清单 2-9　ImplicitConversion. Java

```
1  public class ImplicitConversion{
2      public static void main(String[] args){
3              char ch='a';
4              int num=120;
5              float fNum=12.5f;
6              double dNum=22.5;
7
8              num=ch+num;
9              dNum+=num-fNum;
10             fNum+=ch;
11
12             System.out.println("num="+num+",fNum="+fNum+",dNum="+
dNum);
13     }
14 }
```

运行结果如图 2-12 所示。

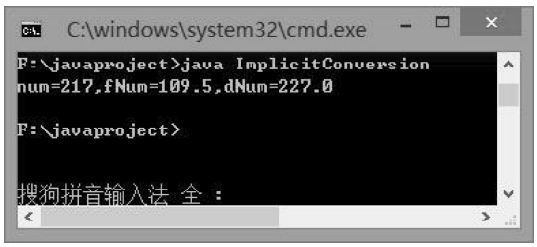


图 2-12　数据类型隐式转换

在代码清单 2-9 中，第 3 行定义了一个字符类型变量 ch，其值为 'a'，Unicode 码值为 97；在执行第 8 行代码 num=ch+num 时，系统将 ch 转换为 int 类型，相当于执行 num=97+120；因此在第 12 行输出 num 时，其值为 217。同样，在第 9 行中执行 dNum+=num-fNum 时，系统将 num 和 fNum 转换为双精度实型后再进行运算，因此得到的结果为 227.0。

有时候我们在处理数据时，系统无法完成隐式的转换，如下面的代码：

```
char ch='A';
ch+=32;
```

这两句代码是想将大写字符 A 转换为小写字符 a，复合赋值运算符先要将 ch 与 32 相加，然后将结果赋值给 ch，因此系统先将 ch 的值转换为整型值即 65，然后做加法操作得到结果 97。在将结果 97 赋值给 ch 时将会发生错误，系统会提示可能损失精度，这是因为当我们将使用内存单元较多的数据类型转换为使用内存单元较少的数据类型时，由于内存单元的减少，可能造成数据的丢失。就像将较大容器中盛装的水导入较小的容器中一样，有部分水会损失。这时我们可以使用强制类型转换。强制类型转换不能解决精度损失的问题，它只是强制将某种类型转换为另一种类型。我们可以按以下格式强制转换数据类型：

（数据类型）数据值

代码清单 2-10 演示了强制转换。

代码清单 2-10　ForcedConversion. Java

```
1  public class ForcedConversion{
2      public static void main(String[] args){
3              char ch='a';
4              int num=20;
5              System. out. println(ch-32);
6              System. out. println(65. 75-num);
7              System. out. println();
8              System. out. println((char)(ch-32));
9              System. out. println((int)(65. 75-num));
10     }
11 }
```

运行结果如图 2-13 所示。

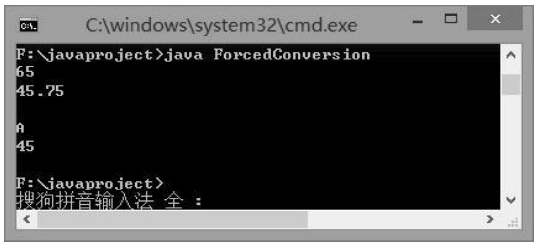


图 2-13　数据类型强制转换

在代码清单 2-10 中，第 5 行的表达式 ch-32 在运算时，为保持运算符两端数据类型一致，将 ch 转换为了整型，计算结果也为整型值，为 65，因此输出的结果为 65。第 6 行的表达式 65. 75-num 同样将 num 转换为了 double 类型，计算结果也为 double 类型，值为 45. 75。第 8 行将 ch-32 的结果做了强制类型转换，将它转换为字符型，因为 Unicode 码 65 对应的字符为 'A'，所以输出结果为 A。第 9 行，将 65. 75-num 的结果强制转换为整型，输出结果为 45，转换丢失了数据的精度。在 Java 中将浮点类型转换为整型值时，将直接丢弃小数部分。

2.5　基于控制台的基本输入和输出

前述内容中给大家提到过，计算机程序的宏观流程可以表示为

数据输入→数据处理→数据输出

Java 中提供了多种数据输入和输出的方式，本节将介绍基于控制台的输入输出。在 Java 中，System. out 表示标准输出设备，System. in 表示标准输入设备。默认情况下，标准输出设备是显示器，而标准输入设备是键盘。如果想在控制台中输出数据，只需要调用 println 方法就能完成大多数任务，就像前面的示例中所演示的一样。Java 并不支持直接从控制台输入数据，若需要从控制台输入数据，可以使用 Scanner 类来创建它的对象，以读取来自 System. in 的输入。Scanner 的使用方法如代码清单 2-11 所示。

代码清单 2-11　ScannerTest. Java

```
1  import Java. util. Scanner;
2  public class ScannerTest{
3      public static void main(String[]args){
4              int num;
5              double dNum;
6              Scanner scanner=new Scanner(System. in);
7              System. out. println("请输入一个整数:");
8              num=scanner. nextInt();
9              System. out. println("请输入一个小数:");
10             dNum=scanner. nextDouble();
11             System. out. println("num="+num+",dNum="+dNum);
12     }
13 }
```

运行结果如图 2-14 所示。

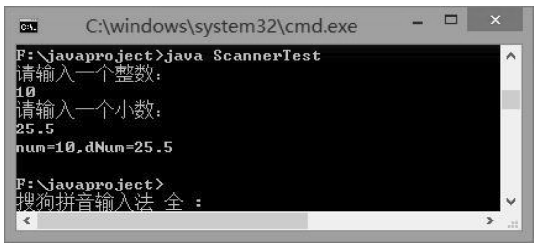


图 2-14　控制台输入

在代码清单 2-11 中，第 1 行的 import 语句用于导入一个包，它的作用是告诉编译器 Scanner 类被定义在 Java. util 包中；第 6 行创建了一个 Scanner 的对象，用于从键盘获取输入；第 7 行输出了一个提示，提示用户输入一个整数；第 8 行使用对象 scanner 的 nextInt 方法接收了一个从键盘输入的整数；第 9、10 行做了类似的处理，使用了 nextDouble 方法接收了一个从键盘输入的双精度浮点数。Scanner 类提供了一系列的方法，来让我们从键盘输入各种类型的数据，见表 2-3 所列。

表 2-3　从键盘输入数据的类型

方法	说明	方法	说明
nextByte	读取一个 byte 型整数	nextShort	读取一个 short 型整数
nextInt	读取一个 int 型整数	nextLong	读取一个 long 型整数
nextFloat	读取一个 float 型的数	nextDouble	读取一个 double 型的数
next	读取一个字符串，以空白符结束	nextLine	读取一行文本，即以按下回车键为结束标志

2.6 实例学习

2.6.1 数据交换

实例 2.1 在程序设计中，经常需要交换两个变量的值，代码清单 2-12 演示了两个整型数据的交换。

代码清单 2-12 ExchangeNums.Java

```
1  public class ExchangeNums{
2     public static void main(String[] args){
3           int num1=5;
4           int num2=10;
5           System.out.println("num1="+num1+",num2="+num2);
6           int temp;
7           temp=num1;
8           num1=num2;
9           num2=temp;
10          System.out.println("num1="+num1+",num2="+num2);
11       }
12 }
```

运行结果如图 2-15 所示。

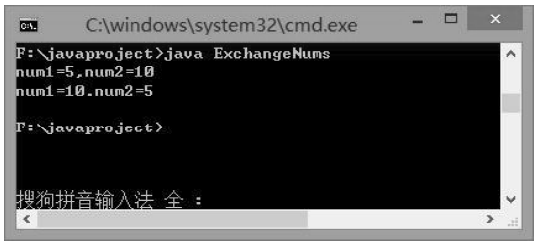


图 2-15 数据交换

在代码清单 2-12 中，第 3、4 行定义了变量 num1 和 num2，它们的值分别为 5 和 10。第 6 行定义了一个中间变量 temp，用于在 num1 和 num2 交换数据时，保存其中某个变量的值。就像我们想要将杯子 A 中的可乐和杯子 B 中的牛奶进行交换时，需要一个空杯子 C 来进行中转。先将杯子 B 中的牛奶倒入空杯子 C，然后将杯子 A 中的可乐倒入已经为空的杯子 B 中，然后再将杯子 C 中的牛奶倒入杯子 A 中，完成交换过程。在程序中，使用中间变量 temp 来保存 num1 的值，是因为当执行第 8 行语句 num1=num2 时，num2 的值会覆盖掉 num1 的值，也就是说当第 8 行代码执行完毕后，num1 和 num2 的值均为 10。图 2-16 演示了整个交换过程。

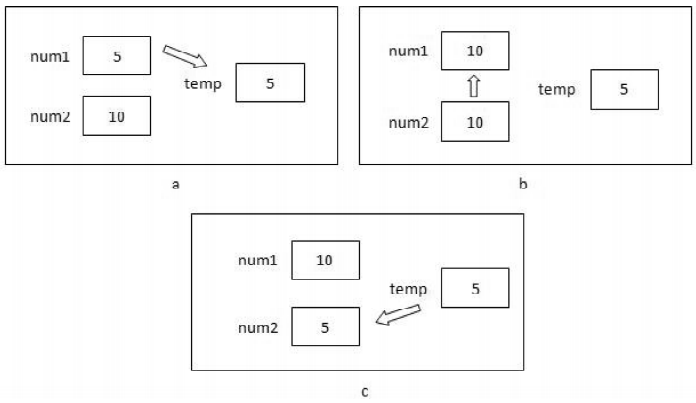


图 2-16　数据交换示意图

2.6.2　求矩形周长和面积

实例 2.2　现在可以编程来计算矩形周长和面积了。编写一个程序，提示用户输入矩形的长度和宽度，然后计算矩形的周长和面积，最后把结果反馈给用户，如代码清单 2-13 所示。

代码清单 2-13　Rectangle.Java

```
1  import Java.util.*;
2  public class Rectangle
3  {
4      public static void main(String[] args)
5      {
6          int width,height;
7          Scanner scanner=new Scanner(System.in);
8          System.out.println("请输入要计算的矩形的长度和宽度:");
9          width=scanner.nextInt();
10         height=scanner.nextInt();
11         System.out.println("矩形的周长为:"+(width+height)*2);
12         System.out.println("矩形的面积为:"+width*height);
13     }
14 }
```

运行结果如图 2-17 所示。

图 2-17　求矩形周长与面积

在代码清单 2-13 中，第 6 行定义了两个整型变量用于接收用户输入的长度和宽度。第 7 行定义了一个 Scanner 的对象 scannner，从控制台（标准输入流）中获取用户输入。第 9 行

与第 10 行调用了 scanner 对象的 nextInt（）方法，接收了两个整数，分别赋值给变量 width 和 height。第 11 行按矩形的周长公式（长＋宽）×2 进行计算，并将结果输出到控制台。第 12 行按矩形的求面积公式（长×宽）进行计算，并将结果输出到控制台。

2.6.3 求三位数中各位之和

实例 2.3 如有三位数 123，则计算 1＋2＋3 的和。

在数据处理过程中，我们经常需要析出数据中的某位数，可以使用求余运算和除法运算来解决这个问题，如代码清单 2-14 所示。

代码清单 2-14 DivAndModEx.Java

```
1  import Java.util.*;
2  public class DivAndModEx{
3      public static void main(String[] args){
4          int num,sum=0;
5          Scanner scanner=new Scanner(System.in);
6          System.out.println("请输入一个三位整数:");
7          num=scanner.nextInt();
8          sum+=num/100;
9          sum+=num%10;
10         sum+=num/10%10;
11         System.out.println(num+"三位数字之和为:"+sum);
12     }
13 }
```

运行结果如图 2-18 所示。

图 2-18 求三位数各位数字之和

在代码清单 2-14 中，第 4 行定义了两个整型变量，num 用于接收输入的三位数，sum 用于保存三位数各位数字之和。第 8 行 num/100 得到三位数的百位数字，并将其加到 sum 中。第 9 行 num%10 得到三位数的个位数字，并将其加到 sum 中。第 10 行 num/10%10 先通过除法运算移出个位数，再通过求余运算得到三位数中的十位数字，并将其加到 sum 中。要特别注意，整数做除法运算，得到的结果仍然是整数。

2.7 本章小结

本章介绍了 Java 程序的基础知识。通过本章的学习，我们可以了解到计算机程序有三种基本结构：顺序、分支、循环。从宏观上看，计算机程序的执行流程一般为：数据输入→数