

21天学通

Java

庞永庆 庞丽娟 等编著

20

小时多媒体
语音视频教学

DVD

1→2→3→4→5→6→7→8→9→10→11→12→13→14→15→16→17→18→19→20→21

1→2→3→4→5→6→7→8→9→10→11→12→13→14→15→16→17→18→19→20→21

本书特色

- 基础知识→核心技术→典型实例→综合练习→项目案例
- 307个典型实例、1个项目案例、246个练习题
- 一线开发人员全程贴心讲解，上手毫不费力
- 20小时多媒体语音视频教学
- 本书源代码+本书电子教案(PPT)
- 1000余页编程参考宝典电子书(免费赠送)

超值DVD

Java高级架构师课程(总共122门课程， 1460GB)

<https://www.consultdog.com/#/courseDetails?id=5>

最强java面试视频课程(21门课程， 126GB)

<https://www.consultdog.com/#/courseDetails?id=16>

本书涵盖主题

- JDK的下载、安装和使用
- 数据类型
- 运算符
- 流程控制
- 数组
- Java面向对象编程思想
- Java中访问的控制逻辑
- 修饰符
- 面向对象特性——继承
- 接口
- 构造器的定义和使用
- Java中异常处理机制
- 内部类的概念和访问
- 多线程
- 桌面程序开发中的控件定义
- 桌面程序开发中的事件
- JDBC数据库编程
- Java中的输入输出流
- 集合框架理论
- Java网络编程
- 学生管理系统

精彩内容，尽在21天学编程

21小时多媒体

语音视频教学

DVD

上架建议:程序设计/Java

网上订购: www.dearbook.com.cn
第二书店·第一服务



责任编辑: 高洪霞



本书贴有激光防伪标志, 凡没有防伪标志者, 属盗版图书。

ISBN 978-7-121-07897-2



9 787121 078972

定价:49.80元(含DVD光盘1张)

以任务驱动方式讲解，用实例引导读者学习

21天学通Java

庞永庆 庞丽娟 等编著

电子工业出版社

Publishing House of Electronics Industry
北京•BEIJING

内 容 提 要

本书是 Java 语言的入门教程，由浅入深，循序渐进地讲授如何使用 Java 语言进行程序开发。全书内容包括 Java 开发环境、Java 基本语法知识、Java 面向对象特点、Java 界面开发，以及 Java 数据库开发和网络编程开发。为了便于读者学习，本书最后一章对一个完整学生管理系统进行了分析。具体讲解了学生模块和老师模块，以及其他各个模块的功能分析。

本书旨在为 Java 语言的初学者和大中专学生提供易于入门，便于全面了解和掌握 Java 编程技术的教辅资料，同时对有一定经验的 Java 编程者和学习者也有一定的参考价值。

本书附赠 DVD 光盘 1 张，内容包括超大容量手把手教学视频、电子教案（PPT）、编程参考宝典电子书、源代码及各章习题答案。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有，侵权必究。

图书在版编目（CIP）数据

21 天学通 Java / 廖永庆等编著. —北京：电子工业出版社，2009.1
ISBN 978-7-121-07897-2

I. 2… II. 廖… III. JAVA 语言—程序设计 IV. TP312

中国版本图书馆 CIP 数据核字（2008）第 186408 号

责任编辑：高洪霞

印 刷：北京智力达印刷有限公司

装 订：北京中新伟业印刷有限公司

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本：787×1092 1/16 印张：26 字数：677 千字

印 次：2009 年 1 月第 1 次印刷

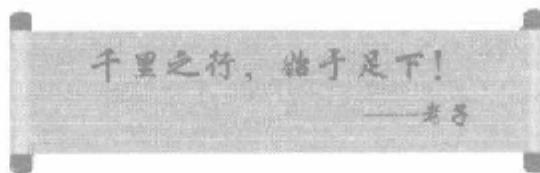
印 数：4000 册 定价：49.80 元（含 DVD 光盘 1 张）

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，
联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zts@phei.com.cn，盗版侵权举报请发邮件至 dhqq@phei.com.cn。

服务热线：（010）88258888。

本书特点



为什么要写这样一本书

作为一个初学 Java 编程的人,最重要的第一步是什么呢?毫无疑问,是选择一本好书。虽然现在书店中讲解 Java 基础的书已经有很多了,但是却很难找到适合初学者使用的一本。很多书中都有许多复杂难记的语法和概念,这些常是最容易使 Java 初学者放弃的地方,就像有一堵无形的墙立在面前而使他们无法逾越。

为了让那些 Java 初学人员少走弯路，快速而轻松地学会 Java 编程，笔者决定总结自己学习 Java 的经验，并结合多年的开发经验，编写一本能够真正让 Java 初学人员容易掌握的书。在这本书中，笔者将通过 21 天的学习规划，帮助读者快速掌握 Java 编程的基本知识，为以后进行 Java 开发打下基础。

本书有何特色

1. 细致体贴的讲解

为了让读者更快地上手,本书特别设计了适合初学者的学习方式,用准确的语言总结概念→用直观的图示演示过程→用详细的注释解释代码→用形象的比方帮助记忆。效果如下:



- ① **知识点介绍** 准确、清晰是其显著特点，一般放在每一节开始位置，让零基础的读者了解相关概念，顺利入门。
- ② **范例** 书中出现的完整实例，以章节顺序编号，便于检索和循序渐进地学习、实践，放在每节知识点介绍之后。
- ③ **范例代码** 与范例编号对应，层次清楚、语句简洁、注释丰富，体现了代码优美的原则，有利于读者养成良好的代码编写习惯。对于大段程序，均在每行代码前设定编号便于学习。
- ④ **运行结果** 对范例给出运行结果和对应图示，帮助读者更直观地理解范例代码。
- ⑤ **代码解析** 将范例代码中的关键代码行逐一解释，有助于读者掌握相关概念和知识。
- ⑥ **综合练习** 为了便于读者巩固所学内容，本书每章中均提供了综合练习，并给出了操作提示和结果，配合读者自己动手实践。
- ⑦ **习题** 每章最后提供专门的测试习题，供读者检验所学知识是否牢固掌握，题目的提示或答案放在光盘中。
- ⑧ **贴心的提示** 为了便于读者阅读，全书还穿插着一些技巧、提示等小贴士，体例约定如下：
- 提示：通常是一些贴心的提醒，让读者加深印象或提供建议，或者解决问题的方法。
 - 注意：提出学习过程中需要特别注意的一些知识点和内容，或者相关信息。
 - 警告：对操作不当或理解偏差将会造成的灾难性后果做警示，以加深读者印象。
- 经作者多年的培训和授课证明，以上讲解方式是最适合初学者学习的方式，读者按照这种方式，会非常轻松、顺利地掌握本书知识。

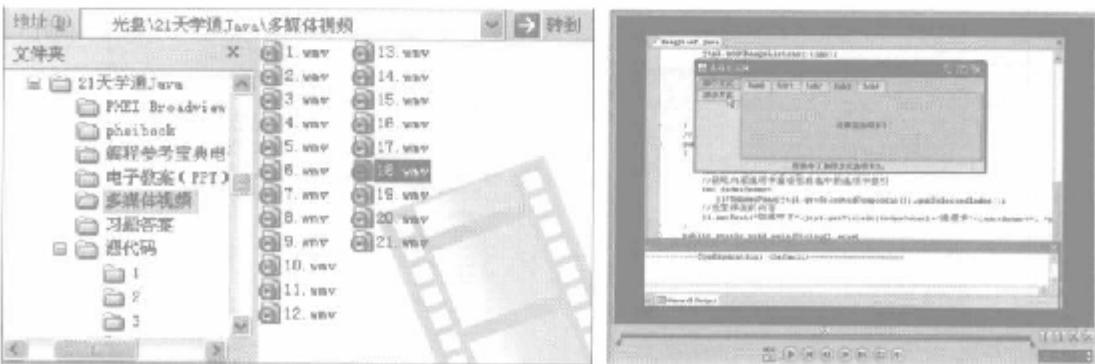
2. 实用超值的 DVD 光盘

为了帮助读者比较直观地学习，本书附赠 DVD 光盘，内容包括多媒体视频、电子教案（PPT）、编程参考宝典电子书、各章习题答案和实例源代码等。

● 多媒体视频

配有长达 20 小时手把手教学视频，讲解关键知识点界面操作和书中的一些综合练习题。作者亲自配音、演示，手把手教会读者使用。

光盘\21天学通Java
└─ 21天学通Java
 └─ PHEI Broadview 2000专业书目
 └─ pheibook
 └─ 编程参考宝典电子书
 └─ 电子教案（PPT）
 └─ 多媒体视频
 └─ 习题答案
 └─ 源代码



● 电子教案 (PPT)

本书可以作为高校相关课程的教材或课外辅导书, 所以笔者特别为本书制作了电子教案 (PPT), 以方便老师教学使用。

● 编程参考宝典电子书

为方便广大读者学习, 特别制作了编程开发参考电子书, 供读者查阅和参考。



3. 提供完善的技术支持

本书提供了论坛: <http://www.rzchina.net>, 读者可以在上面提问交流。另外, 论坛上还有一些小的教程、视频动画和各种技术文章, 可帮助读者提高开发水平。

4. 丰富的额外素材下载

相关的开发素材文件, 在 www.broadview.com.cn 提供下载。

推荐的学习计划

本书作者在长期从事相关培训或教学实践过程中, 归纳了最适合初学者的学习模式, 并参考了多位专家的意见, 为读者总结了合理的学习时间分配方式, 列表如下:

推荐时间安排		自学目标 (框内打钩表示已掌握)	难度指数
第1周	第1天	熟练掌握如何搭建 Java 开发环境, 包括下载、安装和配置 JDK <input type="checkbox"/> 能够编写和编译 Java 程序, 并能够运行生成文件 <input type="checkbox"/>	★
	第2天	了解 Java 有哪些基本数据类型 <input type="checkbox"/> 掌握各种数据类型的基本含义 <input type="checkbox"/> 学会如何进行数据类型转换 <input type="checkbox"/> 了解标识符和保留字等基本概念 <input type="checkbox"/>	★★
		了解如何在 Java 程序中进行注释 <input type="checkbox"/>	
		了解算术运算符的概念和熟练使用算术运算符 <input type="checkbox"/> 了解关系运算符的概念和熟练使用关系运算符 <input type="checkbox"/> 了解逻辑运算符的概念和熟练使用逻辑运算符 <input type="checkbox"/> 了解三元运算符的概念和熟练使用三元运算符 <input type="checkbox"/>	
	第3天	了解位运算符的概念和熟练使用位运算符 <input type="checkbox"/> 了解位移运算符的概念和熟练使用位移运算符 <input type="checkbox"/> 了解赋值运算符的概念和熟练使用赋值运算符 <input type="checkbox"/>	★★
		了解算术运算符的概念和熟练使用算术运算符 <input type="checkbox"/> 了解关系运算符的概念和熟练使用关系运算符 <input type="checkbox"/> 了解逻辑运算符的概念和熟练使用逻辑运算符 <input type="checkbox"/> 了解三元运算符的概念和熟练使用三元运算符 <input type="checkbox"/>	
		了解位运算符的概念和熟练使用位运算符 <input type="checkbox"/> 了解位移运算符的概念和熟练使用位移运算符 <input type="checkbox"/> 了解赋值运算符的概念和熟练使用赋值运算符 <input type="checkbox"/>	
		了解算术运算符的概念和熟练使用算术运算符 <input type="checkbox"/> 了解关系运算符的概念和熟练使用关系运算符 <input type="checkbox"/> 了解逻辑运算符的概念和熟练使用逻辑运算符 <input type="checkbox"/> 了解三元运算符的概念和熟练使用三元运算符 <input type="checkbox"/>	
		了解位运算符的概念和熟练使用位运算符 <input type="checkbox"/> 了解位移运算符的概念和熟练使用位移运算符 <input type="checkbox"/> 了解赋值运算符的概念和熟练使用赋值运算符 <input type="checkbox"/>	

续表

推荐时间安排	自学目标（框内打钩表示已掌握）	难度指数
第1周	第4天 了解 if 条件语句和掌握各种 if 条件语句的使用 了解 switch 分支语句和掌握 switch 分支语句的使用 了解 while 循环语句和掌握 while 循环语句的使用 了解 do-while 循环语句和掌握 do-while 循环语句的使用 了解 for 循环语句和掌握 for 循环语句的使用	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
	第5天 知道如何创建数组，包括创建一维数组和多维数组 能够对数组进行初始化操作 熟练掌握如何借助数组来解决实际问题	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
	第6天 了解什么是面向对象 熟悉 Java 中的类并能够进行类的操作 掌握成员变量和局部变量的区别 掌握 Java 程序中方法的创建和使用	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
	第7天 了解包的概念和如何使用包 知道类的访问级别有哪些，它们有什么区别 重点掌握 final 修饰符和 static 修饰符	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
	第8天 了解什么是继承和继承如何使用 掌握声明成员变量的修饰符 熟练掌握方法的重写和重载 了解枚举、反射和泛型等热门技术	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
	第9天 会定义接口和访问接口中的变量 熟练掌握接口的使用 了解接口和抽象类的区别 了解接口的多态问题 熟练掌握使用 instanceof 判断类型	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
	第10天 了解什么是构造器 熟练掌握如何创建构造器 熟练掌握构造器的使用，包括构造器如何调用等问题 了解构造器的一些基本机制	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
第2周	第11天 了解什么是异常处理 熟练掌握如何进行异常处理 掌握异常的分类和区别不同的异常 能够自定义异常和使用自定义异常	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
	第12天 了解什么是非静态内部类和如何进行非静态内部类和外部类之间的访问 了解什么是局部内部类和如何进行局部内部类和外部类之间的访问 了解什么是静态内部类和如何进行静态内部类和外部类之间的访问 了解什么是匿名内部类和如何进行匿名内部类和外部类之间的访问	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>

续表

推荐时间安排		自学目标（框内打钩表示已掌握）	难度指数
第2周	第 13 天	了解什么是多线程	<input type="checkbox"/>
		熟练掌握如何定义和使用多线程	<input type="checkbox"/>
		了解多线程的生命周期	<input type="checkbox"/>
		掌握多线程的调用的几个情况	<input type="checkbox"/>
		了解多线程的同步问题	<input type="checkbox"/>
	第 14 天	了解 Swing 开发的基本过程	<input type="checkbox"/>
		掌握如何创建窗口、面板、标签和按钮	<input type="checkbox"/>
第3周	第 15 天	掌握和熟练使用 Swing 中的事件	<input type="checkbox"/>
		了解各种布局管理器的样式	<input type="checkbox"/>
		掌握每一种布局管理器的使用	<input type="checkbox"/>
	第 16 天	了解如何创建文本框和文本框的实际应用	<input type="checkbox"/>
		了解如何创建复选框和复选框的实际应用	<input type="checkbox"/>
		了解如何创建单选按钮和单选按钮的实际应用	<input type="checkbox"/>
	第 17 天	对数据库有基本了解	<input type="checkbox"/>
		熟练掌握 JDBC 的编程步骤	<input type="checkbox"/>
		掌握如何在 Java 中进行数据库操作	<input type="checkbox"/>
	第 18 天	了解什么是 I/O 流	<input type="checkbox"/>
		掌握流的分类	<input type="checkbox"/>
		熟练掌握流如何进行文件操作	<input type="checkbox"/>
	第 19 天	了解什么是集合框架和集合框架包括哪些形式	<input type="checkbox"/>
		掌握什么是列表和列表中包括哪些类和接口	<input type="checkbox"/>
		掌握什么是集合和集合中包括哪些类和接口	<input type="checkbox"/>
		掌握什么是映射和映射中包括哪些类和接口	<input type="checkbox"/>
	第 20 天	了解什么是协议，有哪些协议	<input type="checkbox"/>
		了解网络编程的模型	<input type="checkbox"/>
		熟练掌握使用 Socket 进行网络编程	<input type="checkbox"/>
	第 21 天	掌握实际开发的步骤	<input type="checkbox"/>
		能够熟练开发和学生管理系统相类似的系统	<input type="checkbox"/>
		掌握 Java 中的界面开发	<input type="checkbox"/>
		掌握 Java 中如何连接数据库	<input type="checkbox"/>

本书适合哪些读者阅读

本书非常适合以下人员阅读：

- 从未接触过 Java 编程的自学人员；
- 了解一些 Java，但还需要进一步学习的人员；
- 各大中专院校的在校学生和相关授课老师；
- 其他编程爱好者。

本书作者
2009 年 1 月

目 录

第一篇 基础篇

第1章 Java简介 ( 教学视频: 33分钟)	21
1.1 Java的平台简介	21
1.2 安装工具包	22
1.2.1 下载JDK	22
1.2.2 安装JDK	24
1.2.3 查看与设置环境变量	25
1.2.4 JDK常用命令	27
1.2.5 Java各个目录含义	28
1.2.6 要善于使用JDK的帮助文件	28
1.3 程序开发过程	29
1.4 编码规范	29
1.5 HelloWorld: 第一个Java程序	30
1.5.1 编写程序代码	30
1.5.2 编译程序代码并运行	30
1.5.3 注意事项	31
1.6 使用Eclipse集成开发工具开发	31
1.7 综合练习	32
1.8 小结	32
1.9 习题	33
第2章 Java的基本数据类型 ( 教学视频: 38分钟)	34
2.1 数据类型	34
2.1.1 整型	34
2.1.2 浮点型	35
2.1.3 字符型 (char)	36
2.1.4 布尔型 (boolean)	36
2.2 数据类型间的转换	36
2.2.1 自动转换	36
2.2.2 强制转换	37
2.2.3 隐含转换	37
2.3 标识符的命名	38
2.3.1 标识符的命名规则	38
2.3.2 代码演示如何定义标识符	38

2.3.3 不好的标识符命名	38
2.3.4 良好的标识符命名	39
2.4 关键字	39
2.5 代码注释	40
2.5.1 行注释	40
2.5.2 块注释	41
2.5.3 文档注释用户自定义类型	41
2.6 综合练习	43
2.7 小结	43
2.8 习题	43
第3章 运算符 ( 教学视频: 43分钟)	45
3.1 算术运算符	45
3.1.1 “+”: 加法运算符	45
3.1.2 “-”: 减法运算符	46
3.1.3 “*”: 乘法运算符	47
3.1.4 “/”: 除法运算符	48
3.1.5 “%”: 求余运算符	48
3.2 自增自减运算符	49
3.3 关系运算符	51
3.3.1 “==”、“!=”	51
3.3.2 “>”、“<”、“>=”、“<=”	52
3.4 逻辑运算符	53
3.4.1 “&&”: 与运算符	53
3.4.2 “ ”: 或运算符	53
3.4.3 “!”: 非运算符	54
3.4.4 逻辑运算符总结	54
3.5 三元运算符	55
3.6 位运算符	55
3.6.1 “&”: 按位与运算符	56
3.6.2 “ ”: 按位或运算符	56
3.6.3 “^”: 按位异或运算符	57
3.7 位移运算符	57
3.7.1 “>>”: 带符号右移运算符	58
3.7.2 “<<”: 带符号左移运算符	58
3.7.3 “>>>”: 无符号右移运算符	58
3.8 赋值运算符	59
3.8.1 一般赋值运算符	59
3.8.2 运算赋值运算符	59
3.9 运算符之间的优先级	60

3.10 综合练习	61
3.11 小结	62
3.12 习题	62
第4章 流程控制 ( 教学视频: 58分钟)	64
4.1 if 条件语句	64
4.1.1 if语句的语法	64
4.1.2 if语句用法举例	64
4.2 switch 分支语句	67
4.2.1 switch 分支语句的语法	67
4.2.2 switch 分支语句表达式的使用条件	68
4.2.3 switch 分支语句举例	70
4.3 while 循环语句	72
4.3.1 while 循环语句的语法	72
4.3.2 while 循环语句举例	73
4.4 do...while 循环语句	73
4.4.1 do...while 循环语句的语法	74
4.4.2 do ... while 循环语句举例	74
4.5 for 循环语句	75
4.5.1 for 循环语句的语法	75
4.5.2 用 for 循环来实现其他循环语句	76
4.5.3 for 循环语句的举例	77
4.6 如何中断和继续语句的执行	78
4.6.1 break: 中断语句执行	78
4.6.2 continue: 继续语句执行	79
4.7 综合练习	79
4.8 小结	80
4.9 习题	81
第5章 数组 ( 教学视频: 52分钟)	83
5.1 如何创建数组	83
5.1.1 创建数组	83
5.1.2 创建多维数组	84
5.2 数组的初始化	85
5.2.1 创建并初始数组元素	85
5.2.2 循环初始化	87
5.3 数组操作的举例	88
5.3.1 数组元素值的复制	88
5.3.2 数组元素的排序	90
5.3.3 在数组里查找指定元素	91

5.3.4 利用数组打印 26 个英文字母	92
5.4 综合练习	93
5.5 小结	94
5.6 习题	94

第二篇 面向对象篇

第 6 章 类与对象 (教学视频: 48 分钟) 96

6.1 什么是面向对象	96
6.1.1 面向对象编程的特点	96
6.1.2 面向对象编程与面向过程编程的区别	97
6.2 什么是类	97
6.2.1 类的定义和对象的创建	97
6.2.2 如何使用现有类	99
6.2.3 类设计的技巧	100
6.3 成员变量	101
6.3.1 成员变量的创建	101
6.3.2 成员变量的初始化	102
6.4 局部变量	105
6.4.1 局部变量的创建和初始化	105
6.4.2 局部变量和成员变量的区别	106
6.5 方法	106
6.5.1 方法的创建和参数	106
6.5.2 方法参数的传递	108
6.6 对象引用的使用	110
6.6.1 调用不存在的对象或成员变量	110
6.6.2 调用对象为 null 值的引用	111
6.6.3 对象引用间的比较	113
6.7 this	113
6.8 要活用 JDK 已有的类	114
6.8.1 Date 类	114
6.8.2 Integer 类	116
6.9 综合练习	117
6.10 小结	118
6.11 习题	118

第 7 章 控制逻辑 (教学视频: 50 分钟) 120

7.1 包 (package)	120
7.1.1 创建一个包	120
7.1.2 如何使用包	121

7.1.3 什么是静态引入	122
7.2 类的访问级别	123
7.2.1 公开的访问级别	123
7.2.2 默认的访问级别	124
7.3 什么是封装	124
7.4 最终修饰符	125
7.4.1 final 修饰对象类型的成员变量	127
7.4.2 final 修饰基本类型的成员变量	129
7.4.3 final 修饰的局部变量	131
7.4.4 final 修饰的方法	132
7.5 静态修饰符	134
7.5.1 什么是静态变量	134
7.5.2 静态变量的访问	135
7.5.3 什么是静态常量	137
7.6 综合练习	139
7.7 小结	140
7.8 习题	140
第8章 继承 ( 教学视频: 72分钟)	141
8.1 什么是继承	141
8.1.1 类的继承	142
8.1.2 继承的语法	145
8.2 修饰符	146
8.2.1 public: 声明成员变量为公共类型	146
8.2.2 private: 声明成员变量为私有类型	147
8.2.3 default: 声明成员变量为默认类型	148
8.2.4 protected: 声明成员变量为保护类型	149
8.3 成员变量的覆盖	150
8.4 对象引用	151
8.5 方法的重写和重载	152
8.5.1 方法重写的特点	152
8.5.2 方法重载的特点	154
8.5.3 重写的返回类型	156
8.5.4 重写是基于继承的	158
8.5.5 静态方法是不能重写的	159
8.5.6 三者之间的关系	161
8.5.7 重写 <code>toString</code> 方法	162
8.5.8 重写 <code>equals</code> 方法	163
8.6 final 与继承的关系	164
8.7 abstract 与继承的关系	165

8.8 什么是多态	166
8.9 什么是枚举类	168
8.10 什么是反射机制	169
8.11 什么是泛型	170
8.12 综合练习	172
8.13 小结	172
8.14 习题	172
第 9 章 接口 ( 教学视频: 47 分钟)	174
9.1 什么是接口	174
9.1.1 接口的定义	174
9.1.2 访问接口里的常量	176
9.2 接口的使用	177
9.2.1 接口里的方法如何创建	177
9.2.2 接口引用怎么使用	178
9.3 什么是抽象类	180
9.3.1 抽象类的使用和特点	180
9.3.2 抽象类与接口区别	183
9.4 接口的多态	183
9.5 判断类型	185
9.5.1 什么是 instanceof	185
9.5.2 使用 instanceof 的注意事项	188
9.6 综合练习	189
9.7 小结	189
9.8 习题	189
第 10 章 构造器 ( 教学视频: 46 分钟)	191
10.1 什么是构造器	191
10.1.1 构造器的使用	191
10.1.2 被修饰的构造器	193
10.1.3 构造器方法与普通方法的区别	196
10.2 如何实例化一个对象	197
10.3 构造器的使用	199
10.3.1 构造器的调用	199
10.3.2 构造器重载	202
10.3.3 父子类间的构造器的调用流程	204
10.3.4 如何自定义构造器	207
10.4 什么是单子模式	208
10.5 构造器在程序中是何时运行的	211
10.6 综合练习	214

10.7 小结	215
10.8 习题	215
第 11 章 异常处理 ( 教学视频: 60 分钟)	217
11.1 异常处理基本介绍	217
11.1.1 try 和 catch 捕获异常	217
11.1.2 try-catch 语句使用注意点	218
11.1.3 finally 语句的使用	220
11.1.4 再谈异常处理注意点	222
11.2 异常的分类	223
11.2.1 捕获异常	223
11.2.2 未捕获异常	225
11.3 抛出异常	225
11.3.1 抛出异常的简单介绍	225
11.3.2 使用 throws 和 throw 语句抛出异常	227
11.4 自定义异常	227
11.4.1 创建和使用自定义异常类	227
11.4.2 自定义异常的实际应用	228
11.5 综合练习	231
11.6 小结	232
11.7 习题	232
第 12 章 内部类 ( 教学视频: 71 分钟)	234
12.1 非静态内部类	234
12.1.1 创建非静态内部类	234
12.1.2 在外部类中访问内部类	235
12.1.3 在外部类外访问内部类	236
12.1.4 在内部类中访问外部类	237
12.2 局部内部类	240
12.2.1 创建局部内部类	240
12.2.2 在局部内部类中访问外部类成员变量	240
12.2.3 在局部内部类中访问外部类的局部变量	241
12.2.4 静态方法中的局部内部类	243
12.3 静态内部类	244
12.3.1 创建静态内部类	244
12.3.2 在外部类中访问静态内部类	245
12.3.3 在外部类外访问静态内部类	246
12.4 匿名内部类	247
12.4.1 创建匿名内部类	247
12.4.2 匿名内部类的初始化	249

12.5 综合练习	250
12.6 小结	250
12.7 习题	250
第 13 章 多线程 ( 教学视频: 55 分钟)	252
13.1 多线程简介	252
13.2 定义线程和创建线程对象	252
13.2.1 继承 Thread 类定义线程	252
13.2.2 实现 Runnable 接口定义线程	253
13.3 运行线程	254
13.3.1 启动线程	254
13.3.2 同时运行多个线程	256
13.4 线程生命周期	257
13.4.1 新建状态	257
13.4.2 准备状态	257
13.4.3 运行状态	257
13.4.4 等待/阻塞状态	258
13.4.5 死亡状态	258
13.5 线程的调度	258
13.5.1 睡眠方法	258
13.5.2 线程优先级	260
13.5.3 yield 让步方法	261
13.5.4 join 让步方法	262
13.6 综合练习	264
13.7 小结	265
13.8 习题	265

第三篇 应用篇

第 14 章 Swing 桌面程序开发 ( 教学视频: 70 分钟)	268
14.1 开发第一个 Swing 程序	268
14.2 JFrame 窗口类	269
14.2.1 JFrame 窗口类简介	269
14.2.2 创建简单窗体	269
14.2.3 设置窗体	271
14.3 JPanel 面板类	273
14.3.1 容器介绍	273
14.3.2 JPanel 面板类简介	274
14.3.3 创建面板	274
14.4 JLabel 标签类	275

14.4.1 JLabel 标签类简介	275
14.4.2 创建标签	276
14.5 JButton 按钮类	276
14.5.1 JButton 按钮类简介	277
14.5.2 创建按钮	277
14.5.3 按钮动作事件	278
14.6 Swing 中的事件	280
14.6.1 事件简介	280
14.6.2 同一个事件源注册多个监听器	280
14.6.3 同一个监听器注册给多个事件源	282
14.6.4 窗体获取和失去焦点事件	283
14.6.5 窗体打开、关闭和激活事件	284
14.7 综合练习	286
14.8 小结	288
14.9 习题	288
第 15 章 布局管理器 ( 教学视频: 62 分钟)	290
15.1 流布局	290
15.1.1 流布局介绍	290
15.1.2 使用流布局	291
15.2 网格布局	293
15.2.1 网格布局介绍	293
15.2.2 使用网格布局	293
15.3 边框布局	295
15.3.1 边框布局介绍	296
15.3.2 使用边框布局	296
15.4 空布局	298
15.4.1 空布局介绍	298
15.4.2 使用空布局	298
15.5 卡片布局	299
15.5.1 卡片布局介绍	299
15.5.2 使用卡片布局	300
15.6 综合练习	302
15.7 小结	304
15.8 习题	304
第 16 章 Swing 常用控件 ( 教学视频: 90 分钟)	306
16.1 文本框及密码框和多行文本框	306
16.1.1 创建文本框	306
16.1.2 创建密码框	307

16.1.3 创建多行文本框	309
16.2 复选框和单选按钮	310
16.2.1 创建单选按钮	310
16.2.2 创建复选框	313
16.3 选项卡	315
16.3.1 选项卡介绍	315
16.3.2 创建选项卡	315
16.4 分隔窗格	317
16.4.1 分隔窗格介绍	317
16.4.2 创建分隔窗格	317
16.5 滑块和进度条	319
16.5.1 创建滑块	319
16.5.2 创建进度条	320
16.6 列表框	323
16.6.1 列表框介绍	323
16.6.2 创建列表框	324
16.6.3 下拉列表框	326
16.7 菜单	328
16.7.1 菜单介绍	328
16.7.2 创建菜单	329
16.7.3 创建弹出式菜单	330
16.8 综合练习	330
16.9 小结	332
16.10 习题	333
第 17 章 JDBC 数据库编程 ( 教学视频: 63 分钟)	335
17.1 数据库基本介绍	335
17.1.1 数据库介绍	335
17.1.2 数据库应用架构	335
17.1.3 数据库模型	336
17.2 JDBC 数据库编程介绍	336
17.2.1 JDBC 和 ODBC 的关系	337
17.2.2 为什么使用 JDBC 数据库编程	337
17.3 SQL 数据库操作技术	338
17.3.1 什么是 SQL	338
17.3.2 如何进行 SQL 操作	338
17.4 创建数据库	339
17.4.1 创建 Access 数据库	339
17.4.2 创建 SQL Server 数据库	339
17.5 JDBC 编程步骤	342

17.5.1 创建数据源	342
17.5.2 加载驱动程序	344
17.5.3 建立数据库连接	345
17.5.4 进行数据库操作	345
17.5.5 获取数据库中信息	346
17.5.6 JDBC 数据库编程实例	347
17.6 事务处理	348
17.6.1 事务介绍	348
17.6.2 进行事务操作	349
17.7 综合练习	351
17.8 小结	352
17.9 习题	352
第 18 章 Java 中输入/输出流 ( 教学视频: 55 分钟)	353
18.1 I/O 流简介	353
18.1.1 什么是 I/O 流	353
18.1.2 节点流与处理流	353
18.1.3 字节流与字符流	354
18.1.4 抽象基类	354
18.2 使用流进行文件操作	356
18.2.1 使用 File 类进行文件与目录操作	356
18.2.2 FileInputStream 类与 FileOutputStream 类	359
18.2.3 FileReader 类与 FileWriter 类	362
18.3 综合练习	364
18.4 小结	364
18.5 习题	364
第 19 章 集合框架 ( 教学视频: 65 分钟)	366
19.1 集合框架总论	366
19.1.1 什么是集合框架	366
19.1.2 Collection 接口	366
19.2 列表	367
19.2.1 List 列表接口	367
19.2.2 Vector 类	368
19.2.3 ArrayList 类	371
19.2.4 LinkedList 类	373
19.3 集合	376
19.3.1 Set 接口	376
19.3.2 SortedSet 接口	377
19.3.3 TreeSet 类	378

19.3.4 HashSet 类	380
19.4 映射	381
19.4.1 Map 接口	381
19.4.2 HashMap 类	382
19.4.3 TreeMap 类	384
19.5 综合练习	385
19.6 小结	386
19.7 习题	387
第 20 章 网络编程 ( 教学视频: 58 分钟)	389
20.1 网络编程基础	389
20.1.1 TCP/IP 协议	389
20.1.2 网络编程模型	389
20.1.3 网络传输协议	390
20.1.4 端口和套接字	390
20.2 基于 TCP/IP 协议的网络编程	391
20.2.1 Socket 套接字	391
20.2.2 ServerSocket 类	391
20.2.3 Socket 类	392
20.2.4 网络编程 C/S 架构实例	393
20.3 综合练习	396
20.4 小结	398
20.5 习题	398

第四篇 综合案例篇

第 21 章 学生管理系统 ( 教学视频: 54 分钟)	399
21.1 系统设计	399
21.2 数据库设计	399
21.3 登录界面开发	400
21.3.1 界面设计	400
21.3.2 程序开发	400
21.4 学生界面开发	402
21.4.1 界面设计	402
21.4.2 程序开发	402
21.4.3 开发插入学生界面	403
21.4.4 查询学生信息界面	406
21.4.5 查询成绩信息	409
21.5 综合练习	412
21.6 小结	413

第1章 Java简介

本章将介绍 Java 的特点及所应用的平台，然后带领读者从第一步做起，完成一个 Java 的小程序。通过这个小程序，可以让读者了解 Java 平台的搭建及简单的开发步骤。通过本章的学习，读者应该能够实现如下目标。

- 熟练掌握如何搭建 Java 开发环境，包括下载、安装和配置 JDK；
- 能够编写和编译 Java 程序，并能够运行生成文件。

这些是学习本章的目标，同时也是对读者的基本要求。本章内容是后面知识的基础，读者一定要熟练地掌握本章的知识。



1.1 Java 的平台简介

Java 语言在网络编程方面应用得很广，作为一个新的程序设计语言，它具有简单、多变、面向对象、不依赖操作系统的特点，具有很好的移植性和安全性，这些特点给网络编程带来了很多便利。Java 的平台根据用途来区分，可以分为三个版本。

- Java SE——Java Standard Edition，这是 Java 的标准版，主要用于桌面级的应用和数据库开发。
- Java EE——Java Enterprise Edition，这是 Java 的企业版，提供了企业级开发的各种技术，主要用于企业级开发，现在用的最多的也就是这个平台。
- Java ME——Java Micro Edition，这个版本的 Java 平台主要用于嵌入式和移动式的开发，最常用的就是手机应用软件开发。

Java 作为一门优秀的编程语言，相对于其他类似语言具有一定的优越性，这是由 Java 语言的如下特点决定的。

- Java 语言具有简单、面向对象、分布式、安全、可移植、多态等特点。
- Java 语言是一种面向对象的语言，通过理解一些重要的概念就能编写出各种功能的代码。
- Java 语言主要集中用在设计类和接口功能方面，提供了继承及多态的机制，对类中的成员变量和方法可进行覆盖和重写，实现了代码的重复使用，使代码编写更简单。
- Java 语言不支持指针，所有的访问，都必须通过具体的对象变量来实现，这样既保护了对象的私有变量，同时也避免了一些错误。
- Java 语言编译产生的文件是字节码文件，字节码文件和平台无关。



提示：Java 具有很多重要的特点，也是 Java 成为一门流行编程语言的原因。



1.2 安装工具包

学习 Java 程序开发，首先要进行 Java JDK 的安装，JDK 就是提供 Java 服务的系统包。请读者根据操作系统来选择安装哪个版本的 JDK。本节将介绍如何安装 JDK 和配置 JDK 的环境变量，并讲解一些常用命令。

1.2.1 下载 JDK

Java 的系统包即 JDK，JDK 的全称为 Java Development Kit，可以根据不同的操作系统平台来下载不同的 JDK，下面介绍在 32 位的 Windows XP 系统上，如何下载并完成配置。

① 打开 IE 浏览器，并在地址栏输入“<http://java.sun.com/>”，按回车键，如图 1-1 所示。



图 1-1 进入 Java 官方网站

② 在页面里打开菜单“Downloads”，并选择其中菜单项“Java EE”，如图 1-2 所示。

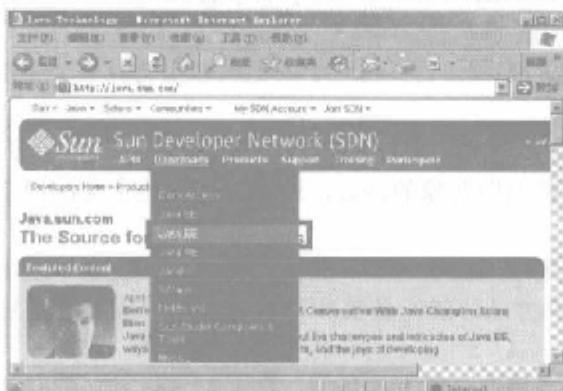


图 1-2 选择菜单项 Java EE 准备进行下载

③ 进入 Java EE 页面后，单击“Download”按钮进行下载，如图 1-3 所示。

④ 根据机器的操作系统平台，选择不同的 JDK 版本，这里选择“Windows”，并选中“*I agree to the Java EE 5 SDK Update 4 License Agreement*”复选框后，单击“Continue”按钮继续前进，如图 1-4 所示。

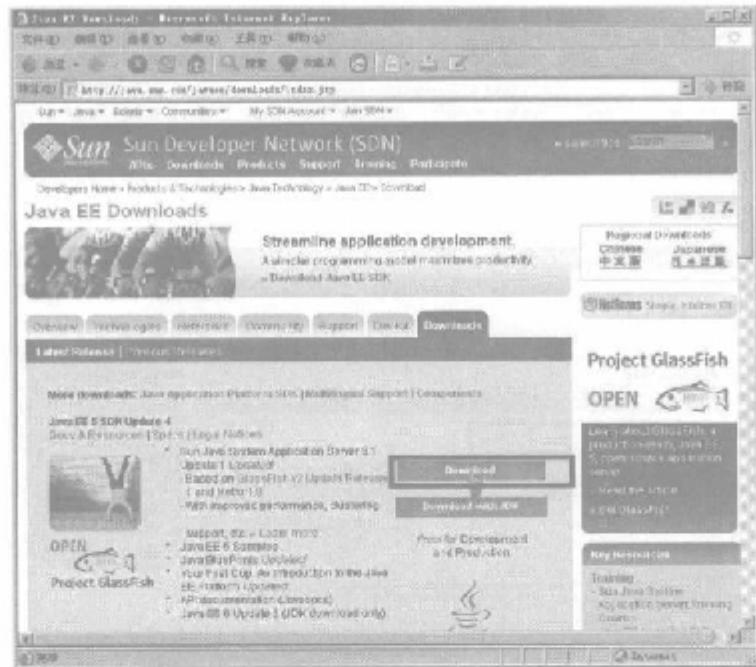


图 1-3 单击“Download”按钮后选择平台种类

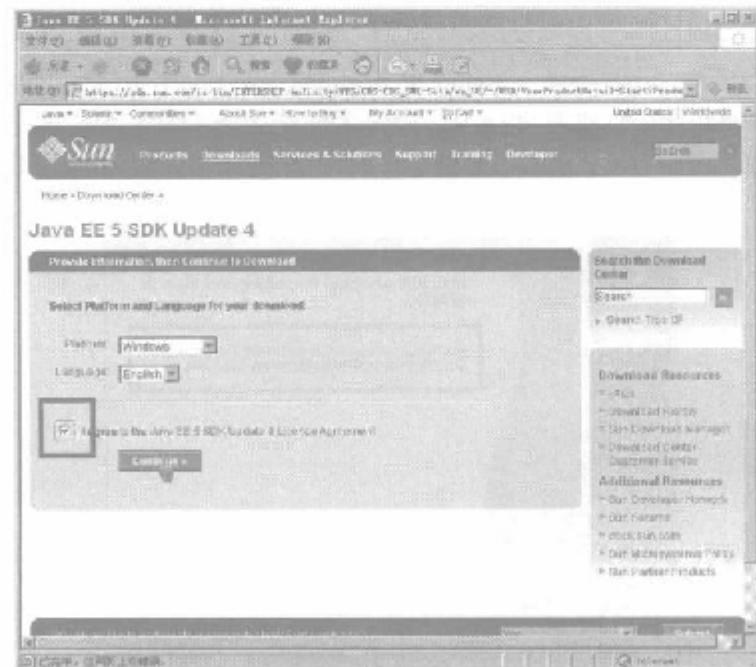


图 1-4 单击 Continue 按钮出现具体的要下载的文件

⑤ 单击“java_ee_sdk-5_04-windows-nojdk.exe”链接进行下载，因为文件比较大，推荐使用下载工具进行下载，如图 1-5 所示。

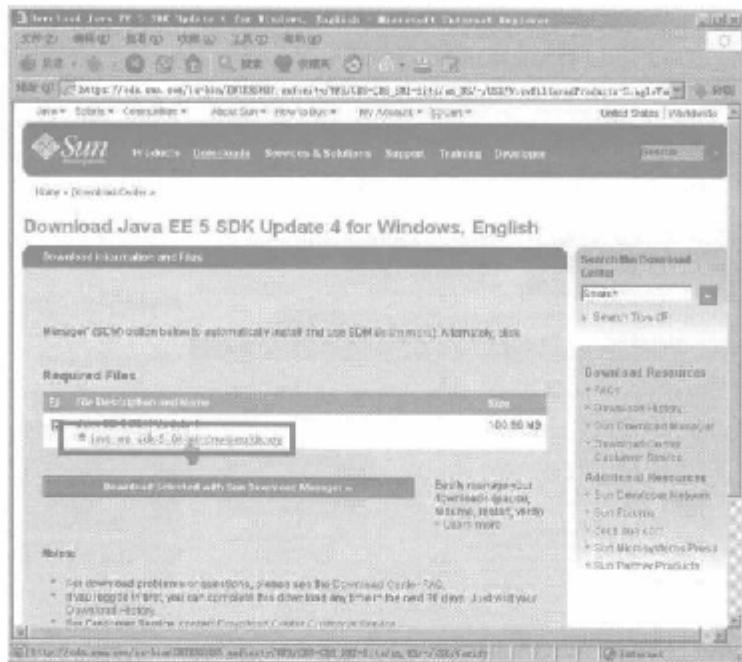


图 1-5 单击“java_ee_sdk-5_04-windows-nojdk.exe”链接进行下载

1.2.2 安装 JDK

下载完成后，便可进行安装，下面介绍 Windows XP 下的 JDK 安装步骤。

- ① 找到安装程序后，双击 jdk-6-windows-i586.exe，运行安装程序进行安装。
- ② 安装程序正在初始化所要安装的文件，稍等后初始化完成，完成后的界面如图 1-6 所示。

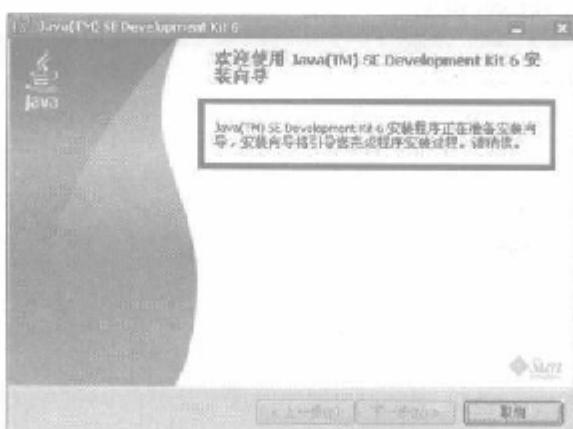


图 1-6 安装 JDK 的开始界面

- ③ 选择接受许可证协议，继续安装，如图 1-7 所示。
- ④ 选择要安装的组件，单击“下一步”按钮继续安装，一般选择默认组件直接进入下一步，如图 1-8 所示。

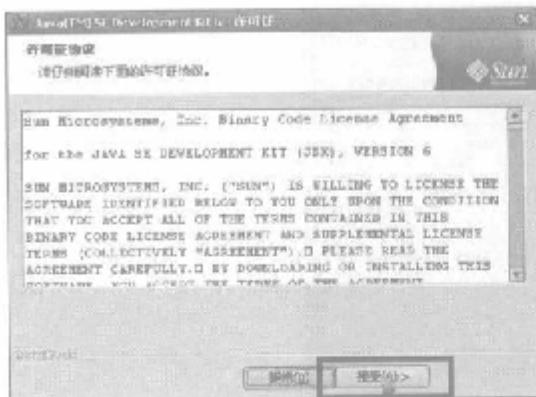


图 1-7 单击“接受”按钮继续安装

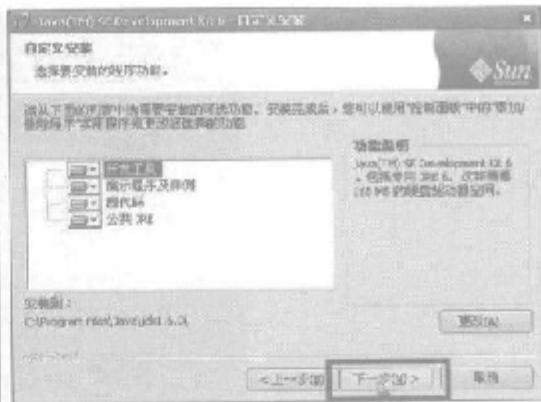


图 1-8 选择要安装的组件，单击“下一步”按钮

⑤ 正在进行安装，需要稍等，如图 1-9 所示。

⑥ 显示安装完成界面，至此 JDK 安装完成，如图 1-10 所示。

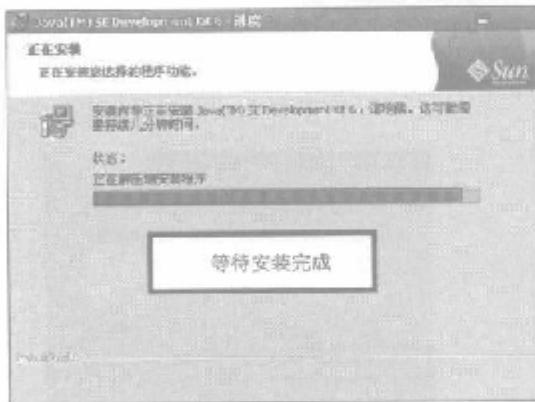


图 1-9 安装程序正在安装，请稍等

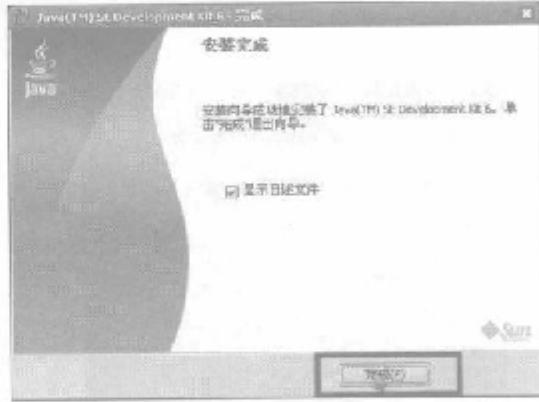


图 1-10 JDK 安装完成，单击“完成”按钮结束安装

1.2.3 查看与设置环境变量

所谓环境变量，是供系统内部使用的变量，包含系统中的当前系统用户的环境信息的字符串和软件的一个确定存放的路径，安装完 JDK 后就必须配置环境变量，方法如下所述。

① 在桌面上右键单击“我的电脑”图标，在出现的菜单中选择“属性”选项，出现的界面如图 1-11 所示。

② 选择“高级”选项，如图 1-12 所示，并单击“环境变量”按钮，弹出“环境变量”对话框，如图 1-13 所示。

③ 在“系统变量”对话框里找到变量名“Path”，并选中，再单击“编辑”按钮进入“编辑系统变量”对话框，如图 1-14 所示。

④ 在所有的变量值前面输入“C:\Program Files\Java\jdk1.5.0_08\bin;”，注意有分号，路径根据情况进行设置，如图 1-15 所示。

⑤ 在“系统变量”对话框里单击“新建”按钮，弹出“新建系统变量”对话框，如图 1-16 所示。



21 天学通 Java

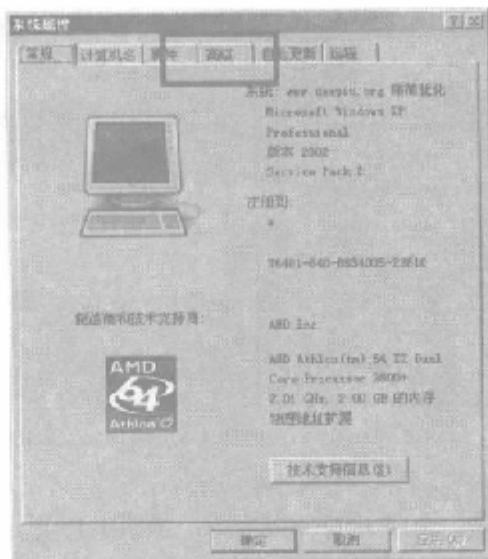


图 1-11 系统属性

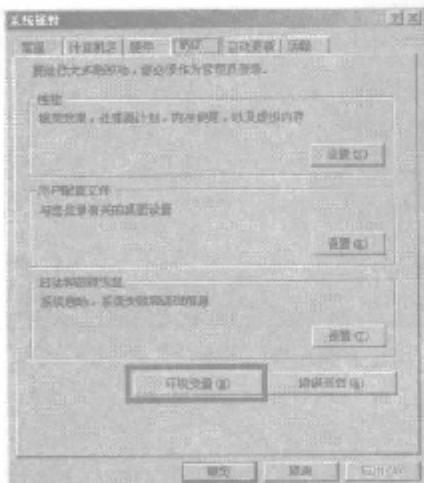


图 1-12 选择“高级”选项



图 1-13 “环境变量”对话框



图 1-14 “编辑系统变量”对话框



图 1-15 编辑系统变量

- ⑥ 设置变量名为“classpath”，变量值为“.;C:\Program Files\Java\jdk1.5.0_08\lib\dt.jar;C:\Program Files\Java\jdk1.5.0_08\lib\tools.jar”，路径根据情况进行设置，如图 1-17 所示。
- ⑦ 最后单击“确定”按钮退出，完成安装。

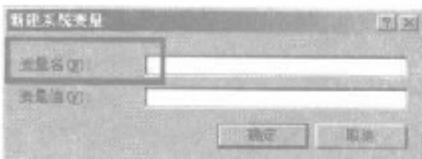


图 1-16 “新建系统变量”对话框



图 1-17 填入新值

配置完环境后，需要测试是否配置正确。下面给出了详细的测试步骤，按照这个步骤操作可以很轻松地完成测试。

- ① 单击“开始”按钮，在弹出的开始菜单中再单击“运行”菜单项，将出现“运行”对话框。在“运行”对话框里输入“cmd”，并单击“确定”按钮，如图 1-18 所示。

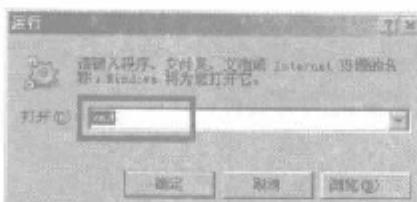


图 1-18 “运行”对话框

- ② 在命令提示符窗口里输入“javac”和“java”，观察是否出现javac或java的用法提示，如图1-19所示。

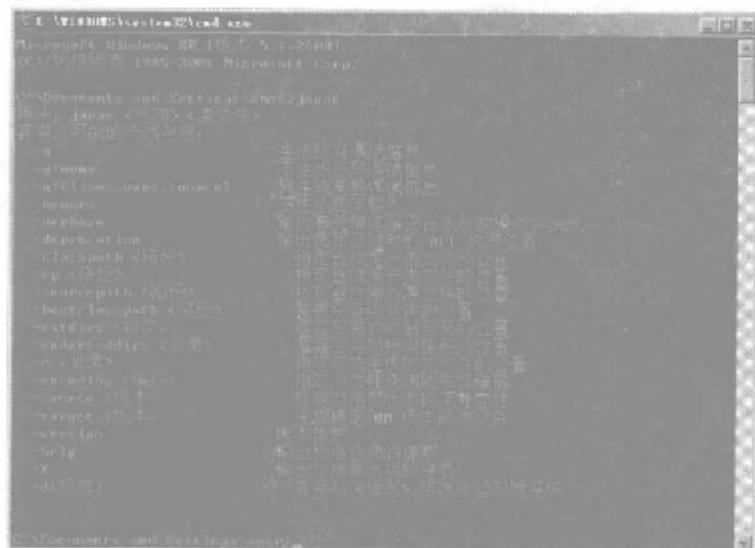


图 1-19 配置正确的环境变量信息

1.2.4 JDK 常用命令

在图 1-19 中显示了 JDK 中的部分命令，本节将对这些命令进行必要的讲解。

1. javac 的常用命令

- -g: 生成调试信息。



- `-g:none`: 生成无调试信息。
- `-g:{lines,vars,source}`: 只生成部分调试信息。
- `-O`: 优化, 可能增大类文件。
- `-nowarn`: 无警告。
- `-verbose`: 输出编译器信息。
- `-deprecation`: 输出不鼓励使用的 API 的程序路径。
- `-classpath + 路径`: 指定用户类文件的路径。
- `-sourcepath + 路径`: 指定输入源文件的路径。
- `-bootclasspath + 路径`: 覆盖自举类文件的路径。
- `-extdirs + 目录`: 覆盖扩展类的路径。
- `-d + 目录`: 指定输出类文件的路径。
- `-encoding + 编码`: 指定源文件中的字符集编码。
- `-target + 版本`: 生成虚拟机的类文件。

2. JDK 的常用命令

- `native2ascii`: 将中文 Unicode 码转换成 ASCII 码, `-reverse` 参数可以将 ASCII 码转换回来。
- `javap`: 将 class 反编译成 Java bytecodes。
- `jdb`: Java 的 debug 工具。
- `jps`: 查看 JVM 进程信息用的。
- `keytool`: 生成 keystore 文件。
- `jar`: 可将多个文件合并为单个 JAR 文件, jar 是个多用途的压缩工具, 它基于 ZIP 和 ZLIB 压缩格式。
- `javadoc`: Javadoc 解析 Java 源文件中的声明和文档注释, 并产生相应的 HTML 页面, 描述公有类、保护类、内部类、接口、构造函数、方法。在实现时, javadoc 要求且依赖于 Java 编译器完成其工作。

1.2.5 Java 各个目录含义

JDK 安装完成后, 在安装目录下会安装很多目录和文件。下面对这些目录进行简单的介绍。分类及说明如表 1-1 所示。

表 1-1 各目录含义

目 录	含 义
bin	JDK 的基本程序都在这里, 如果 javac、java、javadoc 等
demo	和字面意思一样, Java 自带的一些例子程序
jre	Java 运行时的环境都在这里
lib	Java 的类库
src	Java 类库的源代码

1.2.6 要善于使用 JDK 的帮助文件

JDK 的帮助文件有在线版本和离线版本两种, 可以从 Java 的官方网站 <http://java.sun.com>。

com 上下载到最新的 JDK 帮助文件，帮助文件分为两种格式，即 HTML 格式和 CHM 格式。JDK 的帮助文件使用很简单，只需要打开目录下的 index.html 即可。若想查找某个方法是怎么实现的，只需根据包的路径找到此方法就行了。Java EE5 帮助文件的路径和界面如图 1-20 所示。

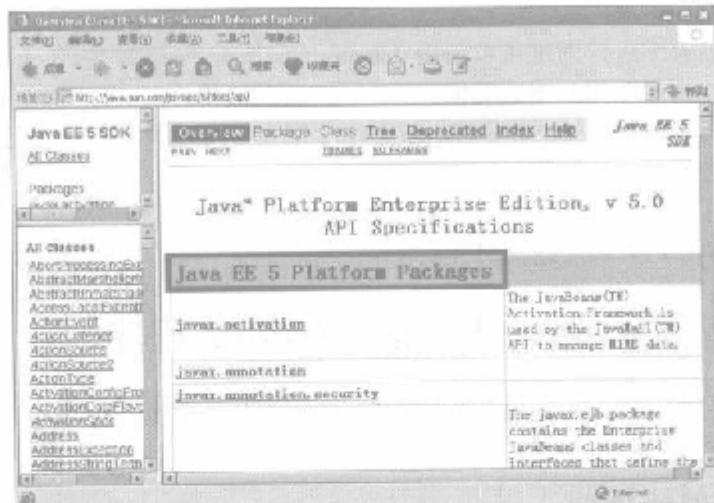


图 1-20 Java EE 5 帮助文件的界面



1.3 程序开发过程

安装好 JDK 及配置好环境变量以后，就可以进行 Java 程序的开发。因为 Java 是一种编译型语言，所以在程序开发过程方面和其他语言有所不同。要开发 Java 程序，要经过以下三个步骤。

- ① 创建一个源文件。Java 源文件就是 Java 代码文件，以 Java 语言编写。Java 源文件是纯文本文件，扩展名为“.java”。可以使用任何文本编辑器来创建和编辑源文件，本书使用 Windows 系统自带的记事本作为 Java 源文件的编辑器。
- ② 将源文件编译为一个.class 文件。使用 JDK 所带的编译器工具 javac.exe，它会读取源文件并将其文本编译为 Java 虚拟机能理解的指令，保存在以后缀.class 结尾的文件中。包含在 CLASS 文件中的指令就是众所周知的字节码（bytecodes），它是与平台无关的二进制文件，执行时由解释器 java.exe 解释成本地机器码，边解释边执行。
- ③ 运行程序。使用 Java 解释器（java.exe）来解释执行 Java 应用程序的字节码文件（.class 文件），通过使用 Java 虚拟机来运行 Java 应用程序。

对 Java 的程序开发过程有所了解后，在第 1.5 节中将按照这个程序开发过程来开发一个最简单的 HelloWorld 程序。



1.4 编码规范

Java 程序要按照 Java 编码规范来进行编写。一个程序不按照编码规范编写可能也是能



够运行的，但是不按照编码规范编写的程序不是一个好程序，这种程序不易于程序的查看和维护。

编码规范包括很多内容，例如代码的编写规则，命名规则，代码注释等多项内容。命名规范和代码注释将在下一章中结合数据类型进行讲解。在本节中，主要讲解一下代码的编写规则。

- 代码必须有缩进，缩进可以使用 Tab 键，或者 4 个空格。因为 4 个空格在 Eclipse 中默认作为一个 Tab 缩进单位。
- 每行代码不要超过 80 个字符，这是由于很多编写工具不能对一行超过 80 个字符的内容进行很好的解释。
- 当代码在一行中放不下时，应进行换行。但是换行不能自动换行，而应该按照级别进行换行，并且同级别对齐。



1.5 HelloWorld：第一个 Java 程序

JDK 安装完毕，环境变量也配置完毕后，下面开始编写第一个 Java 程序，并讲解编译和运行程序的方法。

1.5.1 编写程序代码

打开文本文件编辑器，如 Windows 的记事本，也可使用更高级的编写工具。如 Eclipse、JBuilder、NetBeans 等，这些工具具有更加强大的功能，但现在不推荐使用，不利于初学者打下良好的编程基础。首先，在记事本里添加如下代码。该代码可以直接从光盘中复制到记事本中，当然亲自动手输入是最好的。

【范例 1-1】 使用记事本编写的程序如下所示。

示例代码 1-1

```
//定义一个类名称为 HelloWorld
public class HelloWorld
{
    //类的主入口函数
    public static void main(String args[])
    {
        //System.out.println 为打印语句，用来显示结果
        System.out.println("欢迎使用 Java 来编写程序！");
    }
}
```

【代码解析】 即使读者还没有学习如何编写程序，但是通过讲解还是可以使读者理解的。在本程序中，首先定义了一个类，类的名称为 HelloWorld。在这个类里有一个 main 方法，这是 Java 程序的入口，只要能执行的程序都有这个方法。System.out.println 方法能执行打印操作，还能打印其他类型的数据。

在编写后把这个文本文件保存为 HelloWorld.java，并注意大小写问题。

1.5.2 编译程序代码并运行

编写完 Java 程序的源代码后就可以对该程序进行编译，Java 程序源代码的编译有如

以下几个步骤。

- ① 单击“开始”按钮，在菜单中选择“运行”菜单项，将出现“运行”对话框。在“运行”对话框里填写“cmd”，并单击“确定”按钮。
- ② 在命令行提示符下进入源代码文件的存放目录。
- ③ 输入命令“javac HelloWorld.java”，并按回车键，如图 1-21 所示。

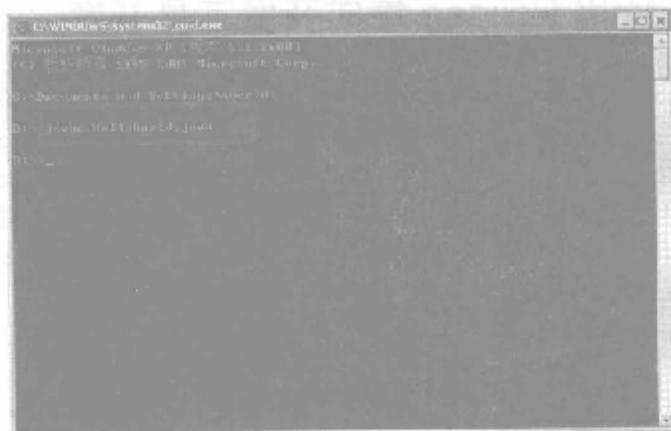


图 1-21 编译 HelloWorld.java

- ④ 编译成功后，在程序源代码的目录里会出现文件 HelloWorld.class，这是 Java 编译的字节码文件，如图 1-22 所示。
- ⑤ 在命令行提示符下使用命令运行程序，如图 1-23 所示。

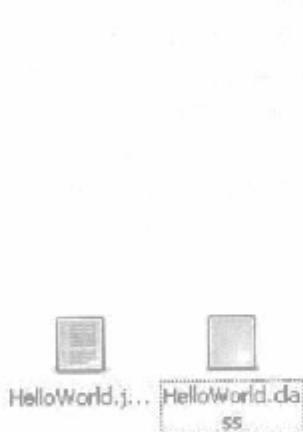


图 1-22 生成的 HelloWorld.class



图 1-23 程序 HelloWorld 的运行结果

1.5.3 注意事项

在编写、编译和运行 Java 程序时有很多注意点，这也是初学者需要注意的地方。

- 在运行时如果提示“java.lang.NoClassDefFoundError”的话，请查找环境变量是否设置正确。
- 在命令提示符下输入命令的时候要注意区分大小写，Java 是区分大小写的。



- 用 javac 编译程序时是有扩展名的。
- 用 java 运行程序时是没有扩展名的。
- 源程序里要有 main 方法。
- 源程序里的类名要和文件名相同，包括大小写。



1.6 使用 Eclipse 集成开发工具开发

Eclipse 是目前最流行的 Java 开发工具，Eclipse 中集成了许多工具和插件，从而使 Java 程序开发更容易。Eclipse 是一个可以免费使用的软件，读者可以从 Eclipse 的官方网站 <http://www.eclipse.org> 上下载。下载后解压缩就可以使用，直接下载的 Eclipse 是英文版，可以下载中文语言包 [NLpack1-eclipse-SDK-3.2.1-win32.zip](#) 从而完成中文版 Eclipse 的安装。

下载和安装 Eclipse 后，就可以使用该集成工具了，双击 `eclipse.exe` 文件可以运行 Eclipse。由于篇幅原因，这里不对 Eclipse 界面进行介绍，读者可以自己熟悉一下 Eclipse 界面内容。这里主要讲解如何在 Eclipse 中进行第一个 `HelloWorld` 程序开发。开发步骤如下所示。

- ① 打开 Eclipse，选择菜单栏“文件”命令，再选择级联菜单“新建”命令，最后选择“项目”子菜单，在弹出的对话框中选择 Java 项目，并单击“下一步”按钮。
- ② 输入项目名称，例如 `FirstProject`；在“内容”选项卡中选择“从现有资源创建项目”，然后在目录中找到前面 `HelloWorld.java` 的路径。单击“完成”按钮完成项目的创建。
- ③ 在“包资源管理器”中单击右键，弹出邮件菜单，选择“新建”→“包”菜单，在弹出对话框的“名称”文本框中输入包名，这里输入“`FirstBao`”。
- ④ 打开 Java 编写界面，输入 `HelloWorld` 程序。单击运行按钮，就会在控制台窗口中输出“`HelloWorld`”内容。这样一个 Java 程序就在 Eclipse 工具中编写、编译和运行完成。

从 Eclipse 开发中可以看到，和在记事本中开发不同的是，缺少编译过程，这是因为在 Eclipse 中可以进行自动编译。Eclipse 有一个增量编译器，每次保存一个 Java 文件时它就自动进行编译。这个特性被称做“自动构建”。如果不需要这个功能，可以进行“窗口”→“首选项”→“工作台”→“对资源修改执行自动构建”操作，关闭这个特性。



1.7 综合练习

编写一个输出“我终于会 Java 了”的程序。

【提示】参考本章中编写的第一个 Java 程序来进行程序编写，代码如下所示。

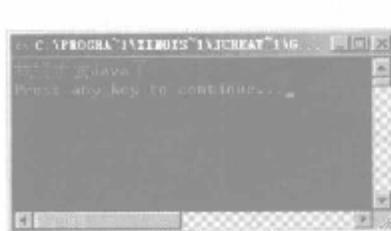


图 1-24 输出相应内容

```
public class LianXil
{
    public static void main(String args[])
    {
        System.out.println("我终于会 Java 了");
    }
}
```

【运行结果】使用 `javac` 编译程序，然后使用 `java` 运行编译产生的 `class` 程序。运行结果如图 1-24 所示。



1.8 小结

本章介绍了 Java 的各种平台, 以及如何下载、安装 JDK, 设置环境变量等, 这些是编写 Java 程序的基础知识; 本章接着通过一个小例子说明 Java 程序是如何编写的。在本章学习中, 读者应重点掌握如下内容。

- JDK 的安装与配置。
- 如何编写、编译和运行 Java 程序。



1.9 习题

一、填空题

1. 在开发 Java 程序前, 需要进行的工作包括_____、_____、_____。
2. Java 的平台根据用途来区分, 可以分为_____、_____、_____三个版本。
3. 开发一个 Java 程序的步骤包括_____、_____、_____。

二、选择题

1. 对编写后的 Java 程序进行编译, 编译后的扩展名为()。
A. .java B. .class C. .txt
2. 对编译后的 class 文件运行, 需要的命令为()。
A. java B. javac C. javap

三、简答题

1. Java 作为一门优秀的编程语言, 它有哪些特点? 这些特点是和其他语言有什么不同?
2. 简述 Java 程序的开发过程。

四、编程题

在一台没有 Java 开发环境的计算机中, 在 15min 内开发出一个输出 “OK” 的 Java 程序。

第2章 Java 的基本数据类型

本章将首先介绍 Java 的基本数据类型,如整型、浮点型等,以及它们之间的转换,最后介绍标识符的命名规则。通过本章的学习,读者应该能够实现如下目标。

- 了解 Java 有哪些基本数据类型。
- 掌握各种数据类型的基本含义。
- 学会如何进行数据类型转换。
- 了解标识符和保留字等基本概念。
- 了解如何在 Java 程序中进行注释。



2.1 数据类型

所谓数据类型,就是对数据的抽象描述,Java 基本数据类型一共有 8 种,如 int 表示整型, float 表示浮点类型,下面将针对部分类型做详细介绍。

2.1.1 整型

整型是 Java 数据类型中的最基本类型,使用 int 表示。所调整型,就好比日常生活中的十进制数,是没有小数点的数据。在 Java 里整型是有符号的,且有正负之分。如-10、20。

Java 整型数据可以使用三种进制的数来表示,下面就对这三种进制进行介绍。

1. 十进制

十进制数在日常生活中最常见,大家天天都在用,Java 里定义一个十进制数的代码如下:

```
//int 为基本数据类型,是最常用的基本数据类型  
//正的十进制数  
int i = 11;  
//负的十进制数  
int j = -12;
```

2. 八进制

八进制数的进制规则是满 8 进 1,包含 0~7 的 8 个数字,在整数前面添加一个“0”就表示为八进制数。

```
//正的八进制数  
int i = 08;  
//负的八进制数  
int j = -12;
```

3. 十六进制数

十六进制数的进制规则是满 16 进 1,包含 0~9, a~f 的 16 个数字,在整数前面添加一个“0x”表示十六进制数。各个类型的取值范围如表 2-1 所示。

```
//正的十六进制数
int i = 0x05;
//负的十六进制数
int j = -0xffaa;
```

表 2-1 整型的取值范围

类 型	bit	byte	取值范围
byte	8	1	-27 ~ 27-1
short	16	2	-215 ~ 215-1
int	32	4	-231 ~ 231-1
long	64	8	-263 ~ 263-1

 **提示：**区分十进制、八进制和十六进制的方法就是通过数字的前面部分来判断的。如果以“0X”或“0x”开头，则该数肯定为十六进制数。如果以“0”开头，则该数肯定为八进制数。

2.1.2 浮点型

浮点型同样也是 Java 数据类型中的基本类型，整型表示整数，浮点型则表示小数。所谓浮点类型，就好比日常生活中的十进制数加上小数点。在 Java 里浮点类型是有符号且有正负之分的。

1. float：单精度浮点数

声明为 float 类型的浮点数时，要在结尾加 F 或 f，浮点类型默认的类型是 double。

```
//正的浮点数
float i1 = 11.11F;
//负的浮点数
float j2 = -17.15f;
```

2. double：双精度浮点数

声明为 double 类型的浮点数时，要在结尾加 D 或 d。但声明为 double 类型时结尾的 D 和 d 可加可不加。这里建议是在 double 数据类型的数后面加上 D 或 d，以便更能够和单精度浮点数区分。浮点型取值范围如表 2-2 所示。

```
//正的浮点数
double i1 = 11.11223D;
//正的浮点数
double i2 = 11.11333d;
//负的浮点数
double j3 = -17.15555;
```

表 2-2 浮点类型的取值范围

类 型	bit	byte	取值范围
float	32	4	+3.40282347E+38F
double	64	8	+1.79769313486231570E+308

 **注意：**如果浮点型数据的结尾处没有字母，则该数据为一个双精度浮点数。



2.1.3 字符型 (char)

字符型是一种表示字符的数据类型。char 型表示一个字符，16 位，占用两个字节。一般一个 char 型数值只用来表示一个字符，用“‘’”单引号来表示。例如下面的例子。

```
//表示一个字符  
char c1 = 'c';  
//表示一个unicode 码  
char c2 = '\u005E';  
//表示一个整数  
char c3 = 56;
```

Java 中还有一种特殊的字符型数值，那就是转义字符。有一些特殊符号是不能通过一般字符来进行显示的，例如换行符和制表符。在表 2-3 中列出了 Java 中比较常用的转义字符。

表 2-3 常用转义字符

转义字符	名 称	unicode 码
\n	换行	\u000a
\r	回车	\u000d
\b	退格	\u0008
\t	制表符	\u0009
\"	双引号	\u0022
\'	单引号	\u0027
\	反斜杠	\u005c



说明：转义字符是一种特殊的形式，在一些情况下是不可缺少的。例如后面将要学习的多行文本框，提交时必须使用转义字符进行操作。

2.1.4 布尔型 (boolean)

布尔型是一种起到判断作用的数据类型。boolean 类型的取值非常简单，就好比日常生活中的真与假，在 Java 中，使用 true 与 false 表示真与假。例如下面的例子。

```
boolean b1 = false;  
boolean b2 = true;
```



2.2 数据类型间的转换

在日常生活中，斤和两都是重量单位，一斤可以转换为十两。在 Java 中，整型、浮点型等都是基本的数据类型，它们是能够进行数据类型转换的。下面介绍不同数据类型之间的数据转换方式。

2.2.1 自动转换

自动转换就是不需要明确指出所要转换的类型是什么，而是由 Java 虚拟机自动转换。转换的规则就是小数据类型变大数据类型，但大数据类型的数据精度有时会被破坏。下面



看一段代码。

```
//定义各种数据类型
int i = 123;
char c1 = '2';
char c2 = 'c';
byte b = 2;

//自动转换的数据类型
int n = b;
long l = i;
```

请读者思考把 byte 类型转换成 float 类型时, 为什么会影响数据的精度?

2.2.2 强制转换

强制转换是带有强制性的, 明明不能自动转换, 而强制性地进行转换。看下面的例子。

```
//定义数据类型
int i = 22;
long L = 33;

//强制转换数据类型
char c = (char)i;
int n = (int)L;
```

其中, i 原来是一个 int 整型, 但要将它强行转换成 char 字符型。同样 L 原来是一个 long 型, 但要将它强行转换成 int 整型。通过前面的学习已经知道, long 型的取值范围最大值可以为 2 的 63 次方减 1, 而 int 型的取值范围最大值只有 2 的 31 次方减 1。所以如果 L 为一个大于 2 的 31 次方减 1, 在强制类型转换时就会丢失精度, 使数值发生变化, 这也是读者需要注意的地方。



注意: 强制类型转换是会丢失精度的, 经常会发生转换后数据发生变换的情况。但是在一些必要的地方又必须进行强制类型转换。所以要谨慎和准确地使用强制类型转换。

2.2.3 隐含转换

隐含转换和自动转换很相似, Java 虚拟机根据数据类型的位数来判断此数据类型是否能装载此数据, 如果能, Java 就默认进行了转换。举例说明如下:

```
//例子1
byte b = 111;

//例子2
int i = 222;
byte c = (byte) i;
```

上面语句中有两个转换, 一个是 111 转换成 byte 类型的数据库, 因为 byte 类型的数据位数能装载下 111, 所以能进行转, 这就是隐含转换。把值为 222 的变量 i 转换成 byte 类型的变量 c 就不能进行隐含转换, 因为能进行隐含转换的只能是常量而不能是变量。



2.3 标识符的命名

在 Java 里，方法名、类名、成员变量名都是标识符。所谓标识符，就好比日常生活中一个物品的名称一样，是一个代号，用来表示该物品。命名标识符的好处就是让外人看了，一下就能了解这个标识符的用途。下面介绍怎样命名标识符。

2.3.1 标识符的命名规则

标识符要以英文字母开头，由英文字母或数字组成，其他的符号不能出现在标识符里。标识符具体说明如下所述。

- 英文字母是大写的 A~Z，小写的是 a~z，以及“_”和“\$”。
- 数字包括 0~9。
- 其他的符号不能用在标识符里。
- 不能用 Java 所保留的关键字。
- Java 标识符是大小写敏感的。



说明：符合标识符的命名规则并不是一种最好的命名方法。给一个标识符命名，首先要符合命名规范，还要负责特点含义。

2.3.2 代码演示如何定义标识符

在本节中，将演示什么是正确，什么是错误的标识符。

```
int i = 22;
int I = 33;
char 2i = 23;
float float = 3f;
```

代码说明：

- 整型 i 和整型 I 为两个不同的标识符，因为在 Java 里标识符是区分大小写的。
- 2i 标识符的第一个字母为数字，所以也不能为正确的标识符。
- float 为 Java 保留的关键字，关键字不能用在标识符里，而是另有用途的。

2.3.3 不好的标识符命名

一个良好的标识符能体现此标识符所描述的方法、成员变量或类的含义。下面看例子。

【范例 2-1】示例代码 2-1 是一个不好的标识符命名程序。

示例代码 2-1

```
//定义了一个类a
public class a
{
    //定义成员变量
    int i = 2;
    char c = 3;
    //定义方法b
```



```

public void b()
{
    System.out.println("b");
}

//Java程序的主入口方法
public static void main(String args[])
{
    System.out.println("main");
}
}

```

【代码解析】定义一个类，名称为 a，在外人看来不知道这个 a 类所描述的是完成什么功能的类。成员变量 i 和 c 在外人看来也不知道具体完成什么功能。对方法 b 同样如此，不知道这样的表述能有什么好处。这样的标识符的命名是不合适的，不仅在外人看不明白，时间长了程序员自己也可能看不懂了。

2.3.4 良好的标识符命名

【范例 2-2】示例代码 2-2 是一个良好的标识符命名程序。

示例代码 2-2

```

//定义了一个类bike，所描述的是一个自行车
public class bike
{
    //定义成员变量
    String bike_color;    //自行车的颜色
    String bike_size;    //自行车的尺寸，即 26 或 28

    //描述的是自行车的基本信息
    public void bike_info()
    {
        System.out.println("自行车的组成与出厂信息");
    }

    //Java程序的主入口方法
    public static void main(String args[])
    {
        System.out.println("main");
    }
}

```

【代码解析】这样定义类名、成员变量名、方法名，在外人和自己看来，通过标识符名称就能知道类、成员变量和方法的功能。读者要养成良好的标识符命名习惯，这样既有利于自己，也帮助了别人。



2.4 关键字

所谓关键字，就好比日常生活中一个物品的标识，和人的名字很相似，具有特殊的含义。Java 保留了很多关键字，这些关键字都有其各自的用途。因此标识符是不能用这些关键字的。

Java 关键字在编写代码的时候是不能使用的，如果使用将会出现编译错误。Java 所保留的关键字如表 2-4 所示。



表 2-4 Java 关键字列表

abstract	boolean
break	byte
case	catch
char	class
continue	default
do	double
else	extends
false	final
finally	float
for	if
implements	import
instanceof	int
interface	long
native	new
null	package
private	protected
public	return
short	static
super	switch
synchronized	this
transient	true
try	void
volatile	while
const	goto



提示：关键字都是小写的，所以如果给出的为大写，肯定不是关键字。



2.5 代码注释

所谓注释，就好比在日常生活中听老师讲课所作的笔记，笔记的作用是解释知识点，帮助加强记忆。在 Java 中，程序中通常给出一些解释，可以提示某段代码的作用，这就是 Java 中的代码注释。注释的代码是不被编译的，所以不用担心执行效率的问题。

2.5.1 行注释

行注释就是一整行的注释信息，单行注释也是最常用的，行注释的符号是“//”，在注释符号后面一整行都被作为注释信息。例如下面的小程序。

【范例 2-3】示例代码 2-3 是一个进行单行注释的程序。



示例代码 2-3

```
public class HelloWorld
{
    //这是Java程序的入口方法
    public static void main(String args[])
    {
        System.out.println("欢迎使用Java来编写程序!");
    }
}
```

【代码解析】注释就是一整行的注释信息，行注释的符号是“//”，在注释符号后面一整行都被作为注释信息。



说明：单行注释是一种常用的注释，本书中大部分地方将使用这种注释方法，从而方便读者学习示例代码。

2.5.2 块注释

所谓块注释和行注释是一个意思，都是注释信息的意思，起到提示的作用。块注释的符号是“/* */”，以“/*”开始，以“*/”结束，在这个区域内的文字都将作为注释信息。例如下面的小程序。

```
/*
 *param name
 *author amer
 */
```

2.5.3 文档注释用户自定义类型

所谓文档注释是描述类的，通过在类里定义的文档注释，可以帮助程序员了解此类具有哪些功能，以及此类的相关信息。文档注释以“/**”开头，以“*/”结尾，把前面的例子加以修改得到如下示例。

【范例 2-4】例如下面的小程序。

示例代码 2-4

```
/**
 * 作者 amer
 */

public class HelloWorld
{
    public String name; //成员变量

    /**
     * Java 程序的主入口方法
     * 参数为 args
     */
    public static void main(String args[])
    {
        System.out.println("环境使用Java来编写程序!");
    }
}
```



【运行结果】通过在命令行下输入命令“javadoc HelloWorld.java”，来生成网页文档，如图 2-1 所示。



图 2-1 用命令 javadoc 来生成文档

命令执行完毕后，会在程序的源代码的同目录下生成和文档相关的 HTML 文件，如图 2-2 所示运行 index.html 打开文档，文档运行的界面如图 2-3 所示。

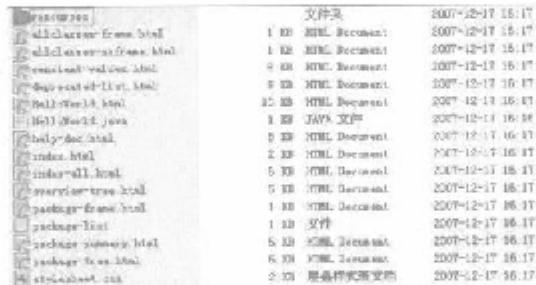


图 2-2 生成的 HTML 文件

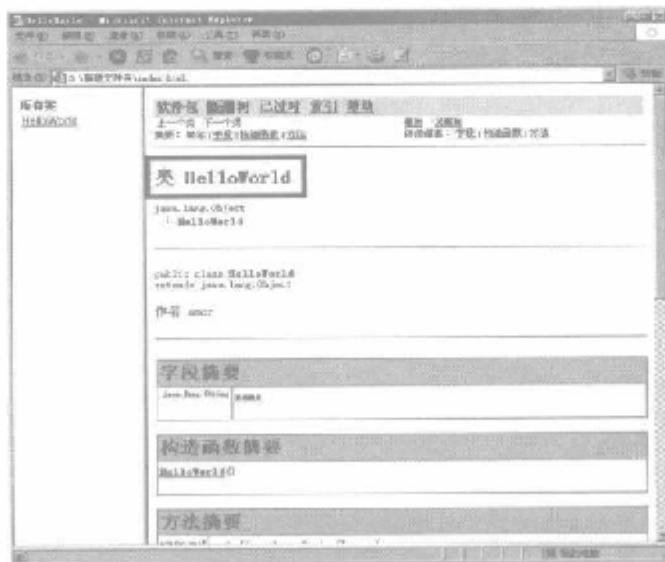


图 2-3 文档注释



2.6 综合练习

判断下面程序是否能够正常运行。

```
public class LianXii1
{
    public static void main(String args[])
    {
        int For=1; //定义一个变量名称为 For 的变量
        int Do=2; //定义一个变量名称为 Do 的变量
        int t=For+Do;
        System.out.println("变量和为"+t);
    }
}
```

提示：关键字都是小写的，大写后就不再为关键字。

【运行结果】使用 javac 编译程序，然后使用 java 运行编译产生的 class 程序，运行结果如图 2-4 所示。



图 2-4 运行结果



2.7 小结

通过对本章内容的学习，可以让读者了解 Java 的基本数据类型的定义和它们之间的转换规则，以及它们之间的注意事项。掌握标识符的定义是学习本章的重点，也是写好程序的基础。如果读者还不完全了解标识符的相关问题，还可以参考电子工业出版社出版的《Java 程序设计应用教程》一书来进行学习。在下一章中将继续讲解 Java 基本语法中的运算符。



2.8 习题

一、填空题

1. Java 中的基本数据类型包括_____、_____、_____、_____。
2. 在 Java 的整型数值中有三种进制方式，分别为_____、_____、_____。
3. Java 中的类型转换方式包括_____、_____、_____。
4. Java 中的注释分为_____、_____、_____三种。

二、选择题

1. 下面哪一个是 Java 中的有效关键字（ ）？
 - A. method
 - B. native
 - C. For
 - D. array
2. 下面程序的运行结果是（ ）？


```
01 public class Hello
02 {
03     public static void main(String args[])
04     {
05         signed int x=3;
06         System.out.println(x);
```



```
07      )  
08  }
```

- A. 3 B. 编译错误 C. 没有任何输出 D. 抛出异常
3. 选择下面正确定义一个字符型数值的选项 ()。
- A. char c1='uface'; B. char c2='iface';
C. char c3='u0123'; D. char c4='face';
E. char c5=0xbeef;
4. 选择下面正确定义一个布尔型数值的选项 ()。
- A. boolean c1=0; B. boolean c2=false; C. boolean c3='false';
D. boolean c4=no; E. boolean c5=Boolean.false();
5. 选择下面正确定义一个浮点型数值的选项 ()。
- A. float c1=-123; B. float c2=3.1415; C. float c3=42E3;
D. float c4=20.1D; E. float c5=20.2F;
6. 选择下面正确进行标识符命名的选项 ()。
- A. 3AAA B. AAA3
C. _AAA D. 2_AAA

三、简答题

1. 简述什么是正确的标识符命名。
2. 什么情况下能够进行自动转换？什么情况下需要进行强制类型转换？

第3章 运 算 符

所谓运算符，就好比日常生活中的运算符号“+”、“-”、“*”、“/”，这些符号几乎天天都要用到。在Java中，运算符就和日常生活中的运算符号一样，起到运算的作用，但它们不再是这么简单的运算符。本章将介绍这些运算符，通过本章，读者应该实现如下目标。

- 了解算术运算符的概念和熟练使用算术运算符。
- 了解关系运算符的概念和熟练使用关系运算符。
- 了解逻辑运算符的概念和熟练使用逻辑运算符。
- 了解三元运算符的概念和熟练使用三元运算符。
- 了解位运算符的概念和熟练使用位运算符。
- 了解位移运算符的概念和熟练使用位移运算符。
- 了解赋值运算符的概念和熟练使用赋值运算符。



3.1 算术运算符

算术运算符是读者最熟悉不过的了，比如“+”、“-”、“*”、“/”。本节先来介绍算术运算符如何应用，以及有哪些注意事项。

- `+`: 加法运算符，也可做字符串的连接用途。
- `-`: 减法运算符。
- `*`: 乘法运算符。
- `/`: 除法运算符。
- `%`: 求余运算符。

3.1.1 “+”: 加法运算符

加法运算符和日常生活中“+”是一样的，都是做两个数值的加法运算。下面举例在Java中的形式。

```
//声明两个整数
int i = 33;
int j = 44;

//将33和44做加法运算
int n = i + j;
```

【范例 3-1】下面是一个使用加法运算符的程序。

示例代码 3-1

```
01 //修改HelloWorld例子，编程如下形式
02 public class HelloWorld
03 {
```



```
04 //定义两个整型变量
05 int i = 33;
06 int j = 44;
07 //定义两个字符型变量
08 char c1 = 'a';
09 char c2 = 'b';
10
11 public static void main(String args[])
12 {
13     //创建对象,对象引用为hw
14     HelloWorld hw = new HelloWorld();
15
16     //将两个整型变量做加法运算
17     int n = hw.i + hw.j;
18
19     //将两个字符型的变量进行相加
20     int c = hw.c1 + hw.c2;
21
22     //打印并显示结果.
23     System.out.println(n);
24     System.out.println(c);
25 }
26 }
```



图 3-1 加法运算

【运行结果】使用 `javac` 编译程序, 然后使用 `java` 运行编译产生的 `class` 程序, 运行结果如图 3-1 所示。

【代码解析】`c1` 和 `c2` 的类型为 `char`, 因为 `char` 型的位数比 `int` 型的位数小, 所以能进行转换。加法运算法除了具有加法运算的功能, 还有连接两个字符的功能。

提示: 在进行多个字符连接时, 可以使用多个加法运算法来进行连接。但是这样是存在缺点的, 这在后面将学习到。

3.1.2 “-”: 减法运算符

减法运算符和日常生活中 “-” 是一样的, 都是做两个数值的减法运算。下面举例在 Java 中的形式。

```
//声明两个整数
int i = 66;
int j = 77;

//将 66 和 77 做减法运算
int n = i - j;
```

【范例 3-2】下面是一个使用减法运算符的程序。

示例代码 3-2

```
01 //修改上一节例子, 如下形式
02 public class HelloWorld
03 {
04     //定义两个整型变量
05     int i = 55;
```

```

06     int j = 22;
07     //定义两个字符型变量
08     char c1 = 'f';
09     char c2 = 'a';
10     public static void main(String args[])
11     {
12         //创建对象，对象引用为h1
13         HelloWorld h1 = new HelloWorld();
14         //将两个整型变量做减法运算
15         int n = h1.i - h1.j;
16         //将两个字符型的变量进行相减
17         int c = h1.c1 - h1.c2;
18         //打印并显示结果
19         System.out.println(n);
20         System.out.println(c);
21     }
22 }

```

【运行结果】 使用 `javac` 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 3-2 所示。

【代码解析】 变量 `i` 和变量 `j` 通过减法运算符做减法运算。变量 `c1` 和变量 `c2` 做减法运算，虽然是 `char` 型的，通过转换运算结果为 `int` 型。



图 3-2 减法运算

3.1.3 “*”: 乘法运算符

乘法运算符 “*” 和日常生活中乘号类似，只是符号不一样而已，都是做两个数值的乘法运算。下面举例在 Java 中的形式。

```

//声明两个整数
int i = 6;
int j = 7;

//将 6 和 7 做乘法运算
int n = i * j;

```

【范例 3-3】 下面是一个使用乘法运算符的程序。

示例代码 3-3

```

01 //修改上一节例子，如下形式
02 public class HelloWorld
03 {
04     //定义两个整型变量
05     int i = 6;
06     int j = 7;
07
08     public static void main(String args[])
09     {
10         //创建对象，对象引用为h1
11         HelloWorld h1 = new HelloWorld();
12
13         //将两个整型变量做乘法运算
14         int n = h1.i * h1.j;
15
16         //打印并显示结果
17         System.out.println(n);

```



```
18  }
19 }
```



图 3-3 乘法运算

【运行结果】使用 `javac` 编译程序，然后使用 Java 运行编译产生的 `class` 程序，运行结果如图 3-3 所示。

【代码解析】乘法运算符不仅可以做整型数值的乘法运算，也可以做浮点型数值的乘法运算。在做乘法运算的时候应注意运算的优先级。

3.1.4 “/”：除法运算符

除法运算符 “/” 和日常生活中的除号类似，只是符号不一样而已，都是做两个数值的除法运算。下面举例在 Java 中的形式。

```
//声明两个整数
int i = 24;
int j = 2;

//将 24 和 2 做除法运算
int n = i / j;
```

【范例 3-4】下面是一个使用除法运算符的程序。

示例代码 3-4

```
01 //修改上一节例子，如下形式
02 public class HelloWorld
03 {
04     //定义两个整型变量
05     int i = 24;
06     int j = 2;
07     public static void main(String args[])
08     {
09         //创建对象，对象引用为 h1
10         HelloWorld h1 = new HelloWorld();
11         //将两个整型变量做除法运算
12         int n = h1.i / h1.j;
13         //打印并显示结果
14         System.out.println(n);
15     }
16 }
```

【运行结果】使用 `javac` 编译程序，然后使用 Java 运行编译产生的 `class` 程序，运行结果如图 3-4 所示。

【代码解析】除法运算符不仅可以做整型数值的除法运算，也可以做浮点型数值的除法运算。在做除法运算的时候应注意运算的优先级。在做浮点型数值的除法运算时，如果被除数为 0，那么结果为无穷大。

3.1.5 “%”：求余运算符

求余运算符 “%” 和日常生活中的除法求余类似，也是求两个数值的除法运算的余数。下面举例在 Java 中的形式。



图 3-4 除法运算

```
//声明两个整数
int i = 24;
int j = 5;

//将 24 和 5 做求余运算
int n = i % j;
```

【范例 3-5】下面是一个使用求余运算符的程序。

示例代码 3-5

```
01 //修改上一节例子, 如下形式
02 public class HelloWorld
03 {
04     //定义两个整型变量
05     int i = 24;
06     int j = 5;
07     public static void main(String args[])
08     {
09         //创建对象, 对象引用为 h1
10         HelloWorld h1 = new HelloWorld();
11         //将两个整型变量做求余运算
12         int n = h1.i % h1.j;
13         //打印并显示结果
14         System.out.println(n);
15     }
16 }
```

【运行结果】使用 javac 编译程序, 然后使用 Java 运行编译产生的 class 程序, 运行结果如图 3-5 所示。

【代码解析】整数在做求余运算时, 被除数不能为 0。浮点数在做求余运算时, 除数为 0, 结果为 NaN。求余时, 被除数为负数时, 结果依然为正数。求余时, 除数为负数时, 结果负数。



图 3-5 求余运算



3.2 自增自减运算符

所谓自增减运算符 “++”、“--”，就是变量和数字 1 做加减法运算将运算的结果赋值给做运算的变量，如下所示。

```
int i = 4;
i++;
int j = 4;
j = j + 1;
```

【范例 3-6】下面是一个使用自增运算符的程序。

示例代码 3-6

```
01 //修改上一节例子, 如下形式
02 public class HelloWorld6
03 {
04     public static void main(String args[])
05     {
06         //定义两个整型变量
07         int i = 5;
```



```
08         int j = 5;
09         //创建对象，对象引用为 hw
10         HelloWorld6 hl = new HelloWorld6();
11         //将变量 i 自增
12         i++;
13         j = j + 1;
14         //打印并显示结果
15         System.out.println(i);
16         System.out.println(j);
17     }
18 }
```

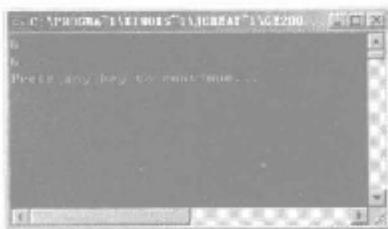


图 3-6 自增运算

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 3-6 所示。

【代码解析】读者可以发现，这两个运算的结果相等。其实自增运算符“++”对变量 i 进行运算就相当于 $i = i + 1$ ，只是一个简写的形式。“++”即是自增运算符，只能操作变量，不能操作具体的数值。自增运算符放在变量的前面和后面，其意义是不一样的。在前面是先加 1 再做运算，后面是先做运算再加 1。

和自增运算符相对应的就是自减运算符“-”，自增运算符是为原数增加 1，自减运算符是为原数减少 1。如下所示。

```
int i = 4;
i--;
int j = 4;
j = j - 1;
```

【范例 3-7】下面举一个完整的例子，来演示自减运算符是如何使用的。

示例代码 3-7

```
01 //修改上一节例子，如下形式
02 public class HelloWorld7
03 {
04     public static void main(String args[])
05     {
06         //定义两个整型变量
07         int i = 5;
08         int j = 5;
09         //创建对象，对象引用为 hw
10         HelloWorld hl = new HelloWorld();
11         //将变量 i 自减
12         i--;
13         j = j - 1;
14         //打印并显示结果
15         System.out.println(i);
16         System.out.println(j);
17     }
18 }
```

【运行结果】使用 javac 编译程序，然后使用 java 运行编译产生的 class 程序，运行结果如图 3-7 所示。

【代码解析】自减运算符和自增运算符类似。其实

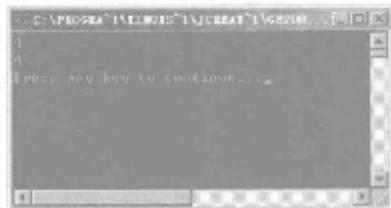


图 3-7 自减运算

就相当于实现 `i = i - 1;`，只是一个简写的形式。“`--`”即是自减运算符，只能操作变量，不能操作具体的数值。自减运算符放在变量的前面和后面，其意义是不一样的。在前面是先减 1 再做运算，后面是先做运算再减 1。



注意：自增和自减运算符放在数字的前面和放在数字的后面是有很大不同的，读者可以做实验体验一下。



3.3 关系运算符

关系运算符描述的是一种关系，既然描述的是关系，那结果就为对或不对。在 Java 里就表示为真或假。下面是关系运算符的分类。

- “`==`”：表示等于。
- “`!=`”：表示不等于。
- “`>=`”：表示大于等于。
- “`<=`”：表示小于等于。
- “`>`”：表示大于。
- “`<`”：表示小于。

关系运算符比较基本类型时，表示比较的值是否相等。如果用 “`==`” 和 “`!=`” 比较对象，就表示比较对象的引用是否相等。

3.3.1 “`==`”、“`!=`”

等于和不等于运算符比较的是运算数的等于和不等于，结果为 `true` 或 `false`，即真或假。例如下面的例子。

```
// 定义两个整型的变量
int i = 4;
int j = 4;

boolean b1 = i == j;
boolean b2 = i != j;

// 创建两个对象
String s1 = new String();
String s2 = new String();

boolean b3 = b1 == b2;
boolean b4 = s1 != s2;
```

【范例 3-8】下面是一个使用等于和不等于运算符的程序。

示例代码 3-8

```
01 // 修改上一节例子，如下形式
02 public class HelloWorld8
03 {
04     public static void main(String args[])
05     {
06         // 定义两个整型的变量
07         int i = 4;
```



```
08         int j = 4;
09         boolean b1 = i == j;
10         boolean b2 = i != j;
11         //创建两个对象
12         String s1 = new String();
13         String s2 = new String();
14         boolean b3 = (s1 == s2);
15         boolean b4 = (s1 != s2);
16         System.out.println(b1);
17         System.out.println(b2);
18         System.out.println(b3);
19         System.out.println(b4);
20     }
21 }
```

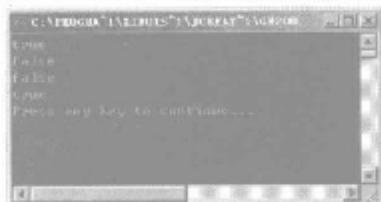


图 3-8 等于和不等于运算

【运行结果】使用 `javac` 编译程序，然后使用 Java 运行编译产生的 `class` 程序，运行结果如图 3-8 所示。

【代码解析】在本程序中，使用关系运算符来比较整数和字符串及布尔值。从该程序可以看到等于和不等于关系运算符使用得非常广泛。

3.3.2 “>”、“<”、“>=”、“<=”

这 4 个运算符比较的是运算数的大小关系，结果为 `true` 或 `false`，即真或假。例如下面是使用这些关系运算符的例子。

```
//定义两个整型变量
int i = 5;
int j = 4;

boolean b1 = i > j;
boolean b2 = i < j;
boolean b3 = i >= j;
boolean b4 = i <= j;
```

【范例 3-9】下面是一个使用大小关系运算符的程序。

示例代码 3-9

```
01 //修改上一节例子，如下形式
02 public class HelloWorld9
03 {
04     public static void main(String args[])
05     {
06         //定义两个整型的变量
07         int i = 5;
08         int j = 4;
09         boolean b1 = i > j;
10         boolean b2 = i < j;
11         boolean b3 = i >= j;
12         boolean b4 = i <= j;
13         System.out.println(b1);
14         System.out.println(b2);
15         System.out.println(b3);
16         System.out.println(b4);
17     }
18 }
```



【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 3-9 所示。

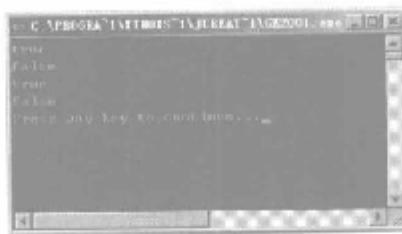


图 3-9 大小关系运算

【代码解析】大小关系运算符比较的是具体数值，结果为 true 或 false。大小关系运算符不能比较对象。



3.4 逻辑运算符

逻辑运算符，其实就是比较二进制数的逻辑关系，运算结果为 true 或 false。逻辑运算符包括如下几种。

- 与运算符：“`&&`”。
- 或运算符：“`||`”。
- 非运算符：“`!`”。

3.4.1 “`&&`：与运算符

“`&&`”运算符比较的是符号两边的表达式的真假。

【范例 3-10】通过下面代码说明“`&&`”运算符。

示例代码 3-10

```

01 //修改上一节例子，如下形式
02 public class HelloWorld10
03 {
04     public static void main(String args[])
05     {
06         boolean n = (4 > 3) && (2 < 8);
07         System.out.println(n);
08     }
09 }
```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 3-10 所示。

【代码解析】首先判断与运算符的两边是否为 true，如果有一边为 false，结果就为 false，并退出这个表达式；如果运算符的两边都为 true，结果即为 true。

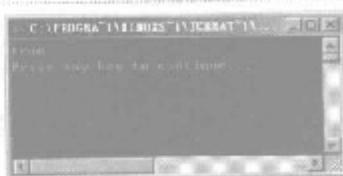


图 3-10 与运算

3.4.2 “`||`”：或运算符

“`||`”运算符比较的是符号两边的表达式的真假。



【范例 3-11】通过下面代码说明“||”运算符。

示例代码 3-11

```
01 //修改上一节例子,如下形式
02 public class HelloWorld11
03 {
04     public static void main(String args[])
05     {
06         boolean n = (4 > 3) || (2 > 8);
07         //打印并显示结果
08         System.out.println(n);
09     }
10 }
```



图 3-11 或运算

【运行结果】使用 javac 编译程序,然后使用 Java 运行编译产生的 class 程序,运行结果如图 3-11 所示。

【代码解析】首先判断或运算符两边的表达式值,只要有一边为 true,结果即为 true;如果运算符的两边都为 false,那么结果为 false。

3.4.3 “!”：非运算符

“!”运算符是把符号右边的表达式的结果取反。如为 true,取反为 false;如为 false,取反为 true。

【范例 3-12】通过下面代码说明“!”运算符。

示例代码 3-12

```
01 //修改上一节例子,如下形式
02 public class HelloWorld12
03 {
04     public static void main(String args[])
05     {
06         boolean n = !(2 > 8);
07         //打印并显示结果
08         System.out.println(n);
09     }
10 }
```

【运行结果】使用 javac 编译程序,然后使用 Java 运行编译产生的 class 程序,运行结果如图 3-12 所示。

【代码解析】在本程序中,2 大于 8 肯定是不正确的,其结果为 false。但是在前面使用非运算符,结果就会变为 true,由此可以知道非运算符的作用。



图 3-12 非运算

3.4.4 逻辑运算符总结

- “&&”运算符,符号的两边表达式值都为 true 时,结果为 true;只要有一边不为 true,结果即为 false。
- “||”运算符,符号的两边表达式值只要有一边为 true,结果就为 true;如果都为 false,结果即为 false。
- “!”运算符,原表达式值为 true,结果为 false;如果为 false,结果即为 true。



3.5 三元运算符

三元运算符是对三个表达式进行的集中比较，表达式1的值为true时，结果就为第二个表达式值；如果表达式1的值为false时，结果就为第三个表达式值。语法如下：

表达式1? 表达式2: 表达式3

【范例 3-13】通过下面代码说明三元运算符。

示例代码 3-13

```

01 //修改上一节例子，如下形式
02 public class HelloWorld13
03 {
04     public static void main(String args[])
05     {
06         //4 < 3表达式的结果为true 和 false 中的一个
07         boolean n = (4 < 3) ? true : false;
08         //打印并显示结果
09         System.out.println(n);
10     }
11 }
```

【运行结果】使用javac编译程序，然后使用Java运行编译产生的class程序，运行结果如图3-13所示。

【代码解析】使用三元运算符经常能在实际开发中起到事半功倍的效果。在本程序中，只是一个简单的应用，使用三元运算符来判断4是否小于3，如果是则返回第二项内容“true”，如果不是返回第三项内容“false”。

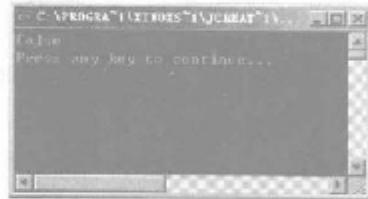


图3-13 三元运算符运算

提示：三元运算符是一种比较特殊的运算符，它和后面将要学习的if语句具有同样的功能，但是使用三元运算符编写的代码要比if语句编写的代码少得多。



3.6 位运算符

位运算符是将操作数转换成二进制，然后按位进行比较。

运算符包括如下几种。

- “&”：按位与运算符。
- “|”：按位或运算符。
- “^”：按位异或运算符。

比较规则如表3-1所示。

表3-1 位运算符的比较规则

x	y	x&y	x y	x^y
0	0	0	0	0



续表

x	y	$x \& y$	$x \mid y$	$x \wedge y$
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

3.6.1 “&”: 按位与运算符

按位与运算符，两个数同位都为 1 的时候该位结果即为 1，有一边不是 1 的话该位结果就为 0。

【范例 3-14】通过下面代码的演示来说明“&”运算符。

示例代码 3-14

```

01 //修改上一节例子，如下形式
02 public class HelloWorld14
03 {
04     public static void main(String args[])
05     {
06         int n = 4 & 3;
07         //打印并显示结果
08         System.out.println(n);
09     }
10 }
```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 3-14 所示。

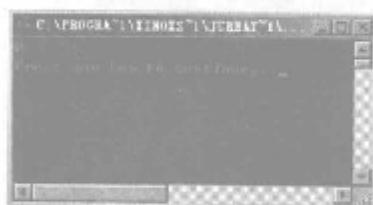


图 3-14 按位与运算

【代码解析】4 转换为二进制数为 0100，3 转换为二进制数为 0011。

0100

0011

—

0000

转换为十进制数为 0。

3.6.2 “|”: 按位或运算符

按位或运算符，两个数同位有一个为 1 的时候该位结果即为 1。

【范例 3-15】通过下面代码的演示来说明“|”运算符。

示例代码 3-15

```

01 //修改上一节例子，如下形式
02 public class HelloWorld15
03 {
04     public static void main(String args[])
05     {
06         int n = 4 | 3;
07         //打印并显示结果
08         System.out.println(n);
09     }
10 }
```

【运行结果】使用 `javac` 编译程序，然后使用 Java 运行编译产生的 `class` 程序，运行结果如图 3-15 所示。

【代码解析】4 转换为二进制数为 0100，3 转换为二进制数为 0011。

9100

0011

0111

转换为十进制数为 7。

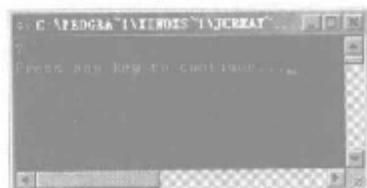


图 3-15 按位或运算

3.6.3 “ \wedge ”：按位异或运算符

按位异或运算符，两个数同位都为 1 的时候该位结果即为 0。有一个为 1 该位结果即为 1。

【范例 3-16】通过下面代码的演示来说明“`^`”运算符。

示例代码 3-16

```
01 //修改上一节例子,如下形式
02 public class HelloWorld16
03 {
04     public static void main(String args[])
05     {
06         int n = 4 ^ 3;
07         //打印并显示结果
08         System.out.println(n);
09     }
10 }
```

【运行结果】使用 `javac` 编译程序，然后使用 Java 运行编译产生的 `class` 程序，运行结果如图 3-16 所示。

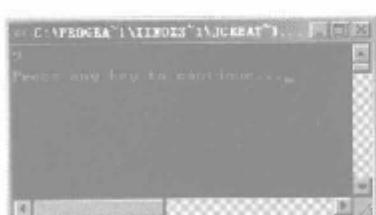


图 3-16 异或运算符

【代码解析】4 转换为二进制数为 0100，3 转换为二进制数为 0011。

0100

0011

011

转换为十进制数为 7。

3.7 位移运算符

位移运算符先把操作数转换成二进制数，然后向右向左移动相应的位数。位移运算符包括如下几种。

- “ $>>$ ” : 带符号右移运算符。
 - “ $<<$ ” : 带符号左移运算符。
 - “ $>>>$ ” : 无符号右移运算符。



3.7.1 “>>”: 带符号右移运算符

右移运算符是把操作数转换成二进制数向右移动指定的位数。该右移运算符是有符号的，如果为正数就补 0，如果为负数就补 1。

【范例 3-17】通过下面代码的演示来说明“>>”运算符。

示例代码 3-17

```
01 //修改上一节例子，如下形式
02 public class HelloWorld17
03 {
04     public static void main(String args[])
05     {
06         int n = 7 >> 2 ;
07         //打印并显示结果
08         System.out.println(n);
09     }
10 }
```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 3-17 所示。

【代码解析】7 转换为二进制数：0000 0111。
0000 0111 >> 2 = 0000 0001，转换为十进制数为 1。

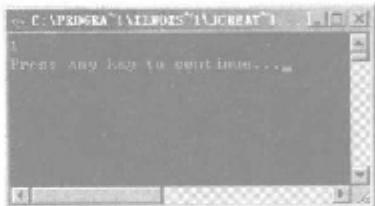


图 3-17 带符号右移运算

3.7.2 “<<”: 带符号左移运算符

左移运算符是把操作数转换成二进制数向左移动指定的位数。该左移运算符是有符号的，如果为正数就补 0，如果为负数就补 1。

【范例 3-18】通过下面代码的演示来说明“<<”运算符。

示例代码 3-18

```
01 //修改上一节例子，如下形式
02 public class HelloWorld18
03 {
04     public static void main(String args[])
05     {
06         int n = 7 << 2 ;
07         //打印并显示结果
08         System.out.println(n);
09     }
10 }
```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 3-18 所示。

【代码解析】7 转换为二进制数：0000 0111。
0000 0111 << 2 = 0001 1100，转换为十进制数为 28。

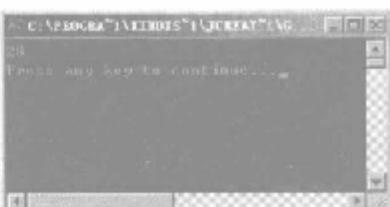


图 3-18 带符号左移运算

3.7.3 “>>>”: 无符号右移运算符

无符号右移运算符是把操作数转换成二进制数向右移动指定的位数。无符号右移运算符全在最高位

上补0。

【范例 3-19】通过下面代码的演示来说明“>>>”运算符。

示例代码 3-19

```
01 //修改上一节例子，如下形式
02 public class HelloWorld19
03 {
04     public static void main(String args[])
05     {
06         int n = 7 >>> 3 ;
07         //打印并显示结果
08         System.out.println(n);
09     }
10 }
```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 3-19 所示。

【代码解析】7 转换为二进制：0000 0111。

$0000\ 0111\ >> 2 = 0000\ 0001$ ，转换为十进制为 1。



图 3-19 无符号右移运算



3.8 赋值运算符

赋值运算符就好比在日常生活中进行 $a=3$ 操作，以后就可以用 a 表示 3 这个数值一样，即把 3 赋给变量 a 。

3.8.1 一般赋值运算符

一般运算符使用“ $=$ ”，在编写代码时最常用，也很容易理解。示例如下：

int n = 3;

这一条语句的含义是把数值 3 赋值给整型变量 n 。

3.8.2 运算赋值运算符

运算赋值运算符和一般赋值运算符很相似，也是赋值用的，但它具有运算的功能。

【范例 3-20】通过下面代码的演示来说明运算赋值运算符。

示例代码 3-20

```
01 public class HelloWorld20
02 {
03     public static void main(String args[])
04     {
05         int n = 7;
06         int j = 0;
07         j += n;
08         //打印并显示结果
09         System.out.println(j);
10     }
11 }
```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结



果如图 3-20 所示。

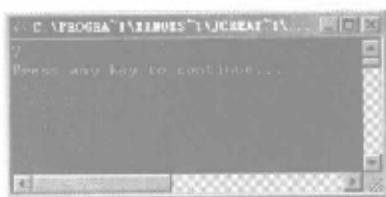


图 3-20 运算赋值运算

【代码解析】“`+=`”就是一个运算赋值运算符，它和自增运算符很相似，本程序中相当于实现 `j = j + n`。

运算赋值运算符有很多，这里不一一举例，下面给出这些运算赋值运算符的含义。

- `a+=b` : 相当于 `a = a + b;`
- `a-=b` : 相当于 `a = a - b;`
- `a*=b` : 相当于 `a = a * b;`

- `a/=b` : 相当于 `a = a / b;`
- `a%*=b` : 相当于 `a = a % b;`
- `a^=b` : 相当于 `a = a ^ b;`
- `a|=b` : 相当于 `a = a | b;`
- `a&=b` : 相当于 `a = a & b;`
- `a>>=b` : 相当于 `a = a >> b;`
- `a<<=b` : 相当于 `a = a << b;`
- `a>>>=b` : 相当于 `a = a >>> b.`



3.9 运算符之间的优先级

运算符之间的运算优先级是有一定的顺序的。括号拥有最高的优先级，接下来是一元运算符，最后是二元运算符，如表 3-2 所示。

表 3-2 各个运算符的优先级

运算符优先级从高到低	含 义
<code>()</code>	括号
<code>++ -- ~ !</code>	自增和自减运算符
<code>*/%</code>	算术运算符
<code>+-</code>	算术运算符
<code>>> >>> <<</code>	位移运算符
<code>< <= > >=</code>	关系运算符
<code>== !=</code>	关系运算符
<code>&</code>	位与运算符
<code>^</code>	位异或运算符
<code> </code>	位或运算符
<code>&&</code>	逻辑与运算符
<code> </code>	逻辑或运算符
<code>?:</code>	三元选择运算符
<code>= += -= *= /= %= &= >>= >>>= <<=</code>	赋值运算符



说明：虽然运算符间具有严格的优先级，但是一般不提倡直接使用。读者在写自己的表达式时应尽量多使用括号，从而让其他人很容易读懂程序。



3.10 综合练习

1. 区分前置自增减运算符和后置自增减运算符的不同。

【提示】通过程序来看这个问题。

```
01  public class LianXii
02  {
03      public static void main(String args[])
04      {
05          int a=1;
06          int b=1;
07          System.out.println("使用后置运算符的结果为: "+(a++)); //显示后置结果
08          System.out.println("使用前置运算符的结果为: "+(++b)); //显示前置结果
09      }
10  }
```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 3-21 所示。

2. 三元运算符的应用有哪些？

【提示】通过程序来看这个问题。

```
01  public class Lianxi2
02  {
03      public static void main(String args[])
04      {
05          int a=3;
06          int b=4;
07          System.out.println("使用条件运算符显示");
08          String s=(a<b)?"a 小于 b":"a 大于 b";
09          System.out.println(s);
10          System.out.println("使用 if 条件语句显示");
11          if(a<b)
12          {
13              System.out.println("a 小于 b");
14          }
15          else
16          {
17              System.out.println("a 大于 b");
18          }
19      }
20  }
```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 3-22 所示。



图 3-21 前置和后置的区别



图 3-22 三元运算符



3.11 小结

通过对本章各种运算符的学习，读者应对 Java 基本类型的运算有了新的认识，虽然很简单，但对以后学习很重要。虽然运算符比较简单，但其中也是有很多知识需要讲解的。在本章中最常用的运算符是自增自减运算符和赋值运算符，这也是本章的重点。如果读者想了解更多的关于 Java 运算符的内容，可以参考电子工业出版社出版的《Java 程序设计教程（第五版）（英文版）》一书来进一步学习。下一章将继续学习 Java 基本语法中的流程控制语句。



3.12 习题

一、填空题

1. 算术运算符包括_____、_____、_____、_____、_____。
2. 位运算符包括_____、_____、_____。
3. 位移运算符包括_____、_____、_____。

二、选择题

1. 下面的哪两个选项的结果相同（ ）。
A. 32/2; B. (8>>2) <<4; C. 5^3;
D. 128>>>2; E. 2>>5;
2. 选择下面程序的运行结果（ ）。

```
public class Hello
{
    public static void main(String args[])
    {
        int x=11&9;
        int y=x^3;
        System.out.println(y/12);
    }
}
```

A. 14; B. 17; C. 5; D. 0;
3. 选择该程序的运行结果（ ）。

```
public class Hello
{
    public static void main(String args[])
    {
        boolean b1=true;
        boolean b2=false;
        boolean b3=true;
        if(b1&b2|b2&b3|b2)
        {
            System.out.println("aaa");
        }
        if(b1&b3|b2&b3|b2|b3);
        {
            System.out.println("bbb");
        }
    }
}
```

- A. aaa
- B. bbb
- C. aaaabbbb
- D. 编译发生错误

4. 选择下面程序的运行结果 ()。

```
public class Hello
{
    public static void main(String args[])
    {
        int x=0;
        int y=0;
        for(int k=0;k<5;k++)
        {
            if(((++x>2)||(++y>2))
            {
                x++;
            }
        }
        System.out.println(x+"and"+y);
    }
}
```

A. 8and2 B. 5and3 C. 10and5 D. 编译发生错误

三、简略地

1. 判断下面的程序是否能够正常运行。如果不能运行, 请说出不能运行的原因。

```
01 public class Hello
02 {
03     public static void main(String args[])
04     {
05         int x=3;
06         double y=3.0;
07         boolean b=(x==y);
08         System.out.println(b);
09     }
10 }
```

2. 简述运算符的优先级，以及如何使用优先级。

第4章 流程控制

在日常生活中，人们早上起床后，通常要做洗脸、刷牙等事情；如果有好看的电视节目，可能会打开电视机收看；这些事情的先后顺序，每一个人都有自己的安排。在 Java 中，洗脸、刷牙等事情就好像程序代码，它们是由流程控制语句来控制的。例如，在流程控制语句中有 if 语句，它的作用就是根据条件来执行程序，就好像根据是否有好看的电视节目来决定是否打开电视机一样。在 Java 里控制流程语句主要有条件语句、分支语句、循环语句，本章将分别介绍。通过本章的学习，读者应该能够实现下面的几点目标。

- 了解 if 条件语句和掌握各种 if 条件语句的使用。
- 了解 switch 分支语句和掌握 switch 分支语句的使用。
- 了解 while 循环语句和掌握 while 循环语句的使用。
- 了解 do-while 循环语句和掌握 do-while 循环语句的使用。
- 了解 for 循环语句和掌握 for 循环语句的使用。



4.1 if 条件语句

在前面已经提到，如果有好看的电视节目时，就打开电视收看。在 Java 中，if 条件语句就用来实现这个功能，如果 if 条件中的条件语句是正确的，就会执行 if 语句中的语句。

4.1.1 if 语句的语法

if 语句的基本语法如下：

```
if(表达式){方法体}else if(表达式){方法体}else{方法体}
```

下面用代码演示。

```
if (a > 3)  
    条件成功的方法体
```

语法解释：

- if 语句的执行条件是，当表达式值为 true 时，执行方法体的部分。
- 如果表达式值为 false，执行 else if 的部分或 else 部分的方法体。

4.1.2 if 语句用法举例

if 语句的用法有好几种，下面列举 if 语句的几种形式。

- 简写形式：if ...
- 一般形式：if ... else
- 完整形式：if ... else if ... else

1. if语句的简写形式

if(表达式){方法体}

if语句简写形式的流程图如图4-1所示。

【范例4-1】通过程序的方式来讲解if简写形式。

示例代码4-1

```

01 public class If1
02 {
03     public static void main(String args[])
04     {
05         int n = 7;
06         //if语句执行判断,如果大于3打印变量n大于3,否则不打印
07         if (n > 3)
08             System.out.println("变量n大于3");
09     }
10 }

```

【运行结果】使用javac编译程序,然后使用Java运行编译产生的class程序,运行结果如图4-2所示。

【代码解析】当n的值大于3时就执行System.out.println()语句,否则不执行任何语句。

2. if语句的一般形式

if(表达式){方法体}else{方法体}

if语句一般形式的流程图如图4-3所示。

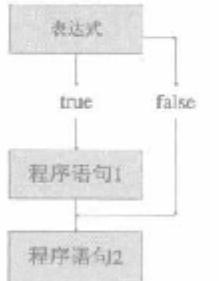


图4-1 if语句简写形式流程图

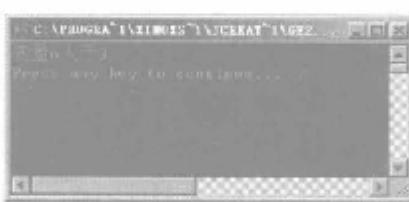


图4-2 if语句简写形式



图4-3 if语句一般形式流程图

【范例4-2】通过程序的方式来讲解if一般形式。

示例代码4-2

```

01 public class If2
02 {
03     public static void main(String args[])
04     {
05         int i = 3;
06         //如果i大于5
07         if (i > 5)
08         {
09             //执行下面的语句
10             System.out.println("变量i大于5");
11         }
12         //如果i不大于5

```



```
13         else
14         {
15             //执行下面的语句
16             System.out.println("变量i小于5");
17         }
18     }
19 }
```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 4-4 所示。

【代码解析】当 i 大于 5 的时候打印“变量 i 大于 5”，否则打印“变量 i 小于 5”。

3. if 语句的完整形式

if(表达式) {方法体} else if(表达式) {方法体} else {方法体}

if 语句完整形式的流程图如图 4-5 所示。

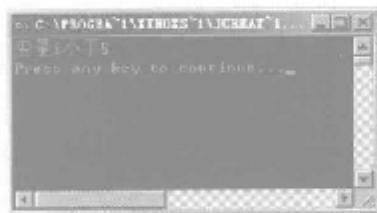


图 4-4 if 语句一般形式



图 4-5 if 语句完整形式流程图

【范例 4-3】通过程序的方式来讲解 if 完整形式。

示例代码 4-3

```
01 public class If0
02 {
03     public static void main(String args[])
04     {
05         int i = 3;
06         if (i > 5)
07         {
08             System.out.println("变量i大于5");
09         }
10         else if (i > 4)
11         {
12             System.out.println("变量i小于4");
13         }
14         else
15         {
16             System.out.println("其他");
17         }
18     }
19 }
```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结

果如图 4-6 所示。

【代码解析】当 i 的值大于 5 的时候，打印“变量 i 大于 5”，如果 i 的值大于 4，打印“变量 i 大于 4”，否则打印“其他”。

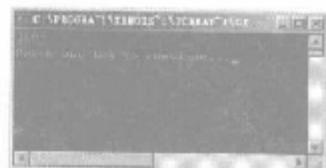


图 4-6 if 语句完整形式



说明：if 语句有三种形式，巧妙地选择使用不同的形式，既能使程序美观，又能起到简化程序的效果。



4.2 switch 分支语句

switch 语句和 if 相似，它是根据条件表达式的值来判断执行的程序语句。在日常生活中也经常有这样的情况，例如根据星期几来判断是否上课，如果为星期一到星期五中的一天就上课，如果为星期六或星期日就不上课。switch 分支语句要比 if 语句复杂得多。但当判断的条件很多时，switch 分支语句要比 if 语句要方便很多。

4.2.1 switch 分支语句的语法

switch 分支语句和 if 语句一样都是通过表达式的成立与否，来选择执行哪条语句的。先来看一下 switch 语句的组成部分。

```
switch (表达式)
{
    case 表达式 1:
    {
        表达式的结果与表达式 1 相匹配时，所执行的方法体。
        break;
    }
    case 表达式 2:
    {
        表达式的结果与表达式 2 相匹配时，所执行的方法体。
        break;
    }
    case 表达式 3:
    {
        表达式的结果与表达式 3 相匹配时，所执行的方法体。
        break;
    }
    ...
    default:
        表达式的结果与上述表达式的结果都不匹配时，所执行的方法体。
}
```

语法解释：

- 如果 switch 分支语句的表达式的结果和 case 语句中的表达式相等时，就执行该 case 语句的方法体，最后执行 break 语句退出 switch 分支语句。
- 当 switch 语句的表达式的结果没有和 case 语句中的表达式相匹配时，就执行 default 分支语句的方法体。



图 4-7 switch 分支语句基本流程图

switch 分支语句的基本流程图如图 4-7 所示。

4.2.2 switch 分支语句表达式的使用条件

switch 分支语句的表达式的使用有一定的条件，不是什么类型都能使用的。一般能使用的条件是具体的整型数值和一些有顺序的数列。下面先来对整型数值进行讲解。在 Java 中，整数类型包括 byte、char、short、int 型。

【范例 4-4】通过程序的方式来讲解 switch 分支语句。

示例代码 4-4

```

01 //测试 switch 分支语句
02 public class testSwitch
03 {
04     public static void main(String args[])
05     {
06         int i = 3;
07         char c = 4;
08         //switch 分支语句，表达式 i 只能是整型，这里的表达式 i 为 int 型
09         switch(i)
10         {
11             //具体的条件分支
12             case 3:
13             {
14                 System.out.println("所执行的分支是 case 3 分支");
15                 break;
16             }
17             default:
18                 System.out.println("其他分支语句");
19         }
20
21         //switch 分支语句，表达式 c 只能是整型，这里的表达式 i 为 char 型
22         switch(c)
23         {
24             //具体的条件分支
25             case 4:
26             {
27                 System.out.println("所执行的分支是 case 4 分支");
28                 break;
29             }
30             default:
31                 System.out.println("其他分支语句");
32         }
33     }
34 }
  
```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 4-8 所示。

【代码解析】通过判断 switch 分支语句的表达式 i 的结果，再和 case 语句相匹配。如果 i 的值和 case 语句的值相匹配的话，就执行该 case 语句的方法体。通过判断 switch 分支语句的表达式 c 的结果，再和 case 语句相匹配。如果 c 的值和 case 语句的值相匹配



图 4-8 switch 分支语句

的话，就执行该 case 语句的方法体。如果表达式没有与 case 语句相匹配的，就执行 default 分支的方法体。

除了具体的整型数值，还可以用表达式来判断。用表达式的结果和 case 语句相匹配，再执行相应的方法体。

【范例 4-5】通过程序的方式来讲解 switch 分支语句为表达式的情况。

示例代码 4-5

```

01 //测试switch分支语句
02 public class testSwitch2
03 {
04     public static void main(String args[])
05     {
06         int i = 7;
07         int n = 4;
08         //switch 分支语句，表达式 i - n 的结果为具体的整型数值
09         switch(i - n)
10         {
11             //具体的条件分支
12             case 3:
13             {
14                 System.out.println("所执行的分支是 case 3 分支");
15                 break;
16             }
17             case 4:
18             {
19                 System.out.println("所执行的分支是 case 4 分支");
20                 break;
21             }
22             default:
23                 System.out.println("其他分支语句");
24         }
25     }
26 }
```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 4-9 所示。

【代码解析】通过计算表达式 $i - n$ 的结果与 case 语句进行匹配，如果匹配就执行其相应的方法体。如果表达式 $i - n$ 的结果没有相匹配的，就执行 default 分支的语句。

switch 分支语句中的条件还可以是枚举类型。所谓枚举类型，就好比在日常生活中的星期一到星期日的一段有序的数列。先简单说明枚举类型如何定义，以后再做详细说明。

语法: enum 名称(各种有序的数据)
enum TestEnum{one, two, three, four}

【范例 4-6】下面举例说明枚举类型在表达式里如何应用。

示例代码 4-6

```

01 //定义一个枚举类型
02 enum TestEnum{one, two, three, four}
```

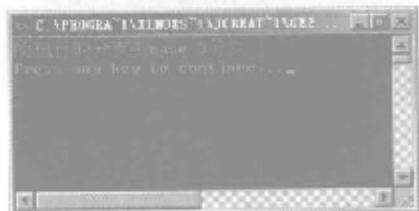


图 4-9 switch 语句条件为表达式



```
03 //测试switch分支语句
04 public class testSwitch3
05 {
06     public static void main(String args[])
07     {
08         //switch分支语句，表达式为一个枚举类型
09         switch(testEnum.two)
10         {
11             //具体的条件分支
12             case one:
13             {
14                 System.out.println("所执行的分支是 case one 分支");
15                 break;
16             }
17             case two:
18             {
19                 System.out.println("所执行的分支是 case two 分支");
20                 break;
21             }
22             case three:
23             {
24                 System.out.println("所执行的分支是 case three 分支");
25                 break;
26             }
27             case four:
28             {
29                 System.out.println("所执行的分支是 case four 分支");
30                 break;
31             }
32             default:
33                 System.out.println("其他分支语句");
34         }
35     }
36 }
```

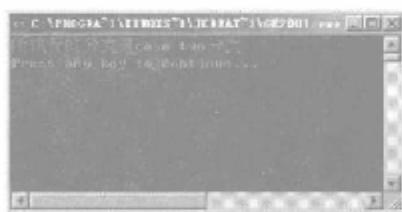


图 4-10 switch 语句条件为枚举类型

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 4-10 所示。

【代码解析】枚举类型和表达式很相似，通过查找枚举类型里的具体数据来和 case 语句相匹配，如果相匹配就执行其 case 语句的方法体。如果表达式的结果没有和 case 语句相匹配的，就执行 default 的方法体。



注：switch 语句的表达式可以为整型和枚举类型。case 语句的表达式一般为 switch 表达式的运行结果，一般为常量或具体的枚举数据。当 case 语句与表达式的结果没相匹配的，就执行 default 分支的方法体。每个 case 语句在执行完毕后要用 break 关键字来退出该 switch 分支语句。

4.2.3 switch 分支语句举例

上一节介绍了 switch 分支语句的各个组成部分的使用及注意事项，下面用一个完整的例子来说明 switch 语句。



【范例 4-7】一个比较大的使用 switch 分支语句的程序。

示例代码 4-7

```

01 //定义一个枚举类型
02 enum testEnum{one, two, three, four}
03 //测试switch 分支语句
04 public class testSwitch4
05 {
06     public static void main(String args[])
07     {
08         int i = 3;
09         //switch 分支语句, 表达式 i 只能是整型, 这里的表达式 i 为 int 型
10         switch(i)
11         {
12             //具体的条件分支
13             case 2:
14             {
15                 System.out.println("所执行的分支是 case 2 分支");
16                 break;
17             }
18             case 2 + 1:
19             {
20                 System.out.println("所执行的分支是 case 2 + 1 分支");
21                 break;
22             }
23             default:
24                 System.out.println("其他分支语句");
25         }
26         int ii = 7;
27         int nn = 4;
28
29         //switch 分支语句, 表达式 i - n 的结果为具体的整型数值
30         switch(ii - nn)
31         {
32             //具体的条件分支
33             case 3:
34             {
35                 System.out.println("所执行的分支是 case 3 分支");
36                 break;
37             }
38             default:
39                 System.out.println("其他分支语句");
40         }
41
42         //switch 分支语句, 表达式为一个枚举类型
43         switch(testEnum.two)
44         {
45             //具体的条件分支
46             case one:
47             {
48                 System.out.println("所执行的分支是 case one 分支");
49                 break;
50             }
51             case two:
52             {
53                 System.out.println("所执行的分支是 case two 分支");
54                 break;
55             }
56             default:

```



```

57             System.out.println("其他分支语句");
58         }
59     }
60 }

```

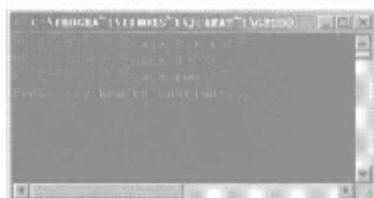


图 4-11 switch 分支语句

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 4-11 所示。

【代码解析】本程序将本节中学到的 switch 分支语句三种不同表达式的情况都包含在内，从而方便读者进行对照学习。



4.3 while 循环语句

所谓 while 循环语句，就是先进行判断再进行循环的语句。通过判断表达式的值，来决定具体的循环次数。下面先介绍 while 循环语句的语法并举例说明。

4.3.1 while 循环语句的语法

通过判断表达式的成功与否，来决定循环的次数。其基本语法如下：

```

while(表达式)
{
    方法体
}

```

while 循环语句的流程图如图 4-12 所示。

使用 while 循环语句时，一般先不知道要循环的次数，通过判断表达式的结果真与假，来决定具体的循环次数。

【范例 4-8】 通过程序来讲解 while 语句。

示例代码 4-8

```

01 public class testWhile
02 {
03     public static void main(String args[])
04     {
05         int i = 0;
06         //当整型 i 小于 3 的时候循环
07         while (i < 3)
08         {
09             System.out.println(i);
10             //每次递增整型变量 i，在下次循环的时候进行判断
11             i++;
12         }
13     }
14 }

```

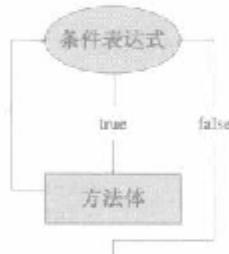


图 4-12 while 循环语句流程图

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 4-13 所示。

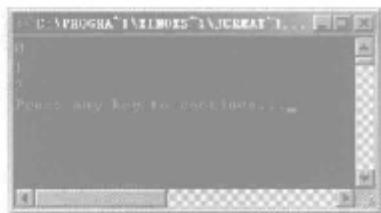


图 4-13 while 循环语句

4.3.2 while 循环语句举例

上一节介绍了 while 循环语句的具体语法后，下面用一个详细例子进行说明。

【范例 4-9】 在下面的程序中，将讲解如何显示乘法表。

示例代码 4-9

```
//testWhile2 类，所描述的是一个 9*9 乘法表
public class testWhile2
{
    public static void main(String args[])
    {
        //定义整型变量
        int i = 9;
        int j = 9;
        //当变量 i 大于或等于 1 的时候执行循环
        while(i >= 1)
        {
            while(j <= i) && (j > 0)
            {
                System.out.print(i + "*" + j + "=" + j * i + " ");
                j--;
            }
            System.out.println(" ");
            i--;
            //把每次循环后的 j 值赋值给 i
            j = i;
        }
    }
}
```



图 4-14 使用 while 循环语句

【运行结果】 使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 4-14 所示。

【代码解析】 该程序是有一定难度的，这里使用到了嵌套的 while 语句。在外层，且 i 大于等于 1 的情况下循环。在内层，则是以更复杂的条件进行循环，读者可以自己分析该程序。



4.4 do...while 循环语句

所谓 do...while 循环语句，就是先进行循环，再进行表达式的判断，如果表达式不成立就退出循环。下面先介绍 do...while 循环语句的语法并举例说明。



4.4.1 do...while 循环语句的语法

do...while 循环语句是先进行循环，再进行判断。其基本语法如下：

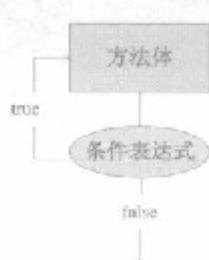


图 4.15 do-while 循环语句流程图

```
do  
{  
    方法体  
}  
while(表达式);
```

do ... while 循环语句的流程图如图 4-15 所示。

使用 `do ... while` 循环语句时, 无论表达式的结果真与假, 都进行一次循环。

【范例 4-10】下面是简单使用 do-while 循环语句的

示例代码 4-10

```
public class testDoWhile
{
    public static void main(String args[])
    {
        int i = 0;
        //当整型 i 小于 3 的时候结果循环
        do
        {
            System.out.println(i);
            //每次递增整型变量 i，在下次循环的时候进行判断
            i++;
        }
        while (i < 3);
    }
}
```

【运行结果】使用 `javac` 编译程序，然后使用 `Java` 运行编译产生的 `class` 程序，运行结果如图 4-16 所示。

【代码解析】在 do-while 语句中, 首先执行方法体, 然后判断条件是否符合。do-while 语句和 while 语句最大的区别就是, do-while 语句至少执行一次。

4.4.2 do ... while 循环语句举例

上一节介绍了 `do...while` 循环语句的具体语法，下面用一个详细例子进行说明。

【范例 4-11】下面是使用 do...while 循环语句完成显示乘法表功能的程序。

示例代码 4-11

```
//testDoWhile2类, 所描述的是一个9*9 乘法表
public class testDoWhile2
{
    public static void main(String args[])
    {
        for (int i = 1; i <= 9; i++)
        {
            for (int j = 1; j <= i; j++)
            {
                System.out.print(j + " * " + i + " = " + (j * i));
            }
            System.out.println();
        }
    }
}
```



图 4-16 使用 do-while 循环语句



```

    int i=9;
    int j=9;
    do
    {
        do
        {
            System.out.print(i + "*" + j + "=" + j * i + " ");
            j--;
        }
        while((j <= i) && (j > 0));
        System.out.println(" ");
        i--;
        j = i;
    }
    while(i >= 1);
}

```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 4-17 所示。

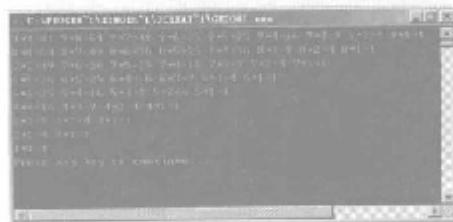


图 4-17 do-while 循环语句

【代码解析】在本程序中，使用的是 do_while 循环语句来完成显示乘法表的功能。从运行结果中可以看出，使用 do_while 循环语句同样能够完成 while 循环语句的功能。



4.5 for 循环语句

所谓 for 循环语句，就是明确了循环的次数，再进行循环的。下面先介绍 for 循环语句的语法并举例说明。

4.5.1 for 循环语句的语法

for 循环语句通过判断表达式的成立与否，来决定循环的次数。其基本语法如下：

```

for(变量初始化, 表达式, 递增表达式)
{
    方法体
}

```

for 循环语句的流程图如图 4-18 所示。

for 循环语句的变量是为循环设定循环次数的初始值，每次递增变量后，判断表达式的成功与否来决定循环的次数。



图 4-18 for 循环语句



【范例 4-12】下面是一个简单使用 for 循环语句的程序。

示例代码 4-12

```
public class testFor
{
    public static void main(String args[])
    {
        //当整型 i 小于 10 的时候循环
        for(int i = 0; i < 10; i++)
        {
            System.out.println(i);
        }
    }
}
```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 4-19 所示。



图 4-19 for 循环语句

【代码解析】for 循环语句里声明变量 i 只在循环的内部有效。如果在外面调用将出现编译错误，如果要在 for 循环外面使用变量 i，就要在外面声明变量，如 int i。

4.5.2 用 for 循环来实现其他循环语句

【范例 4-13】修改上一节中的显示 9×9 乘法表的程序代码。

示例代码 4-13

```
/*testFor2 类，所描述的是一个 9×9 乘法表
public class testFor2
{
    public static void main(String args[])
    {
        for (int i = 1; i < 10; i++)
        {
            for (int j = 1; j < 10; j++)
            {
                if (j <= i)
                {
                    System.out.print(i + "*" + j + "=" + (i * j) + " ");
                }
            }
            System.out.println(" ");
        }
    }
}
```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结

果如图 4-20 所示。

【代码解析】在本程序中同样要显示出一个乘法表，只是在该程序中使用的是 for 循环语句。在该程序中同样使用嵌套 for 循环，并且在内层 for 循环中使用到了 if 条件语句，读者可以看一下没有 if 条件语句出现什么结果。



图 4-20 嵌套 for 循环

4.5.3 for 循环语句的举例

下面介绍 for 循环语句的其他用法，并解释其含义。

【范例 4-14】看下面生成正三角形的程序。

示例代码 4-14

```
public class testFor3
{
    public static void main(String args[])
    {
        int i;
        int j;
        int input = 10;
        System.out.println("正三角:");
        for (i = 1; i <= input; i++)
        {
            for (j = i; j < input; j++)
            {
                System.out.print(" ");
            }
            //打印左半边三角
            for (j = i; j >= 1; j--)
            {
                if (j >= 10)
                    System.out.print("++"); //大于 9 的数字用*符号表示
                else
                    System.out.print(j); //如果要全部打印*号，此处将 j 用***代替
                //下边程序做相同改动
            }
            //打印右半边三角
            for (j = 2; j <= i; j++)
            {
                if (j >= 10)
                    System.out.print("++");
                else
                    System.out.print(j);
            }
            System.out.println();
        }
    }
}
```



图 4-21 生成正三角

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 4-21 所示。

【代码解析】本程序使用到了多个 for 循环，读者应该非常细心地分析该程序，分清每一个 for 循环的作用。对该程序全面了解后，就可以自己来编写一个输出倒三角的程序。如果读者能够写出这个程序，证明已经把 for 循环彻底掌握了。



4.6 如何中断和继续语句的执行

在学校中，有时会发生临时放假的情况，可能会临时放假一天，也可能会长期放假。Java 循环语句中也有这种情况，可以使用 break 语句和 continue 语句来中断程序，就好比中断上课一样。不同的是 break 语句类似于一直放假，而 continue 语句类似于放假一天。语句的中断就是指在语句的执行过程中，用代码中断语句的执行并退出此代码块；继续和中断类似，是用代码中断本次循环的执行而进入下一次循环。中断和继续在 Java 里用 break 和 continue 关键字来表示。

4.6.1 break：中断语句执行

break 关键字在前面学习的 switch 分支语句中已经使用过了，下面直接用代码进行说明。

【范例 4-15】下面是使用 break 来中断 for 循环的程序。

示例代码 4-15

```
public class testBreak
{
    public static void main(String args[])
    {
        for(int i = 0; i < 10; i++)
        {
            System.out.println(i);
            if (i == 5)
            {
                break;
            }
        }
    }
}
```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 4-22 所示。

【代码解析】for 循环的初始变量 i 的值为 0，每次递增 1，当 i 小于 10 的时候退出循环。使用 break 关键字是在循环变量 i 的值等于 5 的时候退出循环。



图 4-22 break 语句

4.6.2 continue: 继续语句执行

continue 语句表示跳出本循环，继续执行下一次循环，同样还是采用程序来讲解 continue 语句。

【范例 4-16】下面是使用 continue 语句的程序。

示例代码 4-16

```
public class TestContinue
{
    public static void main(String args[])
    {
        for(int i = 0; i < 10; i++)
        {
            if (i == 5)
            {
                continue;
            }

            System.out.println(i);
        }
    }
}
```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 4-23 所示。

【代码解析】for 循环的初始变量 i 的值为 0，每次递增 1，当 i 小于 10 的时候退出循环。当循环变量 i 的值等于 5 的时候，退出本次循环直接进入到下一次循环。

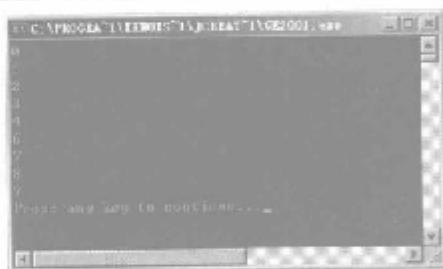


图 4-23 continue 语句



4.7 综合练习

1. 编写一个求从 1 到 100 中所有奇数和的程序。

【提示】使用 for 循环，将循环条件改为 $i+=2$ 。

```
01  public class Lianxi1
02  {
03      public static void main(String args[])
04      {
05          int t=0;
06          //执行 for 循环，使循环间隔为 2，从而只进行奇数操作
07          for(int i=1;i<=100;i+=2)
08          {
09              t+=i;    //每次循环时，为表示总和的变量加上本次循环的变量值
10          }
11          System.out.println("从 1 到 100 的奇数和为: "+t);
12      }
13  }
```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结



果如图 4-24 所示。

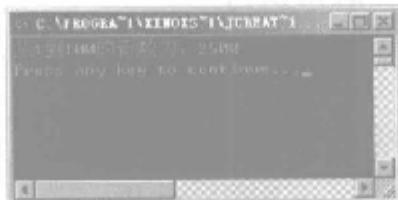


图 4-24 求奇数和运行结果

2. 求下面程序的输出结果，程序如下所示。

```
public class LianXi2
{
    public static void main(String args[])
    {
        int i=5;
        do
        {
            System.out.println(i);
            i++;
        }
        while(i<3);
    }
}
```

【提示】`do...while` 循环至少执行一次，所以虽然 `i=5` 第一次循环时并不符合`if` 条件，但是仍然会先执行方法体显示一次变量 `i` 的值。

3. 将下面的程序和练习 2 的程序相比较, 观察一下将出现什么错误。

```
public class LianXi3
{
    public static void main(String args[])
    {
        int i=5;
        do
        {
            System.out.println(i);
            i++;
        }
        while(i<3)
    }
}
```

【运行结果】运行该程序则发生错误，告诉程序员缺少分号。

【提示】在 do...while 循环语句中，在 $i < 3$ 条件后应该有一个分号，这是初学者需要特别注意的。



4.8 小结

通过本章的学习,可以让读者了解 Java 如何控制程序的执行和中断,学好这些知识可以为以后编写代码打下基础。其中对流程语句的讲解读者还可以参考电子工业出版社出版的《Java 优化编程(第 2 版)》一书进行更详细的学习。本章的重点是 for 循环语句和 if

条件语句的使用。下一章将学习数组的创建和操作。



4.9 习题

一、填空题

1. if 条件语句具有的三种形式分别为_____、_____、_____。
 2. 所谓 while 循环语句，就是先进行_____再进行_____。
 3. Java 中的循环语句包括_____、_____、_____。
 4. Java 中的跳转语句包括_____、_____。

二、选择题

1. 选择下面程序的运行结果 ()。

```
01 public class Hello
02 {
03     final static int x=0;
04     public static int y=1;
05     public static void main(String args[])
06     {
07         for(int k=0;k<3;k++)
08         {
09             switch(k)
10             {
11                 case x:System.out.println("0");
12                 case y:System.out.println("1");
13                 case y+1:System.out.println("2");
14             }
15         }
16     }
17 }
```

- A. 012
 - B. 第 11 行发生编译错误
 - C. 第 12 行发生编译错误
 - D. 第 12 行和第 13 行发生编译错误

2. 选择下面程序的运行结果 ()。

```
01  public class Hello
02  {
03      static boolean b1;
04      static boolean b2;
05      public static void main(String args[])
06      {
07          int x=1;
08          if(!b1)
09          {
10              if(!b2)
11              {
12                  b1=true;
13              }
14              if(!b1)
15              {
16                  x=x+10;
17              }
18          else if(b2=true)
19          {
20              x=x+100;
21          }
22      }
23  }
```



```
22         else if(b1|b2)
23         {
24             x=x+1000;
25         }
26     }
27     System.out.println(x);
28 }
29 }
```

- A. 1 B. 101 C. 110 D. 1101

3. 先来看下面的程序 ()。

```
01 public class Hello
02 {
03     public static void main(String args[])
04     {
05
06         System.out.println("x=" + x);
07     }
08 }
09 }
```

在该程序的第 5 行加入下面的哪一个选项可以得到如下结果 ()。

x=0
x=1

- A. for(int x=-1;x<2;++x){ B. for(int x=1;x<3;++x){

C. for(int x=0;x>2;++x){ D. for(int x=0;x<2; x++){

4. 选择该程序的运行结果 ()。

```
public class Hello
{
    public static void main(String args[])
    {
        int x=5;
        do
        {
            while(x<5)
            {
                System.out.println(x);
            }
        } while(x>5);
    }
}
```

- A. 5 B. 55
C. 没有任何输出 D. 编译错误

三、简答题

1. 简述 while 循环语句和 do...while 循环语句有什么不同。
2. 简述 break 中断语句和 continue 继续语句有什么不同。

四、编程题

1. 使用控制语句输出九九乘法表。
2. 开发一个程序，程序的功能是如果今天是周一到周五，就直接输出“上班”；如果今天是周六或者周日，就直接输出“休息”。

第5章 数组

在日常生活中，盒子的作用是存放东西，但是是不可能把衣服和食品放在一起的，会有专门放衣服的盒子，也会有专门放食品的盒子。在 Java 中，数组就好比日常生活中的盒子，用来存储数据。每一个数组也是有类型的，用来放相应类型的数据。数组是一种存储数据的数据结构。本章将介绍数组的创建和使用，以及其注意事项。通过本章的学习，读者应该能够实现下面的目标。

- 知道如何创建数组，包括创建一维数组和多维数组。
- 能够对数组进行初始化操作。
- 熟练掌握如何借助数组来解决实际问题。



5.1 如何创建数组

假设现在有 100 个苹果，如果分散存放会很不好管理，如果集中存放的话，大家会想到放在一个盒子里。在 Java 里也是这样，如果 100 个苹果每一个都放在一个变量里，会显得代码很凌乱，如果用数组存放会显得代码很整洁。所以说数组是一种存放数据的数据结构。下面讲解数组是如何创建的。

5.1.1 创建数组

定义 9 个 int 型的变量，分别存放 1~9 的数字。代码如下：

```
int i1 = 1;
int i2 = 2;
int i3 = 3;
int i4 = 4;
int i5 = 5;
int i6 = 6;
int i7 = 7;
int i8 = 8;
int i9 = 9;
```

这样定义完后代码看起来很长，使用起来也会很不方便。如果把上面的代码转换成数组，代码如下所示。

```
//定义了一个一维数组，长度为 9，数组的名称为 i
int i[] = {1,2,3,4,5,6,7,8,9};
```

【范例 5-1】下面是一个创建一维数组的程序。

示例代码 5-1

```
01  public class ChuangTian1
02  {
03      public static void main(String args[])
04      {
```



```
05         //定义了一个一维数组，长度为100，数组的名称为i
06         int i[] = new int[100];
07     }
08 }
```

【代码解析】创建了 int 型，长度为 100 的一维数组，数组下标从 0 到 99，共能存储 100 个数据。如果用数组存放数据将会使管理非常容易和方便。

创建数组后，在使用数组时需要注意只能使用数组长度里的元素，如果使用数组长度以外的下标进行访问就会发生错误。

【范例 5-2】下面是一个数组下标越界的程序。

示例代码 5-2

```
01 //类 TestArray 所描述的是一个测试数组的类
02 public class TestArray
03 {
04     public static void main(String args[])
05     {
06         //定义了 int 型的数组，长度为 10，名称为 k
07         int k[] = new int[10];
08         //对数组下标为 10 的元素进行赋值
09         k[10] = 11;
10     }
11 }
```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行该程序是会发生异常的。

【代码解析】上面一段代码将在编译的时候提示错误，因为数组的长度为 10，也就是说有 10 个元素。因为数组的下标为 0~9，所以 k[10] = 11; 是错误的。将 10 改为 0~9 的任意一个数字后，程序将编译通过。



注意：对数组进行访问时，不能使用超过数组长度的下标进行访问，否则就会发生异常。

数组除了能在创建后对其元素的值进行赋值，还能在创建的同时对其元素进行赋值。

```
//创建一个一维数组，名称为 i，其长度为 9，数组各个元素为 1,2,3,4,5,6,7,8,9
int i[] = {1,2,3,4,5,6,7,8,9}
```

5.1.2 创建多维数组

在一座楼中通常有多个单元，一个单元中又有多个房间。在 Java 中多维数组就是这种设计，多维数组是一种嵌套的数组，一维数组的每个元素又是一个一维数组。多维数组的创建如下所示。

【范例 5-3】下面是创建一个多维数组的程序。

示例代码 5-3

```
01 public class ChuangJian2
02 {
03     public static void main(String args[])
04     {
05         //定义了一个多维数组
```



```

06         int i[][] = new int[4][4];
07     }
08 }
```

【代码解析】创建了一个 int 型的多维数组，长度 4×4 ，即 16 个元素。

多维数组 i 的数据结构如下：

```

1-1 2 3 4
2-1 2 3 4
3-1 2 3 4
4-1 2 3 4
```



注意：多维数组同样也只能使用数组长度里的元素。这里包括多维数组中的任意一维都不能越界。

【范例 5-4】下面程序是对多维数组进行访问越界时的情况。

示例代码 5-4

```

01 //类 TestArray 所描述的是一个测试数组的类
02 public class TestArray
03 {
04     public static void main(String args[])
05     {
06         //定义了 int 型的数组，长度为 10，名称为 k
07         int k[][] = new int[4][4];
08         //对数组下标为 10 的元素进行赋值
09         k[1][5] = 6;
10     }
11 }
```

【代码解析】上面一段代码在编译的时候同样也提示错误，因为数组的长度为 4×4 ，也就是说有 16 个元素。因为该数组是二维数组，每一维的每个元素又是一个一维数组，所以元素为 16 个。因为数组的第二维的长度也是从 0 开始的，所以 $K[1][5]$ 是错误的。



5.2 数组的初始化

初始化是给数组中的元素进行赋值，数组的赋值有创建赋值和动态赋值。这就好比确定一座楼有多少个单元，一个单元中又有多少个房间。下面介绍数组的赋值方法，并举例说明。

5.2.1 创建并初始化数组元素

数组的创建初始化是数组创建完后，系统对各个元素进行的默认赋值，系统对各个基本类型的默认初值如下：

- boolean : false
- byte : 0
- char : '\u0000'
- short : 0
- int : 0



- long : 0L
- float : 0.0f
- double : 0.0

数组中对象类型的系统初始值默认为 null。

【范例 5-5】下面用代码演示各个基本数据类型的初始值。

示例代码 5-5

```
01 // testArray 类, 所描述的是测试数组初始值
02 public class Chushihua1
03 {
04     public static void main(String args[])
05     {
06         // 创建一个 boolean 型的数组, 长度为 2
07         boolean bo[] = new boolean[2];
08         // 创建一个 byte 型的数组, 长度为 2
09         byte b[] = new byte[2];
10         // 创建一个 char 型的数组, 长度为 2
11         char c[] = new char[2];
12         // 创建一个 short 型的数组, 长度为 2
13         short s[] = new short[2];
14         // 创建一个 int 型的数组, 长度为 2
15         int i[] = new int[2];
16         // 创建一个 long 型的数组, 长度为 2
17         long l[] = new long[2];
18         // 创建一个 float 型的数组, 长度为 2
19         float f[] = new float[2];
20         // 创建一个 double 型的数组, 长度为 2
21         double d[] = new double[2];
22         System.out.println(bo[0]);
23         System.out.println(b[0]);
24         System.out.println(c[0]);
25         System.out.println(s[0]);
26         System.out.println(i[0]);
27         System.out.println(l[0]);
28         System.out.println(f[0]);
29         System.out.println(d[0]);
30     }
31 }
```



图 5-1 各种数据类型的初始化值

【运行结果】使用 javac 编译程序, 然后使用 Java 运行编译产生的 class 程序, 运行结果如图 5-1 所示。

【代码解析】本程序中演示的是当没有对数组进行初始化时, 数组中元素的默认值。从运行结果中可以看出不同类型的默认值。



注意: 布尔型的默认值为 false, 这是最不容易记住的。

【范例 5-6】下面是对象类型的数组默认初始值。

示例代码 5-6

```

01  public class ChuShiHua2
02  {
03      public static void main(String args[])
04      {
05          //创建一个boolean型的数组, 长度为 2
06          Boolean bo[] = new Boolean[2];
07          //创建一个byte型的数组, 长度为 2
08          byte b[] = new Byte[2];
09          System.out.println(bo[0]);
10          System.out.println(b[0]);
11      }
12  }

```

【运行结果】使用 javac 编译程序, 然后使用 Java 运行编译产生的 class 程序, 运行结果如图 5-2 所示。

【代码解析】创建对象类型的数组, 数组的各个元素为对象类型, 因为没有指定其具体的对象, 所以元素的值为 null。



图 5-2 对象数组初始化

数组除了可以在创建的同时进行初始化, 也能在运行期间对数组各个元素进行赋值。对数组元素进行赋值通常使用 for 循环语句来进行。

【范例 5-7】下面是一个使用 for 循环为数组进行赋值的程序。

示例代码 5-7

```

01  //testArray 类, 所描述的是用 for 语句进行数组初始化
02  public class ChuShiHua3
03  {
04      public static void main(String args[])
05      {
06          //下面创建一个 int 型的数组, 数组的长度为 10
07          int a[] = new int[10];
08          for(int i = 0; i < a.length; i++)
09          {
10              a[i] = i + 1;
11              System.out.println("数组的各个元素的值为 : " + a[i]);
12          }
13      }
14  }

```



图 5-3 循环赋值

【运行结果】使用 javac 编译程序, 然后使用 Java 运行编译产生的 class 程序, 运行结果如图 5-3 所示。

【代码解析】创建了 int 型的数组 a, 长度为 10 个元素, 其下标是从 0 开始, 到 9 结束。通过从 0 开始循环到数组的长度, 把每次循环的变量 i 的值赋值给数组当前的元素, 打印并显示数组的各个元素的值。



5.3 数组操作的举例

前面章节介绍了创建一维数组，以及多维数组的方法，并演示了数组的初始化等操作。下面介绍操作数组的常用方法。

5.3.1 数组元素值的复制

数组里各个元素的值可以用数组的引用和使用循环对其进行赋值，但要注意两个不同长度的数组进行复制的时候下标越界的问题。下面通过代码来演示。

【范例 5-8】用 for 循环演示对数组各个元素的值的复制。

示例代码 5-8

```
01 // FuZhi1 类，所描述的是用 for 语句进行数组各元素的复制
02 public class FuZhi1
03 {
04     public static void main(String args[])
05     {
06         //下面创建一个 int 型的数组，数组的长度为 10
07         int a[] = new int[10];
08         int b[] = new int[a.length];
09         System.out.print("数组 a 的各个元素的值为： ");
10         //先通过 for 循环对数组 a 进行初始化赋值
11         for(int i = 0; i < a.length; i++)
12         {
13             a[i] = i + 1;
14             //打印数组 a 的各个值
15             System.out.print(a[i] + " ");
16         }
17         System.out.println();
18         System.out.print("数组 b 的各个元素的值为： ");
19         for (int j = 0; j < b.length; j++)
20         {
21             b[j] = a[j];
22             System.out.print(b[j] + " ");
23         }
24         System.out.println();
25     }
26 }
```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 5-4 所示。



图 5-4 数组元素值复制

【代码解析】用 for 循环语句来对数组 a 各个元素进行初始化，即对其进行赋值；再通过 for

循环遍历数组 a 的各个元素并把值赋给数组 b。



注意：两个不同长度的数组进行复制的时候一定要注意下标越界的问题。

【范例 5-9】修改上面的代码使其看起来更简洁。

示例代码 5-9

```

01 //Fuzhi2 类, 所描述的是用 for 语句进行数组初始化
02 public class Fuzhi2
03 {
04     public static void main(String args[])
05     {
06         //下面创建一个 int 型的数组, 数组的长度为 10
07         int a[] = new int[10];
08         int b[] = new int[a.length];
09
10         //初始化数组 a 的各个元素的值, 并把值赋给数组 b 的各个元素
11         for(int i = 0; i < a.length; i++)
12         {
13             a[i] = i + 1;
14             //将值赋给数组 b 的各个元素
15             b[i] = a[i];
16
17             //打印并显示数组 b 的各个值
18             System.out.print(b[i] + " ");
19         }
20         System.out.println("");
21     }
22 }
```

【运行结果】使用 javac 编译程序, 然后使用 Java 运行编译产生的 class 程序, 运行结果如图 5-5 所示。

【代码解析】这段代码和前面代码的结果是一样的, 区别就是在对数组 a 的各个元素的值初始化的同时把值赋给数组 b 的各个元素。

【范例 5-10】用方法 arraycopy 对数组进行值的复制。

示例代码 5-10

```

01 //FuZhi3 类, 所描述的是用 for 语句进行数组初始化并复制数组
02 public class FuZhi3
03 {
04     public static void main(String args[])
05     {
06         //下面创建一个 int 型的数组, 数组的长度为 10.
07         int a[] = new int[10];
08         int b[] = new int[a.length];
09         //初始化数组 a 的各个元素的值, 并把值赋给数组 b 的各个元素
10         for(int i = 0; i < a.length; i++)
11         {
12             a[i] = i + 1;
13             //通过方法 arraycopy 将值赋给数组 b 的各个元素
14             System.arraycopy(a, 0, b, 0, 10);
15             //打印并显示数组 b 的各个值
16             System.out.print(b[i] + " ");
17         }
18     }
19 }
```



```
18         System.out.println("++");
19     }
20 }
```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 5-6 所示。



图 5-5 修改后复制数组值

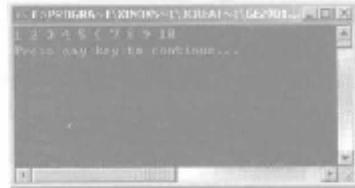


图 5-6 使用 arraycopy 方法复制

【代码解析】这个例子和前面两段代码的效果是一样的，都是对数组各个元素的复制。方法 arraycopy (src, srcPos, dest, destPos, length) 各个参数的表示含义如下所述。

- src : src 为 object 类型的数组引用，即要复制的数组。
- srcPos: srcPos 为 int 型的要复制数组的下标值，即要复制数组的开始位置。
- dest : dest 为 object 类型的新数组引用。
- destPos : destPos 为 int 型新数组的下标开始值。
- length : length 为 int 型的数组的长度。

5.3.2 数组元素的排序

数组元素的排序在数组的操作中是很常见的。下面分别介绍两种数组元素的排序方法。

【范例 5-11】下面是使用冒泡排序法对数组中元素进行排序的程序。

示例代码 5-11

```
01 public class PaiXu
02 {
03     public static void main(String args[])
04     {
05         //定义一个数组a
06         int a[] = { 1, 3, 2, 5, 6, 8, 4 };
07         for (int i = 0; i < a.length; i++)
08         {
09             //第二层循环从第一层循环的元素后面哪个元素开始
10             for (int j = i + 1; j < a.length; j++)
11             {
12                 //如果第二个元素比第一个小的话就换位置
13                 if (a[j] < a[i])
14                 {
15                     int n = a[i];
16                     a[i] = a[j];
17                     a[j] = n;
18                 }
19             }
20         }
21         //打印数组各个元素
22         for (int i = 0; i < a.length; i++)
23         {
24             System.out.println(a[i]);
25         }
26     }
27 }
```

```

26     }
27 }

```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 5-7 所示。

【代码解析】一个排序错乱的数组通过第二个元素和第一个元素的比较，如果第二个元素小的话就换位置，再和下一个元素进行比较，直到元素顺序正确为止。

【范例 5-12】用 sort 方法进行数组排序。

示例代码 5-12

```

01 //引入包 Arrays，因为 Arrays.sort 需要此包
02 import java.util.Arrays;
03 public class PaiXu2
04 {
05     public static void main(String[] args)
06     {
07         //定义一个数组 a，并确定其元素
08         int[] a = {12, 3, 19, 2, 10, 13, 91};
09         //用 sort 方法对数组进行排序
10         Arrays.sort(a);
11         System.out.println("数组排序后：");
12         //调用方法进行数组排序的显示
13         printa(a);
14     }
15     private static void printa(int[] a)
16     {
17         for (int i = 0; i < a.length; i++)
18         {
19             System.out.println("a[" + i + "]=" + a[i] + " ");
20         }
21     }
22 }

```



图 5-7 冒泡排序法

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 5-8 所示。

【代码解析】在本程序中使用 sort(a)方法对数组进行排序。sort(a)方法中具有一个参数，该参数表示一个数组类型，是要排序的数组。

5.3.3 在数组里查找指定元素

下面演示用代码怎么实现在数组里查找指定的元素。

【范例 5-13】利用 for 循环语句来查找。

示例代码 5-13

```

01 public class Chazhaol
02 {

```



```
03     public static void main(String[] args)
04     {
05         //定义一个数组a，并确定其元素
06         int[] a = {12, 3, 19, 2, 10, 13, 9};
07         //要查找的元素2
08         int n = 2;
09         for(int i = 0; i < a.length; i++)
10         {
11             //如果数组的第i个元素和n相等的话
12             if (a[i] == n)
13             {
14                 System.out.println("要查找的元素在数组的第" + (i + 1) + "个位置");
15             }
16         }
17     }
18 }
```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 5-9 所示。

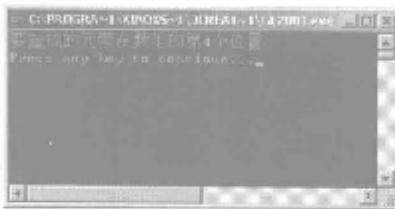


图 5-9 使用 for 循环查找

【代码解析】利于 for 循环遍历数组里每一个元素，如果是要查找的元素就把其位置打印出来。

5.3.4 利用数组打印 26 个英文字母

在实际开发中，有时会让开发员编写一个生成连续字母的字符串，该功能利用 26 个英文字母的 ASCII 码结合 for 循环来完成。

【范例 5-14】下面是一个显示生成连续字母的程序。

示例代码 5-14

```
01 public class ChaZhao2
02 {
03     public static void main(String[] args)
04     {
05         //定义了一个 char 型的数组 a
06         char[] a;
07         //实例对象数组，长度为 26
08         a = new char[26];
09         for (int i = 0; i < 26; i++)
10         {
11             //通过把 A 的 ASCII 码和循环变量进行相加来转换各个字母的 ASCII 码
12             a[i] = (char) ('A' + i);
13             System.out.print(a[i]);
14             if(a[i]=='Z')
15             {
16                 System.out.println("");
17             }
18 }
```

```

18
19
20

```

【运行结果】使用 javac 编译程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 5-10 所示。

【代码解析】先创建一个 char 型的数组，来存放 26 个英文字母。在循环的时候把 A 转换为 ASCII 码和循环变量进行相加，并打印出 char 型数组的各个元素。

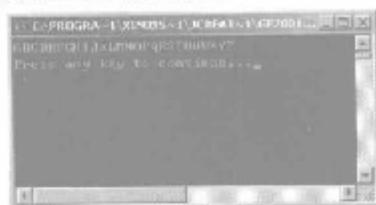


图 5-10 显示连续字母

 **注意：**循环打印字母时，必须要进行强制类型转换，从而能够进行循环操作。



5.4 综合练习

1. 利用数组打印连续小写字母

【提示】可以使用示例代码 5-14 的形式作为参考来进行该程序的开发。

```

01  public class Lianxi1
02  {
03      public static void main(String[] args)
04      {
05          //定义了一个char型的数组 a
06          char[] a;
07          //实例对象数组，长度为 26
08          a = new char[26];
09          for (int i = 0; i < 26; i++)
10          {
11              //通过把 A 的 ASCII 码和循环变量进行相加来转换各个字码的 ASCII 码
12              a[i] = (char) ('a' + i);
13              System.out.print(a[i]);
14              if(a[i]=='z')
15              {
16                  System.out.println("结束");
17              }
18          }
19      }
20  }

```

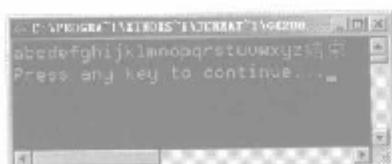


图 5-11 练习题 1

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 5-11 所示。

2. 对已有数组元素按照一定规则进行排序

【提示】可以使用 sort 方法将数组中的所有数组元

素按照从小到大的顺序进行排列。

```

01 //引入包 Arrays，因为 Arrays.sort 需要此包
02 import java.util.Arrays;
03 public class Lianxi2

```



```

04  {
05      public static void main(String[] args)
06  {
07          //定义一个数组 a，并确定其元素
08          int[] a = {22, 33, 11, 56, 5};
09          System.out.println("数组排序前:");
10          paiXu(a);
11          //用 sort 方法对数组进行排序
12          Arrays.sort(a);
13          System.out.println("数组排序后:");
14          //调用方法进行数组排序的显示
15          paiXu(a);
16      }
17      private static void paiXu(int[] a)
18  {
19      for (int i = 0; i < a.length; i++)
20  {
21      System.out.println("a["+i+"]=" + a[i]
22  }
23  }
24 }

```

【运行结果】 使用 `javac` 编译程序将产生一个和该程序对应的 `class` 程序, 然后使用 `Java` 运行编译产生的 `class` 程序, 运行结果如图 5-12 所示。

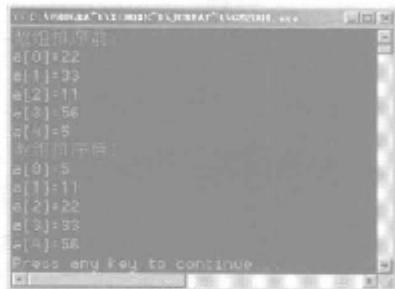


图 5-12 对元素排序



5.5 小结

本章通过对数组的创建, 以及数组初始化的讲解, 让读者对数组有了基本的了解; 并演示了数组的常用方法等操作, 且对前面的知识进行了巩固。本章的重点是对数组的创建和初始化。如果读者想了解更多的关于数组的内容, 可以参考电子工业出版社出版的《JAVA 面向对象编程》一书进行学习。



5.6 习题

一、填空题

- 在访问数组元素时, 是通过数组_____进行访问的。
- 数组下标从_____开始的。
- _____是给数组中的元素进行赋值, 数组的赋值有_____和_____。

二、选择题

- 选择下面正确声明数组的选项 ()。

A. <code>int [] myArray [];</code>	B. <code>int [] myArray ;</code>
C. <code>int myArray [];</code>	D. <code>int [5] myArray;</code>
- 先看下面的程序。

```

01  public class Hello
02  {

```

```
03     public static void main(String args[])
04     {
05         int [][] b1=new int[2][3];
06         int b2=5;
07         int [][][][][] b3 = new int[2][3][4][5];
08     }
09 }
```

选择能够放在第 8 行正常编译的选项 ()。

- A. `b1[0][1]=b2;` B. `b3[0][1][2]=b1;`
C. `b3[0][1][2][3]=b2;` D. `b1[1]=b2;`

3. 选择下面正确声明和初始化数组的选项 ()。

A. `int [] a={1,2,3,4};` B. `int a []=new int(5);`
C. `int a []=new int [5];` D. `int [] b;`

4. 选择下面正确声明和初始化数组的选项 ()。

A. `int [] a={"1","2","3","4"};` B. `int a []=new int(5);`
C. `int a [][]={1,2,3,4};` D. `int [] a={1,2,3,4};`

三、简答题

- ### 1. 简述多维数组的含义。

四、编程题

1. 编写一个对数组元素进行复制的程序。
 2. 编写一个对 `int a[] = { 12, 26, 27, 58, 6, 89, 42 }` 数组进行排序的程序。
 3. 编写一个输出从 E 到 H 连续字母的程序。

第6章 类与对象

在日常生活中，盖房子之前要首先设计一个建筑图纸，然后根据图纸来盖房子。在 Java 中，类好比在日常生活中描述一个物品的信息，如房子的建筑图纸；而对象就好比实实在在的房子。本章将要介绍类的定义、类成员变量的定义和方法的定义、方法的参数等知识。通过本章的学习，读者应该能够实现如下几个目标。

- 了解什么是面向对象。
- 熟悉 Java 中的类并能够进行类的操作。
- 掌握成员变量和局部变量的区别。
- 掌握 Java 程序中方法的创建和使用。



6.1 什么是面向对象

所谓面向对象，是指编写程序的时候要围绕着一个对象的功能进行编写。本节将要介绍面向对象的特点，以及它与面向过程编程的区别。

6.1.1 面向对象编程的特点

面向对象编程的英文缩写是 OOP，全称为 Object Oriented Programming。在进行面向对象编程时，方法和成员变量都写在具体的对象里，并对其成员变量和方法有很好的隐藏性；对象之间的访问都是通过其接口进行的。下面列举面向对象编程的特点，分为如下几种。

首先说的是继承。继承是发生在类与类之间的，是子类共享父类成员变量和方法的一种模式。通过扩展子类的方法可以使子类有比父类更加强大的功能。



说明：继承是面向对象编程的特点，同样也是 Java 的特点，这里和其他语言有很大不同。

【范例 6-1】下面代码演示类的继承。

示例代码 6-1

```
01 //bike 类描述的是一个自行车
02 class bike
03 {
04 }
05
06 // racing_cycle 类描述的是一个公路赛车，继承自 bike
07 class racing_cycle extends bike
08 {
09 }
```

【代码解析】在继承一个类的时候，可以在一个已经存在的类的基础之上进行继承。



通过继承共享了父类的成员变量和方法，并加入新的方法使其子类比父类更加强大。



提示：继承是发生在类与类之间的。继承可以是单继承，也可以多层继承。

多态是指对象在运行期和编译期具有两种状态，多态的使用使代码具有了更多的灵活性和重用性。

抽象是指在定义类的时候，确定了该类的一些行为和动作。比如自行车可以移动，但怎么移动不进行说明。这种提前定义一些动作和行为的类为抽象的。

封装是指对一件物品的描述信息是这个物品所特有的，是不能让外界看到的一些成员变量和方法。在 Java 里成员变量和方法就被封装在类里，需要通过一些特有的方法访问它们。

6.1.2 面向对象编程与面向过程编程的区别

面向过程是指在遇到问题的时候，怎么去解决这个问题，而分析问题的步骤，就是解决这个问题的方法，是通过方法一步一步来完成的。面向对象是指在遇到问题的时候，把问题分解成各自独立功能的类，而这个类是完成各自问题的。总结如下所述。

- 面向过程和面向对象最明显的区别就是，面向对象是按照要完成的功能来实现的，而面向过程是按照解决这个问题的步骤来实现的。
- 面向对象是按照程序中的功能进行划分的。
- 面向过程是按照问题的解决思路来划分的，是一步一步来解决问题的。
- 面向过程更看重的是完成问题的过程。
- 面向对象更看重的是功能，通过各种功能模块的组合来完成问题。



6.2 什么是类

类是一种抽象的东西，描述的是一个物品的完整信息，比如房子和图纸的关系。在 Java 里，图纸就是类，定义了房子的各种信息，而房子是类的实体。

6.2.1 类的定义和对象的创建

定义一个类表示定义了一个功能模块。下面先介绍如何定义一个类，以及如何创建这个类的实例，即对象。类是通过关键字 `class` 来定义的，在 `class` 关键字后面加上类的名称，这样就创建了一个类。在类里面可以定义类的成员变量和方法。类的语法代码如下所示。

```
class 类的名称
{
    //类的成员变量
    //类的方法
}
```

创建类的实例是通过 `new` 关键字来定义的，后面加上定义类时为类起的名称，需要注意的是在类名后还需要一个括号。创建类的实例的代码如下所示。

```
new 类的名称();
```

【范例 6-2】下面用代码来演示创建类。



示例代码 6-2

```
01 //bike 类,描述的是一个自行车
02 class bike
03 {
04     //自行车的颜色
05     String color;
06     //自行车所具有的方法
07     void getMes()
08     {
09     }
10 }
```

【代码解析】在本程序中创建了一个名称为 bike 的类，在该类中定义了一个表示自行车颜色的成员变量 color，还定义了一个叫做 getMes 的方法，但是在该方法中不做任何事情。下面的语句创建类的实例。

```
//创建一个 bike 类的对象实例,对象实例的名称为 b,即对象引用
bike b = new bike();
```

这并不是一个完整的程序，只是演示如何创建类的实例。其中 b 是为创建类的实例起的名称，它也是创建类实例的对象引用。

【范例 6-3】下面演示一个完整的类定义和对象创建的例子。

示例代码 6-3

```
01 //bike 类,描述的是一个自行车
02 class bike
03 {
04     //自行车的颜色
05     String color = "黄色";
06     //自行车所具有的方法
07     void getMes()
08     {
09         System.out.println("类的方法");
10     }
11     //main 方法为运行一个类的主入口方法
12     public static void main(String args[])
13     {
14         //创建一个 bike 类的对象实例,对象实例的名称为 b,即对象引用
15         bike b = new bike();
16         //显示此类的颜色,并显示出来
17         System.out.println(b.color);
18         //调用 bike 类里的 getMes()方法
19         b.getMes();
20     }
21 }
```

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 6-1 所示。

【代码解析】用 new 关键字创建对象 bike，这个对象在内存中是存在的。用 b 表示在内存中对这个 bike 类的对象的引用，使用 b 就能使用这个对象的数据。

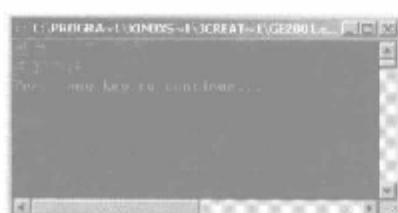


图 6-1 创建类



6.2.2 如何使用现有类

在定义一些类的时候，如何使用它们呢？这里需要分为多种情况。定义的类可以在一个包下面，也可以不在一个包下面，这在使用时是不同的。类又分为已有类和自定义类，它们之间的使用也是有区别的。下面就通过范例来讲解在不同情况下如何使用类。

【范例 6-4】在同目录下使用类。首先是定义一个 bike 类，在该类中不存在任何成员变量和方法，这里只是演示如何在同一目录下使用类。

示例代码 6-4

```
01 //bike.java
02 class bike
03 {
04 }
```

接下来定义一个使用 bike 类的类。

```
01 //testBike.java
02 //在 testBike 类里使用了 bike 类
03 class TestBike
04 {
05     bike b = new bike();
06 }
```

【代码解析】从程序中可以看到，当使用的类和被使用的类在同一目录下时，就可以直接使用创建类的实例，从而使用该类。

【范例 6-5】在不同目录下使用类。同样还是创建一个 bike 类，只是这次比上一个范例中多出了使用包的程序语句。使用包后将使编译后的 class 程序不在当前目录下，而是放在定义的包目录下，从而使使用的类和被使用的类不在同一目录下。

示例代码 6-5

```
01 //bike.java
02
03 //bike 类在 a 目录下，即 a 包
04 package a;
05
06 class bike
07 {
08 }
```

接下来定义使用 bike 类的类。

```
01 //testBike.java
02
03 //引入 a 目录下的 bike 类
04 import a.bike;
05 //在 testBike 类里使用了 bike 类
06 class TestBike
07 {
08     bike b = new bike();
09 }
```

【代码解析】从程序中可以看到如果想使用不同目录下的类，则首先需要使用 import 将该类引入到程序中，其中 a 是在定义 bike 类时定义的包。引入 bike 类后，就可以像在



同一目录下一样来使用 bike 类。

【范例 6-6】使用系统自带的类。

示例代码 6-6

```
01 // test 类
02 public class test
03 {
04     public static void main(String[] args)
05     {
06         //String 类为 Java 自带的类，描述的是一个字符串
07         String s = new String();
08     }
09 }
```

【代码解析】该程序中使用了 Java 语言中自带的类 String 类，从程序中可以看到当使用 String 类时，也是可以直接使用的。Java 语言自带的类分为两种，这里的 String 类是位于 lang 包下，所以是不需要引入的。但是如果在别的包下，在使用时就需要使用 import 将包引入。

6.2.3 类设计的技巧

设计一个类要明确这个类所要完成的功能，类里的成员变量和方法是描述类的功能的。如果定义了和这个类不相关的成员变量和方法将不是一个良好的设计。

【范例 6-7】示例代码 6-7 是一个不太好的类设计。

示例代码 6-7

```
01 public class bike
02 {
03     //这个成员变量描述的是自行车的颜色
04     String color = "黄色";
05
06     //这个成员变量描述的是公路赛车的颜色，所以在这里不太合适
07     String racing_color = "绿色";
08 }
```

【代码解析】在本程序中定义了一个表示自行车颜色的 color 成员变量，又定义了一个表示赛车颜色的 racing_color 成员变量；而该程序是定义的一个 bike 自行车类，所以定义表示赛车颜色的 racing_color 成员变量是不太好的选择。

【范例 6-8】示例代码 6-8 是一个良好的类设计。

示例代码 6-8

```
01 public class bike
02 {
03     //这个成员变量描述的是自行车的颜色
04     String color = "黄色";
05 }

01 public class racing
02 {
03     //这个成员变量描述的是公路赛车的颜色
04     String racing_color = "绿色";
05 }
```



【代码解析】在该范例中，定义了两个类。其中 bike 类中只定义了一个表示自行车颜色的 color 成员变量。同样在 racing 类中只定义了一个表示赛车颜色的 racing_color 成员变量。这种设计相对上一个范例中的设计要好得多，这样使类和成员变量相对应，也使别人更容易读懂代码。



6.3 成员变量

成员变量就是这个类里定义的一些私有变量，这些变量属于这个类。这就好比日常生活中的自行车的大小，即这个车子是 26 还是 28 的，这个尺寸就是自行车的成员变量，是描述这个自行车的。下面开始介绍成员变量。

6.3.1 成员变量的创建

成员变量描述的是这个类的一些属性或状态，下面通过代码来演示怎么定义成员变量。语法如下：变量的类型 变量的名称

【范例 6-9】 创建成员变量的一般形式。

示例代码 6-9

```

01 //bike 类描述的是一个自行车
02 public class bike
03 {
04     //这个成员变量描述的是自行车的颜色
05     String color;
06
07     //这个成员变量描述的是自行车的大小，即尺寸
08     String size;
09 }
```

【代码解析】在该程序中，定义了一个叫做 bike 的类，在该类中定义了两个成员变量，一个是表示自行车颜色的 color 成员变量，一个是表示自行车型号的 size 成员变量。

【范例 6-10】 下面看一个创建成员变量的完整形式。

示例代码 6-10

```

01 //bike 类描述的是一辆自行车
02 public class bike2
03 {
04     //这个成员变量描述的是自行车的颜色
05     String color;
06
07     //这个成员变量描述的是自行车的大小，即尺寸
08     String size;
09
10     //程序的运行函数即主入口函数
11     public static void main(String args[])
12     {
13         //创建 bike 类的对象实例，即 bike 类的对象引用 b
14         bike2 b = new bike2();
15
16         //给对象引用 b 的成员变量赋值
17         b.color = "黄色";
```



```

18         b.size = "26 尺寸";
19
20         //打印并显示成员变量
21         System.out.println("该自行车的颜色为: " + b.color);
22         System.out.println("该自行车的型号为: " + b.size);
23     }
24 }

```



图 6-2 创建成员变量

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 6-2 所示。

【代码解析】通过 new 关键字来创建这个 bike 类的对象，用 bike 类的对象引用 b 来给其成员变量赋值。因为成员变量是在这个类实例化后才能访问到的。成员变量赋完值后，调用 println 语句来打印并显示结果。

6.3.2 成员变量的初始化

通过 new 关键字来创建一个对象后，会有一个系统默认的初始值，所以不管有没有在创建成员变量的时候给变量一个值，系统都会有一个默认的值。

成员变量和对象的引用在声明的时候不对其赋初值，那么系统都会赋一个初值，具体的信息如表 6-1 所示。

表 6-1 成员变量和对象引用的系统默认值

类 型	默 认 值
Byte	0
Char	'\u0000'
Boolean	false
Short	0
Int	0
Long	0L
Float	0F
Double	0.0D
对象引用	null

【范例 6-11】下面是一个基本类型的例子。

示例代码 6-11

```

01 //test 类描述的是基本类型的初始化
02 public class test
03 {
04     //基本类型的成员变量
05     int size;
06     boolean b;
07
08     //程序的运行函数即主入口函数
09     public static void main(String args[])

```



```

10  {
11      //创建 bike 类的对象实例, 即 bike 类的对象引用 t
12      test t = new test();
13
14      //打印并显示成员变量
15      System.out.println(t.size);
16      System.out.println(t.b);
17
18      //给对象引用 t 的成员变量赋值
19      t.size = 123;
20      t.b = true;
21
22      //打印并显示成员变量
23      System.out.println(t.size);
24      System.out.println(t.b);
25  }
26 }

```

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序, 然后使用 Java 运行编译产生的 class 程序, 运行结果如图 6-3 所示。

【代码解析】比较两次的打印结果, 开始发现成员变量 size 和 b 的值是不一样的。第一次在创建对象后打印其结果, 打印出来的值就是系统默认的初始值。第二次给成员变量赋值后, 打印后就是赋值后的值了。

【范例 6-12】下面是一个基本类型和对象引用的例子。

示例代码 6-12

```

01  //test 类描述的是基本类型和对象引用的初始化
02  public class test2
03  {
04      //基本类型的成员变量
05      int size;
06      boolean b;
07
08      //String 类为一个封装类, s 为一个对象引用.
09      String s;
10
11     //程序的运行函数即主入口函数
12     public static void main(String args[])
13     {
14         //创建 bike 类的对象实例, 即 bike 类的对象引用 t
15         test2 t = new test2();
16
17         //打印并显示成员变量
18         System.out.println(t.size);
19         System.out.println(t.b);
20         System.out.println(t.s);
21
22         //给对象引用 t 的成员变量赋值
23         t.size = 123;
24         t.b = true;
25         t.s = "为一个对象引用类型";
26     }

```



图 6-3 初始化成员变量



```
27     //打印并显示成员变量
28     System.out.println(t.size);
29     System.out.println(t.b);
30     System.out.println(t.s);
31 }
32 }
```



图 6-4 对象引用

null 正是系统默认的初始值。

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 6-4 所示。

【代码解析】这个例子和上一个例子类似，不同点就是加入了对象引用类型。对象引用 s 的类型为一个 String 类型，是一个封装类。在创建对象后，打印其对象引用 s 的值为 null，这个

【范例 6-13】在创建成员变量的时候进行初始化。

示例代码 6-13

```
01 //test 类描述的是基本类型和对象引用的初始化
02 public class test3{
03 {
04     //基本类型的成员变量
05     int size = 26;
06     boolean b = true;
07
08     //String 类为一个封装类，s 为一个对象引用
09     String s = "为一个对象引用类型";
10
11     //程序的运行函数即主入口函数
12     public static void main(String args[])
13     {
14         //创建 bike 类的对象实例，即 bike 类的对象引用 b
15         test3 t = new test3();
16
17         //打印并显示成员变量
18         System.out.println(t.size);
19         System.out.println(t.b);
20         System.out.println(t.s);
21     }
22 }
```

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 6-5 所示。

【代码解析】在创建成员变量的同时对其进行赋值，系统检测这个成员变量有默认值的话就不再对其进行默认初值赋值了。创建完对象实例后，打印其结果，可以看出结果就是创建的时候所赋的值。

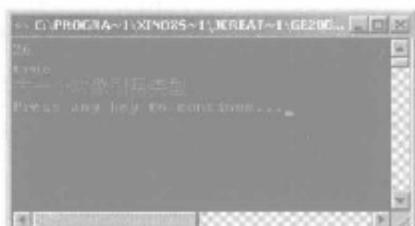


图 6-5 创建时初始化



6.4 局部变量

局部变量和成员变量很相似，都是描述信息的。局部变量和成员变量的不同点就是局部变量是在方法体里创建的，在方法体外是访问不到这个变量的。

6.4.1 局部变量的创建和初始化

局部变量描述的是方法体的一些属性或状态的，下面通过代码来演示怎么定义局部变量。创建局部变量的基本语法如下：

变量的类型 变量的名称

【范例 6-14】下面是演示局部变量的例子。

示例代码 6-14

```

01 //test 类描述的是基本类型的初始化
02 public class test4
03 {
04     //程序的运行方法，即主入口方法
05     public static void main(String args[])
06     {
07         //基本类型的局部变量
08         int size = 123;
09         boolean b = true;
10         //打印并显示局部变量
11         System.out.println(size);
12         System.out.println(b);
13     }
14 }
```

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 6-6 所示。

【代码解析】在本程序中定义了两个局部变量，并显示这两个变量，该程序是很简单的。

【范例 6-15】没有给局部变量赋值的例子。



图 6-6 局部变量

示例代码 6-15

```

01 //test 类描述的是基本类型的初始化
02 public class test
03 {
04     //程序的运行方法，即主入口方法
05     public static void main(String args[])
06     {
07         //基本类型的局部变量
08         int size;
09         boolean b;
10
11         //创建 bike 类的对象实例，即 bike 类的对象引用 b
12         test t = new test();
13
14         //打印并显示局部变量
```



```
15         System.out.println(size);
16     }
17 }
18 }
```

【运行结果】使用 `javac` 编译程序将产生一个和该程序对应的 `class` 程序, 然后使用 Java 运行编译产生的 `class` 程序, 运行将产生错误, 错误信息如下所示。

```
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
  The local variable size may not have been initialized
  The local variable b may not have been initialized

  at test.main(test.java:15)
```

【代码解析】这两个例子的区别就是没有给局部变量赋初值, 只要给局部变量赋值就可解决此问题。局部变量是不对其赋默认值的。

6.4.2 局部变量和成员变量的区别

局部变量描述的是这个方法体内的属性, 而成员变量描述的是这个对象里的属性的, 它们之间的区别, 即访问区别如下:

- 成员变量可以被 `public`、`protected`、`default`、`private`、`static`、`final` 修饰符修饰。
- 局部变量可以被 `final` 修饰符修饰, 但不能修饰为 `public`、`protected`、`default`、`private`、`static`。
- 成员变量是在堆里进行创建的, 而局部变量是在栈里进行创建的。
- 成员变量是系统默认值。
- 局部变量没有系统默认值, 必须手动赋值。



6.5 方法

每个人都有走、吃和睡等动作。在 Java 中, 方法就好比日常生活中的一个动作, 是用来完成一系列操作的。方法收到对象的信息, 并进行处理的操作。

6.5.1 方法的创建和参数

方法的参数是提供外界在执行方法的时候提供给方法的特殊描述信息的, 在本程序中定义了一个 `add` 方法。当调用该方法时, 需要给该方法传递两个参数, 从而求出这两个参数的和。

```
方法修饰符 方法的返回类型 方法名称(方法参数)
{
    方法体
}
```

方法的定义如下所示。

```
public void add(int i, int n)
{
    System.out.println(i+n);
}
```

代码说明:

- 方法名称为 add，有两个参数都是 int 类型的。
- 方法体是打印 $i+n$ 的值，并显示出来。
- 方法的修饰符为 public 类型的，修饰符可有可无。

【范例 6-16】下面用一个完整的例子来演示创建方法。

示例代码 6-16

```

01 //test 类描述的是基本类型的初始化
02 public class test6
03 {
04     //基本类型的成员变量
05     int add1 = 123;
06     int add2 = 222;
07
08     //创建了一个方法，参数有两个，都为 int 类型的
09     public void add(int i, int n)
10     {
11         System.out.println(i+n);
12     }
13
14     //程序的运行方法，即主入口方法
15     public static void main(String args[])
16     {
17         //创建 bike 类的对象实例，即 bike 类的对象引用 b
18         test6 t = new test6();
19
20         //执行方法 add，并传入两个参数值
21         t.add(t.add1, t.add2);
22     }
23 }
```

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 6-7 所示。

【代码解析】通过创建了对象实例后，将两个成员变量作为方法的参数传入到方法 add 中，进行计算。

方法的返回类型有很多种，主要分为如下几类。

- 方法返回值为 void 类型时为无返回值。
- 方法返回值还可以为任意的类型，如 String、Boolean、int。如果定义了方法的返回类型就必须在方法体内用 return 把返回值进行返回。
- 方法的返回值可以为 null，但必须是对象类型。基本类型不能返回 null。
- 在返回值为基本类型的时候，只要能够自动转换就可返回。

方法的参数也有多种形式，下面是对方法参数的讨论。

- 方法的参数可以为基本数据类型，也可以为对象引用类型。
- 每个参数都有完整的声明该变量的形式。
- 方法的参数可以有一个，也可有多个。
- Java 程序的入口 main 就为一个方法，参数为 String[] args，它是个特殊的方法。

接下来就来讨论一下方法的调用问题。方法是定义在对象里的，只能通过其对象引用来调用。通过对象引用名称，加上“.”点号，再加上方法名称来进行访问的。

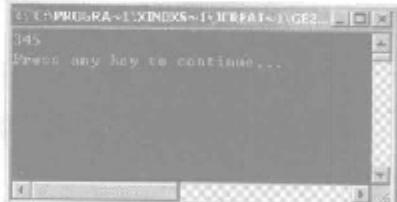


图 6-7 方法



【范例 6-17】下面通过一个例子来说明方法的创建。

示例代码 6-17

```
01 public class test7
02 {
03     public void add(int i, int n)
04     {
05         System.out.println(i+n);
06     }
07
08     //程序的运行方法, 即主入口方法
09     public static void main(String args[])
10     {
11         //创建 bike 类的对象实例, 即 bike 类的对象引用 b
12         test7 t = new test7();
13
14         //执行方法 add, 并传入两个参数值
15         t.add(11,22);
16     }
17 }
```



图 6-8 方法调用

【运行结果】使用 `javac` 编译程序将产生一个和该程序对应的 `class` 程序, 然后使用 Java 运行编译产生的 `class` 程序, 运行结果如图 6-8 所示。

【代码解析】在本程序中定义了一个 add 方法，该方法需要两个参数，方法的功能是将两个参数相加并显示出来。当调用该方法时，需要给该方法传递两个参数，从而也求出这两个参数的和。

6.5.2 方法参数的传递

参数的传递是传递的值还是引用呢？下面通过例子来分别说明，请读者仔细考虑。

[案例 6-18]当传递类型为基本类型时，传递的是该类型的值。下面通过代码来演示。

示例代码 6-18

```
01 //test 类描述的是基本类型的传递
02 public class test
03 {
04     //方法 add 是把传入的参数进行+1，并显示其结果
05     public void add(int i)
06     {
07         i = i + 1;
08         System.out.println(i);
09     }
10
11     //程序的运行方法，即主入口方法
12     public static void main(String args[])
13     {
14         //基本类型的局部变量
15         int size = 44;
16
17         //创建 bike 类的对象实例，即 bike 类的对象引用 b
18         test t = new test();
19     }
20 }
```

```

20     //打印原来的值
21     System.out.println(size);
22     //运行时的值
23     t.add(size);
24     //打印运行后的值
25     System.out.println(size);
26 }
27

```

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 6-9 所示。

【代码解析】在参数为基本类型进行传递的时候，是传递的这个值的备份，即第二份。不论在方法中怎么改变这个备份，都不是操作原来的数据，所以原来的值是不会改变的。

【范例 6-19】当传递的参数为对象引用类型时，也是利用的传值的方式进行的。下面通过代码说明。

示例代码 6-19

```

01 //test 类描述的是方法的传递
02 public class test
03 {
04     public static void main(String[] args)
05     {
06         //创建一个对象类型
07         String s = new String("Hello ");
08
09         //打印其值
10         System.out.println("before : " + s);
11
12         //通过方法去改变其值
13         changeString(s);
14
15         //打印方法改变的值和原值
16         System.out.println("changeString : " + s);
17         System.out.println("after : " + s);
18     }
19
20     public static void changeString(String str)
21     {
22         str = new String("hi");
23         str = str + "china!";
24     }
25 }

```

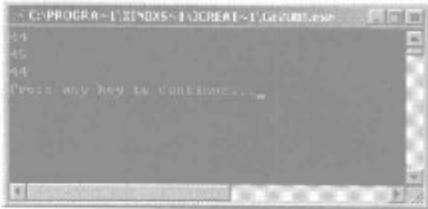


图 6-9 传递类型值

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 6-10 所示。

【代码解析】当把对象引用 s 传递到一个方法后，这个方法可以改变这个对象的属性，并能返回相应的改变。但这个对象引用指向的这个字符串 s 是永远不会改变的。这里传递对象引用后，又通过这个引用去创建了一个新的 String 类型的字符串，这两个字符串在内存中当然不是同一个了。

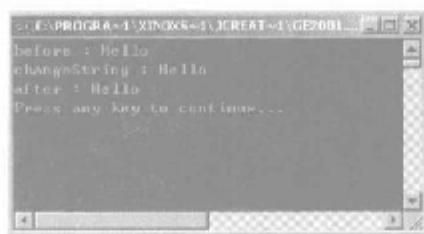


图 6-10 传递参数



6.6 对象引用的使用

对象引用就是该引用名称指向内存中的一个对象，通过调用该引用即可完成对该对象的操作。本节将要讨论一些操作对象引用中将出现的一些常见问题。例如不存在的对象、空引用、对象间的比较等问题，下面分别来说明。

6.6.1 调用不存在的对象或成员变量

如果调用的对象或成员变量没有创建，那么在编译的时候编译器将出现错误。下面用代码演示这个错误，并演示如何修正。

【范例 6-20】代码演示访问不存在的成员变量。

示例代码 6-20

```
01 //TestL 类描述的是测试访问不存在的成员变量
02 public class test
03 {
04     //main 方法为程序的入口方法
05     public static void main(String[] args)
06     {
07         //创建 TestL 类的对象实例
08         TestL t = new TestL();
09         //t.a 访问的是一个不存在的成员变量，将提示不可识别的字段
10         System.out.println(t.a);
11     }
12 }
```

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，然后使用 Java 运行编译产生的 class 程序，运行将会发生如下异常。

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
  t.a cannot be resolved or is not a field
  at TestL.main(test.java:7)
```

【代码解析】对象引用 t 要访问的是 a 这个成员变量，而 a 没有声明，在编译的时候将提示错误信息。在错误提示里，已经提示为 main 方法里的第 7 行，只需查看这里就能找到错误的所在。

【范例 6-21】修改上述代码使程序运行通过。



示例代码 6-21

```

01 //Test 类描述的是测试访问不存在的成员变量
02 public class test
03 {
04     //a 为 test 类的成员变量
05     String a;
06
07     //main 方法为程序的入口方法
08     public static void main(String[] args)
09     {
10         //创建 test 类的对象实例
11         test t = new test();
12         //t.a 访问的是一个不存在的成员变量, 将提示不可识别的字段
13         System.out.println(t.a);
14     }
15 }

```

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序, 然后使用 Java 运行编译产生的 class 程序, 运行结果如图 6-11 所示。

【代码解析】根据上例中的错误提示在 test 类声明了一个名称为 a 的成员变量。因为 String 类型的 a 没有进行赋值, 所以打印出来为 null。

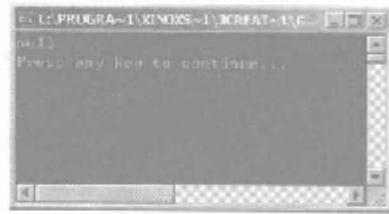


图 6-11 修改后访问

6.6.2 调用对象为 null 值的引用

任何操作的对象的值为 null 的时候都将出现空指针错误, 即“NullPointerException”错误。因为成员变量和方法是属于对象的, 即属于用 new 关键字创建出来的对象的。下面用代码来演示这个错误, 并演示如何进行修正。

【范例 6-22】访问对象值为 null 的成员变量或方法。

示例代码 6-22

```

01 //ArrayList 类所需要的
02 import java.util.ArrayList;
03
04 //test 类测试访问 null 值的对象
05 public class test
06 {
07     //声明一个成员变量 a 并进行初值
08     public String a = "test 类的成员变量";
09
10     //Java 程序的主入口方法
11     public static void main(String[] args)
12     {
13         //创建 test 类的对象实例
14         test t = new test();
15
16         //创建一个集合类, 对象引用为一个 null 值
17         ArrayList al = null;
18
19         //向一个 null 的集合对象里添加数据
20         al.add(t.a);
21     }
22 }

```



【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序, 然后使用 Java 运行编译产生的 class 程序, 运行将会发生如下异常。

```
Exception in thread "main" java.lang.NullPointerException
at test.main(test.java:20)
```

【代码解析】ArrayList 类为一个集合类和数组很相似, 都是用来存储数据用的。错误提示在 main 方法里的 20 行, 提示为 NullPointerException, 即空指针错误。对象引用 al 声明为一个 null 值, 表示这个对象并没有创建其对象的实例, 只是一个引用而已。当操作任意一个为 null 的对象的时候都将提示空指针错误。

【范例 6-23】修改上述代码使程序运行通过。

示例代码 6-23

```
01 //ArrayList 类所需要的
02 import java.util.ArrayList;
03
04 //test 类测试访问 null 值的对象
05 public class test
06 {
07     //声明一个成员变量 a 并进行初值
08     public String a = "Test 类的成员变量";
09
10     //Java 程序的主入口方法
11     public static void main(String[] args)
12     {
13         //创建 test 类的对象实例
14         test t = new test();
15
16         //创建一个集合类, 对象引用为一个 null 值
17         ArrayList al = new ArrayList();
18
19         //向一个 null 的集合对象里添加数据
20         al.add(t.a);
21
22         //打印集合 al 里的各个元素
23         System.out.println(al);
24     }
25 }
```

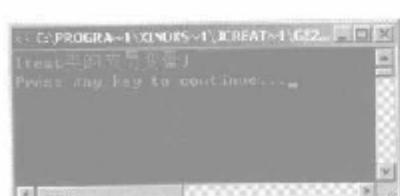


图 6-12 调用对象

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序, 然后使用 Java 运行编译产生的 class 程序, 运行结果如图 6-12 所示。

【代码解析】上述错误出现在 “al.add(t.a)”, 因为 al 为一个 null 的引用, 只需将集合引用 al 的实例创建出来就行了。当创建了集合的对象实例后, 程序将打印集合引用 al 的各个元素。

对本节的内容进行总结, 可以概括成如下几点。

- 任何操作的对象的值为 null, 都将出现空指针错误, 即 “NullPointerException”。
- NullPointerException 错误是运行期的错误, 在编译的时候系统是不进行提示的。
- 在声明一个对象引用后尽量为其赋一个初值, 来避免空指针的出现。

6.6.3 对象引用间的比较

两个对象引用进行比较，比较的是这两个对象的引用，而引用是在内存中的一个地址。地址当然是不能相同的了。下面通过一个例子来演示引用间的比较。

【范例 6-24】演示两个对象引用的比较。

示例代码 6-24

```

01 //test 类测试访问 null 值的对象
02 public class Test
03 {
04     //Java 程序的主入口方法
05     public static void main(String[] args)
06     {
07         //创建 test 类的对象实例
08         Test t1 = new Test();
09         Test t2 = new Test();
10
11         //这里的 equals 方法比较的是地址
12         if(t1.equals(t2))
13         {
14             System.out.println("t1 和 t2 相等");
15         }
16         else
17         {
18             System.out.println("t1 和 t2 不相等");
19         }
20     }
21 }
```

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 6-13 所示。

【代码解析】equals 方法在这里比较的是对象的引用，因为 equals 方法是 Object 类的方法，而任何类的父类都为 Object，equals 方法是继承过来的。继承将在后面的章节里做详细讲解。用 new 关键字创建的对象地址是重新分配的，它们进行比较，地址当然是不同的了。



图 6-13 对象引用比较



6.7 this

this 是 Java 保留的一个关键字，this 就好比日常生活中的“你我他”中的我，表示自己、本身的意思。在 Java 里也是如此，表示类的本身。下面通过代码来演示。

【范例 6-25】通过 this 来访问本类的成员变量。

示例代码 6-25

```

01 //test 类测试访问 null 值的对象
02 public class Test
03 {
04     //定义一个成员变量 color，类型为 String
```



```
05     String color;
06
07     //Test 类的构造方法
08     public test(String color)
09     {
10         //this 这里表示为创建这个对象的引用是谁, 这个 this 就是什么
11         this.color = color;
12     }
13
14     //打印并显示 color 这个成员变量的值
15     public void getColor()
16     {
17         System.out.println(color);
18     }
19
20     //Java 程序的主入口方法
21     public static void main(String[] args)
22     {
23         //创建 Test 类的对象实例, 引用为 t
24         test t = new test("黄色");
25
26         //调用引用 t 的 getColor 方法
27         t.getColor();
28     }
29 }
```

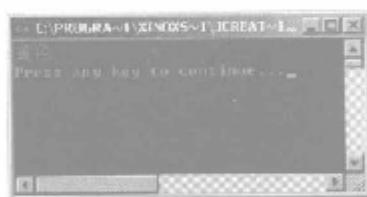


图 6-14 this 的使用

【运行结果】使用 `javac` 编译程序将产生一个和该程序对应的 `class` 程序, 然后使用 Java 运行编译产生的 `class` 程序, 运行结果如图 6-14 所示。

【代码解析】在 `test` 类的构造方法里的 `this` 关键字, 表示类的本身的意思, 即创建这个类的引用是什么, 这个 `this` 就是什么。`this` 只能在方法里使用, 不能在修饰为 `static` 的方法里使用。



6.8 要活用 JDK 已有的类

JDK 已经提供了解决问题的很多方法, 下面介绍常用的类及其方法。通过使用这些类可以让读者更了解 JDK。

6.8.1 Date 类

`Date` 类是 Java 定义的系统时间类, 里面存放的是和时间相关的方法。使用 `Date` 类, 首先要引用这个类, 即 “`import java.util.Date`”。下面通过代码来演示。

【范例 6-26】代码演示如何打印系统时间。

示例代码 6-26

```
01 //引用 Date 所需要的包
02 import java.util.Date;
03
04 //test 类测试访问 null 值的对象
05 public class test
06 {
```



```

07     public void showDate(Date d)
08     {
09         System.out.println(d);
10     }
11
12     //Java 程序的主入口方法
13     public static void main(String[] args)
14     {
15         //创建 test 类的对象实例, 引用为 t
16         test t = new test();
17
18         //创建一个 Date, 即日期类的对象实例, 引用为 d
19         Date d = new Date();
20
21         //调用 showDate 方法来显示时间
22         t.showDate(d);
23     }
24 }

```

【运行结果】 使用 javac 编译程序将产生一个和该程序对应的 class 程序, 然后使用 Java 运行编译产生的 class 程序, 运行结果如图 6-15 所示。

【代码解析】 在本程序中首先导入 util 包, 该包中有 Date 类。在该程序的 main 主方法中定义了一个 Date 类对象, 使用该对象就能够显示当前系统的时间。

接下来介绍获得时间后, 如何得到各个分项, 即“分、秒、时”等时间。创建 Date 对象后, 即可得到系统当前的时间。如果要得到时间的具体信息呢, JDK 也提供了获取例如“年、月、日”等信息的方法, 下面具体介绍它们, 如表 6-2 所示。

表 6-2 获取时间具体信息的方法列表

方法名称	使用说明
getYear()	获得当前时间的年份数-1900 的结果, 返回类型为 int 型
getMonth()	获得当前时间的月份数, 而 Java 中月份是从 0 开始的, 所以结果是从 0~11 中间的数, 返回类型为 int 型
getDate()	获得当前时间的日期数, 结果是 1~31 中间的一个数, 返回类型为 int 型
getDay()	获得当前时间的天数, 星期日是一个星期中的第一天, 它的取值范围是 0~6, 即 0 为星期日, 返回类型为 int 型
getHours()	获得当前时间的小时数, 取值范围为 0~23, 返回类型为 int 型
getMinutes()	获得当前时间的分钟数, 取值范围为 0~59, 返回类型为 int 型
getSeconds()	获得当前时间的秒钟数, 返回类型为 int 型

Date 类封装了获取当前系统日期的各种方法, 通过使用 Date 类可以在操作日期的时候带来很多方便。各个方法如表 6-3 所示。

表 6-3 Date 类各个方法

after(Date)	测试该日期是否在某指定的日期之后
before(Date)	测试该日期是否在某指定的日期之前

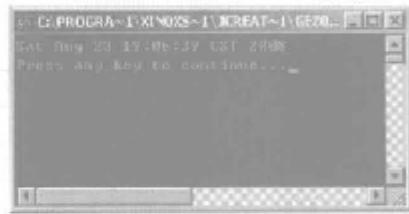


图 6-15 显示系统时间



续表

equals(Object)	比较两个日期
getDate()	返回该日期表示的一月中的日
getDay()	返回该日期表示的星期
getHours()	返回该日期表示的时
getMinutes()	返回该日期表示的分
getMonth()	返回该日期表示的月
getSeconds()	返回该日期表示的秒
getTime()	返回该日期表示的从 GMT 1970 年 1 月 1 日 00:00:00 起的毫秒数
getTimezoneOffset()	返回本地时区的偏移量
getYear()	返回该日期表示的年，并减去 1900
hashCode()	返回该对象的散列码
parse(String)	给定一个表示时间的字符串，分析它并返回时间值
setDate(int)	将一月中的日设置为指定的数值
setHours(int)	将该日期的时设置为指定的数值
setMinutes(int)	将该日期的分设置为指定的数值
setMonth(int)	将该日期的月设置为指定的数值
setSeconds(int)	将该日期的秒设置为指定的数值
setTime(long)	设置日期表示从 GMT 1970 年 1 月 1 日 00:00:00 起的毫秒数
setYear(int)	设置该日期的年为指定数值加 1900
toGMTString()	创建该日期的字符串表示
toLocaleString()	以依赖实现的形式创建该日期的字符串表示
toString()	返回该日期规范的字符串表示
UTC(int, int, int, int, int, int)	根据参数确定日期和时间

6.8.2 Integer 类

Integer 类封装了基本类型 int 的值，Integer 类型对象包含 int 型的单个域。Integer 类的方法如表 6-4 所示。

表 6-4 Integer 类各个方法

byteValue()	以 byte 类型返回该 Integer 的值
decode(String)	将字符串译码成 Integer
doubleValue()	以 double 类型返回该 Integer 的值
equals(Object)	将该对象与指定对象比较
floatValue()	以 float 类型返回该 Integer 的值
getInteger(String)	用指定名字来确定系统特性的整数值
getInteger(String, int)	用指定名字来确定系统特性的整数值
getInteger(String, Integer)	用指定名字来确定系统特性的整数值
hashCode()	返回该 Integer 的散列码
intValue()	以 int 类型返回该 Integer 的值
longValue()	以 long 类型返回该 Integer 的值
parseInt(String)	将字符串参数作为带符号十进制整数来分析



续表

parseInt(String, int)	以第二个参数所指定的基将字符串参数分析为一个带符号的整数
shortValue()	以 short 类型返回该 Integer 的值
toBinaryString(int)	创建一个整数参数的以 2 为基数的无符号整数的字符串表示
toHexString(int)	创建一个整数参数的以 16 为基数的无符号整数的字符串表示
toOctalString(int)	创建一个整数参数的以 8 为基数的无符号整数的字符串表示
toString()	返回一个表示该 Integer 值的 String 对象
toString(int)	返回表示指定整数的一个新 String 对象
toString(int, int)	创建第二个参数指定基下的第一个参数的字符串表示
valueOf(String)	返回用指定 String 值初始化的新 Integer 对象
valueOf(String, int)	返回用指定 String 值初始化的新 Integer 对象

【范例 6-27】下面通过代码演示如何进行 Integer 类型的转换。parseInt 可以将 String 类型，即 String 类型转换为 int 类型，代码如下所示。

示例代码 6-27

```

01 //test 类测试 Integer 的转换
02 public class test
03 {
04     //Java 程序的主入口方法
05     public static void main(String[] args)
06     {
07         //创建 Test 类的对象实例，引用为 t
08         Test t = new Test();
09
10         //创建一个 String 的字符串，值为 123
11         String s = "123";
12
13         //调用 parseInt 方法，将字符串转换为 int 类型
14         int n = Integer.parseInt(s);
15
16         //打印并显示其值
17         System.out.println(n);
18     }
19 }
```

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 6-16 所示。

【代码解析】parseInt 方法是将 String 类型的数据转换为 int 类型的数据。转换的前提是字符串的值为一个数值型的字符串。



图 6-16 Integer 类

6.9 综合练习

1. 成员变量和局部变量相同情况下时如何访问？

【提示】成员变量和局部变量的访问方式是不同的。



21 天学通 Java

```
public class LianXil
{
    int i=5;          //定义一个成员变量 i
    public static void main(String args[])
    {
        int i=6;      //定义一个局部变量 i
        System.out.println("局部变量的值为: "+i);
        LianXil lx=new LianXil();
        System.out.println("成员变量的值为: "+lx.i);
    }
}
```

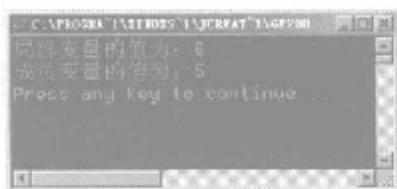


图 6-17 访问变量

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 6-17 所示。



6.10 小结

通过学习本章，可以让读者了解面向对象的基本思想、类的创建和使用、成员变量和局部变量的区别，以及对象引用的一些注意事项等问题，学好本章可以为以后的深入学习打下基础。如果读者想了解更多关于本章的相关知识，可以参考《Java 编程思想》一书进行学习。



6.11 习题

一、填空题

1. 所谓继承，是发生在_____与_____之间的，是子类共享父类成员_____和_____的一种模式。
2. 类是通过关键字_____来定义的，在该关键字后面加上类的名称，这样就创建了一个类。
3. 所谓多态是指对象在_____和_____具有两种状态，多态的使用使代码具有了更多的灵活性和重用性。
4. 类又分为_____和_____，它们之间的使用也是有区别的。
5. 成员变量描述的是这个类的_____和_____。
6. 通过 new 关键字来创建一个对象后，会有一个系统默认的初始值。boolean 类型的默认值为_____。
7. 局部变量可以被_____修饰符修饰，但不能修饰为_____、_____、_____、_____、_____。
8. _____类是 Java 定义的系统时间类，里面存放的是和时间相关的方法。
9. _____类封装了基本类型 int 的值，Integer 类型对象包含 int 型的单个域。

二、选择题

1. 选择下面程序的运行结果 ()。

01 public class Hello



```

02  {
03      public static void main(String args[])
04      {
05          new Hello(3L);
06      }
07      public Hello(long x)
08      {
09          this((int)x);
10          System.out.println("a");
11      }
12      public Hello(int x)
13      {
14          this();
15          System.out.println("b");
16      }
17      public Hello()
18      {
19          System.out.println("c");
20      }
21  }

```

- A. abc B. cba C. 编译错误

2. 选择下面程序的运行结果 ()。

```

01  public class Hello
02  {
03      public static void main(String args[])
04      {
05          new Hello();
06      }
07      public void Hello(long x)
08      {
09          System.out.println("a");
10      }
11      public void Hello(int x)
12      {
13          System.out.println("b");
14      }
15      public void Hello()
16      {
17          System.out.println("c");
18      }
19  }

```

- A. c B. cba C. 编译错误 D. 没有任何输出

三、简答题

- 概述面向对象编程的特点，以及面向对象编程和面向过程有什么不同。
- 概述成员变量和局部变量有什么不同。

四、编程题

- 编写一个输出当前时间的程序。
- 编写一个比较两个对象是否相等的程序。

第7章 控制逻辑

上一章介绍了类和对象的概念，以及成员变量、局部变量和方法的概念。本章将介绍如何通过修饰符来控制变量的访问。首先介绍包的概念，后面将介绍各个控制权限的修饰符。通过本章的学习，读者应该能够实现如下几个目标。

- 了解包的概念和如何使用包。
- 知道类的访问级别有哪些，它们有什么区别。
- 重点掌握 final 修饰符和 static 修饰符。



7.1 包 (package)

包就好比日常生活中的箱子，是一个存放东西的空间。在 Java 中，包的概念就好比 Windows 里的目录的概念，是一层一层的关系。类按照其功能分别存放在各个包里。

7.1.1 创建一个包

使用包是为了更好地将代码进行管理，首先介绍如何创建一个包，语法如下：

```
package 包名;
```

例如下面的程序语句。

```
package a.b;
```

创建包时有以下几个注意事项。

- package 为 Java 保留的关键字，不能使用别的符号进行代替。
- package 语句必须在第一行。
- package 语句只能有一个，不能有多个。
- 如果包有多层的话用句点“.”分隔。

【范例 7-1】下面的代码演示如何运行带包的程序。

示例代码 7-1

```
01 package a.b;
02
03 //test 类测试包
04 public class test
05 {
06     //Java 程序的主入口方法
07     public static void main(String[] args)
08     {
09         //创建 test 类的对象实例，引用为 t
10         test t = new test();
11
12         //打印并显示结果
13     }
14 }
```

```

13         System.out.println("程序在包中运行了");
14     }
15 }

```

【运行结果】 使用 `javac` 编译程序将产生一个和该程序对应的 `class` 程序，该程序位于当前程序的包目录下，然后使用 Java 运行编译产生的 `class` 程序，运行结果如图 7-1 所示。

【代码解析】 有包的代码在运行的时候要注意目录的问题，要找到 `class` 文件的所在目录再运行文件，如果提示“`NoClassDefFoundError`”，表示输入的包的路径是错误的。在编译的时候用命令“`java -d 文件名`”进行编译。

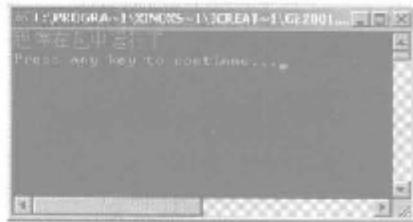


图 7-1 使用包

7.1.2 如何使用包

当创建了一个包时就要引入一个包，引入一个包的关键字为 `import`，语法如下：

```

import 包名.*;
import 包名.类名;

```

【范例 7-2】 下面通过代码来演示如何引入一个包。

示例代码 7-2

```

01 package a;
02
03 public class aaa
04 {
05     String emp = "包中的成员变量";
06
07     public void getMes()
08     {
09         System.out.println(emp);
10     }
11 }

```

如果想在其他程序中使用该类，则在其他程序中加入如下语句。

```
import a.aaa;
```

【范例 7-3】 下面通过一个完整代码来演示如何运行包里的方法。首先还是创建一个基本类。

示例代码 7-3

```

01 //创建一个包，名字为 a
02 package a;
03
04 //创建一个类
05 public class aaa
06 {
07     //声明一个 String 型的成员变量
08     String emp = "包中的成员变量";
09
10     //该方法打印其成员变量的值
11     public void getMes()
12     {
13         System.out.println(emp);
14     }
15 }

```



```
14  }
15 }
```

然后创建使用该类的类。

```
01 //引入 a 包里的 aaa 类
02 import a.aaa;
03
04 //test 类测试包
05 public class test
06 {
07     //Java 程序的主入口方法
08     public static void main(String[] args)
09     {
10         //创建 aaa 类的对象实例，引用为 a
11         aaa a = new aaa();
12
13         //调用该方法打印其成员变量的值
14         a.getMes();
15     }
16 }
```

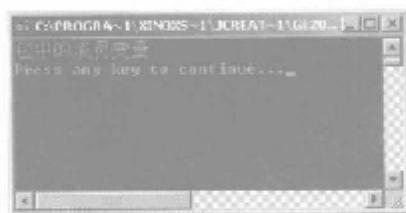


图 7-2 使用包的成员

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，该程序位于当前程序的包目录下，然后使用 Java 运行编译产生的 class 程序，运行结果如图 7-2 所示。

【代码解析】通过使用 import 来引入包 a 的 aaa 类后，就和使用类一样了，没有别的区别。注意 import 关键字的大小写。

学习了如何创建和如何使用包，下面对这两方面的知识进行一下总结。

- 一段程序可以有多 import 语句。
- 当程序有多个 import 语句时，没有其先后顺序。
- import 包名.* 表示引入该包下的所有类。
- import 包名.aaa 表示引入该包下的 aaa 类。
- import 语句要在 package 语句之后使用。

7.1.3 什么是静态引入

静态引入就是引入包中的静态成员变量和静态方法。静态引入的关键字为 static，静态的其他内容将在后面章节进行讲解。静态引入的语法如下：

```
import static 包名.aaa.*;
import static 包名.aaa.方法名称;
```

【范例 7-4】下面通过代码来演示如何静态引入。

示例代码 7-4

```
01 //静态引入 System.out.println 方法
02 import static java.lang.System.out;
03
04 //test 类测试包
05 public class test
06 {
```

```

07  //Java 程序的主入口方法
08  public static void main(String[] args)
09  {
10      //打印并显示结果
11      out.println("通过静态引入来打印数据");
12  }
13 }

```

【运行结果】 使用 `javac` 编译程序将产生一个和该程序对应的 `class` 程序, 该程序位于当前程序的包目录下, 然后使用 Java 运行编译产生的 `class` 程序, 运行结果如图 7-3 所示。

【代码解析】 带 “*” 的形式表示引入包下这个类里的所有的静态方法和静态成员变量。当只需引入具体类的静态方法时用第二种形式。不推荐使用这种方式的包引入, 因为其不利于代码的阅读, 降低程序可读性。

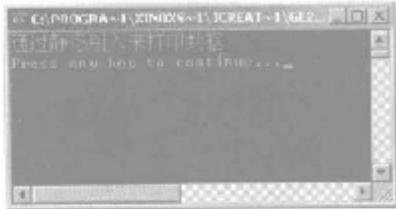


图 7-3 静态引入



7.2 类的访问级别

类的访问级别, 就好比日常生活中常见的大树, 要想到达树顶, 要从树底下慢慢地爬上去。树底下的树枝能看到旁边的树枝, 但看不到树顶的树枝。在 Java 中, 类的访问也是有一种级别关系的。下面介绍类的访问级别和成员变量的访问级别。本节所提到的修饰符请读者先行了解, 将在第 8 章对其含义做详细说明。

7.2.1 公开的访问级别

公开的访问级别在 Java 中表示为 `public`, 即在类的名称前面带有 `public` 修饰符。用 `public` 修饰符修饰该类, 表示在任何包中的任何类都能访问该类。但要注意不同包的问题。下面代码演示如何用 `public` 修饰符修饰一个类。

```

//test 类描述的是用修饰符修饰类
public class test
{
    ...//方法体
}

```

【范例 7-5】 下面通过一个完整的例子来演示 `public` 修饰一个类。

示例代码 7-5

```

01 //创建一个包 a
02 package a;
03
04 import b.*;
05
06 //test 类描述的是用修饰符修饰类
07 public class test
08 {
09     //Java 程序的主入口方法
10     public static void main(String[] args)
11     {
12         //创建 test1 的对象实例

```



```
13         test1 t1 = new test1();
14
15         //将 emp 的值赋值给 String 类型的 s
16         String s = t1.emp;
17
18         //打印并显示结果
19         System.out.println(s);
20     }
21 }
```

在该程序中使用到了 test1 类，所以还需要定义一个 test1 类。

```
01 //创建一个包
02 package b;
03
04 public class test1
05 {
06     String emp = "不同包中的成员变量";
07 }
```

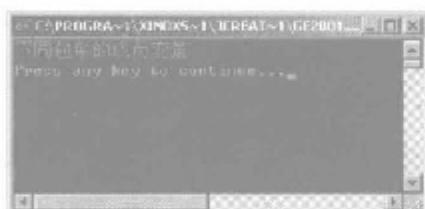


图 7-4 public 修饰符

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，该程序位于当前程序的包目录下，然后使用 Java 运行编译产生的 class 程序，运行结果如图 7-4 所示。

【代码解析】public 修饰的类，能在同一包下进行访问。当在不同包进行访问时，需要用 import 来引入该包。

7.2.2 默认的访问级别

默认的访问级别和公开的访问级别很相似，不同点就是默认的访问级别不能访问不同包下的类。只能访问同包下的类。默认的访问级别不需要在类前面加任何修饰符。下面代码演示如何定义一个默认的访问级别的类。

```
//test 类描述的是默认访问级别的类
class test
{
    ...//方法体
}
```

【范例 7-6】下面用代码来说明默认的访问级别在不同包下产生的错误。

示例代码 7-6

```
01 //创建一个包 a
02 package a;
03
04 import b.*;
05
06 //test 类描述的是用修饰符修饰类
07 public class test
08 {
09     //Java 程序的主入口方法
10     public static void main(String[] args)
11     {
12         //创建 test1 的对象实例
13     }
14 }
```

```

13         test1 t1 = new test1();
14
15         //将 emp 的值赋值给 string 类型的 s
16         String s = t1.emp;
17
18         //打印并显示结果
19         System.out.println(s);
20     }
21 }

```

同样在该程序中使用到了 test1 类，所以要定义一个 test1 类。

```

01 //创建一个包
02 package b;
03
04 class test1
05 {
06     String emp = "不同包中的成员变量";
07 }

```

【运行结果】 使用 javac 编译程序将产生一个和该程序对应的 class 程序，该程序位于当前程序的包目录下，然后使用 Java 运行编译产生的 class 程序，运行结果如图 7-5 所示。

【代码解析】 将提示找不到 test1 类，因为 test1 被修饰的是被默认的访问级别。默认的访问级别是不能访问到不同包里的类。将 test1 类修饰为 public 的就可解决上述问题。



图 7-5 默认访问级别



7.3 什么是封装

封装就好比用一个盒子把一些东西装起来，让外界能看到这些东西。在 Java 中，封装就是在一个类里定义了一些成员变量和方法，通过限制其成员变量和方法的可见性，使得外界不能访问它们。因此封装展现了接口，隐藏了细节。本节所提到的修饰符请读者先行了解，将在第 8 章对其含义进行详细的说明。

【范例 7-7】 下面通过一个例子来演示如何进行封装。

示例代码 7-7

```

01 class bike
02 {
03     //定义私有类型的 String 类型的成员变量 name
04     private String name;
05
06     //定义私有类型的 String 类型的成员变量 color
07     private String color;
08
09     //定义私有类型的 String 类型的成员变量 size
10     private String size;
11
12     //通过 get 方法来取得 name 值
13     public String getName()

```



```
14     {
15         return name;
16     }
17
18     //通过 get 方法来取得 name 值
19     public void setName(String name)
20     {
21         this.name = name;
22     }
23
24     //通过 get 方法来取得 name 值
25     public String getColor()
26     {
27         return color;
28     }
29
30     //通过 get 方法来取得 name 值
31     public void setColor(String color)
32     {
33         this.color = color;
34     }
35
36     //通过 get 方法来取得 name 值
37     public String getSize()
38     {
39         return size;
40     }
41
42     //通过 get 方法来取得 name 值
43     public void setSize(String size)
44     {
45         this.size = size;
46     }
47 }
48
49 //test 类描述的是如何进行封装
50 public class test
51 {
52     //Java 程序的主入口方法
53     public static void main(String[] args)
54     {
55         //创建 test1 的对象实例
56         bike b = new bike();
57
58         //调用 set 方法进行属性设置
59         b.setName("自行车");
60         b.setColor("黄色");
61         b.setSize("26 尺寸");
62
63         //通过 get 方法来取得其私有属性值
64         String name = b.getName();
65         String color = b.getColor();
66         String size = b.getSize();
67
68         //打印并显示各个属性的值
69         System.out.println(name + " : " + color + " : " + size);
70     }
71 }
```

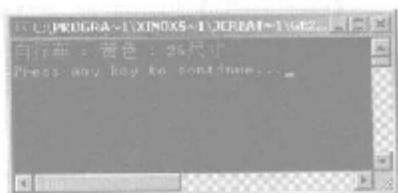


图 7-6 封装

【运行结果】 使用 javac 编译程序将产生一个和该程序对应的 class 程序，该程序位于当前程序的包目录下，然后使用 Java 运行编译产生的 class 程序，运行结果如图 7-6 所示。

【代码解析】 通过把 bike 类的各个成员变量声明为 private 即私有类型的，可以让其父类和其他类不能访问，这样具有了很好保护性。只需提供给外界 set 和 get 方法进行操作。当外界用 set 和 get 方法进行操作的时候不需要知道类的具体实现细节，只需要知道该方法返回的是什么类型的值。

总结如下：

- 虽然在定义类的时候可以将类里的成员变量声明为 public，即公共类型。但这样就不能体现类封装数据的特性了。
- 推荐在编写代码的过程中将成员变量修饰为私有类型的，即 private。声明为 private 类型后有利于代码的可读性，便于维护。



7.4 最终修饰符

所谓最终修饰符，在字面上可以说为最终的，不变的意思。修饰符 final 可以修饰很多类型的数据，被其修饰的数据所具有的含义也各有不同。下面将分别介绍修饰成员变量、局部变量、方法，以及基本类型所具有的含义。

7.4.1 final 修饰对象类型的成员变量

final 关键字修饰成员变量，其值是不能改变的。必须进行初始化。在一般情况下创建对象的时候，系统都对其成员变量进行默认初始化，被 final 关键字修饰的成员变量是不会被初始化的。

【范例 7-8】 下面用代码来演示 final 关键字修饰对象类型的成员变量没有初值的错误。

示例代码 7-8

```

01 //test 类描述的是 final 修饰的成员变量
02 public class test
03 {
04     //把 String 变量声明为 final 类型
05     final String color;
06
07     //Java 程序的主入口方法
08     public static void main(String[] args)
09     {
10         //创建 test 类的对象实例
11         test t = new test();
12
13         String s = t.color;
14
15         //打印并显示各个属性的值
16         System.out.println(s);
17     }
18 }
```



【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序, 然后使用 Java 运行编译产生的 class 程序, 运行出现如下异常。

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:  
  The blank final field color may not have been initialized  
  
  at test.<init>(<test.java:2>)  
  at test.main(<test.java:9>)
```

【代码解析】上面代码的错误提示为 color 成员变量没有在 test 对象实例前进行初始化, 也就是没有进行赋值。

【范例 7-9】修改上面的代码使其成员变量进行显式的初始化。

示例代码 7-9

```
01 //test 类描述的是 final 修饰的成员变量  
02 public class test  
03 {  
04     //把 String 变量声明为 final 类型  
05     final String color = "黄色";  
06  
07     //Java 程序的主入口方法  
08     public static void main(String[] args)  
09     {  
10         //创建 test 类的对象实例  
11         test t = new test();  
12  
13         String s = t.color;  
14  
15         //打印并显示各个属性的值  
16         System.out.println(s);  
17     }  
18 }
```

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序, 然后使用 Java 运行编译产生的 class 程序, 运行结果如图 7-7 所示。

【代码解析】在本程序中为 final 修饰的 color 变量定义了值, 从而能够正常地显示。

【范例 7-10】修改上面的代码使其成员变量在构造方法里初始化。

示例代码 7-10

```
01 //test 类描述的是 final 修饰的成员变量  
02 public class test  
03 {  
04     //把 String 变量声明为 final 类型  
05     final String color;  
06  
07     //创建 test 类的构造方法  
08     public test(String color)  
09     {  
10         this.color = color;  
11     }  
12  
13     //Java 程序的主入口方法  
14     public static void main(String[] args)  
15     {  
16         test t = new test("黄色");  
17     }
```

```

17      //打印并显示各个属性的值
18      String s = t.color;
19      System.out.println(s);
20  }
21 }

```

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序, 然后使用 Java 运行编译产生的 class 程序, 运行结果如图 7-8 所示。

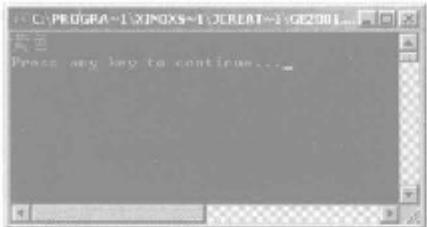


图 7-7 修改后运行结果

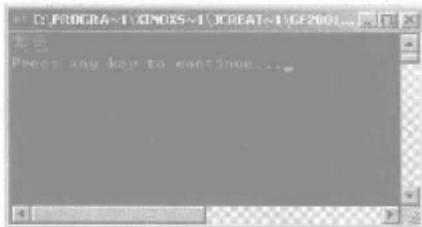


图 7-8 修改后运行结果

【代码解析】该程序中采用的是和上一个范例不同的赋值方法。使用该范例中的赋值方法同样能够正常显示 color 变量的值。

7.4.2 final 修饰基本类型的成员变量

本节内容和上一节里对象类型的成员变量很相似。当 final 修饰基本类型的成员变量的时候, 其值是不能改变的, 也就是人们常说的常量; 而对象类型的成员变量是指其引用不能改变。下面通过代码来介绍 final 修饰基本类型的成员变量有哪些特点。

【范例 7-11】下面用代码演示 final 关键字修饰基本类型的成员变量没有初值的错误。

示例代码 7-11

```

01 //test 类描述的是 final 修饰的成员变量
02 public class test
03 {
04     //把 int 变量声明为 final 类型
05     final int i;
06
07     //Java 程序的主入口方法
08     public static void main(String[] args)
09     {
10         //创建 test 类的对象实例
11         test t = new test();
12
13         int n = t.i;
14
15         //打印并显示各个属性的值
16         System.out.println(n);
17     }
18 }

```

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序, 然后使用 Java 运行编译产生的 class 程序, 运行出现如下异常。

```

Exception in thread "main" java.lang.Error: Unresolved compilation problem:
The blank final field i may not have been initialized

```



```
at test.<init>(test.java:2)
at test.main(test.java:9)
```

【代码解析】上面代码的错误提示为 i 成员变量没有在 test 对象实例前进行初始化，也就是没有进行赋值。这和对象类型错误提示相同。

【范例 7-12】修改上面的代码使其成员变量进行显式的初始化。

示例代码 7-12

```
01 //test 类描述的是 final 修饰的成员变量
02 public class test
03 {
04     //把 int 变量声明为 final 类型
05     final int i = 123;
06
07     //Java 程序的主入口方法
08     public static void main(String[] args)
09     {
10         //创建 test 类的对象实例
11         test t = new test();
12
13         int n = t.i;
14
15         //打印并显示各个属性的值
16         System.out.println(n);
17     }
18 }
```

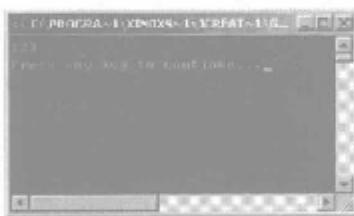


图 7-9 final 修饰基本类型

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 7-9 所示。

【代码解析】在本程序中将 int 基本类型变量声明为 final 类型，从而使得在开始定义时，必须给出初始值。在该程序中给出的初始值为“123”，所以该程序是能够正常工作的，从而显示该值。

【范例 7-13】修改上面的代码使其成员变量在构造方法里初始化。

示例代码 7-13

```
01 //test 类描述的是 final 修饰的成员变量
02 public class test
03 {
04     //把 int 变量声明为 final 类型
05     final int i;
06
07     public test(int i)
08     {
09         this.i = i;
10     }
11
12     //Java 程序的主入口方法
13     public static void main(String[] args)
14     {
15         //创建 test 类的对象实例
16         test t = new test(155);
```

```

18         int n = t.i;
19
20         //打印并显示各个属性的值
21         System.out.println(n);
22     }
23 }

```

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 7-10 所示。

【代码解析】该程序中采用的是和上一个范例不同的赋值方法。使用该范例中的赋值方法同样能够正常显示变量 i 的值。



图 7-10 构造器初始化

7.4.3 final 修饰的局部变量

final 关键字修饰的局部变量和成员变量很相似，都是使其值不能被修改。但是被修饰的局部变量一旦被赋值后就不能进行修改了。如果在创建的时候没有对其进行赋值，那么在使用前还是可以对其进行赋值的。这就是成员变量和局部变量的不同点。下面用代码来演示。

【范例 7-14】代码演示修饰局部变量可以不进行初始化赋值。

示例代码 7-14

```

01 //Test 类描述的是 final 修饰的局部变量
02 public class test
03 {
04     //定义了一个方法
05     public void getMes()
06     {
07         System.out.println("程序顺利运行");
08     }
09
10     //Java 程序的主入口方法
11     public static void main(String[] args)
12     {
13         //创建 test 类的对象实例
14         test t = new test();
15
16         //调用方法打印结果
17         t.getMes();
18     }
19 }

```

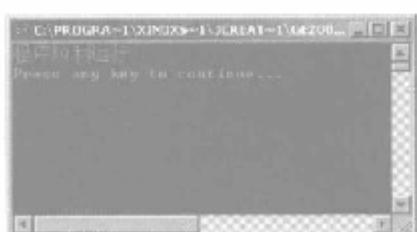


图 7-11 局部变量

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 7-11 所示。

【范例 7-15】下面用代码演示对 final 关键字修饰的局部变量进行重新赋值。



示例代码 7-15

```
01 //test 类描述的是 final 修饰的局部变量
02 public class test
03 {
04     public void getMes()
05     {
06         System.out.println("程序顺利运行");
07     }
08
09     //Java 程序的主入口方法
10     public static void main(String[] args)
11     {
12         //创建 test 类的对象实例
13         test t = new test();
14
15         //把 String 变量声明为 final 类型
16         final String s = "aaa";
17
18         s = "www";
19
20         System.out.println(s);
21         //调用方法打印结果
22         t.getMes();
23     }
24 }
```

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，然后使用 Java 运行编译产生的 class 程序，运行出现如下异常。

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
  The final local variable s cannot be assigned. It must be blank and not using
  a compound assignment
  at test.main(test.java:18)
```

【代码解析】因为 final 关键字所修饰的成员变量的值不能被改变，而局部变量也是如此。上面的错误提示就是 String 类型的变量 s 不能被重新赋值。解决这个问题只能重新声明一个 String 类型的变量或去掉 final 关键字。

7.4.4 final 修饰的方法

当用 final 关键字修饰方法时，和修饰成员变量、局部变量不太一样。被修饰的方法能被该类的子类所继承，但不能重写了。这样保护了父类某些特殊的数据。下面用代码来演示使用 final 关键字修饰方法和不使用的区别。

【范例 7-16】不使用 final 关键字的代码例子。

示例代码 7-16

```
01 //bike 类描述的是一辆自行车
02 class bike
03 {
04     //声明一个 String 类型的变量 color
05     String color = "黄色";
06
07     //声明一个方法
08     public void getMes()
```

```

09     {
10         System.out.println("父类的成员变量 color : " + color);
11     }
12 }
13
14 //test 类描述的是 final 修饰的成员变量
15 public class test extends bike
16 {
17     //声明一个 String 类型的变量 color
18     String color = "绿色";
19
20     //重写了父类的方法
21     public void getMes()
22     {
23         System.out.println("子类的成员变量 color : " + color);
24     }
25
26 //Java 程序的主入口方法
27 public static void main(String[] args)
28 {
29     //创建 test 类的对象实例
30     test t = new test();
31
32     //调用方法打印结果
33     t.getMes();
34 }
35 }

```

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 7-12 所示。

【代码解析】在该程序中并没有使用 final 修饰符，在下一个范例中将使用 final 修饰符，同时讲解使用和不使用 final 修饰符的区别。

【范例 7-17】使用 final 关键字的代码例子。

示例代码 7-17

```

01 //bike 类描述的是一辆自行车
02 class bike
03 {
04     //声明一个 String 类型的变量 color
05     String color = "黄色";
06
07     //声明一个方法
08     public final void getMes()
09     {
10         System.out.println("父类的成员变量 color : " + color);
11     }
12 }
13
14 //test 类描述的是 final 修饰的成员变量
15 public class test extends bike
16 {
17     //声明一个 String 类型的变量 color
18     String color = "绿色";

```



图 7-12 不使用 final



```
19  
20     //Java 程序的主入口方法  
21     public static void main(String[] args)  
22     {  
23         //创建 test 类的对象实例  
24         test t = new test();  
25  
26         //调用方法打印结果  
27         t.getMes();  
28     }  
29 }
```

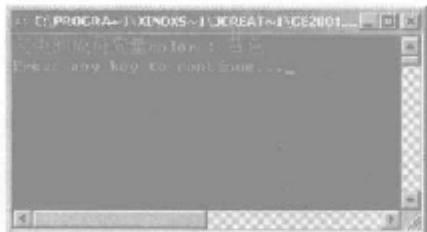


图 7-13 使用 final 修饰符

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序, 然后使用 Java 运行编译产生的 class 程序, 运行结果如图 7-13 所示。

【代码解析】在这两个例子里可以看出, 被修饰为 final 类型的方法通过继承, 该方法被继承了过来, 而不能通过继承进行方法的重写。父类的方法被修饰为 final 类型, 如果子类进行了重写, 那么将在编译的时候提示错误。原因就是被修饰为 final 类型的方法引用不能改变。



7.5 静态修饰符

静态修饰符 static 是 Java 保留的关键字, static 即静态的意思。所谓静态就是在内存中只能有一份。static 能修饰变量、方法、语句块、内部类, 下面分别对它们做介绍。

7.5.1 什么是静态变量

静态变量只能存在一份, 它属于类, 不随着对象的创建而建立副本。如果不想在创建对象的时候就需要知道一些相关信息, 那么就声明为 static 类型的, 被修饰为 static 类型的成员变量不属于对象, 它是属于类的。下面通过代码来演示这一特性。用 static 关键字修饰成员变量的语法及示例代码如下:

```
static 成员变量类型 成员变量名称  
  
static String color = "绿色";
```

【范例 7-18】通过代码来演示 static 修饰的成员变量是属于类的, 只存在一份。

示例代码 7-18

```
01 //test 类描述的是 static 修饰的成员变量  
02 public class test  
03 {  
04     //声明一个 static 类型的 String 类型的变量 color  
05     static String color = "绿色";  
06  
07     //创建 test 类的构造器  
08     public test(String color)  
09     {  
10         this.color += color;  
11     }  
12 }
```



```

13 //Java 程序的主入口方法
14 public static void main(String[] args)
15 {
16     //创建 test 类的对象实例
17     test t1 = new test("黄色");
18     test t2 = new test("红色");
19
20     //打印并显示
21     System.out.println(t1.color);
22     System.out.println(t2.color);
23     System.out.println(color);
24 }
25

```

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序, 然后使用 Java 运行编译产生的 class 程序, 运行结果如图 7-14 所示。

【代码解析】上面代码中的构造器, 只需知道每次创建对象的时候都执行这里就可以了, 构造器的知识将在以后的章节里进行介绍。通过两次执行构造器里的方法, 对 String 类型的字符 color 进行相加后, 结果证明了静态变量在所有对象里只存在一份。



图 7-14 静态变量

7.5.2 静态变量的访问

通过上一节里的说明, 让读者基本了解了什么是静态成员变量, 下面介绍如何访问静态的成员变量。主要通过如下方式。

- 类名.静态成员变量名称
- 静态成员变量名称

【范例 7-19】下面通过代码来演示在静态的方法里使用非静态成员变量出现的错误。

示例代码 7-19

```

01 //test 类描述的是 static 修饰的成员变量
02 public class test
03 {
04     //声明一个 static 类型的 String 类型的变量 color
05     String color = "绿色";
06
07     //Java 程序的主入口方法
08     public static void main(String[] args)
09     {
10         //打印并显示
11         System.out.println(color);
12     }
13 }

```

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序, 然后使用 Java 运行编译产生的 class 程序, 运行出现如下异常。

```

Exception in thread "main" java.lang.Error: Unresolved compilation problem:
  Cannot make a static reference to the non-static field color
  at test.main(test.java:11)

```



【代码解析】上面代码的错误提示说明代码的第 11 行出现错误，原因就是在 static 类型的方法里使用了 static 类型的成员变量 color。将成员变量 color 修饰为 static 类型的就可以解决上述错误。

【范例 7-20】将上面出现错误的例子进行修改，使代码运行成功。

示例代码 7-20

```
01 //test 类描述的是 static 修饰的成员变量
02 public class test
03 {
04     //声明一个 static 类型的 String 类型的变量 color
05     static String color = "绿色";
06
07     //Java 程序的主入口方法
08     public static void main(String[] args)
09     {
10         //打印并显示
11         System.out.println(color);
12     }
13 }
```



图 7-15 修改后运行结果

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 7-15 所示。

【代码解析】程序里的 main 方法为一个 static 类型的静态方法，如果要使用 color 成员变量的话，color 变量就要被修饰为 static 类型的。静态变量不依赖与对象的加载而是依赖于类的加载的。当类加载后对象可能还没有被加载，所以说静态成员变量是属于类的，其生命周期比非静态成员变量要长。在静态方法里只能使用静态成员变量。

【范例 7-21】下面代码演示非静态方法访问静态成员变量。

示例代码 7-21

```
01 //test 类描述的是 static 修饰的成员变量
02 public class test
03 {
04     //声明一个 static 类型的 String 类型的变量 color
05     static String color = "绿色";
06
07     //定义一个方法来访问静态变量
08     public void getMe()
09     {
10         System.out.println(test.color);
11     }
12
13     //Java 程序的主入口方法
14     public static void main(String[] args)
15     {
16         //创建 test 类的对象实例，引用名称 t
17         test t = new test();
18
19         //执行方法显示结果
20     }
21 }
```

```

20         t.getMes();
21     }
22 }

```

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，然后使用 Java 运行编译产生的 class 程序，运行结果如图 7-16 所示。

【代码解析】方法 getMes 里调用了静态成员变量 color，为什么在这里不提示错误呢？这是因为 getMes 是属于对象的，对象存在了该方法就存在了。在方法 getMes 存在的时候静态成员变量早就已经存在了。在任何静态方法里都可以访问静态成员变量。

【范例 7-22】代码演示在静态方法里不能使用 this 关键字。

示例代码 7-22

```

01 //test 类描述的是 static 修饰的成员变量
02 public class test
03 {
04     //声明一个 static 类型的 String 类型的变量 color
05     static String color = "绿色";
06
07     //定义一个方法来访问静态变量
08
09     //Java 程序的主入口方法
10     public static void main(String[] args)
11     {
12         //打印并显示
13         System.out.println(this.color);
14     }
15 }

```

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，然后使用 Java 运行编译产生的 class 程序，运行出现如下异常。

```

Exception in thread "main" java.lang.Error: Unresolved compilation problem:
  Cannot use this in a static context
  at test.main(test.java:13)

```

【代码解析】this 关键字指的是类的本身，是属于对象的，而静态成员变量是属于类的。它们之间的访问关系不一样。对象存在的时候静态成员变量一定存在，但 this 不知道是指向的是哪个对象；而静态成员变量存在的时候对象不一定存在。

7.5.3 什么是静态常量

通过前面章节的学习让读者了解了 static 关键字的使用及注意事项。下面介绍使用修饰符 static 的另一种形式——常量。所谓静态常量指的就是唯一的、不可变的、只存在一份的数据。在 Java 里，用 static final 两个关键字来修饰成员变量。下面用代码来演示如何声明静态常量。

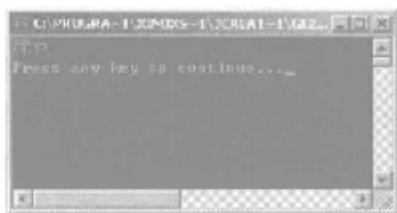


图 7-16 非静态访问静态



```
//声明两个静态常量
public static final int i = 11;
public static final float i = 11.0f;
public static final double PI = 3.14;
```

- static 关键字修饰成员变量是属于类，随着类的创建而创建。
- final 关键字修饰成员变量的值是不能改变的。
- static 关键字和 final 关键字没有前后顺序之分。

【范例 7-23】下面用一个完整的例子来演示没有给常量赋值的错误。

示例代码 7-23

```
01 //test 类描述的是 static 修饰的成员变量
02 public class Test
03 {
04     //声明一个 final static 类型的 String 类型的常量 color
05     public final static String color = "绿色";
06
07     //声明一个 final static 类型的 double 类型的变量 PI
08     public static final double PI;
09
10     //定义一个方法来访问静态变量
11
12     //Java 程序的主入口方法
13     public static void main(String[] args)
14     {
15         //打印并显示结果
16         System.out.println(color);
17
18         //打印并显示结果
19         System.out.println(PI);
20     }
21 }
```

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序，然后使用 Java 运行编译产生的 class 程序，运行出现如下异常。

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
  at test.main(test.java:13)
```

【代码解析】错误的提示说明常量 PI 必须要初始化，因为 static 关键字的关系要在对象创建前对其值进行初始化。对常量 PI 进行初始化赋值就可以解决这个问题。

【范例 7-24】用静态块的方法修改上面代码使其能正常运行。

示例代码 7-24

```
01 //test 类描述的是 static 修饰的成员变量
02 public class Test
03 {
04     //此为一个静态块
05     static
06     {
07         PI = 3.14;
08     }
09
10     //声明一个 final static 类型的 String 类型的常量 color
11     public final static String color = "绿色";
```



```

12 //声明一个final static类型的double类型的变量PI
13     public static final double PI;
14
15     //定义一个方法来访问静态变量
16
17     //Java程序的主入口方法
18     public static void main(String[] args)
19     {
20         //打印并显示结果
21         System.out.println(color);
22
23         //打印并显示结果
24         System.out.println(PI);
25     }
26 }
27

```

【运行结果】使用javac编译程序将产生一个和该程序对应的class程序，然后使用Java运行编译产生的class程序，运行结果如图7-17所示。

【代码解析】静态块和静态成员变量一样都属于类的，不属于对象的。常量和静态块随着类的创建而存在，当然可以给常量进行赋值。

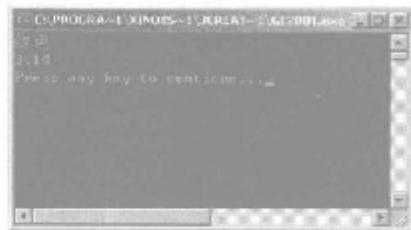


图7-17 静态常数



7.6 综合练习

1. 访问静态变量有哪两种方法？

【提示】访问静态变量可以直接访问，也可以使用类来访问。

```

01 public class Lianxil
02 {
03     static int i=5;
04     static int j=6;
05     public static void main(String args[])
06     {
07         System.out.println(i);           //直接访问
08         Lianxil lx=new Lianxil();
09         System.out.println(lx.j);       //使用对象调用
10     }
11 }

```

【运行结果】使用javac编译程序将产生一个和该程序对应的class程序，然后使用Java运行编译产生的class程序，运行结果如图7-18所示。

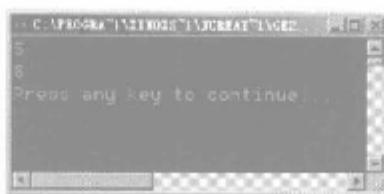


图7-18 访问静态变量



7.7 小结

通过学习本章中包、类的访问级别，以及 final 关键字、static 关键字的知识，读者了解了成员变量的访问是如何控制的。在后面的章节里将要学习继承这一重要概念，以及更多的关键字。如果读者想了解更多关于本章的相关知识，可以参考电子工业出版社出版的《Java 程序设计经典教程：融合上机操作实例》一书进行学习。



7.8 习题

一、填空题

1. 所谓静态引入就是引入包中的_____和_____，静态引入的关键字为_____。
 2. 默认的访问级别和公开的访问级别很相似，不同点就是_____。
 3. 修饰符 final 可以修饰_____等数据。
 4. static 修饰符能够修饰_____、_____、_____、_____。

一、选择题

1. 选择下面选项中的访问修饰符 ()。
A. final B. static C. private D. protected
2. 选择下面程序的运行结果 ()。

```
01  public class Hello
02  {
03      public static void main(String args[])
04      {
05          newHello nh=new newHello();
06          nh.count();
07      }
08  }
09  protected class newHello
10  {
11      void count()
12      {
13          System.out.println("aaa");
14      }
15  }
```

- A. aaa
 - B. 第5行发生编译错误
 - C. 第9行发生编译错误
 - D. 第11行发生编译错误

三、简答题

1. 什么是封装，封装有什么特点。
 2. 简述 final 修饰符的使用，以及 final 修饰符的作用。

第8章 继承

经过了前面章节的学习，读者应该对面向对象有了一定的认识。本章将开始学习面向对象的一个重要的概念——继承，并在此基础上讨论重写、重载、重写与重载之间的区别，以及多态和如何灵活运用 final、abstract 关键字等概念，学好这些概念是灵活运用多态的基础。通过本章的学习，读者应该能够实现如下几个目标。

- 了解什么是继承和继承如何使用。
- 掌握声明成员变量的修饰符。
- 熟练掌握方法的重写和重载。
- 了解枚举、反射和泛型等热门技术。



8.1 什么是继承

在日常生活中，经常遇到这样的问题。有一辆自行车，自行车有颜色和型号大小之分，而公路赛车也有颜色和型号大小之分，公路赛车多了一项速度的优势。自行车有的东西公路赛车全都有，而公路赛车有的东西自行车不一定有，它们相同地方有很多。在 Java 中，就采用继承来完成处理这种情况的功能。

【范例 8-1】通过示例代码 8-1 来理解什么是继承。

示例代码 8-1

```
01 //这是一个类，表述的是一个自行车
02 public class bike
03 {
04     public String color; //自行车的颜色
05     public int size; //自行车的大小，即型号
06 }
07
08 //这是一个类，表述的是一个公路赛类
09 public class racing_cycle
10 {
11     public String color; //自行车的颜色
12     public int size; //自行车的大小，即型号
13     public String speed; //公路赛车的速度
14 }
```

【代码解析】既然公路赛车是自行车的一种，那么通过继承可以很方便地解决这个问题，让自行车类定义自行车所应该具备的所有属性和状态，再让公路赛车继承自行车类就行了。公路赛车只需定义自己所特有的属性和状态。就好比公路赛车是自行车的一个升级版本，很自然地拥有自行车的特性。

【范例 8-2】下面就来使用继承来简化上面的程序。



示例代码 8-2

```
01 //这是一个类, 表述的是一个自行车
02 public class bike
03 {
04     public String color; //自行车的颜色
05     public int size; //自行车的大小, 即型号
06 }
07 //这是一个类, 表述的是一个公路赛车, 它继承于自行车
08 public class racing_cycle extends bike
09 {
10     public String speed; //公路赛车的速度
11 }
```

【代码解析】继承是为了让代码重复使用, 提高效率, 在此基础上衍生出更多的新类。继承是面向对象编程的特点, 没有继承就不是面向对象编程, 而是面向过程了。Java 提供了单一继承, 通过接口可以实现多重继承。本节要说明什么是继承, 继承有哪些特点。

8.1.1 类的继承

在 Java 中, 被继承的类叫超类 (superclass), 继承超类的类叫子类 (subclass)。因此, 子类是超类的一个功能上的扩展, 它继承了超类定义的所有属性和方法, 并且添加了特有功能方法。

首先举一个典型例子来说明继承有什么特点, 然后再结合代码学习。

有一对爷俩, 爸爸和儿子, 爸爸的眼睛是单眼皮, 个子很高, 头发很好, 皮肤很黑, 而儿子同样有他爸爸的一些特征, 但是儿子的皮肤很白, 双眼皮, 戴眼镜, 在外人看来他们是爷俩。儿子具有爸爸的所有特征, 但是儿子的皮肤很白和戴眼睛这些是儿子自己所特有的, 也是和爸爸不一样的地方。这个小例子正是日常生活里常见的。

换到 Java 里, 类与类之间的关系, 可以看成倒置的金字塔, 爸爸在上面, 儿子在下面。爸爸可能有多个儿子, 但是一个儿子只能有一个爸爸, 这在日常生活里也是如此。

【范例 8-3】下面是表示父类和子类的代码。

示例代码 8-3

```
01 public class Father
02 {
03     public String name; //父亲的名字
04     public int age; //父亲的年龄
05     public String eye; //父亲眼睛的样子
06     public String height; //父亲的身高
07     public String cutis; //父亲的皮肤的颜色
08 }
09
10 public class son
11 {
12     public String name; //儿子的名字
13     public int age; //儿子的年龄
14     public String eye; //儿子的眼睛的样子
15     public String height; //儿子的身高
16     public String cutis; //儿子的皮肤颜色
17     public String spectacle frame; //这个属性是儿子所特有的, 表示儿子是否戴眼镜
18 }
```



【代码解析】爸爸拥有所有的共同特征，而儿子同样拥有爸爸的所有特征，但是儿子可以比爸爸多出一些特征来。爸爸就相当于父类，儿子相当于子类，子类继承与父类。父类是一对多的关系，子类是一对一的关系，这个是Java所特有的。

【范例 8-4】把上面的代码换成父子类关系，可以如下面代码所示。

示例代码 8-4

```

01  public class Father
02  {
03      public String name;          //父亲的名字
04      public int age;             //父亲的年龄
05      public String eye;          //父亲眼睛的样子
06      public String height;       //父亲的身高
07      public String cutis;        //父亲的皮肤的颜色
08  }
09
10 public class Son extends Father //Son 类继承与 Father 类
11 {
12     public String spectacle frame; //这个属性是儿子所特有的，表示儿子是否戴眼镜
13 }

```

【范例 8-5】下面这段代码演示了两个类继承关系，以及它们之间的注意事项和解释。通过这些可以让读者有更多的了解。

示例代码 8-5

```

01  class Father
02  {
03      public String name;          //父亲的名字
04      public int age;             //父亲的年龄
05      public String eye;          //父亲眼睛的样子
06      public String height;       //父亲的身高
07      public String cutis;        //父亲的皮肤的颜色
08
09      public Father()           //Father 类的构造器
10  {
11  }
12
13 //Father 类的有两个参数的构造器
14     public Father(String name, int age)
15  {
16         this.name = name;        //对 Father 类的成员变量 name 赋值
17         this.age = age;         //对 Father 类的成员变量 age 赋值
18     }
19
20 //此方法重写了 Object 类的 toString 方法，读者可以把它去掉看有什么效果
21     public String toString()
22  {
23         String emp = name + " " + age;
24         return emp;             //把字符串 emp 的结果返回出去，并退出这个方法
25     }
26 }
27
28 public class Son extends Father //Son 类继承与 Father 类
29 {
30     public String spectacle_frame; //Son 类定义的特有的成员变量
31
32 //Son 类的构造器

```



```
33     public Son()
34     {
35     }
36
37     //Son类的有一个参数的构造器
38     public Son(String spectacle_frame)
39     {
40         this.spectacle_frame = spectacle_frame;
41     }
42
43     public void getMes()
44     {
45         System.out.println(spectacle_frame);
46     }
47
48     //Java程序的主入口方法，程序运行先从这里运行
49     public static void main(String args[])
50     {
51         Father f = new Father("father", 40); //创建了一个对象，对象引用是f,
52         //并应用的是两个参数的构造器
53         System.out.println(f); //打印对象引用f
54
55         Son s = new Son("blue"); //创建一个对象，引用为s
56         s.name="son"; //给s对象引用里的成员变量name赋值
57         s.age=15;
58         System.out.println(s); //打印到屏幕上，并输出
59         s.getMes(); //执行getMes方法
60     }
}
```

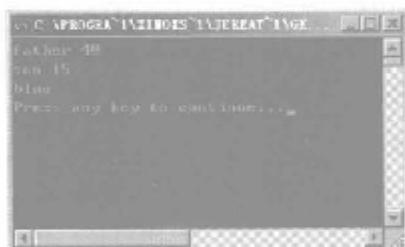


图 8-1 父子关系

【运行结果】使用 javac 编译程序将产生一个和该程序对应的 class 程序, 然后使用 Java 运行编译产生的 class 程序, 运行结果如图 8-1 所示。

【代码解析】Father 为父类, Son 为子类, 子类拥有父类所有的公共方法和属性。子类比父类多出自己所特有的属性和方法。如果父类现在所有的方法和属性不能满足功能需要, 那么只需要新建一个父类的子类, 在子类里添加所需要的方法和属性来满足功能上的需要。

一般在父类定义一些公共的属性和方法, 而子类定义这个子类所要完成的特定功能的相应方法和属性, 上面的小例子中已经演示了这些。

```
Son s = new Son("blue");
```

代码说明:

- 调用时子类里有一个参数的构造器, 默认的是调用父类的无参构造器。
- 父类里如果没有无参构造器, 将不能编译通过, 但子类里有没有无参的构造器并不重要。

```
01 public Son()
02 {
03     super(); //这里调用的是Father的无参构造器, 不写也行
04 }
05
```

```

06 public Son(String name, int age)
07 {
08     super(name, age); //调用的是父类的有两个参数的构造器,父类必须要有两个参数的构造器
09 }

```

本节学习了继承的使用,下面对其内容做如下总结。

- 通过继承定义类,可以简化类的定义,让所需要的功能用相应的子类去定义和实现。
- Java 是单继承的,子类可以有很多,父类只能有一个。上面的例子,如果加一个 Friend 类, Son 只能继承自 Father, 要么继承 Friend, 不能同时继承 Father 和 Friend。
- Java 的继承是多层继承的,是一个类可以有很多子类,而子类下面又可以有很多子类。
- 父类里的属性和方法可以让子类所有,父类里的属性和方法可以使子类同样拥有,而子类的不能调用父类的方法和属性,子类的无参构造器默认是调用的父类的无参构造器。
- 父类没有定义一个无参的构造器,那么编译器就默认生成一个无参的构造器,也可以在子类构造器里显示使用 super 方法调用父类构造器,super 方法里写几个参数就可以表示调用的是父类的哪一个构造器。
- 一般情况下,定义了一个有参的构造器,就应该定义一个无参的构造器。

8.1.2 继承的语法

类的继承是通过 Java 保留的关键字 extends 来修饰的,通过 extends 关键字表明前者具备后者的公共的成员变量和方法,在具备了所有的公共的成员变量和方法后,本身还能定义一些特有的成员变量和方法。基本语法如下:

```
class 类名 extends 父类名称
```

【范例 8-6】下面是使用继承的程序。

示例代码 8-6

```

01 public class Father
02 {
03     public String name; //父亲的名字
04     public int age; //父亲的年龄
05     public String eye; //父亲眼睛的样子
06     public String height; //父亲的身高
07     public String cutis; //父亲的皮肤的颜色
08 }
09
10 public class Son extends Father //Son 类继承与 Father 类
11 {
12     public String spectacle frame; //这个属性是儿子所特有的,表示儿子是否戴眼镜
13 }

```

【代码解析】extends 为 Java 保留的关键字,所有类的父类为 Object。如果 Son 子类没有继承关系,那么 Son 的父类为 Object, Object 类的包路径为 java.lang.Object。

【范例 8-7】下面代码演示了类的多层继承,类的继承是一个单项关系,也就是说是从头到尾的继承,而兄弟类是不能继承的。



示例代码 8-7

```
01 //这是一个类, 描述的是一个水果
02 class fruit ()
03 {
04 }
05
06 //这是一个类, 描述的是一个苹果, 继承自水果
07 class apple () extends fruit
08 {
09 }
10
11 //这是一个类, 描述的是一个黄苹果, 继承自苹果
12 class yellow_apple() extends apple
13 {
14 }
```

【代码解析】fruit 为父类, apple 和 yellow_apple 都为子类, apple 继承与 fruit, apple 能使用父类 fruit 的所有属性和方法, 而 yellow_apple 继承 apple, 拥有 apple 和 fruit 的属性和方法。上面一段代码演示了类的多层继承, 如果把代码改为 yellow_apple 继承与 fruit 的话, apple 和 yellow_apple 都继承与 fruit, apple 和 yellow_apple 共享 fruit 的所有属性和方法, apple 和 yellow_apple 自己的属性和方法是不能互相访问的, 此时 apple 和 yellow_apple 是兄弟类的关系。而 fruit 就继承于 Object 类。



8.2 修饰符

修饰符是修饰的当前成员变量的访问限制和状态的。就好比一个眼镜, 颜色是黑色的, 这个黑色就修饰了这个眼镜, 而眼镜的种类很多, 可以让不同的人群使用, 如近视眼镜就由有近视眼的人群使用, 别人使用的话就不好了。示例如下:

```
public String name; //public 就是一个修饰符
```

成员变量的继承是指 B 继承于 A 后, B 能使用 A 的属性和方法, 是受成员变量的修饰符决定的。在上一节的例子里的成员变量都是使用的默认修饰符, 本节将详细介绍修饰符是如何限制成员变量的继承的。主要有 4 个修饰符, 分别为 public、private、default、protected, 下面分节对其详细介绍。

8.2.1 public: 声明成员变量为公共类型

public 表明被它修饰的成员变量为公共类型, 那么这个成员变量在任何包里都能访问, 包括子类也能访问到, 下面用代码来说明。

【范例 8-8】下面是使用 public 修饰符的程序。

示例代码 8-8

```
01 //此类描述的是一个门
02 class door
03 {
04     public String color; //为定义为 public 类型, 描述门的颜色
05
06     //门的构造器
07     public door ()
08     {
```



```
09         this. color = "门的颜色为黄色";
10     }
11
12     public void openDoor()
13     {
14         System.out.println("此方法为 public 方法,为开门的动作");
15     }
16 }
17
18 public class wood_Door extends door
19 {
20     public String wood_color;
21
22     public wood_Door ()
23     {
24         this. wood_color = "是本类里的成员变量";
25     }
26
27     public static void main(String[] args)
28     {
29         wood_Door wd = new wood_Door ();
30         //访问的是父类的成员变量和方法
31         System.out.println(wd. color);
32         wd. openDoor ();
33         //访问的是本类的成员变量
34         System.out.println(wd. wood_color);
35     }
36 }
```

【运行结果】使用 `javac` 编译程序将产生一个和该程序对应的 `class` 程序，然后使用 Java 运行编译产生的 `class` 程序，运行结果如图 8-2 所示。

【代码解析】wood_Door 继承与 door, color 成员变量和 openDoor 方法声明为 public 类型, 那么 door 的子类 wood_Door 能够访问到 door 的成员变量 color 和方法 openDoor。wood_color 是 wood_Door 的本类成员变量, 也能访问到, 如果 wood_Door 有子类的话。

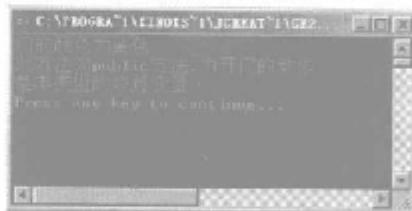


图 8-2 public 修饰符

8.2.2 private: 声明成员变量为私有类型

`private` 表明被它修饰的成员变量为私有类型，表示除了本类外任何类都不能访问到这个成员变量，具有很好的保护性。下面用代码来说明。

【范例 8-9】下面是使用 private 修饰符的程序。

示例代码 8-9

```
01 class door
02 {
03     private String color; //为定义为public类型
04
05     //door类的构造器
06     public door ()
07     {
08         this.color = "父类里的private类型成员变量";
09     }
10 }
```