模块 1

认识第一个 Java 应用程序

Java 语言是目前国内外使用最为广泛的程序设计语言之一，它具有功能丰富、表达能力强、使用方便灵活、执行效率高、跨平台、可移植性好等优点，几乎可应用于所有领域。使用 Java 语言进行程序设计和软件开发，可熟悉并理解面向对象思想的精髓。熟练掌握 Java 语言，便能更加深入地掌握计算机技术。

Java 程序开发的前提是安装 JDK 并配置环境变量，搭建开发环境。Java 程序代码编写的规范与 C 语言的相比有很大区别，如严格区分大小写等。本模块首先介绍 Java 开发环境的搭建，然后通过几个简单的实例程序介绍 Java 程序的编写、编译和运行方法。

学习目标

- 了解 Java 程序编写的基本规范。
- 掌握 Java 程序开发环境的搭建。
- 掌握 Java 程序的开发流程。
- 掌握 Java 程序的编译和运行方法。
- 掌握数据输入/输出的基本方法。

1.1 案例引入——第一个 Java 应用程序

案例说明

Java 程序可分为应用程序(Application)和小应用程序(Applet)两大类。根据介绍程序语言的传统,首先来编写第一个 Java 应用程序,即 Hello World 程序。

本案例程序编辑、编译、运行后将会显示“Hello World”文本信息。Hello World 程序将是接触 Java 的第一个应用程序,通过这个友好的应用程序,可以了解 Java 语言的特点,以及如何编辑、编译和运行一个 Java 程序。

案例分析

为完成本案例程序,首先要在 Windows 系统上(本书所有案例及实训练习均在 Windows 操作系统上实现)搭建 Java 开发环境,然后使用文本编辑器编辑源程序,而后完成编译和运行。

1.2 知 识 准 备

1.2.1 Java 语言简介

Java 是一种跨平台的面向对象的程序设计语言,由 Sun Microsystems 公司的 James Gosling 于 20 世纪 90 年代初开发出来的。它最初被命名为 Oak,目标设定在家用电器等小型系统的编程语言,以解决诸如电视机、电话、闹钟、烤面包机等家用电器的控制和通信问题。由于这些智能化家电的市场需求没有预期的高,因此 Sun 公司放弃了该项计划。就在 Oak 几近失败之时,随着互联网的发展,Sun 公司看到了 Oak 在计算机网络上的广阔应用前景,于是改造了 Oak,于 1995 年 5 月以 Java 的名称正式发布。

Java 编程语言是一种简单、面向对象、分布式、解释性、健壮、安全与系统无关、可移植、高性能、多线程和动态的语言。Java 技术具有卓越的通用性、高效性、平台移植性和安全性,广泛应用于 PC、数据中心、游戏控制台、科学超级计算机、移动电话和互联网,同时拥有全球最大的开发者专业社群。在全球云计算和移动互联网的产业环境下,Java 更具备了显著优势和广阔前景。

1999 年 6 月,Sun 公司发布 Java 的 3 个版本:标准版(J2SE)、企业版(J2EE)和微型版(J2ME)。2005 年 6 月,JavaOne 大会召开,Sun 公司公开 Java SE 6。此时,Java 的各种版本已经更名,以取消其中的数字“2”:J2EE 更名为 Java EE,J2SE 更名为 Java SE,J2ME 更名为 Java ME。2009 年 4 月 20 日,Sun 公司被 Oracle 公司(甲骨文公司)收购。本书所有内容基于 Java SE 版本。

1.2.2　Java 开发环境的搭建

了解 Java 的开发环境是使用 Java 语言编程的第一步，也是学好 Java 语言的基础。目前已经有很多 Java 开发环境，但对于初学者而言，Sun 公司提供的免费的 Java 开发和运行环境——JDK 仍然是最简单易用的。JDK（Java development kit，Java 开发工具包）是整个 Java 的核心，包括了 Java 运行环境、Java 工具和 Java 基础的类库。掌握 JDK 的安装是学好 Java 的第一步。

1. 下载 JDK

Oracle 公司收购 Sun 公司以后，仍然不断推出新的 JDK 版本，目前最新的版本为 JDK 7。可以直接访问下面的网址下载 JDK：

http://www.oracle.com/technetwork/java/javase/downloads/index.html

在下载页面中，需要根据自己的操作系统选择合适的 JDK，如图 1-1 所示。以最常见的 Windows 操作系统为例，本书写作时下载的 JDK 的安装文件名为 jdk-7u2-windows-i586.exe，以后均以此版本为例进行讲解。需要注意的是，JDK 的版本会不断更新。

Java SE Development Kit 7u2

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

○ Accept License Agreement　◉ Decline License Agreement

Product / File Description	File Size	Download
Linux x86	63.62 MB	jdk-7u2-linux-i586.rpm
Linux x86	78.62 MB	jdk-7u2-linux-i586.tar.gz
Linux x64	64.51 MB	jdk-7u2-linux-x64.rpm
Linux x64	77.46 MB	jdk-7u2-linux-x64.tar.gz
Solaris x86	135.87 MB	jdk-7u2-solaris-i586.tar.Z
Solaris x86	81.37 MB	jdk-7u2-solaris-i586.tar.gz
Solaris SPARC	138.94 MB	jdk-7u2-solaris-sparc.tar.Z
Solaris SPARC	86.05 MB	jdk-7u2-solaris-sparc.tar.gz
Solaris SPARC 64-bit	16.13 MB	jdk-7u2-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	12.31 MB	jdk-7u2-solaris-sparcv9.tar.gz
Solaris x64	14.45 MB	jdk-7u2-solaris-x64.tar.Z
Solaris x64	9.25 MB	jdk-7u2-solaris-x64.tar.gz
Windows x86	84.04 MB	jdk-7u2-windows-i586.exe
Windows x64	87.35 MB	jdk-7u2-windows-x64.exe

图 1-1　下载 JDK

2. 安装 JDK

下载完成后，就可以安装 JDK 了。安装 JDK 的操作步骤如下。

（1）双击下载的 JDK 安装文件 jdk-7u2-windows-i586.exe，即可进入 JDK 安装向导界面，如图 1-2 所示。

（2）单击“下一步”按钮进入 JDK 的安装选择界面，如图 1-3 所示。

图 1-2　JDK 安装向导界面

图 1-3　JDK 安装选择界面

(3)在图 1-3 中,单击“更改”按钮,可以选择安装目录,在此保持默认安装目录。在自定义安装程序的功能时,建议选择全部功能。

(4)单击“下一步”按钮进行安装,在安装过程中,会弹出 JRE(Java 运行环境)的安装提示,如图 1-4 所示,单击“下一步”按钮即可。

(5)随即弹出如图 1-5 所示的界面,单击“完成”按钮,即可完成 JDK 的安装。

3. 设置环境变量

JDK 安装成功后,还需要对操作系统的环境变量进行设置。

(1)在 Windows 操作系统桌面上右击“我的电脑”图标,在弹出的快捷菜单中选择“属性”命令,弹出“系统属性”对话框,切换到“高级”选项卡,如图 1-6 所示。

(2)单击“环境变量”按钮,打开“环境变量”对话框,如图 1-7 所示。

图 1-4　JRE 安装提示

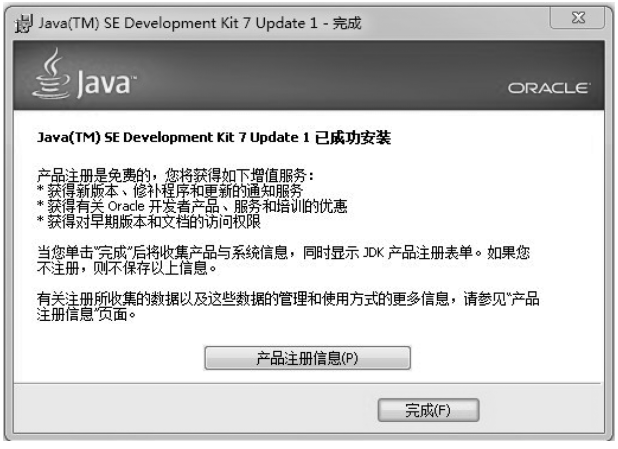

图 1-5　JDK 安装成功

图 1-6　“系统属性”对话框

图 1-7　“环境变量”对话框

(3)在“环境变量”对话框中单击“系统变量”选项组下方的“新建”按钮,在弹出的“新建系统变量”对话框中输入变量名“JAVA_HOME”,变量值为“C:\Program Files\Java\jdk1.7.0_02”,即 JDK 的安装目录,如图 1-8 所示,然后单击“确定”按钮即可。

图 1-8 “新建系统变量”对话框

(4)按照同样的方法,新建系统环境变量“CLASSPATH”,变量值为“.;%JAVA_HOME%\lib”,并单击“确定”按钮完成设置,如图 1-9 所示。

图 1-9 新建 CLASSPATH 系统变量

(5)在“系统变量”选项组中选择 Path 选项,并单击其下方的“编辑”按钮,弹出“编辑系统变量”对话框,在当前变量值的基础上,增加“%Java_home%\bin”,并用分号隔开,如图 1-10 所示。

图 1-10 编辑 Path 系统变量

(6)单击“确定”按钮,完成环境变量的设置。

1.2.3 Java Application 开发

1. Java Application 开发流程

Java Application 开发的基本步骤是:首先编写源程序,并保存为.java 文件;然后编译源程序得到字节码文件,即.class 文件;最后通过 Java 解释器解释执行字节码文件。整个开发流程如图 1-11所示。

编译器 javac.exe 和解释器 java.exe 都安装在 JDK 安装目录下的 bin 文件夹中。bin 文件夹中还有其他一些 Java 的开发工具。

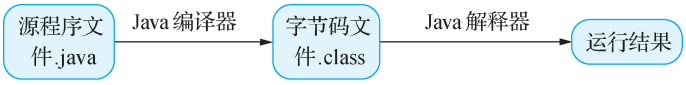

图 1-11　Java 程序开发流程

跨平台性可以说是 Java 语言的最优秀特性之一，这个特性经常被称为“一次编译，到处执行”。Java 源程序经过编译后得到的是一个特殊的文件——字节码文件，凡是安装了 Java 运行环境(JRE)的平台，都能执行这个字节码文件。JRE 主要由 Java 虚拟机(JVM)和一些标准类库组成。

2. Java 程序编写的基本规范

对于初学者而言，编写 Java 程序的一些基本规范必须要牢记。

(1)Java 语言严格区分字母的大小写，如 Hello 和 hello 是不同的。

(2)一条 Java 语句必须以分号结束。

(3)大括号“{ }”用于构成一个语句块，总是成对出现的。

3. Java 程序的注释

注释是程序中的说明文字，便于程序的阅读。注释不是 Java 的语句，不会影响程序的功能和运行，一个 Java 程序中可以有多条注释。Java 程序的注释有以下 3 种形式。

(1)单行注释。以“//”开始到本行结束的内容都是注释。例如：

```
//这是单行注释
```

(2)块注释。在“/*”和“*/”之间的所有内容都是注释。例如：

```
/*这是块注释
它可以包含很多行内容 */
```

(3)文档注释。在“/**”和“*/”之间的内容都是文档注释。可以通过 JDK 提供的工具 javadoc 提取程序的文档注释，生成程序的 HTML 文档。

1.3 案 例 实 施

1.3.1 编写源代码

【程序 1-1】 编写 HelloWorld.java 文件。

打开记事本(或其他文本编辑器)，输入以下 Java 源代码。

```
public class HelloWorld{                          //定义一个类，类名为 HelloWorld
    public static void main(String[] args){       //main()是 Java 应用程序的主方法
        System.out.println("Hello World");         //在命令行下输出 Hello World
    }
}
```

一个 Java Application 程序由若干个类组成，【程序 1-1】只有一个类，类名为 HelloWorld。最外层的大括号及括号之间的内容称为类体。main()是类 HelloWorld 的一

个方法，而且是主方法。一个 Application 程序有且只有一个类包含 main()方法，这个类就是程序的主类。【程序 1-1】中类 HelloWorld 就是主类。

Application 程序的执行是从 main()方法开始的，main()方法的格式是固定不变的：

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

Java 源程序文件的命名有着严格的限制，文件的扩展名为“.java”。源文件中只能有一个类用 public 修饰，源程序文件的名字必须和这个 public 类的名字一致。因此，【程序 1-1】编辑完成后应该保存为“HelloWord.java”。

1.3.2 编译源代码

Java Application 程序需要先将源程序文件编译成字节码文件，才能被 Java 解释器解释运行。执行“开始”→“运行”命令，在弹出的“运行”对话框中输入“cmd”，单击“确定”按钮，打开命令提示符窗口。将命令提示符的当前路径切换到 Java 源程序文件所在的目录，如 D:\project，然后输入以下命令完成对 HelloWorld.java 的编译：

```
javac HelloWorld.java
```

可执行文件 javac.exe 是 Java 的编译工具，用于对 Java 源文件进行编译。若源代码没有错误，编译成功后在 D:\project 目录下会生成一个字节码文件 HelloWorld.class。

1.3.3 运行程序

使用 JDK 的解释器 java.exe 就可以对编译后得到的字节码文件进行解释执行了。在命令提示符后输入下面的命令，并按 Enter 键：

```
java HelloWorld
```

程序运行结果如图 1-12 所示。

图 1-12 Hello Word 程序运行结果

1.4 训练与实战

下面通过两个具体的实战训练项目来进一步巩固 Java 应用程序开发的步骤，熟悉 Java 程序的基本规范。

1.4.1　带命令行输入参数的 Java 程序

命令行参数就是程序执行时在命令行中紧跟在程序名后的信息，这个参数的本质是一个字符串数组。Java 应用程序执行时的命令行参数传递给了 main()方法，由 main(String[] args)方法中的形式参数 args 接收。args 是一个字符串数组，和其他程序设计语言一样，对数组元素的引用是通过下标来实现的，下标从 0 开始。关于数组在后面的模块中有详细介绍。

训练内容

编写一个简单的应用程序，实现将两个字符串参数交换次序后输出。

训练过程

1)编写源代码

【程序 1-2】　编写 Exchange.java 文件。

打开记事本，输入以下代码，将程序以 Exchange.java 为名称保存到 D:\project 下。

```
public class Exchange
{
    public static void main(String[] args)
    {
        System.out.println(args[1]+"!"+args[0]);  /*将命令行的两个参数输
                                                    出时，中间用"!"隔开 */
    }
}
```

2)编译和运行程序

在命令提示符窗口下将路径切换到 D:\project，然后输入：

```
javac Exchange.java
```

编译完成后，使用下面的命令传递参数：

```
java Exchange first second
```

first 为第一个参数，将传递给 args[0]；second 为第二个参数，将传递给 args[1]。两个参数之间用空格隔开。程序运行结果如图 1-13 所示，先输出第二个参数，再输出第一个参数。

总结与体会

通过命令行参数传递参数给 main()方法是最简单的参数输入形式。在没有系统地讲解Java的输入/输出方法之前，这将是一种非常有效的用户和程序进行交互的方式。需要注意的是，这样的参数输入只能是字符串数据，如果需要传递其他类型的数据，则需要进行类型转换。

图 1-13　交换参数次序

1.4.2　简单的输入/输出处理

训练内容

带命令行参数的运行方式能够将信息传递给程序，但这样的交互只能在运行程序时才可以。更普遍的需求是，在程序运行过程中与用户进行交互。下面使用 Java 的标准输入/输出对象实现对输入的字母进行大小写转换，而后输出。

训练过程

1）编写源代码

【程序 1-3】　编写 Trans.java 文件。

打开记事本，输入以下代码，将程序以 Trans.java 为名称保存到 D:\project 下。

```
import java.io.IOException;
public class Trans
{
    public static void main(String[] args) throws IOException
    {
        int a,b;
        System.out.println("please input a letter:");
        a = System.in.read();
        if(a>=65&&a<=90)
          b = a + 32;
        else
          b = a - 32;
        System.out.println((char)a + "has been transformed into:" + (char)b);
    }
}
```

程序首行 import java.io.IOException 用于导入程序将用到的类 IOException。程序中

的 System. in 和 System. out 分别是标准输入对象和标准输出对象。throws IOException 子句表明该方法可能会抛出一个输入/输出异常对象。

使用 Java 的输入/输出对象就必须加入对输入/输出异常的处理，有关异常的处理和输入/输出将在以后的模块中详细介绍。

2)编译和运行程序

在命令提示符窗口中将路径切换到 D:\project 下，然后输入下面的命令：

```
javac Trans. java
```

编译完成后输入下面的命令运行程序：

```
java Trans
```

根据提示信息输入大写字母“A”，然后按 Enter 键，运行结果如图 1-14 所示。

图 1-14　转换字母的大小写

总结与体会

本节介绍了使用系统提供的标准输入/输出对象进行操作的简单程序，通过这种方式可以实现更好的人机交互。关于标准的输入/输出对象的使用，后面还有更多的训练内容。

1.5 实际工作中常见问题解析

1. 问题

虽然安装了 JDK，却不能使用 javac 和 java 命令来运行程序。

2. 问题分析与解决

出现此类问题的主要原因是没有配置系统环境变量或系统环境变量配置不正确。因此，需要正确配置 JAVA_HOME、CLASSPATH、Path 这 3 个环境变量。

(1)JAVA_HOME 变量是为了确保系统能正确地找到 JDK 的位置。

(2)CLASSPATH 变量是为了让 Java 编译器能正确地找到程序所需要的包或类库。

(3)Path 变量是为了确保系统能找到 JDK 的开发工具 javac 和 java。

1.6 习　　题

一、思考题

1. 简述 Java Application 程序的开发流程。
2. 查阅资料，比较 Java 语言和 C 语言的异同。

二、编程题

1. 编写一个 Java 程序，在屏幕上输出“这是我的第一个 Java 程序！”。
2. 编写一个 Java 程序，通过键盘输入两个整数，求这两个数的和并输出结果。

模块2

Java 基本语法

在程序设计语言中都有一套基本的语法规则，如标识符和关键字、运算符和表达式、程序流程控制语句等。对于学过 C 语言或者其他编程语言的读者来说，Java 语言的语法规则并不复杂，而且和其他高级程序设计语言的语法有着很多相似之处。本模块将通过几个具体的实例来介绍 Java 语言的基本语法规则，尤其是 Java 语言特有的语法规则。

学习目标

- 掌握 Java 标识符的命名规则。
- 掌握 Java 的基本数据类型。
- 了解 Java 的程序流程控制语句。
- 理解数学函数的作用。

2.1 案例引入——数据类型转换

案例说明

与其他程序设计语言一样，Java 语言的语法中也包含一套基本的符号，这些基本符号构成了 Java 语言的标识符和关键字。同时，Java 语言是一种强类型语言，即每个变量或者表达式都必须有确定的数据类型，在对变量赋值时要进行数据类型的兼容性检查。本案例是一个简单的对不同数据类型的变量进行赋值的程序。

案例分析

为完成本案例程序，首先要熟悉 Java 语言的标识符的定义、变量的声明以及变量的赋值，掌握 Java 基本数据类型，然后使用文本编辑器编辑源程序，而后完成编译和运行。

2.2 知 识 准 备

2.2.1 标识符和关键字

1. 字符集

Java 字符集采用的是通用的国际标准字符集 Unicode(万国码)，而并非计算机系统常用的 ASCII 码字符集。Unicode 字符集使用 2 字节即 16 位来表示一个字符，共有 65 536 个字符。Unicode字符集能表示迄今为止人类的所有语言文字，包括以下几类字符：

(1)ASCII 码字符集里的英文字母 A～Z、a～z 以及数字 0～9；

(2)多国文字字符，包括汉字、日文、朝鲜文、希腊文等；

(3)常见的特殊符号字符集，如“&”、“¥”、“@”等。

2. 标识符

标识符用于命名程序对象，如 Java 中的变量名、类名、方法名等。Java 语言中的标识符命名规则如下。

(1)标识符由文字字符(包括字母、中文等)、数字、下划线(_)、美元符号($)组成，并且首字母不能是数字，不能包含任何嵌入的空格或点“.”以及除下划线“_”、“$”字符之外的特殊字符。

(2)不能把 Java 关键字作为标识符。

(3)标识符的长度没有限制。

(4)标识符严格区分大小写，如 My 和 my 是不同的标识。

例如，“student”、“s1”、“_A101”、“性别”都是合法的标识符，即符合 Java 标识符的命名规则；而“2stu”是非法的标识符，即不符合 Java 标识符的命名规则。

3. 关键字

和其他高级语言一样，Java 语言也保留一部分关键字。这些作为关键字的标识符具有特殊的意义和用途，不能作为一般的标识符使用。Java 关键字详细信息参见附录 I。关键字又称保留字，主要用于描述程序结构、声明 Java 类、定义类的成员、基本数据类型名、异常处理等。例如，关键字“class”用来声明一个 Java 类，“int”用来定义一个整型类型的数据或变量。

2.2.2　数据类型

在计算机语言里，数据类型是对内存位置的一个抽象的表达方式，可以理解为针对内存的一种抽象表达方式。Java 是强类型语言，所以对数据类型的规范更严格。在 Java 中，数据类型可分为简单数据类型和复合数据类型两大类，如表 2-1 所示。

表 2-1　Java 数据类型

简单数据类型								复合数据类型		
整数类型				浮点类型		字符类型	布尔类型	类	接口	数组
byte	short	int	long	float	double	char	boolean	class	interface	

简单数据类型是系统预先定义的一些常用类型，包括整数类型、浮点类型、字符类型和布尔类型。用户只能使用这些简单数据类型，而不能修改。

复合数据类型是由简单数据类型组合而成的新类型，包括数组、类和接口。

1. 整数类型

Java 语言把整数类型的数据按照数值范围的大小划分为 byte、short、int、long 4 种类型。表 2-2 列出了各种类型在内存中所占的位数及其数值范围。

表 2-2　整数类型

类 型 名	所占位数	数值范围
byte	8	$-2^7 \sim (2^7-1)$
short	16	$-2^{15} \sim (2^{15}-1)$
int	32	$-2^{31} \sim (2^{31}-1)$
long	64	$-2^{63} \sim (2^{63}-1)$

整型常量有十进制、八进制和十六进制 3 种表示方法，如下所示。

(1)十进制整数，如 35、−64、0 等。

(2)八进制整数以数字“0”开头，如 035 表示十进制数 29，−064 则表示十进制数−52。

(3)十六进制整数以“0x”或“0X”开头，如 0x35 表示十进制数 53，−0x11 表示十进制数−17。

对于 long 型(长整型)常量，则要在整数的后面加上“L”或“l”，如 35L 表示一个 long 型长整数，这个数在内存中存储时需要占用 64 位的存储空间。

注意：Java 语言不提供任何类型的无符号整数。

2. 浮点类型

浮点类型又称实数类型、实型，用于表示带小数点的数据，有单精度和双精度两类，即 float 和 double。表 2-3 列出了这两种类型在内存中所占的位数及其数值范围。

表 2-3　浮点类型

类 型 名	所占位数	数值范围
float	32	3.4e－38～3.4e＋38
double	64	1.7e－308～1.7e＋308

浮点型常量可以用十进制数形式来表示，即由数字和小数点组成，且必须有小数点，如 0.25、－32.432、35.0 等；也可以用科学计数法的形式来表示，如 1.2e3 或 1.2E3，都表示 1 200.0。这里的 e 或 E 代表底数 10，e 或 E 之前必须有数字，且 e 或 E 后的指数必须为整数。

float 型常量（单精度）需在数字后面加上 f 或 F，如 1.23F 或 1.23f。

double 型常量（双精度）既可以在数字后面加 d 或 D，如 1.23D 或 1.23d，也可以直接书写。也就是说，实数默认为 double 类型。

3. 字符型数据

Java 语言采用 Unicode 字符集，即用 2 字节来存储一个字符。字符常量有普通字符常量和转义字符常量两种。普通字符常量是用单引号括起来的一个字符，如′w′、′A′、′中′等。对于被 Java 语言用做特定意义的字符，或者不能显式显示的字符，则需要用转义字符来实现，如用′\n′表示换行符。表 2-4 列出了 Java 中常见的转义字符。

表 2-4　常见转义字符

转义字符	意　　义
\ddd	1～3 位八进制数表示的字符
\uxxxx	1～4 位十六进制数表示的字符
\′	单引号字符
\″	双引号字符
\\	反斜杠字符
\r	回车
\n	换行
\f	走纸换页
\b	退格
\t	横向跳格

在 C 语言中，通常 char 型变量可与整型变量互换，Java 语言也是如此。需要注意的是，char 类型的值可以自然转换为 int 类型，而从 int 类型转换为 char 类型时需要强制执行，即强制类型转换。

由于 char 型可以自然转换为 int 型，在许多情况下可以对字符进行算数运算操作，就好像它们是整数一样。例如，可以将两个字符相加，或者对一个字符变量的值进行增减操作。

【例 2-1】 对 char 型数据进行加法运算。

程序代码如下：

```
char c = 'a';
int n;
n = c + 32;
c = (char)n;
```

后 3 行代码可以简化为：

```
c = (char)(c + 32);
```

4. 布尔类型

布尔类型数据只有两个值：true(真)和 false(假)。通过使用 Java 提供的关系运算符和逻辑运算符，对关系表达式或逻辑表达式进行运算后得到其布尔值，一般用于逻辑判断。

布尔型数据在计算机内占 1 位。需要注意的是：Java 中的布尔值与数字 0 和 1 之间是不能自由转换的，即 false 和 true 并不对应 0 和任何非 0 数值，这一点和 C 语言是不一样的。

2.2.3　运算符和表达式

1. 运算符

Java 语言的运算符和其他高级语言的运算符基本相同。按运算符需要的操作数来分，有一元运算符(如++、--等)，二元运算符(如+、-、* 等)和三元运算符(如?:)。

若按运算符的功能来分，则分为以下几类：

(1)算术运算符(+，-，*，/，%，++，--)；

(2)关系运算符(>，<，>=，<=，==，!=)；

(3)逻辑运算符(&&，||，!)；

(4)位运算符(>>，<<，>>>，&，^，|，~)；

(5)赋值运算符(=，+=，-=，*=，/=)；

(6)条件运算符(?:)；

(7)其他运算符。

2. 表达式

由运算符和操作数按照一定的语法规则组成的有意义的式子称为表达式。一个常量或者一个变量是最简单的表达式，其值就是该常量的值或变量的值。表达式的值可以作为其他运算符的操作数，从而形成更复杂的表达式。

在对一个由各种运算符组成的复杂的表达式进行求值运算时，要按照运算符的优先级顺序从高到低进行，同优先级别的运算符按照从左至右的方向进行。Java 语言的各种运算符的优先级别如表 2-5 所示。

表 2-5　运算符优先级

优 先 级	运 算 符
1	. [] ()
2	++ -- ! ~ instanceof
3	new (type)
4	* / %
5	+ -
6	>> << >>>
7	< > >= <=
8	== !=
9	&
10	^
11	|
12	&&
13	||
14	?:
15	= += -= *= /= %= ^=
16	&= |= <<= >>= >>>=

2.2.4　流程控制语句

Java 语言程序通过控制语句来执行程序流，完成一定的任务。程序流是由若干个语句组成的，语句可以是一条单一的语句，如“c=a+23;”，也可以是用花括号{}括起来的一个复合语句。Java 中的控制语句有以下几类。

(1)分支语句：if-else，switch。

(2)循环语句：while，do-while，for。

(3)跳转语句：break，continue，return。

(4)异常处理语句：try-catch-finally，throw。

(5)包处理语句：package，import。

(6)注释语句：//，/*…*/，/**…*/。

Java 语言的流程控制结构与 C 语言或者C++语言的基本相同，但仍然存在一些差异，如 Java 语言的 break 语句与 continue 语句有更多的用法，Java 中没有 goto 语句。异常处理语句是 Java 语言所特有的，将在后面的模块中详细介绍。

2.2.5　数学函数

在高级语言中，数学函数通常是作为一个系统预定义的函数库提供给用户使用的。Java 语言同样也提供了大量的库函数，通常称为类库或者程序包，同样包括了数学函数。常用数学函数如表 2-6 所示，更详细的信息可以查看 Java API 文档。

表 2-6　常用数学函数

函　数	功　能
abs(double x)	返回 x 的绝对值(x 也可以是整型)
max(double x,double y)	返回 x、y 中的较大数
exp(double x)	返回 e^x
log(double x)	返回 x 的自然对数函数值
sqrt(double x)	返回 x 的平方根
pow(double x,double y)	返回 x 的 y 方值
sin(double x)	返回 x 弧度的正弦函数值
random(double x)	返回随机数值,为 0～1 之间的 double 值

这些数学函数都包含在 Java API 预定义的类库 Math 中,其路径为 java.lang.Math。由于 java.lang 是每一个 Java 程序都会自动包含的基本类库,所以 Math 类库不用使用 import 语句导入,可以直接使用,但首字母必须大写。

【例 2-2】 计算公式 $\sqrt{a^2+b^3}$。

程序代码如下:

```
public class TestM{
    public static void main(String[] args){
        double a = 2.5,b = 3.14;
        System.out.println("" + Math.sqrt(Math.pow(a,2) + Math.pow(b,3)));
    }
}
```

编译并运行程序后的结果如图 2-1 所示。

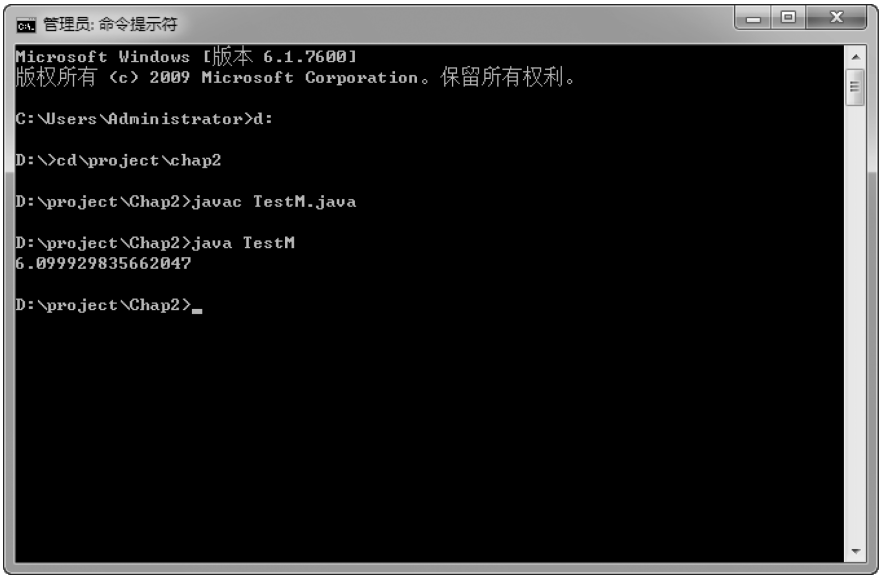

图 2-1　数学函数应用举例

2.3 案 例 实 施

2.3.1 编写源代码

【程序 2-1】 编写 DataTypeConversion.java 文件。

打开记事本(或其他文本编辑器),输入以下源代码,将程序以 DataTypeConversion.java为文件名保存到 D:\project\chap2 下。

```
public class DataTypeConversion
{
    public static void main(String[] args)
    {
        byte a = 65;                          //声明 byte 型变量并初始化
        short b = 135;
        int c = 21341;
        long d = 31422100;
        float f;
        double g = 123456789.987654321;
        b = a;                                //自动类型转换
        c = (int)d;                           //强制类型转换,把 long 型转换为 int 型
        f = (float)g;
        System.out.print("a = " + a + ";");
        System.out.println("b = " + b + ";");
        System.out.print("c = " + c + ";");
        System.out.println("d = " + d + ";");
        System.out.println("f = " + f + ";");
        System.out.println("g = " + g + ";");
    }
}
```

程序首先声明了 4 个整型变量并为其赋初值,分别是 a(byte 型)、b(short 型)、c(int 型)和 d(long 类型),然后声明了两个浮点型变量 f 和 g,分别是 float 类型和 double 类型。这 6 个数值类型变量的精度不同,在内存中所占的存储空间也不同,因此,在不同类型的变量之间赋值时需要考虑数据的兼容性问题。

一般情况下,低精度的类型向高精度的类型转换时系统可以自动实现,如本程序中的语句“b=a;”,byte 类型自动转换为 short 类型。反之,即高精度类型向低精度类型转换时,则必须进行强制类型转换,如本程序中的语句“c=(int)d;”,把 long 类型强制转换为 int 类型,然后赋值给变量 c。以下所示的从左到右的转换都是自动转换的:

byte→short→int→long→float→double

2.3.2　编译并运行程序

在命令提示符窗口下将路径切换到 D:\project\chap2，然后输入：

```
javac DataTypeConversion.java
```

编译完成后，输入以下命令运行程序：

```
java DataTypeConversion
```

程序运行结果如图 2-2 所示。

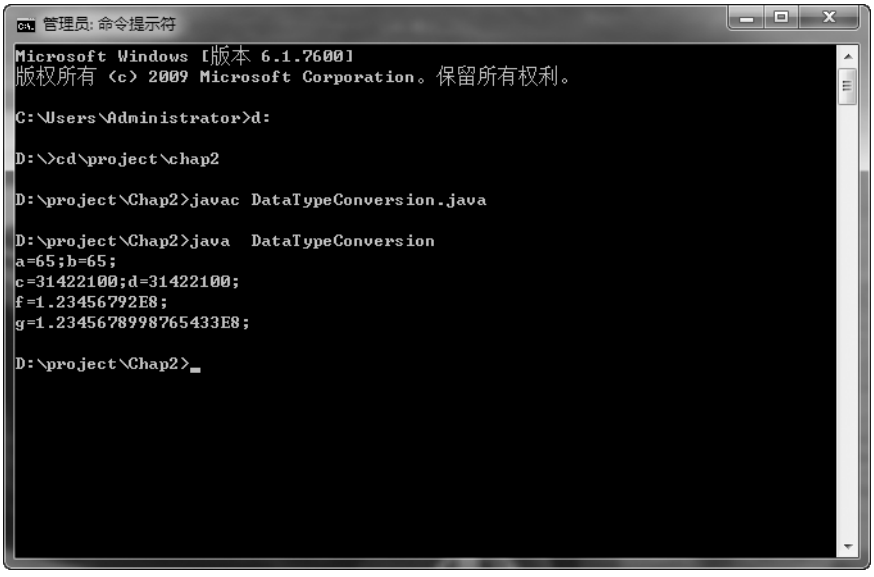

图 2-2　数据类型转换程序运行结果

从运行结果中可以发现：double 类型的变量 g 通过强制类型转换，赋值给 float 类型的变量 f 时出现了数据精度的损失。这是强制类型转换后精度不匹配的必然结果。

2.3.3　调试程序

我们尝试将【程序 2-1】中的赋值语句“b＝a;”改写为“a＝b;”。显然这是一个错误的语句，因为变量 b 是 short 类型，不能直接赋值给 byte 类型的变量。编译更改后的程序，javac 编译器将报错并指出错误之处，如图 2-3 所示。

从图 2-3 中可以看出，错误代码位于第 11 行，原因是数据类型不匹配导致的精度损失。这也是以后的学习中经常会遇到的情况，因此，我们要学会分析编译错误的原因，定位错误位置并改正错误。

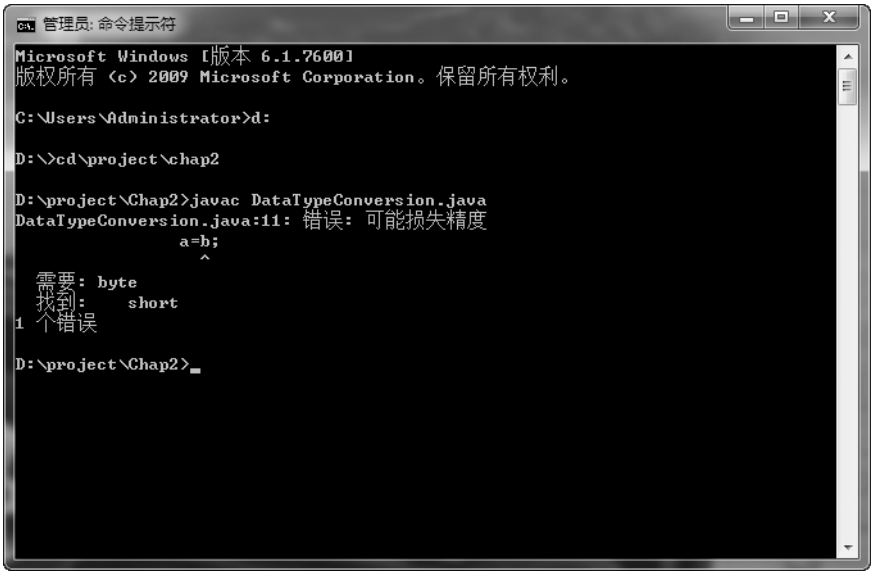

图 2-3　系统报错

2.4 训练与实战

通过上面的案例可以看到，Java 语言在标识符、关键字以及基本数据类型方面与 C 语言或者 C++语言非常相似，同时 Java 是一种强类型语言，即所有的变量、表达式都必须有确定的类型，Java 编译器对所有的表达式都进行类型相容性检查，以保证类型是兼容的。任何类型的不匹配都是错误的，在编译器完成编译以前，错误必须被改正。下面通过两个具体的实战训练项目来熟悉 Java 的流程控制语句等 Java 的基本语法规则。

2.4.1 输出小于 N 的所有素数

输出小于 N 的所有素数是一个经典的程序控制范例。完成该项目需要使用 Java 的循环控制语句逐一测试从 1～N 之间的所有正整数，并使用条件控制语句来判断每个整数是不是素数，把其中的素数逐一显示输出。

判断一个整数 m 是否为素数，只需要用 2～$\sqrt{m}$ 之间的所有整数去除 m，只要其中有一个数能整除 m，则 m 为非素数，否则 m 为素数。

训练内容

编写一个 Java 应用程序，输出小于 100 的所有素数。

训练过程

1）编写源代码

【程序 2-2】　编写 PrimeNumber. java 文件。

打开记事本，输入以下代码，将程序以 PrimeNumber.java 为文件名保存到 D:\project\chap2 下。

```
public class PrimeNumber
{
    public static void main(String[] args)
    {
        int n,m,k;
        n = 100;
        System.out.println("小于" + n + "的素数有:");
        for(m = 2;m<n;m++)                    //测试 2～100 之间的所有整数
        {
            for(k = 2;k<=Math.sqrt(m);k++)
                if (m%k == 0) break;          //判断 m 能不能被 k 整除
            if(k>Math.sqrt(m))                //2～√m都不能整除 m,则 m 是素数
                System.out.print("\t" + m);   //输出素数,但不换行
        }
        System.out.println();
    }
}
```

本程序使用了 for 循环的嵌套来实现程序流程的控制，用 n 来控制外层 for 循环，用 k 来控制内层 for 循环；用 if 语句来判断整除性以及是否为素数；用数学函数 Math.sqrt(m)返回 m 的平方根的值。

2)编译和运行程序

在命令提示符窗口下将路径切换到 D:\project\chap2，然后输入：

```
javac PrimeNumber.java
```

编译完成后，使用下面的命令运行程序：

```
java PrimeNumber
```

运行结果如图 2-4 所示。

从运行结果上可以看出两个素数之间都空有一定的距离，这其实就是程序中输出的转义字符“\t”的作用，代表一个制表符，即横向跳格，相当于按一次 Tab 键。

总结与体会

本训练用 Java 实现了一个经典的判断素数的程序，在程序中使用了顺序、选择和循环 3 种基本程序控制结构，从中可以看出 Java 语言在程序控制语句方面和 C 语言并没有太大的区别。

在编程时，通过 for 循环语句可以很容易实现类似的循环控制问题。如果用 break 语句配合 if 语句一同使用，也可以达到控制程序流程的目的。【程序 2-2】中当某个 m 和 k 求余结果等于 0 时(m%k==0)，即可确定 m 不是素数了，将提前终止内层的 for 循环，提前开始新的外层 for 循环。

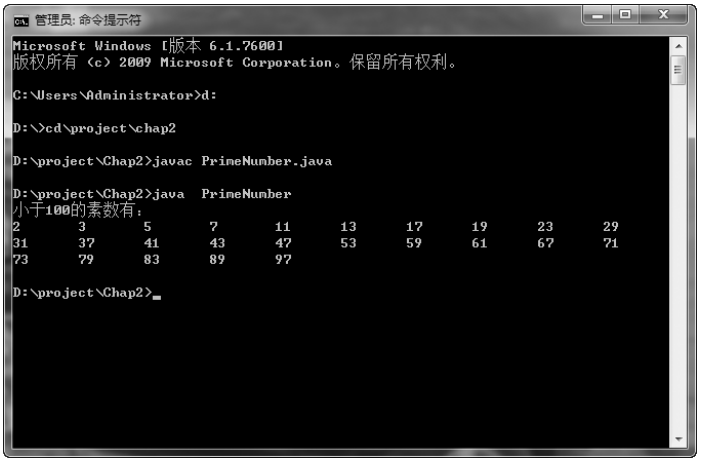

图 2-4　输出小于 100 的素数程序运行结果

2.4.2　猜数游戏

猜数游戏要实现的功能并不复杂，从程序控制的角度来说主要是使用循环控制语句，但需要用户在程序运行的过程中与程序进行交互，因此需要用到全新的数据输入方式。

训练内容

猜数游戏的规则是：程序随机产生一个 100 以内的正整数 k，然后由用户猜测这个数，用户猜测的数输入后保存在 g 中，程序通过比较 k 和 g 大小，反馈给用户“猜中了”、“猜大了”和“猜小了”3 种信息中的一种，当且仅当用户猜中了这个数，游戏才会结束，否则用户需要继续猜下去，直到猜中为止。

训练过程

1）编写源代码

【程序 2-3】　编写 GuessingGame.java 文件。

打开记事本程序，输入以下代码，将程序以 GuessingGame.java 为文件名保存到 D:\project\chap2 下。

```
import java.io.*;                                    //导入输入/输出包
public class GuessingGame
{
    public static void main(String[] args) throws IOException
    {
        int k=(int)(Math.random()*100);      //生成一个 0～100 之间的整数
        int g;
        do
        {
            System.out.println("请输入您猜测的数字(0-100)");
```

```
            BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
            g = Integer.parseInt(in.readLine());
                                    //将输入的字符串型数据强制转换为整型
            if(g! = k){
                if(g>k)
                    System.out.println("猜大了!");
                else
                    System.out.println("猜小了!");
                System.out.println();
            }
        }while(g! = k);
        System.out.println("恭喜你! 猜中了,数字为:" + g);
    }
}
```

在【程序 1-3】中，我们已经接触了 Java 通过控制台输入数据的简单方法。本程序中，使用了新的 IO 流类来读取数据。代码“BufferedReader in＝new BufferedReader(new InputStreamReader(System.in));”表示从标准输入设备 System.in 读取数据到一个字符输入流 InputStreamReader，再将字符数据送入字符缓冲流 BufferedReader。

代码“g＝Integer.parseInt(in.readLine())”表示从字符缓冲流中读取一行字符，形成字符串，并将其强制转换为 int 型数据，然后赋值给 g。从输入流读取的数据常常是字符串的形式，而在程序中可能要使用各种其他的数据类型，这就需要进行数据类型转换。将输入字符串转换为其他数据类型的方法如表 2-7 所示。

表 2-7　字符串转换

方　法	功　能
Integer.parseInt(String s)	返回 int 型
Long.parseLong(String s)	返回 long 型
Float.parseFloat(String s)	返回 float 型
Double.parseDouble(String s)	返回 double 型

由于程序用到了输入/输出流类，所以使用 import java.io.* 导入相关的类包，所涉及的 Java 输入/输出的内容将在后续模块中详细讲解。

2)编译和运行程序

在命令提示符窗口下将路径切换到 D:\project\chap2，然后输入：

```
javac GuessingGame.java
```

编译完成后输入下面的命令运行程序：

```
java GuessingGame
```

根据程序提示信息输入数字，即可进行猜数游戏，程序一次运行的结果如图 2-5 所示。

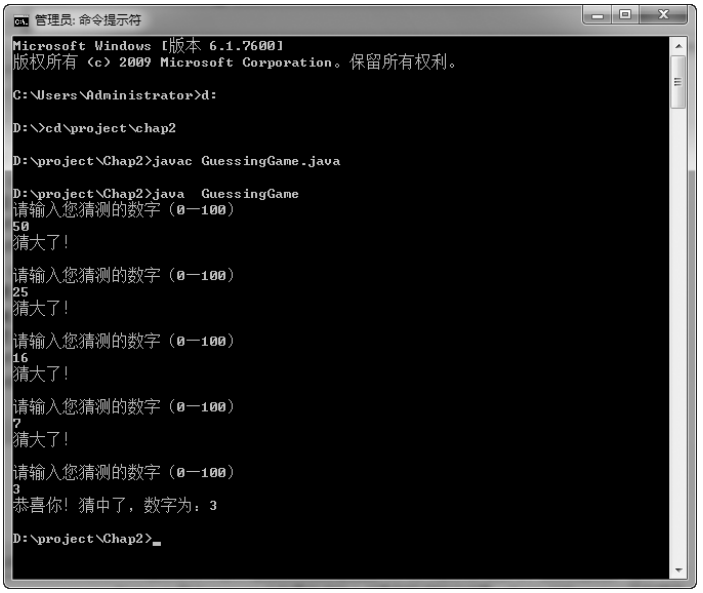

图 2-5　猜数游戏程序某次的运行结果

程序首先提示用户输入猜测的数字，输入 50，程序反馈“猜大了”，并提示用户继续猜。不断重复，直到猜到正确数字 3 结束。由于数字是系统随机产生的，程序每次运行的情况都可能不一样。

总结与体会

本项目实现了一个简单的猜数游戏。这个程序如果用 C 语言来实现或许更简单些，因为使用 Java 语言来实现，需要用到全新的输入数据和转换数据类型的方法。想要彻底理解这些新方法有待继续学习，目前应该学会在自己的程序中使用这些方法。

2.5 实际工作中常见问题解析

2.5.1 死循环问题

1. 问题

在使用循环控制语句编程时出现死循环现象，即循环无法终止或者程序的运行结果不正确。

2. 问题分析与解决

此类问题通常是由于循环控制的条件设定不当造成的。无论是 while 语句、do-while 语句，还是 for 语句，在构造循环结构程序时，必须处理好 3 个环节才能保证循环的顺利运行：一是循环之前循环变量的初始化，即循环的开始点；二是循环条件的设定必须准确；三是循环语句中必须有能够改变循环条件的关键性语句，否则循环可能不正确或者造成死循环。

下面的代码段用于实现前 100 个数求和。

```
int sum = 0;
int i = 1;                    //循环之前完成循环变量 i 的初始化,即赋值为 1
while(i<=100)                 //循环条件的设定,i 小于等于 100
{
    sum = sum + i;
    i ++ ;                    /* 能够改变循环条件的关键性语句,随着 i 的不断增加,总有
                                 一个时刻使循环条件(i<=100)为假,即循环结束 */
}
System.out.println("sum = " + sum);
```

另外,当循环体有多条语句时,必须使用花括号({})把这些语句括起来,构成一个复合语句,否则循环也达不到预期的目的。上面的代码段就使用了复合语句。

2.5.2　使用输入/输出类导致编译错误

1. 问题

程序代码中使用了 Java 的输入/输出类,程序编译时报错,或者运行时出现异常。

2. 问题分析与解决

此类现象常常是因为没有在程序中引入对应的 Java API 类库造成的。解决方法很简单,只需在程序的前面加上语句“import java.io.*;”即可。

2.6　习　　题

一、思考题

1. Java 的基本数据类型与 C 语言的相比有哪些不同?
2. 为什么高精度的数据类型不能自动转换为低精度的数据类型?
3. 为什么 char 类型数据可以直接进行算术运算?

二、编程题

1. 输入圆的半径 r,计算并输出圆的面积和周长。
2. 输入正整数 n,输出 n 行由“*”组成的等腰三角形图案,其中第一行为 1 个“*”,第二行为 3 个“*”,第 i 行为 $2i-1$ 个“*”($i \leqslant n$)。下面是当 $n=5$ 时的图案。

```
    *
   * * *
  * * * * *
 * * * * * * *
* * * * * * * * *
```

3. 用 Java 语言编写程序，计算下列公式的值，变量的值自拟。

(1) $a^2+b^5+c^{\frac{1}{2}}$ 。

(2) $x\log_2 y$ 。

(3) $\sin x+|\cos(\frac{x}{2})|$ 。

4. 编写程序，输出 Fibonacci 数列(斐波纳契数列)的前 50 项。

提示：斐波纳契数列指的是这样一列数：1、1、2、3、5、8、13、21、…即第 1 项和第 2 项的值为 1，从第 3 项开始，每一项都等于其前两项之和。

模块3

类和对象

有一个问题：世界是由什么组成的？不同的人肯定有不同的答案。作为面向对象编程的程序员来说，要站在分类的角度去考虑问题：这个世界由动物、植物、物品、名胜等组成，动物分为单细胞动物、多细胞动物、哺乳动物等，哺乳动物又分为人、老虎、狮子、大象……我们看到的每一个事物都是一个对象。在 Java 的世界中，万物皆对象。

本模块讲解面向对象编程的一些基本概念——对象、类和封装，以及如何在 Java 中实现类、对象和方法。

学习目标

- 理解对象。
- 理解类。
- 理解封装。
- 理解对象与类之间的关系。
- 会定义和使用类的方法。
- 理解变量的作用域。

3.1 案例引入——用程序来描述学生特征

案例说明

现实生活中“学生”身份的人群特点很明显，可以从不同的方面来描述这一类人，如姓名、年龄、班级编号及兴趣爱好等，也可以从行为描述。现在要使用计算机程序语言来描述学生特征。

案例分析

使用 Java 语言描述学生特征，首先要掌握对象、类、属性、方法的概念，学会封装的方法，进而使用程序解决描述学生特征的需求。

3.2 知 识 准 备

3.2.1 面向对象概述

面向对象编程（object oriented programe，OOP）是指将现实世界中的概念模拟到计算机程序中，它将现实世界中的所有事物视为对象，如一辆自行车、一辆汽车、一本书等。对象具有属性和行为，例如，汽车有颜色、速度等属性，还有加速、减速和制动等行为。Java 就是一种面向对象的编程语言，我们要学会用面向对象的思想思考问题，编写程序。

面向对象编程思想的核心是对象（object）。对象是指现实世界中的实体。OOP 能够将现实世界中的实体模拟为计算机上的类似实体，有关这个过程如图 3-1 所示。

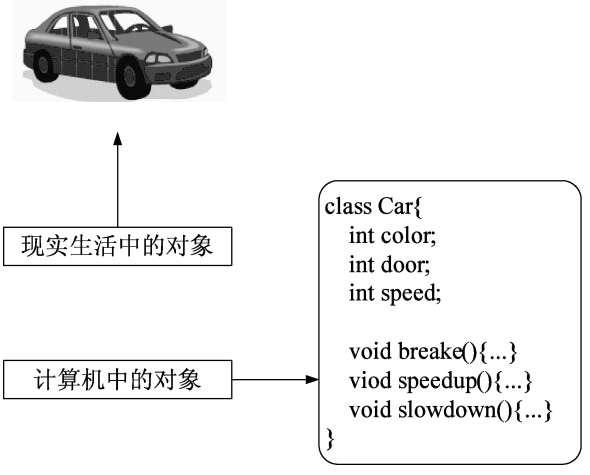

图 3-1 将现实问题模拟为计算机实体

为了更好地理解如何将现实生活中的实体或事物模拟成面向对象世界中的对象，不妨以超市为例。假设某超市设有采购部、仓储部和销售部等不同的部门，每个部门在超市中都起着特定的作用：采购部负责发出所需商品的订单和采购这些商品；仓储部负责组织商品、跟踪库存和维护库存商品；销售部负责生成账单和与顾客打交道等事务。

以下是超市中涉及的几个元素。

1. 实际对象

实际对象主要有：

- 超市雇员（收银员、促销员、系统管理员、仓库管理员和采购经理）；
- 超市购物车；
- 顾客；
- 商品。

2. 业务对象

业务对象主要有：

- 超市库存；
- 顾客的订单。

其中每个对象都有各自的特征和行为，用于描述它是什么和能执行什么任务。以收银员对象为例，收银员对象的特征包括：

- 姓名；
- 职衔；
- 年龄；
- 体重。

收银员对象能够执行的操作包括：

- 向顾客收款；
- 打印账单。

收银员对象的属性和行为描述如图 3-2 所示。此时的收银员对象就可以看做是计算机对象。

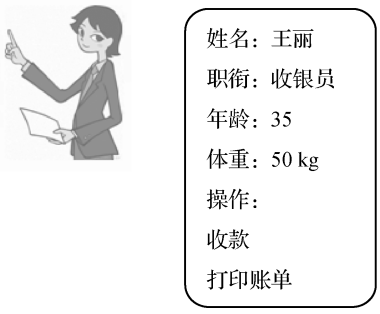

图 3-2　收银员对象

3.2.2 类和对象的相关概念

多个对象所共有的属性和操作需要组合成一个单元，称为“类”。如果将对象看做房子，那么类就是房子的设计图纸。

1. 类的定义

类是具有相同特征和共同行为的一组对象的集合。

类定义了对象所拥有的特征（属性）和行为（方法），定义了一种对象所能拥有的数据和能完成的操作。在面向对象的程序设计中，类是程序的基本单元。程序中的对象是类的一个实例，是一个软件单元，由一组结构化的数据和在其上的一组操作构成。

在现实世界中，对象正是拥有了这些特性才变得与众不同，因此构成对象的两个最重要的因素就是特征（属性）和行为（方法）。在软件技术领域，我们尽可能用对象来模拟现实世界的实体，因此对象都具有属性和方法。对象的属性就是现实世界中对应实体的特征，对象的方法就是现实世界中对应实体能够执行的动作。

在前面讨论的内容中，收银员、促销员、系统管理员、仓库管理员和采购经理等均为雇员类的对象。

1）类的属性

对象或者实体的特征在类中表示为成员变量，称为类的属性。例如，每一个雇员对象都有姓名、年龄和体重，它们是类中所有雇员共有的公共属性。

属性的定义：对象或实体拥有的特征在类中的表示称为属性。

2）类的方法

方法是对象执行操作的一种规范。方法用于指定以何种方式操作对象的数据，是操作的实际实现。

方法的定义：对象执行的操作称为方法。

2. 类和对象的区别

类和对象是有本质的区别的：类是用来描述实体的“模板”或“原型”，而对象是实际的实体，每一个对象都是类的一个实例。所以，通常“为类创建一个对象”也称做“为类创建一个实例”。

对象是实体，而类是概念模型，用来定义对象的所有特征和所需操作。同一类的所有对象都拥有相同的特征和操作。

（1）类是对象的“原型”，它为特定类型的对象指定了允许的操作和必要的特征。表 3-1 给出了类和对象的更多示例。

表 3-1　类和对象示例

类	对　象
动物	一只叫“咪咪”的花猫
	一只叫“笨笨”的猎狗
汽车	一辆红旗轿车
	一辆奔驰轿车

当为类创建一个对象后，对象的存在期限是有限制的，被创建的对象在适当的时期将被销毁。

(2)类是对象的类型。到目前为止，我们已经学习了 Java 的不少数据类型，如整型、单精度浮点型、双精度浮点型、字符型、布尔型。这些都是 Java 语言中定义好的数据类型，当运算时直接使用它们即可。而定义类就是定义了一个用户自己需要的类型，不是系统预定义的，例如，“人”、“动物”、“车”等。类与其他简单数据类型不同的是，类具有方法。

3. Java 的类模板

在面向对象程序设计中，类是程序的基本单元。Java 是一门纯粹的面向对象的程序设计语言，所有程序都是以类为组织单元。

Java 类模板如下：

```
public class 类名
{
        //定义属性部分
        属性 1 的类型 属性 1;
        属性 2 的类型 属性 2;
        …
        属性 n 的类型 属性 n;
        //定义方法部分
        方法 1;
        方法 2;
        …
        方法 m;
}
```

在 Java 中创建一个类，需要使用 class 关键字、一个类名和一个表示程序体的花括号。其中，class 是创建类的关键字。在 class 前面有一个 public，表示“公有”的意思，用于修饰类的使用范围，其具体的含义在后面的模块介绍。class 后面是类名，并且要写一对花括号({})，类的主体部分就写在这对花括号中。

类名也是标识符，所以给类命名也要遵守标识符命名的规则和规范。另外，Java 类的命名通常还应遵守以下命名规范。

(1)类名是一个名词，采用大小写混合的方式，每个单词的首字母大写。

(2)类名应该简洁且易于描述，能够见名知义，并使用完整的单词，避免缩写词。

4. 定义一个类的步骤

类将现实世界中的概念模拟到计算机程序中，类是对象的模型，确定对象拥有的特征(属性)和行为(方法)。因此，定义一个类需要抽取同一种对象的属性和方法的共同特征，步骤如下。

(1)定义类名。编写类的最外层框架：

```
public class 类名
{
```

```
    //程序体
}
```

(2)编写类的属性。在类中,数据和下面要提到的方法统称为类成员。其中,类的属性就是类的数据成员。我们通过在类的主体中定义变量来描述类所具有的特征(属性),在这里声明的变量称为类的成员变量。

(3)编写类的方法。类的方法描述类所具有的行为,是类的方法成员。

以下代码段定义了一个类 Person,用以描述它的特征和行为:

```
class Person
{
    //定义属性
    String  name;          //姓名
    int  age;              //年龄
    //定义方法
    public void hello()
    {
        System.out.println("喂,你好,我的名字叫"+name+",今年"+age+"岁!");
    }
}
```

类 Person 定义了姓名和年龄两个属性,同时定义了自我介绍方法 hello()。

5. 创建和使用对象

Person 类定义好了以后,就可以根据定义的模型创建对象了。由类生成对象的过程称为类的实例化过程。一个实例就是一个对象,一个类可以生成多个对象。创建对象的语法如下:

```
类名  对象名 = new 类名();
```

Java 中使用 new 关键字来创建类的一个对象,如"Person firstPerson=new Person();",这里的 firstPerson 就是 Person 类的一个对象。但是这时我们并没有给它的数据成员(即对象的属性)指定特定的数据,创建对象后,就需要给对象的数据成员赋予特定的数据。在Java中,要引用对象的属性和方法,需要使用点"."操作符来访问。引用对象的属性和方法的语法格式如下:

```
对象名.属性
对象名.方法名()
```

【例 3-1】 定义一个测试类,创建和使用 Person 类的对象。

程序代码如下:

```
class Person
{
    //定义属性
    String name;      //姓名
    int age;          //年龄
    //定义方法
```

```
    public void hello()
    {
        System.out.println("喂,你好,我的名字叫"+name+",今年"+age+"岁!");
    }
}
public class TestPerson
{
    public static void main(String[] args)
    {
        Person person = new Person();          //创建一个 Person 对象
        person.name = "李丽";                  //引用 name 属性
        person.age = 18;                       //引用 age 属性
        person.hello();                        //调用 hello()方法
    }
}
```

程序编译、运行之后的输出结果如图 3-3 所示。

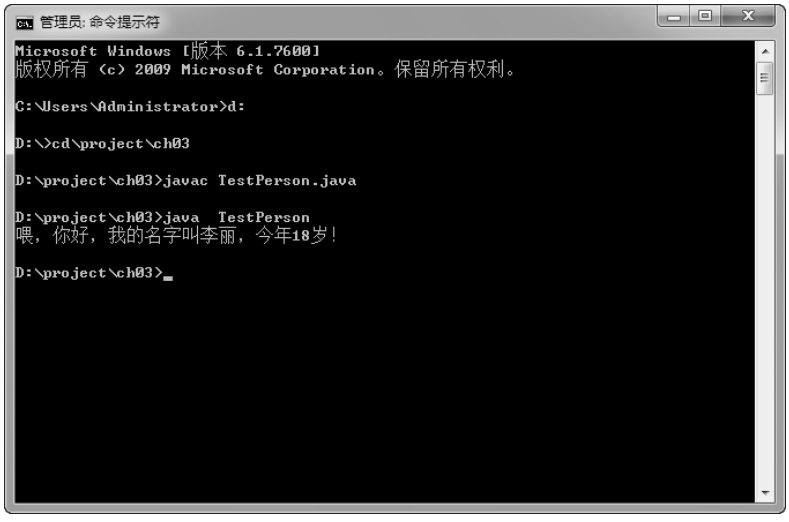

图 3-3　创建和使用类的对象程序运行结果

3.2.3　方法

类由一组相同特征和共同行为的实体抽象而来。对象的行为通过编写类的方法来实现。

1. 引例

显而易见,类的方法是一个功能模块,其作用是“做一件事情”。我们举例说明什么是类的方法。如图 3-4 所示是一只电动玩具狮子,在它身上有一个按钮,装上电池,按动按钮,电动狮子就会跑起来。

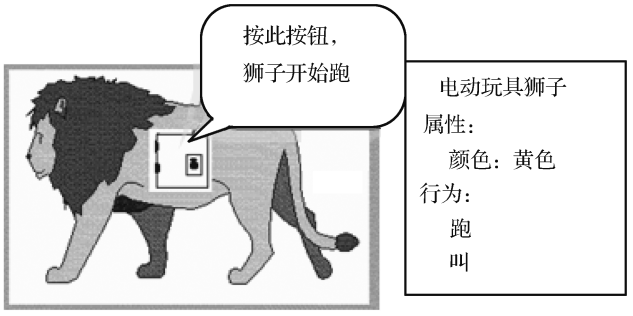

图 3-4　电动玩具狮子

以下代码段创建一个电动狮子(AutoLion)类，描述它的属性和行为：

```
class AutoLion
{
    String color = "黄色";
    public void run(){
        System.out.println("正在以 0.1 米/秒的速度向前奔跑");
    }
    public String bark(){
        String sound = "吼";
        return sound;
    }
}
```

在 AutoLion 类中定义了两个行为(方法)："跑"和"叫"。厂商提供了电动玩具，也提供了一个功能接口，也即那个"按钮"。我们不需要了解电动玩具的内部构造，只要按这个按钮即可。电动玩具能跑能叫的内部实现就相当于类中的方法，这正是面向对象语言描述现实世界事物的组织形式。

类的每个方法定义一个功能，定义了类的对象之后就可以调用这些方法了，调用某个方法时无须知道这个方法的代码是如何编写的。例如，电动狮子(AutoLion)类定义了 run()方法和 bark()方法，在创建了电动狮子对象之后就可以调用这两个方法了。可见，类的方法定义了类的某种行为(功能)，而且由于被封装在类中，实施细节被隐藏起来了。

2. 方法的定义

方法的定义共有 4 个部分，具体包括：

(1)方法的名称；

(2)方法返回的值的数据类型；

(3)参数列表；

(4)方法的主体。

方法的声明是上述前 3 部分的组合。

定义类的方法的语法格式如下：

```
<returntype> <methodname> (<type1> <arg1>,<type2> <arg2>,…)
```

```
{
    <set of statements>
}
```

其中，returntype 是方法返回值的数据类型；methodname 是用户自定义的方法名称；参数列表是一组变量声明，多个参数之间用逗号隔开。

通常，定义类的方法需要以下两个步骤来完成。

(1)定义方法名、参数以及返回值类型。

(2)编写方法体。

另外，在定义类的方法时要注意以下几点。

(1)由一对花括号括起来的语句是方法的主体，它包含一段程序代码，执行时完成一定的功能。

(2)方法名主要是在调用此方法时使用，其命名方法应遵守标识符命名的规则。另外，通常方法名是一个动词，如果由两个以上单词组成，第一个单词的首字母小写，其后单词首字母大写。

(3)方法就像“黑匣子”，完成一个功能，并且在执行后可能返回一个结果。在方法的主体内，如果方法具有返回类型(非 void 类型)，则必须使用关键字 return 返回数据。即方法的返回值有以下两种情况。

①如果方法具有返回值，那么方法中必须使用关键字 return 返回该值，返回类型为该返回值的类型。return 语句的语法格式如下：

```
return <表达式>;
```

②如果方法没有返回值，则方法的返回值的数据类型为 void。

例如，有下面的方法：

```
public String toString(){
    return "这是一个" + color + "的玩具狮子," + "可以跑可以叫。";
}
```

在该方法中，返回值类型是 String，因此必须用 return 返回一个字符串。

如果方法没有返回值，则返回值类型应该使用 void，用以说明无返回值。例如：

```
public void run()
{
    System.out.println("正在以 0.1 米/秒的速度向前奔跑");
}
```

因此，在编写程序时一定要注意方法声明中返回值的类型和方法体中真正返回值的类型是否一致。

Java 提供了 3 种跳转语句：break、continue 和 return。break、continue 语句在模块 2 中已经介绍过，不再赘述，这里简单介绍 return 语句的用法。

return 语句有以下两种功能。

(1)该段程序代码已经运行完成，现在要离开这个方法。

(2)如果方法产生一个值，这个值放置在 return 后面，即<表达式>部分，把数据带回到调用此方法的位置。

常见错误 1：

```
public void hello(){
    …
    return "大家好!";
}
```

错误原因是方法声明中返回值的类型和方法体中真正返回值的类型不一致。

常见错误 2：

```
public class Student{
    public double getInfo(){
        double weight = 95.5;
        double height = 1.69;
        return weight,height;
    }
}
```

错误原因是方法至多可以返回一个值，不能返回多个值。

3. 方法的调用

在程序中，使用方法的名称可以执行方法中包含的语句，这个过程称为方法调用。方法调用的一般形式如下：

```
对象名.方法名();
```

在 Java 中，每个对象都需要完成特定的应用程序功能。当需要某一对象执行一项特定操作时，通过调用该对象的方法来实现。另外，类的不同成员方法之间也可以相互调用。

【例 3-2】 定义电动狮子类，即 AutoLion 类，并在该类的 showLion()方法中调用该类的另一个方法 getColor()。

程序代码如下：

```
class  AutoLion{
    String color = "黄色";
    public void run(){
        //方法 1:跑
        System.out.println("正在以 0.1 米/秒的速度向前奔跑。");
    }
    public String bark(){
        //方法 2:叫
        String sound = "吼";
        return sound;
    }
    public String getColor(){
        //方法 3:获得颜色属性
        return color;
    }
```

```
    public String showLion(){
        //方法 4:描述狮子特性
        return "这是一个" + getColor() + "的玩具狮子!";
    }
}
public class TestLion{
    public static void main(String[ ] args){
        AutoLion lion = new AutoLion();
        System.out.println(lion.showLion());
        lion.run();
        System.out.println(lion.bark());
    }
}
```

程序编译、运行后的结果如图 3-5 所示。

图 3-5　方法调用程序运行结果

上例在测试类(TestLion)中定义程序入口,检验电动狮子类的跑、叫功能是否可以正常进行,即在测试类的 main()方法中调用 AutoLion 类的 run()方法和 bark()方法。

类中的成员方法相对独立地完成程序的某个功能,它们之间可以相互调用,调用时仅仅使用成员方法名即可。showLion()是 AutoLion 类的方法成员,因此它可以直接调用该类另一个方法成员 getColor(),调用时直接引用方法的名称 getColor()即可。但是,其他类的方法要调用 AutoLion 类的成员方法时,就必须创建 AutoLion 类的一个对象,然后通过“.”运算符使用对象的成员方法。

4. 构造方法

构造方法是指在创建给定类的实例时调用的一个方法。它们与类同名,但不具备任何返回类型。

当使用关键字 new 创建类的实例时,Java 为对象分配内存,初始化实例变量并调用构

造方法。每个类都定义有构造方法，以便初始化其成员变量。在创建类的对象时，将自动调用该类的构造方法，可以将值传递给构造方法。因此，构造方法可通过初始化成员变量和创建对象的环境来初始化对象。

构造方法有两种类型，分别是参数化构造方法和隐式/默认构造方法。

1）参数化构造方法

此类构造方法可在类定义中进行编码。在创建类的对象时，传递的值和构造方法的参数应当在个数、次序和数据类型上匹配。下面的代码片段是带参数构造方法的定义：

```
class Book{
    String bookName;
    String authorName;
    int nopages;
    boolean available;
    Book(String book,String author,int pages,boolean status){
        bookName = book;
        authorName = author;
        nopages = pages;
        available = status;
}
...
```

2）隐式/默认构造方法

当一个类未定义构造方法时，Java 虚拟机（JVM）便提供一个默认构造方法。它不带任何参数，其主体不含任何语句。【例 3-2】中演示了隐式构造方法的使用。

构造方法的特点如下。

（1）它的名字与类同名。

（2）没有返回值类型也不能用 void 修饰。

（3）一个类可以有不同参数列表的构造方法，即构造方法可以重载（方法重载在模块 4 中介绍）。

（4）如果程序员未定义构造方法，系统会提供默认构造方法。

（5）如果程序员定义了一个或多个构造方法，则系统将自动屏蔽默认的构造方法。

（6）构造方法不能说明为 native、abstract、synchronized 或 final，也不能从父类继承构造方法。

5. 变量的作用域

Java 中用类来组织程序，类中可以定义成员变量和成员方法。在类的方法内也可以定义变量，但方法内的变量和方法外的变量在使用上是有区别的，即涉及变量的作用域问题。

变量作用域是指可在程序中按变量名访问该变量的区域。变量的作用域与变量的声明位置有关，不同的声明位置决定了变量不同的作用域。

图 3-6 所示表明了变量的作用域。在类中定义的变量称为类的成员变量，如变量 1、变量 2 和变量 3，AutoLion 类的方法可以直接使用它们；如果要在其他类的方法访问它们，必须首先创建 AutoLion 类的对象，然后通过对象的点运算符来引用。在方法中定义的变量称

为局部变量，如变量 4 和变量 5。局部变量的作用区域只是在定义它的方法内，因此只有这个方法可以使用它。

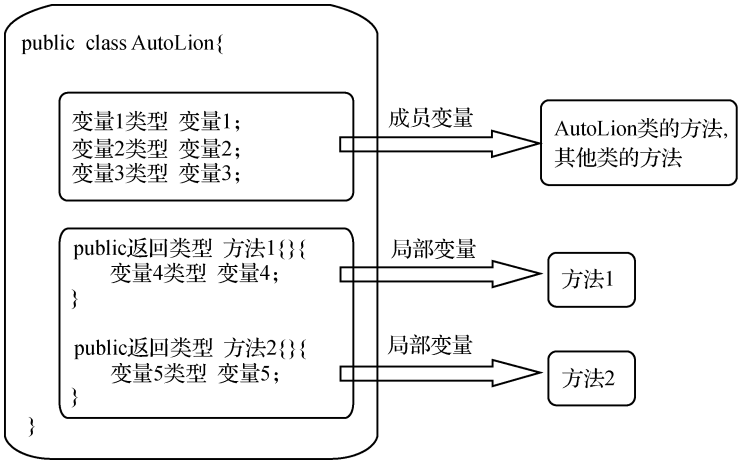

图 3-6　变量的作用域

3.2.4　this 关键字和 static 关键字

1. this 关键字

Java 中包含一个名为 this 的特殊引用值。this 关键字在任何实例方法中都可使用，以引用当前对象。this 是在对象的内部指代自身的引用。以关键字 static 声明的方法（类方法）不能使用 this 关键字。下面的代码片段演示了 this 关键字的用法。

```
public class Pixel{
    int x;
    int y;
    void init (int x,int y){
        this.x = x;
        this.y = y;
    }
    public static void main(String args[])
    {
        Pixel p = new Pixel();
        p.init(4,3);
    }
}
```

因为 this 可以直接引用对象，所以可以用它来解决 Java 的同一程序段内不同作用域的同名变量问题，即解决 Java 类中实例变量和局部变量的同名冲突。

2. static 关键字

有时候程序员需要定义一个类成员，对它的使用不依赖于该类的任何对象。要创建这

样的成员，成员变量声明前必须加上关键字 static。成员声明为 static 后，在创建类的任何对象之前就可以访问它。例如，类中的 main()方法声明为 static，则不用创建该类的实例就可以在运行时由系统调用 main()方法。

关键字 static 可应用于变量、方法，甚至不属于方法的一段代码块。创建包含静态成员的类的对象时，不会生成静态变量的副本。正是由于静态变量没有副本，所以某些静态变量为类的所有实例共享。可以通过类名称来调用静态变量，也可以通过对类实例的引用来调用静态变量，但建议使用第一种方法。

方法声明为静态时具有某些限制，具体内容如下。

(1)它们只能调用其他静态方法。

(2)它们必须只访问静态数据。

(3)不能使用关键字 this 和 super。

【例 3-3】 关键字 static 的用法。

```
/**
 * 这个程序演示关键字 static 的用法.
 * @author jake
 */
public class InchesToFeet
{
    /** 初始化静态变量 */
    private static final int inches = 12;
    /** 构造方法 */
    protected InchesToFeet(){
    }
    /**
     * 显示值
     * @param in 传递至 convert 方法的参数
     * @return 传递给方法的 in
     */
    public static double convert(double in)
    {
        return (in / inches);
    }
    static{
        System.out.println("这是首先执行的静态块");
    }
    /**
     * 这是 main 方法.
     * @param arg 传递至 main 方法的参数
     */
```

```
    public static void main(String [] arg)
    {
        // 初始化变量
        double inch = 66;
        double feet = InchesToFeet.convert(inch);
        System.out.println(inch + "inch 为" + feet + "英尺.");
    }
}
```

程序经过编译、运行后的结果如图 3-7 所示。

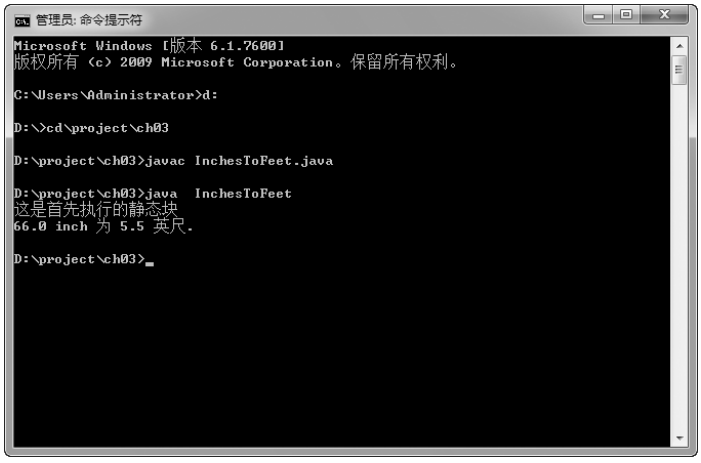

图 3-7　【例 3-3】程序运行结果

此程序一加载，所有的静态语句便开始运行。首先，inches 设置为 12，然后，执行 static 块。变量 inches 将用来存放常量值，直到程序员想要更改。由 public 和 static 修饰的 convert()方法返回一个双精度值。main()方法调用静态方法 convert()，并用 inch 传入值。

在 main()方法中是用点操作符来调用静态方法 convert()的。这就说明可以从类中直接调用静态方法，而不用实例化该类的任何对象。其格式为：

```
Classname.method();
```

3.2.5　封装与访问控制

面向对象编程(OOP)最重要的特点之一是信息隐藏。类是数据和方法的集合，这意味着类以外的代码不能直接使用类的数据，而只能通过方法来访问。

1. 封装的概念

顾名思义，封装是指将东西包装在一起，然后以新的完整形式呈现。对于 OOP 而言，封装是将方法和属性一起包装到程序单元中，这些单元以类的形式实现。以药用胶囊为例，胶囊中存在着各种各样的化学药品，这些化学药品共同形成了药物，用于治疗各种疾病。

在 OOP 中，每当定义类/对象时，往往会将相互关联的数据和功能绑定在一起，就像用药用胶囊把化学药品包装起来一样，这种做法称为封装。用计算机专业术语讲，隐藏属性、

方法和实现的详细信息的处理方式称为封装。

封装的好处之一是隐藏信息。“信息隐藏”是一个强大的技术工具，它能降低程序的复杂性。在创建类时，会根据完成任务的需要创建许多属性和方法，而只有被其他程序访问的那些属性和方法才对外公开。

下面再举一个例子来说明封装的概念。一家名为法拉利的公司是一家汽车备件制造商，该公司以其产品质量优良而享有盛名；另一家公司迪斯尼是一家汽车制造公司，迪斯尼公司需要为其新型汽车购买一些品质优良的变速箱和离合器。迪斯尼公司的采购经理罗杰斯先生与法拉利公司的市场经理雪莉女士进行接洽，罗杰斯先生向她提供了所需变速箱和离合器的规格，如图 3-8 所示。

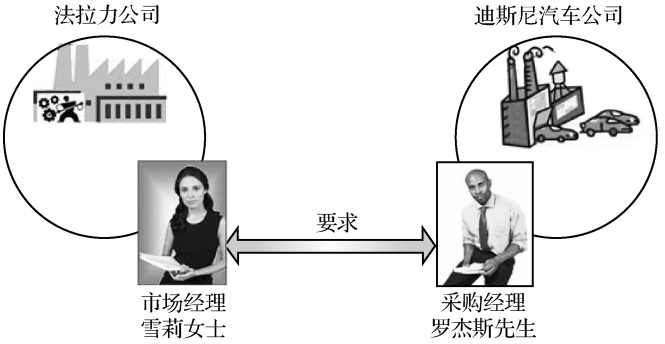

图 3-8　封装举例

雪莉女士收到上述请求，就将订单信息传递给法拉利公司的相关部门，然后公司将这些零件交付给迪斯尼公司。如果法拉利公司库房中有所需数量的零件，该过程将十分简单；否则，将需要制造这些零件。无论属于哪种情况，罗杰斯先生不需要知道有关过程以及如何制造这些零件等信息。同样，雪莉女士也不需要知道迪斯尼公司将如何使用这些变速箱和离合器来组装汽车。因此，除了提供给对方的必须信息，双方的其余信息均未披露。图 3-9 所示显示了双方的公开信息和隐藏的私有信息。

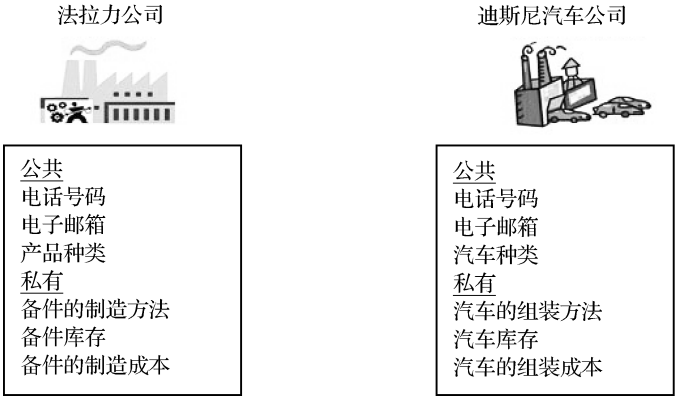

图 3-9　双方的公共信息和私有信息

法拉利公司的工作方式及其存储和使用的数据可一起装入一个名为法拉利的实体中。除了法拉利公司为便于交流而提供的公共信息外，迪斯尼公司不需要知道法拉利公司所有

的工作过程或属于法拉利公司的私有信息。这种处理方式就称为封装。

2. 封装的实现

封装就是将属性私有化，提供公有方法访问私有属性。做法是：修改属性的可见性来限制对属性的访问，并为每个属性创建一对取值(getter)和赋值(setter)方法，用以访问这些属性。下面首先了解 Java 中数据的可见性。

Java 语言具有隐藏类中复杂的实现细节的机制，这是通过为每个成员指定访问修饰符来实现的。访问修饰符可以确定如何访问某个成员。访问控制在某些方面与继承或包密切相关。前面模块中的程序已经使用过这些访问修饰符，此处提供它们的含义。

1)public

类的 public(公有)成员可以被该类的成员和非该类的成员访问。例如，如果 User 类具有一个名为 username 的公有成员，则该成员可以被 User 类的所有成员方法和用户自定义的其他类的成员方法访问。

2)private

类的 private(私有)成员只能被该类的成员访问。任何非该类成员的代码不能访问任何私有方法或变量。使用 private 访问修饰符的主要目的是隐藏数据。

3)protected

类的 protected(保护)成员可以被该类的成员以及其子类的成员访问，还可以被同一个包内其他类的成员访问。

还有一个缺省(default)情况，即类的成员没有修饰符，此时只有它本身的类和在同一个包的类里可以访问它。

访问修饰符及其缺省情况对类成员访问的限制总结如表 3-2 所示。

表 3-2　修饰符的可访问性

位　　置	private	default	protected	public
同一个类	可访问	可访问	可访问	可访问
同一个包内的类	不可访问	可访问	可访问	可访问
不同包内的子类	不可访问	不可访问	可访问	可访问
不同包并且不是子类	不可访问	不可访问	不可访问	可访问

Java 中类的属性的具体封装步骤如下。

(1)修改属性的可见性来限制对属性的访问。

(2)为每一个属性创建一对赋值和取值方法，用于对这些属性进行访问。

(3)在赋值方法和取值方法中，加入对属性的存取限制。

【例 3-4】 定义 Student 类，实现封装。

```
class Student
{
    private String name;          //姓名
    private int age;              //年龄
    public String getName(){
```

```
        return name;
    }
    public void setName(String MyName){
        name = MyName;
    }
    public int getAge(){
        return age;
    }
    public void setAge(int age){
        if(age<6){
            System.out.println("错误！最小年龄应为 6 岁!");
            this.age = 6;   //如果不符合年龄要求,则赋予默认值
        }else{
            this.age = age;
        }
    }
    /**
     * 返回自我介绍的内容
     */
    public String introduction(){
        return "大家好！我是" + name + ",我今年" + age + "岁";
    }
}
public class TestStudent{
    public static void main(String args[]){
        Student s = new Student();
        s.setName("Jake");
        s.setAge(20);
        System.out.println(s.introduction());
    }
}
```

程序编译、运行后的结果如图 3-10 所示。

3. 封装的优势

封装的优势主要有以下几个。

(1)隐藏实现细节。这样做可以防止程序员依赖这些细节。也就是说,程序员无须担心对任何实现细节所做的更改,因为这些更改不会影响使用该类的其他代码。

(2)实现对属性的访问限制,增强程序的可维护性。

(3)防止用户意外地删除数据。例如,类可以包含许多相互依赖的字段,如果程序员对其中任何一个字段做了更改,而未更改相关的字段,则可能会产生错误。然而,如果用方法

来更改字段，则可以确保更改所有相关字段的值，使类的状态保持一致。类还可以隐藏它的一些方法，以防止用户调用这些方法。

(4)隐藏类的详细信息，使类更容易使用和理解。

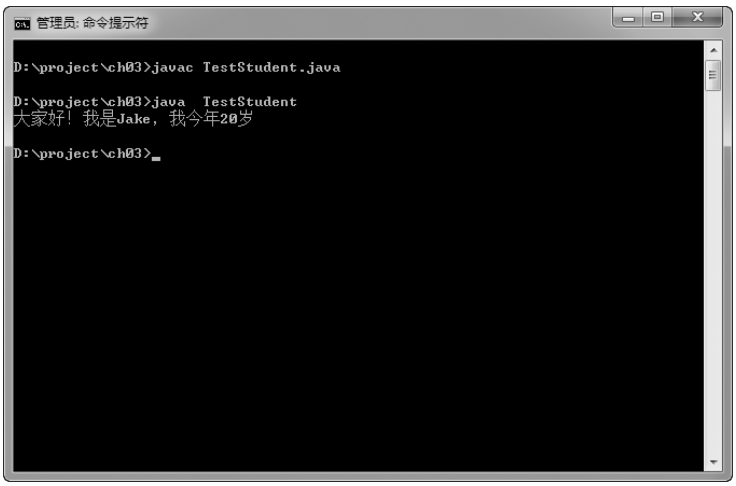

图 3-10　类封装程序运行结果

3.3 案 例 实 施

编写学生类，并输出学生相关信息，操作步骤如下。

3.3.1 编写源代码

打开记事本(或其他文本编辑器)，输入以下 Java 源代码。

【程序 3-1】 编写 StudentTest.java 文件。

程序代码如下：

```
/**
 * 学生类
 */
class Student{
    String name;                    // 姓名
    int age;                        // 年龄
    String hobby;                   // 兴趣
    String classNo;                 // 班级编号
    /**
     * 自我介绍方法
     */
        public void introduction(){
```

```
                System.out.println("大家好！ 我是" + name + ",我今年" + age + "岁,兴
趣是" + hobby + ",我的班级编号是" + classNo + "!");
        }
    }
    /**
     * 学生测试类
     */
    public class StudentTest{
        /**
         * 测试
         * @param args 传递至 main()方法的参数
         */
        public static void main(String[] args)
        {
            /** 创建一个学生对象 */
            Student stu = new Student();
            stu.name = "赵强";
            stu.age = 20;
            stu.hobby = "踢足球";
            stu.classNo = "soft101";
            stu.introduction();
        }
    }
```

编辑完成后保存为“StudentTest.java”。

3.3.2 编译源代码

打开命令提示符窗口,将命令提示符的当前路径切换到 Java 源程序文件所在的目录,然后输入以下命令:

```
javac StudentTest.java
```

若源代码没有错误,编译成功后 Java 源程序文件所在的目录下将会生成一个字节码文件 StudentTest.class。

3.3.3 运行程序

使用 JDK 的解释器 java.exe 就可以对编译后得到的字节码文件进行解释执行了。在命令提示符后输入下面的命令:

```
java StudentTest
```

程序运行结果如图 3-11 所示。

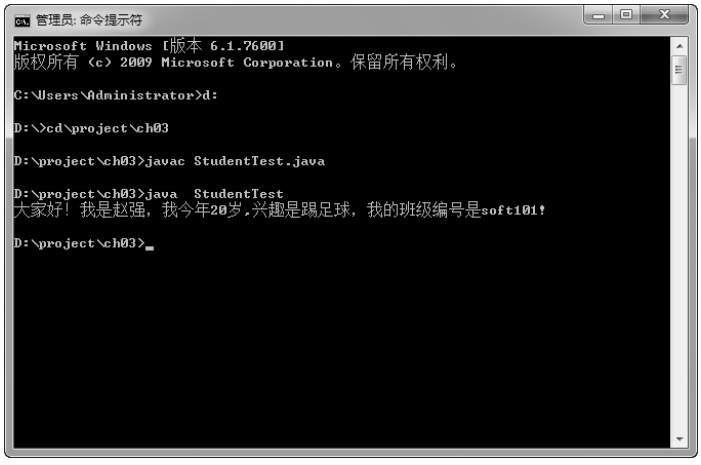

图 3-11　学生类程序运行结果

3.4 训练与实战

在上面的案例实施过程中，我们用 Java 程序描述了学生的基本信息，学会了类、对象、属性和方法的使用。下面通过两个具体的实战训练项目来进一步巩固 Java 中类和对象的应用，强化构造方法、封装和 this 关键字的应用。

3.4.1 计算两点之间的距离

平面直角坐标系中点的位置是通过横坐标、纵坐标来描述的，那么任意两点可以用 $p_1(x_1,y_1)$、$p_2(x_2,y_2)$来表示，两点之间的距离可以通过数学公式 $\sqrt{(y_2-y_1)^2+(x_2-x_1)^2}$ 来计算。其中，公式中涉及算术平方根和指数运算，因此需要使用 java.lang 包中 Math 类，该类提供了一些基本数学运算和几何运算的方法。该类中的所有方法都是静态的。Math 类的部分方法如表 3-3 所示。

表 3-3　Math 类的部分方法

方　法	说　明
double sin(double numvalue)	计算角 numvalue 的正弦值
double cos(double numvalue)	计算角 numvalue 的余弦值
double pow(double a,double b)	计算 a 的 b 次方
double sqrt(double numvalue)	计算给定值的平方根
int abs(int numvalue)	计算 int 类型值 numvalue 的绝对值，也接收 long、float 和 double 类型的参数
double ceil(double numvalue)	返回大于等于 numvalue 的最小整数值

续表

方　法	说　明
double floor(double numvalue)	返回小于等于 numvalue 的最大整数值
int max(int a,int b)	返回 int 类型值 a 和 b 中的较大值，也接收 long、float 和 double 类型的参数
int min(int a,int b)	返回 a 和 b 中的较小值，也可接收 long、float 和 double 类型的参数

训练内容

编写 Java 程序，计算平面直角坐标系中任意两点之间的距离。

训练过程

1)编写源代码

【程序 3-2】 编写 TestPoint.java 文件。

程序代码如下：

```
public class TestPoint{
    double dx;
    double dy;
    TestPoint(double x,double y){
        this.dx = x;
        this.dy = y;
    }
    public double getX()
    {
        return dx;
    }
    public double getY()
    {
        return dy;
    }
    public double distance(TestPoint p){
        return Math.sqrt(Math.pow((dx - p.dx),2) + Math.pow((dy - p.dy),2));
    }
    public static void main(String[] args){
        TestPoint tp = new TestPoint(2.3d,6.7d);
        TestPoint tp1 = new TestPoint(10.3d,16.7d);
        System.out.println("两点之间的距离:" + tp.distance(tp1));
    }
}
```

2)编译和运行程序

在命令提示符窗口下将路径切换到文件 TestPoint.java 所属目录,然后输入:

```
javac TestPoint.java
```

按 Enter 键编译完成后,输入以下的命令:

```
java TestPoint
```

程序运行结果如图 3-12 所示。

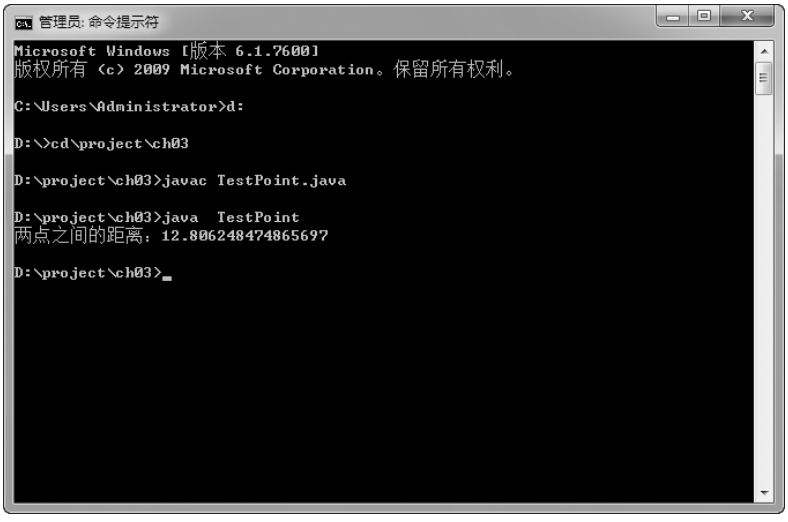

图 3-12　计算两点之间的距离程序运行结果

总结与体会

本训练使用了带参数的构造方法,它在定义一个对象的同时也把对象的属性进行了初始化,简化了对象初始化代码。注意,在使用带参数的构造方法实例化对象时,传递的值和构造方法的参数应当在个数、次序和数据类型上应互相匹配。

3.4.2　复数的加法运算

数学中的复数分为实部和虚部两部分,在进行复数运算时要考虑实部和虚部的运算,输出复数时同样要考虑复数的组成。

训练内容

编写程序,对任意两个复数进行加法运算,并输出运行结果。

训练过程

1)编写源代码

启动编辑器编写源代码程序,保存为 Complex.java 文件。

【程序 3-3】　编写 Complex.java 文件。

```
public class Complex{
    double real,img;          //实部和虚部
```

```
public Complex()          //默认构造方法
{
    this.real = 0;
    this.img = 0;
}
public Complex(double real,double img)     //带参数的构造方法
{
    this.real = real;
    this.img = img;
}
public double getReal(){ return this.real; }          //得到实部
public double getImage(){ return this.img; }          //得到虚部
public void setReal(double real){ this.real = real; }   //设置实部
public void setImage(double img){ this.img = img; }   //设置虚部
public Complex addComplex(Complex a,Complex b)  //两个复数相加,结果返回
{
    Complex temp = new Complex();
    temp.real = a.real + b.real;
    temp.img = a.img + b.img;
    return temp;
}
public String toString()  //将复数转化为"a + bi"形式的字符串
{
    String s;
    if (real! = 0.0){
        if(img>0)
            s = new Float(real).toString() + "+" + new Float(img).toString
                () + "i";
        else if(img<0)
            s = new Float(real).toString() + "-" +
                new Float( - 1 * img).toString() + "i";
        else
            s = new Float(real).toString();
    }
    else{
        if(img>0)
            s = new Float(img).toString() + "i";
        else if(img<0)
            s = new Float( - 1 * img).toString() + "i";
        else
            s = new Float(real).toString();
```

```
        }
        return s;
    }
    public void printComplex()          //输出复数
    {
        System.out.println("" + this.real + "+" + this.img + "i");
                                        //可以添加代码以判断虚部的正负
    }
    public static void main(String[] args)          //测试代码
    {
        Complex cc = new Complex(4,8);
        cc.printComplex();
        Complex dd = new Complex(2,2);
        dd.printComplex();
        System.out.println("--------------");
        cc.addComplex(cc,dd).printComplex();
        System.out.println("--------------");
        System.out.println(cc.addComplex(cc,dd).toString());
    }
}
```

2)编译和运行程序

在命令提示符窗口下将路径切换到文件 Complex.java 所属目录,然后输入:

```
javac Complex.java
```

编译完成后输入下面的命令:

```
java Complex
```

程序运行结果如图 3-13 所示。

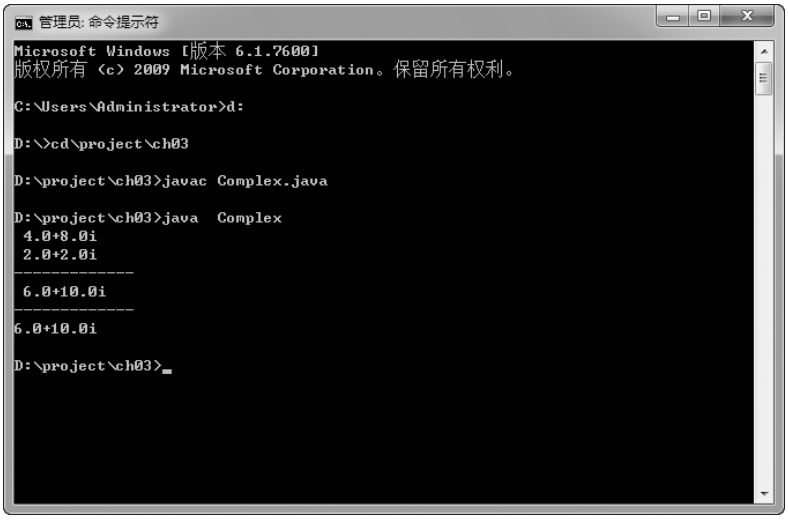

图 3-13　复数相加程序运行结果

总结与体会

面向对象编程解决问题非常灵活，只有多实战训练，才能够真正掌握解决问题的办法。

3.5 实际工作中常见问题解析

3.5.1 Java 文件命名的问题

1. 问题

Java 源程序文件名与程序中的 public 修饰的类名不同时，编译出错。

编辑下面 Java 源代码，保存文件名为 Student.java。

```
/**
 * 学生类
 */
class Student{
    String name;                    // 姓名
    int age;                        // 年龄
    String hobby;                   // 兴趣
    String classNo;                 // 班级编号
    public void introduction(){
    System.out.println("大家好！ 我是" + name + "，我今年" + age + "岁，兴趣是" +
hobby + "，我的班级编号是" + classNo + "！");
    }
}
/**
 * 学生测试类
 */
public class StudentTest{
    public static void main(String[] args){
        /** 创建一个学生对象 */
        Student stu = new Student();
        stu.name = "赵强";
        stu.age = 20;
        stu.hobby = "踢足球";
        stu.classNo = "soft101";
        stu.introduction();
    }
}
```

按照正常的操作步骤编译，系统会报错，如图 3-14 所示。

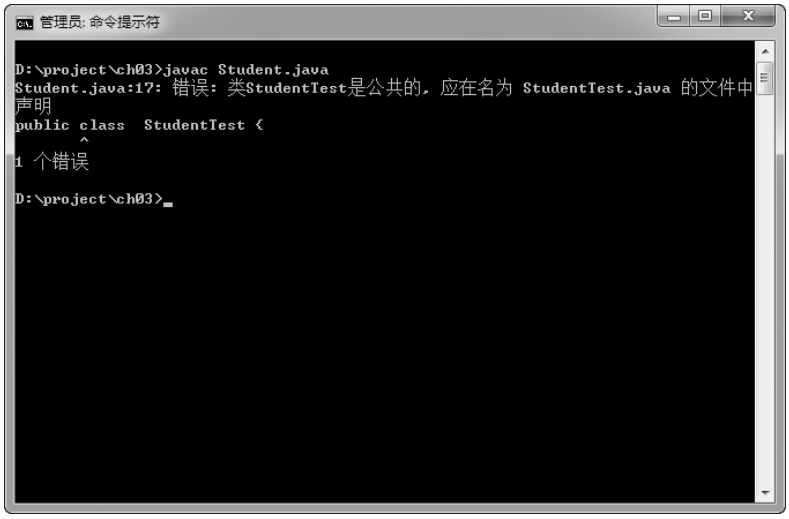

图 3-14　运行 Student. java 文件出现的出错信息

2. 问题分析与解决

问题出在文件名和文件中 main()方法所在的 public 类名不一致。main()方法是一个程序的入口，要保证在一个程序(可能有多个 java 文件组成)中只有一个 main()方法，运行时应运行含有 main()方法的文件。

3. 解决办法

文件名改为 StudentTest. java 即可。另外，Java 程序可以将学生类 Student 和测试类 StudentTest 放在不同的文件中。

3.5.2　默认构造方法问题

1. 问题

一个类中定义了构造方法，欲使用该类的默认构造方法实例化对象，但没有成功。

2. 问题分析与解决

出错原因是如果程序员定义了一个或多个构造方法，将自动屏蔽默认的构造方法。

3. 解决办法

显式定义无参数构造方法。注意：使用类的默认构造方法实例化对象所生成的对象属性值为空或零。可以在显式定义的无参数构造方法中为属性设置定值，从而完成初始化工作。

3.6 习　　题

一、选择题

1.（　　）和（　　）是拥有属性和方法的实体。

A. 对象　　B. 类　　C. 方法　　D. 类的实例

2. 对象的特征在类中表现为变量，称为类的（　　）。

A. 对象　　B. 属性　　C. 方法　　D. 数据类型

3. 小慧定义了一个汽车类（Car），包含的属性有：颜色（color）、型号（type）、品牌（brand）。现在小强要在 main()方法中创建 Car 类的对象，在他的编码中，（　　）是正确的。

A.
```
Car myCar = new Car();
myCar color = "黄色";
```

B.
```
Car myCar = new Car();
myCar.brand  = "宝马";
```

C.
```
Car myCar;
myCar.color = "黄色";
```

D.
```
Car myCar = new Car();
color = "黄色";
```

4. 阅读下列代码，（　　）是正确的。

A.
```
public class Dog{
        String color = "黑色";
        String type = "德国牧羊犬";
        public void toString(){
            return "这是一只" + color + "的" + type;
        }
    }
```

B.
```
public class Dog{
        String color = "黑色";
        String type = "德国牧羊犬";
        public String toString(){
            return "这是一只" + color + "的" + type;
        }
    }
```

C.
```
public class Dog{
        String color = "黑色";
        String type = "德国牧羊犬";
        public toString(){
            return "这是一只" + color + "的" + type;
```

```
        }
    }
```

D.
```
public class Dog{
    String color = "黑色";
    String type = "德国牧羊犬";
    public String toString(){
        "这是一只" + color + "的" + type;
    }
}
```

5. 在类的修饰符中，只允许该类自身访问的是(　　)。

A. private　　B. public　　C. default　　D. protected

6. 在成员方法的访问控制修饰符中，规定访问权限包含该类自身、同包的其他类和其他包的该类子类的修饰符是(　　)。

A. public　　B. private　　C. default　　D. protected

7. 下列关于构造方法的特点的描述中，错误的是(　　)。

A. 不可重写　　B. 方法同名类

C. 无返回类型　　D. 系统自动调整

8. 下列关于静态方法的描述中，错误的是(　　)。

A. 在类体内说明静态方法使用关键字 static

B. 静态方法只能处理静态变量和静态方法

C. 静态方法不占用对象的内存空间，非静态方法占用对象的内存空间

D. 静态方法只能用类名调用

9. 下列对封装的描述中，错误的是(　　)。

A. 封装体包含了属性和行为

B. 封装体中的访问权限是相同的

C. 被封装的某些信息在封装体外是不可见的

D. 封装使得抽象的数据类型提高了可重用性

二、编程题

用代码实现计算器类(Calculator)。

提示：

(1)定义成员变量运算数 1(number1)和运算数 2(number2)。

(2)定义成员方法“加”(add)、“减”(subtract)、“乘”(multiple)和“除”(divide)。

(3)定义测试类 Test，对任意两数进行加、减、乘、除运算。