第1章　　java 语言基础

本章内容(一般掌握 3%~5%)

本章重点：应用程序和小应用程序的基础知识，程序的开发过程，基本数据类型。

本章难点：程序的开发过程，了解一种 Java 程序的开发环境。

1.1 Java 语言特点

Java 语言最初的应用对象是消费性电子产品(即 PDA 、电子游戏机、电视机顶盒之类的产品)。为了进入消费性电子产品市场，SUN 公司专门成立了一个项目开发小组，目标是设计嵌在消费性电子产品的小型分布式系统软件，能够适用于异构网络、多主机体系结构，能实现信息安全传递。项目小组的最初设想是用 C++语言完成这个目标。由于 C++语言的复杂性和不安全性，不能胜任这项工作。为此，项目小组开发一个取名为 Oak 的语言。

Oak 语言在消费性电子产品市场上没有获得青睐。但在当时，Internet 开始流行，人们发明了一种网络传输协议，这种协议可以在文本中插入图片和声音，能使单调的 Internet 世界变得图文并茂。虽然 Web 页面拥用图文和声音，但仍然是静态的，不具备交互性。要让页面拥有动态画面，并能交互，需要在 Web 页面中嵌入一会段程序。由于在 Internet 上运行的数以千计不同类的计算机，这就要求编写这种程序的语言必须具有平台无关性，并要求语言必须简练，支撑环境要小，而安全性却很高。Oak 语言恰好能够满足这些要求。

将 Oak 语言正式应用于 Internet 还需要进一步完善，还要求有一个支持它的 Web 浏览器。能用于编写浏览器，并获得浏览器扶持的新版本 Oak 语言取名 Java。Internet 得到 Java 语言的支持，可以实现真正的交互，人们使用浏览器能“漫游”丰富多彩的 Internet 世界。

Java 语言是适用于分布式计算环境的面向对象编程语言，它虽类似 C 和 C++，但比 C++简单，忽略了许多为提高计算效率，初学者较难掌握的程序语言特性。

Java 语言主要有以下**特点**：

1. 强类型

Java 语言是一种强类型语言，强类型能约束程序员必须遵守更多的编程规定，也能让编译器检测出程序中尽可能多的错误。

2. 编译和解释

Java 语言是一种高级编程语言，用 Java 语言编写的源程序在计算机上运行需经过编译和解释执行两个严格区分的阶段。Java 语言的编译程序先将 Java 源程序翻译成机器无关的字节码(bytecode)，不是通常的编译程序将源程序翻译成计算机的机器代码。运行时，Java 的运行系统和链接需要执行的类，并作必要的优化后，解释执行字节码程序。

3. 自动无用内存回收功能

Java 语言具有自动无用内存回收功能，程序可以按需使用内存，但不需要对无用内存显式地撤销分配。系统有一个垃圾收集器(garbage collector)，自动收集程序不再使用的内存。这样，能避免显式的撤销分配所引起的安全问题。Java 语言不再含有任何不安全的语言成分。例如，没有指针，数组元素都要检查下标是否越界。

4. 面向对象

面向对象是程序员编写大型程序、有效控制程序复杂性的重要手段。Java 语言在面向对象方面，比 C++更“纯”，它的所有数据类型，包括布尔类型、整形、字符型等，都有相应的类，程序可完全基于对象编写。

面向对象语言主要有**封装性、继承性和多态性**三个特点。封装就是将实现细节隐藏起来，只给出如何使用的信息。数据及数据上的操作用类封装，对象是类的实例，外界使用对象中的数据及可用的操作受到一定的限制。继承体现众多的一种层次对象的特性，下一层的类可从上一层的类继承定义，从上一层类派生的类的对象能继承上一层对象的特性，同时可以改变和扩充一些特性，以适应其自身的特点。多态性的意义主要体现在逻辑上相同的不同层次上的操作，使用相同的操作名，根据具体对象，能自动选择对应的操作。Java 语言很实用地实现了这三种特性。

5. 与平台无关

与平台无关是对程序**可移植性最直接最有效的支持**。Java 语言的设计者在设计时重点考虑了 Java 程序的可移植性，采用多种机制来保证可移植性，其中最主要的是定义了一种虚拟机(virtual machine),以及虚拟机使用的 java 字节码。在任何平台上,Java 源程序被 Java 编译器编译成虚拟机能够识别的字节码。这样，只要有 Java 虚拟机的平台，就能解释执行 java 字节码程序，从而实现 Java 与平台无关。另外，Java 语言还采用基于国际标准的数据类

型，在任何平台上，同上种数据类型是一致的。例如，用 int 标识 32 位二进制位(bit)整型数据，那么无论在哪一台计算机上，Java 的 int 数据都是 32 位整数。相反，C 语言会随着硬软件平台的改变，用 int 标识的整数位数也可能不全相同。

Java 语言提高可移植性的代价是降低程序的执行效率。出于 java 语言也是一种解释执行的语言，Java 程序的执行速度与 C 程序的执行速度有较大的差别。不过，为了尽量弥补执行效率低的缺陷，java 的字节码在设计上非常接近现代计算机的机器码，这有助于提高解释执行的速度。

6. 安全性

Java 是在网络环境中使用的编程语言，必须考虑安全性问题，主要有以下两个方面：

设计的安全防范：Java 语言没有指针，避免程序因为指针使用不当，访问不应该访问的内存空间；提供数组元素上标检测机制，禁止程序越界访问内存；提供内存自动回收机制，避免程序遗漏或重复释放内存。

运行安全检查：为了防止字节码程序可能被非法改动，解释执行前，先对字节码程序作检查，防止网络“黑客”对字节码程序已作了恶意改动，达到破坏系统的目的。最后，浏览器限制下载的小应用程序不允许访问本地文件，避免小应用程序破坏本地文件。

7. 分布式计算

Java 语言支持客户机/服务器计算模式。Java 程序能利用 URL 对象，能访问网络上的对象，如同访问本地的文件一样，实现数据分布。另外，Java 的客户机/服务器模式也可以把计算从服务器分散到客户机端，实现操作分布。

8. 多线程

线程是比进程更小的一种可并发执行的单位，每个进程都有自己独立的内存空间和其他资源，当进程切换时需要进行数据和资源的保护与恢复。若干协同工作的线程可以共享内存空间和资源，线程切换不需要数据的保护与恢复。

Java 的运行环境采用多线程实现，可以利用系统的空闲时间执行诸如内存回收等操作；Java 语言提供语言级多线程支持，用 Java 语言能直接编写多线程程序。

1.2 Java 应用程序和小应用程序

Java 程序可以是独立的应用程序和能在浏览器上执行的小应用程序(Applet)。两种 Java 程序都由一个或多个扩展名为“.class”的文件组成。都需要 java 虚拟机(JVM)载入并翻译。这两种程序的主要区别是：小应用程序只能在与 Java 兼容的容器中运行，可以嵌入在 HTML 网页内，在网络上发布，当网页被浏览时，在浏览器中运行。小应程序的运行还要受到严格的安全限制，例如，它不能访问用计算机上的文件。Java 应用程序没有这些限制，也不支持网页嵌入和下载运行。

小应用程序和应用程序在代码编写上也有很大差异。一个小应用程序必须定义成一个 Applet 类的子类，应用程序可以是 Applet 类的子类，也可以不是。应用程序必须在一个类中定义一个 main()方法，该方法代表应用程序的入口。而小应用程序不必定义 main()方法，它的执行由 Applet 类中定义的多个方法控制。

[例 1.1] 一个非常简单的应用程序。

```
public class Example1_1{//这是我的第一个应用程序
    public static void main(String []args){
        System.Out.println(“你好!欢迎你学习 Java 语言。”);
    }
}
```

上述 java 程序的执行将输出以下字样：

```
你好!欢迎你学习 Java 语言。
```

一个应用程序由若干个类组成，上面这个应用程序只有一个类，类的名字是 Example1_1。public 是 java 语言的关键字，表示声明的类 Example1_1 是公用的。class 也是关键字，用来声明类。最外层的一对花括号以及括号内的内容叫做类体。public static void main(String []args)是类 Example1_1 的一个方法。一个应用程序必须只有一个类含有 main()方法，这个类是应用程序的主类。public static void 是对 main()方法的说明。应用程序的 main()方法必须被说明成 public static void。表示 main()方法的访问权限是公有的，它是一个类方法，没有结果返回，参见第 3 章。String []args 或 String args[]，声明 main()方法的参数是一个字符串数组，参见第 4 章。

Java 源程序命名受严格的限制。Java 源文件的扩展名必须是“.java”如果源文件中有多个类，那么只能有一个 public 类；如果源文件中有 public 类，那么源文件的名字必须与这个类的名字完全相同。例如，例 1.1 应用程序的源文件名必须是 Example1_1.java。如果源文件没有 public 类，那么源文件的名字只要和某个类的名字相同即可。

[例 1.2]一个简单的小应用程序，用一同颜色显示两行文字：

欢迎你学习 Java 语言。

只要认真学习，多上机实习，一定能学好 Java 语言。

```
import java.applet.*;
import java.awt.*;
public class Example1_2 extends Applet{
    public void paint(Graphics g){
    g.setColor(Color.blue);//设置显示的颜色为 blue
    g.drawString("欢迎你学 Java 语言",30,20);
    g.setColor(Color.red);//设置显示的颜色为 red
    g.drawString("只要认真学习，多上机实习，一定能学好 Java 语言。",30,50);
    }
}
```

一个小应用程序也出若干个类组成,其中必须有一个类,它继承系统提供的 Applet 类,这个类是小应用程序的主类。主类必须是 public 的，源文件名必须与小应用程序的主类名相同。上述程序的源文件名必须是 Example1_2.java。小应用程序的结构参见 3.5 节。

在结束这一节之前，先讨论一下程序中经常出现的注释。注释是程序中的说明文字，用于帮助阅读程序，它不是语句，不会影响程序的执行效率。Java 语言的注释有三种形式：

行注释 // 变量 r 是贺的半径

块注释 /* 以下程序段是采用冒泡排序对数组元素实现从小到大排序 */

文档注释 /* *类 Student 是学生类*/

行注释用于简短地说明前面代码的意义或功能。注释内容至行末结束，不能跨行。

例如：

```
int fontsize =30 ;//标题字体大小
```

块注释是多行注释，用于说明下面程序段的算法、功能等，块注释不可嵌套使用。

文档注释能被实用程序 javadoc 接受，它能自动生成程序的 HTML 文档。

1.3 Java 程序的开发过程

Java 程序的开发过程如图 1.1 所示。对于 Java 应用程序，**先编写 Java 源程序**，源程序**经 Java 编译器编译后产生码文件**，最后由 Java 解释器**解释执行字节码**文件。对于小应用程序，**先编写源程序**，然后经 Java 编译器**编译后，产生字节码文件**，最后由 **Web 浏览器解释执行字节码文件**。

以下以使用 SUN 公司提供的 Java JDK(java Developer’s Kit)为例，并假设 Java JDK 安装在 d:\java 目录下，则在该目录下还应包含以下几个文件和子目录：

(1) 子目录 d:\java\bin,这个子目录是 Java JDK 的核心，其中最主要的可执行文件有：

1) javac---编译器
2) java---解释器
3) jdb---调试器
4) appletviewer---java Applet 解释器

(2) 子目录 d:\java\lib，这个子目录存储的是函数库。

编写源程序时，使用一个文字编辑器，输入源程序文件，并保存。键入编译命令，使用 Java 编译器编译 Java 源程序。键入解释命令，通过解释器解释执行 Java 应用程序的字节码文件。对于小应用程序需通过支持 Java 的浏览器解释执行字节码文件。

设输入前面所述 Java 程序，源文件名为 Example1_1.java，保存在 D：\java 目录下。

Java 编译器的使用格式是：

javac [选项] 源文件

例如，可用以下命令编译以上输入的 Java 应用程序：

```
javac d:\java\Example1_1.java
```

编译时，可能编译器指定选项，java JDK 编译器的选项参见表 1-1。

选项	说明
-classpath<路径>	引用类的路径表
-d<目录>	编译后类文件存放的目录
-g	生成调试信息表
-ng	不生成调试信息表
-nowarn	关闭编译器警告功能
-o	优化类文件
-verbose	显示编译过程中的详细信息

Java 解释器 java.exe 用来解释执行 Java 类文件，解释器的使用格式为：

java [选项]类名[参数]

其中，类名指定的类必须是 main()方法的，即是一个 Java 应用程序。

对于使用开发环境情况来说，编辑源程序、编译、执行和调试直接使用环境提供的工具，使开发程序变得非常简单。例如，使用 Eclipse 开发环境。在 eclipse 环境下开发 Java 程序的方法请参见附录 F。

1．4 标识符和关键字

如同别的程序语言一样，Java 语言也有基本符号，由基本符号按一定的构词规则构成标识符等基本词汇，再由基本词汇和关键字按语言的句法构成 Java 程序。

1． 字符集

Java 语言使用 Unicode 字符集，共有 65535 个字符，包括：

(1) 数字符 10 个(0~9)。

(2) 英文字母大、小写字符各 26 个(A~Z 和 a~z)。

(3) 下划线字符，美元符号。

(4) 文字字符，(汉字，日文片假名、平假名和朝鲜文字等)。

(5) 其他用于构成特殊符号的字符集。

2． 标识符

标识符用于命名程序对象。例如，类名、变量名、方法名等。Java 语言的标识符可以按以下规则任意命名：

(1) 由字母(包括英文字母、下划线字符、美元字符、文字字符)和数字字符组成。

(2) 限定标识符的第一个字符不能是数字符。

例如，下列 4 个字符串都可以个为标识符：

Boy_$、$63Girl、颜色、小伙子

而以下 4 个字符不能作为标识符：

46A、%Val、c+2、Hello!

前两是因为首字符不合理，后两个含有不允许在标识符中出现的字符。

除了按上述规则命名的习惯约定：

一般变量名、方法名用小写英文字母开头；文字和常数用大写字母命名；类名首字母用大写字母；包名全部用小写字母。命名习惯是一种公共约定，Java 语言的语法并没有这种限制，但是培养良好的编程习惯有利于程序维护和协作开发。

3． 关键字

为了表示程序结构、定义和修饰等，Java 语言引入一组关键字，参见附录 A。关键字是上些英文单词，在语言中，关键字已经被赋予特定的意义，不能用这些关键字作为标识符命名程序对象。在 Java 语言中，关键字的作用有以下多个方面：

描述程序结构、声明类、定义类的成员、简单数据类型、值和变量、异常情况处理、实例(对象)的创建和检验、流程控制等。

Java 语言中所有关键字均由小写字母组成。每个关键字在程序中出现有特定位置和使用方法的要求。关键字不可以用做变量、方法、类或标号的标识符(名字)。

1.5 基本数据类型

Java 语言是强类型语言，每个变量和表达式都有确定的类型，在变量赋值时要进行**类型兼容性检验**。数据类型可分

为基本数据类型和非基本数据类型两大类。

基本数据类型也称为原始数据类型，是系统预先规定的一些常用类型。它们是：整数类型、浮点数(实数)类型、字符(文字)类型、逻辑类型(布尔型)。

非基本数据类型也称为复合数据类型，是由基本类型组合的新类型。**非基本数据类型**分为：数组、字符串、类、接口。

本小节只介绍基本数据类型，非基本数据类型将在以后章节中分别介绍。

1.5.1 逻辑类型

逻辑类型用关键字 boolean 标识，所以也称布尔型。逻辑类型只有真和假两个值，true 表示真，false 表示假。以下是逻辑变量定义的例子：

```
boolean b; boolean bool;
boolean flg1,flg2, 美丽; //一次定义多个变量
boolean b1 = true,b2= false,丑=false;//定义时可以赋初值
```

逻辑变量用于记录某种条件成立与否，也用语句中作条件判断。

1.5.2 字符类型

字符类型用关键字 char 标识。字符型数据是一个字符，内部表示是字符的 Unicode 代码，共用 65535 个不同的字符。在 Java 程序中，**字符常量**有**普通字符**和**转义字符**常两种。

用单引号括住一个字符，表示一个普通字符常量。例如，’a’、’B’、’$’、’国’。

对于被语言用作特定意义的字符，或者不能显式显示的字符，需用转义字符标记它们。

例如，换行符用\n 标记，水平制表符用\t 标记。常用的转义字符的标记方法见表 1-2。其中，\nnn 是用八进制代码表示的字符，\unnnn 是用十六进制代码表示的字符。例如：\141 表示字母 a，\u0061 也表示字母 a.以下是字符变量定义的例子：

```
char ch; char ch1,ch2,漂亮;
char ch3 = ‘A’,ch4 = ‘家’,丑=’假’;//定义时可以赋给初值
```

表 1-2 转义字符及其含义

转义符	含义
\b	退格(Backspace 键)
\n	换行符，光标位置移到下一行首
\r	回车符，光标位置移到当前行首
\t	水平制表符(Tab 键)
\v	竖向退格符
\f	走纸换页
\\	反斜杠符\
\’	单引号符’
\”	双引号符”
\nnn	n 为 8 进制数字，用八进制数据表示字符的代码
\unnnn	n 为 16 进制数字，用 16 进制数据表示字符的代码

1.5.3 整数类型

整数是不带小数点和指数的数值数据。由于计算机只能表示整数的一个子集，表达更大范围内的整数需要更多的二进制位，Java 语言将整型数据按数值范围大水不同分成四种：

(1) 基本型：用 int 标识。4 个字节，32 位，取值范围是-2^{31}~$2^{31}-1$，即-2147483648~2147483647。

(2) 字节型：用 byte 标识。1 个字节，8 位，取值范围是-2^{7}~$2^{7}-1$，即-128~127。

(3) 短整型：用 short 标识。2 个字节，16 位，取值范围是-2^{16}~$2^{16}-1$，即-23768~32767

(4) 长整型：用 long 标识。8 个字节，64 位，取值范围是-2^{64}~$2^{64}-1$，即-9233720368547758O8L~9233720368547758O7L

每种整数类型处理不同范围的整数值，并且都是带符号的。

整型常量有十进制、八进制和十六进制三种写法。如下 面的示例所示：

1234(十进制)，0777(八进制，以数字 0 开头)，0x3ABC(十六进制，以 0x 开头，后随数字符和英文 A 到 F)，5333L 或 53331(以字母 L 或 l 结尾是长整数)。

以下是整型变量定义的实例：

```
int I; int x,积;byte 字节变量;//一个字节整型变量，该变量的名为字节变量
```

long z,sum;int w =12,len =-1230;long big = 9876L;//定义同时可以赋初值。

1.5.4 浮点数类型

浮点数类型也称实型，浮点数是带小数点或指数的数值数据。 Java 语言的浮点数有单精度和双精度两种。

1. 单精度型

单精度型 用 float 标识，占 4 个字节，32 位，取值-10^{38}~10^{38}。float 型常是的书写的方法是在实数之后加上字母 **F 或 f.** 例如：23.54f,12389.987F。

2. 双精度型

双精度型 用 double 标识，占 8 个字节，64 位，取值-10^{308}~10^{308}。double 型常是的**书写的方法有两种：一种是直接写一个实数，或在实数后面加上字母 D 或 d.** 例如：123.5439、123.5439D, 123.5439d。另一种是科学计数法，用 10 的方幂表示(用字符 e 或 E 表示幂底 10)。例如：123.24e40(科学计数法表示，值为 123.24 乘 10 的 40 次方)。

以下是浮点数类型变量定义的例子:

```
float x,y;
double v=12.86,u=2431098.987D;
float u=12.36f;
```

浮点数类型与整数不同，当执行运算时如果出现某种不正常的状态，浮点数类型不会抛出异常。例如，如果用 0 除浮点数类型的变量，则计算结果是一个特别的无限值。

习题

1. 指出 Java 语言的主要特点和 Java 程序的执行过程。

答：主要特点：

(1)强类型，(2) **编译和解释，(3) 自动无用内存回收功能，(4) 面向对象，(5) 与平台无关，(6)安全性，(7)分布式计算，(8)多线程。**

Java 程序的执行过程：是解释执行。

1.javac [选项] 源文件==编译

2. java [选项] 类名[参数] ==执行

2. 说出开发与运行 Java 程序的主要步骤。
3. 如何区分应用程序和小应用程序?
4. 说出 Java 源文件的命名规则。
5. 选择一种上机环境，参照实例，编写一个输出“hello world!”字样的 Java 程序。
6. Java 语言使用什么字符集？共有多少个不同的字符?
7. Java 语言标识符的命名规则是什么、
8. Java 有哪些基本数据类型，它们的常量又是如何书写的?
9. 指出下列内容哪些是 Java 语言的整型常，哪些是浮点数类型常量，哪些两者都不是。

1)E-4， 2)A423 ，

3)-1E-31,4)0xABCL,5).32E31,6)087,7)0xL,8)003,9)0x12.5,10)077,11)11E,12)056L,13)0.,14).0

第 2 章 运算和语句

本章主要内容(**次重点 10%**)

- 数据运算
- 语句
- Java 程序实例

本章重点：算术运算、自增和自减运算、关系运算和逻辑，Java 语句以及 java 程序实例。

本章难点：自增和自减运算、位运算和移位运算，while 语句、do…while 语句、for 语句和 Java 程序实例。

2．1 数据运算

在高级语言中，运算由表达式表示。**表达式**由**运算符**和**运算分量**组成，运算分量可以是常量、变量和方法调用。Java 语言的基本运算可分成以下几类：**赋值运算，算术运算，自增和自减运算，关系运算，逻辑运算，条件运算和字符串连接运算**等。

2．1．1 赋值运算

在 Java 语言中，符号“=”是赋值运算符，不是“相等”(相等运算符是“==”，见关系运算符的叙述)。赋值运算分为两类：一是简单赋值运算；二是复合赋值运算。

1． 简单赋值运算

简单赋值运算的一般形式如下：

变量 = 表达式

赋值运算的执行过程是：

(1) 计算赋值运算符的右端的表达式。

(2) 当赋值运算符两侧不一致时，将表达式值的类型自动转换成变量的类型。

(3) 将表达式的值赋值给变量，即存储到与变量对应的存储单元中。

完成一个赋值运算的表达式称为赋值表达式，赋值表达式是先计算表达式的值，然后将表达式的值赋值给变量。

例如，表达式 x=x+1，表示完成表达式 x+1 的计算，将计算结果赋值给变量 x。

这里的类型转换是指数值数据的类型自动转换，这样的**自动转换只能由简单类型向复杂类型转换，不能从复杂的转换成简单的**。即如下所示的从左到右转换：

byte->short->int->long->float->double

例如，以下代码说明 int 类型能自动转换成 double 类型：

```
int j=3;
double y=2.0;
y=j;//j 的值为 3，y 的值为 3.0
```

以下则是不正确的代码，double 类型不能自动转换成 int 类型。

```
j=y;
```

Java 语言约定赋值运算也有值，它的值就是赋予变量的值，因此，赋值运算的结果可以继续赋值。例如：

```
int j;
double y=2.0;
y=j=3;//j 的值为 3，,y 的值为 3.0
```

赋值运算符结合性“自右至左”，当连续有多上赋值运算时，是从右至左逐个赋值。

2． 复合赋值运算

在程序中，经常遇到在变量现在值的基础上作某种修正的运算。例如

x=x+5

这类运算的特点是：变量既要参与运算，又要接受赋值。为避免对同一个变量的地址重复计算，引入复合赋值运算符。常用的复合赋值运算符有：

+=、-=、*=、/=、%=

例如：

```
x+=5;//等价于 x=x+5
x *=u+v;//等价于 x=x*(u+v),这里括号不能省略
a+=a-=b+2;//等价于 a=a+(a=a-(b+2))
```

记 θ 为某个双目运算符，复合赋值运算

x θ =e

的等效表达式为

x= x θ (e)

当 e 是一个复杂表达式时，等效表达式的括号是必需的。

2．1．2 算术运算

算术运算要求**运算分量**的类型是**数值类型**的(整数类型和浮点数类型)。运算时，只需一个运算分量的是单目运算，需两个运算分量的是双目运算。算术运算的运算符是：

单目算术运算符：＋（取正）、－（取负）

双目算术运算符：＋（加）、－（减）、＊（乘）、/（除）、％（求余数）

说明：

⑴ 加、减、乘、除和求余数运算都是双目运算符，结合性都是从左至右。取正和取负是单目运算符，结合性是从右至左，其优先级高于＋、－、＊，％等双目运算符。

⑵ “/”为**除法运算符**，当**除数和被除数均为整数**类型数据时，则**结果也是整数类型数据**。例如 7 /4 的结果为 1 。

⑶ “％”为求余数运算符，**求余数运算所得结果的符号**与**被除的符号相同**。例如：５％３的结果为２，－５％３的结果为－２，５％－３的结果为２。

用算术运算符、运算分量和括号连接起来，符合 Java 语言语法规则的计算式，称为算术表达式。例如，如果变量 x 和 y 已经被正确声明，并且已经赋予初值，则以下的式子就是一个正确的算术表达式：

X*2+y*(x-5)

2.1.3 自增和自减运算

自增运算符“++”和自减运算符“--”是单目运算符，要求运算分量是数值类型的变量。其作用是变量的值增 1 或减 1 。这两个运算符与变量结合有以下四种可能形式：

++i 前缀形式，表示在引用变量 i 之前，先使 i 加 1 ，以加 1 后的 i 值为运算结果。

--i 前缀形式，表示在引用变量 i 之前，先使 i 减 1 ，以减 1 后的 i 值为运算结果。

i++后缀形式，表示在收用变量 i 之后，才使 i 加 1 ，即以增 1 前的 i 值为运算结果。

i--后缀形式，表示在引用变量 i 之后，才使 i 减 1 ，即以减 1 前的 i 值为运算结果。

例如：

```
i=4;j=++i;//i 结果为 5，j 的结果为 5
i=4;j=i++;//i 结果为 5，j 的结果为 4
i=4;j=--i;//i 结果为 3，j 的结果为 3
i=4;j=i--;//i 结果为 3，j 的结果为 4
```

上述例子说明，**对变量采用自增或自减，用前缀形式或用后缀形式，对变量本身来说，效果是相同的，但表达式的值不相同。前缀形式是变量运算之后的新值，后缀形式是变量运算之前的值。**

自增自减运算能使程序更为简洁和高效，但在使用时需注意“＋＋”和“－－”运算的运算**只能是变量，不能是常量或表达式**。例如，４++或（i+j）++都不是合法的。

2.1.4 关系运算

关系运算用来表达两个表达式值的比较，运算结果是布尔型。有 6 上关系运算符：

＜（小于）、＜＝（小于等于）、＞（大于）、＞＝（大于等于）、＝＝（等于）、！＝（不等于）

关系运算对左右两侧的值进行比较，如果比较运算的结果成立，则值为 true；不成立为 false.

上述 6 个关系运算符的优先级不完全相同。＜、＜＝、＞、＞＝的优先级高于＝＝，！＝。

例如，表达式 x>y==c<d,等价于(x>y)==(c<d)。该表达式的意义是 x>y 与 c<d,或同时成立或同时不成立。

关系运算符的优先级低于算术运算符的优先级。

例如：x>u+v,等价于 x>(u+v).

关系运算符的结合方向是自左至右。

2.1.5 逻辑运算

逻辑运算用于描述逻辑表达式，实现连续多个条件的**逻辑与、逻辑或、逻辑否定**的判定。有 3 个逻辑运算符：

＆＆（逻辑与）、||（逻辑或）、!（逻辑否定）

其中：运算符＆＆和||是双目运算符、运算符！是单目运算符。逻辑运算的操作数必须是布尔型的，结果也是布尔型的。

逻辑否定“!”的优先级**高于算术运算符**的优先级。**逻辑与**“＆＆”和**逻辑或**“||”的优先级**低于关系运算符**的优先

级。

表 2-1 是逻辑运算的“真值表”，表中列出当运算分量 a 和 b 的值在不同组合情况下，各种逻辑运算的结果。

表 2-1　逻辑运算真值表

a	b	!a	!b	a&&b	a||b
true	true	false	false	true	true
true	false	false	true	false	true
false	true	true	false	false	true
false	false	true	true	false	false

例如：

a>b&&x>y　　等价于(a>b)&&(x>y)

a!=b||x!=y　　等价于(a!=b)||(x!=y)

x==0||x<y&&z>y　　等价于(x==y)||((x<y)&&(z>Y))

!a&&b||x>y&&z<y　　等价于((!a)&&b)||((x>y)&&(z<y))

逻辑运算符用来描述逻辑表达式。例如，闰年的条件是：每 4 年一个闰年，但每 1 0 0 年少一个闰年，每 4 0 0 年又增加一个闰年。如果年份用整数类型变量 year 表示，则 year 年是闰年的条件是：

（year 能被 4 整除，但不能被 1 0 0 整除）或（year 能被 4 0 0 整除）

用逻辑表达式可描述如下：

（year/%4==0&&year%100!=0）||year%400==0

需要**特别指出的是，逻辑与和逻辑或的运算符有以下性质，Java 语言利用这些性质，在进行连续的逻辑运算时，不分逻辑与和逻辑或的优先级进行计算，而是顺序进行逻辑与和逻辑或的计算，一旦逻辑子表达式或逻辑子表达式或逻辑表达式能确定结果，这不再继续计算。**

（1） 对表达式 a&&b, 当 a 为 false 时，结果为 false，不必再计算 b；仅当 a 为 true 时，才需计算 b。

（2） 对表达式 a||b，当 a 为 true 是，结果为 true，不心再计算 b;仅当 a 为 false 时，才需计算　b。

例如：设有 a=b=c=1, 计算++a>=1||++b<++c。从左到右顺序逻辑或表达式，先计算子表达式++a>=1, 变量 a 的值变为 2，++a>=1 为 true, 整个逻辑或表达式的值已经为 true，不再计算右边的子表达式++b<++c。因而变量 b 和 c 的值不变，仍为 1 。

在具体编写程序时，也应利用以上性质。用逻辑与表达两个条件必须同时成立时，如果条件不成立，条件 2 的值不便计算，则逻辑表达式应写成：

条件 1 &&条件 2

避免在条件 1 不成立情况下，计算条件 2 。例如，要表示 y/x>2 和 x!=0 同时成立，应写成：

x!=0&&y/x>2

当 x 为 0 时，不会 y/x。而写成：

y/x>2&&x!=0

是不正确的，因为当 x 为 0 时，不能计算 y/x。对于逻辑或也有类似情况。

2.1.6 条件运算

条件运算是一个三目运算，一般形式如下：

逻辑表达式？表达式 1 ：表达式 2

条件运算的执行过程是：

⑴ 计算逻辑表达式

⑵ 如果逻辑表达式的值为 true，则计算表达式 1 ，并以表达式 1 的值为条件运算的结果（不再计算表达式 2 ）

⑶ 如果逻辑表达式的为 false, 则计算表达式 2 ，并以表达式 2 的值为条件运算的结果（未计算表达式 1 ）。

例如：

x>y ？ x+5 ： y-4

如果 x>y 条件为 true，则上述表达式取 x+5 的值，否则取 y-4 的值。

条件运算符（？ ：）的优先级高于赋值运算符，低于逻辑运算符，也低于关系运算符和算术运算符。

例如：max = x>y　?x+5:y-4

等价于：max = ((x>y)?x+5:(y-4))

条件运算符的性为“自右至左”。例如

x>y?x:u?v?u:v

等价于：x>y?x:(u>v?u:v)

条件表达式的返回值类型由表达式 1 和表达式 2 的类型确定。如果表达式 1 值的字节数比表达式 2 的值的字节数多，则条件表达式值的类型与表达式 1 的类型相同；反之，则与表达式 2 的类型相同。

2.1.7 其他运算

除前面介绍的运算外，还有许多运算，本节只介绍位运算和移位运算。位运算和移位运算实现对二进制位串数据的运算，主要应用于与计算机内部表示直接有关的运算，读者可以跳过这些内容。

2.2 语句

一个计算过程由一系列计算步骤组成。一个计算步骤或用一个，或用一个计算流程控制实现。程序语言用描述计算步骤。在 Java 语言中，**语句分为基本语句、控制结构语句以及 package 语句和 import 语句**等。其中控制结构语句包括复合语句、if 语句、switch 语句、循环语句和 try…catch 语句。其中循环语句有 while 语句、do…while 语句、for 语句三种。

2.2.1 基本语句

基本语句主要有表达式语句、空语句、break 语句、continue 语句、return 语句等。基本语句都以分号为结束符。

1. 表达式语句

在赋值表达式、自增自减表达式和方法调用表达式之后加上分号即变成语句，称它们是表达式语句。例如，表达式“k++”,写成 “k++;”就是一个表达语句。最典型的表达式语句是赋值表达式构成的语句，譬如：

```
k=k+2;
m=n=j=3;
```

赋值表达式语句在程序中经常使用，习惯又称为赋值语句。

另一个典型的表达式语句是方法调用表达式之后接上分号：

```
方法调用;
```

该表达式语句虽未保留方法调用的返回值，但方法调用会引起实参向形参传递信息和执行方法体，将使变量获得输入数据；调用输出方法使程序输出计算结果等。

2. 空语句

空语句是只有一个分号的语句，其形式为

```
;
```

实际上，空语句是什么也不做的语句。语言引入空语句是出于以下实用上的考虑。例如，循环控制结构的句法需要一个语句作为循环体，当要循环执行的动作由循环控制部分完成时，就不需要有一个实际意义的循环体，这时就需要用一个空语句作为循环体。另外，语言引入空语句使语句序列中连续出现多个分号不再是一种错误，编译系统遇到这种情况，就认为单独的分号是空语句。

3. break 语句

break 语句必须出现在多路按值选择结构或循环结构中，break 语句的执行强制结束它所在的控制结构，让程序从这个控制结构的后继语句继续执行。break 语句的书写形式是

```
break;
```

break 语句的应用，将在介绍 switch 语句和循环语句时作进一步讨论。

4. continue 语句

continue 语句只能出现在循环结构中，continue 语句的执行将忽略它所在的循环体中在它之后的语句。如果 continue 语句在 while 语句或 do…while 语句的循环体中，使控制转入对循环条件表达式的计算和测试；如果出现在 for 语句的循环体中，使控制转入到对 for 控制结构的表达式 3 的求值。简单地说，continue 语句提早结束当前轮次循环，进入下一轮次循环。continue 语句的书写形式是

```
continue;
```

continue 语句的应用，将在循环语句中进一步讨论。

5. return 语句

return 语句只能出现在方法体中，return 语句的执行将结束方法的执行，将控制返回到方法调用处。return 语句有两种形式：

```
return ; 或 return 表达式;
```

第一种形式只有用于不返回结果的方法体中，第二种形式用于有返回结果的方法体中。执行第二种形式的 return 语

句时，方法在返回前先计算 return 后的表达式，并以该表达式值作为方法返回值，带回到方法调用处继续计算。

2．2．2 复合语句

复杂计算经常被分解为一个计算步骤序列。整个计算步骤序列在逻辑上是一个整体，要求计算机从计算步骤序列的第一个计算步骤开始，顺序执行每个计算步骤，直至最后一个计算步骤。

在 Java 语言中，用花括号括住一个顺序执行的计算步骤序列描述顺序结构，这样的顺序结构称作复合语句。复合语句中的每个计算步骤可以是单个语句，也可以是一个控制结构，特别情况也可以为空。

以交换两个整型变量 x 和 y 的值为例，实现变量 x 和 y 值的交换可分解为以下顺序执行的三个赋值步骤：

```
temp =x;
x=y;
y=temp;
```

把交换变量 x 和 y 的值作为一个不可分割的整体来考虑，应把上述 3 个语句写成如下形式的复合语句:

```
{
    int temp;
    temp = x;
    x =y;
    y=temp;
}
```

在构造复合语句时，这完成指定的工作，可能需要临时工作变量。例如，以上例子中的 temp 变量。在语句序列中插入变量定义，引入只有复合语句内的语句可使用的临时变量。用复合语句描述计算步骤序列，并定义自己专用的局部变量，使复合语句有很强的独立性，它不再要求外面为它定义专用变量。一个计算步骤序列用复合语句描述后，它已经是一种单个语句。复合语句常被用作其他控制结构的成分语句。

2．2．3 if 语句

根据当前情况选择不同的计算，需用选择控制结构实现。有两种选择控制结构：两路条件选择结构和多路按值选择结构。

两路条件选择由一个条件和两个供选择的分支语句执行。

两路条件选择结构用 if 语句描述。if 语句根据条件表达式的值为 true 或 false，从两个语句中选取一个语句执行。if 语句的一般形式为

```
if(条件表达式)
    语句 1
else
    语句 2
```

if 语句的执行过程是：

(1) 计算条件表达式的值。

(2) 测试表达式的值并选择语句执行。如果表达式的值为 true，则执行语句 1；否则执行语句 2 。

注意，无论条件表达的值是 true 还是 false，只执行语句 1 或语句 2 中的一个，不会两个都执行。

当 if 语句中的语句 2 为空语句时，可简写成：

```
if(条件表达式)
    语句
```

这种形式的 if 语句的执行过程是：

(1) 计算表达式的值

(2) 测试表达式的值。若表达式的值为 true，则执行它的成分语句后结束 if 语句。否则，立即结束 if 语句。

在 if 语句中的语句 1、语句 2 可以是任何语句。当它们中的某一个是由多个语句组成时，必须将它们用花括号括住。

当 if 语句中供选择的语句又是 if 语句时，if 语句就呈嵌套的形式，这时应注意 else 与 if 的对应关系。java 语言约定：else 与它前面最接近的 if 对应。

2．2．4 switch 语句

经常遇到这样的选择控制，对变量表达式的每一个可能的值分别作不同的计算。如果用两路条件选择结构描述这样的选择控制，由于要逐一测试是否等于某个值，if 语句嵌套的层次就很深，程序的可读性和可修改性也很差。多路按值选择结构可避免这个问题。**表达式的值的类型只能是 char,byte short int 类型。**

在 Java 语言中，用 switch 语句描述多路按值选择结构。switch 语句包含一个表达式，用于按表达式值的不同作相应选择。另外，还有一系列由 case 开头的子句和一个可选的 default 开头的子句，每个子句有一个可能为空的语句序列。switch 语句一般形式为：()

```
switch(表达式){
case 常量表达式 1：语句序列 1;break;
case 常量表达式 2：语句序列 2;break;
case 常量表达式 n: 语句序列 n;break;
default:语句序列 n+1;break;
}
```

2.2.5 try…catch 语句

```
try{
    语句块//执行该语句块，可能会发生异常
}catch(异常类型 e){
异常发生时要执行的语句;
}finally{
无论异常发生与否，都要执行的语句。
}
```

一个 try 可以有多个 catch 和它标识的语句块。**catch 的内容可以为空，但是{}不可省**。

2．2．6 while 语句。

2．2．7 do…while 语句。

2．2．8 for 语句

2．2．9 嵌套的循环结构

2．2．10 循环体内的 continue 和 break 语句

2．2．11 package 语句

```
package 包名;
```

package 语句必须出现在源程序文件的起始行，表示出现在该文件中的所有类都属于这个程序包。如果有多个源程序的程序包声明使用相同的包名，则表示这些源程序文件中的全部类都属于这个程序名。包名是一个目录名，其中也可以有路径。如果 源程序文件的首行没有 package 语句，则被默认为是无名包，存于当前目录中。

2．2．12 import 语句

import 语句用于引入所需要的类。import 语句的格式为：

```
import 包名.类名;
```

如果要从一个包中引入多个类，则可以用通配符“* ”。

在一个 Java 源程序中可以有多个 import 语句，它们必须出现在 package 之后，类的定义之前。

2．2．13 java 应用程序的基本结构

```
[package 包名; ]
import 包名.类名;
…
[public] class 类名{
int val= …
  public static void main(String args[])
  {
  …
  }
}
```

java 程序由类和对象构成，而类又由方法和变量构成。java 的方法由语句构成，而语句又由标识符和运算符构成。

2．3 java 程序实例

第 3 章　面向对象编程基础

本章主要内容(重点内容：15%)

面向对象的基本概念

Java 的类和对象

接口

基本类

Java 小应用程序基础

本章重点：Java 语言的类、对象、继承、接口以及 Java 小应用程序基础

本章难点：继承、多态性、接口

3．1 面向对象的基本概念

自计算机诞生到现在，程序设计语言发展经历了面向机器、面向过程和面向对象 3 个阶段。面向机器阶段用机器指令，即机器语言，或用助记符代替机器指令的汇编语言编写程序。使用**面向机器**的语言编写程序，编程模式是：

程序＝数据＋指令

编程时，需要考虑的内容包括计算机的硬件特性、数据位置安排、I／O 设备的控制细节等。这类程序的缺陷是程序的可读性、可维护性、可移植性极差，因此编程效率很低。面向机器语言主要应用于一般的科学计算和简单控制处理。

面向过程阶段是使用高级程序设计语言编程，有代表性的语言有 FORTRAN、PASCAL、C、COBOL 等。用面向过程语言编程，编程模式是：

程序＝数据结构＋算法

编程时，需要考虑的内容是程序做什么、怎么做，重点考虑每个实现细节。采用的主要技术是结构化控制结构和模块化设计。这类程序除科学计算外，大量应用于一般的应用程序开发。这种编程技术的缺陷是不适宜应用于图形用户界面、事件驱动编程，难以开发超大规模的应用程序，特别是随着程序规模的进一趟扩大，系统变和非常难以维护。

面向对象阶段是目前正广泛流行的编程方法。面向对象编程语言有 Smalltalk、C++、Java 等。面向对象编程语言提供一种全新的编程技术。用面向对象编程，编程模式改为：

程序＝对象＋消息

程序设计者考虑的是对象的描述、对象间的关系、类的管理、什么时候和什么地方调用对象的哪一种方法。面向对象编程的适应范围大大扩大，能适用于大规模应用程序的解决方案、网络计算等。对于编程者来说，最大的优点是面向对象编程能有效支持重用，使超大规模的程序也变行相对容易维护。

面向对象程序设计语言引入许多概念和机制，包括**抽象、对象、消息、类、继承、多态性**等。本节先介绍这些概念和机制，随后将详细介绍 Java 语言如何体现和应用这些概念和机制。

1．抽象

抽象（abstraction）是程序设计中最经常使用的技术之一，因为**抽象是有效控制程序复杂性的重要手段**。在设计初始阶段，采用自顶向下的设计方法，暂时不关心具体细节，首先设计出抽象的算法；随后，抽象的算法步骤逐步被具体的实现替换。程序设计中利用抽象，在面向过程阶段，系统按函数和模块进行设计。在面向对象阶段，利用抽象按类设计。类是目前支持抽象的最好工具。

2．对象

在实际生活中，人们每时每刻与对象打交道，例如，汽车、自行车，在特定的应用领域都是对象（Object）。这些现实世界中的**对象**都有**状态、行为和名称**。例如，自行车的品牌、两个轮子的尺寸、行进中的速度等能描述自行车的状态；自行车提供包括加速、减速、刹车等行为。在面向对象语言中，**对象的名称用于程序引用对象，对象的成员变量，用于存储对象的状态信息，对象的方法用于描述对象的行为**。

3．消息

一个应用程序总会包含许多对象，通过这些对象之间的交互，实现更高级、更复杂的行为。例如，小张用他的自行车上学读书，在上学的过程中，小张骑着自行车，一会儿让车加速，一会儿让车减速，一会儿又刹车。总之，一路上，小张与他的自行车一直在进行交互，直到学校。

在程序系统中，**对象之间的交互**通过相互发送消息（Message）实现。当对象 A 希望对象 B 执行 B 的一个方法时，用对象 A 发送消息给对象 B 来实现。消息发送时，如果还需要其他参数，消息可带参数一起发送。

4．类

通常程序系统中会有许多同样类型的对象。例如，每个同学都有一辆自行车。尽管每辆自行车的主人不同，各自行车当时所处的状态不同，但是，自行车所包含的属性、自行车能提供的行为是相同的人们可以统一给出同类对

象的各项属性和能提供的所有行为，据此，区别自行车和汽车是不同种类的对象。如同避免重复描述每辆自行车的属性和行为一样，为了避免程序逐一描述同类中的每个对象的所有属性，详细给出对象的每个方法的描述，**把同一类对象的所有共同的属性和行为放在一起描述。这种声明对象共有属性和行为的机制称为类**。类的声明定义了类的所有对象的共有的属性和方法。这样，如果程序声明了自行车类的共有属性和方法，则小张的自行车是自行车类一个实例。如果程序需要，可随时由自行车类创建小王的自行车、小李的自行车等。所以类是对一组相同对象的描述，**类概括了同类对象的共有性质：数据和方法。类的每个对象都有自己的标识，但它们具有相同的一组属性和提供相同的一组方法**。

5．继承

除对象按类划分外，不是同一类的对象可能会存在某些相似性。**继承**（Inheritance）**就是在已有类基础上，扩充属性，或扩充与改写其某些方法，生成新的类，这个方式称为继承**。继承定义的类称为子类，被继承的类称为**超类（父类）**，子类自动含有父类具有的属性和方法。继承具有传递性。例如，自行车可分为山地自行车、比赛用自行车等。山地自行车、比赛用自行车都是自行车类的子类。所之，自行车类是山地自行车、比赛用自行车的超类。**超类声明定义共同的特性，子类继承超类的共有特性，还可增加某些特殊性，并可扩充和修正部分行为。**如同大家非常熟悉的几何图形，抽象的几何图形有位置、面积等共同特性，有求面积等方法。四边形、三角形、点都是几何图形的子类。同样四边形又可以有普通的四边形、长方形、菱形等。采用继承声明定义的子类可以有父类的属性和方法，也可以增加新的属性和方法，并可以对父类的某些方法给出新的定义。例如，各种类别的几何图形有不同的求面积的方法。继承的**最大好处**是**对象能按层次划分**，并在子类中，与父类相同的属性和方法可以不再重复声明。继承体现了面向对象方法与现实世界中人们的抽象思维方式保持一致。

6．多态性

多态性（Polymorphism）有多种表现形式，这里所说的**多态性是指类型适应性多态性，这来自继承，不同继承层次的对象对同样的方法有不同的实现**。类型适应性多态性要求自动按对象的实际类型正确调用它的方法。例如，各类几何图形构成一个继承体系，每种几何图形有特定的求面积方法。一个以几何图形对象为参数的方法 f()运行时，如果需要求这个参数对象的面积，系统根据对象的实际类型，自动按实际对象的类型选取正确的求面积方法。在方法 f()运行之前，只知道参数将对应一个几何图形对象，不知道它是三角形、圆，还是其他别的几何图形。这样，在方法 f()运行之前不能确定参数的求面积方法。这种多态性要求直至方法 f()运行时，知道了与参数对应的实际对象的类型，才确定求面积方法。这种类型适应性的多态性需要采用动态联编（Dynamic Binding）技术实现。

联编是将发送给对象的**消息与**含执行该消息方法的**对象连接起来**。当联编在**编译和连接阶段实现**时，这样的联编过程称为**静态联编**；当**联编**推迟至**运行时间实现**时，该联编过程称为**动态联编**。动态联编是面向对象语言必须具有的一种能力，是实现上述多态性的技术基础。

3．2　Java 的类和对象

本节介绍 Java 语言的面向对象机制，内容包括类和对象，以及支持面向对象编程提供的一些机制。

3．2．1　类

类是一种类型，类封装对象的属性和方法，是同一类对象的模板。Java 程序的主要部分是描述类。

1．类的声明

声明一个新类的基本格式如下：

```
class 类名//声明新类的基本格式
{
        类体
}
```

其中关键字 class 引导一个类的声明，类名是一个标识符。类体给出类的成员变量定义和方法定义，其中类的成员变量用于存储对象的属性，方法描述对象的行为。类体中**不能有独立的执行代码，所有的执行代码只能出现在方法中。**

[例　3.1]学生类 Student 的声明。

```
class Student{
    float height,weight;
    String name,sex,no;
    void setStudent(String n, String s, String o){
        name= n;sex =s;no=o;
        System.out.println(“name:”+name);
```

```
        System.out.println(“sex:” +sex);
        System.out.println(“no:” +no);
    }
    void setWH(float w,float h){
        weight = w; height = h;
    }
}
```

在类 Student 中，定义了 5 个成员变量：height、weight、name、sex 和 no；2 个方法：setStudent()和 setWH()。

2．成员变量

成员变量定义的一般形式如下：

[修饰字]　类型　变量名；

其中，修饰字可以空缺，修饰字用于对成员变量限制其访问权限，成员的访问权限有 4 种：private、protected、public 和友好的，参见 3.2.5 节。变量的类型可以是任何数据类型，变量的名字是一个标识符。Java 中**成员变量的名字可以与**类的**方法的名字相同**。

成员变量对类内定义的方法都有效。类的方法中也可定义变量，这种变量称为局部变量，局部变量只在定义它的方法中有效。以下代码示意引用成员变量和局部变量的合法性：

```
class A{
    int x;
    int f(){
        int a = 1;
        x = a;
    }
    int g()
    {
        int y;
        y= a+x;
    }
}
```

3．方法

方法给出对象的行为，方法的声明如同 C 语言函数定义。方法声明的一般形式如下：

返回值类型　方法名（类型　参数名，…，类型　参数名）

```
 {
    方法体
}
```

返回值类型声明方法返回值的数据类型。如果方法无返回值，就用 void 关键字。方法可以没有参数，多个参数用逗号分隔，参数类型可以是任何数据类型。

[例 3.2]方法的例子，类 B 有方法 u()和 v()。

```
class B{
    double x,y;
    int u(int x,int y){
        return x*x+y*y+1;
    }
    float v(int a, float b){
    return a*x+b*y;
    }
}
```

如果某方法中的**参数名或局部变量的名与类的成员变量**的名相同，则**成员变量**在这个方法内暂时被隐藏。例如，在以下类 B 的声明中，方法 u()中的参数 x 和 y 与成员变量 x 和 y 同名，方法 u()中的 x 和 y 引用的参数 x 和 y。**如果成员变量在方法内被隐藏，又在方法中要引用成员变量，必须使用** this（参见 3.2.7 中关于关键字 this

的叙述），this 表示调用该方法的当前对象。

```
class Jerry{
    int x,z;
    void g(int z){
      int x = 5+z;
      this.x = x;
      this .z = z;
    }
}
```

参数名和局部变量名可随意命名，建议尽量使用互不相同的名。

在 Java 程序中，类的方法可以重载。方法重载是指在一个类中定义多个相同名字的方法，但这些方法或者参数个数不同或者顺序参数的类型不同。

[例 3.4]方法重载的例子，类 C 的 4 个 fun（）方法或因参数个数不同，或因参数的类型顺序不同，是 4 个合理的重载方法。

```
class C{
    float fun(float s){
        return s*s;
    }
    float fun(float x,int y){
        return x*x+y*y;
    }
    float fun(int x,float y){
        return x*x+y*y;
    }
    float fun(float x,float y){
        return x*x+y*y;
    }
}
```

编译器将根据方法调用时的参数个数和参数类型及顺序确定调用的是哪一个方法。例如，调用方法 fun()时，如果提供一个 float 参数，则是调用第一个 fun()方法，如果参数有两个，且第一个是 float 参数，第二个是 int 参数，则是调用第二个 fun()方法。方法参数的名称不能用来区分重载方法。

4．构造方法

构造方法是一种特殊的方法，这种方法的名与它的类名相同，并且不返回结果，也不写上 void 关键字。构造方法的作用是创建类的对象，并给对象初始化。构造方法是公共方法，但程序不能显式调用构造方法。程序运行时，当有对象要创建时，由系统自动调用构造方法。

[例 3.5]类内定义构造方法的例子。类 Point 定义了两个构造方法。

```
class Point{
    int x,y;
    Point(){
        x=10;y=20;
    }
    Point(int x,int y){
        this.x=x;
        this.y=y;
}
int getX(){return x;}
int getY(){return y;};
}
```

如果类的声明没有定义构造方法，系统就增补一个没有参数的默认构造方法。

3.2.2 对象

类被声明后，就可用类创建对象，被创建的对象称为类的实例。程序使用对象需依次经历 4 个步骤：**声明对象、创建对象、使用对象和撤销对象**。

1．声明对象

由于类是一种引用类型（参见节后面叙述的对象的内存模型），声明对象只是命名一个变量，这个变量能引用类的对象。由于对象还没有创建，所以也暂不要为对象分配内存。声明对象的一般形式为：

类名　　对象名;

例如，代码：

```
Point p1,p2;
```

这里的 Point 是前面声明的类名，上述代码声明 p1,p2 两个对象。

2．创建对象

创建对象就是为对象**分配内存**，为对象分配内存也称为类的**实例化**。一般形式为：

new 构造方法([参数表])

其中参数被构造方法用于给对象设置初值。例如，代码：

```
p1 = new Point();p2= new Point(30,40);
```

对象 p1 用无参数的构造方法初始化，使 p1 的 x 坐标为 10,y 的坐标为 20,对象 p2 用带两个参数的构造方法初始化，使 p2 的 x 坐标为 30,y 坐标为 40。

对象创建的两个也可一起完成，一般格式为：

　　类名　对象名　＝　new 构造方法（[参数表]）

例如，代码：

```
Point p3 = new Point(),p4=new Point(60,70);
```

同时创建两个 Point 对象，p3 的坐标为 10，y 的坐标为 20;p4 的坐标为 60,y 坐标为 70。

3．对象的内存模型

Java 语言将类型分成**基本类型和引用类型**两种。第 2 章介绍的整型、浮点型、字符型等是基本类型，程序引用这种类型的变量，采用直接访问形式。

在 Java 语言中，数组类型和类类型是引用类型。程序访问引用类型的变量采用间接访问方式。

4．使用对象

程序使用对象有多种情况：或为对象**设置状态**、或**获取**对象的**状态**、或**改变**对象的**状态**、或**应用**对象的某种**方法**。前三种需要访问对象的成员变量，最后一种要调用对象的方法。程序通过操作符“.”对某对象的成员变量进行访问和方法调用。一般形式为：

　　对象名.成员变量

　　对象名.方法（[参数表]）

参见例 3.6，程序首先创建对象 p1 和 p2，并利用对象提供的方法 getX()和 getY()，用代码 p1.getX()获得对象 p1 的属性值,p2.getY()获得对象 p2 的 y 属性值。由于类 Point 声明中,成员变量是 public 的(参见 3.2.5 访问权限)，程序也可以直接用代码 p1.x 获得对象 p1 的 x 属性值；用代码 p2.y 获得对象 p2 的 y 的属性值。

[例 3.6]使用对象的程序例子

```
public class Examle3_1{
    public static void main(String []args){
    Point p1,p2,p3;
    p1 = new Point();
    p2 = new Point(40,50);
    p3= new Point(p1.getX()+p2.getX(),p1.getY()+p2.getY());
    System.out.println("p3.x=" +p3.getX()+",p3.y=" +getY());
    Point p4 = new Point(p1.x,p.y);
    System.out.println("p4.x=" +p4.x+",p4.y=" +p4.y);
    }
}
//这里是类 Point 声明代码，限于篇幅，这里不再给出，参见 3.5 中 Point 类的定义。
```

3.2.3 实例变量和类变量

类的成员变量又分为实例变量和类变量。在定义成员变量时，用关键字 static 修饰的是**类变量**，定义时**未用 static 修饰**的是**实例变量**。例如，以下代码定义成员变量 x 是实例变量，成员变量 y 是类变量。

```
class D{
float x;
static int y;
…
}
```

由前面的示例程序看出，程序定义的每个对象都有自己的实例变量。例如，前面程序中，对象 p1,p2,p3 和 p4 都有自己的实例变量 x 和 y。类变量是类的所有对象共享成员变量，一个类变量在类中只有一个，它属于整个类，而不属于类的某个对象。**引用类变量**途径有两条，或**通过类**，或**通过对象**，格式如下：

类名.类变量名　　或　对象名.类变量名

不同对象的实例变量将被分配不同的内存空间。改变类的某一个对象的实例变量的值不会影响其他对象的实例变量。

类变量的内存只有一处，让类的所有对象共享。从类的任一对象改变类变量，类的其他对象都能发现这个改变。

3．2．4　实例方法和类方法

如同类的成员变量有实例变量和类变量两种一样，类的方法也有两种：实例方法和类方法。在方法定义时，冠以修饰字 static 的方法称为类方法，没有冠以 static 修饰字的方法是实例方法。

[例 3.7]类 D 定义了一个实例方法，两个类方法。

```
class D{
    int a;//实例变量
    static int c;//类变量
    float max(int x,int y)//实例方法
     {a = x>y?x:y;}
    static int setCount(int c0)//类方法
    { c=c0;}
static void incCount(int step)//类方法
 {c+=step;}
}
```

类中的实例方法可以互相调用，并可调用类方法。**类方法也能相互调用，但不能直接**调用实例方法，**除非**类方法**引入局部对象**，然后**通过局部对象调用实例方法**。另外，**类方法能直接引用类变量，不能引用实例变量**。实例方法可引用实例变量，也可引用类变量。例如 ，例 3.8 给出的类声明中有些是合法的代码，而有些是不合法的代码。

[例 3.8]含不合法的代码的例子。程序的注释指明合法和不合法的原因

```
class E{
    float u;
    static float v;
    static void setUV(boolean f){
    u = s_m(f);// 非法，类方法不可以调用实例变量 u
    v = s_m(f);//合法，类方法可以调用类方法
    v = r_m(!f);//非法，类方法不能直接调用实例方法。
    }
    static float s_m(boolean f){
    return f?u:v;//非法，类方法只能 类变量
    }
    float r_m(boolean f){
    return f?u:v;//合法，实例方法能引用实例变量和类变量
    }
}
```

实例方法可以访问类变量和当前对象的实例变量。**实例方法必须通过对象调用**，不能通过类名调用。类方法只能类变量，

不能够访问实例变量。类方法除了可以通过实例对象调用之外，还可以通过类名调用。

[例 3.9]说明类变量用法的，应用程序。改写 Point 类的声明，在 Point 类中增加一个类变变量 pCount,它的初值为 0。在构造方法中，有类变量 pCount 增 1 的代码，这能记录类的对象个数。

```
public class Example3_2{
    public static void main(String args[]){
      Point p1,p2,p3;
      p1 = new Point();
      p2 = new Point(40,50);
      p3 = new Point(p1.getX()+p2.getX(),p1.getY()+p2.getY());
      System.out.println(“p3.x=”+p3.getX()+”,p3.y=”+p3.getY());
      Point p4 = new Point(p1.x,p2.y);
      System.out.println(“p4.x=”+p4.x+”,p4.y=”+p4.y);
      System.out.println(“程序共有 Point 对象”+Point.pointNum()+ “个”);
    }
}
class Point{
  int x,y;
  static int pCount =0;
  Point(){
    x=10;
    y=20;
    pCount++;
  }
  Point(int x,int y){
    this.x = x;
    this.y=y;
    pCount++;
  }
  static int pointNum(){return pCount;}
  int getX(){ return x; }
  int getY(){  return y; }
}
```

由于 java 系统内设废弃内存回收程序，所以一般情况下，一个对象使用结束后，程序不必特别通知系统撤销对象。但有时为提高系统资源的利用率，程序也可通过调用方法 finialize()**显式通知系统，请求系统撤销对象**。

3．2．5 访问权限

访问权限实现访问控制。在类的成员中引入权限控制，保护类的成员不在非期望的情况下被引用。在类声明中，除了类中方法总能访问类的成员变量外，Java 语言为其他类的方法访问本类成员变量和方法，提供以下 4 种访问权限：

public:设置没有限制的访问权限，其他类的方法都能访问。

private：设置最强限制的访问权限，其他类的方法都不能访问。

protected:只限于子类和同一包中的类的方法能访问。

<default>：(无修饰，友好的)只允许同一包中类的方法访问。

1． public(共有)

类的成员变量被设置成 public 访问权限，则类外的任何方法都能访问它。这样的成员变量就没有任何安全性，在应用程序中很少使用。通常，只有为对象设定的功能性方法被设置 public 访问权限，让类外的方法可以通过对象调用这样的方法，让对象完成它的服务功能。

2． private(私有)

类的成员变量被设置 private 访问权限，则类外的任何方法都不能访问它。方法被设置成私有的，通常这些方法是类内部专用的方法。类通常另定义一些 public 访问权限的方法，通过这些方法访问的成员变量，这样的成员变量的安全性能得到有效的保证。

3. protected(受保护)

受保护访问权限是一种公有权限和私有权限之间的访问权限。例如，在类 A 的声明中，成员 x 被定义成是受保护的，则类 A 的子类和与类 A 同一包中的别的类可以访问类 A 的受保护成员 x；但对于不是类 A 的子类或与类 A 不在同一包中别的类来说，不可访问受保护成员 x。通常同一包中的一些类与定义受保护成员的类有许多相关性，为了提高系统的效率，让这些相关类的方法可直接访问，这样的成员可考虑设置受保护访问权限。

4. 友好变量和友好方法

没有修饰的成员变量和方法称为友好变量和友好方法。与受保护访问权限比较，如果只允许同一包中的相关类的方法访问，不让类的子类和其他类的方法访问时，可设置成友好变量和友好方法。

在同一源程序文件中的类，总是在同一包中，如果声明类 A 的源文件中用 import 语句引入了另外一个包中的类 C，并用类 C 创建了一个对象 c，那么对象 c 将不能访问类 A 的友好变量和友好方法。如果一个类被修饰为 public 的，那么可以在任何另外一个类中使用该类创建对象。如果一个类不加任何修饰，那么在另外一个类中使用这个类创建对象时，要保证它们是在同一包中。

访问权限说明能访问的范围，表 3-1 是访问权限表，其中打勾的表示可访问，没有打勾的表示不可访问。

表 3-1 访问权限表

	同类	同包的其他类	所有其他类	不同包的子类
public	YES	YES	YES	YES
private	YES			
protected	YES	YES		YES
<无修饰>(友好)	Yes	Yes		

3．2．6 继承

继承是面向对象语言的重要机制。借助继承，可以扩展原有的代码，应用到其他程序中，而不必重新编写这些代码。在 java 语言中，**继承是通过扩展原有的类，声明新类来实现的**。扩展声明的新类称为子类，原有的类称为超类(父类)。继承机制规定，子类可以拥有超类的所有属性和方法，也可以扩展定义自己特有的属性，增加新方法和重新定义超类的方法。

java 语言**不支持多重继承**，限定一个类只能有一个超类。在子类声明中加入 extends 子句来指定超类。格式如下：

```
class 子类名 extends 超类名
{
    类体
}
```

例如，代码：

```
class E extends D
{…}
```

声明类 E 的超类是类 D，这里假定类 D 已在某处声明。类声明时，如果缺省 extends 子句，未指定超类，则该类的超类是系统声明的类 java.lang.Object.

子类对父类的继承性，java 语言作以下规定：

子类自然地继承其父类的不是 private 的成员变量作为自己的成员变量，并且自然地继承父类中不是 private 的方法作为自己的方法。

[例 3.10]继承声明子类的例子

程序先声明 Mother 类，定义了一些成员变量和方法。类 Daughter 继承类 Mother,并增加了一些成员变量和方法。 通过 Daughter 类对象 girl，能调用从 Mother 类继承的方法，也能调用 Daughter 类新增的方法，或重新定义的方法。

```
class Mother{
    private int money;
    float weith,height;
    String speak(String s){return s;}
    float getWeitht(){return weight;}
    float getHeight(){return height;}
    String dance(){return “我会跳舞”;}
}
```

```
class Daughter extends Mother{
    String cat;
    String sing(String s){return s;}
    String dance(){return “我是小舞蹈演员”;}
}
public class Example3_10{
    public static void main(String argc[]){
      Daughter girl =new Daughter();
      girl.cat=”漂亮的帽子”;
      girl.weight = 35.0f;
      girl.height =120.0f;
      System.out.println(girl.speak(“我是女儿”));
      System.out.println(girl.speak(“我像母亲一样很会说话”));
      System.out.println(girl.speak(“我重”+girl.weight + “公斤”));
      System.out.println(girl.speak(“我高”+ girl.height + “公分”));
      System.out.println(girl.speak(“我还比母亲多一顶”+girl.cat));
      System.out.println(girl.sing(“我还能唱歌”));
      System.out.println(girl.dance());
    }
}
```

程序的输出结果是：

我是女儿
我像母亲一样很会说话
我重 35.0 公斤
我高 120.0 公分
我还比母亲多一顶漂亮的帽子
我还能唱歌

在例 3.10 的程序中，子类 Daughter 中声明的方法 dance()，跟超类中的方法 dance(),方法的**名字、参数的个数和类型完全相同，超类的 dance()方法在子类中**就被**隐藏**。当子类对象调用方法 dance()时，自然调用子类的 dance()方法。**超类方法在子类中隐藏称为重写或置换**。当子类中定义的成员变量和超类中成员变量同名时，超类的成员变量同样会在子类中被隐藏。子类对成员变量的隐藏和方法的重写可以把超类的状态和行为改变为自身的状态和行为。对于子类对象，如果子类重写了超类的方法，则子类对象调用这个方法时，调用子类方法。如果子类继承了超类的方法(未重写)，则会调用超类方法。

1. 多态性

参见例 3.11，程序声明表示几何形状的类 Shape，通过继承类 Shape 声明圆类 Circle 和长方形类 Rectangle.这 3 个类都定义了求面积方法 area()。另在主类 Example3_11 中定义了一个方法 returnArea()，该方法以 Shape 类对象为参数，利用参数对象求出几何图形的面积。

面向对象语言规定，**子类对象也是超类对象，凡超类对象可以应用的地方，子类对象也适用**。将子类对象交给原本处理超类对象的方法 returnArea()时，方法 returnArea()也一样能正确工作。这样，调用方法 returnArea()时，可以提供 Shape 类的对象，也可 Circle 类对象，或 Rectangle 类对象。即程序分别用 Circle 类对象 c 和 Rectangle 类对象 r 调用方法 returnArea()也能正确执行。

上述要求，给编译系统带来了一个新问题。因为方法 reaturnArea()在被调用之前，是不知道调用的参数对象具体是哪一种类型，编译暂时不能利用参数 s 的类型是 Shape,就推断代码 s.area()是调用 Shape 类的求面积方法。如是这样，就会产生错误结果。这种**编译时暂不绑定调用哪个方法，必须在运行时才绑定调用方法的技术称为动态联编**。而代码 s.area()根据 s 在执行时实际对象的类型不同，调用同名的不同方法，是面向对象语言的一种多态性。解决这种多态性必须采用动态联编技术。由于 Java 语言采用动态联编技术，保证以下程序能得到希望的结果。

有人也将调用**重载方法作为多态性之一**。重载能由调用时提供的参数个数和参数的类型顺序，在编译时就能确定被调用的方法，这种多态性**不需要**动态联编技术的支持。

[例 3.11]说明多态性的程序例子

```
class Shape{
  float area()//
  {return 0.0f;}
}
class Circle extends Shape{
     float R;
     Circle(float r)
     {
          R = r;
     }
    float area()
   {return 3.1415926f*R*R;}
}
class Rectange extends Shape{
  float W,H;
  Rectangle(float w,float h)
  {W=w;H=h;}
  float area()//
  {return W*H;}
}
public class Example3_11{
    public static void main(String args[]){
    Circle c;
    Rectangle r;
    c = new rectangle(3.0f,4.0f);
    System.out.println(“圆面积=”+returnArea(c));
    System.out.println(“长方形面积=”+returnarea(r));
  }
  static float returnArea(Shape s){
     return s.area();
  }
}
```

程序的输出结果是：

圆面积=31.415925

长方形面积=12.0

2. 多层继承

继承声明的新类，又可被其他类再继承，从而构成多层继承。参见例 3.12，程序首先声明交通工具的类(Vehicle)，再利用交通工具类，继承声明一个飞行器子类(Aircraft)；

又从飞行器(Aricraft)类继承声明两个子类，螺旋直升机(Whirlybird)类和喷气机(Jet)类。这样，Whirlybird 类和 Jet 类就是多层继承的类。

[例 3.12]一个多层继承的例子。

```
class Vehircle{
     public void start(){System.out.println(“Starting…”);}
}
class Aricraft extends Vehicle{
     public void fly(){System.out.println(“Flying…”);}
}
class Whirlybird extends Aircraft{
     public void whirl(){System.out.println(“Whirling…”);}
```

```
}
class Jet extends Aircraft{
     public void zoom(){System.out.println(“Zooming…”);}
}
```

3. 多层继承中构造方法的调用顺序

参见例 3.13，类 D 继承类 C，类 C 继承类 B，类 B 继承 A。当创建一个类 D 的对象 obj 时，由于类 D 的继承性，对象 obj 含有它所继承的各超类的成员变量，需要调用所继承各超类的构造方法。问题是这样构造方法的调用顺序是怎样的。例子说明构造方法的**调用顺序与类的继承顺序一致**，从最高层次的超类开始，按继承调用各类的构造方法，如果子类的构造方法要调用超类的构造方法，就从超类继承的成员变量初始化，用代码 super(参见 3.2.7 关于关键字 super 的叙述)，**并且代码必须是构造方法的第一条语句**。

[例 3.13] 说明多层继承中构造方法调用顺序的例子

```
class A{
  int a;
A(int a)
{this.a = a; System.out.println(“Constructing A”);}
}
class B extends A{
     int b;
     B(int a,int b)
     {super(a);this.b=b;System.out.println(“Constructing B”);}
}
class C extends B{
     int c;
     C(int a,int b,int c)
     {super(a,b);this.c = c;System.out.println(“Constructing C”);}
}
class D extends C{
     int d;
     D(int a,int b,int c,int d)
     {super(a,b,c);this.d = d;System.out.println(“Constructing D”);}
}
class Example3_13{
     public static void main(String args[]){
     D obj = new D(1,2,3,4);
     System.out.println(“对象 obj 的值是:”+obj.a+
                        “,”+obj.b+ “,”+obj.c + “,”+obj.d);
     System.out.println(“Main Program!”);
     }
}
```

程序的输出结果是：

```
Constructing A
Constructing B
Constructing C
Constructing D
对象 obj 的值是：1，2，3，4
Main Program!
```

3.2.7 Java 面向对象的其他机制

为了构造和编写面向对象程序的需要，Java 语言还引入一些非常用用的机制。本小节介绍其中对程序构造和编写比较有用的一些机制。

1. **final 类、final 方法和 final 变量**

final 的意义是**最终的**，用于修饰类、方法和变量，其意思是“**不能改变**”。**禁止改变**可能是考虑到“设计”和“效率”两个方面。

在类声明之前用 final 修饰，声明类是 final 类，final 类是**不能被再继承**的类，即它**不能再有子类**。例如，以下代码声明类 C 是 final 类：

```
final class C{
…
}
```

用 final 修饰的变量，声明该变量的值不能被改变。不能被改变的变量相当于一个常量。final 变量主要用于以下**两个方面：或是一个编译期的常数，它永远不会改变；或在初始化后，不希望它再改变**。例如，以下代码声明 FINALVAR 是一个 final 变量：

```
final int FINALVAR = 100;
```

final 局部变量在定义时可以暂不设定初值，但是一经初始化，以后就不能再改变。例如，以下代码：

```
 final int AFINAL;
… …//假设这里的代码没有对 AFINAL 赋值
AFINAL=1;//以后不能再给 AFINAL 再赋值
```

常用 final 声明常量，常量名习惯全部用大写字母命名。

2. **abstract 类和 abstract 方法**

abstract 类称为抽象类。抽象类只声明一种模板，没有具体实现代码的类。只有它的**子类才能**是有实际意义的类。所以**抽象类不可有实例**。即不能用抽象类声明或创建对象。

abstract 方法称为抽象方法。抽象方法只能出现在抽象类中，抽象方法没有实现的代码。如果一个类是抽象类的子类，则必须具体实现超类的抽象方法。**抽象类可以没有抽象方法，但有抽象方法的类一定是抽象类**。

[例 3.14]含抽象类和抽象方法的程序。

```
abstract class Shape{
 int x,y;
void MoveTo(int newx,int newy){x = newx;y=newy;}
Shape(int newx,int newy){x=newx;y=newy;}
abstract void Draw();
}
class Square extends Shape{
  int len;
   Square(int px,intpy,int l){super(px,py);len =l;}
   void Draw(){
        System.out.print(“我是正方形”);
        System.out.print(“我的中心位置是：”+”(“+x”,”+y+”),”);
        System.out.println(“我的边长是：”+len);
        //以 x,y 为中心，边长 len 的正方形
   }
}
class Circle extends Shape{
   int radius;
   Circle(int px,int py,int r){super(px,py);radius=r;}
   void Draw(){
     System.out.print(“我是圆形”);
     System.out.print(“我的中心位置是：”+”(“+x”,”+y+”),”);
     System.out.print(“我的半径是:”+radius);
     //以 x,y 为圆心，半径为 radius 的圆
   }
}
```

```
class ShapeDraw{
     void draw(Shape obj)
     {obj.Draw();}//利用动态联编，按实际对象调用相应的 Draw()方法
}
public class Example3_14{
  public static void main(String args[]){
    ShapeDraw sd = new ShapeDraw();
    Square s = new Square(10,10,5);
    Circle c = new Circle(30,30,5);
    sd.draw(s);//以 s 为实参调用 sd 的 draw 方法
    sd.draw(c);//以 s 为实参调用 sd 的 draw 方法
  }
}
```

程序的输出结果是：

我是正方形，我的中心位置是：(10，10)，我的边长是：20

我是圆形，我的中心位置是：(30，30)，我的半径是 5

3. 关键字 super

子类隐藏了超类的成员变量或者**覆盖**了**超类方法**后，利用关键字 super，子类方法可以引用超类的成员变量和被覆盖的方法。使用 super 有 3 种情况：使用 super 调用超类的构造方法，参见多层继承中构造方法的调用顺序；使用 super 调用超类被子类覆盖的方法；使用 super 访问超类被子类隐藏的成员变量。

[例 3.15]使用 super 调用超类构造方法、超类方法和超类成员变量的程序。

```
class A{
int x;
A( int inf){x = inf;}
void method(){System.out.println(“我是类 A 的方法!值是”+x);}
int getX(){return x;}
}
class B extends A{
double x;
B(int a,double b)
{
super(a);
x = b;
}
void method(){System.out.println(“我是类 B 的方法!值是”+x);}
}
class C extends B{
char x;
C(int a,double b,char c){
super(a,b);
x=c;
}
void method(){
char chCx =x;
int iAx = this.getX();
super.method();
System.out.println(“我是类 C 的方法!值是”+x);
System.out.println(“我获得的信息是：”+“chCx=”+x+chCx+”,dBx=”+dBx+”,iAx=”+iAx);
}
```

```
}
public class Example3_15{
public static void main(String args[]){
C c = new C(2,3,0,' C' );
c.method();
}
}
```

程序的输出结果是：

我是类 B 的方法!值是 3。0

我是类 C 的方法!值是 C

我获得的信息是：chCx=CC,dBx=3.0,iAx=2

4. **关键字 this**

关键字 this 用来指对象自己。this 的作用有多种，例如，将对象自己作为参数，调用别的方法；**当方法的局部变量隐藏成员变量时，利用 this 引用成员变量；在构造函数内调用另一构造函数等**。参见以下示例。

```
Class B{
 int b,c;
B(){this(2,3);}
B(int x,int y){b =x ;c=y;}
}
```

5. 类之间的 is-a 和 has-a 关系

在两个相关的类之间，例如，类 A 和类 B，可能会有 is-a 和 has-a 关系。参见例 3.16，类 A 是继承扩展类 B，则子类 A 和超类 B 之间是 is-a 关系，类 B 的所有成员类 A 也有，类 A 也是一个类 B。

[例 3.16]类 A 继承扩展类 B，类 A 和类 B 之间有 is-a 关系。

```
class B{
int b;
B(int x){b =x;}
void write(){System.out.println(“这是来自 B 的输出!”);}
}
class A extends B{
  int a;
  A(int x,int y){
    super(x);
    a = y;
    write();
    System.out.printn(“我是子类 A!” + “b=”+b+ “,a=”+a);
  }
}
public class Example3_16{
  public static void main(String args[])
  {
     a obj = new A(1,2);
  }
}
```

程序的输出结果是：

这是来自类 B 的输出!

我是子类 A!b=1,a=2

如果类 A 的某个成员变量的类型是 B，则类 A 和类 B 之间是 has-a 关系，即**类 A 包含有类 B**。例 3.17 是 has-a 关系例子，类 A 中成员变量 b 的类型是 B。

[例 3.17]类 A 的某个成员变量的类型是 B，类 A 和类 B 之间是 has-a 关系。

```
class A{
    B b;
    int a;
    A(int x,int y,int z){
        b = new B(x,y);
        a = z; b.write();
    }
}
class B{
  int b1,b2;
  B(int x,int y){
     b1= x;b2=y;
  }
  void write(){
     System.out.println(“这是来自类 B 的输出!”);
 }
}
public class Example3_17{
  public static void main(String args[]){
     A obj = new A(1,2,3);
  }
}
```

程序输出结果是:

这是来自类 B 的输出!

6. 内部类(inner class)

Java 语言也允许在类声明内部嵌套类声明。嵌套的类可以是静态的或者是非静态的。**静态类不能直接引用其包含类的成员，必须实例化**，所以不经常使用。非静态的嵌套类，因为需要进行事件处理，非常流行。参见例 3.18，类 B 在类 A 的内部声明，而且在类 A 的构造方法中实例化一个类 B 的内部对象。

[例 3.18]一个含内部类的程序

```
class A{
  B obj;
  A(){obj = new B(); obj.print();}
  class B{
    public void print(){
    System.out.println(“内部类 B…”);
    }
  }
}
public class Example3_18{
  public static void main(String args[])
  {
        A obj = new A();
  }
}
```

3.3 接口

Java 程序中的类只能继承一个类，这种继承称为单继承。Java 语言虽不允许一个类继承多个类，但允许一个类实现多个接口。接口(Interface)有与类相似的基本形式。**接口有常量定义和方法定义，但没有方法的实现代码**。可以将接口想象为一个“纯“抽象类。接口给出某种特定功能的一组方法的规范，而并未真正地实现每个方法，方法的实现要在实现这个接口的类中完成。接口也是对一些类为实现某些控制所建立的一个”协议“。例如，为键盘操作建立一个键盘操作，

定义了响应键盘操作的一组方法，给出每个方法的规范：方法的名称、返回值类型，参数个数与参数类型。实现键盘控制的一些类，加为它们对键盘操作的目的各有不同，各有不同的实现方案，但都必须按接口给定的规范给出自己的实现。形象地说，接口对实现接口的类提出这样的要求：“实现我的所有类，都应该包含像我现在这种样子，并给出方法的全部实现 ”。

类除了能继承一个类之外，还可实现多个接口。这样，对有继承并实现接口的类来说，实际上有了多个超类，实现有控制的多重继承。

接口的优点是很明显的。例如，在程序开发的早期建立一组协议而不给出具体实现，便于设计更合理的类层次。所以，接口比多继承更强、更容易控制。

1. 接口的定义

接口定义包括接口声明和接口体两部分。一般形式如下？

[public]interface 接口名 [extends 超接口表]{

接口体

}

接口名通常以 able 或 ible 结尾，意指能做什么。接口是一种只由常量定义和抽象方法组成的特殊类。public 修饰的类是公共接口，可以被所有的类和接口使用；而没有 public 修饰的接口只能被同一个包中的其他类和接口使用。

2. 接口体

接口体中的内容是一系列常量定义和方法定义。其中方法定义是一种方法声明，不提供方法的实现(没有方法体)，用分号“;”结尾。方法的实现在“使用”这个接口的各个类中，故称使用接口的类为接口的实现。以下是一个接口定义的示例：

```
interface Sleepable{
final int max=100;
void sleepSometime(int deltaT);
float  work (float x,float y);
}
```

接口 的所有变量都默认为 final static 属性；所有的方法都默认为是 public abstract 属性。

一个类通过使用关键字 implements 声明自己使用一个或多个接口。实现多个接口时，接口名之间用逗号隔开。以下示例代码说明类 Pig 继承类 Animal 并实现两个接口 eatable 和 sleepable。

```
class Pig extends Animal implements Eatable, Sleepable
{…}
```

如果一个类要实现某个接口，那么这个类必须实现该接口的所有方法，为这些方法实现的方法体。在实现接口的方法时，方法的名字、返回值类型、参数个数及类型必须与接口中的定义的方法完全一致，并一定要用 public 修饰。如果接口的方法的返回值类型不是 void,那么在实现该方法的体中至少要有一个 return 语句；如果方法不返回值，则为 void 类型，类体除了两个大括号外，可以没有任何语句。

[例 3.19]声明接口和类实现接口的例子。

```
interface Computable{
final int MAX =100;
void speak(String s);
int f(int x);
int g(int x,int y);
}
class A implements Computable{
int no;
public int f(int x){
int sum = 0;
for(int i=0;i<=x;i++){sum =sum+i;}
return sum;
}
public int g(int x,int y)
{
```

```
return x*y;
}
public void speak(String s){
}
}
class B implements Computable{
int no;
public int f(int x){
int sum =0;
for(int i=0;i<=x;i++){sum = sum+i*i;}
return sum;
}
public int g(int x,int y){return x+y;}
public void speak(String s){
}
public class Example3_19{
  public static void main(String args[]){
    A Li;
    B Tom;
    Li = new A(); Li.no =951898;
    Tom = new B();Tom.no = 951899;
    System.out.print("Li 的编号：”+Li.no+ “,最大值：”+Li.MAX);
    System.out.print(“,从 1 到 100 求和=”+Li.f(100));
    System.out.print(“,3 与 4 的积=”+Li.g(3,4));
    System.out.print(“,Tom 的编号：”+Tom.no+ “， 最大值：”+ Tom.MAX);
    System.out.print(“,从 1 平方加到 9 平方=”+Tom.f(9));
    System.out.print(“,5 与 6 的和=”+Tom.g(5,6));
  }
}
```

程序的输出结果是：

Li 的编号：951898，最大值：100，从 1 到 100 求和=5050，3 与 4 的积=12

Tom 的编号：951899，最大值：100，从 1 平方加到 9 平方=285，5 与 6 的和=11

3．4 基本类

Java 语言为一些基本类型设有对应的基本类，如表 3-2 所示。

表 3-2 基本类型与基本类

基本类型	对应的基本类	基本类型	对应的基本类
boolean	Boolean	int	Integer
char	Character	long	Long
byte	Byte	float	Float
short	Short	double	Double

基本类型值与对应基本类对象能相互转换。可以利用以下形式的基本类构造方法，由基本类型值得到对应基本类对象：

public 基本类(基本类型 v)

例如，以下代码得到一个 Integer 对象，其值是 123

```
Integer n = new Integer(123);
```

也可以**由基本类对象得到基本类型值**。实现方法如以下例子所示：

设有 Double 类对象 dObj，Boolean 类对象 bObj,代码：

```
double d = dObj.doubleValue();
```

使得 dObj 对象的值赋值给 double 类型变量 d。代码：

```
boolean b = bObj.booleanValue();
```

bObj 对象的值赋值给 boolean 类型变量 b。类似的代码可以用于其他类对象与对应的基本类型变量。

3．5 Java 小应用程序基础

Java 小应用程序即 Java Applet，是嵌入在 Web 面上供浏览器运行的程序，可放在服务器上供客户端下载使用。小应用程序的主要功能是显示图形、动画、播放声音、响应用户事件等。每个小应用程序都继承 java.applet.Applet 类。

如果小应用程序已用 import java.applet.Applet 导入系统的 Applet 类，小应用程序的主类必须按以下形式声明：

public class 主类名 extends Applet

如果没有用 import 导入系统 Applet 类，则主类的声明应写成：

public 类名 extends java.applet.Applet

为了系统的安全，对小应用程序有一定限制:**不能调用本地的应用程序和运行时的动态连接库；不能访问本地文件**，如读、写文件；**除原小应用程序所在的位置**(站点)外，**不能再做网络连接与 WWW 建立联系，不能获得任何有关系统设置的信息**。

1． 小应用程序的建立和运行

例 3．20 是一个简单的小应用程序的例子，说明小应用程序的建立和运行的步骤。

[例 3.20]说明小应用程序的建立和运行步骤的小应用程序。有关图形界面的内容参见第 5 章。类中各方法参见后面关于 Applet 类的方法的叙述。该程序启动后，如果不断地切换屏幕，文字的显示位置会往下移动。

```
import java.applet.*
import java.awt.*;
public class Example3_20 extends Applet{
  int pos;
  public void init(){pos =5;}
  public void start(){repaint();}
  public void stop(){}
  public void paint(Graphics g){
    g.drawString(“我们正在学习 java 程序设计”，20，pos+10);
    pos = (pos+20)%100+5;
  }
}
```

一个小应用程序从建立到运行需经历 4 个步骤：

(1) 用 Java 语言编写小应用程序的源程序。

小应用程序由若干类组成，在源程序中必须有一个类继承 java.applet.Applet 类，这个类是小应用程序的主类。小应用程序的主类，用 public 关键字修饰。小应用程序的源文件名必须与它的主类名完全相同。

(2) 把小应用程序的源程序编译成字节码.class 文件。

如果源文件有多个类，那么经编译后将生成多个.class 文件。

(3) 将小应用程序的字节码文件嵌入在 Web 页中，设计一个指定小应用程序的 HTML 文件.html。以下是启动小应用程序的 HTML 文件结构：

```
<HTML>
    <HEAD>
        <TITLE>String named by Programmer </TITLE>
    </HEAD>
    <BODY>
            <APPLET>
            [CODE BASE =Applet 的路径]
            CODE = Applet 的字节码文件名
            WIDTH = Applet 在 Web 页中的宽度
            HEIGHT = Applet 在 Web 页中的高度
            [<PARAM NAME =参数名 VALUE= 参数值>]
        [<!注释>][可选的辅助文本]
            </APPLET>
    </BODY>
</HTML>
```

(4) 在支持 Java 的浏览器上运行。

或用观察器 appletviewer，或用 Netscape、Hotjava、IE 等。如果采用某个 Java 开发环境。也可在开发环境下执行小应用程序。

2. Applet 类的基本方法

(1)init()方法

初始化，由浏览器载入内存时调用 init()方法，**该方法只运行一次**。其用途是创建所需要的对象、设置初始状态、装载力图像、设置参数等。

(2)start()方法

初始化后，**紧接着调用 start()方法**、启动小应用程序主线程，或当用户从小应用程序所在的 Web 页面转到其他页面，然后又**返回该页面时，start()方法将再次被调用** 。

(3)stop()方法

当**浏览器**离开小应用程序所在的页面**转到其他页面时**，调用 stop()方法，挂起小应用程序的执行。

(4)destroy()方法

当关闭浏览器时，系统调用 destroy()方法，结束小应用程序的生命，关闭线程释放对象资源。

(5) paint(Graphics g)方法

该方法用于在屏幕窗口上显示某些信息。如文字、色彩、背景和图像等。当小应用程序启动时，浏览器产生一个 Graphics 类的对象，供小应用程序使用，并以参数 g 传递给方法 paint()。小应用程序可以把 g 理解为一支画笔。

小应用程序主类的程序结构如下：

```
public class AppletName extends java.applet.Applet{
public void init(){…}
pulic void start(){…}
public void stop(){…}
public void destroy(){…}
public void paint(Graphics g){…}
…
}
```

(6) repaint()方法

为在 java 页面上显示动态效果，需要 repaint()方法，repaint()调用 update(),repaint()先清除 paint()方法以前所画的内容，然后再调用 paint()方法。

在例 3.20 的小应用程序中，start()方法调用 repaint()方法，因此每当小程序调用 start()方法时，将会导致以前用 paint()方法所画的内容消失，并紧接着再调用 paint()方法。由于 paint()方法对显示字符串的位置有修改，这样每当调用 start()方法时，字符串的显示位置会移动。

(7) update()方法

调用 repaint()方法时，系统会自动调用 update()方法。update(Graphics g)方法是从父类 Applet 继承来的，该方法的功能是清除 paint()方法以前所画的内容，然后再调用 paint()方法。小应用程序重写 update()方法，可以达到特定的要求。参见 7.2.1 例 7.3。

3. 小应用程序的生命周期(执行过程)

(1)下载并实例化小应用程序，系统调用 init()方法。

(2)启动，系统调用 start()方法。

(3)小应用程序暂停执行,或因 Browser 转向其他网页,或小应用程序调用 stop()方法,或小应用程序执行 paint(Graphics g)方法

(4)重新启动，或重新进入小应用程序所在页面，重新激活小应用程序，或小应用程执行 paint(Graphics g)方法完成。

(5)终止，或退出浏览器，或执行 destroy()消亡方法。

当浏览器打开超文本文件，例如 Example3_20.html,发现有 APPLET 标记时，就创建主类 Example3_20 对象，该对象是小应用程序主类的实例，具体表现是一个视窗。视窗是一个容器，它的大小由超文本中的 width 和 height 来确定，参见第 5 章。

4. 小应用程序数据输入输出

小应用程序的数据输入有多种方式。从图形界面的文本框、文本区输入；也可以采用系统标准输入；还可以由 HTML 文件

中的参数导入。
这里只小应用程序从它的 HTML 文件导入 参数的方法。设小应用程序有以下成员变量定义：
int sleepTime;String filename;
该小应用程序的 HTML 文件 用以下形式的代码指定小应用程序主类成员变量 sleepTime 与 fileName 的值：
<PARAM NAME = “sleepTime” VALUE= “1000”>
<PARAM NAME = “filename” VALUE = “mypic.jpg”>
小应用程序在 init()方法中可帮以下代码读取上述两个成员变量的值：
String s = getParameter(“sleeptime”);
sletpTime = Integer.parseInt(s);
fineName = getParameter(“filename”);
小应用程序在窗口中输出数据，需要重新设计 paint()方法。该方法主要调用 drawstring()方法。调用 drawString()方法的格式如下：

```
g.drawString(string,xPos,yPos);
```

参数 string 是要输出的字符串，xPos 和 yPos 是字符串在输出窗口的像素坐标。

习题
3．1 什么是面向对象技术？它有什么优点？
3．2 面向对象的程序设计与面向过程的程序设计有什么区别？
3．3 在程序中类和对象有什么区别？
3．4 举例说明类方法和实例方法，以及类变量和实例变量的区别。什么情况下用实例变量？什么情况下用类变量？
3．5 子类继承超类的哪些成员变量和方法？
3．6 子类在什么情况下能隐藏超类的成员变量和方法？
3．7 在子类中是否允许有一个方法和超类的方法名字相同，而类型不同？
3．8 试说出对象、类、继承和多态性的概念。
3．9 为什么要定义抽象类？为什么抽象类没有具体的对象？
3．10 试比较接口和抽象类的异同。使用接口有哪些注意事项？
3．11 指出 Applet 的程序结构及各方法的作用。
3．12 以下程序有什么错误？

3．13 试声明一个复数类 Complex，要求该类提供：由已知实部和虚部构造复数的构造方法；复数与实数和复数与复数的四则运算方法复数对象的实部、虚部的方法；输出复数等方法。
并要求编写一个应用程序实现对复数类的完整测试。

```
class Float //实数类
{
 float m;
 Float(){m=0;}        //无参构造函数
 Float(float n){m=n;}//有参构造函数
 public String toString()//将实数 m 转换为字符串
 {return (String.valueOf(m));}
}
class Complex //复数类
{
 float real;
 float imag;
 Complex(){real=0;imag=0;}   //无参构造函数
 Complex(float r,float i){real=r;imag=i;};}//有参构造函数
 float getReal(){return real;}//取复数的实部
 float getImag(){return imag;}//取复数的虚部
 Complex complexAdd(Complex x)//复数相加
```

```
{ Complex temp=new Complex();//创建临时复数
 temp.real=real+x.real;//实部相加
 temp.imag=imag+x.imag;//虚部相加
 return temp;
}
Complex complexFloatAdd(Float x)//复数加实数
{
 Complex temp=new Complex();
 temp.real=real+x.m;
 temp.imag=imag;
 return temp;
}
Complex complexSubtract(Complex x)//复数相减
{
 Complex temp=new Complex();
 temp.real=real-x.real;
 temp.imag=imag-x.imag;
 return temp;
}
Complex complexFloatSubtract(Float x)//复数减实数
{
 Complex temp=new Complex();
 temp.real=real-x.m;
 temp.imag=imag;
 return temp;
}
Complex complexMulti(Complex x)//复数相乘
{
 Complex temp=new Complex();
 temp.real=real*x.real-imag*x.imag;
 temp.imag=imag*x.real+real*x.imag;
 return temp;
}
Complex complexFloatMulti(Float x)//复数乘以实数
{
 Complex temp=new Complex();
 temp.real=real*x.m;
 temp.imag=imag*x.m;
 return temp;
}
Complex complexDivide(Complex x)//复数相除
{
 Complex temp=new Complex();
 temp.real=(real*x.real+imag*x.imag)/(x.real*x.real+x.imag*x.imag);
 temp.imag=+(imag*x.real-real*x.imag)/(x.real*x.real+x.imag*x.imag);
 return temp;
}
Complex complexFloatDivide(Float x)//复数除以实数
{
```

```
  Complex temp=new Complex();
  temp.real=real/x.m;
  temp.imag=imag/x.m;
  return temp;
 }
 public String toString()//输出复数
 {if(imag>0)
     return (real+"+"+imag+"i");
  else if(imag<0)
       return (String.valueOf(real)+imag+"i");
  else
       return (String.valueOf(real));
   }
}
public class ComplexTest
{
 public static void main(String[]args)
 {
          Complex a,b,c;
   a=new Complex(1,3);
   b=new Complex(2,-3);
   c=a.complexAdd(b);
   System.out.println("("+a.toString()+")+("+b.toString()+")="+c.toString());
   c=a.complexSubtract(b);
   System.out.println("("+a.toString()+")-("+b.toString()+")="+c.toString());
   c=a.complexMulti(b);
   System.out.println("("+a.toString()+")*("+b.toString()+")="+c.toString());
   c=a.complexDivide(b);
   System.out.println("("+a.toString()+")/("+b.toString()+")="+c.toString());
   Float d;
  d=new Float(4);
  c=a.complexFloatAdd(d);
  System.out.println("("+a.toString()+")+"+d.toString()+"="+c.toString());
  c=a.complexFloatSubtract(d);
  System.out.println("("+a.toString()+")-"+d.toString()+"="+c.toString());
  c=a.complexFloatMulti(d);
  System.out.println("("+a.toString()+")*"+d.toString()+"="+c.toString());
  c=a.complexFloatDivide(d);
  System.out.println("("+a.toString()+")/"+d.toString()+"="+c.toString());
  System.out.println("real="+c.getReal()+"   "+"imag="+c.getImag());
   }
}
```

3.14 度声明一个有理数 RationalNum,要求提供有理数的四则运算、以分数形式输出有理数和以浮点数形式输出浮点 等方法。

```
class RationalNum
{
 private int mole,deno;
 RationalNum(){mole=1;deno=1;}            //无参构造函数，因 deno 是分母，所以 deno 不能为 0
 RationalNum(int x,int y){mole=x;deno=y;}//有参构造函数
```

```
RationalNum rationalNumAdd(RationalNum r)//有理数加法方法
{
 RationalNum temp=new RationalNum();//创建临时有理数 temp
 temp.mole=mole*r.deno+deno*r.mole;//如 3/4+5/7 的分子应为 3*7+5*4，分母为 4*7
 temp.deno=deno*r.deno;
 int c=greatestCommonDivisor(temp.mole,temp.deno);//取最大公约数
 temp.mole=temp.mole/c;//最简分数的分子
 temp.deno=temp.deno/c;//最简分数的分母
 return temp;
}
RationalNum rationalNumSubtract(RationalNum r)
{
 RationalNum temp=new RationalNum();//创建临时有理数 temp
 temp.mole=mole*r.deno-deno*r.mole;//如 3/4+5/7 的分子应为 3*7+5*4，分母为 4*7
 temp.deno=deno*r.deno;
 int c=greatestCommonDivisor(temp.mole,temp.deno);//取最大公约数
 temp.mole=temp.mole/c;//最简分数的分子
 temp.deno=temp.deno/c;//最简分数的分母
 return temp;
}
RationalNum rationalNumMulti(RationalNum r)
{
 RationalNum temp=new RationalNum();//创建临时有理数 temp
 temp.mole=mole*r.mole;//如 3/4+5/7 的分子应为 3*7+5*4，分母为 4*7
 temp.deno=deno*r.deno;
 int c=greatestCommonDivisor(temp.mole,temp.deno);//取最大公约数
 temp.mole=temp.mole/c;//最简分数的分子
 temp.deno=temp.deno/c;//最简分数的分母
 return temp;
}
RationalNum rationalNumDivide(RationalNum r)
{
 RationalNum temp=new RationalNum();//创建临时有理数 temp
 temp.mole=mole*r.deno;//如 3/4+5/7 的分子应为 3*7+5*4，分母为 4*7
 temp.deno=deno*r.mole;
 int c=greatestCommonDivisor(temp.mole,temp.deno);//取最大公约数
 temp.mole=temp.mole/c;//最简分数的分子
 temp.deno=temp.deno/c;//最简分数的分母
 return temp;
}
int greatestCommonDivisor(int a,int b)//求最大公约数
{
 int c1=Math.abs(a);//取绝对值是为了不影响分子分母的符号
 int c2=Math.abs(b);
 int c3=0;
 //以下为用辗转相除法求最大公约数
 if(c1<c2)
 {
  c3=c1;c1=c2;c2=c3;
```

```
  }
  while(c1%c2!=0)
  {
   c3=c2;
   c2=c1%c2;
   c1=c3;
  }
  return c2;
 }
 public String toString1()
 {if((mole<0&&deno<0)||(mole>0&&deno>0))
  return (Math.abs(mole)+"/"+Math.abs(deno));
  else
   return ("-"+Math.abs(mole)+"/"+Math.abs(deno));
 }
  public String toString2()
 {
  return String.valueOf((float)mole/deno)+"f";
 }
}
public class RationalNumTest
{
 public static void main(String[]args)
 {
  RationalNum m,n,t;
   m=new RationalNum(-1,2);
   n=new RationalNum(-8,-9);
   t=m.rationalNumAdd(n);
   System.out.println("以分数形式输出有理数："+"("+m.toString1()+")+("+n.toString1()+")="+t.toString1());
   System.out.println("以浮点数形式输出浮点数："+"("+m.toString2()+")+("+n.toString2()+")="+t.toString2());
   t=m.rationalNumSubtract(n);
   System.out.println("以分数形式输出有理数："+"("+m.toString1()+")-("+n.toString1()+")="+t.toString1());
   System.out.println("以浮点数形式输出浮点数："+"("+m.toString2()+")-("+n.toString2()+")="+t.toString2());
   t=m.rationalNumMulti(n);
   System.out.println("以分数形式输出有理数："+"("+m.toString1()+")*("+n.toString1()+")="+t.toString1());
   System.out.println("以浮点数形式输出浮点数："+"("+m.toString2()+")*("+n.toString2()+")="+t.toString2());
   t=m.rationalNumDivide(n);
   System.out.println("以分数形式输出有理数："+"("+m.toString1()+")/("+n.toString1()+")="+t.toString1());
   System.out.println("以浮点数形式输出浮点数："+"("+m.toString2()+")/("+n.toString2()+")="+t.toString2());
   }
}
```

/**用辗转相除法求最大公约数

用辗转相除法求两个数的最大公约数的步骤如下：

先用小的一个数除大的一个数，得第一个余数；

再用第一个余数除小的一个数，得第二个余数；

又用第二个余数除第一个余数，得第三个余数；

这样逐次用后一个数去除前一个余数，直到余数是 0 为止。那么，最后一个除数就是所求的最大公约数（如果最后的除数是 1，那么原来的两个数是互质数）。

例如求 1515 和 600 的最大公约数，

第一次：用 600 除 1515，商 2 余 315；

第二次：用 315 除 600，商 1 余 285；

第三次：用 285 除 315，商 1 余 30；

第四次：用 30 除 285，商 9 余 15；

第五次：用 15 除 30，商 2 余 0。

1515 和 600 的最大公约数是 15。

辗转相除法是求两个数的最大公约数的方法。如果求几个数的最大公约数，可以先求两个数的最大公约数，再求这个最大公约数与第三个数的最大公约数。这样依次下去，直到最后一个数为止。最后所得的一个最大公约数，就是所求的几个数的最大公约数。

```
int   f(int   m,int   n)
  {
    int   tempm=m;
  int   tempn=n;
  int   temp=0;
  if(tempm<tempn)   {temp=tempm;tempm=tempn;tempn=temp;}
  //下面使用辗转相除法
          while(tempm%tempn!=0){
          temp=tempn;
          tempn=tempm%tempn;
          tempm=temp;
      }
  return   tempn;
  }

*/
```

第 4 章 数组和字符串

本章内容(次重点：10%)

4．1 数组

数组是将一组相同类型的数据顺序存储，组成一种复合数据类型。数组的特点主要是：一个数组中的所有元素的数据类型相同，数组中的元素连续顺序存储，数组中每个元素按存储顺序对应一个下标，下标从 0 开始顺序编号，引用数组元素通过数组名和它的下标确定。数组有一维数组和多维数组之分，数组元素的下标个数确定数组的维数。

4．1．1 一维数组

数组元素只有一个下标的数组是一维数组。

1. 声明一维数组

在 Java 语言中，数组是一种引用类型，数组名如同对象名一样，是一种引用。声明数组是声明一个标识符能引用数组，只要指定数组元素的类型。声明一维数组有两种格式：

数组元素类型 数组名[]； 或者 数组元素类型[] 数组名；

```
float boy[];double girl[];char cat[];
Student stu[];//与 Student[] stu;  等价
```

2. 创建一维数组

创建一维数组就是为数组分配存储空间，需指定数组长度。数组创建后，数组名就标识用于存储数组元素的存储空间。创建一维数组的方法有 3 种：

（1） 先声明后创建

创建数组代码的一般形式为

```
数组名字 = new 数组元素类型[元素个数];
```

例如，代码：

```
int  intArray [];
intArray = new  int[5];
```

（2） 声明时创建

把声明数组和创建数组的代码结合在一起。例如，代码：

```
int  intArray[] = new int[5];
```

（3） 直接赋值创建

在声明数组时，直接给出数组的初值。例如代码：

```
int [] intArray = {1,2,3,4,5};
```

直接赋值创建数组的方法也称为数组初始化。在数组初始化的代码中，花括号{}，内初值间的分隔符是逗号”,”。系统根据数组初始化时，提供的初值个数确定数组的元素个数，然后为数组分配空间，并为数组各元素依次赋初值。

上述代码相当于：

```
int  intArray[] = new int[5];
intArray[0]=1; intArray[1]=2; intArray[2]=3; intArray[3]=4; intArray[4]=5;
```

前两种方法创建数组时，系统会给数组元素提供默认初始值：数值型数组元素的默认值是 0；char 类型数组元素的默认值是全 0 代码字符；布尔类型数组元素的默认值是 false。

在 Java 中，所有数组都有一个成员 **length**，程序可以访问它，获得数组的元素个数，但不能改变它。例如，按照前面数组 intArray 的定义，intArray.length 的值是 5 。

3. 引用一维数组元素

引用一维数组元素代码的一般形式为

```
数组名[下标]
```

数组下标可以是整型常数，或者是整型表达式。例如：

```
intArray[3]=25;
```

数组下标范围是 0 到数组名.length-1。为了确保程序安全性，Java 运行系统对数组下标自动进行范围越界检查。如果发现下标越界，系统将发生异常。

4. 数组是一种引用类型

数组也是一种引用类型，一个数组名在不同时间可引用不同数组。参见例 4.2。代码：

```
myArray =firstArray;
```

```
    myArray = secondArray;
```

4．1.2 多维数组

数组元素有多个下标的数组就是多维数组。多维数组是一种数组的数组，例如，当数组的元素又是一维数组时，就是一个二维数组。在 Java 程序中，可以有三维数组，或四维数组等。以下以二维数组为例说明多维数组声明、创建和引用的方法。

1. 声明二维数组

声明二维数组的一般形式有以下 3 种：

```
    类型  数组名 [ ][ ]
    类型 [ ][ ] 数组名
    类型 [ ] 数组名 [ ]
```

类似的代码可以声明多维数组。

2. 创建二维数组

创建二维数组对象的方法有多种：

（1） 直接分配（平衡二维数组－－矩阵）

类型 数组名[][]=new 类型[长度 1][长度 2]

例如，以下代码创建一个 3 行 3 列的矩阵：

```
int myTowArray[][]= new int[3][3];
```

（2） 从最高维开始，分别对每一维分配不等长的空间（非平衡数组）

以二维数组为例，先指定第一维，创建有指定子数组个数的二维数组然后，依次对每个子数组确定元素个数，并创建子数组。例如，以下代码创建一个二维数组 a[][]，它有 3 个子数组，第一个子数组有 4 个元素，第 2 个子数组有 5 个元素，第 3 个子数组有 2 个元素：

```
 int a[][] = new int[3];
a[0]= new int [4];
a[1]=new int[5];
a[2]=new int[2];
```

（3） 直接赋值创建

声明二维数组，同时直接给出各子数组的元素。如果子数组的元素个数不同，则创建是一个非平衡的二维数组。例如，代码：

```
int [][]x = {{5,6,7},
         {8,9,10,11},
         {18,19,20,15},
         {2,9}
};
```

3. 引用二维数组

二维数组元素的引用方式是：

```
    数组名[下标 1 ][下标 2 ]
```

其中下标 1 是第一维下标，下标 2 是第二维下标。例如，对上述二维数组 x,x[0][1]是 6 ， x[2][3]是 15。

4. 使用二维数组要注意的问题

对于二维数组，要注意以下两点：

（1） 二维数组名，length 和二维数组名[下标 1].length 的区别。二维数组名.length 是二维数组的子数组个数；二维数组名[下标 1].length 是指定子数组的元素个数。

（2） 二维数组名[下标]是一维数组。

[例 4.3]一个含三角二维数组的应用程序

```
public class Example4_3{
public static void main(String args[]){
    boolean bTb1[][]= new boolean[4][];
for(int i=0;i<bTb1.length;i++)
{
    bTb1[i] = new boolean[i+1];
```

```
}
for(int i=0;i<bTb1.length;i++){
    for(int k=0;k<bTb1[i].length;k++){
        System.out.print(bTb1[i][k]+"  ");
    }
    System.out.println("");
}
}
}
```

运行以上程序，输出结果是：

```
false
false false
false false false
false false false false
```

4．2字符串

字符串是由0个或多个字符组成的序列，Java语言提供两个用于处理字符串的类：

String类用于处理**不可变的字符串**。**StringBuffer**类用于处理**可改变的字符串**。

4．2．1字符串常量

字符串常量是用**双引号括起来的一个字符串**。例如，“我正在学习Java程序设计”、”123.456”。在Java语言中，字符串常量是匿名的String对象。

4．2．2　字符串声明和创建

如同一般用类声明和创建对象一样，可用String类声明和创建字符串。例如，以下代码声明字符串：

```
String s;
```

利用String类的构造方法可以创建字符串对象，String类的构造方法有：

（1） String()，创建一个空字符串对象

例如：

```
s = new String();
```

（2） String(String str)，根据给定的字符串str创建字符串对象

例如，利用字符串常量创建字符串：

```
s = new String("I am a student.");
```

或简写成：

```
 s = "I am a student.";
```

字符串声明和创建也可一起完成：

```
    Sting s = new String("I am a student.");
```

以下代码用字符串对象创建新字符串：

```
    String newStr = new String(s);
```

以上代码使字符串newStr与字符串s的内容相同，但是，它们是两个字符串。

（3） String(char a[])，根据字符数组a创建一个字符串。

例如，代码：

```
char charArray[4] = { 'g' ,' I' ,' r' ,' l' };
String str = new String(charArray);
```

所生成的字符串内容是”girl”。

（4） String (char a[], int startIndex, int charNum)，根据字符数组的起始下标位置以及字符个数创建字符串。

例如，以下代码输出3456。

```
char a[] = { '1' ,' 2' ,' 3' ,' 4' ,' 5' ,' 6' ,' 7' };
String s = new String (a,2,4);
System.out.println(s);
```

4．2．3字符串基本操作

Java 语言为便于程序使用字符串，为字符串提供非常丰富的操作，本书只介绍其中一些常用的字符串操作。

(1) 字符串连接(+ 或 concat())

字符串有一个**连接运算符+**，得到连接两个字符串的结果；一个连接方法 concat (String s)，实现复制参数 s 字符串的内容，连接在字符串对象之后，得到一新的字符串。

例 4.4 是说明字符串连接运算的应用程序。

[例 4.4] 一个说明字符串连接运算和连接方法的应用程序。

```
public class Example4_4{
  public static void main(String args[]){
      String s1 = “ABC”;
      String s2 = “DEFG”;
      System.out.println(s1+s2);
      s1 = “ABC”;
      s2 = “XYZ”;
      s1 = s1.concat(s2);
      System.out.println(s1);
  }
}
```

程序输出：

```
ABCDEFG
ABCXYZ
```

(2) 获取字符串的长度 length()

length()方法可以获取一个字符串长度。例如，以下代码输出 4 。

```
String s = “我是学生”;
int strlen = s.length();
System.out.println(strlen);
```

字符串常量也可以使用 length()方法获得长度。例如：”我是好学生”.length()的值是 5 。

(3) 字符串前缀或后缀的比较

用 boolean **startsWith**(String s)方法**判断一个字符串的前缀**是否为字符串 s，例如：

```
String str1 = “220302620629021”;
String str2 = “21079670924022”;
```

str.startsWith(“ 220”);的值是 true；str2.startsWith(“ 220”)的值是 false。

用 boolean **endsWith**(String s)方法**判断一个字符串的后缀**是否为字符串 s，例如：

```
String str1 = “220302620629021”;
String str2 = “21079670924022”;
```

str1.endsWith(“ 021”)的值是 true；str2.endsWith(“ 021”)的值是 false。

(4) 比较两个字符串是否相同

用 boolean equals(String s)方法比较某字符串是否与字符串 s 相同。例如：

```
String str1 = “abc”;
String str2 = “Abc”;
String str3 = “abc”;
String str4 = new String(“abc”);
```

str1.equals(str3)的值是 true。str1.equals(str2)的值是 false，因为两字符串的第一个字母不同。

注意两字符串比较与两个字符串引用对象比较的区别。比如，表达式 str1.equals(str2)与表达式 str1==str2 的差异。前者表示所引用的两个字符串的内容是否相同，后者表示 str1 与 str2 是否引用同一个对象。在以上 4 行代码中，由于 str1 和 str2 分别引用不同的对象,所以 str1==str2 的值是 false;因为 str1 和 str3 引用同一个字符串常量,str1==str3 的值是 true。尽管 str1 与 str4 引用的对象的内容相同，因为它们引用的是不同对象，str1 引用的是编译时能确定的字符串常量，str4 引用的是程序运行时创建的对象，所以表达式 str1==str4 的值是 false。

用 boolean **equalsIgnoreCase**(String s)方法，忽略大小写比较某字符串是否与字符串 s 相同。例如：对于前面的字符串 str1 和 str2，str1.equalsIgnoreCase(str2)的值是 true。

用 int compareTo(String another)方法按字典顺序与参数 another 字符串比较大小。代码 s1.compareTo(s2)，如果 s1 和 s2 相同，方法返回值 0；如果 s1 大于 s2，方法返回正值；如果 s1 小于 s2，方法返回负值。

用 int compareToIgnoreCase(String another)方法，忽略大小写，按字典顺序与参数 another 字符串比较大小。

(5) 字符串检索

用 String 类中方法

int indexOf(String s)或

int indexOf(String s,int startpoint)

实现字符串检索。前一个方法是从指定字符串的头开始检索参数字符串 s，返回字符串 s 首次出现的位置。后一个方法则在指定字符串中从某个位置开始检索参数字符串 s,返回字符串 s 首次出现的位置。**如果没有检索到。则返回-1**。例如：

```
String s=”ABCDEFGHIJABC”;
s.indexOf(“C”);//值是 2
s.IndexOf(“EFG”,2);//4
s.indexOf(“A”,7);//值是 10
s.IndexOf(“D”,4);//-1
```

(6) 取字符串的子串

使用 String 类中的方法：

String substring(int startpoint);

String substring(int startpoint, int end);

其中 **startpoint 是字符串的开始下标，end 是截止下标**。子串是从开始下标开始，至截止下标的前一个下标为止范围内的子串。例如，以下代码的结果如注释所示。

```
String s =”0123456789”,s1,s2;
s1 = s.substring(2);//s1  是 23456789
s2 = s.substring(2,5);//s2 是 234
```

(7) 替换字符串的某字符得到一个新字符串

使用 String 类中的方法

String replace(char oldChar,char newChar)

用参数 newChar 指定的字符替换 s 中由 oldChar 指定的所有字符，产生一个新的字符串。例如，代码

```
String s1 = “1234567788”;
String s2 = s1.replace(“7”,”A”);
```

创建两个 String 对象，s1 为”1234567788”,s2 为”123456AA88”。参见例 4.5。

(8) 去掉前后空格得到一个新字符串

方法 trim()可以去掉字符串的前后空格。参见例 4.5。

[例 4.5] 一个说明字符串的字符替换和去掉字符串前后空格的应用程序。

```
public class Example4_5{
public static void main(String args[]){
String s =”1234567788”,str;
str = s.replace(‘7’,’A’);
System.out.println(“s=”+s+” str=”+str);
String s2= “ 1234567788”,str2;
str2 = s2.trim();
System.out.println(“s2=”+s2+”|str2=”+str2+”|”);
}
}
```

程序输出

```
s=1234567788  str = 123456AA88
s2=  123456  77  88  |  str2=123456  77  88|
```

注意：方法 trim()不能改变字符串自己，而是产生一个新的字符串。

(9) 基本类型值的字符串表示

例如，设有 double 类型变量 d，boolean 类型变量 b，代码：

String dStr = String.valueOf(d);
String bStr = String.valueOf(b);
分别将数值和逻辑值转换成字符串。

(10) 字符串得到基本类型对象

字符串能转换成基本类对象，例如，代码：
Double dObj = Double.valueOf(“1234.567”);
将字符串转换成 Double 类的对象。类似地，可以将字符串转换到其他基本类对象。

(11) 字符串得到基本类型值

1）字符串转化为整型值

字符串转化为 int 类型值用 Integer.parseInt(String s)方法。例如：
int x;String s = “6542”;x= Integer.parseInt(s);
字符串转化为 long 类型值用 Long.parseLong(String s)方法。例如：
long x;String s =”46046542”;x = Long.parseLong(s);

2）字符串转化为实型值

字符串转化为 float 类型值用 Float.parseFloat(String s)方法。例如：
float f = Float.parseFloat(“12.35”)，使 f 的值为 12.35。
字符串转化 double 类型用代码 Double.parseDouble(String s)方法。例如：
double d = Double.parseDouble (“12.35”)，使 f 的值为 12.35。

4.2.4 StringTokenizer 类

java.util 包中的类 StringTokenizer 用于语言符号（单词）的分析。例如，如果把空格作为分隔符，字符串“We are learning java programming” 有 5 个单词。要分析出字符串中的单词，要用 StringTokenizer 类的构造方法，由给定的字符串构造一个 StringTokenizer 对象，StringTokenizer 对象称作字符串分析器。然后利用一个单词方法 nextToken()，结合循环，从字符串分析器中逐一取出单词。用 hasMoreTokens()方法控制循环，只要字符串中还有语言符号，该方法就返回 true，否则返回 false。调用 countTokens()方法能得到字符串分析器中一共有多少个单词。

StringTokenizer 类有两个常用构造方法：
StringTokenizer(String s)，为字符串 s 构造一个分析器。使用默认的分隔符集合，即空格符（若干个空格被看作一个空格）、换行符、回车符、Tab 符、进纸符。
StringTokenizer（String s,String delim），为字符串 s 构造一个分析器，以字符串参数 delim 中的字符作为分隔符。

4.2.5 字符串与字节数组

(1) String (byte[]b)
(2) String(byte []b,int offset,int length)

String 也提供实例方法 getBytes()：
byte [] getBytes(),使用默认字节字符对应表将字符串转化为字节数组。

4.2.6 对象的字符串表示

Object 类有方法 String toString(),一上对象通过调用该方法可以获得该对象的字符串表示。

1.举例说明如何声明、创建和初始化数组。
如声明数组：int a[] 或 int[]a
创建数组：a=new int[]
初始化 int a[]={1,2,3,4,5}

2.举例说明如何声明、创建和初始化数组。
如声明多维数组：int a[][] 或 int[]a[] 或 int[][]a
创建数组：a=new int[][]
初始化 int a[][]={{1,2,3,4,5}
{6,7,8,9,10}}

3.一个数组能够存储不同类型的元素吗?

答：不能。因为数组是将一组相同类型的数据顺序存储，组成一种复合数据类型。

4.4 答案

```
public class IntArrayTest
{
 public static void main(String args[])
 {
  int intArray[][]={{1,2,9,10,25},
                    {4,3,8,11,24},
                    {5,6,7,12,23},
                    {16,15,14,13,22},
                    {17,18,19,20,21}};
  for(int i=0;i<=4;i++)
    {
      for(int j=0;j<=4;j++)
       {
        if(intArray[i][j]/10==0)
        System.out.print(" "+intArray[i][j]+"    ");
        else
        System.out.print(intArray[i][j]+"    ");
       }
       System.out.println("");
      }
  }
}
```

4.5 java 中的字符数组与字符串有什么区别?

答：字符数组是由类型为 char 的字符元素组成的，每个元素位置存储一个字符元素，对于数组名为 s 的数组，用 char s[] 表示。字符串是由 0 个或多个字符组成的序列，是一个对象，对于对象名为 s 用 String s 表示。

4.6 确定一个字符数组长度与确定一个 String 对象的长度有什么不同?

答：确定一个字符数组的长度用数组名.length,而确定一个 String 对象的长度用对象名.length()

4.7 此题紧扣的是必须用题中的方法实现大小写转换

```
public class UpperAndLowerTest
{
 public static void main(String args[])
 {
     String s1=("ABCdefgHIJkhl123"),s2,s3;
     System.out.println("未转换时的字符串 s1="+s1);
     byte t1[]=s1.getBytes();//将字符串 s1 转换成字节数组 t1
     s2=s1.toUpperCase();//将字符串 s1 的字母全部转换成大写并保存到 s2
             s3=s1.toLowerCase();//将字符串 s1 的字母全部转换成小写并保存到 s3
             byte t2[]=s2.getBytes();//将字符串 s2 转换成字节数组 t2
     byte t3[]=s3.getBytes();//将字符串 s1 转换成字节数组 t3
             for(int i=0;i<t1.length;i++)
     {
       if(t1[i]!=t2[i])//如果 t1[i]不是大写字母而是小写字母
           t1[i]=t2[i];//将小写字母转换成大写字母
       else //如果是大写字母
   t1[i]=t3[i];//将大写字母转换成小写字母
     }
```

```
            s1=new String(t1);//将字节数组 t1 转化成字符串保存到 s1 中
            System.out.println("进行字母大小写转换后的 s1="+s1);
 }
}
```

4.8

```
public class ConcatTest
{
 public static void main(String args[])
 {
  String str1="abcd";
  String str2="efgh";
  String str3=str1.concat(str2);
  System.out.println("str3="+str3);
 }
}
```

4.9 试利用 java.util.Date 类继承声明 MyDate 类增加以下功能：增加 1 天、增加 1 月和增加 1 年的方法；输出 MyDate 对象日期的方法；求两个 MyDate 对象日期差的方法。并提供能用当前日期初始化 MyDate 类对象的构造方法。同时编写应用程序能实现对 MyDate 对象的测试。

```
import java.util.*;
class MyDate extends Date
{
 Date x;
 MyDate()//用当前日期初始化 MyDate
 {
  x=new Date();
 }
  MyDate( int year,int month,int date1)//按输入的日期初始化 MyDate
  {
   x=new Date(year-1900,month-1,date1);//系统会在输入的日期上自动加 1900 所以要减，月份是 0 对应 1 月份，所以对应的月份应减 1
  }
 void addDate()//增加一天
 {
  x.setDate(x.getDate()+1);
 }
 void addMonth()//增加一月
 {
  x.setMonth(x.getMonth()+1);
 }
 void addYear()//增加一年
 {
  x.setYear(x.getYear()+1);
 }
 void getMyDate()//输出 MyDate 对象日期,并利用 substring()去掉 Date 的时间
 {String r=x.toString();
  String r1=r.substring(0,11);
  String r2=r.substring(r.indexOf('C'));
   String r3=r1+r2;
  System.out.println(r3);
 }
```

```
 void dateSubtract(Date t2)//两个 MyDate 对象日期差
 {
  long day=(x.getTime()-t2.getTime())/(1000*60*60*24);
  System.out.println("两个日期相差:"+day);
  }
}
public class DateTest
{
  public static void main(String[]args)
 { MyDate today,nextday;
  nextday=new MyDate(2008,9,17);
  nextday.getMyDate();
   today=new MyDate(1998,11,30);
   today.getMyDate();
  today.addDate();
  System.out.print("today 对象增加一天后的日期：");
   today.getMyDate();
   today.addMonth();
   System.out.print("today 对象增加一月后的日期：");
   today.getMyDate();
   today.addYear();
   System.out.print("today 对象增加一年后的日期：");
   today.getMyDate();
   nextday.dateSubtract(today.x);
 }
}
```

4.10 求前 n 个质数。要求确定数 m 是否是质数，用早先求出的质数对 m 的整除性来确定。

```
public class PrimeTest
{
 public static void main(String[]args)
 {
  int n=40;
  int primeArray[];
  primeArray=new int[n];
      primeArray[0]=2;
  int m=2;
  int i=0;
              int s=0;
  while(i<n)
    { m=m+1;
     for(int t=0;t<=i;t++)
      {
        if(m%primeArray[t]==0)
          { s=0;break;}
         else
          {s=m%primeArray[t];}
      }
         if(s!=0)
```

```
            {
             primeArray[++i]=m;
             if(i==n-1)break;
               }
      }
      System.out.println("前"+n+"个质数是：");
      for(i=0;i<n;i++)
      {
                 if(i%20==0)System.out.println();
        System.out.print(primeArray[i]+"  ");
               }
  }
```

4.11 *编写实现从两字符串中找出最长的相同字符列的代码*

```
public class StringCompareTest
{
 public static void main(String[]args)
 {String str1="abcdefghijkABCD";
  String str2="ilmabnopqefghrstABCD";
   String str3,str4,t1,t2,t3="";
  int i,j,k;
  int len1=str1.length();
  int len2=str2.length();
  int len3;
  if(len1>len2)//确定使 str3 字符串的长度小于或等于 Str4 的长度
  {len3=len1;len1=len2;len2=len3;str3=str2;str4=str1;}
  else
  {str3=str1;str4=str2;}
  for( i=len1;i>=0;i--)//通过这个 for 嵌套循环找出一个最长相同字符列
   {
      for( j=0;j<=i;j++)
      {
       for( k=0;k<=len2;k++)
         {
          t1=str3.substring(j,i);
          if((len2-k)<(i-j))break;
          t2=str4.substring(k,k+i-j);
          if(t1.equals(t2)&&t1.length()>t3.length())
           {t3=t1;break;}
          }
       }
    }
  System.out.println("两字符串中最长的相同字符列代码是:");
  //根据上一个循环确定的最长相同字符列的长度找出两字符串中所有的最长相同字符列
  for(i=len1;i>=0;i--)

   {
      for(j=0;j<=i;j++)
      {
```

```
        for(k=0;k<=len2;k++)
         {
          t1=str3.substring(j,i);
          if((len2-k)<(i-j))break;
          t2=str4.substring(k,k+i-j);
          if(t1.equals(t2)&&t1.length()==t3.length())
          System.out.println(t1);
         }
      }
    }
 }
}
```

4.12 整理字符串，将字符串的前导空白符和后随空白符删去，并将字符串中非空白符列之前的连续的多个空白符只保留一个，而去掉多余的空白符。

```
import java.util.*;
public class StringMangeTest
{
 public static void main(String[]args)
 {
   String str1="   123456    77    88 abdde  fg  ";
   System.out.println("整理前的字符串 str1="+str1);
   //String str2=str1.trim();
   StringTokenizer pas=new StringTokenizer(str1," ");//将非空白字符列看作单词进行分析
   int n=pas.countTokens();//单词（没有空格分格的一串字符列）个数
    int i=0;
    String s="";
    String s1="";
    while(i<=n-1)
     {if(i<n-1)//对除最后一个单词的前面的单词加一个空格赋给字符串 s1
        s1=pas.nextToken()+' ';//当 i=0 时，s1 得到第一个单词
      if(i==n-1)//将最后一个单词赋给字符串 s1
        s1=pas.nextToken();
      s+=s1;//将各字符串连接起来组成一个新串 s
      i++;
   }
   str1=s;
   System.out.println("整理后的字符串 str1="+str1);
  }
}
```

4.13 编写用数组实现大整数的类，提供大整数的加、减、乘等运算。

这是一道比较复杂的题目，我通过参考网上相关资料，终于做出了一份答案，关于除法，题目没有要求，同时感觉除法实现难度很大，所以这里没有做出，请有兴趣的同学以后跟进。

```
public class BigNumber
{
 public static int[] add(int a[],int b[])//大整数加法实现
 {
  int carry=0;//进位设置
```

```
 int c[]=new int[a.length];
  for(int i=c.length-1;i>=0;i--)
  {
   c[i]=a[i]+b[i]+carry;
   if(c[i]>10000)
   {c[i]=c[i]-10000;carry=1;}
   else
   carry=0;
  }
  return c;
}
public static int[] subtract(int a[],int b[])//减法实现
{
 int borrow=0;
 int c[]=new int[a.length];
  for(int i=c.length-1;i>=0;i--)
 {
  if(a[0]<b[0])
  c[i]=b[i]-a[i]-borrow;
   else
   c[i]=a[i]-b[i]-borrow;
   if(c[i]>=0)
   borrow=0;
   else
   {
    borrow=1;c[i]=c[i]+10000;
    }
 }
 return c;
}
 public static int[] multi(int a[],int b[])//乘法实现
{
 int carry=0;
 int c[]=new int[a.length+b.length];//两数相乘会使结果数组长度变长
 int d[]=new int[c.length];int r,s;
 for(r=c.length-1,s=a.length-1;r>=c.length-a.length;r--,s--)
 d[r]=a[s];
 int i,j,k,t;
  for( i=c.length-1;i>=c.length-a.length-1 ;i--)
    { for(k=i,t=i;k>=0;k--)
    {for( j=b.length-1;j>=0;j--)
       {
        int temp=d[k]*b[j]+carry;
        if(t==0)break;
         c[t]=(c[t]+temp)%10000;
         carry=(c[t]+temp)/10000;
         t=t-1;
       }
    }
```

```
   }
   return c;
 }
 public static void getBigNumber(int x[])//输出大整数
 {int j=0;
  while(x[j]==0)//当数组中第一个不为 0 的元素左边有 0 的元素时，不输出 0
  {if(j==x.length-1)//当循环已到数组最后一个元素时，退出循环
    break;
    j++;}
  for(int i=j; i<=x.length-1;i++)//对从第一个不为 0 的元素进行输出，或输出 0
  {
   if(i>j)
   {
    if(x[i]<10)
       System.out.print("000"+x[i]);
     else if(x[i]<100)
            System.out.print("00"+x[i]);
         else if(x[i]<1000)
                System.out.print("0"+x[i]);
              else
                System.out.print(String.valueOf(x[i]));
   }
   if(i==j)
   System.out.print(String.valueOf(x[i]));
  }
 }

 public static void main(String[]args)
 {
   System.out.println("以数组形式显示两个大整数，如果两个数长度不等时，在短数前补 0 使两数长度相等");
   int a[]={11,0000};
   int b[]={11,0000};
    int c1[]=BigNumber.add(a,b);
   BigNumber.getBigNumber(c1);
   System.out.println();
   int c2[]=BigNumber.subtract(a,b);
   if(a[0]<b[0])
   System.out.print("-");
   BigNumber.getBigNumber(c2);
   System.out.println();
   int c3[]=BigNumber.multi(a,b);
    BigNumber.getBigNumber(c3);
  }
}
```

第 5 章 图形界面设计(一)

本章内容(重点：15%)

5．1 图形界面设计基础

早先程序使用最简单的输入输出方式，用户在键盘输入数据，程序将信息输出在屏幕上。现代程序要求使用图形用户界面(Graphical User Interface,GUI)，界面中有菜单、按钮等，用户通过鼠标选择菜单中的选项和点击按钮，命令程序功能模块。本章学习如何用 Java 语言编写 GUI 科学试验，如何通过 GUI 实现输入和输出。

5．1．1 AWT 和 Swing

先前用 Java 编写 GUI 程序，是使用抽象窗口工具包 AWT(Abstract Window Toolkit).现在多用 Swing。Swing 可以看作是 AWT 的改良版，而不是代替 AWT，是对 AWT 的提高和扩展。所以，在写 GUI 程序时，Swing 和 AWT 都要作用。它们共存于 Java 基础类(Java Foundation Class,JFC)中。

尽管 AWT 和 Swing 都提供了构造图形界面元素的类，但它们的重要方面有所不同：AWT 依赖于主平台绘制用户界面组件；而 Swing 有自己的机制，在主平台提供的窗口中绘制和管理界面组件。Swing 与 AWT 之间的最明显的区别是界面组件的外观，AWT 在不同平台上运行相同的程序，界面的外观和风格可能会有一些差异。然而，一个基于 Swing 的应用程序可能在任何平台上都会有相同的外观和风格。

Swing 中的类是从 AWT 继承的，参见图 5.1。有些 Swing 类直接扩展 AWT 中对应的类。例如，JApplet、JDialog、JFrame 和 JWindow。

使用 Swing 设计图形界面，主要引入两个包：

javax.swing 包含 Swing 的基本类；java.awt.event 包含与处理事件相关的接口和类。

由于 Swing 太丰富，不可能在一本教科书中给出 Swing 的全面介绍，但本书所介绍的有关 Swing 的知识，已足以让读者编写相当精美的 GUI 程序。

5．1．2 组件和容器

组件(component)是图形界面的基本元素，用户可以直接操作，例如按钮。容器(Container)是图形界面的的复合元素，容器可以包含组件，例如面板。

Java 语言为每种组件都预定义类，程序通过它们或它们的子类各种组件对象，如，Swing 中预定义的按钮类 JButton 是一种类，程序创建的 JButton 对象，或 JButton 子类的对象就是按钮。Java 语言也为每种容器预定义类，程序通过它们或它们的子类创建各种容器对象。例如，Swing 中预定义的窗口类 JFrame 是一种容器类，程序创建的 JFrame 或 JFrame 子类的对象就是窗口。

为了统一管理组件和容器，为所有组件类定义超类，把组件的共有操作都定义在 Component 类中。同样，为所有容器类定义超类 Container 类，把容器的共有操作都定义在 Container 类中。例如，Container 类中定义了 add()方法，大多数容器都可以用 add()方法向容器添加组件。

Component、Container 和 Graphics 类是 AWT 库中的关键类，其中 Graphics 类将在第 7 章中介绍。为能层次地构造复杂的图形界面，容器被当作特殊的组件，可以把容器放入另一个容器中。例如，把若干按钮和文本框分放在两个面板中，再把这两个面板和另一些按钮放入窗口中。这种层次地构造界面的方法，能以增量的方式构造复杂的用户界面。

5．1．3 事件驱动程序设计基础

1．事件、监视器和监视器注册

图形界面上的事件是指在某个组件上发生用户操作。例如，用户单击了界面上的某个按钮，就说在这个按钮上发生了事件，这个按钮对象就是事件的击发者。**对事件作监视的对象称为监视器**，监视器提供响应事件的处理方法。**为了让监视器与事件对象关联起来，需要对事件对象作监视器注册，告诉系统事件对象的监视器。**

以程序响应按钮事件为例，参见 5.3.2 节关于按钮事件处理方法的介绍。程序要创建按钮对象，把它添加到界面中，要为按钮作监视器注册，程序要有响应按钮事件的方法。当“单击按钮”事件发生时，系统就调用已为这个按钮注册的事件处理方法，完成处理按钮事件的工作。

2．实现事件处理的途征

java 语言编写事件处理程序主要有两种方案;**一个是程序重设 handleEvent(Eventevt)**,采用这个方案的程序工作量稍大一些。**另一个方案是程序实现一些系统设定的接口**。java 按事件类型提供多种接口，作为监视器对象的类需要实现相应的接口，即实现响应事件的方法。当事件发生时，系统内设的 handleEvent(Event evt)方法就自动调用监视器的类实现的响应事件的方法。

java.awt.event 包中用来检测并对事件做出反应的模型包括以下**三个组成元素**：

(1) **源对象**：事件“发生”这个组件上，它与一组“侦听”该事件的对象保持着联系。

(2) **监视器对象**：一个实现预定义的接口的类的一个对象，该对象的类要提供对发生的事件作处理的方法。

(3) **事件对象**：它包含描述当事件发生时从源传递给监视器的特定事件的信息。

一个事件驱动程序要做的工作除创建源对象和监视器对象之外，还必须安排监视器了解源对象，或向源对象注册监视器。每个源对象有一个已注册的监视器列表，提供一个方法能向该列表添加监视器对象。只有在源对象注册了监视器之后，系统才会将源对象上发生的事件通知监视器对象。

3. 事件类型和监视器接口

在 java 语言中，为了便于系统管理事件，也为了便于程序作监视器注册，系统将事件分类，称为事件类型。系统为每个事件类型提供一个接口。要作为监视器对象的类必须实现相应的接口，提供接口规定的响应事件的方法。

再以程序响应按钮事件为例，JButton 类对象 button 可以是一个事件的激发者。当用户点击界面中与 button 对应的按钮时，button 对象就会产生一个 ActionEvent 类型的事件。如果监视器对象是 obj，对象 obj 的类是 Obj,则类 Obj 必须实现 AWT 中的 ActionListener 接口，实现监视按钮事件的 actionPerformed 方法。button 对象必须用 addActionListener 方法注册它的监视器 obj。

程序运行时，当用户点击 button 对象对应的按钮时，系统就将一个 ActionEvent 对象从事件激发对象传递到监视器。ActionEvent 对象包含的信息包括事件发生在哪一个按钮，以及有关该事件的其他信息。

表 5-1 给出有一定代表性的事件类型和产生这些事件的部分 Swing 组件。实际事件发生时，通常会产生一系列的事件，例如，用户点击按钮，会产生 ChangeEvent 事件提示光标到了按钮上，接着又是一个 ChangeEvent 事件表示鼠标被按下，然后是 ActionEvent 事件表示鼠标已松开，但光标依旧在按钮上，最后是 ChangeEvent 事件，表示光标已离开按钮。但是应用程序通常只处理按下按钮的完整动作的单个 ActionEvent 事件。

表 5-1 组件和事件类型

事件类型	组件	描述
ActionEvent	JButton,JCheckBox JComboBox,JMenuItem JRadioButton	点击、选项或选择
ChangeEvent	JSlider	调整一个可移动元素的位置
AdjustmentEvent	JScrollBar	调整滑块位置
ItemEvent	JComboBox,JCheckBox JRadioButton JRadioButtonMenuItem JCheckBoxMenuItem	从一组可选方案中选择一个项目
ListSelectionEvent	JList	选项事件
KeyEvent MouseEvent	JComponent 及其派生类	操纵鼠标或键盘
CareEvent	JTextArea,JTextField	选择和编辑文本
WindowEvent	Window 及其派生类 JFrame	对窗口打开、关闭和图标化

每个事件类型都有一个相应的监视器接口，表 5-2 列出了每个接口的方法。实现监视器接口的类必须实现所有定义在接口中的方法。

表 5-2 监视器接口方法

监视器接口	方法	获取事件源的方法
ActionListener	actionPerformed	getSource,getActionCommand
ChangeListener	stateChanged	getSource
AdjustmentListener	adjustmentValueChanged	getAdjustable
FocusListener	focusGained,focusLost	
ItemListener	itemStateChanged	getItemSelectable(),getSource()
ListSelectionListener	valueChanged	e.getSource().getSelectedValue()
KeyListener	keyPressed,keyReleased,keyTyped	
CareListener	careUpdate	
MouseListener	mouseClicked,mouseEntered, mouseExited,mousePressed, mouseReleased	
MouseMontionListener	mouseDragged,mouseMoved	
WindowListener	windowClosed,windowClosing, windowDeactivated,windowDeiconified, windowIconified,windowOpened	

事件驱动程序的实际例子请参见以后各小节含事件处理的实例程序。

5．2 框架窗口

窗口是 GUI 编程的基础，小应用程序或图形界面的应用程序的可视组件都放在窗口中，在 GUI 中，窗口是用户屏幕的一部分，起着在屏幕中一个小屏幕的作用。有以下三种窗口：

(1) Applet 窗口，Applet 类管理这个窗口，当应用程序程序启动时，由系统创建和处理。

(2) 框架窗口(JFrame),这是通常意义上的窗口，它支持窗口周边的框架、标题栏，以及最小化、最大化和关闭按钮。

(3) 一种无边框窗口(JWindow)，没有标题栏，没有框架，只是一个空的矩形。

用 Swing 中的 JFrame 类或它的子类创建的对象就是 JFrame 窗口。

JFrame 类的主要构造方法：

(1) JFrame(),创建无标题的窗口对象。

(2) JFrame(String s),创建一个标题名是字符串 s 的窗口对象。

JFrame 类的其他常用方法：

(1) setBounds(int x,int y,int width,int height),参数 x,y 指定窗口出现在屏幕的位置；参数 width,height 指定窗口的宽度和高度。单位是像素。

(2) setSize(int width,int height),设置窗口的大小，参数 width 和 height 指定窗口的宽度和高度，单位是像素。

(3) setBackground(Color c),以参数 c 设置窗口的背景颜色。

(4) setVisible(boolean b),参数 b 设置窗口是可见或不可见。JFrame 默认是不可见的。

(5) pack(),用紧凑方式显示窗口。如果不使用该方法，窗口初始出现时可能看不到窗口中的组件，当用户调整窗口的大小时，可能才能看到这些组件。

(6) setTitle(String name),以参数 name 设置窗口的名字。

(7) getTitle()，获取窗口的名字。

(8) setResiable(boolean m),设置当前窗口是否可调整大小(默认可调整大小)。

Swing 里的容器都可以添加组件，**除了 JPanel 及其子类(JApplet)之外，其他的 Swing 容器不允许把组件直接加入**。其他容器添加组件有两种方法：

一种是用 getContentPane()方法获得内容面板，再将组件加入。例如，例 5.1 程序中的代码：

```
mw.getContentPane().add(button);
```

该代码的意义是获得容器的内容面板，并将按钮 button 添加到这个内容面板中。

另一种是建立一个 JPanel 对象的中间容器，把组件添加到这个容器中，再用 setContentPane()把这个容器置为内容面板。例如，代码：

```
JPanel contentPane = new JPanel();
```

```
    …
    mw.setContentPane(contentPane);
```

以上代码把 contentPane 置成内容面板。

[例 5.1]一个用 JFrame 类创建窗口的 Java 应用程序。窗口只有一个按钮。

```
import javax.swing.*;
public class Example5_1{
    public static void main(String args[]){
        JFrame mw = new JFrame(“我的第一个窗口”);
        mw.setSize(250,200);
        JButton button = new JButton(“我是一个按钮”);
        mw.getContentPane().add(button);
        mw.setVisible(true);
    }
}
```

用 Swing 编写 GUI 程序时，通常不直接用 JFrame 创建窗口对象，而用 JFrame 派生的子类创建窗口对象，在子类中可以加入窗口的特定要求和特别的内容等。

[例 5.2]定义 JFrame 派生的子类 MyWindowDemo 创建 JFrame 窗口。类 MyWindowDemo 的构造方法有五个参数：窗口的标题名，加放窗口的组件，窗口的背景颜色以及窗口的高度和宽度。在主方法中，利用类 MyWindowDemo 创建两个类似的窗口。

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Example5_2{
    public static MyWindowDemo mw1;
    public static MyWindowDemo mw2;
    public static void main(String args[]){
        JButton static butt1 = new JButton(“我是一个按钮”);
        String name1 = “我的第一个窗口”;
        String name2 =  “我的第二个窗口”;
        mw1 = new MyWindowDemo(name1,butt1,Color.blue,350,450);
        mw1.setVisible(true);
        JButton butt2 = new JButton(“我是另一个按钮”);
        mw2 = new MyWindowDemo(name2,butt2,Color.magenta,300,400);
        mw2.setVisible(true);
    }
}
class MyWindowDemo extends JFrame{
    public MyWindowDemo(String name,JButton button,Color c,int w,int h){
        super();
        setTitle(name);
        setSize(w,h);
        Container con = getContentPane();
        con.add(button);
        con.setBackground(c);
    }
}
```

显示颜色由 java.awt 包的 Color 类管理，在 Color 类中预定了一些常用的颜色，参见表 7-1。JFrame 类的部分常用方法参见表 5-3。

表 7-1Color 类中定义的常用颜色

字段摘要		
static Color	black	黑色。
static Color	BLACK	黑色。
static Color	blue	蓝色。
static Color	BLUE	蓝色。
static Color	cyan	青色。
static Color	CYAN	青色。
static Color	DARK_GRAY	深灰色。
static Color	darkGray	深灰色。
static Color	gray	灰色。
static Color	GRAY	灰色。
static Color	green	绿色。
static Color	GREEN	绿色。
static Color	LIGHTGRAY	浅灰色。
static Color	lightGray	浅灰色。
static Color	magenta	洋红色。
static Color	MAGENTA	洋红色。
static Color	orange	桔黄色。
static Color	ORANGE	桔黄色。
static Color	pink	粉红色。
static Color	PINK	粉红色。
static Color	red	红色。
static Color	RED	红色。
static Color	white	白色。
static Color	WHITE	白色。
static Color	yellow	黄色。
static Color	YELLOW	黄色。

表 5-3 JFrame 类的部分常用方法

方法	意义
JFrame()	构造方法，创建一个 JFrame 对象
JFrame(String title)	创建一个以 title 为标题的 JFrame 对象
add()	从父类继承的方法，向窗口添加窗口元素
void addWindowListener(WindowListener ear)	注册监视器，监听由 JFrame 对象击发的事件
Container getContentPane()	返回 JFrame 对象的内容面板
void setBackground(Color c)	设置背景色为 c
void setForeground(Color c)	设置前景色为 c
void setSize(int w,int h)	设置窗口的宽为 w,高为 h
vid setTitle(String title)	设置窗口中的标题
void setVisible(boolean b)	设置窗口的可见性，true 可见，false 不可见

5．3 标签、按钮和按钮事件

标签和按钮也许是图形界面中最常见的两种组件，按钮又总是与激发动作事件有关。

5．3．1 标签

标签(JLabel)是最简单的 Swing 组件。标签对象的作用是对位于其后的界面组件作说明。可以设置标签的属性,即前景色,背景色、字体等，**但不能动态地编辑标签中的文本**。

程序关于标签的基本内容有以下几个方面：

(1) 声明一个标签名

(2) 创建一个标签对象

(3) 将标签对象加入到某个容器。

JLabel 类的主要构造方法是：

(1) JLabel (),构造一个无显示文字的标签

(2) JLabel (String s),构造一个显示文字为 s 的标签。

(3) JLabel(String s, int align),构造一个显示文字为 s 的标签。align 为显示文字的水平方式，对齐方式有三种：左对齐：JLabel.LEFT。中心对齐：JLabel.CENTER,右对齐：JLabel.RIGHT.

JLabel 类的其他常用方法是：

(1) setText(String s),设置标签显示文字。

(2) getText(),获取标签显示文字。

(3) setBackground(Color c)，设置标签的背景颜色，默认背景颜色是容器的背景颜色。

(4) setForeground(Color c),设置标签上的文字的颜色，默认颜色是黑色。

5.3.2 按钮

按钮(JButton)在界面设计中用于激发动作事件。按钮可显示文本，当按钮被激活时，能激发动作事件。

JButton 常用构造方法有：

(1) JButton(),创建一个没有标题的按钮对象。

(2) JButton(String s),创建一个标题为 s 的按钮对象。

JButton 类的其他常用方法有：

(1) setLabel(String s)，设置按钮的标题文字。

(2) getLabel(),获取按钮的标题文字。

(3) setMnemonic(char mnemonic)，设置热键

(4) setToolTipText(String s),设置提示文字。

(5) setEnabled(boolean b),设置是否响应事件

(6) setRolloverEnabled(boolean b)设置是否可滚动。

(7) addActionListener(ActionListener aL),向按钮添加动作监视器。

(8) removeActionListener(ActionListener aL),移动按钮的监视器。

按钮处理动作事件的基本内容有以下几个方面：

(1) 与按钮动作事件相关的接口是 ActionListener，给出实现该接口的类的定义。

(2) 声明一个按钮名。

(3) 创建一个按钮对象。
(4) 将按钮对象加入到某个容器。
(5) 为需要控制的按钮对象注册监视器，对在这个按钮上产生的事件实施监听。

如果是按钮对象所在的类实现监视接口，注册监视器的代码形式是

```
addActionListener(this);
```

参见例 5.3。 **如果是别的类 A 的对象 a 作为监视器，类 A 必须实现 ActionListener 接口**，完成监视器注册需用以下形式的两行代码：

```
A a = new A();//创建类 A 的实例 a
addActionListener(a);//用对象 a 作为监视器对事件进行监视。
```

(6) 在实现接口 ActionListener 的类中，给出处理事件的方法的定义：

```
public void actionPerformed(ActionEvent e);
```

在处理事件的方法中，用获取事件源信息的方法获得事件源信息，并判断和完成相应处理。获得事件源的方法有：方法 getSource()获得事件源对象；方法 getActionCommand()获得事件源按钮的文字信息。

[例 5.3]处理按钮事件实例，应用程序定义了一个窗口，窗口内设置两个按钮，当点击 Red 按钮时，窗口的背景色置成红色；点击 Green 按钮时，窗口的背景色置成绿色。

```
import javax.swing.*;import java.awt.*;import java.awt.event.*;
public class J503{
 public static void main(String[]args){
      ButtonDemo myButtonGUI=new ButtonDemo();//声明并创建按钮对象
      myButtonGUI.setVisible(true);
 }
}
class ButtonDemo extends JFrame implements ActionListener{
 public static final int Width=250;
 public static final int Height=200;
 ButtonDemo(){
      setSize(Width,Height);     setTitle("按钮事件样例");
      Container conPane=getContentPane();
      conPane.setBackground(Color.BLUE);
      conPane.setLayout(new FlowLayout());//采用 FlowLayout 布局
      JButton redBut=new JButton("Red");
      redBut.addActionListener(this);//给 Red 按钮注册监视器
      conPane.add(redBut);//在窗口添加 Red 按钮
      JButton greenBut=new JButton("Green");
      greenBut.addActionListener(this);//给 Green 按钮注册监视器
      conPane.add(greenBut);//在窗口添加 Green 按钮
 }
 public void actionPerformed(ActionEvent e){//实现接口处理事件的方法
      Container conPane=getContentPane();
      if(e.getActionCommand().equals("Red"))//是 Red 按钮事件
           conPane.setBackground(Color.RED);
      else if(e.getActionCommand().equals("Green"))//是 Green 按钮事件
           conPane.setBackground(Color.GREEN);
      else{}
 }
}
```

用鼠标点击按钮产生事件对象，将事件送达对象，这个过程称为激发事件。当一个事件被送到监视器对象时，监视器对象实现的接口方法被调用，调用时系统会提供事件对象的参数。程序中虽然没有调用监视器方法的的代码，但是程序做了两

件事：第一，指定哪一个对象是监视器，它将响应由按钮的激发的事件，这个步骤称为监视器注册。第二，必须定义一个方法，当事件送到监视器时，这个方法将被调用。程序中没有调用这个方法的代码，这个调用是系统执行的。

在上面的程序中，代码

```
redBut.addActionListener(this);
```

注册 this 作为 redBut 按钮的监视器，随后的代码也注册 this 作为 greenBut 按钮的监视器。在上述的程序中，this 就是当前的 ButtonDemo 对象 myButtonGUI。这样，ButtonDemo 类就是监视器对象的类，对象 MyButtonGUI 作为两个按钮的监视器。在类 ButtonDemo 中有监视器方法的实现。当一个按钮被点击时，系统以事件的激发者为参数，自动调用方法 actionPerformed ()。

组件不同，激发的事件种类也不同，监视器类的种类也不同。按钮激发的事件称为 action 事件，相应的监视器称为 action 监视器。一个 action 监视器对象的类型为 ActionListener，类要实现 ActionListener 接口。程序体现这些内容需要做到两点：

(1) 在类定义的首行接上代码 implements ActionListener;

(2) 类内定义方法 actionPerformed ()。

前面程序中的类 ButtonDemo 正确地做到了这两点。

每个界面元素当激发事件时，都有一个字符串与这个事件相对应，这个字符串称为 action 命令。用代码 e.getActionCommand()就能获取 action 事件参数 e 的命令字符串，据此，方法 actionPerformed()就能知道是哪一个按钮激发的事件。在默认情况下，按钮的命令字符串就是按钮上的文字。如有必要可以用方法 setActionCommand()为界面组件设置命令字符串。

5.4 面板

面板有两种，一种是普通面板(JPanel)，另一种是滚动面板(JScrollPane).

5.4.1 JPanel

面板是一种通用容器，JPanel 的作用是实现界面的层次结构，在它上面放入一些组件，也可以在上面绘画，将放有组件和有画的 JPanel 再放入另一个容器里。JPanel 的默认布局为 FlowLayout。

面板处理程序的基本内容有以下几个方面

(1) 通过继承声明 JPanel 类的子类，子类中有一些组件，并在构造方法中将组件加入面板。

(2) 声明 JPanel 子类对象。

(3) 创建 JPanel 子类对象。

(4) 将 JPanel 子类对象加入到某个容器。

JPanel 类的常用构造方法有：

(1) JPanel(),创建一个 JPanel 对象。

(2) JPanel(LayoutManager layout),创建 JPanel 对象时指定布局 layout。

JPanel 对象添加组件的方法：

(1) add(组件)，添加组件。

(2) add(字符串，组件)，当面板采用 GardLayout 布局时，字符串是引用添加组件的代号。

[例 5.4] 小应用程序有两个 JPanel 子类对象和一个按钮。每个 JPanel 子类对象又有两个按钮和一个标签。

```
import java.applet.*;
import javax.swing.*;
class MyPanel extends JPanel{
    JButton button1,button2;
    JLabel Label;
    MyPanel(String s1,String s2,String s3){
        //Panel 对象被初始化为有两个按钮和一个文本框
        button1=new JButton(s1);button2=new JButton(s2);
        Label=new JLabel(s3);
        add(button1);add(button2);add(Label);
    }
}
public class J504 extends Applet{
    MyPanel panel1,panel2;
```

```
    JButton Button;
    public void init(){
        panel1=new MyPanel("确定","取消","标签，我们在面板 1 中");
        panel2=new MyPanel("确定","取消","标签，我们在面板 2 中");
        Button=new JButton("我是不在面板中的按钮");
        add(panel1);add(panel2);add(Button);
        setSize(300,200);
    }
}
```

5.4.2 JScrollPane

当一个容器内放置了许多组件，而容器的显示区域不足以同时显示所有组件时，如果让容器带滚动条，通过移动滚动条的滑块，容器中位置上的组件就能看到。滚动面板 JScrollPane 能实现这样的要求，JScrollPane 是带有滚动条的面板。JScrollPane 是 Container 类的子类，**也是一种容器，但是只能添加一个组件。JScrollPane 的一般用法是先将一些组件添加到一个 JPanel 中，然后再把这个 JPanel 添加到 JScrollPane 中。**这样，从界面上看，在滚动面板上，好像也有多个组件。在 Swing 中，像 JTextArea、JList、JTable 等组件都没有自带滚动条，都需要将它们放置于滚动面板，利用滚动面板的滚动条，浏览组件中的内容。

JScrollPane 类的构造方法有：

(1) JScrollPane()，先创建 JScrollPane 对象，然后再用方法 setViewportView(Component com)为滚动面板对象放置组件对象。

(2) JScrollPane(Component com),创建 JScrollPane 对象，参数 com 是要放置于 JScrollPane 对象的组件对象。为 JScrollPane 对象指定了显示对象之后，再用 add()方法将 JScrollPane 对象放置于窗口中。

JScrollPane 对象设置滚动条的方法是：

(1) setHorizontalScrollBarPolicy(int policy),policy 取以下列 3 个值之一：

JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS

JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEED

JScrollPane.HORIZONTAL_SCROLLBAR_NEVER

(2) setVerticalScrollBarPolicy(int policy), policy 取以下列 3 个值之一：

JScrollPane.VERTICAL_SCROLLBAR_ALWAYS

JScrollPane.VERTICAL_SCROLLBAR_AS_NEED

JScrollPane.VERTICAL_SCROLLBAR_NEVER

以下代码将文本区放置于滚动面板，滑动面板的滚动条能浏览文本区

```
JTextArea textA = new JTextArea(20,30);
JScrollPane jsp = new JScrollPane(TextA);
getContentPane().add(jsp);//将含文本区的滚动面板加入到当前窗口中。
```

5.5 布局设计

在界面设计中，一个容器要放置许多组件，为了美观，为组件安排在容器中的位置，这就是布局设计。java.awt 中定义了多种布局类，每种布局类对应一种布局的策略。常用的有以下布局类：

FlowLayout,依次放置组件。

BoarderLayout,将组件放置在边界上。

CardLayout,将组件像扑克牌一样叠放，而每次只能显示其中一个组件。

GridLayout,将显示区域按行、列划分成一个个相等的格子，组件依次放入这些格子中。

GridBagLayout,将显示区域划分成许多矩形小单元，每个组件可占用一个或多个小单元。

其中 GridBagLayout 能进行精细的位置控制，也最复杂，本书不讨论这种布局策略。

每个容器都有一个布局管理器，由它来决定如何安排放入容器内的的组件。布局管理器是实现 LayoutManager 接口的类。

5. 5.1 FlowLayout 布局 (JApplet,JPanel,JScrollPane 默认布局)

FlowLayout 布局是将其中的组件按照加入的先后顺序从左到右排列，一行满之后就转到一下行继续从左到右排列，每一行中的组件都居中排列。这是一种最简便的布局策略，一般用于组件不多的情况，当组件较多时，容器中的组件就会显得高低不平，各行长短不一。

FlowLayout 是小应用程序和面板默认布局，FlowLayout 布局的构造方法有：

(1) FlowLayout(),生成一个默认的 FlowLayout 布局。**默认情况下，组件居中，间隙为 5 个像素**。

(2) FlowLayout(int aligment),设定每珩的组件的对齐方式。alignment 取值可以为 FlowLayout.LEFT,FlowLayout.CENTER,FlowLayout.RIGHT。

(3) FlowLayout(int aligment,int horz, int vert),设定对齐方式，并设定组件的水平间距 horz 和垂直间距 vert.

用超类 Container 的方法 setLayout()为容器设定布局。例如，代码 setLayout(new FlowLayout())为容器设定 FlowLayout 布局。将组件加入容器的方法是 add(组件名)。

5.5.2 BorderLayout 布局(JWindow、JFrame,JDialog 的默认布局)

BorderLayout 布局策略是把容器内的空间简单划分为东“East”,西“West”,南“South”,北 “North”,中“Center”五个区域。加入组件时，都应该指明把组件放在哪一个区域中。一个位置放一个组件。如果某个位置要加入多个组件，应先将要加入该位置的组件放放另一个容器中，然后再将这个容器加入到这个个位置。

BorderLayout 布局的构造方法有：

(1) BorderLayout(),生成一个默认的 BorderLayout 布局。**默认情况下，没有间隙。**

(2) BorderLayout(int horz,int vert),设定组件之间的水平间距和垂直间距。

BorderLayout 布局策略的设定方法是 setLayout(new BorderLayout())。将组件加入到容器的方法是 add(组件名,位置)，如果加入组件时没有指定位置，则**默认为“中”位置**。

BorderLayout 布局是 **JWindow、JFrame,JDialog 的默认布局**。

[例 5.5]应用程序设有五个标签、分别放于窗口的东、西、南、北和中五个区域。

```
import javax.swing.*;import java.awt.*;
public class J505{
    public static void main(String[]args){
        JLabel label1,label2,label3,label4,label5;
        JFrame mw=new JFrame("我是一个窗口");//创建一个窗口容器对象
        mw.setSize(250,200);
        Container con=mw.getContentPane();
        con.setLayout(new BorderLayout());
        label1=new JLabel("东标签");//默认左对齐
        label2=new JLabel("南标签",JLabel.CENTER);
        label3=new JLabel("西标签");
        label4=new JLabel("北标签",JLabel.CENTER);
        label5=new JLabel("中标签",JLabel.CENTER);
        con.add(label1,"East");
        con.add(label2,"South");
        con.add(label3,"West");
        con.add(label4,"North");
        con.add(label5,"Center");
        mw.setVisible(true);
    }
}
```

5. 5.3 GridLayout 布局

GridLayout 布局是把容器划分成若干行和列的**网格状**，行数和列数由程序控制，组件放在网格的小格子中。GridLayout 布局的特点是组件定位比较精确。由于 GridLayout 布局中每个网格具有相同形状和大小，要求放入容器的组件也应保持相同的大小。

GridLayout 布局的构造方法有：

(1) GridLayout(),生成 一个单列的 GridLayout 布局。**默认情况下，无间隙**。

(2) GridLayout(int row,int col),设定一个有行 row 和列 col 的 GridLayout 布局。

(3) GridLayout(int row,int col,int horz,int vert),设定布局的行数和列数、组件的水平间距和垂直间距。

GridLayout 布局以**行**为基准，当放置的组件个数超额时，自动增加列；反之，组件太少也会自动减少列，行数不变，组件按行优先顺序排列(**根据组件自动增减列**)。GridLayout 布局的每个网格必须填入组件，如果希望某个网格为空白，可以用一个空白标签(add(new Label()))顶替。

[例 5.6]小应用程序先将若干个按钮和若干个标签放入 JPanel 中，然后将 JPanel 放入 JScrollPane 中，最后，将 JScrollPane 放入小程序的窗口中，程序所创建的 JScrollPane 总是带水平和垂直滚动条，滚动面板的可视范围小于面板的实际要求，可以移动滚动条的滑块显示面板原先不在可视范围内的区域。

```
import java.applet.*;
import javax.swing.*;
import java.awt.*;
class MyWindow extends JFrame{
    public MyWindow(int w,int h){
        setTitle("滚动面板实例");
        Container con=getContentPane();
        con.setPreferredSize(new Dimension(w,h));
        con.setLayout(new BorderLayout());
        JPanel p=new JPanel();
        p.setLayout(new GridLayout(6,6));
        for (int i=0;i<6;i++){
            p.add(new JLabel());
            for(int j=1;j<=2;j++){
                p.add(new JButton("按钮"+(2*i+j)));
                p.add(new JLabel("标签"+(2*i+j)));
            }
            p.add(new JLabel());
        }
        p.setBackground(Color.blue);
        p.setPreferredSize(new Dimension(w+60,h+60));
        JScrollPane ScrollPane=new JScrollPane(p);
        ScrollPane.setPreferredSize(new Dimension(w-60,h-60));
        add(ScrollPane,BorderLayout.CENTER);//小程序添加滚动面板
        setVisible(true);     pack();
    }
}
class ScrollPane extends JScrollPane{
    public ScrollPane(Component p){
        super(p);
        setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
    }
}
public class J506 extends Applet{
    MyWindow myWindow;
    public void init(){
        myWindow=new MyWindow(400,350);
    }
}
```

GridLayout 布局要求所有组件的大小保持一致，这可能会使用界面外观不够美观。一个补救的办法是让一些小组件合并放在一个容器中，然后把这个容器作为组件，再放入到 GridLayout 布局中。这就是前面所说的容器嵌套。例如，容器 A 使用 GridLayout 布局，将容器均分为网格；另有容器 B 和 C 各放入若干组件后，把 B 和 C 分别作为组件添加到容器 A 中。容器 B 和 C 也可以设置为 GridLayout 布局，把自己分为若干网格，也可以设置成其他布局。这样，从外观来看，各组件的大小就有了差异。

5.5.4 CardLayout 布局

采用 CardLayout 布局的容器虽可容纳多个组件，但是多个组件拥有同一个显示空间，某一时刻只能显示一个组件。就像一叠扑克牌每次只能显示最上面的一张一样，这个显示的组件将占据容器的全部空间。CardLayout 布局设计步骤如下：
先创建 CardLayout 布局对象。然后，使用 setLayout()方法为容器设置布局。最的，调用容器的 add()方法将组件加入容器。
CardLayout 布局策略加入组件的方法是：

add(组件代号，组件)；

其中组件代号是字符串，是另给的，与组件名无关。例如，以下代码为一个 JPanel 容器设定 CardLayout 布局：

```
CardLayout myCard = new CardLayout();//创建 CardLayout 布局对象
JPanel p = new JPanel();//创建 Panel 对象
p.setLayout(myCard);
```

用 CardLayout 类提供的方法显示某一组件的方式有两种：

(1) 使用 show(容器名，组件代号)形式的代码，指定某个容器中的某个组件显示。

例如，以下代码指定容器 p 的组件代号 k，显示这个组件：

```
myCard.show(p,k);
```

(2) 按组件加入容器的顺序显示组件。

first(容器)：例如，代码 myCard.first(p)；

last(容器)：例如 ， myCard.last(p)；

next(容器)：例如，myCard.next(p)；

previous(容器):myCard.previous(p)；

[例 5.7]小应用程序使用 CardLayout 布局，面板容器 p 使用 CardLayout 布局策略设置 10 个标签组件。窗口设有 4 个按钮，分别负责显示 p 的第一个组件、最后一个组件、当前组件的前一个组件和当前的组件的最后一个组件。

```
import java.applet.*;import java.awt.*;
import java.awt.event.*;import javax.swing.*;
class MyPanel extends JPanel{
    int x;JLabel label1;
    MyPanel(int a){
        x=a;getSize();
        label1=new JLabel("我是第"+x+"个标签");add(label1);
    }
    public Dimension getPreferredSize(){
    return new Dimension(200,50);}
}
public class J507 extends Applet implements ActionListener{
    CardLayout mycard;MyPanel myPanel[];JPanel p;
    private void addButton(JPanel pan,String butName,ActionListener listener){
        JButton aButton=new JButton(butName);
        aButton.addActionListener(listener);
        pan.add(aButton);
    }
    public void init(){
        setLayout(new BorderLayout());//小程序的布局是边界布局
        mycard=new CardLayout();
        this.setSize(400,150);
        p=new JPanel();p.setLayout(mycard);//p 的布局设置为卡片式布局
        myPanel=new MyPanel[10];
        for(int i=0;i<10;i++){
            myPanel[i]=new MyPanel(i+1);
            p.add("A"+i,myPanel[i]);
        }
        JPanel p2=new JPanel();
```

```
        addButton(p2,"第一个",this);
        addButton(p2,"最后一个",this);
        addButton(p2,"前一个",this);
        addButton(p2,"后一个",this);
        add(p,"Center");       add(p2,"South");
    }
    public void actionPerformed(ActionEvent e){
        if (e.getActionCommand().equals("第一个"))mycard.first(p);
        else if(e.getActionCommand().equals("最后一个"))mycard.last(p);
        else if(e.getActionCommand().equals("前一个"))mycard.previous(p);
        else if(e.getActionCommand().equals("后一个"))mycard.next(p);
    }
}
```

5.5.5 null 布局与 setBounds 方法

空布局就是把一个容器的布局设置为 null 布局。空布局采用 setBounds()方法设置组件本身的大小和在容器中的位置：

setBounds(int x,int y,int width,int height)

组件所占区域是一个矩形，参数 x,y 是组件的左上角在容器中的位置坐标；参数 weight,height 是组件的宽和高。

空布局安置组件的办法分两个步骤：先使用 add()方法身容器添加组件。然后调用 setBounds()方法设置组件在容器中的位置和组件本身的大小。

与组件相关的其他方法：

(1) getSize().width,

(2) getSize().height

(3) setVgap(ing vgap)

(4) setHgap(int hgap);

5.6 文本框和文本区

在图形界面中，文本框和文本区是用于信息输入输出的组件。

5.6.1 文本框

文本框(JTextField)是界面中用于输入和输出一行文本的框。JTextField 类用来建立文本框。 与文本框相关的接口是 ActionListener。

文本框处理程序的基本内容有以下几个方面：

(1) 声明一个文本框名。

(2) 建立一个文本框对象。

(3) 将文本框对象加入到某个容器。

(4) 对需要控制的文本框对象注册监视器，监听文本框的输入结束(即输入回车键)事件。

(5) 一个处理文本框事件的方法，完成对截获事件进行判断和处理。

JTextField 类的主要构造方法：

(1) JTextField()，文本框的字符长度为 1。

(2) JTextField(int columns)，文本框初始值为空字符串，文本框的字符长度设为 columns.

(3) JTextField(String text),文本框初始值为 text 的字符串。

(4) JTextField(String text,int columns);文本框初始值为 text，文本框的字符长度为 columns。

JTextField 类的其他方法：

(1) setFont(Font f),设置字体

(2) setText(String text)，在文本框中设置文本

(3) getText()，获取文本框中的文本。

(4) setEditable(boolean)，指定文本框的可编辑性，默认为 true，可编辑。

(5) setHorizontalAlignment(int alignment)设置文本对齐方式。对齐方式有：JTextField.LEFT, JTextField.CENTER, JTextField.RIGHT。

(6) requestFocus()，设置焦点。

(7) addActionListener(ActionListener)，为文本框设置动作监视器，指定 ActionListener 对象接收该文本框上发生的输入结束动作事件。
(8) removeActionListener(ActionListener)移去文本框监视器。
(9) getColumns(),返回文本框的列数。
(10) getMinimumSize(),返回文本框所需的最小尺寸。
(11) getMinimumSize(int)，返回文本框在指定的字符数情况下，所需的最小尺寸。
(12) getPreferredSize(),返回文本框希望具有的尺寸。
(13) getPreferredSize(int),返回文本框在指定字符数情况下，希望具有的尺寸。
[例 5.8]小应用程序有两个文本框。一个文本用于输入一个整数，另一个文本框显示这个整数的平方值。程序用字符串转基本类型的方法 Long.parseLong(text1.getText())，读取文本框 text1 中的字符串，并将它转换成整数。程序用 Sqr 类的实例作为监视器，但为了让监视器能访问主类的变量，主类中的变量被声明为类变量，并且不设置访问权限。

```
import java.applet.*;import javax.swing.*;import java.awt.event.*;
public class J508 extends Applet{
    static JTextField text1,text2;
    Sqr s=new Sqr();//创建监视器
    public void init(){
        text1=new JTextField(10);
        text2=new JTextField(10);
        add(text1);
        add(text2);
        text1.addActionListener(s);//类 Sqr 的实例 s 作为 text1 的监视器
    }
}
class Sqr implements ActionListener{
    public void actionPerformed(ActionEvent e){//实现接口 ActionListener
        if(e.getSource()==J508.text1){
            long n=Long.parseLong(J508.text1.getText());
            //将 text1 的文本转换成 long 型数据
            J508.text2.setText(String.valueOf(n*n));
            //将 n*n 转化为字符串
        }
        else{}
    }
}
```

密码框(JPasswordField)是一个单行的输入组件，与 JTextField 基本类似。密码框多一个屏蔽功能，就是在输入时，都会以一个别的指定的字符(一般是*字符)输出。除了前面介绍的文本框的方法外，另有一些密码框常用的方法：
(1) getEchoChar(),返回密码的回显字符。
(2) setEchoChar(char)，设置密码框的回显字符。

5.6.2 文本区

文本区(JTextArea)是窗体中一个放置文本的区域。文本区与文本框的主要区别是文本区可存放多行文本。javax.swing 包中的 JTextArea 类用来建立文本区。JTextArea 组件没有事件。

文本区处理程序的基本内容有以下几个方面：
(1) 声明一个文本区名。
(2) 建立一个文本区对象。
(3) 将文本区对象加入到某个容器。
JTextArea 类的主要构造方法：
(1) JTextArea()，以默认的列数和行数，创建一个文本区对象。
(2) JTextArea(String s),以 s 为初始值，创建一个文本区对象。
(3) JTextArea(Strings ,int x,int y)，以 s 为初始值，行数为 x,列数为 y,创建一个文本区对象。

(4) JTextArea(int x,int y)以行数为 x,以列数为 y，创建一个文本区对象。

JTextArea 类的其他常用方法：

(1) setText(String s),设置显示文本，同时清除原有文本。

(2) getText()，获取文本区的文本。

(3) insert(String s,int x),在指定的位置插入指定的文本。

(4) replace(String s,int x,int y),用给定的一替换从 x 位置开始到 y 位置结束的文本。

(5) append(String s)，在文本区追加文本。

(6) getCarePosition(),获取文本区中活动光标的位置。

(7) setCarePosition(int n)，设置活动光标的位置。

(8) setLineWrap(boolean b)，设置自动换行，缺省情况，不自动换行。

以下代码创建一个文本区，并设置能自动换行。

```
JTextArea textA = new JTextArea(“我是一个文本区”,10,15);
textA.setLineWrap(true);//设置自动换行
```

当文本区中的内容较多，不能在文本区全部显示时，可给文本区配上滚动条。给文本区设置滚动条可用以下代码：

```
JTextArea ta = new JTextArea();
JScrollPane jsp = new JScrollPane(ta);//给文本区添加滚动条
```

5.6.3 数据输入和输出

在 GUI 中，常用文本框和文本区实现数据的输入和输出。如果采用文本区输入，通常另设一个数据输入完成按钮。当数据输入结束时，点击这个按钮。事件处理程序利用 **getText()方法从文本区中读取字符串信息**。对于采用文本框作为输入的情况，最后输入的回车符可以激发输入完成事件，通常不用另设按钮。事件处理程序可以利用单词分析器分析出一个个数，再利用字符串转换数值方法，获得输入的数值。对于输出，程序先将数值转换成字符串，然后通过 setText()方法将数据输出到文本框或文本区。

[例 5.9]小应用程序设置一个文本区、一个文本框和两个按钮。用户在文本区中输入整数序列，单击求和按钮，程序对文本区中的整数序列进行求和，并在文本框中输出和。单击第二个按钮，清除文本区和文本框中的内容。

```
import java.util.*;import java.applet.*;import java.awt.*;
import javax.swing.*;import java.awt.event.*;
public class J509 extends Applet implements ActionListener{
    JTextArea textA;JTextField textF;JButton b1,b2;
    public void init(){
        setSize(250,150);
        textA=new JTextArea("",5,10);
        textA.setBackground(Color.cyan);
        textF=new JTextField("",10);
        textF.setBackground(Color.pink);
        b1=new JButton("求    和");    b2=new JButton("重新开始");
        textF.setEditable(false);
        b1.addActionListener(this);    b2.addActionListener(this);
        add(textA);  add(textF);  add(b1);add(b2);
    }
    public void actionPerformed(ActionEvent e){
        if(e.getSource()==b1){
            String s=textA.getText();
            StringTokenizer tokens=new StringTokenizer(s);
            //使用默认的分隔符集合：空格、换行、Tab 符合回车作分隔符
            int n=tokens.countTokens(),sum=0,i;
            for(i=0;i<=n-1;i++){
                String temp=tokens.nextToken();//从文本区取下一个数据
                sum+=Integer.parseInt(temp);
            }
```

```
            textF.setText(""+sum);
        }
        else if(e.getSource()==b2){
            textA.setText(null);
            textF.setText(null);
        }
    }
}
```

[例5.10]小应用程序计算从起始整数到终止整数中是因子倍数的所有数。小程序容器用GridLayout布局将界面划分为3行列，第一行是标签，第二行和第三行是两个Panel。设计两个Panel容器类Panel1,Panel2,并分别用GridLayout布局划分。Panel1为1行6列，Panel2为1行4列。然后将标签和容器类Panel1,Panel2产生的组件加入到窗口的相应位置中。

```
import java.applet.*;import javax.swing.*;
import java.awt.*;import java.awt.event.*;
class Panel1 extends JPanel{
    JTextField text1,text2,text3;
    Panel1(){//构造方法。当创建Panel对象时，Panel被初始化为有三个标签
        //三个文本框，布局为GridLayout(1,6)
        text1=new JTextField(10);text2=new JTextField(10);
        text3=new JTextField(10);setLayout(new GridLayout(1,6));
        add(new JLabel("起始数",JLabel.RIGHT));add(text1);
        add(new JLabel("终止数",JLabel.RIGHT));add(text2);
        add(new JLabel("因子",JLabel.RIGHT));add(text3);
    }
}
class Panel2 extends JPanel{//扩展Panel类
    JTextArea text;JButton Button;
    Panel2(){//构造方法。当创建Panel对象时，Panel被初始化为有一个标签
        //一个文本框，布局为GridLayout(1,4)
        text=new JTextArea(4,10);text.setLineWrap(true);
        JScrollPane jsp=new JScrollPane(text);
        Button=new JButton("开始计算");
        setLayout(new GridLayout(1,4));
        add(new JLabel("计算结果：",JLabel.RIGHT));
        add(jsp);
        add(new Label());add(Button);
    }
}
public class J510 extends Applet implements ActionListener{
    Panel1 panel1;Panel2 panel2;
    public void init(){
        setLayout(new GridLayout(3,1));
        setSize(400,200);panel1=new Panel1();panel2=new Panel2();
        add(new JLabel("计算从起始数到终止数是因子倍数的数",JLabel.CENTER));
        add(panel1);add(panel2);
        (panel2.Button).addActionListener(this);
    }
    public void actionPerformed(ActionEvent e){
        if(e.getSource()==(panel2.Button)){
            long n1,n2,f,count=0;
```

```
                n1=Long.parseLong(panel1.text1.getText());
                n2=Long.parseLong(panel1.text2.getText());
                f=Long.parseLong(panel1.text3.getText());
                for(long i=n1;i<=n2;i++)
                {if(i%f==0)
                     panel2.text.append(String.valueOf(i)+"");
                }
            }
        }
}
```

习题

5.1 Swing 与 AWT 有何关系？

答：Swing 可以看作是 AWT 的改良版，而不是代替 AWT,是对 AWT 的提高和扩展。在写 GUI 程序时,Swing 和 AWT 都要使用，它们共存于 JAVA 基础类中。Swing 中的类是从 AWT 中继承的。

但它们也有重要的不同：AWT 依赖于主平台绘制用户界面组件，而 Swing 有自己的机制，在主平台提供的窗口中绘制和管理界面组件。Swing 和 AWT 之间的最明显的区别是界面组件的外观，AWT 在不同平台上运行相同的程序，界面的外观和风格可能会有一些差异，而一个基于 Swing 的应用程序可能在任何平台上都会有相同的外观和风格。

5.2 什么是组件对象？什么是容器?容器与其他类型的组件有何不同？

答：组件是图形界面的基本元素，用户可以直接操作，java 语言为每种组件都预定义类，程序通过它们或它们的子类创建各种组件对象。

容器是图形界面的复合元素，容器可以包含组件。java 语言也为每种容器预定义类，程序通过它们或它们的子类创建各种容器对象。

容器与其他组件的不同是：为了能层次地构造复杂的图形界面，容器被当作特殊的组件，可以把容器放入另一个容器中，可以向容器中添加组件。

5.3 以下程序中，有多少个组件，哪些既是组件又是容器？

```
import javax.swing.*;
public class Test3{
    public static void main(String args[]){
    Jframe fra = new JFrame(“?”);
    fra.setVisible(true);
    fra.setSize(120,100);
    JPane p = new JPanel();
    JButton b = new JButton(“Java”);
    JTextField text = new JTextField(10);
    JLabel label = new JLabel(“how are you”);
    p.add(b);
    fra.add(label, “North”);
    fra.add(p, “Center”);
    fra.add(text, “South”);
    fra.pack();//用紧凑方式显示窗口。
    }
}
```

题中有五个组件:JFrame,JPanel，JButton,JTextField,JLabel.其中 JFrame,JPanel 既是组件也是容器。

5.4 什么是事件？什么是监听器？如何进行事件注册？

答：图形界面上的事件是指在某个组件上发生的用户操作。

对事件作监视的对象称为监视器，监视器提供响应事件的处理方法。

为了让监视器与事件对象关联起来，需要对事件对象作监视器注册，告诉系统事件对象的监视器。以程序响应按钮事件为例，当用户点击用户界面中与 button 对应的按钮时，button 对象就会产生一个 ActionEvent 类型的事件。如果监视器对象是

obj，对象的类是 Obj,则类 Obj 必须实现 AWT 中的 ActionListener 接口，实现监视按钮事件睥 actionPerformed 方法。buton 对象必须用 addActionListener 方法注册它的监视器对象 obj.

5.5 如何进行事件处理？如何获取事件源？

答：java 语言编写事件处理程序主要有两种主要方法：一个是程序重设方法 handleEvent(Event evt);一个是程序实现一些系统设定的接口。java 按事件类型提供多种接口，作为监视器对象的类需要实现相应的接口，即实现响应事件的方法。当事件发生时，系统内设的 handleEvent(Event evt)方法就自动调用监视器的类实现响应事件的方法。

一个事件驱动程序要作的工作除创建源对象和监视器对象之外，还安排监视器了解源对象，或向源对象注册监视器。每个源对象有一个已注册的监视器列表，提供一个方法向该列表添加监视器对象。在源对象注册了监视器之后，系统会将源对象上发生的事件通知监视器对象。 getSource() 。

5.6 编写一个小程序，小应用程序窗口有一个按钮，点击这个按钮时，点击按钮的次数会显示在按钮上。

```
import java.applet.*;
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class ButtonNumberTest extends Applet
{
    static JFrame mw;
     static JButton buttonx;
     Sqr s=new Sqr();
     public void init()
     {
        mw=new JFrame("按钮点击次数统计");
        mw.setSize(200,150);
       mw.setLayout(new FlowLayout());
        buttonx=new JButton("0");
        mw.add(buttonx);
       buttonx.addActionListener(s);
       mw.setVisible(true);
    }
}
 class Sqr implements ActionListener
 {
   public void actionPerformed(ActionEvent e)
   {
    if(e.getSource()==ButtonNumberTest.buttonx)
    {
     int n=Integer.parseInt(ButtonNumberTest.buttonx.getLabel());
     n=n+1;
     ButtonNumberTest.buttonx.setLabel(String.valueOf(n));
    }
   else{}
  }
 }
```

5.7 创建一个有文本框和三个按钮的程序。当按下某个按钮时，使不同的文字显示在文本框中。

```
import java.applet.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class ButtonTest extends Applet implements ActionListener
```

```
{
 JTextField text;JButton b1,b2,b3;
 public void init()
 {
  setSize(250,150);
  text=new JTextField("",20);
  b1=new JButton("中文");
  b2=new JButton("英文");
  b3=new JButton("数字");
  text.setEditable(false);
  b1.addActionListener(this);
  b2.addActionListener(this);
  b3.addActionListener(this);
  add(text);add(b1);add(b2);add(b3);
 }
 public void actionPerformed(ActionEvent e)
 {
  if(e.getSource()==b1)
     text.setText("中文按钮");
  else if(e.getSource()==b2)
      text.setText("English");
  else if(e.getSource()==b3)
     text.setText("123456");
 }
}
```

5.8 编写一个有两个文本框的小应用程序，在第一个文本框输入英语单词，在第二个文本框会自动显示汉语解释；在第一个文本框输入汉语单词，在第二个文本框中显示英语解释。设英语单词表只有少许几个

```
import java.applet.*;
import javax.swing.*;
import java.awt.event.*;
public class FieldTest extends Applet implements ActionListener
{
 JTextField text1,text2;
 public void init()
 {
  setSize(250,150);
  text1=new JTextField(10);
  text2=new JTextField(10);
  text1.addActionListener(this);
  text2.addActionListener(this);
  text2.setEditable(false);
  text1.requestFocus();
  add(text1);
  add(text2);
 }
 public void actionPerformed(ActionEvent e)
 {
  if(e.getSource()==text1)
  {
```

```
   String s=text1.getText();
   if(s.equals("中国"))
     {text2.setText("China");}
   else if(s.equals("China"))
     {text2.setText("中国");}
   else if(s.equals("美国"))
     {text2.setText("America");}
   else if(s.equals("America"))
     {text2.setText("美国");}
   else
     {text2.setText("查无此单词");}
   }
  }
}
```

5.9 编写有一个标签、一个文本框、一个文本区和两个按钮的小应用程序。当在一个文本区输入若干数据 后，点击求和按钮，在文本框显示输入炎的和，标签显示“输入数的和”；点击求平均值按钮，在文本框显示输入数的平均值，标签显示“输入数的平均值”。要求文本区设有滚动条。

```
import java.util.*;
import java.applet.*;
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
class Panel1 extends JPanel
{
 JLabel label;JTextField textF;
 Panel1()
 {
  label=new JLabel("",JLabel.RIGHT);
   textF=new JTextField(6);
   setLayout(new GridLayout(1,2));
   add(label);
   add(textF);
 }
}
class Panel2 extends JPanel
{
 JButton button1,button2;
 Panel2()
 {
  button1=new JButton("求    和");
  button2=new JButton("求平均值");
  setLayout(new GridLayout(1,2));
  add(button1);add(button2);
 }
}
public class PanelAreaField extends Applet implements ActionListener
{
 Panel1 panel1;
```

```
 Panel2 panel2;
 JTextArea textA;
 public void init()
 {
  setLayout(new GridLayout(3,1));
   setSize(400,200);
   panel1=new Panel1();
   panel2=new Panel2();
   textA=new JTextArea(4,10);
   textA.setLineWrap(true);
   JScrollPane jsp=new JScrollPane(textA);
   add(jsp);
   add(panel1);
   add(panel2);
   (panel2.button1).addActionListener(this);
   (panel2.button2).addActionListener(this);
  }
  public void actionPerformed(ActionEvent e)
  {
   if(e.getSource()==(panel2.button1))
   {
    (panel1.label).setText("输入数的和");
    String s=textA.getText();
    StringTokenizer words=new StringTokenizer(s);
    int n=words.countTokens(),sum=0,i;
    for(i=0;i<=n-1;i++)
    {
     String temp=words.nextToken();
     sum+=Integer.parseInt(temp);
    }
    (panel1.textF).setText(" "+sum);
   }
   else if(e.getSource()==(panel2.button2))
   {
    (panel1.label).setText("输入数的平均值");
    String s=textA.getText();
    StringTokenizer words=new StringTokenizer(s);
    int n=words.countTokens(),i,sum=0;
    for(i=0;i<=n-1;i++)
     {
      String temp=words.nextToken();
      sum+=Integer.parseInt(temp);
     }
     (panel1.textF).setText(" "+sum/n);
    }
   }
}
```

5-10 利用布局嵌套的办法实现如下形状的布局设计

<table>
<tr><td colspan="3">文本区(带滚动条)</td><td>标签
文本框</td><td colspan="2">面板</td><td>面板</td></tr>
<tr><td>按钮
按钮</td><td>面板</td><td>面板</td><td>文本区
(带滚动条)</td><td>面板</td><td>面板</td><td>面板</td></tr>
</table>

```
import java.applet.*;
import javax.swing.*;
import java.awt.*;
class MyPanel extends JPanel
{
 JLabel lab;
 MyPanel(){}
 MyPanel(String s)
 {
  lab=new JLabel(s,JLabel.CENTER);
  add(lab);
 }
}
class Pan1 extends JPanel
{
 JButton b1,b2;
 Pan1()
 {
  b1=new JButton("按钮");
  b2=new JButton("按钮");
  setLayout(new GridLayout(2,1));
  add(b1);
  add(b2);
 }
}

class Pan3 extends JPanel
{
 JLabel la;
 JTextField f;
 Pan3()
 {
  la=new JLabel("标签",JLabel.CENTER);
  f=new JTextField("文本框",5);
  setLayout(new GridLayout(2,1));
  add(la);
  add(f);
 }
}
class Pan4 extends MyPanel
{
 Pan4()
 {
  MyPanel pan41=new MyPanel("面板");
  MyPanel pan42=new MyPanel("面板");
```

```
  add(pan41);
  add(pan42);
 }
}
class Pan5 extends MyPanel
{
 Pan5()
 {
  MyPanel pan51=new MyPanel("面板");
  MyPanel pan52=new MyPanel("面板");
  MyPanel pan53=new MyPanel("面板");
  add(pan51);
  add(pan52);
  add(pan53);
 }
}
class Pan6 extends MyPanel
{
 Pan1 pan1;
 Pan6()
 { pan1=new Pan1();
   MyPanel pan21=new MyPanel("面板");
  MyPanel pan22=new MyPanel("面板");
   setLayout(new GridLayout(1,3));
   add(pan1);
  add(pan21);
  add(pan22);
 }
}
public class SetLayoutTest extends Applet
{
 Pan6 pan6;Pan3 pan3;Pan4 pan4;Pan5 pan5;
 JTextArea a1,a2;
 public void init()
 {
  setSize(400,200);
  setLayout(new GridLayout(2,3));
  pan6=new Pan6();
  pan3=new Pan3();
  pan4=new Pan4();
  pan5=new Pan5();
  a1=new JTextArea("文本区 1",5,10);
  a2=new JTextArea("文本区 2",5,10);
  JScrollPane jsp1=new JScrollPane(a1);
  jsp1.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
  jsp1.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
  JScrollPane jsp2=new JScrollPane(a2);
  jsp2.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
  jsp2.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
```

```
    add(jsp1);add(pan3);add(pan4);
    add(pan6);add(jsp2);add(pan5);
  }
}
```

第 6 章 图形界面设计(二)

本章内容(重点：15%)

6．1 选择框和单选按钮

选择框、单选框和单选按钮都是选择组件，选择组件有两种状态，一种是选中(on)，另一种是未选中(off)，它们提供一种简单的 “on/off”选择功能，让用户在一组选择项目中作选择。

6．1．1 选择框

选择框(JCheckBox)的选中与否开状是一个小方框，被选中则在框中打勾。**当在一个容器中有多个选择框，同时可以有多个选择框被选中，这样的选择框也称复选框**。与选择框相关的接口是 ItemListener，事件类是 ItemEvent。

JCheckBox 类常用的构造方法有以下 3 个：

(1) JCheckBox()，用空标题构造选择框。

(2) JCheckBox(String s),用给定的标题 s 构造选择框。

(3) JCheckBox(String s, boolean b)，用给定的标题 s 构造选择框，参数 b 设置选中与否的初始状态。

JCheckBox 类的其他常用方法如下：

(1) getState(),获取选择框的状态。

(2) setState(boolean b),设置选择框的状态

(3) getLabel()，获取选择框的标题。

(4) setLabel(String s)，设置选择框的标题。

(5) isSelected()，获取选择框是否被选中的状态。

(6) itemStateChanged(ItemEvent e),处理选择框事件的接口方法。

(7) getItemSelectable()，获取可选项。获取事件源。

(8) addItemListener(ItemListener l),为选择框设定监视器。

(9) removeItemListener(ItemListener l),移去选择框的监视器。

[例 6.1]声明一个面板子类，面板子类对象有 3 个选择框。

```
class Panel1 extends JPanel{
    JCheckBox box1,box2,box3;
    Panel1(){
        box1 = new JCheckBox(“足球”);
        box2 = new JCheckBox(“排球”);
        box2 = new JCheckBox(“篮球”);
    }
}
```

6．1．2 单选框

当在一个容器中放入多个选择框，且没有 ButtonGroup 对象将它们分组，则可以同时选中多个选择框。如果使用 ButtonGroup 对象将选择框分组，同一时刻组内的多个选择框只允许有一个被选中，称同一组内的选择框为单选框。单选框分组的方法是先创建 ButtonGroup 对象，然后将希望为同组的选择框添加到同一个 ButtonGroup 对象中。参见例 6.2 程序的面板子类 Panel2 的声明，组内有 3 个单选框。

6．1．3 单选按钮

单选按钮(JRadioButton)的功能与单选框相似。使用单选按钮的方法是将一些单选按钮用 ButtonGroup 对象分组，使同一组的单选按钮只允许有一个被选中。单选按钮与单选框的差异是显示的样式不同，单选按钮是一个圆形的按钮，单选框是一个小方框。

JRadioButton 类的常用构造方法有以下几个：

(1) JRadioButton()，用空标题构造单选按钮。

(2) JRadioButton(String s),用给定的标题 s 构造单选按钮。

(3) JRadioButton(String s,boolean b),用给定的标题 s 构造单选按钮,参数 b 设置选中与否的初始状态。

单选按钮使用时需要使用 ButtonGroup 将单选按钮分组，单选按钮的分组方法是先创建对象，然后将同组的单选按钮添加到同一个 ButtonGroup 对象中。参见例 6.2 程序的子类 panel1 的声明，组内有 3 个单选按钮。

6．1．4 选择项目事件处理

用户对选择框或单选按钮做出选择后，程序应对这个选择作出必要的响应，程序为此要处理选择项目事件。选择项目处理程序的基本内容有：

(1)监视选择项目对象的类要**实现接口** ItemListener,(2)程序要**声明和建立选择对象**，(3)为选择对象**注册监视器**，(4)**编写处理选择项目事件的接口方法** itemStateChanged(ItemEvent e),在该方法内用 getItemSelectable()方法获取事件源，并作相应处理。

[例 6.2]处理选择项目事件的小应用程序。一个由 3 个单选按钮组成的产品选择组，当选中某个产品时，文本区将显示该产品的信息。一个由 3 个选择框组成的购买产品数量选择框组，当选择了购买数量后，在另一个文本框显示每台价格。

```
import java.applet.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class Panel1 extends JPanel{
    JRadioButton box1,box2,box3;
    ButtonGroup g;
    Panel1(){
        setLayout(new GridLayout(1,3));
        g = new ButtonGroup();
        box1 = new JRadioButton(MyWindow.fName[0]+"计算机", false);
        box2 = new JRadioButton(MyWindow.fName[1]+"计算机", false);
        box3 = new JRadioButton(MyWindow.fName[2]+"计算机", false);
        g.add(box1);g.add(box2);g.add(box3);
        add(box1);add(box2);add(box3);
        add(new JLabel("计算机 3 选 1") );
    }
}
class Panel2 extends JPanel{
    JCheckBox box1,box2,box3;
    ButtonGroup g;
    Panel2(){
        setLayout(new GridLayout(1,3));
        g = new ButtonGroup();
        box1 = new JCheckBox("购买 1 台 ");
        box2 = new JCheckBox("购买 2 台 ");
        box3 = new JCheckBox("购买 3 台 ");
        g.add(box1);g.add(box2);g.add(box3);
        add(box1);add(box2);add(box3);
        add(new JLabel(" 选择 1、2 或 3"));
    }
}
class MyWindow extends JFrame implements ItemListener{
    Panel1 panel1;
    Panel2 panel2;
    JLabel label1,label2;
    JTextArea text1,text2;
    static String fName[] = {"HP","IBM","DELL"};
    static double priTbl[][]={{1.20,1.15,1.10},{1.70,1.65,1.60},{1.65,1.60,1.58}};
    static int productin = -1;
    MyWindow(String s)
    {
```

```
        super(s);
        Container con = this.getContentPane();
        con.setLayout(new GridLayout(3,2));
        this.setLocation(100,100);
        this.setSize(400,100);
        panel1 = new Panel1();panel2 = new Panel2();
        label1 = new JLabel("产品介绍",JLabel.CENTER);
        label2 = new JLabel("产品价格",JLabel.CENTER);
        text1 = new JTextArea();text2 = new JTextArea();
        con.add(label1);con.add(label2);con.add(panel1);
        con.add(panel2);con.add(text1);con.add(text2);
        panel1.box1.addItemListener(this);
        panel1.box2.addItemListener(this);
        panel1.box3.addItemListener(this);
        panel2.box1.addItemListener(this);
        panel2.box2.addItemListener(this);
        panel2.box3.addItemListener(this);
        this.setVisible(true);this.pack();
    }
    public void itemStateChanged(ItemEvent e)//选项状态已改变
    {
        if(e.getItemSelectable()==panel1.box1)//获取可选项
        {
            production =0;
            text1.setText(fName[0]+"公司生产");text2.setText("");
        }
        else if(e.getItemSelectable()==panel1.box2)
        {
            production =1;
            text1.setText(fName[1]+"公司生产");text2.setText("");
        }
        else if(e.getItemSelectable()==panel1.box3)
        {
            production =2;
            text1.setText(fName[2]+"公司生产");text2.setText("");
        }
        else
        {
            if(production ==-1) return;
            if(e.getItemSelectable()==panel2.box1)
            {
                text2.setText(""+priTbl[production][0]+"万元/台");
            }
            else if(e.getItemSelectable()==panel2.box2)
            {
                text2.setText(""+priTbl[production][1]+"万元/台");
            }
            else if(e.getItemSelectable()==panel2.box3)
            {
```

```
                    text2.setText(""+priTbl[production][2]+"万元/台");
                }
            }
        }
    }
    public class Example6_2 extends Applet{
        MyWindow myWin = new MyWindow("选择项目处理示例程序");
    }
```

6.2 列表和组合框

列表和组合框是又一类供用户选择的界面组件，用于在一组选择项目选择，组合框还可以输入新的选择。

6．2．1 列表

列表(JList)在界面中表现为列表框，是 JList 类或它的子类的对象。程序可以在列表框中加入多个文本选择项条目。列表事件的事件源有两种：一是鼠标双击某个选项，二是鼠标单击某个选项。

双击选项是动作事件，与该事件相关的接口是 ActionListener，注册监视器的方法是 addActionListener()，接口方法是 actionPerformed(ActionEvent e)。

单击选项是选项事件，与选项事件相关的接口是 ListSelectionListener,注册监视器的方法是 addListSelectionListener,接口方法是 valueChanged(ListSelectionEvent e)。

JList 类的常用构造方法：

(1) JList(),建立一个列表。

(2) JList(String list[]),建立列表，list 是字符串数组，数组元素是列表的选择条目。

JList 类的常用方法：

(1) getSelectedIndex(),获取选项的索引。返回最小的选择单元索引；只选择了列表中单个项时，返回*该选择*。

(2) getSelectedValue(),获取选项的值。

(3) getSelectedIndices()，返回所选的全部索引的数组（按升序排列)。

(4) getSelectedValues()， 返回所有选择值的数组，根据其列表中的索引顺序按升序排序。

(5) getItemCount(),获取列表中的条数。

(6) setVisibleRowCount(int n),设置列表可见行数。

(7) setSelectionMode(int seleMode),设置列表选择模型。选择模型有单选和多选两种。

单选：ListSelectionModel.SINGLE_SELECTION.

多选：ListSelectionModel.MULTIPLE.INTERVAL_SELECTION.

(8) remove(int n),从列表的选项菜单中删除指定索引的选项。

(9) removeAll()，删除列表中的全部选项。

列表可以添加滚动条，列表添加滚动条的方法是先创建列表，然后再创建一个 JScrollPane 滚动面板对象，在创建滚动面板对象时指定列表。以下代码示意为列表 list2 添加滚动条：

```
JScrollPane jsp = new JScrollPane(list2);
```

[例 6.3]小应用程序有两个列表，第一个列表只允许单选，第二个列表允许多选。

```
import java.applet.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class MyWindow extends JFrame implements ListSelectionListener{
    JList list1,list2;
    String news[]={"人民日报","新民晚报","浙江日报","文汇报"};
    String sports[]={"足球","排球","乒乓球","篮球"};
    JTextArea text;
    MyWindow(String s){
        super(s);
        Container con = getContentPane();
        con.setBackground(Color.BLUE);
```

```
            con.setLayout(new GridLayout(2,2));
            con.setSize(200,500);
            list1 = new JList(news);
            list1.setVisibleRowCount(3);
            list1.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
            list1.addListSelectionListener(this);
            list2 = new JList(sports);
            list2.setVisibleRowCount(2);
            list2.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
            list2.addListSelectionListener(this);
            con.add(list1);
            con.add(list2);
            text= new JTextArea(10,20);
            con.add(text);
            this.setVisible(true);
            this.pack();
        }
        public void valueChanged(ListSelectionEvent e){// 每当选择值发生更改时调用
            if(e.getSource()==list1){
                text.setText(null);
                Object listValue = ((JList) e.getSource()).getSelectedValue();
                String seleName = listValue.toString();
                for(int i=0;i<news.length;i++)
                if(news[i].equals(seleName)){
                    text.append(seleName+ "被选中\n");
                }
            }
            else if(e.getSource()==list2){
                text.setText(null);
                int tempList[] =list2.getSelectedIndices();
                for(int i=0;i<tempList.length;i++)
                text.append(sports[tempList[i]] + "被选中\n");
            }
        }
    }
    public class Example6_3 extends Applet{
        MyWindow myWin = new MyWindow("列表示例");
    }
```

6.2.2 组合框

组合框(JComboBox)是文本框和列表的组合，可以在文本框中输入选项，也可以单击下拉按钮从显示的列表中进行选择。

组合框的常用构造方法：

(1) JComboBox(),建立一个没有选项的 JComboBox 对象。

(2) JComboBox(JComboBoxModel aModel),用数据模型建立一个 JComboBox 对象。

(3) JComboBox(Object[]items),利用数组对象建立一个 JComboBox 对象。

组合框的其他常用方法有以下几个：

(1) addItem(Object obj),向组合框加选项。

(2) getItemCount(),获取组合框的条目总数。

(3) removeItem(Object ob),删除指定选项。

(4) removeItemAt(int index),删除指定索引的选项。

(5) insertItemAt(Object ob,int index),在指定的索引处插入选项。

(6) getSelectedIndex(),获取所选项的索引值(从 0 开始)。

(7) getSelectedItem(),获得所选项的内容。

(8) setEditable(boolean b),设为可编辑。组合框的默认状态是不可编辑的，需要调用本方法设定为可编辑，才能响应选择输入事件。

在 JComboBox 对象上发生事件分为两类。**一是用户选定项目，事件响应程序获取用户所选的项目。二是用户输入项目后按回车键，事件响应程序读取用户的输入。**第一类事件的接口是 ItemListener;第二类事件是输入事件,接口是 ActionListener。

[例 6.4]一个说明组合框用法的应用程序。程序中声明的组合框子类实现 ItemLister 接口和 ActionListener 接口。组合框子类的窗口中设置了一个文本框和一个组合框，组合框中有三个选择。实现接口的监视方法将组合框的选择结果在文本框中显示。

```
public class Example6_4{
    public static void main(String args[]){
        ComboBoxDemo mycomboBoxGUI = new ComboBoxDemo();
    }
}
class ComboBoxDemo extends JFrame implements ActionListener,ItemListener{
    public static final int Width = 350;
    public static final int Height = 150;
    String proList[] = { "踢足球","打篮球","打排球" };
    JTextField text;
    JComboBox comboBox;
    public ComboBoxDemo(){
        setSize(Width,Height);
        setTitle("组合框使用示意程序");
        Container conPane = getContentPane();
        conPane.setBackground(Color.BLUE);
        conPane.setLayout(new FlowLayout());
        comboBox = new JComboBox(proList);
        comboBox.addActionListener(this);
        combobox.addItemListener(this);
        comboBox.setEditable(true);//响应键盘输入
        conPane.add(comboBox);
        text = new JTextField(10);
        conPane.add(text);
        this.setVisible(true);
    }
    public void actionPerformed(ActionEvent e){
        if(e.getSource()==comboBox)
            text.setText(comboBox.getSelectedItem().toString());
    }
    public void itemStateChanged(ItemEvent e){
        if(e.getSource()==comboBox){
            text.setText(comboBox.getSelectedItem().toString());
        }
    }
}
```

6．3 菜单

有两种类型的菜单：下拉式菜单和弹出式菜单。本书只讨论下拉式菜单编程方法。菜单与 JComboBox 和 JCheckBox 不同，它们在界面中是一直可见的。菜单与 JComboBox 的相同之处是每次只可选择一个项目。

在下拉式菜单或弹出式菜单中选择一个选项就产生一个 ActionEvent 事件。该事件被发送给那个选项的监视器，事件的意义由监视器解释。

6．3．1 菜单条、菜单和菜单项

下拉式菜单通过出现在菜单条上的名字可视化表示，菜单条(JMenuBar)通常出现在 JFrame 的顶部，一个菜单条显示多个下拉式菜单的名字。可以用两种方式来激活下拉式菜单。一种是按下鼠标的按钮，并保持按下状态，移动鼠标，直至释放鼠标完成选择，高亮度显示的菜单项即为所选择的。另一种方式是当光标位于菜单条中的菜单名上时，点击鼠标，在这种情况下，菜单会展开，且高亮度显示菜单项。

一个菜单条可以放多个菜单(JMenu)，每个菜单又可以有许多菜单项(JMenuItem).例如，Eclipse 环境的菜单条有 File、Edit、Source、Refactor 等菜单，每个菜单又有许多菜单项。例如，File 菜单有 New、Open File、Close 、Close All 等菜单项。

向窗口增设菜单的方法是：先创建一个菜单条对象，然后再创建若干菜单对象，把这些菜单对象放在菜单条里，再按要求为每个菜单对象添加菜单项。

菜单中的菜单项也可以是一个完整的菜单。由于菜单项又可以是另一个完整菜单，因此可以构造一个层次状菜单结构。

1． 菜单条

类 JMenuBar 的实例就是菜单条。例如，以下代码创建菜单条对象 menubar:

```
JMenuBar menubar = new JMenuBar();
```

在窗口中增设菜单条，必须使用 JFrame 类中的 setJMenuBar()方法。例如，代码：

```
setJMenuBar(menubar);
```

类 JMenuBar 的常用方法有：

(1) add(JMenu m),将菜单 m 加入到菜单条中。
(2) countJMenus(),获得菜单条中菜单条数。
(3) getJMenu(int p),取得菜单条中的菜单。
(4) remove(JMenu m),删除菜单条中的菜单 m。

2． 菜单

由类 JMenu 创建的对象就是菜单。类 JMenu 的常用方法如下：

(1) JMenu()，建立一个空标题的菜单。
(2) JMenu(String s),建立一个标题为 s 的菜单。
(3) add(JMenuItem item),向菜单增加由参数 item 指定的菜单选项。
(4) add(JMenu menu),向菜单增加由参数 menu 指定的菜单。实现在菜单嵌入子菜单。
(5) addSeparator(),在菜单选项之间画一条分隔线。
(6) getItem(int n)，得到指定索引处的菜单项。
(7) getItemCount(),得到菜单项数目。
(8) insert(JMenuItem item,int n),在菜单的位置 n 插入菜单项 item.
(9) remove(int n),删除菜单位置 n 的菜单项
(10) removeAll(),删除菜单的所有菜单项。

3． 菜单项

类 JMenuItem 的实例就是菜单项。类 JMenuItem 的常用方法如下：

JMenuItem()构造无标题的菜单项。
JMenuItem(String s),构造有标题的菜单项。
setEnabled(boolean b),设置当前单项是否可被选择。
isEnabled(),返回当前菜单项是否可被用户选择。
getLabel(),得到菜单项的名称。
setLabel(),设置菜单选项的名称。
addActionListener(ActionListener e),为菜单项设置监视器。监视器接受点击某个菜单的动作事件。

4． 处理菜单事件

菜单的事件源是用鼠标点击某个菜单项。处理该事件的接口是 ActionListener,要实现的接口方法是 actionPerformed(ActionEvent e),获得事件源的方法 getSource().

[例 6.5]小应用程序示意窗口有菜单条的实现方法。设有一个按钮，当按钮处于打开窗口状态时，单击按钮将打开一个窗口，窗口设有一个菜单条，有两个菜单，每个菜单又各有三个菜单项。当一个菜单项被选中时，菜单项监视方法在文本框中显示相应菜单项被选中字样。

```
import java.applet.*
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class MenuWindow extends JFrame implements ActionListener{
    public static JtextField text;
    private void addItem(JMenu Menu,String menuName,ActionListener listener){
        JMenuItem anItem = new JMenuItem(menuName);
        anItem.setActionCommand(menuName);
        anItem.addActionListener(listener);
        Menu.add(anItem);
    }
    public MenuWindow(String s,int w,int h){
        setTitle(s);
        Container con = this.getContentPane();
        con.setLayout(new BorderLayout());
        this.setLocation(100,100);
        this.setSize(w,h);
        JMenu menu1 = new JMenu("体育");
        addItem(menu1," 跑步",this);
        addItem(menu1," 跳绳",this);
        addItem(menu1,"打球",this);
        JMenu menu2 = JMenu("娱乐");
        addItem(menu2,"唱歌",this);
        addItem(menu2,"跳舞",this);
        addItem(menu2,"游戏",this);
        JMenuBar menubar = new JMenuBar();
        text = new JTextField();
        menubar.add(menu1);
        menubar.add(menu2);
        setJMenuBar(MenuBar);
        con.add(text,BorderLayout.NORTH);
    }
    public void actionPerformed(ActionEvent e){
        text.setText(e.getActionCommand()+"菜单项被选中!");
    }
}
public class Example6_5 extends Applet implements ActionListener{
    MenuWindow window;
    JButton button;
    boolean bflg;
    public void init(){
        button = new JButton("打开我的体育娱乐之窗");bflg =true;
        window = new MenuWindow("体育娱乐之窗",100,100);
        button.addActionListener(this);
        add(button);
```

```
    }
    public void actionPerformed(ActionEvent e){
        if(e.getSource()==button){
            if(bflg){
                window.setVisible(true);
                bflg = false;
                button.setLabel("关闭我的体育娱乐之窗");
            }
            else{
                window.setVisible(false);
                bflg = true;
                button.setLabel("打开我的体育娱乐之窗");
            }
        }
    }
}
```

5. 嵌入子菜单

创建了一个菜单，并创建多个菜单项，其中某个菜单项又是一个(含其他菜单项的)菜单，这就构成菜单嵌套。例如，将上述程序中的有关代码改成如下：

```
    Menu menu1,menu2,item4;
    MenuItem item3,item5,item6,item41,item42;
```

另插入以下代码创建 item41 和 item42 菜单项，并把它们加入到 item4 菜单中：

```
    item41= new MenuItem(“东方红”);
    item42 = new MenuItem(“牡丹”);
    item4.add(item41);
    item4.add(item42);
```

则点击 item4 菜单时，又会打开两个菜单项供选择。

6. 向菜单增加退出项

增设一个新的菜单项，对该菜单项加入监视，对应的监视方法中使用 System.exit()方法，就能实现单击该菜单项时退出 Java 运行环境。例如，以下示意代码：

```
…
item7 = new MenuItem(“退出”);
item7.addActionListener(this);
…
public void actionPerformed(ActionEvent e){
    if(e.getSource()==item7)
    {
        System.exit(0);
    }
}
```

7.设置菜单项的快捷键

用 MenuShortcut 类为菜单项设置快捷键。构造方法是 MenuShortcut(int key)。其中 key 可以取值 KeyEvent.VK_A 至 KenEvent.VK_Z,也可以取 ‘a’到 ‘z’键码值。菜单项使用 setShortcut(MenuShortcut k)方法来设置快捷键。例如，以下代码设置字母 e 为快捷键。

```
class Herwindow extends Frame implements ActionListener{
    MenuBar menbar;
    Menu menu;
    MenuItem item;
    MenuShortcut shortcut = new MenuShortcut(KeyEvent.VK_E);
```

…

item.setShortcut(shortcut);

…

}

6.3.2 选择框菜单项

菜单也可以包含具有持久的选择状态的选项，这种特殊的菜单可由 JCheckBoxMenuItem 类来定义。JCheckBoxMenuItem 对象像选择框一样，能表示一个选项被选中与否，也可以作为一个菜单项加到下拉菜单中。点击 JCheckBoxMenuItem 菜单时，就会在它的左边出现打勾符号或清除打勾符号。例如，在例 6.5 程序的类 MenuWindow 中，将代码

addItem(menu1,“跑步”,this);addItem(menu1,”跳绳”,this);

改写成以下代码，就将两个普通菜单项“跑步“和“跳绳”改成两个选择框菜单项：

```
JCheckBoxMenuItem item1 = new JCheckBoxMenuItem(“跑步”);
JCheckBoxMenuItem item2 = new JCheckBoxMenuItem(“跳绳”);
item1.setActionCommand(“跑步”);
item1.addActionListener(this);
menu1.add(item1);
item2.setActionCommand(“跳绳”);
item2.addActionListener(this);
menu1.add(item2);
```

6.4 对话框

对话框是为了人机对话过程提供交互模式的工具。应用程序通过对话框，或给用户提供信息，或从用户获得信息。对话框是一个临时窗口，可以在其中放置用于得到用户输入的控件。在 Swing 中，有两个对话框类，它们是 JDialog 类和 JOptionPane 类。JDialog 类提供构造并管理通用对话框;JOptionPane 类给一些常见的对话框提供许多便于使用的选项,例如,简单的“yes-no”对话框等。

6．4．1 JDialog 类

JDialog 类作作对话框的基类。对话框与一般窗口不同，对话框依赖其他窗口，当它所依赖的窗口消失或最小化时，对话框也将消失；窗口还原时，对话框又会自动恢复。

对话框分为强制和非强制两种。强制型对话框不能中断对话过程，直至对话框结束，才让程序响应对话框以外的事件。非强制型对话框可以中断对话过程，去响应对话框以外的事件。**强制型也称有模式对话框，非强制对话框也称非模式对话框。**

JDialog 对象也是一种容器，因此也可以给 JDialog 对话框指派布局管理器，对话框的默认布局为 BoarderLayout 布局。**但组件不能直接加到对话框中**，对话框也包含一个内容面板，应当把组件加到 JDialog 对象的内容面板中。**由于对话框依赖窗口，因此要建立对话框，必须先要创建一个窗口。**

JDialog 类常用的构造方法有 3 个：

(1) JDialog(),构造一个初始化不可见的非强制型对话框。

(2) JDialog(JFrame f,String s),构造一个初始化不可见的非强制型对话框，参数 f 设置对话框所依赖的窗口，参数 s 用于设置标题。通常先声明一个 JDialog 类的子类，然后创建这个子类的一个对象，就建立了一个对话框。

(3) **JDialog(JFrame f,String s,boolean b),构造一个标题为 s，初始化不可见的对话框。参数 f 设置对话框所依赖的窗口，**参数 b 决定对话框是否强制或非强制型。

JDialog 类的其他常用方法有以下几个：

(1) getTitle(),获取对话框的标题。

(2) setTitle(String s),设置对话框的标题。

(3) setModal(boolean b)，设置对话框的模式。

(4) setSize(),设置框的大小。

(5) setVisible(boolean b),显示或隐藏对话框。

[例 6.6]小应用程序声明一个用户窗口类和对话框类，用户窗口有两个按钮和两个文本框，当点击某个按钮时，对应的对话框被激活。在对话框中输入相应信息，按对话框的确定按钮。确定按钮的监视方法，将对话框中输入的信息传送给用户窗口，并在用户窗口的相应文本框中显示选择信息。

```
import java.applet.*
import javax.swing.*;
```

```
import java.awt.*;
import java.awt.event.*;
class MyWindow extends JFrame implements ActionListener{
    private JButton button1,button2;
    private static int flg=0;
    private static JTextField text1,text2;
    Mywindow(String s){
        super(s);
        Container con = this.getContentPane();
        con.setLayout(new GridLayout(2,2));
        this.setSize(200,100);
        this setLocation(100,100);
        button1 = new JButton("选择水果");
        button2 = new JButton("选择食品");
        button1.addActionListener(this);
        button2.addActionListener(this);
        text1 = new JTextField(20);
        text2 = new JTextField(20);
        con.add(button1);
        con.add(button2);
        con.add(text1);
        con.add(text2);
        this.setVisible(true);
        this.pack();
    }
    public static void returnName(String s){
        if(flg ==1)
            text1.setText("选择的水果是："+s);
        else if(flg == 2)
            text2.setText("选择的食品是："+s);
    }
    public void actionPerformed(ActionEvent e){
        MyDialog dialog;
        if(e.getSource()==button1)
        {
            dialog = new MyDialog(this,"水果");
            dialog.setVisible(true);
            flg =1;
        }
        else if(e.getSource()==button2)
        {
            dialog =new MyDialog(this,"食品");
            dialog.setVisible(true);
            flg=2;
        }
    }
}
class MyDialog extends JDialog implements ActionListener{
    JLabel title;
```

```
    JTextField text;
    JButton done;
    Mydialog(JFrame F,String s){
        super(F,s,true);//模态
        Container con = this.getContentPane();
        title = new JLabel("输入"+s+"名称");
        text = new JTextField(10);
        text.setEditable(true);
        con.setLayout(new FlowLayout());
        con.setSize(200,100);
        setModal(false);
        done = new JButton("确定");
        done.addActionListener(this);
        con.setVisible(true);
        this.pack();
    }
    public void actionPerformed(ActionEvent e){
        MyWindow.returnName(text.getText());
        setVisible(false);
        dispose();
    }
}
public class Example6_6 extends Applet{
    MyWindow window;
    MyDialog dialog;
    public void init(){
        window = new MyWindow("带对话框窗口");
    }
}
```

上述例子创建的是强制型对话框，改为非强制型对话框就允许用户在对话过程中暂停，与程序的其他部分进行交互。这样，在界面中可以看到部分对话的效果。

将上述例子改为非强制型对话框只要作少量的改动即可。首先是将对话框构造方法中的代码“super(F,s,true);”改为“super(F,s,false);”。

第二个变化：原来是响应确定按钮事件时，才调用方法returnName()，将对话框得到的字符串返回给程序。现在当文本框输入选择字符串结束后，就应该立即调用该方法。为此，需要对文本框的输入事件作监视，为文本注册监视器：

```
    public void actionPerformed(ActionEvent e){
        if(e.getSource()==text){
            MyWindow.returnName(text.getText());
        }
        else if(e.getSource()==done){
            MyWindow.returnName(text.getText());
            setVisible(false);
            dispose();//清除资源
        }
    }
```

6.4.2 JOptionPane 类

经常遇到非常简单的对话情况，为了简化常见对话框的编程，JOptionPane 类定义了四个简单对话框类型，参见表 6-1。JOptionPane 类提供一组静态方法，让用户选用某种类型的对话框。下面的代码是选用确认对话框：

int result = JOptionPane.showConfirmDialog(parent,“确实要退出吗”, “退出确认“, JOptionPane.YES_NO_CANCEL_OPTION);

其中方法名的中间部分文字“Confirm”是创建对话框的类型，文字 Confirm 指明是选用确认对话框。将文字 Confirm 改为另外三种类型的某一个，就成为相应类型的对话框。上述代码的四个参数的意义是：第一个参数指定这个对话框的父窗口；第二个参数是对话框显示的文字；第三个参数是对话框的标题；最后一个参数指明对话框有三个按钮，分别为“是(Y)”，“否(N)”，和“撤销”。方法的返回结果是用户响应了这个对话框后的结果，参见表 6-2 给出的可能答案。

如果程序只要求输入一行信息，创建输入对话框的代码早在第 2 章已有叙述。输入对话框以列表或文本框形式请求用户输入选择信息，用户可以从列表中选择选项或从文本框中输入信息。以下是一个从列表中选择运行项目的输入对话框的示意代码：

```
String result = (String)JOptionPane.showInputDialog(parent,
                “请选择一项运动项目”, “这是运动项目选择对话框”,
                    JOptionPane.QUESTION_MESSAGE,null,
                new Object[]{“踢足球“, “打篮球”, “跑步”, “跳绳”}, “跑步”);
```

第四个参数是信息类型，参见 6-3，第五个参数在这里没有特别的作用，总是用 null;第六个参数定义了一个供选择的字符串数组，第七个参数是选择的默认值。对话框还包括“确定”和“撤销”按钮。

表 6-1 JOptionPane 对话框类型

输入	通过文本框、列表或其他手段输入，另有“确定”和“撤销”按钮
确认	提出一个问题，待用户确认，另有“是(Y)”、“否(N)”和“撤销”按钮
信息	显示一条简单的信息，另有“确定”和“撤销”按钮
选项	显示一列供用户选择的选项

表 6-2 由 JOptionPane 对话框返回的结果

YES_OPTION	用户按了“是(Y)”按钮
NO_OPTION	用户按了“否(N)”按钮
CANCEL_OPTION	用户按了“撤销”按钮
OK_OPTION	用户按了“确定”按钮
CLOSED_OPTION	用户没按任何按钮，关闭对话框窗口

表 6-3 JOptionPane 对话框的信息类型选项

PLAIN_MESSAGE	不包括任何图标
WARNING_MESSAGE	包括一个警告图标
QUESTION_MESSAGE	包括一个问题图标
INFORMATIN_MESSAGE	包括一个信息图标
ERROR_MESSAGE	包括一个出错图标

有时，程序只是简单地输出一些信息，并不要求用户有反馈。这样的对话框可用以下形式的代码创建：

```
JOptionPane.showMessageDialog(parent, “这是一个 Java 程序”,
    “我是输出信息对话框”, JOptionPane.PLAIN_MESSAGE);
```

上述代码中前三参数的意义与前面所述相同，最后参数是指定信息类型为不包括任何图标，参见表 6-3。

6．5 滚动条

滚动条(JScrollBar)也称为滑块，用来表示一个相对值，该值代表指定范围内的一个整数。例如，用 Word 编辑文档时，编辑窗右边的滑块对应当前编辑位置在整个文档中的相对位置，可以通过移动选择新的编辑位置。在 Swing 中，用 JScrollBar 类实现和管理可调界面。JScrollBar 类常用的构造方法是：

```
JScrollBar(int dir,int init,int width,int low,int high)
```

其中，dir 表示滚动条的方向。JScrollBar 类定义了两个常量，JScrollBar.VERTICAL 表示垂直滚动条；JScrollBar.HORIZONTAL 表示水平滚动条。init 表示滚动条的初始值，该值确定滚动条滑块开始时的位置；width 是滚动条滑块的宽度；最后两个参数指定滚动的下界和上界。注意滑块的宽度可能影响滚动条可得到的实际的最大值。例如，滚动条的范围是 0 至 255，滑块的宽度是 10，并利用滑块的左端或顶端来确定它的实际位置。那么滚动条可以达到的最大值是指定最大值减去滑块的宽度。所以滚动条的值不会超过 245。

JScrollBar 类其他常用方法是：

(1) setUnitIncrement(),设置增量，即单位像素的增值。

(2) getUnitIncrement(),获取增量。

(3) setBlockIncrement(),设置滑块增量，即滑块的幅度。

(4) getBlockIncrement(),获取滑块增量。

(5) setMaxinum()，设置最大值。

(6) getMaxinum()，获取最大值。

(7) setMininum(),设置最小值

(8) getMininum()，获取最小值

(9) setValue(),设置新值

(10) getValue()，获取当前值。

JScrollBar 类对象的事件类型是 AdjustmentEvent；类要实现的接口是 AdjustmentListener,接口方法是 adjustmentValueChanged();注册监视器的方法是 addAdjustmentListener();获取事件源对象的方法是 getAdjustable().

[例 6.7]应用程序将滚动条作为值的选择。容器有一个开/关滚动条的按钮，一个文本框和一个滚动条，当滚动条处于打开状态时，移动滚动条上的滑块，滑块的对应值显示在文本框中。如果滚动条处于关闭状态 ，则移动滚动条上的滑块，滑块的对应值在文本框中不显示。

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class MyScrollBar extends JScrollBar{
    public MyScrollBar(int init,int len, int low, int high){
            super(JScrollBar.HORIZONTAL,init,len,low,high);
    }
    public Dimension getPreferredSize(){
        return new Dimension(125,20);
    }
}
class MyWindow extends JFrame implements ActionListener, AdjustmentListener{
    private JButton Button;
    private JTextField text;
    private boolean barOpened;
    MyWindow(String s){
        super(s);
        MyScrollBar tempBar = new MyScrollBar(10,10,0,255);
        Container con = this.getContentPane();
        con.setLayout(new GridLlayout(2,1));
        this.setSize(200,100);
        this.setLocatin(100,100);
        Button = new JButton("开/闭滚动条");
        Button.addActionListener(this);
        barOpened = false;
        tempBar.addAdjustmentListener(this);
        text = new JTextField("滚动条关闭",20);
        con.add(Button);
        con.add(text);
        con.add(tempBar);
        this.setVisible(true);
        this.pack();
    }
```

```
    public void actionPerformed(ActionEvent e){
        if(e.getSource()== Button){
            if(barOpened)text.setText("滚动条关闭");
            else text.setText("滚动条打开");
            barOpened !=barOpened;
        }
    }
    public void adjustmentValueChanged(AdjustmentEvent e){
        if(barOpened){
            MyScrollBar myBar = (MyScrollBar) e.getAdjustable();
            text.setText("选择的值是："+MyBar.getValue());
        }
    }
}
public class Example6_7{
    public static void main(String[] args){
        MyWindow myWindow = new MyWindow("滚动条实例");
    }
}
```

MyScrollBar 类定义的方法 getPreferedSize()也是 Component 类中定义的方法，界面组件通过覆盖定义该方法确定界面组件的大小。当布局字处理器在安排组件布局时，就会调用该方法来确定组件的大小。这个方法返回一个 Dimension 类型的对象，Dimension 对象含两个整数，分别为组件的宽和高。在上述程序中，为滚动条指派的区域的宽是 125 像素，高是 20 个像素。任何组件都可用上述方法来指定大小。滚动条应用的更多实例参见 6.6.2 的例 6.9。

6．6 鼠标事件

鼠标事件的事件源往往与容器相关，当鼠标进入容器、离开容器，或者在容器中单击鼠标、拖动鼠标时都会发生鼠标事件。java 语言为处理鼠标事件提供两个接口：MouseListener，MouseMotionListener 接口。

6．6．1 MouseListener 接口

MouseListener 接口能处理 5 种鼠标事件：按下鼠标，释放鼠标，点击鼠标、鼠标进入、鼠标退出。相应的方法有：

(1) getX(),鼠标的 X 坐标

(2) getY(),鼠标的 Y 坐标

(3) getModifiers(),获取鼠标的左键或右键。

(4) getClickCount(),鼠标被点击的次数。

(5) getSource()，获取发生鼠标的事件源。

(6) addMouseListener(监视器),加放监视器。

(7) removeMouseListener(监视器)，移去监视器。

要实现的 MouseListener 接口的方法有：

(1) mousePressed(MouseEvent e);

(2) mouseReleased(MouseEvent e);

(3) mouseEntered(MouseEvent e);

(4) mouseExited(MouseEvent e);

(5) mouseClicked(MouseEvent e);

[例 6.8]小应用程序设置了一个文本区，用于记录一系列鼠标事件。当鼠标进入小应用程序窗口时，文本区显示“鼠标进来”；当鼠标离开 窗口时，文本区显示“鼠标走开”；当鼠标被按下时，文本区显示“鼠标按下”，当鼠标被双击时，文本区显示“鼠标双击”；并显示鼠标的坐标。程序还显示一个红色的圆，当点击鼠标时，圆的半径会不断地变大。

[例 6.8]小应用程序设置了一个文本区，用于记录一系列鼠标事件。当鼠标进入小应用程序窗口时，文本区显示"鼠标进来";当鼠标离开窗口时，文本区显示"鼠标走开"鼠标被按下时，文本区显示"鼠标键按下"鼠标双击时，文本区显示"鼠标双击"；并显示鼠标的坐标。程序还显示一个红色的圆，当点击鼠标时，中=圆的半径会不断变大。

```
import java.applet.*;
```

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class MyPanel extends JPanel{
    public void print(int r){
        Graphics g = getGraphics();
        g.clearRect(0,0,this.getWidth(),this.getHeight());
        g.setColor(Color.red);
        g.fillOval(10,10,r,r);
    }
}
class MyWindow extends JFrame implements MouseListener{
    JTextArea text;
    MyPanel panel;
    int x,y,r =10;
    int mouseFlg=0;
    static String mouseStates[]={"鼠标键按下","鼠标松开","鼠标进来","鼠标走开","鼠标双击"};
    MyWindow(String s){
        super(s);
        Container con = this.getContentPane();
        con.setLayout(new GridLayout(2,1));
        this.setSize(200,300);
        this.setLocation(100,100);
        panel = new MyPanel();
        con.add(panel);
        text = new JTextArea(10,20);
        text.setBackground(Color.blue);
        con.add(text);
        addMouseListener(this);
        this.setVisible(true);
        this.pack();
    }
    public void paint(Graphics g){
        r = r+4;
        if(r>80)
        {
            r=10;
        }
        text.append(mouseStates[mouseFlg]+"了，位置是：" +x+","+y+"\n");
        panel.print(r);
    }
    public void mousePressed(MouseEvent e){
        x = e.getX();
        y = e.getY();
        mouseFlg = 0;
        repaint();
    }
    public void mouseRelease(MouseEvent e){
        x = e.getX();
```

```
            y = e.getY();
            mouseFlg = 1;
            repaint();
        }
        public void mouseEntered(MouseEvent e){
            x = e.getX();
            y = e.getY();
            mouseFlg = 2;
            repaint();
        }
        public void mouseExited(MouseEvent e){
            x = e.getX();
            y = e.getY();
            mouseFlg = 3;
            repaint();
        }
        public void mouseClicked(MouseEvent e){
            if(e.getClickCount()==2)
            {
            x = e.getX();
            y = e.getY();
            mouseFlg = 4;
            repaint();
            }
            else
            {}
        }
}
public class Example6_8 extends Applet{
        public void init(){
            MyWindow myWnd = new MyWindow("鼠标事件示意程序");
        }
}
```

任何组件上都可以发生鼠标事件：鼠标进入、鼠标退出、按下鼠标等。例如，在上述程序中添加一个按钮，并给按钮对象添加鼠标监视器，将上述程序中的 init()方法修改成如下形式，即能示意按钮上的所有鼠标事件。

```
        JButton button;
        public void init(){
            button = new JButton(“按钮也能发生鼠标事件”);
            r = 10;
            text = new JTextArea(15,20);
            add(button);
            add(text);
            button.addMouseListener(this);
        }
```

如果程序希望进一步知道按下或点击的是鼠标左键或右键，鼠标的左键或右键可用 InputEvent 类中的常量 BUTTON1_MASK 和 BUTTON3_MASK 来判定。例如，以下表达式判断是否按下或点击了鼠标右键：

```
e.getModifiers()==InputEvent. BUTTON3_MASK
```

6.6.2 MouseMotionListener 接口

MouseMotionListener 接口处理拖动鼠标和鼠标移动两种事件。

注册监视器的方法是：

addMouseMotionListener(监视器)

要实现的的接口方法有两个：

(1) mouseDragged(MouseEvent e)

(2) mouseMoved(MouseEvent e)

例 6.9 一个滚动条与显示窗口同步变化的应用程序。窗口有一个方块，用鼠标拖运方块，或用鼠标点击窗口，方块改变显示位置，相应水平和垂直滚动条的滑块也会改变它们在滚动条中的位置。反之，移动滚动条的滑块，方块在窗口中的显示位置也会改变。

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class MyWindow extends JFrame{
    public MyWindow(String s){
        super(s);
        Container con = this.getContentPane();
        con.setLayout(new BorderLayout());
        this.setLocation(100,100);
        JScrollBar xAxis = new JScrollBar(JScrollBar.HORIZONTAL,50,1,0,100);
        jScrollBar yAxis = new jScrollBar(JScrollBar.VERTICAL,50,1,0,100);
        MyListener listener = new MyListener(xAxis,yAxis,238,118);
        Jpanel scrolledCanvas = new JPanel();
        scrolledCanvas.setLayout(new BorderLayout());
        scrolledCanvas.add(listener,BorderLayout.CENTER);
        scrolledCanvas.add(xAix,BorderLayout.SOUTH);
        scrolledCanvas.add(yAix,BorderLayout.EAST);
        con.add(scrolledCanvas,BorderLayout.NORTH);
        this.setVisible(true);
        this.pack();
    }
    public Dimension getPreferredSize(){
        return new Dimension(500,300);
    }
}
class MyListener extends JComponent implements MouseListener, MouseMotionListener,AdjustmentListener{
    private int x,y;
    private JScrollBar xScrollBar;
    private JScrollBar yScrollBar;
    private void updateScrollBars(int x,int y){
        int d;
        d = (int)(((float)x/(float)getSize().width)*100.0);
        xScrollBar.setValue(d);
        d = (int)(((float)y/(float)getSize().height)*100.0);
        yScrollBar.setValue(d);
    }
    public MyListener(JScrollBar xaxis,JScrollBar yaxis,int x0,int y0){
        xScrollBar =xaxis;
        yScrollBar =yaxis;
```

```
        x = x0;
        y=y0;
        xScrollBar.addAdjustmentListener(this);
        yScrollBar.addAdjustmentListener(this);
        this.addMouseListener(this);
        this.addMouseMotionListener(this);
    }
    public void paint(Graphics g){
        g.setColor(getBackground());
        Dimension size = getSize();
        g.fillRect(0,0,size.width,size.height);
        g.setColor(Color.blue);
        g.fillRect(x,y,50,50);
    }
    public void mouseEntered(MouseEvent e){}
    public void mouseExited(MouseEvent e){}
    public void mouseClicked(MouseEvent e){}
    public void mouseRelease(MouseEvent e){}
    public void mouseMoved(MouseEvent e){}
    public void mousePressed(MouseEvent e){
        x = e.getX();
        y = e.getY();
        updateScrollBars(x,y);
        repaint();
    }
    public void mouseDragged(MouseEvent e){
        x = e.getX();
        y = e.getY();
        updateScrollBars(x,y);
        repaint();
    }
    public void adjustmentValueChanged(AdjustmentEvent e){
        if(e.getSource()==xScrollBar)
        x=(int)((float)(xScrollBar.getValue()/100.0)*getSize().width);
        else if(e.getSource()==yScrollBar)
            y = (int)((float)(yScrollBar.getValue()/100.0)*getSize().height);
        repaint();
    }
}
public class Example6_9{
    public static void main(){
        MyWindow myWindow = new MyWindow("滚动条示意程序");

    }
}
```

上述例子中，如果只要求通过滑动滑块，改变内容的显示位置，可以简单地使用滚动面板 JScrollPane。如果是这样，关于滚动条的创建和控制都可以免去，直接由 JScrollPane 内部实现。参见以下修改后的 MyWindow 的定义：

```
    class MyWindow extends JFrame{
    public MyWindow(String s){
```

```
            super(s);
            Container con = this.getContentPane();
            con.setLayout(new BorderLayout());
            this.setLocaltion(100,100);
            MyListener listener = new MyListener();
            listener.setPreferredSize(new Dimension(700,700));
            JScrollPane scrolledCanvas = new JScrollPane(listener);
            this.add(scrolledCanvas,BorderLayout.CENTER);
            this.setVisible(true);
            this.pack();
        }
        public Dimension getPreferredSize(){
            return new Dimension(400,400);
        }
    }
```

鼠标指针形状也能由程序控制 ， setCursor()方法能设置鼠标指针形状。例如，代码

setCursor(Cursor.getPredefinedCursor(cursor.WAIT_CURSOR))。

Windows 平台上的常见鼠标指针形状定义见表 6-4。

HAND_CURSOR,WAIT_CURSOR,NE_RESIZE_CURSOR,N_RESIZE_CURSOR,MOVE_CURSOR,CROSHAIR_CURSOR,SE_RESIZE_CURSOR,E_RESIZE_CURSOR.

6. 7 键盘事件

键盘事件的事件源一般丐组件相关，当一个组件处于激活状态时，按下、释放或敲击键盘上的某个键时就会发生键盘事件。

键盘事件的接口是 KeyListener，注册键盘事件监视器的方法是 addKeyListener(监视器)。实现 KeyListener 接口有 3 个：

(1) keyPressed(KeyEvent e),键盘上某个键被按下。

(2) keyReleased(KeyEvent e),键盘上某个键被按下，又释放。

(3) keyTyped(KeyEvent e),keyPressed 和 keyReleased 两个方法的组合。

管理键盘事件的类是 KeyEvent，该类提供方法：

public int getKeyCode(),获得按动的键码，键码表在 KeyEvent 类中定义，参见附录 E。

[例 6.10]小应用程序有一个按钮和一个文本区，按钮作为发生键盘事件的事件源，并对它实施监视。程序运行时，先点击按钮，让按钮激活。以后输入英文字母时，在正文区显示输入的字母。字母显示时，字母之间用空格符分隔，且满 10 个字母时，换行显示。

```
import java.applet.*
import java.awt.*;
import java.awt.event.*;
public class Example6_10 extends Applet implements KeyListener{
    int count =0;
    Button button = new Button();
    TextArea text = new TextArea(5,20);
    public void init(){
        button.addKeyListener(this);
        add(button);add(text);
    }
    public void keyPressed(KeyEvent e){
        int t = e.getKeyCode();
        if(t>=KeyEvent.VK_A&&t<=KeyEvent.VK_Z){
            text.append((char)t+" ");
            count++;
```

```
            if(count%10==0)
            text.append("\n");
        }
    }
    public void keyTyped(KeyEvent e){}
    public void keyReleased(KeyEvent e){}
}
```

习题

1. 设计一个面板，该面板中有四个运动项目选择框和一个文件区。当某个选择项目被选中时，在文本区中显示该选择项目。

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class MyPanel extends JPanel implements ItemListener
{
 JCheckBox ch1,ch2,ch3,ch4;
 JTextArea text;
 JLabel la;
 MyPanel()
 {
  setLayout(new GridLayout(6,1));
    la=new JLabel("请选择运动项目");
  ch1=new JCheckBox("踢足球");
  ch2=new JCheckBox("打篮球");
  ch3=new JCheckBox("跑步");
  ch4=new JCheckBox("跳远");
  text=new JTextArea(5,5);
  ch1.addItemListener(this);
  ch2.addItemListener(this);
  ch3.addItemListener(this);
  ch4.addItemListener(this);
  add(la);add(ch1);add(ch2);
  add(ch3);add(ch4);add(text);
 }
 public void itemStateChanged(ItemEvent e)
 {
  if(e.getItemSelectable()==ch1)
  text.setText("你选中的运动项目是踢足球。");
  else if(e.getItemSelectable()==ch2)
  text.setText("你选中的运动项目是打篮球。");
  else if(e.getItemSelectable()==ch3)
  text.setText("你选中的运动项目是跑步。");
  else if(e.getItemSelectable()==ch4)
  text.setText("你选中的运动项目是跳远。");
 }
}
 class MyWindow extends JFrame
{
```

```
 MyPanel myPanel;
 MyWindow(String s)
 {
  super(s);
  setSize(150,150);
  Container con=this.getContentPane();
  myPanel=new MyPanel();
  con.add(myPanel);
  this.setVisible(true);//this.pack();
 }
}
public class PanelCheckBoxTest
{
 public static void main(String[]args)
 {
  MyWindow myWindw=new MyWindow("运动项目选择");
 }
}
```

2. 设计一个面板，该面板中有四个运动项目单选框和一个文本框。当某个选择项目被选中时，在文本框中显示该选择项目。

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
class MyPanel extends JPanel implements ItemListener
{
 JCheckBox ch1,ch2,ch3,ch4;
 JTextField text;
 JLabel la;
 ButtonGroup g;
 MyPanel()
 {
  setLayout(new GridLayout(6,1));
    la=new JLabel("请选择运动项目");
    g=new ButtonGroup();
  ch1=new JCheckBox("踢足球");
  ch2=new JCheckBox("打篮球");
  ch3=new JCheckBox("跑步");
  ch4=new JCheckBox("跳远");
  text=new JTextField(5);
  ch1.addItemListener(this);
  ch2.addItemListener(this);
  ch3.addItemListener(this);
  ch4.addItemListener(this);
  add(la);g.add(ch1);g.add(ch2);
  g.add(ch3);g.add(ch4);
  add(ch1);add(ch2);
  add(ch3);add(ch4);add(text);
 }
 public void itemStateChanged(ItemEvent e)
```

```
 {
  if(e.getItemSelectable()==ch1)
  text.setText("你选中的运动项目是踢足球。");
  else if(e.getItemSelectable()==ch2)
  text.setText("你选中的运动项目是打篮球。");
  else if(e.getItemSelectable()==ch3)
  text.setText("你选中的运动项目是跑步。");
  else if(e.getItemSelectable()==ch4)
  text.setText("你选中的运动项目是跳远。");
 }
}
 class MyWindow extends JFrame
{
 MyPanel myPanel;
 MyWindow(String s)
 {
  super(s);
  setSize(300,300);
  Container con=this.getContentPane();
  myPanel=new MyPanel();
  con.add(myPanel);
  this.setVisible(true);this.pack();
 }
}
public class PanelCheckBoxTest2 extends Applet
{
 MyWindow myWindow;
 public void init()
 {
  myWindow=new MyWindow("运动项目选择");
 }
}
```

3. 设计一个面板，该面板中有四个运动项目单选按钮和一个文本框。当某个选择项目被选中时，在文本框中显示该选择项目。

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
class MyPanel extends JPanel implements ItemListener
{
 JRadioButton ch1,ch2,ch3,ch4;
 JTextField text;
 JLabel la;
 ButtonGroup g;
 MyPanel()
 {
  setLayout(new GridLayout(6,1));
    la=new JLabel("请选择运动项目");
    g=new ButtonGroup();
```

```
  ch1=new JRadioButton("踢足球");
  ch2=new JRadioButton("打篮球");
  ch3=new JRadioButton("跑步");
  ch4=new JRadioButton("跳远");
  text=new JTextField(5);
  ch1.addItemListener(this);
  ch2.addItemListener(this);
  ch3.addItemListener(this);
  ch4.addItemListener(this);
  add(la);g.add(ch1);g.add(ch2);
  g.add(ch3);g.add(ch4);
  add(ch1);add(ch2);
  add(ch3);add(ch4);add(text);
 }
 public void itemStateChanged(ItemEvent e)
 {
  if(e.getItemSelectable()==ch1)
  text.setText("你选中的运动项目是踢足球。");
  else if(e.getItemSelectable()==ch2)
  text.setText("你选中的运动项目是打篮球。");
  else if(e.getItemSelectable()==ch3)
  text.setText("你选中的运动项目是跑步。");
  else if(e.getItemSelectable()==ch4)
  text.setText("你选中的运动项目是跳远。");
 }
}
 class MyWindow extends JFrame
{
 MyPanel myPanel;
 MyWindow(String s)
 {
  super(s);
  setSize(300,300);
  Container co 的介绍; n=this.getContentPane();
  myPanel=new MyPanel();
  con.add(myPanel);
  this.setVisible(true);this.pack();
 }
}
public class PanelRadioTest extends Applet
{
 MyWindow myWindow;
 public void init()
 {
  myWindow=new MyWindow("运动项目选择");
 }
}
```

4. 设计一个窗口，取默认布局 BorderLayout 布局。北面添加一个列表，列表有 4 门课程选项。中心添加 一个文本区，当选择列表中的某门课程后，文本区显示相应课程；当双击列表中的某个选项后，文本区就显示相应课程的开课时间。

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
import java.applet.*;
class MyWindow extends JFrame implements MouseListener,ListSelectionListener
{
  static JList listX;
  static String lesson[]={"语文","数学","英语","地理"};
  static JTextArea text;
  MyWindow(String s)
  {
  super(s);
  Container con=getContentPane();
  con.setSize(200,100);
  con.setLayout(new BorderLayout());
  listX=new JList(lesson);
  listX.setVisibleRowCount(2);
  JScrollPane jsp=new JScrollPane(listX);
  //list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
  listX.addMouseListener(this);
  listX.addListSelectionListener(this);
  text=new JTextArea();
  con.add(jsp,BorderLayout.NORTH);con.add(text,BorderLayout.CENTER);
  this.setVisible(true);this.pack();
   }
    public void mouseClicked(MouseEvent e)
  { int index;
   if(e.getSource()==listX)
   {
    if(e.getClickCount()==2)
      {
       text.setText(null);
       index=listX.locationToIndex(e.getPoint());
       String seleName=lesson[index];
       if(lesson[0].equals(seleName))
       text.setText("语文 9 月 4 日下午开课");
       else if(lesson[1].equals(seleName))
       text.setText("数学 9 月 5 日上午开课");
       else if(lesson[2].equals(seleName))
       text.setText("英语 9 月 6 日下午开课");
       else if(lesson[3].equals(seleName))
       text.setText("地理 9 月 7 日开课");
       }
     }
    }
    public void mouseExited(MouseEvent e){}
```

```
    public void mouseEntered(MouseEvent e){}
    public void mousePressed(MouseEvent e){}
    public void mouseReleased(MouseEvent e){}
    public void valueChanged(ListSelectionEvent e)
   {
      if(e.getSource()==listX)
       {
         text.setText(null);
           Object listvalue=((JList)e.getSource()).getSelectedvalue();
          String seleName=listvalue.toString();
           if(lesson[0].equals(seleName))
          text.setText("语文是对汉语的认识，了解和运用");
          else if(lesson[1].equals(seleName))
          text.setText("数学是一门应用科学，应用于日常生活");
          else if(lesson[2].equals(seleName))
          text.setText("英语是一门国际语言，是对外交流的必须");
          else if(lesson[3].equals(seleName))
          text.setText("地理是对各个地方的特征进行介绍");
        }
      }
}
public class ListAreaTest extends Applet
{
 MyWindow myWindow;
 public void init()
 {
  myWindow=new MyWindow("课程信息");
 }
}
```

5. 设计一个窗，取默布局 BorderLayout 布局。北面添加一个组合框，组合框有多门课程选项。中心添加一个文本框，当在组合框中选定一门课程后，文本框显示相应课程。

```
import java.applet.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
class MyWindow extends JFrame implements ItemListener
{
 JComboBox box;
 String lesson[]={"语文","数学","英语","地理","政治"};
 JTextField textF;
 MyWindow(String s)
 {
  super(s);
  Container con=this.getContentPane();
  con.setSize(200,100);
  con.setLayout(new BorderLayout());
  con.setBackground(Color.yellow);
  box=new JComboBox(lesson);
```

```
  box.addItemListener(this);
  box.setEditable(true);
  textF=new JTextField();
  con.add(box,BorderLayout.NORTH);con.add(textF,BorderLayout.CENTER);
  this.setVisible(true);this.pack();
 }
 public void itemStateChanged(ItemEvent e)
 {
  if(e.getSource()==box)
  textF.setText(box.getSelectedItem().toString());
 }
}
public class ComboBoxTest extends Applet
{
 MyWindow myWindow;
 public void init()
 {
  myWindow=new MyWindow("课程信息");
 }
}
```

6. 设计一个 JFrame 窗口，窗口中心添加一个文本区。另添加 4 个菜单，每个菜单都有菜单项，每个菜单项都对应有快捷键，选择某个菜单项时，窗口中心的文本区显示相应信息。

```
import java.applet.*;
import javax.swing.*;
import java.awt.*;
import java.awt.MenuShortcut;
import java.awt.event.*;
class MyWindow extends JFrame implements ActionListener
{
 JMenuBar bar;
 JMenu menu1,menu2,menu3,menu4;
  JTextArea text;

  private void addItem(JMenu menux,String itemName,int Key,ActionListener listener)
  {
   JMenuItem anItem=new JMenuItem(itemName);
   anItem.setAccelerator(KeyStroke.getKeyStroke(Key, InputEvent.CTRL_MASK));//菜单项快捷键设置方式
   anItem.setActionCommand(itemName);
   anItem.addActionListener(listener);
   menux.add(anItem);
  }
  MyWindow(String s)
  {
   super(s);
   Container con=getContentPane();
   con.setSize(150,150);
   con.setLayout(new BorderLayout());
   con.setBackground(Color.red);
```

```
   menu1=new JMenu("文件");
   menu2=new JMenu("编辑");
   menu3=new JMenu("格式");
   menu4=new JMenu("窗口");
   addItem(menu1,"打开",KeyEvent.VK_O,this);
   addItem(menu1,"关闭",KeyEvent.VK_C,this);
    addItem(menu2,"复制",KeyEvent.VK_C,this);
   addItem(menu2,"粘贴",KeyEvent.VK_P,this);
   addItem(menu3,"大写",KeyEvent.VK_A,this);
   addItem(menu3,"小写",KeyEvent.VK_L,this);
   addItem(menu4,"水平",KeyEvent.VK_H,this);
   addItem(menu4,"垂直",KeyEvent.VK_V,this);
   bar=new JMenuBar();
   text=new JTextArea();
   bar.add(menu1);bar.add(menu2);bar.add(menu3);bar.add(menu4);
   setJMenuBar(bar);con.add(text,BorderLayout.CENTER);
   this.setVisible(true);this.pack();
  }
  public void actionPerformed(ActionEvent e)
  {
     text.setText(e.getActionCommand());
  }
}
public class MenuTest extends Applet
{
 MyWindow myWindow;
 public void init()
 {
    myWindow=new MyWindow("菜单测试");
  }
}
```

7.设计一个窗口，窗口有两个依赖于它的对话框，一个对话框负责求圆的面积；另一个负责求三角形的面积。窗口中一个菜单负责打开这两个对话框。几何图形的数据通过对话框输入。

```
import java.applet.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
class MyWindow extends JFrame implements ActionListener
{
 JMenuBar bar;
 JMenu menu1;
 public void addItem(JMenu menux,String s,ActionListener listener)
 {
  JMenuItem anItem=new JMenuItem(s);
  anItem.setActionCommand(s);
```

```
  anItem.addActionListener(listener);
  menux.add(anItem);
 }
 MyWindow(String t)
 {
  super(t);
  Container con=getContentPane();
  con.setBackground(Color.red);
  con.setSize(300,300);
  menu1=new JMenu("对话框选择");
  addItem(menu1,"求圆面积",this);
  addItem(menu1,"求三角形面积",this);
  bar=new JMenuBar();
  bar.add(menu1);
  setJMenuBar(bar);
  this.setVisible(true);
  this.pack();
 }
 public void actionPerformed(ActionEvent e)
 {
  MyDialog dialogx;
  if(e.getActionCommand()=="求圆面积")
    {
     dialogx=new MyDialog(this,"圆的半径R=");
     dialogx.setVisible(true);
    }
    else if(e.getActionCommand()=="求三角形面积")
    {
     dialogx=new MyDialog(this,"三角形的底和高分别是");
     dialogx.setVisible(true);

    }
  }
}
class MyDialog extends JDialog implements ActionListener
{
 static JLabel title;static JTextField text;
 MyDialog(JFrame F,String s)
 {
  super(F,s,false);
  Container con=this.getContentPane();
  title=new JLabel("请输入"+s);
  text=new JTextField(5);
  text.setEditable(true);
  con.setLayout(new FlowLayout());
  con.setSize(250,250);
  setModal(false);
  text.addActionListener(this);
  con.add(title);con.add(text);
```

```
  con.setVisible(true);this.pack();
 }
 public void actionPerformed(ActionEvent e)
 {
  if(e.getSource()==text)
  {
   String t=text.getText();
   StringTokenizer tokens=new StringTokenizer(t);
   int n=tokens.countTokens();
   if(n==1)
   {
    title.setText("求得圆的面积是:");
    float R,S;
    String temp=tokens.nextToken();
    R=Float.parseFloat(temp);
    S=3.14159f*R*R;
    text.setText(String.valueOf(S));
   }
   if(n==2)
   {
    title.setText("求得三角形的面积是: ");
    float S,A=0,H=0;int i;
     for(i=0;i<=n-1;i++)
     {
      String temp=tokens.nextToken();
      if(i==0)
      A=Float.parseFloat(temp);
      if(i==1)
      H=Float.parseFloat(temp);
     }
     S=A*H/2.0f;
     text.setText(String.valueOf(S));
   }
  }
 }
}
public class DialogTest extends Applet
{
 MyWindow myWindow;
 public void init()
 {
  myWindow=new MyWindow("对话框测试");
 }
}
```

8. 设计一个 JFrame 窗口。程序有两个按钮，一个负责“体育之窗”的打开和关闭，一个负责“音乐之窗”的打开和关闭。当负责“体育之窗”的捉上的文字为“打开体育之窗”时，单击该按钮，“体育之窗”打开，同时按钮上文字改为“关闭体育之窗”。这时，再单击它，“体育之窗”关闭，按钮上的文字又改为“打开体育之窗”。“音乐之窗”按钮也有类似的处理效果。

```
import java.applet.*;
```

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class MyWindow extends JFrame
{
 JTextField text;
 MyWindow(String s)
 {
  super(s);
  Container con=this.getContentPane();
  con.setSize(200,200);
  text=new JTextField("欢迎光临"+s);
  con.add(text);this.pack();
 }
}
public class WindowOpenTest extends JFrame implements ActionListener
{
  MyWindow window1,window2;
   JButton b1,b2;boolean flg1,flg2;
   public void init()
   {
  b1=new JButton("打开体育之窗");flg1=true;
  b2=new JButton("打开音乐之窗");flg2=true;
  window1=new MyWindow("体育之窗");
  window2=new MyWindow("音乐之窗");
  b1.addActionListener(this);
  b2.addActionListener(this);
  add(b1);add(b2);
   }
   public void actionPerformed(ActionEvent e)
   {
    if(e.getSource()==b1)
    {
     if(flg1)
     {
      window1.setVisible(true);
      flg1=false;
      b1.setLabel("关闭体育之窗");
      }
     else
     {
        window1.setVisible(false);
      flg1=true;
      b1.setLabel("打开体育之窗");
     }
    }
    else if(e.getSource()==b2)
    {
     if(flg2)
```

```
    {
      window2.setVisible(true);
     flg2=false;
     b2.setLabel("关闭音乐之窗");
    }
    else
    {
      window2.setVisible(false);
      flg2=true;
      b2.setLabel("打开音乐之窗");
       }
   }
  }
 }
```

9. 设计一个窗口，窗口中有一个文本框。程序对键盘输入进行监视，当输入一个整数和回车符时，在文本框输出 1 至该整数的数列和；如果中间输入数字的其他字符，在文本框中显示输入错误，重新输入的字样。

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class KeyFieldTest extends Applet implements KeyListener,ActionListener
{
 static TextField text;
 static String s="";
 public void init()
 {
  text=new TextField(10);
  text.addKeyListener(this);
  text.addActionListener(this);
  add(text);

 }
 public void keyPressed(KeyEvent e)
 {
  int t=e.getKeyCode();
  if(t!=KeyEvent.VK_ENTER)
     s+=String.valueOf((char)t);
   if(t==KeyEvent.VK_DELETE)
   {text.setText(null);s="";}
  }
 public void keyTyped(KeyEvent e){}
  public void keyReleased(KeyEvent e){}
  public void actionPerformed(ActionEvent e)
  {
   byte c[]=s.getBytes();int i;
   for(i=0;i<=c.length-1;i++)
   {
       if(c[i]>=KeyEvent.VK_0&&c[i]<=KeyEvent.VK_9){}
       else
```

```
        { text.setText("输入错误，重新输入");break;}
    }
    if(i==c.length)
    { int y=0;
     int x=Integer.parseInt(s);
     for(int j=1;j<=x;j++)
     y+=j;
     text.setText(String.valueOf(y));
    }
  }
}
```

10．编写一个演示鼠标拖动和移动的程序。界面设有一个文本区，当鼠标拖动或移动时，在文本区中输出指明鼠标拖动或移动，及鼠标位置的字样。

```
import java.applet.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.*;
class MyWindow extends JFrame implements MouseMotionListener
{
 JTextArea text;
 int x,y;int mouseFlg=0;
 static String mouseStates[]={"鼠标拖动","鼠标移动"};
 MyWindow(String s)
 {
  super(s);
  Container con=this.getContentPane();
  con.setSize(200,200);
  con.setLayout(new GridLayout(2,1));
  this.setLocation(100,100);
  text=new JTextArea(10,20);
  text.setBackground(Color.blue);
  con.add(text);
  addMouseMotionListener(this);
  this.setVisible(true);this.pack();
 }
 public void paint(Graphics g)
 {
  text.setText(mouseStates[mouseFlg]+"了，位置是："+x+","+y+"\n");
 }
 public void mouseDragged(MouseEvent e)
 {
  x=e.getX();y=e.getY();mouseFlg=0;repaint();
 }
 public void mouseMoved(MouseEvent e)
 {
  x=e.getX();y=e.getY();mouseFlg=1;repaint();
```

```
 }

}
public class MouseTest1 extends Applet
{
 MyWindow myWindow;
 public void init()
 {
  myWindow=new MyWindow("鼠标测试");
 }
}
```

第 7 章　图形、图像与多媒体

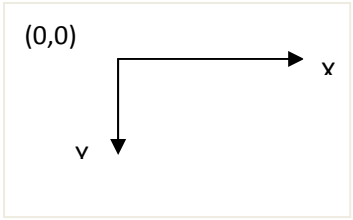
本章内容(次重点：10%)

本章重点：设置字型、颜色，几何图形绘制方法、图像显示技术基础。

本章难点：ＸＯＲ绘图模式、设置线型、绘制函数曲线，图像缓冲显示技术。

７．１　绘图基础

要在平面上显示文字和绘图，首先要确定一个平面坐标系。Java 语言约定，显示屏上一个长方形区域为程序绘图区域，坐标原点（０，０）位于整个区域的左上角。一个坐标点（x,y）对应屏幕窗口中的一个像素，是整数。如图７．1 所示。



窗口大小由超文本文件中的 width 和 height 指定。例如，以下超文本确定窗口宽为 250 像素，高为 400 像素：

```
<applet code = h.class width=250 height=400></applet>
```

７．１．１ Graphics 类的基本功能

在 java.awt 包中，类 Graphics 提供的功能有：**建立字体、设定显示颜色、显示图像和文本，绘制和填充各种几何图形**。可以从图形对象或使用 Component 的 getGraphics()方法得到 Graphics 对象。Graphics2D 类继承 Graphics 类，并且增加了许多状态属性，使应用程序可以绘制出更加丰富多彩的图形。

在某个组件中绘图，一般应该为这个组件所属的子类重写 paint()方法，在该重写的方法中进行绘图。但要在 JComponent 子类中进行绘图。例如，继承定义一个文本区子类，要在这样的文本区子对象中绘图，就应给这个文本区子类重写 **paintComponent**()。系统自动为程序提供图形对象，并以参数 g 传递给 paint()方法和 paintComponent()方法。

７．１．２ 字型和颜色

显示文字的方法主要有三种：

(1) drawString(String str,int x,int y),在指定的位置显示字符串。

(2) drawChars(char data[],int offset,int length, int x, int y),在指定的位置显示字符数组中的文字，从字符数组的 offset 位置开始，最多显示 length 个字符。

(3) drawBytes(byte data[],int offset,int length,int x,int y), 在指定的位置显示字符数组中的文字，从字符数组的 offset 位置开始，最多显示 length 个字符。

这里给出的显示位置(x,y)为文字的基线的开始坐标，不是文字显示的矩形区域的左上角坐标。

文字字型有三个要素：字体、风格和字号。常用的字体有 Times New Roman、Symbol、宋体、楷体等。常用的风格有三种：正常、粗体和斜体，分别用三个常量表示：Font.PLAIN(正常)、Font.BOLD(粗体)和 Font.ITALIC（斜体）。风格可以组合使用，例如 ，Font.BOLD+Font.ITALIC。字号是字的大小，单位是磅。

在 Java 语言中，用类 Font 对象字型。Font 类构造方法有：

Font(String fontName,int style,int size)，3 个参数分别表示字体、风格和字号。例如，代码：

```
Font fnA = new Font(“细明本”,Font.PLAIN,12)
```

设置的字型的是：细明体、正常风格， 12 磅字号。

Font 类的其他常用方法：

(1) getStyle()，返回字体风格。

(2) getSize()，返回字体大小。

(3) getName()，返回字体名称。

(4) isPlain()，测试字体是否是正常字体。

(5) isBold(),测试字体是否是粗体。

(6) isItalic()，测试字体是否是斜体。

[例７．１]小应用程序用 6 种字型字符串，显示内容说明本身的字型。

```
import java.applet.*;
import java.awt.*;
public class Example7_1 extends Applet{
    Font f1 = new Font("Helvetica",Font.PLAIN,18);
    Font f2 = new Font("Helvetica", Font.BOLD,10);
    Font f3 = new Font("Helvetica",Font.ITALIC,12);
    Font f4 = new Font("Courier",Font.PLAIN,12);
```

```
    Font f5 = new Font("TimesRoman", Font.BOLD+Font.ITALIC,14);
    Font f6 = new Font("Dialog",Font.ITALIC,14);
    public void paint(Graphics g){
        setSize(250,200);
        g.setFont(f1);drawString("18pt plain Helvetica",5,20);
        g.setFont(f2);drawString("10pt bold  Helvetica",5,43);
        g.setFont(f3);drawString("12pt italic Helvetica",5,58);
        g.setFont(f4);drawString("12pt plain courier",5,75);
        g.setFont(f5);drawString("14pt bold & italic times Roman",5,92);
        g.setFont(f6);drawString("14pt italic dialog",5,111);
    }
}
```

用类 Color 的对象设置颜色，有两种方法生成各种颜色。一是用类 Color 预定议的颜色：black，red, white, yellow ……；二是通过红绿蓝（RGB）的值合成颜色。

与颜色有关的常用方法：

(1) 用类 Color 的构造方法 Color（int R, int G,int B）创建一个颜色对象，参数 R,G,B 分别表示红色、绿色和蓝色，它们的取值是从 0 到 255。

(2) 用类 Graphics 的方法 setColor(Color c)，参数 c 的取值参见表 7 － 1 。

(3) 用类 Component 的方法 setBackground(Color c)设置背景颜色。因为小程序是组件类的子类，直接可用 setBackground()方法改变背景色。

(4) 用类 Graphics 的方法 getColor()获取颜色。

表 7 － 1 Color 类预定义颜色常量

black	blue	cyan
darkGray	gray	green
lightGray	magenta	orange
pink	red	white
yellow		

[例 7.2]小应用程序设置颜色并涂方块，其中绘制方块的方法参见 7．2 节。

```
import java.applet.*;
import java.awt.*;
public class Example7_2 extends Applet{
    public void paint(Graphics g){
        setSize(380,200);
        for(int i=0;i<=10;i++)
        {
            Color myredcolor = new Color(i*25+5,0,0);
            g.setColor(myredcolor);
            g.fillRect(i*32+5,2,28,28);
        }
        for(int i=0;i<=10;i++)
        {
            Color mygreencolor = new Color(0,i*25+5,0);
            g.setColor(mygreencolor);
            g.fillRect(i*32+5,32,28,28);
        }
        for(int i=0;i<=10;i++)
        {
            Color mybluecolor = new Color(0,0,i*25+5);
            g.setColor(mybluecolor);
```

```
            g.fillRect(i*32+5,62,28,28);
        }
    }
}
```

7．1．3绘图模式

绘图模式是指后绘制的图形与早先绘制的图形有重叠时，如何确定重叠部分的颜色。例如，后绘制的覆盖早先绘制的；或者后绘制与早先绘制的两种颜色按某种规则混合。主要有正常模式和异或模式两种：正常模式是后绘制的图形覆盖在早先绘制的图形之上，使早先贩图形的重叠部分不再可见。异或模式把绘图看作是按图形着色。异或模式绘图时，将当前正要绘图的颜色、原先绘制的颜色以及异或模式设定的颜色作特定的运算，得到实际绘图颜色。设置绘图模式的方法有：

（1）setPaintMode()，设置绘图模式为**覆盖模式**（正常模式）。正常模式是绘图的默认模式。

（2）setXORMode(Color c)，设置绘图模式为**异或模式**，参数 c 为异或模式设定的绘图颜色。

设背景色为B，用 setXORMode()设置的颜色为C，另用某个非背景色D绘图，XOR 模式有以下确定实际绘图颜色的法则：

B + B = C,用背景色绘图，出现C颜色。

D + D = B,当一个图形重画时就能清除原先画的图形。

B + D = B和D的混合色（当B，D不相同时）。

如果某区域已用D着色，再用E着色，则结果是：

D + E = D和E的混合色（当D，E不相同时）。

XOR 绘图模式实例参见7.2.1节的例7.4。

7．2　绘图

Graphics 类提供基本绘图方法，Graphics2D 类提供更强大的绘图能力。

7.2.1 Graphics 类的绘图方法

Graphics 类提供基本的几何图形绘制方法，主要有：**画线段、画矩形、画圆、画带颜色的图形、画椭圆、画圆弧、画多边形**等。

（1）画线

在窗口画一条线段，可以使用 Graphics 类的 drawLine()方法：

drawLine(int x1,int y1,int x2,int y2)

例如，以下代码在点（3，3）与点（50，50）之间画线段，在点（100,100）处画一个点。

```
g.drawLine(3,3,50,50);//画一条线段
g.drawLine(100,100,100,100);//画一个点。
```

（2）画矩形

有两种矩形：普通型和圆角型。

画普通矩形有两个方法：

drawRect(int x,int y,int width,int height),画线框围起来的矩形。其中参数 x 和 y 指定左上角的位置，参数 width 和 height 是矩形的宽和高。

fillRect(int x,int y,int width,int height)，是用预定的颜色填充一个矩形，得到一个着色的矩形块。

以下代码是画矩形的例子：

```
g.drawRect(80,100,40,25);//画线框
g.setColor(Color.yellow);g.fillRect(20,70,20,30);//画着色块
```

画圆角矩形也有两个方法：

drawRoundRect(int x,int y,int width, int height, int arcWidth, int arcHeight),是用线围起来的圆角矩形。其中参数 x 和 y 指定矩形左上角的位置；参数 width 和 heigth 是矩形的宽和高；arcWidth 和 arcHeight 分别是圆角弧的横向直径和圆角弧的纵向直径。

fillRoundRect(int x,int y,int width,int height,int arcWidth,int archeight),**是用预定的颜色填充的圆角矩形**。各参数的意义同前一个方法。

以下代码是画矩形的例子：

```
g.drawRoundRect(10,10,150,70,40,25);//画一个圆角矩形
g.setColor(Color.blue);
g.fillRoundRect(80,100,100,100,60,40);//涂一个圆角矩形块
```

```
g.drawRoundRect(10,150,40,40,40,40);//画圆
g.setColor(Color.red);
g.fillRoundRect(80,100,100,100,100,100);//画圆块
```

可以用画圆角矩形方法画圆形，当矩形的宽和高相等，圆角弧的横向直径和圆角弧的纵向直径也相等，并等于矩形的宽和高时，画的就是圆形。参见上述例子中的注释，前一个是画圆，后一个是涂圆块。

（3）画三维矩形

画三维矩形有两个方法：

draw3DRect(int x,int y,int width,int height, boolean raised),画一个突出显示的矩形。其中 x 和 y 指定矩形左上角的位置，参数 width 和 height 是矩形的宽和高，参数 raised 是突出与否。

fill3DRect(int x,int y,int width,int height,boolean raised)，用预定的颜色填充一个突出显示的矩形。

以下代码是画突出矩形的例子：

```
g.draw3DRect(80,100,40,25,true);//画一个线框
g.setColor(Color.yellow);
g.fill3DRect(20,70,20,30,true);//画一个着色块
```

（4）画椭圆形

椭圆形由椭圆的横轴和纵轴确定。画椭圆形有两个方法：

drawOval(int x,int y,int width,int height),是画用线围成的椭圆形.其中参数 x 和参数 y 指定椭圆形左上角的位置,参数 width 和 height 是横轴和纵轴.

fillOval(int x,int y,int width,int height),是用预定的颜色填充的椭圆形,是一个着色块.也可以用画椭圆形方法画圆形,当横轴和纵轴相等时,所画的椭圆形即为圆形.以下代码是画椭圆形的例子.

```
g.drawOval(10,10,60,120);//画椭圆
g.setColor(Color.cyan);g.fillOval(100,30,60,60);//涂圆块
g.setColor(Color.magenta);g.fillOval(15,140,100,50);//涂椭圆
```

（5）画圆弧

画圆弧有两个方法:

drawArc(int x,int y,int width,int height,int startAngle, int arcAngle),画椭圆一部分的圆弧线.椭圆的中心是它的外接矩形的中心,其中参数是外接矩形的左上角坐标(x,y),宽是 width,高是 heigh.参数 startAngle 的单位是 “度”.起始角度 0 度是指 3 点钟方位.参数 startAngle 和 arcAngle 表示从 startAngle 角度开始,逆时针方向画 arcAngle 度的弧.约定,正值度数是逆时针方向,负值度数是顺时针方向,例如-90 度是 6 点钟方位.

fillArc(int x,int y,int width, int height, int startAngle, int arcAngle),用 setColor()方法设定的颜色,画着色椭圆的一部分.

以下代码是画圆弧的例子:

```
g.drawArc(10,40,90,50,0,180);//画圆弧线
g.drawArc(100,40,90,50,180,180);//画圆弧线
g.setColor(Color.yellow);
g.fillArc(10,100,40,40,0,-270);//填充缺右上角的四分之三的椭圆
g.setColor(Color.green);
g.fillArc(60,110,110,60,-90,-270);//填充缺左下角的四分之三的椭圆
```

（6）画多边形

多边形是用多条线段首尾连接而成的封闭平面图.多边形线段端点的 x 坐标和 y 坐标分别存储在两个数组中,画多边形就是按给定的坐标点顺序用直线段将它们连起来.以下是画多边形常用的两个方法:

drawPolygon(int xpoints[],int yPoints[],int nPoints),画一个多边形

fillPolygon(int xPoints[],int yPoints[],int nPoints),用方法 setColor()设定的颜色着色多边形.

其中数组 xPoints[]存储 x 坐标点,yPoints[]存储 y 坐标点,nPoints 是坐标点个数.

注意,**上述方法并不自动闭合多边形,要画一个闭合的多边形,给出的坐标点的最后一点必须与第一点相同.**

以下代码实现填充一个三角形和画一个八边形.

```
int px1[]={50,90,10,50};//首末点相重,才能画多边形
```

```
int py1[]={10,50,50,10};
int px2[]={140,180,170,180,140,100,110,140};
int py2[]={5,25,35,45,65,35,25,5};
  g.setColor(Color.blue); g.fillPolygon(px1,py1,4);
  g.setColor(Color.red);  g.drawPolygon(px2,py2,9);
```

也可以用多边形对象画多边形. **用多边形类 Polygon** 创建一个多边形对象,然后用这个对象绘制多边形.Polygon 类的主要方法:

1) Polygon(),创建多边形对象,暂时没有坐标点.
2) Polygon(int xPoints[],int yPoints[],int nPoints),用指定的坐标点创建多边形对象.
3) addPoint(),将一个坐标点加入到 Polygon 对象中.
4) drawPolygon(Polygon p),绘制多边形
5) fillPolygon(Polygon p),和指定的颜色填充多边形.

例如,以下代码,画一个三角形和填充一个黄色的三角形. **注意,用多边形对象画封闭多边形不要求首末点重合.**

```
int x[]={140,180,170,180,140,100,110,100};
int y[]={5,25,35,45,65,45,35,25};
Polygon ponlygon1=new Polygon();
polygon1.addPoint(50,10);polygon1.addPoint(90,50);
polygon1.addPoint(10,50);g.drawPolygon(polygon1);
g.setColor(Color.yellow);
Polygon polygon2 = new Polygon(x,y,8);
g.fillPolygon(polygon2);
```

(7) 擦除矩形块

当需要在一个着色图形的中间有一个空缺的矩形的情况,可用背景色填充一矩形块实现,相当于在该矩形块上使用了 “橡皮擦”.实现的方法是:

clearRect(int x,int y, int width,int height),擦除一个由参数指定的矩形块的着色.

例如,以下代码实现在一个圆中擦除一个矩形块的着色.

```
  g.setColor(Color.blue);
  g.fillOval(50,50,100,100);g.clearRect(70,70,40,55);
```

(8) 限定作图显示区域

用一个矩形表示图形的显示区域,要求图形在指定的范围内有效,不重新计算新的坐标值,自动实现超出部分不显示.方法是 **clipRect**(int x,int y,int width,int height),限制图形在指定区域内的显示,超出部分不显示.多个限制区有覆盖时,得到限制区域的**交集区域**.例如,代码:

```
  g.clipRect(0,0,100,50);g.clipRect(50,25,100,50);
```

相当于 g.clipRect(50,25,50,25);

(9) 复制图形

利用 Graphics 类的方法 copyArea()可以实现图形的复制,其使用格式是:

copyArea(int x,int y,int width,int height, int dx, int dy),dx 和 dy 分别表示将图形粘贴到原位置偏移的像素点数,正值为往右或往下偏移是,负值为往左或往上偏移量.位移的参考点是要复制矩形的左上角坐标.例如,以下代码示意图形的复制,将一个矩形的一部分、另一个矩形的全部分别自制.

```
  g.drawRect(10,10,60,90);
  g.fillRect(90,10,60,90);
  g.copyArea(40,50,60,70,-20,80);
  g.copyArea(110,50,60,60,10,80);
```

[例 7.3]小应用程序重写 update()方法,只清除圆块,不清除文字.窗口显示一个不断移动的红色方块.

```
import java.applet.*;
import java.awt.*;
public class Example7_3 extends Applet{
   int i=1;
```

```
    public void init()
    {
        setBackground(Color.yellow);
    }
    public void paint(Graphics g)
    {
        i = i+8; if(i>160)i=1;
        g.setColor(Color.red);g.fillRect(i,10,20,20);
        g.drawString("我正学习 update()方法",100,100);
        try{
            Thread.sleep(100);
        }
        catch(InterruptedException e){}
        repaint();
    }
    public void update(Graphics g)
    {
        g.clearRect(i,10,200,100);//不清除"我正在学习 update()方法"
        paint(g);
    }
}
```

一般的绘图程序要继承 JFrame,定义一个 JFrame 窗口子类,还要继承 JPanel,定义一个 JPanel 子类.**在 JPanel 子类 中重定义方法 paintComponent(),在这个方法中调用绘图方法,绘制各种图形.**

[例 7.4]使用 XOR 绘图模式的应用程序,绘图效果参见图 7.2

```
import javax.swing.*;
import java.awt.*;
public class Example7_4 extends JFrame
{
  public static void main(String args[])
  {
    GraphicsDemo myGraphicsFrame = new GraphicsDemo();
  }
}
class ShapesPanel extends JPanel
{
  SharpesPanel()
  {
    setBackground(Color.white);
  }
  public void paintComponent(Graphics g)
  {
    super.paintComponent(g);
    setBackground(Color.yellow); //背景色为黄色
    g.setXORMode(Color.red); //设置 XOR 绘图模式,颜色为红色
    g.setColor(Color.green);
    g.fillRect(20, 20, 80, 40); //实际颜色是 green + yellow 的混合色=灰色
    g.setColor(Color.yellow);
    g.fillRect(60, 20, 80, 40); //后一半是 yellow+yellow=read,前一半是 yellow+灰色
    g.setColor(Color.green);
```

```
            g.fillRect(20, 70, 80, 40); //实际颜色是 green+yellow 的混合色=灰色.
            g.fillRect(60, 70, 80, 40);
              //前一半是(green+yellow)+gray =背景色,后一半是 green+yellow = gray
            g.setColor(Color.green);
            g.drawLine(80, 100, 180, 200); //该直线是 green+yellow = gray
            g.drawLine(100, 100, 200, 200); //同上
        /*再绘制部分重叠的直线.原直线中间段是灰色+灰色=背景色,延长部分是 green+yellow=gray.*/
            g.drawLine(140, 140, 220, 220);
            g.setColor(Color.yellow); //分析下列直线颜色变化,与早先的力有重叠
            g.drawLine(20, 30, 160, 30);
            g.drawLine(20, 75, 160, 75);
          }
        }
        class GraphicsDemod extends JFrame
        {
          public GraphicsDemo()
          {
            this.getContentPane().add(new ShapesPanel());
            setTile("基本绘图方法演示");
            setSize(300, 300);
            setVisible(true);
          }
        }
```

7.2.2 Graphics2D 类的绘图方法

Java 语言在 Graphics 类提供绘制各种基本的几何图形的基础上,扩展 Graphics 类提供一个 Graphics2D 类,它拥用更强大的二维图形处理能力,提供、坐标转换、颜色管理以及文字布局等更精确的控制。

1. 绘图属性

Graphics2D 定义了几种方法，用于添加或改变图形的状态属性。可以通过设定和修改状态属性，指定画笔宽度和画笔的连接方式；设定平移、旋转、缩放或修剪变换图形；以及设定填充图形的颜色和图案等。图形状态属性用特定的对象存储。

(1) stroke 属性

stroke 属性控制线条的宽度、笔形样式、线段连接方式或短划线图案。该属性的设置需要先创建 BasicStroke 对象，再调用 setStroke()方法来设置。创建 BasicStroke 对象的方法有：

BasicStroke(float w)，指定线条宽 w。

BasicStroke(float w,int cap, int join),cap 是端点样：CAP_BUTT(无修饰)，CAP_ROUND(半圆形末端)，CAP_SQUARE(方形末端，默认值)。Join 定义两线段交汇处的连接方式：JOIN_BEVEL(无修饰),JOIN_MTTER(尖形末端，默认值),JOIN_ROUND(圆形末端)。

(2) paint 属性

paint 属性控制填充效果。先调用以下方法确定填充效果，理用 setPaint()方法设置。

GradientPaint(float x1,float y1,Color c1,float x2,flaot y2,Color c2),从(x1,y1)到(x2,y2)颜色从 c1 渐变到 c2。其中：参数 c1,c2 决定这个渐变色是从颜色 c1 渐变到颜色 c2。参数 x1,y1,x2,y2 决定了渐变的强弱，即要求从点(x1,y1)出发到达点(x2,y2)，颜色从 c1 变成 c2。

GradientPaint(float x1,float y1,Color c1,float x2,float y2,Color c2,Boolean cyclic),如果希望渐变到终点又是起点的颜色，应将 cyclic 设置为 true。

(3) transform 属性

transform 属性用来实现常用的图形平移、缩放和斜切等变换操作。首先创建 AffineTransform 对象，然后调用 setTransform()方法设置 transform 属性。最后，用具有指定属性的 Graphics2D 对象绘制图形。创建 AffineTransform 对象的方法有：

getRotateinstrance(double theta),旋转 theta 弧度。

getRotateInstance(double theta,dioble x,double y),绕旋转中心(x,y)旋转。

getScaleInstance(double sx,double sy)，x 和 y 方向分别按 sx,sy 比例变换。

getTranslateInstance(double tx,double ty),平移变换。

getShearInstance(double shx,double shy)，斜切变换，shx 和 shy 指定斜拉度。

也可以先创建一个没有 transform 属性的 AffineTransform 对象，然后用以下方法指定图形平移、旋转、缩放变换属性。

transelate(double dx,double dy)，将图形在 x 轴方向平移 dx 像素。

scale(double sx,double sy),图形在 x 轴方向缩放 sx 倍，纵向缩放 sy 倍。

rotate(double arc,double x, double y),图形以点(x,y)为轴点，旋转 arc 弧度。

例如，创建 AffineTransform 对象：

```
AffineTransform trans = new AffineTransform();
```

为 AffineTransform 对象指定绕点旋转变换属性：

```
Trans.rotate(50.0*3.1415927/180.0,90,80);
```

接着为 Graphics2D 的对象 g2d 设置具有上述旋转变换功能的“画笔”：

```
Graphics2D g2d = (Graphics2D)g;g2d.setTranstorm(trans);
```

最后，以图形对象为参数调用具有变换功能的 Graphics2D 对象的 draw()方法。例如，设已有一个二次曲线对象 curve，以下代码实现用上述旋转功能的 g2d 对象绘制这条二次曲线：

```
g2d.draw(curve);
```

(4) clip 属性

clip 属性用于实现剪裁效果。设置剪裁属性可调用 setClip()方法确定剪裁区的 Shape。连续多个 setClip()得到它们交集的剪裁区。

(5) composit 属性

composit 属性设置图形重叠区域的效果。先用方法 AlphaComposite.getInstance(int rule, float alpha)得到 AlphaComposite 对象，再通过 setComposite()方法设置混合效果。Alpha 值的范围为 0.0f(完全透明)-0.1f(完全不透明)。

2. Graphics2D 类的绘图方法

Graphics2D 类仍然保留 Graphics 类的绘图方法，同时增加了许多新方法。新方法将几何图形(线段、圆等)作为一个对象来绘制。在 java.awt.geom 包中声明的一系列类，分别用于创建各种身体图形对象。主要有：

Line2D 线段类，RoundRectangle2D 圆角矩形类，Ellipse2D 椭圆类，Arc2D 圆弧类，QuadCurve2D 二次曲线类，CubicCurve2D 三次曲线类。

要用 Graphics2D 类的新方法画一个图形。**先在重画方法 paintComponent()或 paint()中，把参数对象 g 强制转换成 Graphics2D 对象；然后，用上述图形类提供的静态方法 Double()创建该图形的对象；最后，以图形对象为参数调用 Graphics2D 对象的 draw()方法绘制这个图形**。例如以下代码用 Graphics2D 的新方法绘制线段和圆角矩形：

```
Graphics2D g2d = (Graphics2D)g;//将对象 g 类型从 Graphics 转换成 Graphics2D
Line2D line = new Line2D.Double(30.0,30.0,340.0,30.0);
g2d.draw(line);
RoundRectangle2D rRect = new RoundRectangle2D.Double(13.0,30.0,100.0,70.0,40.0,20.0);
g2d.draw(rRect);
```

也可以先用 java.awt.geom 包提供的 Shape 对象，并用单精度 Float 坐标或双精度 Double 坐标创建 Shape 对象，然后再用 draw()方法绘制。例如，以下代码先创建圆弧对象，然后绘制圆弧：

```
Shape arc = new Arc2D.Float(30,30,150,150,40,100,Arc2D.OPEN);
g2d.draw(arc)/绘制前面创建的图形对象 arc
```

3. Graphics2D 的几何图形类

(1) 线段

```
Line2D line = new Line2D.Double(2,3,200,300);//声明并创建线段对象
//起点是(2，3)，终点是(200，300)
```

(2) 矩形

```
Rectangle2D rect = new Rectangle2D.Double(20,30,80,40);//声明并创建矩形对象，矩形的左上角是(20，30)，宽是 300，高是 40
```

(3) 圆角矩形

RoundRectangle2D rectRound = new RoundRectangle2D.Double(20,30,130,100,18,15);

//左上角是(20，30)，宽是130，高是100，圆角的长轴是18，短轴是15。

(4) 椭圆

Ellipse2D ellipse = new Ellipse2D.Double(20,30,100,50);

//左上角 (20，30)，宽是100，高是50

(5) 圆弧

Arc2D arc1 = new Arc2D.Double(8,30,85,60,5,90,Arc2D.OPEN);

//外接矩形的左上角(10，30)，宽85，高60，起始角是5度，终止角是90度

Arc2D arc2 = new Arc2D.Double(20,65,90,70,0,180,Arc2D.CHORD);

Arc2D arc3 = new Arc2D.Double(40,110,50,90,0,270,Arc2D.PIE);

参数Arc2D.OPEN、Arc2D.CHORD、Arc2D.PIE分别表示圆弧是开弧、弓弧和饼弧。

(6) 二次曲线

二次曲线用二阶多项式表示：

$y(x)=ax^2+bx+c$

一条二次曲线需要三个点确定：始点、控制点和终点。

QuadCurve2D curve1 = new QuadCurver2D.Double(20,10,90,65,55,115);

QuadCurve2D curve2 = new QuadCurver2D.Double(20,10,15,63,55,115);

QuadCurve2D curve3 = new QuadCurver2D.Double(20,10,54,64,55,115);

方法Double()中的6个参数分别是二次曲线的始点、控制点和终点。以上3条二次曲线的开始点和终点分别相同。

(7) 三次曲线

三次曲线用三阶多项式表示：

$y(x)=ax^3+bx^2+cx+d$

一条三次曲线需要四个点确定：始点、两个控制点和终点。

CubicCurve2D curve1 = new CubicCurve2D.Double(12,30,50,75,15,15,115,93);

CubicCurve2D curve2 = new CubicCurve2D.Double(12,30,15,70,20,25,35,94);

CubicCurve2D curve3 = new CubicCurve2D.Double(12,30,50,75,20,95,95,95);

方法Double()中的8个参数分别是三次曲线的始点、两个控制点和终点。

一般的方程曲线的绘制过程用一个循环控制。通过循环产生自变量的值，按照方程计算出函数值，再作必要的坐标转换：原点定位的平移变换，图像缩小或放大的缩放变换，得到曲线的图像点，并绘制这个点。以绘制以下曲线方程为例：

Y=sin(x)+cos(x),x

绘制的部分代码可以写成如下：

```
double x0,y0,x1,y1,x2,y2,scale;
x0=100;y0=80;
scale =20.0;
for(x1=-3.1415926d;x1<=2*3.1415926d;x1+=0.01d)
{
    y1=Math.sin(x1)+Math.cos(x1);
    x2=x0+x1*scale;y2=y0+y1*scale;//(x2,y2)是图像点
    g.fillOval((int)x2,(int)y2,1,1);//画一个圆点作为图像点
}
```

7．3 图像处理基础

图像是由一组像素构成，用二进制形式保存的图片。java语言支持GIF、JPEG和BMP这3种主要图像文件格式。java语言的图像处理功能被封装在Image类中。

7．3．1 图像载入和输出

在java程序中，图像也是对象，所以载入图像时，先要声明Image对象，然后，利用getImage()方法把Image对

象与图像文件联系起来。载入图像文件的方法有两个：

(1) Image getImage(URL url),url 指明图像所在位置和文件名。

(2) Image getImage(URL url,String name),url 指明图像所在位置，name 是文件名。

例如，以下代码声明 Image 对象，并用 getImage()对象与图像文件联系起来：

Image img = getImage(getCodeBase(),”family.jpg”);

URL(uniform Resource Location 统一资源定位符)对象用于标识资源的名字和地址，在 WWW 客户机访问 Internet 网上资源时使用。确定图像位置的方法有两种：绝对位置与相对位置。取相对位置的方法有：

(1) URL getCodeBase(),取小应用程序文件所在的位置。

(2) URL getDocumentBase(),取 HTML 文件所在的位置。

例如，代码：

```
URL picURLA = new URL(getDocumentBase(),”imageSample1.gif”),
                        picURLB = new URL(getDocumentBase(),”pictures/imageSample.gif”);
                        Image imageA = getImage(picURLA),imageB = getImage(picURLB);
```

获取图像信息(属性)的方法有：

(1) getWidth(ImageObserver observer),取宽度。

(2) getHeight(ImageObserver observer),取高度。

输出图像的代码写在 paint()方法中，有 4 种显示图像的方法：

(1) boolean drawImage(Image img,int x,int y,ImageObserver observer)

(2) boolean drawImage(Image img,int x,int y,Color bgcolor,ImageObserver observer)

(3) boolean drawImage(Image img,int x,int y,int width,int height,ImageObsever observer)

(4) boolean drawImage(Image img,int x,int y,int width,int height,Color bgcolor,ImageObsever observer)

参数 img 是 Image 对象，x,y 是绘制图像矩形的左上角位置，**observer 是加载图像时的图像观察器**，bgcolor 是显示图像用的底色，width 和 height 是显示图像的矩形区域，当这个区域与图像的大小不同时，显示图像就会有缩放处理。

Applet 类也实现 ImageObserver 接口，常用 this 作为实参。参见以下代码及注释：

(1) g.drawImage(image1,0,0,this);//原图显示

(2) g.drawImage(image2,10,10,Color.red,this);//图形加底色显示

注意：如原图的大小与给定的范围不同，系统会自动缩放

(3) g.drawImage(labImag,0,0,this);// 原图显示

(4) g.grawImage(labImag,0,120,100,100,this);//缩放显示

(5) g.grawImage(labImag,0,240,500,100,this);//缩放显示

[例 7.5]小应用程序用 init()或 start()方法下载(获取)图像，用 paint()方法显示得到的图像。

```
import java.applet.*;import java.awt.*;
public class Example7_5 extends Applet{
    Image myImag;
    public void start(){
        myImag = getImage(getCodeBase(),”myPic.jpg”);
    }
    public void paint(Graphics g){
        g.drawImage(myImg,2,2,this);
    }
}
```

由于在 Frame、JFrame 和 JPanel 等类中没有提供 getImage()方法，它们载入图像需要使用 java.awt.Toolkit 中的 Toolkit 抽象类，该类有载入图像文件的方法：

(1) Image.getImage(String name),按指定的文件名载入图像文件。

(2) Image.getImage(URL url),统一资源定位符载入图像文件。

这样，各种组件可以用 getToolkit()方法得到 Toolkit 对象，然后在组件的 paint()方法中通过 Toolkit 对象显示图像。以下代码示意这样的用法：

```
Toolkit tool = getToolkit();
URL url = new URL(http://www.uvw.com/image.gif);
```

```
Image img = tool.getImage(url);
```

组件也可以使用 Toolkit 提供的静态方法 getDefaultToolkit()获得一个缺省的 Toolkit 对象,并用它加载图像。此时，载入图像的代码常写成这样：

```
Image img = Toolkit.getDefaultToolkit().getImage(url);
```

7．3．2 图像缓冲技术

当图像信息量较大，采用以上直接显示的方法，可能前面一部分显示后，显示后面一部分时，由于后面一部分还未从文件读出，使显示呈斑驳现象。为了提高显示效果，许多应用程序都采用图像缓冲技术，即先把图像完整装入内存，在缓冲区中绘制图像或图形，然后将缓冲区中绘制好的图像或图形一次性输出在屏幕上。缓冲技术不仅可以解决闪烁问题，并且由于在计算机内存中创建图像，程序可以对图像进行像素级处理，完成复杂的图像变换后再显示。

[例 7.6]小应用程序程序演示图像缓冲显示技术。程序运行时，当鼠标在图像区域内按下时，图像会出现边框，托动鼠标时，图像也随之移动。抬起鼠标后，边框消失。程序将两种状态的图像先放入两个缓冲区，当鼠标拖动时，不断地在新的位置重绘鼠标按下样式的图像鼠标抬起时，重绘鼠标抬起样式的图像。

```
import java.applet.*;
import java.awt.*;
imprt java.awt.image. * ;
import javax.swing.*;
import java.event.*;
public class Example7_6 extends Applet
{
  Image myPicture;
  /*init()方法中，先定义一个 Image 对象，并赋予 createImage()方法的返回值，接着创建 Graphics 对象并赋予其图形环境。最后，让 Graphics 对象调用 drawImage()方法显示图像。
  由于这里的 Graphics 对象 offScreenGc 是非屏幕对象是，小程序窗口不会有图像显示*/
  public void init()
  {
    myPicture = getImage(getCodeBase(), "myPic.JPG");
    Image offScreenImage = createImage(size().width, size().height);
    Graphics offScreenGc = offScreenImage.getGraphics();
    new BufferedDemo(myPicture);
  }
  /*drawImage()方法的第四个参数是实现 ImageObserver 接口,在 init()方法中,调用 drawImage()方法的参数是 this，所以小程序要定义 imageUpdate()方法*/
  public boolean imageUpdate(Image img, int infoFlg, int x, int y, int w, int h)
  {
    if (infoFlg = ALLBITS)// 表示图像已全部装入内存
    {
      repaint();
      return false;// 防止线程再次调用 imageUpdate()方法
    }
    else
      return true;
  }
}
```

/*程序的执行过程是,当小程序调用 drawImage()方法时,drawImage()方法将创建一个调用 imageUpdate()方法的线程，在 imageUpdate()方法中，测定图像是否已在部分调入内存。创建的线程不断调用 imageUpdate()方法，直到该方法返回 false 为止。参数 infoFlg 使小程序能知道图像装入内存的情况。当 infoFlg 等于 ALLBITS 时，表示图像已全部装入内存。当该方法发现图像已全部装入内存后，置 imageLoaded 为真，并调用 repaint()方法重画小程序窗口。方法返回 false 防止线程再次调用 imageUpdate()方法。*/

```
class BufferedDemo extends JFrame
{
  public BufferedDemo(Image img)
  {
    this.getContentPane().add(new PicPanel(img));
    setTile("双缓技术演示");
    setSize(300, 300);
    setVisible(true);
  }
}

class PicPane extends JPanel implements MouseListener, MouseMotionListener
{
  int x = 0, y = 0, dx = 0, cy = 0;
  BufferedImage bimg1, bimg2;
  boolean upstate = true;
  public picPanel(Image img)
  {
    this.setBackground(Color.white);
    this.addMouseListener(this);
    this.addMouseMotionListener(this);
    bimg1 = new BufferedImage(img.getWidth(this), img.getHeight(this),
      BufferedImage.TYPE_INT_ARGB);
    bimg2 = new BufferedImage(img.getWidth(this), img.getHeight(this),
      BufferedImage.TYPE_INT_ARGB);
    Graphics2D g2D1 = bimg1.createGraphics();
    Graphics2D g2D2 = bimg2.createGraphics();
    g2D1.drawImage(img, 0, 0, this);
    g2D2.drawImage(img, 0, 0, this);
    g2D2.drawRect(1, 1, img.getWidth(this) - 3, img.getHeight(this) - 3);
  }
  public void paintComponent(Graphics g)
  {
    super.painComponent(g);
    Graphics2D g2D = (Graphics2D)g;
    if (upState)
      g2D.drawImage(bimg1, x, y, this);
    else
      g2D.drawImage(bimg2.x, y, this);
  }
  public void mousePress(MouseEvent e)
  {
    if (e.getX() >= x && e.getX() < x + bimg1.getWidth(this) && e.getY() >= y
      && e.getY() < y + bimg1.getHeight(this))
    {
      upstate = false;
      setCursor(Cursor.getPredefinedCursor(Coursor.HAND_CURSOR));
      dx = e.getX() - x;
      dy = e.getY() - y;
```

```
      repain();
    }
  }
  public void mouseExited(MouseEvent e){}
  public void mouseClicked(MouseEvent e){}
  public void mouseEntered(MouseEvent e){}
  public void MouseReleased(MouseEvent e)
  {
    this.setCursor(Cursor.getpredefinedCursor(Cursor.DEFAULT_CURSOR));
    upState = true;
    repaint();
  }
  public void mouseMoved(MouseEvent e){}
  public void mouseDragged(MouseEvent e)
  {
    if (!upState)
    {
      x = e.getX() - dx;
      y = e.getY() - dy;
      repaint();
    }
  }
}
```

程序要创建缓冲区图像，需要引入 java.awt.image 包中的 BufferedImage 类。要创建一个缓冲区图，可以调用 createImage()方法，该方法返回一个 Image 对象，然后再将它转换成一个 BufferedImage 对象。例如，代码：

```
BufferedImage bimage = (BufferedImage)this.createImage(this.getWidth(),this.getHeight());
```

也可利用以下构造方法来建立。

```
BufferedImage(int width,int heigh, int imageType);
```

其中参数 imageType 是图像类型。

使用缓冲区显示图像，需先在缓冲区中准备好图像，再将缓冲区中的图像显示在界面上。显示图像需要图形对象 Graphics,可以通过以下方法建立：

```
Graphics2D g2d = bimge.createGraphics();
```

7．4 多媒体基础

本节介绍 Java 程序播放幻灯片和动画，播放声音和视频的方法。

7．4．1 播放幻灯片和动画

用实例说明播放幻灯片和动画的方法

[例 7.7]小应用程序先将幻灯片读入数组在存储，单击鼠标变换幻灯片，逐张显示。

```
import java.applet.*import java.awt.*;
import java.awt.event.*;
public class Example7_7 extends Applet implements MouseListener
{
  final int number = 50; //假定幻灯片有 50 张
  int count = 0;
  Image[] card = new Image[number];
  public void init()
  {
    addMouseListener(this);
    for (int i = 0; i < number; i++)
    {
```

```
      card[i] = getImage(getCodeBase(), "DSC0033" + i + ".jpg");
    }
  }
  public void paint(Graphics g)
  {
    if ((card[count]) != null)
      g.drawImage(card[count], 10, 10, card[count].getWidth(this),
        card[count].getHeitht(this), this);
  }
  public void mousePressed(MouseEvent e)
  {
    count = (count + 1) % number; //循环逐张显示
    repaint();
  }
  public void mouseRelease(MouseEvent e){}
  public void mouseEntered(MouseEvent e){}
  public void mouseExited(Mouse Event e){}
  public void mouseClicked(MouseEvent e){}
}
```

[例 7.8]小应用程序说明播放动画的方法，要求播放的图片和小程序放在相同的目录中，程序通过快速显示一组图片造成显示动画的效果。小应用程序利用线程控制动画图片的逐显示，有关线程编程技术参见第 8 章。

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class Example7_8 extends Applet implements Runnable
{
  final int number = 50;
  int count = 0;
  Thread mythread;
  Image[] pic = new Image[number];
  public void init()
  {
    setSize(300, 200);
    for (int i = 0; i <= number; i++)
    {
      //载入动画图片
      pic[i - 1] = getImage(getCodeBase(), "DSC0033" + i + ".jpg");
    }
  }
  public void start()
  {
    mythread = new Thread(this); //创建一个线程
    mythread.start(); //启动线程执行
  }
  public void stop()
  {
    mythread = null;
  }
  public void run()
```

```
  {
    //线程的执行代码
    while (true)
    {
      repaint();
      count = (count + 1) % number; //改变显示的图片号
      try
      {
        mhythread.sleep(200);
      }
      catch (InterruptedExeception e){}
    }
  }
  public void paint(Graphics g)
  {
    if ((pic[count] != null)g.drawImage(pic[count], 10, 10, pic[count].getwidth
      (this), pic[count].getHeight(this), this);
  }
}
```

7.4.2 播放声音

Java 语言老根据地的音频格式有多种：au、aiff、wav、midi、rfm 等。小程序要播放音频文件，可使用类 AudioClip，该类在 java.applet.AudioClip 类库中定义。小程序先创建 AudioClip 对象，并用 getAudioClip()方法为其初始化。代码形式如下：

```
AudioClip audioClip = getAudioClip(getCodeBase(),” myAudioClipFile.au”);
```

如果要从网上获得音频文件，可用方法 getAudioClip(URL url, String name),根据 url 地址及音频文件 name 获得可播放的音频对象。

控制声音的播放有 3 个方法：play()播放声音，loop()循环播放和 stop()停止播放。

[例 7.9]能播放声音的小应用程序。

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class Example7_9 extends Applet implements ActionListener
{
  AudioClip clip; //声明一个音频对象
  Button buttonPlay, buttonLoop, buttonStop;
  public void init()
  {
    clip = getAudioClip(getCodeBase(), "2.wav");
    //根据程序所在地址处声音文件 2.wav 创建音频对象，
    //Applet 类的 getCodeBase()方法可以获得小程序所在的 html 页面的 URL 地址。
    buttonPlay = new Button("开始播放");
    buttonLoop = new Button("循环播放");
    buttonStop = new Button("停止播放");
    buttonPlay.addActionListener(this);
    buttonStop.addActionListener(this);
    buttonLoop.addActionListener(this);
    add(buttonPlay);
    add(buttonLoop);
    add(buttonStop);
```

```
  }
  public void stop()
  {
    clip.stop(); //当离开此页面时停止播放
  }
  public void actionPerformed(ActionEvent e)
  {
    if (e.getSource() == buttonPlay)
    {
      clip.play();
    }
    else if (e.getSource() == buttonLoob)
    {
      clip.loop();
    }
    else if (e.getSource() == buttonStop)
    {
      clip.stop();
    }
  }
}
```

如果声音文件较大或网络速度慢会影响小程序的初始化工作。这可用多线程技术解决。在一个级别较低的线程中完成音频对象的创建，即由后台载入声音文件，前台播放。

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class Hanoi extends applet implements Runnable, ActionListener
{
    AudioClip clip; //声明一个音频对象
    textField text;
    Thread thread;
    Button buttonPlay, buttonLoop, buttonStop;
    public void init()
    {
        thread = new Thread(this); //创建新线程
        thread .setPriority(Thread.MIN_PRIORITY);
        buttonPlay = new Button("开始播放");
         buttonLoop = new Button( "循环播放");
        buttonStop = new Button("停止播放");
        text = new textField(12);
        buttonPlay.addActionListener(this);
        buttonStop.addActionListener(this);
        buttonLoop.addActionListener(this);
        add(buttonPlay);
        add(buttonLoop);
        add(buttonStop);
        add(text);
    }
    public void start()
```

```
    {
        thread.start();
    }
    public void stop()
    {
        clip.stop();
    }
    public void actionPerformed(ActionEvent e)
    {
        if (e.getSource() == buttonPlay()
        {
            clip.play();
        }
        else if (e.getSource() == buttonLoop()
        {
            clip.loop();
        }
        else if (e.getSource() == buttonStop()
        {
            clip.stop();
        }
    }
    public void run()
    {
        //在线程 thread 中创建音频对象
        clip = getAudioclip(getCodeBase(), "2.wav"); text.setText("请稍等"); if
            (clip ! = null)
        {
            buttonPlay.setBackground(Color.red); buttonLoop.setBackground
                (Color.green); text.setText("您可以播放了");
        } //获得音频对象后通知可以播放
    }
}
```

习题

1. 编写一个应用程序，绘制一个五角星。

```
    import java.applet.*;
    import java.awt.*;
    import javax.swing.*;
    public class StarTest2 extends Applet
    {
     public void init()
     {
      GraphicsDemo myGraphicsDemo=new GraphicsDemo();
     }
    }
    class MyPanel extends JPanel
    { int x0=100,y0=100;
     int x[]=new int[6];
     int y[]=new int[6];
```

```
 int i=0;
 double r=60,t;

 MyPanel()
 {
  setBackground(Color.blue);
   for(t=0.0d;t<=4*3.1415926d;t+=3.1415926*4.0/5.0d)
  {
  x[i]=(int)(r*Math.cos(t)+x0);
  y[i]=(int)(r*Math.sin(t)+y0);
  i++;
  if(i==6)break;
  }
 }
 public void paintComponent(Graphics g)
 {
  super.paintComponent(g);
  setBackground(Color.green);
  g.setColor(Color.red);
  g.drawPolygon(x,y,6);
 }
}
class GraphicsDemo extends JFrame
{
 public GraphicsDemo()
 {
  this.getContentPane().add(new MyPanel());
  setTitle("五角星绘制");
  setSize(300,300);
  setVisible(true);
 }
}
```

2. 用 Graphics2D 绘制一个抛物线。设抛物线方程的系数从图形界面输入。

```
import java.awt.*;
import java.awt.geom.*;
import java.applet.*;
import java.awt.event.*;
import javax.swing.*;
public class TTest extends Applet
{
 public void init()
 {
  GraphicsDemo myGraphics=new GraphicsDemo();
 }
}
class MyPanel extends JPanel implements ActionListener
{
 double x,y,p=0.0,scalex=5.0,scaley=10.0,dx=20.0;
 JTextField text1;
```

```
 JLabel la1;

 MyPanel()
 {
  setSize(200,200);
  setBackground(Color.blue);
  text1=new JTextField(5);

  text1.addActionListener(this);

  la1=new JLabel("请输入抛物线方程的系数 p:");

  add(la1);add(text1);
 }
 public void actionPerformed(ActionEvent e)
 {
  if(e.getSource()==text1)
  p=Double.parseDouble(text1.getText());
 }
 public void paintComponent(Graphics g)
 {
  super.paintComponent(g);
  Graphics2D g2D=(Graphics2D)g;
  setBackground(Color.green);
  QuadCurve2D curve=new QuadCurve2D.Double((dx-1.0*Math.sqrt(2*10*p)),10.0*scaley,(0+dx)*scalex,0,(dx+Math
.sqrt(2*10*p))*scalex,10*scaley);
  g2D.setColor(Color.red);
  g2D.draw(curve);repaint();
 }
}
class GraphicsDemo extends JFrame
{
 public GraphicsDemo()
 {
  this.getContentPane().add(new MyPanel());
  setTitle("抛物线演示");
  setSize(300,300);
  setVisible(true);
 }
}
```

3. 利用 Graphics2D 的平稳、缩放、旋转功能。绘制一个六角星。

```
import java.applet.*;
import java.awt.*;
import javax.swing.*;
import java.awt.geom.*;
public class SixStarTest extends Applet
{
 public void init()
```

```
 {
  GraphicsDemo myGraphicsDemo=new GraphicsDemo();
 }
}
class MyPanel extends JPanel
{ double x0=100,y0=100;
 double sx=1.5,sy=1.5;
 int x[]=new int[4];
 int y[]=new int[4];
 int i=0;
 double r=60,t;

 MyPanel()
 {
  setBackground(Color.blue);
   for(t=0.0d;t<=2.0*3.1415926d;t+=3.1415926*2.0/3.0d)
  {
  x[i]=(int)(r*Math.cos(t));
  y[i]=(int)(r*Math.sin(t));
  i++;
  if(i==4)break;
  }
 }
 public void paintComponent(Graphics g)
 {
  super.paintComponent(g);
  setBackground(Color.green);
  Graphics2D g2D=(Graphics2D)g;
  g2D.setColor(Color.red);
  Polygon polysix2=new Polygon(x,y,4);
  AffineTransform trans2=new AffineTransform();
  trans2.translate(x0,y0);
  trans2.scale(sx,sy);
  g2D.setTransform(trans2);
  g2D.draw(polysix2);
  trans2.rotate(3.1415927/3.0,sx,sy);
  g2D.setTransform(trans2);
  g2D.draw(polysix2);
 }
}
class GraphicsDemo extends JFrame
{
 public GraphicsDemo()
 {
  this.getContentPane().add(new MyPanel());
  setTitle("六角星绘制");
  setSize(300,300);
  setVisible(true);
 }
```

```
    }
```

4. 编写个用鼠标自由作画的程序。作画程序设置选择画笔颜色、画笔粗细及选用橡皮擦和设定橡皮擦大小的界面。

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.geom.*;
import java.util.*;

    public class PaintFreeTest extends Applet implements ItemListener,MouseMotionListener,ActionListener
    {
      JComboBox combo1,combo2;
     String s1[]={"红色","绿色","蓝色"};
     String s2[]={"2","8","16"};
     JCheckBox box;
     JTextField text1,text2,text3,text4,text5;
     BasicStroke stro;
     int rubber=0,x=-1,y=-1,L=0,W=0,con=3;
     Color c;
     public void init()
     {
      addMouseMotionListener(this);
      combo1=new JComboBox(s1);
      combo2=new JComboBox(s2);
      text1=new JTextField("请选画笔颜色：");
      text2=new JTextField("请选画笔粗细:");
      text3=new JTextField("橡皮擦选择确认");
      text4=new JTextField("输入橡皮擦长 L 和宽 W");
      box=new JCheckBox("橡皮擦");
      text5=new JTextField(5);
      text1.setEditable(false);text2.setEditable(false);
      text3.setEditable(false);text4.setEditable(false);
      combo1.addItemListener(this);
      combo2.addItemListener(this);
      box.addItemListener(this);
      text5.addActionListener(this);

      add(text1);add(combo1);add(text2);add(combo2);
      add(text3);add(box);add(text4);add(text5);
     }
     public void itemStateChanged(ItemEvent e)
     {
      if(e.getSource()==combo1)
      {
       if(combo1.getSelectedItem().toString()=="红色")
       {rubber=0;
        c=new Color(255,0,0);}
       else if(combo1.getSelectedItem().toString()=="绿色")
       {rubber=0;
```

```
    c=new Color(0,255,0);}
   else if(combo1.getSelectedItem().toString()=="蓝色")
   {rubber=0;
    c=new Color(0,0,255);}
  }
    if(e.getSource()==combo2)
  {
   if(combo2.getSelectedItem().toString()=="2")
   {rubber=0;
    stro=new BasicStroke(2f);}
   else if(combo2.getSelectedItem().toString()=="8")
   {rubber=0;stro=new BasicStroke(8f);}
   else if(combo2.getSelectedItem().toString()=="16")
   {rubber=0;stro=new BasicStroke(16f);}
  }
     if(e.getSource()==box)
    {rubber=1;}
  }
public void actionPerformed(ActionEvent e)
{
 if(e.getSource()==text5)
 {
  String s=text5.getText();
  StringTokenizer tokens=new StringTokenizer(s);
  int n=tokens.countTokens();
  for(int i=0;i<=n-1;i++)
  { String temp=tokens.nextToken();
   if(i==0)
   L=Integer.parseInt(temp);
   else if(i==1)
   W=Integer.parseInt(temp);
  }
 }

}

public void paint(Graphics g)
{

 setBackground(Color.orange);
  Graphics2D g2d=(Graphics2D)g;
 if(rubber==0&&x!=-1&&y!=-1)
 {
  g2d.setColor(c);
  g2d.setStroke(stro);
  Line2D line=new Line2D.Double(x,y,x+1,y+1);
  g2d.draw(line);
 }
 else if(rubber==1)
```

```
  {g2d.clearRect(x,y,L,W);}
  //repaint();

 }
 public void update(Graphics g)
  {Graphics2D g2d=(Graphics2D)g;
   paint(g2d);}
 public void mouseDragged(MouseEvent e)
 {
  x=(int)e.getX();y=(int)e.getY();
  repaint();
 }
 public void mouseMoved(MouseEvent e){}
}
```

5. 编写一个应用程序，程序从界面输入二次曲线的系数以及区间，然后画出这条二次曲线。

```
import java.awt.*;
import java.awt.geom.*;
import java.applet.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;

public class CurveTest extends Applet
{
 public void init()
 {
  GraphicsDemo myGraphics=new GraphicsDemo();
 }
}
class MyPanel extends JPanel implements ActionListener
{
 double x,y,a=0,b=0,c=0,e=0,f=0,scalex=10,scaley=0.5,dx=20.0;
 JTextField text1,text2;
 JLabel la1,la2;

 MyPanel()
 {
  setSize(200,200);
  setBackground(Color.blue);
  text1=new JTextField(8);
  text2=new JTextField(8);
  text1.addActionListener(this);
  text2.addActionListener(this);
  la1=new JLabel("请按顺序输入二次曲线方程的系数a,b,c:");
  la2=new JLabel("请输入二次曲线的区间[e f]");
  add(la1);add(text1);add(la2);add(text2);
 }
 public void actionPerformed(ActionEvent e)
```

```
 {
  if(e.getSource()==text1)
  {
   String s=text1.getText();
    StringTokenizer tokens=new StringTokenizer(s);
    int n=tokens.countTokens();
    for(int i=0;i<=n-1;i++)
    { String temp=tokens.nextToken();
     if(i==0)
      a=Double.parseDouble(temp);
      else if(i==1)
      b=Double.parseDouble(temp);
      else if(i==3)
      c=Double.parseDouble(temp);
     }
   }
   if(e.getSource()==text2)
  { String s=text2.getText();
    StringTokenizer tokens=new StringTokenizer(s);
    int n=tokens.countTokens();
    for(int i=0;i<=n-1;i++)
    { String temp=tokens.nextToken();
     if(i==0)
      c=Double.parseDouble(temp);
      else if(i==1)
      f=Double.parseDouble(temp);
    }
  }
 }
 public void paintComponent(Graphics g)
 {
  super.paintComponent(g);
  Graphics2D g2D=(Graphics2D)g;
  setBackground(Color.green);
  QuadCurve2D curve=new QuadCurve2D.Double(e,a*e*e+b*e+c,15*scalex,380*scaley,f*scalex,(a*f*f+b*f+c)*scale
y);
  g2D.setColor(Color.red);
  g2D.draw(curve);repaint();
 }
}
class GraphicsDemo extends JFrame
{
 public GraphicsDemo()
 {
  this.getContentPane().add(new MyPanel());
  setTitle("二次曲线演示");
  setSize(300,300);
  setVisible(true);
```

```
 }
}
```

6. 参照 7.9，编写一个播放音乐的小应用程序。要求音乐文件由对话框指定。

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class AudioTest2 extends Applet implements ActionListener
{
 AudioClip clip;
 MyDialog log;
 Button buttonPlay,buttonLoop,buttonStop;
  static String string1;
 public void init()
 {
  MyDialog log=new MyDialog();
  log.setVisible(true);
  buttonPlay=new Button("开始播放");
  buttonLoop=new Button("循环播放");
  buttonStop=new Button("停止播放");
  buttonPlay.addActionListener(this);
  buttonLoop.addActionListener(this);
  buttonStop.addActionListener(this);
  add(buttonPlay);add(buttonLoop);add(buttonStop);
   }
 public void stop()
 {
  clip.stop();
 }
 public void actionPerformed(ActionEvent e)
 { clip=getAudioClip(getCodeBase(),string1);
  if(e.getSource()==buttonPlay){clip.play();}
  else if(e.getSource()==buttonLoop){clip.loop();}
  if(e.getSource()==buttonStop){clip.stop();}
 }
}
class MyDialog extends JDialog implements ActionListener
{
 JLabel title;JTextField text;
 MyDialog()
 {
  Container con=this.getContentPane();
  title=new JLabel("请输入音乐文件名称");
  text=new JTextField(5);text.setEditable(true);
  con.setLayout(new FlowLayout());
  con.setSize(200,100);setModal(false);
  text.addActionListener(this);
```

```
  con.add(title);con.add(text);
  con.setVisible(true);this.pack();
 }
 public void actionPerformed(ActionEvent e)
 {
  if(e.getSource()==text)
  AudioTest2.string1=text.getText();
  setVisible(false);
 }
}
```

第 8 章　多线程

本章主要内容(重点：15%)

线程的基本概念

Thread 类和 Runnable 接口

线程互斥和同步

本章重点：线程创建和基本控制方法，线程的互斥和同步

本章难点：线程的互斥和同步

8．1　线程的基本概念

要深入了解线程，先要弄清程序、进程与线程 3 个相互关联的基本概念。

程序是一段代码，是计算机执行的蓝本。编写程序就是希望计算机按程序蓝本执行。

进程是程序的一次执行过程，从代码加载、执行，直至完成的一个完整过程。这个过程也是进程从产生、运行至消亡的过程。程序与进程之间的关系如同乐谱与一次演奏关系。乐谱好比程序，演奏好比进程，演奏的依据是乐谱，进程执行的依据是程序。

线程是一个控制流，也是一个执行过程，但单位比进程小。一个进程在执行过程中，可以产生多个线程，形成多条执行线索。每条线索，即每个线程也有它自身的产生、运行和消亡的过程。如果把进程比作一次乐曲的演奏，线程可以比作每个乐师的奏乐。从外面看，一场音乐会给人们一次美好的音乐享受；从内部看，每个乐师正在按要求认真工作。乐师在工作时，相互之间会有协调和配合。

线程和进程比较，它们共同点，都是程序的一个执行过程。不同点是进程是一个实体，每个进程有自己的状态、专用数据段（独立内存资源）；同一个进程下的线程则共享进程的数据段。创建进程时，必须建立其专用数据段；创建线程时不必建立新的数据段。线程不是能够独立运行的程序，而只是某个进程内的一个执行流。

线程的建立和线程间的切换速度大大超过进程，不需要数据段的保护和恢复。同时，又具备进程的大多数优点，所以线程的执行效率比进程的执行效率高。缺点是由于多个线程共享数据段，带来数据访问过程中的互斥和同步问题，使系统管理变得复杂。

多线程在提高输入/输出设备平等工作能力、有效利用系统资源、改善计算机通信及发挥硬件的多处理器功能等方面有很大的优势。

8.1.1 线程的生命周期

一个线程“创建→工作→死亡”的过程称为**线程的生命周期**.线程生命周期共有五个状态:**新建状态、就绪状态、运行状态、阻塞状态和死亡状态**。

1. 新建状态

新建状态指创建了一个线程，但它还没有启动。处于新建状态的线程对象，只能够被启动或终止。例如，以下代码使线程 myThread 处于新建状态：

```
Thread mythread = new Thread();
```

2. 就绪状态

就绪状态是当线程处于新建状态后，**调用了 start()方法**，线程就处于就绪状态。就绪状态线程具备了运行条件，但尚未进入运行状态。处于就绪状态的线程可有多个，这些就绪状态的线程将在就绪队列中排队， 等待 CPU 资源。线程通过线程调试获得 CPU 资源变成运行状态。例如，以下代码使线程 myThread 进入就绪状态：

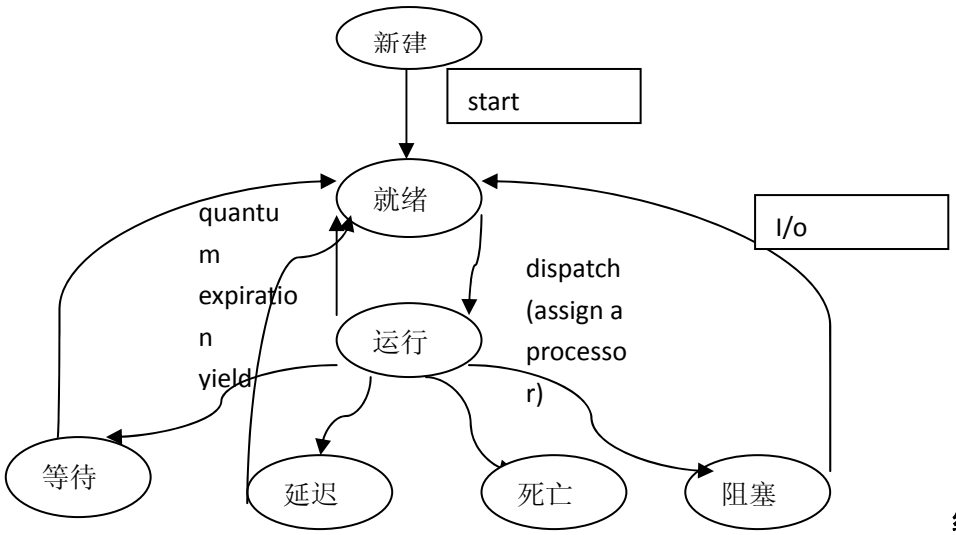
```
myThread.start();
```

3. 运行状态

运行状态是某个就绪状态的线程获得 CPU 资源，正在运行。如果有更高优先级的线程进入就绪状态，则该线程将被迫放弃对 CPU 的控制权，进入就绪状态。使用 yield()方法可以**使线程主动放弃** CPU。线程也可能由于执行结束或执行 stop()方法进入死亡状态。每个线程对象都有一个 run()方法，当线程对象开始执行时，系统就调用该对象的 run()方法。

4. 阻塞状态

阻塞状态是正运行的线程遇到某个特殊情况。例如，延迟、挂起、等待 I／O 操作完成等。进入阻塞状态的线程让出 CPU，并暂时停止自己的执行。线程进入阻塞状态后，就一直等待，直到引起阻塞的原因被消除，线程又转入就绪状态，进入就绪队列排队。当线程再次变成运行状态时，将从原来暂停处开始继续运行。



线程从阻塞状态恢复到就绪状态有三种途径：**自动恢复**（例如：sleep 时间到、I/O 操作完成）；**用 resume()方法恢复**；**用 notify()或 notifyAll()方法通知恢复**。也可能因为别的线程强制某个处于阻塞状态的线程终止，该线程就从阻塞状态进入死亡状态。

5. **死亡状态**

死亡状态是指线程不再具有继续运行的能力，也不能再转到其他状态。一般有两种情况使一个线程终止，**进入死亡状态**。**一是线程完成了全部工作**，即执行完 run()方法的最后一条语句。**另一种是线程被提前强制性终止**。图 8．1 是线程的生命周期图，图中给出从一种状态转变成另一种状态的各种可能的原因。

8.1.2 线程调度与优先级

Java 提供了一个线程调度器来监视和控制就绪状态的线程。线程的调度策略采用**抢占式，优先级高**的线程比优先级低的线程**优先执行**。在**优先级相同**的情况下，就按**“先到先服务”**的原则。**线程的优先级用数值表示，数值越大优先级越高（范围 1~10）**。每个线程根据继承特性自动从父线程获得一个线程的优先级，也可在程序中重新设置。对于任务较紧急的重要线程，可安排较高的优先级。相反，则给一个较低的优先级。

每个 Java 程序都有一个默认的主线程，就是通过 JVM 启动的第一个线程。对于应用程序，主线程执行的是 main()方法。对于 Applet，主线程是指浏览器加载并执行小应用程序的那一个线程。子线程是由应用程序创建的线程。另有一种线程称为**守护线程**（Daemon），这是一种用于监视其他线程工作的服务线程，它的优先级最低。

8.2 Thread 类和 Runnable 接口

Java 程序实现多线程应用有**两种途径**：**一是继承 Thread 类声明 Thread 子类，用 Thread 子类创建线程对象。二是在类中实现 Runnable 接口，在类中提供 Runnable 接口的 run()方法。**无论用哪种方法，都需要 java 基础类库中的 Thread 类及其方法的支持。**程序员能控制的关键性工作有两个方面：一是编写线程的 run()方法；二是建立线程实例。**

8.2.1 Thread 类

Thread 类是用来创建线程和提供线程操作的类。

1．Thread 类为创建线程和线程控制提供以下常用的方法：

(1) Thread()，创建一个线程。线程名按创建顺序为 Thread_1、Thread_2…等。

(2) Thread(String m)，创建一个以 m 命名的线程。

(3) Thread(Runnable target),创建线程，参数 target 是创建线程的目标。目标是一个对象，对象的类要实现 Runnable 接口，类中给出接口的 run()方法。

(4) public Thread(Runnable target,String m)，创建线程，参数 target 是创建线程的目标，m 是线程名。

(5) Thread(ThreadGroup g,String m)，创建一个以 m 命名的线程，该线程属于指定线程组 g。

(6) Thread(ThreadGroup g,Runnable target),创建一个属于指定线程组 g 的线程，target 是线程的目标。

(7) Thread(ThreadGroup g,Runnable target,String m),创建一个线程，名为 m，属于指定的线程组 g，target 是线程的目标。

(8) getPriority()，获得线程的优先级。**（范围 1~10）**。

(9) setPriority(int p),设定线程的优先级为 p。线程创建时，子线程继承父线程的优先级。

优先级的数值越大优先级越高（缺省为 5）。常用以下 3 个优先级：

Thread.MIN_PRIORITY（最低：1），Thread.MAX_PRIORITY（最高：10）和 Thread.NORMAL_PRIORITY（标准）。

(10) start(),启动线程，让线程从新建状态到就绪状态。

(11) run()，实现线程行为（操作）的方法。

(12) sleep(int dt)，让线程休眠 dt 时间，单位是毫秒。例如，代码

```
sleep(100);
```

让线程休眠 100 毫秒时间段。在以后的这个时间段中，其他线程就有机会被执行。当休眠时间一到，将重新到就绪队列排队。由于 sleep()方法可能会产生 InterruptedException 异常，应将 sleep()方法写在 try 块中，并用 catch 块处理异常。**sleep()方法是 static 方法，不可重载。**

(13) currentThread(),获得当前正在占有的 CPU 的那个线程。

(14) getName(),获得线程的名字。

(15) setName(),设置线程的名字。

(16) isAlive()，返回 boolean 类型值，查询线程是否还活跃。

(17) destroy()，强制线程生命期结束。

(18) stop(),强制线程生命期结束，并完成一些清理工作，及抛出异常。

有时，小应用程序的界面已从屏幕消失，但并没有停止线程的执行，会一直到浏览器关闭才结束。要在小应用程序的 stop()方法中终止线程的执行。

(19) suspend()，挂起线程，处理不可运行的阻塞状态。

(20) resume()，恢复挂起线程，重新进入就绪队列排队。

(21) yield()，暂停当前正执行的线程，若就绪队列中有与当前线程同优先□线程，则当前线程让出 CPU 控制权，移到就绪队列的队尾。若队列中没有同优先级或更高优先级的线程，则当前线程继续执行。

2．用 Thread 子类实现多线程

用 Thread 子类实现多线程，得先声明一个 Thread 类的子类，并在子类中重新定义 run()方法。当程序需要建立线程时，就可创建 Thread 子类的实例，并让创建的线程调用 start()方法，这时，run()方法将自动执行。

[例 8.1]应用程序用 Thread 子类实现多线程。在 main()方法中创建了两个线程 threadA 和 threadB，它们分别是 Thread 子类 Athread 和 Bthread 的实例，并分别调用它们的 start()方法启动它们。threadA 先调用 start()方法，类 Athread 中的 run()方法将自动执行。由于 threadA 线程先执行，它的 run()方法输出“我是 ThreadA”和当时的时间后，threadA 线程主动休息 2000 毫秒，让出 CPU。这时正在等待 CPU 的 threadB 线程获得 CPU 资源，执行它的 run()方法，输出“我是 threadB”和当时的时间，threadB 线程主动让出 CPU,1000 毫秒后又来排队等待 CPU 资源。过了 1000 毫秒后，这时 threadA 线程还没有“醒来”，因此又轮到 threadB 执行。再次输出“我是 threadB”……

```
import java.util.Date;
public class Example8_1{
    static Athread threadA;
    static Bthread threadB;
    public static void main(String args[]){
        threadA = new Athread();
        threadB = new Bthread();
        threadA.start();
        threadB.start();
    }
}

class Athread extends Thread{
    public void run(){
        Date timeNow;//为了能输出当时的时间
        for(int i=0;i<=5;i++)
        {
            timeNow = new Date();//得到当前时间
            System.out.println("我是 threadA:" +timeNow.toString());
            try{sleep(2000);}
            catch(InterruptedException e){}
        }
```

```
    }
}

class Bthread extends Thread{
    public void run(){
        Date timeNow;//为了能输出当时的时间
        for(int i=0;i<=5;i++)
        {
            timeNow = new Date();//得到当前时间
            System.out.println("我是 threadB:" +timeNow.toString());
            try{sleep(1000);}
            catch(InterruptedException e){}
        }
    }
}
```

8.2.2 Runnable 接口

Java.lang.Runnable 接口，只有 run()方法需要实现。一个实现 Runnable 接口的类实际上定义了一个在主线程之外的新线程的操作。

用 Runnable 接口实现多线程的主要工作是：声明实现 Runnable 接口的类，在类内实现 run()方法；并在类内声明线程对象，在 init()，方法或 start()方法中创建新线程，并在 **start()方法中启动新线程**。

[例 8.2] 小应用程序通过 Runnable 接口创建线程。在类的 start()方法中构造方法 Thread(this)创建了一个新的线程。this 代表着该类的对象作为这个新线程的目标，因此类必须为这个新创建的线程实现 Runnable 接口，即提供 run ()方法。在 start()方法中构造了一个名为 myThread 的线程，并调用 Thread 类 start()方法来启动这个线程。这样，run()方法被执行，除小应用程序的主线程外，又开始了一个新线程 myThread.在例子中，run()方法在睡眠 1 秒后，调用 repaint()方法重绘 Applet 窗口。在 paint()方法中，在文本区输出线程睡眠的次数，并用随机产生的颜色和半径涂一个圆块。

```
import java.applet.*;
import java.awt.*;
import javax.swing.*;
public class Example8_2 extends java.applet.Applet
implements Runnable{//实现 Runnable 接口
    Thread myThread=null;//声明线程对象
    JTextArea t;
    int k;
    public void start(){
        t = new JTextArea(20,20);
        add(t);k=0;setSize(500,500);
        if(myThread == null){//重新进入小程序时，再次创建线程 myThread
            myThread = new Thread(this);//创建新线程
            mythread.start();//启动新线程
        }
    }
    public void run(){//定义线程的运行代码
        while(myThread!=null){
            try{myThread.sleep(1000);k++;}
            catch(InterruptedException e){}
            repaint();
        }
```

```
    }
    public void paint(Graphics g){
        double i= Math.random();
        if(i<0.5) g.setColor(Color.yellow);
        else  g.setColor(Color.blue);
        g.fillOval(10,10,(int)(100*i),(int)(100*i));
        t.append("我在工作，已休息了"+k+"次\n");
    }
    public void stop(){//离开小程序页时，调用本方法，让线程停止
        if(myThread !=null)
        {
            myThread.stop();
            myThread=null;//重新进入小程序时，再次创建线程
        }
    }
}
```

[例 8.3]　小应用程序创建两个线程，一个顺时针画圆，一个逆时针画圆。例子使用容器类方法 getGraphics()获得 Graphics 对象，给线程作为画笔使用。由于两个线程睡眠时间不同，画圆的速度不一致。程序了为解决图形移动显示时的闪烁问题，采用底色重画原图，将原图擦去的办法。

```
import java.applet.*;
import java.awt.*;
import java.awt.Event.*;
public class Example8_3 extends java.applet.Applet implements Runnable{//实现 Runnable 接口
    Thread redBall,blueBall;
    Graphics redPen,bluePen;
    int blueSeta =0,redSeta=0;
    public void init(){
        setSize(250,200);
        redBall = new Thread(this);blueBall= new Thread(this);
        redPen = getGraphics();bluePen = getGraphics();
        redPen.setColor(Color.red);bluePen.setColor(Color.blue);
        setBackground(Color.gray);
    }
    public void start(){
        redBall.start();
        blueBall.start();
    }

    public void run(){//定义线程的运行代码
        int x,y;
        while(true){
            if(Thread.currentThread() == redBall){
                x= (int)(80.0*Math.cos(3.1415926/180.0*redSeta));
                y= (int)(80.0*Math.sin(3.1415926/180.0*redSeta));
                redPen.setColor(Color.gray);//用底色画圆，擦除原先所画圆点
                redPen.fillOval(100+x,100+y,10,10);
                redSeta +=3;
                if(redSeta>=360)redSeta = 0;
                x= (int)(80.0*Math.cos(3.1415926/180.0*redSeta));
```

```
                y= (int)(80.0*Math.sin(3.1415926/180.0*redSeta));
                redPen.setColor(Color.red);
                redPen.fillOval(100+x,100+y,10,10);
                try{redBall.sleep(20);}
                catch(InterruptedException e){}

            }
            else if(Thread.currentThread == blueBall){
                x= (int)(80.0*Math.cos(3.1415926/180.0*blueSeta));
                y= (int)(80.0*Math.sin(3.1415926/180.0*blueSeta));
                bluePen.setColor(Color.gray);//用底色画圆，擦除原先所画圆点
                bluePen.fillOval(150+x,100+y,10,10);
                blueSeta -=3;
                if(blueSeta<=-360)blueSeta = 0;
                x= (int)(80.0*Math.cos(3.1415926/180.0*blueSeta));
                y= (int)(80.0*Math.sin(3.1415926/180.0*blueSeta));
                bluePen.setColor(Color.blue);
                bluePen.fillOval(150+x,100+y,10,10);
                try{blueBall.sleep(40);}
                catch(InterruptedException e){}
            }
        }
    }
}
```

8.3 线程互斥和同步

通常情况下，程序中的多个线程是相互协调和相互联系的，多线程之间有互斥和同步。

8.3.1 线程互斥

先看线程之间需要互斥的情况。设有若干线程共享某个变量，且都对变量有修改。如果它们之间不考虑相互协调工作，就会产生混乱。比如，线程A和B共用变量x ，都对x执行增1操作。由于A和B没有协调，两线程对x的读取、修改和写入操作会相互交叉，可能两个线程读取同样的x值，一个线程将修改后的x新值写入到x后，另一个线程也把自己对x的修改后的新值写入到x。这样，x只记录后一个线程的修改作用。

```
public class Example8_4{
    public static void main(String arg[]){
        MyResourceClass mrc = new MyResourceClass();
        Thread[] aThreadArray = new Thread[20];
        System.out.println("\t 刚开始的值是:" +mrc.getInfo());
        //20 个线程＊每个线程加 1000 次＊每次加 50
        System.out.println("\t 预期的正确结果是:"+20*1000*50);
        System.out.println("\t 多个线程正在工作，请稍等！");
        for(int i=0;i<20;i++){//产生 20 个线程并开始执行
            aThreadArray[i] = new Thread(new MyMultiThreadClass(mrc));
            aThreadArray[i].start();
        }
    WhileLoop://等待所有线程结束
        while(true){
            for(int i=0;i<20;i++)
            if(aThreadArray[i].isAlive())continue WhileLoop;
            break;
        }
```

```
        System.out.println("\t 最后结果是:" +mrc.getInfo());
    }
}
class MyMultiThreadClass implements Runnable{
    MyresourceClass Useinteger;
    MyMultiThreadClass(MyresourceClass mrc){UseInteger = mrc;}
    public void run(){
        int i, LocalInteger;
        for(i = 0;i< 1000;i++){
            LocalInteger = UseInteger.getInfo();//把值取出来
            LocalInteger +=50;
            try{Thread.sleep(10);//做一些其他的处理
        }catch(InterruptedException e){}
        UseInteger.putInfo(LocalInteger);//把值存回去
        }
    }
}
class MyResourceClass{
    int IntegerResource;
    MyResourceClass(){ IntegerResource =0;}
    public int getInfo(){ return IntegerResource;}
    public void putInfo(int info){IntegerResource = info;}
}
```

程序执行的输出结果是：
刚开始的值是： 0
预期的正确结果是：1000000
多个线程正在工作，请稍等!
最后的结果是：50050
造成最后结果不正确的原因是，可能有多个线程取得的是同一个值，各自累计并存入，累计得慢的，后存入的线程把别的执行快的线程的修改覆盖掉了。所以最终实际结果　比预期的要小的多。**解决多线程互斥的办法是，某个线程在使用共享变量时，别的线程暂时等待，等待正在使用共享变量的线程使用结束**。等到前一个线程使用结束后，才让等待使用共享变量的其他线程中的某一个使用它，而别的线程继续等待。**如能保证它们是逐个使用共享变量的，再多的线程使用共享变量也不会产生混乱。**
多线程互斥使用共享资源的程序段，在操作系统中称为临界段。临界段是一种加锁的机制，与多线程共享资源有关。**临界段的作用是在任何时刻一个共享资源只能供一个线程使用**。当资源未被占用，线程可以进入处理这个资源的段，从而得到该资源的使用权；当线程执行完毕，便退出临界段。如果一个线程已某个共享资源的临界段，并且还没有使用结束，其他线程必须等待。
在 Java 语言中，使用**关键字** synchronized 定义临界段，能对共享对象的操作上锁。如果修改例 8．4 程序，另定义一个累计修改的方法，并将这个方法作为临界段，即某一时刻只允许一个线程做累计修改工作，这就能保证得到正确结果。修改后的 run()方法，及 MyResourceClass 类新增的方法如下：

```
    public void run(){
        for(int i=0;i<1000;i++){
            UseInteger.sumResource(50);
            try{
                Thread.sleep(10);//做一些其他的处理
            }catch(InterruptedException e){}
        }
   }
   class MyResourceClass{
```

```
        int IntegerResource;
        MyResourceClass(){IntegerResource =0;}
        public int getInfo(){return IntegerResource;}
        public void putInfo(int info){ IntegerResource =info;}
        synchronized void sumResource(int q){
            int localInteger;
            LocalInteger = getInfo();//把值取出来
            LocalInteger +=q;
            putInfo(LocalInteger);//把值存回去
        }
    }
```

8.3.2 线程同步

多线程之间除有互斥情况外,还需要同步.**当线程 A 使用到某个对象,而此对象又需要线程 B 修改后才能符合本线程的需要,这时线程 A 就要等待线程 B 完成修改工作.这种线程相互等待称为线程的同步.**

为了实现同步,Java 语言提供 wait()、notify()和 notifyAll()三个方法供线程在临界段中使用。在临界段中使用 wait()方法，使执行该方法的线程等待，并允许其他线程使用这个临界段。wait()方法常用以下两种格式：

wait(),让线程一直等待，直到被 notify()或 notifyAll()方法唤醒。

wait(long timeout),让线程等待到被唤醒，或经过指定时间后结束等待。

当线程使用完临界段后，用 notify()方法通知由于想使用这个临界段而处于等待的线程结束等待。**notify()方法只是通知第一个处于等待的线程。如果某个线程在使用完临界段方法后,其他早先等待的线程都可结束等待,重新竞争 CPU,则可以 notifyAll()方法。**

[例 8.5] 小应用程序模拟一群顾客购买纪念品。设纪念品 5 元钱一个，顾客有人持 5 元、有人持 10 元购买纪念品。持 5 元的顾客能产即买到纪念品，持 10 元的顾客，如果销售员有可找的小面额钱，则也能立即购买；如果销售员没有可找的小面额钱，则这位顾客就得等待。程序模拟这群顾客购买纪念品的过程。

```
class SalesLady{
    int memontoes,five,ten;
    public synchronized String ruleForSale(int num,int money){
        String s = null;
        if( memontoes ==0) return "对不起,已售完!";
        if(money ==5){
            memontoes--;five++;
            s = "给你一个纪念品,"+"你的钱正好.";
        }else if(money ==10){
            while(five<1){
                    try{
                    System.out.println(""+ num+ "号顾客用 10 元钱购票,发生等待!");
                    wait();
                    }catch(InterruptedException e){}
                }
                memontoes--;five -=1;ten++;
                s = "给你一个纪念品,"+"你给了十元,找你 5 元.";
            }
            notify();
            return s;
        }
        SalesLady(int m,int f, int t){
            memontoes =m; five =f; ten =t;
        }
    }
```

```
public class Example8_5 extends java.applet.Applet{
    static SalesLady salesLady = new SalesLady(14,0,0);
    public void start(){
        int moneies[]={10,10,5,10,5,10,5,5,10,5,10,5,5,10,5};
        Thread []aThreadArray = new Thread[20];
        System.out.println("现在开始购票");
        for(int i=0;i<moneies.length;i++){
            aThreadArray[i]= new Thread(new CustomerClass(i+1,moneies[i]))
            aThreadarray[i].start();
        }
        WhileLoop;
        while(true){
            for(int i=0;i<moneies.length;i++)
            if( aThreadArray[i].isAlive() )
                continue WhileLoop;
            break;
        }
        System.out.println("购票结束！");
    }
}
public CustomerClass impements Runnable{
    int num,money;//顾客序号，钱的面值
    public void run(){
        try{Thread.sleep(10);
        }//假定顾客在购买前还做一些其他的事
        catch(InterruptedException e){}
        System.out.println("我是" + num +
        "号顾客，用" + money +"元购纪念品，售货员说："
        + Example8_5.salesLady.ruleForSale(num,money));
    }
    CustormerClass(int n,int m){num =n;money = m;
    }
}
```

请注意实现线程同步的一般原则：如果两个或多个线程修改同一个对象，那么将执行修改的操作方法用关键字 synchronized 修饰之，使它成为临界段。例如，在上述例子中，许多顾客线程共享 saleslady 对象，其中 ruleForSale() 作为临界段。

如果进入临界段的线程必须等待某个对象的状态被改变，那么应调用 wait() 方法，使线程等待。反之，当另一个进入临界段的线程修改了某个对象的状态后，就应该调用 notify() 方法，及时通知那些处于等待的线程，它们等待的环境已经发生了改变。

有三种类型的代码段能够作为**临界段：类方法、实例方法、一个方法中的代码块**。下面的同步方法：

```
synchronized void myMethod(){//需要同步的代码
}
```

可以等价地描述成如下：

```
void myMethod(){
    synchronized(this){需要同步的代码}
}
```

如果发生多个线程形成一个等待环，即第一个线程在等候第二个线程，而第二个线程在等候第三个线程，依次类推，最后一个线程在等候第一个线程。这样，所有线程都陷入相互等待的状态。这个循环等待现象称为死锁。在互斥同步机制的实现中，稍有不当，就可能产生死锁。Java 语言对死锁问题没有特别处理，只能由用户在编程时注意。

习题

1. 建立线程有哪两种方法？

   一是继承 Thread 类声明 Thread 子类，用 Thread 子类创建线程对象。二是在类中实现 Runnable 接口，在类中提供 Runnable 接口的 run()方法。无论用哪种方法，都需要 java 基础类库中的 Thread 类及其方法的支持。程序员能控制的关键性工作有两个方面：一是编写线程的 run()方法；二是建立线程实例。

2. 怎样设置线程的优先级？

   setPriority(int p),设定线程的优先级为 p(1~10)。线程创建时，子线程继承父线程的优先级。

   优先级的数值越大优先级越高（缺省为 5 ）。常用以下 3 个优先级：

   Thread.MIN_PRIORITY（最低），Thread.MAX_PRIORITY（最高）和 Thread.NORMAL_PRIORITY（标准）

3. 编写一个小应用程序，在小应用程序有两个线程，一个负责模仿圆转运动，另一个模仿椭圆运动。

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class ThreadTest3 extends Applet implements Runnable
{
 Thread threadArc,threadOval;
 Graphics arcPen,ovalPen;
 int arcSeta=0,ovalSeta=0;
 public void init()
 {
  setSize(300,300);
  threadArc=new Thread(this);
  threadOval=new Thread(this);
  arcPen=getGraphics();
  ovalPen=getGraphics();
  arcPen.setColor(Color.red);
  ovalPen.setColor(Color.black);
  setBackground(Color.blue);
 }
 public void start()
 {
  threadArc.start();
  threadOval.start();
 }
 public void run()
 {
  int x,y;
  while(true)
  {
  if(Thread.currentThread()==threadArc)
  {
   x=(int)(80*Math.cos(3.1415926/180.0*arcSeta));
   y=(int)(80*Math.sin(3.1415926/180.0*arcSeta));
   arcPen.setColor(Color.blue);
   arcPen.fillOval(100+x,100+y,10,10);
   arcSeta+=3;
   x=(int)(80*Math.cos(3.1415926/180.0*arcSeta));
   y=(int)(80*Math.sin(3.1415926/180.0*arcSeta));
   arcPen.setColor(Color.red);
```

```
        arcPen.fillOval(100+x,100+y,10,10);
         try{threadArc.sleep(20);}
         catch(InterruptedException e){}
       }
       if(Thread.currentThread()==threadOval)
       {
        x=(int)(80*Math.cos(3.1415926/180.0*ovalSeta));
        y=(int)(40*Math.sin(3.1415926/180.0*ovalSeta));
        ovalPen.setColor(Color.blue);
        ovalPen.fillOval(100+x,100+y,10,10);
        ovalSeta+=3;
        x=(int)(80*Math.cos(3.1415926/180.0*ovalSeta));
        y=(int)(40*Math.sin(3.1415926/180.0*ovalSeta));
        ovalPen.setColor(Color.black);
        ovalPen.fillOval(100+x,100+y,10,10);
         try{threadOval.sleep(50);}
         catch(InterruptedException e){}
       }
       }
     }
   }
```

4. 在多线程程序中，要考虑互斥的原因是什么？在 Java 中，是如何解决这个问题的？

多线程之间要共享资源，为了保护资源在使用时，不被其他线程使用。

在 Java 语言中，使用关键字 synchronized 定义临界段，能对共享对象的操作上锁。

5. 在多线程程序中，要考虑同步的原因是什么？在 Java 中，是如何解决这个问题的？

在临界段中使用 wait()方法，使执行该方法的线程等待，并允许其他线程使用这个临界段。wait()方法常用以下两种格式：

wait(),让线程一直等待，直到被 notify()或 notifyAll()方法唤醒。

wait(long timeout),让线程等待到被唤醒，或经过指定时间后结束等待。

当线程使用完临界段后，用 notify()方法通知由于想使用这个临界段而处于等待的线程结束等待。**notify()方法只是通知第一个处于等待的线程**。如果某个线程在使用完临界段方法后，其他早先等待的线程都可结束等待，重新竞争 CPU，则可以 notifyAll()方法。

6. 模拟人排队买球票，设置球票 5 元一张，购票者有持 5 元的、有持 10 元的、有持 20 元的、有持 50 元的、有持 100 元的。已知排队者序列，试给出排队购票情况。

7. 修改例 8．5 程序，使一些暂时找不到零钱的顾客能按先来先买规则，排队等待购买纪念品。

```
class SalesLady
{
 static int memontoes,five,ten;
 SalesLady(int m,int f,int t)
 {
  memontoes=m;five=f;ten=t;
 }
 public synchronized void ruleForSale(int num,int money)
 {
  String s=null;
  if(memontoes==0)
  {
   s="对不起,已销完!";
```

```
    System.out.println("我是"+num+"号顾客，用"+money+"元购纪念品，售货员说："+s);
   }
   if(money==5&&memontoes>0)
   {
     memontoes--;
    five++;
     s="给你一个纪念品,"+"你的钱正好。"+memontoes;
     System.out.println("我是"+num+"号顾客，用"+money+"元购纪念品，售货员说："+s);
     if(five>=1)notify();
   }
   else if(money==10)
   {
    while(five<1)
    {
     try{System.out.println(" "+num+"号顾客用 10 元钱购票，发生等待!");
      wait();if(memontoes<=0){s="对不起,已销完!";System.out.println("我是"+num+"号顾客,用"+money+"元购纪念
品，售货员说："+s);}}
     catch(InterruptedException e){}
    }
    memontoes--;
    five-=1;ten++;
     s="给你一个纪念品,"+"你给了十元，找你五元。"+memontoes;
     System.out.println("我是"+num+"号顾客，用"+money+"元购纪念品，售货员说："+s);
    }
   notifyAll();
 }

}
public class RunnableTest5 extends java.applet.Applet
{
 static SalesLady salesLady=new SalesLady(14,0,0);
 public static Thread[] aThreadArray=new Thread[20];
 public void start()
 {
  int moneies[]={10,10,5,10,5,10,5,5,10,5,10,5,5,10,5};
  System.out.println("现在开始购票:");
  for(int i=0;i<moneies.length;i++)
  {
   aThreadArray[i]=new Thread(new CustomerClass(i+1,moneies[i]));
   aThreadArray[i].start();
  }
  WhileLoop:
  while(true)
  {
   for(int i=0;i<moneies.length;i++)
     if(aThreadArray[i].isAlive())
     continue WhileLoop;
   break;
  }
```

```
  System.out.println("购票结束!");
 }
}
class CustomerClass implements Runnable
{
   int num,money;
   public void run()
 {

  if(Thread.currentThread()==RunnableTest5.aThreadArray[0])
   RunnableTest5.salesLady.ruleForSale(num,money);
   if(Thread.currentThread()==RunnableTest5.aThreadArray[1])
   RunnableTest5.salesLady.ruleForSale(num,money);
   if(Thread.currentThread()==RunnableTest5.aThreadArray[2])
   RunnableTest5.salesLady.ruleForSale(num,money);
   if(Thread.currentThread()==RunnableTest5.aThreadArray[3])
   RunnableTest5.salesLady.ruleForSale(num,money);
   if(Thread.currentThread()==RunnableTest5.aThreadArray[4])
   RunnableTest5.salesLady.ruleForSale(num,money);
   if(Thread.currentThread()==RunnableTest5.aThreadArray[5])
   RunnableTest5.salesLady.ruleForSale(num,money);
   if(Thread.currentThread()==RunnableTest5.aThreadArray[6])
   RunnableTest5.salesLady.ruleForSale(num,money);
   if(Thread.currentThread()==RunnableTest5.aThreadArray[7])
   RunnableTest5.salesLady.ruleForSale(num,money);
   if(Thread.currentThread()==RunnableTest5.aThreadArray[8])
   RunnableTest5.salesLady.ruleForSale(num,money);
   if(Thread.currentThread()==RunnableTest5.aThreadArray[9])
   RunnableTest5.salesLady.ruleForSale(num,money);
   if(Thread.currentThread()==RunnableTest5.aThreadArray[10])
   RunnableTest5.salesLady.ruleForSale(num,money);
   if(Thread.currentThread()==RunnableTest5.aThreadArray[11])
   RunnableTest5.salesLady.ruleForSale(num,money);
   if(Thread.currentThread()==RunnableTest5.aThreadArray[12])
   RunnableTest5.salesLady.ruleForSale(num,money);
   if(Thread.currentThread()==RunnableTest5.aThreadArray[13])
   RunnableTest5.salesLady.ruleForSale(num,money);
   if(Thread.currentThread()==RunnableTest5.aThreadArray[14])
   RunnableTest5.salesLady.ruleForSale(num,money);
   }
 CustomerClass(int n,int m)
 {num=n;money=m;}
}
```

第 9 章 输入和输出流

本章主要内容(一般掌握 3%~5%)

字节流和字符流

缓冲式输入输出

随机访问

文件对话框

本章重点:字节流和字符流的输入和输出/缓冲式输入输出，以及使用文件对话框的方法。

本章难点：缓冲式输入输出、随机访问，使用文件对话框指定输入输出文件。

9.1 字节流和字符流

对 Java 程序而言，输入/输出包括：对外设通道的输入/输出、对文件的读和写、对网络数据的读和写。Java 语言采用流（stream）的机制实现输入和输出。**流是一种有方向的字节/字符数据序列**。程序为获取数据，打开一个来自数据源的流，通过流读取数据。程序为输出数据，打开一个到目的地的流，就可以将数据输出到目的地。对于程序来说，I/O 流提供一条数据通道，输入流指的是数据的来源，程序从输入流中读取数据；输出流是数据要去的目的地，程序向输出流写的数据被传送到目的地。虽然大多数情况，I/O 流与磁盘文件有关，但数据源也可以是键盘、鼠标、内存、数据的目的地也可以是内存、显示器。

实现输入/输出操作的类库在 java.io 包中。包中有一组支持打开流、向流写数据、从流读取数据和关闭流等操作的类，要使用这些类，程序必须导入 java.io 包。

9.1.1 File 对象

建立输入输出流对象之前可以先建立一个 File（文件）对象。File 对象是 File 类的实例。File 对象对应一个目录或文件，对象的属性包括文件路径、名字、文件长度、可否读写等。**File 对象只用来命名文件、查询文件属性和处理目录，不提供读写文件操作**。

File 类常用的构造方法：

（1） File(String s)，由 s 确定 File 对象的文件名。

（2） File(String directory ,String s)，由 directory 确定 File 对象的文件目录，由 s 确定文件名。

以下是创建 File 对象的示意代码：

```
File  f1 = new file(“/mydir/myfile.txt”);
File  f2 = new File(“/mydir”,” myfile.text”);
```

9.1.2 流对象

Java 程序的输入和输出流称为流对象，文件读写的主要操作包括：建立流对象、读数据、写数据和关闭流。输入输出流由文件名或 File 对象创建。

根据输入输出数据的类型是字节还是字符，java.io 提供两种基本抽象类：一种用于**读写字节**，另一种用于**读写字符**。

字节流用于读写二进制数据，比如图像和声音文件等。**字节流数据是 8 位的**，由 InputStream 类和 OutputStream 类为字节流提供 API 和部分实现。**字节流读和写操作由** FileInputStream 类和 FileOutputStream 类提供支持。它们分别是 InputStream 和 OutputStream 类的子类。

字符流数据是 16 位的 Unicode 字符，字符流能处理 Unicode 字符集中的任何字符，由 Reader 类和 Writer 类为字符流提供 API 和部分实现。**读和写操作**由 FileReader 类和 FileWriter 类提供支持。它们分别是 Reader 和 Writer 的子类。

1. 建立流对象

通过流的构造方法建立流对象，创建输入输出流对象的常用构造方法用 2 种，下面以创建 FileInputStream 类的流对象为例说明创建流对象的方法，创建 FileOutputStream 类、FileReader 类和 FileWriter 类的流对象与此完全类似。

（1） FileInputStream(String name),直接指定的文件名创建输入流。

（2） FileInputStream(File filename),用文件对象创建输入流。

由于使用输入流的构造方法创建输入流时，可能会出现 FileNotFoundException 异常。所以创建输入流的代码必须出现在 try…catch 语句结构中，让程序能检测并处理这个异常。所以，创建输入流代码呈以下形式：

```
try{
```

```
    FileInputStream ins = new FileinputStream(“myfile.data”);//创建输入流对象。
} catch(FileNotFoundException e){
    System.out.println(“文件找不到：”+e);
}
```

上述代码把输入流对象与文件关联起来，使用 catch 块捕获并处理文件找不到异常。

2. 输入流的常用方法

FileInputStream 类包含下面用于**读字节和字节数组**的方法：

（1） int read()从输入流读取下一个字节，返回一个整数，整数范围在 0 ~255 之间。

（2） int read(byte b[]),从输入流读取长度为 b.length 的字节数据，写入到字节数组 b,并返回实际所读取的字节数。

（3） int read(byte b[],int off, int len),从数据流中读取长度为 len 的字节数据，写入到字节数组 b 中从下标 off 开始的数组元素中，并返回实际读取的字节数。

使用以上三个方法时，如果在读数据之前，已到达输入的末端位置，没有读入一个字节数据，用返回-1 表示流在读之前已结束。

FileReader 类包含下面用于**读字符和字符数组**的方法：

（1） int read(),从输入流读取下一个字符。如果读前已到流的末尾，则返回-1。

（2） int read(char b[]),从输入流读取长度为 b.length 的字符数据，写入到字符数组 b，并返回实际所读取的字符数。

（3） int read(char b[],int off,int len)，从数据流中读取长度为 len 的字符数据，写入到字符数组 b 中从下标 off 开始的数组元素中，并返回实际读取的字符数。

对于输入流，另外还有一个常用的方法：

long skip(long n)，从输入流的当前读位置起向前移动 n 个字节/字符，并返回实际跳过的字节/字符数。

3. 输出流的常用方法

FileOutputStream 类包含下面用于**输出字节和字节数组**的方法：

(1) int write(int c),输出一个字节。

(2) int write(byte b[]),输出存于字节数组 b 中的字节数据。

(3) int write(byte b[],int off,int len),输出字节数组 b 中从下标 off 开始的多至 len 个字节。

FileWriter 类也包含用于**输出字符和字符数组**的方法：

(1) int write(int c),输出一个字符。

(2) int write(char b[]),输出存于字符数组 b 中的字符数据。

(3) int write(char b[],int off,int len),输出字符数组 b 中人下标 off 开始的多至 len 个字符。

对于输出流，另外还有一个常用方法：

void flush(),刷空输出流，并且输出所有存储在缓冲区的内容。

流使用结束后，关闭流并且释放与该流相关的资源用方法 close()。为了程序安全，也为了提高计算机的资源利用率。流使用结束后，程序应及时关闭它。

[例 9．1]一个文件复制应用程序，将某个文件的内容全部复制另一个文件。

```
import java.io.*;
class Example9_1{
    public static void main(String arg[]){
        File inputFile = new File("file1.txt");
        File outputFile = new File("file2.txt");
        int ch;
        try{
            FileReader in = new FileReader(inputFile);
            FileWriter out = new FileWriter(outputFile);
            System.out.println("文件复制程序开始工作");
            while((ch =in.read())!=-1){
                out.write(ch);
            }
```

```
            in.close();
            out.close();
            System.out.println("文件复制程序工作结束");
        }catch(FileNotFoundException e1){
            System.out.println("文件没有找到"+e1);
        }catch(IOException e2){
            System.out.println("File read Error:"+e2);
        }
    }
}
```

上述程序非常简单，为 file1.txt 创建一个输入流，为 file2.txt 创建一个输出流，程序逐一从输入流读取字符，输出到输出流上，直至输入流结束。

9．2　缓冲式输入输出

当程序处理的文件按行组织，并且行不是定长时，用前面所述的流类实现很不方便。**程序如果要按行输入输出，需采用缓冲式输入输出方式。**

9．2．1 缓冲式输入

采用缓冲式输入时，对于程序的输入请求，系统一次性输入足够多的内容放在内存缓冲区中。供程序以后的输入请求使用，待缓冲区内容用完，再一次性输入足够多的数据。

程序要采用缓冲式输入，只要先创建 FileReader 对象，再利用 FileReader 对象创建 BufferedReader 对象，习惯称为 FileReader 对象接到 BufferedReader 对象上。如以下代码所示：

```
BufferedReader in;
file = new FileReader(“abc.txt”);//创建 fileReader 对象
in = new BufferedReader(file);//接到 BufferedReader 类对象上
in.readLine();
```

这样，就可以对 BufferedReader 对象 in 使用 readLine()方法，发行输入文件“abc.txt”.

[例 9．2]说明 BufferedReader 类的用法的应用程序。

```
import java.io.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
class MyWindow extends jFrame implements ActionListener{
    JTextArea text;
    BufferedReader in;
    JButton  Button1;
    FileReader file;
    MyWindow(){
        super("缓冲区中读取");
        Container con = this.getContentPane();//获得内容面板
        con.setSize(100,400);
        con.setLayout(new BorderLayout());
        Button1 = new JButton("读取文件");
        Button1.addActionListener(this);
        text = new JTextArea(20,20);
        text.setBackground(Color.cyan);
        JScrollPane jsp = new JScrollPane(text);
        con.add(jsp,BorderLayout.CENTER);
        con.add(Button1,"South");
        this.setVisible(true);
        this pack();
```

```
            try{File f = new File("Example9_3.java");
                file = new FileReader(f);// 或者 file = new FileReader(“Example9_3.java”);
                in = new BufferedReader(file);
            }catch(FileNofoundException e1){
                text.setText("文件没有找到");
                button1.removeActionListener(this);
            }
        }
        public void actionPerformed(ActionEvent e)
        {
            String s;
            if(e.getSource()==button1)
            try{while ((s = in.readLine())!=null)
                text.append(s+'\n');
            }catch(IOException exp){}
        }
    }
    public class Example9_2{
        public static void main(String arg[]){
            MyWindow myWin = new MyWindow();
        }
    }
```

9.2.2 缓冲式输出

采用缓冲式输出时，对于程序的输出请求，系统先将内容暂存于缓冲区，待缓冲区满或输出结束，才将暂存于缓冲区中的内容输出到流的目的地。

BufferedWriter 类是提供缓冲式输出的类。程序只要先**创建 FileWriter 对象**，再选用 FileWriter 对象**创建 BufferedWriter 对象**，习惯称为 FileWriter 对象接到 BufferedWriter 对象上。对 BufferedWriter 对象**使用 write()** 方法就能实现缓冲式输出。采用缓冲式输出时，write()方法只是将字符串写入到系统内设的缓冲区，等缓冲区满后，系统自动将缓冲区中内容写入到文件。**如果想立即写入到文件，则需要调用** flush()方法。

[例 9.3]说明 BufferedWriter 类用法的应用程序。

```
import java.util.*;
import java.io.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
class MyWindow extends JFrame implements ActionListener{
    JTextArea text;
    JButton  Button1;
    FileWriter writefile;
    BufferedWriter out;
    MyWindow(){
        super("缓冲式样流的输出");
        Container con = this.getContentPane();//获得内容面板
        //con.setSize(100,400);
        con.setLayout(new BorderLayout());
        Button1 = new JButton("写文件");
        Button1.addActionListener(this);
        text = new JTextArea(20,30);
        text.setBackground(Color.cyan);
```

```
        con.setSize(40,40);
        con.add(text,"Center");
        con.add(Button1,"South");
        con.setVisible(true);
//      this pack();
        try{
            writefile = new FileWriter("line.txt");
            out = new BufferedWriter(writefile);
        }catch(IOException e){}
    }
    public void actionPerformed(ActionEvent e)
    {
        String s;
        if(e.getSource()==button1)
        try{out.write(text.getText(),0,(text.getText()).length());
            out.flush();
            text.setText(null);
            System.exit(0);
        }catch(IOException exp)
        {text.setText("文件写出错!\n");System.exit(-1);
        }
    }
}
public class Example9_3{
    public static void main(String args[]){
        MyWindow myWin = new MyWindow();
        myWin.pack();
    }
}
```

9.3 随机访问

流在某些应用中，不希望顺序读写流。例如，跳过部分不需要的数据，去读更后面的数据；或者对早先读过的数据要重新读入等。这样的应用需采用随机访问方式，本节介绍文件随机访问的方法。

文件随机访问可利用 RandomAccessFile 类实现，RandomAccessFile 类创建的流既可以随机输出，也可以随机输入。

1．RandomAccessFile 类构造方法

RandomAccessFile 类常用的构造方法是 RandomAccessFile(String name, String mode)和 RandomAcessFile(File file,String mode)。参数 mode 取值："r"（只读）或 "rw"（可读写）。构造方法可能产生"文件不存在"异常，创建对象的代码应能捕获 FileNotFoundException 异常。对流进行读写操作时，还应该能捕获 IOException 异常。

2．RandomAcessFile 类读写文件的方法

RandomAccessFile 类常用的读写文件方法有：

（1） int read(),读取一个字节，当前读位置往前移动 8 个二进制位。

（2） int read(byte b[]),把文件中的数据读到一个字节数组，当前读位置顺次往前移动。

（3） int read(byte b[],int offset, int length),从文件中读取 length 个字节，把读入的字节存到数组 b[],存储从偏移量为 offset 的位置开始；当前读位置也往前移动。

（4） String readLine(),读取文件中的一行字节，遇到换行，结束一次读入过程。

（5） String readUTF(),从一个 UTF 格式的文件中读取一个字符串。

（6） boolean readBoolean(),读取一个布尔值。

（7） byte readByte(),读取一个字节。

（8） short readShort(),读取一个 short 型整数。

（9） long readLong(),读取一个 long 型整数。
（10） char readChar(),读取一个字符。
（11） int readInt(),读取一个整数。
（12） void writeBoolean(boolean v),写一个布尔值。
（13） void writeByte(int v),写一个字节。
（14） void writeShort(int v),写一个 short 型整数。
（15） void writeChar(int v),写一个字符。
（16） void writeInt(int v),写一个型整数。
（17） void writeLong(long v),写一个 long 型整数。
（18） void writeFloat(float v),写一个浮点数。
（19） void writeDouble(double v),写一个双精度浮点数。
（20） void writeBytes(String v),写一个字符串作为字符序列，写这个字符串。
（21） void writeChars(String v),写一个字符串作为字符序列，写这个字符串。
（22） void seek(long offset),移动当前读写位置距文件开始 offset 位置。

[例 9.4]一个文件随机读写的应用程序。程序把若干 long 型整数写到一个名字为 longData.dat 的文件中，然后按相反顺序读出这些写入的整数。

```
import java.io.*;
public class Example9_4{
    public static void main(String arg[]){
    RandomAccessFile inOut = null;
    long data[]={151,278,235,206,-170,32,43,21,83,47};
    try{
        inOut = new RandomAccessFile("longData.dat","rw");
        for(int i=0;i<data.length;i++){
            inOut.writeLong(data[i]);
        }
        for(int i=data.length-1;i>=0;i--){
        inOut.seek(i*8);//long  型数据占 8 个字节，第 i 个整数自 i*8 字节开始
            System.out.print(" " + inOut.readLong()+((i==0)?"\n":","));
        }
        inOut.close();
    }
    catch(FileNotFoundException e1){
        System.out.println("文件找不到！" +e1);
    }
    catch(IOException e2){System.out.println("文件读写错误！" +e2);}
    }
}
```

[例 9. 5]应用程序逐行读取自己的源代码显示。程序利用 getFilePointer()得到文件的当前位置，与 fileLength()文件的长度比较用于控制循环。

```
import java.io.*;
public class Example9_5{
    public static void main(String []args){
        try{
            RandomAccessFile File=new RandomAccessFile("Example9_5.java","rw");
            long fileCurPos=0;
            long fileLength=File.length();
            while (fileCurPos<fileLength){
                String s=File.readLine();
```

```
                System.out.println(s);
                fileCurPos=File.getFilePointer();
            }
            File.close();
        }
        catch(FileNotFoundException e1){
            System.out.println("文件找不到!"+e1);
        }
        catch(IOException e2){System.out.println("文件读写错!"+e2);}
    }
}
```

9.4 文件对话框

大多数应用程序要使用文件，或读取文件的内容，或将数据保存于文件中，其中文件通过文件对话框指定。本节介绍程序使用文件对话框打开和保存文件的方法。可以用 javax.swing 包中的类 JFileChooser 实现打开和保存文件对话框。

1．JFileChooser 类的构造方法

(1) JFileChooser()，建立一个 JFileChooser 对象，默认的文件对话框路径是用户的 Home 目录。

(2) JFileChooser(String currentDirectoryPath),建立一个 JFileChooser 对象，并设置文件对话框的打开路径。

2．JFileChooser 类的其他常用方法

(1) showOpenDialog(),打开“打开文件对话框”。

(2) showSaveDialog(),打开“保存文件对话框”。

文件对话框打开后，在用户按下按钮或关闭对话框时，上述两个方法会返回一个整数值，这个整数值是以下三个之一：

JFileChooser.CANCEL_OPTION,用户按下“撤销”按钮。

JFileChooser.APPROVE_OPTION,用户按下“打开/保存”按钮。

JFileChooser.ERROR_OPTION,有错，或是对话框非正常关闭。

当程序发现用户选择了文件并按下了“打开/保存”按钮后，程序就可以利用 getSelectedFile()方法取得文件对象，并利用这个文件对象用方法 getName()取得文件的名称，用方法 getPath()取得文件的路径。

3．设置筛选条件

在打开文件对话框中，还可以设置筛选条件，即指定文件的类型。使用 FileFilter 类，该类预设两个方法，分别是 accept(File f),与 getDescription()。当目录中的文件与筛选条件相符时，方法 accept()返回 true，并将此文件名显示在对话框中。而 getDescription()方法则是对筛选条件的描述，由程序指定，例如，“*.java”等。

由于 FileFilter 类是一个抽象类，程序要设置打开文件对话框的文件筛选条件，就应该继承这个类，编写一个实现类，实现上述两个方法。然后，就可以使用 JFileChooser 类 addChoosableFileFilter()方法，或者是 setFileFilter()方法设置筛选条件。

[例 9.6]应用程序利用文件对话框确定数据文件。程序首先弹出一个界面，有两个按钮：打开文件和保存文件。当点击打开文件按钮时，程序弹出一个打开文件对话框，当点击保存文件时，弹出保存文件对话框。程序还继承 FileFilter 类，定义了 MyFileFilter 类，程序利用这个类设置文件对话框的筛选条件。

```
import java.awt.*;
import javax.swing.*;
import java.io.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.filechooser.*;
import javax.swing.filechooser.FileFilter;
public class Example9_6{
public static void main(String arg[]){
    FrameFiledialog f = new FrameFiledialog();
}
```

```
}
class FrameFiledialog extends JFrame implements ActionListener{
JFileChooser filedialog = null;
JLable lable = new JLable(" " ,JLable.CENTER);
jButton b1,b2;JTextArea text;
FrameFiledialog(){
    super("带文件对话框的窗口");
    Container con = this.getContentPane();
    con.setLayout(new BorderLayout());
    con.setSize(40,50);
    JPanel p = new JPanel();
    b1 = new JButton("打开文件");
    b2 = new jButton("保存文件");
    b1.addActionListener(this);
    b2.addActionListener(this);
    p.add(b1);
    p.add(b2);
    text = new JTextArea(20,30);
    JScrollPane jsp = new JScrollPane(text);
    filedialog = new JFileChooser("D:\\workspace");
    filedialog.setControlButtonsAreShown(true);
    filedialog.addChoosableFileFilter(new MyFileFilter("txt"));
    filedialog.addChoosableFileFilter(new MyFileFilter("java"));
    text.setBackground(Color.cyan);
    con.add(jsp,BorderLayout.CENTER);
    con.add(Label,BorderLayout.NORTH);
    con.add(p,BorderLayout.SOUTH);
    this.setVisible(true);
    this.pack();
}
public void actionPerformed(ActionEvent e){
    File file = null;
    int result;
    if(e.getSource()==b1){//打开文件
        filedialog.setDialogTitle("打开文件");
        result = filedialog.showOpenDialog(this);
        text.setText("");
        if(result ==JFileChooser.APPROVE_OPTION){
            file = filedialog.getSelectedfile();
            lable.setText("你选择打开的文件名称是：" + file.getname());
        }else if (result ==JFileChooser.CANCEL_OPTION){
            lable.setText("你没有选择任何文件");
        }
        FileInputStream fileStream = null;
        if(file ! = null){
            try{
                fileStream = new FileInputStream(file);
            }catch(FileNotFoundException nfe){
                lable.setText("文件没有找到");return;
```

```
            }
            int readByte;
            try{
                while((readByte = fileStream.read())!=-1)
                text.append(String.valueOf((char)readByte));
                fileStream.close();
            }catch(IOException ie){label.settext("读取文件出错");}
        }
    }else if (e.getsource()==b2){//保存文件
        filedialog.setDialogTitle("保存文件");
        result = filedialog.showSaveDialog(this);
        file = null;
        String fileName;
        if(result ==JFileChooser.APPROVE_OPTION){
            file = filedialog.getSelectedFile();
            label.setText("你选择保存的文件名称是：" + file.getName());
        }else if(result ==JFileChooser.CANCEL_OPTION){
            label.setText("你没有选择任何文件");
        }
        FileOutputStream fileStream = null;
        if(File !=null){
            try{fileStream = new FileOutputStream(file);
            }catch(FileNotFoundException nfe){
                label.setText("文件没有发现");return;
            }
            String content = text.getText();
            try{
                fileStream.write(content.getBytes());
                fileStream.close();
            }catch(IOException ie){abel.setText("写文件出错");}
        }
    }
}
}
class MyFileFilter extends FileFilter{
String ext;
MyFileFilter(String t){ext =t;}
public boolean accept(File file){
    if(file.isDirectory()) return true;
    String fn = file.getName();
    int index = fn.lastIndexOf('.');
    if(index >0&& index <fn.length()-1){
        String extension = fn.substring(index+1).toLowerCase();
        if(extension.equals(ext))return true;
    }
    return false;
}
public String getDescription(){
    if(ext.equals("java")){return "JAVA Source File(*.java)";}
```

```
    if(ext.equals("txt")){return "Txt File(*.txt)";}
    return "";
}
}
```

习题

1．指出字节流与字符流的区别。

2．设计一个界面，有一个文本区和一个按钮。在文本区中输入数据，点击按钮后，将文本区中的内容输出　到文件中。要求文件通过文件保存对话框指定。

3．设计一个应用程序，求整数文件各行整数的和。设整数文件中全是整数，整数序列被分成行，各行整数个数不等。要求程序将文件中各行　整数之和输出在另一个文件中。

4．设计一个应用程序，原始数据从程序界面的一个文本区输入。用户点击按钮后，在另一个文本区上输出排序后的数据，并将排序后数据输出到文件中。

5．设计一个应用程序，原始数据从程序界面的一个文本区输入。用户点击按钮后，将文本区上的数据输出到文件中。然后，由用户指定数据的序号，程序从文件读出对应序号的数据输出在文本框中。

6．设计一个界面，有一个文本区和一个按钮。点击按钮后，将文件中的内容输入，显示在文本区。要求文件通过对话框指定。

第 10 章　网络与数据库编程基础

本章主要内容(一般掌握%3~5%)

Java 网络编程基础

Java 数据库编程基础

本章重点：支持 Java 程序访问网上资源的类及其方法，网络编程应用。支持 Java 程序与数据库连接、数据表查询、数据记录修改和插入的类及其方法，数据库编程应用。

本章难点：网络套接字、客户机端与服务器端实现通信的应用程序。支持 Java 实现数据库编程的类，程序实现数据修改和插入的方法，数据库编程应用。

10.1　Java 网络编程基础

Java 语言的优势之一是 Java 程序能访问网络资源。Java 提供一系列的类支持 Java 程序访问网络资源。

10.1.1 IP 地址和 InetAddress 类

1.　TCP/IP 协议和 IP 地址

为了进行网络通信,通信双方必须遵守通信协议.目前最广泛使用的是TCP/IP 协议,它是Internet 中各方所遵循的公共协议.TCP(Transport Control Protocol)是一种传输控制协议,IP(Internet Protocol)是一种网际协议，TCP/IP 代表这两个协议的。

TCP/IP 分为四个层次：**网络接口层**，负责接收和发送物理帧；**网络层**，负责相邻节点之间的通信；**传输层**，负责起点到终点的通信；**应用层**，提供诸如文件传输、电子邮件等应用程序。

TCP 协议将任何网络信息传输当作信息流。例如，机器 A 上的一个长报文发送到机器 B，发送端 A 需要将数据分片，把一片片数据分别打包发送。数据包有一个头，指明该数据包发往何处、凶中数据在接收序列中所处的位置。每个包都按照 I P地址提供的目的地从一个台机器传送到另一台机器，或从一个网络节点传送到另一个网络节点。在接收端 B，这些数据包都能够按照正确的顺序重新组装起来。

TCP/IP 协议是一个协议族，由一组协议组成，主要包含以下更具体的协议：

Telnet，远程登录，允许一台计算机用户登录到另一台远程计算机上，使远程操作如同在本地计算机上操作一样。

FTP（File Transfer protocol）,文件传输协议，允许用户将远程主机上的文件复制到自己的计算机上。

SMTP(simple Mail Transfer Protocol)，简单邮件传输协议，用于传输电子邮件。

NFS(Network file Server),网络文件服务器，使多台计算机透明地访问彼此的目录。

HTTP 是一种超文本传输协议，它是基于 TCP/IP 协议的，是W W W浏览器和服务器之间应用层的通信协议。HTTP 是一种通用、无状态、面向对象的协议。HTTP 会话（事务）包括四个步骤：连接（Connection）、请求（Request）、应答（Response）和关闭（Close）。

Java 语言可编写低层的网络应用。例如，传输文件，建立邮件控制器，处理网络数据等。Java 语言支持的 Internet 协议有 ftp、telnet、www 等，支持网络通信的软件都在 java.net 包中，例如，java.net.ftp,java.net.www 等。

I P地址用于指明因特网上的一台计算机在网络中的地址，用 32 位二进制代码表示一个网络地址。地址分A 、 B 、 C 、 D 、 E五类，常用的是A 、 B 、 C三类：

A （ 1 .0.0.0-126.255.255.255）,0,7 位网络号，后 2 4 位为主机号。

B （ 1 2 8 .0.0.0-191.255.255.255）,10,14 位网络号,后 16 位为主机号

C(192.0.0.0-223.255.255.255),110,21 位网络号,后 8 位为主机号

D(224.0.0.0-239.255.255.255),1110,28 位多点广播组标号.

E(240.0.0.0-254.255.255.255),1111,保留试验使用。

通常，IP 地址用四段十进制数表示（ 8 位一段）。例如：

202.120.224.5

或用文字域名表示。例如：

www.fudan.edu.cn

在因特网上，域名服务器（Domain Name Server,DNS）执行文字名称到二进制网络地址的映射。

2．InetAddress 类

Java.net 包中有 InetAddress 类的定义，InetAddress 类的对象用于 IP 地址和域名，该类提供以下方法：

（1） getByName(String s),获得一个InetAddress 类的对象，该对象中含有主机的 I P地址和域名，该对象用如下格式表示它包含的信息：

www.sina.com.cn/202.108.37.40

（2） String getHostName(),获取 InetAddress 对象的域名。

（3） String getHostAddress(),获取 InetAddress 对象的 IP 地址。

（4） getLocalHost(),获得一个 InetAddress 对象，该对象含有本地机的域名和 IP 地址。

[例 10.1] 说明Inetaddress类的用法的应用程序。程序演示如何获取www.fudan.edu.cn的域名和IP地址。运行结果为：

www.fudan.edu.cn

202.120.224.5

```
Import java.net.*;
Class Example10_1{
    Public static void main(String args[]){
        Try{ //以下代码通过域名建立 InetAddress 对象：
            InetAddress addr = InetAddress.getByname(“www.fudan.edu.cn”);
            String domainName = addr.getHostName();//获得主机名
            String IPName = addr.getHostAddress();//获得 IP 地址
            System.out.println(domainName);
            System.out.println(IPName);
       }catch(UnknownHostException e){
         e.printStackTrace();
       }
    }
}
```

10.1.2 统一资源定位符 URL

统一资源定位符 URL(Uniform Resource Locator)是WWW 客户机访问 Internet 时用来标识资源的名字和地址。超文本链路由统一资源定位符 URL 维持。URL 的格式是：

<METHOD>://<HOSTNAME:PORT>/<PATH>/<FILE>

其中：Method 是传输协议：HOSTNAME 是文档和服务器所在的 Internet 主机名（域名系统中 DNS 中的点地址）;PORT 是服务端口号（可省略）；PATH 是路径名，FILE 是文件名。

例如：

http://www.sun.com/(http是协议名，www.sun.com是主机名)

http://home.netscape.com/home/welcome.html (home.netscape.com是主机名，home/welcome.html是文件路径和文件名)

1. URL 类

Java.net 包有 URL 类，一个 URL 对象可以表示一个网络资源。程序利用 URL 对象能实现 Internet 寻址、网络资源的定位连接、在客户机与服务器之间直接访问等。URL 类的构造方法是

URL(String s)

其中，s 指出网络中的一个资源。

利用 URL 对象访问网上资源的方法是：先创建 URL 对象，如以下代码所示：

```
URL myURL;
try { myURL = new URL(“http://www.fudan.edu.cn:80/”);
}catch(MalformedURLException e)
{System.out.println(“有错的URL:” +url+e);}
```

因创建 URL 对象可能会产生 MalformedURLException 异常。所以，创建 URL 对象的代码应出现在 try…catch 语句块中，以便能捕捉网址错误异常。

2．URLConnection 类

要接收和发关信息还要用 URLConnection 类，程序获得一个 URLConnection 对象，相当于完成对指定 URL 的一个 HTTP 连接。以下是示意获得 URLConnection 对象的代码。

```
URL mu = new URL(“http://www.sun.com/”);//先要创建一个URL对象
URLConnection muC = mu.openConnection();//获得 URLConnection 对象
```

上述代码说明，先要创建一个 URL 对象，然后利用 URL 对象的 openConnection()方法，从系统获得一个 URLConnection

对象。程序有了 URLConnection 对象后，**就可使用 URLConnection 类提供的以下方法获得流对象和实现网络连接：**

（1） getOutputStream()，获得向远程主机发送信息的 OutputStream 流对象。

（2） getInputStream()，获得从远程主机获取信息的 InputStream 流对象。有了网络连接的输入和输出流，程序就可实现远程通信。

（3） connect()，设置网络连接。

3．信息的发送和接收

发送和接收信息要获得流对象，并由流对象创建输入或输出数据流对象。然后，就可以用流的方法访问网上资源。

参见例 10.2 程序中的方法 readByURL()，该方法说明已知网址读取网页内容的过程。方法利用网址参数创建一个 URL 对象 url，接着利用对象 url 的 openConnect()方法，获得 URLConnection 对象 tc，用对象 tc 的 connect()方法建立网络连接，接着获得网络连接的 InputStreamReader 类对象 in，将对象 in，转化成为 BufferedRead 对象 dis，改为缓冲式输入。最后，用对象 dis 的 readLine()方法完成读取网络文本数据。

如同本地数据流一样，网上资源使用结束后，数据流也应及时关闭。例如，代码

```
dis.close();
```

关闭先前代码建立的流 dis。

[例 10.2]以数据流方法读取网页内容的应用程序。程序运行时，网址从文本框中读取。

```
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.javax.swing.*;
public class Example10_2{
    public static void main(String args[]){
        new downNetFile();
    }
}
class DownNetFile extends JFrame implements ActionListener{
        JTextFileld infield = new JTextField(30);
        JTextarea showArea = new JTextArea();
        JButton b = new JButton(“download”);JPanel p = new JPanel();
        DownNetFile(){
            super(“read network text file application”);
        Container con = this.getContentPane();
            p.add(infield);p.add(b);
        JScrollPane jsp = new JScrollPane(showArea);
        b.addActionListener(this);
        con.add(p,” North”);con.add(jsp,” Center”);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(500,400);setVisible(true);
        }
        public void actionPerformed(ActionEvent e){
            readByURL(infield.getText());
        }
        public void readByURL(String urlName){
            try{
                URL url = new URL(urlName);//由网址创建 URL 对象
                URLConnection tc = url.openConnectin();//获得 URLConnection 对象
                tc.connect();//设置网络连接
                InptStreamReader in = new InputStreamReader(tc.getInputStream());
                BufferedReader dis = new BufferedReader(in);//采用缓冲式输入
```

```
            String inline;
            while((inline = dis.readLine())!=null){
                showArea.append(inline +" \n" );
            }
            dis.close();//网上资源使用结束后，数据流及时关闭
        } catch(MalformedURLException e){e.printStackTrace();}
        catch(IOException e){e.printStacktrace();}
        /*访问网上资源可能产生 MalformedURLException 和 IOException 异常*/
        }
    }
```

10.1.3 客户机/服务器模式

网络应用模式主要有：**主机/终端模式**，集中计算，集中管理；**客户机/服务器**（Client/Server,简称C/S）模式，分布计算，分布管理；**浏览器/服务器模式**，利用 Internet 跨平台。

WWW（万维网）就是建立在**客户机/服务器**模式上，以 HTML 语言和 HTTP 协议为基础，能够提供各种 Internet 服务的信息浏览系统。网络信息放在主机的不同位置，WWW服务器利用超文本链路链接各项信息。WWW客户机（浏览器 Brower）负责与服务器建立联系，向服务器发送请求，处理 HTML 超媒体，提供图形用户界面（GUI），显示信息等。

在客户机/服务器工作模式中，在 Server 端，要准备接受多个 Client 端计算机的通信。为此，除用 IP 地址标识 Internet 上的计算机之外，另还引入端口号，用端口号标识正在 Server 端后台服务的线程。端口号与IP地址的组合称为网络套接字（socket）。

Java 语言在实现 C/S 模式中，套接字分为两类：在 **Server 端**，ServerSocket 类支持底层的网络通信；在 **Client** 端，Socket 类支持网络的底层通信。Server 机通过端口（总线 I/O 地址）提供面向 Client 机的服务；Server 机在它的几个不同端口分别同时提供几种不同的服务。Client 接入 Server 的某一端口，通过这个端口提请 Server 机为其服务。规定：端口号 0 ~1023 供系统专用。例如，HTTP 协议在端口 80，telnet 协议在端口 23。端口 1024~65535 供应用程序使用。

当 Client 程序和 Server 程序需要通信时，可以用 Socket 类建立套接字连接。套接字连接可想象为一个电话呼叫：最初是 Client 程序建立呼叫，Server 程序监听；呼叫完成后，任何一方都可以随时讲话。

双方实现通信有**流式 socket** 和**数据报式 socket** 两种可选方式。

流式 socket **是有连接的通信**，即 TCP(Transmission Control Protocol)。每次通信前建立连接，通信结束后断开连接。特点是可以保证传输的正确性、可靠性。

数据报式 socket **是无连接的通信**，即 UDP(User Datagram Protocol)。将欲传输的数据分成 小包，直接上网发送。无需建立连接和拆除连接，速度快，但无可靠保证。

流式 socket 在 Client 程序和 Server 程序间建立通信的通道。每个 socket 可以进行读和写两种操作。对于任一端，与对方的通信会话过程是：

建立 socket 连接，获得输入/输出流，读数据/写数据，通信完成后关闭 socket（拆除连接）。流式 Socket 的通信过程见 10.1。

利用 socket 的构造方法，可以在客户端建立到服务器的套接字对象：

Socket(String host,int port),host 是服务器的 IP 地址，port 是端口号，这些是预先约定的。例如，代码：

```
try{
    Socket mySocket = new Socket( "http://fudan.edu.cn" ,1860);
}catch(IOException e){}
```

然后,用 getInputStream()方法获得输入流,用这个输入流读取服务器放入"线路"的信息;用 getOutputStream()方法获得输出流，用这个输出流将信息写入 "线路"。

利用 ServerSocket 的构造方法可以在服务器建立接受客户套接字的服务器套接字对象：

ServerSocket(int port),指定端口号,创建一个 ServerSocket 对象。端口号 port 要与客户呼叫的端口号相同。为此，用以下形式代码；

```
try{ServerSocket serverSocket = new ServerSocket(1860);
}catch(IOException e){}
```

服务器端程序在指定的端口监听，当收到 Client 程序发出的服务请求时，创建一个套接字对象与该端口对应的 Client 程序通信。例如,执行上述建立服务器套接字对象的代码,确立了对象 serverSocket 后,就可能它使用 accept()

方法，得到Socket对象，接收Client程序来自套接字mySocket的信息。如以下代码所示：

```
try{Socket sc = serverSocket.accept();//ac是一个Socket对象
}catch(IOException e){}
```

要撤销服务，可以关闭Socket对象sc；

```
sc.close();
```

[例10.3]C/S模式中的Client端应用程序。这是一个Client端的流式Socket通信的简单实例，代码说明Client端程序的编写方法。例中，Client程序向服务器主机的端口4441提出请求，连接建立后完成对服务器的读写。

```
import java.io.*;
import java.net.*;
public class Client{
    public static void main(String args[]){
        String s = null;Socket mySocket;
        DataInputStream in = null;DataOutputStream out = null;
        try{
            mySocket = new Socket(“localhost”,4441);
            in = new DataInputStream(mySocket.getInputStream());
            out = new DataOutputStream(mySocket.getOutputStream());
            out.writeUTF(“good server!”);
            while(true)
            {
                s = in.readUTF();
                if(s==null)break;
                else System.out.println(s);
            }
            mySocket.close();
        }catch(IOException e){System.out.println(“can’t connect”);
    }
}
```

[例10.4]与例10.3 Client端应用程序对应的Server端应用程序。程序在4441端口监听，当检测到有客户机请求时，产生一个内为“客户，你好，我是服务器”的字符串输出到客户端。

```
import java.io.*;import java.net.*;
public class Server{
    public static void main(String args[]){
        ServerSocket server = null;
        Socket you = null;String s = null;
        DataOutputStream out = null;
        DataInputStream in = null;
        try{
            server = new ServerSocket(4441);
        }catch(IOException e1){
                system.out.println(“ERROR:” +e1);
        }
        try{
            you = server.accept();
            in = new DataInputStream(you.getInputStream());
            out = new DataOutputStream(you. getOutputStream());
            while(true){
                s = in.readUTF();
                if(s!=null) break;
```

```
                }
                out.writeUTF(“客户，你好，我是服务器”);
                out.close();
                }
                catch(IOException e){System.out.println(“ERROR:”+e);}
            }
    }
```

为了充分发挥计算机的平行工作能力，可以把套接字连接工作让一个线程完成。当客户端要请求服务器给予服务，或当服务器端接收到一个客户的服务请求，就启动一个专门完成信息通信的线程，在该线程中创建输入输出流，并完成客户端与服务器端的信息交流。

[例 10.5] 将套接字连接工作置于线程的客户端小应用程序。界面在有一个发送信息按纽、一个文本框和一个文本区。客户端应用程序首先与服务器建立套接字连接。使用数据输入流 in 反复读取服务器放入线路里的信息，将收到的信息在文本区中显示。娶志取的信息是“结束”，则关闭套接字连接，并结束程序。用户也可在文本框输入信息，并按发送信息按钮，则客户端程序利用数据输出流 out,将文本框中的内容发送给服务器。

```
import java.net.*;
import java.io.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.applet.*;
public class Aclient extends Applet implements Runnable,ActionListener{
        JButton button; JTextField textF; JTextArea textA;
        Socket socket;  Thread thread;
        DataInputStream in; DataOutputStream out;
        public void init(){
            setBackground(new Color(120,153,137));
            setLayout(new BorderLayout());
            Button = new JButton(“发送信息”);
            textF = new JTextField(20);
            textA = new JTextArea(20,30);
            setSize(450,350);
            JPanel p = new JPanel();
            p.add(textF);  p.add(button);
            add(textA,”Center”); add(p,”South”);
            button.addActionListener(this);
        }
        public void start(){
            try{
                socket = new Socket(this.getCodeBase().getHost(),4441);
                in = new DataInputStream(socket.getInputStream());
                out = new DataOutputStream(socket.getOutputStream());
                }catch(IOException e){}
                    if(thread==null){
                        thread = new Thread(this);
                        thread.setPriority(Thread.MIN_PRIORITY);
                        thread.start();
                        }
        }
        public void run(){
```

```
            String s = null;
            while(true){
                try{
                    s = in.readUTF();
                }catch(IOException e){}
                if(s.equals(“结束”))
                {
                    try{socket.close();break;
                    }catch(IOException e){}
                }else texA.append(s + “\n”);
            }
        }
        public void actionPerformed(ActionEvent e){
            if(e.getSource()==button){
                String s = textF.getText();
                if(s! = null)
                {
                    try{
                        out.writeUTF(s);
                    }catch(IOException e1){}
                }
                else{
                    try{ out.writeUTF(“请说话”);
                        }catch(IOException e1){}
                }
            }
        }
    }
```

[例 10.6]对应例 10.5 客户端小应用程序的服务器端小应用程序。程序以端 4441 建立与客户端的套接字连接，服务器端收到客户端的申请后，以客户的套接字建立一个线程，并启动。如果没有客户申请，则继续监听客户的申请。线程按客户的套接字建立输入数据流 in 和输数据流 out。线程利用 in 读取客户放入线路里的信息。如果接受的信息是“结束”，则服务器回复“结束”后关闭套接字连接；否则回复：“我是服务器你对我说“，以及服务器接收到的信息。

```
    import java.net.*;
    import java.io.*;
    import java.awt.*;
    import javax.swing.*;
    import java.awt.event.*;
    import java.applet.*;
    public class Aclient extends Applet implements Runnable,ActionListener{
        JButton button; JTextField textF; JTextArea textA;
        Socket socket;  Thread thread;
        DataInputStream in; DataOutputStream out;
        public void init(){
            setBackground(new Color(120,153,137));
            setLayout(new BorderLayout());
            Button = new JButton(“发送信息”);
            textF = new JTextField(20);
            textA = new JTextArea(20,30);
```

```
        setSize(450,350);
        JPanel p = new JPanel();
        p.add(textF);  p.add(button);
        add(textA,"Center"); add(p,"South");
        button.addActionListener(this);
    }
    public void start(){
        try{
            socket = new Socket(this.getCodeBase().getHost(),4441);
            in = new DataInputStream(socket.getInputStream());
            out = new DataOutputStream(socket.getOutputStream());
            }catch(IOException e){}
                if(thread==null){
                    thread = new Thread(this);
                    thread.setPriority(Thread.MIN_PRIORITY);
                    thread.start();
                }
    }
    public void run(){
        String s = null;
        while(true){
            try{
                s = in.readUTF();
            }catch(IOException e){}
            if(s.equals("结束"))
            {
                try{socket.close();break;
                }catch(IOException e){}
            }else texA.append(s + "\n");
        }
    }
    public void actionPerformed(ActionEvent e){
        if(e.getSource()==button){
            String s = textF.getText();
            if(s! = null)
            {
                try{
                    out.writeUTF(s);
                }catch(IOException e1){}
            }
            else{
                try{ out.writeUTF("请说话");
                    }catch(IOException e1){}
            }
        }
    }
}
```

10.2 Java 数据库编程基础

一个网络关系数据库应用系统是一个三层次结构。客户机与服务器采用网络连接，客户机端应用程序按通信协

议与服务器端的数据库程序通信；数据库服务程序通过 SQL 命令与数据库管理系统通信。

10.2.1 Java 程序与数据库连接

Java 程序与数据库连接方法有两种。一种是使用 JDBC-ODBC 桥接器与数据库连接，一种是用纯 Java 的 JDBC 驱动程序实现与数据库连接。

1. 使用 JDBC-ODBC 桥接器与数据库连接

Java 程序使用 JDBC-ODBC 桥接器与数据库连接，Java 程序与数据库通信的过程是：

先由数据库应用程序向 ODBC 驱动管理器发出 API 调用，ODBC 驱动管理器将这个调用转换成向数据库管理系统的 ODBC 驱动程序调用，数据库管理系统又将这个调用转换成对操作系统的数据输入/输出调用。最后，操作系统从数据库中得到实际数据逐级返回。

数据库编程首先要设置数据源，在 ODBC 中设置数据源的步骤如下：

(1) 打开 Windows 控制面板中的管理工具。对于 windows XP:选择“性能维护”→“管理工具”→“数据源(ODBC)”；对于 windows 2000：选择“管理工具”→“数据源”。

(2) 打开“数据源”。出现 ODBC 数据源管理器对话框，显示现有的数据源名称。

(3) 选择“用户 DSN”，单击“添加”按钮，出现安装数据源驱动程序对话框。Access(*.mdb)数据源，单击“完成”按钮，出现“创建数据源对话框，键入需要创建的数据源名，并为创建的数据源选择一个数据库表。

(4) 单击数据库区域的“选择”按钮，选择需要的数据库表。当需要为数据源授权访问级别时，单击“高级”按钮。设置登录名和密码后，单击“确定”按钮，完成 Access 数据库在 ODBC 管理器中的配置。

(5) 如果还没有数据库表，则需创建一个数据库表。

数据源就是数据库，在设定了数据源的基础上，Java 程序要访问数据库表，还要建立 JDBC-ODBC 桥接器，让程序与数据库连接。以后，程序就可向数据库发送 SQL 语句，处理数据库返回的结果。Java 数据库连接 JDBC(Java DataBase Connectivity)由一组用 Java 语言编写的类和接口组成，JDBC 是 Java 程序与数据库连接 API。它能做以下三件事情：

与某个数据库建立连接、向数据库发送 SQL 语句和处理数据库返回的结果。

调用类方法 Class.forName(String s)能建立 JDBC-ODBC 桥接器。例如，代码：

```
try{Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);
}catch(Exception e){}
```

为 Java 程序加载了驱动程序。

[例 10.7]说明与数据库连接的方法 connectByJdbcOdbc()，该方法按给定的数据库 URL、用户名和密码连接数据库，如果连接成功，方法返回连接对象，连接不成功，则返回空。

```
public static connection connectByjdbcOdbc(String url, String username, String
    password)
{
    Connection con = null;
    try
    {
        Class.forName( //加载 ODBC 驱动程序
        "sun.jdbc.odbc.JdbcOdbcDriver");
    }
    catch (Exception e)
    {
        e.printStackTrace();
        return null; //连接失败
    }
    try
    {
        con = DriverManager.getConnection(url, username, password);
    }
    catch (SQLExceotuib e)
```

```
        {
            e.printStackTrace();
            return null; //连接不成功
        }
        return con; //连接成功
    }
```

以下代码是对 connectByJdbcOdbc()方法的一个调用，数据库连接成功，弹出数据库连接成功信息窗，否则弹出数据库连接不成功信息窗。

```
if ((con = connectByJdbcOdbc("jdbc:odbc:redsun", "xia", "1234")) != null)
  {
    JoptionPane.showMessageDialog(null, "数据库连接成功");
    try
    {
        con.close();
        con = null;
    }
    catch (SOLException e){}
}
else
    JOptionPane.showMessageDialog(null, "数据库连接失败");
```

用纯 Java 的 JDBC 驱动程序实现与数据库连接

Java程序也可以用纯Java的JDBC驱动程序实现与数据库连接。这种方法应用较广泛，但是需要下载相应的驱动程序包，因为不同的数据库的连接代码可能不同，连接不同的数据库，加载的驱动程序也可能不同。例如，连接SQLServer的驱动程序在www.msdn.com网站下载，有3个包：msbase.jar,mssqlserver.jar和msutil.jar，并要求将这 3 个包放在jdk\jre\lib\ext\目录下，或在CLASSPATH中设置其放置位置。

使用纯 Java 的 JDBC 驱动程序实现与数据库连接的过程如下：

(1) 加载驱动程序

有两种加载驱动程序的方式：一各是将驱动程序添加到 java.lang.System 的属性 jdbc.drivers 中。这是一个 DriverManager 类加载驱动程序类名的列表，表元用冒号分隔。另一种方式是从相关的网站下载驱动程序后，在程序中利用 Class.forName()方法加载指定的驱动程序。例如，

```
Class.forName(“com.microsoft.jdbc.sqlserver.SQLServerDriver”);
```

(2) 创建指定数据库的 URL

数据库的 URL 对象类似网络的统一资源定位符，其格式是：

jdbc:subProtocol:subName://hostname:port:Databasename=XXX

其中，subprotocol 是某种驱动程序支持的数据库连接机制； subName 是当前连接机制下的具体名称；hostName 是主机名；port 是相应的连接端口；DatabaseName 是要连接的数据库名称。例如，以下代码可以是一个数据库的 URL：

jdbc:Microsoft:sqlserver://localhost:1433;Databasename=ksinfo

该数据库的 URL 说明利用 miscrosoft 提供的机制，用 sqlserve 驱动，通过 1433 端口访问本机上的 ksInfo 数据库。

(3) 建立连接

驱动程序管理器(DriverManager)的方法 getConnection()建立连接。参见 10.2.2 关于 DriverManager 类的介绍。

[例 10.8]说明与数据库连接的静态方法 connectByJdbc()，该方法按给定的数据库 URL、用户名和密码连接数据库，如果连接成功，方法返回 true，连接不成功，则返回 false。

```
public static Connection conectByJdbc(String url, String username, String
    password)
{
```

```
    Connection con = null;
    try
    {
        Class.forName( //加载特定的驱动程序
        "com.microsoft.jdbc.sqlserver.SQLServerDriver");
    }
    catch (Exception e)
    {
        e.printStackTrace();
        return null; //连接失败
    }
    try
    {
        con = DriverManage.getConnection(url, username, password);
    }
    catch (SQLException e)
    {
        e.printStackTrace();
        return null; //连接失败
    }
    return con; //连接成功
}
```

10.2.2 几个重要的类和接口

编写访问数据库的 Java 程序还需要几个重要的类和接口。

1. DriverManager 类

DriverManager 类处理驱动程序的加载和建立新数据库连接。DriverManager 是 java.sql 包中用于管理数据库驱动程序的类。通常，应用程序只使用类 DriverManager 的 getConnection()静态方法，用来建立与数据库的连接，返回 Connection 对象：

static Connection getConnection(String url,String username,String password)

指定数据的 URL 用户名和密码创建数据库连接对象。url 的语法格式是：

jdbc:<数据库的连接机制>:<ODBC 数据库名>。

2. Connection 类

Connection 类是 java.sql 包中用于处理与特定数据库连接的类。Connection 对象是用来表示数据库连接的对象，Java 程序对数据库的操作都在这种对象上进行。Connection 类的主要方法有：

Statement createStatement(),创建一个 Statement 对象。

Statement createStatement(int resultSetType,int resultSetConcurrency),创建一个 Statement 对象，生成具有特定类型的结果集。

void commit()，提交对数据库的改动并释放当前持有的数据库的锁。

void rollback()，回滚当前事务中的所有改动并释放当前连接持有的数据库的锁。

String getCatalog(),获得连接对象的当前目录。

boolean isClose()，判断连接是否已关闭。

boolean isReadOnly()，判断连接是否为只读模式。

void setReadOnly(),设置连接为只读模式。

void close(),释放连接对象的数据库和 JDBC 资源。

3. Statement 类

Statement 类是 java.sql 包中用于在指定的连接中处理 SQL 语句的类。数据库编程的要点是在程序中嵌入 SQL 命令。程序需要声明和创建连接数据库的 Connection 对象，并让该对象连接数据库。调用类 DriverManager 的静态方法 getConnection()获得 Connection 对象，实现程序与数据库的连。然后，用 Statement 类声明 SQL 语句对象，并调用 Connection 对象的 createStatement()方法，创建 SQL 语句对象。例如，以下代码创建语句

对象 sql:

```
    Statement sql = null;
try{ sql = con.createStatement();
}catch(SQLException e){}
```

4. ResultSet 类

有了 SQL 语句对象后，调用语句对象的方法 executeQuery()执行 SQL 查询，并将查询结果存放在一个用 ResultSet 类声明的对象中，例如，以下代码读取学生成绩表存于 rs 对象中：

```
ResultSet rs = sql.executeQuery(“SELECT  * FROM ksInfo”);
```

ResultSet 对象实际上是一个由查询结果数据的表，是一个管式数据集，由统一形式的数据行组成，一行对应一条查询记录。在 ResultSet 对象中隐含着一个游标，一次只能获得游标当前所指的数据行，用 next 方法可取下一个数据行。用数据行的字段(列)名称或位置索引(自 1 开始)调用形如 getXXX()方法获得记录的字段植 。

以下是 ResultSet 对象的部分方法：

(1) byte getByte(int columnIndex),返回指定字段的字节值。

(2) Date getDate(int columnIndex),返回指定字段的日期值。

(3) float getFloat(int columnIndex),返回指定字段的浮点值。

(4) int getInt(int columnIndex),返回指定字段的整数值。

(5) String getString(int columnIndex),返回指定字段的字符串值。

(6) double getDouble(String columnName),返回指定字段的双精度值。

(7) long getLong(String columnName),返回指定字段的 long 型整值。

(8) boolean next(),返回是否还有下一字段。

以上方法中的 columnIndex 是位置索引，用于指定字段，columnName 是字段名。

用户需要在查询结果集上浏览，或前后移动、或显示结果集的指定记录，这称为可滚动结果集。程序要获得一个可滚动结果集，只要在获得 SQL 的语句对象时，增加指定结果集的两个参数即可。例如，以下代码：

```
    Statement stmt = con.createStatement(type,concurrency);
    ResultSet rs = stmt.executeQuery(SQL 语句)
```

语句对象 stmt 的 SQL 查询就能得到相应类型的结果集。

int 型参数 type 决定可滚动集的滚动方式：

ResultSet.TYPE_FORWORD_ONLY,结果集的游标只能向下滚动。

ResultSet.TYPE_SCROLL_INSENSITIVE,游标可上下移动，当数据库变化时，当前结果集不变。

ResultSet. TYPE_SCROLL_SENSITIVE，游标可上下移动，当数据库变化时，当前结果集同步改变。

int 型参数 concurrency 决定数据库是否与可滚动集同步更新？

ResultSet.CONCUR_READ_ONLY,不能用结果集更新数据库中的表。

ResultSet.CONCUR_UPDATETABLE,能用结果集更新数据库中的表。

例如，以下代码利用连接对象 connect，创建 Statement 对象 stmt，指定结果集可滚动，并以只读方式读数据库：

```
stmt = connect.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                        ResultSet.CONCUR_READ_ONLY);
```

可滚动集上另外一些常用的方法如下：

boolean previous(),将游标向上移动，当移到结果集的第一行时，返回 false。

void beforeFirst()，将游标移结果集的第一行之前。

void afterLast(),将游标移到结果集的最后一行之后。

void first()，将游标移到第一行。

void last(),将游标移到最后一行。

boolean isAfterLast(),判游标是否在最后一行之后。

boolean isBeforeFirst(),判游标是否在第一行之前。

boolean isLast(),判游标是否在最后一行。

boolean isFirst(),判游标是否在第一行。

int getRow(),获取当前所指的行(行号自 1 开始编号，结果集空，返回 0)。

boolean absolute(int row),将游标移到 row 行。

10.2.3 数据库查询

利用 Connection 对象的 createStatement 方法建立 Statement 对象，利用 Statement 对象的 executeQuery()方法执行 SQL 查询语句进行查询，返回结果集，再形如 getXXX()的方法从结果集中读取数据。经过这样的一系列步骤就能实现对数据库的查询。

[例 10.9]Java 应用程序访问数据库。应用程序打开考生信息表 ksInfo，从中取出考生的各项信息。设考生信息数据库的结构如下：

类型	字符串	字符串	整数	字符串	字符串
字段名	考号	姓名	成绩	地址	简历

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;
import java.sql.*;
public class Example10_9 extends JFrame implements ActionListener
{
    public static Connection connectByJdbcodbc(String url, String username,
        String password)
    {
        Connection con = null;
        try
        {
            Class.forName( //加载 ODBC 驱动程序
            "sun.jdbc.odbc.JdbcOdbcDriver");
        }
        catch (Exception e)
        {
            e.printStackTrace();
            return null; //加载失败，连接不成功
        }
        try
        {
            con = DriverManager.getConnection(url, username, password);
        }
        catch (SQLException e)
        {
            e.printStackTrace();
            return null; //连接失败
        }
        return con; //连接成功
    }
    String title[] =
    {
        "考号", "姓名", "成绩", "地址", "简历"
    };
    JTextField txtNo = new JTextField(8);
    JTextField txtName = new JTextField(10);
    JTextField txtScore = new JTextField(3);
    JTextField txtAddr = new JTextField(30);
```

```
JTextArea txtresume = new JTextArea();
JButton prev = new JButton("前一个");
JButton next = new JButton("后一个");
JButton first = new JButton("第一个");
JButton last = new JButton("最后一个");
Statement sql; //SQL 语句对象
ResultSet rs; //存放查询结果对象
Example10_9(Connection connect)
{
    super("考生信息查看窗口");
    setSize(450, 350);
    try
    {
        sql = connect.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_READ_ONLY);
        rs = sql.executeQuery("SELECT * FROM ksInfo");
        Container con = getContentPane();
        con.setLayout(new BorderLayout(0, 6); JPanel p[] = new JPanel[4];
        for (int i = 0; i < 4; i++)
        {
            p[i] = new JPane(new FlowLayout(FlowLayout.LEFT, 8, 0));
                p[i].add(new JLabel(title[i]));
        }
        p[0].add(txtNo);
         p[1].add(txtName);
         p[2].add(txtScore);
         p[3].add(txtAddr);
         JPanel p1 = new JPane(new GridLayout94, 1, 0, 8));
        JScrollPane jsp = new JScrollPane(txtResume,
            JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
            JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
        jsp.setPreforredSize(new Dimension(300, 60);
        for (int i = 0; i < 4; i++)
        {
            p1.add(p[i]);
        }
        JPanel p2 = new JPanel(new FlowLayout(FlowLayout.LEFT, 10, 0);
        p2.add(new JLabel(title[4]));
        p2.add(jsp);
        Jpanel p3 = new Jpanel();
        p3.add(prev);
        p3.add(next);
        p3.add(first);
        p3.add(last);
        prev.addActionListener(this);
        next.addActionListener(this);
        first.addActionListener(this);
        last.addActionlistener(this);
        rs.first();
```

```
        readRecord();
    }
    catch (Exception e)
    {
        e.printStackTrace():
    }
    setVisible(ture);
}
public void modifyRecord(Connection connect)
{
    String stuNo = (String)JOptionPane.showInputDialog(null,
        "请输入考生考号", "输入考号对话框", JOptionPane.PLAIN_MESSAGE, null,
        null, "");
     try {
        sql = connect.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_READ_ONLY); rs = sql.executeQuery(
            "SELECT * FROM ksInfo"); Container con = getContentPane();
            con.setLayout(new Boarderlayout(0, 6));
        Jpanel p[] = new JPanel[4];
        for (int i = 0; i < ; i++)
        {
            p[i] = new JPane(new FlowLayout(flowLayout.LEFT, 8, 0));
            p[i].add(new JLabel(title[i]));
        }
        p[0].add(txtNo);
        p[1].add(txtName);
        p[2].add(txtScore);
        p[3].add(txtAddr);
        Jpanel p1 = new Jpane(new GridLayout(4, 1, 0, 8));
        JScrollPane jsp = new JScrollPane(txtResume,
        JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
        JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
        jsp.setPreferredSize (new dimension(300, 60));
        for (int i = 0; i < 4; i++)
        {
            p1.add(p[i]);
        }
        Jpanel p2 = new JPanel(new FlowLayout(FlowLayout.LEFT, 10, 0));
        p2.add(new JLableI(title[4]));
        p2.add(jsp);
        JPanel p3 = new JPanel();
        p3.add(prev);
        p3.add(next);
        p3.add(first);
        p3.add(last);
        prev.addActionListener(this);
        next.addActionListener(this);
        first.addActionListenerIthis);
        last.addActionListener(this);
```

```
                rs.first();
                readRecord();
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
            setVisible(true);
        }
        boolean readRecord()
        {
            try
            {
                txtNo.setText(rs.getString("考号")); txtName.setText(rs.getString(
                    "姓名")); txtScore.setText(rs.getString("成绩"));
                    txtAddr.setText(rs.getString("地址")); txtResume.setText
                    (rs.getString("简历"));
            }
            catch (SQLException e)
            {
                e.printStackTrace(); return false;
            }
            return true;
        }
        public void actionPerformed(ActionEvent e)
        {
            try
            {
                if (e.getSource() == prev)rs.previous();
                else if (e.getSource() == next)rs.next();
                else if (e.getSource() == first)rs.first();
                else if (e.getSource() == last)rs.last(); readRecord();
            }
            catch (Exception e2){}
        }
        public static void main(String args[])
        {
            connection connect = null;
            JFrame .setDefaultLookAndFeeDecorated(true);
            Font font = new Font("JFrame", Font.PLAIN, 14);
            if ((connect =connectByJdbcOdbc("jdbc:odbc:redsun", "xia", "1234")) == null)
            {
                JOptionPane.showMessageDialog(null, "数据库连接失败!");
                System.exit ( - 1);
            }
            new Example10_9(connect);  //创建对象
        }
    }
```

10.2.4 数据库更新

数据库更新操作包括数据表创建、删除、以及数据表记录的增加、删除、修改等操作。如果利用数据 SQL 命令实现，则利用 Statement 对旬的 executeUpdate()方法，执行 SQL 的 update 语句，实现数据表的修改；执行 SQL 的 insert 语句，实现数据表记录的添加。

例如，在前面数据为查询例子基础上，再增加对数据表的修改和插入。限于篇幅，不再给出完整程序，只给出实现修改和插入的方法。程序可再增设插入、,，除保存按钮，通过已有的浏览，定位到数据表的特定位置，对痈疽记录进行编辑修改，或插入，或删除，然后按保存按钮，完成修改后的数据表保存。

下面用代码说明数据表更新的方法。

与数据表连接时，需指定获得的 ResultSet 对象是可更新的。

```
stmt = connect.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                               ResultSet.CONCUR_UPDATABLE);
```

1. 插入数据表记录

插入数据表记录有 3 种方案

(1) 使用 Statement 对象

实现插入数据表记录的 SQL 语句的语法是：

```
insert into  表名(字段名 1，字段名 2,……)value (字段值 1，字段值 2，……)
```

例如：

```
insert into ksInfo(考号，姓名，成绩，地址，简历)value(‘200701’,’张大卫’534，’上海欧阳路 218 弄 4-1202’,’’)
```

实现同样功能的 Java 程序代码是：

```
sql = “insert intoksIno(考号，姓名，成绩，地址，简历)”;
sql= = sql+ “value(‘”+txtNo.getTxt()+’,’”+txtName.getText(0”’,”;
sql = sql+txtScore.getText();
sql=sql+”,’”+txtAddr.getText()+”’,’”+txtResume.getText()+”’)”;
stmt.executeUpdate(sql);
```

(2) 使用 ResultSet 对象

使用 ResultSet 对象的方法 moveToInsertRow()将数据表游标移到插入位置，输入数据后，用方法 insertRow()插入记录。例如，以下示意代码：

```
String sql= “select * from ksInfo”;//生成 SQL 语句
ResultSet rs = stmt.executeQuery(sql);//获取数据表结果集
rs.moveToInsertRow();//将数据表游标移到插入记录位置
rs.updateString(1,’200701’);//向考号字段填入数据
rs.updateString(2,’张大卫’);//向名字字段填入数据
rs.updateInt(3,534);//向成绩字段填入数据
rs.updateString(4,’上海欧阳路 218 弄 4-1202’);//向地址字段填入数据
rs.updateString(5,’’);//向简历字段填入数据
try{rs.insertRow();}catch(Exception e){};//完成插入
```

(3) 使用 PrepareStatement 对象

与使用 Statement 对象的方法类似，只是创建 SQL 语句时暂时用参数？表示值，然后由 SQL 语句对象生成 PrepareStatement 对象，插入时通过设定实际参数，实现记录的更新。示意代码如下：

```
sql = “insert into ksInfo(考号，姓名，成绩，地址，简历)value (?,?,?,?,’’)”;
PrepareStatement pStmt = connect.prepareStatement(sql);
pStmt.setString(1,’200701’);//向考号字段填入数据
pStmt. setString (2,’张大卫’);//向名字字段填入数据
pStmt.setInt(3,534);//向成绩字段填入数据
pStmt. setString (4,’上海欧阳路 218 弄 4-1202’);//向地址字段填入数据
pStmt. setString (5,’’);//向简历字段填入数据
pStmt.executeUpdate();
```

2. 修改数据表记录

修改数据表记录也有 3 种方案

(1) 使用 Statement 对象

实现修改数据表记录的 SQL 语句的语法是：

update 表名 set 字段名 1 = 字段值 1，字段名 2 = 字段值 2，……where 特定条件

例如：

update ksInfo set 姓名 = ‘张小卫’ where 姓名 = ‘张大卫’

先创建一个 SQL 语句，然砌调用 Statement 对象的 executeUpdate() 方法。例如，

```
sql = “update ksInfo set 姓名 = ‘” +txtName.getText();
sql = sql + “,成绩=” +txtScore.getText();
sql = sql +”,地址=’ ” +txtAddr.getText();
sql= sql+” ’ ,,简历=’ ” +txtResume.getText()+” ’ where 考号=” +txtNo.getText();
stmt.executeUpdate(sql);
```

(2) 使用 ResultSet 对象

先建立 ResultSet 对象，然后直接设定记录的字段值，修改数据表的记录。例如，

```
String sql = “select * from ksInfo where 姓名=’张大卫’ ”;//生成 SQL 语句
ResultSet rs = stmt.executeQuery(sql);//获取数据表结果集
if(rs.next()){
  rs.updateString(2,’ 张小卫’ );
  try{rs.updateRow();}catch(Exception e){}
}
```

(3) 使用 PrepareStatement 对象

创建 SQL 语句时，暂时用参数？表示值，然后由 SQL 语句对象生成 PrepareStatement 对象，接着通过设定实际参数实现记录的更新。示意代码：

```
sql = “update ksInfo set 姓名=? where 姓名 = ‘张大卫’ ;
PrepareStatement pStmt = connect.prepareStatement(sql);
pStmt.setString(2,’ 张小卫’ );//向名字字段填入数据
pStmt.executeUpdate();
```

3. 删除数据表记录

删除数据表也有 3 种方案

(1) 使用 Statement 对象

删除数据表记录的 SQL 语句的语法是

```
delete from 表名 where 特定条件
```

例如 ：

```
delete from ksInfo where 姓名 = ‘张大卫’
```

先创建一个 SQL 语句，然后调用 Statement 对象的 executeUpdate() 方法：

```
stmt.executeUpdate(sql);
```

(2) 使用 ResultSet 对象

先创建一个 SQL 语句，然后调用 Statement 对象的 executeUpdate() 方法。例如，

```
String sql = “select * from ksInfo where 姓名 = ‘张大卫’ ”;//生成 SQL 语句
ResultSet rs = stmt.executeQuery(sql);//获取数据表结果集
if(rs.next()){
rs.deleteRow();try{ rs.updateRow();}catch(Exception e){}
}
```

(3) 使用 PrepareStatement 对象

创建 SQL 语句时，暂时用参数？表示值，然后由 SQL 语句对象生成 PrepareStatement 对象，接着设定实际参数实现特定记录的删除。例如，以下示意代码：

```
sql = “delete form ksInfo where 姓名=?”;
PrepareStatement pStmt = connect.prepareStatement(sql);
pStmt.setString(2,’ 张大卫’ );//给名字字段指定数据
```

```
pStmt.executeUpdate();
```

习题

1. 在 java 程序中，用何种对象存储 IP 地址和域名？

答：用 InetAddress 类的对象

2. 用代码示意程序获取域名和 IP 地址的方法。

```
Import java.net.*;
Class Example10_1{
Public static void main(String args[]){
Try{ //以下代码通过域名建立 InetAddress 对象：
InetAddress addr = InetAddress.getByname(www.fudan.edu.cn);
            String domainName = addr.getHostName();//获得主机名
            String IPName = addr.getHostAddress();//获得 IP 地址
        System.out.println(domainName);
        System.out.println(IPName);
      }
    catch(UnknownHostException e){
    e.printStackTrace();
    }
    }
}
```

3. URL 的作用是什么？

一个 URL 对象可以表示一个网络资源。程序利用 URL 对象能实现 Internet 寻址、网络资源的定位连接、在客户机与服务器之间直接访问等

4. URLConnection 对象的作用是什么？

要接收和发关信息还要用 URLConnection 类，程序获得一个 URLConnection 对象，相当于完成对指定 URL 的一个 HTTP 连接。

5. 用代码示意由网址读取网页内容的过程。

```
public void readByURL(String urlName){
try{
    URL url = new URL(urlName);//由网址创建 URL 对象
    URLConnection tc = url.openConnectin();//获得 URLConnection 对象
    tc.connect();//设置网络连接
    InptStreamReader in = new InputStreamReader(tc.getInputStream());
    BufferedReader dis = new BufferedReader(in);//采用缓冲式输入
   String inline;
   while((inline = dis.readLine())!=null){
        showArea.append(inline +" \n" );
    }
    dis.close();//网上资源使用结束后，数据流及时关闭
 } catch(MalformedURLException e){e.printStackTrace();}
 catch(IOException e){e.printStacktrace();}
 /*访问网上资源可能产生 MalformedURLException 和 IOException 异常*/
}
```

6. 网络编程可分哪些层次？

主机/终端模式；客户机/服务器（Client/Server，简称C/S）模式；浏览器/服务器模式

7. 分别说出 socket 连接和 serverSocket 连接的方法。

Server：创建 ServerSocket 对象监听，等待并接收 Client 的请求，利用返回 Socket 对象与 Client 通信，关闭 Socket，关闭监听

Client：创建 Socket 对象向 Server 请求，利用此 Socket 对象与 Server 通信，关闭 Socket 结束与通信。

8. 说出 java 程序与数据库连接的方法。

(1) 使用 JDBC-ODBC 桥接器与数据连接。

(2) 用纯 Java 的 JDBC 驱动程序实现与数据库连接

9. Connection 对象的作用是什么？

Connection 对象是用来表示数据库连接的对象，Java 程序对数据库的操作都在这种对象上进行。

10. ResultSet 对象的作用是什么？

ResultSet 对象实际上是一个由查询结果数据的表，是一个管式数据集，由统一形式的数据行组成，一行对应一条查询记录。在 ResultSet 对象中隐含着一个游标，一次只能获得游标当前所指的数据行，用 next 方法可取下一个数据行。

11. 如何获得可滚动结果集？

程序要获得一个可滚动结果集，只要在获得 SQL 的语句对象时，增加指定结果集的两个参数即可。例如，以下代码：

Statement stmt = con.createStatement(type,concurrency);

ResultSet rs = stmt.executeQuery(SQL 语句)

语句对象 stmt 的 SQL 查询就能得到相应类型的结果集。

int 型参数 type 决定可滚动集的滚动方式：

ResultSet.TYPE_FORWORD_ONLY,结果集的游标只能向下滚动。

ResultSet.TYPE_SCROLL_INSENSITIVE,游标可上下移动，当数据库变化时，当前结果集不变。

ResultSet. TYPE_SCROLL_SENSITIVE，游标可上下移动，当数据库变化时，当前结果集同步改变。

int 型参数 concurrency 决定数据库是否与可滚动集同步更新？

ResultSet.CONCUR_READ_ONLY,不能用结果集更新数据库中的表。

ResultSet.CONCUR_UPDATETABLE,能用结果集更新数据库中的表。

12. 说出实现数据库查询的方法。

利用 Connection 对象的 createStatement 方法建立 Statement 对象，利用 Statement 对象的 executeQuery()方法执行 SQL 查询语句进行查询，返回结果集，再利用 getXXXX()的方法从结果集中读取数据。

13. 分别说出实现数据表记录插入、修改和删除的方法。