Java 程序设计基础教程

（讲稿）

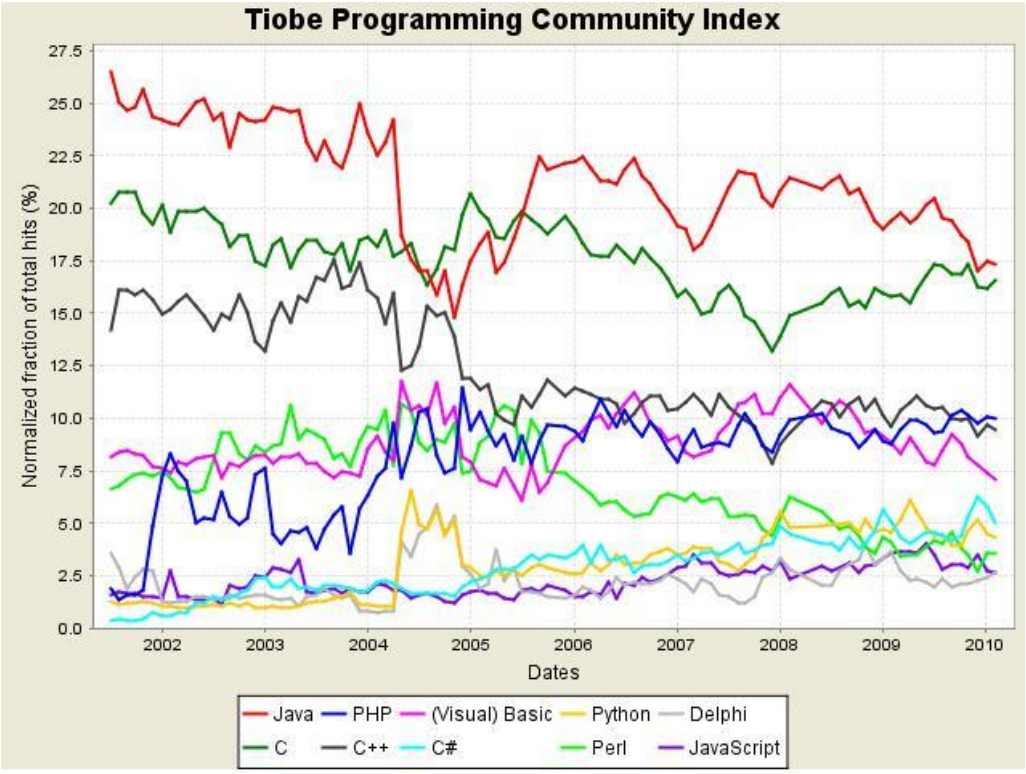
主讲人：靳宗信

黄河科技学院信息工程学院计算机科学系

2011-1-2

一、

二、



三、Java 技术的分类

（1）J2SE：Java 2 Standard Edition

支持所有 JAVA 标准规范中所定义的核心类函数库和所有的 JAVA 基本类别。J2SE 定位在客户端程序的应用上。

（2）J2EE：Java 2 Enterprise Edition

在 J2SE 的基础上增加了企业内部扩展类函数库的支持，比如支持 Servlet/JSP 的 javax.servletr.*和 EJB 的 javax.ejb.*的类函数库。J2EE 定位在服务器端程序的应用上。

（3）J2ME：Java 2 Micro Edition

只支持 JAVA 标准规范中所定义的核心类函数库的子集。定位于嵌入式系统的应用上。

四、Java 程序的分类（J2SE）

（1）Java Application

例：

```
class HelloWorld{
 public static void main(String[] args){
      System.out.println("Hello World!");
 }
```

```
}
```

（2）Java Applet

例：

```
import java.applet.Applet;
import java.awt.*;
public class TestApplet extends Applet{
 public void paint(Graphics g){
        g.drawString("Hello World",10,100);
 }
}
```

```
<HTML>
  <HEAD>
    <TITLE> New Document </TITLE>
  </HEAD>
  <BODY>
    <applet code="TestApplet.class" height=400 width=400>
    </applet>
  </BODY>
</HTML>
```

五、Java 开发工具

（1）JDK：Java Software Develop Kit

可以从下面的网站下载:

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

(http://java.sun.com)

（2）EditPlus：源程序的编辑器

（3）Java 的集成开发环境

Eclipse (MyEclipse)

Netbeans

JBuilder

Jcreator

……

六、JDK 的安装、配置

（1）下载 JDK

（2）安装 JDK

（3）配置环境变量

PATH

CLASSPATH

（4）安装 EditPlus

七、Java 程序的开发过程

（1）Java 应用程序的开发过程

编辑源程序 (EditPlus)

保存源程序 (.java)

编译源程序 (javac.exe)

调试源程序

运行

例子：

```
class HelloWorld{
 public static void main(String[] args){
      System.out.println("Hello World!");
 }
}
```

（2）Java Applet 的开发过程

编辑源程序 (EditPlus)

保存源程序 (.java)

编译源程序 (javac.exe)

调试源程序

运行 (浏览器运行)

例子：

```
import java.applet.Applet;
import java.awt.*;
public class TestApplet extends Applet{
```

```
 public void paint(Graphics g){
        g.drawString("Hello World",10,100);
 }
}

<HTML>
 <HEAD>
  <TITLE> New Document </TITLE>
 </HEAD>
 <BODY>
  <applet code="TestApplet.class" height=400 width=400>
  </applet>
 </BODY>
</HTML>
```

八、参考书籍

（1）《Thinking in Java》Fourth Edition

1999 年 Java 世界最具有影响力的书籍

作者：Bruce Eckel

《Thinking in C++》

《Thinking in Patterns》

《C++Inside & Out》

《Using C++》

（2）最重要的资料：Java API Documentation

可以从以下网站下载:

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

(http://java.sun.com)

作业：

（1）配置 Java 的开发环境

（2）编写一个 Java 应用程序，并运行…

（3）编写一个 Java Applet，并运行…

（4）叙述 Java 为什么是跨平台的？

第二章 Java 语言基础

一、标识符和关键词

（1）标识符：用来表示类名、变量名、方法名、类型名、数组名、文件名的有效字符序列

Java 语言规定：标识符有字母、下划线、美元符号和数字组成，并且第一个字符不能是数字。

Java 语言中的字母不仅是指常用的拉丁字母 **a**、**b**、**c** 等，还包括汉字、日文、朝鲜文、俄文、希腊字母以及其他许多语言中的文字。

（2）关键词：**java** 语言中被赋予特定意义的一些单词。

不能把这些单词作为名字来使用。

常见的关键词如下：

abstract	**boolean**	**break**	**byte**	**case**	**catch**	**char**	**class**
continue	**do**	**double**	**else**	**enum**	**extends**	**false**	**final**
finally	**float**	**for**	**implements**	**import**	**instanceof**	**int**	**interface**
long	**native**	**new**	**null**	**package**	**private**	**protected**	**public**
return	**short**	**static**	**super**	**switch**	**sychronized**	**this**	**throw**
throws	**true**	**try**	**void**	**while**			

二、Java 的数据类型

（一）基本数据类型

基本类型	大小	最小值	最大值	对应包装类
boolean	**—**	**—**	**—**	**Boolean**
char	**16 bit**	**Unicode 0**	**Unicode 216-1**	**Character**
byte	**8 bit**	**-128**	**+127**	**Byte**

short	**16 bit**	**-215**	**-215-1**	**Short**
int	**32 bit**	**-231**	**-231-1**	**Integer**
long	**64 bit**	**-263**	**-263-1**	**Long**
float	**32 bit**	**IEEE754**	**IEEE754**	**Float**
double	**64 bit**	**IEEE754**	**IEEE754**	**Double**

基本数据类型的转换：（按精度从高到底的顺序）

byte →short →int→long→float→double

（1）自动类型转换

（2）强制类型转换

（二）复合数据类型

类型性、接口类型、数组类型等。

三、运算符

（1）+ — * / %

例题 1：

```
public class TestOne{
    public static void main(String[] args){
        System.out.println(7/2);
        System.out.println(7.0/2);
        System.out.println(7.0/2.0);
        System.out.println(7.0/2.0);
    }
}
```

例题 2：

```
public class TestTwo{
    public static void main(String[] args){
        System.out.print(7%-2+"\t");
        System.out.print(7%2+"\t");
        System.out.print(-7%2+"\t");
        System.out.println(-7%-2+"\t");
        System.out.println("7.2%2.8="+7.2%2.8);
    }
```

}

（2）++ ——

例题 3：

```
public class TestThree{
    public static void main(String[] args){
        int a=2;
        a++;
        System.out.println("a="+a);
        int m=a++;
        System.out.println("a="+a+",m="+m);
        int n=++a;
        System.out.println("a="+a+",n="+n);
    }
}
```

算术混合运算的精度：

byte short int long float double

★ char 类型和整数类型运算的结果？

例题 4：

```
public class TestFour{
    public static void main(String[] args){
        byte x=7;
        //那么'B'+x 的结果是什么类型？
    }
}
```

（3）关系运算符：<= == !=

（4）逻辑运算符（短路运算）：&& || !

op1	op2	op1&&op2	op1||op2	!op1
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

例题 5：（短路运算）

```
public class TestFive{
    public static void main(String[] args){
        int x=1,y=1;
        boolean b=(((y=1)==0)&&((x=6)==6));
        System.out.println("x="+x);
        System.out.println("y="+y);
    }
}
```

（5）位运算：

按位逻辑运算：& | ~ ^

例题 6：

```
public class TestSix{
    public static void main(String[] args){
        int a=3;
        int b=5;
        System.out.println("a&b="+(a&b));
        System.out.println("a|b="+(a|b));
        System.out.println("a^b="+(a^b));
        System.out.println("~a="+(~a));
    }
}
```

移位运算：>> >>> <<

例题 7：

```
public class TestSeven{
    public static void main(String[] args){
        int a=5;
        System.out.println("a>>2="+(a>>2));
        System.out.println("a>>>2="+(a>>>2));
        System.out.println("a<<2="+(a<<2));
    }
}
```

（6）其它运算符：

运算符	描述
？：	所用相当于 **if-else**
[ ]	用于声明数组、创建数组及访问数组元素
.	用于访问对象实例或者类的成员
(type)	强制类型转换
new	创建对象或数组
instanceof	判断对象是否为类的实例

四、Java 中的注释

（1）单行注释

```
//此行为单行注释
```

（2）多行注释

```
/*
此处为多行注释
*/
```

（3）文档注释

```
/**
此处为文档注释
*/
```

五、字符界面常见类型数据的输入

（1）字符的输入

例题：

```
import java.io.*;
public class InputChar{
    public static void main(String[] args){
        char c=' ';
        System.out.println("Enter a character please:");
        try{
           c=(char)System.in.read();
        }
        catch(IOException e){
           System.out.println(e);
```

```
        }
        System.out.println("You've entered a character:"+c);
    }
}
```

★上例中如果输入英文准确无误，那么输入汉字呢？

例题：

```
import java.io.*;
public class InputChar{
    public static void main(String[] args){
        char c=' ';
        System.out.println("Enter a character please:");
        try{
            InputStreamReader ln=new InputStreamReader(System.in);
          c=(char)ln.read();
        }
        catch(IOException e){
          System.out.println(e);
        }
        System.out.println("You've entered a character:"+c);
    }
}
```

（2）字符串的输入

```
import java.io.*;
public class InputString{
    public static void main(String[] args){
        String s="";
        System.out.println("Enter a String please:");
        try{
            InputStreamReader ln=new InputStreamReader(System.in);
            BufferedReader in=new BufferedReader(ln);
             s=in.readLine();
        }
        catch(IOException e){
          System.out.println(e);
```

```
        }
        System.out.println("You've entered a String: "+s);
    }
}
```

（3）基本数据类型的输入

```
import java.io.*;
public class InputInteger{
    public static void main(String[] args){
        int a=0;
        System.out.println("Enter a Integer please:");
        try{
            InputStreamReader ln=new InputStreamReader(System.in);
            BufferedReader in=new BufferedReader(ln);
             String s=in.readLine();
               a=Integer.parseInt(s);
        }
        catch(IOException e){
           System.out.println(e);
        }
        System.out.println("You've entered a Integer: "+a);
    }
}
```

★其它输入方式

```
import java.util.*;
public class OtherInput{
    public static void main(String[] args){
        Scanner scan=new Scanner(System.in);
        int a=scan.nextInt();
        System.out.println(a);
    }
}
```

六、流程控制语句

（1）条件控制语句

if 语句

例题：2-6 从键盘上输入 3 个数，输出其中的最大者。

```
import java.io.*;
public class Max{
    public static void main(String[] args){
        int a,b,c,max;
        String s;
        try{
            System.out.print("输入第一个整型数：");
    BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
            s=br.readLine();
            a=Integer.parseInt(s);
            System.out.print("输入第二个整数：");
            s=br.readLine();
            b=Integer.parseInt(s);
            System.out.print("输入第三个整数：");
            s=br.readLine();
            c=Integer.parseInt(s);
            max=a;
            if(b>max)max=b;
            if(c>max)max=c;
            System.out.println("三个数种最大的数是："+max);
        }
        catch (IOException e){}
    }
}
```

switch 语句

例题 2-8 从键盘输入一行字符，分别统计其中数字字符、空格以及其它字符的数量。

```
import java.io.*;
class SwitchTest{
    public static void main(String[] args)throws IOException{
```

```
        int numberOfDigits=0,numberOfSpaces=0,numberOfOthers=0;
        char c;
        while((c=(char)System.in.read())!='\n'){
            switch(c){
                case '0':
                case '1':
                case '2':
                case '3':
                case '4':
                case '5':
                case '6':
                case '7':
                case '8':
                case '9': numberOfDigits++;break;
                case ' ': numberOfSpaces++;break;
                default:    numberOfOthers++;break;
            }
        }
        System.out.println("number Of Digits="+numberOfDigits+"");
        System.out.println("number Of spaces="+numberOfSpaces+"");
        System.out.println("number Of others="+numberOfOthers+"");
    }
}
```

★ Enter 键：' \r\n'

（2）循环语句

while 循环：

例题 2-9：在三位数中找出所有水仙花数，水仙花数的条件是该数等于其各位数字的立方和。

```
public class Narcissus{
    public static void main(String[] args){
        int i,j,k,n=100,m=1;
        while(n<1000){
          i=n/100;   //获取最高位
          j=(n-i*100)/10;   //获取第 2 位
```

```
            k=n%10;  //获取最低位
            if(Math.pow(i,3)+Math.pow(j,3)+Math.pow(k,3)==n)
                System.out.println("找到第"+m++ +"个水仙花数："+n);
            n++;
        }
    }
}
```

do-while 循环

例题 2-11：求 1+2+3+4+…+100 的值。

```
public class Add{
    public static void main(String[] args){
        long sum=0;
        int k=1;
        do{
            sum=sum+k;
            k++;
        }
        while (k<=100);
        System.out.println("1+2+3+...+100="+sum);
    }
}
```

for 语句

例题 2-13：求 Fibonacci（斐波那契）数列的前 10 个数。

```
public class Fibonacci{
    public static void main(String[] args){
        int n0=0,n1=1,n2;
        System.out.print(n0+" "+n1+" ");
        for (int i=0;i<8 ;i++ ){
            n2=n1+n0;
            System.out.print(n2+" ");
            n0=n1;
            n1=n2;
        }
        System.out.println();
```

```
    }
}
```

（3）跳转语句

break 语句：

例题 2-16：Break 语句使用实例。

```
public class UsingBreak{
    public static void main(String[] args){
        int i=0;
        while(true){
            i++;
            System.out.print(i+" ");
            if(i>=9)break;
        }
    }
}
```

带标号的 break

例题 2-17：证明 3~100 之间的数是否符合角谷猜想。

角谷猜想：任何正整数 n，如果是偶数，则除 2，如果是奇数，则乘 3 加 1，得到一个新数，继续这样的处理，最后得到的数一定是 1。

```
public class JGuess{
    public static void main(String[] args){
        int k;
        outer:
            for(k=3;k<=100;k++){
                int n=k;
                do{
                    if(n%2==0)
                        n=n/2;
                    else
                        n=n*3+1;
                    if(n==0){
                        System.out.print(k+"不满足角谷猜想");
                        break outer;
                    }
```

```
                }
                while (n!=1);
            }
            if(k==101)
                System.out.print("数 3~100 之间的数满足角谷猜想");
    }
}
```

continue 语句（带标号的 continue）

七、方法

Java 中的函数称为方法。

①实例方法：对象所拥有的方法。

②静态方法（类方法）：类所拥有的方法。

```
public class TestStaticMethod{
    public static void f(){
       System.out.println("i am a static fun");
    }
    public static void main(String[] args){
       f();
    }
}
```

参数传递：

①基本类型的参数传递：传递的是值。

例题 2-21：基本数据类型参数传递演示。

```
public class SwapTest{
    static void swap(int x,int y){
         int temp;
        temp=x;
        x=y;
        y=temp;
        System.out.println("x="+x+",y="+y);
    }
    public static void main(String[] args){
        int n=7,m=5;
```

```
        swap(n,m);
        System.out.println("n="+n+",m="+m);
    }
}
```

②引用类型的参数传递：传递的是地址。

```
public class RefSwap{
    int value=0;
    static void swap(RefSwap x,RefSwap y){
         int temp;
        temp=x.value;
        x.value=y.value;
        y.value=temp;
        System.out.println("in calling method:");
        System.out.println("x.value="+x.value+",y.value="+y.value);
    }
    public static void main(String[] args){
        RefSwap n=new RefSwap();
        RefSwap m=new RefSwap();
        n.value=4;
        m.value=5;
        System.out.println("before calling:");
        System.out.println("n.value="+n.value+",m.value="+m.value);

        swap(n,m);
          System.out.println("after called:");
        System.out.println("n.value="+n.value+",m.value="+m.value);
    }
}
```

递归：在方法内又调用方法自己。

例题 2-23：求 1+1/2！+1/3！+……+1/10!。

```
public class Factorial{
    static int fac(int n){
         if(n==1)
             return 1;
```

```
        else
            return n*fac(n-1);
    }
    public static void main(String[] args){
        double s=1;
        for (int k=2;k<=10;k++ ){
            s=s+1.0/fac(k);
        }
        System.out.println("1+1/2！+1/3！+·····+1/10!="+s);
    }
}
```

八、Java 数组

一维数组：

（1）声明数组

```
int a[];
int[] a;
```

（2）数组初始化

① int[] a={1,2,3,4,5,6,7,8,9};

② int[] a=new int[9];

```
    for(int i=0;i<9;i++){
        a[i]=i+1;
    }
```

③ int[] a=new int[]{1,2,3,4,5,6,7,8,9};

演变->可变参数

例题：

```
class TestArg{
    static void f(int[] a){
        for (int x:a){
            System.out.print(x+" ");
        }
    }
    public static void main(String[] args){
        int[] b=new int[]{1,2,3,4,5,6,7};
```

```
        f(b);
    }
}
```

可以演变为：

```
class TestArg{
    static void f(int[] a){
        for (int x:a){
            System.out.print(x+" ");
        }
    }
    public static void main(String[] args){
        f(new int[]{1,2,3,4,5,6,7});
    }
}
```

继而演变为：

```
class TestArg{
    static void f(int... a){
        for (int x:a){
            System.out.print(x+" ");
        }
    }
    public static void main(String[] args){
        f(1,2,3,4,5,6,7);
    }
}
```

（3）一维数组的长度

a.lenth

例题：

```
class TestArray{
    public static void main(String[] args){
        int[] a=new int[]{1,2,3,4,5,6,7,8,9};
        for(int i=0;i<a.length;i++){
            System.out.print(a[i]+" ");
        }
```

```
    }
}
```

二维数组：

（1）声明数组

```
int a[][];
int[][] a;
```

（2）数组初始化

① int[][] a={{1,2,3},{4,5,6},{7,8,9}};

```
class TestArray{
    public static void main(String[] args){
        int[][] a={{1,2,3},{4,5,6},{7,8,9}};
        for(int i=0;i<a.length;i++){
            for (int j=0;j<a[i].length;j++ ){
              System.out.print(a[i][j]+" ");
            }
           System.out.println();
        }
    }
}
```

② int[][] a=new int[3][];

```
         a[0]=new int[3];
        a[1]=new int[3];
        a[2]=new int[3];
        for(int i=0;i<a.length;i++){
            for (int j=0;j<a[i].length;j++ ){
              a[i][j]=k;
              k++;
            }
        }
```

例题：

```
class TestArray{
    public static void main(String[] args){
        int k=1;
```

```
        int[][] a=new int[3][];
          a[0]=new int[3];
        a[1]=new int[3];
        a[2]=new int[3];
        for(int i=0;i<a.length;i++){
              for (int j=0;j<a[i].length;j++ ){
               a[i][j]=k;
               k++;
              }
        }
        for(int i=0;i<a.length;i++){
              for (int j=0;j<a[i].length;j++ ){
               System.out.print(a[i][j]+" ");
              }
            System.out.println();
        }
    }
}
```

★ java 中的二维数组可以看做是数组的数组。（每行可以有不同的列数）

例题：二维数组动态创建。

```
public class ArrayOfArrayDemo2{
    public static void main(String[] args){
        int[][] aMatrix=new int[4][];
        for (int i=0;i<aMatrix.length;i++ ){
            aMatrix[i]=new int[i+1];
            for (int j=0;j<aMatrix[i].length;j++ ){
                aMatrix[i][j]=i+j;
            }
        }
        //输出数组
          for (int i=0;i<aMatrix.length;i++ ){
            for (int j=0;j<aMatrix[i].length;j++ ){
                System.out.print(aMatrix[i][j]+" ");
            }
```

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

③ int[][] a=new int[][]{{1,2,3},{4,5,6},{7,8,9}};

```
 class TestArray{
    public static void main(String[] args){
        int[][] a=new int[][]{{1,2,3},{4,5,6},{7,8,9}};
        for(int i=0;i<a.length;i++){
              for (int j=0;j<a[i].length;j++ ){
                System.out.print(a[i][j]+" ");
              }
            System.out.println();
        }
    }
}
```

数组作为方法参数

例题：一维数组作为参数，计算数组中所有元素的和。

```
public class Sum{
    static int sum(int a[]){
         int s=0;
        for(int i=0;i<a.length;i++){
              s=s+a[i];
        }
    return s;
    }
    public static void main(String[] args){
        int a[]={1,2,3,4,5,6,7,8,9,10};
        System.out.println("数组 a 的总和是："+sum(a));
    }
}
```

九、Java 命令行参数

public static void main(String[] args)

例题 2-30：利用命令行参数实现两整数相乘。

```
public class UseComLParameter{
    public static void main(String[] args){
        int a1,a2,a3;
        if (args.length<2){
            System.out.println("运行本程序应该提供两个命令行参数");
            System.exit(0);
        }
        a1=Integer.parseInt(args[0]);
        a2=Integer.parseInt(args[1]);
        a3=a1*a2;
        System.out.println(a1+"与"+a2+"相乘的积为："+a3);
    }
}
```

例题 2-31：输出命令行所有参数。

```
public class AllCommandPara{
    public static void main(String[] args){
        for(int i=0;i<args.length;i++)
            System.out.println(args[i]+" ");
    }
}
```

作业：编写程序用 foreach 语法输出命令行所有参数。

十、java 语言的一些说明

（一）Java 中的注释

（1）单行注释

```
//此行为单行注释
```

（2）多行注释

```
/*
此处为多行注释
```

*/

（3）文档注释

/**

此处为文档注释

*/

（二）java 程序的说明

（1）java 文件的命名：java 文件的名字与主类的名字相同。

（2）如果一个 java 程序只有一个类，那么这个类就是主类。如果一个 java 程序包含多个类，那么其中最多只能有一个类是 public 的，则这个类为主类。

（3）如果一个 java 程序包含多个类，那么每个类都可以含有 main 函数。

（三）java 程序中元素的命名

符合 java 中标识符的命名规则，尽量做到见名知义。

（1）类名首字母大写，如果类名由多个单词组成，则每个单词的首字母都要大写。

（2）变量和函数首字母小写，如果变量和函数由多个单词组成，则从第二个单词开始，首字母大写。

（3）包的名字小写。

（4）在一个完整的 java 程序中，第一行语句为 package 语句，第二行开始为 import 语句，第三部分才是类的定义。如果一个 java 程序由多个类组成，则类之间并无先后次序。

第三章 类与对象

一、编程语言发展的几个阶段

（1）机器语言

这些语言中的指令都是由 0 和 1 组成的序列，称这样的序列为一条机器指令。

（2）面向过程的语言

以事件为中心的编程思想。就是分析出解决问题所需要的步骤，然后用函数把这些步骤一步一步实现，使用的时候一个一个依次调用。

这种语言对底层硬件，内存等操作比较方便，但是写代码和调试维护等会很麻烦。

（3）面向对象的语言

把构成问题事务分解成各个对象，建立对象的目的不是为了完成一个步骤，而是为了描叙某个事物在整个解决问题的步骤中的行为。其对底层的操作不是很方便。

二、面向对象编程取代面向过程编程有哪些原因

（1）以对象为模型去描述现实世界，更接近于人类对现实世界各种事物的理解。

（2）通过封装、继承、多态，更有利于代码重用。

三、面向对象编程的 3 个特性

（1）封装性：将数据和对数据的操作封装到一起。

例题：

```
class Person {
    String name;
    int age;
    void walk(){
        System.out.println("会走路");
    }
}
```

（2）继承性：子类可以继承父类的属性和功能，同时又可以增添自己的属性和功能。

例题：

```
class Person {
    String    name;
    int age;
    void walk(){
        System.out.println("会走路");
    }
}
class Student extends Person{
    String no;
    void takeLessons(){
        System.out.println("我喜欢所有的课程");
    }
}
```

（3）多态性：即多种形态。有 2 中意义的多态：一种是操作名称的多态，即方法的重载。另一种是和继承有关的多态，是指统一操作被不同类型的对象调用时可能产生不同的行为，即运行时刻多态。

方法重载：

```
class Person {
    String name;
    int age;
    void walk(){
        System.out.println("can walk");
    }
}
class Student extends Person{
    String no;
    void takeLessons(){
        System.out.println("我喜欢所有的课程");
    }
    void takeLessons(String cour){
        System.out.println("我也喜欢"+cour);
    }
    public static void main(String[] args){
```

```
            Student zhangSan=new Student();
            zhangSan.name="张三";
            zhangSan.age=18;
System.out.println("我叫"+zhangSan.name+"，我的年龄是"+zhangSan.age+"
岁");
            zhangSan.takeLessons();
            zhangSan.takeLessons("java");
        }
    }
    运行时刻多态：
    class Animal{
        void cry(){}
    }
    class Cat extends Animal{
        void cry(){
            System.out.println("喵喵...");
        }
    }
    class Dog extends Animal{
        void cry(){
            System.out.println("汪汪...");
        }
    }
    public class TestCry{
        public static void main(String[] args){
            Animal a=new Cat();
            a.cry();
            a=new Dog();
            a.cry();
        }
    }
```

四、类

（一）系统定义的类

Java 系统提供的类库也称为 java API，它是系统提供的已实现的标准类的集合，按其用途被划分为若干个不同的包。J2SE 中将 java API 主要分成 3 个包：

（1）java 核心包：java.*;

包括：applet、awt、beans、io、lang、math、net、sql、text、util 等。

（2）java 扩展的包：javax.*;

包括：swing、security、rmi 等。

（3）组织的包：org.*;

主要用于 CORBA 和 XML 处理。

★关键词 import

例题：

```
import java.util.Date;
class TestP{
    public static void main(String[] args){
        Date d=new Date();
        System.out.println(d);
    }
}
```

（二）用户自己定义的类

（1）类的声明和类体

```
public class Person{
    private String address;
    private String name;
    private int age;
    public String getName(){
       return name;
    }
    public int getAge(){
       return age;
    }
    public void changeName(String new_name){
       name=new_name;
    }
    public void incAge(){
       age++;
```

```
    }
    public void setAge(int new_age){
       age=new_age;
    }
    public String getAddress(){
       return address;
    }
    public void setAddress(String x){
       address=x;
    }
    public String toString(){
       String s="Name:"+name+"\n";
       s+="Age:"+age+"\n";
       s+="Address:"+address+"\n";
       return s;
    }
    public static void main(String[] args){

    }
}
```

五、对象的创建和引用

（1）创建对象及访问对象成员

例题：为上例添加 main 方法。

```
public static void main(String[] args){
        Person p1=new Person();
        Person p2=new Person();
        p1.changeName("John");
        p1.setAge(23);
        p1.setAddress("江西");
        p1.incAge();
        p2.changeName("Mary Ann");
        p2.setAge(22);
        p2.setAddress("北京");
```

```
        Person p3=p1;
        p3.age++;
        System.out.println("姓名="+p1.getName()+"，地址="+p1.getAddress());
        System.out.println("姓名="+p2.getName()+"，地址="+p2.getAddress());
    }
```

（2）对象的初始化和构造方法

给对象的变量赋值有以下几种方法：

① 创建对象时赋初值，（全部清零）。

例题：

```
class TestInit{
  int x;
  public static void main(String[] args){
        TestInit t=new TestInit();
        System.out.println(t.x);
  }
}
```

★ 基本数据类型作为成员变量时的初始值：

基本数据类型	默认值
boolean	false
char	‘\u0000’(null)
byte	(byte)0
short	(short)0
int	0
long	0L
float	0.0F
double	0.0D

例题：

```
class TestValue{
    boolean b1;
    byte b2;
    short b3;
    int b4;
```

```
    long b5;
    char b6;
    float b7;
    double b8;
    String b9;
    public static void main(String[] args){
        TestValue t=new TestValue();
        System.out.println("t.b1="+t.b1);
        System.out.println("t.b2="+t.b2);
        System.out.println("t.b3="+t.b3);
        System.out.println("t.b4="+t.b4);
        System.out.println("t.b5="+t.b5);
        System.out.println("t.b6="+t.b6);
        System.out.println("t.b7="+t.b7);
        System.out.println("t.b8="+t.b8);
        System.out.println("t.b9="+t.b9);
    }
}
```

② 定义初始化。

例题：

```
class TestInit{
  int x=1;
  public static void main(String[] args){
      TestInit t=new TestInit();
      System.out.println(t.x);
  }
}
```

③ 初始化块。

例题：

```
class TestInit{
  int x;
  {
    x=2;
```

```
    }
    public static void main(String[] args){
        TestInit t=new TestInit();
        System.out.println(t.x);
    }
}
```

④ 构造方法。

例题：

```
class TestInit{
    int x=1;
    TestInit(int k){
        x=k;
    }
    public static void main(String[] args){
        TestInit t=new TestInit(10);
        System.out.println(t.x);
    }
}
```

六、变量的作用域

① 成员变量：整个类体内有效。

② 局部变量：从声明点到方法体结束或代码块结束。

③ 方法的参数：整个方法体。

例题：

```
public class Scope{
    int x;//成员变量
    int y;
    {
        x=2;
        y=1;
    }
    public void method(int a){
        int x=5;
        for(int i=1;i<a;i++){
```

```
            x=x+i;
        }
        System.out.println("x="+x+",y="+y+",a="+a);
    }
    public static void main(String[] args){
        Scope x=new Scope();
        x.method(6);
    }
}
```

七、类变量和静态方法

成员变量分为两类：

① 实例成员变量，实例变量：对象所有。

② 静态成员变量，静态变量，类变量：类所有。

例题：

```
class TestStatic{
    int x;
    static int y=1;
    public static void main(String[] args){
        TestStatic t1=new TestStatic();
        t1.x=1;
        t1.y=2;
        TestStatic t2=new TestStatic();
        t2.x=3;
        t2.y=4;
        y=y+1;
        TestStatic.y=9;
        System.out.println("t1.x="+t1.x);
        System.out.println("t1.y="+t1.y);
        System.out.println("t2.x="+t2.x);
        System.out.println("t2.y="+t2.y);
        System.out.println("y="+y);
        System.out.println("TestStatic.y="+TestStatic.y);
    }
```

```
    }
```

例题：

```
class TalkPlace{
    static String talkArea="";
}
public class User{
    static int count=0;
    String username;
    int age;
    public User(String name,int yourage){
       username=name;
       age=yourage;
    }
    void login(){
       count++;
       System.out.println("you are no "+count+"user");
    }
    void speak(String words){
       TalkPlace.talkArea=TalkPlace.talkArea+username+"说： "+words+"\n";
    }
    public static void main(String[] args){
       User x1=new User("张三",20);
       x1.login();
       x1.speak("hello");
       User x2=new User("李四",16);
       x2.login();
       x2.speak("good morning");
       x1.speak("bye");
       System.out.println("--讨论区内容如下： ");
       System.out.println(TalkPlace.talkArea);
    }
}
```

成员方法分为两类：

① 实例方法：由对象名调用。

② 静态方法，类方法：类名调用或对象名调用。

★ 只能处理静态变量，只能调用静态方法。

例题：

```
class TestStaticMethod{
    void f(){
       System.out.println("我是实例方法，必须由对象来调用");
    }
    static void g(){
 System.out.println("我是类方法，可以由类的名字调用，也可以直接调用");
    }
    public static void main(String[] args){
        TestStaticMethod t=new TestStaticMethod();
        t.f();
        TestStaticMethod.g();
        g();
    }
}
```

例题：

```
public class FindPrime{
    public static boolean prime(int n){
      for (int k=2;k<Math.sqrt(n);k++ ){
          if(n%k==0)return false;
      }
      return true;
    }
    public static void main(String[] args){
        for (int m=10;m<=100 ;m++ ){
            if(prime(m))
                System.out.print(m+",");
        }
    }
}
```

八、使用包组织类

包是 Java 语言中有效的管理类的一个机制。

（1）用关键词 package 声明包语句。

例题：

```
package jin;
public class TestPackage{
    public void f(){
       System.out.println("i am testing package...");
    }
}
```

（2）引入包：

① 引入系统的包；

例题：

```
import java.util.Date;
class TestImport1{
    public static void main(String[] args){
        Date d=new Date();
        System.out.println(d);
    }
}
```

② 引入自定义包；

```
import jin.*;
class TestImport2{
    public static void main(String[] args){
        TestPackage t=new TestPackage();
        t.f();
    }
}
```

★ 制作工具类。

```
package jin.util;
import java.io.*;
public class InputData{
    public static char getChar()throws IOException{
```

```
        System.out.println("请输入一个字符：");
        InputStreamReader in=new InputStreamReader(System.in);
        char c=(char)in.read();
        return c;
    }
    public static byte getByte()throws Exception{
        System.out.println("请输入一个 byte 值：");
    BufferedReader in=new BufferedReader(new
InputStreamReader(System.in));
        String s=in.readLine();
        byte b=Byte.parseByte(s);
        return b;
    }
     public static short getShort()throws Exception{
        System.out.println("请输入一个 short 值：");
    BufferedReader in=new BufferedReader(new
InputStreamReader(System.in));
        String s=in.readLine();
        short b=Short.parseShort(s);
        return b;
     }
     public static int getInt()throws Exception{
        System.out.println("请输入一个 int 值：");
    BufferedReader in=new BufferedReader(new
InputStreamReader(System.in));
        String s=in.readLine();
        int b=Integer.parseInt(s);
        return b;
    }
     public static long getLong()throws Exception{
        System.out.println("请输入一个 long 值：");
    BufferedReader in=new BufferedReader(new
InputStreamReader(System.in));
        String s=in.readLine();
```

```
        long b=Long.parseLong(s);
        return b;
    }
    public static float getFloat()throws Exception{
        System.out.println("请输入一个 float 值： ");
    BufferedReader in=new BufferedReader(new
InputStreamReader(System.in));
        String s=in.readLine();
        float b=Float.parseFloat(s);
        return b;
    }
    public static double getDouble()throws Exception{
        System.out.println("请输入一个 double 值： ");
    BufferedReader in=new BufferedReader(new
InputStreamReader(System.in));
        String s=in.readLine();
        double b=Double.parseDouble(s);
        return b;
    }
    public static String getString()throws IOException{
        System.out.println("请输入一个字符串： ");
    BufferedReader in=new BufferedReader(new
InputStreamReader(System.in));
        String s=in.readLine();
        return s;
    }
}
```

第四章 继承、多态和接口

一、访问控制符

（1）类的访问权限：

① 公共访问权限（public）。

② 包访问权限。

（2）成员的访问权限

① public

② private

③ protected

④ 默认包访问

★ 结论：

	public	包访问	protected	private
相同包中的非子类中被访问	YES	YES	YES	NO
相同包中的子类中被访问	YES	YES	YES	NO
不同包中的非子类中被访问	YES	NO	NO	NO
不同包中的子类中被访问	YES	NO	NO	NO

二、继承

（一）Java 中继承的实现

① extends 关键字

例题：

```
class Person {
    String  name;
    int age;
    void walk(){
        System.out.println("会走路");
    }
}
class Student extends Person{
    String no;
```

```
    void takeLessons(){
        System.out.println("我喜欢所有的课程");
    }
}
```

（二）构造方法在继承中的作用

① 若子类没有定义构造方法，创建对象时将无条件的调用父类的无参构造方法。

② 对于父类的有参构造方法，子类可以在自己的构造方法中使用关键字 **super** 来调用它，但 **super** 语句必须是子类构造方法的第一条语句。

③ 子类在自己的构造方法中如果没有使用 **super** 明确调用父类的构造方法，则在创建对象时，将自动先执行父类的无参构造方法，然后再执行自己定义的构造方法。

例题：

```
class Parent{
    String my;
    public Parent(String x){
        my=x;
    }
}
public class   SubClass extends Parent{
    public static void main(String[] args){


    }
}
```

（三）子类的继承性

① 子类和父类在同一包中的继承性：

例题：

② 子类和父类不在同一个包中的继承性：

例题：

★ 结论：

	public	包访问	protected	private

可被相同包中的子类继承后直接访问	**YES**	**YES**	**YES**	**NO**
可被不同包中的子类继承后直接访问	**YES**	**NO**	**YES**	**NO**

（四）变量的继承和隐藏

例题：

```
class Parent{
    int a=3;
    int m=2;
}
public class SubClass extends Parent{
    int a=4;
    int b=1;
    public static void main(String[] args){
        SubClass my=new SubClass();
        System.out.println("a="+my.a+",b="+my.b+",m="+my.m);
    }
}
```

三、多态性

① 方法的重载

例题：

```
public class A{
    int x=0;
    void test(int x){
       System.out.println("test(int):"+x);
    }
    void test(long x){
       System.out.println("test(long):"+x);
    }
    void test(double x){
       System.out.println("test(double):"+x);
    }
    public static void main(String[] args){
```

```
        A a1=new A();
        a1.test(5.0);
        a1.test(5);
    }
}
```

★ 方法的覆盖

例题：

```
class A{
    int x=0;
    void test(int x){
        System.out.println("in A.test(int):"+x);
    }
    void test(long x){
        System.out.println("in A.test(long):"+x);
    }
    void test(double x){
        System.out.println("in A.test(double):"+x);
    }
}
public class B extends A{
    void test(int x){
         System.out.println("in B.test(int)"+x);
    }
    void test(String x,int y){
         System.out.println("in B.test(String,int):"+x+","+y);
    }
    public static void main(String[] args){
        B b1=new B();
        b1.test("abcd",10);
        b1.test(5);
        b1.test(5.0);
    }
}
```

② 运行时刻多态

例题：

```
class Animal{
    void cry(){}
}
class Cat extends Animal{
    void cry(){
        System.out.println("喵喵...");
    }
}
class Dog extends Animal{
    void cry(){
        System.out.println("汪汪...");
    }
}
public class TestCry{
    public static void main(String[] args){
        Animal a=new Cat();
        a.cry();
        a=new Dog();
        a.cry();
    }
}
```

四、this 和 super

（一）关键字 this

关键字 **this** 表示“当前对象”。有下面几种用法：

（1）调用当前对象的其他方法或访问当前对象的实例变量；

例如：

```
class TestThis1{
    int x=1;
    void f(){}
    void g(){
        int a=this.x+1;//此时 this 可以省略。
```

```
        this.f();//此时 this 可以省略。
    }
    public static void main(String[] args){

    }
}
```

（2）当做方法的返回值返回或作为参数传递；

例题：当作方法的返回值：

```
public class Leaf {
  int i = 0;
  Leaf increment() {
    i++;
    return this;
  }
  void print() {
    System.out.println("i = " + i);
  }
  public static void main(String[] args) {
    Leaf x = new Leaf();
    x.increment().increment().increment().increment().increment().print();
  }
}
```

例题：作为参数传递：

```
class A{
    void f(){
        System.out.println("I am f in class A");
    }
    A(){
      new TestThis2(this);
    }
};
public class TestThis2{
    TestThis2(A a){
        a.f();
```

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

（3）区分成员变量和局部变量同名的情况；

例题：

```
class TestThis3{
    int x;
    public TestThis3(int x){
    this.x=x+10;
    }
    public static void main(String[] args){
        TestThis3 t=new TestThis3(5);
        System.out.println(t.x);
    }
}
```

（4）一个构造方法中调用另外一个构造方法。

例题：

```
class TestThis4 {
    int x,y;
    public TestThis4(int x,int y){
        this.x=x;
        this.y=y;
        }
    public TestThis4(int x){
    this(x,3);
    }
    public static void main(String[] args){
        TestThis4 t=new TestThis4(1);
        System.out.println("t.x="+t.x);
        System.out.println("t.y="+t.y);
    }
}
```

（二）关键字 **super**

① 通过 super 访问父类成员

例题：

```
public class SubClass extends Parent{
   int a=6;
   void f(){
      super.f();
      a=a+super.a-3;
   }
   public static void main(String[] args){
      SubClass my=new SubClass();
      my.f();
      System.out.println("a="+my.a);
   }
}
```

② 调用父类的构造方法

例题：

```
class Parent{
   Parent(){
     System.out.println("i am Parent()");
   }
}
public class SubClass extends Parent{
   SubClass(){
        super();
         System.out.println("i am SubClass()");
   }
   public static void main(String[] args){
      SubClass my=new SubClass();
   }
}
```

五、final 修饰符

① final 修饰的类：不能有子类。

② final 修饰的函数：不能被重写。

③ final 修饰的域：不能被修改。

④ final 修饰的数组（引用）：不能被修改，但所指向的数组（对象）中的内容可以被修改。

⑤空白 final：必须在域的定义处或者每个构造器中用表达式对 final 进行赋值。

六、抽象类和抽象方法

由 **abstract** 修饰的类为抽象类，由 **abstract** 修饰的方法为抽象方法。

性质：

（1）抽象类不能制造对象。

（2）抽象类中可以包含一般函数和抽象函数，可以没有抽象函数，但抽象函数必须在抽象类中定义。（接口是完全抽象类）

（3）如果抽象类的子类不是抽象类，它必须实现父类的所有抽象函数。

（4）如果抽象类的子类也是抽象类，那么它可以实现一部分或者不实现父类的抽象函数。

七、接口

（1）接口的说明

例题：

```
interface MyInterface{
   public static final int I=1;
   public abstract void f();
}
```

★ 接口中的变量是常量，默认的修饰符为 **public static final**，可以省略。

★ 接口中的方法是抽象方法，默认的修饰符为 **public abstract**，可以省略。

★ 一个要实现某个接口，它必须实现接口中的所有抽象方法，并且必须加上 **public** 的访问权限。

（2）接口回调

例题：

```
interface    ShowMessage{
    void  显示商标(String s);
}
class TV implements ShowMessage{
    public void  显示商标(String s){
        System.out.println(s);
    }
}
class PC implements ShowMessage{
    public void  显示商标(String s){
        System.out.println(s);
    }
}
public class Example4_27{
    public static void main(String args[]){
        ShowMessage sm;
          sm=new TV();
          sm.显示商标("长城牌电视机");
          sm=new PC();
          sm.显示商标("联想奔月 5008PC 机");
    }
}
```

（3）接口作为参数

例题：

```
interface SpeakHello{
    void speakHello();
}
class Chinese implements SpeakHello{
    public    void speakHello(){
        System.out.println("中国人习惯问候语:你好,吃饭了吗？  ");
    }
}
class English implements SpeakHello{
    public    void speakHello(){
```

```
        System.out.println("英国人习惯问候语：你好，天气不错 ");
     }
}
class KindHello{
    public void lookHello(SpeakHello hello){
        hello.speakHello();
    }
}
public class Example4_29{
    public static void main(String args[]){
       KindHello kindHello=new KindHello();
        kindHello.lookHello(new Chinese());
        kindHello.lookHello(new English());
    }
}
```

八、内嵌类

（1）成员类

例题：

```
public class OuterOne{
    private int x=3;
    InnerOne ino=new InnerOne();
    //成员类开始
    public class InnerOne{
        private int y=5;
        public void innerMethod(){
           System.out.println("y is "+y);
        }
        public void innerMethod2(){
           System.out.println("x2 is "+x);
        }
    };//成员类结束
     public void outerMethod(){
```

```
        System.out.println("x is "+x);
        ino.innerMethod();
        ino.innerMethod2();
    }
    public static void main(String[] args){
        OuterOne my=new OuterOne();
        my.outerMethod();
    }
}
```

★ 从上面的程序可以看出，内嵌类和外层类的其它成员处于同级别位置，所以也称为成员类。

★ 在内嵌类中可以访问外围类的成员。

★ 成员类中不能定义静态方法。

★ 在外围类中访问内嵌类的方法：

① 在外围类的成员定义处创建内嵌类的对象。

② 在外围类的某个成员方法中创建内嵌类的对象，然后通过该对象访问内嵌类的成员。

③ 在外围类的静态方法中不能直接创建内嵌类的对象，此时，必须先创建外围类的对象，然后通过外围类的对象创建内嵌类的对象。

例如：

```
public static void main(String[] args){
        OuterOne.InnerOne i=new OuterOne().new InnerOne();
        i.innerMethod();
    }
```

★ 在内嵌类中使用关键字 this：在内嵌类中，this 是指内嵌类的对象，要访问外层类的当前对象必须加上外层类的名作为前缀。

例题：

```
public class A{
    private int x=3;
    //内嵌类
    public class B{
        private int x=5;
        public void M(int x){
          System.out.println("x="+x);
```

```
            System.out.println("this.x="+this.x);
            System.out.println("A.this.x="+A.this.x);
        }
    };
    public static void main(String[] args){
        A a=new A();
        A.B b=a.new B();
        b.M(6);
    }
}
```

（2）嵌套类：内嵌类可以定义为静态的，也称为静态内部类。

例题：

```
public class OuterTwo{
    private static int x=3;
    private int y=5;
    //静态内部类
    public static class InnerTwo{
        public static void Method(){
            System.out.println("x is "+x);
        }
        public void Method2(){
            System.out.println("x is "+x);
            //System.out.println("y is "+y);
        }
    };//静态内部类结束
    public static void main(String[] args){
        OuterTwo.InnerTwo.Method();
        new OuterTwo.InnerTwo().Method2();
    }
}
```

★ 静态内部类中可以定义实例方法和静态方法。静态方法可以通过类名直接访问，而实例方法必须通过内部类的对象访问。

（3）局部内部类

① 方法中的内部类

例题：

```
public class OuterThree{
    private int x=3;
    public void outerMethod(int m){
       final int n=x+2;
       //局部内部类
       class InnerThree{
            private int y=5;
            public void innerMethod(){
              System.out.println("y is "+y);
              System.out.println("n is "+n);
              //System.out.println("m is "+m);
              System.out.println("x is "+x);
            }
       };//局部内部类结束
       InnerThree in3=new InnerThree();
       in3.innerMethod();
    }
    public static void main(String[] args){
       OuterThree my=new OuterThree();
       my.outerMethod(8);
    }
}
```

★ 局部内部类与局部变量相似，前面不能用访问修饰符修饰。（public 等）

② 匿名内部类

★与接口有关的匿名内部类：

例题：

```
interface Sample{
   void testMethod();
}
public class AnonymouseInner{
    void outerMethod(){
       new Sample(){
          public void testMethod(){
```

```
                System.out.println("just test");
              }
            }.testMethod();
          }
           public static void main(String[] args){
            AnonymouseInner my=new AnonymouseInner();
            my.outerMethod();
          }
      };
```

★与类（一般类或抽象类）有关的匿名内部类：

例题：

```
class Sample{
  void testMethod(){};
}
public class AnonymouseInner{
     void outerMethod(){
       new Sample(){
          public void testMethod(){
            System.out.println("just test");
          }
       }.testMethod();
     }
      public static void main(String[] args){
       AnonymouseInner my=new AnonymouseInner();
       my.outerMethod();
     }
};
```

九、对象引用转换

向上转型：父类类型的引用指向子类创建的对象。（Java 允许）

例题：

```
class SuperClass{
};
```

```
class SubClass extends SuperClass{
};
public class TestUpCasting{
    public static void main(String[] args){
        SuperClass s=new SubClass();
    }
}
```

向下转型：子类类型的引用指向父类创建的对象。（可能出错）

例题：

```
class SuperClass{
};
class SubClass extends SuperClass{
};
public class TestUpCasting{
    public static void main(String[] args){
        SubClass s=new SuperClass();
    }
}
```

★ 向上转型后，分清楚对同名成员的引用。

例题：

```
class SuperClass{
    int x=1;
    void f(){
       System.out.println("A");
    }
};
class SubClass extends SuperClass{
    int x=2;
    void f(){
       System.out.println("B");
    }
};
public class TestUpCasting{
    public static void main(String[] args){
```

```
        SuperClass s=new SubClass();
        System.out.println("s.x="+s.x);
        s.f();
    }
}
```

★ 只有非私有的实例方法是多态的。

★ 域、静态方法、**final** 的方法、私有方法都不是多态的。

第五章 常用系统类

一、语言基础类

（1）Object 类

Java 中所有类的祖先类。

（2）Math 类

包含用来完成常用的数学运算的方法及 Math.PI 和 Math.E 两个数学常量。

例题：利用随机函数产生 10 道二位数的加法测试题，根据用户输入计算得分。

```
import java.io.*;
public class AddTest{
    public static void main(String[] args)throws IOException{
        int score=0;
BufferedReader in=new BufferedReader(new InputStreamReader
(System.in));
        for (int i=0;i<10;i++){
            int a=10+(int)(90*Math.random());
            int b=10+(int)(90*Math.random());
            System.out.print(a+"+"+b+"=?");
            int ans=Integer.parseInt(in.readLine());
            if(a+b==ans)
                score=score+10;
        }
        System.out.print("you score="+score);
    }
}
```

（3）数据类型包装类

基本类型	对应包装类
boolean	Boolean
char	Character

byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

例题：

例题：

```
public class E437{
    public static void main(String args[ ]){
        char a[]={'a','b','c','D','E','F'};
          for(int i=0;i<a.length;i++){
           if(Character.isLowerCase(a[i])){
                a[i]=Character.toUpperCase(a[i]);
                }
                else if(Character.isUpperCase(a[i])){
                a[i]=Character.toLowerCase(a[i]);
                }
          }
          for(int i=0;i<a.length;i++){
           System.out.print(" "+a[i]);
          }
    }
}
```

二、字符串

（1）String 类：创建不可变长的字符串

① 创建字符串

String s1=“abc”;

String s2=new String(“abc”);

…

② 比较两个字符串

==和 equals()方法。

例题：

```
class TestEquals{
   public static void main(String[] args){
      String s1="abc";
      String s2="abc";
      String s3=new String("abc");
      String s4=new String("abc");
      System.out.println(s1==s2);
      System.out.println(s2==s3);
      System.out.println(s3==s4);
      System.out.println(s1.equals(s2));
      System.out.println(s1.equals(s3));
   }
}
```

③ 字符串的其他常用方法

length()：字符串的长度，即字符串中字符的个数。

concat(String str):字符串的连接。

“+”：字符串的连接。

indexOf(int ch):在当前的字符串中查找字符 **ch**，从前向后找。

lastIndexOf(int ch): 在当前的字符串中查找字符 **ch**，从后向前找。

字符串的分析：**StringTokenizer** 类。

例题：

```
import java.util.*;
public class WordAnalyse{
   public static void main(String[] args){
   StringTokenizer st=new StringTokenizer("hello everybody");
       while(st.hasMoreTokens()){//判断是否有后续单词。
           System.out.println(st.nextToken());//取下一个单词。
       }
   }
}
```

public String[ ] split(String regex)方法：使用指定分隔符分离字符串。

例题：

```
public class TestSplit{
   public static void main(String[] args){
        String s=new String("boo:and:foo");
        for(String s1:s.split(":"))
            System.out.println(s1);
   }
}
```

（2）**StringBuffer** 类：创建可变长的字符串

① 创建 **StringBuffer** 对象

public StringBuffer():创建一个空的 **StringBuffer** 对象。

public StringBuffer(int length): 创建一个长度为 **length** 的 **StringBuffer** 对象。

public StringBuffer(String str):用字符串 **str** 初始化 **StringBuffer** 对象。

② **StringBuffer** 的主要方法

例题：

```
class TestStringBuffer{
   public static void main(String[] args){
        StringBuffer str=new StringBuffer();
        str.append("Hello ");
        str.append("Mary!");
        str.insert(6,"good ");
        str.setCharAt(6,'G');
        str.deleteCharAt(15);
        str.replace(11,15,"Tom");
        System.out.println(str);
   }
}
```

（3）**StringBuilder** 类**(JDK5.0** 开始**)**：

StringBuilder 不支持同步。

StringBuffer 支持同步。

性能：StringBuilder>StringBuffer>String

例题：

```
public class TestB{
        final static int ttime=100000;//测试循环次数
        public TestB(){}
        public void test(String s){
            long begin=System.currentTimeMillis();
            for(int i=0;i<ttime;i++){
                s+="add";
            }
            long over=System.currentTimeMillis();
            System.out.println("操作"+s.getClass().getName()+"类型
使用的时间为："+(over-begin)+"毫秒");
        }
        public void test(StringBuffer s){
            long begin=System.currentTimeMillis();
            for(int i=0;i<ttime;i++){
                s.append("add");
            }
            long over=System.currentTimeMillis();
            System.out.println("操作"+s.getClass().getName()+"类型
使用的时间为："+(over-begin)+"毫秒");
        }
           public void test(StringBuilder s){
            long begin=System.currentTimeMillis();
            for(int i=0;i<ttime;i++){
                s.append("add");
            }
            long over=System.currentTimeMillis();
            System.out.println("操作"+s.getClass().getName()+"类型
使用的时间为："+(over-begin)+"毫秒");
        }
        public void test2(){
            String s2="abadf";
```

```
            long begin=System.currentTimeMillis();
            for(int i=0;i<ttime;i++){
                String s=s2+s2+s2;
            }
            long over=System.currentTimeMillis();
            System.out.println("操作字符串对象引用相加类型使用的
时间为："+(over-begin)+"毫秒");
        }
        public void test3(){
            String s2="abadf";
            long begin=System.currentTimeMillis();
            for(int i=0;i<ttime;i++){
                String s="abadf"+"abadf"+"abadf";
            }
            long over=System.currentTimeMillis();
            System.out.println("操作字符串相加使用的时间为：
"+(over-begin)+"毫秒");
        }
      public static void main(String[] args){
          String s1="abc";
        StringBuffer sb1=new StringBuffer("abc");
        StringBuilder sb2=new StringBuilder("abc");
        TestB t=new TestB();
        t.test(s1);
        t.test(sb1);
        t.test(sb2);
        t.test2();
        t.test3();
    }
}
```

三、Vector 类

Vector 类实现了可扩展的对象数组。

例题：测试向量的大小及容量的变化。

```
import java.util.*;
public class TestCapacity{
    public static void main(String[] args){
        Vector v=new Vector();
        System.out.println("size="+v.size());
        System.out.println("capacity="+v.capacity());
        for(int i=0;i<14;i++){
            v.add("hello");
        }
        System.out.println("After added 14 Elements");
        System.out.println("size="+v.size());
        System.out.println("capacity="+v.capacity());
    }
}
```

（1）给向量序列尾部添加新元素

（2）获取向量序列中元素

（3）查找向量序列中元素

（4）修改向量序列中元素

（5）删除向量序列中元素

（6）向量的遍历访问

例题：向量的添加、获取、修改、删除和遍历。

```
import java.util.*;
public class TestVector{
    public static void main(String[] args){
        Vector v=new Vector();
        v.add("abc");
        v.addElement("def");
        v.add("xyz");
        System.out.println("v(1)="+v.elementAt(1));
        System.out.println("v(1)="+v.get(1));
        System.out.println(v);
        v.setElementAt("DEF",1);
          System.out.println(v);
        v.removeElementAt(1);
```

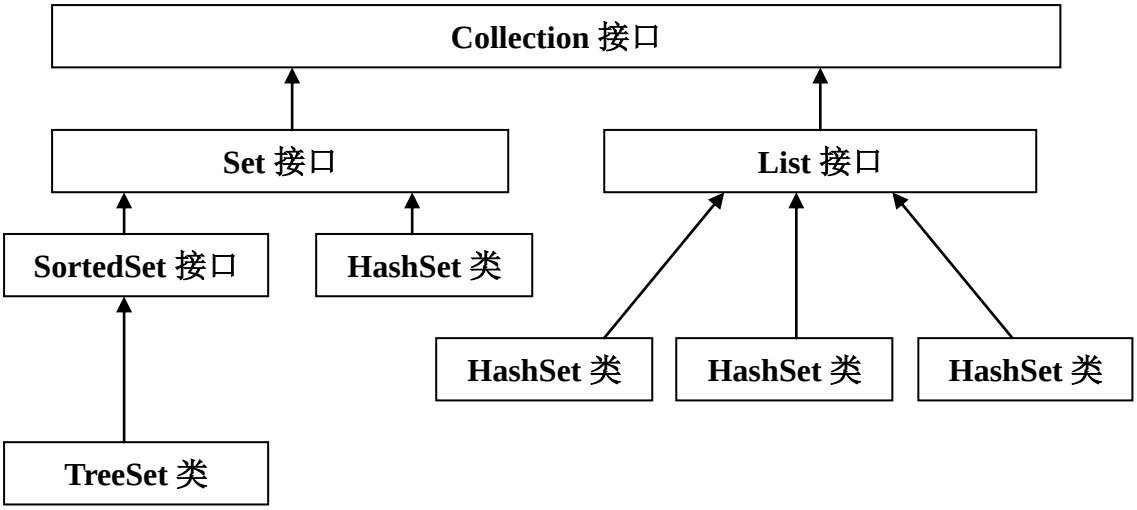
```
        System.out.println(v);
        for(int i=0;i<v.size();i++){
            System.out.print(v.get(i)+" ");
        }
        System.out.println();
        Iterator x=v.iterator();
        while(x.hasNext())
            System.out.print(x.next()+" ");
    }
}
```

四、Collection API 简介

在 **Java API** 中为了支持各种对象的存储访问提供了 **Collection**（收集）系列 **API**，**Vector** 是这种类型的一种。

（1）Collection 接口及实现层次

接口 **Collection** 处于 **Collection API** 的最高层次，其中定义了所有底层接口或类的公共方法，如下图所示：



① Collection 接口

- boolean add(Object o)：将一个对象加入到收集中。
- boolean contains(Object o)：判断收集中是否包含指定对象。
- boolean isEmpty()：判断收集是否为空。
- Iterator iterator()：取得遍历访问收集的迭代子对象。
- boolean remove(Object o)：从收集中删除某对象。
- int size()：获取对象的大小。

- Object[] toArray()：将收集元素转化为对象数组。
- void clear()：删除收集中所有的元素。

② Set 接口

该接口是数学上集合模型的抽象，有两个特点：一是不含重复元素，而是无序。

例题：

```
import java.util.*;
public class TestSet{
   public static void main(String[] args){
       HashSet h=new HashSet();
       h.add("Str1");
       h.add(new Integer(12));
       h.add(new Double(4.2));
       h.add("Str1");
       h.add(new String("Str1"));
       System.out.println(h);
   }
}
```

③ List 接口

该接口类似于数学上数列的模型，也称序列。其特点是可含有重复元素，而且是有序的。

★ 类 ArrayList 内部使用基于动态数组的数据结构。类 **LinkedList** 内部使用基于链表的数据结构。

★ 对于随机访问 get 和 set，ArrayList 要优于 LinkedList，因为 linkedList 要移动指针。

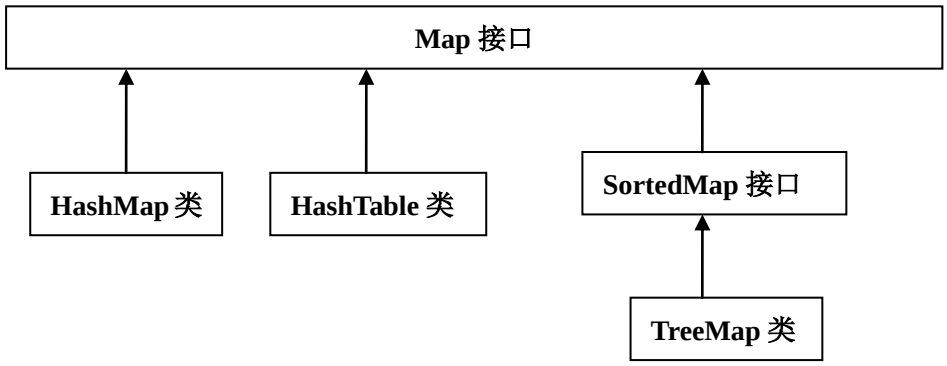
★ 对于新增和删除元素操作 add 和 remove，LinkedList 比较占优势，因为 ArrayList 要移动数据。

例题：

```
import java.util.*;
public class TestList{
    public static void main(String[] args){
        ArrayList a=new ArrayList();
        a.add("Str1");
        a.add(new Integer(12));
        a.add(new Double(4.2));
        a.add("Str1");
        a.add(new String("Str1"));
        System.out.println(a);
        //或者使用迭代器访问 a 中的元素
        Iterator p=a.iterator();
        while(p.hasNext()){
          System.out.print(p.next()+",");
        }
    }
```

```
}
```
例题：
```
import java.util.*;
public class ListDemo2{
       final int N=50000;
       long timeList(List st){
            long start=System.currentTimeMillis();
           for(int i=0;i<N;i++){
                 Object obj=new Integer(i);
                st.add(obj);
           }
           long stop=System.currentTimeMillis();
           return (stop-start);
     }
     public static void main(String[] args){
           ListDemo2 l=new ListDemo2();
           System.out.println("time for ArrayList="+l.timeList(new ArrayList()));
        System.out.println("time for LinkedList="+l.timeList(new LinkedList()));
     }
}
```
（2）Map 接口及实现层次

除了 Collection 表示的这种单一对象数据集合，对于“”表示的数据集合在 Collection API 提供了 Map 接口。Map 接口及其子接口的实现层次如下图：



例题：
```
import java.util.*;
public class MapDemo{
     public static void main(String[] args){
           Map m=new HashMap();
           m.put("张三","200311");
           m.put("李四","200312");
           m.put("王五","200313");
           Set allEntry=m.entrySet();
```

```
        for(Iterator i=allEntry.iterator();i.hasNext();){
            System.out.println(i.next());
        }
        System.out.println(m.get("李四"));
    }
}
```

五、日期和时间

（一）Date 类

（1）java.util.Date 类：

例题：输出当前的时间

```
import java.util.*;
class OutDate{
    public static void main(String[] args){
         Date d=new Date();
        System.out.println("当前的时间是："+d);
    }
}
```

（2）按格式输出时间对象

Java.text.SimpleDateFormat 类

例题：按照 xxxx 年 xx 月 xx 日 星期 x xx 时 xx 分 xx 秒的格式输出当前时间。

```
import java.util.*;
import java.text.SimpleDateFormat;
class OutDate{
    public static void main(String[] args){
         Date d=new Date();
        SimpleDateFormat sm=new SimpleDateFormat("现在的时间是：yyyy 年 MM 月 dd 日 E HH 时 mm 分 ss 秒");
        System.out.println(sm.format(d));
    }
```

（二）Calendar 类

java.util 包中的类为抽象方法，可以使用 static 的方法 getInstance()初始化一个日立对象。

set()方法可以将日立翻到任何一个时间。

例题：

```
import java.util.*;
```

```
class Example6_2{
    public static void main(String args[]){
        Calendar calendar=Calendar.getInstance();
          calendar.setTime(new Date());
  String  年=String.valueOf(calendar.get(Calendar.YEAR)),
          月=String.valueOf(calendar.get(Calendar.MONTH)+1),
   日=String.valueOf(calendar.get(Calendar.DAY_OF_MONTH)),
星期=String.valueOf(calendar.get(Calendar.DAY_OF_WEEK)-1);
          int hour=calendar.get(Calendar.HOUR_OF_DAY),
             minute=calendar.get(Calendar.MINUTE),
             second=calendar.get(Calendar.SECOND);
          System.out.println("现在的时间是：");
          System.out.println(""+年+"年"+月+"月"+日+"日 "+ "星期"+星
期);
          System.out.println(""+hour+"时"+minute+"分"+second+"秒");
          calendar.set(1962,5,29);  //将日历翻到 1962 年 6 月 29 日,注意 5 表
示六月。
          long time1962=calendar.getTimeInMillis();
          calendar.set(2006,9,1);
          long time2006=calendar.getTimeInMillis();
          long  相隔天数=(time2006-time1962)/(1000*60*60*24);
          System.out.println("2006 年 10 月 1 日和 1962 年 6 月 29 日相隔"+
相隔天数+"天");
    }
}
```

例题：输出 2006 年 12 月的日历页

```
import java.util.*;
class Example6_3{
    public static void main(String args[]){
        System.out.println(" 日 一 二 三 四 五 六");
          Calendar 日历=Calendar.getInstance();
          日历.set(2006,11,1);  //将日历翻到 2006 年 12 月 1 日。
          int 星期几=日历.get(Calendar.DAY_OF_WEEK)-1;
```

```
            String a[]=new String[星期几+31];
            for(int i=0;i<星期几;i++){
               a[i]="**";
            }
            for(int i=星期几,n=1;i<星期几+31;i++){
               if(n<=9)
                        a[i]=String.valueOf(n)+" ";
                  else
                        a[i]=String.valueOf(n) ;
                  n++;
            }
            for(int i=0;i<a.length;i++){
               if(i%7==0){
                  System.out.println("");
                  }
            System.out.print(" "+a[i]);
         }
      }
}
```

第六章 Java Applet

一、Applet 简介

可以在浏览器中运行的小应用程序。
例题：

```
import java.awt.*;
import java.applet.Applet;
public class MyApplet extends Applet{
    public void paint(Graphics g){
        g.drawString("HelloWorld",100,100);
    }
}
```

编写 HTML 文件，并把 MyApplet.class 嵌入到 HTML 文件中。

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<applet code=MyApplet.class codebase=jin height=400 width=400></applet>
</BODY>
</HTML>
```

二、Applet 方法介绍

（1）init()方法
（2）start()方法
（3）paint()方法
（4）stop()方法
（5）destroy()方法
（6）update()方法
（7）repaint()方法

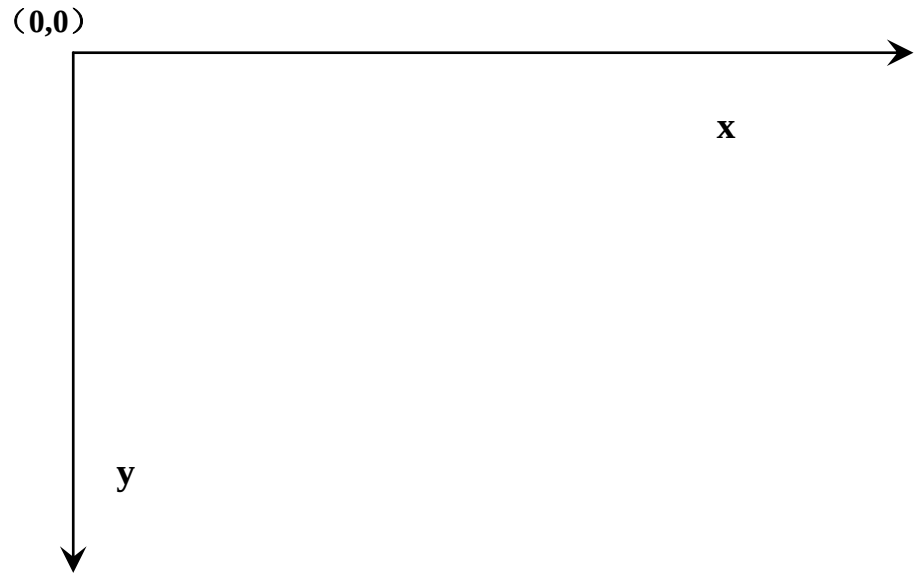
例题 6-1 一个验证 Applet 方法执行次数的测试程序

```
import java.awt.*;
import java.applet.Applet;
public class Count extends Applet{
    static int initCount=0;
    static int startCount=0;
    static int paintCount=0;
    static int stopCount=0;
    static int destroyCount=0;
    public void init(){
       initCount++;
    }
```

```
    public void start(){
       startCount++;
    }
    public void stop(){
       stopCount++;
    }
    public void destroy(){
       destroyCount++;
    }
    public void paint(Graphics g){
       paintCount++;
       g.drawString("init()="+initCount,50,30);
       g.drawString("start()="+startCount,100,30);
       g.drawString("paint()="+paintCount,150,30);
       g.drawString("stop()="+stopCount,250,30);
    }
}
```

三、Applet 的 AWT 绘制

（1）Java 图形坐标



（2）各类图形的绘制方法

例题：绘制一个微笑的人脸。

```
import java.awt.*;
import java.applet.Applet;
public class SimlePeople extends Applet{
    public void paint(Graphics g){
        g.drawString("永远的微笑！！ ",50,30);
```

```
        g.drawOval(60,60,200,200);
        g.fillOval(90,120,50,20);
        g.fillOval(190,120,50,20);
        g.drawLine(165,125,165,175);
        g.drawLine(165,175,150,160);
        g.drawArc(110,130,95,95,0,-180);
    }
}
```

（3）显示文字

例题：在 Applet 的正中央显示“欢迎您!”。

```
import java.awt.*;
import java.applet.Applet;
public class FontDemo extends Applet{
    public void paint(Graphics g){
        String str="欢迎您！ ";
        Font f=new Font("黑体",Font.PLAIN,24);
        g.setFont(f);
        FontMetrics fm=getFontMetrics(f);
        int x=(getWidth()-fm.stringWidth(str))/2;
        int y=getHeight()/2;
        g.drawString(str,x,y);
    }
}
```

（4）颜色控制

例题：用随机定义的颜色填充小方块。

```
import java.awt.*;
import java.applet.Applet;
public class Colors extends Applet{
    public void paint(Graphics g){
      int red,green,blue;
      for(int i=10;i<200;i+=40){
           red=(int)Math.floor(Math.random()*256);
          green=(int)Math.floor(Math.random()*256);
          blue=(int)Math.floor(Math.random()*256);
          g.setColor(new Color(red,green,blue));
          g.fillRect(i,20,30,30);
      }
    }
}
```

（5）Java 2D 图形绘制

Graphics2D 类在其父类 Graphics 类的基础上做了扩展，可以对为二维图形进行几何形状控制、坐标变换、颜色管理和文本布置等操作。Java2D 还提供了大量的属性，用于指定颜色、线宽、填充图案、透明度和其他属性。

① Graphics2D 的图形对象

所有 **Graphics2D** 的图形都在 **java.awt.geom** 包中定义。包括线段、矩形、椭圆、弧、多边形等。

② 指定填充图案

用 **setPaint(Paint)**方法指定填充方式，可以使用单色、渐变、纹理或用户自己设计的图案来填充对象区域。

★ **Color:**单色填充

★ **GradientPaint:**渐变填充

★ **TexturePaint:**纹理填充

③ 设置画笔

使用 **setStroke()**方法并用 **BasicStroke** 对象作为参数，可以设置绘制图形线条的宽度和连接形状。

④ 绘制图形

无论什么样的图形对象，都使用相同的 **Graphics2D** 方法：

★ **void fill(Shape s):**绘制一个填充的图形。

★ **void draw(Shape s):**绘制图形的边框。

例题：利用 **Graphics2D** 绘制矩形

```
import java.awt.*;
import java.applet.*;
import java.awt.geom.*;
public class GradientTest extends Applet{
    public void paint(Graphics g){
        Graphics2D g2d=(Graphics2D)g;
        Rectangle2D r=new Rectangle2D.Double(25,20,150,50);
        GradientPaint                                          p=new
GradientPaint(25,20,Color.yellow,300,90,Color.green);
        g2d.setPaint(p);
        g2d.fill(r);
        g2d.setPaint(Color.blue);
        g2d.setStroke(new
BasicStroke(5,BasicStroke.CAP_BUTT,BasicStroke.JOIN_ROUND));
        g2d.draw(r);
    }
}
```

★ 可以使用渐变填充的构造方法：

```
        GradientPaint                                          p=new
GradientPaint(25,20,Color.yellow,30,25,Color.green，true);
```

⑤ 坐标变换

利用 **AffineTransform** 类可以实现图形绘制的各类变换，包括：平移、缩放、旋转等。具体步骤如下：

★ 创建 **AffineTransform** 的对象

★ 设置变换形式

★ 将 **Graphics2D** 对象设置为采用该变换的“画笔”。

★ 绘制变换图形

例题：利用旋转绘制图形

```
import java.awt.*;
import java.applet.*;
import java.awt.geom.*;
public class Rotate extends Applet{
    public void paint(Graphics g){
        Graphics2D g2d=(Graphics2D)g;
        Ellipse2D ellipse=new Ellipse2D.Double(20,50,100,60);
        AffineTransform trans=new AffineTransform();
        for(int k=0;k<=36;k++){
            trans.rotate(10.0*Math.PI/180,80,75);
            g2d.setTransform(trans);
            g2d.draw(ellipse);
        }
    }
}
```

四、Applet 参数传递

（1）在 HTML 文件中给 Applet 提供参数

```
<HTML>
<BODY>
<applet code=My_param.class height=200 width=300>
<param name=vs value=可变大小的字符串>
<param name=size value=24>
</applet>
</BODY>
</HTML>
```

例题：

（2）在 Applet 代码中读取 Applet 参数值

例题：

```
import java.awt.*;
import java.applet.Applet;
public class My_param extends Applet{
    private String s="";
    private int size;
    public void init(){
        s=getParameter("vs");
        size=Integer.parseInt(getParameter("size"));
    }
    public void paint(Graphics g){
        g.setFont(new Font("宋体",Font.PLAIN,size));
        g.drawString(s,30,40);
    }
```

```
}
```

例题：利用 Applet 参数传递绘制图形信息。

```
import java.awt.*;
import java.applet.Applet;
public class ParaDraw extends Applet{
    String graph;
    public void init(){
      graph=getParameter("graph");
    }
    public void paint(Graphics g){
      String[] para;
      int x,y,w,h;
      String[] commands=graph.split("/");
      for(int k=0;k<commands.length;k++){
           para=commands[k].split(",");
          if(para[0].equals("oval")){
              x=Integer.parseInt(para[1]);
             y=Integer.parseInt(para[2]);
             w=Integer.parseInt(para[3]);
             h=Integer.parseInt(para[4]);
             g.drawOval(x,y,w,h);
          }
          else if(para[0].equals("rect")){
              x=Integer.parseInt(para[1]);
             y=Integer.parseInt(para[2]);
             w=Integer.parseInt(para[3]);
             h=Integer.parseInt(para[4]);
             g.drawRect(x,y,w,h);
          }
      }
    }
}
<HTML>
<BODY>
<applet code=ParaDraw.class height=200 width=200>
<param                                                    name=graph
value="rect,10,20,100,110/oval,40,60,50,50/rect,20,30,110,120">
</applet>
</BODY>
</HTML>
```

五、Applet 多媒体支持

（1）绘制图像

Java 对图像的处理包括图像的获取和图像绘制两个环节。

① 图像的获取

public Image getImage(URL,String)：从指定的 **URL** 位置获取某个名称的图像文件。如果 **URL** 中直接包含图像文件名时，也可以使用 **getImage(URL)**方法。

URL 是图像的绝对地址，为了增加程序的通用性，通常使用下面的方法得到图像的地址：

★ getCodeBase()：返回 Applet 字节码文件的 URL 地址。

★ getDocumentBase()：返回 HTML 文件的 URL 地址。

② 图像绘制

可以使用 **Graphics** 类的 **drawImage()**方法：

★ public void drawImage(Image,x,y,imageObserver)

★ public void drawImage(Image,x,y,width,height,imageObserver)

例题：

```
/*
<applet code=DrawMyImage.class height=400 width=400></applet>
*/
import java.awt.*;
import java.applet.*;
public class DrawMyImage extends Applet{
    Image myImage;
    public void init(){
        myImage=getImage(getDocumentBase(),"m.gif");
    }
    public void paint(Graphics g){
         g.drawImage(myImage,0,0,400,400,this);
    }
}
```

③ 利用双缓冲区绘图

由于 **drawImage** 方法绘制图像时是边下载边绘制，所以画面有爬行现象，为了提高显示效果，可以开辟一个内存缓冲区，先将图像绘制到缓冲区，然后再将缓冲区图像绘制到 **Applet** 画面上。

建立图形缓冲区的方法为：

createImage(width, height)

使用 **getGraphihcs()**方法可以得到该图像缓冲区的 **Graphics** 对象。

例题：绘制移动的笑脸。

```
/*
<applet code=Mthread.class height=300 width=300></applet>
*/
import java.awt.*;
import java.applet.Applet;
public class Mthread extends Applet{
    Image img;
    int pos=0;
    public void init(){
```

```
        img=createImage(300,300);
        Graphics gimg=img.getGraphics();
        gimg.drawOval(60,60,100,100);
        gimg.fillOval(75,90,25,10);
        gimg.fillOval(120,90,25,10);
        gimg.drawLine(110,95,110,130);
        gimg.drawArc(85,110,50,40,0,-180);
        gimg.drawLine(110,130,100,120);
    }
    public void paint(Graphics g){
        g.drawImage(img,pos,100,this);
        pos=++pos%200;
        for(int j=0;j<9900000;j++);
        repaint();
    }
}
```

（2）实现动画

例题：

```
/*
<applet code=ShowAnimator.class height=300 width=500></applet>
*/
import java.awt.*;
import java.applet.Applet;
public class ShowAnimator extends Applet{
    Image[] m_Images;
    int totalImages=10;
    int currentImage=0;
    public void init(){
        m_Images=new Image[totalImages];
        for(int i=0;i<totalImages;i++)

m_Images[i]=getImage(getDocumentBase(),"images\\img00"+(i+1)+".gif");
    }
    public void start(){
        currentImage=0;
    }
    public void paint(Graphics g){
        g.drawImage(m_Images[currentImage],50,50,this);
        currentImage=++currentImage%totalImages;
        try{
            Thread.sleep(200);
        }
        catch(InterruptedException e){}
        repaint();
```

```
    }
}
```

（3）播放声音文件

① 利用 Applet 类的 play()方法直接播放

② 使用 AutoClip 接口

例题：

```
/*
<applet code=SoundA.class height=300 width=500></applet>
*/
import java.applet.*;
public class SoundA extends Applet{
    AudioClip ac;
    public void init(){
        ac=getAudioClip(getCodeBase(),"sloop.wav");
    }
    public void start(){
       ac.loop();
    }
    public void stop(){
       ac.stop();
    }
}
```

六、Java 存档文件

（一）创建存档文件

（1）普通 JAR 文件的打包

例题：把下面的程序编译后压缩成 TestJar.jar 文件。

```
public class TestJar {
     public void f(){
         System.out.println("i am f() in TestJar");
    }
}
class Test1{
    public void g(){
       System.out.println("i am g() in Test1");
    }
}
```

命令：jar cvf TestJar.jar TestJar.class Test1.class

（2）制作可执行的 JAR 文件

例题：把下面程序制作成可执行的 JAR 文件。

```
import javax.swing.*;
import java.awt.*;
public class TestFrame extends JFrame{
```

```
    JButton b;
    public TestFrame(){
       setLayout(new FlowLayout());
       b=new JButton("确定");
       add(b);
       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       setSize(400,400);
       setVisible(true);
    }
    public static void main(String[] args){
       new TestFrame();
    }
}
```

步骤：

① 编译 **TestFrame.java**

② 编写清单文件 **m.mf**，清单文件的内容如下：

Main-Class: TestFrame

★ ：和 **TestFrame** 之间要有空格，**TestFrame** 后面要有回车键

③ 利用命令：**jar cvfm TestFrame.jar m.mf TestFrame.class** 制作可执行文件。

④运行可执行文件 **TestFrame.jar**。

双击运行或使用命令：

java -jar TestFrame.jar

（二）在 **HTML** 文件中指定 **Applet** 的存档文件

（1）在**<APPLET>**标记中指定 **JAR** 文件

例题：

```
<applet code=MyClass.class archive=resource.jar height=300 width=300>
</applet>
```

（2）在**<OBJECT>**标记中指定 **JAR** 文件

例题：

```
<object code=MyClass.class height=300 width=300>
<param name=archive value=resource.jar>
</object>
```

第七章 图形用户界面编辑

一、图形用户界面的核心概念

（1）一个简单的 GUI 示例。

例题：将二进制数据转换为十进制

```
import java.awt.*;
import java.awt.event.*;
public class ConvertToDec extends Frame implements ActionListener{
    Label dec;
    TextField input;
    public ConvertToDec(){
        super("binary to decimal");
        dec=new Label("......结果......");
        input=new TextField(15);
        Button convert=new Button("转换");
        setLayout(new FlowLayout());
        add(input);
        add(convert);
        add(dec);
        convert.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e){
        String s=input.getText();
        int x=Integer.parseInt(s,2);
        dec.setText("result="+x);
    }
    public static void main(String[] args){
        Frame x=new ConvertToDec();
        x.setSize(400,100);
        x.setVisible(true);
    }
}
```

（2）创建窗体

Frame 的创建大致有两种方式：

① 通过继承 Frame 来创建窗体。

② 直接由 Frame 类创建。

（3）创建 GUI 部件

由 add()方法把 GUI 布局加入到容器中。

（4）事件处理

① 事件处理的流程

★ 给事件源对象注册监听者

★ 给监听者编写事件处理代码

★ 发生事件时调用监听者的方法进行相关处理

② 事件监听者接口及其方法

Java 中的所有事件类都定义在 **java.awt.event** 包中，该包中还定义了 **11** 个监听者接口，每个接口内部包含了若干处理相关事件的抽象方法。

见 **P158**：表 **7-1**

（**5**）在事件处理代码中区分事件源

一个事件源对象可以注册多个监听者，一个监听者也可以监视多个事件源。不同类型的事件提供了不同的方法来区分事件源对象。如 **ActionEvent** 类中提供了两个方法：

① **getSource()**：用来获取事件对象名。

② **getActionCommand()**：用来获取事件对象的命令名。

例题：有两个按钮，点击按钮 **b1** 画圆，点击按钮 **b2** 画矩形。

```
import java.awt.*;
import java.awt.event.*;
public class TwoButton extends Panel implements ActionListener{
    Button b1,b2;
    Panel draw;
    public TwoButton(Panel draw){
        this.draw=draw;
        b1=new Button("circle");
        b2=new Button("rectangle");
        add(b1);
        add(b2);
        b1.addActionListener(this);
        b2.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e){
        Graphics g=draw.getGraphics();
        g.setColor(draw.getBackground());
        g.fillRect(0,0,draw.getSize().width,draw.getSize().height);
        g.setColor(Color.blue);
        String label=e.getActionCommand();
        if(label.equals("circle"))
            g.drawOval(20,20,50,50);
        else
            g.drawRect(20,20,40,60);
    }
    public static void main(String[] args){
        Frame f=new Frame("to Button event Test");
        Panel draw=new Panel();
```

```
            TwoButton two=new TwoButton(draw);
            f.setLayout(new BorderLayout());
            f.add("North",two);
            f.add("Center",draw);
            f.setSize(300,300);
            f.setVisible(true);
        }
    }
```

（6）关于事件适配器类

例题：处理窗体关闭

```
    /*
    <applet code=TestFrame.class height=500 width=500></applet>
    */
    import java.awt.*;
    import java.awt.event.*;
    import java.applet.*;
    public class TestFrame extends Applet{
        public void init(){
            new MyFrame();
        }
    }
    class MyFrame extends Frame implements ActionListener{
        Button btn;
        MyFrame(){
          super("MY WINDOWS");
          btn=new Button("关闭");
          setLayout(new FlowLayout());
          add(btn);
          btn.addActionListener(this);
          addWindowListener(new CloseWin());
          setSize(300,200);
          setVisible(true);
        }
        public void actionPerformed(ActionEvent e){
          if(e.getActionCommand()=="关闭"){
             dispose();
          }
        }
    };
    class CloseWin extends WindowAdapter{
        public void windowClosing(WindowEvent e){
          Window w=e.getWindow();
```

```
            w.dispose();
          }
      };
```

二、容器与布局管理

Java.awt 包中定义了 5 种布局管理器：

（1）FlowLayout：流式布局

例题：

```
import java.awt.*;
import java.applet.Applet;
public class FlowLayoutExample extends Applet{
    public void init(){
      setLayout(new FlowLayout(FlowLayout.LEFT,10,10));
      String spaces="";
      for(int i=1;i<=9;i++){
          add(new Button("B #"+i+spaces));
         spaces+=" ";
      }
    }
    public static void main(String[] args){
      Frame x=new Frame("FlowLayout");
      FlowLayoutExample y=new FlowLayoutExample();
      x.add(y);
      y.init();
      x.setSize(200,100);
      x.setVisible(true);
    }
}
```

（2）BorderLayout：边缘布局

例题：

```
import java.awt.*;
import java.applet.Applet;
public class BorderLayoutExample extends Applet{
    String[] borders={"North","East","South","West","Center"};
    public void init(){
      setLayout(new BorderLayout(10,10));
      String spaces="";
      for(int i=0;i<5;i++){
          add(borders[i],new Button(borders[i]));
      }
    }
    public static void main(String[] args){
      Frame x=new Frame("BorderLayout");
```

```
        BorderLayoutExample y=new BorderLayoutExample();
        x.add(y);
        y.init();
        x.setSize(300,200);
        x.setVisible(true);
    }
}
```

（3）GridLayout：网格布局

例题：

```
import java.awt.*;
import java.applet.Applet;
public class GridLayoutExample extends Applet{
    public void init(){
        setLayout(new GridLayout(3,3,10,10));
        for(int i=1;i<=9;i++){
            add(new Button("Button #"+i));
        }
    }
    public static void main(String[] args){
        Frame x=new Frame("GridLayout");
        GridLayoutExample y=new GridLayoutExample();
        x.add(y);
        y.init();
        x.setSize(300,200);
        x.setVisible(true);
    }
}
```

（4）CardLayout：卡片布局

例题：

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class CardLayoutExample extends Applet{
    public void init(){
        final CardLayout cardlayout=new CardLayout(10,10);
        setLayout(cardlayout);
        ActionListener listener=new ActionListener(){
            public void actionPerformed(ActionEvent e){
                cardlayout.next(CardLayoutExample.this);
            }
        };
        for(int i=1;i<=9;i++){
            Button b=new Button("Button #"+i);
            b.addActionListener(listener);
```

```
            add("Button"+i,b);
          }
      }
      public static void main(String[] args){
         Frame x=new Frame("CardLayout");
         CardLayoutExample y=new CardLayoutExample();
         x.add(y);
         y.init();
         x.setSize(300,200);
         x.setVisible(true);
      }
}
```

（5）GridBagLayout：网格快布局

例题：

```
import java.awt.*;
public class Test extends Frame{
      public static void main(String[] args){
         new Test();
      }
      public Test(){
         Label receiveLabel=new Label("收件人： ");
         Label ccLabel=new Label("抄送： ");
         Label subjectLabel=new Label("主题： ");
         Label accessoryLabel=new Label("附件： ");
         TextField receiveField=new TextField();
         TextField ccField=new TextField();
         TextField subjectField=new TextField();
         TextArea accessoryArea=new TextArea(1,40);
         TextArea mailArea=new TextArea(8,40);
          setLayout(new GridBagLayout());
         GridBagConstraints gridBag=new GridBagConstraints();
         gridBag.fill=GridBagConstraints.HORIZONTAL;
         gridBag.weightx=0;
         gridBag.weighty=0;
         receiveLabel.setFont(new Font("Alias",Font.BOLD,16));
         addToBag(receiveLabel,gridBag,0,0,1,1);
         ccLabel.setFont(new Font("Alias",Font.BOLD,16));
          addToBag(ccLabel,gridBag,0,1,1,1);
          subjectLabel.setFont(new Font("Alias",Font.BOLD,16));
         addToBag(subjectLabel,gridBag,0,2,1,1);
          accessoryLabel.setFont(new Font("Alias",Font.BOLD,16));
         addToBag(accessoryLabel,gridBag,0,3,1,1);
          gridBag.weightx=100;
         gridBag.weighty=0;
```

```
        addToBag(receiveField,gridBag,1,0,2,1);
        addToBag(ccField,gridBag,1,1,2,1);
        addToBag(subjectField,gridBag,1,2,2,1);
        accessoryArea.setEditable(false);
        addToBag(accessoryArea,gridBag,0,3,2,1);
        gridBag.fill=GridBagConstraints.BOTH;
        gridBag.weightx=100;
        gridBag.weighty=100;
        addToBag(mailArea,gridBag,0,4,3,1);
        setSize(300,300);
        setVisible(true);
    }
    void addToBag(Component c,GridBagConstraints gbc,int x,int y,int w,int h){
        gbc.gridx=x;
        gbc.gridy=y;
        gbc.gridheight=h;
        gbc.gridwidth=w;
        add(c,gbc);
    }
}
```

（6）BoxLayout：在 javax.swing.border 包中。

例题：

```
import javax.swing.*;
import java.awt.*;
import javax.swing.border.*;
public class E716{
    public static void main(String args[]){
        new WindowBox();
    }
}
class WindowBox extends Frame{
    Box baseBox ,boxV1,boxV2;
    WindowBox(){
        boxV1=Box.createVerticalBox();
        boxV1.add(new Label("姓名"));
        boxV1.add(Box.createVerticalStrut(8));
        boxV1.add(new Label("email"));
        boxV1.add(Box.createVerticalStrut(8));
        boxV1.add(new Label("职业"));
        boxV2=Box.createVerticalBox();
        boxV2.add(new TextField(12));
        boxV2.add(Box.createVerticalStrut(8));
        boxV2.add(new TextField(12));
        boxV2.add(Box.createVerticalStrut(8));
```

```
        boxV2.add(new TextField(12));
        baseBox=Box.createHorizontalBox();
        baseBox.add(boxV1);
        baseBox.add(Box.createHorizontalStrut(10));
        baseBox.add(boxV2);
        setLayout(new FlowLayout());
        add(baseBox);
        setBounds(120,125,250,150);
        setVisible(true);
     }
}
```

（7）null

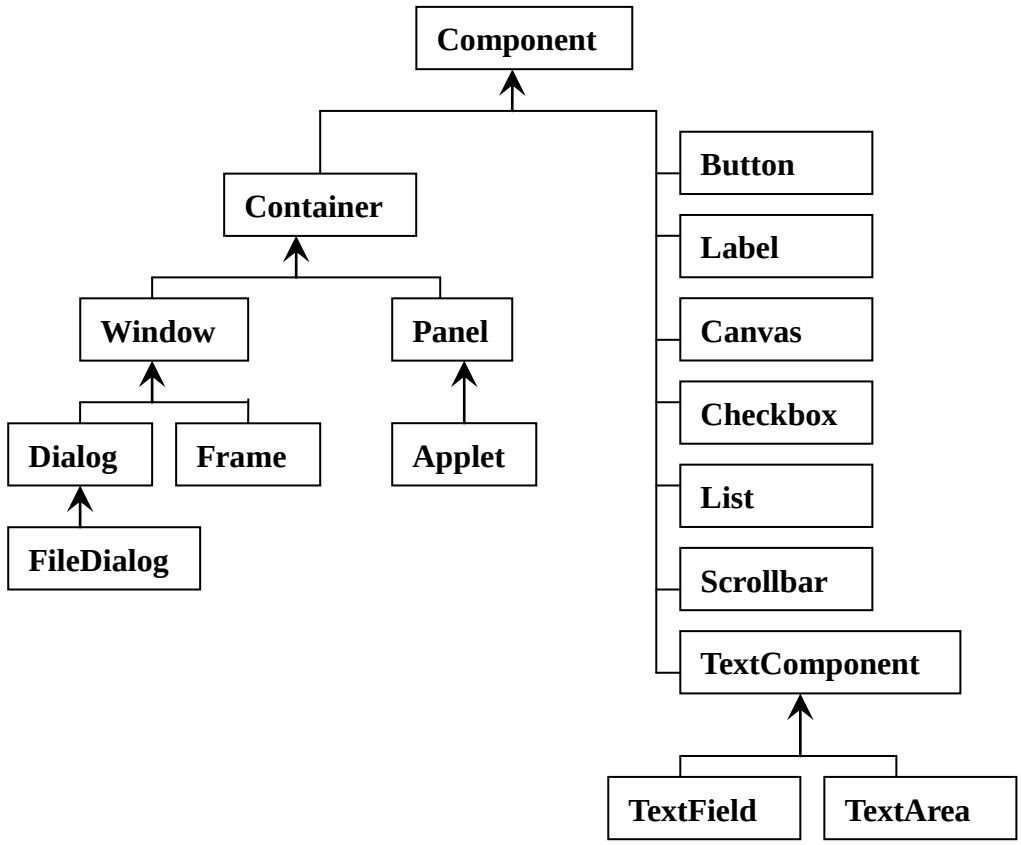
例题：

```
import java.awt.*;
import java.awt.event.*;
class WindowNull extends Frame{
     WindowNull(){
        setLayout(null);
         MyButton button=new MyButton();
         Panel p=new Panel();
         p.setLayout(null);
         p.setBackground(Color.cyan);
         p.add(button);
         button.setBounds(20,10,25,70);
         add(p);
         p.setBounds(50,50,90,100);
         setBounds(120,125,200,200);
         setVisible(true);
    }
}
class MyButton    extends Button implements ActionListener{
    int n=-1;
    MyButton(){
         addActionListener(this);
    }
    public void paint(Graphics g){
        g.drawString("单",6,16);
         g.drawString("击",6,36);
         g.drawString("我",6,56);
    }
    public void actionPerformed(ActionEvent e){
         n=(n+1)%3;
         if(n==0)
             setBackground(Color.red);
```

```
        else if(n==1)
            setBackground(Color.yellow);
        else if(n==2)
            setBackground(Color.green);
    }
}
public class E717{
    public static void main(String args[]){
        new WindowNull();
    }
}
```

三、常用 GUI 标准组件

（1）GUI 标准组件概述



基本 GUI 部件被加入到容器中，用来完成一种具体的与用户交互的功能。其步骤为：

① 创建某种基本控制部件类的对象，指定该对象的属性，如外观、大小等。

② 将该对象加入到某个容器的合适位置。

③ 为该对象注册事件监听者。

（2）文本框与文本域

① 文本框
构造方法：
· TextField()
· TextField(int)
· TextField(String)
· TextField(String,int)
设置回显字符：
例题：
TextField pass=new TextField(8);
pass.setEchoChar(‘*’);
② 文本域
构造方法：
· TextArea()
· TextArea(int,int)
· TextArea(String)
· TextArea(String,int,int)
③ 常用方法
数据的写入与读取：
· getText()
· setText()
· isEditable()
指定和获取文本域中“选定状态”文本：
· select(int start,int end)
· selectAll()
· setSelectionStart()
· setSelectionEnd()
· getSelectionStart()
· getSelectionEnd()
· getSelectedText()
屏蔽回显：
· setEchoChar(char c)
· echoCharIsSet()
· getEchoChar()
· 添加数据：
· append(String s)
· insert(String s,int pos)
④ 事件响应
· 文本框中按回车键时产生 ActionEvent 事件：
事件：ActionEvent
处理事件的接口：ActionListener()
处理事件的方法：actionPerformed(ActionEvent e)
· 文本框和文本域进行任何数据的更改时，产生 TextEvent 事件。
事件：TextEvent
处理事件的接口：TextListener()

处理事件的方法：textValueChanged(TextEvent e)

例题：7-10

```
/*
<applet code=InsertNewLine.class width=500 height=500></applet>
*/
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class InsertNewLine extends Applet implements ActionListener{
    TextField input;
    TextArea display;
    final static int lineSize=20;
    public void init(){
       input=new TextField(45);
       display=new TextArea(10,45);
       add(input);
       add(display);
       input.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e){
       String s=input.getText();
       String r=convert(s);
       display.setText(r);
    }
    private String convert(String str){
       String result="";
       int count=0;
       for(int i=0;i<str.length();i++){
          int c=str.charAt(i);
          if(c>=32&&c<=126)
               count+=1;
          else
               count+=2;
          result=result+(char)c;
          if(count>=lineSize){
             result+="\n";
             count=0;
          }
       }
       return result;
    }
}
```

（3）选项按钮与列表框的使用

① 选择事件类：ItemEvent

· public ItemSelectable getItemSelectable()：返回引发事件的事件源。

· public Object getItem()：返回引发事件的具体选项。

· public int getStateChange()：返回具体的选中状态变化类型，它的返回值在 ItemEvent 的几个静态常量列举之内：

ItemEvent.SELECTED：代表选项被选中。

ItemEvent.DESELECTED：代表选项被放弃不选。

② 复选按钮：Checkbox

创建：

Checkbox backg=new Checkbox(“背景色”);

常用方法：

· boolean getState()

· void setState(Boolean value)

事件响应：

事件：ItemEvent

处理事件的接口：ItemListener()

处理事件的方法： itemStateChanged(ItemEvent e)

例题：7-11

```
/*
<applet code=FuXuan.class width=500 height=500></applet>
*/

import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class FuXuan extends Applet implements ActionListener{
   String[] question={"Java test question1\n A.choice1\n B.choice2\n C.choice3","Java test question2\n A.good\n B.bad\n C.luck"};
   String[] ch={"A","B","C"};
   String[] answer={"AB","BC"};
   Checkbox[] cb=new Checkbox[3];
   Label hint;
   TextArea content;
   Button ok;
   int bh=0;
   Button next;
   Button previous;
   public void init(){
     setLayout(new BorderLayout());
     content=new TextArea(10,50);
     add("Center",content);
     content.setText(question[bh]);
     Panel p=new Panel();
```

```
  p.setLayout(new GridLayout(2,1));
  Panel p1=new Panel();
  for(int i=0;i<ch.length;i++){
    cb[i]=new Checkbox(ch[i]);
   p1.add(cb[i]);
  }
  p.add(p1);
  Panel p2=new Panel();
  ok=new Button("确定");
  p2.add(ok);
  hint=new Label("对错提示");
  p2.add(hint);
  next=new Button("下一题");
  p2.add(next);
  previous=new Button("上一题");
  p2.add(previous);
  p.add(p2);
  add("South",p);
  next.addActionListener(this);
   previous.addActionListener(this);
  ok.addActionListener(this);
}
public void actionPerformed(ActionEvent e){
  if(e.getSource()==ok){
    String s="";
   for(int i=0;i<ch.length;i++)
       if(cb[i].getState())
          s=s+cb[i].getLabel();
   if(s.equals(answer[bh]))
       hint.setText("对");
   else
       hint.setText("错");
  }
  else if(e.getSource()==next){
    if(bh<question.length-1)
       bh++;
   content.setText(question[bh]);
  }
  else{
    if(bh>0)
       bh--;
   content.setText(question[bh]);
  }
}
```

```
}
```

③ 单选按钮：CheckBoxGroup

例题：7-12

```
/*
<applet code=ChangeColor.class width=500 height=500></applet>
*/
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class ChangeColor extends Applet implements ItemListener{
   String[] des={"红色","蓝色","绿色","白色","灰色"};
   Color[]
c={Color.red,Color.blue,Color.green,Color.white,Color.gray};
   Color drawColor=Color.black;
   public void init(){
     CheckboxGroup style=new CheckboxGroup();
     for(int i=0;i<des.length;i++){
        Checkbox one=new Checkbox(des[i],false,style);
        one.addItemListener(this);
        add(one);
     }
   }
   public void paint(Graphics g){
     g.setColor(drawColor);
     g.setFont(new Font("变色字",Font.BOLD,24));
     g.drawString("变色字",80,80);
   }
   public void itemStateChanged(ItemEvent e){
     Checkbox temp=(Checkbox)e.getItemSelectable();
     for(int i=0;i<des.length;i++){
       if(temp.getLabel()==des[i]){
         drawColor=c[i];
         repaint();
         break;
       }
     }
   }
}
```

④ 下拉列表：Choice

例题：7-13

```
/*
<applet code=ChangeColor2.class width=500 height=500></applet>
*/
```

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class ChangeColor2 extends Applet implements ItemListener{
   String[] des={"红色","蓝色","绿色","白色","灰色"};
   Color[]
c={Color.red,Color.blue,Color.green,Color.white,Color.gray};
   public void init(){
     Choice color=new Choice();
     for(int i=0;i<des.length;i++){
       color.add(des[i]);
     }
     color.addItemListener(this);
     add(color);
   }
   public void itemStateChanged(ItemEvent e){
     Choice temp=(Choice)e.getItemSelectable();
     int k=temp.getSelectedIndex();
     setBackground(c[k]);
   }
}
```

⑤ 列表：List

列表 List 与下拉列表 Choice 的区别：

· 列表可以在屏幕上看到一定数目的选择项，而下拉列表只能看到一项。

· 用户可以选中列表中的多项，而下拉列表只能被选中一项。

★ 列表可产生 2 类事件：

· ItemEvent 类选择事件：当单击某选项时触发。

· ActionEvent 类动作事件：当双击选项时触发。

例题：7-14

```
/*
<applet code=TestList.class width=500 height=500></applet>
*/
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class TestList extends Applet implements ActionListener,ItemListener{
    List myList;
    Label result;
    String[] unit={"总务处","教务处","工会","科研处","信息学院","机械学院
"};
    public void init(){
        myList=new List(5,true);
        for(int i=0;i<unit.length;i++)
            myList.add(unit[i]);
```

```
        add(myList);
        myList.addActionListener(this);
        myList.addItemListener(this);
    }
    public void actionPerformed(ActionEvent e){
        if(e.getSource()==myList)
            showStatus("您双击了选项"+e.getActionCommand());
    }
    public void itemStateChanged(ItemEvent e){
        String[] sList;
        String str="";
        List temp=(List)(e.getItemSelectable());
        sList=temp.getSelectedItems();
        for(int i=0;i<sList.length;i++)
            str=str+sList[i]+" ";
        showStatus("您选择了选项："+str);
    }
}
```

（4）滚动条的使用：Scrollbar

例如：

```
Scrollbar mySlider=new Scrollbar(Scrollbar.HORIZONTAL,0,1,0,255);
```

· 第 1 个参数为常量，代表水平滚动条。

· 第 2 个参数代表初始值。

· 第 3 个参数代表滚动条的滑块长度。

· 第 4、5 个参数跟别代表滚动条的最小值和最大值。

★AdjustmentEvent 事件：

事件：AdjustmentEvent

处理事件的接口：AdjustmentListener

处理事件的函数：adjustmentValueChanged(AdjustmentEvent e)

例题:7-15

```
/*
<applet code=TestSlider.class width=500 height=500></applet>
*/
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
class ColorBar extends Scrollbar{
    public ColorBar(){
      super(Scrollbar.HORIZONTAL,0,40,0,295);
      this.setUnitIncrement(1);
      this.setBlockIncrement(50);
    }
};
```

```
class ColorPanel extends Panel implements AdjustmentListener{
    Scrollbar redSlider=new ColorBar();
    Scrollbar greenSlider=new ColorBar();
    Scrollbar blueSlider=new ColorBar();
    Canvas mycanvas=new Canvas();
    Color color;
    public ColorPanel(){
       Panel x=new Panel();
       x.setLayout(new GridLayout(3,2,1,1));
       x.add(new Label("red"));
       x.add(redSlider);
       x.add(new Label("green"));
       x.add(greenSlider);
       x.add(new Label("blue"));
       x.add(blueSlider);
       setLayout(new GridLayout(2,1,5,5));
       add(mycanvas);
       add(x);
       redSlider.addAdjustmentListener(this);
       greenSlider.addAdjustmentListener(this);
       blueSlider.addAdjustmentListener(this);
    }
    public void adjustmentValueChanged(AdjustmentEvent e){
       int value1,value2,value3;
       value1=redSlider.getValue();
       value2=greenSlider.getValue();
       value3=blueSlider.getValue();
       color=new Color(value1,value2,value3);
       mycanvas.setBackground(color);
    }
};
public class TestSlider extends Applet{
    public void init(){
       setLayout(new BorderLayout());
       add("Center",new ColorPanel());
    }
}
```

四、鼠标和键盘事件

（1）鼠标事件

① 鼠标事件类

鼠标事件共有 7 种情形，用 MouseEvent 类中的静态整形常量表示。

- 鼠标事件
  - 按下鼠标按键：**MOUSE_PRESSED**
  - 释放鼠标按键：**MOUSE_RELEASED**
  - 单击鼠标按键（按下并释放）：**MOUSE_CLICKED**
  - 鼠标光标进入组件几何形状的未遮掩部分：**MOUSE_ENTERED**
  - 鼠标光标离开组件几何形状的未遮掩部分：**MOUSE_EXITED**
- 鼠标移动事件
  - 移动鼠标：**MOUSE_MOVED**
  - 拖动鼠标：**MOUSE_DRAGGED**
- **MouseEvent** 类的主要方法：
  - **public int getX()**：返回发生鼠标事件的 **x** 坐标。
  - **public int getY()**：返回发生鼠标事件的 **y** 坐标。
  - **public Point getPoint()**：返回 **Point** 对象，即鼠标事件发生的坐标点。
  - **public int getClickCount()**：返回鼠标点击事件的连击次数。
- 处理鼠标事件的接口：

  MouseListener 接口：
  - 注册监视器的方法：**addMouseListener()**
  - 接口中的方法：
    - ✧**mouseClicked(MouseEvent e)**： 鼠标按键在组件上单击（按下并释放）时调用。
    - ✧**mouseEntered(MouseEvent e)**：鼠标进入到组件上时调用。
    - ✧**mouseExited(MouseEvent e)**： 鼠标离开组件时调用。
    - ✧**mousePressed(MouseEvent e)**：鼠标按键在组件上按下时调用。
    - ✧**mouseReleased(MouseEvent e)**：鼠标按钮在组件上释放时调用

  MouseMotionListener 接口：
  - 注册监视器的方法：**addMouseMotionListener()**
  - 接口中的方法：
    - ✧**mouseDragged(MouseEvent e)**：鼠标按键在组件上按下并拖动时调用。
    - ✧**mouseMoved(MouseEvent e)**：鼠标光标移动到组件上但无按键按下时调用。

例题：7-16 一个鼠标跟踪程序，在当前位置画红色小圆。

```
/*
<applet code=TraceMouse.class height=400 width=400></applet>
*/
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class TraceMouse extends Applet implements MouseMotionListener {
    private static final int RADIUS=7;
    private int posx=10,posy=10;
    public void init(){
```

```
        addMouseMotionListener(this);
    }
    public void paint(Graphics g){
        g.setColor(Color.red);
        g.fillOval(posx-RADIUS,posy-RADIUS,RADIUS*2,RADIUS*2);
    }
    public void mouseMoved(MouseEvent event){
        posx=event.getX();
        posy=event.getY();
        repaint();
    }
    public void mouseDragged(MouseEvent event){}
}
```

★ 频繁调用 **repaint()**方法会出现闪烁现象。**repaint()**方法会调用 **update(g)**方法，**update(g)**先清除画面上的内容再调用 **paint()**方法。所以产生闪烁现象。

★ 改进：改写 **update(g)**方法，让其直接调用 **paint()**方法，使用利用背景颜色填充矩形的办法清除 **applet** 屏幕。创建双缓冲区来提高显示效果。

例题：修改例题 **7-16**

```
/*
<applet code=TraceMouse2.class height=400 width=400></applet>
*/
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class TraceMouse2 extends Applet implements MouseMotionListener
{
    private static final int RADIUS=7;
    private int posx=10,posy=10;
    protected Image offscreen;
    protected Graphics offg;
    public void init(){
        addMouseMotionListener(this);
        offscreen=createImage(getSize().width,getSize().height);
        offg=offscreen.getGraphics();
    }
    public void paint(Graphics g){
        offg.setColor(Color.white);
        offg.fillRect(0,0,getSize().width-1,getSize().height-1);
        offg.setColor(Color.black);
         offg.drawRect(0,0,getSize().width-1,getSize().height-1);
        offg.setColor(Color.red);
        offg.fillOval(posx-RADIUS,posy-RADIUS,RADIUS*2,RADIUS*2);
```

```
        g.drawImage(offscreen,0,0,this);
    }
    public void update(Graphics g){
        paint(g);
    }
    public void mouseMoved(MouseEvent event){
        posx=event.getX();
        posy=event.getY();
        repaint();
    }
    public void mouseDragged(MouseEvent event){}
}
```

② 高级语义事件和低级语义事件

例题：7-17

```
/*
<applet code=MouseLevel.class width=700 height=700></applet>
*/
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class    MouseLevel extends Applet{
    Button btn;
    int pos=0;
    public void init(){
        btn=new Button("click me");
        add(btn);
        btn.addMouseListener(new MouseAdapter(){
            public void mouseClicked(MouseEvent e){
                Graphics g=getGraphics();
                pos+=40;
                g.drawString("lower level",pos,50);
            }
        });
        btn.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                Graphics g=getGraphics();
                pos+=40;
                g.drawString("high level",pos,80);
            }
        });
    }
}
```

（2）键盘事件

★使用 KeyListener 接口处理键盘事件

接口中的方法：

public void keyPressed(KeyEvent e)

public void keyTyped(KeyEvent e)

public void keyReleased(KeyEvent e)

★ 注册监视器的方法：

addKeyListener()

★ **KeyEvent** 类的方法：

public int getKeyCode()：返回键盘码。

public char getKeyChar()：返回键盘上的字符。

例题：**7-18** 可变小方框的移动，通过键盘的方向键控制小方框的移动，通过字母键 **B**、**G**、**R**、**K** 更改小方框的颜色

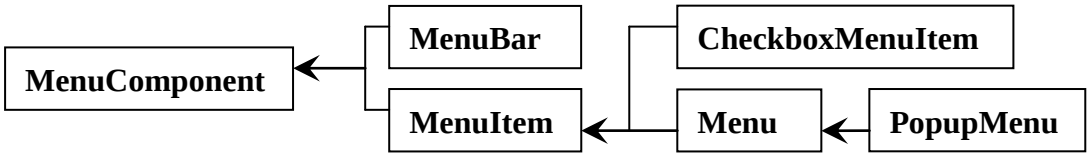
```
/*
<applet code=KeyboardDemo.class width=500 height=500></applet>
*/
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class KeyboardDemo extends Applet implements KeyListener{
    static final int SQUARE_SIZE=20;
    Color squareColor;
    int squareTop,squareLeft;
    public void init(){
       setBackground(Color.white);
       squareTop=getSize().height/2-SQUARE_SIZE/2;
        squareLeft=getSize().width/2-SQUARE_SIZE/2;
       squareColor=Color.red;
       addKeyListener(this);
    }
    public void paint(Graphics g){
       g.setColor(Color.cyan);
       int width=getSize().width;
       int height=getSize().height;
       g.drawRect(0,0,width-1,height-1);
       g.drawRect(1,1,width-3,height-3);
       g.drawRect(2,2,width-5,height-5);
       g.setColor(squareColor);
       g.fillRect(squareLeft,squareTop,SQUARE_SIZE,SQUARE_SIZE);
    }
    public void keyTyped(KeyEvent evt){
       char ch=evt.getKeyChar();
       if(ch=='B'||ch=='b'){
          squareColor=Color.blue;
         repaint();
```

```
        }
        else if(ch=='G'||ch=='g'){
            squareColor=Color.green;
          repaint();
        }
         else if(ch=='R'||ch=='r'){
        squareColor=Color.red;
          repaint();
        }
        else if(ch=='K'||ch=='k'){
        squareColor=Color.black;
          repaint();
        }
    }
    public void keyPressed(KeyEvent evt){
        int key=evt.getKeyCode();
        if(key==KeyEvent.VK_LEFT){
            squareLeft-=8;
          if(squareLeft<3)
                squareLeft=3;
          repaint();
        }
        else if(key==KeyEvent.VK_RIGHT){
            squareLeft+=8;
          if(squareLeft>getSize().width-3-SQUARE_SIZE)
                squareLeft=getSize().width-3-SQUARE_SIZE;
          repaint();
        }
        else if(key==KeyEvent.VK_UP){
            squareTop-=8;
          if(squareTop<3)
                squareTop=3;
          repaint();
        }
        else if(key==KeyEvent.VK_DOWN){
            squareTop+=8;
          if(squareTop>getSize().height-3-SQUARE_SIZE)
                squareTop=getSize().height-3-SQUARE_SIZE;
          repaint();
        }
    }
    public void keyReleased(KeyEvent evt){}
}
```

五、菜单的使用

（1）下拉菜单

AWT 菜单的继承层次图：



菜单编程的基本步骤：

① 创建菜单条（MenuBar）；

② 创建不同的菜单（Menu）并加入到菜单条中；

③ 创建不同的菜单项（MenuItem）并加入到菜单中；

④ 给窗体设定菜单条；

⑤ 各菜单项注册监视器；

⑥ 在 actionPerformed(ActionEvent e)方法中编写事件处理代码。

例题：7-19 一个简单的菜单

```
import java.awt.*;
import java.awt.event.*;
public class MenuDemo extends Frame implements ActionListener{
    TextArea ta;
    public MenuDemo(){
       ta=new TextArea(30,60);
       add(ta);
       MenuBar menubar=new MenuBar();
       setMenuBar(menubar);
       Menu file=new Menu("File");
       menubar.add(file);
       MenuItem open=new MenuItem("open",new
MenuShortcut(KeyEvent.VK_O));
       MenuItem quit=new MenuItem("Quit",new
MenuShortcut(KeyEvent.VK_Q));
       file.add(open);
       file.addSeparator();
       file.add(quit);
       addWindowListener(new WindowAdapter(){
          public void windowClosing(WindowEvent e){
             Frame frm=(Frame)(e.getSource());
            frm.dispose();
          }
```

```
            });
            open.addActionListener(this);
            quit.addActionListener(this);
            setSize(300,300);
            setVisible(true);
        }
        public void actionPerformed(ActionEvent e){
            if(e.getActionCommand()=="Quit"){
                dispose();
            }
            else{
                ta.setText("read a file content and write here");
            }
        }
        public static void main(String[] args){
            new MenuDemo();
        }
    }
```

例题：7-20 多级菜单

```
    import java.awt.*;
    public class MultiLevelMenu{
        public static void main(String[] args){
            Frame f=new Frame("多级菜单演示");
            MenuBar mb=new MenuBar();
            Menu m1=new Menu("File");
            Menu m2=new Menu("View");
            MenuItem m21=new MenuItem("Screen Font");
            Menu m22=new Menu("Print Font");
            CheckboxMenuItem m221=new CheckboxMenuItem("Mirror
Screen Font");
            MenuItem m222=new MenuItem("Set Printer Font");
            f.setMenuBar(mb);
            mb.add(m1);
            mb.add(m2);
            m2.add(m21);
            m2.add(m22);
            m22.add(m221);
            m22.add(m222);
            f.setSize(400,400);
            f.setVisible(true);
        }
    }
```

（2）弹出式菜单

弹出式菜单的编程要点：

① 创建弹出式菜单（**PopupMenu**）对象；

② 为弹出式菜单加入菜单项；

③ 组件用 **add** 方法将弹出式菜单附着在组件上，这样在该部件上点击鼠标右键会显示弹出式菜单；

④ 在该组件或容器上注册鼠标事件的监视器；

⑤ 重写 **processMouseEvent(MouseEvent e)**方法判断是否触发弹出式菜单，如果是点击了鼠标右键，则调用 **show** 方法把菜单显示到产生鼠标事件处。

例题：**7-21** 设计一个画图程序，可以通过弹出式菜单选择画笔的颜色，通过鼠标拖动画线。

```
/*
<applet code=MenuScribble.class width=500 height=500></applet>
*/
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class MenuScribble extends Applet implements ActionListener{
    protected int lastx,lasty;
    protected Color color=Color.black;
    protected PopupMenu popup;
    public void init(){
      popup=new PopupMenu("Color");
      String[] labels={"Clear","Red","Green","Blue","Black"};
      for(int i=0;i<labels.length;i++){
         MenuItem mi=new MenuItem(labels[i]);
        mi.addActionListener(this);
        popup.add(mi);
      }
      this.add(popup);
      this.addMouseListener(new MouseAdapter(){
        public void mousePressed(MouseEvent e){
            lastx=e.getX();
            lasty=e.getY();
        }
      });
      this.addMouseMotionListener(new MouseMotionAdapter(){
          public void mouseDragged(MouseEvent e){
            Graphics g=getGraphics();
            int x=e.getX();
            int y=e.getY();
            g.setColor(color);
            g.drawLine(lastx,lasty,x,y);
            lastx=x;
            lasty=y;
          }
```

```
        });
    }
    public void processMouseEvent(MouseEvent e){
        if((popup!=null)&&e.isPopupTrigger())
            popup.show(this,e.getX(),e.getY());
        else
            super.processMouseEvent(e);
    }
    public void actionPerformed(ActionEvent e){
        String name=((MenuItem)e.getSource()).getLabel();
        if(name.equals("Clear")){
          Graphics g=this.getGraphics();
          g.setColor(this.getBackground());
          g.fillRect(0,0,this.getSize().width,this.getSize().height);
        }
        else if(name.equals("Red"))color=Color.red;
        else if(name.equals("Green"))color=Color.green;
        else if(name.equals("Blue"))color=Color.blue;
        else if(name.equals("Black"))color=Color.black;
    }
}
```

六、对话框（Dialog）的使用

（1）对话框的创建和使用

① 模态对话框

② 非模态对话框

例题：7-22 将例题 7-15 的颜色面板放入一个对话框，给窗体设置背景颜色。

```
import java.awt.*;
import java.awt.event.*;
public class TestDialog extends Frame{
     Button change=new Button("Change Color");
    ColorPanel m;
    Button b=new Button("确定");
    Dialog my;
    public TestDialog(){
        super("Parent");
        setLayout(new FlowLayout());
        m=new ColorPanel();
        add(change);
        change.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                my=new Dialog(TestDialog.this,"Select Color",true);
```

```
                my.addWindowListener(new WindowAdapter(){
                    public void windowClosing(WindowEvent e){
                       my.dispose();
                    }
                });
                my.setLayout(new BorderLayout());
                my.add("Center",m);
                my.add("South",b);
                my.setBounds(100,100,300,200);
                my.setVisible(true);
             }
          });
          setBounds(300,300,300,300);
          setVisible(true);
          b.addActionListener(new ActionListener(){
             public void actionPerformed(ActionEvent e){
               my.dispose();
               setBackground(m.mycanvas.getBackground());
             }
          });
          addWindowListener(new WindowAdapter(){
               public void windowClosing(WindowEvent e){
                  dispose();
               }
          });
      }
      public static void main(String[] args){
           new TestDialog();
      }
}

class ColorBar extends Scrollbar{
      public ColorBar(){
         super(Scrollbar.HORIZONTAL,0,40,0,295);
         this.setUnitIncrement(1);
         this.setBlockIncrement(50);
      }
};

class ColorPanel extends Panel implements AdjustmentListener{
      Scrollbar redSlider=new ColorBar();
      Scrollbar greenSlider=new ColorBar();
      Scrollbar blueSlider=new ColorBar();
      Canvas mycanvas=new Canvas();
```

```
        Color color;
        public ColorPanel(){
           Panel x=new Panel();
           x.setLayout(new GridLayout(3,2,1,1));
           x.add(new Label("red"));
           x.add(redSlider);
           x.add(new Label("green"));
           x.add(greenSlider);
           x.add(new Label("blue"));
           x.add(blueSlider);
           setLayout(new GridLayout(2,1,5,5));
           add(mycanvas);
           add(x);
           redSlider.addAdjustmentListener(this);
           greenSlider.addAdjustmentListener(this);
           blueSlider.addAdjustmentListener(this);
        }
        public void adjustmentValueChanged(AdjustmentEvent e){
           int value1,value2,value3;
           value1=redSlider.getValue();
           value2=greenSlider.getValue();
           value3=blueSlider.getValue();
           color=new Color(value1,value2,value3);
           mycanvas.setBackground(color);
        }
    };
```

（2）文件对话框（FileDialog）

① String getDirectory()：返回对话框选择的文件的目录。

② Sting getFile()：返回对话框选择的文件名，当对话框未选文件时返回 null。

例题：7-23 利用文件对话框获取文件名写入文本框中

```
    import java.awt.*;
    import java.awt.event.*;
    public class TestFileDialog extends Frame{
        Button selectButton=new Button("选择文件");
        TextField file=new TextField(20);
        public TestFileDialog(){
            setLayout(new FlowLayout());
            add(selectButton);
            add(file);
            selectButton.addActionListener(new ActionListener(){
                public void actionPerformed(ActionEvent e){
                    FileDialog my=new FileDialog(TestFileDialog.this);
```

```
                my.setVisible(true);
                file.setText(my.getDirectory()+my.getFile());
            }
        });
        setSize(200,100);
        setVisible(true);
    }
    public static void main(String[] args){
         new TestFileDialog();
    }
}
```

第八章 异常处理

一、异常的概念

（1）什么是异常

异常是指程序运行时出现的非正常情况。可能导致程序发生非正常的情况很多，如：数组下标越界、算数运算被 0 除、空指针访问、试图访问不存在的文件等。

例题：从键盘上输入两个整数，求这两个整数的和。

```
public class Add{
    public static void main(String[] args){
        int a=Integer.parseInt(args[0]);
        int b=Integer.parseInt(args[1]);
        int c=a+b;
        System.out.println("a+b="+c);
    }
}
```

★ 用户输入的数少于 2 个时，修改程序。

修改程序如下：

```
public class Add{
    public static void main(String[] args){
        if (args.length<2){
            System.out.println("输入的整数个数小于 2 个");
        }
        else{
            int a=Integer.parseInt(args[0]);
             int b=Integer.parseInt(args[1]);
             int c=a+b;
             System.out.println("a+b="+c);
        }
    }
}
```

★ 用户输入的数不是整数时，修改程序。

修改程序如下：

```
public class Add{
    public static boolean isInteger(String s){
         boolean b;
          int index=s.indexOf('.');
        if(index==-1)b=true;
        else b=false;
        return b;
    }
    public static void main(String[] args){
```

```
        if (args.length<2)
        {
            System.out.println("输入的整数个数小于 2 个");
        }
        else if(!(isInteger(args[0])&isInteger(args[1]))){
            System.out.println("请输入 2 个整数!!!  ");
        }
        else{
             int a=Integer.parseInt(args[0]);
             int b=Integer.parseInt(args[1]);
             int c=a+b;
             System.out.println("a+b="+c);
        }
    }
}
```

使用 java 中的异常机制：

程序如下：

```
public class Add{
    public static void main(String[] args){
        int a,b,c;
        try{
            a=Integer.parseInt(args[0]);
            b=Integer.parseInt(args[1]);
            c=a+b;
            System.out.println("a+b="+c);
        }
        catch(ArrayIndexOutOfBoundsException e){
           System.out.println("输入的整数个数小于 2 个");
        }
          catch(NumberFormatException e){
           System.out.println("请输入 2 个整数!!!  ");
        }
    }
}
```

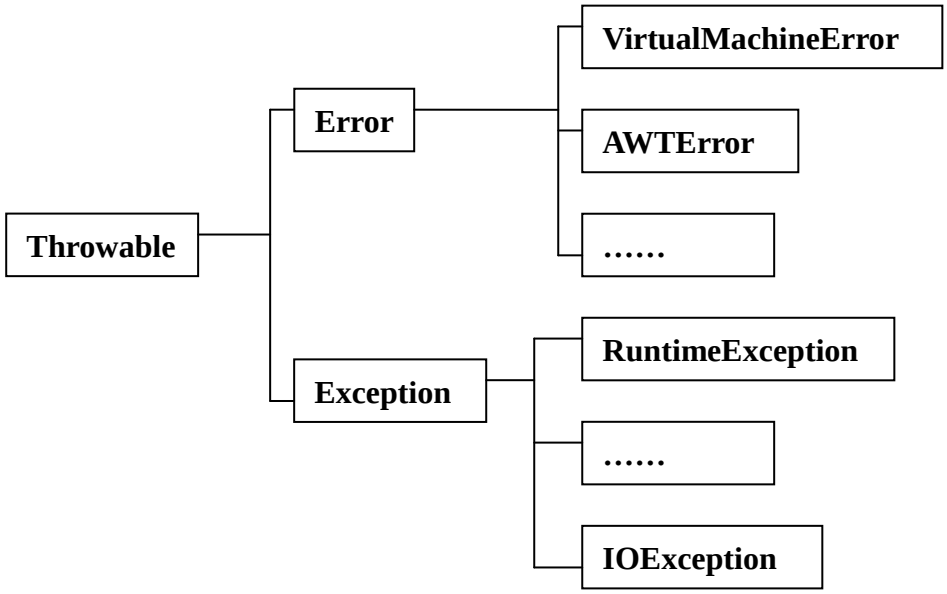
或者：

```
public class Add{
    public static void main(String[] args){
        int a,b,c;
        try{
            a=Integer.parseInt(args[0]);
            b=Integer.parseInt(args[1]);
            c=a+b;
            System.out.println("a+b="+c);
        }
```

```
        catch(Exception e){
           System.out.println("请输入 2 个整数");
        }
    }
}
```

（2）异常的类层次

Throwable 类是异常的根类，**Error** 是 **JVM** 系统内部错误，与具体程序无关，**Exception** 类是程序代码中需要处理的异常。



（3）系统定义的异常

Exception 类的子类，每个类代表一种运行时刻的异常，这些类都实现定义好放在了系统的类库中，称为系统定义的异常。

二、异常的处理

（1）**try…catch…finally** 结构

例题：

```
class TestFinally{
    public static void main(String[] args){
        try{
            System.out.println("A");
            //int a=8/0;
           //retrun;
        }
        catch (Exception e){
             System.out.println("B");
        }
        finally{
```

```
                System.out.println("C");
            }
            System.out.println("D");
        }
}
```

（2）多异常的程序举例

一个 try 可以引导多个 catch。

例题：8-3 根据输入的元素位置值查找数组元素的值。

```
import java.io.*;
public class MultiException{
    public static void main(String[] args){
        int[] arr={100,200,300,400,500,600};
        String index;
        int index1;
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("输入序号(输入 end 结束)： ");
        while(true){
            try{
              index=br.readLine();
              if(index.equals("end"))
                  break;
              index1=Integer.parseInt(index);
              System.out.println("元素的值为： "+arr[index1]);
            }
            catch(ArrayIndexOutOfBoundsException a){
              System.out.println("数组下标越界");
            }
            catch(NumberFormatException a){
              System.out.println("请输入一个整数");
            }
            catch(IOException a){}
        }
    }
}
```

三、自定义异常

（1）自定义异常类设计

例题：

```
class MyException extends Exception{
 String message;
 MyException(String s){
    message=s;
```

```
    }
    public String toString(){
       return message;
    }
   };
   class TestMyException{
    public static void main(String[] args){
          try{
              throw new MyException("我自己定义的异常类");
          }
          catch(MyException e){
              System.out.println(e);
          }
    }
   }
```

（2）抛出异常：throw

例题：

```
   import java.io.*;
   class TestThrow{
    public static void main(String[] args){
          try{
               throw new IOException();
          }
          catch (IOException e){
               System.out.println(e);
          }
    }
   }
```

（3）方法的异常声明：throws

例题：

```
   import java.io.*;
   class TestThrow{
    public static void main(String[] args)throws IOException{

    }
   }
```

第九章 流式输入/输出与文件处理

一、输入/输出的基本概念

（1）输入/输出设备与文件

外部设备：

① 存储设备：硬盘、光盘、U 盘等。

② 输入输出设备：鼠标、键盘、扫描仪、显示器、打印机、绘图仪等。

文件：

① 输入输出设备可以看成是一类特殊的文件；

② 文件可以看成是字节的序列；

③ 文件分为文本文件和二进制文件，文本文件存放的是 ASCII 码（或其他编码）表示的字符，而二进制文件是具有特定结构的数据。

（2）流的概念

Java 中的输入/输出是以流的方式来处理的。流是在计算机的输入/输出操作中流动的数据系列。

流中有未经加工的二进制数据，也有特定格式的数据。

★ 流的分类：

① 输入流和输出流；

② 字节流和字符流。

★ 流的特点：

① 先进先出。先被写入的数据，在读取时先被读取；

② 顺序存取。不能随机访问中间的数据；

③ 只读或只写。要么是输出流，要么是输入流。

★ 针对一些频繁的设备交互，Java 系统预先定义了 3 个可以直接使用：

① 标准输入（System.in）：InputStream 类型，通常代表键盘的输入；

② 标准输出（System.out）：PrintStream 类型，通常写往显示器；

③ 标准错误输出（System.err）：PrintStream 类型，通常写往显示器。

★ 可以使用下面的方法对流对象进行重定向：

① static void setIn(InputStream in)：重新定义标准输入流；

② static void setErr(PrintStream err)：重新定义标准错误输出；

③ static void setout(PrintStream out)：重新定义标准输出。

二、面向字节的输入/输出流

（1）面向字节的输入流

★ 面向字节的输入流都是 **InputStream** 的子类，此类为抽象类，类中定义了多个方法：

① **public int read()**：读取一个字节；

② **public int read(byte[] b)**：读多个字节到字节数组；

③ **public int read(byte[] b,int off,int len)**：从输入流读指定长度的数据到字节数组，数组从字节数组的 **off** 处开始存放；

④ **public long skip(long n)**：指针跳过 **n** 个字节；

⑤ **public void mark()**：在当前位置指针处做一标记；

⑥ **public void reset()**：将位置指针返回标记处；

⑦ **public void close()**：关闭流。

★ 使用 **InputStream** 类的子类

① 文件输入流（**FileInputStream**）

例题：**9.1** 在屏幕上显示文件内容

```
import java.io.*;
public class DisplayFile{
    public static void main(String[] args){
        try{
            FileInputStream infile=new FileInputStream(args[0]);
            int byteRead=infile.read();
            while(byteRead!=-1){
                System.out.print((char)byteRead);
               byteRead=infile.read();
            }
        }
        catch(ArrayIndexOutOfBoundsException e){
            System.out.println("需要提供一个文件名作为命令行参数");
        }
        catch(FileNotFoundException e){
           System.out.println("file not find!");
        }
        catch(IOException e){}
    }
```

```
}
```
改进此程序：每次读入多个字节。
```
import java.io.*;
public class DisplayFile2{
    public static void main(String[] args){
        try{
            FileInputStream infile=new FileInputStream(args[0]);
            byte b[]=new byte[100];
            int byteRead=0;
            while((byteRead=infile.read(b))!=-1){
                 String s=new String(b);
                System.out.print(s);
            }
        }
        catch(ArrayIndexOutOfBoundsException e){
            System.out.println("需要提供一个文件名作为命令行参数");
        }
        catch(FileNotFoundException e){
           System.out.println("file not find!");
        }
        catch(IOException e){}
    }
}
```
按字符流读取：
```
import java.io.*;
public class DisplayFile3{
    public static void main(String[] args){
        try{
            FileInputStream infile=new FileInputStream(args[0]);
            String s;
BufferedReader ln=new BufferedReader(new InputStreamReader(infile));
            while((s=ln.readLine())!=null){
                System.out.println(s);
```

```
            }
        }
        catch(ArrayIndexOutOfBoundsException e){
            System.out.println("需要提供一个文件名作为命令行参数");
        }
        catch(FileNotFoundException e){
           System.out.println("file not find!");
        }
        catch(IOException e){}
    }
}
```

② 数据输入流（DataInputStream）

为了规范对各类基本数据类型数据的读取操作，Java 定义了 DataInput 接口，该接口中规定了基本类型数据的输入方法：

- readByte()
- readBoolean()
- readShort()
- readCahr()
- readInt()
- readLong()
- readFloat()
- readDouble()

DataInputStream 类实现了 DataInput 接口。

※ 利用 DataInputStream 类可以从键盘上得到一个整数吗？

例题：9.2 从键盘上输入一个整数，求该数各位数字之和，并输出。

```
import java.io.*;
public class BitSum{
    public static void main(String[] args)throws IOException{
        DataInputStream din=new DataInputStream(System.in);
        System.out.print("input a integer:");
        int x=din.readInt();
        int sum=0;
        int n=x;
        while(n>0){
```

```
                int lastbit=n%10;
                n=n/10;
                sum=sum+lastbit;
            }
            System.out.println(x+"的各位数字之和="+sum);
        }
    }
```

（2）面向字节的输出流

★ 面向字节的输出流都是 OutputStream 的子类，此类为抽象类，类中定义了多个方法：

① public void write(int b)：将参数 b 的低字节写入输出流；

② public void write(byte[] b)：将字节数组写入输出流；

③ public void flush()：强制将缓冲区数据写入输出流对应的外设；

④ public void close()：关闭输出流；

例题：9.3 将一个大文件分拆成若干个小文件。

```
import java.io.*;
public class BigToSmall{
    public static void main(String[] args){
        int number=0;
        final int size=Integer.parseInt(args[1]);
         byte[] b=new byte[size];
        try{
            FileInputStream infile=new FileInputStream(args[0]);
            while(true){
               FileOutputStream outfile=new FileOutputStream("file"+number);
                number++;
                int byteRead=infile.read(b);
                if(byteRead==-1)break;
                outfile.write(b,0,byteRead);
                outfile.close();
            }
        }
        catch(IOException e){}
    }
```

```
}
```

例题：9.4 找出 10~100 之间的姐妹素数，写入到文件中，所谓姐妹素数是指相邻两个奇数均为素数。

```
import java.io.*;
public class FindSisterPrime{
    public static boolean isPrime(int n){
        for(int k=2;k<=Math.sqrt(n);k++){
            if(n%k==0)return false;
         }
         return true;
    }
    public static void main(String[] args){
       try{
          FileOutputStream file=new FileOutputStream("x.dat");
         DataOutputStream out=new DataOutputStream(file);
         for(int n=11;n<100;n+=2){
            if(isPrime(n)&&isPrime(n+2)){
               out.writeInt(n);
              out.writeInt(n+2);
            }
         }
       }
       catch(IOException e){}
    }
}
```

※ 打开 x.dat 文件发现是乱码。因为该文件中数据不是文本数据。要读取其中的数据需要以输入流的方式访问文件。

例如：

```
import java.io.*;
public class OutSisterPrime{
    public static void main(String[] args){
        try{
            FileInputStream file=new FileInputStream("x.dat");
            DataInputStream in=new DataInputStream(file);
```

```
            try{
                while(true){
                    int n1=in.readInt();
                   int n2=in.readInt();
                   System.out.println(n1+":"+n2);
                }
            }
            catch(EOFException e){
               in.close();
            }
        }
        catch(IOException e){}
    }
}
```

三、面向字符的输入/输出流

（1）面向字符的输入流

面向字符的输入流都是 Reader 的子类，该类中定义了读取字符的函数。

例题 9-5：从一个文件中读取数据加上行号后输出。

```
import java.io.*;
public class AddLineNo{
    public static void main(String[] args){
        try{
            FileReader file=new FileReader("AddLineNo.java");
            LineNumberReader in=new LineNumberReader(file);
            boolean eof=false;
            while(!eof){
                String x=in.readLine();
               if(x==null)
                    eof=true;
               else
                    System.out.println(in.getLineNumber()+":"+x);
            }
            in.close();
```

```
            }
            catch(IOException e){}
        }
    }
```

以上程序采用逻辑变量 eof，判断是否读到文件末尾，修改本程序：

```
    import java.io.*;
    public class AddLineNo2{
        public static void main(String[] args){
            try{
                FileReader file=new FileReader("AddLineNo.java");
                LineNumberReader in=new LineNumberReader(file);
                String s;
                while((s=in.readLine())!=null){
                    System.out.println(in.getLineNumber()+":"+s);
                }
                in.close();
            }
            catch(IOException e){}
        }
    }
```

（2）面向字符的输出流

面向字符的输出流都是 Writer 的子类，该类中定义了写入字符的函数。

例题：9.6 利用 FileWriter 类将 ASCII 英文字符集写入到文件。

```
    import java.io.*;
public class CharWrite{
    public static void main(String[] args){
        try{
          FileWriter fw=new FileWriter("charset.txt");
          for(int i=32;i<126;i++)
              fw.write(i);
          fw.close();
        }
        catch(IOException e){}
    }
```

}

四、转换流

（1）转换输入流：InputStreamReader

一个 **InputStreamReader** 对象接收一个字节输入流作为源，产生相应的 **UTF-16** 字符。其常用方法如下：

- ✧ **public InputStreamReader(InputStream in)**：创建转换输入流，按默认字符集的编码从输入流读取数据。
- ✧ **public InputStreamReader(InputStream in,Charset c)**：创建转换输入流，按指定字符集的编码从输入流读取数据。
- ✧ **public InputStreamReader(InputStream in String enc)throws UnsupportedEncodingException**：创建转换输入流，按名称所指定的字符集的编码从输入流读数据。

例如：按“**ISO 8859-6**”编码从文件中读取字符，将其转换为相应的 **UTF-16** 字符。

```
FileInputStream filein=new FileInputStream(file);
InputStreamReader in=new InputStreamReader(filein,”iso-8859-6”);
```

★ 强行将任意的字节流转换为字符流是没有任何意义的。

★ 标准输入设备（键盘）提供的流是字节流，实际上从键盘输入的数据是字符系列。

```
BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
String s=in.readLine();
```

（2）转换输出流：OutputStreamWtiter

一个 **OutputStreamWriter** 对象将 **UTF-16** 字符转换为指定的字符编码形式写入到字节输出流。其构造方法如下：

- ✧ **public OutputStreamWriter(OutputStream out)**：创建转换输出流，按默认字符集的编码往输出流写数据。
- ✧ **public OutputStreamWriter(OutputStream out, Charset c)**：创建转换输出流，按指定字符集的编码往输出流写数据。
- ✧ **public OutputStreamWriter(OutputStream out, String enc)**：创建转换输出流，按名称所指字符集的编码往输出流写数据。

五、文件处理(File 类)

File 类有一个欺骗性的名字——通常会认为它对付的是一个文件，但实情并非如此。它既代表一个特定文件的名字，也代表目录内一系列文件的名字。

（1）文件与目录管理

① 创建 **File** 对象

- ✧ **public File(String path)**
- ✧ **public File(String path,String name)**
- ✧ **public File(File dir, String name)**

② 获取文件或目录属性

- ✧ **public String getName():**获取文件的名字；
- ✧ **public boolean canRead():**判断文件是否可读；
- ✧ **public boolean canWrite():**判断文件是否可写；
- ✧ **public boolean exits():**判断文件是否存在；
- ✧ **public long length():**获取文件的长度；
- ✧ **public String getAbsolutePath():**获取文件的绝对路径；
- ✧ **public String getParent():**获取文件的父目录；
- ✧ **public boolean ifFile():**判断文件是否是一个正常文件
- ✧ **public boolean isDirectory():**判断文件是否是一个目录；
- ✧ **public boolean isHidden():**判断文件是否是隐藏文件；
- ✧ **public long lastModified():**获取文件最后的修改时间。

③ 文件或目录操作

- ✧ **public boolean mkdir():**创建一个目录，如果创建成功返回 **true**，否则返回 **false**，如果目录已经存在也返回 **false**。
- ✧ **public boolean createNewFile():**创建文件；
- ✧ **public boolean delete():**删除当前文件。
- ✧ **public boolean renameTo(File newFile):**将文件改名为新文件名。
- ✧ **public boolean equals(File f):**比较两个文件或目录是否相等。
- ✧ **public String[] list():**用字符串形式返还目录下的全部文件；
- ✧ **public File[] listFiles():**用 **File** 对象的形式返回目录下的全部文件；
- ✧ **public String[] list(FilenameFilter obj):**用字符串形式返回目录下指定类型的所有文件；
- ✧ **public File[] listFiles():**用 **File** 对象形式返回目录下制定类型的所有文件；
- ◆ 接口 **FilenameFilter**：文件过滤器

该接口中的方法：

public Boolean accept(File dir,String name);

例题：列出 **E:\chaper9** 目录下的扩展名为**.java** 的文件，并删除 **E.java** 文件。

```
import java.io.*;
class FileAccept implements FilenameFilter {
String str=null;
    FileAccept(String s){
    str="."+s;
    }
    public   boolean accept(File dir,String name){
    return name.endsWith(str);
    }
}
public class Example10_2{
public static void main(String args[ ]){
     File dir=new File("E:\\chaper9");
        File deletedFile=new File(dir,"E.java");
        FileAccept acceptCondition=new FileAccept("java");
        File   fileName[]=dir.listFiles(acceptCondition);
        for(int i=0;i< fileName.length;i++){
      System.out.println("文件名称:"+fileName[i].getName());
        }
        boolean boo=deletedFile.delete();
        if(boo){
System.out.println("文件:"+deletedFile.getName()+"被删除");
        }
    }
}
```

例题：9-7 显示若干指定文件的基本信息，文件通过命令行参数提供。

```
import java.io.*;
class FileInfo{
  static File fileToCheck;
  public static void main(String[] args)throws IOException{
```

```
        if(args.length>0){
            for(int i=0;i<args.length;i++){
                fileToCheck=new File(args[i]);
               info(fileToCheck);
            }
        }
        else{
            System.out.println("用法：java FileInfo file1 file2 ...");
        }
    }
    public static void info(File f)throws IOException{
        System.out.println("Name:"+f.getName());
        System.out.println("Path:"+f.getPath());
        System.out.println("Absolute Path:"+f.getAbsolutePath());
        if(f.exists()){
            System.out.println("File exists");
           System.out.println("and is Readable:"+f.canRead());
           System.out.println("and is Writeable:"+f.canWrite());
           System.out.println("File is "+f.length()+"bytes");
        }
        else{
            System.out.println("file does not exist");
        }
    }
}
```

（2）文件的顺序访问

根据文件中数据的类型不同，有两种不同的流访问文件：

- ✧ **FileInputStream 和 FileOutputStream**
- ✧ **FileReader 和 FileWriter**

★ 向文件中写数据要建立 **FileInputStream** 的对象或 **FileReader** 的对象，从文件中读数据要建立 **FileOutputStream** 的对象或 **FileWriter** 的对象。不能在一个流中即对文件进行读操作也进行写操作。

（3）文件的随机访问（**RandomAccessFile** 类）

RandomAccessFile 类提供了对流进行随机读写的能力，此类实现了 **DataInput** 和 **DataOutput** 接口，可对数据进行格式化读取。同时还增加了如下方法：

- ✧ **public long getFilePointer():**返回当前指针。
- ✧ **public void seek(long pos):**将文件指针定位到一个绝对地址。地址是相对于文件开头的偏移量，地址 **0** 表示文件的开头。
- ✧ **public void length():**返回文件的长度。
- ✧ **public void setLength(long newLength):**设置文件的长度。

RandomAccessFile 类的构造方法：

- ✧ **public RandomAccessFile(String name, String mode)**
- ✧ **public RandomAccessFile(File file, String mode)**

其中 **mode** 表示访问权限：**r** 表示只读，**rw** 表示可读写。

例题：

```
import java.io.*;
public class E1{
    public static void main(String args[]){
        RandomAccessFile in_and_out=null;
        int data[]={1,2,3,4,5,6,7,8,9,10};
        try{
            in_and_out=new RandomAccessFile("tom.dat","rw");
        }
        catch(Exception e){}
        try{
            for(int i=0;i<data.length;i++){
                in_and_out.writeInt(data[i]);
             }
             for(long i=data.length-1;i>=0;i--){
                in_and_out.seek(i*4);
                      //每隔 4 个字节往前读取一个整数
                  System.out.print(","+in_and_out.readInt());
             }
             in_and_out.close();
            }
        catch(IOException e){}
```

```
    }
}
```

例题：

```
import    java.io.*;
class E2{
    public static void main(String args[]){
        try{
    RandomAccessFile in=new RandomAccessFile("E2.java","rw");
                long filePoint=0;
                long fileLength=in.length();
                while(filePoint<fileLength){
                   String s=in.readLine();
                      System.out.println(s);
                      filePoint=in.getFilePointer();
                }
                in.close();
            }
         catch(Exception e){}
    }
}
```

例题：9-8 模拟应用日志处理，将键盘输入数据写入到文件尾部。

```
    import java.io.*;
    public class RaTest{
        public static void main(String[] args){
            try{
BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
                String s=in.readLine();
    RandomAccessFile myRAFile=new RandomAccessFile("java.log","rw");
                myRAFile.seek(myRAFile.length());//将指针定到文件尾部
                  myRAFile.writeBytes(s+"\n");
                myRAFile.close();
            }
            catch(IOException e){}
        }
```

```
}
```

例题：9-9 应用系统访问统计

```
import java.io.*;
public class Count{
    public static void main(String[] args)throws IOException{
        long count;
        RandomAccessFile fio=new RandomAccessFile("count.txt","rw");
        if(fio.length()==0)
            count=1L;
        else{
           fio.seek(0);
           count=fio.readLong();
           count=count+1L;
           //System.out.println(count);
        }
        fio.seek(0);
        fio.writeLong(count);
        fio.close();
    }
}
```

六、对象串行化

对象的串行化也称对象的序列化，是指将对象写入输出流。

对象的反串行化也称对象的反序列化，是指将对象从输出流中读出。

ObjectOutputStream 类和 **ObjectInputStream** 类中的 **writeObject()**方法和 **readObject()**实现了对象的串行化和反串行化。

例题：9-10 系统对象的串行化处理

程序一：将系统对象写入文件。

```
import java.io.*;
import java.util.*;
public class WriteDate{
    public static void main(String[] args){
        try{
         ObjectOutputStream out=new ObjectOutputStream(new
```

```
FileOutputStream("storedate.dat"));
                out.writeObject(new Date());
                out.writeObject("hello world");
                System.out.println("写入完毕");
            }
            catch(IOException e){}
        }
    }
```

程序二：读取文件中的对象并显示出来。

```
import java.io.*;
import java.util.*;
public class ReadDate{
    public static void main(String[] args){
        try{
            ObjectInputStream in=new ObjectInputStream(new
FileInputStream("storedate.dat"));
            Date current=(Date)in.readObject();
            System.out.println("日期："+current);
            String str=(String)in.readObject();
            System.out.println("字符串："+str);
        }
        catch(IOException e){}
        catch(ClassNotFoundException e){}
    }
}
```

★ 当输入源的数据不符合对象规范或无数据时将产生 IOException 异常。

★ 调用对象输入流的 **readObject()** 方法时必须捕获 **ClassNotFoundException** 异常。

例题：**9-11** 利用对象串行化将各种图形元素以对象的形式存储，从而实现图形的保存。

程序一：图形对象的串行化设计。

```
import java.awt.Graphics;
import java.io.*;
abstract class Graph implements Serializable{
```

```
    public abstract void draw(Graphics g);
}
class Line extends Graph implements Serializable{
    int x1,y1;
    int x2,y2;
    public void draw(Graphics g){
       g.drawLine(x1,y1,x2,y2);
    }
    public Line(int x1,int y1,int x2,int y2){
       this.x1=x1;
       this.y1=y1;
       this.x2=x2;
       this.y2=y2;
    }
}
class Circle extends Graph implements Serializable{
    int x,y;
    int r;
    public void draw(Graphics g){
       g.drawOval(x,y,r,r);
    }
    public Circle(int x,int y,int r){
       this.x=x;
       this.y=y;
       this.r=r;
    }
}
```

程序二：测试将图形对象串行化写入文件。

```
import java.io.*;
public class WriteGraph{
    public static void main(String[] args){
         Line k1=new Line(20,20,80,80);
         Circle k2=new Circle(60,50,80);
         try{
```

```
                ObjectOutputStream out=new ObjectOutputStream(new
FileOutputStream("storedate.dat"));
                out.writeObject(new Integer(2));
                out.writeObject(k1);
                out.writeObject(k2);
            }
            catch(IOException e){
                System.out.println(e);
            }
        }
    }
```

程序三：从文件读取串行化对象并绘图。

```
    import java.awt.*;
    import java.io.*;
    public class DisplayGraph extends Frame{
        public static void main(String[] args){
            new DisplayGraph();
        }
        public DisplayGraph(){
            super("读对象文件显示图形");
            setSize(300,300);
            setVisible(true);
            Graphics g=getGraphics();
            try{
                ObjectInputStream in=new ObjectInputStream(new
FileInputStream("storedate.dat"));
                int n=((Integer)in.readObject()).intValue();
                for(int i=1;i<=n;i++){
                    Graph me=(Graph)in.readObject();
                    me.draw(g);
                }
            }
            catch(IOException e){
                System.out.println(e);
```

```
        }
        catch(ClassNotFoundException e){
           System.out.println(e);
        }
    }
}
```

第十章 多线程

一、Java 多线程的概念

分析下面程序：

```
public class SingleThread{
    public static void main(String[] args){
        while(true){
            System.out.println("hello");
        }
        while(true){
            System.out.println("您好");
        }
    }
}
```

其中第二个 while 循环是无法访问的语句。

使用 Java 中的多线程：

```
public class MultiThread{
    public static void main(String[] args){
        MyThread1 t1=new MyThread1();
        t1.start();
        MyThread2 t2=new MyThread2();
        t2.start();
    }
}
class MyThread1 extends Thread{
    public void run(){
       while(true){
          try{
               sleep(1000);
          }
          catch(InterruptedException e){}
           System.out.println("hello");
       }
```

```
    }
  }
  class MyThread2 extends Thread{
    public void run(){
      while(true){
        try{
          sleep(1000);
        }
        catch(InterruptedException e){}
        System.out.println("您好");
      }
    }
  }
```

（1）多进程与多线程

- ✧ 一个进程要消耗大量的资源，所以多进程开销很大。多个进程之间的通信不方便，大多数操作系统不允许进程访问其他进程的内存空间。
- ✧ 多线程是指在一个进程中可以运行多个不同的线程，执行不同的任务。同进程中的多个线程共享代码和数据空间，每个线程有独立的运行栈和程序计数器。线程的开销较小。

（2）线程的状态

- ✧ 新建状态：创建线程对象后。
- ✧ 就绪状态：线程对象调用 **start()**方法后。
- ✧ 运行状态：就绪状态的线程获得 **CPU** 资源后。
- ✧ 阻塞状态：线程因某些事情的发生或等待某个资源。

  线程进入阻塞状态的原因：

  ① 通过调用 **sleep(milliseconds)**方法使任务进入休眠状态，在这种情况下任务在指定时间内不会运行。

  ② 通过调用 **wait()**方法使线程挂起。直到线程得到了 **notify()**或 **notifyAll()**消息（或者在 **JavaSE5** 的 **java.util.concurrent** 类库中的 **signal()**或 **signalAll()**消息），线程才会进入就绪状态。

  ③ 任务在等待某个输入/输出完成。

  ④ 任务试图在某个对象上调用其同步控制方法，但是对象锁不可用，因为另一个任务已经获取了这个锁。
- ✧ 死亡状态：线程对象释放所拥有的资源。

（3）线程调度与优先级

Java 提供了一个线程调度器来负责线程调度，并采用抢占式调度策略。优先级高的线程先运行，线程优先级相同的线程按“先到先服务”的原则调度。每个线程安排一个时间片，执行完时间片将轮到下一个线程。

当遇到下面几种情况时，当前线程会放弃 CPU 资源：

- ✧ 当前时间片用完。
- ✧ 线程在执行时调用了 yield()方法或 sleep()方法主动放弃。
- ✧ 进行 I/O 访问，等待用户输入，导致线程阻塞，或者为等候一个条件变量，线程调用 wait()方法。
- ✧ 有优先级高的线程参与调度。

线程的优先级用数字来表示，其范围是 1~10。主线程的默认优先级是 5。

Thread 类中提供了几个常量表示优先级：

- ✧ Thread.MIN_PRIORITY=1;
- ✧ Thread.MAX_PRIORITY=10;
- ✧ Thread.NORM_PRIORITY=1;

例题：

```
public class MultiThread{
    public static void main(String[] args){
        MyThread1 t1=new MyThread1();
        t1.start();
        MyThread2 t2=new MyThread2();
        t2.start();
        t1.setPriority(Thread.MAX_PRIORITY);
        t2.setPriority(Thread.MIN_PRIORITY);
    }
}
class MyThread1 extends Thread{
    public void run(){
      while(true){
          System.out.println("hello,hello,hello,hello,hello");
      }
    }
}
class MyThread2 extends Thread{
```

```
        public void run(){
           while(true){
               System.out.println("您好");
           }
        }
    }
```

二、Java 多线程编程方法

（1）Thread 类简介

常用方法参看 Java API。

（2）继承 Thread 类实现多线程

例题：10-1 直接继承 Thread 类实现多线程。

```
import java.util.*;
class TimePrinter extends Thread{
    int pauseTime;
    String name;
    public TimePrinter(int x,String n){
        pauseTime=x;
        name=n;
    }
    public void run(){
       while(true){
           try{
              System.out.println(name+":"+new
Date(System.currentTimeMillis()));
              Thread.sleep(pauseTime);
           }
           catch(Exception e){
               System.out.println(e);
           }
       }
    }
    public static void main(String[] args){
```

```
        TimePrinter tp1=new TimePrinter(1000,"Fast Guy");
        tp1.start();
         TimePrinter tp2=new TimePrinter(3000,"Slow Guy");
        tp2.start();
    }
}
```

（3）实现 Runnable 接口编写多线程

例题：10-2 计数按钮的统计

```
/*
<applet code=CountApplet.class width=500 height=500></applet>
*/
import java.applet.*;
import java.awt.*;
class CountButton extends Button implements Runnable{
    int count=0;
    public CountButton(String s){
       super(s);
    }
    public void run(){
       while(count<1000){
           try{
              this.setLabel(""+count++);
             Thread.sleep((int)(1000*Math.random()));
           }
           catch(Exception e){}
       }
    }
}
public class CountApplet extends Applet{
    public void init(){
        setLayout(null);
        CountButton t1=new CountButton("first");
        t1.setBounds(30,10,80,40);
        add(t1);
```

```
            CountButton t2=new CountButton("second");
            t2.setBounds(130,10,80,40);
            add(t2);
            (new Thread(t1)).start();
            (new Thread(t2)).start();
        }
    };
```

三、线程的控制

（1）放弃运行

yield()方法：线程对象主动放弃本线程的 CPU 资源，从“运行”状态转到“就绪”状态。

例题：

```
public class TestYield extends Thread{
    public static void main(String[] args){
        for(int i=1;i<=2;i++){
            new TestYield().start();
        }
    }
    public void run(){
        System.out.println("A");
        Thread.yield();
        System.out.println("B");
    }
}
```

（2）无限等待

suspend()方法和 resume()方法现已过时。

现在使用 wait()和 notify()方法。

（3）睡眠一段时间

sleep(millisecond)方法。

（4）阻塞

造成阻塞的原因有多种。

（5）关于用户线程和看守线程

可以通过 setDaemon(boolean on)：设置线程为看守线程（on 为 true），或

用户线程（on 为 false）。

例题：

```
public class TestDaemon{
public static void main(String args[ ]){
        DaemonOne    a=new DaemonOne();
        DaemonTwo    b=new DaemonTwo();
        Thread t1=new Thread(a);
        Thread t2=new Thread(b);
         t1.start();
         //t2.setDaemon(true);
         t2.start();
    }
}
class DaemonOne    implements Runnable{
    public void run(){
       for(int i=0;i<8;i++){
          System.out.println("i="+i) ;
                try{
                   Thread.sleep(1000);
                }
                catch(InterruptedException e) {}
        }
     }
}
class DaemonTwo implements Runnable{
     public void run(){
         while(true){
            System.out.println("我是线程 B");
                 try{
                   Thread.sleep(1000);
                 }
                 catch(InterruptedException e){}
         }
     }
```

```
}
```

（6）线程的联合

一个线程 A 在占用 CPU 资源期间，可以让其它线程调用 join()方法与本线程联合：

```
B.join();
```

称线程 A 在运行期间联合了线程 B。

★如果 A 联合了 B，那么 A 立刻中断执行，一直等到与它联合的 B 运行完毕，线程 A 再重新排队等待 CPU 资源。

例题：

```
public class TestJoin{
    public static void main(String[] args)throws InterruptedException{
        B b=new B();
        //b.start();
        b.join();
        System.out.println("end");
    }
}
class B extends Thread{
    int sum=0;
    public void run(){
        for(int i=0;i<=10;i++){
            sum=sum+i;
         }
        System.out.println(sum);
    }
};
```

四、线程资源的同步处理

多个线程共享同一块内存空间和一组系统资源，可能造成某些资源利用上的协调问题。Java 已经提供了解决的办法：避免多个线程同时访问同一个资源。

（1）临界资源问题

多个线程共享的数据称为临界资源。对临界资源的访问会造成混乱。

例题：未使用同步机制的多线程程序：

```
class DataClass{
    private int data=0;
    public void increase(){
        int nd=data;
        try{
            Thread.sleep(100);
        }
        catch (Exception e){}
        data=nd+1;
    }
    public int getData(){
        return data;
    }
};
class NThread extends Thread{
    DataClass d;
    NThread(DataClass d){
      this.d=d;
    }
    boolean alive=true;
    public void run(){
        for(int i=0;i<100;i++){
          d.increase();
        }
        alive=false;
    }
};
public class   NoSyn{
    public static void main(String[] args){
        DataClass d=new DataClass();
        NThread t1=new NThread(d);
        NThread t2=new NThread(d);
        t1.start();
        t2.start();
```

```
        while(t1.alive||t2.alive);
        System.out.println("data="+d.getData());
    }
}
```

例题：使用同步机制的多线程程序：

public synchronized void increase(){……}

将例题 1 进行修改，加入同步机制。

★ 同步块：

synchronized 修饰的方法虽然可以解决同步问题，但也存在缺陷，如果一个 **synchronized** 的方法需要执行的时间很长，将会大大影响系统的效率，所有 **Java** 提供了一个解决办法：使用 **synchronized** 块。

例题：使用 **Synchronized** 块的方法：

```
class CallMe{
    void call(String msg){
        synchronized(this){
            System.out.print("["+msg);
            try{
                Thread.sleep(1000);
            }
            catch (Exception e){}
            System.out.println("]");
        }
    }
};
class Caller implements Runnable{
    String msg;
    CallMe target;
    public Caller(CallMe t,String s){
        target=t;
        msg=s;
        new Thread(this).start();
    }
    public void run(){
        target.call(msg);
```

```
    }
};
public class SynBlock{
    public static void main(String[] args){
        CallMe target=new CallMe();
        new Caller(target,"hello");
        new Caller(target,"synchronized");
        new Caller(target,"world");
    }
}
```

总结：synchronized 关键字的使用方法有两种：

- ✧ 用在对象的前面，限制一段代码的执行，表示执行该段代码必须取得对象锁。
- ✧ 在方法前面，表示该方法为同步方法，执行该方法必须取得对象锁。

（2）wait()方法和 notify()方法

- ✧ wait(): 使线程进入阻塞状态，并释放该线程所占用的对象锁。由 notify()或 notifyAll()唤醒。
- ✧ wait(long timeout): 使线程进入阻塞状态，并释放该线程所占用的对象锁。由 notify()或 notifyAll()唤醒，或者经过 timeout 时间后自动唤醒。
- ✧ notify(): 在等待队列中随机选择一个线程唤醒。
- ✧ notifyAll(): 把等待队列中的所有线程唤醒。

★ 采用 wait 和 notify 可以解决很多临界资源问题。如：假设两个线程都要访问某个数据区，要求线程 1 的访问先于线程 2，则仅用 synchronized 不能解决。

使用 wait 和 notify 的解决方法如下：

```
synchronized method1(...){//由线程 1 调用
    ...//此处访问共享资源
    available=true;
    notify();
}
synchronized method2(...){//由线程 2 调用
    while(!avaliable){
        try{
```

```
                wait();
              }
              catch(InterruptedException e){}
            }
            ...//此处访问共享资源
        }
```

（3）生产者和消费者模型

特例：当一个线程正在使用 synchronized 方法时，此线程要想完成此方法，它必须使用“其它线程使用此 synchronized 方法产生”的结果，而其它线程又不能使用此 synchronized 方法。

例题：10-3 生产者与消费者模型

```
class Consumer extends Thread{
    private ShareArea sharedObject;
    public Consumer(ShareArea shared){
        sharedObject=shared;
    }
    public void run(){
        int value;
        do{
          try{
            Thread.sleep((int)(Math.random()*3000));
          }
          catch(InterruptedException e){}
          value=sharedObject.getSharedInt();
        }
        while(value!=10);
    }
}
class Producer extends Thread{
    private ShareArea sharedObject;
    public Producer(ShareArea shared){
        sharedObject=shared;
    }
    public void run(){
```

```
        for(int count=1;count<=10;count++){
            try{
                Thread.sleep((int)(Math.random()*3000));
            }
            catch(InterruptedException e){}
            sharedObject.setSharedInt(count);
        }
    }
}
class ShareArea{
    private int sharedInt=-1;
    private boolean writeable=true;
    public synchronized void setSharedInt(int value){
        while(!writeable){
          try{
            wait();
          }
          catch(InterruptedException e){}
        }
        System.out.println("生产："+value);
        sharedInt=value;
        writeable=false;
        notify();
    }
    public synchronized int getSharedInt(){
        while(writeable){
            try{
              wait();
            }
            catch(InterruptedException e){}
        }
        System.out.println("消费："+sharedInt);
        writeable=true;
        notify();
```

```
            return sharedInt;
        }
    }
    public class SharedTest{
        public static void main(String args[]){
            ShareArea sharedObject=new ShareArea();
            Producer p=new Producer(sharedObject);
            Consumer c=new Consumer(sharedObject);
            p.start();
            c.start();
        }
    }
```

（4）死锁

对象 a 的锁 A，对象 b 的锁 B

线程 t1：获得 A，等待 B…

线程 t2：获得 B，等待 A…

例题：死锁的示例程序

```
class A{
    synchronized void print(){
        System.out.println("Aprint");
    }
    synchronized void callB(B bObject){
        System.out.println(Thread.currentThread().getName()+":");
        System.out.println("Lock aObject,and wait bObject...");
        try{
            Thread.sleep(100);
        }
        catch (Exception e){}
        bObject.print();
    }
};
class B{
    synchronized void print(){
```

```
            System.out.println("Bprint");
        }
        synchronized void callA(A aObject){
            System.out.println(Thread.currentThread().getName()+":");
            System.out.println("Lock bObject,and wait aObject...");
            try{
                Thread.sleep(100);
            }
            catch (Exception e){}
            aObject.print();
        }
};
public class DeadLock implements Runnable{
        A a=new A();
        B b=new B();
        DeadLock(){
            new Thread(this).start();
            a.callB(b);
        }
        public void run(){
            b.callA(a);
        }
        public static void main(String[] args){
            new DeadLock();
        }
}
```

第十一章 JDBC 技术和数据库应用

一、关系数据库概述

关系数据库：数据以行、列的表格形式存储、通常一个数据库由一组表构成，表中的数据项以及表之间的链接通过关系来组织和约束。

- ✧ 常用小型数据库：**Microsoft Access、MySql** 等。
- ✧ 常用大型数据库：**IBM DB2、Microsoft SQL Server、Oracle、Sybase** 等。

表 **11-1** 常用 **SQL** 命令的使用举例

命令	功能	举例
Create	创建表格	**create table coffees(cof_name varchar(32), price integer)**
Drop	删除表格	**drop tabe coffees**
Insert	插入数据	**insert into coffees values('Colombian', 101)**
Select	查询数据	**select cof_name, price from coffees where price>7**
Delete	删除数据	**delete * from coffees where cof_name='Colombian'**
Update	修改数据	**update coffees set price=price+1**

二、JDBC

JDBC:Java DataBase Connectivity。

（1）JDBC 驱动程序

- ✧ **JDBC-ODBC** 桥接器驱动程序
- ✧ 本地 **API** 结合部分 **java** 驱动程序
- ✧ **JDBC-Net** 纯 **java** 驱动程序
- ✧ 本地协议纯 **java** 驱动程序

（2）ODBC 数据源配置

ODBC 数据源的配置过程：

① “控制面板”->“管理工具”->“数据源（ODBC)”。

② 选择“系统 DNS 选项卡”。

③ 单击“添加”按钮，选择数据库驱动程序。本教材选择“Microsoft Access Driver(*.mdb)”。

④ 出现“ODBC Microsoft Access Driver 安装”对话框，在“数据源名”文本框中输入数据源名称。

⑤ 单击数据库的“选择”按钮，选择数据库。

（3）JDBC API

java.sql 包提供了多种 JDBC API：

✧ Connection 接口：代表与数据库的连接。

✧ Statement 接口：用来执行 SQL 语句并返回结果记录集。

✧ ResultSet：SQL 语句执行后的结果记录集。必须逐行访问数据行

① 使用 JDBC 连接数据库

调用 DriverManager.getConnection()方法与数据库建立连接;

例题：11-1 通过 ODBC 数据源与 Access 数据库的连接。

```
import java.sql.*;
public class ConnectDataBase{
    public static void main(String[] args){
        String url="jdbc:odbc:mydata";
        try{
           Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        }
        catch(ClassNotFoundException e){
           System.out.println(e.getMessage());
        }
        try{
           Connection con=DriverManager.getConnection(url,"",null);
           System.out.println("Connection Succeed!");
           con.close();
        }
        catch(SQLException ex){
           System.out.println("Message:"+ex.getMessage());
        }
    }
}
```

② 创建 Statement 对象

使用 Connection 类的 createStatement()方法创建：

Statement stmt=con.createStatement();

使用 stmt 对象调用 executeQuery()方法执行 SQL 语句：

ResultSet rs=stmt.executeQuery(“select a,b,c from tableName”);

例题：11-2 在一个空库中创建数据表

```
import java.sql.*;
```

```
public class CreateStudent{
    public static void main(String[] args){
        String url="jdbc:odbc:mydata";
        String sql="create table student (name varchar(20),sex
char(2),birthday Date, leave bit, stnumber integer)";
        try{
          Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        }
        catch(ClassNotFoundException e){
          System.out.println(e.getMessage());
        }
        try{
          Connection con=DriverManager.getConnection(url,"",null);
          Statement stmt=con.createStatement();
          try{
            stmt.executeUpdate(sql);
           System.out.println("student table created");
          }
          catch(Exception ex){}
          stmt.close();
          con.close();
        }
        catch(SQLException ex){
          System.out.println(ex.getMessage());
        }
    }
}
```

三、JDBC 基本应用

（1）数据库的查询

① 获取列表信息

② 遍历访问结果集（定位行）

③ 访问当前行的数据项（具体列）

例题：11-3 查询学生表信息

```
import java.sql.*;
public class QueryStudent{
    public static void main(String[] args){
        String url="jdbc:odbc:mydata";
        String sql="select * from student";
        try{
           Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        }
        catch(ClassNotFoundException e){
           System.out.println(e.getMessage());
        }
        try{
           Connection con=DriverManager.getConnection(url,"",null);
           Statement stmt=con.createStatement();
           ResultSet rs=stmt.executeQuery(sql);
           while(rs.next()){
              String s1=rs.getString("name");
             String s2=rs.getString("sex");
             Date d=rs.getDate("birthday");
             boolean v=rs.getBoolean("leave");
             int n=rs.getInt("stnumber");
             System.out.println(s1+","+s2+","+d+","+v+","+n);
           }
           rs.close();
           stmt.close();
           con.close();
        }
        catch(SQLException e){
           System.out.println(e);
        }
    }
}
```

④ 创建滚动结果集

例题：11-4 游标的移动

```
import java.sql.*;
public class MoveCursor{
    public static void main(String[] args){
        String url="jdbc:odbc:mydata";
        String sql="select * from student";
        try{
           Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        }
        catch(ClassNotFoundException e){
           System.out.println(e.getMessage());
        }
        try{
           Connection con=DriverManager.getConnection(url,"",null);
           Statement
stmt=con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.
CONCUR_READ_ONLY);
           ResultSet rs=stmt.executeQuery(sql);
           rs.last();
           int num=rs.getRow();
           System.out.println("共有学生数量="+num);
           rs.beforeFirst();
           while(rs.next()){
              String s1=rs.getString("name");
             String s2=rs.getString("sex");
             Date d=rs.getDate("birthday");
             boolean v=rs.getBoolean("leave");
             int n=rs.getInt("stnumber");
             System.out.println(s1+","+s2+","+d+","+v+","+n);
           }
           rs.close();
           stmt.close();
           con.close();
        }
        catch(SQLException e){
```

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

（2）数据库的更新

① 数据的插入

例题：向空表 student 中插入数据

```
import java.sql.*;
public class InsertData{
    public static void main(String[] args){
        String url="jdbc:odbc:mydata";
        String sql="insert into student values(' 王 五 ',' 女 ',DateValue('81/02/13'),false,20010845)";
        try{
          Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        }
        catch(ClassNotFoundException e){
          System.out.println(e.getMessage());
        }
        try{
          Connection con=DriverManager.getConnection(url,"",null);
          Statement stmt=con.createStatement();
          try{
            stmt.executeUpdate(sql);
           System.out.println("Insert Data Successed");
          }
          catch(Exception ex){}
          stmt.close();
          con.close();
        }
        catch(SQLException ex){
          System.out.println(ex.getMessage());
        }
```

```
        }
    }
```

② 数据的修改和数据的删除

```
    sql="update student set sex='女' where name='张三'";
    sql="delete * from student where stnumber=20010846";
```

（3）用 PreparedStatement 类实现 SQL 操作

例题：11-6 采用 PreparedStatement 实现数据写入。

```
import java.sql.*;
public class InsertStudent2{
    public static void main(String[] args){
        String url="jdbc:odbc:mydata";
        try{
           Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        }
        catch(ClassNotFoundException e){
           System.out.println(e.getMessage());
        }
        try{
           Connection con=DriverManager.getConnection(url,"",null);
           Statement stmt=con.createStatement();
           String sql="insert into student values(?,?,?,?,?)";
           PreparedStatement ps=con.prepareStatement(sql);
           ps.setString(1,"王五");

            ps.setString(2,"男");
           ps.setDate(3,java.util.Date.valueOf("1982-02-15"));
           ps.setBoolean(4,true);
           ps.setInt(5,20001);
           ps.executeUpdate();
           System.out.println("add 1 Item");
           stmt.close();
           con.close();
        }
        catch(SQLException ex){
```

```
            System.out.println(ex.getMessage());
        }
    }
}
```

四、数据库应用举例

第十二章 Java 的网络编程

一、网络编程基础

网络协议

网络上计算机之间的通信必须遵循一定的协议，目前最广泛的网络协议是 TCP/IP。

IP 主要负责网络主机的定位，实现数据传输的路由选择。实际应用中使用域名地址，域名和 IP 之间的转换通过域名解析完成。

传输层负责数据的正确性，有两类典型的通信协议：

① TCP：Transfer Control Protocol

通过 TCP 协议传输，得到的是一个顺序的无差错的数据流。使用 TCP 通信，发送方和接收方首先要建立 socket 连接，在客户/服务器通信中，服务方在某个端口提供服务，等待客户方的访问连接，建立连接后，双方就可以发送和接收数据了。

② UDP：User Datagram Protocol

UDP 是一种无连接的协议，每个数据报都是一个独立的信息，包括完整的源地址或目的地址，它在网络上以任何可能的路径传往目的地。因此能否到达目的地、到达目的地的时间及内容的正确性都不能保证，但 UDP 无需进行连接，传输效率高。如传输声音信号或视频信号等。

java.net 包中提供了丰富的网络功能：

① InetAddress 类表示 IP 地址；

② URL 类封装了对资源的访问；

③ ServerSocket 类和 Socket 类实现面向连接的网络通信；

④ DatagramPacket 类和 DatagrameSocket 类实现数据报的收发。

二、InetAddress 类

因特网上用 IP 地址或域名标识主机，InetAddress 对象封装了这两部分内容。InetAddress 对象使用如下格式表示主机的信息：

www.hhstu.edu.cn/125.46.34.241

InetAddress 类的主要方法有：

① static InetAddress getByName(String host)：根据主机名 InetAddress 对象，使用该方法必须捕获 UnknownHostException 异常。

② static InetAddress getLocalHost()：返回本地主机对应的 InetAddress 对

象，如果该主机无 IP 地址，则产生 UnknownHostException 异常。

③ String getHostAddress()：返回 UnknownHostException 异常的 IP 地址。

④ String getHostName()：返回 UnknownHostException 异常的域名。

例题：

```
import java.net.*;
public class WhoAmI {
   public static void main(String[] args)
         throws Exception {
      if(args.length != 1) {
         System.err.println(
            "Usage: WhoAmI MachineName");
         System.exit(1);
      }
      InetAddress a =
         InetAddress.getByName(args[0]);
      System.out.println(a);
   }
}
```

运行：

```
F:/>java WhoAmI    www.baidu.com
```

或

```
F:/>java WhoAmI    61.135.169.105
```

得到本地主机可以：

```
F:/>java WhoAmI    jzx
```

或

```
F:/>java WhoAmI    localhost
```

★ jzx 本地主机的名字

也可以使用 InetAddress.getLocalHost()函数得到本地主机的 InetAddress 对象。

例如：

```
import java.net.*;
public class WhoAmI2 {
   public static void main(String[] args)
         throws Exception {
```

```
        InetAddress a =
          InetAddress.getLocalHost();
        System.out.println(a);
    }
}
```

可以使用 InetAddress.getAllByName(String)方法。

例如：

```
import java.net.*;
public class WhoAmI3 {
   public static void main(String[] args)
         throws Exception {
        //InetAddress[] a = InetAddress.getAllByName("www.baidu.com");
        //InetAddress[] a = InetAddress.getAllByName("61.135.169.105");
        InetAddress[] a = InetAddress.getAllByName("jzx");
        for(InetAddress x:a)
              System.out.println(x);
   }
}
```

三、URL 类

URL：Uniform Resource Locator，统一资源定位符。用于从主机上读取资源。（只能读取，不能向主机写）

（1）一个 URL 地址通常由 4 部分组成：

协议名：如 http、ftp、file 等

主机名：如 www.baidu.com、220.181.112.143 等

路径文件：如/java/index.jsp

端口号：如 8080、8081 等

（2）URL 类的常用方法：

String getFile()：获取 URL 的文件名，它是带路径的文件标识；

String getHost()：获取 URL 的主机名；

String getPath()：取得 URL 的路径部分；

int getPort()：取得 URL 的端口号；

URLConnection openConnection()：返回代表与 URL 进行连接的 URLConnection 对象；

InputStream openStream()：打开与 URL 的连接，返回来自连接的输入流；

Object getContent()：获取 URL 的内容。

例题：通过操作读取 URL 访问结果。

```
import java.io.*;
import java.net.*;
public class GetURL{
    public static void main(String[] args){
        InputStream in=null;
        OutputStream out=null;
        try{
            if((args.length!=1)&&(args.length!=2))
            throw new IllegalArgumentException("参数的个数不对！ ");
            URL url=new URL(args[0]);
            in=url.openStream();
            if(args.length==2)
                out=new FileOutputStream(args[1]);
            else
                out=System.out;//重定向到屏幕
            byte[] buffer=new byte[4096];
            int bytes_read;
            while((bytes_read=in.read(buffer))!=-1)
                out.write(buffer,0,bytes_read);//将指定 buffer 数组中从偏移量 0 开始的 bytes_read 个字节写入此输出流。
        }
        catch(Exception e){
            System.err.println("Usage:java GetURL<RUL>[<filename>]");
        }
        finally{
            try{
               in.close();
              out.close();
            }
            catch(Exception e){
                System.out.println(e);
```

```
            }
        }
    }
}
```

修改上面的程序，使用字符流。

```
import java.io.*;
import java.net.*;
public class GetURLS{
    public static void main(String[] args){
        BufferedReader in=null;
        BufferedWriter out=null;
        try{
            if((args.length!=1)&&(args.length!=2))
              throw new IllegalArgumentException("参数的个数不对！");
            URL url=new URL(args[0]);
    in=new BufferedReader(new InputStreamReader(url.openStream()));
            if(args.length==2)
                    out=new BufferedWriter(new OutputStreamWriter(new
FileOutputStream(args[1])));
            /*
            else
                out=System.out;//重定向到屏幕,类型不兼容
            */
            String s;
            while((s=in.readLine())!=null)
                out.write(s);
                 out.write("\n\r");
        }
        catch(Exception e){
            System.err.println("Usage:java GetURL<RUL> <filename>");
        }
        finally{
            try{
               in.close();
```

```
                out.close();
            }
            catch(Exception e){
                System.out.println(e);
            }
        }
    }
}
```

四、URLConnection 类

URLConnection 类可实现与 **URL** 资源双向通信。

它代表应用程序和 **URL** 之间的通信链接。此类的实例可用于读取和写入此 **URL** 引用的资源。通常，创建一个到 **URL** 的连接需要几个步骤：

（1）通过在 **URL** 上调用 **openConnection** 方法创建连接对象。

（2）处理设置参数和一般请求属性。

（3）使用 **connect** 方法建立到远程对象的实际连接，或者使用 **URL** 类的 **openConnection()**方法建立实际连接。

（4）远程对象变为可用。远程对象的头字段和内容变为可访问。

例题：下载指定的 **URL** 文件。

```
import java.net.*;
import java.io.*;
public class DownloadFile{
    public static void main(String[] args){
        try{
            URL path=new URL(args[0]);
            saveFile(path);
        }
        catch(MalformedURLException e){
            System.err.println("URL error");
        }
    }
    public static void saveFile(URL url){
        try{
            URLConnection uc=url.openConnection();
```

```
                int len=uc.getContentLength();
                InputStream stream=uc.getInputStream();
                byte[] b=new byte[len];
                stream.read(b,0,len);
                String theFile=url.getFile();
                theFile=theFile.substring(theFile.lastIndexOf('/')+1);
                FileOutputStream fout=new FileOutputStream(theFile);
                fout.write(b);
            }
            catch(Exception e){
                System.out.println(e);
            }
        }
    }
```

F:/javapro/java DownloadFile http://localhost:8080/mxjg/images/header.jpg

★ 在本地服务器目录下必须存在 **mxjg/images/header.jpg**

如果是本地硬盘上的文件可以使用文件协议 **file**

如：**F:/javapro/java DownloadFile file://localhost/mxjg/images/header.jpg** 须存在 **mxjg/images/header.jpg**

五、Socket 络通信

Java 提供了 **Socket** 类和 **ServerSocket** 类分别用于 **Client** 端和 **Server** 端的 **Socket** 通信。

（1）Socket 类

构造方法：

① **Socket(String, int)**：构造一个指定主机，指定端口号的 **Socket**。

② **Socket(InetAddress, int)**：构造一个指定 **Internet** 地址，指定端口号的 **Socket**。

（2）ServerSocket 类

① **ServerSocket(int)**：创建绑定到特定端口的服务器套接字。

② **ServerSocket(int，int)**：创建服务器套接字并将其绑定到指定的本地端口号，其中第二个参数是监听时间的长度。

（3）建立连接与数据通信

首先，在服务器端创建一个 **ServerSocket** 对象，此对象通过执行 **accept()**

方法监听客户端连接，此时服务器端线程处于等待状态。然后在客户端构造 Socket，与某服务器的指定端口进行连接。服务器监听到连接请求后，就可在两者之间建立连接，连接建立后，就可以取得相应的输入、输出流进行通信。

例题 1：一个简单的 Socket 通信演示程序。

服务器端程序：

```
import java.net.*;
import java.io.*;
public class SimpleServer{
    public static void main(String[] args){
        ServerSocket s=null;
        try{
            s=new ServerSocket(5432);
        }
        catch(IOException e){}
        while(true){
            try{
                 Socket s1=s.accept();
                OutputStream slout=s1.getOutputStream();
                DataOutputStream dos=new DataOutputStream(slout);
                dos.writeUTF("Hello World!");
                System.out.println("a client is conneted...");
                s1.close();
            }
            catch(IOException e){}
        }
    }
}
```

客户端程序：

```
import java.net.*;
import java.io.*;
public class SimpleClient{
    public static void main(String[] args)throws IOException{
        Socket s=new Socket("localhost",5432);
        InputStream sIn=s.getInputStream();
```

```
            DataInputStream dis=new DataInputStream(sIn);
            String message=new String(dis.readUTF());
            System.out.println(message);
            s.close();
        }
}
```

例题 2：另一个 Socket 通信程序。

服务器端程序：

```
import java.io.*;
import java.net.*;
public class JabberServer {
  // Choose a port outside of the range 1-1024:
  public static final int PORT = 8080;
  public static void main(String[] args)
      throws IOException {
    ServerSocket s = new ServerSocket(PORT);
    System.out.println("Started: " + s);
    try {
      // Blocks until a connection occurs:
      Socket socket = s.accept();
      try {
        System.out.println(
          "Connection accepted: "+ socket);
        BufferedReader in =
          new BufferedReader(
            new InputStreamReader(
              socket.getInputStream()));
        // Output is automatically flushed
        // by PrintWriter:
        PrintWriter out =
          new PrintWriter(
            new BufferedWriter(
              new OutputStreamWriter(
                socket.getOutputStream())),true);
```

```
            while (true) {
               String str = in.readLine();
               if (str.equals("END")) break;
               System.out.println("Echoing: " + str);
               out.println(str);
            }
         // Always close the two sockets...
         } finally {
            System.out.println("closing...");
            socket.close();
         }
      } finally {
         s.close();
      }
   }
}
```

客户端程序：

```
import java.net.*;
import java.io.*;
public class JabberClient {
   public static void main(String[] args)
         throws IOException {
      // Passing null to getByName() produces the
      // special "Local Loopback" IP address, for
      // testing on one machine w/o a network:
      InetAddress addr =
         InetAddress.getByName(null);
      // Alternatively, you can use
      // the address or name:
      // InetAddress addr =
      //       InetAddress.getByName("127.0.0.1");
      // InetAddress addr =
      //       InetAddress.getByName("localhost");
      System.out.println("addr = " + addr);
```

```
        Socket socket =
          new Socket(addr, JabberServer.PORT);
        // Guard everything in a try-finally to make
        // sure that the socket is closed:
        try {
          System.out.println("socket = " + socket);
          BufferedReader in =
            new BufferedReader(
              new InputStreamReader(
                socket.getInputStream()));
          // Output is automatically flushed
          // by PrintWriter:
          PrintWriter out =
            new PrintWriter(
              new BufferedWriter(
                new OutputStreamWriter(
                  socket.getOutputStream())),true);
          for(int i = 0; i < 10; i ++) {
            out.println("howdy " + i);
            String str = in.readLine();
            System.out.println(str);
          }
          out.println("END");
        } finally {
          System.out.println("closing...");
          socket.close();
        }
      }
    }
```

★ 服务多个客户

JabberServer 可以正常工作，但每次只能为一个客户程序提供服务。在典型的服务器中，我们希望同时能处理多个客户的请求。解决这个问题的关键就是多线程处理机制。

最基本的方法是在服务器（程序）里创建单个 **ServerSocket**，并调用 **accept()**

来等候一个新连接。一旦 **accept()**返回，我们就取得结果获得的 **Socket**，并用它新建一个线程，令其只为那个特定的客户服务。然后再调用 **accept()**，等候下一次新的连接请求。

对于下面这段服务器代码，大家可发现它与 **JabberServer.java** 例子非常相似，只是为一个特定的客户提供服务的所有操作都已移入一个独立的线程类中：

服务器端程序：

```
import java.io.*;
import java.net.*;
class ServeOneJabber extends Thread {
  private Socket socket;
  private BufferedReader in;
  private PrintWriter out;
  public ServeOneJabber(Socket s)
      throws IOException {
    socket = s;
    in =
      new BufferedReader(
        new InputStreamReader(
          socket.getInputStream()));
    // Enable auto-flush:
    out =
      new PrintWriter(
        new BufferedWriter(
          new OutputStreamWriter(
            socket.getOutputStream())), true);
    // If any of the above calls throw an
    // exception, the caller is responsible for
    // closing the socket. Otherwise the thread
    // will close it.
    start(); // Calls run()
  }
  public void run() {
    try {
      while (true) {
```

```
            String str = in.readLine();
            if (str.equals("END")) break;
            System.out.println("Echoing: " + str);
            out.println(str);
          }
          System.out.println("closing...");
        } catch (IOException e) {
        } finally {
          try {
            socket.close();
          } catch(IOException e) {}
        }
      }
}

public class MultiJabberServer {
    static final int PORT = 8080;
    public static void main(String[] args)
          throws IOException {
        ServerSocket s = new ServerSocket(PORT);
        System.out.println("Server Started");
        try {
          while(true) {
            // Blocks until a connection occurs:
            Socket socket = s.accept();
            try {
              new ServeOneJabber(socket);
            } catch(IOException e) {
              // If it fails, close the socket,
              // otherwise the thread will close it:
              socket.close();
            }
          }
        } finally {
```

```
            s.close();
        }
    }
}
```

客户端程序：

```
import java.net.*;
import java.io.*;
class JabberClientThread extends Thread {
    private Socket socket;
    private BufferedReader in;
    private PrintWriter out;
    private static int counter = 0;
    private int id = counter++;
    private static int threadcount = 0;
    public static int threadCount() {
        return threadcount;
    }
    public JabberClientThread(InetAddress addr) {
        System.out.println("Making client " + id);
        threadcount++;
        try {
            socket =
                new Socket(addr, MultiJabberServer.PORT);
        } catch(IOException e) {
            // If the creation of the socket fails,
            // nothing needs to be cleaned up.
        }
        try {
            in =
                new BufferedReader(
                    new InputStreamReader(
                        socket.getInputStream()));
            // Enable auto-flush:
            out =
```

```
            new PrintWriter(
              new BufferedWriter(
                new OutputStreamWriter(
                  socket.getOutputStream())), true);
        start();
      } catch(IOException e) {
        // The socket should be closed on any
        // failures other than the socket
        // constructor:
        try {
          socket.close();
        } catch(IOException e2) {}
      }
      // Otherwise the socket will be closed by
      // the run() method of the thread.
    }
    public void run() {
      try {
        for(int i = 0; i < 25; i++) {
          out.println("Client " + id + ": " + i);
          String str = in.readLine();
          System.out.println(str);
        }
        out.println("END");
      } catch(IOException e) {
      } finally {
        // Always close it:
        try {
          socket.close();
        } catch(IOException e) {}
        threadcount--; // Ending this thread
      }
    }
}
```

```
public class MultiJabberClient {
  static final int MAX_THREADS = 40;
  public static void main(String[] args)
      throws IOException, InterruptedException {
    InetAddress addr =
      InetAddress.getByName(null);
    while(true) {
      if(JabberClientThread.threadCount()
          < MAX_THREADS)
        new JabberClientThread(addr);
      Thread.currentThread().sleep(100);
    }
  }
}
```

六、无连接的数据报

数据报是一种无连接的通信方式，它的速度比较快，但是由于不建立连接，不能保证所有数据都能送到目的地，一般用于传送非关键性的数据。

（1）DatagramPacket 类

此类是进行数据通信的基本单位，包含需要传送的数据、数据报的长度、IP 地址和端口号等信息。其构造方法如下：

① DatagramPacket(byte[] buf, int length) ：构造 DatagramPacket，用来接收长度为 length 的数据包。

② DatagramPacket(byte[] buf, int length, InetAddress address, int port)：构造数据报包，用来将长度为 length 的包发送到指定主机上的指定端口号。

③ DatagramPacket(byte[] buf, int offset, int length)：构造 DatagramPacket，用来接收长度为 length 的包，在缓冲区中指定了偏移量。

④ DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)：构造数据报包，用来将长度为 length 偏移量为 offset 的包发送到指定主机上的指定端口号。

⑤ DatagramPacket(byte[] buf, int offset, int length, SocketAddress address)：构造数据报包，用来将长度为 length 偏移量为 offset 的包发送到指定主机上的指定端口号。

⑥ DatagramPacket(byte[] buf, int length, SocketAddress address)：构造数

据报包，用来将长度为 **length** 的包发送到指定主机上的指定端口号。

★ 常用函数：

void setData(byte[] buf)：设置数据缓冲区。

byte[] getData()：返回数据缓冲区。

getLength()：返回发送或接收数据报的长度。

（2）**DatagramSocket** 类

DatagramSocket 类是用来发送或接收数据报的 **Socket**，其构造方法如下：

① **DatagramSocket()**：构造一个用于发送的 **DatagramSocket**。

② **DatagramSocket(int port)**：构造一个用于接收的 **DatagramSocket**，参数为接收端口号。

（3）发送和接收过程

下面代码给出了数据报接收和发送的编程要点，接收端的 **IP** 地址是 **192.168.0.3**，端口号是 **80**，发送的数据在缓冲区 **message** 中，长度为 **200**。

接收端的程序：

```
byte[] inbuffer=new byte[1024];//设置缓冲区
DatagramPacket inpacket=new DatagramPacket(inbuffer,inbuffer.length);
DatagramSocket insocket=new DatagramSocket(80);//设置端口号
insocket.receive(inpacket);//接收数据报
String s=new String(inbuffer,0,0,inpacket.getLength());//将接收的数据存入字符串。
```

② 发送端程序：

```
//message 为存放发送数据的字节数组
DatagramPacket outpacket=new DatagramPacket(message,200,"192.168.0.3",80);
DatagramSocket outsocket=new DatagramSocket();
outsocket.send(outpacket);
```

例题：利用数据报发送信息或文件

★发送程序：

```
import java.io.*;
import java.net.*;
public class UDPSend{
    public static final String usage=" 用 法 ： java UDPSend <hostname><port><msg>...\n"+"或 java UDPSend<hostname><port>-f<file>";
    public static void main(String[] args){
```

```
                try{
                  if(args.length<3)
                       throw new IllegalArgumentException("参数个数不对");
                  String host=args[0];
                  int port=Integer.parseInt(args[1]);
                  byte[] message;
                  if(args[2].equals("-f")){
                       File f=new File(args[3]);
                      int len=(int)f.length();
                      message=new byte[len];
                      FileInputStream in=new FileInputStream(f);
                      int bytes_read=0;
                      in.read(message,bytes_read,len);
                  }
                  else{
                       String msg=args[2];
                      for(int i=3;i<args.length;i++)
                           msg+=""+args[i];
                      message=msg.getBytes();
                  }
                  InetAddress address=InetAddress.getByName(host);
                  DatagramPacket  packet=new  DatagramPacket(message,messag
e.length,address,port);
                  DatagramSocket dsocket=new DatagramSocket();
                  dsocket.send(packet);
                  dsocket.close();
              }
              catch(Exception e){
                 System.err.println(e);
                 System.err.println(usage);
              }
          }
      }
```

★接收程序：

```
import java.io.*;
import java.net.*;
public class UDPReceive{
    public static final String usage="用法：java UDPReceive<port>";
    public static void main(String[] args){
        try{
            if(args.length!=1)
                throw new IllegalArgumentException("参数个数不足");
            int port=Integer.parseInt(args[0]);
            DatagramSocket dsocket=new DatagramSocket(port);
            byte[] buffer=new byte[2048];
    DatagramPacket  packet=new  DatagramPacket(buffer,buffer.length);
            for(;;){
                dsocket.receive(packet);
               String msg=new String(buffer,0,packet.getLength());

System.out.println(packet.getAddress().getHostName()+":"+msg);
            }
        }
        catch(Exception e){
            System.err.println(e);
            System.err.println(usage);
        }
    }
}
```

（4）数据报多播

多播就是发送一个数据报文，所有组内成员均可接收到。多播通信使用 D 类 IP 地址，地址范围：224.0.0.1~239.255.255.255。

发送广播的主机给指定多播地址的特定端口发送消息。

接收广播的主机必须加入到同一多播地址指定的多播组中，并从同样端口接收数据报。

★ MulticastSocket 类

构造函数：

① MulticastSocket()：创建一个多播 Socket，可用于发送多播消息。

② **MulticastSocket(int port)**：创建一个与指定端口绑定的多播 **Socket**，可用于收发多播消息。

★ 通过 **setTimeToLive(int)**方法设置多播消息的生命周期，默认消息的生命周期为 **1**，这种情况下，消息只能在局域网内部传递。

★ 接收和发送多播数据：

① 接收多播数据

接收方首先通过使用发送方数据报指定的端口创建一个 **MulticastSocket** 对象，通过该对象调用 **joinGroup(InetAddress group)**方法将自己登记到一个多播组中。然后，就可以使用 **MulticastSocket** 对象的 **receive()**方法接收数据报了。在不接收数据时，可以调用 **leaveGroup(InetAddress group)**方法离开多播组。

② 发送多播数据

用 **MulticastSocket** 对象的 **send()**方法发送数据报。由于在发送数据报中已指定了多播地址和端口，发送方创建 **MulticastSocket** 对象时可以使用不指定端口的构造方法。

③ 实现数据多播的关键代码：

```
String msg="Hello";
InetAddress group=InetAddress.getByName("228.5.6.7");//创建多播组
MulticastSocket s=new MulticastSocket(6789);//创建 MulticastSocket 对象
s.joinGroup(group);//加入多播组
DatagramPacket hi=new DatagramPacket(msg.getBytes(),msg.length(),group,6789);//创建要发送的数据报
s.send(hi);//发送
/*下面接收多播数据报*/
byte[] buf=new byte[1024];
DatagramPacket recv=new DatagramPacket(buf,buf.length);
s.receive(recv);
```

例题：广播信息的主机程序

```
import java.net.*;
public class BroadCast extends Thread{
    String s="天气预报,最高温度 32 度,最低温度 25 度";
     int port=5858;//组播的端口
```

```
    InetAddress group=null;//组播组的地址
    MulticastSocket socket=null;//多点广播套接字
    BroadCast(){
     try{
          //设置广播组的地址为 239.255.8.0
          group=InetAddress.getByName("239.255.8.0");
         //多点广播套接字将在 port 端口广播
          socket=new MulticastSocket(port);
          //多点广播套接字发送数据报范围为本地网络
          socket.setTimeToLive(1);
         /*加入广播组,加入 group 后,socket 发送的数据报可以被加入到
group 中的成员接收到*/
          socket.joinGroup(group);
       }
       catch(Exception e){
         System.out.println("Error: "+ e);
       }
    }
    public void run(){
       while(true){
          try{
            DatagramPacket packet=null;//待广播的数据包
              byte data[]=s.getBytes();
              packet=new DatagramPacket(data,data.length,group,port);
              System.out.println(new String(data));
              socket.send(packet); //广播数据包
              sleep(2000);
            }
            catch(Exception e){
              System.out.println("Error: "+ e);
            }
       }
    }
    public static void main(String args[]){
```

```
        new BroadCast().start();
    }
}
```

接收广播的主机程序：

```
import java.net.*;
import java.awt.*;
import java.awt.event.*;
public class Receive extends Frame implements Runnable,ActionListener{
   int port;//组播的端口.
   InetAddress group=null;//组播组的地址.
   MulticastSocket socket=null;//多点广播套接字.
   Button  开始接收,停止接收;
   TextArea  显示正在接收内容,显示已接收的内容;
   Thread thread; //负责接收信息的线程.
   boolean  停止=false;
   public Receive(){
       super("定时接收信息");
       thread=new Thread(this);
       开始接收=new Button("开始接收");
       停止接收=new Button("停止接收");
       停止接收.addActionListener(this);
       开始接收.addActionListener(this);
       显示正在接收内容=new TextArea(10,10);
       显示正在接收内容.setForeground(Color.blue);
       显示已接收的内容=new TextArea(10,10);
       Panel north=new Panel();
       north.add(开始接收);
       north.add(停止接收);
       add(north,BorderLayout.NORTH);
       Panel center=new Panel();
       center.setLayout(new GridLayout(1,2));
       center.add(显示正在接收内容);
       center.add(显示已接收的内容);
       add(center,BorderLayout.CENTER);
```

```
        validate();
        port=5858;//设置组播组的监听端口
        try{
    //设置广播组的地址为 239.255.8.0
    group=InetAddress.getByName("239.255.8.0");
    //多点广播套接字将在 port 端口广播
          socket=new MulticastSocket(port);
         /*加入广播组,加入 group 后,socket 发送的数据报可以被加入到
group 中的成员接收到*/
          socket.joinGroup(group);
        }
        catch(Exception e){}
        setBounds(100,50,360,380);
        setVisible(true);
        addWindowListener(new WindowAdapter(){
          public void windowClosing(WindowEvent e){
              System.exit(0);
            }
        });

      }
    public void actionPerformed(ActionEvent e){
        if(e.getSource()==开始接收){
          开始接收.setBackground(Color.blue);
            停止接收.setBackground(Color.gray);
            if(!(thread.isAlive())){
              thread=new Thread(this);
            }
            try{
              thread.start();
                停止=false;
              }
            catch(Exception ee) {}
         }
```

```
        if(e.getSource()==停止接收){
            开始接收.setBackground(Color.gray);
              停止接收.setBackground(Color.blue);
              停止=true;
            }
        }
        public void run(){
            while(true){
              byte data[]=new byte[8192];
                DatagramPacket packet=null;
              //待接收的数据包。
                packet=new DatagramPacket(data,data.length,group,port);
                try {
                   socket.receive(packet);
             String message=new String(packet.getData(),0,packet.getLength());
                       显示正在接收内容.setText("正在接收的内容:\n"+message);
                       显示已接收的内容.append(message+"\n");
                    }
                catch(Exception e) {}
                if(停止==true){
                   break;
                }
            }
        }
        public static void main(String args[]){
            new Receive();
        }
    }
```

作业：

（1）编写 java 程序，利用 URL 对象读取网络上文件的内容。

（2）编写程序实现：客户端（Client.java）向服务器（Server.java）端请求（请求是一句话，为一个 String）。如果这句话的内容字符串是字符串“plain”

的话，服务器仅将“Hello”字符串返回给用户。否则将用户的话加到当前目录的文本稳健 Memo.txt 中，并向用户返回“OK”。

（3）让（2）中的 Server.java 能并发的处理多用户，并编写程序模拟多个用户向服务器发送请求。

（4）用一个套接字(Socket)完成，由客户端指定一个服务器上的文件名，让服务器发回该文件的内容，或者提示文件不存在。

（5）编写程序，用面向连接的网络通信实现一个远程加法器；客户端向服务器发送两个数；服务器计算两个数的和，返回给客户端。分别写出客户端和服务器端的程序。

第十三章 Swing 编程

一、Swing 包简介

（1）Swing 组件的特性

（2）Swing 的功能分类

Swing 组件从功能上可以分为如下几种：

① 顶层容器：**JFrame、JApplet、JDialog、JWindow。**

② 中间容器：**JPanel、JScrollPane、JSplitPane、JToolBar。**

③ 特殊容器：**JInternalFrame JLayeredPane JRootPane**

④ 基本控件：**JButton JComboBox JList JMenu JSlider JTextField**

⑤ 不可编辑信息的显示：**JLabel JProgressBar JToolTip**

⑥ 可编辑信息的显示：**JColorChooser JFileChooser JTextArea**

二、Swing 包典型部件的使用

（1）JFrame 类

（2）JApplet

（3）Swing 中的按钮和标签

（4）滚动窗格

（5）工具栏

（6）Swing 中对话框

（7）选项卡

（8）表格

（9）树