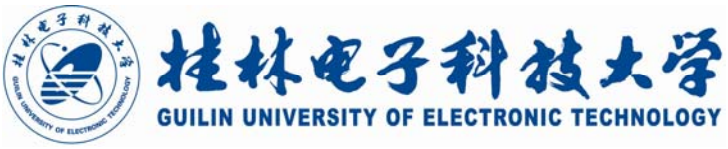


Java 程序设计实验指导书

桂林电子科技大学计算机科学与工程学院

汪华登 编著

修订日期：2013.3.7

前　言

面向对象是当今最流行的软件系统设计和开发思想。Java 作为一门完全面向对象的程序设计语言，具有可移植性强、健壮安全、支持多线程与分布式系统设计等优点，已在很多大中型企业和移动设备的应用系统开发中广泛应用。Java 语言的以上特点，使得它成为当前的主流软件开发语言之一。作为推出 Java 这门语言及其开发包 JDK 的 SUN 公司（现已被 Oracle 公司合并），它所推出的关于 Java 程序设计的 SCJP(Sun 认证 Java 程序员)以及 SCJD(Sun 认证 Java 开发员)等认证也广受业界认可。目前的全国计算机水平考试和计算机等级考试也都设置了 Java 语言语种。由于 Java 具有完全面向对象的特性而语法又相对比 C++语言简单，它也越来越多地被用在面向对象程序设计的教学当中，避免了学生因 C++语法太难而难以入门的缺点。

除了涉及桌面应用软件开发的 Java SE 开发技术（即以前的 J2SE）之外，以 Java 语言为基础的 Java EE(即以前的 J2EE)开发技术，通过 JSP、Servlet、JavaBean、EJB 等组件技术和 Struts、Spring、Hibernate 等框架技术的应用来实现企业版应用系统，在目前的企业级系统开发中占有重要的市场份额，与 Microsoft 公司的.Net 平台一起成为企业级系统开发的两大主流方向。此外以 Java 语言为基础的 Java ME（即以前的 J2ME）开发技术，则广泛应用在手机、信息家电等各类移动设备中。

Java 语言最大的特点是入门容易，但要想掌握它所被主要应用的 Java EE 和 Java ME 两个方向的开发技术，则需要付出很大的努力。尤其是 Java EE，它所涉及到的语法以及组件、容器和框架技术非常多，知识点庞杂，并要求具有 Java 语言、XML、数据库、计算机网络、数据结构以及 UML 等课程和开发技术作为基础。因此要想真正学好 Java，是需要下一番功夫的。

我们目前学习的 Java 程序设计，只是掌握 Java 语言本身，重在语法和面向对象编程思想。根据近几年来承担 Java 程序设计实践教学的经验和对桂电计算机学院 7 个计算机相关专业同学们学习状况的了解，我们编写了这本实验指导书。

本指导书的特点和希望达到的目的是：

1、每个实验项目中都会给大家提供完成实验内容所需的知识储备和讲解，并配合特别设计的典型示例，给大家提供完成实验所需的各种引导和指导，使同学们凭借本指导书书即可基本满足实验所需。本实验教材独自成书，无论理论教材怎么更换，基本不会影响同学们完成实验。

2、实验项目和题目尽量达到难易适当和均衡，在几年来的实践教学中不断检验和完善。

3、每个实验都特别编写了预备知识部分，对完成该次实验所需要了解的理论知识点尽量进行简练的讲述。使得大家基本不需要再查阅过多的理论教材和其它参考书，凭借本实验教材即可基本了解所需的预备知识。

4、每个实验的预备知识部分都设计和安排了各种典型示例。这些示例前面均有详细文字说明，示例代码有详细注释。这些示例会详细告诉大家例子的目的、所要解决的问题，在例子当中展现完成实验项目所需的知识点的应用，并展示程序设计的规范、方法和思想。这些例子虽然并不直接告诉大家该怎么完成实验题目，但同学们在阅读、模仿这些例子程序的过程中，会潜移默化地学到相应所需掌握的知识点和设计方法，自然而言地能够想到解决实验题目的方法。

5、实验内容安排上，注重从基础到复杂，从易到难，在前面的实验中会提前把后面实验所需的知识进行铺垫。使得大家的学习具有渐进性。杜绝突兀的难点和无法下手的情况，尽量做到前后连贯，统筹兼顾。

6、关注 java 语言的版本变化和 java 语言权威认证的考点覆盖范围，在有限的学时内尽量使实验项目内容的取舍和安排更合理。

因学时所限，Java 程序设计中的集合框架等部分重要内容在实验教材和课堂上无法全部涉及，大家可以在课余时间继续扩大阅读面和学习深度。

目　录

《Java 语言程序设计》实验要求和注意事项（根据学校相关教学管理规定）

1、 **预习报告：**每次实验前必须先写好预习报告，在讲解完该次实验后老师将逐个检查并记录预习报告成绩作为平时成绩的参考。实验的预习报告主要是该次实验内容所对应的程序，而且必须是对应该次实验内容完整的、全部的程序，仅仅从所给的指导资料或他人处抄写一部分代码或算法的视为无效。程序可手写或打印在普通的稿纸上，也可用 U 盘等工具带过来以电子版的形式展示。在实验完后再将完整的实验程序手写或打印在正规的实验报告纸上，附录在最终的实验报告的后面。

2、 **实验报告：**按照学校实验规定，每个实验最后都要交一份实验报告。大家每做完一个实验，回去后即利用课余时间写好实验报告。实验报告基本格式参考如下：

一、 实验目的与内容。

二、 分析设计过程：主要包括自己对题目的分析，自己的设计思想和理由，自己的设计方法等。

三、 测试数据和程序运行的结果：使用了哪些典型数据进行了一些怎样的测试，运行结果如何，程序的交互性如何。

四、 问题与总结：所遇到的具体问题和分析解决过程，尚存在的一些问题，所获得的与课程相关、技术相关的心得与体会。

五、 附录： 包括完整、正确的程序代码（若代码较多可截取关键代码段）及注释，特别是主要的算法或代码必须有适当的注释，使得他人可凭借注释较快地读懂该程序。无注释的程序无成绩。

【注意】: **最终的实验报告可打印，但必须使用正规实验报告纸或 A4 大小的纸张。报告雷同者成绩为不及格！**

3、 **实验验收：**无论是预习报告还是最后的实验报告均会被打分，均严禁抄袭、雷同和潦草，否则会酌情扣分直至无成绩。凡无预习报告者，老师可按实验教学规定给予该次实验成绩为 0 分。每次实验原则上应当堂完成，做出来后可请老师验收，并获得动手能力这一项的成绩，原则上每次实验时只能验收当次实验以及前 1 次实验，验收时会适当询问，非自己完成以及自己不懂者请勿申请验收，浪费彼此的时间，请抓紧时间及时自己完成。

4、 **课堂纪律：**请勿迟到早退。上课时请不要听歌或聊天，不要带耳塞，不要在实验室内吃东西和扔任何形式的废弃物。未经允许不能随便调整实验批次，有客观原因不能上课者必须有医院、年级主任等单位或领导的书面证明和签字盖章，随意口头请假者一般均视为无效。

JAVA语言程序设计的预备知识

这里的“预备知识”是贯穿于 java 程序设计始终的一些必须要注意的思想和方法，如果对这些方面不了解、不注意，那么很可能一动手写出来的 java 程序就有错误，根本不可能通过编译，直接造成信心和兴趣上的打击。所以请大家务必先建立起这些概念。

一、java 程序从编写到运行的一般过程和原理：先由编译器编译成后缀为 class 的字节码文件，然后由 JVM(java 虚拟机)来解释运行，可移植于不同的操作系统平台上（只要有 java 运行环境）。

二、java 的特点：从 C++演变而来，保留其优点，去除了易产生错误的功能，简化了内存管理。Java 中没有虚函数、多继承、模板、运算符重载等 C++中相对较难的语法，其基本语法比 C++简单了很多。具体特点有：完全面向对象、平台独立性、安全性（不支持指针，一切对内存的访问都必须通过对象的实例变量来实现）、网络功能、数据库功能、内建的多线程功能等。

三、程序设计过程：先用文本编辑器如记事本、UltraEdit 编辑器、EditPlus 编辑器等文本编辑工具或者用其它的 IDE（集成开发环境）软件所带的编辑功能，来编写好源代码。存成后缀为 java 的文件。然后用 javac 和 java 命令，或其它 IDE 的编译运行功能选项，对程序进行编译、运行。

四、常用的集成开发环境（IDE）软件有：Eclipse、NetBeans、JBuilder、IntelJ、Jcreater、BlueJ 等，其中前三种为实际开发中常用的中大型开发工具，后三种为小型、教学型编译器。Eclipse 来源于 IBM 等公司的开源项目，它是基于插件的开发环境，除可支持 java 开发之外，如果安装了相应的插件，也可支持其它诸如 C++等的开发，因其优异的性能和开源的特点，现在已经成为非常知名的开发平台。其官方站点是 www.eclipse.org ，下载后解压即可使用。NetBeans 为 Sun 公司（现已被知名的甲骨文公司（ORACLE）收购）自己推出的编译器。其特点是免费下载使用，版本更新快，汉化版也能及时推出，并且有各种集成 Tomcat 等服务器的版本，安装方便。JBuilder 为 Borland 公司产品，其功能也较强较全面，但比较庞大，属于不开源的商业软件。本实验指导书推荐使用 Eclipse。

五、关于 java 程序的结构。对于 java 的 Application（应用程序）而言，一个 java 程序中最多只能有一个（也可以没有）以 public 说明的类，而且当存在 public 类时，该文件保存时的名字必须和这个类的名称相同。而且注意，如果是在控制台下编译、执行过程中，编译命令 javac 后面应跟文件名，而执行命令 java 后面应该跟主类名。当然如果是在 IDE 环境中则不必考虑太多这个问题，只要编译成功，一般 IDE 软件都具有自动搜寻主类来执行的功能，但有时需要向 IDE 指明主类。java 程序文件的命名是学习者初始阶段经常容易出错的一个问题，会导致无法编译文件，所以请务必注意。另外请特别注意，Application 程序中是必须有 main 函数的，而且 main 函数除了其参数的名字可以改变以外，不能遗漏任何其它的修饰符。 下面通过两个基本示例程序程序命名的问题进行理解：

```
//以下为第一个示例程序Helloworld.java，用于练习。这是一个非常简单的java程序：
public class HelloWorld
{
    public static void main(String args[])
    {
        System.out.println(“Hello world!”);
    }
}
```

```
//以下为第二个示例程序，在此程序中，按前述规则文件名应为Hello2.java，但是主类名
//却是Hello1，所以在控制台环境下编译执行时候的命令应该为javac Hello2.java ，会
//生成Hello1.class和Hello2.class两个类，然后运行的时候的命令却是java  Hello1
public class Hello2
{
    int a1,a2;
    void pntOut(String s)
    {
        System.out.println(s);
    }
}
class Hello1
{
    public static void main (String args[ ])
    {
        Hello2 a_string;
        a_string=new Hello2();
        a_string.a1=1;
        a_string.a2=2;
        System.out.print("1+2 = ");
        System.out.println(a_string.a2+ a_string.a1);
        a_string.pntOut("hello world! ");
    }
}
```


请注意：java 编程中，文件名、系统和自定义的类名和标识符等都是区分大小写的！标识符是由字母、下划线或美元符开头，后接数字、字母等组成的字符序列，标识符长度不限，且不能和关键字（如 if、while、do、int 等）同名。注释符和 C++中一样有“/* 代码段 */”和“//代码行”两种。

实验一　Java开发环境及基本语法

一、实验目的

1、掌握常用的 Java 集成开发环境的使用，特别是 Eclipse 和 NetBeans。

2、掌握 Java 基本语法，重点是面向对象的思想和语法。

3、掌握控制台下（应用程序）的输入输出方法，作为后续部分实验的基础。

二、实验类型

设计型。

三、实验内容

1、打开实验室计算机上的集成开发环境 Eclipse 或 NetBeans（重点是 Eclipse），掌握其基本使用方法。了解开发软件的各个菜单功能。会创建 java 项目(Project)，会编辑和编译、运行项目代码。

2、验证和学习所给的几个例子程序及其讲解。然后自己编写一个至少由一个类构成的 Java 程序，其功能是在运行后，能接收用户输入一个学生的姓名以及 java 课的成绩（百分制），并输出对该学生成绩是否达到 60 分的及格分的判断（可使用 if 语句）。例如，输入学生姓名李明，然后输入成绩 50，则输出“李明的成绩为不及格”。

3、编写一个程序来表示长方体的长宽高等数据并有计算体积的函数可供调用，长方体类从矩形类继承而来。程序运行时能接受输入任意的 3 个数作为长方体的长、宽和高，然后能够输出所输入的长宽高及所计算出的体积。注意不能把代码全部写在 main 函数里面，不能直接输出长宽高的乘积。在解决该问题的程序设计中，将体积计算等功能封装成方法调用，长宽高等属性封装到矩形类和长方体类中，尽量多地用到接口的定义、类的定义、成员变量的定义、成员函数的定义、类的继承等面向对象的语法和知识点。在此基础上熟悉、理解和解释类的封装、继承等面向对象编程思想和概念。

四、预备知识

作为一门完全面向对象的程序设计语言，Java 语言编写的所有程序都是由类（以及接口）构成的。所以我们需要重点掌握类的定义、修饰，对象的创建，类的继承，以及接口等面向对象语法知识。Java 中没有模板、多继承、运算符重载等 C++中比较复杂的语法部分，其面向对象语法相对简单。

大家此前一般学过 C 或 C++语言等，那么请注意，由于 Java 是“完全”面向对象的语言，任何程序代码都是类或接口构成，所以请务必摒弃在 C 语言等的学习过程中形成的喜欢在 main 函数外部定义全局变量等习惯，因为这些做法在 Java 中将直接成为错误。Java 程序都只由接口和（或）类构成，类中才可定义普通数据类型或复合数据类型的成员变量。对于初学面向对象程序设计的学习者而言，类看起来很复杂，实际上我们应该注意到：类只是由两种东西构成的，一种是成员变量，另一种是成员函数（或称成员方法）。对于 Java 程序而言，只有在成员函数中，才可以编写和存在具体的执行语句。而在 Java 程序的类中的成员函数中，就涉及到数据类型和循环语句等基本语法。

Java 的基本语法与 C 语言是很接近的。同样有普通数据类型变量（在 Java 中包括整型、字符型、布尔型等）的定义，同样有 if 语句，if-else 语句，switch 语句，以及 while 循环，for 循环，do-while 循环。此处不单独举例，大家首先可在 main 函数中自己编写简单的代码对这些基本语法概念进行练习和验证。然后应用 Java 的面向对象语法，设计类和定义对象等，进行 Java 面向对象编程思想和语法的熟悉。

下面有多个例子程序给大家演示 Java 的部分基础语法以及如何进行数据输入和输出。

```
/*
 * 例子1.1，Output.java如下。本例告诉大家如何在控制台下输出数据。System.out对象的print方法
 * 输出数据和println方法输出数据有何区别。
 */
public class Output // Java程序都由类及接口构成，所以程序至少要定义一个类
{
    public static void main(String[] args) {
        System.out.println("这是一行会换行的输出。"); // println输出一串字符并换行
        System.out.print("这是一行不会换行的输出。");
        System.out.print("@这也是一行不会换行的输出，会跟前一行输出连在一起。发现了吗？");
    }
}
```

```
/*
 * 例子1.2，Input.java如下，本例介绍一种在控制台下接受用户输入数据的方法：使用
 * Scanner类中的方法来接收用户输入，大家可打开JDK的API帮助文档查看Scanner类及其
 * 所具有的方法的介绍。
 */
import java.util.Scanner;

public class Input {
    public static void main(String[] args) {
        String str; // 定义一个字符串变量str
        int i; // 定义一个整型变量i
        float f; // 定义一个浮点型变量f
        Scanner sc = new Scanner(System.in); // 定义一个Scanner对象，从System.in接受输入
        str = sc.next(); // 等待用户输入任意一个字符串，它会被存到str中
        System.out.println("你刚输入的一串字符是：" + str); // 输出字符串str
        i = sc.nextInt(); // 等待用户输入任意一个整数，它会被存到i中
        System.out.println("你刚输入的一个整数是：" + i); // 输出整数i
        f = sc.nextFloat(); // 等待用户输入任意一个浮点型的数，它会被存到str中
        System.out.println("你刚输入的一个浮点型的数是：" + f); // 输出浮点型数f

    }
}
```



```
/*
 * 例子1.3 ，Stream.java如下，下面介绍另一种接受用户输入的方法：该程序段涉及
 * 到java中“流”和“异常”的概念，大家可先使用做初步体会，后面的实验和讲解会
 * 逐步学习这方面内容。如下代码可以接受从键盘输入一字符串并存入字符串变量str中。
 */
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
```

```
public class Stream {
    static String str; // 定义辅助变量str用于存放用户输入的字符串

    public static void main(String[] args) {
        try {
            BufferedReader keyin = new BufferedReader(new InputStreamReader(
                    System.in));
            str = keyin.readLine(); // 运行后在此处会等待从键盘读取一个字符串并存放到str中。
        } catch (IOException e) {
        }
        System.out.println("你输入的字符串是" + str);
    }
}
```

```
/*
 * 例子1.4。也可通过JOptionPane类的showInputDialog和showMessageDialog之类的方法，
 * 来接收用户输入和进行信息提示。具体查阅API帮助文档的JOptionPane类。
 */
import javax.swing.JOptionPane;
public class Input {
    public static void main(String[] args) {
    String str1,str2;  // 定义字符串变量str1和str2
    str1 = JOptionPane.showInputDialog("请输入内容：");  //弹出接受用户输入的提示窗口
    System.out.println("你输入的内容是：" + str1);  //在控制台输出用户输入的内容进行验证
    str2 = JOptionPane.showInputDialog("另一种输入窗口","请输入内容：");  //另一种提示
System.out.println("你第二次输入的内容是：" + str2);//在控制台输出用户输入的内容进行验证
        JOptionPane.showMessageDialog(null, str2);   //在消息窗口显示第二次输入的内容
    }
}
```

```
/*
 * 例子1.5 ，类和对象，显式构造函数。 此例特别提醒大家，构造函数是没有返回类型的，连void
 *都不能放在函数声明前面，如果加了void，该函数会被当成普通函数，并因此可能造成整个程序编译
 *出错(当设计者还视该函数为构造函数时)。
 */
public class Hello2 {
    int a1, a2;

    public Hello2() {
        a1 = 1;
        a2 = 2;
    }
```

```
    void pntOut(String s) {
        System.out.println(s);
    }
}

class Hello1 {
    public static void main(String args[]) {
        Hello2 obj; // 定义一个hello2类的对象obj
        obj = new Hello2(); // 生成该对象实例。此时会调用对应的构造函数生成该对象，
                            // 使其成员变量初始化
        System.out.print("1+2 = ");
        System.out.println(obj.a2 + obj.a1);
        obj.pntOut("这是传递给obj对象的pntout函数的一个字符串，你会看到它被输出");
    }
}


/*
 * 例子1.6，关于类的创建、类的继承、封装等:
 * 编写一个JAVA应用程序，设计一个汽车类Vehicle，包含的属性有车轮个数wheels和车重weight。
 * 小汽车类Car是Vehicle类的子类，其中包含的属性有载人数loader。卡车类Truck是Car类的子类，
 * 其中包含属性有载重量payload。每个类都有相关数据的输出方法。
 * [解析]
 * 第一步：定义汽车类Vehicle
 * class Vehicle {
 *    int wheels;      //车轮数
 *    float weight;    //车重
 * }
 * 第二步：定义小汽车类Car
 * class Car extends Vehicle {
 * int loader;      //载人数
 * }
 * 第三步：定义卡车类Truck
 * class Truck extends Car {
 * float payload;    //载重量
 * }
 */
class Vehicle
{
    int wheels;      // 车轮数
    float weight;    // 车重
    Vehicle(int wheels,float weight)
    {
        this.wheels=wheels;
        this.weight=weight;
    }
```

```
    int getWheels()
    {
        return wheels;
    }
    float getWeight()
    {
        return weight;
    }
    void show()
    {
        System.out.println("车轮："+wheels);
        System.out.println("车重："+weight);
    }
}

class Car extends Vehicle
{
    int loader;     // 载人数
    Car(int wheels,float weight,int loader)
    {
        super(wheels,weight);   // 调用父类的构造函数，对从父类继承而来的成员变量进行初始化
        this.loader=loader;
    }
                                Java实验指导书（桂林电子科技大学汪华登）
    void show1()
    {
        System.out.println("车型：小车");
        super.show();     // 调用父类方法
        System.out.println("载人："+loader+"人");
    }
}

class Truck extends Car
{
    float payload;
    Truck(int wheels,float weight,int loader,float payload)
    {
        super(wheels,weight,loader);  // 调用父类构造方法
        this.payload=payload;
    }
    void show2()
    {
        System.out.println("车型：卡车");
        super.show();     // 调用父类方法
        System.out.println("载人："+loader+"人");
        System.out.println("载重量："+payload);
```

```
    }
}
public class VehicleClient
{
    public static void main(String args[])
    {
        System.out.println("输出相关数据");
        Car car=new Car(4,1500,4);  // 创建一个Car类对象
        //car.show1();
        Truck truck=new Truck(8,7000,3,25000);
        truck.show2();
    }
}
```

/* 例 1.5 的运行结果如下：*/

```
输出相关数据
车型：小车
车轮：4
车重：1500.0
载人：4人
车型：卡车
车轮：8
车重：7000.0
载人：3人
载重量：25000.0
```



```
/*
 * 例子1.7，关于接口的使用。Java中的接口(关键字interface)类似于C++中的抽象类，但又
 * 与其不同。接口如同一些预定义的规范一样可以由多个类去实现。 Java中的类只能继承自一个父
 * 类（即单继承），但却可以实现(关键字implements)多个接口，达到类似C++中多继承的效果。在
 * 接口中，只能有变量的定义（而且变量必须立即初始化）和函数的原型的声明（即函数没有函数体，
 * 直接以分号结束，实现该接口的类必须实现接口中所有的函数）。
 */
interface A1 {
    int aa = 100;

    float ab = 100.00f;

    void pntout(String ss);
}

class B1 implements A1 {
    public void pntout(String ss) {
        System.out.println(ss);
    }
```

```
}

public class inteface {
    public static void main(String args[]) {
        B1 abc;
        abc = new B1();
        System.out.println(abc.aa);
        System.out.println(abc.ab);
        abc.pntout("I am a teacher.");
    }
}
```

实验二　系统类和数组

一、实验目的

1、通过字符串处理类的应用，掌握系统类的使用方法。

2、掌握数组的定义和使用。

3、进一步掌握 Java 程序设计的基本过程和基本方法。

二、实验类型

设计型

三、实验内容

1、应用数组和字符串，编写一个用于统计学生成绩的程序，运行之后，首先能接受用户输入不超过 10 个学生的姓名和 Java 课的分数。输入结束之后，能输出这批学生的 Java 课最高分者和最低分者的姓名和分数。还能接受用户输入学生姓名来查询某学生的成绩，当姓名不存在时，向用户提示不存在该学生。

2、通过应用 Java 中常用的数组和字符串，编写一个简单的通讯录程序，来熟悉系统类和数组的应用。通讯录由多条记录构成。每条记录包括一个联系人的姓名、性别、电话、通信地址，并具有显示、查询、增加、修改、删除等功能，且每执行一次功能操作后，可以选择用类似以下的格式，输出通讯录的所有信息记录：

编号	姓名	性别	电话	通信地址
01	张三	男	123456	上海
02	李四	女	456789	桂林

可考虑用数组分别存储姓名，性别，电话，通信地址，那么显示、查询，增加、修改、删除操作就可转换为对数组元素的操作。通讯录中的所有记录的每一个字段可以采用一维或二维或其它类型数组来存放。 请发挥自己的思考能力，用自己熟悉的或者觉得更合理的方式来设计程序解决问题，完成该实验。

四、预备知识

面向对象是 Java 语言的重要特点，所有 Java 程序都必须是完全面向对象的，需要一个或多个类来完成某一个任务。如果每次要完成什么项目，这些类都要重头开始全都由开发者设计，其工作量很可能是很大的，这显然不现实。在前面的实验里面，我们自己定义了一些类来构成程序（我们称之为用户自定义类），也用到了一些系统提供的类（即系统类）来封装数据和实现功能（例如使用字符串 String 类定义字符串对象）。实际上系统类是 Java 面向对象程序设计的基础，系统类贯穿到所有 Java 程序的设计之中。Java 开发包中提供了非常多的系统类，而且除了 Sun（现已并入 Oracle）公司自己推出的标准开发包之外，很多其它公司或个人也开发了各种开发包用于在设计中的复用。这些包中的系统类有的是用于输入输出的，有的是用于网络程序设计的，有的是用于图形图像处理的，种类繁多，数量庞大，并被组织成很多个包来存放。对于非官方的开发包，我们需要在编译路径下导入。对于 JDK 本身提供的系统类，我们在具体需要用到它们的时候，只需要在程序的开头，用 import 关键字将其导入即可。常用的有“Java.lang” 包 、 ”java.io” 包 、 ”java.util” 包 、 ”java.net” 包 、 ”java.awt” 包、”java.applet”包等。大家可以打开 JDK API 帮助文档观察，其中就列出了几乎所有的包和类。

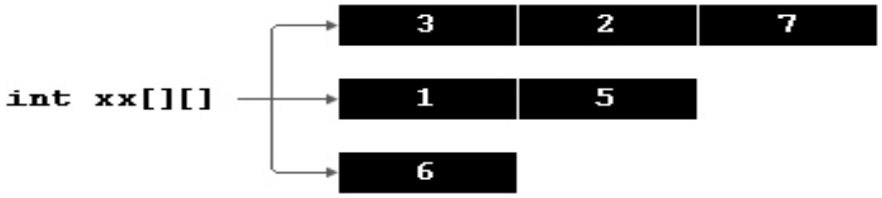
1、Java 中数组的定义与使用

（1）先定义，方法是：类型说明符 数组名[]； 或 类型说明符[] 数组名。

（2）然后再创建，方法是：数组名=new 类型说明符[数组元素的个数] 或 数组名={初值表列}；

也可以将上述两步合并为一步创建：类型说明符 数组名[]={初值表列}； 或 类型说明符[] 数组名={初值表列}；

注意 Java 中的数组可以是不规则的，即不一定是几行几列并且每行元素个数都一样多的。例如语句 int[][] xx={{3,2,7},{1,5},{6}}; 所定义出来的数组结构如下图：




```
/*
 * 例子2.1，关于一维数组的定义和初始化。
 */
public class ArrayTest1 {
    public static void main(String args[]) {
        int AB[]; // 第一步：定义数组AB
        AB = new int[3]; // 第二步：对数组AB进行实例化
        AB[0] = 1;
        AB[1] = 2;
        AB[2] = 3; // 赋值
        System.out.println(AB[0]); // 测试结果
        System.out.println(AB[1]);
        System.out.println(AB[2]);

        int AC[] = new int[2]; // 定义数组AC同时进行实例化
        AC[0] = 100;
        AC[1] = 200;
        System.out.println(AC[0]);
        System.out.println(AC[1]);

        int AD[] = { 10, 20, 30, 40 }; // 定义数组AD同时给定元素值
        System.out.println(AD[0]);
        System.out.println(AD[1]);
        System.out.println(AD[2]);
        System.out.println(AD[3]);

        System.out.println(AB.length); // 输出每个数组中的元素个数
        System.out.println(AC.length);
        System.out.println(AD.length);
    }
    }
```

```
/*例子2.2。 下面所特别设计的一个例子程序向大家展示定义和初始化二维数组的几种情况，以及这
 * 个过程中 容易犯的错误，请仔细运行体会一下，避免在完成本次实验题目的过程中犯同样的错误。
 * 另外，下面这个例子程序的运行结果为：
 * 数组num第1维的维数为2
```

```
 * 数组num第1行的元素个数3
 * 数组str第一维的维数为2
 * 数组str第1行的元素个数5
 * 数组str第2行的元素个数4
 */
public class ArrayTest {
    public static void main(String[] args) {
        char ch[][] = new char[3][4]; // 第一种情况：在定义的同时初始化多维数组

        int num[][]; // 第二种情况
        // num=new int[][]; //错！在初始化时必须至少确定数组num第一维的维数
        num = new int[2][];
        // num[0][0]=8; //此时这么写编译器不会报错，但运行时会提示有空指针异常
//(NullPointerException)
        // num[1][4]=8; //此时同上
        num[0] = new int[3];

        System.out.println("数组num第1维的维数为" + num.length);
        System.out.println("数组num第1行的元素个数" + num[0].length);
        // System.out.println("数组num第2维的元素个数"+num[1].length);
        // 此时这么写编译器不会报错，但运行时会提示有空指针异常(NullPointerException)产生,
        // 因为num[1]的元素个数还没确定

        String str[][] = new String[2][]; // 第三种情况：在定义的同时设定第一维为2
        str[0] = new String[5]; // 第一行有5个元素
        str[1] = new String[4]; // 第二行有4个元素

        System.out.println('\n' + "数组str第一维的维数为" + str.length);
        System.out.println("数组str第1行的元素个数" + str[0].length);
        System.out.println("数组str第2行的元素个数" + str[1].length);
    }
}


/*
 * 例子2.3，本例演示对象数组的定义和初始化问题。
 * 基本数据类型（包括byte、short、int、long、float、double、char、boolean）的数组元素会
 * 自动初始化成“空”值（对于数值，空值就是零；对于char，它是null；而对于boolean，是false）。
 * 在Java中对非基本数据类型初始化时，必须使用new。在使用new创建非基本数据类型的（对象 ）数组
 * 后，此时数组还只是一个引用数组。必须再创建数组的每一个对象，并把对象赋值给数组引用。但String
 * 类型的数组也是特例。 具体请看下例演示。
 */
class MultiDimensionArray // 一个自定义的类，用于定义一维和多维数组来用。
{
    int a; // 定义成员变量a
    int b; // 定义成员变量b
```

```
    public MultiDimensionArray() // 无参构造函数
    {
        this.a = 0;
        this.b = 0;
    }

    public MultiDimensionArray(int i, int j) // 有参构造函数
    {
        this.a = i;
        this.b = j;
    }

    public void resetValue() // 用于将成员变量值复位为0的成员函数
    {
        this.a = 0;
        this.b = 0;
    }
}

public class ArrayDefinitionTest {
    public static void main(String[] args) {

        String str1[] = new String[2];  //String类型的一维数组
        str1[1] = "a String"; // 允许，无语法错误

        String str2[][] = new String[2][3];  //String类型的二维数组
        str2[1][2] = "元素str[1][2]可以直接赋值呢，发现了吗";  //仍然可以

        MultiDimensionArray x = new MultiDimensionArray(2, 3); // 定义一个普通的对象
        x.a = 88; // 允许
        x.b = 99; // 允许

        MultiDimensionArray objArray[] = new MultiDimensionArray[2]; // 定义一个对
象数组!
        // objArray[0].a=888; //有错！运行时会导致NullPointerException，因为
                           //objArray[0]还未通过new生成!
        // objArray[0].resetValue(); //与上一行有同样的问题!

        for (int i = 0; i < objArray.length; i++)
            // 将对象数组每个元素初始化生成，并通过数组引用。
            objArray[i] = new MultiDimensionArray();

        objArray[1].a = 999;
        System.out.println("objArray[1].a=" + objArray[1].a);
    }
```

}

2、 字符串类 String 及其常用方法（函数）：

（1）比较字符串（compareTo()方法和 equals()方法）

public int compareTo(String anotherString)　//compareTo 方法的原型

例如：String s1=“a student”;　　String s2=“A student”;

int n=s2.compareTo(s1);　//字母 A 和 a 的 unicode 码值分别为 65 和 97，所以 A 比 a 小 32，执行结果应为 A 和 a 的差值，即 n 的值为－32 。

public boolean equals(Object anObject)　//equals 方法的原型

例如：String s1=“a student”;　　String s2=“A student”;

boolean n=s1.equals(s2);　System.out.println(n);　　运行结果输出为”false”

（2）对字符串前后缀的判断（startsWith() 方法和 endsWith()方法）

boolean a2=s1.startsWith(“you”,4);　//判断 you 是不是 s1 的起始子字符串，从 4 号位置开始判断。

boolean a3=s1.endsWith(“student”);　//判断 student 是不是 s1 的终止子字符串，从尾部开始判断。

（3）查找指定字符串（indexOf()方法）

例如：String s1=“are you a student?”;

int n2=s1.indexOf(‘e’); //从头开始寻找字符 e

int n3=s1.indexOf(‘e’，5);//从 5 号位置开始寻找字符 e，两语句都是指出首先发现的字符 e 的位置。

（4）提取字符串(charAt()方法和 subString()方法)

String s1＝”Are you a student?”;

String s2=s1.substring(3,8); //提取 s1 中的字符，从 3 号字符开始到 8 号字符结束。

Strint s3=s1.substring(0); //提取 s1 中的字符，从 0 号开始直到最后。

（5）字符串转换（toLowerCase()方法和 toUpperCase()方法）

将字符串转换为小写或大写，格式为：字符串.toLowerCase()或字符串.toUpperCase()

诸如此类对一个字符串进行各种操作的方法（函数）还有很多，我们只需记住几种常用的，其它的方法不必死记硬背，只要打开 JDK API 帮助文档，查找到 String 类，其所有方法的原型及用法就都列出来了。除了功能不一样，它们的使用方法都是一样的。

```
/*
 * 例子2.4，关于关于字符串操作的一些知识。请注意其中使用的一些对字符串进行各种操作的
 * 方法（函数），通过API帮助文档了解这些方法的功能。
 */
public class str_mthd {
    public static void main(String args[]) {
        String s1 = "Are you a student?";
        String s2 = "Are you A student?";
        String s3 = "   Are you a seacher?   ";
```

```
        int n1 = s1.length();
        System.out.println(n1);
        boolean a1 = s1.equals(s2); // equals方法
        System.out.println(a1);
        String ss1 = s1.substring(3, 8); // substring方法
        System.out.println(ss1);
        String ss2 = s1.substring(0);
        System.out.println(ss2);
        int n2 = s2.indexOf('e'); // indexOf方法
        System.out.println(n2);
        int n3 = s1.indexOf('e', 5);
        System.out.println(n3);
        int n4 = s2.compareTo(s1); // compareTo方法
        System.out.println(n4);
        String ss3 = s3.replace('s', 't'); // replace方法
        System.out.println(ss3);
        String ss4 = ss3.trim(); // trim方法
        System.out.println(ss4);
        boolean a2 = s1.startsWith("you", 4); // startsWith方法
        System.out.println(a2);
        boolean a3 = s1.endsWith("student");
        System.out.println(a3);
    }
}
```

实验三　图形界面程序的界面设计

一、实验目的

1、掌握 Java 图形界面程序设计的基本思想和步骤。

2、掌握 JDK 中 AWT 包和 Swing 包的基本使用方法和区别。

3、掌握容器和布局的概念和使用。

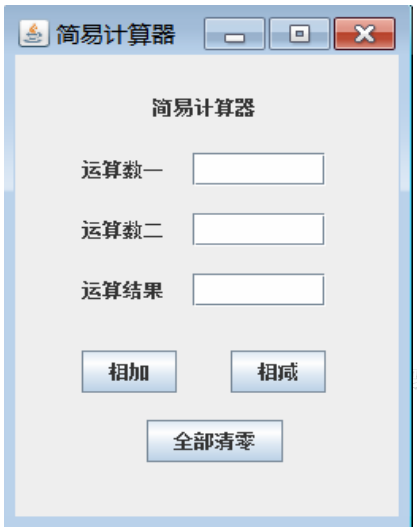
4、掌握图形界面程序的界面设计方法和步骤。

二、实验类型

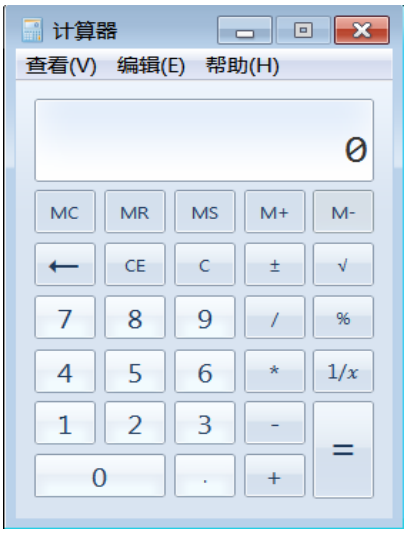
设计型

三、实验内容

1、自己编写所有代码，设计一个程序的界面。该程序的功能要求是：运行之后，界面要求基本如左图所示。其中界面设计时的顶层容器要求使用 JFrame（或 Frame），界面上的“简易计算器”、“运算数一”、“运算数二”、“运算结果”这些提示标签使用组件 JLabel（或 Label），输入运算数和输出运算结果使用 JTextField（或 TextField），“相加”、“相减”和“全部清零”使用按钮 JButton（或 Button）。 请注意查阅 JDK 的 API 文档中这些相关的类，看看它们具有哪些构造函数可以用来生成所需的对象，它们具有哪些方法可以用来进行调用。注意保存好本次实验代码，下一次实验将会要求实现其事件处理以继续完善。



2、下面的左图和右图分别是 windows 系统中的标准型和科学型计算器的界面截图。打开你使用的微软公司 Windows 操作系统或 Linux 等其它操作系统中自带的“计算器”软件，观察和参照其界面和功能，实现一个接近其界面和功能的计算器（标准型或科学型）的界面。并保存好代码，在下一次实验中实现其事件处理（即运算功能）。



四、预备知识

本次实验和下次实验主要是了解图形界面（GUI）程序设计。软件以图形界面形式运行时，用户可借助于菜单、按钮、标签等组件和通过鼠标、键盘等的操作共同完成对软件的应用。当程序运行后，出现图形窗口，我们称之为程序的界面。界面上可能有输入框、输入区、菜单项或者按钮等各种图形组件。当用户通过键盘往这些输入区域输入内容后回车或用鼠标点击了其中一些可以被点击的组件以后，能产生预期的、相应的合理响应和结果，称为事件处理。

目前 NetBeans 这样的开发软件可以直接通过拖放组件的方式设计界面，界面生成之后，完善好图形组件的事件处理的代码，就可完成图形界面程序设计了。Eclipse 在安装了可视化开发插件之后也支持拖放式放置组件来设计界面。但在初学阶段，我们最好先掌握通过自己编写代码来设计图形界面程序的方法，这样更有利于掌握图形界面程序运行的原理，有利于掌握面向对象程序设计的思想。

从大的角度而言，Java 图形界面程序设计就可以分为“界面设计”和“事件处理”两部分。这两部分可以分开进行，一般可以先设计界面，然后进行事件处理。我们将在本次实验中学习界面设计，在下一次实验中学习事件处理。

界面是需要有载体的。而“容器”是所有组件或容器的载体，是图形用户界面设计的基础。图形用户界面上所有的元素都要装载在容器中。容器其实就是 Java 图形界面开发包中的系统类，它们只是一类比较特殊的图形组件。它们在数量上基本上就只有几个。设计界面时，先要定义“容器”对象作为载体，然后定义按钮、输入框之类的普通图形界面组件对象，把这些组件对象添加（调用容器对象的 add 方法）到容器上，才能构成界面。而这些组件放到容器上去之后，按什么样的格局、什么样的顺序摆放，属于容器的“布局”问题。“布局”是对容器而言的，是对放置到容器内的组件的一种位置的约束。给任何一种容器对象设置布局的方法都是调用该容器对象的 setLayout 方法，该方法需要带相应的布局类对象作为参数。然后普通组件就会被按照这种布局参数放到容器上显示了。布局常用的有五六种。

使用图形组件类和容器类进行 GUI 设计时，需要引入相应的包或者类。在 Java 中，能够实现图形用户界面的类库有两个：java.awt 和 javax.swing。前者称为抽象窗口工具库 AWT（Abstract Windows Toolkit），后者是 Java 基础类库 JFC（Java Foundation Classes）的一个组成部分，它提供了一套功能更强、数量更多、更美观的图形用户界面组件。Swing 组件名称和 AWT 组件名称基本相同，但以 J 开头，例如 AWT 按钮类的名称是 Button，在 Swing 中的名称则是 JButton。在 java 中，AWT 包中的类是用来处理图形的最基本的方式，它是 Sun 公司早期所推出和使用的图形组件包，其中的组件包含基本 GUI 组件类、容器类、布局管理类，事件处理类和基本图形类这五种。AWT 包中的组件被认为是重量级组件，AWT 在实际运行中是调用所在平台的图形系统，它们依赖于本地系统来支持绘图与显示，其运行速度慢效果差，系统相关性较强。在 1998 年 Sun Microsystem 推出 JDK1.2 版本时，新的 javax.Swing 包被增加到 java 的基础类库中。Swing 包可以认为是 AWT 包的升级。它不仅拥有几倍于 AWT 包的用户界面组件，而且同样的组件，Swing 包中的组件往往可设置属性更多，功能更强大丰富。例如 Swing 包中按钮(Button)的功能较 AWT 包中的按钮功能更加强大，包括给按钮添加图像、使用快捷键以及设置按钮的对齐方式，还可以给按钮加入图片做背景。

Swing 采用 MVC(模型-视图-控制)的设计范式，使程序员可以根据不同的操作系统来选择不同的外观。Swing 组件完全是用 Java 代码实现的，可以跨平台，使用 Swing 组件的程序在 Mac、Windows 或 Unix 平台上的观感都一样。Swing 组件通常被称为轻量级组件。程序设计过程中最好不要混用 Swing 组件和 AWT 组件，虽然有时候也可以同时用，但有可能在视觉效果和响应上造成不稳定隐患。

AWT 包中常用的顶层容器为 Frame、Dialog、Applet，中间层容器为 Panel（中间层容器是一种可以放到顶层容器上的组件，它可以把顶层容器进行进一步的区域划分以设计所需格局的界面）。常用的普通组件有 Button、TextField、TextArea、Label、List 等。

Swing 包中的组件分类主要为：

（1）顶层容器：JFrame、JApplet、JDialog。

（2）中间层容器：JPanel、JScrollPane、JSplitPane、JToolBar。

（3）其它基本组件：JButton、JList、JTextField 等。

关于布局，其实每种容器都有默认的布局，当设计者没有指定使用另外的布局时，放到它上面的组

件就会按默认布局方式摆放。在 AWT 和 Swing 中都可对容器设置布局，都是调用容器对象的 setLayout 方法。布局有几种，这里作简单说明，具体请大家从示例程序的运行以及 JDK API 文档中查阅这些布局类的说明文档来体会：

（1）FlowLayout 布局：顺序布局，有时又称流式布局。使用该布局时，容器上的元素会按容器显示时的大小，按照从左到右、从上倒下的一行一行摆放下去。

（2）BorderLayout 布局：边界布局。它可以把容器分为东西南北中 5 个方位，可以指定某个组件在哪个方位（但注意并非一定要有 5 个组件摆上去，不足 5 个也可指定位置摆放）。

（3）GridLayout 布局：网格布局。可以指定容器上的元素按几行几列的位置顺序摆放。

（4）CardLayout 布局：卡片布局。使用该布局的容器可放置多个组件，但同一时刻只能显示其中一个，就像一叠叠整齐的纸牌中每次只能最上面一张一样。可以指定哪个组件被显示。

（5）null 布局：空布局。空布局并非没有布局，而是容器被设置为这种布局以后，放到它上面的每个组件就可以通过 setBounds(int a,int b,int width,int height)来设置其具体的位置。

（6）另外还有 BoxLayout（盒式布局）、GridBagLayout（网格包布局）等不常用布局。

在具体的程序设计时，上述每一种布局都对应是一个类，用它们定义出来的对象作为容器对象的 setLayout 方法的参数，就可以把容器设置成该参数所对应的布局了。

仔细阅读、体会、运行和模仿学习后面的多个例子程序（注意代码注释中所提示的各种细节问题），我们可以最后再次总结出“界面设计”的步骤：（1）定义顶层容器对象。 （2）用容器对象的 setLayout 方法给容器设置一种布局。 （3）定义普通图形组件。 （4）调用容器对象的 add 方法将普通图形组件添加到容器上。 其中步骤（1）和（3）一般放在图形界面程序类的成员变量声明部分。（2）和（4）一般放在类的构造函数部分。 最后在 main 函数中生成图形界面程序类的实例运行，显示出界面来。

```
/*
 * 例子 3.1，关于容器 Frame 的使用。
 */                                   Java实验指导书（桂林电子科技大学汪华登）
import java.awt.Frame;
public class FrameApp {
    public static void main(String[] args) {
        Frame frame=new Frame();  //定义一个Frame容器对象frame
        frame.setSize(200,100);
        frame.setVisible(true); //注意Frame对象默认是不可见的，尝试注释掉此句，你会发现看
                                //不到运行结果的窗口出现。
    }
}


/*
 * 例子3.2，关于容器JFrame的使用。 同上例。
 */
import javax.swing.JFrame;

public class JButtonApp {
    public static void main(String[] args) {
        JFrame jframe = new JFrame(); // 定义一个JFrame容器对象jframe
        jframe.setSize(200, 100);
        jframe.setVisible(true);// 注意JFrame对象默认是不可见的，若注释掉此句，你会发现看
        // 不到运行结果的窗口出现。
```

```
        jframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);//JFrame表面上可以关闭，
        // 实际上只是被隐藏为不显示。要真正可以关闭它，必须让JFrame对象
    // 调用setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)这样的方法来实现关闭功能。
    }
}


/*
 * 例子3.3，本例演示使用容器JFrame，设计一个基本的图形界面程序的过程和方法。请注意注释
 * 所提示的一些应该注意的细节问题。
 */
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Color;
import javax.swing.JButton;
import javax.swing.JFrame;
public class JButtonApp{
    JFrame jf=new JFrame();   //定义设计图形界面程序所需的容器对象
    JButton button1=new JButton("按钮1"); //定义界面上需要的按钮button1，以下类似
    JButton button2=new JButton("按钮2");
    JButton button3=new JButton("按钮3");

    public JButtonApp(){    //一般总是用构造函数用于把普通组件添加到容器上构造出界面来！
        jf.setLayout(new FlowLayout());//当没有使用任何布局的时候，JFrame容器只会显示代码中最
                                       //后使用add方法添加上去的组件。假如这条设置布局的语句被
                                       //注释掉的话，程序运行后界面上将会只显示按钮3这一个按钮
        jf.getContentPane().add(button1);//往容器上面添加按钮button1。注意如果是往AWT中的Frame
                                         //容器对象上面添加任何组件的话，都是调用该对象的add方
                                         //法，以组件对象名作为参数添加即可。但对于Swing中的
                                       //JFrame，必须先通过getContentPane()方法获取其内容面
                                         //板，然后再调用add方法添加组件。
        jf.getContentPane().add(button2);
        jf.getContentPane().add(button3);
        jf.getContentPane().setBackground(Color.RED);       //把jf的背景色设置为红色。
        button1.setFont(new   Font("隶书",Font.PLAIN,20));  //给按钮button1设置字体。

        jf.setVisible(true);//JFrame容器对象默认是不可见的，必须要设置为可见，界面才会出来。
        jf.pack(); //如果尝试不调用pack函数，默认出来的界面不会显示全部组件。pack的作用是使
                   //界面上的所有组件以最经济的方式全部显示出来。

    }
    public static void main(String[] args) {
        JButtonApp jba=new JButtonApp();     //生成对象实例，在此过程中构造函数会被调用
```

```
                                        //运行，界面即出现
    }
}


/*
 * 例子3.4，本例演示容器Applet的使用。
 */
import java.applet.Applet;
import javax.swing.JTextField;
public class JTextFieldApp extends Applet //这是一个Applet程序，Applet实际上也是一种容器！
{
    JTextField jtxtfld1,jtxtfld2;  //定义两个JTextField类型的组件
    public void init()
    {
        jtxtfld1=new JTextField(20);   //生成对象实例
        jtxtfld2=new JTextField(20);
        add(jtxtfld1);   //将组件添加到容器Applet上
        add(jtxtfld2);
        jtxtfld2.setEditable(false); //将其中的jtxtfld2设置为不可编辑的，即不能往里输入东西
    }
}
```



```
/*
 * 例子3.5，本例演示FlowLayout布局(顺序布局)的使用。这种布局就是按照调用add方法添加组件的顺序
 * 从左到右、从上到下摆放组件，下面的程序中的5个按钮就是按照这样布局的。
 */
import java.awt.*;
import java.awt.event.*;
public class FlowLayoutApp extends Frame  //这个例子程序不是在程序中定义Frame容器对象作为
                                          //容器，而是继承自Frame类。这隐含着是以Frame作
                                          //为顶层容器。
{
    public FlowLayoutApp()     //构造函数
    {
        super("An FlowLayout Example");  //调用父类的构造函数设置顶层容器的标题
        setLayout(new FlowLayout());
        setBackground(Color.green);  //以绿色作为界面的背景色
        setVisible(true);            //容器设置为可见
        add(new Button("按钮1"));    //添加一个按钮到容器上面
        add(new Button("按钮2"));
        add(new Button("按钮3"));
        add(new Button("按钮4"));
        add(new Button("按钮5"));
        pack();   //使界面上的元素在程序一运行之后，就全部显示出来。
```

```
    }
    public static void main(String args[])
    {
        FlowLayoutApp fla=new FlowLayoutApp();  //生成对象实例
    }
}


/*
 * 例子3.6，本例演示BorderLayout布局(边界布局)的使用。这种布局会把容器上的组件按照
 * 东西南北中各个方位进行排列。可以通过参数具体指定某个组件在东西南北中的哪一个方向。
 * 通过这种布局，可以把容器细分为几个部分，以便设置更好的界面。请注意并不是一定要东
 * 西南北中五个方向都要放组件的。不足五个的组件同样可以分别被指定放在容器上东西南北
 * 中的任何一个方位。自己可以尝试放置少于五个组件的情况。
 */
import javax.swing.JButton;
import javax.swing.JFrame;
public class BorderLayoutApp
{
    JFrame jf;
    JButton eastButton;
    JButton southButton;
    JButton westButton;
    JButton northButton;
    JButton centerButton;

    public BorderLayoutApp()
    {
        jf=new JFrame("这是一个演示BorderLayout布局的例子程序");
        eastButton=new JButton("东按钮");
        southButton=new JButton("南按钮");
        westButton=new JButton("西按钮");
        northButton=new JButton("北按钮");
        centerButton=new JButton("中央按钮");

        jf.add(eastButton,"East");  //注意：Frame容器的默认布局就已经是BorderLayout
        jf.add(southButton,"West");
        jf.add(westButton,"North");
        jf.add(northButton,"South");
        jf.add(centerButton,"Center");
        jf.setVisible(true);
        jf.pack();
    }
    public static void main(String args[])
    {
        BorderLayoutApp bla=new BorderLayoutApp();
```

```
    }
}
```

```
/*
例子3.7，本例演示null布局(顺序布局)的使用。有时会用到null布局（即无布局管理器），程序员必须自己为容器中的组件设置大小和位置，因此其好处是可以随心所欲地设置组件的大小和位置得到自己想要的布局，但跨平台使用时，界面可能发生变化，且程序编写和调试相对比较繁琐，示例如下。
*/
import javax.swing.JButton;
import javax.swing.JFrame;
public class NullLayoutApp
{
    JFrame jfr=new JFrame("null布局的使用示例");
    JButton b1=new JButton("open");
    JButton b2=new JButton("close");
    JButton b3=new JButton("ok");

    public NullLayoutApp(){  //构造函数，用于生成图形程序的界面
        jfr.setSize(300,300);   //设置容器对象jfr的宽度和高度
        jfr.setLayout(null);   //设置为null布局。注意null四个字母全部小写!
        b1.setSize(80,20);       //设置按钮b1的宽度和高度
        b1.setLocation(10,30);  //设置按钮b1在界面上的左上角的位置坐标
        b2.setSize(90,40);
        b2.setLocation(40,60);
        b3.setSize(100,50);
        b3.setLocation(100,80);
        jfr.add(b1);    //依次将每个组件添加到容器上
        jfr.add(b2);
        jfr.add(b3);
        jfr.setResizable(true);
        jfr.setVisible(true);
        //jfr.pack();   注意：使用null布局之后，pack方法反而不需要使用了!
    }
    public static void main(String args[])
    {
        NullLayoutApp nla=new NullLayoutApp(); //定义对象实例，调用构造函数生成界面
    }
}
```

实验四　图形界面程序的事件处理

一、实验目的

1、掌握 Java 图形界面程序设计的基本思想和步骤。

2、掌握图形界面程序设计中键盘鼠标事件处理的机制。

3、了解常用的监听器接口及其方法和作用。

4、掌握图形界面程序设计中事件处理的方法和步骤。

二、实验类型

设计型

三、实验内容

1、在上一个实验（实验三）的第 1 题中，我们设计了一个简易计算器的界面。但仅仅要求设计出其界面，而没有要求进行事件处理。在本次实验中，请实现其事件处理。使得程序运行之后，当输入两个小数作为运算数后，点击“相加”或者“相减”按钮，两数相加或相减的结果就会显示在“运算结果”框中。 如果点击“全部清零”按钮，那么两个运算数输入框和运算结果框中就会完全被清空。

2、对于第 1 题，除了正常数据之外，用户还可能因为好奇或误操作导致这样几种情况： （1）1 个或 2 个运算数均未输入；（2）输入的数据串中含有除小数点和数字之外的非法字符；（3）输入的数据串中不含有除小数点和数字之外的非法字符，但小数点的个数超过 1 个或小数点的位置在数据的开头或结尾处。 请改进你的程序，对这几种情况进行判断，**不允许使用正则表达式或异常处理机制**，完全使用字符串类的各种方法，对这几种输入情况得到的用户所输入的字符串进行分析判断，并在你的程序图形界面上向用户进行相应的提示。

3、在上一个实验（实验三）的第 2 题中，我们设计了一个仿 Windows 系统自带的计算器的界面。但仅仅要求设计出其界面，而没有要求进行事件处理。在本次实验中，请实现其事件处理。使得程序运行之后，各项主要运算功能能良好运行。请自己输入数据进行测试。并尽量实现对非法数据的判断和处理。

四、预备知识

图形界面程序是事件驱动的，当用户与程序之间交互时产生事件，触发程序中的事件处理代码。事件一般分为用户事件和系统事件。通常关注比较多的是用户事件。典型的交互和要处理的用户事件包括用户移动鼠标、单击鼠标，点击按钮、点击下拉列表框或选项卡、在文本框中输入内容后回车、点击菜单项或关闭图形窗口等。

事件处理，处理的是谁的事件？实际上处理的就是界面上的组件的事件。比如被点击的按钮，被输入内容后按下了回车键的单行文本框等等。事件的处理并非自动完成。事件发生后，应该产生什么结果，其实完全是由程序设计者所设计的代码来决定——由每一行具体的程序代码来指定在事件发生后，界面上的各个组件产生什么显示结果，或者产生什么改变。

Java 事件处理的机制是产生一个事件的对象必须设定其事件处理的监听者对象（Listener）。当事件发生时，监听者便会依据事件的类型来执行相应的程序。

要处理一个对象所产生的事件，首先必须注册该对象的监听者。Java.awt.event 包中按照不同的事件类型定义了 11 个监听器接口，每类事件都有对应的事件监听器，监听器实际上就是接口，接口中定义了事件发生时可调用的、必须要实现的方法，一个类可实现监听器的一个或多个接口。与 AWT 事件有关的所有事件都是由 java.awt.AWTEvent 类派生而来的。

系统对事件进行了分类。每一种事件用一种系统定义好的系统类来表示。不同的组件会对应产生不同的事件，也就对应到不同的事件类。而每一种事件类，又对应要实现(implements)不同的接口。而每一个接口中，又有一个或者多个方法必须要被实现（这也是接口被 implements 时的语法要求），这些方法中的代码，就是对应的事件的具体的处理代码。

常见的 AWT 事件以及与其关联的组件如下表所示,Swing 也是类似的。

另外，由于监听器实际上就是实现了相应接口的类，而接口一般要求实现许多方法，如接口 WindowListener 中有 7 个方法，并不是所有方法都是需要的，但接口却要求实现其所有的方法。为了简化，java 语言为一些接口统一提供了事件适配器（Adapter)。事件适配器是抽象类。通过继承事件适配器，重写需要的方法，不要的方法可以不写。例如 Java.awt.event 包中提供了以下几个事件适配器：ComponentAdapter——组件适配器，ContainerAdapter——容器适配器，FocusAdapter——焦点适配器，KeyAdapter——键盘适配器，MouseAdapter——鼠标适配器，MouseMotionAdapter——鼠标移动适配器，WindowAdapter——窗口适配器。

表 4-1 常见的 AWT 事件及其关联组件、接口、方法

事件类	可关联的组件	要实现的接口	接口中必须实现的方法
ActionEvent	Button,List, MenuItem,TextField	ActionListener	ActionPerformed(Actionevent e)
ItemEvent	Checkbox,Choice, List,MenuItem	ItemListener	ItemStateChanged(ItemEvent e
ContainerEvent	Container,Dialog, Frame,Panel, ScrollPane,Window	ContainerListener	componentAdded(ContainerEvent e) componentRemoved(…)
MouseMotionEvent	Button,Canvas,Checkbox, choice,Component,Contai ner,Dialog, Frame,Label,List, Panel,Scrollbar, ScrollPane,TextArea, TextField,Window	MouseMotionListener	mouseGragged(MouseEvent e) mouseMoved(MouseEvent e)
MouseEvent	同 MouseMontionEvent	MouseListener	mousePressed(MouseEvent e) mouseReleased(…) mouseEntered(…) mouseExited(…) mouseClicked(…)
KeyEvent	同 MouseMontionEvent	KeyEvent	keyPressed(KeyEvent e) keyReleased(…) keyTyped()
FocusEvent	同 MouseMontionEvent	FocusListener	focusGained(FocusEvent e) focusLost(FocusEvent e)
AdjustmentEvent	Scrollbar	AdjustmentListener	adjustmentValueChanged(Adjustment Event e)
ComponentEvent	同 MouseMontionEvent	ComponentListener	ComponentMoved(ComponentEvent e) ComponentHidden(…) ComponentResized(…) ComponentShown(…)
TextEvent	TextField,TextArea	TextListener	textValueChanged(TextEvent e)
WindowEvent	Dialog,Frame,Window	WindowListener	WindowClosing(WindowEvent e) WindowOpened(…)

			WindowIconified(…) WindowClosed(…) WindowDeiconfied(…) WindowActivated(…) WindowDeactivated(…)

总结一下，图形界面程序设计中的事件处理的步骤是：首先要让可能产生事件的组件对事件进行监听，那么就要实现监听器接口，实现（implements）了接口的话，就必须要实现接口中所有的方法（这些方法就是具体进行事件处理的代码）。每个组件的事件处理，在程序代码设计上都是要处理这三点。下面我们以例子程序来进行演示和体会。

通过键盘接收到的用户输入或者从图形界面程序的 TextField（或 JTextField）等组件中通过 getText() 方法获取到的内容，一般多为字符串类型的数据。在需要将这些数据用来进行运算等操作时，需要**将字符串和数值两种数据类型进行转换**，下面各列举两种方法（具体请查阅 API 文档中对应的 String 类和 Float、Integer 等数据类型类）：

（1）通过 parseInt 等方法将字符串转换为数值，例如：

```
String s = “12.35”;          //定义一个字符串 s，其内容为“12.35”
float y1=Float.parseFloat(s);       //将 s 转换为 Float 类型变量后转赋给 y1 ，后面的语句类似。
double y2=Double.parseDouble(s);
int y3=Integer.parseInt(s);
long y4=Long.parse.Long(s);
byte y5=Byte.parseByte(s);
```


（2）通过 valueOf 方法将字符串转换成数值，例如：

```
String str=“23.54”;
float y1＝Folat.valueOf(str).floatValue();      //其它数据类型类似。
```

（3）通过 toString 方法将数值转换成字符串，例如：

```
int   i=1234;                    //定义一个整数 x1。
String s1=Integer.toString(i);      //将整数 x1 转换为字符串，存到 s1 中。后面语句功能类似。
float   f=12.34f;
String s2=Float.toString(f);
double   d=123.4;
String s3=Double.toString(d);
```

（4）通过 String 类的 valueO 方法将数值转换为字符串，例如假设 x1 和 x2 是数值：

```
String s1=String.valueOf(x1);       //将数值 x1 转换为字符串 s1。
String s2=String.valueOf(x2);
```

```
/*
例子4.1，按钮事件的处理。这个例子，代表了java图形界面程序设计中所有的事件处理的思想和方法、步骤。
也就是说除了按钮以外的组件，它们的事件处理，可能要实现的接口、接口中的方法、注册监听器的方法等不
一样之外，其基本方法和过程，跟这个例子是完全一样的。请仔细体会理解本例。
事件发生者：myButton;     事件监听者：实现了ActionListener接口的MyEvent类
发生者对事件的监听形式：调用addActionListener()方法;  事件处理者：actionPerformed方法
*/
```

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
public class ButtonEventApp {
    JFrame jf=new JFrame("一个按钮事件处理的例子");
    JButton myButton=new JButton("测试按钮");
    public ButtonEventApp(){
        myButton.addActionListener(new MyEvent()); //对按钮myButton，调用其监听方法
              //addActionListener,对其可能发生的点击事件（MyEvent类型对象表示）进行监听
              //有很多书上称这一步为注册监听器接口
        jf.add(myButton);
        jf.setVisible(true);   //把容器jf设置为可见
        jf.pack();   //让容器jf上的组件在程序运行之后就直接全部显示出来
    }

    public static void main(String args[]){
        new ButtonEventApp();    //产生一个匿名对象，让程序运行起来。
    }
}

class MyEvent implements ActionListener{ //ActionListener接口为“事件监听器”，MyEvent类
          //实现了它，用来接收“产生事件的对象”传递来的信息，以实现ActionListener接口的
          //actionPerformed(ActionEvent e)方法，由参数e接收事件信息，然后对传入的事件信息
          //进行处理

    public void actionPerformed(ActionEvent e){     //如同这里一样，每一个图形界面程序中
                                          //具体的事件处理的代码，都是由接口中的方法来体现
    System.out.println("在控制台输出信息，进行按钮事件处理的测试...");  //本例中事件处理的结果
                                                //是当点击了按钮以后,让控制台输出一行提示信息
    }
}


/*
例子4.2，按钮事件的处理。   这个例子程序是对例子4.1的一个改写，它们本质上是一样的。
【它与例子4.1功能完全一样，经常各类书籍资料上也采用这种代码设计形式，请体会一下跟例4.1有何异同】
这个例子，其实一样代表了java图形界面程序设计中所有的事件处理的思想和方法、步骤。

请仔细体会理解本例。
事件发生者：myButton;    事件监听者：实现了ActionListener接口的ButtonEventApp类
发生者对事件的监听形式：调用addActionListener()方法;  事件处理者：actionPerformed方法
*/
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
```

```
import javax.swing.JFrame;
public class ButtonEventApp implements ActionListener{//在定义类的同时，就implements监听器接口
    JFrame jf=new JFrame("一个按钮事件处理的例子");
    JButton myButton=new JButton("测试按钮");
    public ButtonEventApp(){   //构造函数
        myButton.addActionListener(this); //对按钮myButton，调用其监听方法
                        //addActionListener,对其可能发生的点击事件进行监听
                                    //有很多书上称这一步为注册监听器接口
        jf.add(myButton);
        jf.setVisible(true);   //把容器jf设置为可见
        jf.pack();   //让容器jf上的组件在程序运行之后就直接全部显示出来
    }

    public void actionPerformed(ActionEvent e){     //如同这里一样，每一个图形界面程序中
        //具体的事件处理的代码，都是由接口中的方法来体现
        System.out.println("在控制台输出信息,进行按钮事件处理的测试...");//本例中事件处理的结果
                //是当点击了按钮以后，让控制台输出一行提示信息
        }

    public static void main(String args[]){
        new ButtonEventApp();    //产生一个匿名对象，让程序运行起来。
    }
}
```



```
/*
 * 例子4.3。这是对例子进行了一点改进后的例子，它通过事件监听处理类传递参数
 * 来对界面上在另一个类中定义的对象进行操作
 */
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JTextField;

public class ButtonEventApp {
    JFrame jf = new JFrame("一个按钮事件处理的例子");
    JButton myButton = new JButton("请点击我");
    JTextField jtf = new JTextField(10);

    public ButtonEventApp() {
        myButton.addActionListener(new MyEvent(jtf));//对按钮myButton，调用其监听方
法
            //addActionListener,对其可能发生的点击事件（MyEvent类型对象表示）进行监听
            //有很多书上称这一步为注册监听器接口 ，  jtf会被传递到MyEvent中进行控制处理
```

```
        jf.setLayout(new FlowLayout());
        jf.add(myButton);
        jf.add(jtf);
        jf.setVisible(true); // 把容器jf设置为可见
        jf.pack(); // 让容器jf上的组件在程序运行之后就直接全部显示出来
        jf.setLocation(300, 200);   //让界面出来之后窗口的左上角位于坐标(300,200)处
    }

    public static void main(String args[]) {
        new ButtonEventApp(); // 产生一个匿名对象，让程序运行起来。
    }
}

class MyEvent implements ActionListener { // ActionListener接口为“事件监听器”，MyEvent
类
    // 实现了它，用来接收“产生事件的对象”传递来的信息，以实现ActionListener接口的
    // actionPerformed(ActionEvent e)方法，由参数e接收事件信息，然后对传入的事件信息
    // 进行处理
    JTextField jt = null;

    public MyEvent(JTextField jtf) {
        jt = jtf;
    }
```



```
    public void actionPerformed(ActionEvent e) { // 如同这里一样，每一个图形界面程序中
        //具体的事件处理的代码，都是由接口中的方法来体现本例中事件处理的结果是当点击了
        //按钮以后，让控制台输出一行提示信息
        jt.setText("你点击了按钮！");
    }
}



/*
例子4.4，文本框(JTextField)对象的使用及其事件处理。
*/
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;
import javax.swing.JTextField;

public class JTextFieldApp2 implements ActionListener //实现监听器接口ActionListener
{
    JFrame jf;   //定义顶层容器
    JTextField txtfld1,txtfld2;  //定义两个JTextField组件
    public JTextFieldApp2()    //构造函数，用于生成界面
```

```
    {
        jf=new JFrame();
        jf.setLayout(new FlowLayout());  //设置容器jf的布局
        txtfld1=new JTextField(20);
        txtfld2=new JTextField(20);
        jf.add(txtfld1);   //添加组件到容器上
        jf.add(txtfld2);
        txtfld2.setEditable(false); //将第二个JTextField对象设置为不可编辑，它会变灰
        txtfld1.addActionListener(this);   //事件监听
        jf.setVisible(true);
        jf.pack();
    }

    public void actionPerformed(ActionEvent evt) //实现接口中的方法，进行具体的事件处理
    {
        if((evt.getSource()==txtfld1)&&(txtfld1.getText().equals("black"))) //当用户是
        {                                          //往txtfld1中输入了black并且按了回车时
            txtfld1.setText("");      //就把第一个输入框清空
            txtfld2.setText("Tel of black is: 123");//让第二个输入框显示black的电话号码
        }
        else if((evt.getSource()==txtfld1)&&(txtfld1.getText().equals("smith")))//当
        {                                      //用户是往txtfld1中输入了smith并且按了回车时
            txtfld1.setText("");    //就把第一个输入框清空
            txtfld2.setText("Tel of smith is: 456");//让第二个输入框显示black的电话号码
        }
        else if((evt.getSource()==txtfld1)&&(txtfld1.getText().equals("jack")))
        {
            txtfld1.setText("");
            txtfld2.setText("Tel of jack is: 789");
        }
        else      //当输入的不是其中任何一个人时
        {
            txtfld1.setText("");
            txtfld2.setText("can't find his(her) number");
        }
    }
    public static void main(String args[]){
        JTextFieldApp2 tfa=new JTextFieldApp2();
    }
}
```


/*
例子4.5，选择框（Checkbox对象）的使用及其鼠标点击事件的处理机制。注意此例中显示，Checkbox产生的事件对应的事件类不再同于Button或者TextField了，它对应的事件监听方法（函数）和监听器接口以及接口中要实现的事件处理方法（函数）也都不同于Button或者TextField了。 同样的道理，其它还有很多图形组件，

```
虽然事件处理思想和方法也是完全一样的，但对应的事件监听方法（函数）和监听器接口以及接口中要实现的
事件处理方法（函数）也可能完全不同于Checkbox组件了。 请仔细体会。
*/
import java.awt.Checkbox;
import java.awt.CheckboxGroup;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Frame;
import java.awt.Panel;
import java.awt.TextField;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

public class CheckBoxApp implements ItemListener //Checkbox对应的监听器接口是ItemListener
{
    Frame f;
    Checkbox ckbx1,ckbx2;
    CheckboxGroup ckbx;//把多个Checkbox组件放到一个Checkbox组ckbx里面，使得只能点击组内其中一个
    Panel pnl;  //定义一个面板(Panel)对象做中间层容器
    TextField txt;
    public CheckBoxApp()
    {
        f=new Frame();
        f.setLayout(new FlowLayout());  //改变Frame的默认布局BorderLayout为FlowLayout布局
        txt=new TextField(20);
        pnl=new Panel();
        ckbx=new CheckboxGroup();
        ckbx1=new Checkbox("ckbx1",false,ckbx); //把Checkbox组件放到一个组内
        ckbx2=new Checkbox("ckbx2",false,ckbx);
        pnl.setBackground(Color.yellow);
        pnl.add(ckbx1);  pnl.add(ckbx2);
        f.add(pnl);
        f.add(txt);
        ckbx1.addItemListener(this);  //对ckbx1进行事件监听(监听对它的鼠标点击)
        ckbx2.addItemListener(this);  //对ckbx2也进行事件监听(监听鼠标点击)
        f.setVisible(true);
        f.pack();
    }

    public void itemStateChanged(ItemEvent e) //接口里面的方法相应的应为itemStateChanged
    {
        if(e.getItemSelectable()==ckbx1)  //如果点击的是ckbx1
        {
            txt.setText("ckbx1 is selected");//就让TextField组件txt显示为"ckbx1 is selected"
        }
        else if(e.getItemSelectable()==ckbx2)  //如果点击的是ckbx2
```

```
        {
            txt.setText("ckbx2被选中了");
        }
    }
    public static void main(String args[])
    {
        new CheckBoxApp();    //生成匿名对象实例，让程序运行起来
    }
}
```

实验五　异常

一、实验目的

1、掌握异常的概念、作用、分类和进行异常处理的语法结构。

2、了解系统异常类的体系和系统定义的常见标准异常。

3、掌握用户自定义异常类的定义和抛出异常的方法。

4、通过掌握异常来分析调试程序的编译错误和编写更健壮的程序。

二、实验类型

设计型

三、实验内容

1、标准异常：设计一个程序，使得其运行时，可以接受用户输入若干个整数并依次存入数组中，不限定输入数的个数，使得在输入数的个数超过数组的实际大小时产生数组下标越界异常（ArrayIndexOutOfBoundsException）。程序代码中“**不需要**”对异常进行捕获处理，观察异常产生时程序运行是否被中断，编译器产生了怎样的提示。

2、自定义异常：设计一个新的程序或者完善题目 1 的程序，使得其运行时，可以接受用户输入若干个数并依次存入数组中。要求用户输入的整数必须大于 0 小于 100。如果不符合这个范围则抛出异常。请自定义一个异常类来表示这种情况，并在用户的输入不符合要求时抛出该异常。

四、预备知识

1、异常的定义、作用

异常是在程序运行过程中发生的、会打断程序正确执行的事件。比如数组元素下标越界等。在编写程序时，必须考虑到可能发生的异常事件并做出相应的处理。在一般的高级语言中，要由编程者自己测试出错误发生的原因，通过使用 if 语句或 switch 语句来判断是否出现异常事件，并进行相应的处理。 对问题很多的情况，就有大量的判断语句和正常的执行语句混杂在一起，使得代码结构很不清晰，浪费设计者的精力。Java 语言用面向对象的方法处理异常，每一个异常事件分别由一个异常类的对象来代表。这种办法使得程序员不必再花费很大的精力去编写异常程序。

2、异常的体系、表示及分类

Java 中任何异常对象都是系统中的 java.lang.Throwable 类或其子类的对象。Throwable 类是 java 异常类体系中的根类，它有两个子类：一个是 Error 类，另一个是 Exception 类。Error 类及其子类的对象，代表了程序运行时 java 系统内部的错误，这种错误通常是无法控制的，如磁盘不能读写等等。一旦发生这种错误，一般只能停止程序的运行。而 Exception 类及其子类的对象是程序员应该认真关心并尽可能加以处理的“异常”。Exception 类有许多子类，其中最常见的如 IOException 类，该类及其子类对象表示一个 I/O 错误，程序员应该在程序中进行处理。

对于一些常见的典型异常事件，系统中已经定义好了各种异常来表示它们。大家可以打开 JDK 的 API 帮助文档，找到 Exception 类，会发现它下面有大量的子类，子类下面又可能有大量子类，它们都是系统定义好用来表示各种异常事件的异常类。

常见的系统定义好的标准 java 异常有很多，这里随便列举几种如下：

空指针：java.lang.NullPointerException

整数被 0 除：java.lang.ArithmeticException

数组下标越界：java.lang.ArrayIndexOutOfBoundsException

输入输出流异常：java.lang.IOException

字符转换为数值时的格式异常：java.lang.NumberFormatException
无法找到文件异常：java.lang.FileNotFoundException

在进行程序设计时，我们会发现某些可能会产生的异常（比如数组下标越界），即使我们不编写对其捕获处理的代码，编译的时候也会被编译器通过，只是在运行时就可能因为有异常产生（实际上是异常被系统抛出来）而终止运行，这类异常被称为非受查异常。但有些可能会产生异常的代码，例如 BufferedReader.readLine()方法或者 System.in.read()方法有可能会导致系统抛出的 IOException，在编译时系统就提示我们必须要进行处理，要求我们要么使用 try-catch-finally 的捕获机制进行处理，要么用 throws 声明异常可能会产生，交由调用者（比如 JVM 自身）来进行处理，这类异常通常被称为受查异常。

3、异常处理的语法

Java 中的异常处理涉及到五个关键字：try、catch、finally、throw 和 throws。 Java 语言中用 try-catch-finally 这一套语句来捕获并处理异常，语法格式如下：

```
try
{
  //此处是可能会产生异常的程序代码
}
catch(Exception_1 e1)   { //处理异常 Exception_1 的代码}
catch(Exception_2 e2)   { //处理异常 Exception_2 的代码}
……….
catch(Exception_n en)   { //处理异常 Exception_n 的代码}
```

[finally { /*此处为必定会执行的其它代码。finally 子句不是必须的。但一旦有，则不论 try 中的代码产生异常与否，finally 部分的语句一定会被执行。所以此部分通常放置最后一定需要做的，比如释放资源、标志变量复位等工作对应的程序代码。*/}]

try-catch-finally 语句的 try 块、catch 块、finally 块中的代码都可以嵌套另外的 try-catch-finally 语句，并且可以嵌套任意多层次。但不建议嵌套多次。

异常处理中，还有两个关键字 throw 和 throws 要理解和区分。首先我们说说 throws 这个关键字。它的作用是放在方法（函数）的声明后面，用于表示该方法（函数）可能会产生它所指明的异常，但这个函数体内部并不对其进行捕获处理，谁调用了这个函数，这个调用者就要进行捕获处理（也就是说调用这个函数的代码是要放在 try 中，后面进行 catch 处理）。

throws 在使用时的语法格式为：

```
throws <此括号内为异常列表，当有多个时用逗号分隔>
如 public int handle() throws IOException, NullPointerException
   {
      //可能会产生异常的代码
    }
```

表示函数 handle 可能会抛出 IOException 和（或）NullPointerException，谁调用 handle，调用代码就要进行异常的捕获处理。

若一个方法声明抛出异常，则表示该方法可能会抛出所声明的那些异常，从而要求该方法的调用者，在程序中对这些异常加以注意和处理。若一个方法没有声明抛出某个异常，则该方法仍可能会抛出异常，只不过那些异常不要求调用者加以注意。即使一个方法声明抛出异常，则该方法仍可能抛出不在逗号分隔的异常列表之中列出的异常。通常用逗号分隔的异常列表中的异常，都是要求调用者一定要在程序中明确加以处理的，或者用 try-catch-finally 捕获处理，或者再次声明抛出异常，或者用 try-catch-finally 捕获处理侯再次抛出异常，从而形成异常处理链。

声明抛出异常（throws）是一个说明性的子句，只是表明一个方法可能会抛出异常，而真正抛出异常的动作都是由抛出异常语句来完成的，要用到 throw 关键字。其格式如下：

throw <异常对象>

其中<异常对象>必须是 Throwable 类或其子类的对象。如：throw new Exception(“这是一个异常”);而：throw new String(“抛出异常”);则在编译时会报错。因为 String 类不是 Throwable 类的子类。

系统中所有的异常（系统定义的标准异常以及用户自己定义的异常）都是通过 throw 这个关键字来抛出的。

4、自定义异常

前面我们说到，对于常见的出问题的情况，比如数组下标越界等，系统都定义好了各种异常类来表示，当这些异常事件发生的时候，系统就抛出对应的异常来。但是对于某些特定的情况（特别是跟设计者设计的程序的特定情况相关的情形），例如，设计者要求用户输入的数据在 30 到 50 之间这种情况，这就不具有普遍性，类似这样的情形就有无数种了，系统不可能全部都考虑到了，都定义好了异常来表示。这时候，就需要用户自己定义异常类来表示这种情况，当这种情况发生的时候，就抛出自己定义的异常类的对象来表示异常情况的发生。系统用户自定义异常类必须是 Throwable 类的子类，通常都是从 Exception 类或其子类来继承。一般是通过继承 Exception 类来定义自己的异常类。具体见示例。

```
/*
例子5.1，一个异常处理的例子。本例中System.in.read可能会产生的IOException为受查异常，在编译阶段就必须进行处理，否则编译器会报错。在当前状态下（即4、7、10、11、12行都注释掉的情况下），如果将该程序放在Eclipse中编译的话，会报错提示while语句所在的行有未处理的异常IOException。有两种解决方式：（1）去掉第4行中“throws IOException”前的注释符号，让main函数声明抛出异常。因为System.in.read()方法会抛出IOException异常，而程序中没有try-catch-finally语句进行捕获处理，故必须在main()方法的头部加上throws IOException，明确表示对该异常，程序不想处理，交由调用者处理。而main()方法的调用者是JVM（java虚拟机），一旦有异常，程序即使运行结束。（2）保留第4行的注释，而把7、10、11、12行的注释都去掉，用try－catch语句来捕获处理可能会出现的异常。
*/
1:    import java.io.*;
2:    public class Excep
3:    {
4:      public static void main(String args[])    //throws IOException
5:      {
6:         int c;
7:         //try{
8:         while((c=System.in.read())!=-1)
9:               System.out.println(c);
10:        //}
11:        //catch(IOException e)
12:        //{    e.printStackTrace();    }
13:     }
14:   }
```



```
/*
例子5.2，测试几个系统已定义的标准异常ArithmeticException。此异常在main函数调用会产生该异常的函数时产生。由于main函数的由JVM(java虚拟机)处理，所以该异常会导致程序终止。
在Eclipse平台上运行该程序后，程序终止，控制台提示如下信息：
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at ExcTest.doMoreStuff(ExcTest.java:9)
    at ExcTest.doStuff(ExcTest.java:6)
    at ExcTest.main(ExcTest.java:3)
*/
class ExcTest {
    public static void main (String [] args) {
        doStuff();
    }
    static void doStuff() {
        doMoreStuff ();
    }
    static void doMoreStuff () {
        int x = 5/0;    // 整数不能被0除! ArithmeticException在此被抛出。
    }
}


/*
例子5.3，测试几个系统已定义的标准异常。本例中有可能会产生两种异常，均用try－catch语句来捕获处理可能会出现的异常。
*/
import java.io.*;
public class test
{
    public static void main(String args[])
    {
        System.out.println("Please Input an Interger:");
        try{    //try块中的in.readLine可能会抛出IOException
            BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
            int a=Integer.parseInt(in.readLine());//把接收到的一行输入转换为整数，可能其中含有
                                                  //除数字以外的字符，因此可能会抛出NumberFormatException
            System.out.println("the Integer you input is:"+a);
            }
        catch(IOException e){           //捕获IOException并处理(输出提示信息)
            System.out.println("I/O error");
            }
        catch(NumberFormatException ne){ //捕获NumberFormatException并处理(输出提示信息)
            System.out.println("what you input is not an Integer!");
            }
    }
}


/*
例子5.4。 本例的功能是以只读的方式读取一个在D盘根目录的文件myfile.txt。其try块中可能会产生多种异
```

常。本例会告诉我们，尽管异常类有一个体系（从Exception继承而来依次有很多子类以及子类的子类用于表示异常），但我们在捕获处理异常时最好尽量明确。如果用父类，则在捕获时可以对应到多种异常。例如本例中的IOException类，它的子类有FileNotFoundException和EOFException等（可查API文档看看IOException有哪些子类）。所以下面的程序，第二个catch块实际上不会起作用了。它的情况已经被父类包含了。但如果把第二个catch块放在第一个catch块之前（即把10~14行这一段代码和15~19行这一段代码交换位置），那么编译阶段编译器就会提示一个错误：执行不到的FileNotFoundException的catch块，它已由IOException的catch块处理。 请大家测试、体会（**自己测试时请把代码前的行号去掉**）。

*/

```
 1: import java.io.*;
 2: public class ReadData {
 3:   public static void main(String args[]) {
 4:     try {
 5:       RandomAccessFile raf =
 6:         new RandomAccessFile("d://myfile.txt", "r");
 7:       byte b[] = new byte[1000];
 8:       raf.readFully(b, 0, 1000);
 9:     }
10:     catch(FileNotFoundException e) {
11:       System.err.println("File not found（文件不存在！）");
12:       System.err.println(e.getMessage());
13:       e.printStackTrace();
14:     }
15:     catch(IOException e) {
16:       System.err.println("IO Error（产生了I/O错误！）");
17:       System. err. println(e. toString());
18:       e.printStackTrace();
19:     }
20:   }
21: }
```



```
/*
```

例子5.5。一个关于“自定义异常”的例子。对于常见的出问题的情况，比如数组下标越界等，系统都定义好了各种异常类来表示，当这些异常事件发生的时候，系统就抛出对应的异常来。但是对于某些特定的情况（特别是跟设计者设计的程序的特定情况相关的情形），例如，设计者要求用户输入的数据在30到50之间这种情况，这就不具有普遍性，类似这样的情形就有无数种了，系统不可能全部都考虑到了，都定义好了异常来表示。这时候，就需要用户自己定义异常类来表示这种情况，当这种情况发生的时候，就抛出自己定义的异常类的对象来表示异常情况的发生。系统用户自定义异常类必须是Throwable类的子类，通常都是从Exception类或其子类来继承。一般是通过继承Exception类来定义自己的异常类。

```
*/
class MyExcep extends Exception{  //自定义一个异常类MyExcep
    public MyExcep()  //异常类的构造函数，一般根据需要而设计
    {
        System.out.println("本行的输出，是自定义的异常产生的！你注意到了吗？");
    }
}
```

```
public class comtest{

    static void func() throws MyExcep{ //func是一个会产生异常的函数，此处是为演示而设计的
        throw new MyExcep();  //func函数通过此语句抛出异常
    }
    public static void main(String args[]){
        try{
            func(); //func函数会产生异常，因此在main函数中调用它时要把它放在try中进行异常处理
        }
        catch(MyExcep e1){
            System.out.println("程序运行时产生了自己定义的一个异常，被捕获到了！");
        }
        finally   { //可要可不要的finally部分
            System.out.println("本行为finally部分产生的输出，finally部分是可选的!");
        }
    }
}


/*
 例子5.6，自己定义异常的另一个例子。请阅读程序及其注释来理解和体会。
 */
import java.util.Scanner;
```



```
class ArgumentOutOfBoundsException extends Exception { // 自定义一种异常
    ArgumentOutOfBoundsException() {
        System.out.println("输入错误！欲判断的数不能为负数！");
    }
}

public class UserExcp {
    public static boolean even(int m) throws ArgumentOutOfBoundsException {
        if (m % 2 != 0) { // 当发现m不是偶数时抛出自定义的异常ArgumentOutOfBoundsException
            ArgumentOutOfBoundsException ae = new ArgumentOutOfBoundsException();
            throw ae;
        } else { // 当m是偶数时，函数返回值为true
            return true;
        }
    }
    public static void main(String args[]) {
        int num;
        Scanner input = new Scanner(System.in);
        System.out.println("请输入一个正整数：");
        num = input.nextInt(); // 等待用户输入正整数num
        try {
            boolean result = even(num); // 调用方法even判断num是否为偶数
```

```
            System.out.println("你输入的是一个偶数！");
        } catch (ArgumentOutOfBoundsException e) {// num不是偶数时会产生异常并被此处捕获后输出提示
            System.out.println("产生的异常名称为：" + e.toString());
        }
    }
}
```

实验六　多线程

一、实验目的

1、掌握线程和多线程的概念。

2、掌握创建线程的两种方法及其区别。

3、了解线程的启动、终止、同步、互斥和优先级等概念。

二、实验类型

设计型

三、实验内容

1、编写一个程序，其功能是运行之后，其中有一个线程可以输出 20 次你的学号，另一个线程会输出 20 次你的姓名。

2、编写一个图形界面程序，运行之后，让其中有一个线程能在界面上“实时”显示系统当前时间（精确到秒获取时间可查询 java.util.Calendar 类，它包含了多个用于获得系统时间的函数）。另让一个线程可以在界面上提示当前系统时间下用户该做什么工作（例如当程序判断出系统时间现在是 8：00 到 9：00，则提示用户该上课；现在是 23：00 到 8：00，则提示用户该休息。 具体测试时可以将时间段限制到秒级，以便及时查看到程序运行中提示信息的变化）。

四、预备知识

一般操作系统中的多线程指的是在操作系统中同时运行几个应用程序，每个应用程序占用一个进程。实际上这些程序在单处理器的系统中并不是同时运行，而是操作系统将系统资源分配给各个程序，每个程序在 CPU 中交替执行，由于 CPU 的速度比较快，我们就感觉不出各程序是交替执行，而感觉多个程序是同时运行的。

在 java 虚拟机上执行的一个 java 程序，是操作系统中的一个进程。同一个 java 程序中的各个并发执行的代码片断，称为线程。线程可以说是程序中的一个可执行语句序列。每一个进程都有独立的代码和数据空间（进程上、下文），同一个进程中的各个线程拥有自己的运行栈和程序计数器，但都共享该进程的进程上、下文。线程是 cpu 调度切换的最小单位，创建和撤销一个线程比创建和撤销进程所需的开销小很多。进程间的通信比线程要慢得多，要求也相对高得多。

每个 java 应用程序运行时都至少有一个默认的线程，就是由 main 方法开始的线程，称为主线程。它的作用是启动程序进程，负责创建其它子线程，并负责完成整个进程最后的各种清理和关闭任务。通过 Thread 类，设计者自己可以建立其它所需的线程。有两种途径可以创建新的线程。

需要注意的是：在应用程序中使用多线程不会增加 CPU 的数据处理能力。只有在多 CPU 的计算机或者在网络计算体系结构下，将 Java 程序划分为多个并发执行线程后，同时启动多个线程运行，使不同的线程运行在基于不同处理器的 Java 虚拟机中，才能提高应用程序的执行效率。

多线程可用于一些动态效果的生成。另外，如果应用程序必须等待网络连接或数据库连接等数据吞吐速度相对较慢的资源时，多线程应用程序是非常有利的。基于 Internet 的应用程序有时须是多线程类型的，例如，当开发要支持大量客户机的服务器端应用程序时，可以将应用程序创建成多线程形式来响应客户端的连接请求，使每个连接用户独占一个客户端连接线程。这样，用户感觉服务器只为连接用户自己服务，从而缩短了服务器的客户端响应时间。

创建线程有两种方法，一是声明一个类的同时实现 Runnable 接口，这个类必须实现一个没有参数的 run 方法，run 方法中的代码即线程所要做的事情。

```
/*
 例子6.1，线程的第一种创建方法。该程序运行之后，系统中存在t1和t2两个线程并发执行。另外请注意由于
```

```
这两个线程是用同一个线程类定义的，因此它们是做同样事情的两个线程（或者更多个也可以）。如果两个线程所做的工作不同，则可以分别各自定义自己的线程类，然后用于创建对应的线程。
 */
class MyThread implements Runnable {
    public MyThread() {
        //构造函数的代码，根据需要来写
    }

    public void run() {
        for (int i = 1; i <= 10; i++) {
            System.out.println("第" + i + "次执行线程"+ Thread.currentThread().getName());
            try {
                Thread.currentThread().sleep(500);//睡眠500ms给其它线程运行机会，试试不要会怎样
            }
            catch (InterruptedException e) {
            }
        }
    }

    public static void main(String args[]) {
        Thread t1 = new Thread(new MyThread(),"thread 1"); // 创建线程1的对象,并
                                                           //通过第二个参数将其命名为thread 1
        Thread t2 = new Thread(new MyThread(),"thread 2"); // 创建线程2的对象,并
                                                           //通过第二个参数将其命名为thread 2
        t1.start(); //启动两个线程运行
        t2.start(); //虽然t2的启动表面上好像在后面，实际上两个线程的执行并无先后之分，
                    //t2的执行甚至有可能在t1之前，看系统的调度情况，可以多执行几次看看
    }
}
```


创建新线程的另一种途径是声明一个类为 Thread 的子类，并在子类中重写 Thread 类的 run 方法。run 方法中的代码即线程所要做的事情。

```
/*
例子6.2，线程的第二种创建方法。该程序运行之后，系统中存在t1和t2两个线程并发执行。另外请注意由于这两个线程是用同一个线程类定义的，因此它们是做同样事情的两个线程（或者更多个也可以）。如果两个线程所做的工作不同，则可以分别各自定义自己的线程类，然后用于创建对应的线程。
 */
class MyThread extends Thread { //通过继承系统提供的Thread类
    public MyThread() {
        //重写无参构造函数，若不需要也可去掉此函数
    }

    public MyThread(String name) {//重写另一个构造函数，若不需要也可去掉此函数
        super(name);  //调用父类的构造函数
```

```
    }

    public void run() {
        for (int i = 1; i <= 10; i++) {
            System.out.println("第" + i + "次执行线程"+ Thread.currentThread().getName());
            try {
                sleep(500);//睡眠500ms给其它线程运行机会，试试不要会怎样
            }
            catch (InterruptedException e) {
            }
        }
    }

    public static void main(String args[]) {
        Thread t1 = new Thread(new MyThread(),"thread 1"); // 创建线程1的对象,并
                                                           //通过第二个参数将其命名为thread 1
        Thread t2 = new Thread(new MyThread(),"thread 2"); // 创建线程2的对象,并
                                                           //通过第二个参数将其命名为thread 2
        t1.start(); //调用start方法启动两个线程运行
        t2.start(); //虽然t2的启动表面上好像在后面，实际上两个线程的执行并无先后之分，
                    //t2的执行甚至有可能在t1之前，看系统的调度情况，可以多执行几次看看
    }
}
```



```
/*
 例子6.3，一个源自scjp认证指南的例子，请体会。
 */
class NameRunnable implements Runnable {
    public void run() {
        for (int x = 1; x <= 3; x++) {
            System.out.println("Run by "
                    + Thread.currentThread().getName()+ ", x is " + x);
        }
    }
}
public class ManyNames {
    public static void main(String [] args) {
        // Make one Runnable
        NameRunnable nr = new NameRunnable();
        Thread one = new Thread(nr);
        Thread two = new Thread(nr);
        Thread three = new Thread(nr);

        one.setName("Fred");
        two.setName("Lucy");
```

```
        three.setName("Ricky");
        one.start();
        two.start();
        three.start();
    }
}
```

```
/*
 例子6.4。在java.util包中有很多工具类型的类，例如数学函数类等。其中还有一个Calendar类，其功能是用于获取日期时间等。可以将其import到自己的程序中进行使用。下面的例子演示其部分功能的使用。具体请查阅API帮助文档中关于Calendar类的详细说明。
*/
import java.text.SimpleDateFormat;
import java.util.Calendar;

public class myClock {
    public void showtime() // 定义一个用于显示当前系统时刻的函数
    {
        int y, m, d, h, mi, s;
        Calendar cal = Calendar.getInstance(); // 获取一个Calendar类的实例对象
        y = cal.get(Calendar.YEAR); // 获取年份
        m = cal.get(Calendar.MONTH); // 获取月份，获取的月份是从0到11表示一到十二月
        d = cal.get(Calendar.DATE); // 获取日期
        h = cal.get(Calendar.HOUR_OF_DAY); // 获取小时
        mi = cal.get(Calendar.MINUTE); // 获取分钟
        s = cal.get(Calendar.SECOND); // 获取秒钟
        System.out.println("现在时刻是" + y + "年" + (m + 1) + "月" + d + "日" + h
                + "时" + mi + "分" + s + "秒"); // 输出时刻
    }

    public static void main(String[] args) { // 通过Calendar类获取系统时刻

        myClock c = new myClock();

        for (int i = 0; i < 6; i++) { // 显示6次系统当前时间
            try {
                Thread.sleep(1000); // 两次显示时间之间休眠1000ms（即1秒），以便观察显示出的时间变化
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            c.showtime(); // 显示系统当前时刻
        }

        // 下面的4行代码演示如何以一种指定的格式显示时间
```

```
        SimpleDateFormat SDF = new SimpleDateFormat("yyyy'年'MM'月'dd'日'
'HH:mm:ss");// 格式化时间显示形式
        Calendar now = Calendar.getInstance();
        String time = SDF.format(now.getTime()); // 得到当前日期和时间
        System.out.println(time); // 以前面所设定的格式显示当前时间
    }
}


/*
例子6.5。在本次实验中或者其它有些时候，我们也许需要让两个线程都往一个图形界面上
各自显示一些内容。下面的例子演示了两个线程都往一个界面上显示字符串的参考做法。
*/
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JTextArea;
public class ThreadExample
{
    JFrame jf=new JFrame("线程都往界面上显示内容的例子");
    static JTextArea jta1,jta2;

    Thread trda=new thread1();      //线程trda
    Thread trdb=new thread2();      //线程trdb
                                    Java实验指导书（桂林电子科技大学汪华登）
    public ThreadExample()   //构造函数，生成图形界面
    {
        //setBounds(100,100,500,200);
        jf.setLayout(new FlowLayout());
        jta1=new JTextArea(15,30);
        jta2=new JTextArea(15,30);
        jf.add(jta1);    jf.add(jta2);  //将2个组件添加到界面上
        jf.setLocation(100, 150);
        jf.setVisible(true);
        jf.pack();

        trda.start();     //两个线程都启动
        trdb.start();
    }


    public static void main(String args[])
    {
        ThreadExample frm=new ThreadExample();
    }
}
```

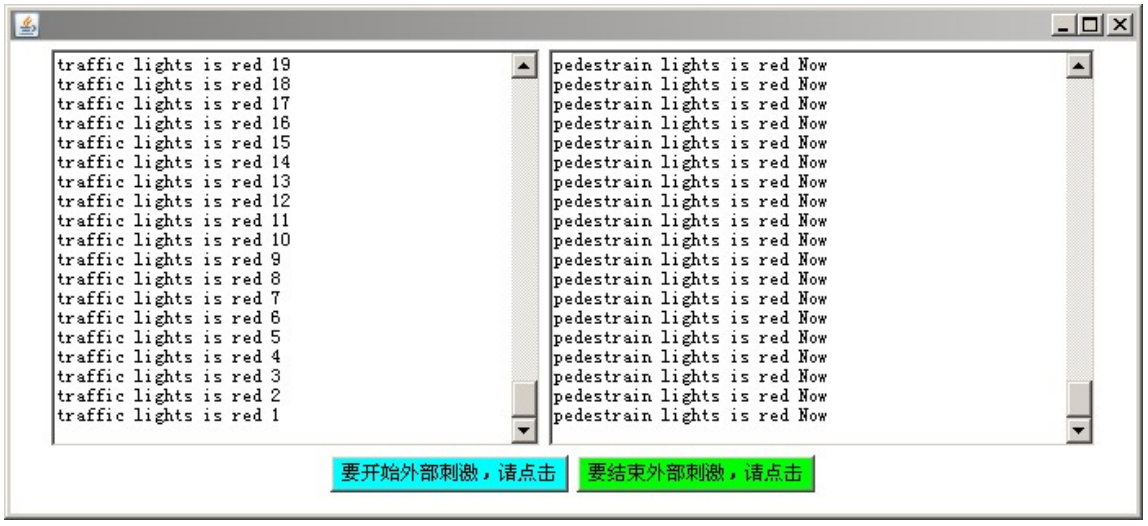
```
class thread1 extends Thread    //线程类thread1
{

    public void run()
    {

        ThreadExample.jta1.append("   这行代码是通过 thread 1 线程放进来的");

    }
}

class thread2 extends Thread    //线程类thread2
{
    public void run()
    {
        ThreadExample.jta2.append("   这行代码是通过 thread 2线程放进来的");
    }
}



/*
例子6.6，一个用多线程来模拟演示交通灯状态变化的图形界面例子程序。其运行结果如例子6.3的代码之后的图所示。要解决的问题是：
正常情况下，交通灯红、绿状态每次显示30秒，交替显示，而人行灯一直为红色。
当有外部刺激事件发生之后，人行灯变为绿色让行人走，此时交通灯就必须一直为红色，不能走车，

当外部刺激事件结束后，人行灯就恢复为红色，然后交通灯又不断地红绿交替各显示30秒，
直到外部刺激事件又再次发生，再重复上面的过程。

下面是对问题进行具体解决实现的图形界面参考程序，在Eclipse平台上运行测试通过，
如果不完善可以再根据要求对代码略作改进即可。
 */
import java.awt.*;
import java.awt.event.*;

class ShareData              //这个类，用于定义两个线程共享的对象
{
    private int sd;          //共享对象的成员变量sd
    public ShareData()
    {
        sd=0;                 //共享对象的成员变量初值初始化为0
    }
    public int getsd()       //这个函数用于获取sd的值
    {
        return sd;
    }
```

```
    public void change_sd()  //用于外部刺激事件发生的时候来调用，用于改变sd的值为1
    {
        sd=1;
    }
    public void reset_sd()   //当刺激事件结束，恢复sd的值为0
    {
        sd=0;
    }
}

class traffic extends Frame implements ActionListener
{
    static TextArea p1,p2;
    Button b1=new Button("要开始外部刺激，请点击");   //定义用于点击来表示刺激事件发生的按钮
    Button b2=new Button("要结束外部刺激，请点击");     //定义用于点击来表示刺激事件结束的按钮
    ShareData so=new ShareData();                //定义两个线程用来共享的对象so

    Thread trda=new trafficlight(so);       //交通灯线程
    Thread trdb=new pedestrianlight(so);    //人行灯线程

    public traffic()   //构造函数，生成图形界面
    {
        b1.setBackground(Color.CYAN);
        b2.setBackground(Color.green);
        setBounds(100,100,500,200);
        setVisible(true);
        setLayout(new FlowLayout());
        p1=new TextArea(15,40);
        p2=new TextArea(15,45);
        add(p1);    add(p2);    add(b1);     add(b2);  //几个组件添加到界面，显示图形界面
        b1.addActionListener(this);   b2.addActionListener(this);
        pack();
        addWindowListener(
                new WindowAdapter()
                {
                    public void windowClosing(WindowEvent e)
                    {
                        System.exit(0);
                    }
                }
        );
        trda.start();
        trdb.start();
    }

    public void actionPerformed(ActionEvent button)  //事件处理的代码
```

```
    {
        // TODO 自动生成方法存根
        if(button.getSource()==b1)  //当点击按钮b1时，刺激事件“发生”
        {
            so.change_sd();          //共享对象的成员变量sd的值从0变为1
        }
        if(button.getSource()==b2)  //当点击按钮b2时，刺激事件“结束”
        {
            so.reset_sd();           //共享对象的成员变量sd的值从1变为0
        }
    }

    public static void main(String args[])
    {

        traffic frm=new traffic();
    }
}

class trafficlight extends Thread    //交通灯线程
{
    ShareData tl=new ShareData();
    public trafficlight(ShareData so)//构造函数，传递共享对象so给trafficlight线程
    {
        tl=so;
    }
    public void run()
    {
        int i,j;
        while(true)  //无限循环，表示交通灯一直工作
        {

            for(i=30;i>0;i--)   //让交通灯显示绿灯30秒
            {
                while(tl.getsd()==1)   //判断是否发生刺激事件
                {
                    traffic.pl.append("because of stimulation,traffic lights is red "+"\n");
                    try
                    {
                        sleep(400);  //为了看运行结果比较快，设置为400ms而非1秒
                    }
                    catch(InterruptedException e)
                    {
                        e.printStackTrace();
                    }
                    i=30;   //这是一个细节处理，当刺激事件结束，交通灯重新开始工作亮绿灯时，
```

```
                        //为了让它能重新把绿灯完整显示30秒，把i值复位为30。 因为i有可能
                        //是在交通灯亮绿灯 X 次之后因为刺激事件而有一个(30-X) 的值,如果
                        //不需要考虑这个问题，那么就可以把“i=30;”这条语句注释掉。
                    }

                    traffic.pl.append("traffic lights is green "+i+"\n");
                    try
                    {
                        sleep(400);
                    }
                    catch(InterruptedException e)
                    {
                        e.printStackTrace();
                    }
                }  //end for

                for(j=30;j>0;j--) //让交通灯显示红灯30秒
                {
                    traffic.pl.append("traffic lights is red "+j+"\n");
                    try
                    {
                        sleep(400); //为了看运行结果比较快，设置为400ms而非1秒
                    }
                    catch(InterruptedException e)
                    {
                        e.printStackTrace();
                    }

                } //end for
            }//end while
        }//end run
}

class pedestrianlight extends Thread    //人行灯线程
{
    ShareData pl=new ShareData();
    public pedestrianlight(ShareData so)  //构造函数，传递共享对象so给pedestrainlight线程
    {
        pl=so;
    }
    public void run()
    {
        while(true)   //无限循环，表示人行灯一直工作
        {
            if(pl.getsd()==0)  //当没有外部事件发生，或者外部刺激事件结束时
            {
```

```
                    traffic.p2.append("pedestrain lights is red Now\n");    //就显示红色
                    try{
                        sleep(400);  //为了看运行结果比较快，设置为400ms而非1秒
                    }
                    catch(InterruptedException e)
                    {
                        e.printStackTrace();
                    }
                }
                else
                {
                    traffic.p2.append("because of stimulation,pedestrain lights is Green \n"); //外部事件发生，显示绿色
                    try
                    {
                        sleep(400); //为了看运行结果比较快，设置为400ms而非1秒

                    }
                    catch(InterruptedException e)
                    {
                        e.printStackTrace();
                    }
                }
            }
        }
}
```

线程具有启动、终止、同步、互斥和优先级等概念以及相应的程序设计要求，在进一步学习或有需要的时候可以通过查阅相关参考书和资料进行了解。

```
/*
 * 例子6.7。下面是一个应用了线程的、简单的万年历的例子。其项目（project）由两个
```

```
 * 程序构成，分别是MainFrame.java和Clock.java，两个程序的源代码如下。供参考。
 */
////////////////////////////////////////////////////
////   万年历例子的2个程序之第1个：MainFrame.java     ////
////////////////////////////////////////////////////
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Date;
import java.util.Calendar;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class MainFrame extends JFrame {
    private static final long serialVersionUID = 1L;
    JPanel panel = new JPanel(new BorderLayout());
    JPanel panel1 = new JPanel();
    JPanel panel2 = new JPanel(new GridLayout(7, 7));
    JPanel panel3 = new JPanel();
    JLabel[] label = new JLabel[49];
    JLabel y_label = new JLabel("年份");
    JLabel m_label = new JLabel("月份");
    JComboBox com1 = new JComboBox();
    JComboBox com2 = new JComboBox();
    int re_year, re_month;
    int x_size, y_size;
    String year_num;
    Calendar now = Calendar.getInstance(); // 实例化Calendar

    MainFrame() {
        super("一个简单的应用了线程的万年历的例子");
        setSize(300, 350);
        x_size = (int) (Toolkit.getDefaultToolkit().getScreenSize().getWidth());
        y_size = (int) (Toolkit.getDefaultToolkit().getScreenSize().getHeight());
        setLocation((x_size - 300) / 2, (y_size - 350) / 2);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        panel1.add(y_label);
        panel1.add(com1);
        panel1.add(m_label);
        panel1.add(com2);
        for (int i = 0; i < 49; i++) {
```

```
            label[i] = new JLabel("", JLabel.CENTER);// 将显示的字符设置为居中
            panel2.add(label[i]);
        }
        panel3.add(new Clock(this));
        panel.add(panel1, BorderLayout.NORTH);
        panel.add(panel2, BorderLayout.CENTER);
        panel.add(panel3, BorderLayout.SOUTH);
        panel.setBackground(Color.white);
        panel1.setBackground(Color.white);
        panel2.setBackground(Color.white);
        panel3.setBackground(Color.white);
        Init();
        com1.addActionListener(new ClockAction());
        com2.addActionListener(new ClockAction());

        setContentPane(panel);
        setVisible(true);
        setResizable(false);
    }

    class ClockAction implements ActionListener {
        public void actionPerformed(ActionEvent arg0) {
            int c_year, c_month, c_week;
            c_year = Integer.parseInt(com1.getSelectedItem().toString()); // 得到
当前所选年份
            c_month = Integer.parseInt(com2.getSelectedItem().toString()) - 1; //
得到当前月份，并减1,计算机中的月为0－11
            c_week = use(c_year, c_month); // 调用函数use，得到星期几
            Resetday(c_week, c_year, c_month); // 调用函数Resetday
        }
    }

    public void Init() {
        int year, month_num, first_day_num;
        String log[] = { "日", "一", "二", "三", "四", "五", "六" };
        for (int i = 0; i < 7; i++) {
            label[i].setText(log[i]);
        }
        for (int i = 0; i < 49; i = i + 7) {
            label[i].setForeground(Color.red); // 将星期日的日期设置为红色
        }
        for (int i = 6; i < 49; i = i + 7) {
            label[i].setForeground(Color.green);// 将星期六的日期设置为绿色
        }
        for (int i = 1; i < 10000; i++) {
            com1.addItem("" + i);
```

```
        }
        for (int i = 1; i < 13; i++) {
            com2.addItem("" + i);
        }
        month_num = (int) (now.get(Calendar.MONTH)); // 得到当前时间的月份
        year = (int) (now.get(Calendar.YEAR)); // 得到当前时间的年份
        com1.setSelectedIndex(year - 1); // 设置下拉列表显示为当前年
        com2.setSelectedIndex(month_num); // 设置下拉列表显示为当前月
        first_day_num = use(year, month_num);
        Resetday(first_day_num, year, month_num);
    }

    public int use(int reyear, int remonth) {
        int week_num;
        now.set(reyear, remonth, 1); // 设置时间为所要查询的年月的第一天
        week_num = (int) (now.get(Calendar.DAY_OF_WEEK));// 得到第一天的星期
        return week_num;
    }

    public void Resetday(int week_log, int year_log, int month_log) {
        int month_day_score; // 存储月份的天数
        int count;
        month_day_score = 0;
        count = 1;

        Date date = new Date(year_log, month_log + 1, 1); // now
        Calendar cal = Calendar.getInstance();
        cal.setTime(date);
        cal.add(Calendar.MONTH, -1); // 前个月
        month_day_score = cal.getActualMaximum(Calendar.DAY_OF_MONTH);// 最后一天

        for (int i = 7; i < 49; i++) { // 初始化标签
            label[i].setText("");
        }
        week_log = week_log + 6; // 将星期数加6，使显示正确
        month_day_score = month_day_score + week_log;
        for (int i = week_log; i < month_day_score; i++, count++) {
            label[i].setText(count + "");
        }
    }

    public static void main(String[] args) {
        JFrame.setDefaultLookAndFeelDecorated(true);
        new MainFrame();
    }
}
```

```
////////////////////////////////////////////////////
////     万年历例子的2个程序之第2个：Clock.java    ////
////////////////////////////////////////////////////
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.text.SimpleDateFormat;
import java.util.Calendar;

class Clock extends Canvas implements Runnable {
    MainFrame mf;
    Thread t;
    String time;

    public Clock(MainFrame mf) {
        this.mf = mf;
        setSize(280, 40);
        setBackground(Color.white);
        t = new Thread(this); // 实例化线程
        t.start(); // 调用线程
    }
                                Java实验指导书（桂林电子科技大学汪华登）
    public void run() {
        while (true) {
            try {
                Thread.sleep(1000); // 休眠1秒钟
            } catch (InterruptedException e) {
                System.out.println("异常");
            }
            this.repaint(100);
        }
    }

    public void paint(Graphics g) {
        Font f = new Font("宋体", Font.BOLD, 16);
        SimpleDateFormat SDF = new SimpleDateFormat("yyyy'年'MM'月'dd'日
'HH:mm:ss");// 格式化时间显示类型
        Calendar now = Calendar.getInstance();
        time = SDF.format(now.getTime()); // 得到当前日期和时间
        g.setFont(f);
        g.setColor(Color.orange);
        g.drawString(time, 45, 25);
    }
}
```

实验七　流与文件

一、实验目的

1、掌握 java 中流的概念和作用。

2、掌握文件读写所使用的相关流（类）。

二、实验类型

设计型

三、实验内容

1、编写一个 Java 程序，能将硬盘上某个文件夹下的一个纯文本文件（如 txt 文件）拷贝到另一个指定的文件夹中。

2、假设已经有一个文本文件中存放着职工的工资记录。每个职工的记录包含姓名、工资、津贴三项。每条记录可以存放于一行（三项之间的间隔符可以自己决定），也可以将每条记录的三项依次分别存放在文本文件中的每一行。请设计一个程序，可以让用户选择打开这个文件查看其内容，并可以让用户选择把每个职工的工资增加 10%之后又存回原来的文件。

四、预备知识

完成本题可能需要用到字符串与数值之间的相互转换，请查阅指导书实验二的内容。

本次实验中，我们首先要了解 java 中一个重要的概念——流。java 并不在语言层面上对输入输出提供支持，而是将这个任务交由类库的类来完成。Java 语言中，输入/输出由两个包组成：java.io 包和 java.nio 包。“流”在 java 的 I/O 中起着重要的作用，它是一个逻辑概念，一个字节输入流是指一个字节序列。可从中依次读出字节，用户可以不必关心它的内部结构、来源。字节输出流类似。“流”模型带来的最大好处是：将文件读/写，网络读/写，内存读/写等操作全部统一起来，在“流”的层次上达到操作的一致性。Java.io 包中约有 60 个类，主要分为：字节流的处理、字符流的处理、对象序列化和随机文件处理等。其中所有字节“输入”流类都是抽象类 InputStream 类的子类，所有字节“输出”流类都是抽象类 OutputStream 类的子类；所有字符“输入”流类都是抽象类 Reader 类的子类，所有字符“输出”流类都是抽象类 Writer 类的子类。下面作简单说明。

1、InputStream 类

该类是所有字节输入流的根类。方法均 throws IOException，其所具有的部分方法如下：

int read() throws IOException 方法的用途是从输入流中读取下一个字节流数据，其返回值是 0～255 之间的一个整数。若读到流结束，返回－1，若流中暂时无数据可读，则阻塞。

void close()函数可以关闭流并释放系统资源，通常系统对流对象进行垃圾收集时会自动调用此函数。

long skip(long n)方法将输入流中当前读取的位置向后移动 n 字节，并返回实际跳过的字节数。

void mark(int readlimit)方法是在输入流的当前读取位置作标记，从该位置开始读取 readlimit 所指的数据后，所做的标记失效。

2、OutputStream 类

该类是所有字节输出流的根类。方法均 throws IOException，其所具有的部分方法如下：

void write(int b)方法为抽象方法，必须被子类实现。该方法用来将指定的字节 b 作为数据写入输出流。

void write(byte b[])将字节数组 b 中长度为 b.length 个字节的数据写入输出流。

void write(byte b[],int off,int len)是将字节数组 b 中从索引 off 开始的长度为 len 个字节的数据写入输出流。

flush()是清空输出流，并强制输出流中剩余的字节。

3、本次实验可能会用到的 FileInputStream 类和 FileOutputStream 类

它们分别直接继承自 InputStream 类和 OutputStream，而且它们重写或实现了父类中的所有方法，通过这两个类可以打开本地机器的文件，按字节进行顺序的读写。

（1） 在初始化 FileInputStream 对象时，必须捕获异常或声明抛出异常，否则编译会出错。

（2） FileInputStream 类不支持 mark（）和 reset（）方法。

（3） 在构造 FileOutputStream 对象时，如果指定路径的文件不存在，会自动构建一个新的文件，如果原来的文件存在，则本次写入的内容会覆盖原来文件的内容。

4、字符输入输出流的基类 Reader 和 Writer

这两个类是字符输入输出流类的根类。其子类 FileReader 类和 FileWriter 类可以字符为单位读写文件。他们支持 Unicode 字符流的读写。其所具有的方法请查阅 API 文档了解。

5、PrintWriter

它是 Writer 类的一个子类，用于以文本格式输出字符串和数字。与 DataOutputStream 类似，PrintWriter 也是有输出方法但无目的地。它必须与一个输出流（如 OutputStreamWriter，FileOutputStream）相结合使用。

Java 中的流的基本分类表

<table>
<tr><th colspan="4">字节流</th><th colspan="4">字符流</th></tr>
<tr><td colspan="2">输入流</td><td colspan="2">输出流</td><td colspan="2">输入流</td><td colspan="2">输出流</td></tr>
<tr><td>以字节为基本单位</td><td>以java中基本数据类型为单位</td><td>以字节为基本单位</td><td>以 java 中基本数据类型为单位</td><td>以字节为基本单位</td><td>以 java 中基本数据类型为单位</td><td>以字节为基本单位</td><td>以java中基本数据类型为单位</td></tr>
<tr><td>InputStream 类(根类)；

FileInputStream 类(文件字节输入流，在生成这类对象时，需处理 FileNotFoundException 异常)；

System.in 对象</td><td>DataInputStream 类：</td><td>OutputStream 类(根类)；

FileOutputStream 类(文件字节输出流。在生成其对象时，若指定的文件不存在，则新建一个文件，若已存在，则清除原文件的内容，需处理 IOException)；

System.out 对象</td><td>DataOutputStream 类</td><td>Reader 类(抽象类，根类)；
FileReader 类(用于字符文件的读，每次读取一个字符或字符数组)；
InputStreamReader 类(字节流向字符流转换的桥类。字符流是建立在字节流的基础之上的，</td><td>没有专门的类，在 BufferedReader 类读入串后再解析串</td><td>Writer 类(抽象类，是所有字符输出的根类)；

FileWriter 类；

OutputStreamWriter 类(参看 InputStreamReader 类)</td><td>PriterWriter</td></tr>
<tr><td colspan="4">有时按字节为单位进行读写处理并不方便，如一个二进制文件中存放有 100 个整数，从中读取时，希望按 int(4 字节)为单位读取，每次读取一整数而不是一字节。DataInputStream 和 DataOutputStream 类的对象是过滤流，可以将基本字节输入输出流自动转换成基本数据类型进行读写的过滤流(通过 readChar()、readDouble()、readFloat()、readInt()等方法。DataOutputStream 类似。它们可以按与机器无关的风格读取 Java 原始数据。</td><td colspan="4">BufferedReader 类和 BufferedWriter 类：文本行是以回车、换行结束的字符序列，以文本行为基本单位进行文本读取与处理有时更为方便，这两个流是带缓冲的字符流。</td></tr>
<tr><td colspan="4">BufferedInputStream 类和 BufferedOutputStream 类的对象将 1 字节流专成一个带缓冲的字节流。当读取数据时，数据按块读入缓冲区，其后的读写操作直接访问缓冲区，可提高性能，并使得 BufferedInputStream 流支持 mark()、reset()、skip()等方法。 对应于字符流操作的类相应为 BufferedReader 类和 BufferedWriter 类。</td><td colspan="4">文本文件的输出流有 FileWriter 类，该类通常以字符为单位写文本文件。有时需要以 java 中的基本数据类型为单位进行文本文件的写入，如将实数值 13.5 以文本方式写入文本文件中，尽管 FileWriter 也能写入，但处理起来不方便，而 PrintWriter 类可以很方便地进行此类文本文件的输出，例如方法 public void print(×× v)自动以字符串格式输出 v 的值，其中××可为任何类型，包括引用。若 v 是对象，则自动调用 v 中的 toString()方法转换成字符串，然后再输出。</td></tr>
<tr><td colspan="8">RandomAccessFile 类：是直接继承自 Object 的完全独立的类，它的对象提供了对随机读/写文件的支持。如：RandomAccessFile raf=new RandomAccessFile(“d:\\t.dat”,”rw”);它将随机读写的字节文件视为一个巨大的字节数组，这个“数组”的下标就是所谓的文件指针。其操作主要有：long getFilePointer()//得到当前的文件指针位置；void seek(long pos)//移动文件指针到指定的位置，从 0 开始计算位置。int skipBytes(int n)//文件指针向文件末尾移动 n 字节，返回移动的字节数。 以及其它很多方法。</td></tr>
</table>

有时会进行关于文件与目录的操作，很可能会用到 File 类。它是以与平台无关的方式描述一个文件或目录对象的属性。利用 File 类中的方法，可以获取文件或目录的各种属性信息。如名称、文件长度、路径等。还可以创建新的目录、对文件或目录改名、删除文件或目录、列出某目录下所有的文件与子目录等。但 File 类并不涉及对文件的读写操作。

File 类的构造函数以及相关的部分方法有：

public File(String path) //由文件或目录路径名创建 File 对象

public File(String path,String name)

public File(File dir, String name)
String getName() //返回该 File 对象所表示的文件或目录的名称，不含路径
String getPath() //返回文件或目录的路径名
String getAbsolutePath() //返回绝对路径
String getParent() //返回上一级目录名
String rename(File newName) //将该 File 对象所表示的文件名改名为 newName
boolean exists() //测试当前 File 对象所指示的文件或目录是否存在
boolean canWrite() //测试当前文件是否可写
boolean canRead() //测试可读否
boolean delete() //删除 File 对象所表示的文件或目录
boolean mkdir() //创建由该 File 对象所代表的子目录
long length() //返回当前文件的长度，以字节为单位

在本实验中，也将可能在进行工资数据改变时，需要将文件中读到的字符串类型的数据转换成数值，而在将数值存回文件时，可能又需要将数据转换成字符串。将字符串和数值两种数据类型进行转换，在实验 4 中已各列举了两种方法，请查阅本实验指导书中实验四的预备知识部分。也可查阅 API 文档中对应的 String 类和 Float、Integer 等数据类型类。

```
/*
例子7.1。读取一个文本文件并显示其内容。
*/
import java.io.*;
public class File1   //用FileInputStream类读取文件到字节缓冲区buf中，然后又将其在控制台输出。
{
    public static void main(String args[])
    {
        byte buf[]=new byte[1000];  //定义一个1000个字节的缓冲区
        try
        {
            FileInputStream fis=new FileInputStream("d:/file1.txt");  //注意不能用“\”
            int rd=fis.read(buf,0,1000);        //将文件内容读取到字节缓冲区中
            String str=new String(buf,0,rd);   //将缓冲区内容转换为字符串
            System.out.println(str);           //输出转换为字符串后的文件
            fis.close();                      //关闭流
        }
        catch(IOException e)
        {
            System.out.println("File read Error");
        }
    }
}
```

```
/*
例子7.2。读取文件并显示其内容（图形界面程序）。
```

```
*/
import java.io.*;
import java.awt.*;
import java.awt.event.*;

public class File2//一个用AWT组件实现了简单图形界面的、通过FileInputStream实现文件读取功能的程序
{
    public static void main(String args[])
    {
        File2Frm frm=new File2Frm();
    }
}

class File2Frm extends Frame
{
    File2Frm()
    {
        int rd;
        setVisible(true);
        setSize(100,80);
        TextArea tarea=new TextArea(20,80); //定义一个TextArea
        add(tarea);
        byte buf[]=new byte[10];   //定义一个字节缓冲区
        pack();
        addWindowListener(
            new WindowAdapter(){    //使用事件适配器实现关闭图形窗口的功能
                public void windowClosing(WindowEvent e){
                    System.exit(0);}});
        try
        {
            File fl=new File("d:/file2.txt"); //准备读取D盘根目录下的文件file2.txt
            FileInputStream fis=new FileInputStream(fl);
            while((rd=fis.read(buf,0,1))!=-1)
            {
                String str=new String(buf,0,1);  //将读取的字节转换为字符串
                tarea.append(str); //将字符串添加到TextArea中（即将文件内容显示在TextArea中）
            }
            fis.close();  //关闭流
        }
        catch(IOException e)
        {
            System.out.println("File read Error（文件不存在！）");
        }
    }
}
```

```
/*
 * 例子7.3。  将键盘输入的一些内容存到一个文件当中。
 */
import java.io.*;
public class File3
{
    public static void main(String args[])
    {
        byte buf[]=new byte[1000];
        try
        {
            System.out.println("请从键盘输入某些字符，按回车键结束运行！");
            int rd=System.in.read(buf);    //输入内容存入buf中，实际输入字符的个数存入rd中
            FileOutputStream fos=new FileOutputStream("d:\\file3.txt");//准备将输入内容存入D盘的file3,
                                                                       //若该文件不存在，则会先创建该文件
            fos.write(buf,0,rd);   //将输入到缓冲区buf中的内容写入到fos所代表的文件file3.txt中
        }
        catch(IOException e)
        {
            System.out.println("Output Error!");
        }
    }
}
```



```
/*
 * 例子7.4。 应用文件对话框对象，实现具备文件选择、读取、显示、存储等功能的图形界面应用程序。
 */
import java.io.*;
import java.awt.*;
import java.awt.event.*;
public class File4
{
    public static void main(String args[])
    {
        File4Frm frm=new File4Frm();
    }
}

class File4Frm extends Frame implements ActionListener
{
    FileDialog sv,op;   //定义文件对话框对象sv，op
    Button btn1,btn2,btn3;
    TextArea tarea;
```

```
File4Frm()
{
    super("打开和保存文件");
    setLayout(null);
    setBackground(Color.cyan);
    setSize(600,300);
    setVisible(true);
    btn1=new Button("Open");
    btn2=new Button("Save");
    btn3=new Button("Close");
    tarea=new TextArea("");
    add(btn1); add(btn2); add(btn3); add(tarea);
    tarea.setBounds(30,50,460,220);
    btn1.setBounds(520,60,50,30);
    btn2.setBounds(520,120,50,30);
    btn3.setBounds(520,180,50,30);
    sv=new FileDialog(this,"保存",FileDialog.SAVE);  //保存功能
    op=new FileDialog(this,"打开",FileDialog.LOAD);  //打开功能
    btn1.addActionListener(this);
    btn2.addActionListener(this);
    btn3.addActionListener(this);
    addWindowListener   //定义事件适配器实现图形界面窗口的关闭功能
    (
        new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        }
    );
}

public void actionPerformed(ActionEvent e)  //界面上的事件处理
{
    if(e.getSource()==btn1)  //如果点击的是open按钮
    {
        String str;
        op.setVisible(true);
        try   //将文件打开读取到界面上的TextArea组件中显示出来
        {
            File fl=new File(op.getDirectory(),op.getFile());
            FileReader fr=new FileReader(fl);
            BufferedReader br = new BufferedReader(fr);
            tarea.setText("");
            while((str=br.readLine())!=null)tarea.append(str+'\n');
```

```
                fr.close();
            }
            catch(Exception e1)
            {
            }
        }
        if(e.getSource()==btn2)  //如果点击的是save按钮
        {
            sv.setVisible(true);
            try   //将TextArea中的内容写入到文件中保存
            {
                File fl=new File(sv.getDirectory(),sv.getFile());
                FileWriter fw=new FileWriter(fl);
                BufferedWriter bw = new BufferedWriter(fw);
                String gt=tarea.getText();
                bw.write(gt,0,gt.length());
                bw.flush();
                fw.close();
            }
            catch(Exception e1)
            {
            }
        }
        if(e.getSource()==btn3)  //如果点击的是close按钮
        {
            System.exit(0);
        }
    }
}


/*
 例子7.5，关于对象序列化与文件。
 正常情况下，对象用完了以后会被销毁而不复存在。如果要存储对象，可以通过一种叫做“对象序列化”（有时又称作“对象串行化”）的机制，它主要是通过定义类的同时实现（implements）Serializable接口来完成。可以将对象存入文件，并可以从文件中将对象从文件中读出来。下面的例子演示了这一点。
 */
import java.io.*;
import java.util.*;

class Point implements Serializable {  //通过实现Serializable接口来实现对象序列化。
    private int x;
    private int y;

    public Point(int x, int y) {   //定义Point类的第一个构造函数
        this.x = x;
```

```
        this.y = y;
    }

    public Point(int x) {   //定义Point类的第二个构造函数
        this(x, 0);
    }

    public Point() {   //定义Point类的第三个构造函数
        this(0, 0);
    }

    public String toString() {
        return "(" + x + "," + y + ")";
    }
}

public class SeriaTest {    //在这个类中将用到上一个Point类
    public static void main(String args[]) {
        try {
            ObjectOutputStream oos = new ObjectOutputStream(
                    new FileOutputStream("d:\\pt.ser")); //对象“流”
            for (int k = 0; k < 3; k++) {
                oos.writeObject(new Point(k, 2 * k));   //将对象写入oos所代表的文件中
            }
            oos.flush();  //清空
            oos.close();  //关闭

            ObjectInputStream ois = new ObjectInputStream(new FileInputStream(
            "d:\\pt.ser"));   //准备以对象为单位读取文件

            for (int k = 0; k < 3; k++) {
                Point pt = (Point) ois.readObject();  //读取文件中的对象
                System.out.print(pt + "  ");    //输出对象内容
            }
            ois.close();
        }

        catch (FileNotFoundException e) {    //处理上述读写过程中可能发生的异常
        }
        catch (IOException ee) {
        }
        catch (ClassNotFoundException eee) {
        }
    }
}
```

```
/*
 * 例子7.6。将一个文件中的某行数据读出来后，也许我们需要对其所包含的几个字符串（数据）
 * 进行分割。使用本例子程序中所演示的字符串类的split方法，可以将字符串"happy new near"
 * 以空格为间隔分为3个单词。但实际上查看API帮助文档我们会发现使用split方法的前提是其参数应
 * 是一个合法的正则表达式。否则结果将不符合预期。例如下面的例子是可以达到目的的。但是如果把
 * 要分割的* 字符串中的空格替换成符号”|”，那么会发现结果不符合预期。此时我们可以使用的另外
 * 一种方法是使用StringTokenizer类分解字符串，请看后面的例子7.7。
 */
 public class split {
    public static void main(String[] args) {
        String str="happy new near"; //要分割的字符串
        String s[];
        s=str.split(" ");  //以空格为分隔符进行分割，分割结果存放在数组中

        for(int i=0;i<s.length;i++)
            System.out.println(s[i]);  //输出分割结果
    }
}


/*
 * 例子7.7。将一个文件中的某行数据读出来后，也许我们需要对其所包含的几个字符串（数据）
 * 进行分割。下面的例子演示使用StringTokenizer类来分割字符串。
 */
import java.util.StringTokenizer;
public class split {
    public static void main(String[] args) {
        String str="happy|new|near";  //要分割的字符串为str
        StringTokenizer st = new StringTokenizer(str,"|"); //指定用|分割
        while(st.hasMoreTokens())
            System.out.println(st.nextToken()); //把子串依次取得后输出
    }
}
```

实验八　网络

一、实验目的

1、掌握 TCP 和 UDP 网络程序设计的模型。

2、掌握 TCP 和 UDP 程序设计的基本方法和所使用的系统类。

3、了解基于应用层 http 协议的程序的设计以及使用系统类的组播、广播等网络程序设计。

二、实验类型

设计型。

三、实验内容

1、先熟悉所给的示例，了解网络编程的一般基本概念。然后两个同学为一组，分别设计一对控制台下的 TCP 通信程序的客户端和服务器端，然后双方要能够相互发送和反馈信息进行无限次连续通信，直到其中一方发送表示结束通信的“end”字符串，然后接收方也返回一个“end”，双方结束通信。

若可能的话，建议最好设计出图形界面下的程序（即设计成一个简单的网络聊天程序）。可在本地机上做测试，或两个同学为一组在不同的机器上作测试都可以，要将服务器地址和端口号作为参数，在运行中可以进行修改。对于本机上测试，服务器地址使用 127.0.0.1 或者本机实际 IP 地址。在网内不同机器上测试，要给出正确的服务器 IP 地址或名称。

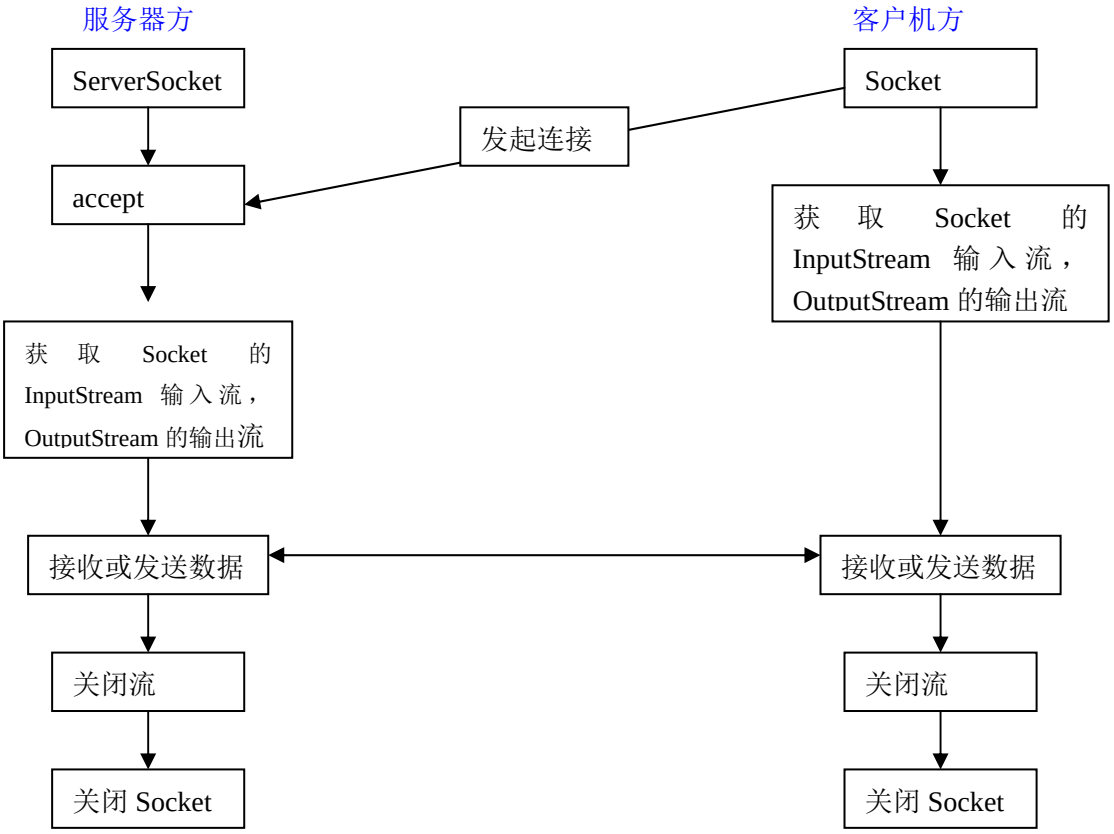
2、观察和分析 QQ 等即时通讯软件的界面和功能，通过查阅资料，思考和分析其设计思想和实现方法，设计一款自己的通讯软件，实现诸如消息传递、好友管理、系统设置、文件收发等若干功能。

四、预备知识

java 对网络程序设计进行支持的类主要包含在 java.net 包中，它主要支持 TCP/UDP 及其上层的网络程序设计。本实验主要涉及：TCP 通信、UDP 通信以及部分 URL 通信的内容。

(一) TCP 网络程序设计

TCP 是面向连接的可靠的字节流的传输。TCP 通信模型是典型的客户机－服务器模型。在 java 语言中，通信双方的通信模型如下：



1、 InetAddress 类

TCP 通信的双方使用 Socket 进行通信，通信一方的 IP 地址及其端口号，合称为一个 Socket，通信双方各自的 Socket，唯一地标识了本次双方的通信。Socket 中的 IP 地址唯一标识了一台主机，而端口号则唯一标识了该通信主机上的一个程序（或进程）。端口号从 0 到 65535，其中 1 到 1023 一般已规定好其用途，最好不要使用。 InetAddress 类的对象用于表示 IP 地址，该类没有明确定义构造函数，常用的方法如下表所示（注意大多为 static 方法）：

方法	功能
Public static InetAddress[] getAllByName(String host)throws UnknowHostException	返回主机名 host 所对应的所有 IP，每一个 IP 用一个 InetAddress 对象表示，结果返回的是一个数组
Public static InetAddress getByName(String host)throws UnknowHostException	返回返回主机名 host 所对应的一个 IP，若该主机对应对歌 IP，则随机返回其中一个 IP
Public static InetAddress getLocalHost()throws UnknowHostException	返回本地主机的 IP 地址
其它方法省略	

```
///////下面的例子，可以返回域名相应的 IP，若参数里没有给出域名，则返回本地主机的 IP 地址//////////////
import java.net.InetAddress;
public class nettest {
    public static void main(String[] args)throws Exception {
        if(args.length>0)
        {
            InetAddress[] addr=InetAddress.getAllByName(args[0]);
            for(int i=0;i<addr.length;i++)
                System.out.println(addr);
        }
```

```
        else
        {
            InetAddress addr=InetAddress.getLocalHost();
            System.out.println("native host ip: "+addr);
        }
    }
}
```

2、 ServerSocker 类

该类的对象代表服务器方的监听 Socket，等待客户机发起 TCP 连接，然后返回一个用于与该主机进行 TCP 通信的 Socket 对象。构造函数有：

public ServerSocket(int port)throws IOException

public ServerSocket(int port, int backlog)throws IOException　//backlog 代表连接队列的最大长度

public ServerSocket(int port, int backlog，InetAddress bindAddr)throws IOException

//只能在指定 IP 地址的网卡上进行监听

该类的常用方法是 public Socket accept()throws IOException 和 public void close()throws IOException

3、 Socket 类

该类的对象表示一个 Socket。客户机使用 Socket 类的构造函数，创建一个 Socket 对象，创建的同时会自动向服务器方发起连接。构造函数：

public Socket(String host,int port)throws UnknowHostException,IOException

public Socket(InetAddress ipaddress,int port)throws UnknowHostException,IOException

常用方法有：

close(),getInputStream() ,getOutputStream() ,getLocatPort(),getLocalAddress(),

getPort(),getInetAddress()

//

////////////////下面是客户机给服务器发信息的单向通信程序，分为客户机程序和服务器程序//////

```
//////////////////首先是客户机程序：
////////////////////////////////////////////////////////////////////////////////////////////
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;
public class Client
{
    public static void main(String args[]) throws IOException
    {
        Socket socket1=new Socket("127.0.0.1",2288); //向地址为127.0.0.1的服务器（即本机）的
                                                     //2288端口发起连接
        try
        {
            System.out.println("connection to server accepted:"+socket1); //连接成功，输出相关
```

```
信息
            PrintWriter out=new PrintWriter(new BufferedWriter(new
                    OutputStreamWriter(socket1.getOutputStream())),true); //获取对输出流（发送
数据）的控制
            //下面向服务器输出数据
                out.println("this is the data from client");  //向服务器端发送一串数据
        }
        finally
        {
            System.out.println("client closing socket");
            socket1.close();   //关闭socket
        }
    }
}


//////////////下面是服务器接收客户机发送过来的信息的程序
////////////////////////////////////////////////////////
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
public class Server
{
    public static void main(String args[]) throws IOException
    {
        ServerSocket s=new ServerSocket(2288); //服务器在2288端口进行监听
        System.out.println("serversocket:"+s);  //服务器在控制台输出自己的地址和端口信息
        try
        {
            Socket socket1=s.accept();   //等待客户端的连接
            try
            {
                System.out.println("connection to client accepted:"+socket1);//连接成功，输出相
关信息
                BufferedReader in=new BufferedReader(new
                        InputStreamReader(socket1.getInputStream())); //获取对输入流（即网络上
传来的数据流）
                                                                      //的控制
                //PrintWriter out=new PrintWriter(new BufferedWriter(new  //如果需要发送数据才
需要这条语句
                        //OutputStreamWriter(socket1.getOutputStream())),true);
```

```
                    String str=in.readLine();    //把收到的数据读取出来
                    System.out.println("服务器端接收到客户端的数据是："+str); //把收到的数据输出来
            }
            finally
            {
                System.out.println("server closing socket");
                socket1.close();  //关闭socket
            }
        }
        finally
        {
            s.close();
        }
    }
}
```

//

//////////下面这两个通信程序中，服务器收到客户机发来的信息后，给客户机回发一条信息，
//////////客户机在收到服务器反馈的信息后将其显示出来，然后终止。////////////////////////////////

```
////////首先是客户机程序如下：
////////////////////////////////////////////////////////////////////////////////////////
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;
public class Client
{
    public static void main(String args[]) throws IOException
    {
        Socket socket1=new Socket("127.0.0.1",2288);
        try
        {
            System.out.println("connection to server accepted:"+socket1);
            PrintWriter out=new PrintWriter(new BufferedWriter(new
                    OutputStreamWriter(socket1.getOutputStream())),true);
            //下面向服务器输出数据
                out.println("this is the data from client");
```

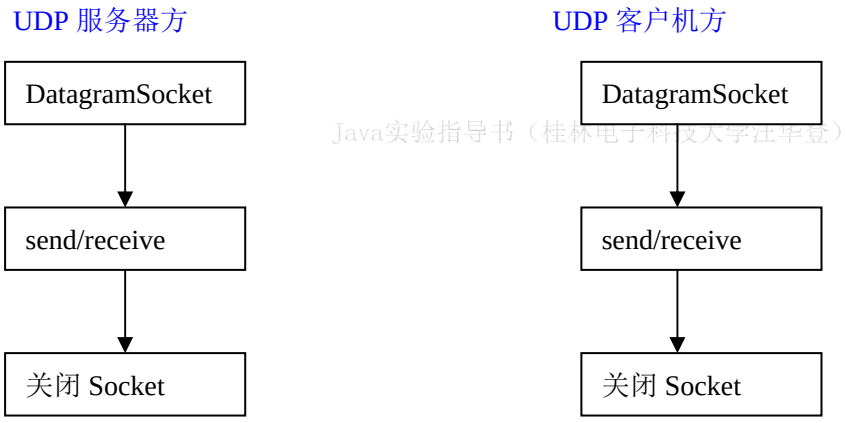
```
            //下面接收服务器反馈过来的数据
            BufferedReader in=new BufferedReader(new
InputStreamReader(socket1.getInputStream()));
            String str=in.readLine();
            System.out.println(str);
        }
        finally
        {
            System.out.println("client closing socket");
            socket1.close();
        }
    }
}


////////然后是服务器端的程序/////////////////////////////////////////////////////
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
public class Server
{
    public static void main(String args[]) throws IOException
    {
        ServerSocket s=new ServerSocket(2288);
        System.out.println("serversocket:"+s);
        try
        {
            Socket socket1=s.accept();
            try
            {
                System.out.println("connection to client accepted:"+socket1);
                BufferedReader in=new BufferedReader(new
                        InputStreamReader(socket1.getInputStream()));
                PrintWriter out=new PrintWriter(new BufferedWriter(new
                        OutputStreamWriter(socket1.getOutputStream())),true);
                    String str=in.readLine();
                    System.out.println("服务器端接收到客户端的数据是："+str);
                    out.println("这一行是服务器发送过来的数据，若你看到则表示收到");
            }
            finally
            {
```

```
                System.out.println("server closing socket");
                socket1.close();
            }

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

当 TCP 服务器需要同时对多个客户机进行响应和服务时，可以使用基于多线程的模型。服务器的主线程将只负责监听（创建 ServerSocket 对象），一旦有客户机发出连接请求，主线程将创建一个子线程，由该子线程负责该客户机的通信。主线程继续监听其它客户机的连接请求。

(二) UDP 网络程序设计

UDP 是面向无连接的不可靠的基于数据包的传输，也是典型的 C/S 模型。UDP 通信的模型如下：



1、 DatagramSocket 类

该类的对象代表了一个 UDP Socket，通过该 Socket 可以发送和接收 UDP 数据包。构造函数有：

public DatagramSocket()throws SocketException

public DatagramSocket(int port)throws SocketException

public DatagramSocket(int port,InetAddress ipaddress)throws SocketException　　//通常服务器使用

2、DatagramPacket 类

该类的对象代表了一个 UDP 数据包。通过 UDP 发送数据时，先要根据发送的数据生城一个 DatagramPacket 对象，然后通过 DatagramSocket 对象的 send()发送这个对象。接收时，先要根据要接收的缓冲区生成一个 DatagramPacket 对象，然后通过 DatagramSocket 对象的 receive()接收这个对象的数据内容。构造函数有：

public DatagramPacket(byte[] buf, int length)　//由接收缓冲区 buf 生成一个 DatagramPacket 对象，该构造函数//常用于构造一个接收用的数据包。

public DatagramPacket(byte[] buf, int length, InetAddress　ip, int　port)　//构造一个指定了接收方 IP 地址和//端口号的、发送用的数据包。

常用方法有：getData()，getLength()，　getOffset()，getPort()，setData()，setPort()等等

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////下面是两个 UDP 通信程序，一个是客户机，一个是服务器。UDP 服务器在 2288 端口//////////
////上接收任一客户机的 UDP 数据包。当客户机向服务器发送任意的字符串，服务器收到////////
////后，向客户机发回字符串字符的个数。 ////////////////////////////////////////////////////////////////////////////////

/////////////// 下面的是UDP通信中的服务器程序//////////////////////////
import java.io.*;
import java.net.*;
public class UdpServer
{
    public static void main(String args[])throws Exception    //为突出重点，不捕获异常
    {
        DatagramSocket ds=new DatagramSocket(2288);
        byte[] buf=new byte[1024];
        DatagramPacket rdp=null;
        DatagramPacket sdp=null;
        boolean flag=true;
        while(flag)
        {
            rdp=new DatagramPacket(buf,1024);  //创建一个用于接收数据的UDP数据包
            ds.receive(rdp);                   //等待一个客户机发送数据包
            InetAddress caddr=rdp.getAddress();   //获取客户机的IP地址
            int cport=rdp.getPort();              //获取客户机的端口号
            String s=new String(rdp.getData(),rdp.getOffset(),rdp.getLength());//获取客户机发送
的文本内容
            if(s.equals("end"))      //服务器端收到end之后，向客户端返回一个回答，然后结束
            {
                String rs=new String("ok,end!");
                byte[] sbuf=rs.getBytes();       //服务器将要发回的串
                sdp=new DatagramPacket(sbuf,sbuf.length,caddr,cport);//生成一个
                                    //发回客户机的UDP数据包的DatagramPacket对象
                ds.send(sdp);
                ds.close();
                break;
            }

            System.out.println("Client's IP is: "+caddr+" Client's Port is: "+cport+" Data from
Client is: "+s+"\r\n");
            String rs=new String("length of "+s+" is  "+s.length());
            byte[] sbuf=rs.getBytes();       //服务器将要发回的串
            sdp=new DatagramPacket(sbuf,sbuf.length,caddr,cport); //生成一个
                                    //发回客户机的UDP数据包的DatagramPacket对象
            ds.send(sdp);
        }
    }
```

```
}


//////////////////下面是客户机程序/////////////////////////////////////////////////
import java.io.*;
import java.net.*;
public class UdpClient
{
    public static void main(String args[])throws Exception  //为突出重点，不捕获异常
    {
        DatagramSocket ds=new DatagramSocket();  //生成一个客户机用的UDP Socket
        DatagramPacket sdp=null;    //发送用的UDP数据包
        DatagramPacket rdp=null;   //接收用的UDP数据包
        BufferedReader kbr=new BufferedReader(new InputStreamReader(System.in)); //键盘输入
        String s=null;
        byte[] rbuf=new byte[1024];   //接收缓冲区大小设置为1024
        byte[] sbuf=null;
        do
        {
            s=kbr.readLine();   //从键盘读入一行文本行
            sbuf=s.getBytes();   //将键盘输入放到字节缓冲区内
            sdp=new DatagramPacket(sbuf,sbuf.length,InetAddress.getByName("127.0.0.1"),2288);
//生成一个
                                Java实验指导书（桂林电子科技大学汪华登）
                                                                    //发送给服务器的UDP
数据包
            ds.send(sdp);
            rdp=new DatagramPacket(rbuf,1024);   //生成一个接收用的数据包
            ds.receive(rdp);        //等待并读取服务器的响应
            String data=new String(rdp.getData(),rdp.getOffset(),rdp.getLength());//获取服务器
发回的数据
            System.out.println("the response of UDP Server is:"+data+"\r\n");
        }while(!s.equals("end"));   //直到客户端用户输入“end”时才结束
        ds.close();         //关闭UDP Socket
    }
}
```

(三) 基于应用层 http 协议的程序（主要涉及到 **URL 类**）

```
////////////下面的程序的功能是获取某个 URL 地址的协议名、主机名、端口号和文件名//////////////////
import java.net.*;
public class useURL
{
  public static void main(String args[])
  {
    URL myurl=null;
```

```
    Try
    {
      myurl=new URL(http://www.guet.edu.cn);
    }
    catch(MalformedURLException e)
    {   System.out.println("MalformedURLException:"+e);   }
    System.out.println("URL String:"+myurl.toString()); //  获取 URL 对象并转换成字符串
    System.out.println("Protocol:"+myurl.getProtocol()); //获取协议名
    System.out.println("Host:"+myurl.getHost()); //获取主机名
    System.out.println("Port:"+myurl.getPort()); //获取端口号
    System.out.println("File:"+myurl.getFile()); //获取文件名
    //URL 类另有 equals(Object obj)和 openStream()等方法可用。另还有 URLConnection 类
    //也可用来访问网络资源，详见相关参考书。
  }
}
////////////////////////////////下面的程序可输出指定网页的源代码//////////////////////////////////////
import java.net.*;
import java.io.*;
public class nettest
{
 public static void main(String args[])throws Exception
 {
    URL url=new URL("http://www.guet.edu.cn");
    BufferedReader br=new BufferedReader(new InputStreamReader(url.openStream()));
    String s;
    while((s=br.readLine())!=null)
        System.out.println(s);
    br.close();
 }
```

(四) Java 的系统类中还定义了一些支持实现组播、广播等功能的实现的类，在有需要研究和使用时请参考其它资料。

参考文献

整理中……