《数据结构--Java 实现》

上机实验指导书

昆明理工大学管理线经济学院

2018 年 月

实验目的与要求

一、实验目的

上机实验是学习《数据结构及算法》必不可少的实践环节。我们知道，在《数据结构及算法》课程的学习中，我们将要学习各种各样的数据结构及各种算法，这些结构及算法比较抽象，难于理解，只有通过充分的大量的上机实践，才能真正掌握这些内容并提高我们的编程能力。

学习《数据结构及算法》除了课堂讲授以外，必须保证有足够的上机时间。因为学时所限，课程不能安排过多的统一上机实验，所以希望学生有效地利用课程上机实验的机会，提高自己的程序开发能力，为专业的学习打下一个良好的基础。为此，我们结合课堂讲授的内容和进度,安排了 16 个上机实验。各实验的训练重点在于基本数据结构的 ADT 及其应用，与教科书的各章具有紧密的对应关系。每实验相对来说，都是比较大的程序，为让学生在有限的时间内完成，大部份内容都已经给出了参考程序，要学生编写的只是一小部份，学生上机实验时，主要的精力应该用于联接和调试。通过本实验达到如下目的：

1．加深对课堂讲授内容的理解

课堂上要讲授计算机程序设计中常见的数据结构及其操作，一般比较抽象，理解起来也不是很容易。通过上机实践，就可以使这些抽象的内容变得容易理解。特别是通过 ADT 的应用实验，使同学不但理解了抽象的数据结构知识，而且知道怎样应用这些知识解决实际的问题，从而加深对这些知识的理解。

2．培养分析解决实际问题的能力

通常，实习题中的问题比平时的习题复杂得多，也更接近实际。实习着眼于原理与应用的结合，使学生学会如何把书上学到的知识运用于解决实际问题的过程中去，培养从事软件开发设计工作所必需的基本技能；另一方面，能使书上的知识变“活”，起到深化理解和灵活掌握教学内容的目的。

3．提高上机实践的动手能力

以往同学们在练习中或是上机遇到的多是一些较小的编程，较偏重于编写功能单一的“小”算法，而在本实验中的题目都是软件设计的综合训练，包括问题分析，总体结构设计，用户界面设计，程序设计基本技能和技巧，多人合作协调，以至一整套软件工程规范的训练和科学作风的培养。通过这些实验，能大提高学生们的动手实践能力。我们知道，机器是比任何教师都严厉而公正的评判者，在这位评判者的帮助下，我们的编程能力才会有突飞猛进。

二、实验要求

本实验课程由 13 个实验构成，每个实验原则上为两节课，如实验课上完不成，学生只有在课后做。本实验都是一些综合的实验，既有一定的难度，又有相当的工作量。为使学生能正常完成实验，我们在算法提示部份给出了相关的算法，在程序提示部份给出了有关的程

序，同时还以电子文档的形式给出了大部分的原代码及可执行的最终程序，学生要编写的只是少部分关键的代码。这样可以大大减少学生的工作量，增加实验涉及知识的复盖面。

每个实验由上机前的预习准备、上机调试运行和实验后的总结三个步骤组成。三个步骤都要求有书面记录，以实验报告的形式交老师批阅并给出评分。期未累计评分后作为最终的实验成绩。每个步骤的具体要求如下：

1．上机前的预习

认真阅读实验指导书中相应实验的目的要求，理解相关的数据结构及算法，读懂给出的提示程序代码。

根据实验的具体要求，进行分析，弄清算法，并编写出需要自己编写的那一部分程序。上机前一定要将有关程序事先编写好，再仔细检查程序直到找不到错误。分析可能遇到的问题及解决的对策。准备几组测试程序的数据和预期的正确结果，以便发现程序中可能存在的错误。上机实验时，应将主要精力用于调试和分析程序中的错误，而不是编写程序。

上机前没有充分的准备，到上机实验时才临时现编写程序，程序可能错误百出，调试时间就不够，无法完成实验内容，也就达不到实验设计预期的目的。

写出实验预习报告。

2．上机过程

事先准备好的源程序，在调试过程中，一般都不会一次通过，一定存在错误。对于语法错误或时连接错误，软件会给出错误提示信息，对同学来说这类错误的修改不算太困难。关键是算法中存在的逻辑错误，需要设计一定的测试数据，并通过跟踪调试来发现。使用预先准备的测试数据运行程序，观察是否得到预期的正确结果。若有问题，则仔细分析、调试，弄清错误的原因，并排除，直到得到正确结果。在调试过程中，要充分利用 Eclipse 集成开发环境提供的调试手段和工具，例如单步跟踪、设置断点、监视变量值的变化等。整个过程应自己独立完成。不要一点小问题就找老师，学会独立思考，勤于分析，通过自己实践得到的经验 用起来更加得心应手。同样在调试程序的过程中，请及时记录出错的现象及解决的方法。

3．实验后的总结

实验结束后，要进行检查验收。学生七人为小组，每组有一组长，组长的检查验收由教师直接进行，其他组员则由组长进行检查验收，将验收结果报教师登记。为方便检查，每个学生的所有实验都必需建立在以自己姓名加学号后两位命名的工作空间目录下（如李明，学号为 201010901180，则工作目录为李明 80），所有的文件名称及自定义标识符后也要加上学号的后两位。每次实验的程序也分别放至一个包中，如实验一的所有程序要放在命名为“实验一”的包中，实验二的程序放在命名为“实验二”的包中，依次类推。实验结束后每位同学都要将工作空间下的内容复制给教师检查并存档。

三、实验报告的格式要求

实验报告包括预习准备、上机过程记录及总结三部份，分别在上机前、上机过程中及上机结束后完成。格式要求如下：

一、预习准备

实验题目内容及要求。(可概括摘取本指导书中的内容)

用文字或流程图程序表示的算法说明，要求文字简练准确。

程序清单（只要给出自己编写的部份，本指导书上已有的不用重复）。

二、上机过程记录

对程序的调试和纠错过程中，请记录出现的**程序逻辑错误：出错现象、原因、解决方法。**

对于调试正确的程序，记录输入的原始数据、相应的运行结果和必要的说明。

三、实验总结

本次上机的收获，包括调试程序的心得与体会，存在的问题，分析程序未调试成功原因等。这部分内容要求简短扼要

实验报告批改时，主要看的是第二部份，因为只有这一部份最能反映实验的实际过程，并且各人都不一样。

实验一 热身练习---- 基本类型变量使用

一、实验目的

1、建立本课程实验的平台，熟悉 Eclipse 编程环境，编程方法。

2、复习 Java 课程中学过的基本类型变量的定义及使用。

3、掌握单步跟踪高度程序的方法。

二、实验内容

1、创建本课程实验所需的环境平台。

2、使用单步跟踪技术运行已给出的程序。

3、根据给出的算法编写并调试运行程序。

三、具体要求

电子文档中已给出了本课程实验所需的有关的工具类及为各实验提供的电子文档。请按下面要求，创建实验所需的平台，并完成本次实验：

1、建立自己的工作目录（Workspace）

在自己的U盘中建立一个以自己姓名加学号后两位命名的目录。

2、运行 Eclipse

将 Eclipse 系统解压到在 U 盘中，找到并双击 Eclipse，启动程序，运行 Eclipse 后将工作目录设置为在步骤 1 中建立的目录。

3、在欢迎屏幕点击 Workbench 图标进行工作台，创建名为 JDS 加学号后两位命名的 Java Project 项目。

4. 将电子文档中的目录“util”，“collection”及“exceptions”分别拖到上一步建立的项目中。

5. 在项目中创建“实验一”包，并在包中创建中创建并运行名为 TestVariable 加学号后两位的类，将电子文档中的 TextVariable.java 程序中 main()函数的代码剪贴到类中。调通并运行给出的参考程序；

6. 单步跟踪调试，观察 sa 在什么时候变成负数？

7. 分别在“实验一”包中按下面的给出的算法要求创建并编写调试运行类名分别为 TestA, TestB, TestC 加学号后两位的程序。

A.求出所有两位数的自守数。自守数是其平方后尾数等于该数自身的自然数。例如：25*25=625，76*76=5776。算法如下：

定义整型变量 x;

变量 x 从 10 到 99 循环做

如果该数的平方除 100 取余等于该数（(x*x)%100==x）

输出该数的自守数形式。

B. 求出 3 位正整数的全部水仙花数。所谓水仙花数是指三位整数的各位上的数字的立方和等于该整数本身。例如：153 就是一个水仙花数:$153 = 1^3 + 5^3 + 3^3$ 。算法如下：

定义整变量 i,j,k;

变量 i 从 1 到 9 循环做

变量 j 从 1 到 9 循环做

变量 k 从 1 到 9 做

如果 i,j,k 组成的数是水仙花数(i*100+j*10+k==i*i*i+j*j*j+k*k*k)

输出该数。

C．求一元二次方程 $ax^2+bx+c=0$ 的两个根。a，b，c 的值分别为 1，5，4 及 1，4，8。算法如下：

定义整型变量 a，b，c；并分别按要求赋；

定义浮点数变量 delta，将赋值为 b*b-4*a*c;

如果 delta 大于零,

定义两个浮点变量 r1，r1;

计算 r1=（float）(-b+ Math.sqrt(delta))/(2*a);

r2=(float) (-b-Math.sqrt(delta))/(2*a);

输出结果 r1 及 r2;

否则

定义两个浮点变量 r,im;分别表示实部和虚部

计算 r=-b/(a*a);

im=(float) Math.sqrt(-dalta)/(2*a);

输出一对共轭复根

四、有关程序提示

电子文档中的程序如下：

```
public class testVariable99 {
    public static void main(String[] args) {
        short sa;  //定义短整型变量sa
        int ia;    //定义整型变量ia
        float fa;  //定义浮点数变量fa
        char ca;   //定义字符变量ca
        //分别给变量赋值
        sa=100;
        ia=32760;
        fa=35.55f;
```

```
        ca='A';
        System.out.println("变量sa="+sa+" ia=" +ia+" fa="+ fa+" ca=" + ca);
        //改变变量的值并输出
        while(ia<32770){
            ia=ia+1;
            ca=(char)((int)ca+1);
            sa=(short)ia;
        }
        System.out.println("变量sa="+sa+" ia=" +ia+" fa="+ fa+" ca=" + ca);
    }
}
```

实验二 热身练习---- 引用变量的使用

一、实验目的

1、复习 java 课程中学过的引用变量（类变量）的定义，实例化及使用。

2、掌握基本图形界面类组件的使用。

3、熟悉布局管理器使用，并用于组织各种复杂的窗口界面。

二、实验内容

使用 Swing 组件创建各种窗口界面。

三、具体要求

1、认真读懂后面程序提示中给出的窗口范例程序代码。创建名为实验二的包，并在包中创建名为 testFrame 加学号后两位的类。将窗口范例程序代码（在电子文档中名为 TestFrame.java 中）剪贴到类的定义中，对程序进行相应修改后，调试并运行。

2、模仿范例窗口代码，编写如下窗口显示程序。

算法提示：在 main()函数中编写如下代码：

定义一窗口类（JFrame）变量，并实例化窗口对象；

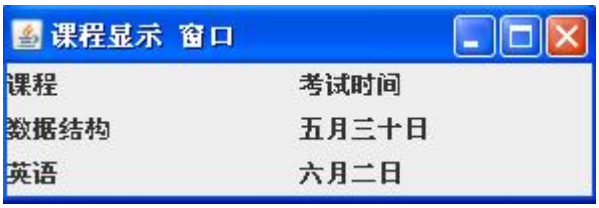
将窗口的布局管理器设置成流布局（FlowLayour）；

依次创建三个标签、三个文本编辑框，按上面窗口的要求加入到窗口中；

设置窗口大小；

让窗口可见。

3、模仿范例窗口代码，编写如下窗口显示程序。



算法提示：在 main()函数中编写如下代码：

定义一窗口类（JFrame）变量，并实例化窗口对象；

将窗口的布局管理器设置三行两列的网格布局（GridLayout）；

按图示界面的要求依次创建六个标签并加入到窗口中；

设置窗口大小；

让窗口可见。

4、模仿范例窗口代码，编写如下窗口显示程序。

算法提示：仿照范例窗口代码创建注册窗口类。该类实现 ActionListener 接口。

该类有如下五个属性变量：

标签（JLabel）类的标题（title）

文本编辑框（JTextField）的姓名（name）

密码编辑框（JPasswordField）类的密码（password）

按纽（JButton）类的注册及退出按纽变量

该类有三个方法：

main()方法：与范例窗口完全相同。

run()方法：算法如下：

按图示创建标题对象；

创建面板 p1，将标题加入到面板中；

创建“用户名”标签，及姓名编辑框对象；

创建面板 p2,将“用户名”标签及姓名编辑框对象加入到面板中；

同样创建“密码”标签、密码编辑框对象及面板 p3,将前两者加入后者中；

创建面板 p4，将布局管理器设成两行一列的网格布局；

将 p2， p3 加入到 p4 中。

创建“注册”、“退出”两个按纽及面板 p5，并将按纽加入面板中；

将面板 p1 加入窗口上边；

将面板 p4 加入窗口中央；

将面板 p5 加入窗口下边；

设置窗口适当大小；

设置窗口可视；

actionPerformed():算法如下：

如果点击的按纽是“注册”

如果用户名是“sys” （ name.getText().equals("sys")并且密码是“123”（password.getText().equals("123")）

将标显示的文本题改成“注册正确”

否则 将标显示的文本题改成“注册错误，请重注册！”

如果点击的按钮是“退出”,退出程序。

四、有关程序提示

电子文档中窗口范例程序如下：

```
public class TestFrame implements ActionListener {
    private JFrame frame;
    private JTextField name,age,id,from;
    private JTextArea message;
    private JButton OK, clear, exit;
    public static void main(String[] args) {
        TestFrame world = new TestFrame();//定义变量并将创建对象实例赋值给它
        world.run();                      //调用对象的方法run()
    }
    public void run(){
        //定义窗口变量并实例化，窗口缺省是边框布局的
        frame = new JFrame("实验二 窗口范例--杜华荣"); //创建窗口
        JLabel title = new JLabel("学生信息输入");    //定义标签变量并实例化
        JPanel p = new JPanel();                     //定义并实例化面板变量
        p.add(title);                                //将标签放到面板中
        frame.add(p, BorderLayout.NORTH);           //将面板放到窗口的上方
        JPanel p1 = new JPanel();                    //定义面板变量并实例化

        //将面板的布局管理设置为2行4列的网格布局
        p1.setLayout(new GridLayout(2,4));
        //创建输入界面中的各字段的标签及编辑框
        JLabel ln = new JLabel("姓  名");
        name = new JTextField();
        JLabel la = new JLabel("年  龄");
        age = new JTextField();
        JLabel lid = new JLabel("学  号");
        id = new JTextField();
        JLabel lf = new JLabel("籍  贯");
        from = new JTextField();

        //将标签及编辑框依次加入到网格布局的面板中
```

```
        p1.add(ln);
        p1.add(name);
        p1.add(la);
        p1.add(age);
        p1.add(lid);
        p1.add(id);
        p1.add(lf);
        p1.add(from);

        message = new JTextArea("\n");  //定义并实例化多行文本框
        JPanel p2 = new JPanel();       //定义并实例化另一面板
        //面板设置为垂直排列的合子布局
        p2.setLayout(new BoxLayout(p2, BoxLayout.Y_AXIS));
        p2.add(p1);                        //加入输入界面面板
        p2.add(message);                    //加入多行文本框
        //将文本框放窗口下边
        frame.add(p2, BorderLayout.CENTER);  //将面板放到窗口的中央

        JPanel p3 = new JPanel();          //定义并实例化另一面板变量
        OK = new JButton("确定");   //定义并实例化两个按钮变量
        clear = new JButton("清屏");
        exit = new JButton("退出");

        //对三个按钮注册监听程序以响应点击事件
        OK.addActionListener(this);
        clear.addActionListener(this);
        exit.addActionListener(this);
        p3.add(OK);      //将两个按钮加入到面板中，面板缺省是流布局的
        p3.add(clear);
        p3.add(exit);
        frame.add(p3,BorderLayout.SOUTH);  //将按钮面板放到窗口下边

        frame.pack();
        frame.setVisible(true);
    }
    public void actionPerformed(ActionEvent e) {
        JButton source = (JButton) e.getSource();//获取事件发生的按钮对象
        if(source==OK){ //是"确认"按钮,将输入信息显示到多行文本框中
            message.setText("输入的学生信息是:\n学生姓名:"+
                    name.getText()+"年龄:"+age.getText()+
                    "学号:"+id.getText()+"籍贯:"+from.getText());
            }
        if(source==clear){//是"清屏"按钮,清空所有文本框
            name.setText("");
```

```
            age.setText("");
            id.setText("");
            from.setText("");
            message.setText("");
        }
        if (source==exit){//是"退出"按钮,退出程序
            System.exit(0);
        }
    }
}
```

实验三 栈的 ADT 的实现

一、实验目的

1、熟悉栈的抽象数据结构。

2、掌握数组栈存储结构的进栈、出栈等操作的算法实现。

3、熟悉 Java 图形界面的编程,学会编写事件监听程序，按纽点击事件进行处理。

二、实验内容

1、实现三章中的栈（ArrayStack）类的所有方法

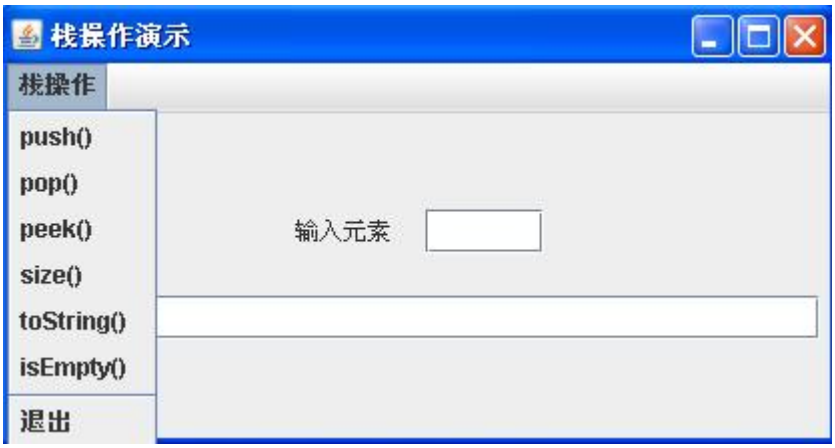
2、掌握图形界面编程中文本输入框、按纽、标签等对象的使用。

3、掌握按纽点击事件监听程序的编写。

三、具体要求

1、创建名为“实验三”的包，在该包中创建名为 ArrayStack 加学号后两位的类，将电子文档中提供的代码剪贴到类代码中,对代码进行相应修改,并加上未实现的方法代码。

2、读懂电子文档中的栈操作演示程序的代码，在包中创建名为 TestStack 加学号后两位的栈操作演示类，将电子文档中提供的代码剪贴到类代码中，对代码进行相应的修改，并调试程序。程序进行后的窗口图形界面类似下图：



在执行过程中，可以通过栈操作菜单调用栈的各种操作，并将结果显示出来。

四、有关程序提示

栈测试的图形窗口程序代码如下，请认真读懂：

```
public class testStack99 implements ActionListener{
    JFrame f;
    JTextField input,output;
    JMenuItem push,pop,peek,size,toString,isEmpty,exit;
    StackADT<String> stack;
    public static void main(String[] args) {
        testStack99 application = new testStack99();
        application.go();
    }
    public void go(){
        stack = new ArrayStack<String>();
        f=new JFrame("栈操作演示");
        //创建栈操作菜单,
        JMenuBar menubar =new JMenuBar();
        JMenu menu = new JMenu("栈操作");
        push= new JMenuItem("push()");
        push.addActionListener(this);
        pop = new JMenuItem("pop()");
        pop.addActionListener(this);
        peek = new JMenuItem("peek()");
        peek.addActionListener(this);
        size = new JMenuItem("size()");
        size.addActionListener(this);
        toString =new JMenuItem("toString()");
        toString.addActionListener(this);
        isEmpty = new JMenuItem("isEmpty()");
```

```
    isEmpty.addActionListener(this);
    exit = new JMenuItem("退出");
    exit.addActionListener(this);
    menu.add(push);
    menu.add(pop);
    menu.add(peek);
    menu.add(size);
    menu.add(toString);
    menu.add(isEmpty);
    menu.addSeparator();
    menu.add(exit);
    menubar.add(menu);
    f.setJMenuBar(menubar);
    //创建窗口屏幕上的对象
    f.setLayout(new GridLayout(4,1,10,10));
    Label lInput = new Label("输入元素");
    Label lOutput = new Label("输出内容");
    Label lSpace = new Label("");
    input = new JTextField(5);
    output= new JTextField(30);
    Panel pan1 = new Panel();
    Panel pan2 = new Panel();
    pan1.add(lInput);
    pan1.add(input);
    pan2.add(lOutput);
    pan2.add(output);
    f.add(lSpace);
    f.add(pan1);
    f.add(pan2);
    f.pack();
    f.setVisible(true);
}
public void actionPerformed(ActionEvent e){
     JMenuItem mi=(JMenuItem)e.getSource();
     if(mi==exit){                    //退出
        System.exit(0);
      }
      if(mi==push){
          if(input.getText()!=null && input.getText().length()>0){
              stack.push(input.getText());
              input.setText(null);
          }
      }
      if(mi==pop){
```

```
                try {
                 output.setText(stack.pop().toString());
             } catch (EmptyCollectionException e1) {
                 e1.printStackTrace();
             }
            }
            if(mi==peek){
                try {
                 output.setText(stack.peek().toString());
             } catch (EmptyCollectionException e1) {
                 e1.printStackTrace();
             }
            }
            if(mi==size){
                output.setText("size()="+stack.size());
            }
            if(mi==toString){
                output.setText(stack.toString());
            }
            if(mi==isEmpty){
                output.setText("isEmpty()="+stack.isEmpty());
            }
    }
}
```

实验四 栈的应用——表达式计算

一、实验目的

1、学会将栈类应用于解决表达式计算的问题。

2、熟悉四则运算表达式的运算处理，编写 calculator 类，实现四则运算表达式的处理。

3、学会字符串扫描识别的 StringTokenizer 类的使用方法。

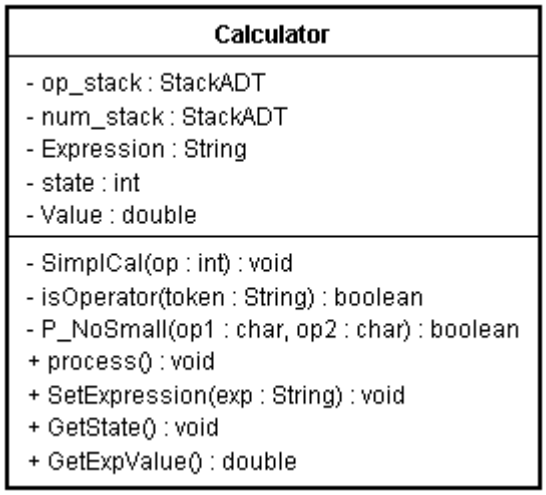
二、实验内容

1、借助栈类编写可进行四则运算表达式计算的 calculator 类。

2、编写窗口驱动程序，可进行输入任意四则运算表达式，并能调用 calculator 类进行运算。

三、具体要求

1、计算类（calculator）类的设计要求如下：



属性：

op_stack，num_stack分别是用于存放操作数（Double）和运算符(Characher)；Expression中用于存放运算表达式的字符串；state用于保存对象的状态，0表示未计算，1表示计算正确，计算如果在Value中，-1表示表达式错误，Value中的值无意义。

方法：

SimplCal(op) 将操作数栈中弹出两个运算数进行op指定的运算，运算结果再压入栈中；

isOperator(token) 用于判断token是否为运算符；

P_NoSmall(op1,op2) 用于判断运算符op1的优先级是否不小于op2；

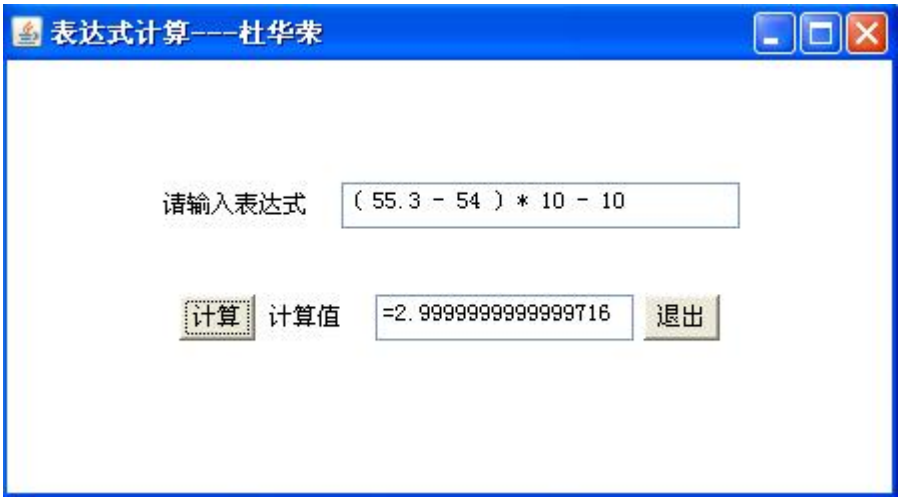
Process() 用于进行表达式的计算；

SetExpression(exp) 用于设置表达式字符串属性的值；

GetState() 用于返回对象的状态值；

GetExpValue() 用于获取表达式的计算值，如还未计算就调用process()先进行计算，如计算正确返回计算值，否则返回零。大部份代码在电子文档中只要将最核心的部份按后面所给的算法实现即可。

2、仿照实验三的窗口驱动程序，编写本实验的窗口驱动程序。窗口界面可设计成如下图所示的形式，在该窗口中输入计算表达式后，点击“计算”按纽时，在事件响应函数中调用 Calculate 类的方法 SetExpression()设置计算表达式，再调用 process()方法进行表达式的计算，最后调用 GetState()方法确定计算是否正确，如计算正确就将结果显示到文本编辑框中，实现任意四则运算表达式的计算，从而调试 Calculate 类。



四、算法思想

类的方法 process()算法如下：

对表达式进行扫描，依次对识别到的对象做：
{ 当对象是操作数：
　　将该操作数压入数据栈num_stack中；
　当对象是运算符op（+,-,*,/)：
　　当运算栈不空并且栈顶的运算级不小于op时循环做
　　{ 弹出运算栈顶的运算符p1;
　　弹出数据栈的两个运算数d1,d2；
　　将d2与d1做p1运算，运算结果压入数据栈；
　　}
　　将运算符op压入运算栈;
　当对象是‘(‘:
　　压入运算栈；
　当对象是‘)‘：
　　当运算栈不是‘(‘时，循环做
　　{ 弹出运算栈顶的运算符p1;
　　弹出数据栈的两个运算数d1,d2；
　　将d2与d1做p1运算，运算结果压入数据栈；
　　}
　　如果运算栈顶是‘(‘，弹出，则否报错；
}
如果运算栈不空，依次弹出栈顶运算符进行相应的运算，直到运算栈为空。
数据栈只有一个数据，该数据即为所求结果。

五、有关程序提示

1、计算类可参考如下程序：

```
public class Calculator {
   final private char  Plus ='+';
   final private char  Minus='-';
   final private char  Mult ='*';
   final private char  Div ='/';
   final private char  LP ='(';
   final private char  RP =')';

   private StackADT<Character> op_stack; //存操作符栈
   private StackADT<Double> num_stack; //存操作数栈
   private String Expression;      //存放输入的表达式
   int state;              //存放表达式计算的状态，0：未计算，1：
                           //计算正确，-1表达式错
   double Value=0;
   private void SimplCal(int op) throws EmptyCollectionException
   {
    double x1,x2;
     x2=num_stack.pop();
     x1=num_stack.pop();
     switch(op)
     { case Plus: num_stack.push(x1+x2);
               break;
       case Minus:num_stack.push(x1-x2);
               break;
       case Mult:num_stack.push(x1*x2);
               break;
       case Div: num_stack.push(x1/x2);
          break;
      }
   }
   private boolean isOperator(String token){
       return (token.equals("+")||token.equals("-")||
              token.equals("*")||token.equals("/")||
              token.equals("(")||token.equals(")"));
   }
   private boolean P_NoSmall(char op1, char op2)
   { if(((op1==Plus||op1==Minus)&&(op2==Plus||op2==Minus))|| //同为加减
      ((op1==Mult||op1==Div)&&(op2==Mult||op2==Div))||     //同为乘除
      ((op1==Mult||op1==Div)&&(op2==Plus||op2==Minus)))    //op1比op2高
       return true;
     else return false;
   }

   private void process()
```

```
   { String token;
     StringTokenizer tokenizer = new StringTokenizer(Expression);
     try{
     while (tokenizer.hasMoreElements())
     {  token = tokenizer.nextToken();
        if (isOperator(token))
           { …… //请根据前面给出算法编写
           }
      else {……  //请根据前面给出算法编写
            }
      }
        ……       //请根据前面给出算法编写
   } catch (EmptyCollectionException e)
   {    state=-1; // 表达式错
   }
}
   public Calculator(){
       op_stack = new ArrayStack<Character>();
       num_stack = new ArrayStack<Double>();
       state=0;
   }
   public void SetExpression(String exp){ //设置要计算的表达式
       Expression=exp;
       state=0;
   }
   public int GetState(){           //获取计算状态
       return state;
   }
   public double GetExpValue(){       //获取表达式的计算值
       if (state==0) process();        //如未计算就计算
       if (state==1) return Value;    //如计算正确就返回计算值
       else return 0;
   }
}
```

实验五 队列 ADT 的两种实现方式

一、实验目的

1、掌握队列抽象数据类型的特点，及操作。

2、掌握用数组方式及链表方式实现队列的抽象数据类型

3、进一步熟练使用图形界面，学会使用检查框组件进行编程。

二、实验内容

1、分别用数组及链表方式实现队列类的所有操作。

2、通过图形界面的方式，调用队列类的所有操作从而实现两种队列类的调试。

3、任选内容：使用实验二学会的方式比较队列的两种实现方式的快慢。

三、具体要求

1、将第五章中两种队列类的实现（LinkedQueue，CircularArrayQueue）操作补齐。

2、模仿实验三设计调试屏幕窗口的界面如下：

在执行过程中，如选择“数组实现”就用 CircularArrayQueue 类的对象演示，否则用 LinkedQueue 类的对象演示。通过队列操作菜单选项可以调用队列除迭代器外的所有操作方法，操作的返回值将显示在输出编辑框中。

四、有关程序提示

1、队列类的大部份程序如下。其余部份，请根据要求自己补齐。

```
public class LinkedQueue<T> implements QueueADT<T> {
    private int count;
    private LinearNode<T> front, rear;
    public LinkedQueue()
    {
    count = 0;
```

```
    front = rear = null;
    }
    public T dequeue() throws EmptyCollectionException {
        if (isEmpty())
            throw new EmptyCollectionException("queue");
        T result = front.getElement();
        front = front.getNext();
        count--;
        return result;
    }
    public void enqueue(T element)
    {
        LinearNode<T> node = new LinearNode<T>(element);
        if(isEmpty())
            front = node;
        else
            rear.setNext(node);
        rear = node;
        count++;
    }

    public T first() throws EmptyCollectionException {
     ... ...
    }
    public boolean isEmpty() {
     ... ...
    }
    public int size() {
      ... ...
    }
    public String toString(){
     ... ...
    }

public class CircularArrayQueue<T> implements QueueADT<T>{
    private final int DDFAULT_CAPACITY =100;
    private int front, rear, count;
    private T[] queue;
    public CircularArrayQueue(){
        front = rear = count = 0;
        queue = (T[])( new Object[DDFAULT_CAPACITY]);
    }
    public CircularArrayQueue(int initialCapacity){
        front = rear = count = 0;
```

```
        queue = ((T[])new Object[initialCapacity]);
    }
    public T dequeue() throws EmptyCollectionException {
        if (isEmpty())
            throw new EmptyCollectionException("queue");
        T result = queue[front];
        queue[front]=null;
        front = (front+1)%queue.length;
        count--;
        return result;
    }
    public void enqueue(T element) {
        if (size()==queue.length)
            expandCapacity();
        queue[rear] = element;
        rear = (rear+1)%queue.length;
        count++;
    }
    public void expandCapacity()
    {
    T[] larger = (T[])(new Object[queue.length*2]);
    for (int scan=0;scan<count; scan++)
    {  larger[scan] =queue[front];
       front = (front+1)% queue.length;
    }
    front = 0;
    rear = count;
    queue = larger;
    }
    public T first() throws EmptyCollectionException {
     ... ...
    }
    public boolean isEmpty() {
        ... ...
    }
    public int size() {
     ... ...
    }
    public String toString(){
     ... ...
    }
}
```

2、窗口程序可参考中如监听事件处理如参考如下进行编写：

```
    public void actionPerformed(ActionEvent e){
        QueueADT<String> queue;
        if(cbg.getSelectedCheckbox()==cbArray)
            queue=queue1;
        else queue=queue2;
        JMenuItem mi=(JMenuItem)e.getSource();
          if(mi==exit){                    //退出
             System.exit(0);
          }
          if(mi==enqueue){
              if(input.getText()!=null && input.getText().length()>0){
                  queue.enqueue(input.getText());
                  input.setText(null);
              }
          }
          if(mi==dequeue){
              try {
               output.setText(queue.dequeue().toString());
           } catch (EmptyCollectionException e1) {
               e1.printStackTrace();
           }
          }
          if(mi==first){
              try {
               output.setText(queue.first().toString());
           } catch (EmptyCollectionException e1) {
               e1.printStackTrace();
           }
          }
          if(mi==size){
              output.setText("size()="+queue.size());
          }
          if(mi==toString){
              output.setText(queue.toString());
          }
          if(mi==isEmpty){
              output.setText("isEmpty()="+queue.isEmpty());
          }
    }
}
```

实验六 队列的应用——毛毛虫游戏

一、实验目的

学会将队列类应用于解决实际应运问题——毛毛虫游戏。

学会在图形界面中使用键盘监听事件进行键盘输入处理。

二、实验内容

1、读懂并调通所给的毛毛虫程序。

2、按要求对程序进行改进。

三、具体要求

1、在电子文档中给出了毛毛虫游戏的代码，请认真仔细读懂程序的各组成部份，并将其拷贝张贴到所创建的实验六包中所创建的类中，并进行适当调整修改后，运行。

2、按下面的提示之一或全部对该程序进行修改完善：

a.增加一记分显示，虫向前移一步加一分，如虫不能移时结束游戏，同时分数每增加十分，等待时间减少一毫秒，即移动速度快一点。

算法提示：在类 Caterpillar 中加一整数的 public 属性 score，初始为 0；

修改 move()方法，当能移动时，移动一步并计分加一；

不能移动时退出程序。

为了使虫一开始就能移动，将 Caterpillar 的方向 direction 初始化为 'E'

在 paint()方法中加入记分显示。

在 run()方法中将睡眠时间从常量 100 改成 100-palyerOne.score/10

b.将游戏改贪吃虫，即在屏幕上随机出现一点食物，虫头碰到食物时身体长长一节。

算法提示：增加一个食物类，该类有一个属性表示食物的位置，三个方法，一个用于获取食物的位置，另一个用于另一个用于改变食物的位置，最后一个是 paint(Graphics)用于将食物指定位置画出来。修改虫类移动的算法，当虫可移动时，查看食物是否在新的虫头位置，如果是，则只在新的位置加一节虫身，并改变食物的位置，否则，同原来一样，加个新的虫头，删除老的虫尾。

c.屏幕增加一条不同颜色的虫，变成两条虫，新增加的虫由键盘的“j”“k”“l”“i”字母控制移动，由两人完，相互撕咬，咬着者得分，被咬者失分。

四、有关程序提示

下面是电子文档中给出的毛毛虫游戏的代码，请认真阅读。

```
public class CaterpillarGame extends Frame{
  public static void main(String []args){
      CaterpillarGame world = new CaterpillarGame();
      world.run();
  }
  public CaterpillarGame(){
      setSize((BoardWidth+2)*SegmentSize,
BoardHeight*SegmentSize+30);
      setTitle("实验六，毛毛虫--杜华荣");
      addKeyListener(new KeyReader());
      addWindowListener(new CloseQuit());
      setVisible(true);
  }

  private Caterpillar playerOne =
      new Caterpillar(Color.blue,new Point(20,10));
  final static int BoardWidth = 60;      //窗口宽以网格为单位
  final static int BoardHeight = 40;    //窗口高,以网格为单位
  final static int SegmentSize = 10;      //网格大小,以象素点为单位

  public void run(){
      while (true){
          movePieces();              //虫移动
          repaint();                 //重画桌面
          try{ Thread.sleep(100);} catch (Exception e){}//等待 100 毫
秒
      }
  }
  public void paint(Graphics g)
  { playerOne.paint(g); }          //画虫
  public void movePieces(){
```

```
            playerOne.move(this);
        }

        private class KeyReader extends KeyAdapter {
            public void keyPressed(KeyEvent e){
                char c = e.getKeyChar();
                switch(c){
                    case 'a': playerOne.setDirection('W');break;
                    case 'd': playerOne.setDirection('E');break;
                    case 'w': playerOne.setDirection('N');break;
                    case 's': playerOne.setDirection('S');break;
                }
            }
        }
        /**
         *
         * @author Huarong Du
         * 毛毛虫类
         */
        class Caterpillar{
            /**
             * 构造函数
             * @param c 虫的颜色
             * @param sp  虫尾所在点(网格为单位)
             */
            public Caterpillar(Color c, Point sp){
                color = c;
                for (int i=0;i<10;i++){   //虫的每一节在一个网格点,共 10 节
                    position = new Point(sp.x+i,sp.y);
                    body.enqueue(position);
                }
            }

            private Color color;
            private Point position;           //虫头所在位置
            private char direction ='W';      //虫移动的方向
            private QueueADT<Point> body = new LinkedQueue<Point>();//虫体
所在位置
            private QueueADT<Character> commands = new
LinkedQueue<Character>(); //键盘命令

            public void setDirection(char d)
            {commands.enqueue(new Character(d));}
```

```
public void move (CaterpillarGame game){
    // 先看是否要改变方向
    if (commands.size()>0){
        Character c;
        try {
            c = (Character) commands.dequeue();
            direction = c.charValue();
        } catch (EmptyCollectionException e) {
            e.printStackTrace();
        }
    }
    //再找到下一点的僧
    Point np =newPosition();
    //去掉尾部一节,在新的一点加下头部一节
    if (game.canMove(np)){
        try {
            body.dequeue();
            body.enqueue(np);
            position = np;}
        catch (EmptyCollectionException e) {
            e.printStackTrace();
        }
    }
}
/**
 * 根据虫头的位置及移动方向确定虫头下一点的应在哪一点
 * @return  下一点的位置
 */
private Point newPosition(){
    int x = position.x; //获取虫头位置
    int y = position.y;
    if(direction =='E') x++;    //根据移动方向确定下一点
    else if (direction =='W') x--;
    else if (direction =='N') y--;
    else if (direction == 'S') y++;
    return new Point(x,y);
}
/**
 * 将虫体画出来
 * @param g 画布对象
 */
public void paint(Graphics g){
    g.setColor(color);
    Iterator<Point> e = body.iterator();
```

```
            while(e.hasNext()){    //将虫体用圆形画出来
                Point p= e.next();
                g.fillOval(5+CaterpillarGame.SegmentSize*p.x,
                        10+CaterpillarGame.SegmentSize*p.y,
                        CaterpillarGame.SegmentSize,
CaterpillarGame.SegmentSize);
            }
        }
        /**
         * @param np  //测试的点
         * @return    //虫体是否在这点上在
         */
        public boolean inPosition (Point np){
            Iterator<Point> e = body.iterator();
            while (e.hasNext()){
                Point location = e.next();
                if(np.equals(location))return true;
            }
            return false;
        }

    }

    /**
     * 测试虫是否能移动
     * @param np 要移到的下一点
     * @return   能否移动
     */

    public boolean canMove (Point np){
          //测试是否到边界
        if ((np.x<=0)||(np.y<=0)) return false;
        if ((np.x >=BoardWidth)|| (np.y >=BoardHeight)) return false;
          //测试是否自己堵住自己
        if (playerOne.inPosition(np)) return false;
          // 可以移动
        return true;
    }
}
```

实验七 队列及栈应用——走迷宫

一、实验目的

1、进一步掌队列及栈类的使用。

3、学会用队列或栈解决探索求解（走迷宫）问题。

二、实验内容

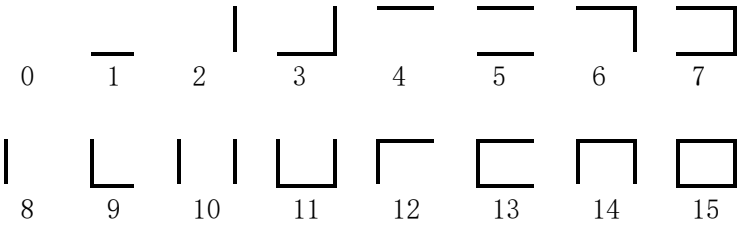
1、读懂本实验提供的程序，分别使用栈和队两种结构保存下一步可能的前进位置。进行实验。

2、自行设计迷宫的初始布局并运行。

3、改进程序，使其在有多条路经时标明最短路线。

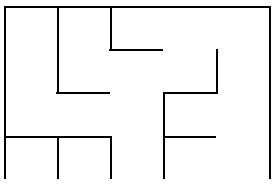
三、具体要求

1、本实验的走迷宫算法与第四章原理是一样，但由于是图形界面，所以迷宫的表示方式不一样。在本实验中，迷宫中的每一点有四个方向，每个方向可能是墙,也可能没有。共有十六种可能的组合，每种组合用一个整数表示。十六种表示如下：



这其实就是用四位二进制表示，第一位表示下边的墙，第二位表示右边的墙，第三位表示上面的墙，第四位表示左边的墙，0表示无，1表示有，故0000（十进制0）表示四面无墙，1111（十进制15）表示四面有墙，0101（十进制5）表示上下有墙，等等。按照这样的表示方法，要判断某个方向是否无墙时可用二进制按位与运算进行，如用walls[i][j] 表示迷宫的(i,j)点，那么walls[i][j] & 1)!=0就表示下方无墙，同理walls[i][j] & 2)!=0表示右面无墙，walls[i][j] & 4)!=0表示上面无墙，walls[i][j] & 8)!=0表示右边无墙。

下图所示的迷宫可用如下的二维数组表示： {{14,14,5,4,6,},
{10,9,4,3,10},



{9,5,2,13,2},
{14,14,10,12,2},
{9,1,1,3,11}}

电子文档中已给出了该程序的所有代码，请请读懂程序，并请自行设计自己的迷宫图。并运行之。

2、将下保存一步要试探的点的队列变成栈，观查有何变化

3、改进程序，使其在有多条路经时标明最短路线，在对每点标数时，不是简单的用一计数器，而是改成看周围连通的点除为 0 的外，最小值加一即可。注意起始点直接标为 1，

四、有关程序提示

下面是电子文档中给出的走迷宫程序的代码，请认真阅读。

```
public class Maze extends Frame{
    public static void main(String []args){
        Maze world = new Maze();
        world.setVisible(true);
        world.solveMaze();
    }
    public Maze (){
        setSize(350,330);
        setTitle("实验七　走迷宫　杜华荣");
        addWindowListener(new CloseQuit());
    }
    private int mazeWidth = 5;
    private int mazeHeight = 5;
    private int [][]walls ={{14,14,5,4,6,},{10,9,4,3,10},
        {9,5,2,13,2},{14,14,10,12,2},{9,1,1,3,11}};
    private int [][]visited = new int[5][5];
    private void solveMaze(){
        //用迷宫的起始点（右下角）初始队列
        QueueADT<Point> que = new LinkedQueue<Point>();
        que.enqueue(new Point(mazeWidth-1,mazeHeight-1));
        int visitCount =0;
        while (!que.isEmpty()){//开始搜索
            Point p;
            try {
                p = (Point) que.dequeue();
                if(visited[p.x][p.y]==0){//未搜索过的新点
```

```
                    visited[p.x][p.y]=++visitCount;
                    repaint();
                if((p.x==0)&&(p.y==0))
                    return;//搜索成功
                putNeighbors(p.x,p.y,que);
                try {Thread.sleep(200);} catch(Exception e){}
            }
            } catch (EmptyCollectionException e1) {
                e1.printStackTrace();
            }
        }
            System.err.println("no solution");
    }
    public void paint (Graphics g){
        int y = 50;
        for(int i=0;i<mazeHeight;i++){
            int x=50;
            for (int j = 0;j<mazeWidth;j++){
                if((walls[i][j] & 1)!=0) g.drawLine(x, y+50, x+50, y+50);
                if((walls[i][j] & 2)!=0) g.drawLine(x+50, y, x+50, y+50);
                if((walls[i][j] & 4)!=0) g.drawLine(x, y, x+50, y);
                if((walls[i][j] & 8)!=0) g.drawLine(x, y, x, y+50);
                if(visited[i][j]!= 0)
                    g.drawString(String.valueOf(visited[i][j]), x+5, y+30);
                x+=50;
            }
            y+=50;
        }
    }
    private void putNeighbors(int x,int y,QueueADT<Point> que){
        if((walls[x][y]&1) ==0) que.enqueue(new Point(x+1,y));
        if((walls[x][y]&2) ==0) que.enqueue(new Point(x,y+1));
        if((walls[x][y]&4) ==0) que.enqueue(new Point(x-1,y));
        if((walls[x][y]&8) ==0) que.enqueue(new Point(x,y-1));
    }
}
```

实验八 线性表（LIST）类的数组实现

一、实验目的

1、进一步熟悉带模板的类及接口。

2、进一步熟悉接口及类的继承方式。

3、掌握线性表类的数组实现。

4、学会用菜单类组件进行编程。

5、掌握 Action Event 及 Item Event 的监听程序的编程。

二、实验内容

按教科书第六章第四节用数组实现线性表的方式，实现 ArrayList 类、ArrayOrderedList 类及 ArrayUnorderedList 类。并用菜单驱动的方式实现一这三个类的调试程序，从而对这三个类进行调试。

三、具体要求

1、按照教科书第六章第四节给出的方法编写线性表数组实现的三个类:ArrayList 类、 ArrayOrderedList 类及 ArrayUnorderedList 类。ArrayList 类实现 ListADT 接口；ArrayOrderedList 类继承 ArrayList 类并实现 OrderedList 接口：而 ArrayUnorderedList 类继承 ArrayList 类并实现 UnorderedList 接口。 电子文档中给出了书中所给出方法的代码，可以直接剪贴到代码编辑窗口中。

2、读懂并熟悉所提供的用菜单驱动的线性表测试程序，并对所实现的线性表进调试。该测试程序的运行界面如下所示：

四、有关程序提示

线性表测试程序的代码如下：

```
public class testArrayList extends Frame implements ItemListener,
ActionListener {
    private static final int TARGET=1;
    private static final int INSERT=2;
```

```
    private static final int DELETE=3;
    private static final int DISPLAY=4;
    private TextArea theArea;
    private CheckboxMenuItem check1,check2;       //检查框菜单选项
    private MenuItem insert1,insert2,insert3,insert4;//增加操作的菜单选项
    private MenuItem remove1,remove2,remove3,exit;   //删除操作的菜间选项
    //显示项
    private MenuItem display1,display2,display3,display4,display5;
    private Button bOK1,bOK2;       //两种输入面板中的确定按纽
    private TextField tf1,tf2,tf3;  //两种输入面板中的文本编辑对象
    private ArrayOrderedList<Integer> OList;    //排序线性表
    private ArrayUnorderedList<Integer> UList;  //未排序线性表
    private ArrayList<Integer> List;            //线性表
    private int operation=0;   //1到4代表:insert1到4,5代表remove3,6代表
display5
    private Panel pen1,pen2;
     //构造函数 ,创建了窗口中的所有显示组件
     //注意创建的两个输入面板并未加入到窗口中
    public testArrayList(){
        this.setTitle("实验八  数组线性表测试---杜华荣");
        theArea = new TextArea(4,50);              //创建文本显示框
        this.add(theArea,BorderLayout.CENTER);     //加到窗口中
        MenuBar MBar = new MenuBar();              //创建菜单栏
        Menu Mtarget = buildMenu(TARGET);           //创建菜单
        Menu MInsert = buildMenu(INSERT);
        Menu MDelete = buildMenu(DELETE);
        Menu MDisplay = buildMenu(DISPLAY);
        MBar.add(Mtarget);                         //将菜单加到菜单栏中
        MBar.add(MInsert);
        MBar.add(MDelete);
        MBar.add(MDisplay);
        pen1=new Panel();                         //创建输入一个数的面板
        Label lb1=new Label("输入整数");           //下面为面板中的对象
        tf1=new TextField(10);
        bOK1 = new Button("确定");
        bOK1.addActionListener(this);              //注册监听程序
        pen1.add(lb1);                            //将对象加入面板中
        pen1.add(tf1);
        pen1.add(bOK1);
        pen2 = new Panel();                       //创建输入两个数的面板
        Label lb2=new Label("输入整数");
        tf2=new TextField(10);
        bOK2 = new Button("确定");
        bOK2.addActionListener(this);              //注册监听程序
```

```
        Label lb3= new Label("指定的整数");
        tf3= new TextField(10);
        pen2.add(lb2);
    pen2.add(tf2);
        pen2.add(lb3);
        pen2.add(tf3);
        pen2.add(bOK2);
        this.setMenuBar(MBar);                    //窗口中设置菜单栏,否则无菜
单
        OList = new ArrayOrderedList<Integer>();
        UList = new ArrayUnorderedList<Integer>();
        List=OList;
    }
    public static void main(String[] args){
    testArrayList world= new testArrayList();
    world.pack();
    world.addWindowListener(new CloseQuit());
    world.setVisible(true);
    }
    //创建菜单
    private Menu buildMenu(int menu){
    Menu result=null;
    switch(menu){
      case TARGET:  //选择线性表
        result= new Menu("操作");
            check1=new CheckboxMenuItem("排序线性表",true);
        check1.addItemListener(this);
        check2 = new CheckboxMenuItem("未排序线性表",false);
        check2.addItemListener(this);
            result.add(check1);
        result.add(check2);
        result.addSeparator();
        exit=new MenuItem("退出");
        exit.addActionListener(this);
        result.add(exit);
        break;
      case INSERT:  //插入
        result = new Menu("增加");
        insert1=new MenuItem("增加数据");
        insert1.addActionListener(this);
        insert2=new MenuItem("增加到表头");
        insert2.addActionListener(this);
        insert2.setEnabled(false);
        insert3=new MenuItem("增加到表尾");
```

```
    insert3.addActionListener(this);
    insert3.setEnabled(false);
    insert4=new MenuItem("增加到指定元素后");
    insert4.addActionListener(this);
    insert4.setEnabled(false);
    result.add(insert1);
    result.add(insert2);
    result.add(insert3);
    result.add(insert4);
        break;
  case DELETE:  //删除
      result = new Menu("删除");
      remove1 = new MenuItem("头节点");
      remove1.addActionListener(this);
      remove2 = new MenuItem("尾节点");
      remove2.addActionListener(this);
      remove3 = new MenuItem("指定元素节点");
      remove3.addActionListener(this);
      result.add(remove1);
      result.add(remove2);
      result.add(remove3);
      break;
       case DISPLAY: //显示
       result = new Menu("显示");
       display1=new MenuItem("线性表内容");
       display1.addActionListener(this);
       display2=new MenuItem("头节点");
       display2.addActionListener(this);
       display3=new MenuItem("尾节点");
       display3.addActionListener(this);
       display4=new MenuItem("节点数");
       display4.addActionListener(this);
       display5=new MenuItem("是否包函某元素");
       display5.addActionListener(this);
       result.add(display1);
       result.add(display2);
       result.add(display3);
       result.add(display4);
       result.add(display5);
}
    return result;
}
//Item 事件监听处理程序, 处理选择排序或未排序线性表事件
public void itemStateChanged(ItemEvent e) {
```

```
            CheckboxMenuItem cbm=(CheckboxMenuItem)e.getSource();
            if(cbm==check1)            //两种线性表同一时刻只能选一种
               check2.setState(!check1.getState());
            else if (cbm==check2);
               check1.setState(!check2.getState());
            if (check1.getState())
            { List=OList;     //设置选择排序线性表可进行的操作
              insert1.setEnabled(true);
              insert2.setEnabled(false);
              insert3.setEnabled(false);
              insert4.setEnabled(false);
           }else
            {  List=UList;     //设置选择排序线性表可进行的操作
               insert1.setEnabled(false);
               insert2.setEnabled(true);
               insert3.setEnabled(true);
               insert4.setEnabled(true);
            }
       }
       //Action事件监听处理程序,处理其它菜单项及按纽事件
       public void actionPerformed(ActionEvent e) {
           if(e.getSource() instanceof MenuItem) //点击菜单命令项
           { MenuItem mi=(MenuItem)e.getSource();
             if(mi==exit){                  //退出
               System.exit(0);
             }
             //需要使用一个整数输入面板的菜单项

if(mi==insert1||mi==insert2||mi==insert3||mi==remove3||mi==display5){
               this.removeAll();                       // 删除窗口中所有对象
               this.add(theArea,BorderLayout.CENTER); //重加入文本显示框
               this.add(pen1,BorderLayout.SOUTH);    //加入一个数的输入面板
              this.pack();
            }
            //用operation记录当前的操作,以便后续处理
            if(mi==insert1) operation=1;
            if(mi==insert2) operation=2;
            if(mi==insert3) operation=3;
            if(mi==remove3) operation=5;
            if(mi==display5) operation=6;
            //需要使用两个整数输入面板的菜单项
            if(mi==insert4){  //增加到指元素后
                operation=4;
                this.removeAll();                       // 删除窗口中所有对象
```

```
        this.add(theArea,BorderLayout.CENTER); //重加入文本显示框
        this.add(pen2,BorderLayout.SOUTH);    //加入两个数的输入面板
       this.pack();
    }
    if(mi==remove1){      //删除头节点
      if(!List.isEmpty()) List.removeFirst();
    }
    if(mi==remove2){      //删除尾节点
        if(!List.isEmpty()) List.removeLast();
    }
    if(mi==display1){     //显示线性表内容
      theArea.append(List.toString()+"\n");
    }
    if(mi==display2){     //显示头节点
        theArea.append("头节点为"+List.first().toString()+"\n");
    }
    if(mi==display3){     //显示尾节点
        theArea.append("尾节点为"+List.last().toString()+"\n");
    }
    if(mi==display4){    //显示线性表中节点数
        theArea.append("节点数为"+List.size()+"\n");
    }
}
if(e.getSource() instanceof Button){  //点击确定按钮事件
    Button bt=(Button) e.getSource();
    if(bt==bOK1){   //输入一个整数面板的确定按钮,先获取整数data
      Integer data=new Integer(Integer.parseInt(tf1.getText()));
       switch(operation)
       {
          case 1: // 增加
              OList.add(data);
              break;
          case 2: // 增加到表头
              UList.addToFront(data);
              break;
          case 3: // 增加到表尾
              UList.addToRear(data);
              break;
          case 5: // 删除指定元素节点
              try {
               List.remove(data);
               } catch (ElementNotFoundException e1) {
               e1.printStackTrace();
               }
```

```
                    break;
                case 6: // 显示是否包函某元素
                    if(List.contains(data))
                       theArea.append("线性表包函"+tf1.getText()+"\n");
                    else
                     theArea.append("线性表不包函"+tf1.getText()+"\n");
                }
            }
            if(bt==bOK2){     //输入两个整数面板的确定按纽
             Integer data1=new Integer(Integer.parseInt(tf2.getText()));
             Integer data2=new Integer(Integer.parseInt(tf3.getText()));
             UList.addAfter(data1, data2);
            }
            //处理后,删除输入面板
            this.removeAll();
            this.add(theArea,BorderLayout.CENTER);
            this.pack();
        }
    }
}
```

实验九 线性表（LIST）类的链表实现

一、实验目的

1、进一步熟悉带模板的类及接口。

2、进一步熟悉接口及类的继承方式。

3、掌握线性表类的链表组实现。

4、进一步熟悉用菜单类组件进行编程。

5、掌握 Action Event 及 Item Event 的监听程序的编程。

二、实验内容

按教科书第六章第五节用链表实现线性表的方式，实现 LinkedList 类、LinkedOrderedList 类及 LinkedUnorderedList 类。仿照上一实验的方式实现一这三个类的调试程序，从而对这三个类进行调试。

三、具体要求

1、按教科书的要求，实现实现 LinkedList 类、 LinkedOrderedList 类及 LinkedUnorderedList 类。ArrayList 类实现 ListADT 接口；LinkedOrderedList 类继承 LinkedList 类并实现 OrderedList 接口：而 LinkedUnorderedList 类继承 LinkedyList 类并实现 UnorderedList 接口。同样，电子文档中给出了书中所给出方法的代码，可以直接剪贴到代码编辑窗口中。

2、与实验八一样，编写用菜单驱动的线性表测试程序，并对所实现的线性表进调试。该测试程序的运行界面与实验八完全相同。

实验十 线性表应用—行星大战

一、实验目的

1、熟悉线性表类的使用。

2、应用线性表实现行星大战游戏。

3、掌握线性表类的链表组实现。

4、熟悉屏幕动画的实现。

5、掌握键盘事件监听程序的编程。

二、实验内容

本实验将实现如下面 UML 类图所示的四个类，从而模拟行星大战的游戏。本游戏中的各个类的简单介绍如下：

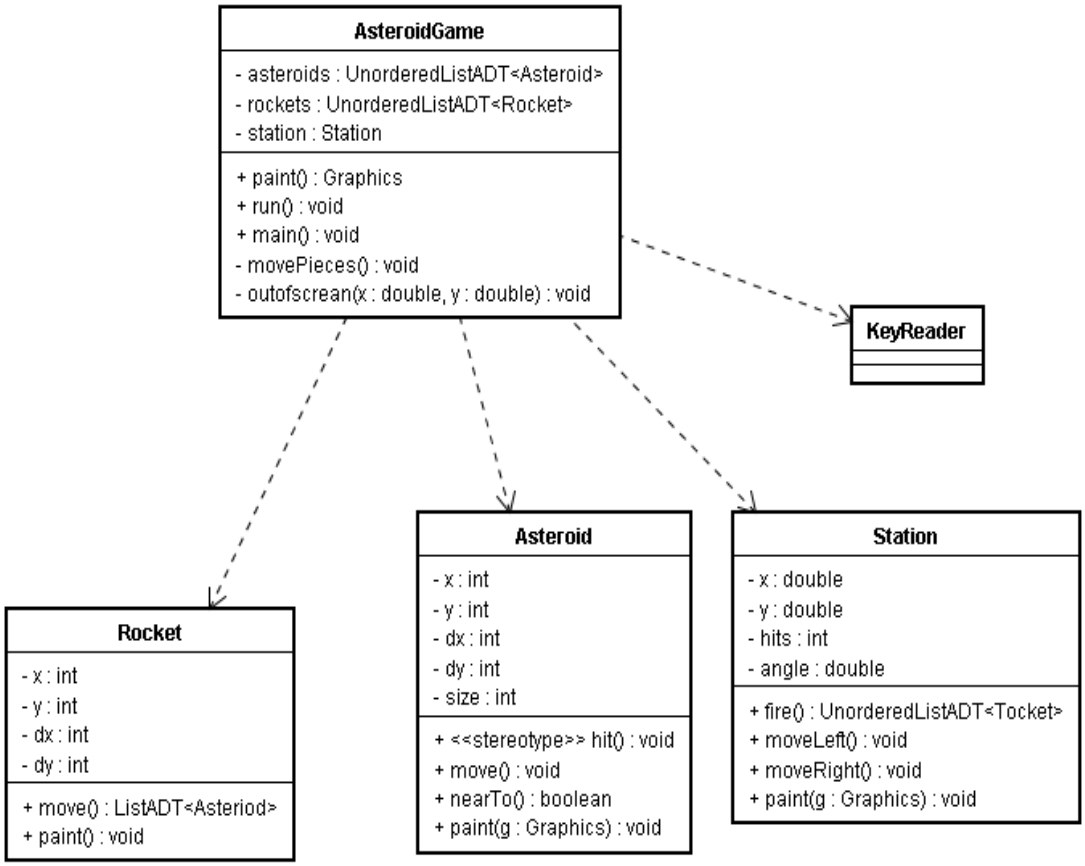
火箭类：有 x,y 坐标及移动量属性 dx,dy 属性，分别表示该火箭在屏幕上的位置及下一画格的相对移动量。其方法主要有使自己移动(move)及将自己在屏幕上画出来(paint)。

行星类：有 x,y 坐标及移动量属性 dx,dy 属性，分别表示该行星在屏幕上的位置及下一画格的相对移动量，此处还有行星的大小量属性 size。行星创建时其大小都是 20,被火箭击中是会变小。其方法主要有使自己移动(move)及将自己在屏幕上画出(paint),此外还有被击中和判断是否靠近指定点两个方法。被击中时，其大小减 4。而是否靠近指定点用来它是否被火箭击中或它是否击中炮站。

炮站类：有 x,y 坐标，被击中量 hits 及炮口的方向 angle 等属性，分别表示该炮站在屏幕上的位置，击中它的行星大小之和，火箭发射的方向。其方法请要有将自己在屏幕上画出（paint），炮口方向向左移(moveLeft)向右移(moveRight)，及发射火箭(fire)。

行星大战游戏类：有三个属性，前两个分别用未排序的线性表保存游戏中的行星和发出的火箭，最后一个属性是炮站，用来指向游戏中的炮站。其方法有，画出屏幕(paint)，它调用游戏中的各个对象的显示画出方法(paint)画也游戏屏幕；移动一格(movePieces)，它也是调用游戏中和个对象的移动方法将这些对象进行移动；游戏运行方法(run)，将循环进行移移动、显示、睡眠。而主函数(main)则创建游戏对象并调用对象的运行方法。该类还实现了键盘事件监听程序，处理键盘指令，实现对大炮方向进行左右移动，发射火箭及退出程序等操作。

请对照程序代码认真阅读程序，调度运行，并对程序加以修改。



三、具体要求

1、下面是所给的行星大战游戏的代码，请认真仔细读懂程序的各组成部份：

火箭类：

```
public class Rocket {
    public Rocket (double ix,double iy,double idx, double idy)
    { x = ix; y=iy; dx = idx; dy=idy;}
```

```
    public double x,y;
    private double dx,dy;

    public void move(ListADT<Asteroid> asteroids){
        x+=dx; y+=dy;
        Iterator<Asteroid> e = asteroids.iterator();
        while (e.hasNext()){
            Asteroid rock = (Asteroid) e.next();
            if (rock.nearTo(x,y))
                rock.hit();
        }
    }
    public void paint(Graphics g){
        g.setColor(Color.black);
        g.fillOval((int) x, (int) y, 5, 5);
    }
}
```

行星类：

```
    public void move(){ x+=dx;y+=dy;};
    public void paint(Graphics g){
        g.setColor(Color.black);
        g.drawOval((int) x, (int) y, size, size);
    }

    public void hit(){size = size -4;}
    /**
     * 判断行星是否靠近指定点
     * @param tx 指定点的x坐标
     * @param ty 指定点的y坐标
     * @return 是不靠近指定点
     */
    public boolean nearTo(double tx,double ty){
        // use Pythagorean theorem to determine distance between points
        double distance = Math.sqrt((x-tx)*(x-tx)+(y-ty)*(y-ty));
        return distance <size;
    }

}
```

炮站类：

```
public class Station {
    public Station( double ix,double iy){x=ix;y=iy;}

    private double angle = Math.PI/2.0;
```

```
    private int hits = 0;
    private final double x;
    private final double y;

    public void moveLeft(){angle = angle +0.1;}
    public void moveRight(){angle = angle -0.1;}
    //
    /**
     * 发射击火箭
     * @param rockets 保存射出火箭的线性表
     */
    public void fire (UnorderedListADT<Rocket> rockets){
        double cosAngle = Math.cos(angle);
        double sinAngle = Math.sin(angle);
          // rocket goes same direction as gun is pointing
        Rocket r= new Rocket(x +15*cosAngle, y-15 *sinAngle, 5*cosAngle,
-5*sinAngle);
        rockets.addToRear(r);
    }

    /**
     * 检查炮站是非被行星击中,并记录伤害程度
     * @param rock 袭击炮站的行星
     */
    public void checkHid(Asteroid rock){
        if (rock.nearTo(x, y))
        hits +=rock.size;
    }
    //画出炮站
    public void paint(Graphics g){
        g.setColor(Color.black);
        double lv = 20 *Math.sin(angle); //炮有 20个象素点长
        double lh = 20 * Math.cos(angle);
        g.drawLine((int)x, (int) y, (int)(x+lh),(int)( y-lv));
        g.drawString("hits:" +hits, (int)(x+10),(int)(y-5));
    }
}
```

游戏类：

```
public class AsteroidGame extends Frame{
    static public void main (String []args){
        AsteroidGame world = new AsteroidGame();
        world.setVisible(true);
        world.run();
    }
```

```
    public AsteroidGame(){
        setTitle("实验十  太空大战  杜华荣");
        setSize(FrameWidth,FrameHeight);
        addKeyListener(new KeyReader());
        addWindowListener(new CloseQuit());

    }

    private int FrameWidth = 500;
    private int FrameHeight = 400;
    private UnorderedListADT<Asteroid> asteroids = new
ArrayUnorderedList<Asteroid>();
    private UnorderedListADT<Rocket> rockets = new
ArrayUnorderedList<Rocket>();
    private Station station = new Station(FrameWidth/2, FrameHeight - 20);

    public void paint(Graphics g){
        station.paint(g);
        Iterator<Asteroid> e = asteroids.iterator();
        while (e.hasNext()){
            Asteroid rock = (Asteroid) e.next();
            rock.paint(g);
        }
        Iterator<Rocket> e1 =rockets.iterator();
        while (e1.hasNext()){
            Rocket rock =(Rocket) e1.next();
            rock.paint(g);
        }
    }
    /**
     * 键盘响应事件函数
     * @author Huarong Du
     *
     */
    private class KeyReader extends KeyAdapter {
        public void keyPressed(KeyEvent e){
            char key = e.getKeyChar();
            switch(key){
              case 'j': station.moveLeft();break;
              case 'k': station.moveRight();break;
              case ' ': station.fire(rockets);break;
              case 'q': System.exit(0);
            }
```

```
    }
}
/**
 * 画面移动一格
 */
private void movePieces(){
      //随机产生新的行星
    if(Math.random()<0.3){
        Asteroid newRock = new Asteroid(
                FrameWidth * Math.random(),20,
                10*Math.random()-5,3+3*Math.random());
        asteroids.addToRear(newRock);
    }
     // 移动画面上的所有物体
    Iterator<Asteroid> e = asteroids.iterator();
    while (e.hasNext()){ //对所有的行星进行移动
        Asteroid rock = (Asteroid) e.next();
        if(rock!=null){
            rock.move();
            if (outofscreen(rock.x,rock.y))
            try {   //删除移到屏幕外的行星
                asteroids.remove(rock);
            } catch (ElementNotFoundException e2) {
                e2.printStackTrace();
            } catch (EmptyCollectionException e2) {
                e2.printStackTrace();
            }
            station.checkHid(rock);
        }
    }
    Iterator<Rocket> e1 = rockets.iterator();
    while (e1.hasNext()){ //对所有火箭进行移动
        Rocket rock = (Rocket) e1.next();
        if(rock!=null){
            if (outofscreen(rock.x,rock.y))
                try {  //删除移到屏幕外的火箭
                    rockets.remove(rock);
                } catch (ElementNotFoundException e2) {
                    e2.printStackTrace();
                } catch (EmptyCollectionException e2) {
                    e2.printStackTrace();
                }
                rock.move(asteroids);
        }
```

```
        }
    }
    /**
     * 判断指定点是否在屏幕内
     * @param x 指定点的x坐标
     * @param y 指定点的y坐标
     * @return  该点是否在屏幕内
     */
    private boolean outofscreen(double x,double y){
        if (x<0||x>FrameWidth ||y<0 ||y> FrameHeight ) return true;
        return false;
    }
    /**
     * 程序运行
     */
    public void run(){
        while (true){ //now move pieces
            movePieces();
            repaint();
            try{
                Thread.sleep(100);
            } catch (Exception e){}
        }
    }

}
```

2、按上面的代码在实验十的包中分别创建各个类，并将电子文档所提供的代码剪贴到类代码中，进行适当修改后调试运行。

3、按下的提示之一或全部对该程序进行修改完善：

a.增加一记分显示，火箭击中行星时得一分，炮站被击中量达某个数值时停止移动，显示游戏结束。

b.修改行星或火箭的颜色或形状

c.让炮站可以用键盘控制移动。

实验十一 递归程序

一、实验目的

1、掌握递归程序的概念及实现方法。

2、应用递归的方法实现迷宫游戏及汉诺塔游戏

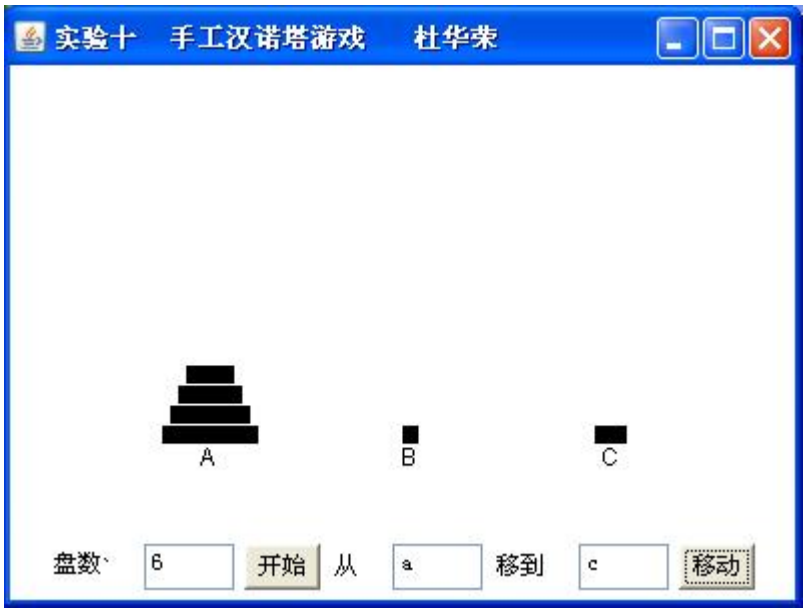
二、实验内容

1、将所给的手工汉诺塔游戏改成自动的方式。

2、利用递归的思想，仿照教科书第七章第三节中走迷宫的方法，将实验七中的迷宫程序改成递归的方式实现。

三、具体要求

1、在电子文档中已给出了将给出了手工汉诺塔游戏的代码，运行该程序将出现下图所示的手工玩游戏的界面：

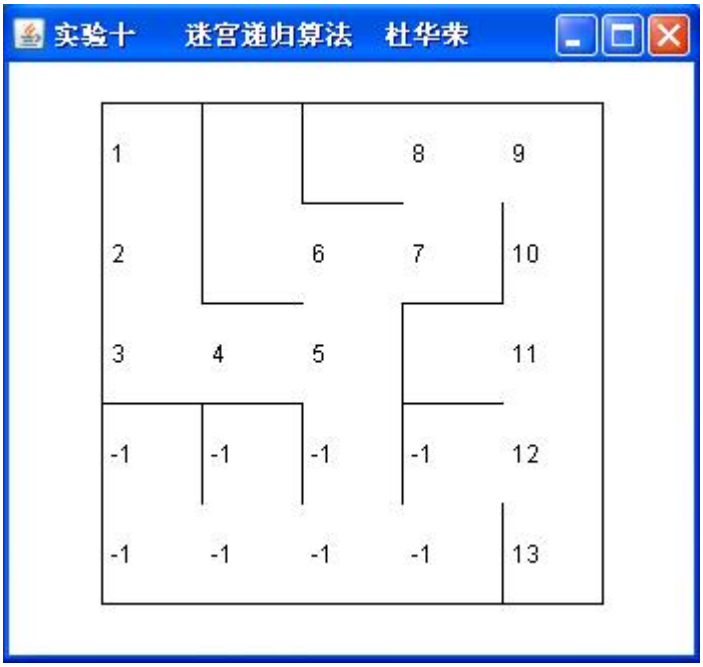


请在认真读懂程序的基础上，将手工移动操作到涉及的组件删除，修改移动按钮的事件响应函数，使点击按钮后自动将 A 柱上的所有盘按游戏规则要求移到 B 柱上，算法采用教科书第七章中的递归算法，即：如 n=1，直接移，否则先将 n-1 块盘移从 A 柱到 C 柱，再将剩下的最后一快盘移到 B 柱，最后将`n-1 块秀从 C 柱移到 B 柱即可。为了能看到移动的过程，请在每次调用移盘过程（moveDish）后让程序睡觉 300 毫秒（Thread.sleep(300)。

2、将实验六中的走迷宫程序改成递归方式。将原来的 soveMaze()方法加入两个参数，分别代表迷宫格的坐标，该方法就变成求出从该点到出口点（0，0）的路经并标注,返回值为该点到出口要走几步，如走不通返回-1。参照书上的思路，算法如下：

如果该点是（0，0），成功，visited[0][0]=1,返回 1;
否则 如果该点是有效点（坐标不出界，本点未访问过）
将该点的 visited 标为-1
对与该点联通的各个点分别调用本算法求到出口的步数，
如有一点走通，标该点的 visited 为走通点的步数加一，返回该值。
如都走不通通 返回-1;
如该点无效，返回-1

为了能看到标注的过程，请在每次调用标注后，更新窗口（repaint()）并让程序睡 300 毫秒（Thread.sleep(300)）。



四、有关程序提示

手工汉诺塔游戏的代码如下：

汉诺塔类：

```
public class Hanoi extends Panel{
    private int[] poleA,poleB,poleC;
    private String message="";
    public Hanoi(int n){
        poleA=new int[n+1];
        poleB=new int[n+1];
```

```
    poleC=new int[n+1];
    for(int i=0;i<=n;i++)
        poleA[i]=n-i;
}
public void paint(Graphics g){
    g.drawString(message,95,20);
    g.drawString("A", 95, 200);
    g.drawString("B", 195, 200);
    g.drawString("C", 295, 200);

    paintStack(poleA,g,100,180);
    paintStack(poleB,g,200,180);
    paintStack(poleC,g,300,180);
}

private void paintStack(int[] stk,Graphics g, int x, int y){
    int i=0;
    while (i<stk.length && stk[i]!=0){
        // each disk is 10 pixels high, 4*size wide
        g.fillRect(x-4*stk[i], y, 8*stk[i], 9);
        y = y-10;
        i++;
    }
}
/**
 * 移盘操作
 * @param a 源柱
 * @param b 目标柱
 * @return 1 正确,-1 盘符名称错 -2 无盘可移-3 大盘压小盘错
 */
public boolean moveDish(char a, char b){
    int i,j, pA[],pB[];
    switch(a)
    { case 'a':
      case 'A':
          pA=poleA;
          break;
      case 'b':
      case 'B':
          pA=poleB;
          break;
      case 'c':
      case 'C':
          pA=poleC;
```

```
            break;
        default:
            message="盘符名称不对";
            this.repaint();
           return false;
      }
      switch(b)
      { case 'a':
        case 'A':
            pB=poleA;
            break;
        case 'b':
        case 'B':
            pB=poleB;
            break;
        case 'c':
        case 'C':
            pB=poleC;
            break;
        default:
            message="盘符名称不对";
             this.repaint();
             return false;
      }
      for(i=0;i<pA.length&&pA[i]!=0;i++);
      for(j=0;j<pB.length&&pB[j]!=0;j++);
      if(i==0) {
      message="无盘可移";
          this.repaint();
          return false; //
      }
      if(j>0&& pA[i-1]>pB[j-1])
      {   message="大盘不能压小盘";
          this.repaint();
      return false;
      }
      else
      { pB[j]=pA[i-1];
        pA[i-1]=0;
        message="";
      }
      this.repaint();
      return true; //移盘正确
  }
```

```
}
```

游戏类：

```
public class hanoiGame extends Frame implements ActionListener{

    public static void main(String[] args) {
        hanoiGame world = new hanoiGame();
        world.setVisible(true);
    }
    Hanoi h;
    Button star,move;
    Panel pan;
    TextField tf1,source,target;
    public hanoiGame(){
        Label lb1=new Label("盘数`");
        Label lb2=new Label("从");
        Label lb3=new Label("移到");
        star = new Button("开始");
        star.addActionListener(this);
        move = new Button("移动");
        move.addActionListener(this);
        tf1 = new TextField(3);
        source = new TextField(3);
        target = new TextField(3);
        pan = new Panel();
        pan.add(lb1);
        pan.add(tf1);
        pan.add(star);
        pan.add(lb2);
        pan.add(source);
        pan.add(lb3);
        pan.add(target);
        pan.add(move);
        this.add(pan,BorderLayout.SOUTH);
        setSize(400,300);
        setTitle("实验十一  手工汉诺塔游戏   杜华荣");
        addWindowListener(new CloseQuit());
    }
    public void actionPerformed(ActionEvent e) {
        if(e.getSource() instanceof Button) //点击按钮命令
        { Button bt=(Button)e.getSource();
          if(bt==star&& tf1.getText()!=null){ //开始命令
              int n =Integer.parseInt(tf1.getText());
              h= new Hanoi(n);
```

```
            this.add(h, BorderLayout.CENTER);
            setSize(400,301);
         }
        if(bt==move && source.getText()!=null &&
target.getText()!=null){
            char c1=source.getText().charAt(0);
            char c2=target.getText().charAt(0);
            h.moveDish(c1, c2);
        }
      }
   }

}
```

实验十二 排序算法—简单排序

一、实验目的

1、掌握选择排序、插入排序及冒泡排序的算法。

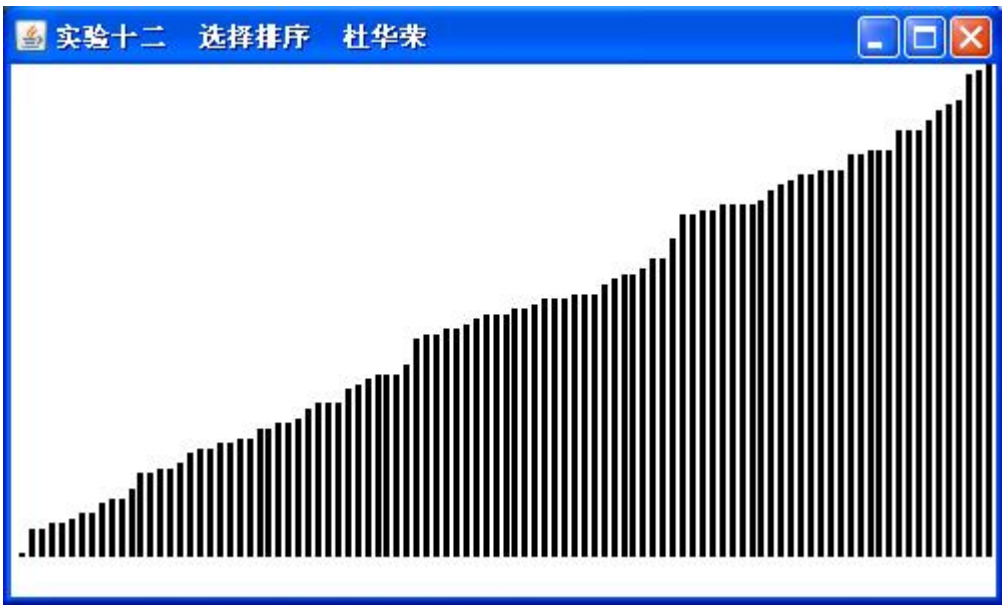
2、应用可视数组类动态显示各种排序过程。

二、实验内容

利用所给的可视化数组类，实现教科书第八章中学到的选择排序、插入排序及冒泡排序的算法的直观演示。

三、具体要求

1、为方便学生完成本实验，教师已经编写好一个可视化数组类，该类继承类，可将其所属性中整型数组中的数据直观显示出来。要更新屏幕时，只要调用该类的数据交换方法（swap）或显示（show）方法即可。同时为了让同学易于掌握该类的使用，同时已经提供了选择排序的演示程序，该程序运行后显示如下图所示，请仔细读懂程序，并运行该演示程序。



2、使用书上学到的插入排序及冒泡排序算法，模仿选择排序程序，分别实现这两种排序的演示程序

四、有关程序提示

可视排序演示代码如下（必须读懂）：

```
public class VisualSort extends Frame{
        public static void main(String[]args){
            VisualSort world = new VisualSort();
             world.selectionSort();

        }
        public VisualSort(){
            setTitle("实验十二 选择排序  杜华荣");
            setSize(500,300);
            add("Center",vv.getPanel());
            addWindowListener(new CloseQuit());
            setVisible(true);
            for (int i=0; i<numberOfElements;i++){
                int d = (int)(numberOfElements * Math.random());
                data[i]=(new Integer(d));
            }
        }
        private Integer[] data = new Integer[numberOfElements];
        private VisualArray vv = new VisualArray(data);
        private final static int numberOfElements = 100;
        public void selectionSort(){
```

```
            for(int index = 0; index<data.length-1;index++)
            {   int min =index;
                for (int scan=index+1;scan<data.length;scan++)
                    if(data[scan].compareTo(data[min])<0)
                        min = scan;
                vv.swap(min,index);
            }
        }

}
```

可视化类的代码如下（不要求读懂）：

```
public class  VisualArray<T> {
        /**
         * 可视化数组
         *
         * @param  欲显示的数组
         */
        public VisualArray (Integer[] data)
            { elementData = data; display = new VectorPanel(); }

        private Integer[] elementData;
        private Panel display;
        public void show(){
             pause(); display.repaint();
        }
        public void swap(int x, int y)
            { Integer temp=elementData[x];
              elementData[x]=elementData[y];
              elementData[y]=temp;;
              show();
             }
        public Panel getPanel () { return display; }
        private void pause () {
            try {
                Thread.sleep(50);
            } catch (Exception e) { }
        }
        private class VectorPanel extends Panel {
            public void paint (Graphics g) {
                int w = getSize().width;    //获取屏幕宽
                int h = getSize().height-20; //确定图的高
                int s = elementData.length;  //显示数据个数
                if (s == 0) return;
```

```
                int m = 0;
                for (int i = 0; i < s; i++) {   //找数据中最大数
                    Integer in = (Integer) elementData[i];
                    if (in.intValue() > m) m = in.intValue();
                }
                double hunit = 0;
                if (m != 0) hunit = h / (double) m;  //求单位高
                double wunit = w/ (double)s;        //求每个数据的宽
                int leftMargin = (int) (w - s*wunit)/2;  //求左边空白
                for (int i = 0; i < s; i++) {     //依次用直条显示每个数据
                    Integer in = (Integer) elementData[i];
                    int dh = (int) (in.intValue() * hunit);
                    g.fillRect((int) (leftMargin+ i * wunit),
                        (int) (h-dh), (int) (wunit-1), (int) dh);
                }
                super.paint(g);
            }
        }
    }
```

实验十三 排序算法—高级排序

一、实验目的

1、掌握快速排序、合并排序的算法。

2、进一步熟悉可视数组类的应用。

二、实验内容

利用所给的可视化数组类，实现教科书第八章中学到的快速排序、合并排序算法的直观演示。

三、具体要求

用上一实验同样的方法实现使用书上学到的快速排序及合并排序算法，进行这两种排序的演示程序