GUI编程

Swing和AWT 是java开发GUI常用的技术 . 但是由于外观不太美观, 组件数量偏少, 并且运行需要JRE环境(动不动就上百M的JRE包....), 所以没有流行起来. 但是 ,建议学还是需要简单的学习和了解的.

1. 组件(JTable,JList等)很多都是MVC的经典示范. 学习也可以了解mvc架构的
2. 工作时,也有可能遇见需要维护N年前awt/swing写的软件 ,虽然可能性很小
3. 可以写一些自己使用用的软件. 还是相当的方便.

艺多不压身

学习了swing还有必要学习awt吗?

swing是建立在awt基础上的。

还是有必要学习一下的.原因如下:

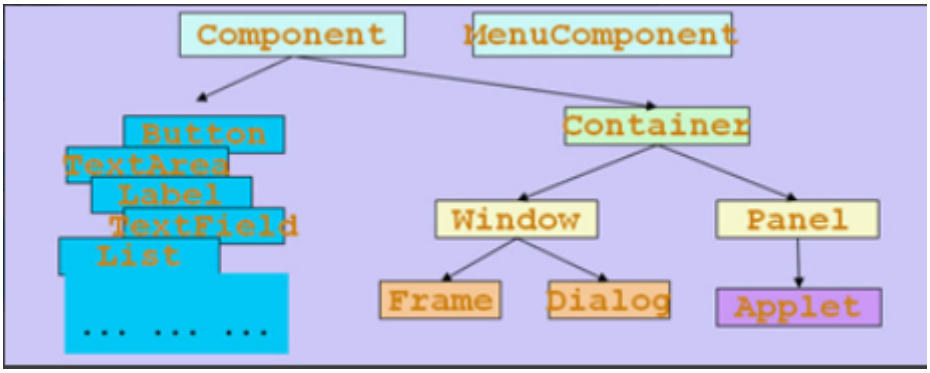
- :知识的关联性 . 比如 布局 , 颜色, 字体,事件机制 等....这些都是awt里的内容. 但在swing里也经常使用到.
- 学习成本低, 因为awt和swing在编码上区别不大, 写法基本一致, 组件使用上也差不多,(只需要记住少数有区别的地方就可以了)
- 使用场景存在不同. awt消耗资源少, 运行速度快. 适合嵌入式等. swing跨平台,组件丰富.

虽然现在用Java做cs的很少，但是对于我们学习Java基础来说，我觉得这个还是很好的资源，我们可以利用它把以前的所有知识贯穿起来，做一些小应用，游戏，等都可以，可以将自己的一些小想法，做成工具分享出来！

AWT

一、AWT介绍

- AWT（Abstract Window Toolkit）包括了很多类和接口，用于Java Application的GUI（Graphics User Interface 图形用户界面）编程。
- GUI的各种元素（如：窗口，按钮，文本框等）由Java类来实现。
- 使用AWT所涉及的类一般在Java.AWT包及其子包中。
- Container和Component是AWT中的两个核心类。



所有的可以显示出来的图形元素都称为Component，Component代表了所有的可见的图形元素，Component里面有一种比较特殊的图形元素叫Container，Container(容器)在图形界面里面是一种可以容纳其它Component元素的一种容器，Container本身也是一种Component的，Container里面也可以容纳别的Container。

Container里面又分为Window和Pannel，Window是可以独立显示出来的，平时我们看到的各种各样的应用程序的窗口都可以称为Window，Window作为一个应用程序窗口独立显示出来，Pannel也可以容纳其它的图形元素，但一般看不见Pannel，Pannel不能作为应用程序的独立窗口显示出来，Pannel要想显示出来就必须得把自己装入到Window里面才能显示出来。

Pannel应用比较典型的就是Applet(JAVA的页面小应用程序)，现在基本上已经不用了，AJAX和JAVASCRIPT完全取代了它的应用。

Window本身又可以分为Frame和Dialog，Frame就是我们平时看到的一般的窗口，而Dialog则是那些需要用户进行了某些操作(如点击某个下拉菜单的项)才出现的对话框，这种对话框就是Dialog。

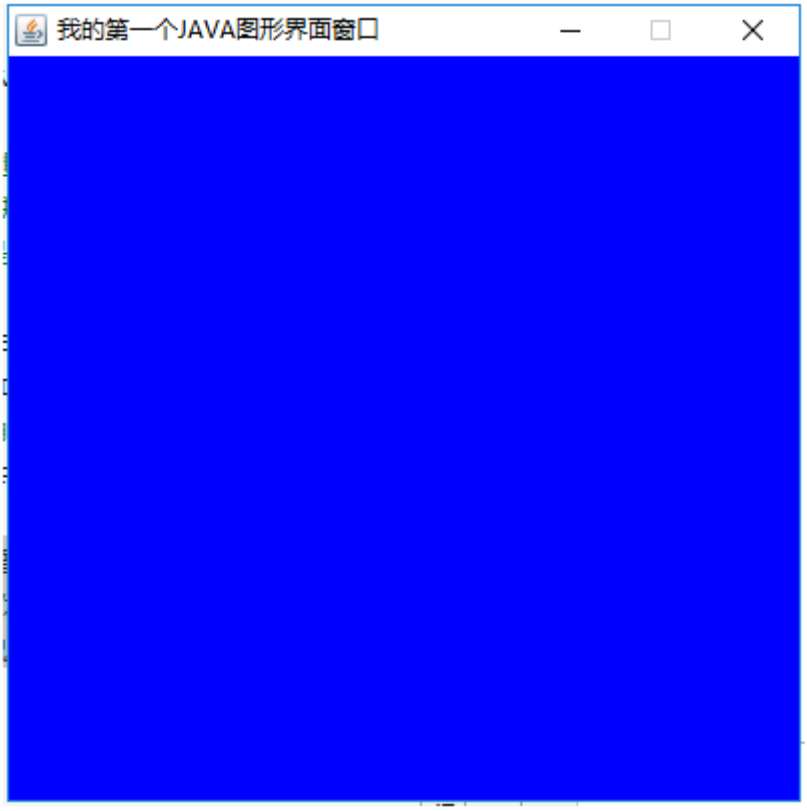
二、组件和容器(Component和Container)

2.1.Frame

【Frame范例】

```java
package com.kuang;

import java.awt.*;

//学习JAVA的GUI编程编写的第一个图形界面窗口
public class TestFrame {
    public static void main(String[] args) {

        //这里只是在内存里面创建了一个窗口对象 还不能真正显示出来然我们看到
        Frame frame = new Frame("我的第一个JAVA图形界面窗口");

        //设置窗体的背景颜色
        frame.setBackground(Color.blue);

        //设置窗体是否可见
        //要想看到在内存里面创建出来的窗口对象
        //必须调用setVisble()方法
        //并且把参数true传入才能看得见窗体
        //如果传入的参数是false
        //那么窗体也是看不见的
        frame.setVisible(true);

        //设置窗体的初始大小
        frame.setSize(400,400);

        //设置窗体出现时的位置，如果不设置则默认在左上角(0,0)位置显示
        frame.setLocation(200,200);

        // 设置窗体能否被改变大小
        // 设置为false后表示不能改变窗体的显示大小
        // 这里将窗体显示的大小设置为200X200
        // 那么窗体的显示只能是这个大小了，不能再使用鼠标拖大或者缩小
        frame.setResizable(false);
    }
}
```

运行结果：



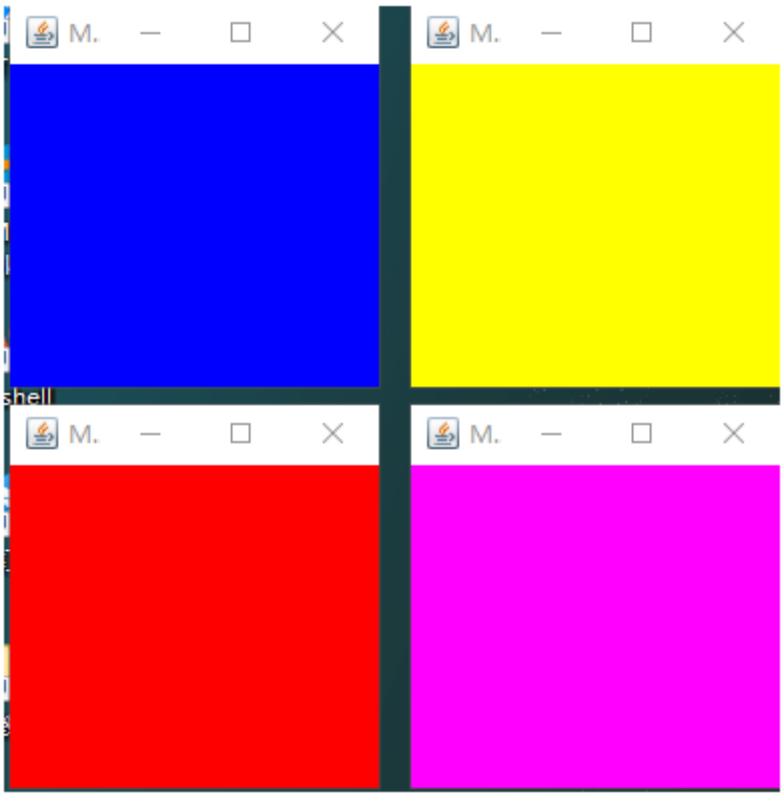
【发现问题：关闭不掉，解决方法：停止Java程序的运行】

【演示二：展示多个窗口】

```
package com.kuang;

import java.awt.*;

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

        MyFrame f1 = new MyFrame(100,100,200,200,Color.blue);
        MyFrame f2 = new MyFrame(300,100,200,200,Color.yellow);
        MyFrame f3 = new MyFrame(100,300,200,200,Color.red);
        MyFrame f4 = new MyFrame(300,300,200,200,Color.MAGENTA);

    }
}

//自定义一个类MyFrame，并且从Frame类继承
//这样MyFrame类就拥有了Frame类的一切属性和方法
//并且MyFrame类还可以自定义属性和方法
//因此使用从Frame类继承而来的自定义类来创建图形窗口比直接使用Frame类来创建图形窗口要灵活
//所以一般使用从Frame类继承而来的自定义类创建图形窗口界面比较好，
//不推荐直接使用Frame类来创建图形窗口界面
class MyFrame extends Frame{

    //定义一个静态成员变量id，用来记录创建出来的窗口的数目
    static int id = 0;

    //自定义构成方法，在构造方法体内使用super调用父类Frame的构造方法
    public MyFrame(int x,int y,int w,int h,Color color){
        super("MyFrame"+(++id));
        /*使用从父类Frame继承而来的方法设置窗体的相关属性*/
```

```
31          setBackground(color);
32          setLayout(null);
33          setBounds(x,y,w,h);
34          setVisible(true);
35      }
36
37  }
```

运行结果：



2.2.Panel

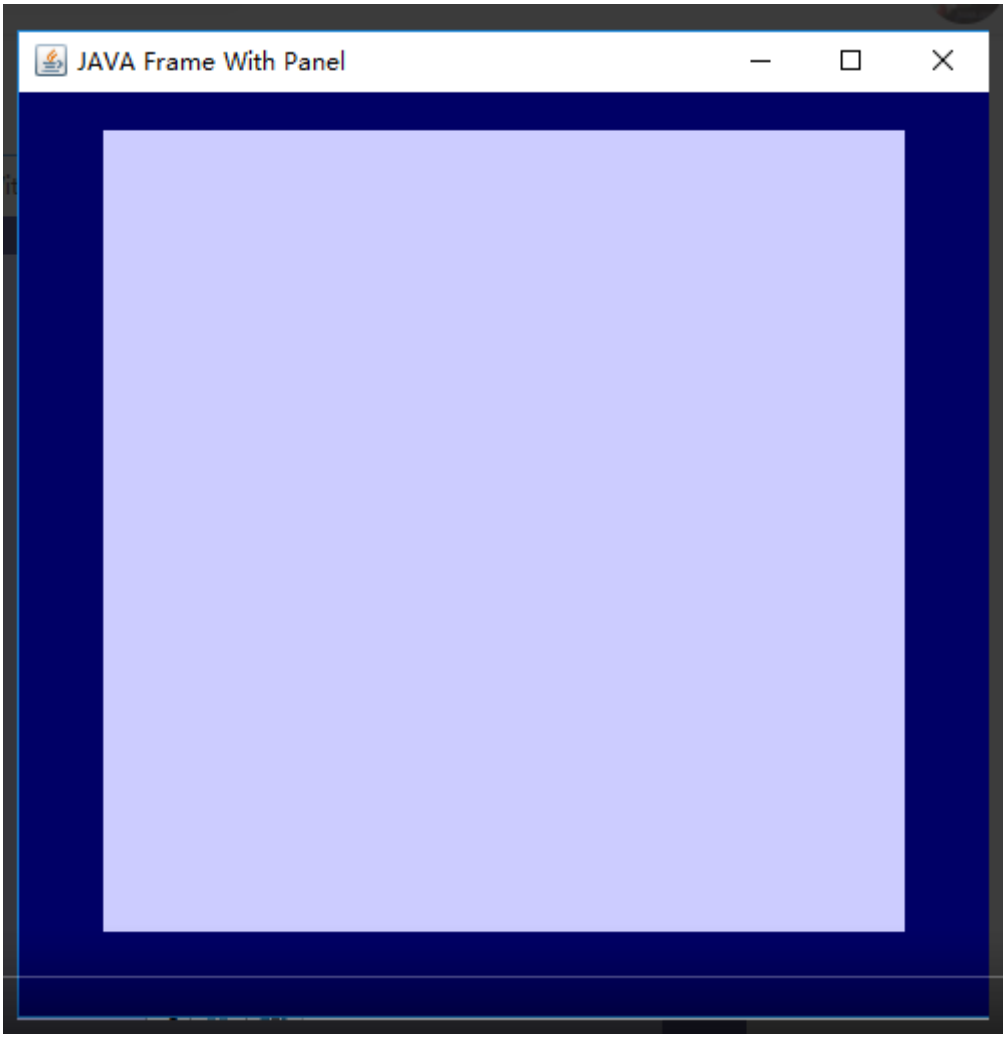
【演示】

```
1  package com.kuang;
```

```java

import java.awt.*;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;

public class TestPanel {
    public static void main(String[] args) {
        Frame frame = new Frame("JAVA Frame With Panel");
        Panel panel = new Panel(null);
        frame.setLayout(null);

        //这里设置的坐标(300,300)是相对于整个屏幕的
        frame.setBounds(300,300,500,500);

        //设置背景颜色时使用三基色(红，绿，蓝)的比例来调配背景色
        frame.setBackground(new Color(0,0,102));

        //这里设置的坐标(50,50)是相对于Frame窗体的
        panel.setBounds(50,50,400,400);
        panel.setBackground(new Color(204,204,255));

        //把Panel容器装入到Frame容器中，使其能在Frame窗口中显示出来
        frame.add(panel);

        frame.setVisible(true);

        //解决关闭问题
        frame.addWindowListener(new WindowListener() {
            @Override
            public void windowOpened(WindowEvent e) {

            }

            @Override
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }

            @Override
            public void windowClosed(WindowEvent e) {

            }

            @Override
            public void windowIconified(WindowEvent e) {

            }

            @Override
            public void windowDeiconified(WindowEvent e) {

            }

            @Override
            public void windowActivated(WindowEvent e) {

            }

```

```
60              @Override
61              public void windowDeactivated(WindowEvent e) {
62
63              }
64          });
65      }
66  }
```

结果如下：



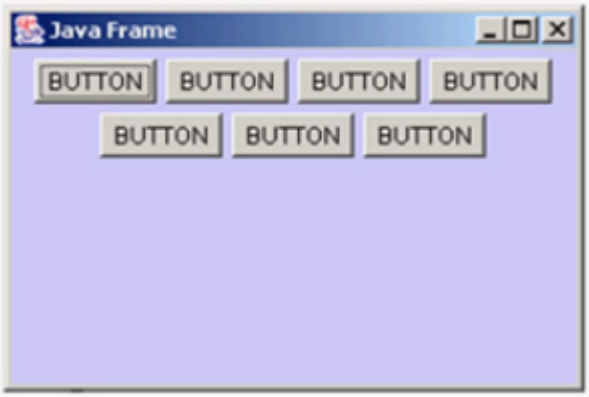
三、布局管理器

- Java语言中，提供了布局管理器类的对象可以管理
  - 管理Component在Container中的布局，不必直接设置Component位置和大小。
  - 每个Container都有一个布局管理器对象，当容器需要对某个组件进行定位或判断其大小尺寸时，就会调用其对应的布局管理器，调用Container的setLayout方法改变其布局管理器对象。
- Awt提供了5种布局管理器类：
  - FlowLayout
  - BorderLayout
  - GridLayou
  - CardLayout
  - GridBagLayout

3.1.第一种布局管理器——FlowLayout

FlowLayout布局管理器

- FlowLayout是Panel类的默认布局管理器。
  - FlowLayout布局管理器对组件逐行定位，行内从左到右，一行排满后换行。
  - 不改变组件的大小，按组件原有尺寸显示组件，可设置不 同的组件间距，行距以及对齐方式。
- FlowLayout布局管理器默认的对齐方式是居中。



FlowLayout 的构造方法

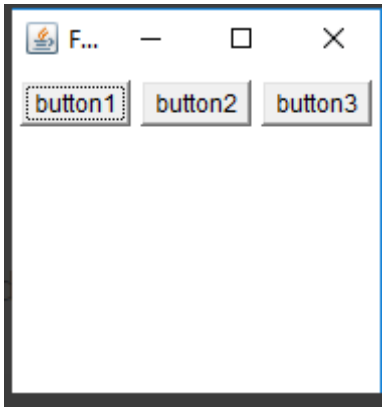
- new FlowLayout(FlowLayout.RIGHT,20,40);
  - 右对齐，组件之间水平间距20个像素，垂直间距40个像素。
- new FlowLayout(FlowLayout.LEFT);
  - 左对齐，水平和垂直间距为缺省值（5）。
- new FlowLayout();
  - 使用缺省的居中对齐方式，水平和垂直间距为缺省值（5）。

【演示】

```
1  package com.kuang;
2
3  import java.awt.*;
4
5  public class TestFlowLayout {
6      public static void main(String[] args) {
```

```
        Frame frame = new Frame("FlowLayout");

        //使用Button类创建按钮
        // 按钮类的其中一个构造方法：Button(String label) label为按钮显示的文本
        Button button1 = new Button("button1");
        Button button2 = new Button("button2");
        Button button3 = new Button("button3");

        // setLayout方法的定义：public void setLayout(LayoutManager mgr)
        // 使用流水(Flow)线般的布局
        frame.setLayout(new FlowLayout());
        // 使用了布局管理器FlowLayout，这里的布局采用默认的水平居中模式

        // frame.setLayout(new FlowLayout(FlowLayout.LEFT));
        // 这里在布局的时候使用了FlowLayout.LEFT常量，这样就将按钮设置为左对齐

        // frame.setLayout(new FlowLayout(FlowLayout.RIGHT));
        //这里在布局的时候使用了FlowLayout.RIGHT常量，这样就将按钮设置为右对齐


        frame.setSize(200,200);

        frame.add(button1); // 把创建出来的按钮放置到Frame窗体中
        frame.add(button2); // 这里并没有设置按钮的大小与位置
        frame.add(button3); // 设置按钮的大小与位置都是由布局管理器来做的

        frame.setVisible(true);

    }
}
```

运行结果：



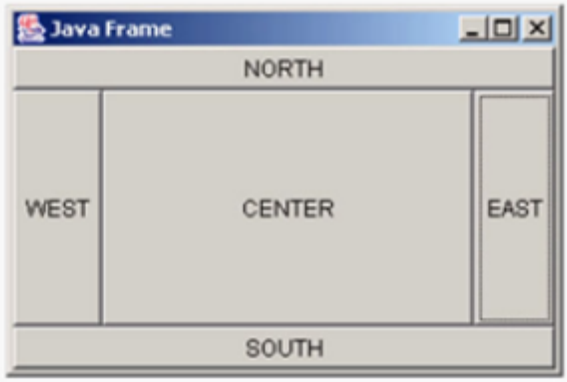
3.2.第二种布局管理器——BorderLayout

BorderLayout 布局管理器

- BorderLayout是Frame类的默认布局管理器。
- BorderLayout将整个容器的布局划分成
  - 东（EAST）
  - 西（WEST）
  - 南（SOUTH）
  - 北（NORTH）
  - 中（CENTER）五个区域，组件只能被添加到指定的区域。
- 如不指定组件的加入部位，则默认加入到CENTER区。
- 每个区域只能加入一个组件，如加入多个，则先前加入的会被覆盖。

BorderLayout 布局管理器

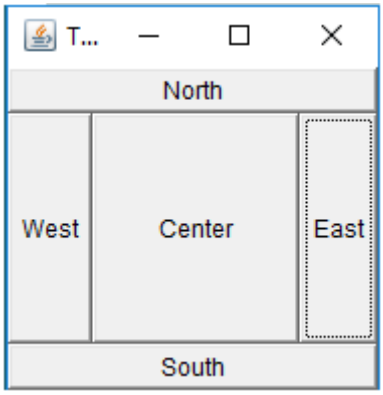
- BorderLayout型布局容器尺寸缩放原则：
  - 北、南两个区域在水平方向缩放。
  - 东、西两个区域在垂直方向缩放。
  - 中部可在两个方向上缩放。




```java
package com.kuang;

import java.awt.*;

public class TestBorderLayout {
    public static void main(String[] args) {
        Frame frame = new Frame("TestBorderLayout");

        Button buttonEast = new Button("East");
        Button buttonWest = new Button("West");
        Button buttonSouth = new Button("South");
        Button buttonNorth = new Button("North");
        Button buttonCenter = new Button("Center");

        //把按钮放置到Frame窗体时按照东西南北中五个方向排列好,推荐使用这种方式去排列窗体元素
        //这样容易检查出错误 因为这样写如果写错了编译器会提示出错

        frame.add(buttonEast,BorderLayout.EAST);
        frame.add(buttonWest,BorderLayout.WEST);
        frame.add(buttonSouth,BorderLayout.SOUTH);
        frame.add(buttonNorth,BorderLayout.NORTH);
```

```
22          frame.add(buttonCenter,BorderLayout.CENTER);
23
24          //也可以使用这样的方式排列按钮 在把按钮放置到Frame窗体时使用方向定位的字符串指定
    按钮的放置位置
25          //这种使用方向定位的字符串指定按钮的放置方式不推荐使用 一旦写错了方向字符串就不好
    检查出来
26          //因为即使是写错了仍然可以编译通过
27          /*
28          frame.add(buttonEast,"EAST");
29          frame.add(buttonWest,"West");
30          frame.add(buttonSouth,"South");
31          frame.add(buttonNorth,"North");
32          frame.add(buttonCenter,"Center");
33          */
34
35          frame.setSize(200,200);
36          frame.setVisible(true);
37
38      }
39  }
```

运行结果：

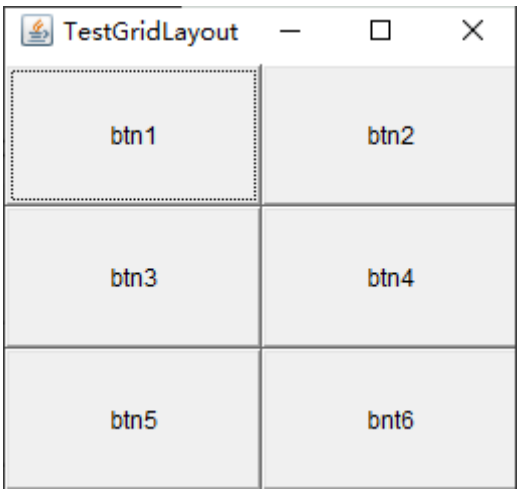


3.3.第三种布局管理器——GridLayout（表格布局管理器）

【演示】

```java
package com.kuang;

import java.awt.*;

public class TestGridLayout {
    public static void main(String[] args) {
        Frame frame = new Frame("TestGridLayout");

        Button btn1 = new Button("btn1");
        Button btn2 = new Button("btn2");
        Button btn3 = new Button("btn3");
        Button btn4 = new Button("btn4");
        Button btn5 = new Button("btn5");
        Button btn6 = new Button("bnt6");

        // 把布局划分成3行2列的表格布局形式
        frame.setLayout(new GridLayout(3,2));

        frame.add(btn1);
        frame.add(btn2);
        frame.add(btn3);
        frame.add(btn4);
        frame.add(btn5);
        frame.add(btn6);

        // Frame.pack()是JAVA语言的一个函数
        // 这个函数的作用就是根据窗口里面的布局及组件的preferredSize来确定frame的最佳大小。
        frame.pack();
        frame.setVisible(true);

    }
}
```

运行结果：



3.4.布局练习

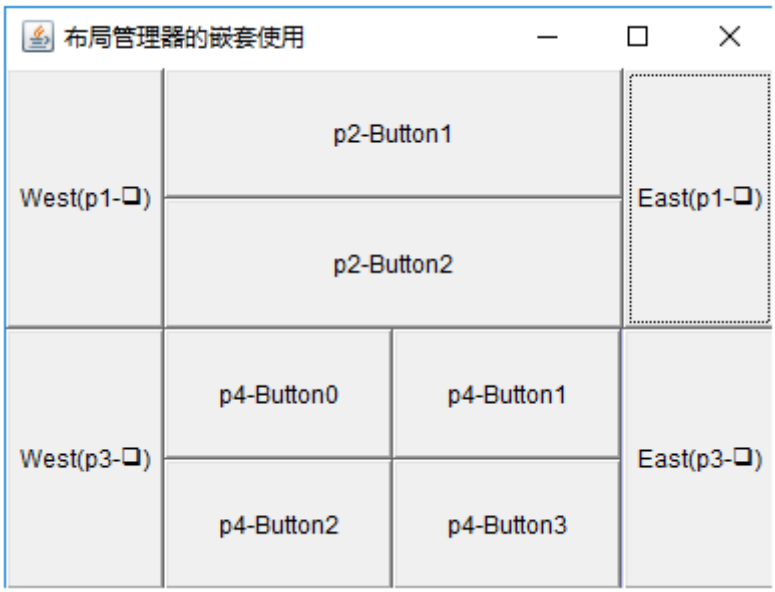
这几种布局管理器可以设置在Frame里面，也可以设置在Panel里面，而Panel本身也可以加入到Frame里面，因此通过Frame与Panel的嵌套就可以实现比较复杂的布局；

【演示】

```
package com.kuang;

import java.awt.*;

public class TestTenButtons {
    public static void main(String[] args) {
        //这里主要是对显示窗体进行设置
        Frame frame = new Frame("布局管理器的嵌套使用");

        //把整个窗体分成2行1列的表格布局
        frame.setLayout(new GridLayout(2,1));

        frame.setLocation(300,400);
        frame.setSize(400,300);
        frame.setVisible(true);
        frame.setBackground(new Color(204,204,255));

        //这里主要是对Panel进行布局的设置
        Panel p1 = new Panel(new BorderLayout());
        //p2使用2行1列的表格布局
        Panel p2 = new Panel(new GridLayout(2,1));
        Panel p3 = new Panel(new BorderLayout());
        //p4使用2行2列的表格布局
        Panel p4 = new Panel(new GridLayout(2,2));

        //这里主要是把按钮元素加入到Panel里面
        p1.add(new Button("East(p1-东)"),BorderLayout.EAST);
        p1.add(new Button("West(p1-西)"),BorderLayout.WEST);

        p2.add(new Button("p2-Button1"));
        p2.add(new Button("p2-Button2"));

        //p1里面嵌套p2，把p2里面的按钮作为p的中间部分装入到p1里面
        //把p2作为元素加入到p1里面
        p1.add(p2,BorderLayout.CENTER);

        p3.add(new Button("East(p3-东)"),BorderLayout.EAST);
        p3.add(new Button("West(p3-西)"),BorderLayout.WEST);
```

```
39
40          for(int i=0;i<4;i++){
41              p4.add(new Button("p4-Button"+i));
42          }
43
44          //p3里面嵌套p4，把p4里面的按钮作为p的中间部分装入到p3里面
45          p3.add(p4,BorderLayout.CENTER);
46
47          //把Panel装入Frame里面,以便于在Frame窗体中显示出来
48          frame.add(p1);
49          frame.add(p3);
50
51      }
52  }
```

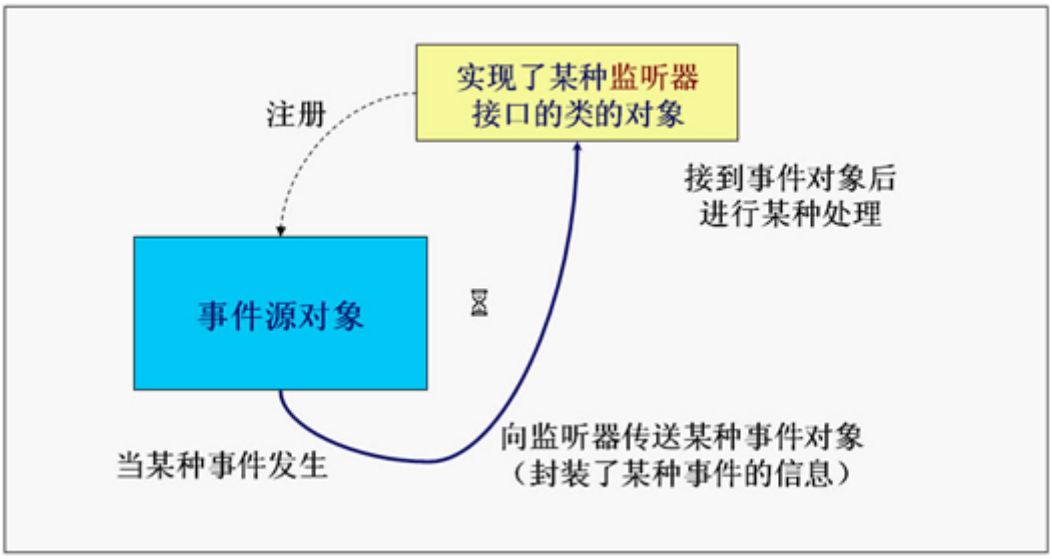
运行结果：



四、布局管理器总结

- Frame是一个顶级窗口，Frame的缺省布局管理器为BorderLayout
- Panel无法单独显示，必须添加到某个容器中。
  - Panel的缺省布局管理器为FlowLayout。
- 当把Panel作为一个组件添加到某个容器中后，该Panel仍然可以有自己的布局管理器。
- 使用布局管理器时，布局管理器负责各个组件的大小和位置，因此用户无法在这种情况下设置组件大小和位置属性，如果试图使用Java语言提供的setLocation()，setSize()，setBounds()等方法，则都会被布局管理器覆盖。
- 如果用户确实需要亲自设置组件大小或位置，则应取消该容器的布局管理器，方法为：
  - setLayout(null)

五、事件监听



【测试代码一】

```java
package com.kuang;

import java.awt.*;
import java.awt.event.*;

public class TestActionEvent {
    public static void main(String[] args) {
        Frame frame = new Frame("TestActionEvent");

        Button button = new Button("Press Me");
        // 创建一个监听对象
        MyActionListener listener = new MyActionListener();

        // 把监听加入到按钮里面，监听按钮的动作，
        // 当按钮触发打击事件时，就会返回一个监听对象e
        // 然后就会自动执行actionPerformed方法
        button.addActionListener(listener);

        frame.add(button, BorderLayout.CENTER);
        frame.pack();

        addWindowClosingEvent(frame);

        frame.setVisible(true);

    }

    //点击窗体上的关闭按钮关闭窗体

    private static void addWindowClosingEvent(Frame frame){
        frame.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }


```

```
}

// 自定义Monitor(监听)类实现事件监听接口ActionListener
// 一个类要想成为监听类，那么必须实现ActionListener接口
class MyActionListener implements ActionListener{

    //重写ActionListener接口里面的actionPerformed(ActionEvent e)方法
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("A Button has been Pressed");
    }
}
```

【测试代码二】

```
package com.kuang;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class TestActionEvent2 {
    public static void main(String[] args) {
        Frame frame = new Frame("TestActionEvent");
        Button btn1 = new Button("start");
        Button btn2 = new Button("stop");

        //创建监听对象
        MyMonitor monitor = new MyMonitor();

        //一个监听对象同时监听两个按钮的动作
        btn1.addActionListener(monitor);
        btn2.addActionListener(monitor);

        //设置btn2的执行单击命令后的返回信息
        btn2.setActionCommand("GameOver");

        frame.add(btn1,BorderLayout.NORTH);
        frame.add(btn2,BorderLayout.CENTER);

        frame.pack();
        frame.setVisible(true);

    }
}

class MyMonitor implements ActionListener{

    @Override
    public void actionPerformed(ActionEvent e) {
        //使用返回的监听对象e调用getActionCommand()方法获取两个按钮执行单击命令后的返回信息
        //根据返回信息的不同区分开当前操作的是哪一个按钮,btn1没有使用setActionCommand()方法设置
        //则btn1返回的信息就是按钮上显示的文本
        System.out.println("a button has been pressed,"+"the relative info is:\n"
                           +e.getActionCommand());
```

```
    }
}
```

六、TextField事件监听


```
package com.kuang;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class TestTextField {
    public static void main(String[] args) {
        new MyFrameTextField();
    }
}

class MyFrameTextField extends Frame{
    MyFrameTextField(){
        TextField textField = new TextField();
        add(textField);
        textField.addActionListener(new MyMonitor2());

        //这个setEchoChar()方法是设置文本框输入时显示的字符，这里设置为*，
        //这样输入任何内容就都以*显示出来，不过打印出来时依然可以看到输入的内容
        textField.setEchoChar('*');
        setVisible(true);
        pack();
    }
}

class MyMonitor2 implements ActionListener{

    //接口里面的所有方法都是public(公共的)
    //所以从API文档复制void actionPerformed(ActionEvent e)时 要在void前面加上
public
    @Override
    public void actionPerformed(ActionEvent e) {
        //事件的相关信息都封装在了对象e里面，通过对象e的相关方法就可以获取事件的相关信息

        //getSource()方法是拿到事件源，注意：拿到这个事件源的时候
        //是把它当作TextField的父类来对待
```

```
        //getSource()方法的定义是：“public Object getSource()”返回值是一个Object对象
        //所以要强制转换成TextField类型的对象
        //在一个类里面想访问另外一个类的事件源对象可以通过getSource()方法
        TextField textField = (TextField) e.getSource();

        // textField.getText()是取得文本框里面的内容
        System.out.println(textField.getText());

        // 把文本框里面的内容清空
        textField.setText("");
    }
}
```

【使用TextField类实现简单的计算器】

```
package com.kuang2;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class TestMath {
    public static void main(String[] args) {
        new Calculator();
    }
}

//这里主要是完成计算器元素的布局
class Calculator extends Frame{
    Calculator(){
        //创建3个文本框，并指定其初始大小分别为10个字符和15个字符的大小 这里使用的是TextField类的另外一种构造方法 public TextField(int columns)
        TextField num1 = new TextField(10);
        TextField num2 = new TextField(10);
        TextField num3 = new TextField(15);

        //创建等号按钮
        Button btnEqual = new Button("=");

        //给等号按钮加上监听，让点击按钮后有响应事件发生
        btnEqual.addActionListener(
            new MyMonitor(num1, num2, num3)
        );

        //“+”是一个静态文本，所以使用Label类创建一个静态文本对象
        Label lblPlus = new Label("+");

        //把Frame默认的BorderLayout布局改成FlowLayout布局
        setLayout(new FlowLayout());

        add(num1);
        add(lblPlus);
        add(num2);
        add(btnEqual);
        add(num3);
        pack();
        setVisible(true);
```

```
42
43      }
44  }
45
46  class MyMonitor implements ActionListener{
47      //为了使对按钮的监听能够对文本框也起作用
48      //所以在自定义类MyMonitor里面定义三个TextField类型的对象 num1,num2,num3,
49      //并且定义了MyMonitor类的一个构造方法 这个构造方法带有三个TextField类型的参数,
50      //用于接收 从TFFrame类里面传递过来的三个TextField类型的参数
51      //然后把接收到的三个TextField类型的参数赋值给在本类中声明的 三个TextField类型的参数
    num1,num2,num3
52      //然后再在actionPerformed()方法里面处理num1,num2,num3
53
54      TextField num1, num2, num3;
55
56      public MyMonitor(TextField num1, TextField num2, TextField num3) {
57          this.num1 = num1;
58          this.num2 = num2;
59          this.num3 = num3;
60      }
61
62      //事件的相关信息都封装在了对象e里面，通过对象e的相关方法就可以获取事件的相关信息
63      @Override
64      public void actionPerformed(ActionEvent e) {
65          // num对象调用getText()方法取得自己显示的文本字符串
66          int n1 = Integer.parseInt(num1.getText());
67          int n2 = Integer.parseInt(num2.getText());
68
69          //num3对象调用setText()方法设置自己的显示文本
70          //字符串与任意类型的数据使用“+”连接时得到的一定是字符串，
71          //这里使用一个空字符串与int类型的数连接，这样就可以直接把(n1+n2)得到的int类型的
    数隐式地转换成字符串了，
72          //这是一种把别的基础数据类型转换成字符串的一个小技巧。
73          //也可以使用“String.valueOf((n1+n2))”把(n1+n2)的和转换成字符串
74          num3.setText("" + (n1 + n2));
75          //num3.setText(String.valueOf((n1+n2)));
76
77
78          //计算结束后清空num1,num2文本框里面的内容
79          num1.setText("");
80          num2.setText("");
81      }
82  }
```

【JAVA里面的经典用法：在一个类里面持有另外一个类的引用】

```
1   package com.kuang2;
2
3   import java.awt.*;
4   import java.awt.event.ActionEvent;
5   import java.awt.event.ActionListener;
6
7   public class TestMath1 {
8
9       public static void main(String[] args) {
10          new Calculator2().launchFrame();
11      }
12
```

```
}

//做好计算器的窗体界面
class Calculator2 extends Frame {
    //把设计计算器窗体的代码封装成一个方法
    TextField num1, num2, num3;

    public void launchFrame() {
        num1 = new TextField(10);
        num2 = new TextField(10);
        num3 = new TextField(15);
        Label lblPlus = new Label("+");
        Button btnEqual = new Button("=");

        btnEqual.addActionListener(new MyMonitorbtnEqual(this));

        setLayout(new FlowLayout());
        add(num1);
        add(lblPlus);
        add(num2);
        add(btnEqual);
        add(num3);
        pack();
        setVisible(true);
    }

}

//这里通过取得Calculator2类的引用，然后使用这个引用去访问Calculator2类里面的成员变量
//这种做法比上一种直接去访问Calculator2类里面的成员变量要好得多
//因为现在不需要知道 Calculator2类里面有哪些成员变量了，
//现在要访问Calculator2类里面的成员变量，直接使用 Calculator2类对象的引用去访问即可
//这个Calculator2类的对象好比是一个大管家， 而我告诉大管家，我要访问Calculator2类里面的那些成员变量，
//大管家的引用就会去帮我找，不再需要我自己去找了。
//这种在一个类里面持有另一个类的引用的用法是一种非常典型的用法
//使用获取到的引用就可以在一个类里面访问另一个类的所有成员了

class MyMonitorbtnEqual implements ActionListener {
    Calculator2 calculator2 = null;

    public MyMonitorbtnEqual(Calculator2 calculator2) {
        this.calculator2 = calculator2;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        int n1 = Integer.parseInt(calculator2.num1.getText());
        int n2 = Integer.parseInt(calculator2.num2.getText());
        calculator2.num3.setText("" + (n1 + n2));
        calculator2.num1.setText("");
        calculator2.num2.setText("");
    }
}
```

结果：

七、内部类

- 好处：
  - 可以方便的访问包装类的成员
  - 可以更清楚的组织逻辑，防止不应该被其他类 访问的类 进行访问
- 何时使用：
  - 该类不允许或不需要其它类进行访问时

【内部类的使用范例】

```
package com.kuang2;

import java.awt.*;
import java.awt.event.*;

public class TestMath3 {

    public static void main(String args[]) {
        new MyMathFrame().launchFrame();
    }
}

class MyMathFrame extends Frame {
    TextField num1, num2, num3;

    public void launchFrame() {
        num1 = new TextField(10);
        num2 = new TextField(15);
        num3 = new TextField(15);
        Label lblPlus = new Label("+");
        Button btnEqual = new Button("=");
        btnEqual.addActionListener(new MyMonitor());
        setLayout(new FlowLayout());
        add(num1);
        add(lblPlus);
        add(num2);
        add(btnEqual);
        add(num3);
        pack();
        setVisible(true);
    }

    /*
     * 这个MyMonitor类是内部类，它在MyFrame类里面定义 MyFrame类称为MyMonitor类的包装类
     */
    /*
     * 使用内部类的好处：
     * 第一个巨大的好处就是可以畅通无阻地访问外部类(即内部类的包装类)的所有成员变量和方法
     * 如这里的在MyFrame类(外部类)定义的三个成员变量num1,num2,num3，
     * 在MyMonitor(内部类)里面就可以直接访问
```

```
     * 这相当于在创建外部类对象时内部类对象默认就拥有了一个外部类对象的引用
     */
    private class MyMonitor implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            int n1 = Integer.parseInt(num1.getText());
            int n2 = Integer.parseInt(num2.getText());
            num3.setText("" + (n1 + n2));
            num1.setText("");
            num2.setText("");
        }
    }
}
```

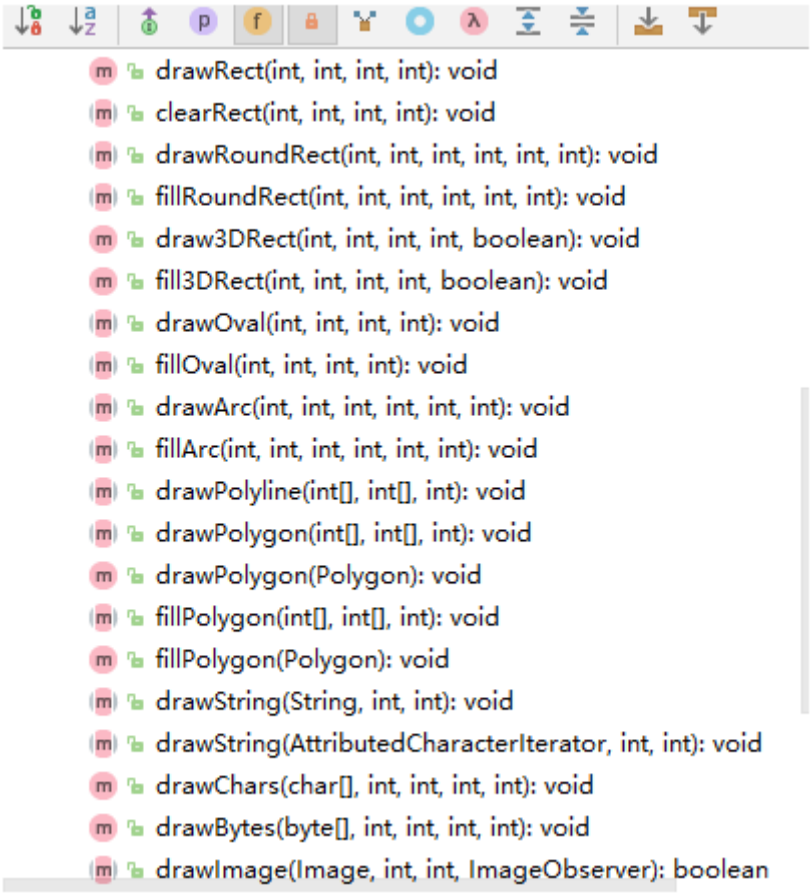
内部类带来的巨大好处是：

1. 可以很方便地访问外部类定义的成员变量和方法
2. 当某一个类不需要其他类访问的时候就把这个类声明为内部类。

八、Graphics 类

每个Component都有一个paint（Graphics g）用于实现绘图目的，每次重画该Component时都自动调用paint方法。

Graphics类中提供了许多绘图方法，如：



【测试代码】

```
package com.kuang3;

import java.awt.*;
```

```
4
5  public class TestPaint {
6      public static void main(String[] args) {
7          new MyPaint().launchFrame();
8          //在main()方法里面并没有显示调用paint(Graphics g)方法
9          //可是当创建出Frame窗体后却可以看到Frame窗体上画出了圆和矩形
10         //这是因为paint()方法是一个比较特殊的方法
11         //在创建Frame窗体时会自动隐式调用
12         //当我们把Frame窗体最小化又再次打开时
13         //又会再次调用paint()方法重新把圆和矩形在Frame窗体上画出来
14         //即每次需要重画Frame窗体的时候就会自动调用paint()方法
15     }
16 }
17
18 class MyPaint extends Frame{
19     public void launchFrame(){
20         setBounds(200,200,640,480);
21         setVisible(true);
22     }
23
24     public void paint(Graphics g){
25         //paint(Graphics g)方法有一个Graphics类型的参数g
26         //我们可以把这个g当作是一个画家，这个画家手里拿着一只画笔
27         //我们通过设置画笔的颜色与形状来画出我们想要的各种各样的图像
28
29         /*设置画笔的颜色*/
30         g.setColor(Color.red);
31         g.fillOval(100,100,100,100);/*画一个实心椭圆*/
32         g.setColor(Color.green);
33         g.fillRect(150,200,200,200);/*画一个实心矩形*/
34
35         //这下面的两行代码是为了写程序的良好编程习惯而写的
36         //前面设置了画笔的颜色，现在就应该把画笔的初始颜色恢复过来
37         //就相当于是画家用完画笔之后把画笔上的颜色清理掉一样
38         Color c = g.getColor();
39         g.setColor(c);
40
41         System.out.println("gogoogo");
42
43     }
44
45 }
```

九、鼠标事件适配器

- 抽象类java.awt.event.MouseAdapter实现了MouseListener接口，可以使用其子类作为MouseEvent的监听器，只要重写其相应的方法即可。
- 对于其他的监听器，也有对应的适配器。
- 适用适配器可以避免监听器定义没有必要的空方法。

【测试代码：画点】

```
1 package com.kuang3;
2
3 import java.awt.*;
```

```java
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.ArrayList;
import java.util.Iterator;

public class TestMouseAdapter {
    public static void main(String[] args) {
        new MyFrame("drawing....");
    }
}

class MyFrame extends Frame{
    ArrayList points = null;
    MyFrame(String s){
        super(s);
        points = new ArrayList();
        setLayout(null);
        setBounds(200,200,400,300);
        this.setBackground(new Color(204,204,255));
        setVisible(true);
        this.addMouseListener(new Monitor());
    }

    public void paint(Graphics g){
        Iterator i = points.iterator();
        while (i.hasNext()){
            Point p = (Point)i.next();
            g.setColor(Color.BLUE);
            g.fillOval(p.x,p.y,10,10);
        }
    }

    public void addPoint(Point p){
        points.add(p);
    }

    private class Monitor extends MouseAdapter{
        @Override
        public void mousePressed(MouseEvent e) {
            MyFrame frame = (MyFrame) e.getSource();
            frame.addPoint(new Point(e.getX(),e.getY()));
            frame.repaint();
        }
    }

}
```

十、window事件

- Window事件所对应的事件类为WindowEvent ，所对应的事件监听接口为WindowListener 。
- WindowListener定义的方法有：

```
public void windowOpened(WindowEvent e)
public void windowClosing(WindowEvent e)
public void windowClosed(WindowEvent e)
public void windowIconified(WindowEvent e)
public void windowDeiconified(WindowEvent e)
public void windowActivated(WindowEvent e)
public void windowDeactivated(WindowEvent e)
```

- 与WindowListener对应的适配器为WindowAdapter。

```java
package com.kuang3;

import java.awt.*;
import java.awt.event.*;

public class TestWindowClose{
    public static void main(String args[]){
        new WindowFrame("关闭WindowFrame");
    }
}

class WindowFrame extends Frame{
    public WindowFrame(String s){
        super(s);
        setBounds(200,200,400,300);
        setLayout(null);
        setBackground(new Color(204,204,255));
        setVisible(true);
        this.addWindowListener(new WindowMonitor());
        /*监听本窗体的动作，把所有的动作信息封装成一个对象传递到监听类里面*/

        this.addWindowListener(
            /*在一个方法里面定义一个类，这个类称为局部类，也叫匿名的内部类，
        这里的{……代码……}里面的代码很像一个类的类体，只不过这个类没有名字，所以叫匿名类
        在这里是把这个匿名类当成WindowAdapter类来使用，语法上这样写的本质意义是相当于这个匿名类
        从WindowAdapter类继承，现在new了一个匿名类的对象出来然后把这个对象当成WindowAdapter来使用
        这个匿名类出了()就没有人认识了*/
            new WindowAdapter(){
                public void windowClosing(WindowEvent e){
                    setVisible(false);
                    System.exit(-1);
                }
            }
        );
    }

    /*这里也是将监听类定义为内部类*/
    class WindowMonitor extends WindowAdapter{
        /*WindowAdapter(Window适配器)类实现了WindowListener监听接口
        重写了WindowListener接口里面的所有方法
```

```
        如果直接使用自定义WindowMonitor类直接去
        实现WindowListener接口，那么就得要重写WindowListener接口
        里面的所有方法，但现在只需要用到这些方法里面的其中一个方法
        所以采用继承实现WindowListener监听接口的一个子类
        并重写这个子类里面需要用到的那个方法即可
        这种做法比直接实现WindowListener监听接口要重写很多个用不到的方法要简洁方便得多
*/
        /*重写需要用到的windowClosing(WindowEvent e)方法*/
        public void windowClosing(WindowEvent e){
            setVisible(false);/*将窗体设置为不显示，即可实现窗体关闭*/
            System.exit(0);/*正常退出*/
        }
    }
}
```

十一、键盘响应事件

【键盘响应事件——KeyEvent】

```
package com.kuang3;

import java.awt.*;
import java.awt.event.*;

public class TestKeyEvent{
    public static void main(String args[]){
        new KeyFrame("键盘响应事件");
    }
}

class KeyFrame extends Frame{
    public KeyFrame(String s){
        super(s);
        setBounds(200,200,400,300);
        setLayout(null);
        setVisible(true);
        addKeyListener(new KeyMonitor());
    }
    /*把自定义的键盘的监听类定义为内部类
    这个监听类从键盘适配器KeyAdapter类继承
    从KeyAdapter类继承也是为了可以简洁方便
    只需要重写需要用到的方法即可，这种做法比
    直接实现KeyListener接口要简单方便，如果
    直接实现KeyListener接口就要把KeyListener
    接口里面的所有方法重写一遍，但真正用到的
    只有一个方法，这样重写其他的方法但又用不到
    难免会做无用功*/
    class KeyMonitor extends KeyAdapter{
        public void keyPressed(KeyEvent e){
            int keycode = e.getKeyCode();
            /*使用getKeyCode()方法获取按键的虚拟码*/
            /*如果获取到的键的虚拟码等于up键的虚拟码
            则表示当前按下的键是up键
            KeyEvent.VK_UP表示取得up键的虚拟码
            键盘中的每一个键都对应有一个虚拟码
```

```
                这些虚拟码在KeyEvent类里面都被定义为静态常量
                所以可以使用“类名.静态常量名”的形式访问得到这些静态常量*/
                if(keycode == KeyEvent.VK_UP){
                    System.out.println("你按的是up键");
                }
            }
        }
}
/*键盘的处理事件是这样的：每一个键都对应着一个虚拟的码，
当按下某一个键时，系统就会去找这个键对应的虚拟的码，以此来确定当前按下的是那个键
*/
```

Swing

Swing是GUI（图形用户界面）开发工具包，内容有很多，这里会分块编写，但在进阶篇中只编写Swing中的基本要素，包括容器、组件和布局等，更深入的内容这里就不介绍了。想深入学习的朋友们可查阅有关资料或图书，比如*《Java Swing图形界面开发与案例详解》——清华大学出版社*。

早期的AWT（抽象窗口工具包）组件开发的图形用户界面，要依赖本地系统，当把AWT组件开发的应用程序移植到其他平台的系统上运行时，不能保证其外观风格，因此AWT是依赖于本地系统平台的。而使用Swing开发的Java应用程序，其界面是不受本地系统平台限制的，也就是说Swing开发的Java应用程序移植到其他系统平台上时，其界面外观是不会改变的。但要注意的是，虽然Swing提供的组件可以方便开发Java应用程序，但是Swing并不能取代AWT，在开发Swing程序时通常要借助与AWT的一些对象来共同完成应用程序的设计。

一、常用窗体

Swing窗体是Swing的一个组件，同时也是创建图形化用户界面的容器，可以将其它组件放置在窗体容器中。

1. JFrame框架窗体

JFrame窗体是一个容器，在Swing开发中我们经常要用到，它是Swing程序中各个组件的载体。语法格式如下：

```
JFrame jf = new JFrame(title);
```

当然，在开发中更常用的方式是通过继承java.swing.JFrame类创建一个窗体，可通过this关键字调用其方法。

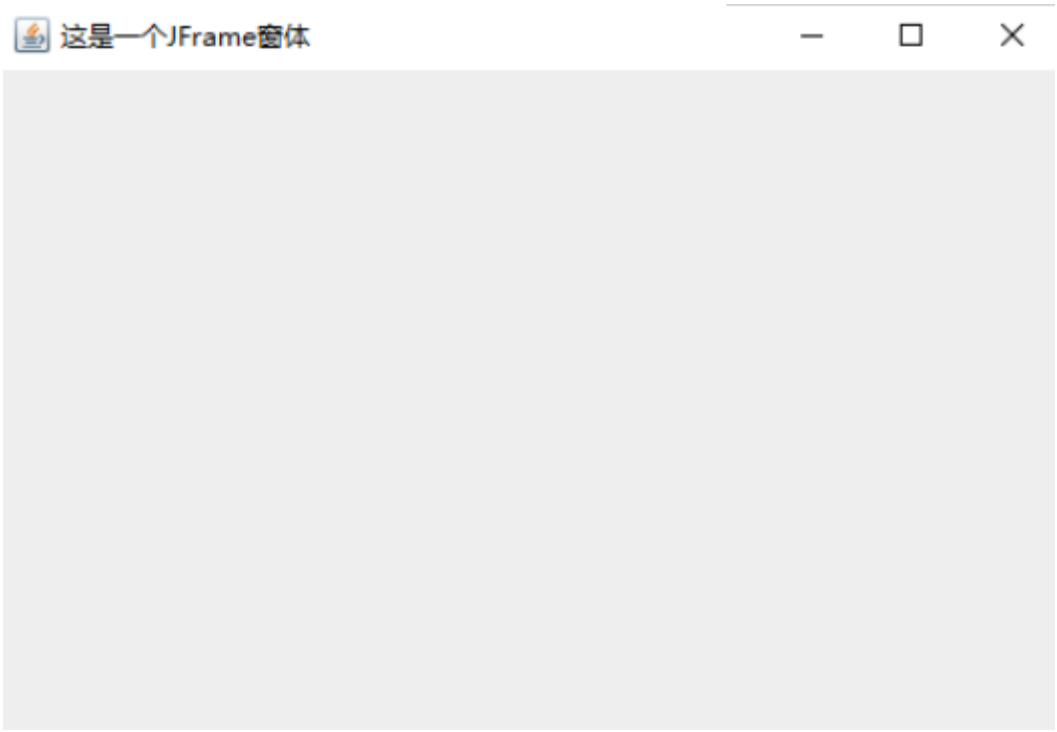
在JFrame对象创建完成后，需要调用getContentPane()方法将窗体转换为容器，然后在容器中添加组件或设置布局管理器，通常这个容器用来包含和显示组件。如果需要将组件添加至容器，可以使用来自Container类的add()方法进行设置。至于JPanel容器会在后面提到。

【下面举一个JFrame窗体的例子】

```
package com.kuang4;

import javax.swing.JFrame;
import javax.swing.WindowConstants;

public class JFrameDemo {
```

```
7
8       public void CreateJFrame() {
9           // 实例化一个JFrame对象
10          JFrame jf = new JFrame("这是一个JFrame窗体");
11          // 设置窗体可视
12          jf.setVisible(true);
13          // 设置窗体大小
14          jf.setSize(500, 350);
15          // 设置窗体关闭方式
16          jf.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
17      }
18
19      public static void main(String[] args) {
20          new JFrameDemo().CreateJFrame();        // 调用CreateJFrame()方法
21      }
22
23  }
```

结果：



这就是一个500*350的窗体，用的是setSize()方法；

标题为“这是一个JFrame窗体”，在实例化对象时就可以定义；

窗体关闭方式见窗体右上角为“EXIT_ON_CLOSE”；

窗体可视setVisible()方法中的参数为“false”或不写setVisible()方法时，此窗体不可见。

常用的窗体关闭方式有四种：

“DO_NOTHING_ON_CLOSE”：什么也不做就将窗体关闭；

“DISPOSE_ON_CLOSE”：任何注册监听程序对象后会自动隐藏并释放窗体；

“HIDE_ON_CLOSE”： 隐藏窗口的默认窗口关闭；

“EXIT_ON_CLOSE”：退出应用程序默认窗口关闭。

【下面再举一个用继承JFrame的方式编写的代码，并加入Container容器及JLabel标签（后面会提到），来看一下具体的流程。】

```java
package com.kuang4;

import java.awt.Color;
import java.awt.Container;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.SwingConstants;
import javax.swing.WindowConstants;

public class JFrameDemo2 extends JFrame{

    public void init() {
        // 可视化
        this.setVisible(true);
        // 大小
        this.setSize(500, 350);
        // 标题
        this.setTitle("西部开源");
        // 关闭方式
        this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        // 创建一个JLabel标签
        JLabel jl = new JLabel("欢迎来到西部开源学习，我是你们的Java老师，秦疆！");

        // 使标签文字居中
        jl.setHorizontalAlignment(SwingConstants.CENTER);

        // 获取一个容器
        Container container = this.getContentPane();
        // 将标签添加至容器
        container.add(jl);
        // 设置容器背景颜色
        container.setBackground(Color.YELLOW);
    }

    public static void main(String[] args) {
        new JFrameDemo2().init();
    }

}
```

运行结果：

这里继承了JFrame类，所以方法中实现时用this关键字即可（或直接实现，不加this）。

2. JDialog窗体

JDialog窗体是Swing组件中的对话框，继承了AWT组件中的java.awt.Dialog类。功能是从一个窗体中弹出另一个窗体。

【下面来看一个实例】

```
package com.kuang4;

import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.WindowConstants;

// 继承JDialog类
public class JDialogDemo extends JDialog {

    // 实例化一个JDialog类对象，指定其父窗体、窗口标题和类型
    public JDialogDemo() {
        super(new MyJFrame(), "这是一个JDialog窗体", true);
        Container container = this.getContentPane();
        container.add(new JLabel("秦老师带你学Java"));
        this.setSize(500, 350);
    }

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

```

```
}

// 下面这部分内容包含监听器，可自行查阅资料
class MyJFrame extends JFrame {
    public MyJFrame() {
        this.setVisible(true);
        this.setSize(700, 500);
        this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        Container container = this.getContentPane();
        container.setLayout(null);

        JButton jb = new JButton("点击弹出对话框");          // 创建按钮
        jb.setBounds(30, 30, 200, 50);          // 按钮位置及大小

        jb.addActionListener(new ActionListener() {          // 监听器，用于监听点击事件
            @Override
            public void actionPerformed(ActionEvent e) {
                new JDialogDemo().setVisible(true);
            }
        });
        container.add(jb);
    }
}
```

当我们点击按钮时，触发点击事件，创建一个JDialog的实例化对象，弹出一个窗口。这里出现了许多我们之前学过的知识，比如super关键字，在之前提到过，这里相当于使用了JDialog(Frame f, String title, boolean model)形式的构造方法；监听器的实现就是一个匿名内部类，之前也提到过。

二、标签组件

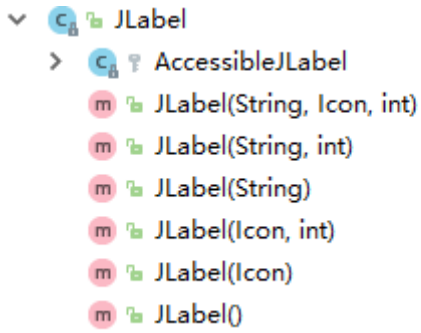
在Swing中显示文本或提示信息的方法是使用标签，它支持文本字符串和图标。上面我们提到的JLabel就是这里的内容。

1. 标签

标签由JLabel类定义，可以显示一行只读文本、一个图像或带图像的文本。

JLabel类提供了许多构造方法，可查看API选择需要的使用，如显示只有文本的标签、只有图标的标签或包含文本与图标的标签等。因为上面已经出现过了，这里就不再举例了。常用语法格式如下，创建的是一个不带图标和文本的JLabel对象：

```
JLabel jl = new JLabel();
```



2. 图标

Swing中的图标可以放置在按钮、标签等组件上，用于描述组件的用途。图标可以用Java支持的图片文件类型进行创建，也可以使用java.awt.Graphics类提供的功能方法来创建。

在Swing中通过Icon接口来创建图标，可以在创建时给定图标的大小、颜色等特性。

注意，Icon是接口，在使用Icon接口的时候，必须实现Icon接口的三个方法：

```
public int getIconHeight()
public int getIconWidth()
public void paintIcon(Component arg0, Graphics arg1, int arg2, int arg3)
```

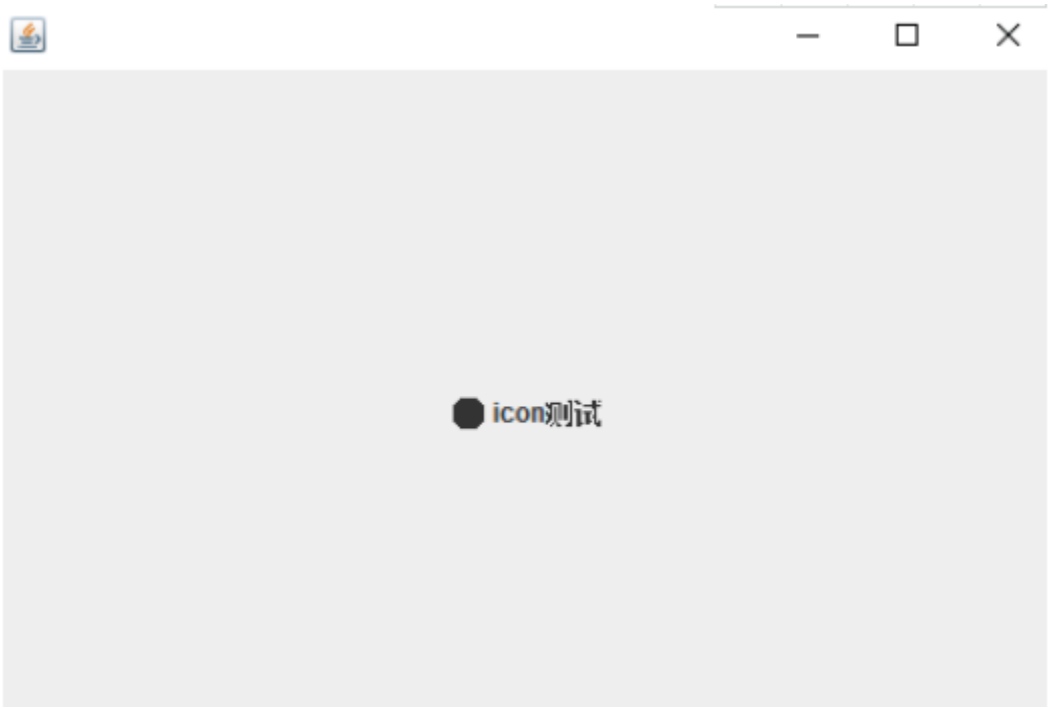
前两个方法用于获取图片的长宽，paintIcon()方法用于实现在指定坐标位置画图。

下面看一个用Icon接口创建图标的实例：

```
package com.kuang4;

import java.awt.Component;
import java.awt.Container;
import java.awt.Graphics;

import javax.swing.Icon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.SwingConstants;
import javax.swing.WindowConstants;

public class IconDemo extends JFrame implements Icon {

    private int width;        // 声明图标的宽
    private int height;        // 声明图标的长

    public IconDemo() {}        // 定义无参构造方法

    public IconDemo(int width, int height) {        // 定义有参构造方法
        this.width = width;
        this.height = height;
    }

    @Override
    public int getIconHeight() {        // 实现getIconHeight()方法
        return this.height;
    }

    @Override
    public int getIconWidth() {            // 实现getIconWidth()方法
```

```
        return this.width;
    }

    @Override
    public void paintIcon(Component arg0, Graphics arg1, int arg2, int arg3)
{        // 实现paintIcon()方法
        arg1.fillOval(arg2, arg3, width, height);        // 绘制一个圆形
    }

    public void init() {    // 定义一个方法用于实现界面
        IconDemo iconDemo = new IconDemo(15, 15);        // 定义图标的长和宽
        JLabel jb = new JLabel("icon测试", iconDemo, SwingConstants.CENTER);
   // 设置标签上的文字在标签正中间

        Container container = getContentPane();
        container.add(jb);

        this.setVisible(true);
        this.setSize(500, 350);
        this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {
        new IconDemo().init();
    }

}
```

运行结果如下：



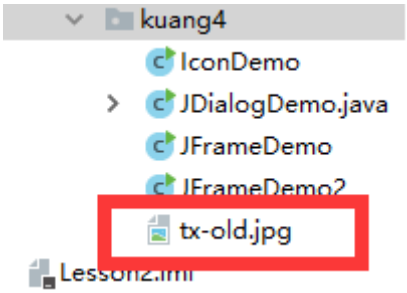
这样如果需要在窗体中使用图标，就可以用如下代码创建图标：

```
IconDemo iconDemo = new IconDemo(15, 15);
```

3. 图片图标

Swing中的图标除了可以绘制之外，还可以使用某个特定的图片创建。利用javax.swing.ImageIcon类根据现有图片创建图标。

下面看一个实例，我们先在包下放一个图片（注意放置位置，不同位置路径不同），如下：



【下面是实现的代码】

```
package com.kuang4;

import java.awt.Container;
import java.net.URL;

import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.SwingConstants;
import javax.swing.WindowConstants;

public class ImageIconDemo extends JFrame {

    public ImageIconDemo() {
        JLabel jl = new JLabel("这是一个JFrame窗体，旁边是一个图片");
        URL url = ImageIconDemo.class.getResource("tx-old.jpg");        //
获得图片所在URL
        Icon icon = new ImageIcon(url);        // 实例化Icon对象
        jl.setIcon(icon);        // 为标签设置图片
        jl.setHorizontalAlignment(SwingConstants.CENTER);
        jl.setOpaque(true);        // 设置标签为不透明状态

        Container container = getContentPane();
        container.add(jl);

        setVisible(true);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setSize(500, 350);
    }

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

}
```

对于图片标签，我们经常将图片放置在标签上，用JLabel中的setIcon()方法即可，当然也可以在初始化JLabel对象时为标签指定图标，这需要获取一个Icon实例。

而getResource()方法可以获得资源文件的URL路径，这里的路径是相对于前面的那个类的，所以可将该图片与该类放在同一个文件夹下；如果不在同一个文件夹下，需通过其它方法获取路径。

三、布局管理器

Swing中，每个组件在容器中都有一个具体的位置和大小，在容器中摆放各自组件时很难判断其具体位置和大小，这里我们就要引入布局管理器了，它提供了基本的布局功能，可以有效的处理整个窗体的布局。常用的布局管理器包括流布局管理器、边界布局管理器、网格布局管理器等。

1. 绝对布局

绝对布局在上一篇的例子中已经出现过了，是硬性指定组件在容器中的位置和大小，可以使用绝对坐标的方式来指定组件的位置。步骤如下：

1. 使用Container.setLayout(null)方法取消布局管理器
2. 使用Container.setBounds()方法设置每个组件的位置和大小

【举一个简单的例子】

```
Container container = getContentPane();     // 创建容器
JButton jb = new JButton("按钮");     // 创建按钮
jb.setBounds(10, 30, 100, 30);     // 设置按钮位置和大小
container.add(jb);     // 将按钮添加到容器中
```

setBounds()方法中，前两个参数是位置的xy坐标，后两个参数是按钮的长和宽。

2. 流布局管理器

流布局管理器是布局管理器中最基本的布局管理器，使用FlowLayout类，像“流”一样从左到右摆放组件，直到占据了这一行的所有空间，再向下移动一行。组件在每一行的位置默认居中排列，要更改位置可自行设置。

在FlowLayout的有参构造方法中，alignment设置为0时，每一行的组件将被指定左对齐排列；当alignment被设置为2时，每一行的组件将被指定右对齐排列；而为1时是默认的居中排列。

下面举个例子，创建10个按钮并用流布局管理器排列。

```
package com.kuang5;

import java.awt.Container;
import java.awt.FlowLayout;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.WindowConstants;

public class FlowLayoutDemo extends JFrame {

    public FlowLayoutDemo() {
        Container container = this.getContentPane();
        // 设置流布局管理器，2是右对齐，后两个参数分别为组件间的水平间隔和垂直间隔
        this.setLayout(new FlowLayout(2, 10, 10));

        // 循环添加按钮
        for(int i=0; i<10; i++) {
            container.add(new JButton("按钮" + i));
        }

        this.setSize(300, 200);
        this.setVisible(true);
        this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }

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

}
```

第一个参数为2是右对齐，每个按钮间的水平、垂直间隔都为10。后两个图分别为参数为1居中排列和参数为0左对齐。运行结果如下：

3. 边界布局管理器

在不指定窗体布局时，Swing组件默认的布局管理器是边界布局管理器，使用的是BorderLayout类。在上篇例子中，一个Label标签占据了整个空间，实质上是默认使用了边界布局管理器。边界布局管理器还可以容器分为东、南、西、北、中五个区域，可以将组件加入这五个区域中。

【演示】

```
1  package com.kuang5;
2
3  import java.awt.BorderLayout;
4  import java.awt.Container;
5
6  import javax.swing.JButton;
7  import javax.swing.JFrame;
8  import javax.swing.WindowConstants;
9
10 public class BorderLayoutDemo extends JFrame {
11
12     private String[] border = {BorderLayout.CENTER, BorderLayout.NORTH,
```

```
        BorderLayout.SOUTH, BorderLayout.WEST, BorderLayout.EAST};    // 此数组用于存放组件摆放位置
    private String[] button = {"中", "北", "南", "西", "东"};    // 此数组用于存放按钮名称

    public BorderLayoutDemo() {
        Container container = this.getContentPane();
        this.setLayout(new BorderLayout());        // 设置容器为边界布局管理器

        // 循环添加按钮
        for(int i=0; i<button.length ; i++) {
            container.add(border[i], new JButton(button[i]));    // 左参数为设置布局，右参数为创建按钮
        }

        this.setVisible(true);
        this.setSize(300, 200);
        this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }

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

}
```

4. 网格布局管理器

网格布局管理器将容器划分为网格，组件按行按列排列，使用GridLayout类。在此布局管理器中，每个组件的大小都相同，且会填满整个网格，改变窗体大小，组件也会随之改变。

【演示】

```
package com.kuang5;

import java.awt.Container;
import java.awt.GridLayout;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.WindowConstants;

class GirdLayoutDemo extends JFrame {

```

```
    public GirdLayoutDemo() {
        Container container = this.getContentPane();
        this.setLayout(new GridLayout(7, 3, 5, 5));    // 前两个参数为7行3列，后两个参数为网格间的间距

        for(int i=0; i<20; i++) {
            container.add(new JButton("按钮" + i));
        }

        this.setVisible(true);
        this.setSize(300, 300);
        this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }

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

}
```

四、面板

面板也是一个容器，可作为容器容纳其他组件，但也必须被添加到其他容器中。Swing中常用面板有JPanel面板和JScrollPane面板。

1. JPanel

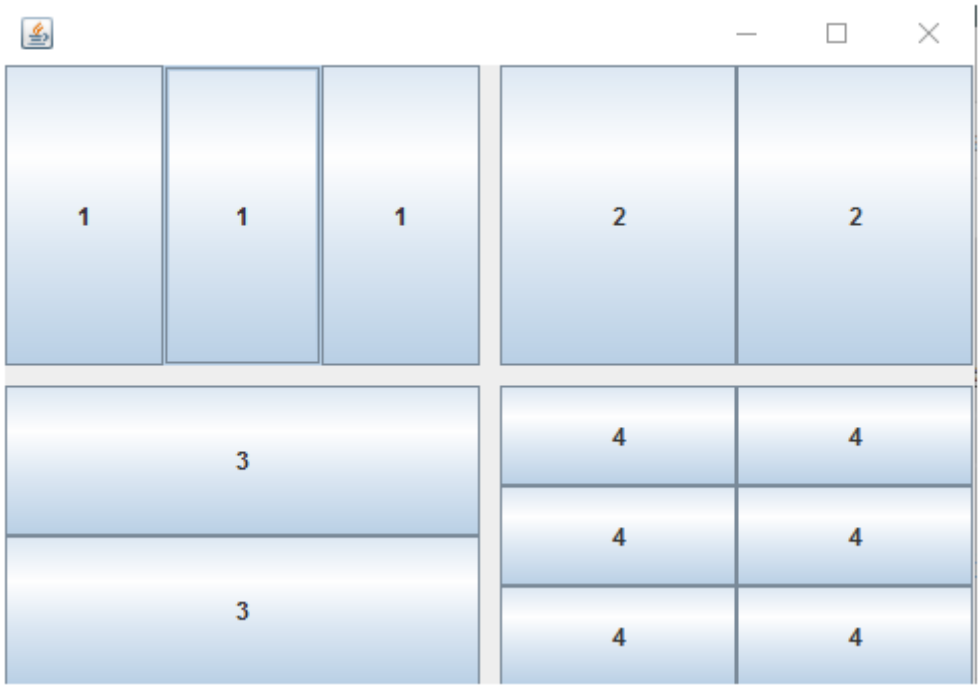
JPanel面板可以聚集一些组件来布局。继承自java.awt.Container类。

【演示】

```
package com.kuang5;

import java.awt.Container;
import java.awt.GridLayout;

import javax.swing.JButton;
```

```java
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.WindowConstants;

public class JPanelDemo extends JFrame {

    public JPanelDemo() {
        Container container = this.getContentPane();
        container.setLayout(new GridLayout(2, 1, 10, 10));    // 整个容器为2行1列

        JPanel p1 = new JPanel(new GridLayout(1, 3));    // 初始化一个面板，设置1行3列的网格布局
        JPanel p2 = new JPanel(new GridLayout(1, 2));    // 初始化一个面板，设置1行2列的网格布局
        JPanel p3 = new JPanel(new GridLayout(2, 1));    // 初始化一个面板，设置2行1列的网格布局
        JPanel p4 = new JPanel(new GridLayout(3, 2));    // 初始化一个面板，设置3行2列的网格布局

        p1.add(new JButton("1"));    // 在JPanel面板中添加按钮
        p1.add(new JButton("1"));    // 在JPanel面板中添加按钮
        p1.add(new JButton("1"));    // 在JPanel面板中添加按钮
        p2.add(new JButton("2"));    // 在JPanel面板中添加按钮
        p2.add(new JButton("2"));    // 在JPanel面板中添加按钮
        p3.add(new JButton("3"));    // 在JPanel面板中添加按钮
        p3.add(new JButton("3"));    // 在JPanel面板中添加按钮
        p4.add(new JButton("4"));    // 在JPanel面板中添加按钮
        p4.add(new JButton("4"));    // 在JPanel面板中添加按钮
        p4.add(new JButton("4"));    // 在JPanel面板中添加按钮
        p4.add(new JButton("4"));    // 在JPanel面板中添加按钮
        p4.add(new JButton("4"));    // 在JPanel面板中添加按钮
        p4.add(new JButton("4"));    // 在JPanel面板中添加按钮

        container.add(p1);        // 在容器中添加面板
        container.add(p2);        // 在容器中添加面板
        container.add(p3);        // 在容器中添加面板
        container.add(p4);        // 在容器中添加面板

        this.setVisible(true);
        this.setSize(500, 350);
        this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }

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

}
```

运行结果如下，可自行对比代码与结果理解JPanel。其中，容器的GridLayout布局设置了横纵都为10的间距，JPanel的GridLayout布局没有设置网格间距。



2. JScrollPane

若遇到一个较小的容器窗体中显示一个较大部分内容的情况，可用JScrollPane面板。这是一个带滚动条的面板，就像平时浏览网页，经常遇到的滚动条一样。

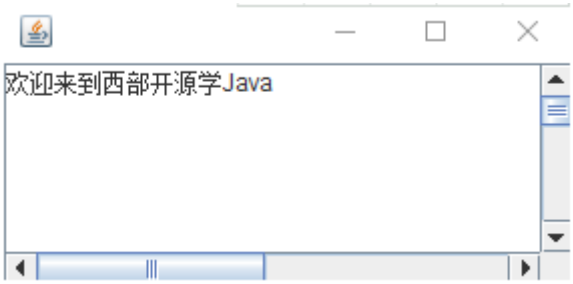
如果需要在JScrollPane面板中放置多个组件，需将这多个组件放置在JPanel面板上，然后将JPanel面板作为一个整体组件添加在JScrollPane面板上。

【演示】

```
package com.kuang5;

import java.awt.Container;

import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.WindowConstants;

public class JScrollPaneDemo extends JFrame {

    public JScrollPaneDemo() {
        Container container = this.getContentPane();

        JTextArea tArea = new JTextArea(20, 50);        // 创建文本区域组件
        tArea.setText("欢迎来到西部开源学Java");

        JScrollPane sp = new JScrollPane(tArea);
        container.add(sp);

        this.setVisible(true);
        this.setSize(300, 150);
        this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }

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

```
    }

}
```

结果：



其中JTextArea是创建一个文本区域组件，大小为20*50，setText()方法是给该文本区域填值。这里在new一个JScrollPane时，就将文本区域组件添加到其上。

五、按钮组件

1. 提交按钮组件（JButton）

JButton在之前的例子中已经出现多次，是较为常用的组件，用于触发特定动作。可以在按钮上显示文本标签，还可以显示图标，如下：

```
package com.kuang5;

import javax.swing.*;
import java.awt.*;

public class Demo extends JFrame {

    public Demo(){
        Container container = this.getContentPane();
        Icon icon = new ImageIcon(Demo.class.getResource("tx-old.jpg"));

        JButton jb = new JButton();

        jb.setIcon(icon);  // 设置图标
        jb.setToolTipText("图片按钮");  // 设置按钮提示

        container.add(jb);

        this.setVisible(true);
        this.setSize(500, 350);
        this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }

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

2. 单选按钮组件（JRadioButton）

默认情况下，单选按钮显示一个圆形图标，通常在其旁放置一些说明性文字。当用户选中某个单选按钮后，按钮组中其它按钮将被自动取消，这时就需要按钮组（ButtonGroup）来将同组按钮放在一起，该按钮组中的按钮只能选择一个，而不在此按钮中的按钮不受影响。语法格式如下：

```
package com.kuang5;

import javax.swing.*;
import java.awt.*;

public class Demo extends JFrame {

    public Demo(){
        Container container = this.getContentPane();


        Icon icon = new ImageIcon(Demo.class.getResource("tx-old.jpg"));

        //单选框
        JRadioButton jr1 = new JRadioButton("JRadioButton1");
        JRadioButton jr2 = new JRadioButton("JRadioButton2");
        JRadioButton jr3 = new JRadioButton("JRadioButton3");

        //按钮组，单选框只能选择一个
        ButtonGroup group = new ButtonGroup();
        group.add(jr1);
        group.add(jr2);
        group.add(jr3);

        container.add(jr1,BorderLayout.CENTER);
        container.add(jr2,BorderLayout.NORTH);
        container.add(jr3,BorderLayout.SOUTH);

        this.setVisible(true);
        this.setSize(500, 350);
        this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }

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

3. 复选框组件（JCheckBox）

复选框是一个方块图标，外加一段描述性文字，与单选按钮的区别就是可以多选。每一个复选框都提供“选中”与“不选中”两种状态。语法格式如下：

```
package com.kuang5;

import javax.swing.*;
import java.awt.*;

public class Demo extends JFrame {
```

```
    public Demo(){
        Container container = this.getContentPane();

        Icon icon = new ImageIcon(Demo.class.getResource("tx-old.jpg"));

        //多选框
        JCheckBox jrb  = new JCheckBox("abc");
        JCheckBox jrb2  = new JCheckBox("abc");
        container.add(jrb);
        container.add(jrb2,BorderLayout.NORTH);


        this.setVisible(true);
        this.setSize(500, 350);
        this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }

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

六、列表组件

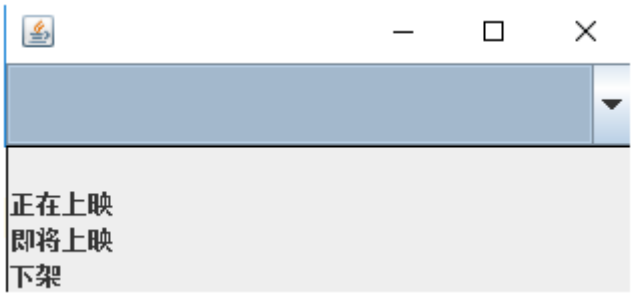
1. 下拉列表（JComboBox）

下拉列表框使用JComboBox类对象来表示，如下方代码：

```
package com.kuang5;

import javax.swing.*;
import java.awt.*;

public class Demo extends JFrame {

    public Demo(){
        Container container = this.getContentPane();

        Icon icon = new ImageIcon(Demo.class.getResource("tx-old.jpg"));

        JComboBox status = new JComboBox();
        status.addItem(null);
        status.addItem("正在上映");
        status.addItem("即将上映");
        status.addItem("下架");

        container.add(status);


        this.setVisible(true);
        this.setSize(500, 350);
        this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
```

```
    }

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

显示的样式如下：



2. 列表框（JList）

列表框只是在窗体上占据固定的大小，如果要使列表框具有滚动效果，可以将列表框放入滚动面板中。

使用数组初始化列表框的参数如下。

```
package com.kuang5;

import javax.swing.*;
import java.awt.*;

public class Demo extends JFrame {

    public Demo(){
        Container container = this.getContentPane();

        Icon icon = new ImageIcon(Demo.class.getResource("tx-old.jpg"));

        //使用数组初始化列表框的参数如下。
        String[] contents = {"1", "2", "3"};
        JList jl = new JList(contents);

        container.add(jl);

        this.setVisible(true);
        this.setSize(500, 350);
        this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }

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

将Vector类型的数据作为初始化JList的参数如下。

```
package com.kuang5;

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

```java
import java.util.Vector;

public class Demo extends JFrame {

    public Demo(){
        Container container = this.getContentPane();

        Icon icon = new ImageIcon(Demo.class.getResource("tx-old.jpg"));

        //将Vector类型的数据作为初始化JList的参数如下
        Vector contents = new Vector();
        JList jl = new JList(contents);
        contents.add("1");
        contents.add("2");
        contents.add("3");

        container.add(jl);

        this.setVisible(true);
        this.setSize(500, 350);
        this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }

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

七、文本组件

1. 文本框（JTextField）

文本框用来显示或编辑一个单行文本，语法格式如下：

```java
JTextField jt = new JTextField("aaa");    // 创建一个文本框，值为aaa
JTextField jt2 = new JTextField("aaa", 20);    // 创建一个长度为20的文本框，值为aaa
jt.setText("");        // 将文本框置空
```

其余构造方法可参考API或源码。

2. 密码框（JPasswordField）

密码框与文本框的定义与用法类似，但会使用户输入的字符串以某种符号进行加密。如下方代码：

```java
JPasswordField jp = new JPasswordField();
jp.setEchoChar('#');        // 设置回显符号
```

3. 文本域（JTextArea）

文本域组件在上面的代码中已经出现了，如下方代码所示：

```
JTextArea tArea = new JTextArea(20, 50);        // 创建文本区域组件
tArea.setText("欢迎来到西部开源学Java");
```

我们对GUI编程就讲到这里了，授人以鱼不如授人以渔，相信大家经过这一小段的学习已经能掌握看方法和源码学习的能力了，之后我们会有一些小游戏专题来巩固我们JavaSE阶段的学习。

小游戏：2048

思路：

使用了4x4的GridLayout作为布局，然后使用16个JLabel作为方块ui。数据上则是使用一个长度为16的int数组储存方块的数值，通过监听上下左右的按键进行相应的数据处理，最后通过刷新函数将数据显示出来并设置颜色。这里提一下胜负判定的实现，胜的判定很简单，就是玩家凑出了至少一个2048的方块即为胜利，而失败的判定思路略复杂，主要是通过模拟用户分别按下上、下、左、右键后，判断格子里是否还有空位，如分别向四个方向移动后都无法产生空位，则判负。

【Game类】

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;

public class Game {

    //用于储存颜色的实体类
    private static class Color {
        public Color(int fc, int bgc) {
            fontColor = fc;//字体颜色
            bgColor = bgc;//背景颜色
        }

        public int fontColor;//字体颜色
        public int bgColor;//背景颜色
    }

    JFrame mainFrame;//主窗口对象
    JLabel[] jLabels;//方块，用jlabel代替
    int[] datas = new int[]{0, 0, 0, 0,
            0, 0, 0, 0,
            0, 0, 0, 0,
            0, 0, 0, 0};//每个方块上的数值
    int[] temp = new int[4];//方块移动算法中抽离的的临时数组
    int[] temp2 = new int[16];//用于检测方块是否有合并


    List emptyBlocks = new ArrayList<Integer>(16);//在生成新方块时用到的临时list，用以存放空方块

```

```
    //存放颜色的map
    static HashMap<Integer, Color> colorMap = new HashMap<Integer, Color>()
{{
        put(0, new Color(0x776e65, 0xCDC1B4));
        put(2, new Color(0x776e65, 0xeee4da));
        put(4, new Color(0x776e65, 0xede0c8));
        put(8, new Color(0xf9f6f2, 0xf2b179));
        put(16, new Color(0xf9f6f2, 0xf59563));
        put(32, new Color(0xf9f6f2, 0xf67c5f));
        put(64, new Color(0xf9f6f2, 0xf65e3b));
        put(128, new Color(0xf9f6f2, 0xedcf72));
        put(256, new Color(0xf9f6f2, 0xedcc61));
        put(512, new Color(0xf9f6f2, 0xe4c02a));
        put(1024, new Color(0xf9f6f2, 0xe2ba13));
        put(2048, new Color(0xf9f6f2, 0xecc400));
    }};

    public Game() {
        initGameFrame();
        initGame();
        refresh();
    }

    //开局时生成两个2的方块和一个4的方块
    private void initGame() {
        for (int i = 0; i < 2; i++) {
            generateBlock(datas, 2);
        }
        generateBlock(datas, 4);
    }

    //随机生成4或者2的方块
    private void randomGenerate(int arr[]) {
        int ran = (int) (Math.random() * 10);
        if (ran > 5) {
            generateBlock(arr, 4);
        } else {
            generateBlock(arr, 2);
        }

    }

    //随机生成新的方块，参数：要生成的方块数值
    private void generateBlock(int arr[], int num) {
        emptyBlocks.clear();

        for (int i = 0; i < 16; i++) {
            if (arr[i] == 0) {
                emptyBlocks.add(i);
            }
        }
        int len = emptyBlocks.size();
        if (len == 0) {
            return;
        }
        int pos = (int) (Math.random() * 100) % len;
        arr[(int) emptyBlocks.get(pos)] = num;
        refresh();
```

```
92
93      }
94
95
96      //胜负判定并做终局处理
97      private void judge(int arr[]) {
98
99          if (isWin(arr)) {
100             JOptionPane.showMessageDialog(null, "恭喜，你已经成功凑出2048的方块", "你赢了", JOptionPane.PLAIN_MESSAGE);
101             System.exit(0);
102         }
103         if (isEnd(arr)) {
104             int max = getMax(datas);
105             JOptionPane.showMessageDialog(null, "抱歉，你没有凑出2048的方块,你的最大方块是：" + max, "游戏结束", JOptionPane.PLAIN_MESSAGE);
106             System.exit(0);
107         }
108
109     }
110
111     //判断玩家是否胜利，只要有一个方块大于等于2048即为胜利
112     private boolean isWin(int arr[]) {
113         for (int i : arr) {
114             if (i >= 2048) {
115                 return true;
116             }
117         }
118         return false;
119
120     }
121
122     //此函数用于判断游戏是否结束，如上下左右移后均无法产生空块，即代表方块已满，则返回真，表示游戏结束
123     private boolean isEnd(int arr[]) {
124
125         int[] tmp = new int[16];
126         int isend = 0;
127
128         System.arraycopy(arr, 0, tmp, 0, 16);
129         left(tmp);
130         if (isNoBlank(tmp)) {
131             isend++;
132         }
133
134         System.arraycopy(arr, 0, tmp, 0, 16);
135         right(tmp);
136         if (isNoBlank(tmp)) {
137             isend++;
138         }
139
140         System.arraycopy(arr, 0, tmp, 0, 16);
141         up(tmp);
142         if (isNoBlank(tmp)) {
143             isend++;
144         }
145
146         System.arraycopy(arr, 0, tmp, 0, 16);
```

```
147             down(tmp);
148             if (isNoBlank(tmp)) {
149                 isend++;
150             }
151 
152             if (isend == 4) {
153                 return true;
154             } else {
155                 return false;
156             }
157         }
158 
159         //判断是否无空方块
160         private boolean isNoBlank(int arr[]) {
161 
162             for (int i : arr) {
163                 if (i == 0) {
164                     return false;
165                 }
166             }
167             return true;
168         }
169 
170         //获取最大的方块数值
171         private int getMax(int arr[]) {
172             int max = arr[0];
173             for (int i : arr) {
174                 if (i >= max) {
175                     max = i;
176                 }
177             }
178             return max;
179         }
180 
181         //刷新每个方块显示的数据
182         private void refresh() {
183             JLabel j;
184             for (int i = 0; i < 16; i++) {
185                 int arr = datas[i];
186                 j = jLabels[i];
187                 if (arr == 0) {
188                     j.setText("");
189                 } else if (arr >= 1024) {
190                     j.setFont(new Font("Dialog", 1, 42));
191                     j.setText(String.valueOf(datas[i]));
192                 } else {
193                     j.setFont(new Font("Dialog", 1, 50));
194                     j.setText(String.valueOf(arr));
195                 }
196 
197                 Color currColor = colorMap.get(arr);
198                 j.setBackground(new java.awt.Color(currColor.bgColor));
199                 j.setForeground(new java.awt.Color(currColor.fontColor));
200             }
201         }
202 
203         //初始化游戏窗口，做一些繁杂的操作
204         private void initGameFrame() {
```

```
205
206         //创建JFrame以及做一些设置
207         mainFrame = new JFrame("2048 Game");
208         mainFrame.setSize(500, 500);
209         mainFrame.setResizable(false);//固定窗口尺寸
210         mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
211         mainFrame.setLocationRelativeTo(null);
212
213         mainFrame.setLayout(new GridLayout(4, 4));
214         mainFrame.getContentPane().setBackground(new
    java.awt.Color(0xCDC1B4));
215         //添加按键监听
216         mainFrame.addKeyListener(new KeyListener() {
217             @Override
218             public void keyTyped(KeyEvent keyEvent) {
219             }
220
221             @Override
222             public void keyPressed(KeyEvent keyEvent) {
223
224                 System.arraycopy(datas, 0, temp2, 0, 16);
225
226                 //根据按键的不同调用不同的处理函数
227                 switch (keyEvent.getKeyCode()) {
228                     case KeyEvent.VK_UP:
229                         up(datas);
230                         break;
231
232                     case KeyEvent.VK_DOWN:
233                         down(datas);
234                         break;
235
236                     case KeyEvent.VK_LEFT:
237                         left(datas);
238                         break;
239
240                     case KeyEvent.VK_RIGHT:
241                         right(datas);
242                         break;
243
244                 }
245
246
247                 //判断移动后是否有方块合并，若有，生成新方块，若无，不产生新方块
248                 if (!Arrays.equals(datas, temp2)) {
249                     randomGenerate(datas);
250                 }
251
252                 refresh();
253                 judge(datas);
254             }
255
256             @Override
257             public void keyReleased(KeyEvent keyEvent) {
258             }
259         });
260
261         //使用系统默认的ui风格
```

```
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, e.getMessage());
        }

        //使用16个JLabel来显示16个方块
        jLabels = new JLabel[16];
        JLabel j; //引用复用，避免for里创建过多引用
        for (int i = 0; i < 16; i++) {
            jLabels[i] = new JLabel("0", JLabel.CENTER);
            j = jLabels[i];
            j.setOpaque(true);
            // 设置边界，参数：上，左，下，右，边界颜色
            j.setBorder(BorderFactory.createMatteBorder(6, 6, 6, 6, new java.awt.Color(0xBBADA0)));

            //j.setForeground(new java.awt.Color(0x776E65));
            j.setFont(new Font("Dialog", 1, 52));
            mainFrame.add(j);
        }
        mainFrame.setVisible(true);
    }

    private void left(int arr[]) {
        moveLeft(arr);

        combineLeft(arr);

        moveLeft(arr);//合并完后会产生空位，所以要再次左移


    }

    //向左合并方块
    private void combineLeft(int arr[]) {
        for (int l = 0; l < 4; l++) {
            //0 1 2
            for (int i = 0; i < 3; i++) {
                if ((arr[l * 4 + i] != 0 && arr[l * 4 + i + 1] != 0) && arr[l * 4 + i] == arr[l * 4 + i + 1]) {
                    arr[l * 4 + i] *= 2;
                    arr[l * 4 + i + 1] = 0;
                }
            }
        }
    }

    //方块左移，针对每一行利用临时数组实现左移
    private void moveLeft(int arr[]) {
        for (int l = 0; l < 4; l++) {


            int z = 0, fz = 0;//z(零）;fz（非零）
            for (int i = 0; i < 4; i++) {
                if (arr[l * 4 + i] == 0) {
                    z++;
```

```
                } else {
                    temp[fz] = arr[l * 4 + i];
                    fz++;
                }
            }
            for (int i = fz; i < 4; i++) {
                temp[i] = 0;
            }
            for (int j = 0; j < 4; j++) {
                arr[l * 4 + j] = temp[j];
            }
        }
    }

    private void right(int arr[]) {

        moveRight(arr);
        combineRight(arr);
        moveRight(arr);

    }

    private void combineRight(int arr[]) {
        for (int l = 0; l < 4; l++) {
            //3 2 1
            for (int i = 3; i > 0; i--) {
                if ((arr[l * 4 + i] != 0 && arr[l * 4 + i - 1] != 0) &&
arr[l * 4 + i] == arr[l * 4 + i - 1]) {
                    arr[l * 4 + i] *= 2;
                    arr[l * 4 + i - 1] = 0;
                }
            }
        }
    }

    private void moveRight(int arr[]) {

        for (int l = 0; l < 4; l++) {

            int z = 3, fz = 3;//z(零);fz（非零）
            for (int i = 3; i >= 0; i--) {
                if (arr[l * 4 + i] == 0) {
                    z--;
                } else {
                    temp[fz] = arr[l * 4 + i];
                    fz--;
                }
            }
            for (int i = fz; i >= 0; i--) {
                temp[i] = 0;
            }
            for (int j = 3; j >= 0; j--) {
                arr[l * 4 + j] = temp[j];
            }
        }
    }


```

```
374     private void up(int arr[]) {
375         moveUp(arr);
376         combineUp(arr);
377         moveUp(arr);
378 
379     }
380 
381     private void combineUp(int arr[]) {
382 
383 
384         for (int r = 0; r < 4; r++) {
385             for (int i = 0; i < 3; i++) {
386                 if ((arr[r + 4 * i] != 0 && arr[r + 4 * (i + 1)] != 0) &&
    arr[r + 4 * i] == arr[r + 4 * (i + 1)]) {
387                     arr[r + 4 * i] *= 2;
388                     arr[r + 4 * (i + 1)] = 0;
389                 }
390             }
391         }
392     }
393 
394     private void moveUp(int arr[]) {
395 
396         for (int r = 0; r < 4; r++) {
397 
398             int z = 0, fz = 0;//z(零）;fz（非零）
399             for (int i = 0; i < 4; i++) {
400                 if (arr[r + 4 * i] == 0) {
401                     z++;
402                 } else {
403                     temp[fz] = arr[r + 4 * i];
404                     fz++;
405                 }
406             }
407             for (int i = fz; i < 4; i++) {
408                 temp[i] = 0;
409             }
410             for (int j = 0; j < 4; j++) {
411                 arr[r + 4 * j] = temp[j];
412             }
413         }
414     }
415 
416 
417     private void down(int arr[]) {
418         moveDown(arr);
419         combineDown(arr);
420         moveDown(arr);
421     }
422 
423     private void combineDown(int arr[]) {
424         for (int r = 0; r < 4; r++) {
425             for (int i = 3; i > 0; i--) {
426                 if ((arr[r + 4 * i] != 0 && arr[r + 4 * (i - 1)] != 0) &&
    arr[r + 4 * i] == arr[r + 4 * (i - 1)]) {
427                     arr[r + 4 * i] *= 2;
428                     arr[r + 4 * (i - 1)] = 0;
429                 }
```

```
            }
        }
    }

    private void moveDown(int arr[]) {
        for (int r = 0; r < 4; r++) {

            int z = 3, fz = 3;//z(零);fz(非零)
            for (int i = 3; i >= 0; i--) {
                if (arr[r + 4 * i] == 0) {
                    z--;
                } else {
                    temp[fz] = arr[r + 4 * i];
                    fz--;
                }
            }
            for (int i = fz; i >= 0; i--) {
                temp[i] = 0;
            }
            for (int j = 3; j >= 0; j--) {
                arr[r + 4 * j] = temp[j];
            }
        }
    }

}
```

【StartFrame类】

```
package com.test2048;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class StartFrame {

    JFrame mainFrame;
    final String gameRule = "2048游戏共有16个格子，开始时会随机生成两个数值为2的方块和一个数值为4的方块，\n" +
            "玩家可通过键盘上的上、下、左、右方向键来操控方块的滑动方向，\n" +
            "每按一次方向键，所有的方块会向一个方向靠拢，相同数值的方块将会相加并合并成一个方块，\n" +
            "此外，每滑动一次将会随机生成一个数值为2或者4的方块，\n" +
            "玩家需要想办法在这16个格子里凑出2048数值的方块，若16个格子被填满且无法再移动，\n" +
            "则游戏结束。";

    public StartFrame() {
        initFrame();
    }

    private void initFrame() {
        mainFrame = new JFrame("2048 Game");
        mainFrame.setSize(500, 500);
        mainFrame.setResizable(false);//固定窗口尺寸
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        mainFrame.setLocationRelativeTo(null);//窗口居中


        JPanel jPanel = new JPanel();
        //BoxLayout.Y_AXIS是指定从上到下垂直布置组件。
        jPanel.setLayout(new BoxLayout(jPanel, BoxLayout.Y_AXIS));

        jPanel.add(newLine(Box.createVerticalStrut(25)));//添加空白区域

        JLabel jLabel = new JLabel("2048");
        jLabel.setForeground(new Color(0x776e65));
        jLabel.setFont(new Font("Dialog", 1, 92));
        jPanel.add(newLine(jLabel));

        /*
        JLabel author = new JLabel("by xxx");
        jPanel.add(newLine(author));
        */


        jPanel.add(newLine(Box.createVerticalStrut(50)));


        JButton btn1 = new JButton("开始游戏");
        btn1.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent actionEvent) {
                new Game();
                mainFrame.dispose();
            }
        });
        jPanel.add(newLine(btn1));


        jPanel.add(newLine(Box.createVerticalStrut(50)));


        JButton btn2 = new JButton("游戏规则");
        btn2.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent actionEvent) {
                JOptionPane.showMessageDialog(null, gameRule, "游戏规则",
JOptionPane.PLAIN_MESSAGE);
            }
        });
        jPanel.add(newLine(btn2));


        jPanel.add(newLine(Box.createVerticalStrut(50)));


        JButton btn3 = new JButton("退出游戏");
        btn3.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent actionEvent) {
                System.exit(0);
            }
        });
```

```
84          jPanel.add(newLine(btn3));
85
86
87          mainFrame.add(jPanel);
88
89          mainFrame.setVisible(true);
90      }
91
92      //添加新一行垂直居中的控件，通过在控件两边填充glue对象实现
93      private JPanel newLine(Component c) {
94
95          JPanel jp = new JPanel();
96          jp.setLayout(new BoxLayout(jp, BoxLayout.X_AXIS));
97          jp.add(Box.createHorizontalGlue());
98          jp.add(c);
99          jp.add(Box.createHorizontalGlue());
100         jp.setOpaque(false);//设置不透明
101
102         return jp;
103     }
104
105 }
```

【Main】

```
1  package com.test2048;
2
3  public class Main {
4
5      public static void main(String[] args) {
6          new StartFrame();
7      }
8
9  }
```