GUI图形界面设计

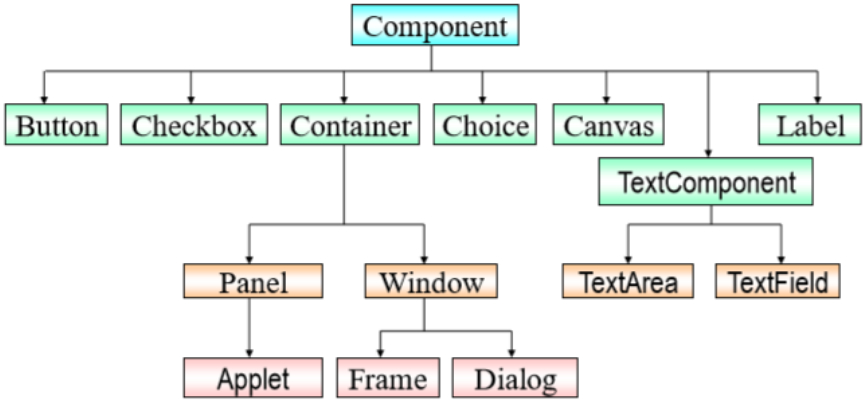
学GUI，学会规划布局

一、awt抽象窗口工具包（早期）

> java.awt.*

java提供的用来建立和设置java图形用户界面的基本工具。java.awt包中的所有类都可以用来建立与平台无关的图形用户界面（GUI）的类，这些类又称为组件。awt包提供的工具类主要有组件（Component)、容器（Container)、布局管理器（LayoutManager)。

- GUI图形组件：java.awt.*;
- GUI事件：java.awt.event.*;



（一）组件（Component)

Component是大部分图形组件的父类，**尤其注意组件是容器的父类！！！** 组件中包含容器、按钮、菜单、文本框、标签、滚动条等。具体怎么用按照所需查看API。

组件放置在容器中需要设定其位置大小，以下是一些共性方法：

- setLocation(int x, int y)
- setSize(int w, int h)

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

- setVisible(boolean b)
- setEnable(boolean b)
- setForeground(Color c)
- setBackground(Color c)
- setFont(Font f)
- ……

（二）容器（Container）

大容器套小容器，小容器里面套组件。

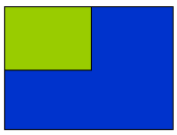
• Container是承载其它组件的容器，AWT的顶级容器包括：Applet、Frame、Dialog

• 只有顶级容器才能显示，其它所有容器 Container及组件Component必须“挂靠” 一个顶级容器，它通过Add()方法来实现

容器名称	含义	默认布局管理器
Panel(面板)	最简单的容器类，包含在窗口中的一个不带边框的区域，辅助GUI布局	`FlowLayout` 布局管理器
applet(小应用程序)	applet 是一种不能单独运行但可嵌入在其他应用程序中的小程序。	`FlowLayout` 布局管理器
Window	一个没有边界和菜单栏的顶层窗口	BorderLayout布局管理器
Frame（框架)	带有标题和边框的顶层窗口	BorderLayout
Dialog（对话框)	带标题和边界的顶层窗口	BorderLayout

例如：

```
/*  范例名称：Panel应用举例
 *  源文件名称：TestFrameWithPanel.java
 *  要  点：
 *      1. Panel组件的性质
 *      2. 容器和组件的概念
 *      3. setSize，setBackground，setLayout，add，setVisible等常用方法
 */

import java.awt.*;

public class TestFrameWithPanel {
    public static void main(String args[]) {
        Frame f = new Frame("MyTest Frame");
        Panel pan = new Panel();
        f.setSize(200, 200);
        f.setBackground(Color.blue);
        f.setLayout(null); // 取消默认布局管理器，采用绝对坐标布局
        pan.setSize(100, 100);
        pan.setBackground(Color.green);
        f.add(pan);//在框架上放一个面板
        f.setVisible(true);// 可视化后才能被显示
    }
}
```



（三）布局管理器（LayoutManager)

所有的布局管理器都是**Layoutmanager接口**的子类，容器Container类都具有设置布局管理器的 方法。

– public void setLayout(LayoutManager mgr)

布局管理器的种类：

种类名称	含义
Flow Layout	流式布局，组件从左向右依次排布
Border Layout	结构化布局
Card Layout	
Grid Layout	网格布局
Grid Bag Layout	网格袋布局

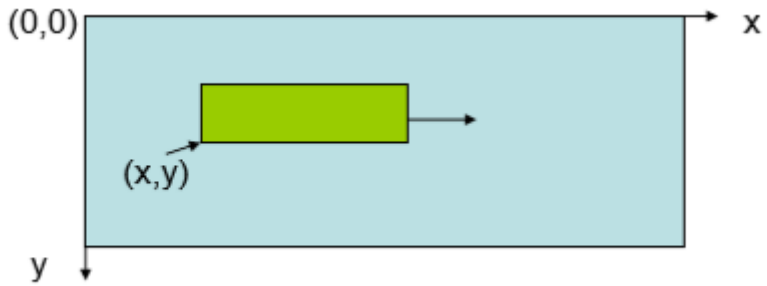
1.绝对布局

•setLayout(null);//如果需要按绝对坐标布局， 可以设布局管理器为null

• 对于需要添加到容器中的component，一定 要调用setBounds()（界限）方法确定具体布局

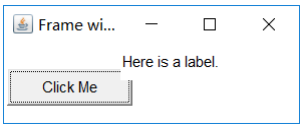
• 注意：setBounds()中(x,y)是component左下角的坐标 ？？？

void	setBounds(int x, int y, int width, int height) 移动组件并调整其大小。
void	setBounds(Rectangle r) 移动组件并调整其大小，使其符合新的有界矩形 r。




```
import java.awt.*;

public class NullLayout{ //与括号之间不能有空格
    public static void main(String[] args) {
        Frame frm = new Frame("Frame with Null Layout Management");
        // 完全由用户自己来布局
        frm.setLayout(null);
        Label lbl = new Label("Here is a label.");
        lbl.setBounds(new Rectangle(100, 30, 100, 30));
        frm.add(lbl);
        Button btn = new Button("Click Me");
```

```
        btn.setBounds(new Rectangle(10, 50, 100, 30));
        frm.add(btn);
        frm.setBounds(100, 100, 250, 100);
        frm.setVisible(true);
    }
}
```



2.相对布局

即使容器改变了大小，布局管理器也可以自己进行相对应的调整。**注意所有布局方式都是类，使用要用new 对象的方式**

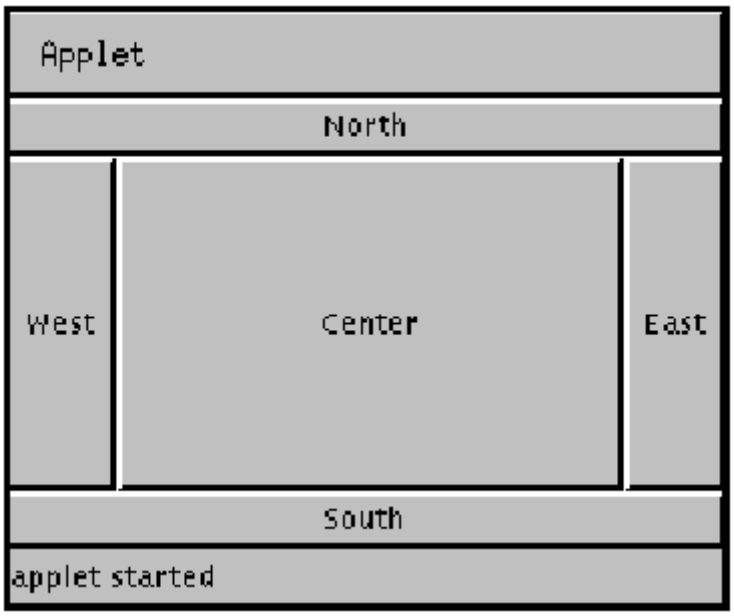
①.流式布局（FlowLayout)

• FlowLayout是流式布局，组件从左向右依 次排布

• 类似网页的缺省布局，当窗体改变时，组 件布局自动发生调整

```
public class FlowLayoutTest {
    public static void main(String[] args) {
        Frame frm = new Frame("Frame with Controls");

        frm.setLayout(new FlowLayout());

        Label lbl = new Label("Here is a label.");
        frm.add(lbl);

        Button btn1 = new Button("Click Me1");
        frm.add(btn1);
        Button btn2 = new Button("Click Me2");
        frm.add(btn2);
        Button btn3 = new Button("Click Me3");
        frm.add(btn3);

        frm.setBounds(0, 0, 400, 300);
        frm.setVisible(true);
    }
}
```

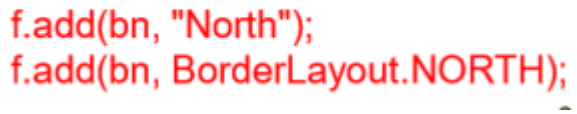
②.结构化布局(BorderLayout)



BorderLayout是结构化布局

• 加入组件的方法 – add(组件, 加入位置) –

加入位置可为： • BorderLayout.CENTER （EAST,SOUTH,WEST,NORTH)




```
Frame f;
f = new Frame("Border Layout");
Button bn = new Button("BN");
f.add(bn, "North");
//f.add(bn, BorderLayout.NORTH);
```

二、事件监听机制

> java.awt.event.*

事件监听机制，简单来说是加一个监视器monitor到控件上，当控件作出动作时，监视器调用响应的事件处理代码，给予相应的响应。

1）事件源对象
事件源对象是指触发事件发生的控件。所有的容器组件和元素组件都可以是事件源对象。比如“登录按钮”。

2）事件监听方法

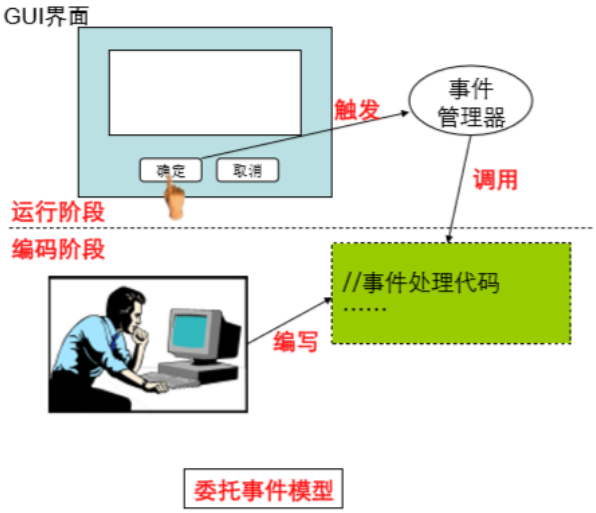
比如我们的需求是如果当登录按钮被点击的时候，程序会采取相应操作。
这就需要我们为事件源对象（登录按钮）添加一个监视器来监视事件源上是否发生了对应的动作。
它的作用是捕获发生在事件源对象上的动作，具体由动作来确定。

3）事件接口及实现（监视器类）

如果我们点击登录按钮后，程序采取相应的动作，我们必须**监听者必须实现ActionLister的接口（一个约定，里面就是事件处理代码）**，在**actionPerformed**(ActionEvent e)这个方法里定义事件的处理方法，控件发生事件时，就会去调用监听器的方法。

双线程，操作系统（事件委托者）得出事件，然后给jvm虚拟机，虚拟机调用事件处理代码。响应式编程

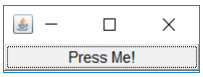
时间逻辑：



1.严格使用监视器三部曲

举例：

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

public class TestActionEvent {
    public static void main(String args[]) {
            Frame f = new Frame("Test");
            Button b = new Button("Press Me!");
            Monitor bh = new Monitor();//1.定义一个监听器
            b.addActionListener(bh);//2.把这个监听器挂到按钮B上
            f.add(b,BorderLayout.CENTER);
            f.pack();
            f.setVisible(true);
    }
}
/*事件处理代码放在被监听者类的外面*/
class Monitor implements ActionListener { //3.实现事件处理的代码(必须写的接口和方法)
    public void actionPerformed(ActionEvent e) {
        System.out.println("a button has been pressed");
    }
}
```

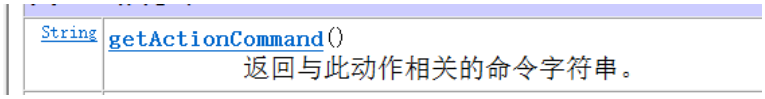


2.利用ActionEvent的方法将事件处理与事件捆绑起来 （鸡肋）

利用事件对象本身传递信息

```
public void actionPerformed(ActionEvent e) {
      System.out.println("a button has been pressed," +
                         "the relative info is:\n " + e.getActionCommand());
}
```

ActionEvent方法：



例如：

```
/*  范例名称：Java事件处理举例
 *  源文件名称：TestActionEvent2.java
 *  要  点：
 *      1. 一个事件源组件上可以同时注册多个监听器
 *      2. 一个监听器对象可以同时注册到多个事件源组件上
 *      3. 事件源的信息可以随它所触发的事件自动传递到所有注册过的监听器
 */

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

public class TestActionEvent2 {
    public static void main(String args[]) {
        Frame f = new Frame("Test");
        Button b1 = new Button("Start");
        Button b2 = new Button("Stop");
        Monitor2 bh = new Monitor2();
        b1.addActionListener(bh);
        b2.addActionListener(bh);//一个监视器检测两个控件
        b2.setActionCommand("game over");//设置事件处理
        f.add(b1, "North");
        f.add(b2, "Center");
        f.pack();
        f.setVisible(true);
    }
}

class Monitor2 implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("a button has been pressed," + "the relative info
is:\n " + e.getActionCommand());//捕捉事件处理代码
    }
}
```

```
a button has been pressed,the relative info is:
 Start
a button has been pressed,the relative info is:
  over
```

3.利用Event的get.Sourse()的方法（鸡肋）

所有event的父类拥有一个getSource()方法， 返回事件发生的对象(Object类型)

```
public void actionPerformed(ActionEvent e) {
      Button b = (Button)e.getSource();
      System.out.println("a button has been pressed," +
                         "the relative info is:\n " + b.getLabel());
}
```

```
import java.awt.*;
import java.awt.event.*;
public class TestActionEvent3 {
    public static void main(String args[]) {
            Frame f = new Frame("Test");
            Button b1 = new Button("Start");//事件的标签在这里！！
            Button b2 = new Button("Stop");
            Monitor2 bh = new Monitor2();
            b1.addActionListener(bh);
            b2.addActionListener(bh);
            f.add(b1,"North");
            f.add(b2,"Center");
            f.pack();
            f.setVisible(true);
    }
}

class Monitor2 implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        Button b = (Button)e.getSource();//返回发生事件的控件
        System.out.println("a button has been pressed," +
        "the relative info is:\n " + b.getLabel());  //读取事件的标签
    }
}
```

```
a button has been pressed,the relative info is:
 Start
a button has been pressed,the relative info is:
 Stop
```

4.通过构造方法传递或者使用内部类（GOOD!）

事件类event和事件类getSource都不足以提供全部信息该如何?

①.通过构造方法传递

事件监听类的接口方法我们不能改变，但可以改变其构造函数，通过构造函数传参 。

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

public class TFMath {
```

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

class TFFrame extends Frame {
    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 MyMonitor(this));//玄机在这里把this传进去了!
        setLayout(new FlowLayout());
        add(num1);
        add(lblPlus);
        add(num2);
        add(btnEqual);
        add(num3);
        pack();
        setVisible(true);
    }

}

class MyMonitor implements ActionListener {
    TFFrame tf = null;
    public MyMonitor(TFFrame tf) {
        this.tf = tf;
    }

    public void actionPerformed(ActionEvent e) {
        int n1 = Integer.parseInt(tf.num1.getText());
        int n2 = Integer.parseInt(tf.num2.getText());
        tf.num3.setText("" + (n1 + n2));

    }
}
```

②.使用内部类

把事件监听类放进被监听类的肚子里面，可以自由访问被监听的相关属性。

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

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

class TFFrame extends Frame {
    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 MyMonitor(this));
        setLayout(new FlowLayout());
        add(num1);
        add(lblPlus);
        add(num2);
        add(btnEqual);
        add(num3);
        pack();
        setVisible(true);
    }

    private class MyMonitor implements ActionListener {
        Frame f;
        public MyMonitor(Frame f) {
            this.f = f;//通过this把这个容器传进去
        }
        public void actionPerformed(ActionEvent e) {
            int n1 = Integer.parseInt(num1.getText());
            int n2 = Integer.parseInt(num2.getText());
            num3.setText("" + (n1 + n2));
        }
    }

}
```

拓展做法:

```
btnEqual.addActionListener(
    // new xxx implements ActionLister()
            new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                int n1 = Integer.parseInt(num1.getText());
                int n2 = Integer.parseInt(num2.getText());
                num3.setText("" + (n1 + n2));
            }
        });
```

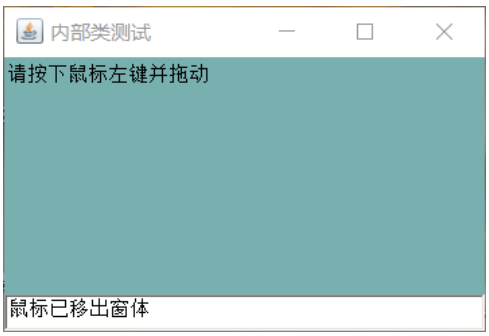
直接把监听器加到按钮上时就把接口实现了，在括号里把处理代码实现了。实际上是new了一个监视器实现了一个接口，但是这里是什么类不重要，所以把监视器名称省略了（匿名内部类），而这种用法建议只在在事件处理类里使用。好处在于控件与方法对应得近，写代码的时候好对照。

附：了解接口Listenr

接口的特点是里面的方法都要实现。

```
/*  范例名称：内部类在事件处理中的使用
 *  源文件名称：TestInner.java
 *  要  点：
 *      1. 内部类的性质和用法
```

```
 *      2. 将监听器类定义为普通内部类的好处----
 *         内部类中可直接访问外层类的属性和方法
 */

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

public class TestInner {
    Frame f = new Frame("内部类测试");
    TextField tf = new TextField(30);

    public TestInner(){
        f.add(new Label("请按下鼠标左键并拖动"), "North");
        f.add(tf, "South");
        f.setBackground(new Color(120,175,175));
        f.addMouseMotionListener(new InnerMonitor());
        f.addMouseListener(new InnerMonitor());
        f.setSize(300, 200);
        f.setVisible(true);
    }

    public static void main(String args[]) {
        TestInner t = new TestInner();
    }

    private class InnerMonitor implements MouseMotionListener,MouseListener {
        public void mouseDragged(MouseEvent e) {
            String s = "鼠标拖动到位置（" + e.getX() + "," + e.getY() + ")";
            tf.setText(s);
        }

        public void mouseEntered(MouseEvent e) {
            String s = "鼠标已进入窗体";
            tf.setText(s);
        }

        public void mouseExited(MouseEvent e) {
            String s = "鼠标已移出窗体";
            tf.setText(s);
        }
        public void mouseMoved(MouseEvent e) { }
        public void mousePressed(MouseEvent e) { }
        public void mouseClicked(MouseEvent e) { }
        public void mouseReleased(MouseEvent e) { }
    }//end of Inner class
}//end of Outer class
```



```
import java.awt.*;
import java.awt.event.*;
/*实现窗口关闭*/
public class TestWindowClose {
  public static void main(String args[]) {
    new MyFrame55("MyFrame");
  }
}

class MyFrame55 extends Frame {
  MyFrame55(String s) {
    super(s);
    setLayout(null);
    setBounds(300, 300, 400, 300);
    this.setBackground(new Color(204, 204, 255));
    setVisible(true);
    // this.addWindowListener(new MyWindowMonitor());

    this.addWindowListener(new WindowAdapter() {//用的是.不同于之前的特定的组件倾听者
      public void windowClosing(WindowEvent e) {
        setVisible(false);
        System.exit(-1);
      }
    });

  }
}

//这个程序等于下面这句话
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);//窗口退出
```

三、Swing工具包（现阶段）

> javax.swing.*

Swing与awt的区别：

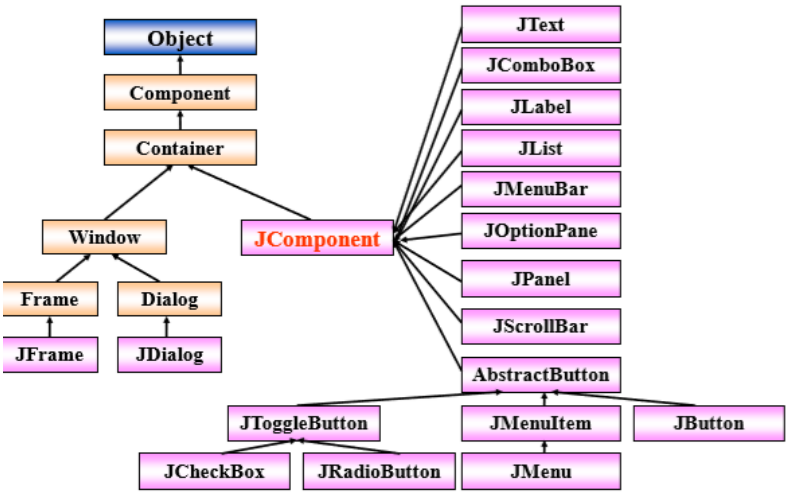
	awt	Swing
实现原理不同	AWT的图形函数与操作系统提供的图形函数有着一一对应的关系。AWT 的图形功能是各操作系统图形功能的“交集”。因为AWT是依靠本地方法来实现功能的，所以AWT控件称为“重量级控件”。（依赖操作系统·的高层用户界面模块）	Swing，不仅提供了AWT 的所有功能，还用纯粹的Java代码对AWT的功能进行了大幅度的扩充。Swing控件在各平台通用。因为Swing不使用本地方法，故Swing控件称为“轻量级控件”。（调用图形子系统的底层例程） java.lang.Object └java.awt.Component └java.awt.Container └java.awt.Panel └java.applet.Applet └javax.swing.JApplet
	基于本地方法的C/C++程序，其运行速度比较快	基于AWT的Java程序，其运行速度比较慢。
	一个嵌入式应用，目标平台的硬件资源往往非常有限，而应用程序的运行速度又是项目中至关重要的因素。在这种矛盾的情况下，简单而高效的AWT当然成了嵌入式Java的第一选择。	在标准版的Java中则提倡使用Swing，也就是通过牺牲速度来实现应用程序的功能。

Swing优势：① 双缓冲区：使用双缓冲技术能改进频繁变化的组件的显示 效果。与AWT组件不同，JComponent组件默认双缓冲区， 不必自己重写代码

② 可插入L&F：每个Jcomponent对象有一个相应的 ComponentUI对象，为它完成所有的绘画、事件处理、决 定尺寸大小等工作。

- 由用户在运行时选择（可选）的“可插入外观”(L&F)。每个组件的外观都由 *UI 委托* 提供，UI 委托是一个继承自 `ComponentUI` 的对象。 具体怎么用有待寒假继续学习！！

（一）基本类结构



（二）容器与组件层次

顶级容器：JFrame框架、JDialog对话框、JApplet小 程序

中间级容器：JPanel面板

原子组件：JButton按钮和JLabel标签等

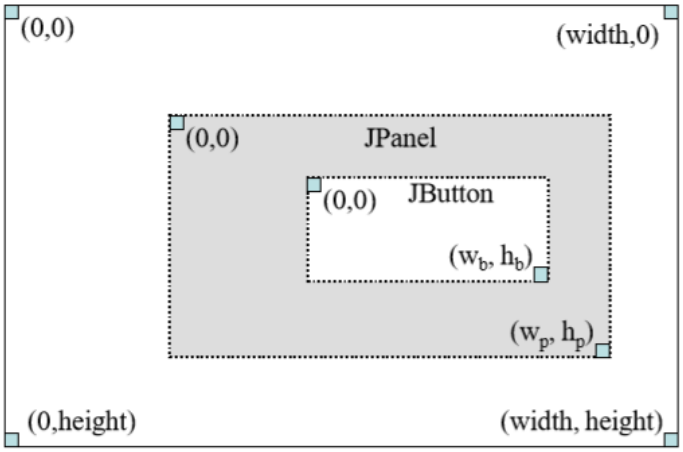
实际中，虽然顶级容器—中间级容器—底层组件的三 层结构不会改变，但每层中可能会分出更多更细的级 别，以满足实际需要

四、Graphics图形绘制

> java.awt.Graphics(是awt里面的一个类哦)

Java自定义的图形绘制，主要是使用Graphics(图形环境)参 数，包括：字体、颜色等绘图元素 。像JButton这样的图形组件不用考虑绘制的问题，而用户自定义的图形组件要考虑自己绘制的问题paint(),repaint() 还没学完

（一）坐标系



（二）在JPanel上绘图

JPanel比JComponent更适合用于用户自定义的 图形组件

• 用户可自由在JPanel上绘制所需的图形效果

• 同时也可以将JPanel加入容器、移动位置、改变 大小等等 • JPanel自带双缓冲功能，避免动画过程中的闪烁

1.如何在JPanel上绘图

步骤:

需要在JPanel上改写，因此继承JPanel类

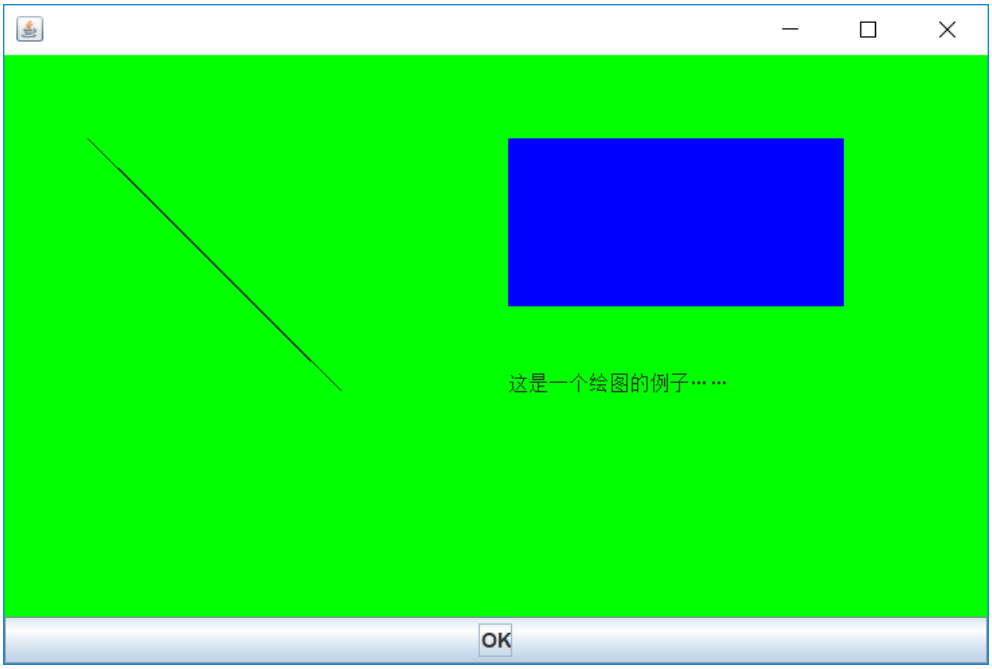
JComponent类中有一个绘制组件的方法 paintComponent(Graphics g)，我们需要重 写(注意super的用法)

注意paintComponent()方法被调用的时机

使用Graphics对象时，注意保存现有环境， 使用完再恢复这是什么意思？？

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

class MyPanel extends JPanel { // 继承JPanel类
    int i = 0;

    public void paintComponent(Graphics g) { // 绘制组件
        System.out.println("paint方法第" + (++i) + "次调用");
        super.paintComponent(g); //重点：调用JPanel父类的方法构造了一个Graphics对象吗？
        g.drawLine(50, 50, 200, 200);
        g.drawString("这是一个绘图的例子……", 300, 200);
```

```
        //Color c = g.getColor();
        //g.setColor(Color.BLUE);
        //g.fillRect(300, 50, 200, 100);
        //g.setColor(c);

        // Color c = g.getColor();
        g.setColor(Color.BLUE);// 设置画笔颜色
        g.fillRect(300, 50, 200, 100);
        // g.setColor(Color.BLUE);
    }
}

public class MyPanelTest {
    public MyPanelTest() {
        JFrame frame = new JFrame();
        frame.setBounds(100, 100, 600, 400);
        frame.setLayout(new BorderLayout());
        MyPanel mp = new MyPanel();//new了一个面板
        mp.setBackground(Color.GREEN);
        JButton bt = new JButton("OK");
        frame.add(mp, BorderLayout.CENTER);
        frame.add(bt, BorderLayout.SOUTH);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);//窗口退出
        frame.setVisible(true);
    }

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



特别值得注意的是java中窗口的移动意味着重新调用函数再画一遍

五、总结

总的来说，gui的学习主要把握组件容器的关系。我们先构思好布局，然后选择采用哪种相对布局。组件的位置大小都布局好后，开始对相应的组件加监听器和事件处理方法（接口ActionListenr中actionPerformed方法的实现）。其中绘图也是布局的一种实现方式