

# 第九章



# 图形程序设计



南京农业大学  
谢忠红



# 图形界面设计

---

- **1.组件和容器**
- **2.GUI(图形用户界面)布局管理器**
- **3.事件处理**
- (举例：一个计算器的图形界面的程序)



# 1 组件和容器

---

- **Java**库提供两种类型的组件
- 第一代组件
- **Java.AWT**组件(**Abstract Window Toolkit**)
- (适合**Applet**小程序)
- 第二代组件
- **Javax.Swing**组件

# ★ AWT的核心内容是组件和容器

- ★ 组件通常为图形用户界面中的可见部分，例如按钮（button）和标签（label）等
- 通过**add( )**方法可将组件加入容器并显示出来。
- 容器是图形用户界面中容纳其他组件的部件，一个容器中可容纳组件或容器。
- **常见容器**：对话框（**Dialog**）、框架（**Frame**）、窗口（**Window**）和面板（**Panel**）。



# 组件的定位

---

- 容器中组件的位置由容器的布局管理器（**Layout Manager**）决定。

## 组件和容器的关系

- 组件自身不能构成独立的图形界面，必须放到容器对象中。

# 基本组件和容器的类层次

**JComponent**

**JContainer**

**JButton**

**JMenu**

**JTextfield**

**JCheckbox**

**JPanel**

**JApplet**

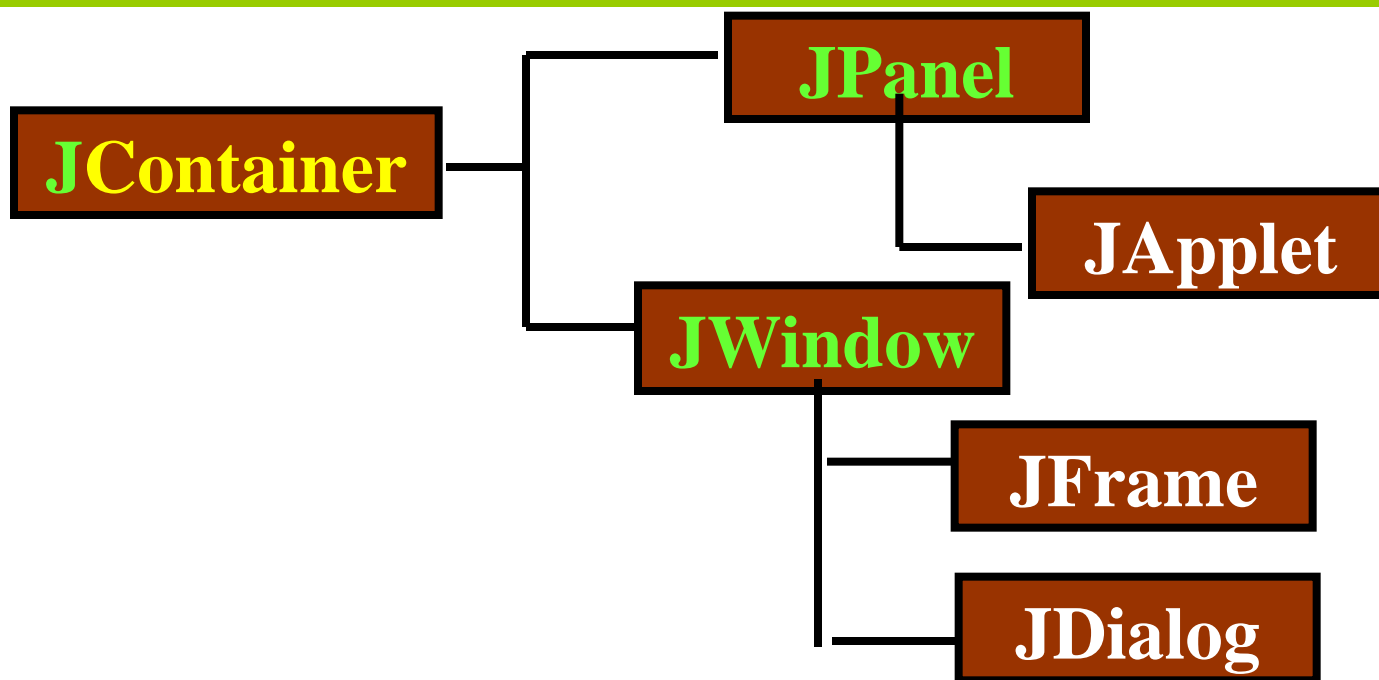
**JWindow**

**JFrame**

**JDialog**

# 常见的容器

- 常用的容器有：对话框（**JDialog**）、框架（**JFrame**）、窗口（**JWindow**）和面板（**JPanel**）。



# 创建简单框架Frame

- 框架（**Frame**）类是**Window**类的子类，它是一种带标题框并且可以改变大小的窗口。
- 构造方法**Frame(String)**可以创建**Frame**的实例，它带有标题框，构造方法中的**String**型参数指定了标题内容。





```
import java.awt.*;  
public class MyFrame extends Frame{  
    public static void main(String args[]){  
        MyFrame fr = new MyFrame("HelloOutThere !");  
        fr.setSize(400,200);  
        fr.setBackground(Color.blue);  
        fr.setVisible(true);  
    }  
    public MyFrame (String str){  
        super(str);  
    }.....  
}
```



这里调用来自  
**Component**类的  
**setSize()**方法


- 
- 使用从 **Component** 类继承过来的 **setSize( )** 方法可以改变 **Frame** 实例的大小。
  - 必须调用 **setVisible( )** 方法和 **setSize()** 方法才能使 **Frame** 的实例可见。

# 创建面板

- 面板（**Panel**）与框架类似，也是一种容器，可以容纳其他**GUI**组件。
- 当一个**Panel**对象被创建之后，使用**Container**类的**add( )**方法将它加入到某个**Window**对象或**Frame**对象中。

# 面板示例

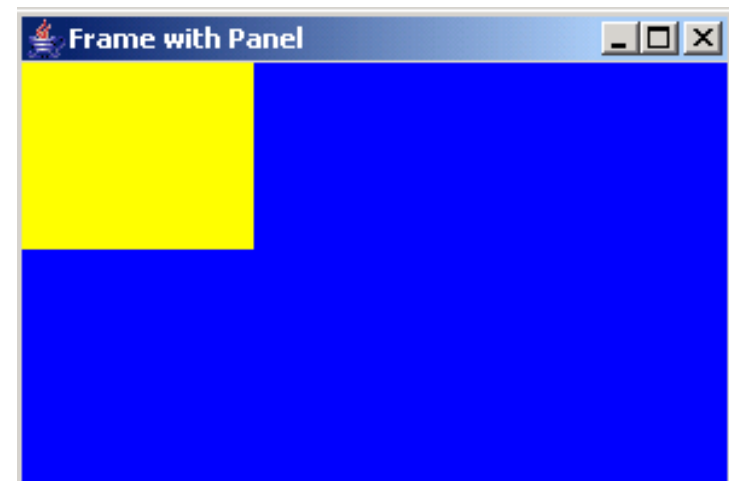
```
import java.awt.*;  
public class FrameWithPanel extends Frame {  
    public FrameWithPanel (String str) {  
        super(str);  
    }  
    public static void main(String args[]) {  
        FrameWithPanel fr = new  
            FrameWithPanel("Frame with Panel");  
        Panel pan = new Panel( );  
    }  
}
```



构造函数



- **fr.setSize(300,200);**
- **fr.setBackground(Color.blue);**
- **fr.setLayout(null);**
- **pan.setSize(100,100);**
- **pan.setBackground(Color.yellow);**
- **fr.add(pan);**
- **fr.setVisible(true);**
- **}**
- **}**



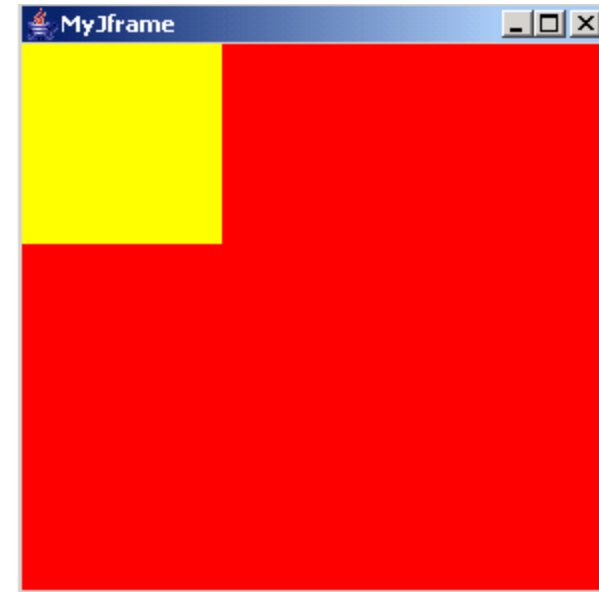
# javax.swing.JFrame



■ **JFrame = Frame + Container**

- 例程:
- `import java.awt.*; import javax.swing.*;`
- `public class MyJFrame extends JFrame{`
- `public MyJFrame (String str) {`
- `super(str);`
- `}`
- `public static void main(String args[]) {`
- `MyJFrame fr = new MyJFrame("MyJframe");`

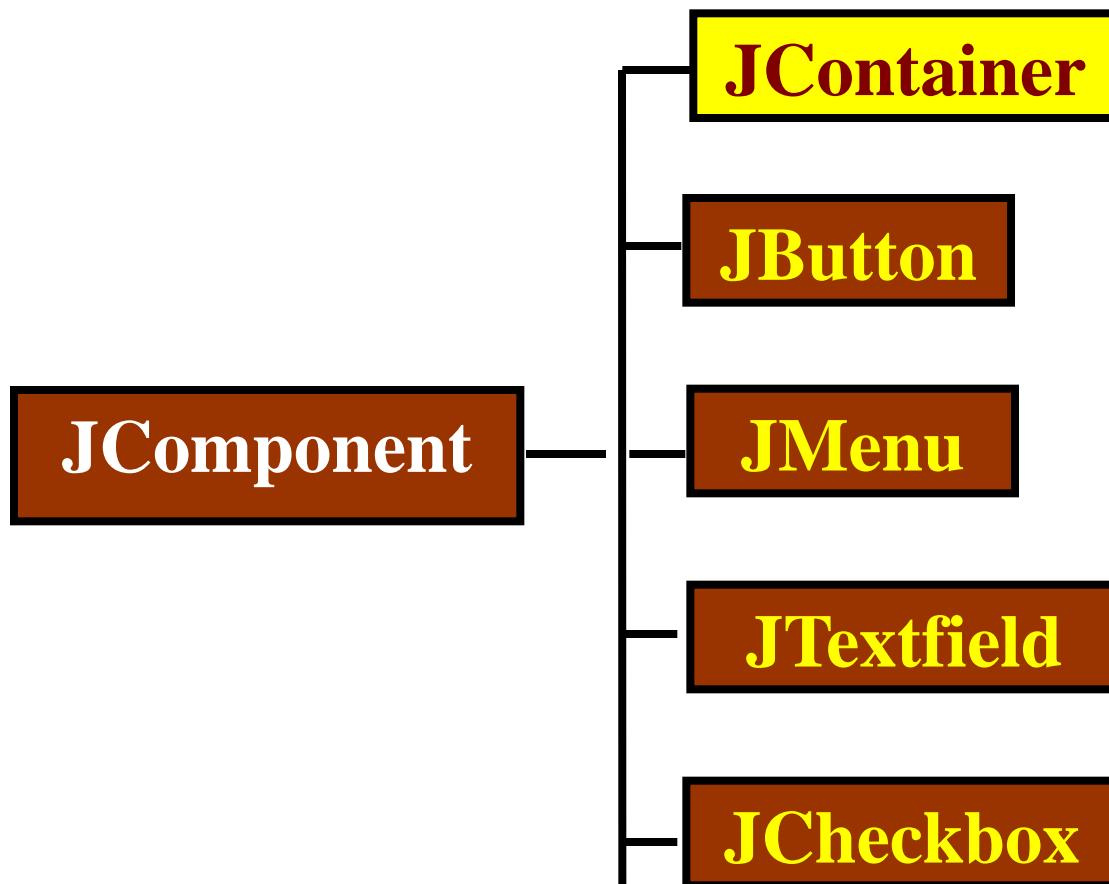
- **Container con=fr.getContentPane();**
- **// 获得面板**
- **fr.setSize(300,300);**
- **con.setBackground(Color.red);**
- **JPanel pan = new JPanel( );**
- **pan.setSize(100,100);**
- **pan.setBackground(Color.yellow);**
- **con.setLayout(null);**
- **con.add(pan);**
- **fr.setVisible(true);**
- **}**
- **}**





# 常见的组件

---







# JButton组件

---

## ■ JButton类

构造函数: **JButton()** ;

**JButton(String label)**

常见的方法:

**void setLabel(String Label)** // 设置按钮的标记

**String getLabel()** // 获取按钮的标记



确定

取消

```
import javax.swing.*;  
import java.applet.Applet;  
public class MyJButton extends Applet{  
    public void init()  
    { JButton b1=new JButton("按钮1");  
        JButton b2=new JButton();  
        b2.setLabel("按钮2");  
        add(b1);    add(b2); }  
}
```

**<APPLET CODE="MyJButton.class"**

**width=300 height=400> </APPLET>**



# 标签 JLabel类

---

- **作用：**一般显示提示信息，用户不能修改
- **构造函数：** `public JLabel()`
- `JLabel(String s)`
- `JLabel(String s,int alignment)`
- **主要方法：**
- `String getString()` // 返回标签内容
- `Void setText(String s)` // 设置标签文本
- `String getAlignment()` // 返回标签得对齐方式



# 文本框JTextField类

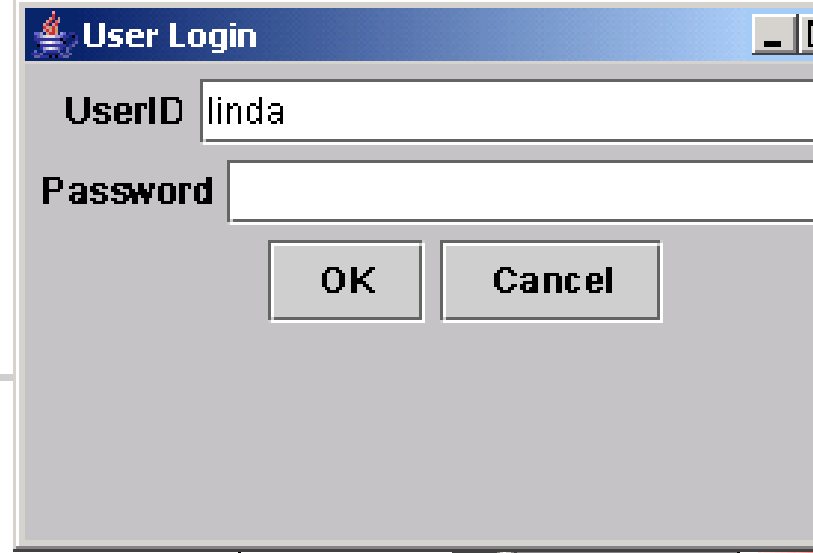
---

- 作用： 生成一个单行文本域
- 构造函数:**JTextField()**
- **JTextField(String s)**
- **JTextField(int col)**
- **JTextField(String s,int col)**
- 主要方法:
- **void setText (String t)**
- **String getText()**//获取响应字符



## 综合例子

- *import java.awt.\*;*
- *import javax.swing.\*;*
- **public class JLogin{**
- **public static void main(String args[]){**
- **JFrame f=new JFrame("User Login");**
- **Container con=f.getContentPane();**
- **con.setBackground(Color.lightGray);**
- **con.setLayout(new FlowLayout());**
- **JLabel lb1=new JLabel("UserID");**
- **JTextField T1=new JTextField("linda",20);**



- **JLabel lb2=new JLabel("Password");**
- **TextField T2=new TextField(20);**
- **Button b1=new Button("OK");**
- **Button b2=new Button("Cancel");**
- **con.add(lb1);            con.add(T1);**
- **con.add(lb2);            con.add(T2);**
- **con.add(b1);            con.add(b2);**
- **f.setSize(280,150);**
- **f.setVisible(true);**
- **}**
- **}**



# 文本域JTextArea

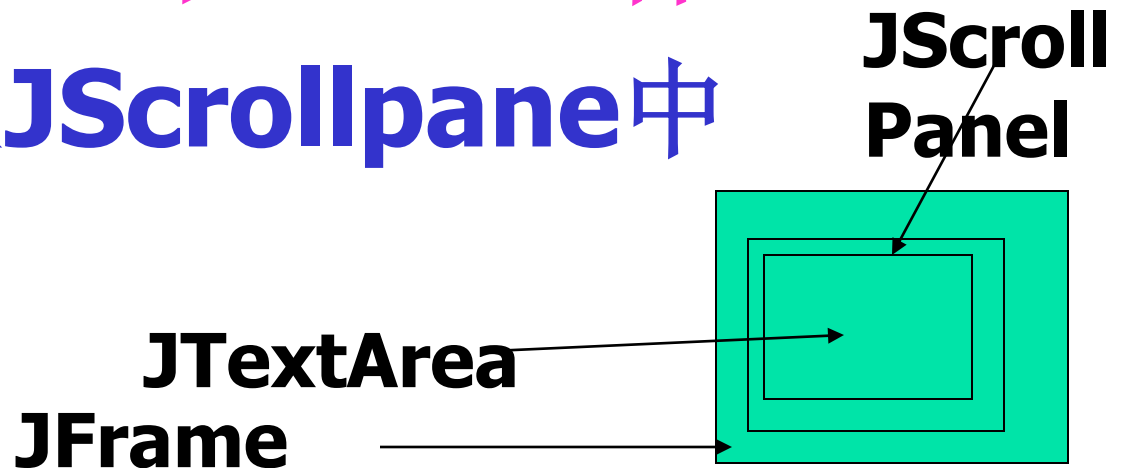
---

- **swing**的**JTextArea**类可生成一个多行的文本域，内容超出显示范围时，具有滚动显示的功能。
- **TextArea**类的构造函数：
  1. **public JTextArea( String text )**
  2. **public JTextArea( int rows, int columns )**
  3. **public JTextArea( String text, int rows, int columns )**

## ■ **TextArea**类的常用方法:

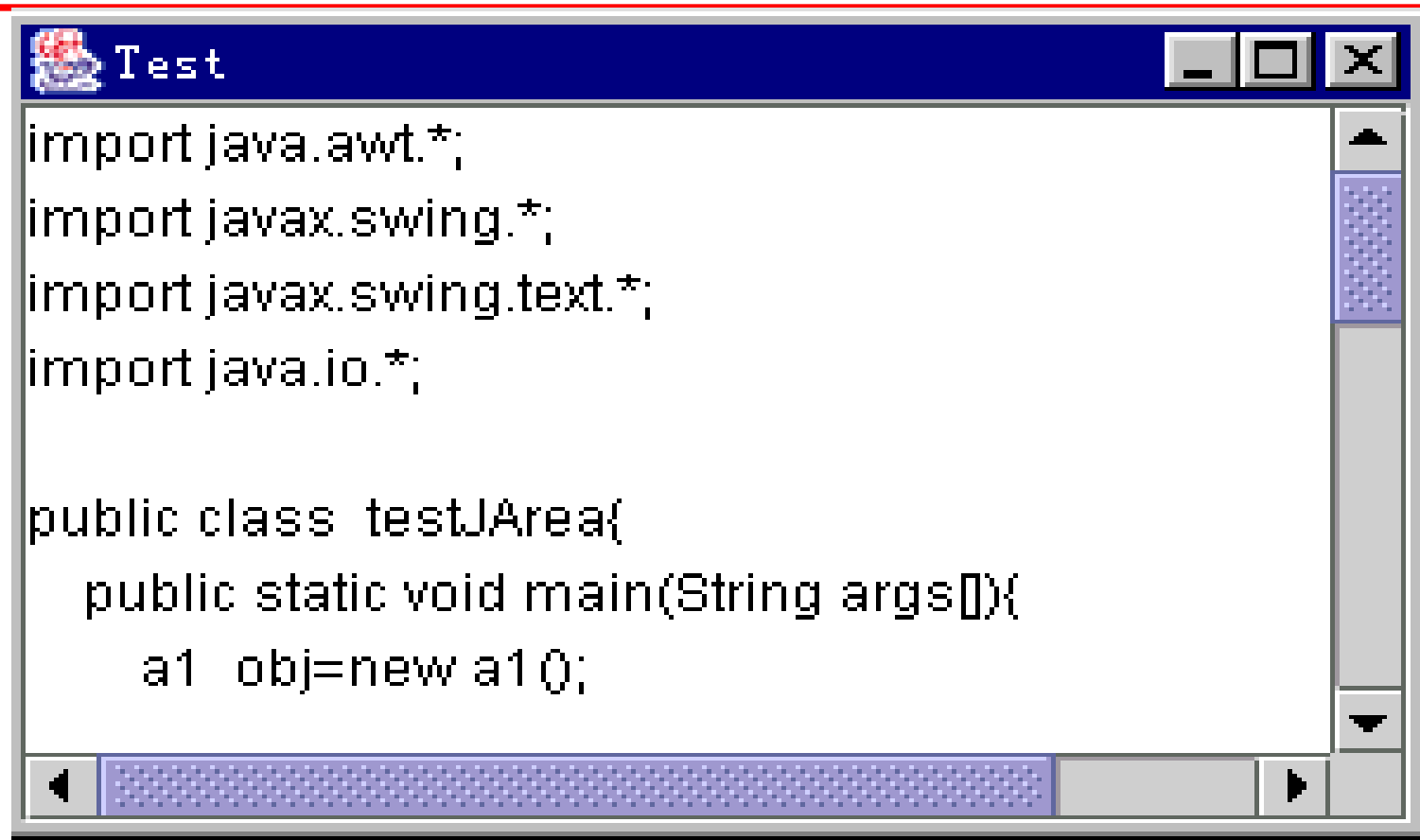
1. **public void append( String str )**
2. **public int getColumns( )**
3. **public int getRows( )**
4. **public void insert( String str, int pos )**
5. **public void replaceRange( String str, int start, int end )**

**注意:** **JTextArea**和**JList**组件  
必须放在**JScrollPane**中





- 程序举例:
- 编写**testJTextArea.java**程序,通过文件输入流将**该文件**从硬盘读入内存并显示在多行文本框中



```
import java.awt.*;  
import javax.swing.*;  
import javax.swing.text.*;  
import java.io.*;  
  
public class testJArea{  
    public static void main(String args[]){  
        a1 obj=new a1();
```

- *import java.awt.\*; import javax.swing.\*;*
- *import javax.swing.text.\*; import java.io.\*;*
- public class testJTextArea{
- public static void main(String args[ ]){
- testJTextArea obj=new testJTextArea ( );
- obj.testArea( );   }
- public void testArea( ){
- JFrame frame=new JFrame("Test");
- Container con=frame.getContentPane();
- JTextArea **ta**=new JTextArea(5,30);
- // 支持滚动的容器组件
- **JScrollPane** pane=**new JScrollPane(ta);**
- con.add( **pane**, BorderLayout.CENTER);

- `try{     // 打开文件`
- `FileReader Rin=new`
- `FileReader("./testJTextArea.java");`
- 
- `ta.read(Rin , null );`
- `}catch(Exception e){`
- `System.exit(0);`
- `}`
- `frame.setSize(200,200);// 设置框架大小`
- `frame.setVisible(true );                         // 显示框架`
- `}`
- `}`

[转到布局管理器](#)

## 复选框javax.Swing.JCheckBox

### ■ 构造函数:

1. public JCheckBox(String S)
2. public JCheckBox(String S,booleanselected)

## 单选框javax.Swing.JRadioButton

### ■构造函数:

1. public JRadioButton(String S)
2. public JRadioButton(String S ,  
boolean selected)



# JavaSwing.ButtonGroup类

---

- **作用：** 将相互独立的单选框构成一组，

- **常用方法：**

1. `public ButtonGroup( )`

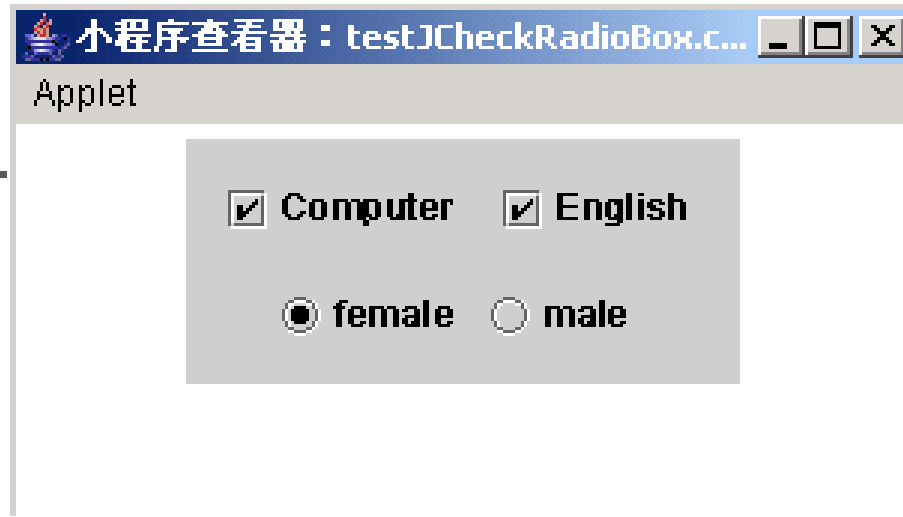
2. `public void add( AbstractButton b):`

**作用：** 将一个单选按钮加到ButtonGroup组件中。

3. `public int getButtoncount( )`

4. `public void remove( AbstractButton b)`

# 两种选择框举例



```
import java.awt.*; import java.applet.Applet;  
import javax.swing.*;  
public class testJCheckBoxBox extends  
Applet {  
    JCheckBox jcb1,jcb2;  
    JRadioButton jr1,jr2;
```

```
ButtonGroup group1;  
JPanel p1;  
public void init() {  
    JPanel pCh=new JPanel();  
    JPanel pRa=new JPanel();  
    p1=new JPanel();  
  
    jcb1=new JCheckBox("Computer ",true);  
    jcb2=new JCheckBox("English ",true);  
  
    jr1=new JRadioButton("female",true);  
    jr2=new JRadioButton("male ",false);
```

```
ButtonGroup group=new ButtonGroup();  
group.add(jr1); group.add(jr2);
```

```
// 不能将group加入到applet。
```

```
p1.setLayout(new GridLayout(2,1));
```

```
pCh.add(jcb1); pCh.add(jcb2);
```

```
pRa.add(jr1); pRa.add(jr2);
```

```
p1.add(pCh);
```

```
p1.add(pRa);
```

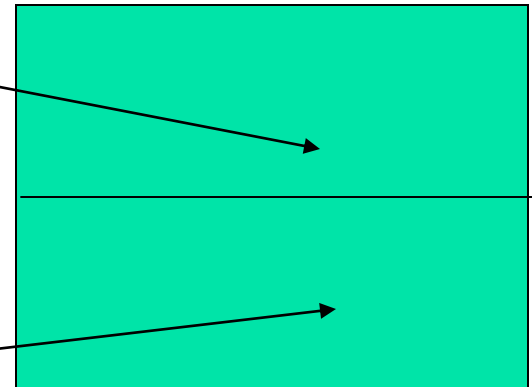
```
add(p1);
```

```
}
```

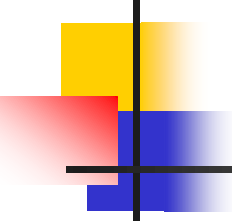
```
}
```

**PCh**

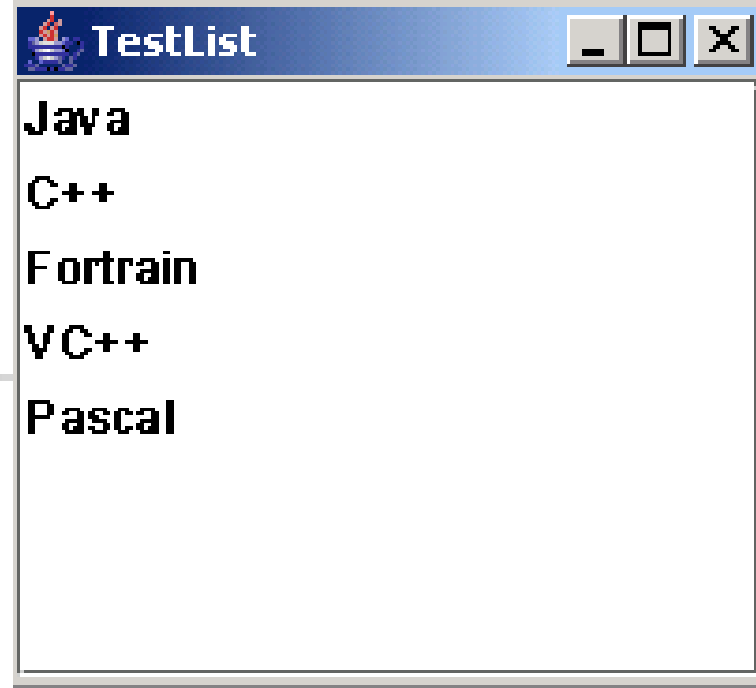
**PRa**







# 列表框:JList类



- **(1)Jlist**组件必须放在**JScrollPane**中, 否则不具滚动功能。
- **(2)构造函数**
- **public JList()**
- **Public Jlist(Object object[])**

- *import java.awt.\*;import javax.swing.\*;*
- public class testJList{
- public static void main(String args[]){
- testJList obj=new testJList(); obj.testJList(); }
- public void testJList(){
- String book[]={"Java","C++","Fortrain",
- "DataBase","OperationSystem"};
- JList list1=new JList(*book*);
- JFrame frame=new JFrame("TestList");
- Container con=frame.getContentPane();
- JScrollPane sp=new JScrollPane(*list1*);
- con.setLayout( new BorderLayout());
- con.add(sp, "Center");
- frame.setSize(50,100);frame.setVisible(true);}}

# 菜单制作

JMenuBar类

JMenu类

JMenuItem类

The screenshot shows the Microsoft PowerPoint application window titled "Microsoft PowerPoint - [第九章图形界面设计.ppt]". The "文件(F)" menu is open, displaying options such as "新建(N)...", "打开(O)...", "保存(S)", "另存为(A)...", "页面设置(U)...", "打印(P)...", and "发送(D)". Below these, a list of recent files is shown, including "1 第九章图形界面设计.ppt", "2 第七章集合列表.ppt.ppt", "3 information第四章数组和字符串1.ppt", and "4 K:\...\information第五章面向对象的高级特征.p...".

Yellow arrows point from the text labels to specific parts of the interface: one from "JMenuBar类" to the menu bar, one from "JMenu类" to the "文件(F)" menu, and one from "JMenuItem类" to the list of recent files.

On the right side of the screenshot, a list of Java Swing classes is displayed, with the first three items underlined:

- MenuBar类
- menu类
- menuItem类

At the bottom left, a list of Java Swing classes is shown, with the first three items underlined:

- 对话框JDialog类
- 对话框JDialog类
- 菜单制作

Below the "菜单制作" item, the class `.JMenuBar类` is listed.

- 简单的菜单程序

- **import javax.swing.\*;**

- **public class testJMenu extends JFrame{**

- **JMenuBar bar;**

- **JMenu fileMenu;**

- **JMenuItem InputMItem,modifyMItem;**

- **JMenuItem queryMItem,deleteMItem;**

- **testJMenu(String s){**

- **super(s);**

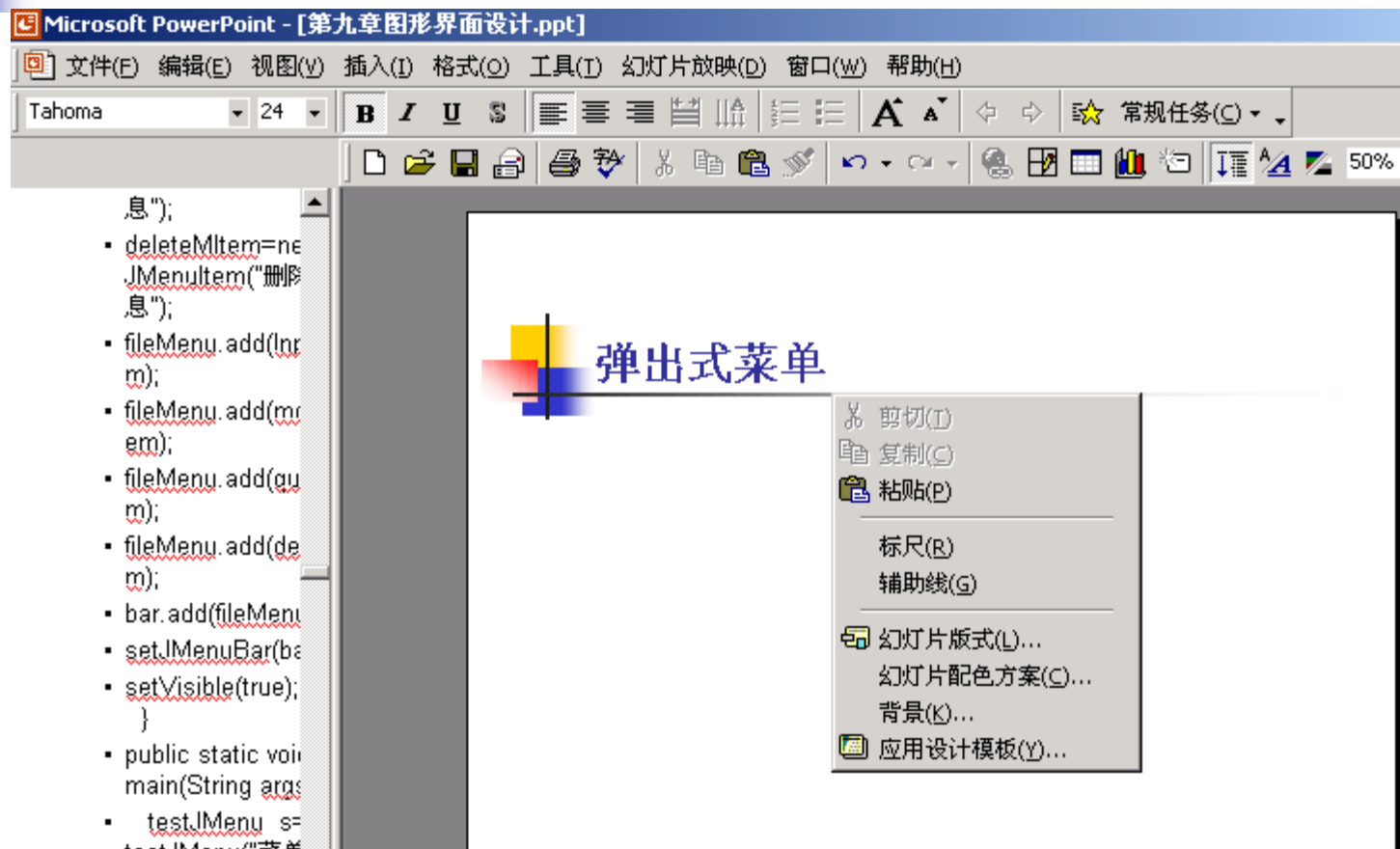
- **bar= new JMenuBar();**

- **fileMenu = new JMenu("菜单选项");**

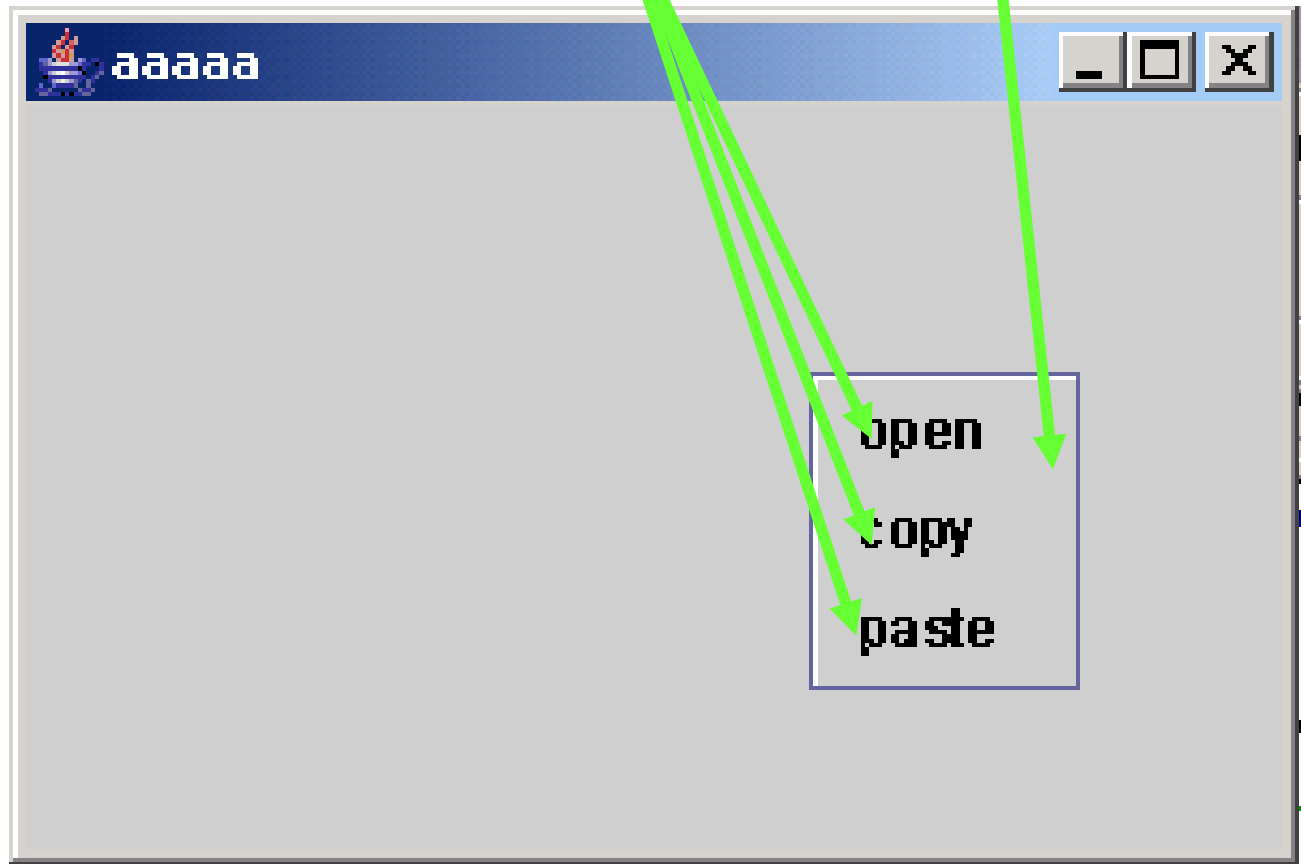
-

- **Input Item =new JMenuItem("录入学生信息");**
- **modifyMItem =new JMenuItem("修改学生信息");**
- **queryMItem=new JMenuItem("查询学生信息");**
- **deleteMItem=new JMenuItem("删除学生信息");**
- **fileMenu.add( InputMItem );**
- **fileMenu.add( modifyMItem );**
- **fileMenu.add( queryMItem );**
- **fileMenu.add( deleteMItem );**
- **bar.add( fileMenu );**
- **setJMenuBar( bar );**
- **setVisible(true);** }
- **public static void main(String args[]){**
- **testJMenu s=new testJMenu("菜单窗体");}**
- **}**

# 弹出式菜单




**JPopupMenu**  
**JMenuItem**



## ■ 弹出式菜单的简单程序

```
■ import javax.swing.*; import java.awt.*;
■ public class pop2 {
■     JFrame      f;
■     JPopupMenu  popUp;
■     JMenuItem   openMItem, copyMItem,
■                 pasteMItem;
■     public pop2() {
■         f = new JFrame("弹出菜单");
■         popUp = new JPopupMenu("File");
■         openMItem = new JMenuItem("open");
■         copyMItem = new JMenuItem("copy");
■         pasteMItem = new JMenuItem("paste");
■     }
■ }
```





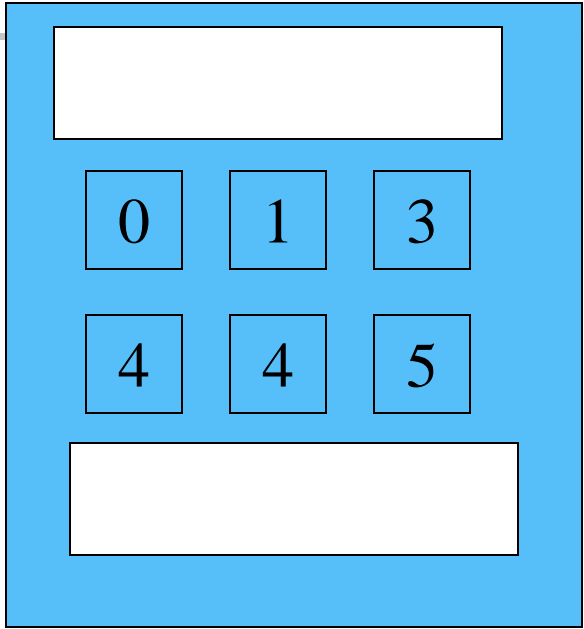
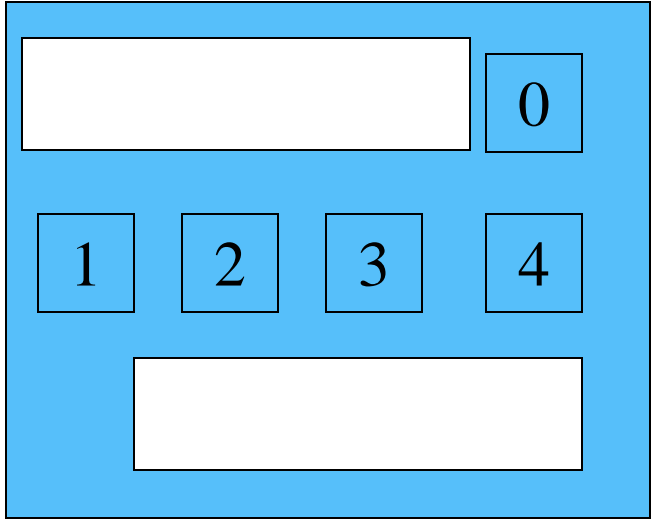
- **popUp** .add(**openMItem**);
- **popUp** .add(**copyMItem**);
- **popUp** .add(**pasteMItem**);
- **f.setSize(300,200);**
- **f.setVisible(true);**
- **popUp.show(f,100,100);**
- **}**
- **public static void main(String args[]){**
- **pop2 s = new pop2();**
- **}**
- **}**

**转到3事件处理**

# 第九章



## 二、GUI布局管理器



## 一个简单的例子

```
import java.awt.*;  
public class ExGui {  
    private Frame    f;  
    private Button   b1;  
    private Button   b2;  
    public static void main(String args[]){  
        ExGui that = new ExGui( );  
        that.go( );  
    }  
}
```



```
public void go( ) {  
    f = new Frame ("GUI example");  
    f.setLayout( new FlowLayout( ) );  
    b1 = new Button("Press me");  
    b2 = new Button("Don't press Me");  
    f.add(b1);  
    f.add(b2);  
    f.pack( );  
    f.setVisible(true);  
}  
}
```

**Java**语言中包含以下几种布局管理器：

- **FlowLayout** (流布局)

**Panel**类和**Applet**类的默认布局管理器。

- **BorderLayout** (边框布局)

**Window**类、**Dialog**类和**Frame**类的默认布局管理器。

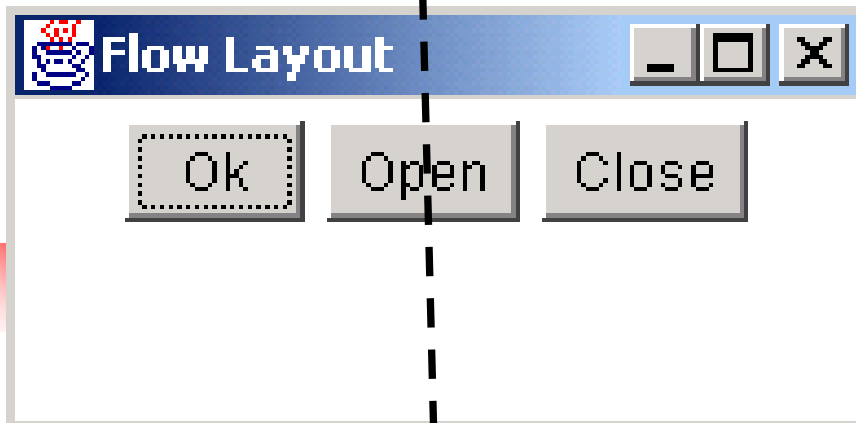
- **GridLayout** (网格布局)

- **CardLayout** (卡片布局)

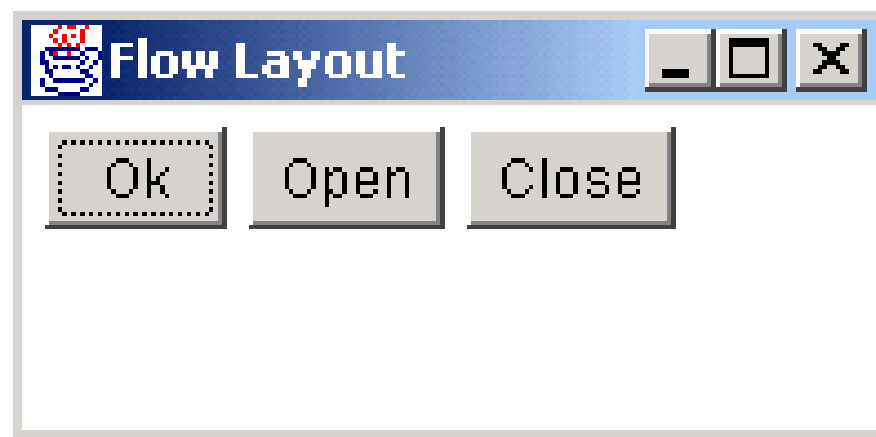
- **GridBagLayout** (网格包布局)

## **FollowLayout( Applet、Panel和Jpanel);**

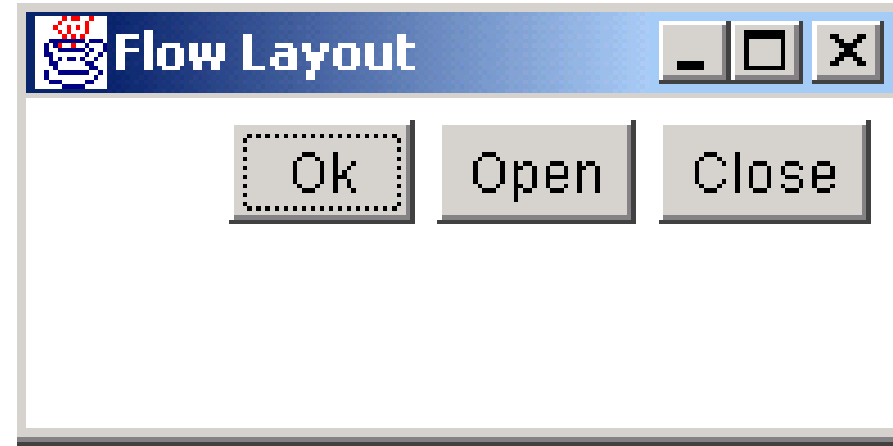




**FlowLayout.CENTER**



**FlowLayout.LEFT**



**FlowLayout.RIGHT**





## ■ 常用构造函数如下：

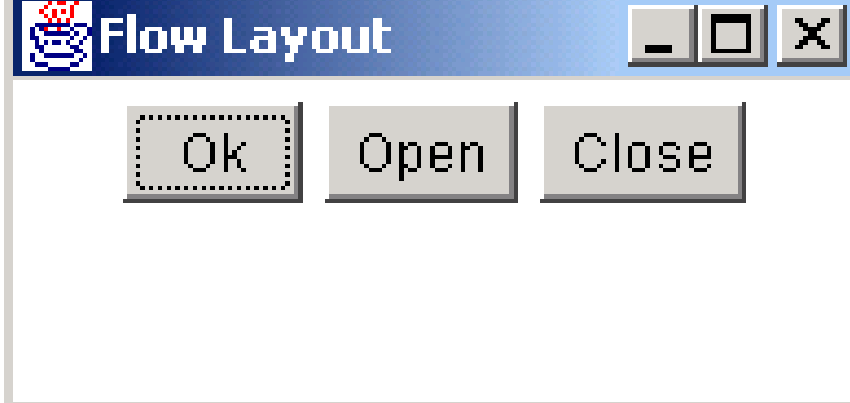
### 1 `public FlowLayout( )`

// 默认居中对齐，垂直和水平间隔为5。

### 2. `public FlowLayout( int align )`

//指定对齐方式的**FlowLayout**构造函数


例： `FlowLayout(FlowLayout.LEFT)`



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

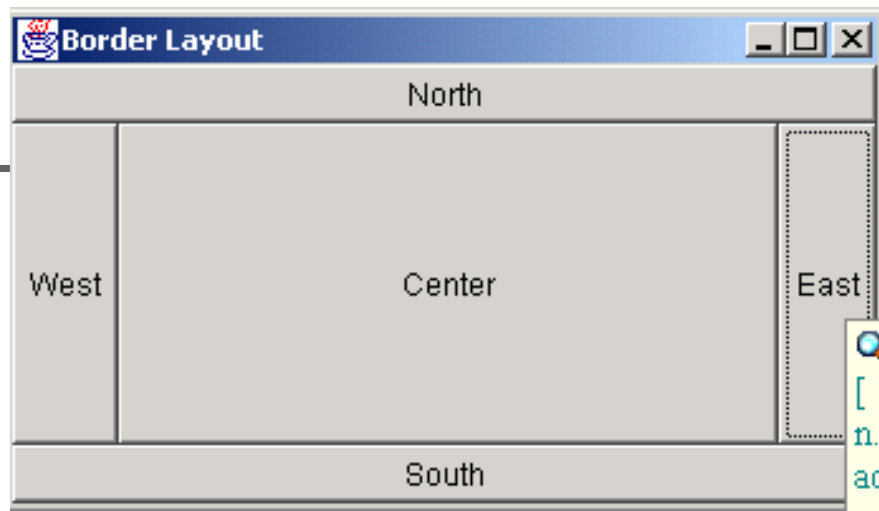
```
public class MyFlow {  
    private Frame f;  
    private Button  
    button1,button2,button3;
```

```
    public static void main(String args[]){  
        MyFlow mflow = new MyFlow( );  
        mflow.go( );  
    }
```



```
public void go( ) {  
    f = new Frame("Flow Layout");  
    f.setLayout(new FlowLayout( ));  
    button1 = new Button("Ok");  
    button2 = new Button("Open");  
    button3 = new Button("Close");  
    f.add(button1);  
    f.add(button2);  
    f.add(button3);  
    f.setSize(100,100);  
    f.setVisible(true);  
}
```

# BorderLayout布局



- **BorderLayout**布局是将空间划分为东、西、南、北、中五个区域；
- 表示为: **"East"、"West"、"South"、**
- **"North"和"Center"。**

## ■ BorderLayout的构造函数如下:

**1. public BorderLayout( )**

// 组件的垂直和水平间隔为0。

**2. public BorderLayout( int hgap, int vgap )**

如: 将一个按钮加到框架的南部

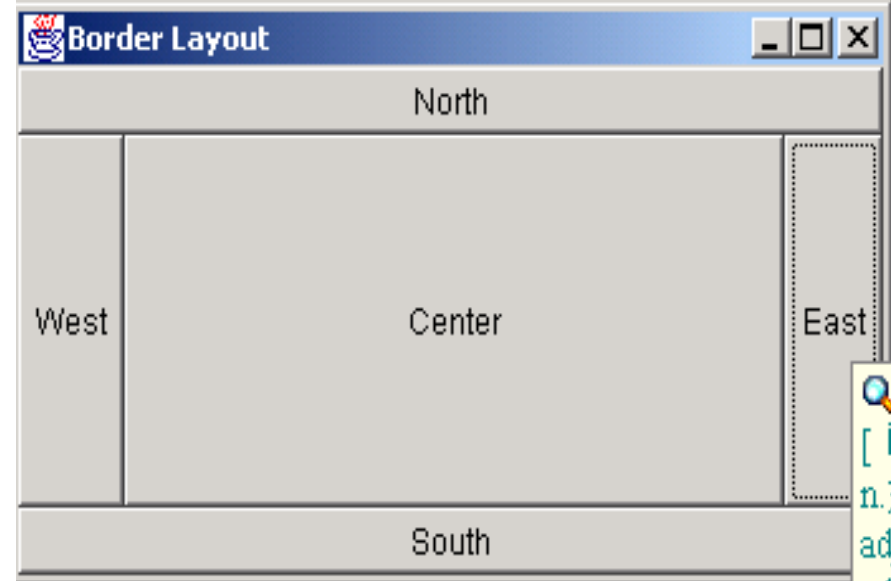
```
f = newFrame("FrameTitle");
```

```
b = newButton("PressMe");
```

```
f.add(b, "South");
```

## 程序举例

```
import java.awt.*;  
public class ExGui2 {  
    private Frame f;  
    private Button be,bw,bn,bs,bc;  
    public static void main(String args[]) {  
        ExGui2 that = new ExGui2( );  
        that.go( );  
    }  
    void go( ) {  
        f = new Frame("Border Layout");  
        be = new Button("East");  
        bs = new Button("South");
```



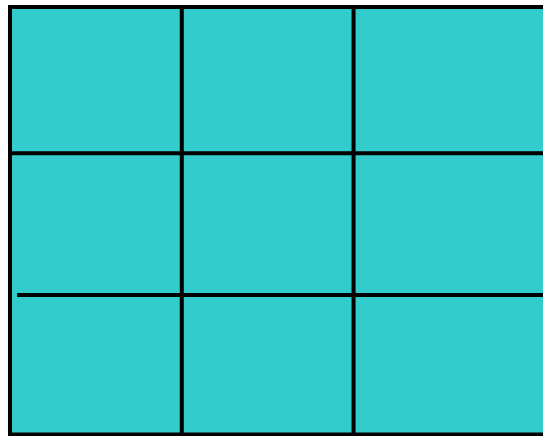
```
bw = new Button("West");  
bn = new Button("North");  
bc = new Button("Center");  
//f.setLayout(new BorderLayout());  
//Frame 默认的布局方式BorderLayout  
f.add(be,"East");  
f.add(bs,"South");  
f.add(bw,"West");  
f.add(bn,"North");  
f.add(bc,"Center");  
f.setSize(350,200);  
f.setVisible(true);
```

```
}
```

```
}
```

# GridLayout布局管理器

- **GridLayout**(网格式的布局管理器)  
将容器空间划分成若干行乘若干列的  
网格，组件从左往右,由上而下依次放  
入其中，每个组件占据一格。





## ■ **GridLayout**类的构造函数:

**1. public GridLayout ( )**

//生成一个行数为**1**的**GridLayout**布局管理器对象。

**2. public GridLayout ( int rows, int cols )**

**3. public GridLayout (int rows, int cols ,  
int hgap, int vgap )**

如:

**new GridLayout(3,2)** , 可以创建一个三行乘两列的布局管理器。

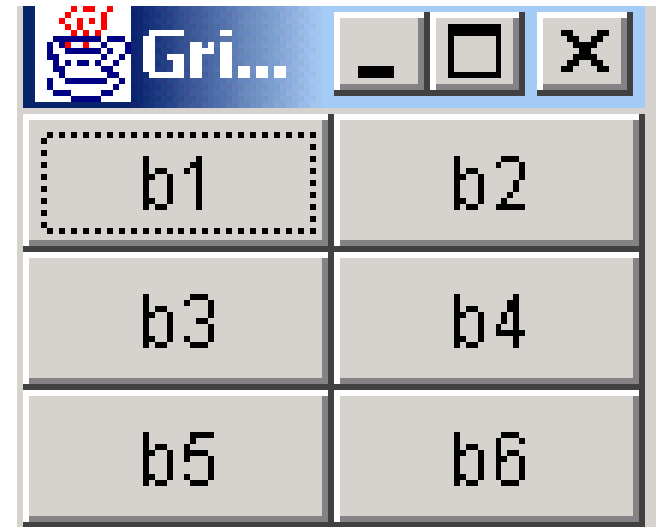
```
import java.awt.*;  
public class GridEx {  
    private Frame f;  
    private Button b1,b2,b3,b4,b5,b6;  
  
    public static void main(String args[]) {  
        GridEx that = new GridEx( );  
        that.go( );  
    }  
    void go( ) {  
        f = new Frame("Gridexample");  
        f.setLayout( new GridLayout(3,2));
```

```
b1 = new Button("b1");  
b2 = new Button("b2");  
b3 = new Button("b3");  
b4 = new Button("b4");  
b5 = new Button("b5");  
b6 = new Button("b6");
```

```
f.add(b1);  
f.add(b2);  
f.add(b3);  
f.add(b4);  
f.add(b5);  
f.add(b6);
```

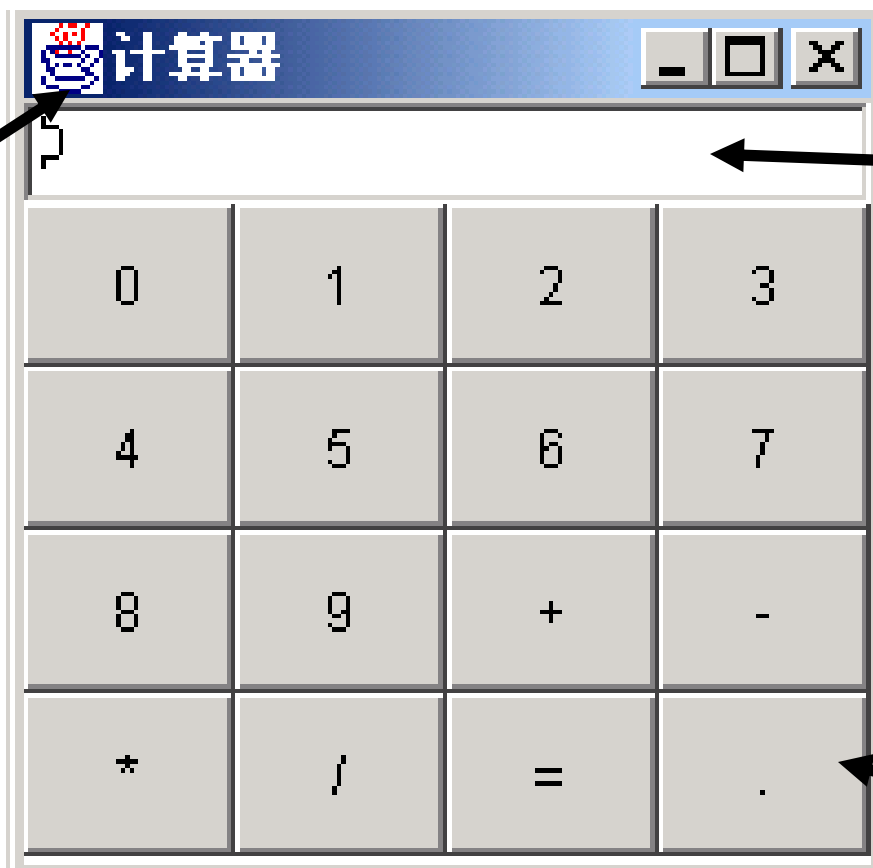
```
f.pack( );  
f.setVisible(true);
```

```
}}
```



```
f.setLayout(new  
GridLayout(3,2,10,10));
```

# 举例1：计算器的界面设计



**Frame**

**TextField**

**Pane**

- **import java.awt.\*;**
- **public class calculate2 {**
- **Button key0,key1,.....key14,key15;**
- **TextField txtAnswer;**
- **Panel p;**
- **Frame f;**
- **public static void main(String args[]){**
- **calculate2 calGUI=new calculate2();**
- **calGUI.go();**
- **}**

```
public void go(){  
    f=new Frame("计算器" ); p=new Panel();  
    txtAnswer=new TextField("0",20);  
    key0=new Button("0");  
        .....  
    key15=new Button(".");  
    p.setLayout(new GridLayout(4,4));  
    p.add(key0);  
        .....  
    p.add(key15);  
    f.setSize(200,200);  
    f.setLayout(new BorderLayout());  
    //默认BorderLayout  
    f.add(txtAnswer,"North");  
    f.add(p, " South");    f.setVisible(true);  
}  
}
```

# 绘图操作:java.awt.Graphics类

作用：进行绘图操作

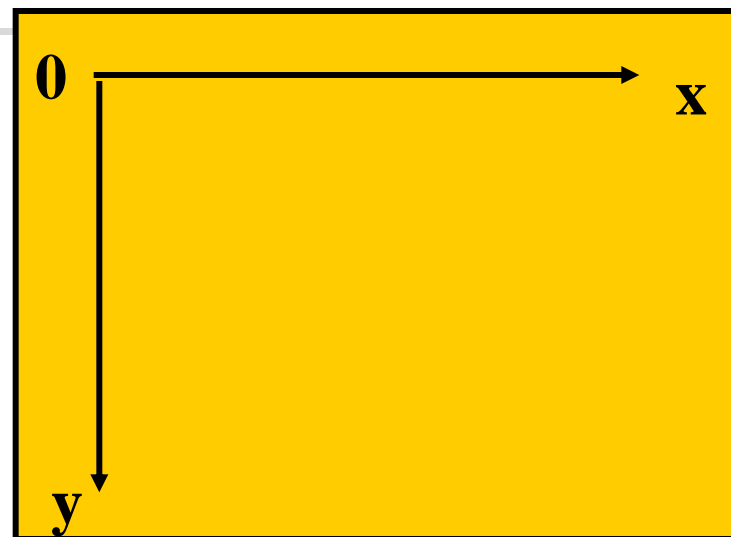
Graphics类的常见方法：

(1)绘制字符串

`drawString( String str, int x, int y)`

(2)绘制线

`drawLine( x1,y1,x2,y2)`



- (3)绘制矩形
- drawRec(x, y, width, height)
- fillRec(x, y, width, height)
- clearRect(x, y, width, height) //清除一个矩形

- (4)绘制椭圆
- drawOval(x, y, width, height)
- fillOval(x, y, width, height)

- (5)颜色管理
- Color getColor() //获取当前颜色
- setColor(Color c)



# 绘图容器: **Java.awt.Comonent**的子类

**例:** **Panel** **Frame** **Applet**

**Comonent**类的两个方法

( 1 ) **void paint(Graphics g){}**

**作用:** 在组件上使用**g**来绘图

( 2 ) **repaint(){}**

**作用:** 自动调用**paint()**方法重新绘图

- 如果声明的类本身就是**Component**的子类则覆盖**paint(方法)**

- `import java.awt.*;`
- `public class testGraphic extends Frame{`
- `public static void main(String args[]){`
- `testGraphic x=new testGraphic();`
- `x.setSize(100,100);`
- `x.setVisible(true);`
- `}`
- `public void paint(Graphics g){`
- `g.drawLine(0,0,100,100);`
- `}`
- `}`

- 使用getGraphics()方法获得Graphics对象
- import java.awt.\*;
- public class testGraphics2 {
- public static void main(String args[]){
- Frame f=new Frame("myframe");
- f.setVisible(true);
- f.setSize(100,100);
- Graphics g=f.getGraphics() ;
- g.drawLine(0,0,100,100);
- }
- }



日期



*Wed Dec 29 23:04:38 CST 2004*

- **import java.awt.\*; import java.util.Date;**
- **public class showdateP extends Panel{**
- **public static void main(String args[]){**
- **Frame f=new Frame ("日期");**
- **showdateP SP =new showdateP();**
- **f.add( SP, "Center");** //SP是一个Panel
- **f.setSize(500,100);**
- **f.setVisible(true);**
- **}**



```
public void paint(Graphics g) {
```

```
    Date timeNow=new Date();
```

- ```
Font msgFont=new  
Font("TimesRoman",Font.ITALIC,30);
```
- ```
g.setFont(msgFont);
```
- ```
g.setColor(Color.blue);  
g.drawString(timeNow.toString(),5,50);
```
- ```
}
```



# 第九章

---

## 三、事件处理

# ■ 1.什么是事件？

- 用户与组件交互而产生的键盘或鼠标等动作。

## • 2.事件源

- 与用户交互的**GUI控件**——例：按钮、文本框等g

### • 3.事件对象？

- 当用户和控件交互产生动作时，AWT事件处理系统会自动产生一个事件对象（描述这次动作的详细信息）
- 常见事件对象类：
- **import java.awt.event.\*;**
- **ActionEvent    KeyEvent    MouseEvent**
- 事件对象产生后应该怎样进行处理呢？
- Java中采用委托处理方式：由添加在控件上的监听器去监听并捕获产生的不同的事件对象然后处理该对象。



## • 4.什么是事件监听器？

- 在某个GUI对象中注册的，用于监视某个事件源 特定事件对象的 *特殊事件监听器*。
- 如：在**Button: Open**需要监听鼠标操作以便及时作出回应，则需要**在Open注册一个鼠标动作事件**的监听器。（如果单击按钮产生**ActionEvent**对象）

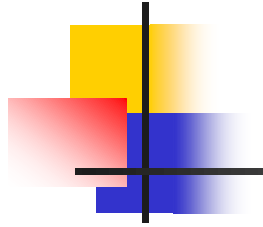
**class BtnClickL implements ActionListener{**

*//定一个监听单击事件的监听器类OneClick*

**public void actionPerformed(ActionEvent e){  
    System.out.println(e.toString());} }**

```
import java.awt.*;  
import java.awt.event.*;  
class TestButton{  
public static void  
main(String args[]){  
    TestButton t=new  
        TestButton();  
    t.go();  
}
```

```
Void go(){  
Frame f=new Frame("Cas");  
Button b1=new Button("OK");  
f.add(b1);  
BtnClickL hear=new BtnClickL ();  
b1.addActionListener(hear);  
f.setSize(100,100);  
f.setVisible(true)} }
```



- **java.awt.event.ActionEvent[ACTION\_PERFORMED,cmd=OK,when=1258270797515,modifiers=] on button0**
- **java.awt.event.ActionEvent[ACTION\_PERFORMED,cmd=OK,when=1258270798437,modifiers=] on button0**
- **java.awt.event.ActionEvent[ACTION\_PERFORMED,cmd=OK,when=1258270798968,modifiers=] on button0**

# 事件处理的三个步骤

1. 定义一个监视器类来，实现事件处理接口。

- **class BtnClickL implements ActionListener{**
- **...}**

2. 完成事件处理接口中定义的各个方法。

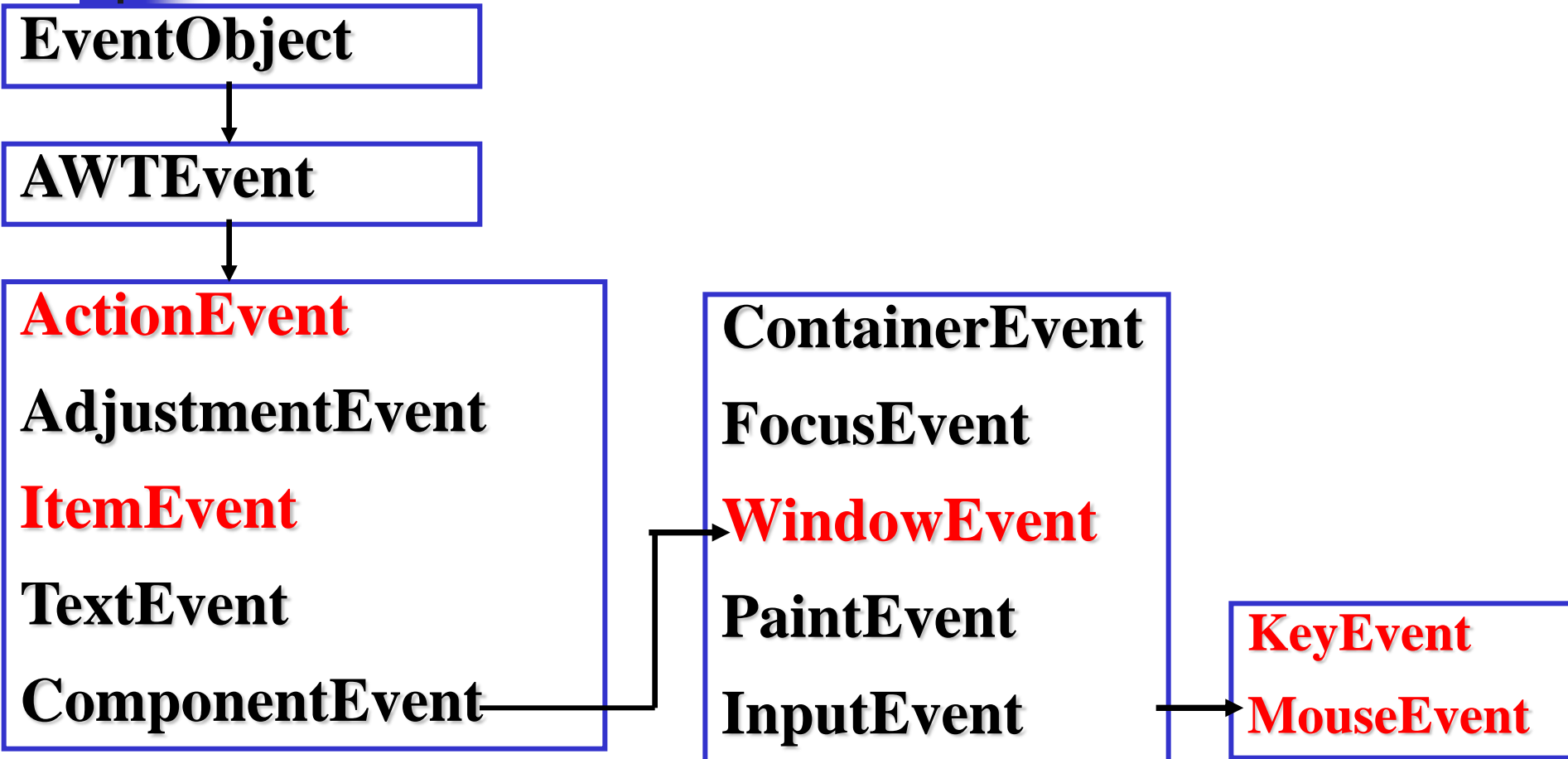
```
public void actionPerformed (ActionEvent e){  
    System.out.println("I'm pressed!");}
```

3. 向某个GUI组件注册该监听器类的一个实例。

- **BtnClickL hear=new BtnClickL ();**
- **button1.addActionListener(hear1);**

# AWTEvent事件类

## (事件对象)



■ 事件类的层次结构



```
■ import java.awt.event.*;
```

---

■ 常见事件对象类:

- **ActionEvent**(单击按钮事件)
- **KeyEvent**(键盘事件)
- **MouseEvent** (鼠标事件)
- **ItemEvent**(选项事件)
- **WindowEvent**(窗口事件)

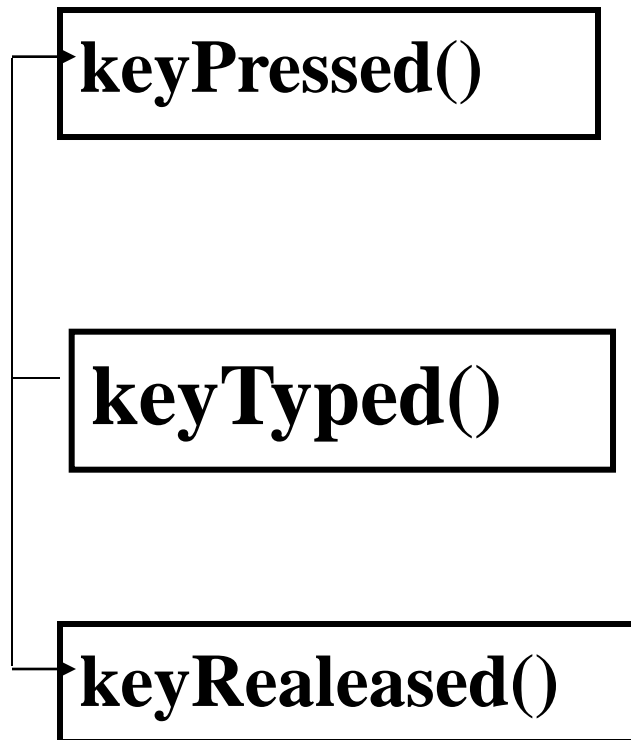


## ■ 各种监听器监听不同的事件对象

---

- 动作事件监听器(接口) : **ActionListener**
- （单击按钮、文本中回车、双击选择项等）
- 方法: **actionPerformed(ActionEvent e);**

**键盘事件监听器（接口）** : **KeyListener**  
**监听的事件对象** : **KeyEvent** (键盘事件)





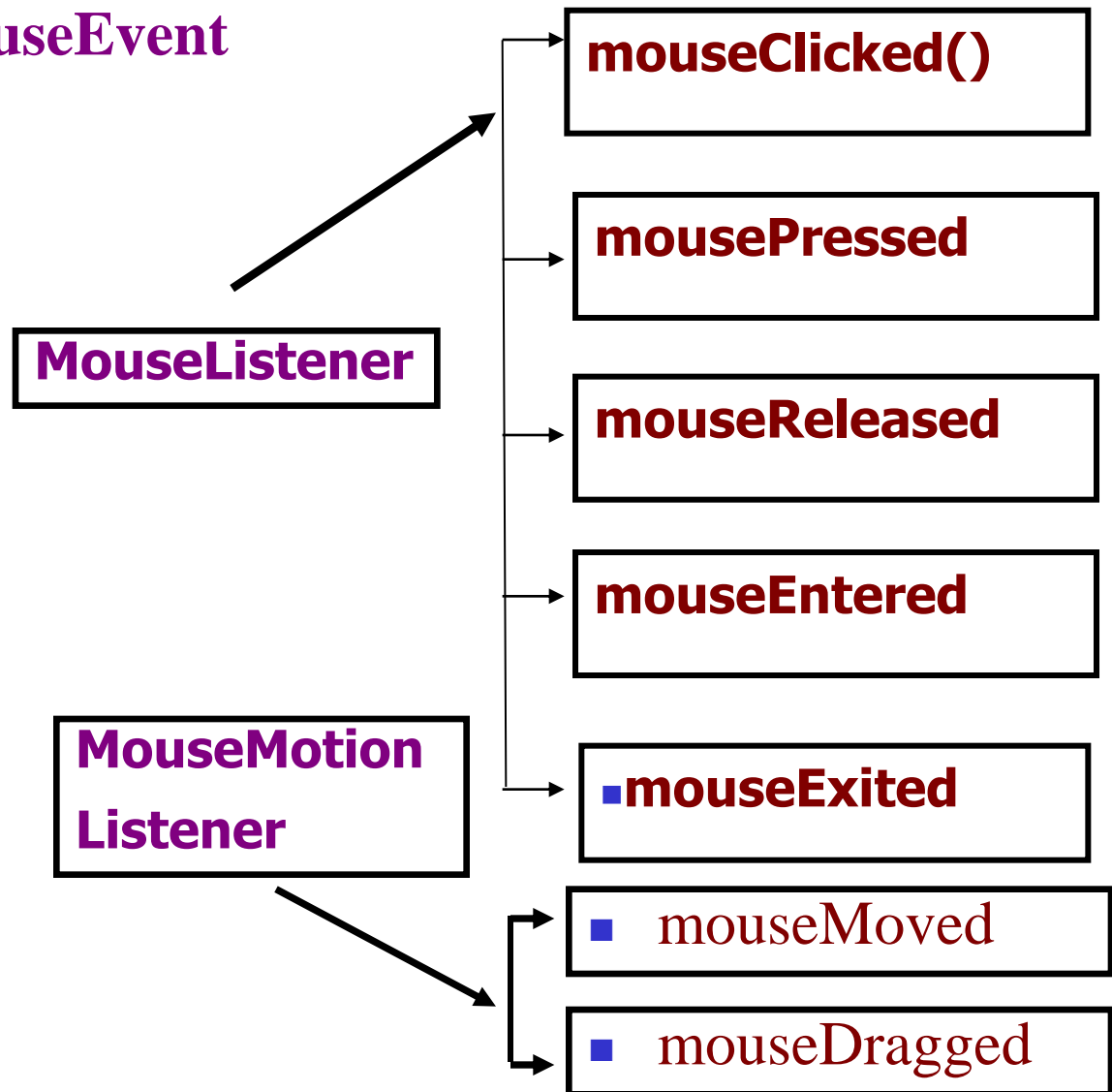
- 举例键盘事件: **testKeyListener.java**
- **import javax.swing.\*;**
- **import java.awt.event.\*;**
- **import java.awt.\*;**
- **class testKeyListener extends JFrame**
- **implements KeyListener{**
- **Container           con =null;**
- **JTextField       t1=null;**
- **JPanel           p1=null;**
-

- `testKeyListener(){`
- `con=getContentPane();`
- `p1=new JPanel();`
- `t1=new JTextField(20);`
- `con.add(p1);   p1.add(t1);`
- `con.validate();         setSize(200,200);`
- `setVisible(true);`
- `t1.addKeyListener( this );`
- `}`
- `public static void main(String args[]){`
- `testKeyListener x=new testKeyListener();`
- `}`


- **public void keyPressed (KeyEvent e){**
- **t1.setText("key is pressed");**
- **}**
- **public void keyReleased(KeyEvent e){**
- **t1.setText(" key is Release");**
- **}**
- **public void keyTyped(KeyEvent e){**
- **t1.setText(" key is Typed");**
- **}**
- **}**

- 鼠标监听器接口: **MouseListener**(鼠标被点击),
- **MouseMotionListener** (鼠标移动)

- 监听对象: **MouseEvent**



- 举例： 鼠标事件处理
- **public class** TestMouseListener **implements**
- **MouseListener, MouseMotionListener**{
- **Frame f; Panel p;**
- **TextField t1, t2; Label lb1, lb2, lb3;**
- **public TestMouseListener()**{
- **f=new Frame();             p=new Panel();**
- **t1=new TextField();     t2=new TextField();**
- **lb1=new Label("x: "); lb2=new Label("y: ");**
- **lb3=new Label("     ");**
- **lb3.setBackground(Color.*RED*);**
- **p.setBackground(Color.*YELLOW*);**



- **p.setLayout( new FlowLayout() ) ;**
- **f.add(p);    p.add(lb1) ;**
- **p.add(t1);    p.add(lb2);**
- **p.add(t2);    p.add(lb3);**
- **p .addMouseListener (this);**
- **p.addMouseMotionListener (this);**
- **f.setSize(400,200);**  
**f.setVisible(true); }**

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

```
■ TestMouseListener ibj=new  
■ TestMouseListener();  
■ }
```

```
public void mouseMoved(MouseEvent e){
```

```
■ t1.setText(" "+e.getX() );  
■ t2.setText(" "+e.getY() );  
■ lb3.setText("mouse is Moving");  
}
```

```
public void mouseDragged(MouseEvent e){
```

```
■ lb3.setText("Dragged");  
}
```

- **public void mouseClicked(MouseEvent e){**
- **lb3.setText("mouse is Clicked"); }**
- **public void mousePressed(MouseEvent e){**
- **lb3.setText("mouse is Pressed"); }**
- **public void mouseReleased(MouseEvent e){**
- **lb3.setText("mouse is Released"); }**
- **public void mouseEntered(MouseEvent e){**
- **lb3.setText("Enter the panel"); }**
- **public void mouseExited(MouseEvent e){**
- **lb3.setText("Exit from the panel"); }**
- **}**



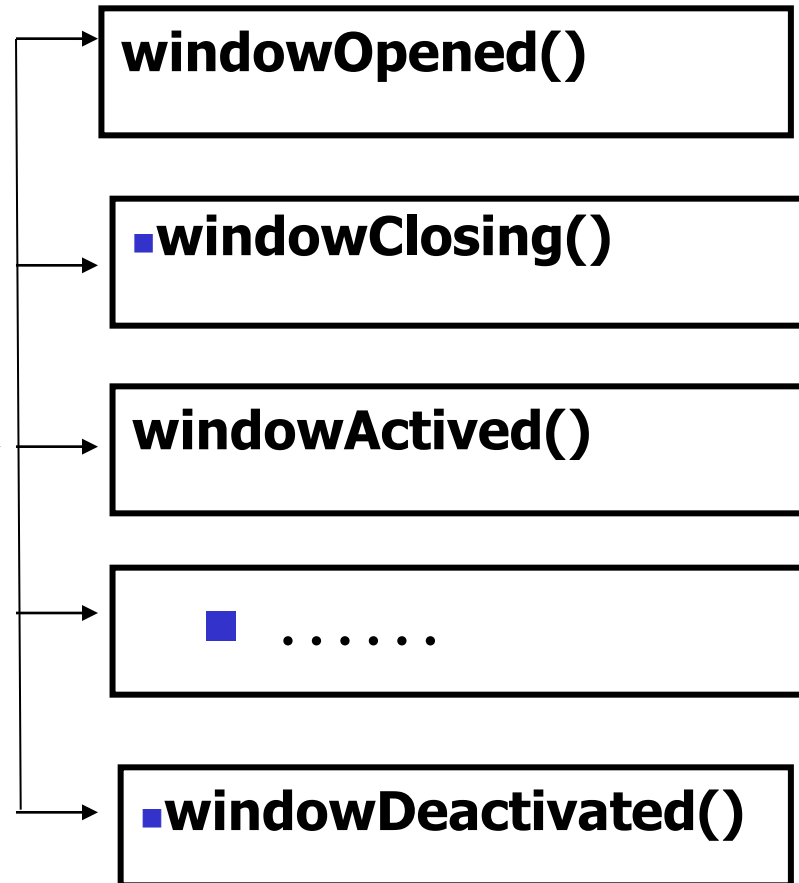


使用接口中的

■ **windowClosing()**方法监听关闭窗口事件


■ / .....

■ **Window**  
■ **Listener**



- 如果我们定义这样一个类(适配器类)
- **Public abstract class WindowAdapter**
- **implements WindowListener{**
- public void windowOpened(WindowEvent e){}
- public void windowClosed(WindowEvent e){}
- public void windowClosing(WindowEvent e){}
- public void windowActivate(WindowEvent e){}
- public void windowDeactivate(WindowEvent e){}
- public void windowIconfied(WindowEvent e){}
- public void windowDeiconfied(WindowEvent e){}
- }

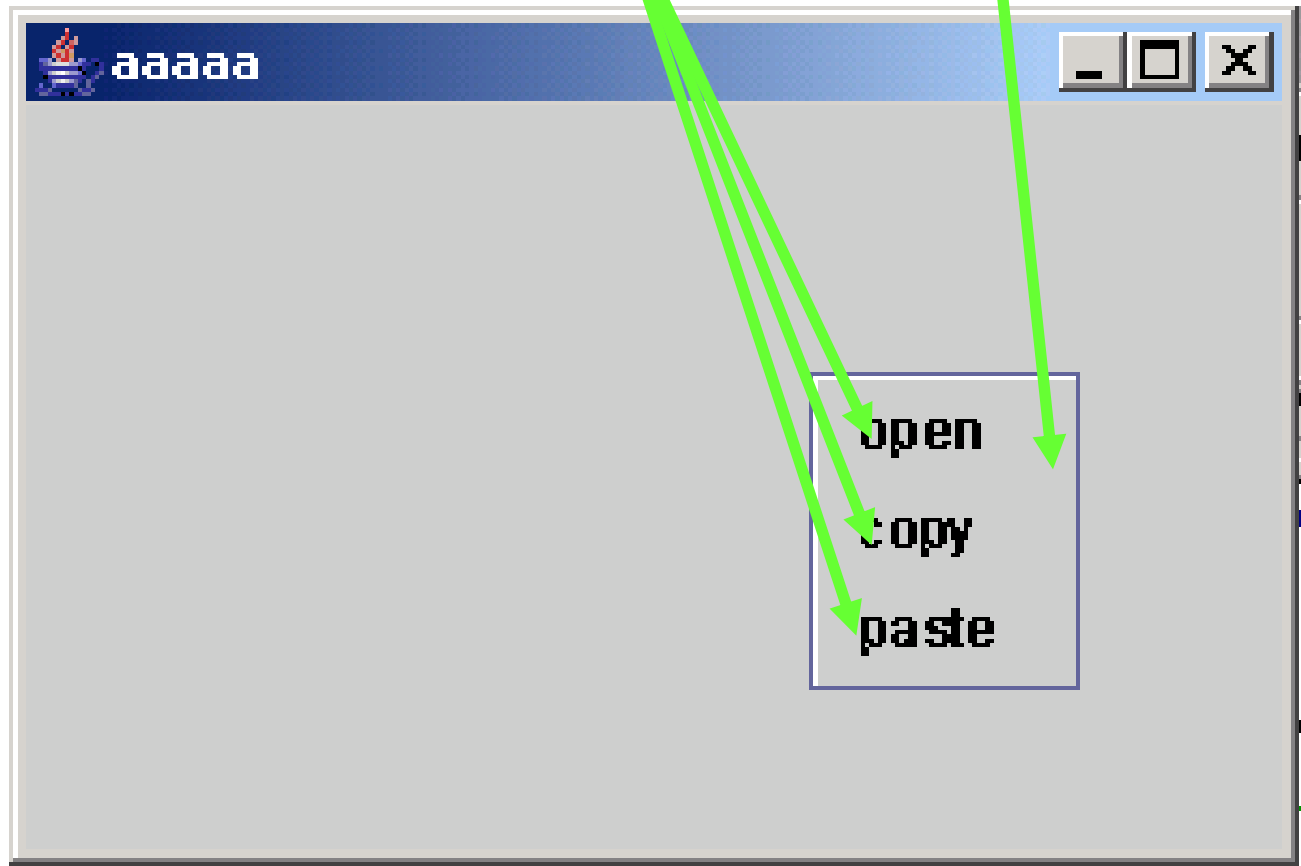
- *import java.awt.\*; import java.awt.event.\*;*
- public class testwindowAdapter **extends**  
**WindowAdapter** implements WindowListener {
- public static void main(String args[]){
- testwindowAdapter x=new testwindowAdapter();
- x.go();
- }
- public void go(){
- Frame f=new Frame("testWindows");
- f.*addWindowListener*(this);
- f.setSize(100,100);         f.setVisible(true);
- }
- public void windowClosing(WindowEvent e){
- System.exit(0); }
- }



常见的监听器接口	对应的适配器
WindowListener	■ WindowAdapter
KeyListener	KeyAdapter
MouseListener	MouseAdapter
MouseMotionListener	MouseMotionAdapter
ActionListener	无

# 单击鼠标 右键弹出菜单

JPopupMenu  
JMenuItem



- **import javax.swing.\*;     import java.awt.\*;**
- **import java.awt.event.\*;**
- **public class testJPopupMenu extends MouseAdapter**
- **implements MouseListener {**
- **JFrame f;     JPopupMenu popUp;**
- **JMenuItem openMItem,copyMItem,**
- **pasteMItem;**
- **public static void main(String args[]){**
- **testJPopupMenu s = new testJPopupMenu("xx");**
- **}**
- **public void mouseClicked(MouseEvent e) {**
- **if (e.getButton()==3 ) {**
- **popUp.show(f,e.getX(),e.getY());     }**
- **}**