Java程序设计

2019 春季

彭启民

pengqm@aliyun.com

事件处理与GUI

- Java图形用户界面
  - java.awt（Abstact Window Toolkits）
    - Java 1.0 AWT built in 30 days, and it shows
    - Java 1.1 AWT significantly improved, but GUI not finished yet
  - JFC/javax.swing
    - Java 2 Swing: very different, vastly improved
    - Very easy to add keyboard accelerators, tooltips, graphics
  - SWT/JFACE

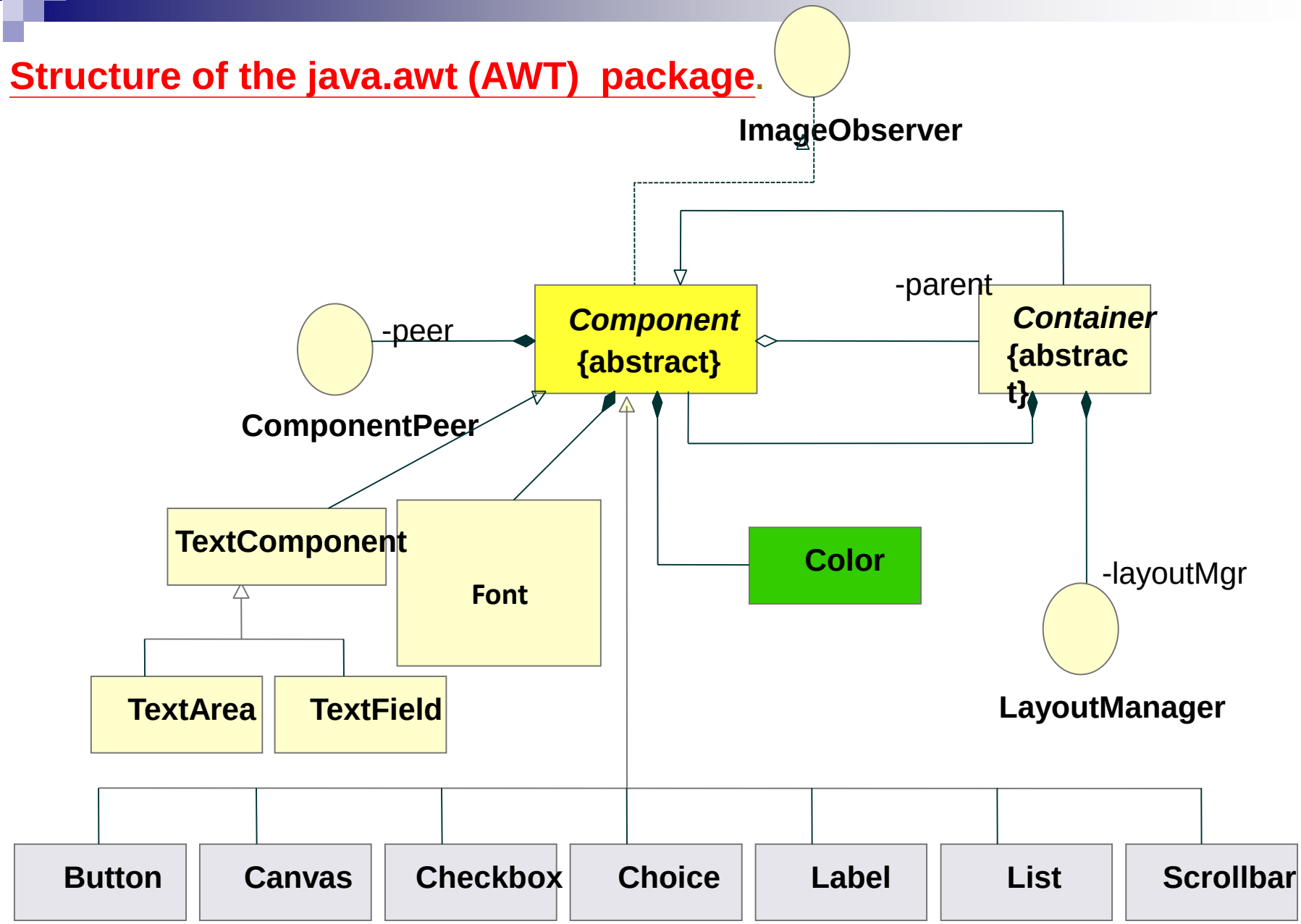
- 基本元素
  - 容器
  - 组件
  - 布局管理器
  - 事件
  - 监听器

- GUI程序设计的主要内容
  - 设置基本容器窗口
  - 设置容器布局
  - 添加所需组件
  - 交互事件处理

AWT

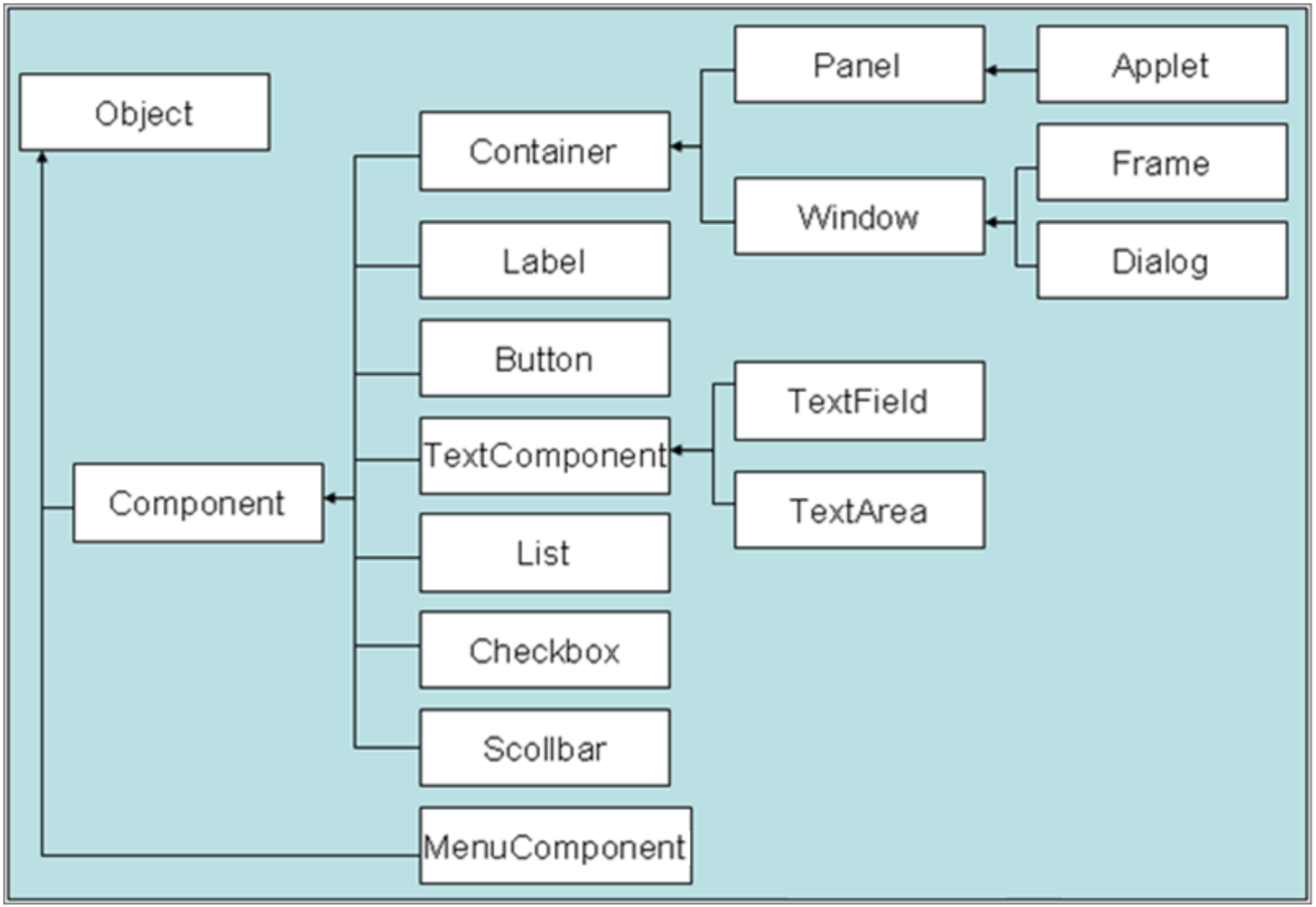
- Java的抽象窗口工具包AWT (Abstract Window Toolkit)提供了创建基于窗口的图形用户界面的便利工具。它的内容相当丰富，共有60多个类和接口
- 利用AWT类库，用户可以方便地建立自己的窗口界面，响应并处理交互事件
- 图形窗口形式的用户界面不同于传统的命令行形式的用户界面，它通过“窗口”、“按钮”、“菜单”等可视的灵活方式提供人机交互的手段，更为直观和生动。
- Java的AWT包定义了窗口系统所显示的各种对象，既包括组织窗口屏幕元素所需的基本类，也包括图形处理，显示所需的基本类

Structure of the java.awt (AWT) package.



- 组件(Component)
  - 包括屏幕上的各种组成部件，如按钮、菜单、画布等
- 容器(Container)
  - 是一种特殊的组件，用来放置、容纳其它组件或容器，如面板、对话框等

- Component类
  - 代表组件的最基本的类。是一个抽象类，封装定义了窗口中各种对象一系列最基本的属性和操作。
  - 子类包括按钮类Button，标签类Label，选择框类Checkbox，画布类Canvas等。
  - 子类Container作为最基本的组件容器
    - Container的两个子类是类Window和类Panel


Object
Component
Container
Label
Button
TextComponent
List
Checkbox
Scollbar
MenuComponent
Panel
Applet
Window
Frame
Dialog
TextField
TextArea

- Component类
  - 表示所有组件的抽象类，定义了所有组件的公共方法
    - public void add(PopupMenu popup);
    - public Color getBackground();
    - public Font getFont();
    - public Graphics getGraphics();
    - public void repaint();
    - public void setEnabled(boolean b);
    - public void setVisible(boolean b);
    - public void setSize(int width, int height);
    - public void requestFocus();
    - ……

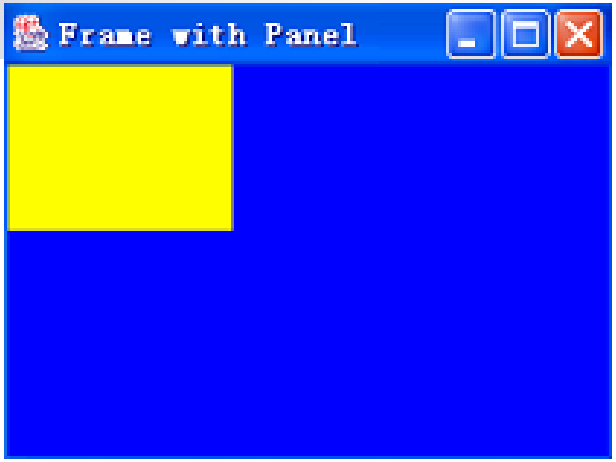
- 组件的颜色
  - 对于GUI的控制组件有四个与颜色有关的方法分别用来设置和获取组件的背景颜色和前景颜色
    - public void setBackground(Color c);
    - public Color getBackground();
    - public void setForeground(Color c);
    - public Color getForeground();

- 控制字体
  - setFont()方法
    - 包括字体类型、字型和字号
    - Font myFont = new Font(“TimesRoman”, Font.BOLD,12);
  - 字型常量
    - Font.PLAIN，Font.BOLD，Font.ITALIC

- 基本容器——Frame类

```
import java. awt.*;
public class MyFrame extends Frame {
    public MyFrame(String str) {
       super( str);
    }
    public static void main(String args[]) {
        MyFrame fr = new MyFrame(" Hello Out There!");
        fr.setSize( 500,500);
        fr.setBackground( Color. blue);
        fr.setVisible( true);
    }
}
```

- 基本容器——Panel类

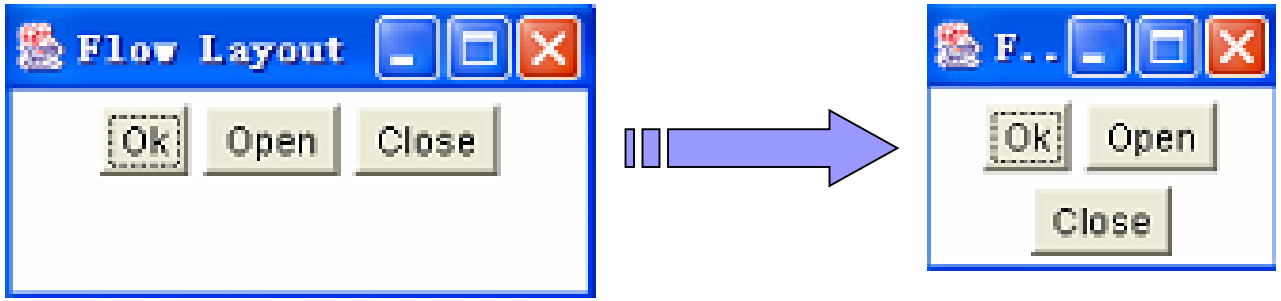



```
import java.awt.*;
public class FrameWithPanel extends Frame {
    public FrameWithPanel (String str) {
        super(str);
    }
    public static void main (String args[]) {
        FrameWithPanel fr = new FrameWithPanel("Frame with
          Panel");
        Panel pan = new Panel();
        fr.setSize(200,200);
        fr.setBackground(Color.blue);
        fr.setLayout(null); //空布局
        //可调用组件的setBounds(int x, int y, int width,int height)方法设
          置组件在容器中的大小和位置，单位均为像素。
        pan.setSize(100,100);
        pan.setBackground(Color.yellow);
        fr.add(pan);
        fr.setVisible(true);
    }
}
```

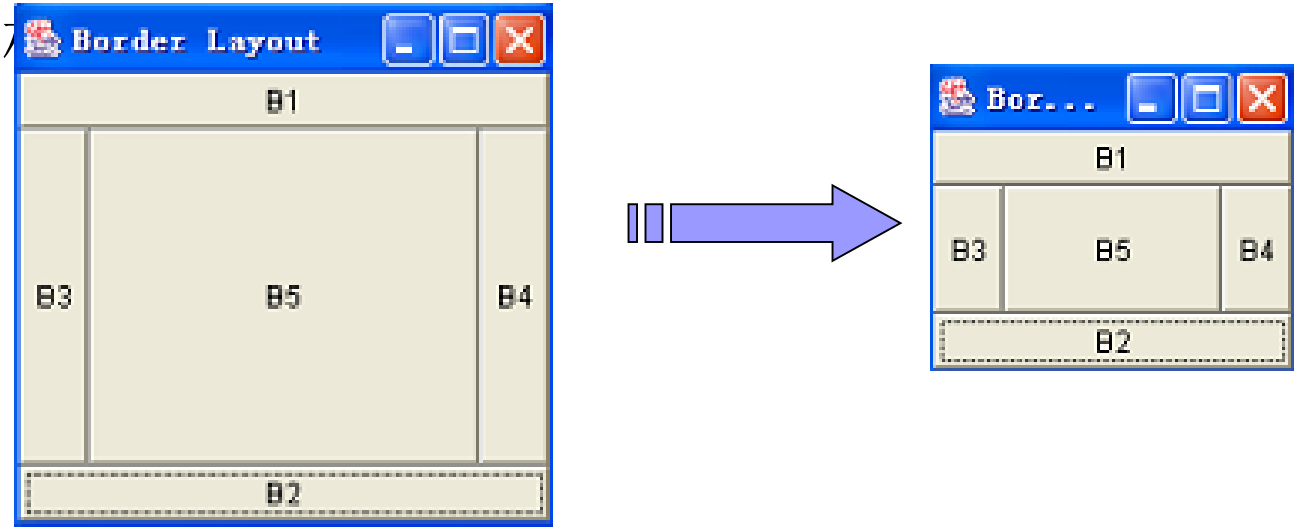
- 容器布局
  - FlowLayout
  - BorderLayout
  - BoxLayout
  - CardLayout(和其他5种配合使用)
  - GridLayout
  - GridBagLayout

通过6种布局管理器组合，设计出复杂的界面，且在不同操作系统平台上显示一致。

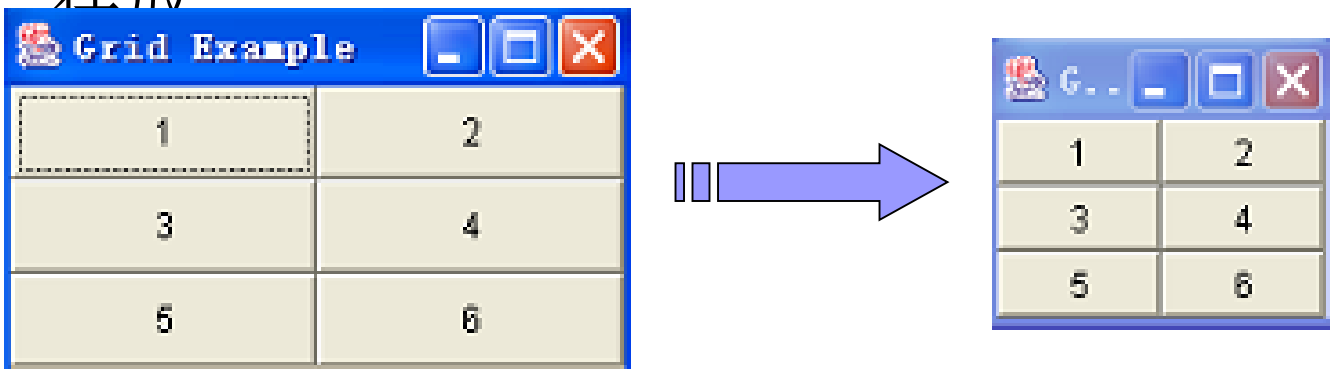
- FlowLayout
  - 顺序布局是容器的缺省布局策略，即将加入容器中的组件依次从左至右，从上至下排列，适用于组件个数较少的情况。



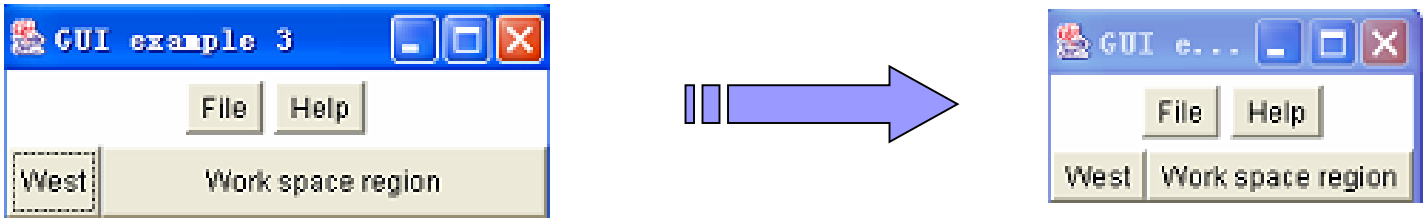
- BorderLayout
  - 区域布局将容器空间分为东西南北中五个区域，加入组件时，应通过字符串“East”、“West”、“South”、“North”、“Center”来



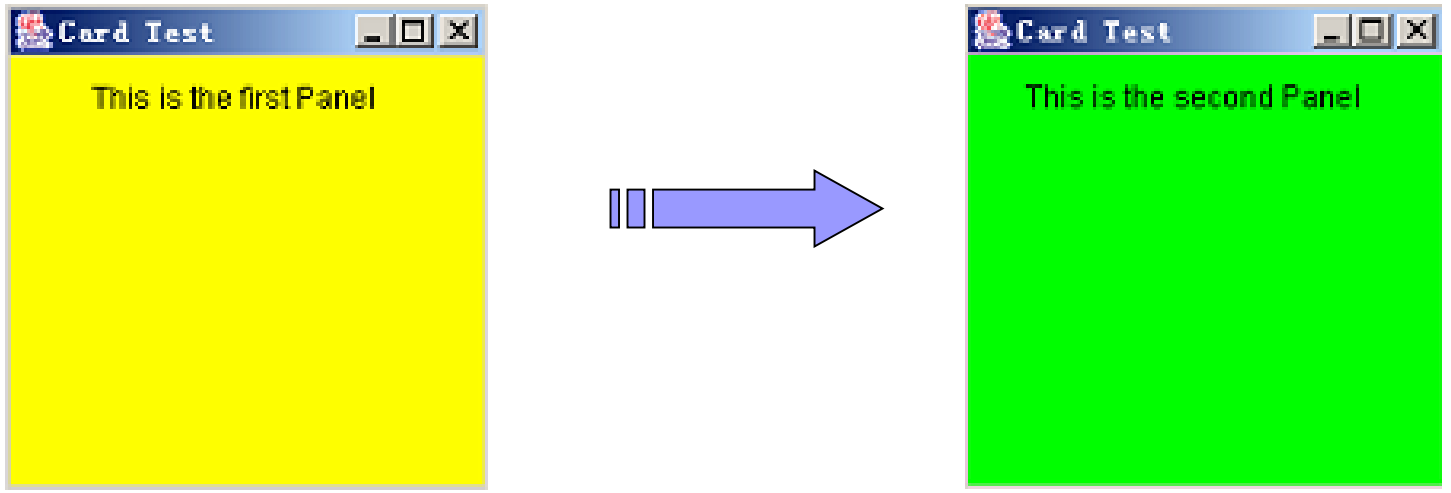
- GridLayout布局
  - 网格布局将容器空间划分为n×m的大小相同的小格，每格区间可摆放一个组件。向容器中增加组件时，按从左至右，从上至下的顺序依次存放

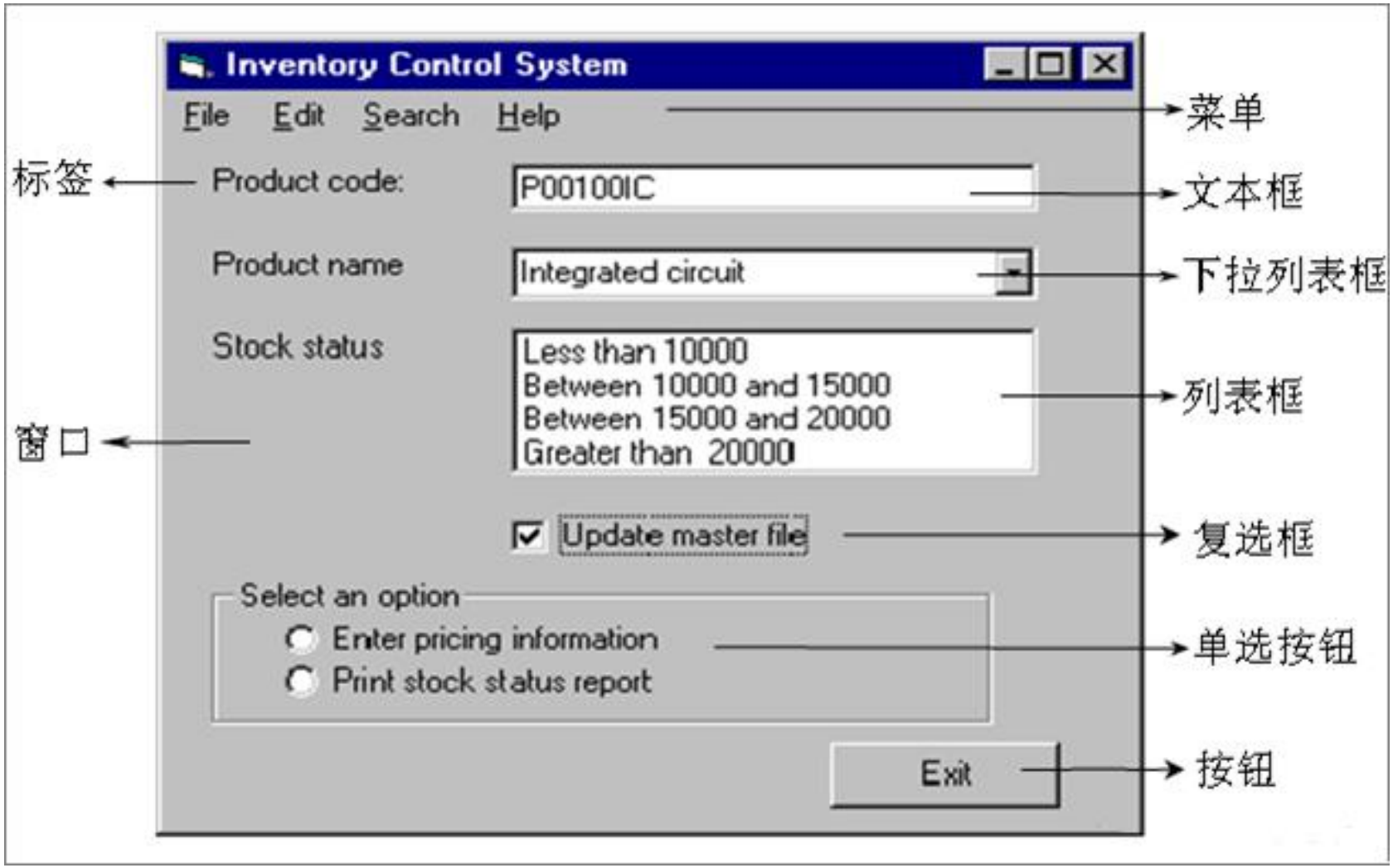


- 使用 Panel 布局
  - 可以满足更复杂的布局要求。



- CardLayout
  - CardLayout布局管理器非常简单，它将容器中的每一个组件当作一个卡片，一次仅有一个卡片可见，最初显示容器时，加入到容器的第一个组件可见。




Inventory Control System
File
Edit
Search
Help
Product code:
P00100IC
Product name
Integrated circuit
Stock status
Less than 10000
Between 10000 and 15000
Between 15000 and 20000
Greater than 20000
Update master file
Select an option
Enter pricing information
Print stock status report
Exit
标签
窗口
菜单
文本框
下拉列表框
列表框
复选框
单选按钮
按钮

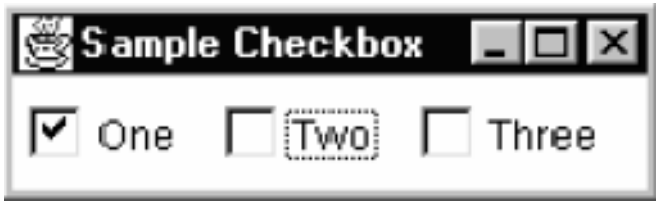
- 标签
  - Label类
    - 可通过getText()和setText()读取和设置提示框文本，不产生特殊事件
  - 用户不能修改的文本提示框
  - 创建方法：
    - Label lb = new Label(“标签”);

- 按钮
  - Button类
    - 可通过getLabel()和setLabel()方法读取和设置按钮文字
    - 可产生ActionEvent事件
  - 创建方法：
    - Button btn = new Button(“OK”);

- 单行文本框
  - TextField
    - TextField tf = new TextField(16);
- 文本区域
  - TextArea
    - TextArea ta = new TextArea(10,40);
- setText()方法和getText()方法设置和获取文本组件的显示内容
- setEditable()方法设置是否允许用户编辑
- isEditable()方法判断是否可编辑

- 复选按钮

```
f = new Frame("Sample Checkbox");
one = new Checkbox("One", true);
two = new Checkbox("Two", false);
three = new Checkbox("Three", false);
one.addItemListener(this);
two.addItemListener(this);
three.addItemListener(this);
f.setLayout(new FlowLayout());
f.add(one);
f.add(two);
f.add(three);
```

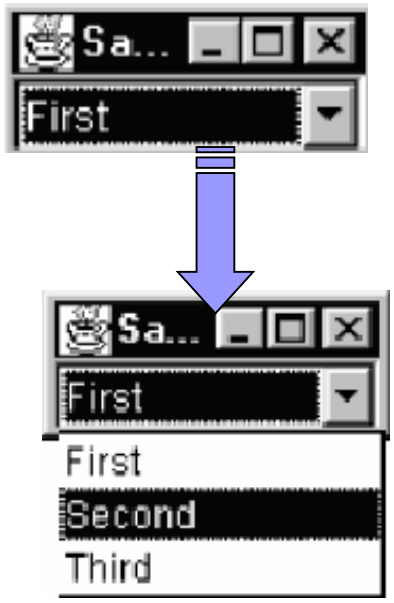


- 单选按钮

```
f = new Frame("CheckBoxGroup");
cbg = new CheckboxGroup();
one = new Checkbox("One", cbg, false);
two = new Checkbox("Two", cbg, false);
three = new Checkbox("Three", cbg, true);
f.setLayout(new FlowLayout());
one.addItemListener(this);
two.addItemListener(this);
three.addItemListener(this);
f.add(one);
f.add(two);
f.add(three);
```

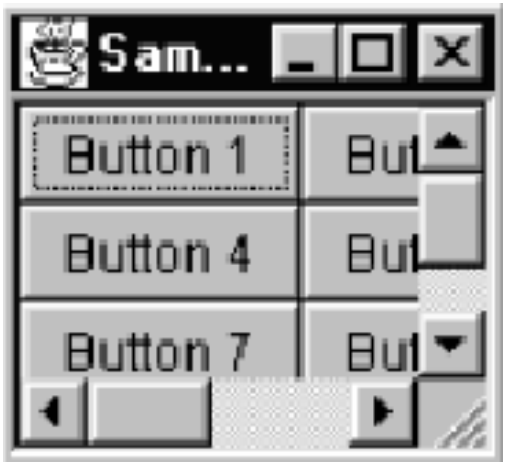
- 下拉框Choice/列表框List

```
f = new Frame("Sample Choice");
choice = new Choice();
choice.addItem("First");
choice.addItem("Second");
choice.addItem("Third");
choice.addItemListener(this);
f.add(choice, BorderLayout.CENTER);
```

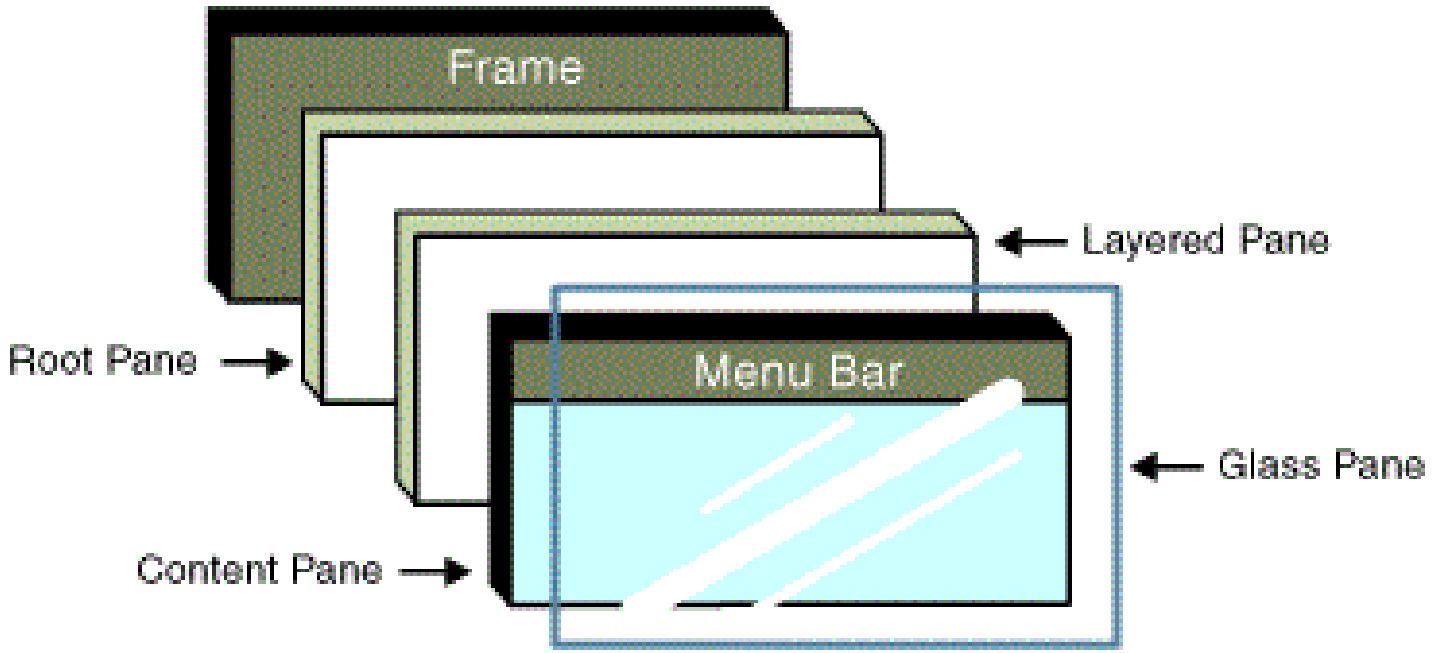


- 滚动条

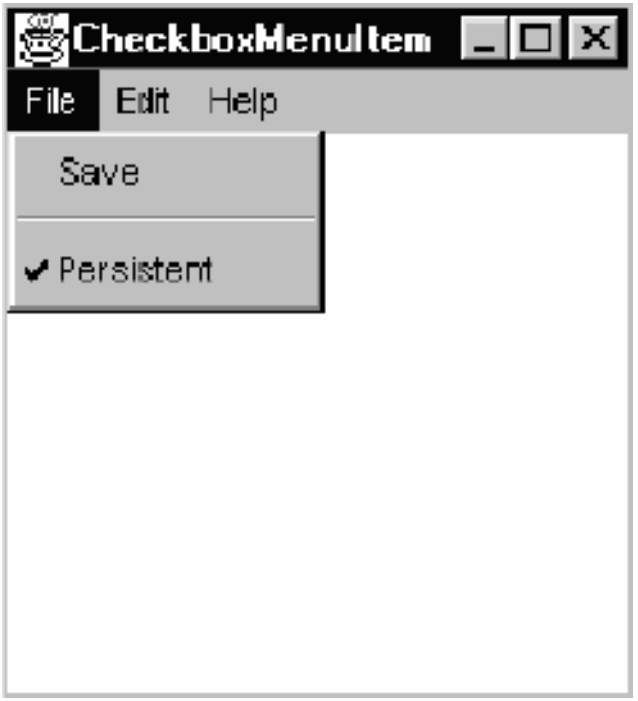
```
Frame f = new Frame("ScrollPane");
Panel p = new Panel();
ScrollPane sp = new ScrollPane();
p.setLayout(new GridLayout(3, 4));
.
.
.
sp.add(p);
f.add(sp, BorderLayout.CENTER");
f.setSize(100, 100);
f.setVisible(true);
```



- 菜单
  - 菜单条MenuBar
  - 菜单Menu
  - 普通菜单项MenuItem
  - 可选择菜单项CheckboxMenuItem

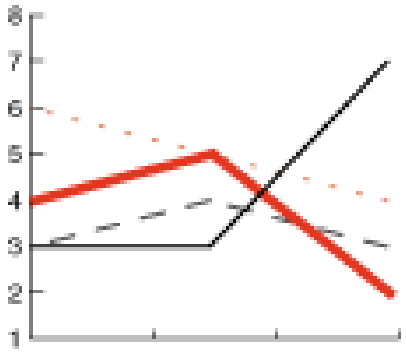

Frame
Layered Pane
Root Pane
Menu Bar
Glass Pane
Content Pane

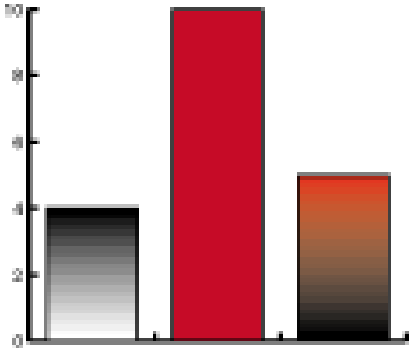
```
mb = new MenuBar();
m1 = new Menu("File");
m2 = new Menu("Edit");
m3 = new Menu("Help");
mb.add(m1);
mb.add(m2);
mb.setHelpMenu(m3);
f.setMenuBar(mb);
.....
mi2 = new MenuItem("Save");
mi2.addActionListener(this);
m1.add(mi2);
......
mi5 = new CheckboxMenuItem("Persistent");
mi5.addItemListener(this);
m1.add(mi5);
```

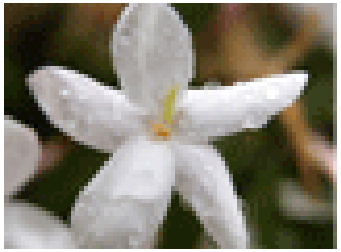


- Java 2D API绘图
  - 是JFC （Java Fundation Classes AWT+Swing+Java2D）的一员（ JDK1.2）
  - 加强了传统AWT（ Abstract Windowing Toolkit ）的描绘功能，可以轻松地描绘出任意的几何图形、运用不同的填色效果、对图形做旋转（ rotate）、缩放（ scale）、扭曲（ shear）等。
  - 增强了AWT能力，支持图像处理，提供滤镜（ filter）、碰撞侦测（hit detection）、图形重叠混色计算（composite）等功能。

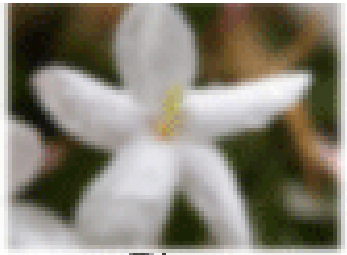
利用不同类型的线或是填色效果绘出统计图



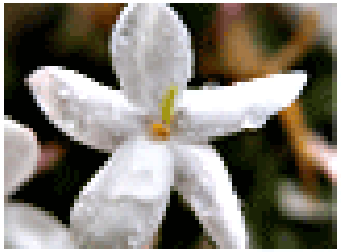




Image



Blur



Sharpen

- Graphics类
  - 是所有图形上下文的抽象基类，这个上下文允许应用将图形绘制到由不同设备实现的组件上，以及绘制到空闲屏幕的映像中。
  - 支持的方法：
    - drawLine(Point p1, Point p2)
    - drawImage(...)
    - drawOval(), drawPolygon(), drawRect, fillArc(), filllOval, etc
    - getColor(), setColor(), getFont(), setFont() etc

- Graphics 对象
  - 封装 基本绘图操作所需的状态信息,包括：
    - 要被绘制到其上的组件对象
    - 绘制和剪贴坐标的平移原点
    - 当前的剪贴区
    - 当前颜色、字体
    - 当前的逻辑像素操作函数 (XOR 或 Paint)、XOR 替换颜色
  - 用某组件或JFrame的getGraphics() 方法来获取对 Graphics 对象的引用

```
Graphics g = this.getGraphics();
```

  - 接收对图形对象的引用，该对象为窗体或组件的 paint方法中的参数或paintComponent方法的参数。

```
public void paint(Graphis g) {
    g.drawLine((11, 11,22, 22));
```

- 绘绘制线条

```
import java.awt.geom.*;
…
Line2D.Float line;
CubicCurve2D.Float cubic;
// create a Line2D
line = new Line2D.Float(20,390,200,390);
// create a CubicCurve2D,比直线多了两个控制点
cubic = new CubicCurve2D.Float(70,100,120,50,170,270,220,100);
// 设定描绘的粗细
g2. setStroke(new BasicStroke(2.0f));
g2.setColor(Color.blue);
g2.draw(line);
line = new Line2D.Float(30,400,250,400);
// 设定描绘的粗细
g2. setStroke(new BasicStroke(5.0f));
g2.draw(line);
```

- 绘制文本

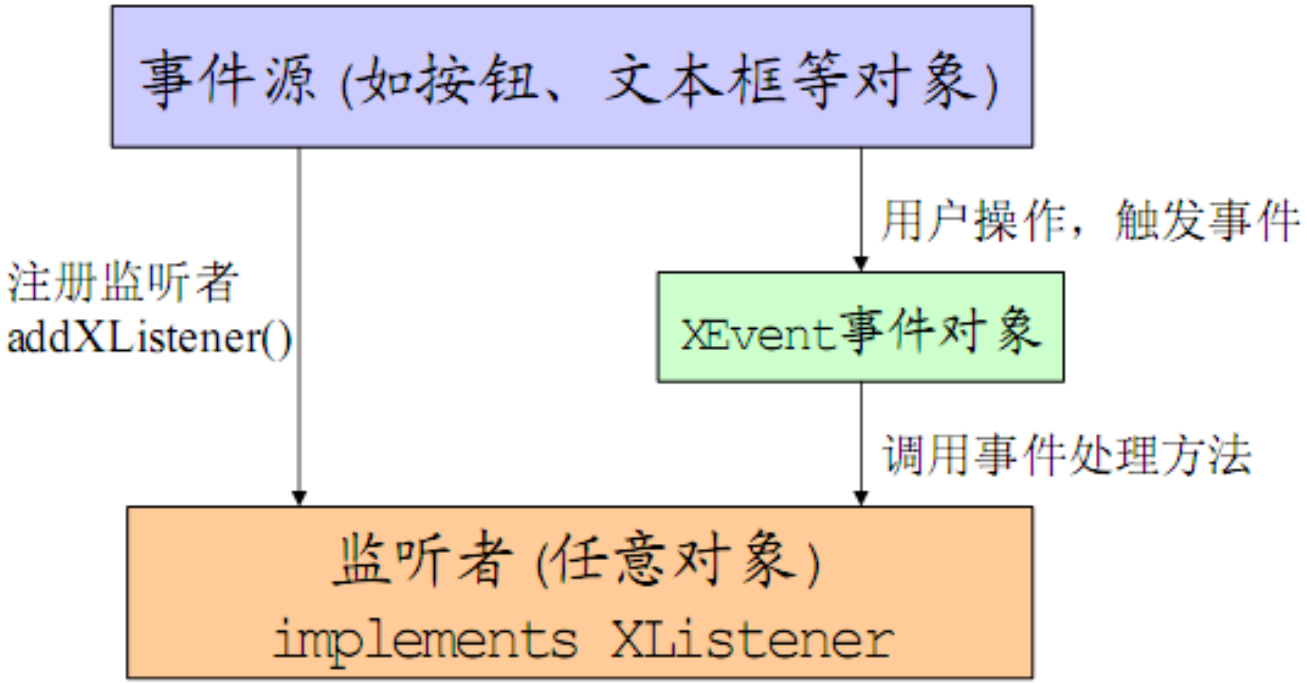
```
import java.awt.font.*;
…
Shape sha;
FontRenderContext frc =g2.getFontRenderContext();
TextLayout tl = new TextLayout("Font Test",new Font("Modern",
                                Font.BOLD+Font.ITALIC,20),frc);
sha=tl.getOutline(AffineTransform.getTranslateInstance(50,380));
g2.setColor(Color.blue);
g2.setStroke(new BasicStroke(2.0f));
g2.draw(sha);
g2.setColor(Color.white);
g2.fill(sha);
```

- 图像处理

```
float[] elements = {0.0f, -1.0f, 0.0f,-1.0f, 4.f, -1.0f,0.0f, -1.0f, 0.0f};
Image img = Toolkit.getDefaultToolkit().getImage("boat.gif");
int w = img.getWidth(this);
int h = img.getHeight(this);
BufferedImage bi = new BufferedImage(w,h,BufferedImage.TYPE_INT_RGB);
Graphics2D big = bi.createGraphics();
big.drawImage(img,0,0,this);
BufferedImageOp biop = null;
AffineTransform at = new AffineTransform();
BufferedImage bimg = new
    BufferedImage(img.getWidth(this),img.getHeight(this),
                  BufferedImage.TYPE_INT_RGB);
Kernel kernel = new Kernel(3,3,elements);
ConvolveOp cop = new ConvolveOp(kernel,ConvolveOp.EDGE_NO_OP,null);
cop.filter(bi,bimg);
biop =
    newAffineTransformOp(at,AffineTransformOp.TYPE_NEAREST_NEIGHBO
    R);
g2 = (Graphics2D)this.getGraphics();
g2.drawImage(bimg,biop,0,0);
```

- Java的事件处理
  - Java语言将每一个键盘或鼠标的操作定义为一个“事件”，在编程中只需定义每个特定事件发生时程序应该做出何种响应即可。这就是图形用户界面中的“事件”和“事件响应”
  - 除了鼠标和键盘的操作，又如系统的状态改变、标准图形界面元素等都可以引发事件，系统对这些事件分别定义处理代码，就可以保证应用程序系统有条不紊地工作。

- 事件处理的模式


事件源 (如按钮、文本框等对象)
用户操作，触发事件
注册监听者
addXListener()
XEvent事件对象
调用事件处理方法
监听者 (任意对象)
implements XListener

- 对java.awt中组件实现事件处理必须使用java.awt.event包

  import java.awt.event.*;
- 设置事件监听器

  事件源.addXXListener
  - 事件监听器对应的类实现事件所对应的接口XXListener，并重写接口中的全部方法。
- 删除事件监听器

  事件源.removeXXListener；

按钮单击事件的处理

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class ActionListenerDemo extends Applet implements
ActionListener{
        Button b=new Button("Press Me");
        public void init(){
                add(b);
                //设置按钮b的监听者
                b.addActionListener(this);
        }
        //实现ActionListener接口所定义的方法actionPerformed
        public void actionPerformed(ActionEvent e){
                showStatus("Button Clicked");
        }
}
```

事件对象

事件类型	事件名称	事件说明	引发情况
semantic	TextEvent	文本事件	文本框输入，修改等
	ItemEvent	选项事件	列表框，选择框等选项选中，撤销选择
	AdjustmentEvent	数值调整事件	滚动条位置改变
	ActionEvent	运作命令事件	选中菜单，按下按钮等
Low-level	WindowEvent	窗口事件	窗口关闭，窗口图标化等
	MouseEvent	鼠标事件	鼠标移动，鼠标按下
	KeyEvent	键盘事件	某键按下，某键松开
	InputEvent	输入事件	键盘输入，鼠标操作
	FocusEvent	焦点事件	窗口激活，拖动滚动条
	ContainerEvent	窗口事件	窗口内组件的添加，删除
	ComponentEvent	组件事件	组件移动，组件大小变化

事件监听接口

接口说明	接口名称
窗口事件监听接口	WindowListener
文本事件监听接口	TextListener
鼠标动作事件监听接口	MouseMotionListener
鼠标事件监听接口	MouseListener
键盘事件监听接口	KeyListener
选项事件监听接口	ItemListener
焦点事件监听接口	FocusListener
容器事件监听接口	ContainerListener
组件事件监听接口	ComponentListener
数值调整监听接口	AdjustmentListener
动作命令监听接口	ActionListener

事件适配器

- 为了方便，Java为那些声明了多个方法的Listener接口提供了一个对应的适配器(Adapter)类，在该类中实现了对应接口的所有方法，只是方法体为空。
- 在创建新类时，可以不实现接口，而是只继承某个适当的适配器，并且仅覆盖所关心的事件处理方法即可。

- 适配器类

对应的接口类	适配器说明	适配器名称
WindowListener	窗口适配器	WindowAdapter
MouseMotionListener	鼠标动作适配器	MouseMotionAdapter
MouseListener	鼠标适配器	MouseAdapter
KeyListener	键盘适配器	KeyAdapter
FocusListener	焦点适配器	FocusAdapter
ContainerListener	容器适配器	ContainerAdapter
ComponentListener	组件适配器	CoponentAdapter

通过适配器来创建一个可关闭的窗口

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class WindowAdapterDemo extends Frame{
        public WindowAdapterDemo(){
                super("可关闭的窗口");
                setSize(300,200);
                setVisible(true);
                addWindowListener(new WinAdapter());
        }
        public static void main(String[] args){
                new WindowAdapterDemo();
        }
        class WinAdapter extends WindowAdapter{
                public void windowClosing(WindowEvent e){
                        System.exit(0);
                }
        }
}
```

常见事件处理

- 按钮点击、菜单选择、输入栏回车
  - 产生ActionEvent事件对象
  - 用addActionListener()方法注册监听对象
  - 监听类必须实现ActionListener接口，在actionPerformed()方法中完成事件的处理。

- 动作事件(ActionEvent)
  - 用户的一个功能性操作通常会产生动作事件，如点击按钮、双击列表项、选择菜单、文本框中输入回车等
  - Java用ActionEvent类对象表示动作事件，通过该对象的方法可得到事件相关信息
    - getActionCommand()得到事件源命令名
    - getModifiers()得到功能键状态
    - getWhen()得到事件产生的绝对时间
    - getSource()得到事件源对象的引用

- 文本事件
  - TextEvent类
  - 用户修改文本框和文本区域中的内容时产生文本事件
  - 要处理文本事件，需用addTextListener()方法注册事件监听对象
    - 监听对象所在的类应该实现TextListener接口
    - 在textValueChanged()方法中完成事件处理代码
  - 当用户在文本框中输入回车时，会产生动作事件ActionEvent

- 鼠标事件
  - MouseEvent
  - 当鼠标事件发生时，系统自动生成一个该类的对象，在鼠标事件的处理方法中，经常需在调用该类的方法来获得关于事件的一些信息。

- MouseEvent类的常用方法
  - public int getX() 返回当前鼠标指针位置的*x*坐标值。
  - public int getY() 返回当前鼠标指针位置的*y*坐标值。
  - public int getClickCount() 返回事件中鼠标点击次数。
  - public String paramString() 返回一个标识该事件的字符串。

- 鼠标点击：
  - 产生MouseEvent事件对象
  - 用addMouseListener()方法注册监听对象
  - 监听类需要实现MouseListener接口，在mouseClick()方法中完成事件的处理。

- MouseListener接口
  - 鼠标监听接口（MouseListener）用于监听发生在一个GUI构件上的鼠标事件，包括鼠标的按下、释放、单击、进入和退出。
  - Java同时也提供了与该接口相对应的称作事件剪裁器的抽象类MouseAdapter。
  - 在一个实现了MouseListener接口或继承了MouseAdapter类的类中可以定义事件的处理方法，而该类的一个对象则应该用addMouseListener()方法注册到发生鼠标事件的构件上。

- MouseListener接口包含的方法
  - public void mouseClicked（MouseEvent e）
    当在一个构件上单击鼠标时被调用。
  - public void mousePressed（MouseEvent e）
    当在一个构件上按下鼠标按钮时被调用。
  - public void mouseReleased（MouseEvent e）　当在一个构件上释放鼠标按钮时被调用。
  - public void mouseEntered（MouseEvent e）

    当鼠标指针进入构件时被调用。
  - public void mouseExited（MouseEvent e）

    当鼠标指针退出构件时被调用。

- 鼠标移动
  - 产生MouseEvent事件对象
  - 用addMouseMotionListener()方法注册监听对象
  - 监听类需要实现MouseMotionListener接口，在mouseMoved()方法中完成事件的处理

- 鼠标移动监听接口
  - MouseMotionListener,用于监听发生在一个GUI构件上的鼠标移动事件，包括鼠标的移动和拖动，其使用方法与MouseListener接口相同。

- MouseMotionListener接口包含的方法
  - public void mouseDragged（MouseEvent e）
    当在一个构件上按下鼠标按钮并且拖动鼠标时该方法被调用。鼠标拖动事件持续到鼠标按钮被释放时为止，而不管鼠标的位置是否超出了原来构件的边界。
  - public void mouseMoved（MouseEvent e）
    当鼠标指针移动时该方法被调用，注意此时鼠标的按钮并没有被按下。

鼠标事件及鼠标移动事件的处理。

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class MouseEventDemo extends Applet implements
MouseListener,MouseMotionListener
{
        int x1,y1,x2,y2;
        public void init(){
                addMouseListener(this);
                addMouseMotionListener(this);
        }
        public void paint(Graphics g){
                g.drawLine(x1,y1,x2,y2);
        }
        public void mousePressed(MouseEvent e){
                x1=e.getX();
                y1=e.getY();
        }
```

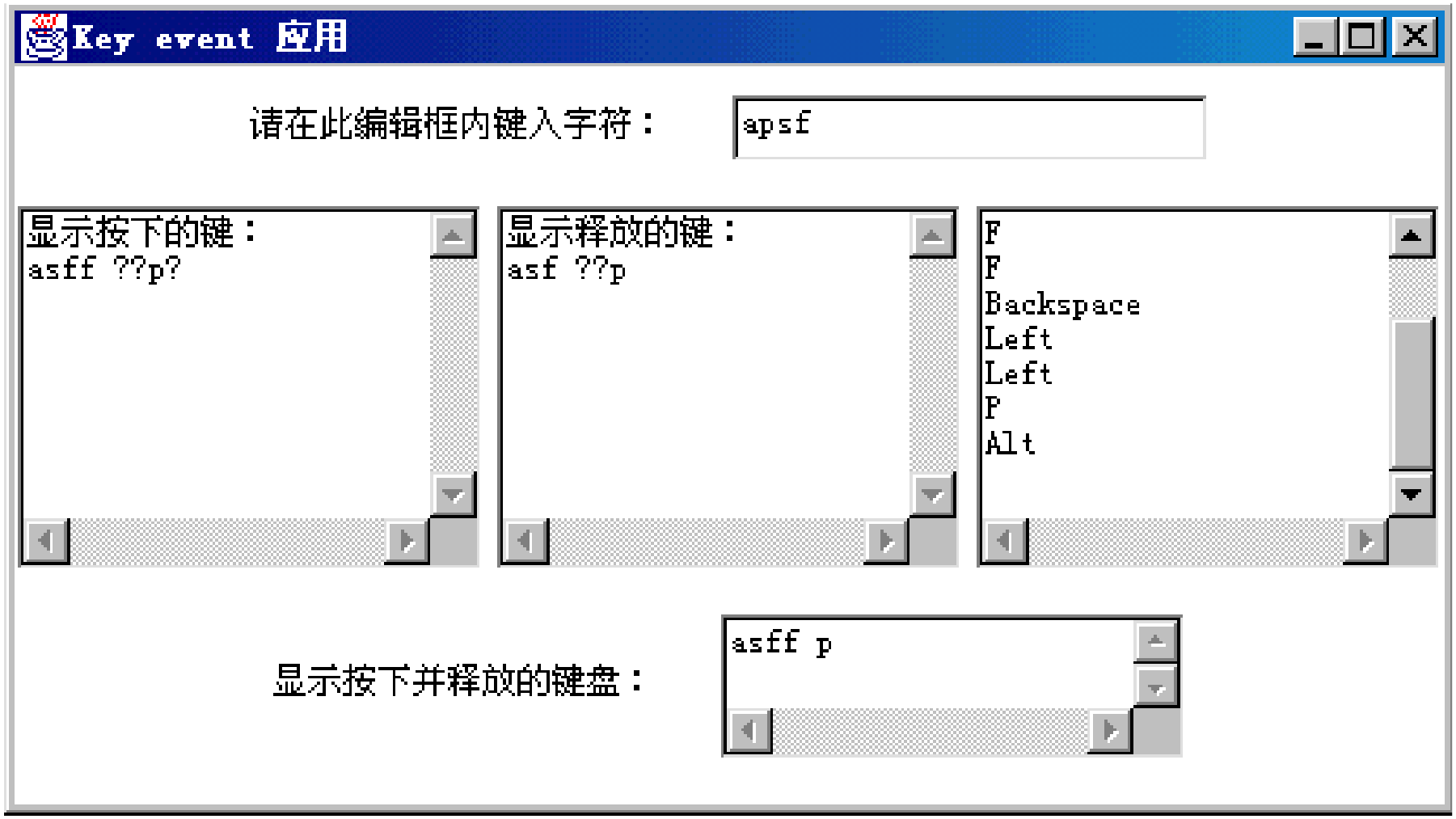
```
        public void mouseClicked(MouseEvent e){}
        public void mouseEntered(MouseEvent e){}
        public void mouseExited(MouseEvent e){}
        public void mouseReleased(MouseEvent e){}
        public void mouseDragged(MouseEvent e){
                x2=e.getX();
                y2=e.getY();
                repaint();
        }
        public void mouseMoved(MouseEvent e){}
}
```

- 键盘事件
  - KeyListener和KeyAdapter用于监听键盘事件的发生并将其传送到相应的事件处理方法中去
  - KeyEvent主要用于提供事件发生时的有关信息

- KeyListener接口
  - 监听3种键盘事件
    - 键按下（Pressed）
    - 键释放（Released）
    - 键按下并释放（Typed）

- KeyListener接口事件处理方法
  - public void keyTyped(KeyEvent e) 当键盘上的一个键被按下并释放后该方法被调用
  - public void keyPressed(KeyEvent e) 当键盘上的一个键盘被按下后该方法被调用。
  - public void keyReleased(KeyEvent e) 当键盘上的一个键盘被释放时该方法被调用

- KeyEvent类中常用的方法
  - public int getKeyCode() 返回键盘事件中相关键的整数类型键码
  - public char getKeyChar() 返回键盘事件中相关键的字符类型键码。例如，对于Shift+a键返回的字符是A
  - public static String getKeyText(int keyCode) 返回一个描述由参数int keyCode指定的键的字符串，如"HOME"，"F1" 或"A".等
  - public String paramString() 返回一个标识该事件的参数字符串


Key event 应用
请在此编辑框内键入字符：
apsf
显示按下的键：
asff ??p?
显示释放的键：
asf ??p
F
F
Backspace
Left
Left
P
Alt
显示按下并释放的键盘：
asff p

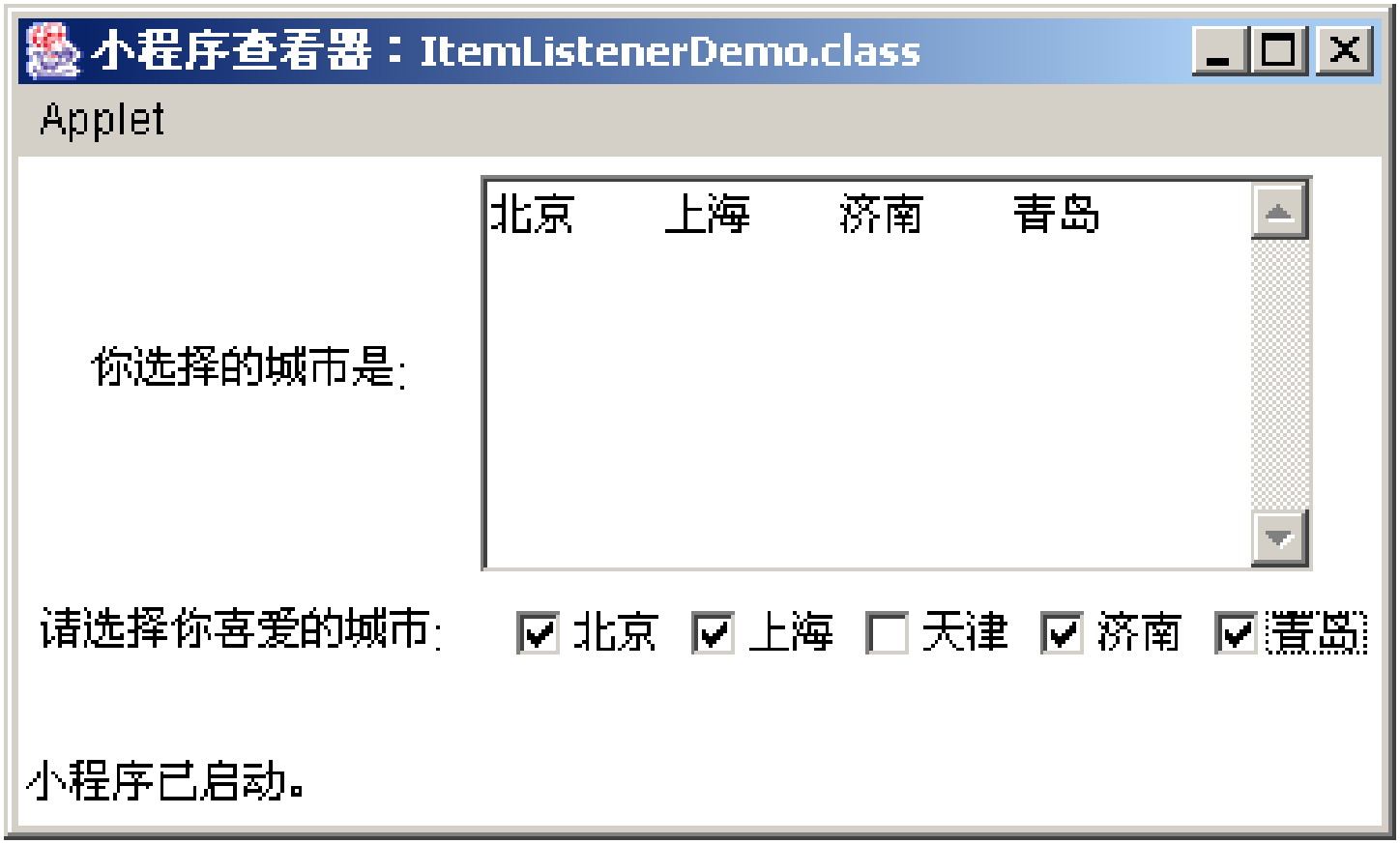
```
import java.awt.event.*;
import java.awt.*;
public class key implements KeyListener{
    TextField tx = new TextField(20);
    TextArea ta1 = new TextArea("显示按下的键：\n",7,20);
    TextArea ta2 = new TextArea("显示释放的键：\n",7,20);
    TextArea ta3 = new TextArea("显示控制与功能键：\n",7,20);
    TextArea ta4 = new TextArea(null,2,20);

    public void keyTyped(KeyEvent e) {
        ta4.append(String.valueOf(e.getKeyChar()));
    }

    public void keyPressed(KeyEvent e) {
        ta1.append(String.valueOf(e.getKeyChar()));
        //ta1.append(String.valueOf(e.getKeyCode()));
        ta3.append(e.getKeyText(e.getKeyCode())+"\n");
    }
```

```
public void keyReleased(KeyEvent e) {
        ta2.append(String.valueOf(e.getKeyChar()));
    }

     key(){
        Frame win1 = new Frame("Key event 应用");
        win1.addWindowListener(new WindowAdapter(){
          public void windowClosing(WindowEvent e) {
            System.exit(0);}});

        win1.setLayout(new FlowLayout());
```

```
        Panel p1 = new Panel();
        win1.add(p1);
        p1.add(new Label("请在此编辑框内键入字符： "));
        p1.add(tx);
        tx.addKeyListener(this);

        Panel p2 = new Panel();
        win1.add(p2);
        p2.add(ta1);
        p2.add(ta2);
        p2.add(ta3);

        Panel p3 = new Panel();
        win1.add(p3);
        p3.add(new Label("显示按下并释放的键盘： "));
        p3.add(ta4);

        win1.setSize(500,280);
        win1.setVisible(true);
    }
    public static void main(String[] args) {
        new key();
    }}
```

- 鼠标事件和键盘事件类都继承自InputEvent
  - getModifiers()方法可获得事件发生时的细节，比如鼠标按键时的键位，敲击键盘时是否同时按下了Alt或Ctrl键等
- 所有的awt组件都能成为鼠标或键盘事件的事件源

- 选择事件
  - 当单选按钮、复选按钮、下拉框、列表框的内容被选中时，将产生选择事件--ItemEvent事件对象
    - ItemEvent对象的getItem()方法可以得到用户的操作的选择项
    - getItemSelectable()可以得到引发选择事件的事件源对象
  - 要处理选择事件，需用事件源的addItemListener()方法注册事件监听对象
    - 监听对象所在的类应该实现ItemListener接口
    - 在itemStateChanged()方法中完成事件处理代码


小程序查看器：ItemListenerDemo.class
Applet
北京 上海 济南 青岛
你选择的城市是:
请选择你喜爱的城市:
北京
上海
天津
济南
青岛
小程序已启动。

选择事件的处理

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class ItemListenerDemo extends Applet implements ItemListener{
    TextArea ta=new TextArea(6,30);
        String[] city={"北京","上海",“天津","济南","青岛"};
    Checkbox cb[]=new Checkbox[5];
    public void init(){
                add(new Label("你选择的城市是: "));
                add(ta);
                add(new Label("请选择你喜爱的城市: "));
                for(int i=0;i<5;i++){
                        cb[i]=new Checkbox(city[i]);
                        add(cb[i]);
                        cb[i].addItemListener(this);
                }
    }
    public void itemStateChanged(ItemEvent e){
                ta.append(e.getItem()+"\t");
   }
}
```

- 调整事件
  - AdjustmentEvent
    - getAdjustmnetType()方法可以得到事件发生的细节
    - getValue()方法可以得到滑块的位置
  - 当用户改变滚动条的滑块位置时，将产生调整事件
  - 要处理选择事件，需用addAdjustmentListener()方法注册事件监听对象
    - 监听对象所在的类应该实现AdjustmentListener接口
    - 在adjustmentValueChanged()方法中完成事件处理代码

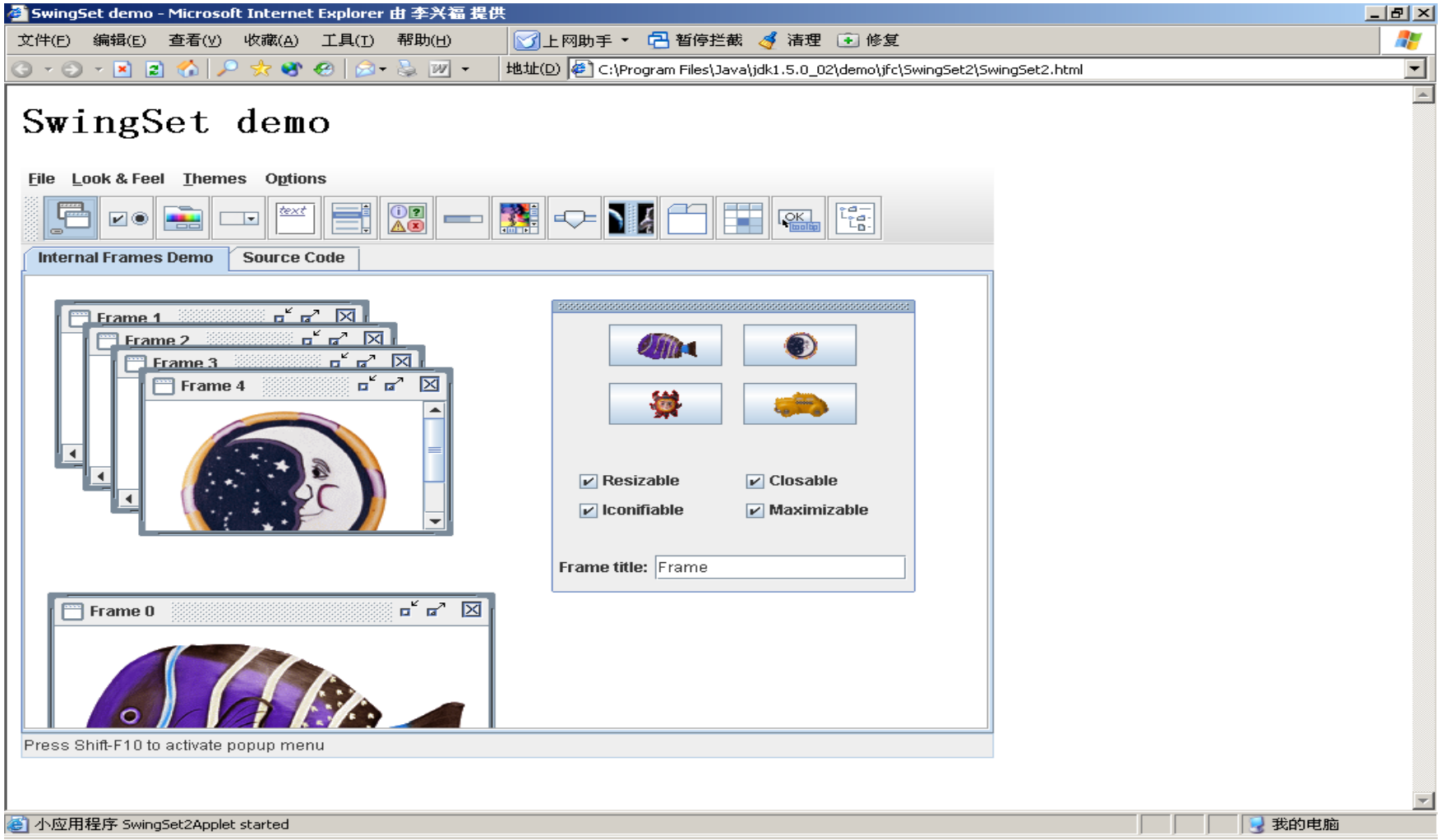
事件类/接口名称	接口方法及说明
ActionEvent 动作事件类 ActionListener 接口	actionPerformed(ActionEvent e) 单击按钮、选择菜单项或在文本框中按回车时
ItemEvent 选择事件类 ItemListener 接口	itemStateChanged(ItemEvent e) 选择复选框、选项框、单击列表框、选中带复选框菜单时
KeyEvent 键盘事件类 KeyListener 接口	keyPressed(KeyEvent e)　键按下时 keyReleased(KeyEvent e)　键释放时 keyTyped(KeyEvent e)　击键时
MouseEvent 鼠标事件类 MouseListener 接口	mouseClicked(MouseEvent e)　单击鼠标时 mouseEntered(MouseEvent e)　鼠标进入时 mouseExited(MouseEvent e)　鼠标离开时 mousePressed(MouseEvent e)　鼠标键按下时 mouseReleased(MouseEvent e)　鼠标键释放时
MouseEvent 鼠标移动事件类 MouseMotionListener 接口	mouseDragged(MouseEvent e)　鼠标拖放时 mouseMoved(MouseEvent e)　鼠标移动时
TextEvent 文本事件类 TextListener 接口	textValueChanged(TextEvent e) 文本框、多行文本框内容修改时
WindowEvent 窗口事件类 WindowListener 接口	windowOpened(WindowEvent e)　窗口打开后 windowClosed(WindowEvent e)　窗口关闭后 windowClosing(WindowEvent e)　窗口关闭时 windowActivated(WindowEvent e)　窗口激活时 windowDeactivated(WindowEvent e)　窗口失去焦点时 windowIconified(WindowEvent e)　窗口最小化时 windowDeiconified(WindowEvent e)　最小化窗口还原时

Swing组件

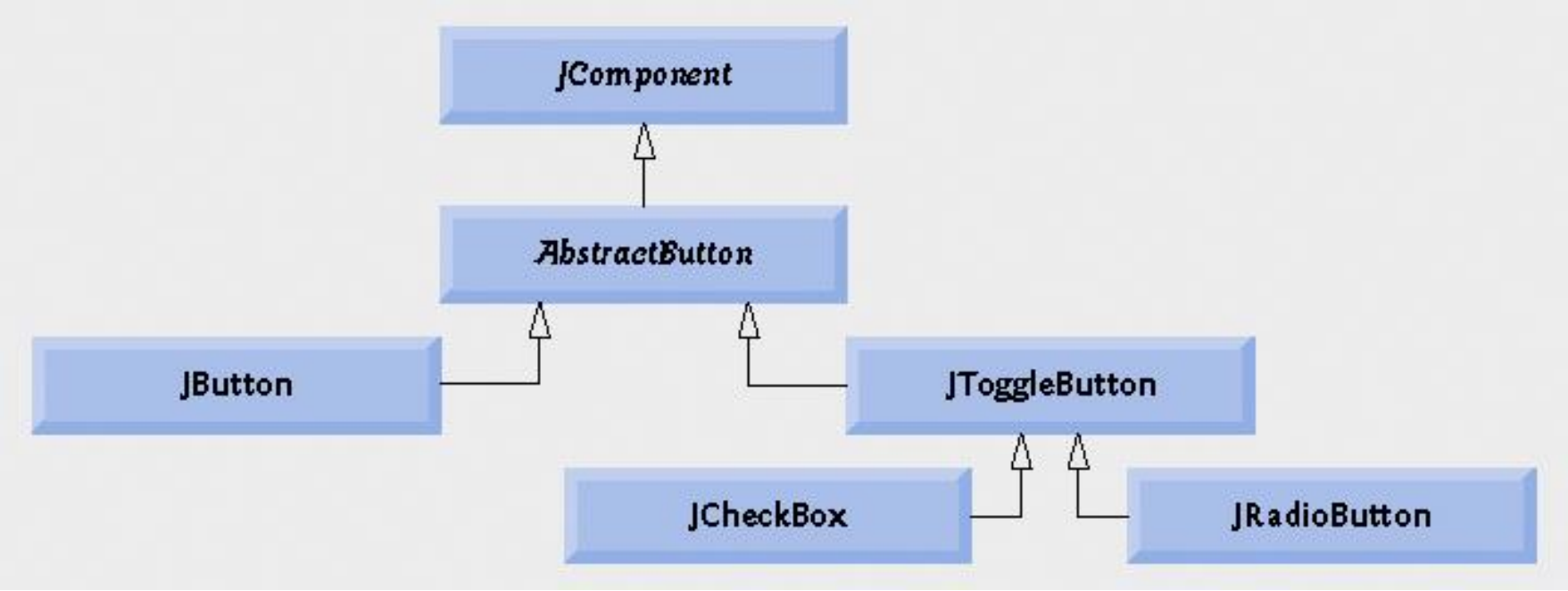
- javax.swing是Java基础类库(JFC)的一个组成部分，提供了一套比前者功能更强、数量更多、更加美观的图形用户界面组件。
- Swing和AWT最大差别在于Swing组件类不带本地代码，因此不受操作系统平台的限制，具有比AWT更强的功能
  - 比如：Swing按钮和标签类可以显示图像标题且可被制作成非矩形形状、可以为Swing组件加边框、能够自动适应操作系统外观等。

Swing Components and the containment hierarchy

- Borders
- Buttons
- Checkboxes
- ComboBoxes
- Image Icons
- Labels
- Layered Panes and Internal Frames (MDI)
- Lists and List Boxes
- Menus
- Popup Menus
- Radio Buttons
- Progress Bars
- Scrolling Support
- Scrollbars
- Splitter Control
- Tabbed Panes


SwingSet demo - Microsoft Internet Explorer 由 李兴福 提供
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)
上网助手 暂停拦截 清理 修复
地址(D) C:\Program Files\Java\jdk1.5.0_02\demo\jfc\SwingSet2\SwingSet2.html
SwingSet demo
File Look & Feel Themes Options
Internal Frames Demo
Source Code
Frame 1
Frame 2
Frame 3
Frame 4
Frame 0
Resizable
Closable
Iconifiable
Maximizable
Frame title:
Frame
Press Shift-F10 to activate popup menu
小应用程序 SwingSet2Applet started
我的电脑

- JButton


JComponent
AbstractButton
JButton
JToggleButton
JCheckBox
JRadioButton

发展

- SWT
- SwingX
- JavaFX

作业（6选1）

- 1.研究报告：Java与大数据分析
- 2.研究报告： Java与人工智能
- 3.研究报告： Java与高性能计算
- 4.研究报告：JavaSE5.0-12.0的变化

  - 内容不少于5000字，独立完成，得分系数0.9
  - 注明姓名学号，5月31日22时前课程网提交

作业（任意选作其一）

- 5.程序设计
  - 数据收发（格式自定、数据自定），用户界面自定
  - 一个接收方，一个发送方；当接收方收到发送方发送的消息后，打印发送的消息及发送方的地址和端口号，之后向发送方反馈信息“收到了！”；发送方打印出接收方反馈的消息。
  - 独立完成，得分系数1.0
  - 注明姓名学号，5月31日22时前课程网提交（源代码文件、编译后的文件）

作业（任意选作其一）

- 6.程序设计
  - 从文件、网络或数据库读取数据（格式自定、数据自定），显示统计结果（包括图形二种以上），用户界面自定
  - 可不超过3人组队完成，说明各自工作（仅设计及文档得分系数0.95，编程及调试得分系数1.05 ）
  - 注明姓名学号，5月31日22时前课程网提交（分析设计文档、源代码文件、编译后的文件，分工情况）