

第7章 图形用户界面（一）

郑 莉

导学

本章主要内容

- 绘图
- Swing基础
- Swing的层次
- 事件处理
- Swing组件
- 其它Swing特性
- 桌面API简介

(说明: 本章部分例题引用自The Java Tutorials <http://docs.oracle.com/javase/tutorial/>)

绘图

绘图

- 图形环境和图形对象
- 颜色和字体
- 使用Graphics类绘图
- 使用Graphics2D类绘图

图形环境和图形对象

- 坐标
 - GUI组件的左上角坐标默认为（0， 0）。
 - 从左上角到右下角，水平坐标x和垂直坐标y增加。
 - 坐标的单位是像素。
- Graphics对象
 - 专门管理图形环境。Graphics类是一个抽象类。
 - 抽象类Graphics提供了一个与平台无关的绘图接口。
 - 各平台上实现的Java系统将创建Graphics类的一个子类，来实现绘图功能，但是这个子类对程序员是透明的。
 - 在执行paint方法时，系统会传递一个指向特定平台的Graphics子类的图形对象g。

颜色

- Java中有关颜色的类是Color类，它在java.awt包中，声明了用于操作颜色的方法和常量

颜色

- **Color**类，以及**Graphics**类中与颜色有关的方法

名称	描述
<code>public final static Color GREEN</code>	常量 绿色
<code>public final static Color RED</code>	常量 红色
<code>public Color(int r,int g,int b)</code>	通过指定红、蓝、绿颜色分量（0~255）， 创建一种颜色
<code>public int getRed()</code>	返回某颜色对象的 红色分量值 (0~255)
Graphics: <code>public void setColor(Color c)</code>	Graphics类的方法，用于 设置组件的颜色
Graphics: <code>public Color getColor()</code>	Graphics类的方法，用于 获得组件的颜色

字体

- `Font`类——有关字体控制，在`java.awt`包中

字体

- **Font**类，以及**Graphics**类中与字体有关的方法

名称	描述
<code>public final static int PLAIN</code>	一个代表 普通字体风格 的常量
<code>public final static int BOLD</code>	一个代表 黑体字体风格 的常量
<code>public final static int ITALIC</code>	一个代表 斜体字体风格 的常量
<code>public Font(String name,int style,int size)</code>	利用指定的字体、风格和大小 创建一个Font对象
<code>public int getStyle()</code>	返回一个表示 当前字体风格 的整数值
<code>public Boolean isPlain()</code>	测试一个字体 是否是普通字体风格
Graphics: <code>public Font getFont()</code>	获得 当前字体
Graphics: <code>public void setFont(Font f)</code>	设置当前字体为f 指定的字体、风格和大小

Graphics类

- Graphics类对象可以绘制文本、线条、矩形、多边形、椭圆、弧等多种图形——这一行文字不显示

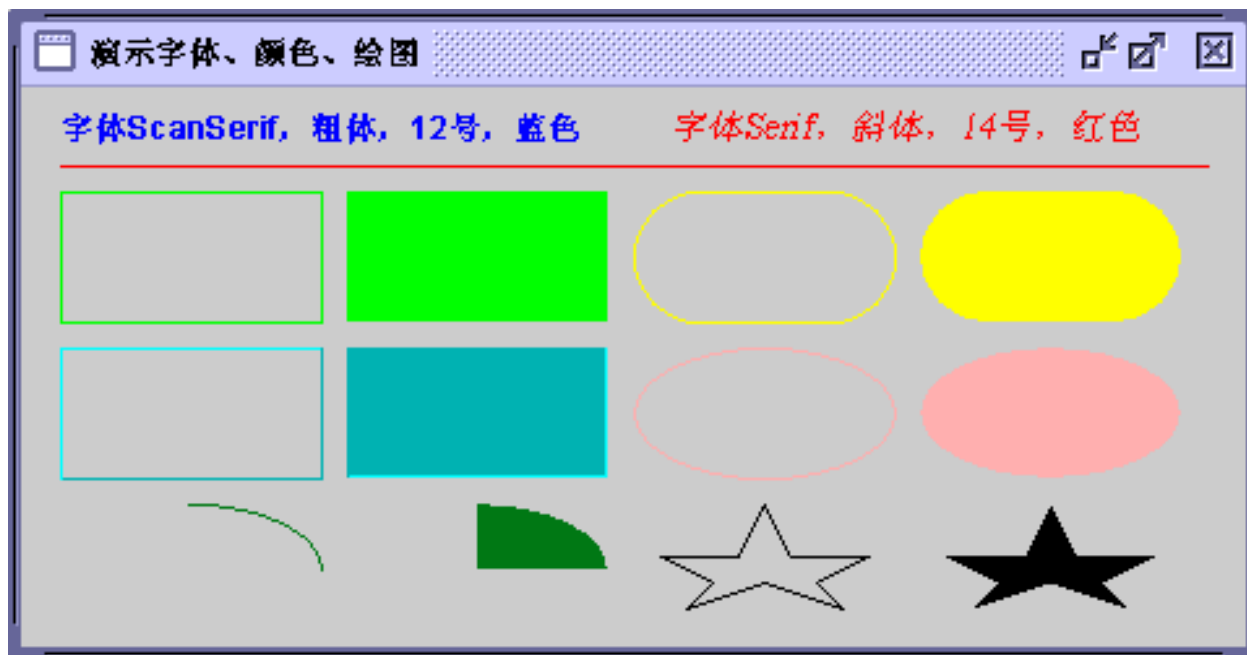
Graphics类常用方法

名称	描述
<code>public void drawString(String str, int x, int y)</code>	绘制字符串，左上角的坐标是 (x, y)
<code>public void drawLine(int x1, int y1, int x2, int y2)</code>	在 (x1, y1) 与 (x2, y2) 两点之间绘制一条线段
<code>public void drawRect(int x, int y, int width, int height)</code>	用指定的width和height绘制一个矩形，该矩形的左上角坐标为(x, y)
<code>public void fillRect(int x, int y, int width, int height)</code>	用指定的width和height绘制一个实心矩形，该矩形的左上角坐标为(x, y)

<code>public void clearRect(int x, int y, int width, int height)</code>	用指定的width和height，以当前背景色绘制一个 实心矩形 。该矩形的左上角坐标为 (x, y)
<code>public void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)</code>	用指定的width和height绘制一个 圆角矩形 ，圆角是一个椭圆的1/4弧，此椭圆由arcWidth、arcHeight确定两轴长。其外切矩形左上角坐标为 (x, y)
<code>public void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)</code>	用当前色绘制 实心圆角矩形 ，各参数含义同drawRoundRect。
<code>public void draw3DRect(int x, int y, int width, int height, boolean b)</code>	用指定的width和height绘制 三维矩形 ，该矩形左上角坐标是(x, y)，b为true时，该矩形为突出的，b为false时，该矩形为凹陷的。
<code>public void fill3DRect(int x, int y, int width, int height, boolean b)</code>	用当前色绘制 实心三维矩形 ，各参数含义同draw3DRect。

<code>public void drawPolygon(int[] xPoints, int [] yPoints, int nPoints)</code>	用xPoints, yPoints数组指定的点的坐标依次相连绘制 多边形 , 共选用前nPoints个点。
<code>public void fillPolygon(int[] xPoints, int [] yPoints, int nPoints)</code>	绘制 实心多边形 , 各参数含义同drawPolygon。
<code>public void drawOval(int x, int y, int width, int height)</code>	用指定的width和height, 以当前色绘制一个 椭圆 , 外切矩形的左上角坐标是(x, y)。
<code>public void fillOval(int x, int y, int width, int height)</code>	绘制 实心椭圆 , 各参数含义同drawOval。
<code>public void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)</code>	绘制指定width和height的 椭圆 , 外切矩形左上角坐标是(x, y), 但只截取从startAngle开始, 并扫过arcAngle度数的弧线。
<code>public void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)</code>	绘制一条 实心弧线 (即扇形) , 各参数含义同drawArc

例：使用Graphics类绘图



例：使用Graphics类绘图

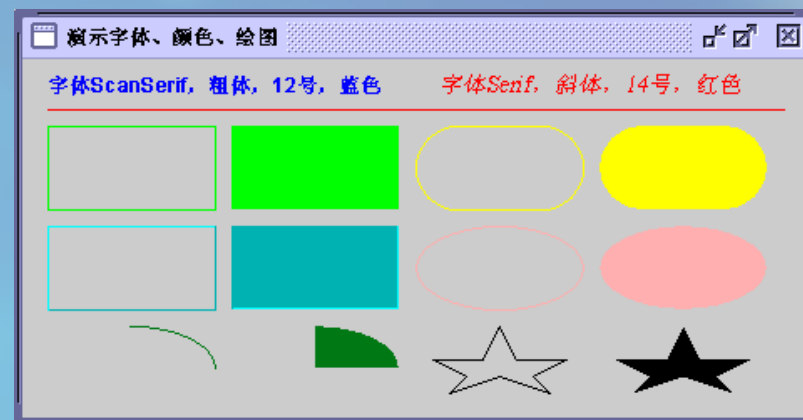
```
import java.awt.*;
import javax.swing.*;

public class GraphicsTester extends JFrame {
    public GraphicsTester ()
    { super( "演示字体、颜色、绘图" );
      setVisible( true ); //显示窗口
      setSize( 480, 250 ); //设置窗口大小
    }
    public void paint( Graphics g ) {
        super.paint( g );
        g.setFont( new Font( "SansSerif", Font.BOLD, 12 ) );
        g.setColor(Color.blue); //设置颜色
        g.drawString("字体ScanSerif，粗体，12号，蓝色",20,50);

        g.setFont( new Font( "Serif", Font.ITALIC, 14 ) );
        g.setColor(new Color(255,0,0));
        g.drawString( " 字体Serif，斜体，14号，红色", 250, 50 );

        g.drawLine(20,60,460,60); //绘制直线
    }
}
```

运行结果



Java2D API

- 提供了高级的二维图形功能。

Java2D API

- 分布在java.awt、java.awt.image、java.awt.color、java.awt.font、java.awt.geom、java.awt.print和java.awt.image.renderable包中。
- 它能轻松使你完成以下功能：
 - 可以很容易绘制各种形状；
 - 可以控制笔画：粗细、端头样式、虚线；
 - 可以用单色、渐变色和纹理填充形状；
 - 平移、旋转、伸缩、切变二维图形，对图像进行模糊、锐化等操作；
 - 构建重叠的文本和图形；
 - 可以对形状进行剪切，将其限制在任意区域内；
 -

Graphics2D 类

- 是Graphics类的抽象子类;
- 要使用Java2D API, 就必须建立该类的对象。

Graphics2D 类

- Graphics类的抽象子类。
- 传递给paint方法的对象是Graphics2D的一个子类实例，被向上转型为Graphics类的实例。要访问Graphics2D功能，必须将传递给paint方法的Graphics引用强制转换为Graphics2D引用：
`Graphics2D g2d=(Graphics2D)g`

例：使用Graphics2D类绘图

使用Java2D使文字出现渐变色效果

```
import java.awt.*;  
import javax.swing.*;  
public class Graphics2DTester extends JApplet{  
    public void paint(Graphics g) {  
        super.paint(g);  
        Graphics2D g2d=(Graphics2D)g;  
        g2d.setPaint(new GradientPaint(0,0,Color.red,180,45,Color.yellow));  
        g2d.drawString("This is a Java Applet!",25,25);  
    }  
}
```

运行结果：



结束语

Swing基础

前面介绍了如何在屏幕上绘制普通的图形，但如果需要绘制一个按钮，并使其可以对点击事件作出响应，就需要使用java Swing提供的组件

其实前面我们已经用到的JFrame、JApplet都是Swing组件，它们分别代表窗口组件和Applet容器组件

JFC 与 Swing

- JFC (Java Foundation Classes)
 - 是关于GUI 组件和服务的完整集合。
 - 作为JAVA SE 的一个有机部分，主要包含5 个部分
 - AWT
 - Java2D
 - Accessibility
 - Drag & Drop
 - Swing
- Swing
 - JFC 的一部分。
 - 提供按钮、窗口、表格等所有的组件。
 - 纯Java组件（完全用Java写的）。

AWT 组件

- 在java.awt包里，包括Button、Checkbox、Scrollbar等，都是Component类的子类。
- 大部分含有native code，所以随操作系统平台的不同会显示出不同的样子，而不能进行更改，是重量级组件。

Swing 组件

- 其名称都是在原来AWT组件名称前加上J，例如JButton、JCheckBox、JScrollbar等，都是JComponent类的子类。
- 架构在 AWT 之上，是AWT的扩展而不是取代。
- 完全是由java语言编写的，其外观和功能不依赖于任何由宿主平台的窗口系统所提供的代码，是轻量级组件。
- 可提供更丰富的视觉感受。

在Applet和Application中应用Swing

- 在Applet中应用Swing，就是要将Swing组件加载到Applet容器上（通常是JApplet），这通常在init方法中完成；
- 在Application中应用Swing，也是要将Swing组件加载到这个Application的顶层容器（通常是JFrame）中。

例：在Applet中应用Swing

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class SwingApplet extends JApplet{
    public void init() {
        Container contentPane=getContentPane();
        contentPane.setLayout(new GridLayout(2,1));
        JButton button=new JButton("Click me");
        final JLabel label=new JLabel();
        contentPane.add(button);
        contentPane.add(label);
        button.addActionListener( new ActionListener(){
            public void actionPerformed(ActionEvent event) {
                String information=JOptionPane.showInputDialog("请输入一串字符");
                label.setText(information);
            } } );//创建监听器语句结束
    } //init方法结束
}
```

例：在Applet中应用Swing (续)

运行结果

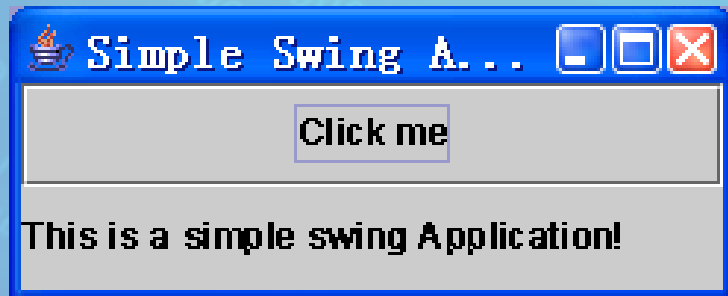
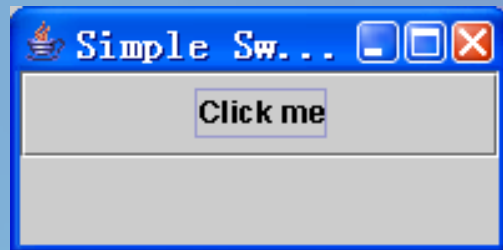


例：在Application中应用Swing

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class SwingApplication {
    public static void main(String[] args){
        JFrame f=new JFrame( "Simple Swing Application" );
        Container contentPane=f.getContentPane();
        contentPane.setLayout(new GridLayout(2,1));
        JButton button=new JButton("Click me");
        final JLabel label=new JLabel();
        contentPane.add(button);//添加按钮
        contentPane.add(label);//添加标签
        button.addActionListener( new ActionListener(){
            public void actionPerformed(ActionEvent event) {
                String information=JOptionPane.showInputDialog("请输入一串字符");
                label.setText(information);
            } } );
        f.setSize(200,100);//设置大小
        f.show();//显示
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

例：在Application中应用Swing (续)

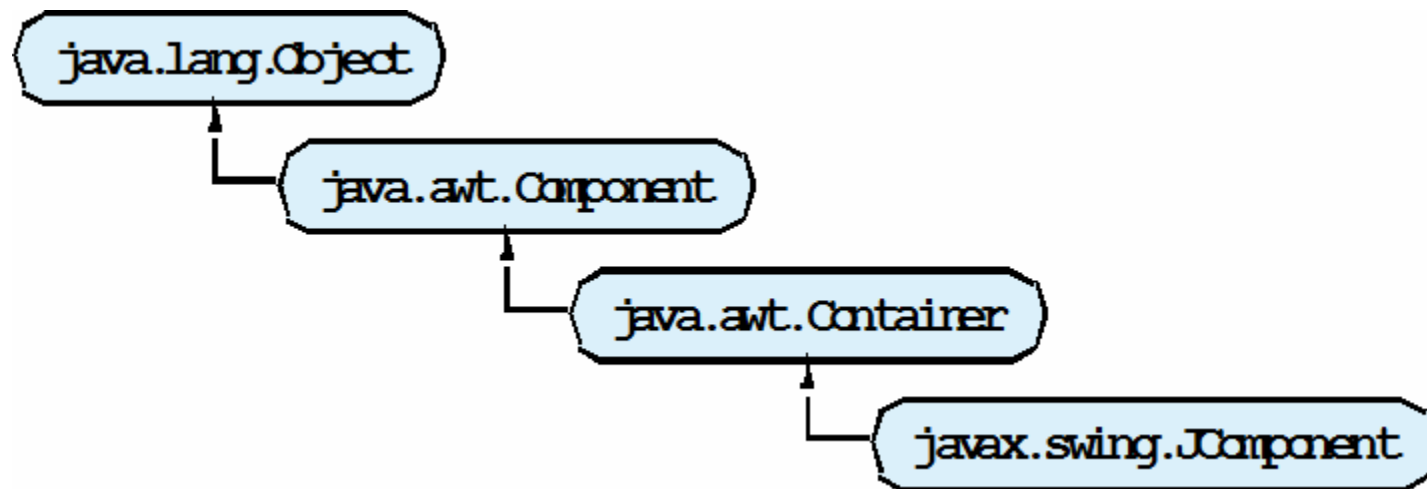
➤ 运行结果



结束语

Swing的层次

多数Swing组件的继承层次



JComponent类是除了顶层容器以外所有Swing组件的超类

- **Component 类**
 - 包含paint、repaint方法，可以在屏幕上绘制组件。
 - 大多数GUI组件直接或间接扩展Component。
- **Container 类**
 - 容纳相关组件。
 - 包括add方法，用来添加组件。
 - 包括setLayout方法，用来设置布局，帮助Container对象对其中的组件进行定位和设置组件大小。
- **JComponent 类——多数Swing组件的超类**
 - 可定制的观感，即可根据需求定制观感。
 - 快捷键 (通过键盘直接访问GUI组件)。
 - 一般的事件处理功能。

Swing的组件和容器层次

- 顶层容器
- 中间层容器
- 原子组件

顶层容器

- JFrame
- JDialog
- JApplet

顶层容器

- Swing的三个顶层容器的类
 - JFrame 实现单个主窗口。
 - JDialog 实现一个二级窗口(对话框)。
 - JApplet 在浏览器窗口中实现一个applet显示区域。
- 必须和操作系统打交道，所以都是重量级组件。
- 从继承结构上来看，它们分别是从原来AWT组件的Frame、Dialog和Applet类继承而来。
- 每个使用Swing组件的Java程序都必须至少有一个顶层容器，别的组件都必须放在这个顶层容器上才能显现出来。

中间层容器

- 是为了容纳别的组件。

中间层容器

- 分为两类
 - 一般用途的
 - JPanel
 - JScrollPane
 - JSplitPane
 - JTabbedPane
 - JToolBar
 - 特殊用途的
 - JInternalFrame
 - JRootPane
- 可以直接从顶层容器中获得一个JRootPane对象来直接使用，而别的中层容器使用的时候需要新建一个对象。

原子组件

- 通常是在图形用户界面中和用户进行交互的组件。

原子组件

- 显示不可编辑信息的
 - 例如：JLabel、JProgressBar、JToolTip
- 有控制功能、可以用来输入信息的
 - 例如：JButton、JCheckBox、JRadioButton、JComboBox、JList、JMenu、JSlider、JSpinner、JTextComponent等
- 能提供格式化的信息并允许用户选择的
 - 例如：JColorChooser、JFileChooser、JTable、JTree

例：三层容器结构

例：三层容器结构

```
import javax.swing.*;
import java.awt.*;
public class ComponentTester {
    public static void main(String[] args){
        JFrame.setDefaultLookAndFeelDecorated(true);
        JFrame frame=new JFrame("Swing Frame");
        Container contentPane=frame.getContentPane();
        JPanel panel=new JPanel();
        panel.setBorder(BorderFactory.createLineBorder(Color.black,5));
        panel.setLayout(new GridLayout(2,1));
        JLabel label=new JLabel("Label",SwingConstants.CENTER);
        JButton button=new JButton("Button");
        panel.add(label);
        panel.add(button);
        contentPane.add(panel);
        frame.pack();//对组件进行排列
        frame.show();//显示
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

运行结果：



常用Swing组件

Box	BoxLayout	JButton
JCheckBox	JCheckBoxMenuItem	JComboBox
JComponent	JDesktopPane	JDialog
JEditorPane	JFrame	JInternalFrame
JLabel	JLayeredPane	JList
JMenu	JMenuBar	JRadioButtonMenuItem
JRootPane	JScrollBar	JScrollPane
JSeparator	JSlider	JSplitPane
JTabbedPane	JTable	JTextArea
JTextField	JTextPane	JToggleButton
JToolBar	JToolTip	JTree
JViewport	JMenuItem	JOptionPane
JPasswordField	JPopupMenu	JProgressBar
JRadioButton	JWindow	OverlayLayout
ProgressMonitor	ProgressMonitorInputStream	Timer
UIDefaults	UIManager	

结束语

布局管理

如何将下级组件有秩序地摆在上一级容器中？

在程序中具体指定每个组件的位置

使用布局管理器（Interface LayoutManager）

布局管理器

- 调用容器对象的setLayout方法，并以布局管理器对象为参数，例如：

```
Container contentPane = frame.getContentPane();  
contentPane.setLayout(new FlowLayout());
```

- 使用布局管理器可以更容易地进行布局，而且当改变窗口大小时，它还会自动更新版面来配合窗口的大小，不需要担心版面会因此混乱。

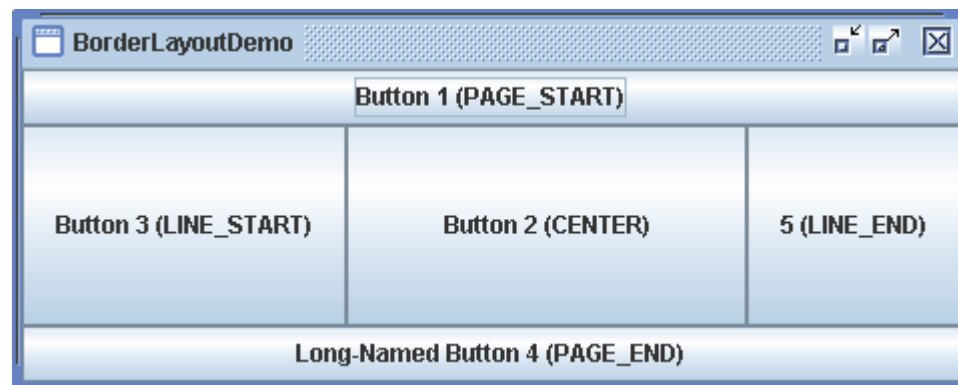
常用的布局管理器类

- 在Java中有很多实现LayoutManager接口的类，经常用到的有以下几个
 - BorderLayout
 - FlowLayout
 - GridLayout
 - CardLayout
 - GridBagLayout
 - BoxLayout
 - SpringLayout
 - 内容面板（content pane）默认使用的就是BorderLayout，它可以将组件放置到五个区域：东、西、南、北、中。

布局示例

布局示例： BorderLayout

- 将组件放置到五个区域：东、西、南、北、中



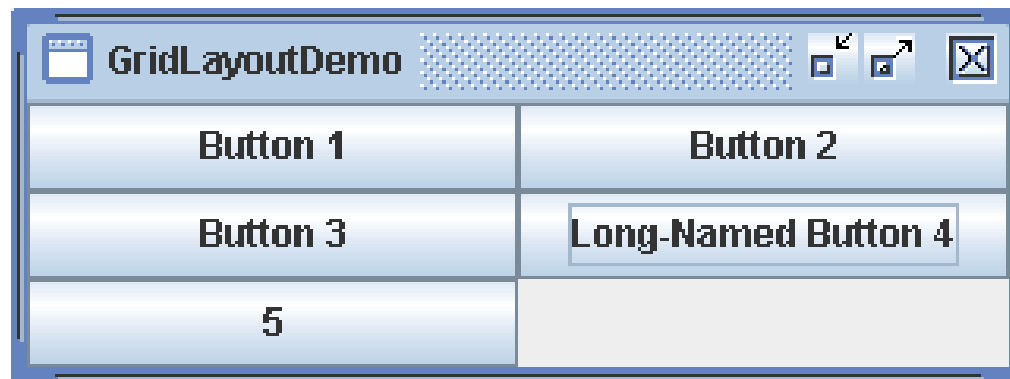
布局示例：FlowLayout

- 是JPanel默认使用的布局管理器，它只是简单地把组件放在一行，如果容器不是足够宽来容纳所有组件，就会自动开始新的一行



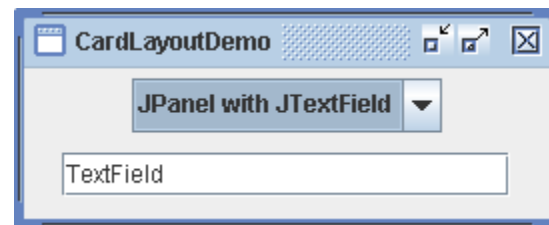
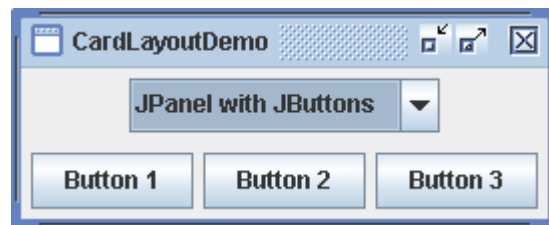
布局示例：GridLayout

- 按照指定的行数和列数将界面分为等大的若干块，组件被等大地按加载顺序放置其中。



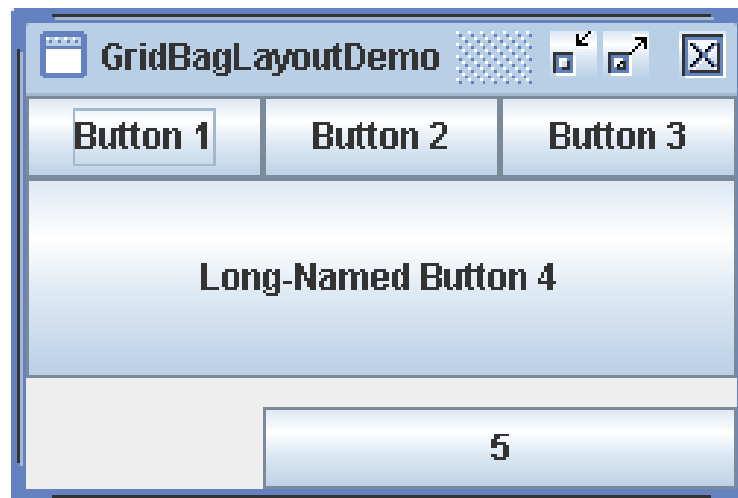
布局示例：CardLayout

- 可以实现在一个区域出现不同的组件布局，就像在一套卡片中选取其中的任意一张一样。它经常由一个复选框控制这个区域显示哪一组组件，可通过组合框像选择卡片一样选择某一种布局



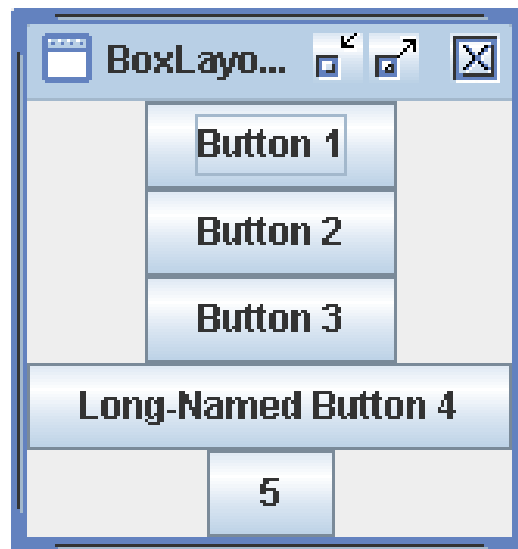
布局示例：GridBagLayout

- 把组件放置在网格中，这一点类似于GridLayout，但它的优点在于不仅能设置组件摆放的位置，还能设置该组件占多少行/列。这是一种非常灵活的布局管理器



布局示例：BoxLayout

- 将组件放在单一的行或列中，和FlowLayout不同的是，它可以考虑组件的对齐方式，最大、最小、优选尺寸



布局示例：SpringLayout

- 是一种灵活的布局管理器。它能够精确指定组件之间的间距。组件之间的距离通过Spring类的对象来表示，每个spring有四个属性，最小值，最大值，优选值和实际值。每个组件的spring对象集合在一起就构成了SpringLayout.Constraints对象



结束语

- 要说明代码实例在讲了事件处理以后给出

内部类

在另一个类或方法的定义中定义的匿名类

内部类

- 在另一个类或方法的定义中定义的类
- 可访问其外部类中的所有数据成员和方法成员
- 可对逻辑上相互联系类进行分组
- 对于同一个包中的其他类来说，能够隐藏
- 可非常方便地编写事件驱动程序
- 声明方式
 - 命名的内部类：可在类的内部多次使用
 - 匿名内部类：可在new关键字后声明内部类，并立即创建一个对象
- 假设外层类名为Myclass，则该类的内部类名为
 - Myclass\$c1.class (c1为命名的内部类名)
 - Myclass\$1.class (表示类中声明的第一个匿名内部类)

例：内部类

例：内部类

```
public class Parcel1 {  
    class Contents { //内部类  
        private int i = 11;  
        public int value() { return i; }  
    }  
    class Destination { //内部类  
        private String label;  
        Destination(String whereTo) { label = whereTo; }  
        String readLabel() { return label; }  
    }  
    public void ship(String dest) {  
        //生成两个内部类对象，并调用了内部类中声明的一个方法  
        Contents c = new Contents();  
        Destination d = new Destination(dest);  
        System.out.println(d.readLabel());  
    }  
}
```

```
public static void main(String[] args)  
{  
    Parcel1 p = new Parcel1();  
    p.ship("Tanzania");  
}
```

注：本例题引自《Java编程思想》

例：外部类方法返回内部类的引用

例：外部类方法返回内部类的引用

```
public class Parcel2 {  
    class Contents {  
        private int i = 11;  
        public int value() { return i; }  
    }  
    class Destination {  
        private String label;  
        Destination(String whereTo) { label = whereTo; }  
        String readLabel() { return label; }  
    }  
    public Destination to(String s)    //to()方法返回内部类Destination的引用  
    { return new Destination(s); }  
    public Contents cont() { return new Contents(); } //cont()方法返回内部类Contents的引用  
}
```

注：本例题引自《Java编程思想》

例：外部类方法返回内部类的引用

```
public void ship(String dest) {  
    Contents c = cont();  
    Destination d = to(dest);  
    System.out.println(d.readLabel());  
}  
public static void main(String[] args) {  
    Parcel2 p = new Parcel2();  
    p.ship("Tanzania");  
    Parcel2 q = new Parcel2();  
    Parcel2.Contents c = q.cont();  
    Parcel2.Destination d = q.to("Borneo");  
}  
}
```

内部类实现接口、继承抽象类

- 可以完全不被看到，而且不能被调用。
- 可以方便实现“隐藏实现细则”，外部能得到的仅仅是指向超类或者接口的一个引用。

例：内部类实现接口、继承抽象类

首先假设有如下抽象类和接口

```
abstract class Contents {  
    abstract public int value();  
}  
  
interface Destination {  
    String readLabel();  
}
```

例：内部类实现接口、继承抽象类

```
public class Parcel3 {  
    private class PContents extends Contents {  
        private int i = 11;  
        public int value() { return i; }  
    }  
    protected class PDestination implements Destination {  
        private String label;  
        private PDestination(String whereTo) { label = whereTo;}  
        public String readLabel() { return label; }  
    }  
    public Destination dest(String s) { return new PDestination(s); }  
    public Contents cont() { return new PContents(); }  
}
```

注：本例题引自《Java编程思想》

例：内部类实现接口、继承抽象类

```
class Test {  
    public static void main(String[] args) {  
        Parcel3 p = new Parcel3();  
        Contents c = p.cont();  
        Destination d = p.dest("Tanzania");  
    }  
}
```

➤ 说明

- 内部类PContents实现了抽象了Contents
- 内部类PDestination实现了接口Destination
- 外部类Test中不能声明对private的内部类的引用

局部作用域中的内部类

- 实现某个接口，产生并返回一个引用
- 为解决一个复杂问题，需要建立一个类，而又不想它为外界所用

例：方法中的内部类

```
public class Parcel4 {  
    public Destination dest(String s) {  
        class PDestination implements Destination {  
            private String label;  
            private PDestination(String whereTo) {  
                label = whereTo;  
            }  
            public String readLabel() { return label; }  
        }  
        return new PDestination(s);  
    }  
    public static void main(String[] args) {  
        Parcel4 p = new Parcel4();  
        Destination d = p.dest("Tanzania");  
    }  
}
```

注：本例题引自《Java编程思想》

例：块作用域中的内部类

```
public class Parcel5 {  
    private void internalTracking(boolean b) {  
        if(b) {  
            class TrackingSlip {  
                private String id;  
                TrackingSlip(String s) { id = s; }  
                String getSlip() { return id; }  
            }  
            TrackingSlip ts = new TrackingSlip("slip");  
            String s = ts.getSlip();  
        }  
    }  
    public void track() { internalTracking(true); }  
    public static void main(String[] args) {  
        Parcel5 p = new Parcel5();  
        p.track();  
    }  
}
```

注：本例题引自《Java编程思想》

匿名的内部类

```
public class Parcel6 {  
    public Contents cont() {  
        return new Contents() {  
            private int i = 11;  
            public int value() { return i; }  
        };  
    }  
    public static void main(String[] args) {  
        Parcel6 p = new Parcel6();  
        Contents c = p.cont();  
    }  
}
```

注：本例题引自《Java编程思想》

结束语

- 说明简单了解即可，主要是为了理解在事件处理时用到匿名内部类

事件处理的基本概念

GUI是由事件驱动的

常见的事件包括

- 移动鼠标
- 单双击鼠标各个按钮
- 单击按钮
- 在文本字段输入
- 在菜单中选择菜单项
- 在组合框中选择、单选和多选
- 拖动滚动条
- 关闭窗口
-
- Swing通过事件对象来包装事件，程序可以通过事件对象获得事件的有关信息

事件处理的几个要素

- 事件源
- 事件监听器
- 事件对象

事件处理的几个要素

- 事件源
 - 与用户进行交互的GUI组件，表示事件来自于哪个组件或对象
 - 比如要对按钮被按下这个事件编写处理程序，按钮就是事件源
- 事件监听器
 - 负责监听事件并做出响应
 - 一旦它监视到事件发生，就会自动调用相应的事件处理程序作出响应
- 事件对象
 - 封装了有关已发生的事件的信息
 - 例如按钮被按下就是一个要被处理的事件，当用户按下按钮时，就会产生一个事件对象。事件对象中包含事件的相关信息和事件源

- 程序员应完成的两项任务
 - 为事件源注册一个事件监听器
 - 实现事件处理方法

事件源

- 提供注册监听器或取消监听器的方法

事件源

- 提供注册监听器或取消监听器的方法
- 如有事件发生，已注册的监听器就会被通知
- 一个事件源可以注册多个事件监听器，每个监听器又可以对多种事件进行相应，例如一个JFrame事件源上可以注册
 - 窗口事件监听器，响应：
 - 窗口关闭
 - 窗口最大化
 - 窗口最小化
 - 鼠标事件监听器，响应：
 - 鼠标点击
 - 鼠标移动

事件监听器

- 是一个对象，通过事件源的`addXXXListener`方法被注册到某个事件源上
- 不同的Swing组件可以注册不同的事件监听器
- 一个事件监听器中可以包含有对多种具体事件的专用处理方法
 - 例如用于处理鼠标事件监听器接口MouseListener中就包含有对应于鼠标压下、放开、进入、离开、敲击五种事件的相应方法`mousePressed`、`mouseReleased`、`mouseEntered`、`mouseExited`、`mouseClicked`，这五种方法都需要一个事件对象作为参数

事件对象

- 常用的事件对象

- **ActionEvent**

- 发生在按下按钮、选择了一个项目、在文本框中按下回车键

- **ItemEvent**

- 发生在具有多个选项的组件上，如JCheckBox、JComboBox

- **ChangeEvent**

- 用在可设定数值的拖曳杆上，例如JSlider、JProgressBar等

- **WindowEvent**

- 用在处理窗口的操作

- **MouseEvent**

- 用于鼠标的操作

常用的Swing事件源可能触发的事件及事件监听器

常用的Swing事件源

事件源	事件对象	事件监听器
JFrame	MouseEvent WindowEvent	MouseListener WindowEventListener
AbstractButton (JButton, JToggleButton, JCheckBox, JRadioButton)	ActionEvent ItemEvent	ActionListener ItemListener
JTextField JPasswordField	ActionEvent UndoableEvent	ActionListener UndoableListener
JTextArea	CareEvent InputMethodEvent	CareListener InputMethodEventListener
JTextPane JEditorPane	CareEvent DocumentEvent UndoableEvent HyperlinkEvent	CareListener DocumentListener UndoableListener HyperlinkListener

常用的Swing事件源

JComboBox	ActionEvent ItemEvent	ActionListener ItemListener
JList	ListSelectionEvent ListDataEvent	ListSelectionListener ListDataListener
JFileChooser	ActionEvent	ActionListener
JMenuItem	ActionEvent ChangeEvent ItemEvent MenuKeyEvent MenuDragMouseEvent	ActionListener ChangeListener ItemListener MenuKeyListener MenuDragMouseListener
JMenu	MenuEvent	MenuListener
JPopupMenu	PopupMenuEvent	PopupMenuListener

常用的Swing事件源

JProgressBar	ChangeEvent	ChangeListener
JSlider	ChangeEvent	ChangeListener
JScrollBar	AdjustmentEvent	AdjustmentListener
JTable	ListSelectionEvent TableModelEvent	ListSelectionListener TableModelListener
JTabbedPane	ChangeEvent	ChangeListener
JTree	TreeSelectionEvent TreeExpansionEvent	TreeSelectionListener TreeExpansionListener
JTimer	ActionEvent	ActionListener

接口与适配器

- 事件监听器接口
 - 规定实现各种处理功能接口。
- 事件监听器适配器类
 - 有时我们并不需要对所有事件进行处理，为此Swing提供了一些适配器类
×××Adapter。

接口与适配器

- 事件监听器接口
 - 例如MouseListener是一个接口，为了在程序中创建一个鼠标事件监听器的对象，我们需要实现其所有五个方法，在方法体中，我们可以通过鼠标事件对象传递过来的信息（例如点击的次数，坐标），实现各种处理功能。
- 事件监听器适配器类
 - 有时我们并不需要对所有事件进行处理，为此Swing提供了一些适配器类×××Adapter，这些类含有所有×××Listener中方法的默认实现（就是什么也不做），因此我们就只需编写那些需要进行处理的事件的方法。例如，如果只想对鼠标敲击事件进行处理，如果使用MouseAdapter类，则只需要重写mouseClicked方法就可以了。

事件处理

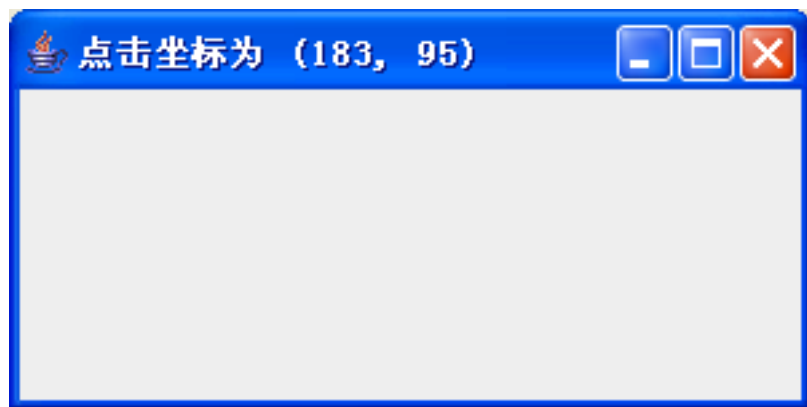
- 实现事件监听器接口
- 继承事件监听器适配器类
- 使用匿名内部类
- lambda表达式

事件处理

- 实现事件监听器接口
 - 这种方法需要实现接口中所有的方法，对我们不需要进行处理的事件方法，也要列出来，其方法体使用一对空的花括号
- 继承事件监听器适配器类
 - 只需要重写我们感兴趣的事件
- 使用匿名内部类
 - 特别适用于已经继承了某个父类（例如Applet程序，主类必须继承JApplet类或Applet类），则根据java语法规则，就不能再继承适配器类的情况，而且使用这种方法程序看起来会比较清楚明了
- lambda表达式
 - 对于只有一个抽象方法的函数式监听器接口，也可以使用lambda表达式。

例：

- 创建一窗口，当鼠标在窗口中点击时，在窗口标题栏中显示点击位置坐标。



- 方法一：实现MouseListener接口。
- 方法二：继承MouseAdapter类。
- 方法三：使用匿名内部类。

例：实现MouseListener接口

```
import java.awt.event.*; //载入MouseListener类所在的包
import javax.swing.*; //载入JFrame所在的包
public class ImplementMouseListener implements MouseListener{
    JFrame f;
    public ImplementMouseListener () {
        f=new JFrame(); //新建一窗口
        f.setSize(300,150);
        f.show();
        f.addMouseListener(this); //为窗口增加鼠标事件监听器
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public void mousePressed(MouseEvent e){}
    public void mouseReleased(MouseEvent e){}
    public void mouseEntered(MouseEvent e){}
    public void mouseExited(MouseEvent e){}
    public void mouseClicked(MouseEvent e){
        f.setTitle("点击坐标为 (" +e.getX()+", " +e.getY());
    }
    public static void main(String[] args){ new ImplementMouseListener ();}
}
```

例：继承MouseListener类

```
import java.awt.event.*; //载入MouseListener所在的包
import javax.swing.*;

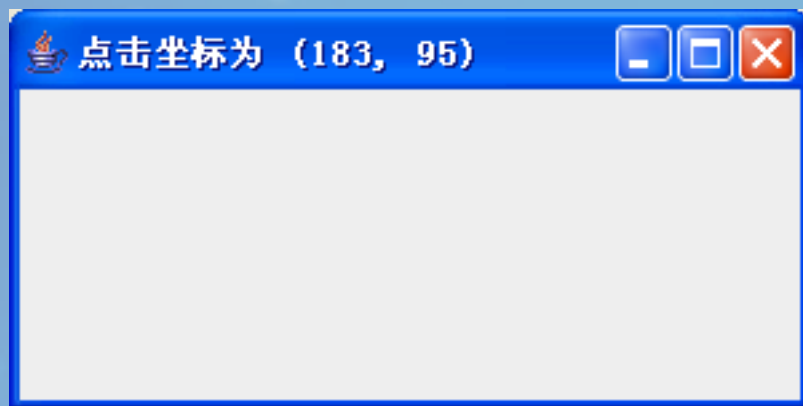
public class ExtendMouseListener extends MouseAdapter{
    JFrame f;
    public ExtendMouseListener () {
        f=new JFrame();
        f.setSize(300,150);
        f.show();
        f.addMouseListener(this);
        f.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
    }
    public void mouseClicked(MouseEvent e){
        f.setTitle("点击坐标为 (" +e.getX()+", " +e.getY()+")");
    }
    public static void main(String[] args){ new ExtendMouseListener();}
}
```


例：使用匿名内部类

```
import java.awt.event.*;
import javax.swing.*;
public class UseInnerClass {
    JFrame f;
    public UseInnerClass () {
        f=new JFrame();
        f.setSize(300,150);
        f.show();
        f.addMouseListener(new MouseAdapter(){
            public void mouseClicked(MouseEvent e){
                f.setTitle("点击坐标为 (" +e.getX()+", " +e.getY()+")");
            }
        }); //为窗口添加鼠标事件监听器语句结束
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args){ new UseInnerClass (); }
}
```


运行结果

- 采用不同方法的程序，其运行效果都是一样的，当鼠标在窗口中点击的时候，窗口标题栏将出现所点位置的坐标信息



结束语

事件派发机制

事件派发机制——事件派发线程

- Swing中的组件是非线程安全的，在Swing中专门提供了一个事件派发线程（EDT）用于对组件的安全访问。

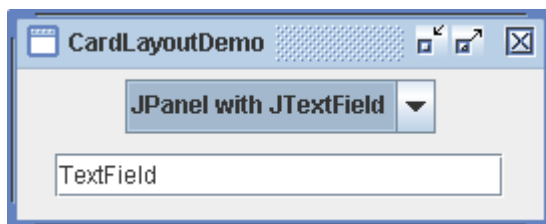
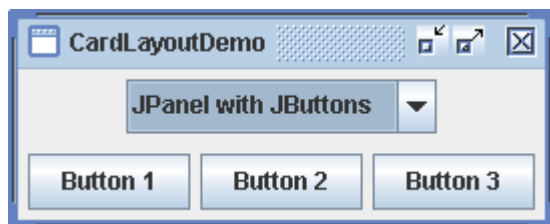
事件派发机制——事件派发线程

- **Swing**中的组件是非线程安全的，在**Swing**中专门提供了一个事件派发线程（**EDT**）用于对组件的安全访问。
 - 用来执行组件事件处理程序的线程（如按钮的点击事件），依次从系统事件队列取出事件并处理，一定要执行完上一个事件的处理程序后，才会处理下一个事件。
 - 事件监听器的方法都是在事件派发线程中执行的，如**ActionListener**的**actionPerformed**方法。

事件派发机制——由事件派发线程启动GUI

- 可以调用`invokeLater`或`invokeAndWait`请事件分发线程以运行某段代码
 - 要将这段代码放入一个`Runnable`对象的`run`方法中，并将该`Runnable`对象作为参数传递给`invokeLater`
- `invokeLater`是异步的，不用等代码执行完就返回。
- `invokeAndWait`是同步的，要等代码执行完才返回。调用时要避免死锁。

重温布局示例：CardLayout



例：CardLayout

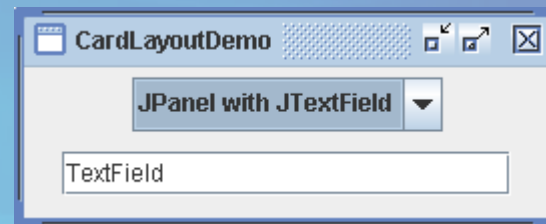
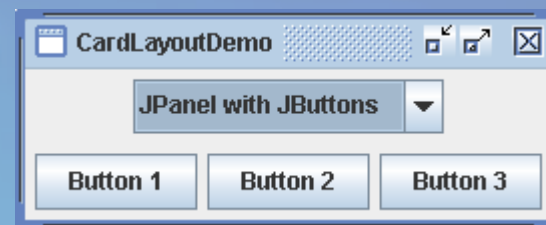
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class CardLayoutDemo implements ItemListener {
    JPanel cards;
    final static String BUTTONPANEL = "JPanel with JButtons";
    final static String TEXTPANEL = "JPanel with JTextField";

    public void addComponentToPane(Container pane) {
        //将JComboBox放进JPanel
        JPanel comboBoxPane = new JPanel(); //默认使用 FlowLayout
        String comboBoxItems[] = { BUTTONPANEL, TEXTPANEL };
        JComboBox cb = new JComboBox(comboBoxItems);
        cb.setEditable(false);
        cb.addItemListener(this);
        comboBoxPane.add(cb);

        JPanel card1 = new JPanel();
```

➤ 运行结果：



结束语

第7章 图形用户界面（二）

郑 莉

顶层容器

容器层次结构

是一个以顶层容器为根的树状组件集合

为了显示在屏幕上，每个组件必须是一套容器层次结构的一部分

每个组件只能放置在某个容器内一次

如果某个组件已经在容器中，又将它加到另外一个容器中，这个组件就会从第一个容器中清除

Swing的3个顶层容器类

- JFrame、JApplet、JDialog。
- 都是重量级组件，分别继承了AWT组件Frame、Applet和Dialog。
- 每个顶层容器都有一个内容面板，通常直接或间接的容纳别的可视组件。
- 可以有选择地为顶层容器添加菜单，菜单被放置在顶层容器上，但是在内容面板之外。

JFrame的继承结构

java.lang.Object

└ **java.awt.Component**

└ **java.awt.Container**

└ **java.awt.Window**

└ **java.awt.Frame**

└ **javax.swing.JFrame**

JApplet的继承结构

```
java.lang.Object
└ java.awt.Component
  └ java.awt.Container
    └ java.awt.Panel
      └ java.awt.Applet
        └ javax.swing.JApplet
```

JDialog的继承结构

java.lang.Object

└ **java.awt.Component**

└ **java.awt.Container**

└ **java.awt.Window**

└ **java.awt.Dialog**

└ **javax.swing.JDialog**

如何获得一个顶层容器

- JApplet类的顶层容器由浏览器提供，通常不需要自己产生JApplet类的对象。
- JFrame和JDialog对象需要创建并通过构造方法初始化。

构造方法

名称	描述
<code>JFrame()</code>	建立一个新的JFrame，默认是不可见的
<code>JFrame(String title)</code>	建立一个具有标题的JFrame，默认是不可见的
<code>JApplet()</code>	建立一个JApplet
<code>JDialog()</code>	建立一个non-modal对话框，无标题
<code>JDialog(Dialog owner)</code>	建立一个属于Dialog组件的对话框，为non-modal形式，无标题
<code>JDialog(Dialog owner, boolean modal)</code>	建立一个属于Dialog组件的对话框，可决定modal形式，无标题
<code>JDialog(Dialog owner, String title)</code>	建立一个属于Dialog组件的对话框，为non-modal形式，有标题

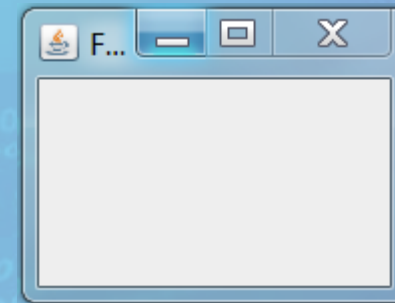
构造方法

<code>JDialog(Dialog owner, String title, boolean modal)</code>	建立一个属于Dialog组件的对话框，可决定modal形式，有标题
<code>JDialog(Frame owner)</code>	建立一个属于Frame组件的对话框，为non-modal形式，无标题
<code>JDialog(Frame owner, boolean modal)</code>	建立一个属于Frame组件的对话框，可决定modal形式，无标题
<code>JDialog(Frame owner, String title)</code>	建立一个属于Frame组件的对话框，为non-modal形式，有标题
<code>JDialog(Frame owner, String title, boolean modal)</code>	建立一个属于Frame组件的对话框，可决定modal形式，有标题

三个顶层容器的例子

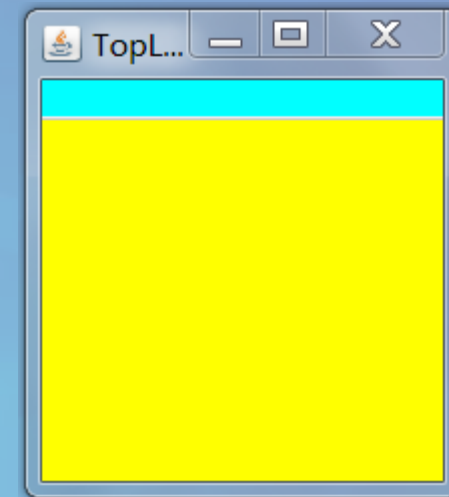
例：FrameDemo.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class FrameDemo {
    public static void main(String s[]) {
        JFrame frame = new JFrame("FrameDemo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel emptyLabel = new JLabel("");
        emptyLabel.setPreferredSize(new Dimension(175, 100));
        frame.getContentPane().add(emptyLabel, BorderLayout.CENTER);
        frame.pack();
        frame.setVisible(true);
    }
}
```



例：TopLevelDemo.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class TopLevelDemo {
    public static void main(String s[]) {
        JFrame frame = new JFrame("TopLevelDemo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel yellowLabel = new JLabel("");
        yellowLabel.setOpaque(true);
        yellowLabel.setBackground(Color.yellow);
        yellowLabel.setPreferredSize(new Dimension(200, 180));
        JMenuBar cyanMenuBar = new JMenuBar();
        cyanMenuBar.setOpaque(true);
        cyanMenuBar.setBackground(Color.cyan);
        cyanMenuBar.setPreferredSize(new Dimension(200, 20));
        frame.setJMenuBar(cyanMenuBar);
        frame.getContentPane().add(yellowLabel, BorderLayout.CENTER);
        frame.pack();
        frame.setVisible(true);
    }
}
```



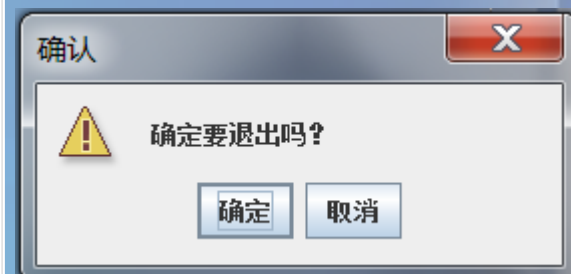
例：JOptionPaneDemo.java

- 通过静态方法 `showxxxDialog`，可以产生四种简单的对话框
 - 它们的方法参数中绝大部分(除了输入对话框可以不指定父窗口)都需要提供一个父窗口组件 `ParentComponent`，只有关闭这些简单的对话框后，才可以返回到其父窗口，也就是说，它们绝大部分都是模态的

例：JOptionPaneDemo.java

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

public class JOptionPaneDemo extends JFrame implements ActionListener{
    public JOptionPaneDemo(){
        super("简单对话框");
        Container c=getContentPane();
        JButton button=new JButton("退出");
        button.addActionListener(this);
        c.setLayout(new FlowLayout());
        c.add(button);
    }
    public void actionPerformed(ActionEvent e){
        //弹出对话框，并用变量select记录用户做的选择
        int select=JOptionPane.showConfirmDialog(this, "确定要退出吗？",
            "确认", JOptionPane.OK_CANCEL_OPTION, JOptionPane.WARNING_MESSAGE);
    }
}
```



结束语

- 这一节我们了解了Swing的顶层容器

中间层容器（一）

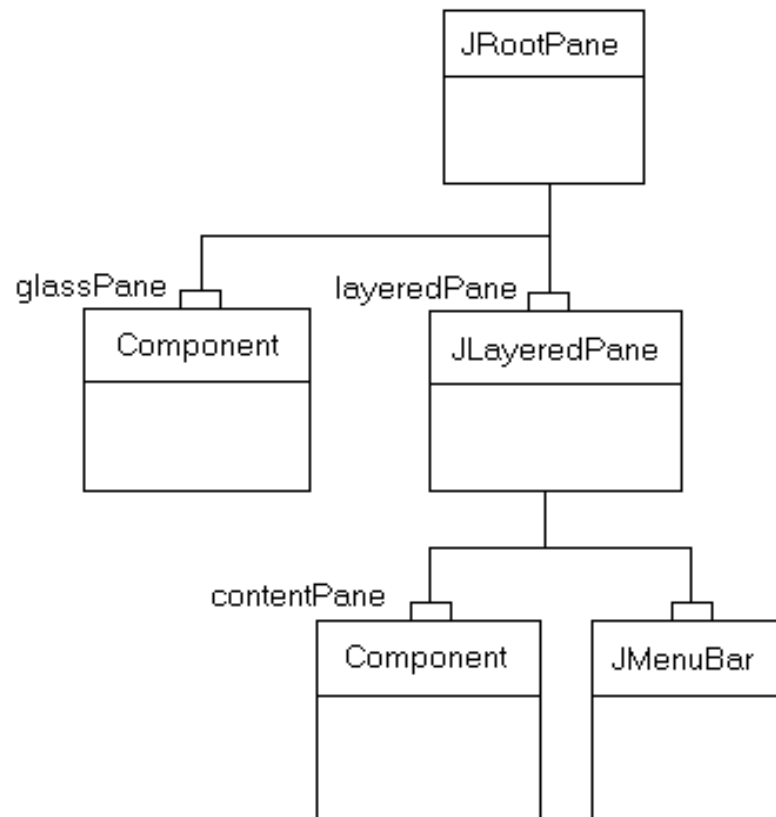
中间层容器存在的目的仅仅是为了容纳别的组件

中间层容器

- 一般用途的
 - JPanel
 - JScrollPane
 - JSplitPane
 - JTabbedPane
 - JToolBar
- 特殊用途的
 - JInternalFrame
 - JRootPane（可以直接从顶层容器中获得）

JRootPane的层次结构

- glassPane
 - 默认状态下是隐藏的。
 - 可以使用glassPane来截获所有要到达JRootPane别的部分的事件。
- layeredPane
 - 分为很多层（layer），每层都有一个代表层深度的整数值（Z-order），深度值高的组件将覆盖在深度值低的组件上。
 - contentPane
 - 一般我们将所有组件添加到contentPane上。
 - JMenuBar
 - 是可选的，如果没有，contentPane就会充满整个顶层容器。



JPanel

- 一种经常使用的轻量级中间容器。

JPanel

- 在默认状态下，除了背景色外不绘制任何东西。
- 可以很容易的为它设置边框和绘制特性，有效地利用JPanel可以使版面管理更为容易。
- 可以使用布局管理器来规划它所容纳的组件的位置和大小
 - 可以通过setLayout方法来改变其布局。
 - 也可以在创建一个JPanel对象时就为它确定某种布局方式。在默认状态下panel使用FlowLayout布局，将各组件布局在一行。

JPanel类 常用API

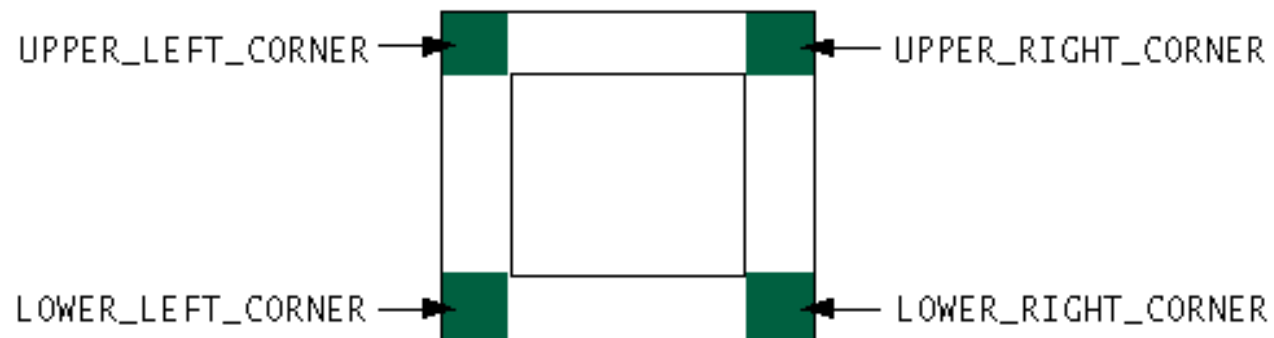
名称	说明
<code>JPanel()</code>	创建一个JPanel，默认布局是FlowLayout
<code>JPanel(LayoutManager layout)</code>	创建一个指定布局的JPanel
<code>void add(Component comp)</code>	添加组件
<code>void add(Component comp, int index)</code>	把组件添加到特定位置上
<code>int getComponentCount()</code>	获得这个panel里所有组件的数量
<code>Component getComponent(int index)</code>	获得指定序号的某个组件
<code>Component getComponentAt(int x, int y)</code>	获得指定位置的某个组件
<code>void remove(Component)</code>	移除某个组件
<code>void removeAll()</code>	移除所有组件
<code>void setLayout(LayoutManager layout)</code>	设置布局
<code>LayoutManager getLayout()</code>	得到现有布局
<code>void setBorder(Border border)</code>	设置边框

JScrollPane

JScrollPane

- JScrollPane容器

- 容器有滚动条，通过拖动滑块，就可以看到更多的内容。
- 由九个部分组成，包括一个中心显示地带、四个角和四条边



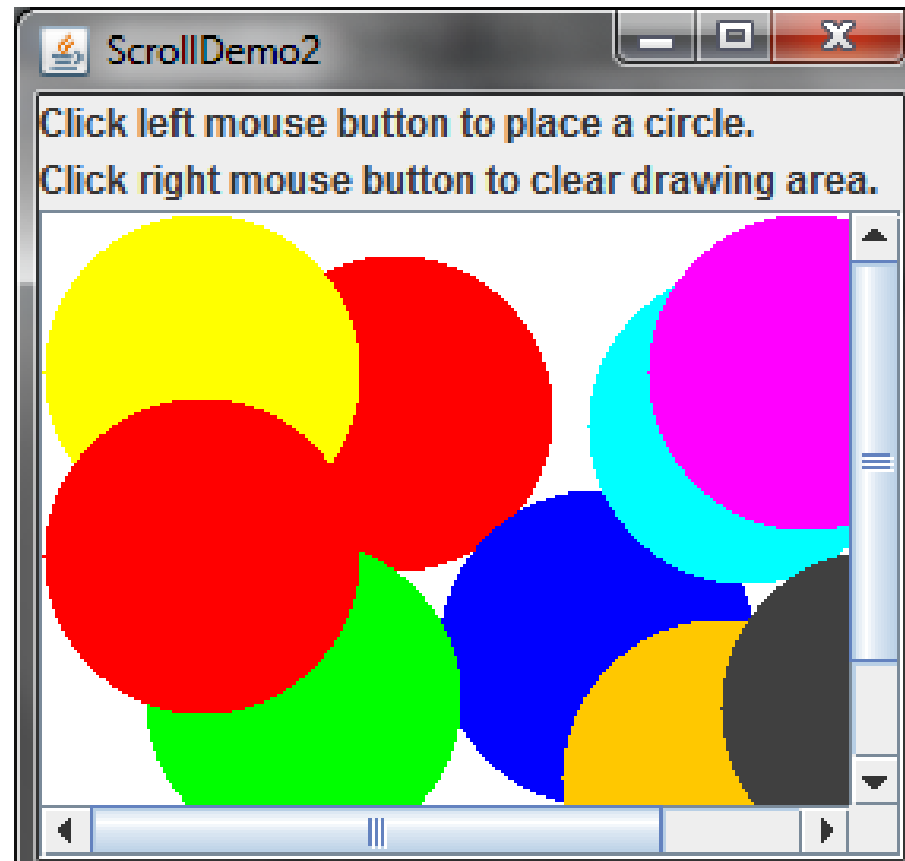
JScrollPane 常用API

名称	说明
<code>static int HORIZONTAL_SCROLLBAR_ALWAYS</code>	水平滚动条策略常数：总是显示
<code>static int HORIZONTAL_SCROLLBAR_AS_NEEDED</code>	水平滚动条策略常数：当显示内容水平区域大于显示区域时才出现
<code>static int HORIZONTAL_SCROLLBAR_NEVER</code>	水平滚动条策略常数：总是不显示
<code>static int VERTICAL_SCROLLBAR_ALWAYS</code>	垂直滚动条策略常数：总是显示
<code>static int VERTICAL_SCROLLBAR_AS_NEEDED</code>	垂直滚动条策略常数：当显示内容垂直区域大于显示区域时才出现
<code>static int VERTICAL_SCROLLBAR_NEVER</code>	垂直滚动条策略常数：总是不显示
<code>JScrollPane()</code>	建立一个空的JScrollPane对象
<code>JScrollPane(Component view)</code>	建立一个显示组件的JScrollPane对象，当组件内容大于显示区域时，自动产生滚动条

JScrollPane 常用API

<code>void setViewportView(Component)</code>	设置JScrollPane中心地带要显示的组件
<code>void setVerticalScrollBarPolicy(int)</code> <code>int getVerticalScrollBarPolicy()</code>	设置或者读取垂直滚动条策略常数，参数为本类的静态常量
<code>void setHorizontalScrollBarPolicy(int)</code> <code>int getHorizontalScrollBarPolicy()</code>	设置或者读取水平滚动条策略常数，参数为本类的静态常量
<code>void setViewportBorder(Border)</code> <code>Border getViewportBorder()</code>	设置或者读取中心显示地带的边框
<code>void setWheelScrollingEnabled(Boolean)</code> <code>Boolean isWheelScrollingEnabled()</code>	设置或者读取是否随着鼠标滚轮滚动出现或隐藏滚动条，默认状态下为真
<code>void setColumnHeaderView(Component)</code> <code>void setRowHeaderView(Component)</code>	设置显示在上面的边上的组件 设置显示在左面的边上的组件
<code>void setCorner(String key, Component corner)</code>	设置要显示在指定角上的组件，key表示角的字符串
<code>Component getCorner(String key)</code>	获得指定角上的组件

例：ScrollDemo2.Java



例：ScrollDemo2.Java

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

public class ScrollDemo2 extends JPanel {
    private Dimension size; // indicates size taken up by graphics
    private Vector objects; // rectangular coordinates used to draw graphics

    private final Color colors[] = {
        Color.red, Color.blue, Color.green, Color.orange,
        Color.cyan, Color.magenta, Color.darkGray, Color.yellow};
    private final int color_n = colors.length;

    JPanel drawingArea;

    public ScrollDemo2() {
```

结束语

- 这一节我们了解了：
 - JRootPane的层次结构
 - JPanel
 - JScrollPane

中间层容器（二）

JSplitPane



JSplitPane

- 可以把两个组件显示在两个显示区域内，且随着区域间分隔线的拖动，区域内组件的大小也随之发生变动。
- 它允许设置水平分割或者垂直分割；也允许设置动态拖曳功能（拖动分界线时两边组件是否会随着拖曳动态改变大小还是在拖曳结束后才改动）。
- 我们通常先把组件放到Scroll Pane中，再把Scroll Pane放到Split Pane中。这样在每部分窗口中，都可以拖动滚动条看到组件的全部内容。

JSplitPane 常用API

名称	说明
<code>static int HORIZONTAL_SPLIT</code>	水平分割常数
<code>static int VERTICAL_SPLIT</code>	垂直分割常数
<code>JSplitPane()</code>	创建一个JSplitPane，以水平方向排列，两边各是一个button，没有动态拖曳功能
<code>JSplitPane(int newOrientation)</code>	建立一个指定分割方向的JSplitPane，没有动态拖曳功能，参数为两个分割常数之一
<code>JSplitPane(int newOrientation, Boolean newContinuousLayout)</code>	指定分割方向，并可指定是否有动态拖曳功能
<code>JSplitPane(int newOrientation, Boolean newContinuousLayout, Component newLeftComponent, Component newRightComponent)</code>	指定分割方向、是否具有动态拖曳功能，和两侧组件

JSplitPane 常用API

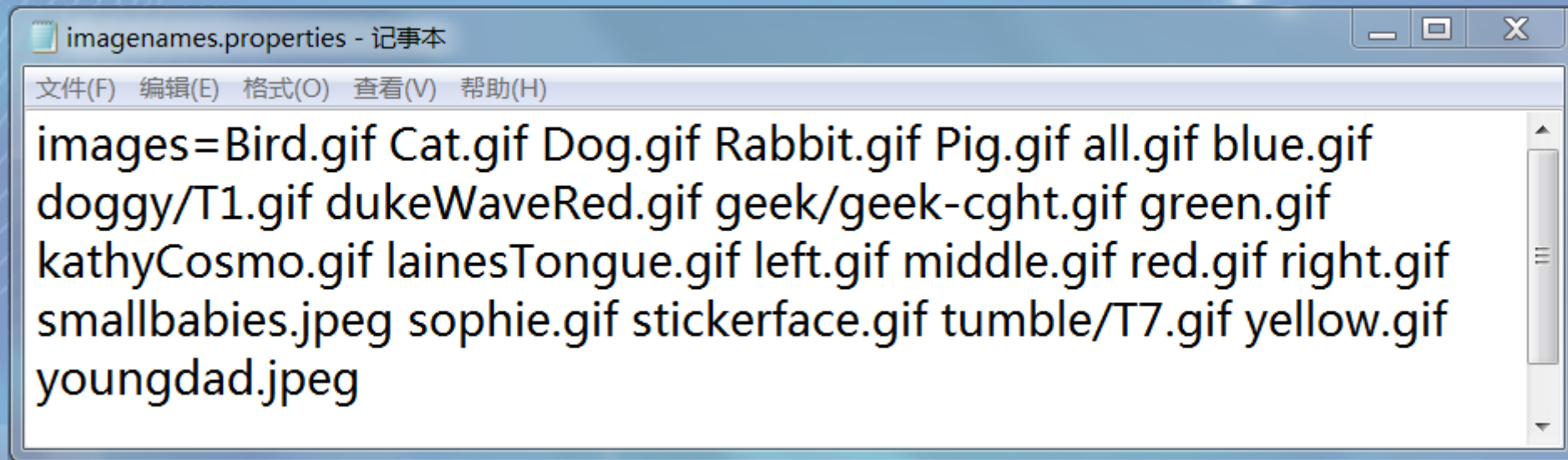
<code>JSplitPane(int newOrientation, Component newLeftComponent, Component newRightComponent)</code>	指定分割方向和两侧组件，无自动拖曳功能
<code>void setOrientation(int newOrientation) int getOrientation()</code>	设置或获得分割方向
<code>void setDividerSize(int) int getDividerSize()</code>	设置或读取分隔线条的粗细
<code>void setContinuousLayout(boolean nCL) boolean isContinuousLayout()</code>	设置或读取是否使用动态拖曳功能
<code>void setOneTouchExpandable(Boolean oTE) boolean is OneTouchExpandable()</code>	设置或读取是否在分隔线上显示一个控键来完全扩展和完全压缩单侧内容。
<code>void remove(Component comp) void add(Component comp)</code>	删除或添加组件。只可以添加两个组件
<code>void setDividerLocation(double) void setDividerLocation(int) int getDividerLocation()</code>	设置或读取分隔线条的位置。设置参数可以是double型的百分数，也可以是int型的像素值

例：SplitPaneDemo.java



例：SplitPaneDemo.Java

➤ imagenames.properties



例：SplitPaneDemo.Java

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

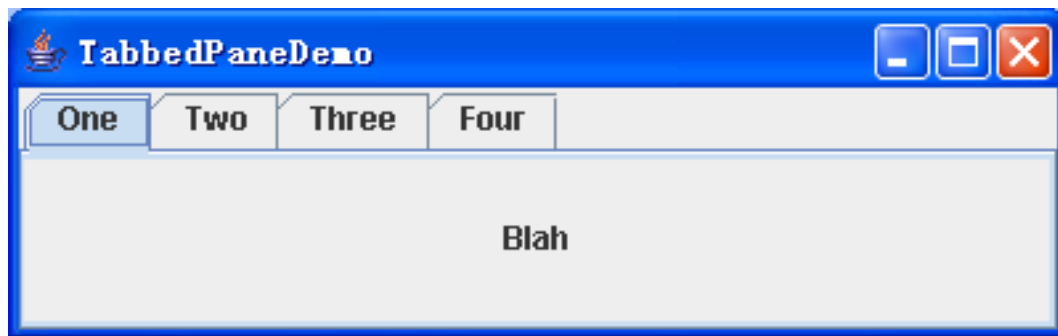
import java.util.*;

//SplitPaneDemo itself is not a visible component.
public class SplitPaneDemo implements ListSelectionListener {
    private Vector imageNames;
    private JLabel picture;
    private JList list;
    private JSplitPane splitPane;

    public SplitPaneDemo() {
        //Read image names from a properties file
```

JTabbedPane

- 如果一个窗口的功能有几项，可以给每项设置一个标签，每个标签下面包含为完成此功能专用的若干组件。
- TabbedPaneDemo.Java:



JTabbedPane 常用API

名称	说明
<code>JTabbedPane()</code>	创建一个tabbed pane。标签条位置在顶部
<code>JtabbedPane(int tabPlacement)</code>	创建一个tabbed pane，并设置其标签位置。参数为从SwingConstants接口中继承来的TOP、BOTTOM、LEFT、RIGHT之一。
<code>void addTab(String title, Icon icon, Component comp, String tip)</code> <code>void addTab(String title, Icon icon, Component comp)</code> <code>void addTab(String, Component)</code>	增加一个标签页，第一个String参数指定显示在标签上的文本，可选的Icon参数制定标签图标，Component参数指定选择此标签页时要显示的组件，最后一个可选的String参数是提示信息
<code>void insertTab(String title, Icon icon, Component comp, String tip, int index)</code>	在指定位置index插入一个标签页，第一个标签页的index是0，其余参数意义同addTab方法
<code>removeTabAt(int index)</code>	删除指定位置的标签页

JTabbedPane 常用API

<pre>int indexOfTab(Component comp) int indexOfTab(String title) int indexOfTab(Icon icon)</pre>	返回有指定组件、标题或图标的标签页序号
<pre>void setSelectedIndex(int) void setSelectedComponent(Component comp)</pre>	选择指定序号或组件的标签页。选择的效果是将显示此标签页所含的所有内容
<pre>void setComponentAt(int index, Component comp) Component getComponentAt(int index)</pre>	设置或读取指定序号标签页上的组件
<pre>void setEnabledAt(int index, Boolean enabled)</pre>	设置指定序号标签页是否可选

例：TabbedPaneDemo.Java

```
import javax.swing.JTabbedPane;
import javax.swing.ImageIcon;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JFrame;

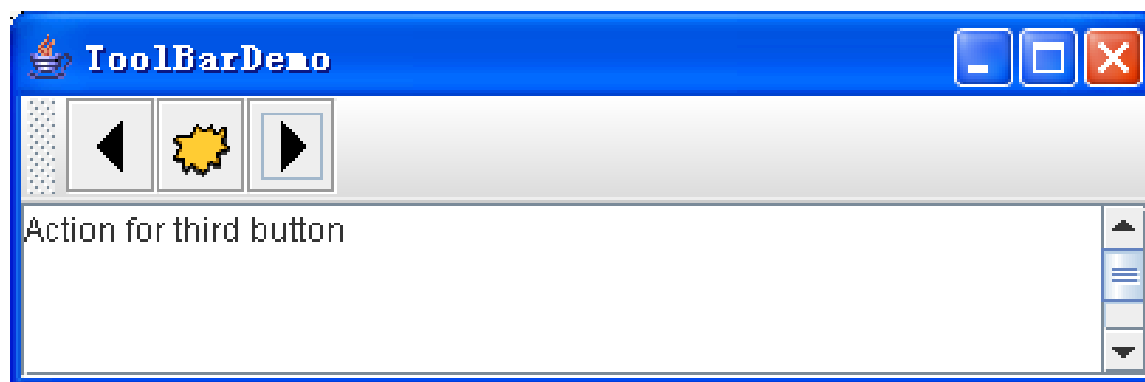
import java.awt.*;
import java.awt.event.*;

public class TabbedPaneDemo extends JPanel {
    public TabbedPaneDemo() {
        ImageIcon icon = new ImageIcon("images/middle.gif");
        JTabbedPane tabbedPane = new JTabbedPane();

        Component panel1 = makeTextPanel("Blah");
        tabbedPane.addTab("One", icon, panel1, "Does nothing");
    }
}
```

JToolBar

- 将一些常用的功能以工具栏的方式呈现。
- ToolBarDemo.java:



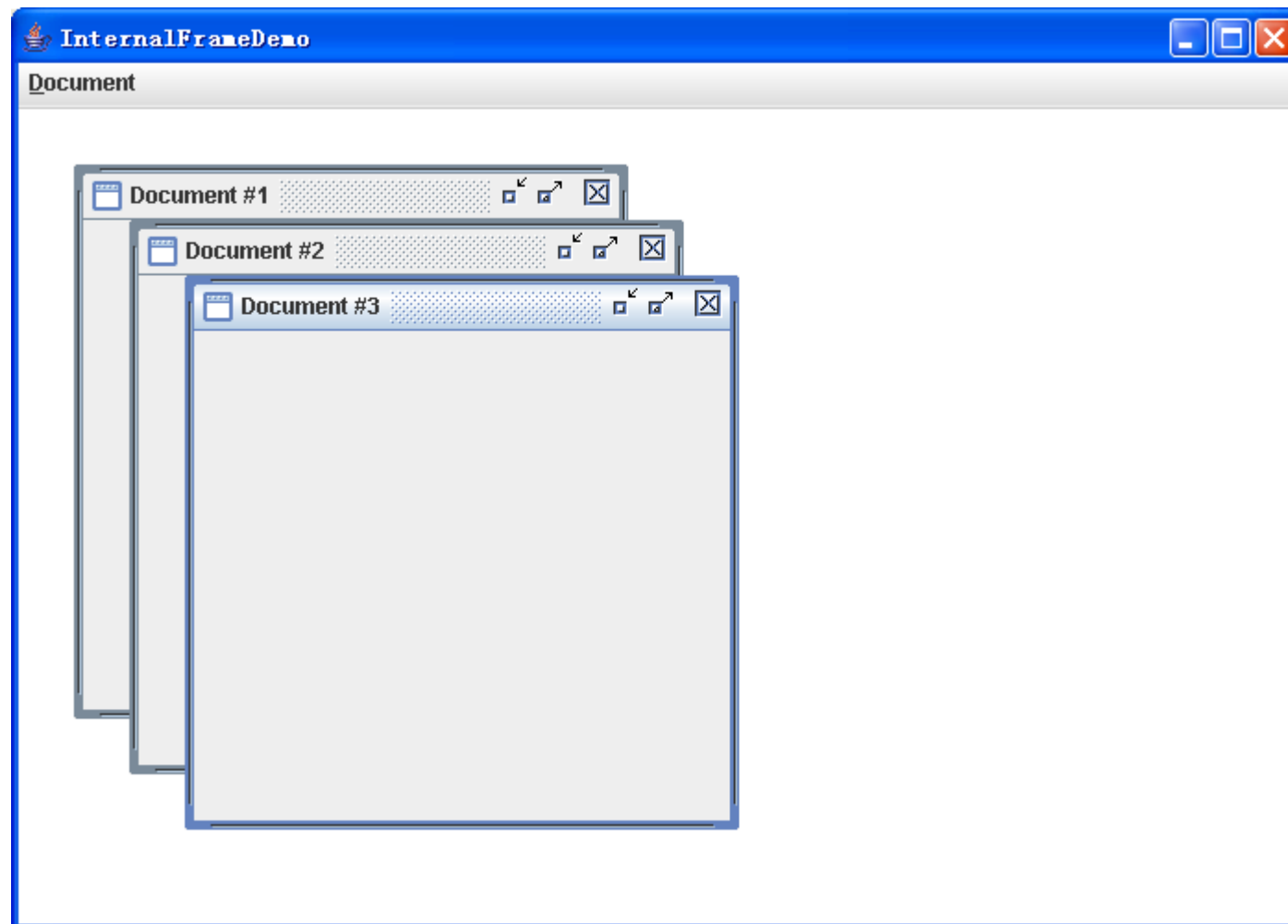
JToolBar 常用API

名称	说明
<code>JToolBar()</code> <code>JToolBar(int orientation)</code> <code>JToolBar(String name)</code> <code>JToolBar(String name, int orientation)</code>	创建一个工具栏，可以指定其朝向orientation，为SwingConstants中的HORIZONTAL或VERTICLE，也可以指定游离工具栏显示的名称name。
<code>void add(Component)</code>	为工具栏添加一个组件
<code>void addSeparator()</code>	在末尾增加一个分隔线
<code>void setFloatable(Boolean floatable)</code> <code>Boolean isFloatable()</code>	设置或读取工具栏是否可以游离，成为一个独立的窗口

例：ToolBarDemo.java

```
import javax.swing.JToolBar;  
import javax.swing.JButton;  
import javax.swing.ImageIcon;  
  
import javax.swing.JFrame;  
import javax.swing.JTextArea;  
import javax.swing.JScrollPane;  
import javax.swing.JPanel;  
  
import java.awt.*;  
import java.awt.event.*;  
  
public class ToolBarDemo extends JFrame {  
    protected JTextArea textArea;  
    protected String newline = "\n";  
}
```

InternalFrame



JInternalFrame

- JInternalFrame
 - 实现在一个主窗口中打开很多个文档，每个文档各自占用一个新的窗口。
 - JInternalFrame的使用跟JFrame几乎一样，可以最大化、最小化、关闭窗口、加入菜单。
 - JInternalFrame是轻量级组件，因此它只能是中间容器，必须依附于顶层容器上。
 - 通常将internal frame加入JDesktopPane类的对象来方便管理，JDesktopPane继承自JLayeredPane，用来建立虚拟桌面。它可以显示并管理众多internal frame之间的层次关系

JInternalFrame 常用API

名称	说明
<code>JInternalFrame()</code> <code>JInternalFrame(String title)</code> <code>JInternalFrame(String title, boolean resizable)</code> <code>JInternalFrame(String title, boolean resizable, boolean closable)</code> <code>JInternalFrame(String title, boolean resizable, boolean closable, boolean maximizable)</code> <code>JInternalFrame(String, boolean resizable, boolean closable, boolean maximizable, boolean iconifiable)</code>	创建一个子窗口对象，依次设置标题、是否可改变大小、可关闭、可最大最小化、是否有图标。默认状态下都是不可以。
<code>void setContentPane(Container c)</code> <code>Container getContentPane()</code>	设置或获得internal frame的内容面板，通常包括除菜单以外的所有可视组件

InternalFrame 常用API

<code>void setJMenuBar(JMenuBar m)</code> <code>JMenuBar getJMenuBar()</code>	设置或获得internal frame的菜单
<code>void setVisible(boolean b)</code>	此方法从Component类继承。设置是否可见。应该在将internal frame加到其父窗口之前调用。
<code>void setLocation(int x, int y)</code>	设置相对于父窗口的位置。指定左上角坐标
<code>void setBounds(int x, int y, int width, int height)</code>	设置位置和大小
<code>Void addInternalFrameListener(InternalFrameListener l)</code>	添加事件监听器，相当于windowlistener
<code>void setSelected(boolean b)</code> <code>boolean isSelected()</code>	设置或获得当前是否被激活。
<code>void setFrameIcon(Icon icon)</code> <code>Icon getFrameIcon()</code>	设置或获得显示在internal frame标题栏中的图标

例：InternalFrameDemo.java

```
import javax.swing.InternalFrame;
import javax.swing.JDesktopPane;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JMenuBar;
import javax.swing.JFrame;

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

public class InternalFrameDemo extends JFrame {
    JDesktopPane desktop;

    public InternalFrameDemo() {
        super("InternalFrameDemo");
    }
}
```

结束语

- 这一节我们了解了：
 - JSplitPane
 - JTabbedPane
 - JToolBar
 - JInternalFrame

原子组件 (一)

原子组件可分为三类

- 显示不可编辑信息
 - JLabel、JProgressBar、JToolTip
- 有控制功能、可以用来输入信息
 - JButton、JCheckBox、JRadioButton、JComboBox、JList、JMenu、JSlider、JSpinner、JTextComponent
- 能提供格式化的信息并允许用户选择
 - JColorChooser、JFileChooser、JTable、JTree

显示不可编辑信息的原子组件

- JLabel
 - 该组件上可以显示文字和图像，并能指定两者的位置。
- JProgressBar
 - 在一些软件运行时，会出现一个进度条告知目前进度如何。通过使用该组件我们可以轻松地给软件加上一个进度条。
- JToolTip
 - 使用setToolTipText()方法为组件设置提示信息。
 - 有的组件例如JTabbedPane由多个部分组成，需要鼠标在不同部分停留时显示不同的提示信息，这时可以在其addTab()方法中设置提示信息参数，也可以通过setToolTipTextAt方法进行设置。

例：进度条、标签、提示信息

```
import javax.swing.*;
import java.awt.*;
import javax.swing.event.*;
import java.awt.event.*;
public class Progress implements ChangeListener
{
    JLabel label;
    JProgressBar pb;
    public Progress()
    {
        int value=0;
        JFrame f=new JFrame("第一类原子组件演示");
        Container contentPane=f.getContentPane ();
        label=new JLabel("",JLabel.CENTER);
        label.setText("第一类进度信息");
    }
}
```



第一类原子组件演示



83.3%

目前已完成进度：83%

结束语

- 这一节我们学习了
 - 显示不可编辑信息的原子组件：
 - JLabel、JProgressBar、JToolTip

原子组件（二）

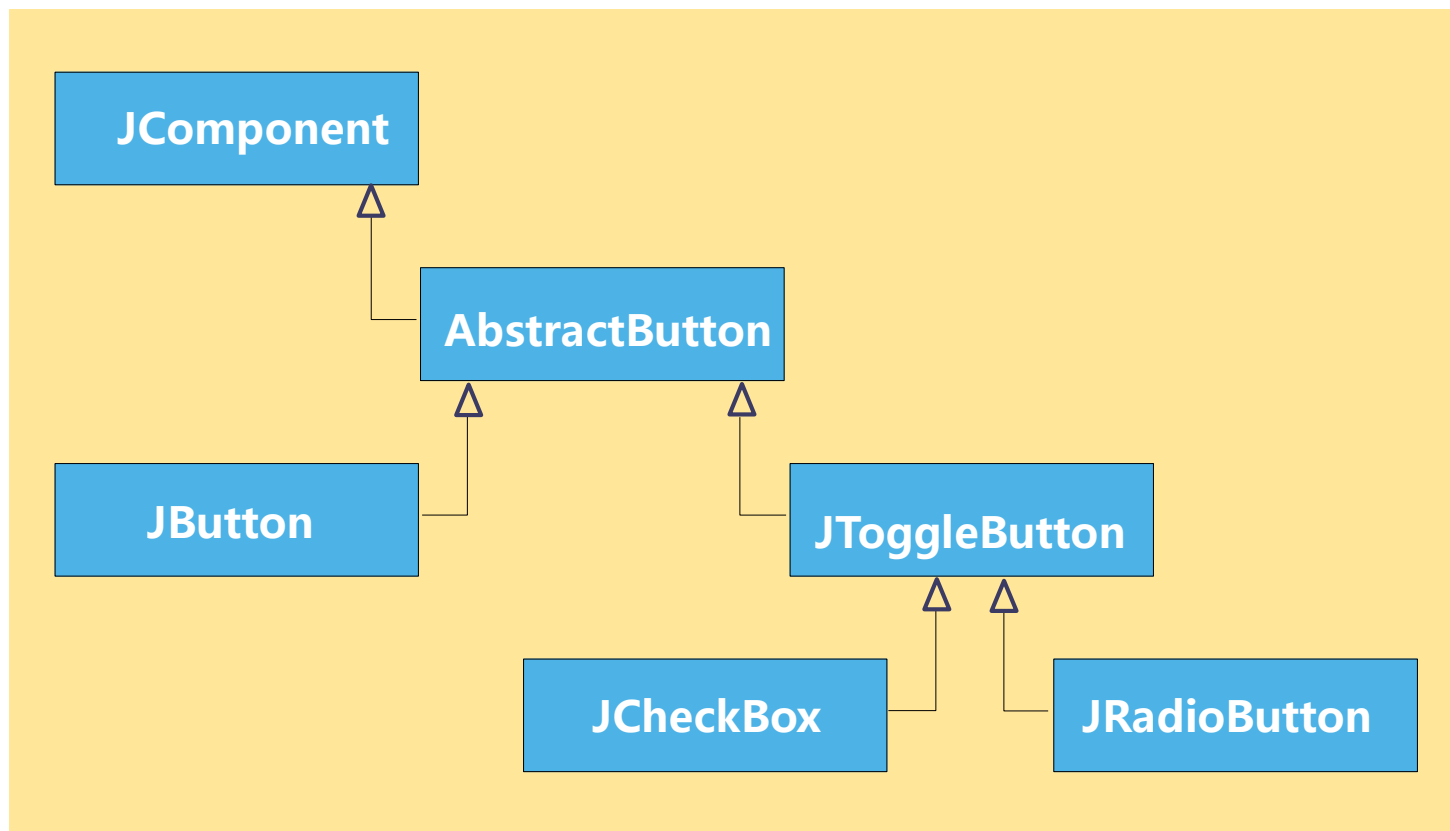
第二类原子组件：有控制功能、可以用来输入信息

按钮类

- `AbstractButton`抽象类是众多按钮类的超类

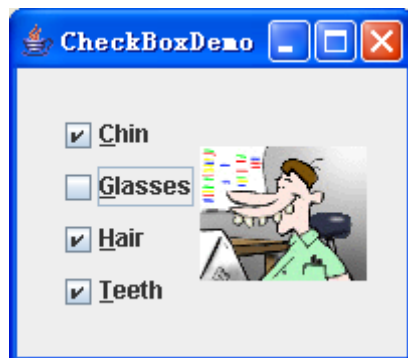
按钮类

AbstractButton抽象类是众多按钮类的超类，继承它的类包括：



- JButton
- JToggleButton——表示有两个选择状态的按钮
 - CheckBox（多选按钮）
 - JRadioButton（单选按钮）
- JMenuItem——在菜单中使用
 - JCheckBoxMenuItem（多选按钮）
 - JRadioButtonMenuItem（单选按钮）
 - JMenu（一般的菜单项）

按钮类示意图



列表框 JList

- JList可以选择一到多个选项

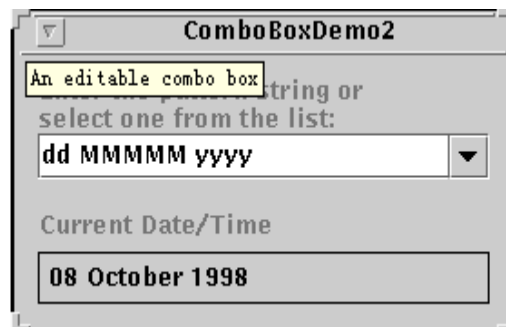
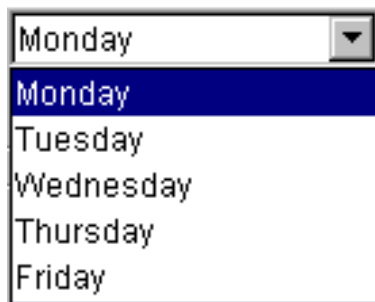


列表框 **JList**

- 有几种各有特色的构造方法（选项是否可添加、删除）。
- 也提供了很多**API**可以用来设置各选项的显示方式（在一列还是若干列）、选择模式（一次可选几项及是否可连续）等。
- 因为**JList**通常含有很多选项，所以经常把它放在一个**JScrollPane**对象里面。
- **JList**的事件处理一般可分为两类
 - 取得用户选取的项目，事件监听器是**ListSelectionListener**。
 - 对鼠标事件作出响应，其事件监听器是**MouseListener**。

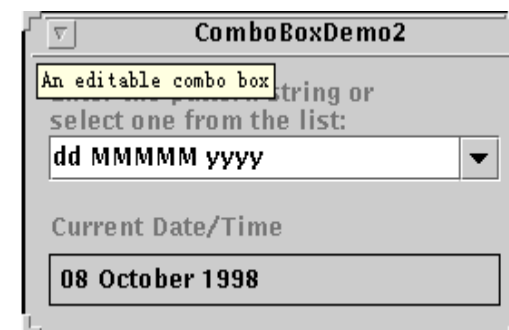
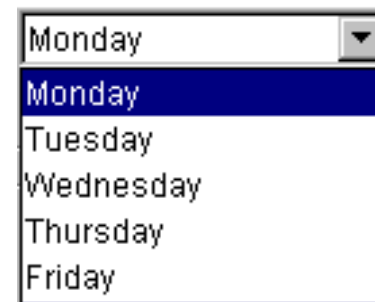
组合框 JComboBox

- 在许多选项中选其一



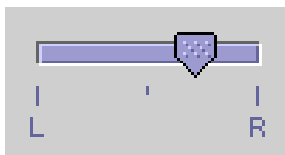
组合框 JComboBox

- 在许多选项中选其一，可以有两种非常不一样的格式：
 - 默认状态是不可编辑的模式，其特色是包括一个按钮和一个下拉列表，用户只能在下拉列表提供的内容中选其一。
 - 另一种是可编辑的模式，其特色是多了一个文本区域，用户可以在此文本区域内填入列表中不包括的内容。
- 组合框只需要很少的屏幕区域，而一组 JRadioButton、JCheckBox、JList 占据的空间比较大



连续数值选择

- JSlider



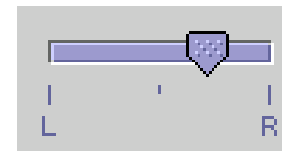
- JSpinner



连续数值选择

- JSlider

- 占空间大。
- 可以设置它的最小、最大、初始刻度，还可以设置它的方向，还可以为其标上刻度或文本。
- 在JSlider上移动滑动杆，会产生ChangeEvent事件。



- JSpinner

- 占空间小。
- 类似于可编辑的JComboBox，是种复合组件，由三个部分组成：向上按钮、向下按钮和一个文本编辑区。
- 可以通过按钮来选择待选项，也可以直接在文本编辑区内输入。
- 和JComboBox不同的是，它的待选项不会显示出来。

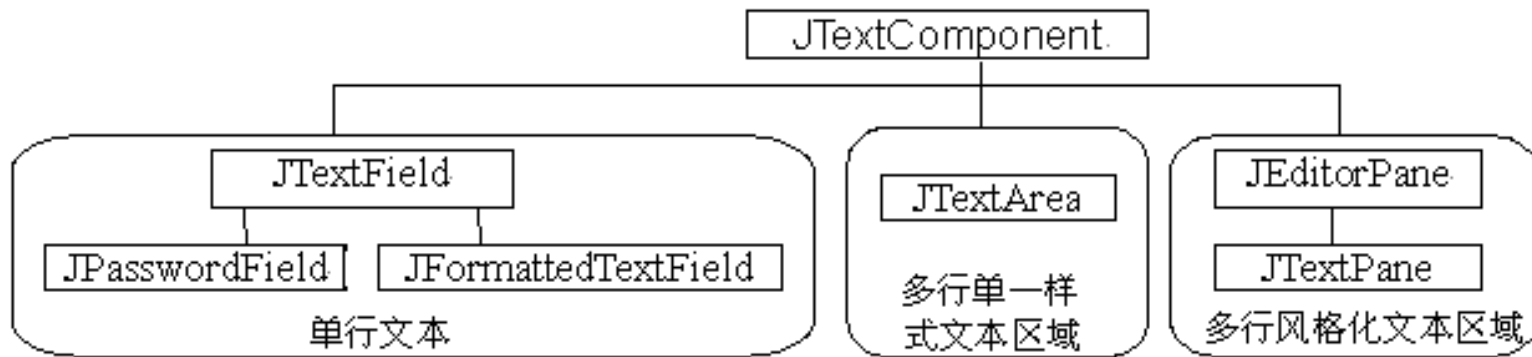


文本组件

- 允许用户在里边编辑文本，能满足复杂的文本需求

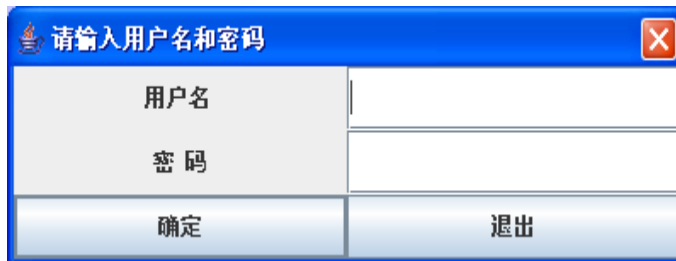
文本组件 <注：图要重画，清晰度不够>

- 都继承自JTextComponent抽象类，可分为三类：
 - JTextField、JPasswordField、JFormattedTextField
 - 只能显示和编辑一行文本。像按钮一样，它们可以产生ActionEvent事件，通常用来接受少量用户输入信息并在输入结束进行一些事件处理。
 - JTextArea
 - 可以显示和编辑多行文本，但是这些文本只能是单一风格的，通常用来让用户输入任意长度的无格式文本或显示无格式的帮助用户信息。
 - JEditorPane、JTextPane
 - 可以显示和编辑多行多种式样的文本，嵌入图像或别的组件。



例：简单计算器（原子组件举例）

- 在第一个窗口输入用户名和密码，要求输入密码使用JPasswordField组件，密码正确后，才能进入第二个窗口进行“+、-、×、÷”计算。运算符号使用单选按钮JRadioButton，操作数使用JcomboBox和Jspinner组件



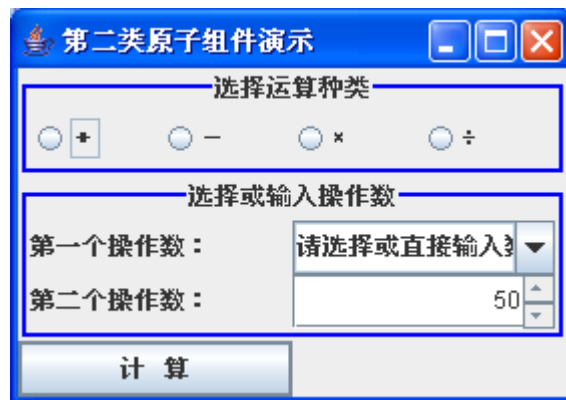
请输入用户名和密码

用户名	
密码	
确定	退出



请输入用户名和密码

用户名	test
密码	****
确定	退出



第二类原子组件演示

选择运算种类

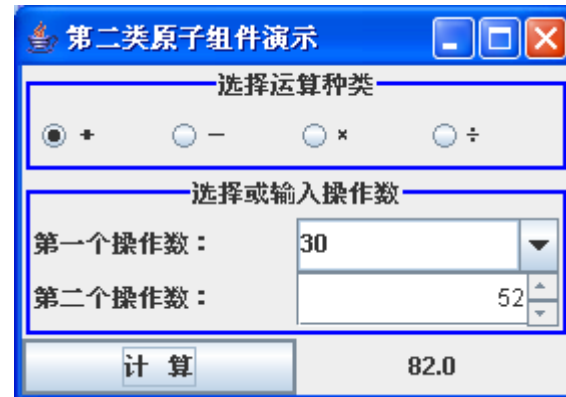
☒ + ☐ - ☐ × ☐ ÷

选择或输入操作数

第一个操作数：请选择或直接输入

第二个操作数：50

计 算



第二类原子组件演示

选择运算种类

☒ + ☐ - ☐ × ☐ ÷

选择或输入操作数

第一个操作数：30

第二个操作数：52

计 算 82.0

例：简单计算器（原子组件举例）

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
public class Calculator implements ActionListener ,ItemListener
{
    static JFrame f=null; //因为要在main静态方法中被引用，所以必须设为static类型
    ButtonGroup bg;      //按钮组，可组合若干单选按钮
    JComboBox combo;     //下拉式列表框
    JSpinner s1;         //有序变化选择框
    JLabel L3;           //显示计算结果的标签
    JRadioButton r1,r2,r3,r4; //单选按钮
    int op=0;

    public Calculator()
    {
```

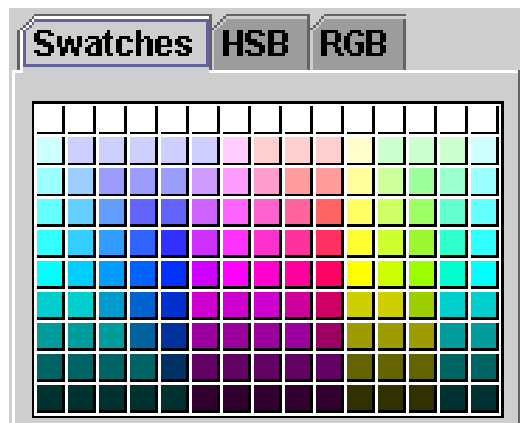
结束语

- 这一节我们学习了
 - 有控制功能、可以用来输入信息的原子组件

原子组件（三）

JColorChooser 颜色选择对话框

- 可以让用户选择所需要的颜色。

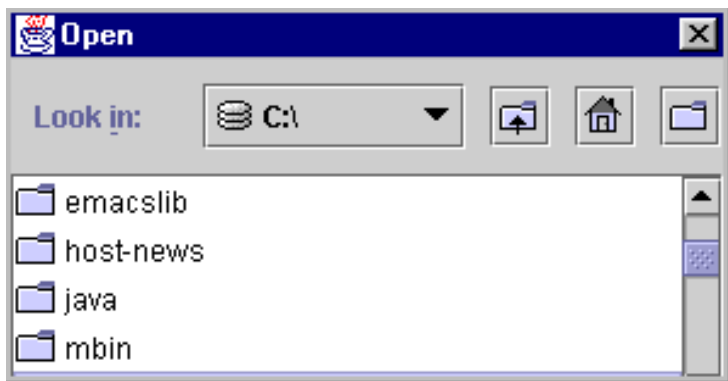


JColorChooser 颜色选择对话框

- 通常使用这个类的静态方法 `showDialog()` 来输出标准的颜色选择对话框，其返回值就是选择的颜色。
- 也可以通过静态方法 `createDialog()` 方式输出个性化的颜色选择对话框，例如为其添加菜单、定义其事件处理程序，这个方法的返回值就是一个对话框。

JFileChooser文件选择对话框

- 让用户选择一个已有的文件或者新建一个文件



JFileChooser文件选择对话框

- 可以使用JFileChooser的showDialog、showOpenDialog或showSaveDialog()方法来打开文件对话框，但是它仅仅会返回用户选择的按钮（确认还是取消）和文件名（如果确认的话），接下来的要实现的功能例如存盘或者打开的功能还需要程序员自己编写。
- 这个类提供了专门的方法用于设置可选择的文件类型，还可以指定每类文件使用的类型图标。

JTable 表 格

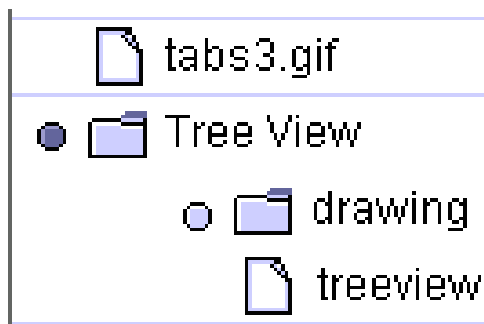
First Na...	Last Name
Mark	Andrews
Tom	Ball
Alan	Chung
Jeff	Dinkins

JTable 表 格

- 可为表格设计显示外观（是否有滚动条、调整某一列宽其它列宽变化情形）、显示模式（根据数据类型的不同有不同的排列显示方式、为某一字段添加组合框JComboBox）、选择模式（单选、多选、连续选、任意选）。
- JTable的事件都是针对表格内容的操作处理，我们称为TableModelEvent事件。通过addTableModelListener方法为表格添加此种事件监听器。
- 和其相关的还有一些接口和类，包括TableModel、AbstractTableModel、DefaultTableModel、SelectionModel、TableColumnModel。

JTree

- 用来产生树状结构来直观地表现层次关系，有根节点、树枝节点、树叶节点。



JTree

- 构造方法有多种，参数可以是一个Hashtable，也可以是TreeNode或TreeModel对象。
- 可以使用JComponent提供的putClientProperty方法来设置JTree外观，也可以使用TreeCellRenderer来个性化各类节点的显示样式。

结束语

- 这一节我们简单了解了：
 - 能提供格式化的信息并允许用户进行选择的原子组件

其它Swing特性

很多Swing组件都要使用一些相同的特性，包括

- Action对象

- 边框

- 观感

Action对象

<以下文字不显示>

- 在很多既有菜单又有工具栏的应用程序中，我们可以看到某条菜单和工具栏的功能是相同的，按照前面所讲的方法，我们需要为每个组件一一添加事件监听器，并在其处理程序中写入相同的代码，程序就会显得比较繁琐。在这种场合，可以考虑使用Action对象实现此功能

Action的用途

- Action接口是对ActionListener接口的一个有用的扩展，它的继承关系如下
`public interface Action extends ActionListener`
- 在很多既有菜单又有工具栏的应用程序中，可以通过Action对象封装事件响应代码和相关设置，并添加到不同的事件源。
- 可以通过它对不同组件的显示文字、图标、快捷键、提示文字、是否可用等属性进行统一的设置

创建 Action 对象

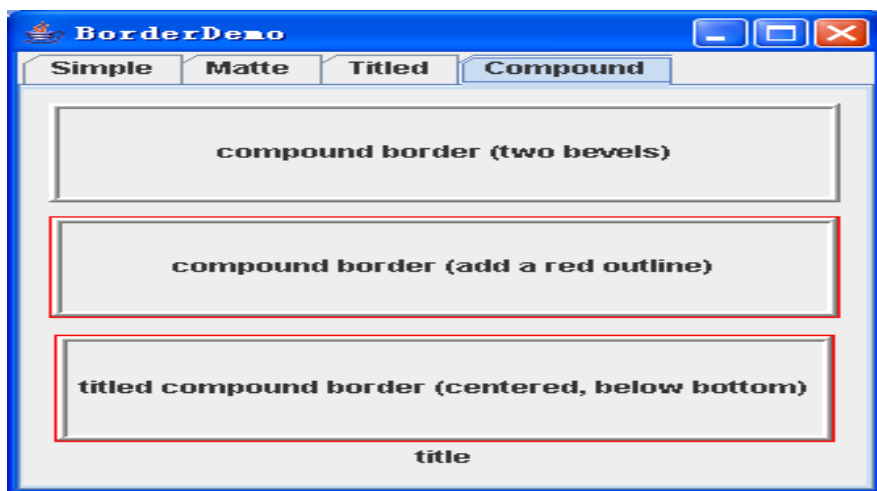
- AbstractAction类实现了Action接口中除了actionPerformed方法以外的其他方法。而且还提供了一些获取和设置Action域属性的方法。
- 首先需要创建一个继承抽象类AbstractAction类的子类，然后再实例化这个子类
 - 设置需要的属性值
 - 定义actionPerformed方法

使用Action对象

- 通过GUI组件的setAction方法将Action对象关联组件。
 - 每个具有addActionListener方法的组件也都具有setAction方法。
 - Action是一个事件监听器，如果需要添加多个监听器，应使用addActionListener方法。一个GUI组件可以调用setAction不止一次，但组件和前一个Action对象之间的关联会被删除。
- 通过setAction方法把Action对象关联到某GUI组件后，会有如下效果：
 - 此组件的属性会被设置为符合这个Action对象的属性。
 - 这个Action对象会被注册为此组件的一个事件监听器对象。
 - 如果改变了Action对象的属性或方法，则和它关联的组件的属性或方法也会自动变更以符合这个改变了的Action对象。

边框

- 每个继承自JComponent的Swing组件都可以有边框。



边框

- 使用组件的**setBorder**方法为组件添加边框， 需要提供一个**Border**类型的对象。
 - 可以使用**BorderFactory**类提供的很多静态方法产生一个常用的Border对象
 - 如果不能够满足要求，可以直接使用javax.swing.border包里的API来定义自己的边框。

有关边框的常用API

名称	说明
BorderFactory类里的静态方法，返回类型都是Border	
<code>createLineBorder(Color color)</code> <code>createLineBorder(Color color, int thickness)</code>	创建指定颜色、线宽的线框
<code>createEtchedBorder()</code> <code>createEtchedBorder(Color highlight, Color shadow)</code> <code>createEtchedBorder(int type)</code> <code>createEtchedBorder(int type, Color highlight, Color shadow)</code>	创建突起或凹陷的边框。可选的Color参数指定了使用的高亮和阴影色。type参数为EtchedBorder.LOWERED或EtchedBorder.RAISED之一，如果不含此参数，则为LOWERED。
<code>createLoweredBevelBorder()</code>	创建一种边框，给人感觉组件比周围低
<code>createRaiseBevelBorder()</code>	创建一种边框，给人感觉组件比周围高

有关边框的常用API

<pre>createBevelBorder(int type, Color highlight, Color shadow) createBevelBorder(int type, Color highlightOuter, Color highlightInner, Color shadowOuter, Color shadowInner)</pre>	创建为组件造成突起或下沉的效果的边框，type为BevelBorder.RAISED或BevelBorder.LOWERED。颜色参数用来指定外层和内层的高亮色和阴影色
<pre>createEmptyBorder() createEmptyBorder(int top, int left, int bottom, int right)</pre>	创建一不显示的边框，无参数的边框不占空间，有参数的指定了边框在四个边上占据的像素数
<pre>MatterBorder createMatteBorder(int top, int left, int bottom, int right, Color color) MatterBorder createMatteBorder(int top, int left, int bottom, int right, Icon tileIcon)</pre>	创建一个不光滑的边框，四个整数参数指定了边框在四个边上占据的像素数，颜色参数指定了边框的颜色，图标参数指定了填充边框的图标

有关边框的常用API

<pre>TitledBorder createTitledBorder(Border border, String title, int titleJustification, int titlePosition, Font titleFont, Color titleColor)</pre>	为已有的border增加标题title，并指定标题位置、字体、颜色。某些参数是可选的，具体用法见API文档。
<pre>CompoundBorder createCompoundBorder(Border outsideBorder, Border insideBorder)</pre>	创建一个双层边框
Swing组件用来设置边框的API	
<pre>void setBorder(Border border) Border getBorder()</pre>	设置或获得JComponent组件的边框
<pre>void setBorderPainted(Boolean) Boolean isBorderPainted()</pre>	设置或获得是否显示边框，用在AbstractButton、JMenuBar、JPopupMenu、JProgressBar和JToolBar中。

设置组件的观感

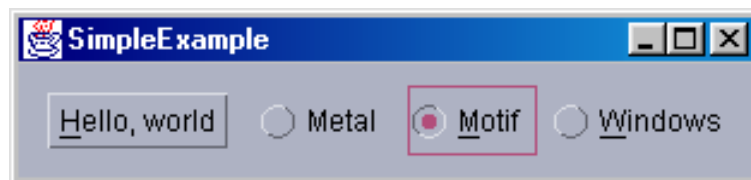
- 在产生任何可视组件以前需要使用UIManager类所提供的setLookAndFeel()静态方法设置好它们的观感。
 - java提供的跨平台的观感。可以利用UIManager类提供的getCrossPlatformLookAndFeelClassName()静态方法获得类名。
 - 程序所处系统的观感。可以利用UIManager类提供的getSystemLookAndFeel()静态方法获得目前操作平台的Look and Feel类名称字符串。

设置顶层容器的观感

- 顶层容器JFrame和JDialog是重量级组件，依赖于操作系统，当使用的操作系统不同时，所显示的顶层容器就会有所不同。针对这两个顶层容器，有一个静态方法专门为其设置观感
 - `static void setDefaultLookAndFeelDecorated(boolean)`
 - 参数是true，就会选用默认的外观感觉
 - 参数是false，就会选用操作平台的外观感觉

观感举例

- Java 观感
- CDE/Motif观感
- Windows观感



桌面API

- 从Java 6开始，对于特定的文件类型，Java程序可以和关联该文件类型的主机应用程序进行交互。这种交互是通过`java.awt.Desktop`类进行的，因此`java.awt.Desktop` API叫做桌面API。

桌面API允许Java应用程序完成以下三件事情：

- 启用主机平台上默认的浏览器打开URL，这个功能由DeskTop的**browse**方法完成；
- 启用主机平台上默认的邮件客户端，这个功能由DeskTop的**mail**方法完成；
- 对特定的文件，启用主机平台上与之关联的应用程序，对该文件进行打开、编辑、打印操作，这些功能分别由DeskTop的**open**、**edit**、**print**方法完成。

结束语

- 这一节我们简单了解了
 - Action对象
 - 边框
 - 观感
 - 桌面API

第7章 小结

本章内容

- Java的绘图机制，及实现更为出色绘图效果的Java 2D。

本章内容

- Swing的结构层次、容器、原子组件、布局管理，以及如何编写事件处理程序。

本章内容

- Action对象、边框、观感、桌面API。

结束语