内容提要

- Swing概述
- Swing容器
- 界面布局管理
- Java事件处理机制
- Swing基本组件
- Swing菜单和工具栏
- Swing高级组件

基于Swing的图形用户界面

- GUI (Graphics User Interface,图形用户界面)
  - 应用程序的外观
  - 用户与程序之间交互的图形化界面（接口）
- Java支持GUI设计的工具包
  - 早期—AWT(Abstract Window Toolkit, 抽象窗口工具)
  - JDK 1.2开始— Swing

Swing概述

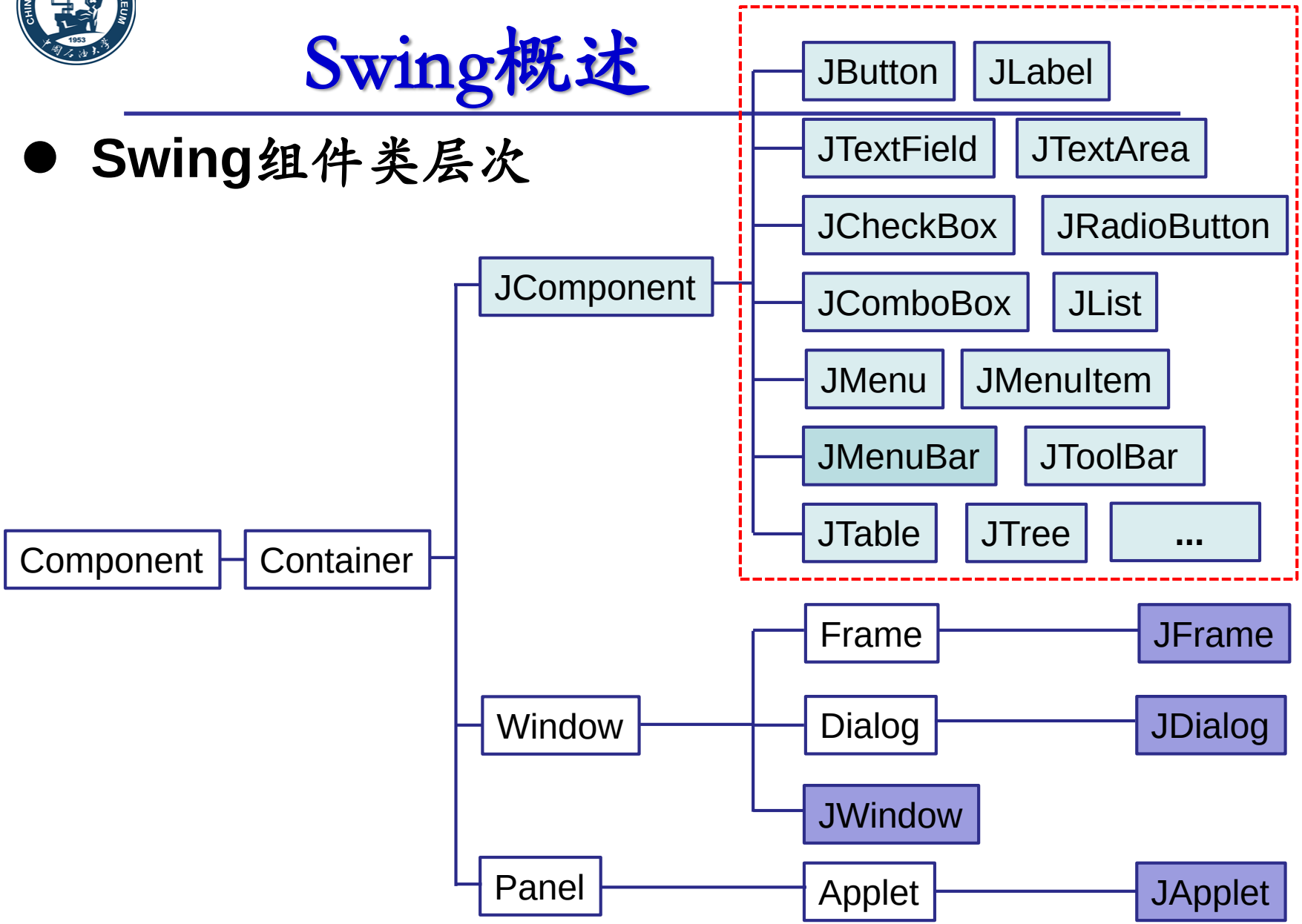
- 早期版本的AWT组件
  - 在 java.awt 包里，包括 Button、Checkbox、Scrollbar组件等，都是Component类的子类
  - 使用了本地平台的GUI功能，大部分含有本地代码(native code)，所以随操作系统平台的不同会显示出不同的外观，而不能进行更改，因此被称为是重量级组件(heavyweight components)

Swing概述

- Swing组件
  - 在javax.swing包，其名称都是在原来AWT组件名称前加上J，例如JButton、JTextField、JCheckBox、JScrollbar等，都是JComponent类的子类
  - JDK1.2推出，架构在 AWT 之上，是AWT的扩展而不是取代，AWT中事件处理机制、布局管理器等在Swing中仍被使用
  - 除了顶层容器，组件完全是由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(lightweight components)
  - 可提供更丰富的视觉感受，被越来越多地使用
  - 然而，Swing中的顶层容器JWindow、Jframe、Jdialog和JApplet仍然是重量级的，是仅有的重量级容器，外观和功能受到本地窗口系统的限制

Swing概述

- Swing组件类层次



Swing概述

- 基于Swing的GUI程序设计一般流程
  - 导入Swing包及其他包，一般包括：
    - javax.swing.*
    - java.awt.*
    - java.awt.event.*
    - 根据需要，其他swing包的子包，例如javax.swing.table.*
  - 在创建顶层容器之前，选择“外观和感觉”
  - 创建顶层容器并进行布局管理等设置，根据需要为顶层容器添加事件处理
  - 创建基本组件，进行相应设置，并根据需要为组件添加事件处理
  - 将组件添加到容器
  - 显示顶层容器，将整个GUI显示出来

Swing容器

- 通常将javax.swing包里的Swing组件归为三个层次
  - 顶层容器：表一个窗体、对话框或applet显示区域
    - JWindow、JFrame、JDialog和JApplet等
    - 有一个内容面板（contentPane）用于容纳中间中层容器和基本组件，调用容器的add方法实际上是向其内容面板添加组件，setLayout方法实际上是设置内容面板的布局。
    - 可以添加一个菜单栏，但在内容面板之外
  - 中间层容器：容纳其他组件，简化组件布局
    JPanel、JScrollPane、JSplitPane、JTabbedPane等
  - 原子组件(基本组件)：直接向用户展示信息或获取用户输入
    - JButton、JLabel、JTextField、JCheckBox等

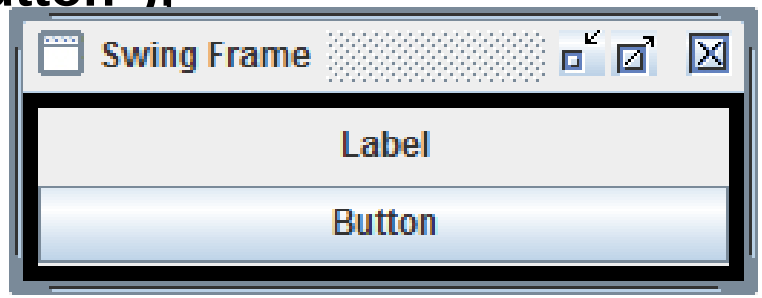
Swing容器

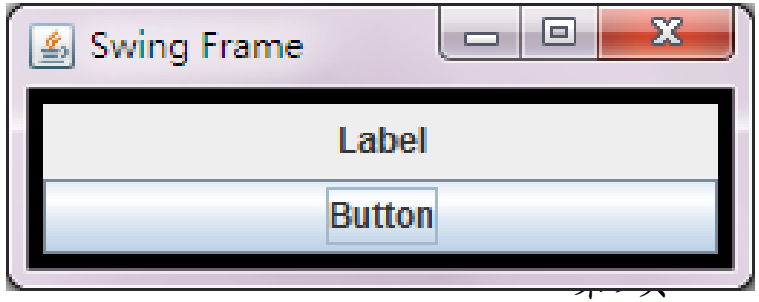
【例8-1】三层容器结构示例

```
import javax.swing.*;
import java.awt.*;
public class ThreeLevelContainerTester {
   public static void main(String[] args) {
      //设置默认外观
     JFrame.setDefaultLookAndFeelDecorated(true);
      //创建顶级容器--窗体
      JFrame frm=new JFrame("Swing Frame");
      frm.setBounds(200, 200, 260, 100); //设置窗体位置和大小
      frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Swing容器

```
    // 创建中间容器--面板
     JPanel panel=new JPanel();
   panel.setBorder(BorderFactory.createLineBorder(Color.black,5));
     panel.setLayout(new GridLayout(2,1));

     //创建原子组件--标签和按钮
     JLabel label=new JLabel("Label",SwingConstants.CENTER);
     JButton button=new JButton("Button");

     //添加组件到容器
     panel.add(label);
     panel.add(button);
     frm.add(panel);

     frm.setVisible(true); //显示窗体
   }
}
```





Swing容器

- 顶层容器之窗体类JFrame
  - 表示一个窗体，包含标题、边框以及最大化、最小化、关闭按钮
  - 默认（左上角）位置为(0, 0)，默认尺寸为0×0
  - 默认不可见
  - 创建窗体的两种方法
    - 直接用JFrame的构造方法创建并设置
    - 定义一个JFrame的派生类再创建，在派生类中对窗体详细设置---更常用
  - 常用构造方法
    - JFrame() 创建一个没有标题的窗体
    - JFrame(String title) 创建一个指定标题的窗体
  - 常用方法

Swing容器

Component **add**(Component comp)	将指定组件追加到此容器的尾部
void **add**(Component comp, Object constraints)	将指定组件追加到此容器的尾部，并指定显示方位(东/南/西/北/中)
void **setBounds**(int x, int y, int width, int height)	设置窗体左上角位置和大小
void **setSize**(int width, int height)	设置窗体大小
void **setLocation**(int x, int y)	设置窗体左上角位置
void **setTitle**(String title)	设置窗体标题
void **setResizable**(boolean resizable)	设置窗体是否可调整大小
void **setVisible**(boolean b)	设置窗体可见性即显示或隐藏窗体
void **setJMenuBar**(JMenuBar menubar)	设置菜单栏
static void **setDefaultLookAndFeelDecorated**(boolean defaultLookAndFeelDecorated)	设置窗体外观，是使用当前 **LookAndFeel** 为其提供的 **Window** 装饰，还是使用当前窗口管理器为其提供的 **Window** 装饰

Swing容器

void **setLayout**(LayoutManager mgr)	设置窗体布局管理器
void **setDefaultCloseOperation**(int operation)	设置用户在此窗体上单击关闭按钮时默认执行的操作
void **dispose**()	释放窗体资源

- operation参数说明
  - **WindowConstants.DO_NOTHING_ON_CLOSE**：不执行任何操作；要求程序在已注册的 **WindowListener** 对象的 **windowClosing** 方法中处理该操作
  - **WindowConstants.HIDE_ON_CLOSE**：调用任意已注册的 **WindowListener** 对象后自动隐藏该窗体
  - **WindowConstants.DISPOSE_ON_CLOSE**：自动隐藏并释放该窗体
  - **JFrame.EXIT_ON_CLOSE**：使用 **System.exit(0)** 方法退出应用程序
  - 注意：**JFrame**实现了接口**WindowConstants**，因此它也继承了前面三个常量

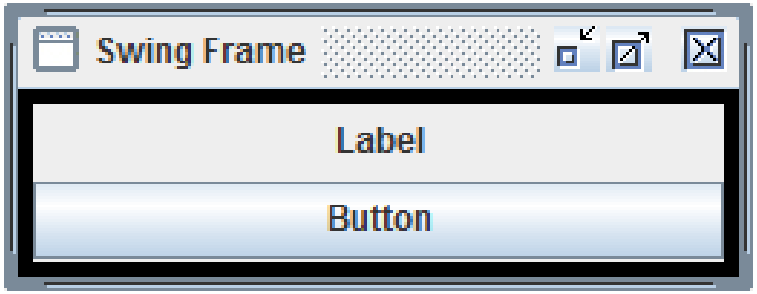
Swing容器

【例8-2】从JFrame派生一个窗体类，然后创建窗体

```
import javax.swing.*;
import java.awt.*;
public class MyJFrame extends JFrame {
  public MyJFrame(){
     this.setTitle("Swing Frame"); // super("Swing Frame");
     this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     this.setBounds(200, 200, 260, 100); //设置窗体位置和大小
      // 创建中间容器--面板
     JPanel panel=new JPanel();
     panel.setBorder(BorderFactory.createLineBorder(Color.black,5));
     panel.setLayout(new GridLayout(2,1));
```

Swing容器

```
    //创建原子组件--标签和按钮
    JLabel label=new  JLabel("Label",SwingConstants.CENTER);
    JButton button=new JButton("Button");
    //添加组件到容器
    panel.add(label);
    panel.add(button);
    this.add(panel);
  }
  public static void main(String[] args) {
    JFrame.setDefaultLookAndFeelDecorated(true); //设置默认外观--应该在窗体创建之前哦
    MyJFrame  frm=new  MyJFrame(); //创建窗体
    frm.setVisible(true); //显示窗体
  }
}
```



Swing容器

- 顶层容器之对话框窗体类JDialog
  - 与JFrame一样，表示一个窗体，但是没有最大化和最小化按钮
  - 主要用于显示提示信息或者接收用户输入（例如用户登录）
  - 对话框具有模式，分为模式的(Modal)与非模式的(Modelness)两种
  - 可以依附于某个窗体
  - 创建窗体的方法与JFrame相同
  - 常用方法与JFrame也几乎相同，但可以设置对话模式：void **setModal**(boolean modal)
  - 一般设置不允许调整大小

Swing容器

- 对话框窗体类JDialog
  - 常用构造方法

JDialog()	创建一个没有标题的无模式对话框。一个共享的、隐藏的窗体将被设置为该对话框的所有者
JDialog(Frame owner)	创建一个没有标题的无模式对话框。如果 **owner** 为 **null**，则一个共享的、隐藏的窗体将被设置为该对话框的所有者
JDialog(Frame owner, boolean modal)	创建一个无标题的模式对话框，所有者为 **owner**
JDialog(Frame owner, String title)	创建一个具有指定标题和所有者的无模式对话框
JDialog(Frame owner, String title, boolean modal)	创建一个具有指定标题和所有者的模式对话框

Swing容器

【例8-3】从JDialg派生一个对话框窗体类，然后创建它

```
import javax.swing.*;
import java.awt.*;
public class MyJDialog extends JDialog {
  public MyJDialog (){
     this.setTitle("Swing 对话框");
     this.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
     this.setBounds(200, 200, 210, 100);
     this.setResizable(false);// 禁止许用户调整大小
    JButton btnOK=new JButton("确定");
    this.add(btnOK);
   }
  public static void main(String[] args) {
     MyJDialog dlg=new MyJDialog ();
     dlg.setVisible(true);
   }
}
```

Swing容器

- 中间层容器之面板类JPanel
  - 最常用的中间层容器，使GUI管理更容易
  - 可以容纳原子组件和中间层面板，作为一个整体加入JFrame等顶层容器；容器不可见，则其内的组件也不可见
  - 常用构造方法
    - JPanel() 创建一个默认布局(即流式布局)的 JPanel
    - JPanel(LayoutManager layout) 创建一个指定布局管理器的JPanel
  - 常用方法
    - void **setLayout**(LayoutManager layout) 设置面板的布局管理器
    - Component **add**(Component comp) 加入组件
    - void **setVisible**(boolean aFlag) 设置可见性

Swing容器

【例8-4】使用JPanel在窗体上布局4个组件


```
import javax.swing.*;        import java.awt.*;
public class JPanelTester extends JFrame {
    public JPanelTester(){
        super("JPanel示例");
        this.setSize(300, 150);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel panel=new JPanel();
        panel.setBackground(Color.GREEN);
        JLabel lblName=new JLabel("请输入姓名： ");
        JTextField txtName=new JTextField("张三",5);
        JButton btnOK=new JButton("确定");
        JButton btnCancel=new JButton("取消");
        panel.add(lblName);          panel.add(txtName);
        panel.add(btnOK);          panel.add(btnCancel);
        this.add(panel);           frm.setVisible(true);
    }
    public static void main(String[] args) {
            JPanelTester frm=new JPanelTester();  }
}
```

Swing容器

- 中间层容器之滚动面板类JScrollPane
  - 用于滚动显示面板中的组件，例如滚动显示文本区域
  - 只允许放入一个组件
  - 有多个组件时，可以先将它们放入一个JPanel内，然后再将该JPanel放入JScrollPane内即可
  - 带有垂直滚动条和水平滚动条
  - 常用构造方法
    - JScrollPane() 创建一个空的（无视口的视图）JScrollPane，需要时水平和垂直滚动条都可显示
    - JScrollPane(Component view) 创建一个显示指定组件内容的 JScrollPane，只要组件的内容超过视图大小就会显示水平和垂直滚动条

Swing容器

- JScrollPane(Component view, int vsbPolicy, int hsbPolicy) 创建一个显示指定组件内容的**JScrollPane**，同时设定滚动条的显示策略；两个策略参数取值为接口 **ScrollPaneConstants**中定义的如下常量：

HORIZONTAL_SCROLLBAR	显示水平滚动条
HORIZONTAL_SCROLLBAR_AS_NEEDED	在需要时显示水平滚动条
HORIZONTAL_SCROLLBAR_NEVER	从不显示水平滚动条
VERTICAL_SCROLLBAR	显示垂直滚动条
VERTICAL_SCROLLBAR_AS_NEEDED	在需要时显示垂直滚动条
VERTICAL_SCROLLBAR_NEVER	从不显示垂直滚动条

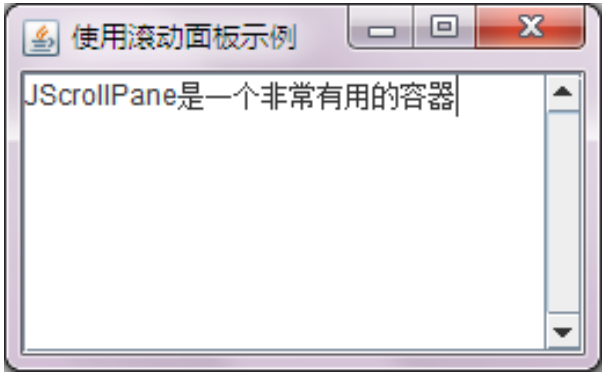
Swing容器

- 中间层容器之滚动面板类**JScrollPane**
  - 常用方法
    - void **setViewportView**(Component view) 指定在滚动面板视图中显示的组件
    - void **setHorizontalScrollBarPolicy**(int policy) 设置水平滚动条显示策略
    - void **setVerticalScrollBarPolicy**(int policy) 设置垂直滚动条显示策略

Swing容器

【例8-5】使用JScrollPane滚动显示JTextArea组件的内容

```
import javax.swing.*;
public class JScrollPaneTester {
   public static void main(String[] args) {
      JFrame frm=new JFrame("使用滚动面板示例");
       frm.setSize(300, 200);
       JScrollPane sp=new JScrollPane();
       sp.setHorizontalScrollBarPolicy(
           ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
       sp.setVerticalScrollBarPolicy(
           ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);

       JTextArea ta=new JTextArea("JScrollPane是一个非常有用的容器");
       sp.setViewportView(ta);

       frm.add(sp);
      frm.setVisible(true);
   }
}
```



Swing容器

- 中间层容器之拆分面板类**JSplitPane**
  - 将面板拆分为两部分，程序中可以设定按水平方向或垂直方向拆分，运行时用户可以调整分隔比例
  - 通过嵌套使用**JSplitPane**，可以将空间分割成更多部分
  - 常用构造方法
    - **JSplitPane**(int newOrientation) 创建一个指定拆分方向的拆分面板，参数取值如下：
      - JSplitPane.HORIZONTAL_SPLIT 水平拆分（左右分割）
      - JSplitPane.VERTICAL_SPLIT 垂直拆分（上下分割）
    - **JSplitPane**(int newOrientation, boolean newContinuousLayout) 创建一个指定拆分方向的拆分面板，并指定当分隔条改变位置时组件是否连续重绘

Swing容器

- 常用方法

void **setOrientation**(int orientation)	设置分割方向
void **setDividerLocation**(int location)	设置分隔条的位置
void **setDividerSize**(int newSize)	设置分隔条的大小
void **setOneTouchExpandable**(boolean newValue)	设置是否显示分割条上的折叠和快速展开箭头
void **setLeftComponent**(Component comp)	将组件设置到分隔条的左边
void **setTopComponent**(Component comp)	将组件设置到分隔条的上面
void **setRightComponent**(Component comp)	将组件设置到分隔条的右边
void **setBottomComponent**(Component comp)	将组件设置到分隔条的下面

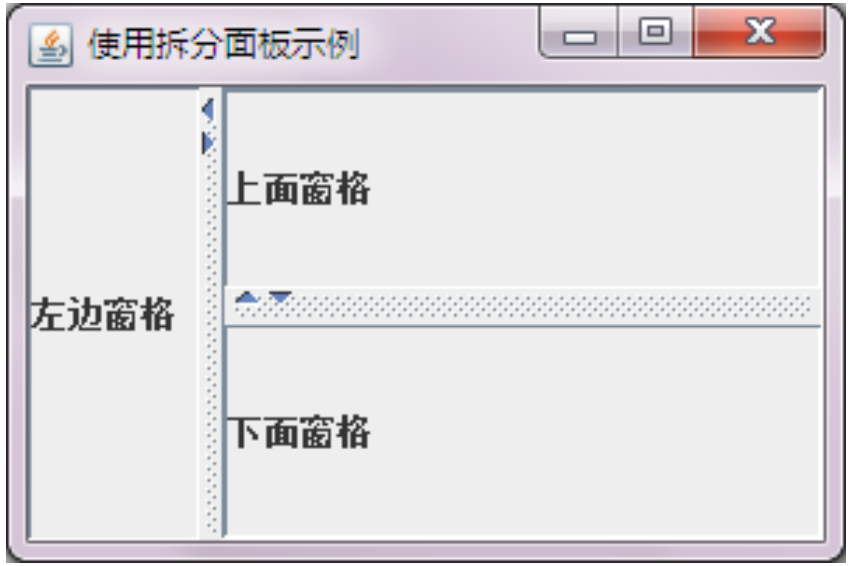
Swing容器

【例8-6】使用拆分面板JSplitPane分割父容器空间

```
import javax.swing.*;
public class JScrollPaneTester {
   public class JSplitPaneTester {
   public static void main(String[] args) {
         JFrame frm=new JFrame("使用拆分面板示例");
         frm.setSize(300, 200);
         frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

         JSplitPane outerSP=new
             JSplitPane(JSplitPane.HORIZONTAL_SPLIT,true);
         outerSP.setOneTouchExpandable(true);
         outerSP.setDividerLocation(60); //分隔条初始位置
         outerSP.setDividerSize(10); //分隔条大小
         outerSP.setLeftComponent(new JLabel("左边窗格"));
```

Swing容器

```
        JSplitPane innerSP=new
                JSplitPane(JSplitPane.VERTICAL_SPLIT,true);
        innerSP.setOneTouchExpandable(true);
        innerSP.setDividerLocation(70); //分隔条初始位置
        innerSP.setDividerSize(15); //分隔条大小
        innerSP.setTopComponent(new JLabel("上面窗格"));
        innerSP.setBottomComponent(new JLabel("下面窗格"));

        outerSP.setRightComponent(innerSP); //外部拆分面板
    的右边窗格

        frm.add(outerSP);
        frm.setVisible(true);
    }
}
```



Swing容器

- 中间层容器之选项卡面板类**JTabbedPane**
  - 允许用户通过单击具有给定标题和/或图标的选项卡，在一组组件之间进行切换。
  - 组件按功能的不同，组织在各个选项卡中。
  - 通过使用 **addTab** 和 **insertTab** 方法将选项卡/组件添加到 **TabbedPane** 对象中。选项卡通过对应于添加位置的索引来表示，其中第一个选项卡的索引为 0。
  - 常用构造方法
    - **JTabbedPane**() 创建一个空的选项卡面板，选项卡布局默认JTabbedPane.TOP，即选项卡位于面板顶端
    - **JTabbedPane**(int tabPlacement) 创建一个空的选项卡面板，使其具有以下指定选项卡布局中的一种：

      JTabbedPane.TOP/BOTTOM/LEFT/RIGHT

Swing容器

- 常用方法
  - void **addTab**(String title, Icon icon, Component component, String tip)：添加由 title 和/或 icon 表示的 component 和 tip
  - void **insertTab**(String title, Icon icon, Component component, String tip, int index)：在 index 位置插入一个 component
  - Component **add**(String title, Component component)：添加由 title 表示的 component
  - void **removeTabAt**(int index)/**remove**(int index)/**removeAll**()：移除指定/所有选项卡
  - int **getTabCount**()：获取选项卡数
  - int **getSelectedIndex**()：返回当前选择的此选项卡窗格的索引
  - String **getTitleAt**(int index)：获取指定选项卡标题

Swing容器

【例8-7】使用选项卡面板JTabbedPane示例


```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;
public class JTabbedPaneTester
        extends JFrame
        implements ChangeListener{
    private JLabel lblInfo;
    private JPanel[] panels;
    public JTabbedPaneTester(){
            super("选项卡面板JTabbedPane示例");
            this.setSize(300, 200);
            this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

            JTabbedPane tp=new JTabbedPane(); //选项卡面板
            panels=new JPanel[3]; // 每个选项卡窗格中放置一个Panel
```

```
public JTabbedPaneTester(){
        …
        String[] tipTexts={"First Panel","Second Panel","Third Panel"};
        for(int i=0;i<panels.length;i++){
            panels[i]=new JPanel();
            panels[i].add(new JLabel("panel"+ (i+1)));
            tp.addTab("Tab"+(i+1), null, panels[i], tipTexts[i]); //添加选项卡
        }
        tp.addChangeListener(this); // 注册ChangeEvent监听器
        lblInfo=new JLabel("状态信息");
        lblInfo.setForeground(Color.RED);
        this.add(tp,"Center");  this.add(lblInfo,"South");
    }
    public void stateChanged(ChangeEvent e){ // 实现ChangeListener接口
        JTabbedPane tp=(JTabbedPane) e.getSource();
        int index = tp.getSelectedIndex();
        lblInfo.setText(tp.getTitleAt(index)+" Selected!");
    }
    public static void main(String[] args) {
        JTabbedPaneTester frm=new JTabbedPaneTester();
        frm.setVisible(true);
    }
}
```

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：

    frame.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

布局管理

- 布局管理器对容器中的组件进行布局，使GUI更专业美观；布局管理器类均实现LayoutManager接口
- 常用
  - FlowLayout：组件自左向右、自上而下放置在容器中
  - BorderLayout：东南西北中各放置一个组件，各方位的组件与容器在该方位的边界的距离始终保持不变
  - GridLayout：容器被均匀划分成一个n行m列的网格，每个网格内放置一个组件，组件自左向右、自上而下放置在网格中
  - GridBagLayout：与GridLayout类似，但网格可以是非均匀的
  - CardLayout：每个组件看做一个卡片，容器看做一个卡包，一个时刻只能显示某个卡片
  - BoxLayout：组件垂直排列成一列或水平排列成一行
- 调用容器的void **setLayout**(LayoutManager layout)方法设置其布局管理器

布局管理：FlowLayout

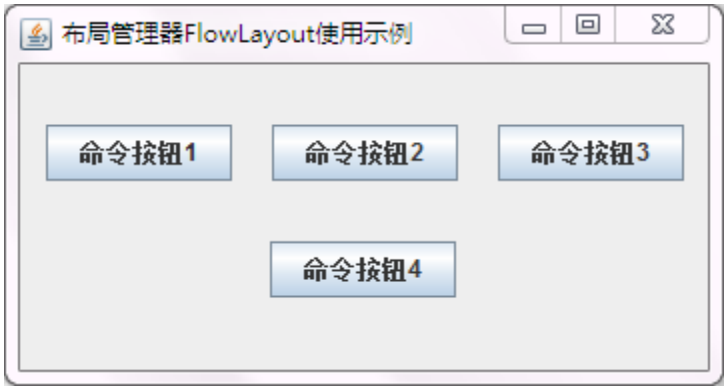
- 组件自左向右、自上而下放置在容器中，且可以设置组件的纵横间距和水平对齐方式
- 为JPanel的默认布局管理器
- 构造方法
  - FlowLayout()：默认组件居中对齐，纵横间距为5个单位
  - FlowLayout(int align)：指定对齐方式，align取值为FlowLayout.LEFT/CENTER/RIGHT/LEADING/TRAILING
  - FlowLayout(int align, int hgap, int vgap)：指定对齐方式和纵横间距
- 常用方法
  - void **setAlignment**(int align) 设置组件的水平对齐方式
  - void **setHgap**(int hgap) 设置组件之间以及组件与容器边界的水平间距
  - void **setVgap**(int vgap) 设置组件之间以及组件与容器边界的垂直间距

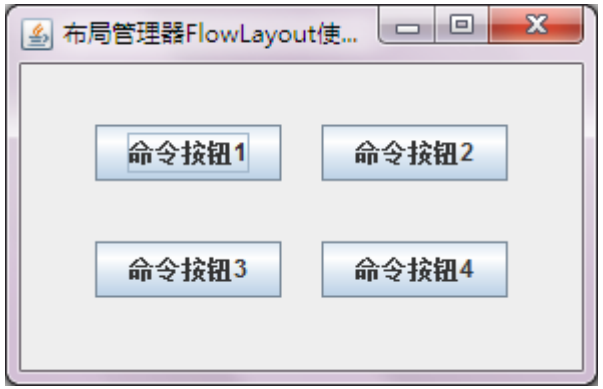
布局管理：FlowLayout

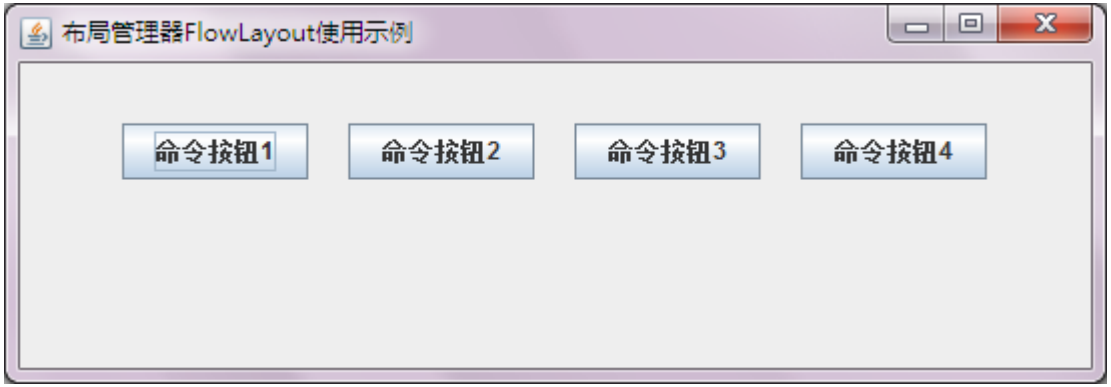
【例8-8】使用FlowLayout布局JFrame中的若干按钮

```
import java.awt.*;
import javax.swing.*;
public class FlowLayoutTester {
   public static void main(String[] args) {
      JFrame frm=new JFrame("布局管理器FlowLayout使用示例");
       frm.setSize(400, 200);
       frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

       FlowLayout layout=new FlowLayout(FlowLayout.CENTER);
       layout.setHgap(20);          layout.setVgap(30);
       frm.setLayout(layout); //给窗体容器设置布局管理器
       for(int i=1;i<=4;i++)   frm.add(new JButton("命令按钮"+i));
       frm.setVisible(true);
   }
}
```

布局管理：FlowLayout







- 第一行放置不下的将在第二行放置，以此类推
- 调整窗体大小，组件的水平对齐方式和纵横间距都不变

布局管理：BorderLayout

- 容器空间被划分成东南西北中5个区域，每个区域放置一个组件
- 某个方位无组件时，其他方位上的组件将自动缩放后占有其位置
- 为JFrame、JDialog、JApplet的默认布局管理器
- 构造方法
  - **BorderLayout()**：组件之间没有间距
  - **BorderLayout(int hgap, int vgap)**：指定纵横间距
- 常用方法
  - void **setHgap**(int hgap) 设置组件之间以及组件与容器边界的水平间距
  - void **setVgap**(int vgap) 设置垂直间距
- 该布局下，组件加入容器的方法
  - void **add**(Component comp, Object constraints)

    BorderLayout.EAST/SOUTH/WEST/NORTH/CENTER

    这些静态常量是字符串类型的，他们对应的值分别是

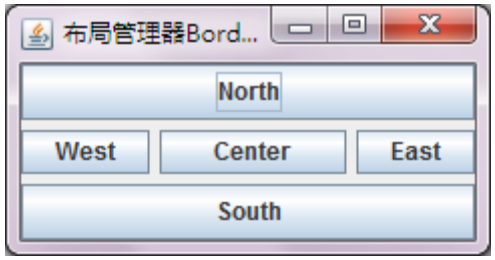
    "East"/"South"/"West"/"North"/"Center"

布局管理：BorderLayout

【例8-9】使用BorderLayout布局JFrame中的若干按钮

```
import java.awt.*;
import javax.swing.*;
public class BorderLayoutTester {
   public static void main(String[] args) {
       JFrame frm=new JFrame("布局管理器BorderLayout使用示例");
       frm.setSize(340, 200);
       frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
       BorderLayout layout=new BorderLayout(5,5); //组件水平和
   垂直间距均为5
       frm.setLayout(layout); //给窗体容器设置布局管理器
       frm.add(new JButton("East"),"East");
       frm.add(new JButton("West"),"West");
       frm.add(new JButton("South"),"South");
       frm.add(new JButton("North"),"North");
       frm.add(new JButton("Center"),"Center");
       frm.setVisible(true);
   }
}
```

布局管理器：BorderLayout



- 五个组件占满整个容器空间
- 调整窗体大小，组件的大小跟随自动调整
  - 南北方位的组件向水平方向缩放
  - 东西方向的组件向垂直方向缩放
  - 中间方位的组件向四周缩放

布局管理：GridLayout

- 容器空间被均匀划分成n行m列的网格，每个网格中放置一个组件，组件按从左到右、由上至下顺序依次放置
- 常用构造方法
  - **GridLayout**(int rows, int cols)：指定网格行数和列数
  - **GridLayout**(int rows, int cols, int hgap, int vgap)：指定网格行数和列数，以及组件的水平和垂直间距
  - 注意：**GridLayout(row,column)**中，列数通过指定的行数和布局中的组件总数来确定。 因此，例如，如果指定了3行和2列，在布局中添加了9个组件，则它们将显示为3行3列。仅当将行数设置为零时，指定列数才对布局有效
- 常用方法
  - void **setRows**(int rows) 设置网格行数
  - void **setColumns**(int cols)
  - void **setHgap**(int hgap) 设置组件之间以及组件与容器边界的水平间距
  - void **setVgap**(int vgap)

布局管理：GridLayout

【例8-10】使用GridLayout布局JFrame中的若干按钮

```
import java.awt.*;
import javax.swing.*;
public class GridLayoutTester {
   public static void main(String[] args) {
      JFrame frm=new JFrame("布局管理器GridLayout使用示例");
       frm.setSize(320, 150);
       frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       GridLayout layout=new GridLayout(2,3,5,5); //2行3列

           frm.setLayout(layout);
           for(int i=1;i<=5;i++)
                    frm.add(new JButton("命令按钮"+i));

           frm.setVisible(true);
   }
}
```

布局管理：GridLayout

- 网格占满整个容器空间
- 调整窗体大小，组件的大小跟随自动调整

布局管理：GridBagLayout

- 与网格布局相似，但每个组件可以占用一个或多个网格单元(称为组件的显示区域)，组件加入容器的顺序可以任意
- 每个由 GridBagLayout 布局的组件都与一个 GridBagConstraints 对象相关联，该对象指定了如何将组件放置到网格内，被称为约束对象
- GridBagConstraints 类封装了一组共有类型的约束变量，以对布局行为进行约束
  - int gridx/gridy: 指定放置组件的左上方网格单元的列/行号
  - int gridwidth/gridheight: 指定组件占有网格单元的列/行数
  - int fill: 当组件的显示区域大于组件尺寸时，如何改变组件的大小。取常量：GridBagConstraints.NONE(默认) /HORIZONTAL /VERTICAL /BOTH
  - int anchor: 当组件的显示区域大于组件尺寸时，确定将组件置于显示区域的何处。取常量：GridBagConstraints.CENTER(默认) / EAST/ SOUTH / WEST/ NORTH/ NORTHWEST/ NORTHEAST/ SOUTHWEST/ SOUTHEAST

布局管理：GridBagLayout

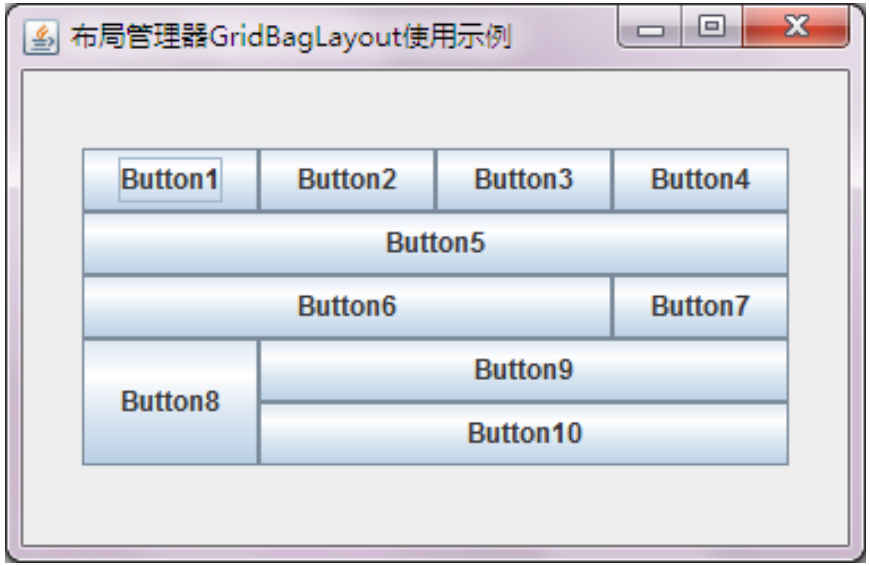
【例8-11】使用GridBagLayout布局JFrame中的若干按钮

```
import java.awt.*;
import javax.swing.*;
public class GridBagLayoutTester  extends JFrame{
   public GridBagLayoutTester(){
      super("布局管理器GridBagLayout使用示例");
      this.setSize(340, 200);
      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      GridBagLayout layout=new GridBagLayout(); //网格包布局
      this.setLayout(layout); //给窗体容器设置布局管理器

      JButton[] btns=new JButton[10];
      for(int i=0;i<btns.length;i++)
            btns[i]=new JButton("Button"+(i+1));

      GridBagConstraints st=new GridBagConstraints();//约束对象
      st.fill=GridBagConstraints.BOTH; //组件充满显示区域
```

布局管理：GridBagLayout

```
        for(int i=0;i<4;i++) add(btns[i], set(st, i, 0, 1, 1));
        add(btns[4], set(st, 0, 1, 4, 1)); add(btns[5], set(st, 0, 2, 3, 1));
        add(btns[6], set(st, 3, 2, 1, 1)); add(btns[7], set(st, 0, 3, 1, 2));
        add(btns[8], set(st, 1, 3, 3, 1)); add(btns[9], set(st, 1, 4, 3, 1));
        pack();
    }
    // 修改约束对象的约束条件
    public GridBagConstraints set(GridBagConstraints st,
                                  int x,int y,int w,int h){
            st.gridx=x;          st.gridy=y;
            st.gridwidth=w;   st.gridheight=h;
            return st;
    }
    public static void main(String[] args) {
            GridBagLayoutTester frm=new GridBagLayoutTester();
            frm.setVisible(true);
    }
}
```

布局管理：GridBagLayout



布局管理：CardLayout

- 容器中的每个组件看做一张卡片，容器看做卡包，一次只能看到一个卡片
- 是选项卡面板**JTabbedPane**的默认布局管理器
- 常用构造方法
  - **CardLayout()**：组件与容器边缘间距的为**0**
  - **CardLayout**(int hgap, int vgap)：指定组件与容器边缘间距，水平间距置于左右边缘，垂直间距置于上下边缘
- 常用方法
  - void **first**(Container parent) 翻转到容器的第一张卡片
  - void **last**(Container parent) 翻转到容器的最后一张卡片
  - void **next**(Container parent) 翻转到指定容器的下一张卡片。如果当前的可见卡片是最后一个，则此方法翻转到布局的第一张卡片

布局管理：CardLayout

- 常用方法
  - void **previous**(Container parent) 翻转到指定容器的前一张卡片。如果当前的可见卡片是第一个，则此方法翻转到布局的最后一张卡片
  - void **show**(Container parent, String name) 翻转到指定名字的卡片
- 该布局下，组件加入容器的方法
  - Component **add**(String name, Component comp) **name**用于指定卡片的名称

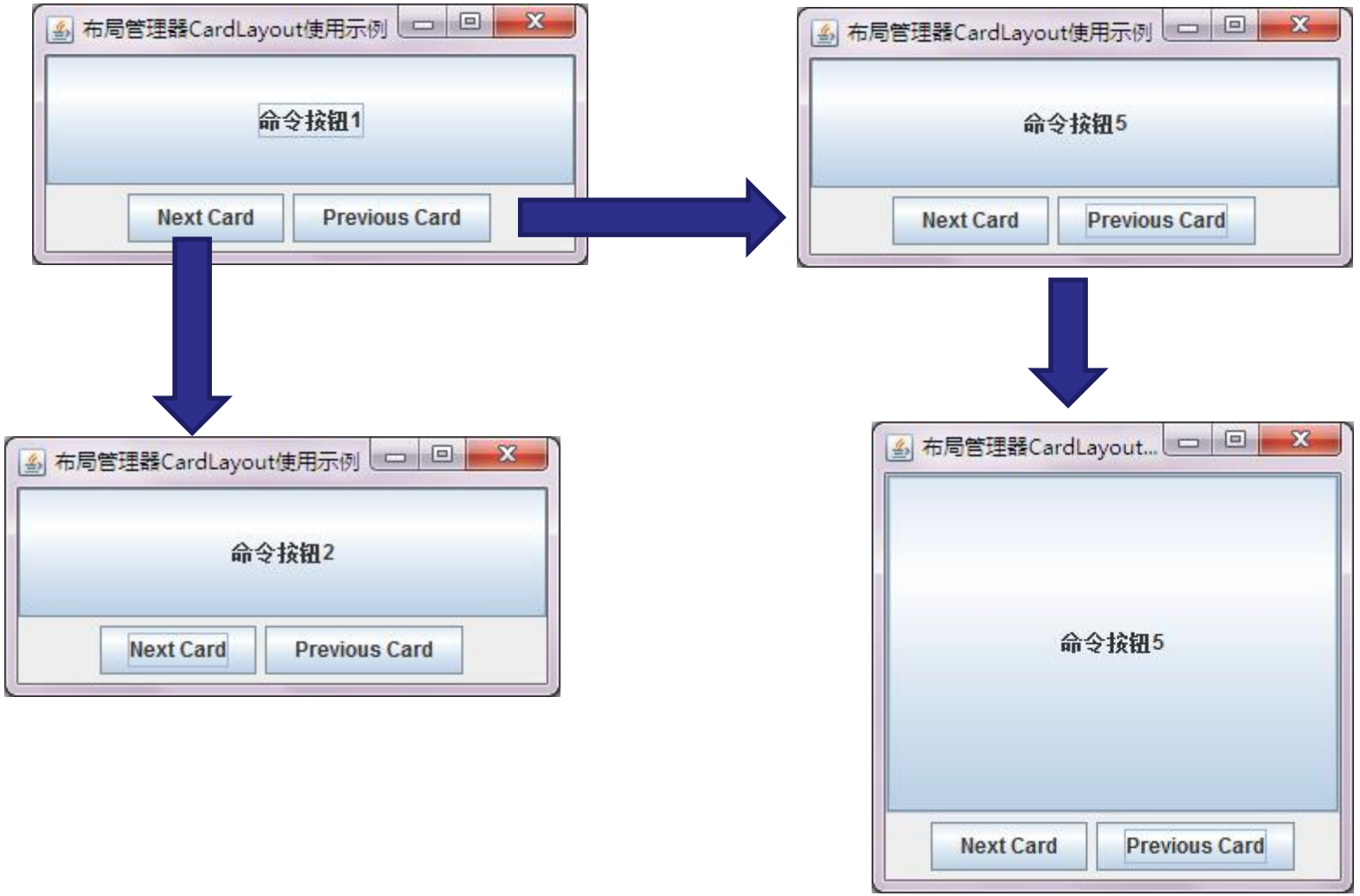
布局管理：CardLayout

【例8-12】使用CardLayout布局JFrame中的若干组件

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class CardLayoutTester extends JFrame {
   public CardLayoutTester(){
        super("布局管理器CardLayout使用示例");
        this.setSize(320, 150);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel cardPanel=new JPanel();
        JPanel controlPanel=new JPanel();
        this.add(cardPanel, "Center"); //添加到默认布局的中部
        this.add(controlPanel, "South"); //添加到默认布局的南部
        CardLayout layout=new CardLayout();
        cardPanel.setLayout(layout); //给面板容器设置布局
        for(int i=1;i<=5;i++)
          cardPanel.add("card"+i,new JButton("命令按钮"+i)); //向
   容器添加卡片
```

```
        JButton btnNext=new JButton("Next Card");
        JButton btnPrevious=new JButton("Previous Card");
        controlPanel.add(btnNext);
        controlPanel.add(btnPrevious);
        //为命令按钮添加事件处理程序：定义事件监听器以及注册监听器
        btnNext.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e)
                {   layout.next(cardPanel); }
        });
        btnPrevious.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e)
                {   layout.previous(cardPanel); }
        });
    }
    public static void main(String[] args) {
        CardLayoutTester frm=new CardLayoutTester();
        frm.setVisible(true);
    }
}
```

布局管理：CardLayout



布局管理：BoxLayout

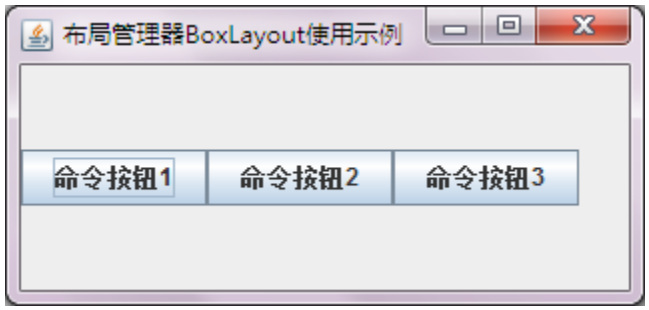
- 容器中组件总是垂直（自上而下）地布置或水平（从左到右）地布置
- 垂直布局试图使所有组件具有相同的宽度（最宽组件的宽度），水平布局试图使所有组件具有相同的高度（最高组件的高度）
- 构造方法
  - **BoxLayout**(Container target, int axis)：创建一个沿给定轴放置组件的布局管理器
    - **target -** 需要布置的容器
    - **axis -** 布置组件时使用的轴。它可以是以下值之一：**BoxLayout.X_AXIS**、**BoxLayout.Y_AXIS**

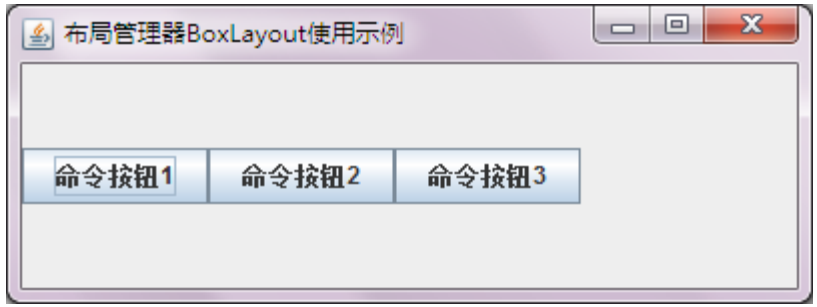
布局管理：BoxLayout

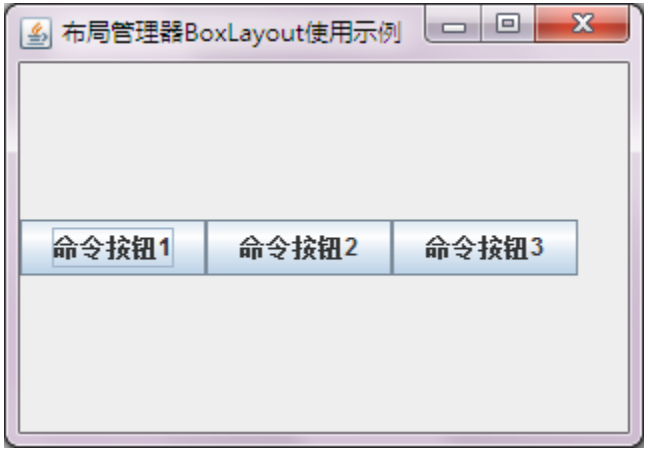
【例8-13】使用BoxLayout布局布置JFrame中的若干按钮

```
import javax.swing.*;
public class BoxLayoutTester {
   public static void main(String[] args) {
      JFrame frm=new JFrame("布局管理器BoxLayout使用示例");
       frm.setSize(320, 150);
       frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       BoxLayout layout=new
            BoxLayout(frm.getContentPane(),BoxLayout.X_AXIS);
       frm.setLayout(layout);

       for(int i=1;i<=3;i++)
                  frm.add(new JButton("命令按钮"+i));

          frm.setVisible(true);
   }
}
```

布局管理：BoxLayout







- 组件之间的间隔为零
- 许多程序使用 Box 类，而不是直接使用 BoxLayout。Box 类是默认使用 BoxLayout 的轻量级容器。

布局管理器：BoxLayout

- **Box类**
  - **BoxLayout布局通常与 Box 容器联合使用**
  - 常用创建方法
    - **Box**(int axis) 创建一个沿指定坐标轴显示其组件的 **Box**
    - static Box **createHorizontalBox()** 创建一个从左到右显示其组件的 **Box**
    - static Box **createVerticalBox()** 创建一个从上到下显示其组件的 **Box**

布局管理器：BoxLayout

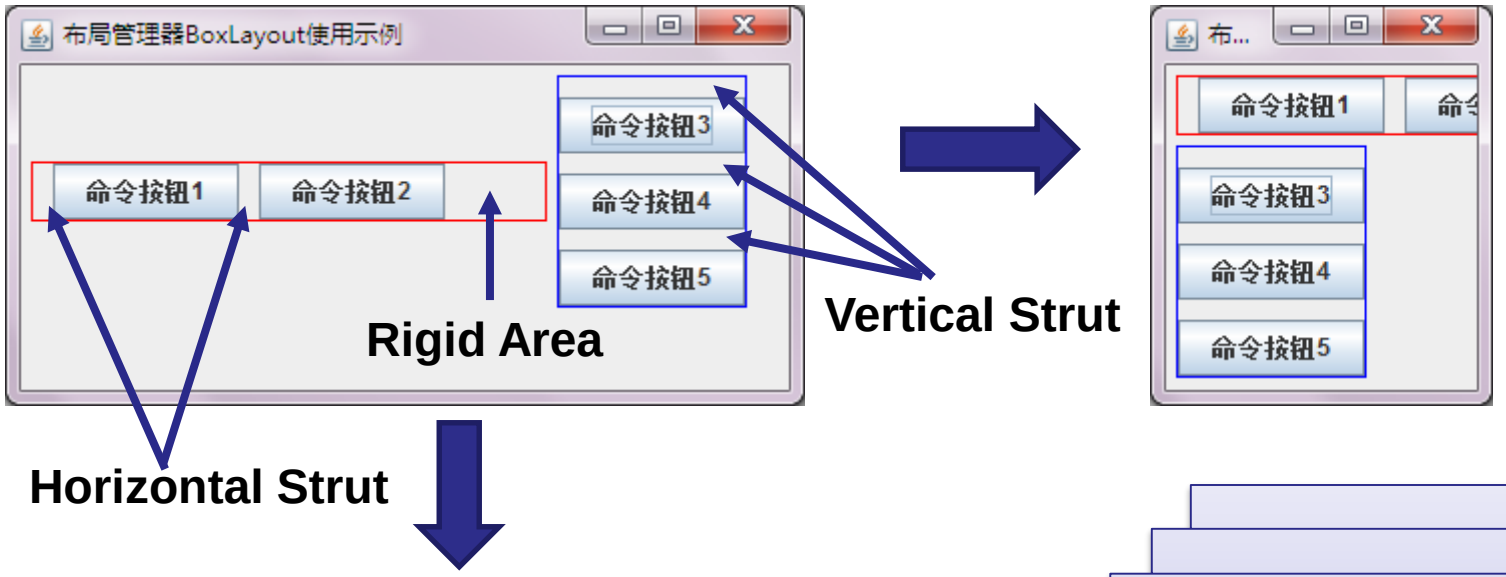
- Box类
  - Box 还提供了若干静态方法，用于创建决定组件之间间隔的的不可见组件
    - 水平/垂直Strut组件：具有固定的宽度/高度，用于确保GUI组件之间保持固定的间隔
      - static Component **createHorizontalStrut**(int height)
      - static Component **createVerticalStrut**(int height)
    - RigidArea组件：具有固定尺寸，容器调整大小时，其尺寸不变
      - static Component **createRigidArea**(Dimension d)
    - 水平/垂直Glue组件：占据Box容器的剩余空间，容器缩放时自动缩放
      - static Component **createHorizontalGlue**()
      - static Component **createVerticalGlue**()

布局管理器： BoxLayout

【例8-14】使用Box容器，该容器默认布局为BoxLayout

```
import java.awt.*;  import javax.swing.*;
public class BoxAndBoxLayoutTester {
   public static void main(String[] args) {
      JFrame frm=new JFrame("布局管理器BoxLayout使用示例");
       frm.setSize(400, 200);
       frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       frm.setLayout(new FlowLayout(FlowLayout.LEFT));
       Box hBox=Box.createHorizontalBox();
       for(int i=1;i<=2;i++){
            hBox.add(Box.createHorizontalStrut(10));
            hBox.add(new JButton("命令按钮"+i));   }
       hBox.add(Box.createRigidArea(new Dimension(50,20)));
       hBox.setBorder(BorderFactory.createLineBorder(Color.RED, 1));
       frm.add(hBox);
       Box vBox=Box.createVerticalBox();
       for(int i=3;i<=5;i++){
           vBox.add(Box.createVerticalStrut(10));
           vBox.add(new JButton("命令按钮"+i));   }
       vBox.setBorder(BorderFactory.createLineBorder(Color.BLUE, 1));
       frm.add(vBox);
       frm.setVisible(true);  }
}
```

布局管理：BoxLayout

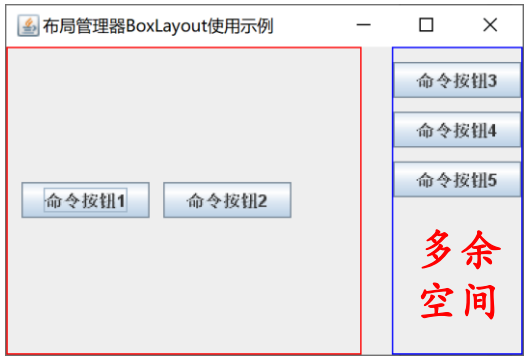


- Box容器中组件保持垂直或水平排列
- 不可见组件用于分隔可见组件

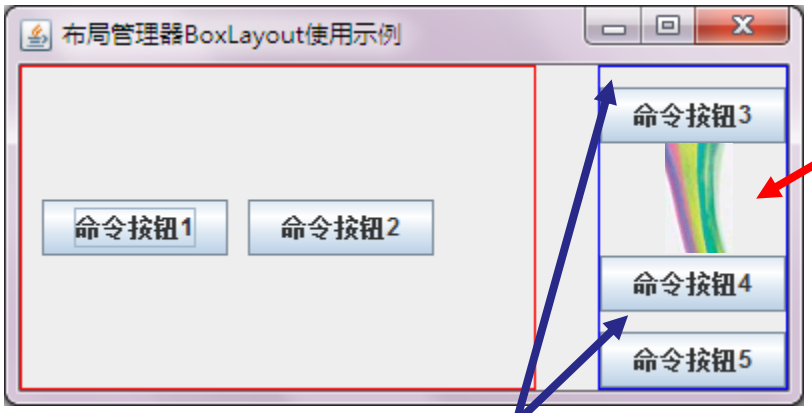
布局管理：BoxLayout

【例8-14'】将上例中窗体布局改为默认的边框布局，此时Box容器可能有多余的空间，因此采用Glue组件来填充。

```
// frm.setLayout(new FlowLayout(FlowLayout.LEFT));
Box hBox=Box.createHorizontalBox();
for(int i=1;i<=2;i++){
    hBox.add(Box.createHorizontalStrut(10));
    hBox.add(new JButton("命令按钮"+i));  }
hBox.add(Box.createRigidArea(new Dimension(50,20)));
hBox.setBorder(BorderFactory.createLineBorder(Color.RED, 1));
frm.add(hBox,"West");

Box vBox=Box.createVerticalBox();
for(int i=3;i<=5;i++){
    if(i==4)
        vBox.add(Box.createVerticalGlue());
    else
        vBox.add(Box.createVerticalStrut(10));
    vBox.add(new JButton("命令按钮"+i));
}
vBox.setBorder(BorderFactory.createLineBorder(Color.BLUE, 1));
frm.add(vBox,"East");
frm.setVisible(true);
```

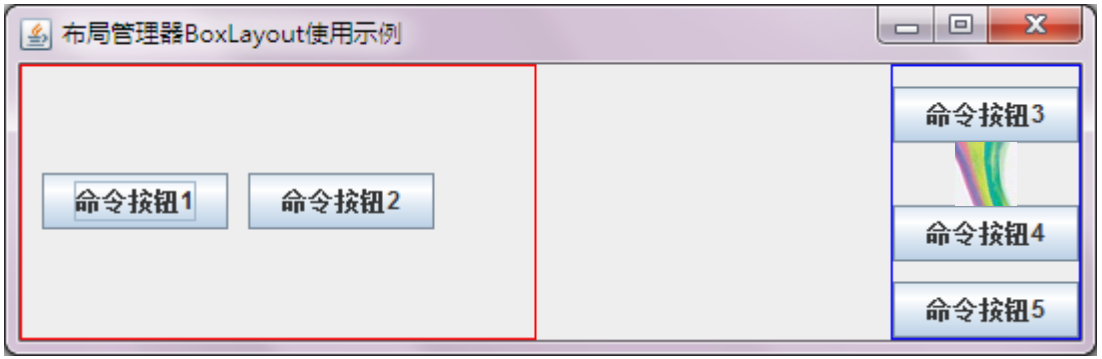


布局管理：BoxLayout



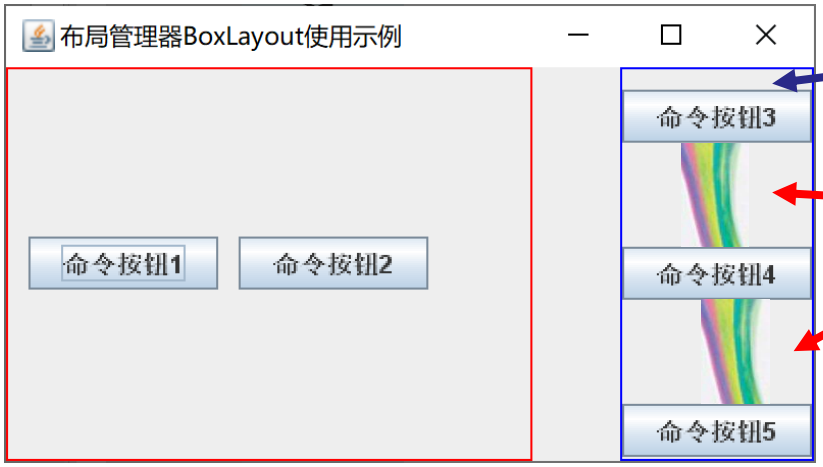
Vertical Glue

Vertical Strut

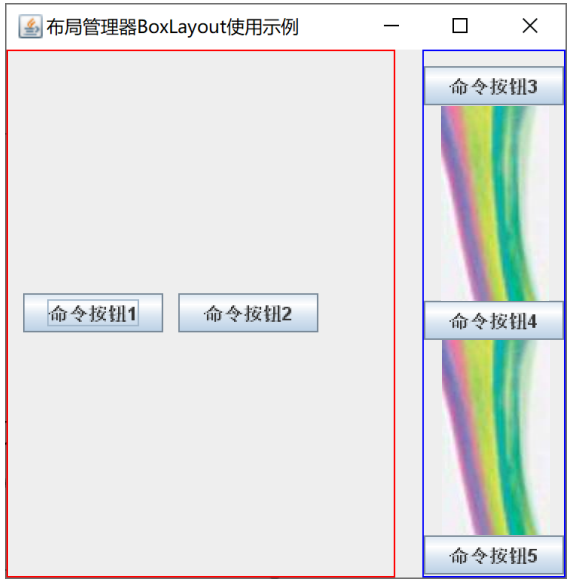


- Box容器中的不可见组件Glue大小随Box缩放而缩放
- Box中有剩余空间时才起作用

布局管理：BoxLayout



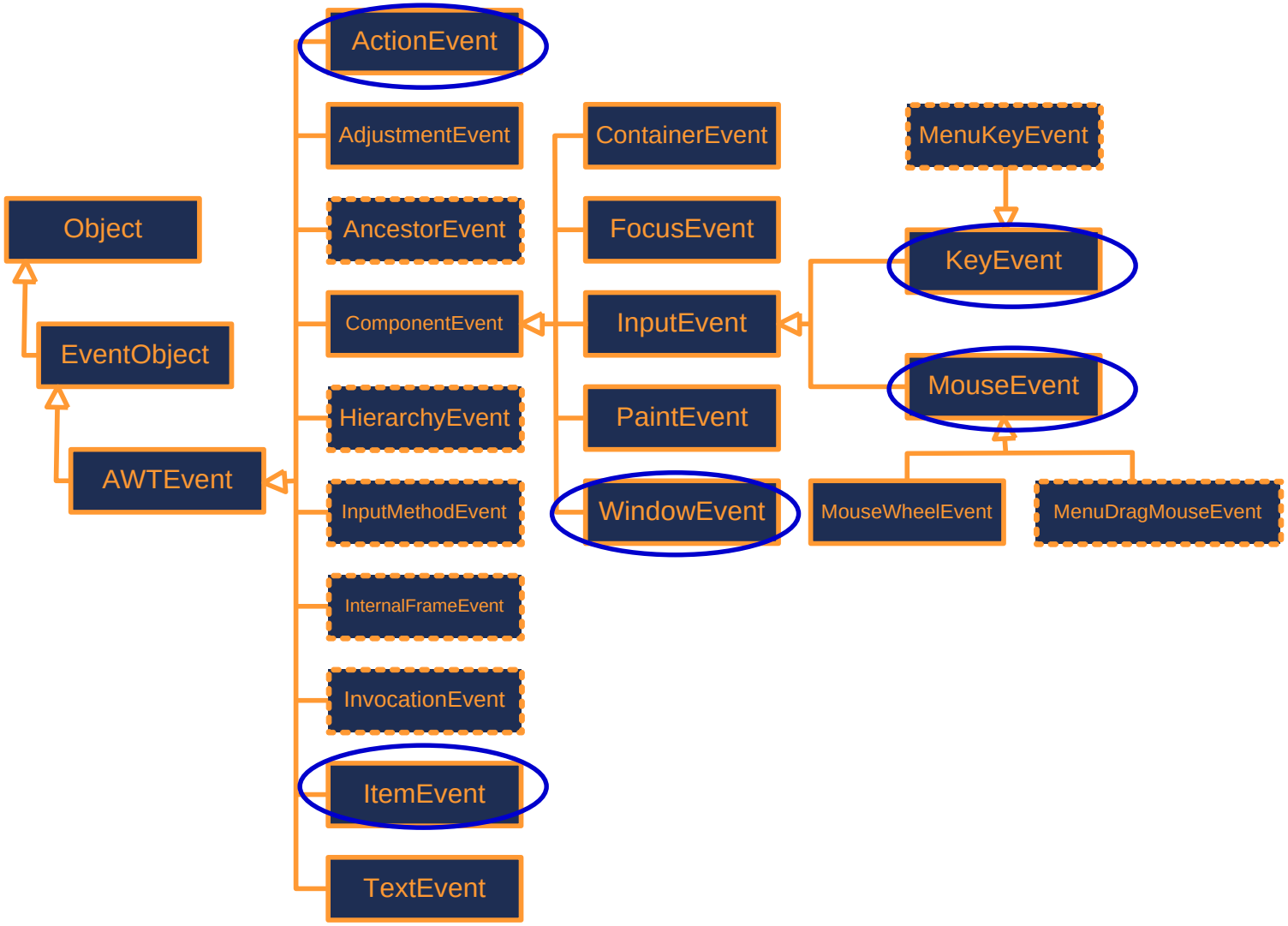

```
Box vBox=Box.createVerticalBox();
for(int i=3;i<=5;i++){
   if(i>3)
       vBox.add(Box.createVerticalGlue());
   else
       vBox.add(Box.createVerticalStrut(10));
   vBox.add(new JButton("命令按钮"+i));
}
```



Java事件处理机制

- Java程序与用户的交互是通过响应各种事件来实现的
- 用户操作GUI组件时可能引发各种事件。常见的事件有
  - 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，JVM会将事件对象传递给GUI应用程序，程序通过事件对象获得事件的有关信息，然后根据事件的类型做出相应的响应

Java事件处理机制



Swing组件的事件对象层次结构

Java事件处理机制：三类对象

- 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

Java事件处理机制：三类对象

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

Java事件处理机制：事件源

- 事件源
  - 提供注册监听器或取消监听器的方法
  - 维护一个已注册的监听器列表，如有事件发生，就会通知每个已注册的监听器
- 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响应，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

Java事件处理机制：事件监听器

- 是一个对象，需要先定义实现了对应事件监听接口的监听器类，然后创建其实例(即监听器)，并通过事件源的add×××Listener方法将其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，用于处理鼠标事件的监听器接口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

Java事件处理机制：事件对象

- Swing事件源可能触发的事件及监听器要实现的接口

事件源	事件对象	事件监听器要实现的接口
JFrame	MouseEvent WindowEvent	MouseListener WindowListener
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

Java事件处理机制：事件对象

- Swing事件源可能触发的事件及事件监听器

事件源	事件对象	事件监听器（接口）
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

Java事件处理机制：事件对象

- Swing事件源可能触发的事件及事件监听器

事件源	事件对象	事件监听器（接口）
JProgressBar JSlider JTabbedPane	ChangeEvent	ChangeListener
JScrollBar	AdjustmentEvent	AdjustmentListener
JTable	ListSelectionEvent TableModelEvent	ListSelectionListener TableModelListener
JTree	TreeSelectionEvent TreeExpansionEvent	TreeSelectionListener TreeExpansionListener
Timer	ActionEvent	ActionListener

Java事件处理机制：事件对象

- 每种事件类型都提供了一些get方法用于获取事件对象中的信息。例如：

事件类型	常用方法
ActionEvent	String **getActionCommand()** 获取引发事件的组件的命令名
ItemEvent	Object **getItem()** 获取引发事件的组件对象引用 int **getStateChange()** 获取当前组件的状态值（**ItemEvent.SELECTED**或**ItemEvent.DESELECTED**）
KeyEvent	char **getKeyChar()** 返回与此事件中的键关联的字符。例如，**shift + "a"** 的 **KEY_TYPED** 事件返回值 **"A"**
MouseEvent	int **getX()** 返回事件相对于源组件的水平 **x** 坐标 int **getY()** 返回事件相对于源组件的垂直 **y** 坐标 Point **getPoint()** 返回事件相对于源组件的 **x, y** 坐标

Java事件处理机制：监听器接口与适配器类

- 事件监听器接口
  - 每种事件都有一个对应的事件监听接口，接口中声明了一个或多个抽象的事件处理方法。
  - 常见事件监听器接口及其声明的方法见表格
  - 如果一个类实现了某个事件监听接口，则该类就具备了响应和处理该事件的能力
  - 例如MouseListener是一个MouseEvent事件的监听接口，为了在程序中创建一个鼠标事件监听器的对象，我们需要实现该接口，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

事件监听器接口	接口中声明的抽象方法
ActionListener 动作事件监听器接口	void **actionPerformed**(ActionEvent e) 处理动作事件。单击按钮、菜单项等时调用
ItemListener 选项事件监听器接口	void **itemStateChanged**(ItemEvent e) 处理选项事件。单复选按钮、列表等选项改变时调用
KeyListener 键盘事件监听器接口	void **keyPressed**(KeyEvent e) 按下某个键时调用 void **keyReleased**(KeyEvent e) 释放某个键时调用 void **keyTyped**(KeyEvent e) 返回按下并释放某个键时调用
MouseListener 鼠标事件监听器接口	void **mouseClicked**(MouseEvent e) 鼠标按键在组件上单击（按下并释放）时调用 void **mousePressed**(MouseEvent e) 鼠标按键在组件上按下时调用 void **mouseReleased**(MouseEvent e) 鼠标按钮在组件上释放时调用 void **mouseEntered**(MouseEvent e) 鼠标进入到组件上时调用 void **mouseExited**(MouseEvent e) 鼠标离开组件时调用

事件监听器接口	接口中声明的抽象方法
MouseMotionListener 鼠标事件监听器接口	void **mouseMoved**(MouseEvent e) 鼠标光标在组件上移动时调用 void **mouseDragged**(MouseEvent e)鼠标按键在组件上按下并拖动时调用。
WindowListener 窗口事件监听器接口	void **windowClosing**(WindowEvent e) 用户试图从窗口的系统菜单中关闭窗口时调用 void **windowClosed**(WindowEvent e) 因对窗口调用 **dispose** 而将其关闭时调用 void **windowOpened**(WindowEvent e) 窗口首次打开并变为可见时调用 void **windowIconified**(WindowEvent e) 窗口从正常状态变为最小化状态时调用 void **windowDeiconified**(WindowEvent e) 窗口从最小化状态变为正常状态时调用 void **windowActivated**(WindowEvent e)窗口被激活时调用 void **windowDeactivated**(WindowEvent e)窗口去激活时调用

Java事件处理机制：监听器接口与适配器类

- 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

事件监听器接口	对应于的适配器类
KeyListener	KeyAdapter
MouseListener	MouseAdapter
MouseMotionListener	MouseMotionAdapter
WindowListener	WindowsAdapter

Java事件处理机制：三种实现事件处理的方法

- 方法1：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
- 方法2：事件监听器从其对应的适配器类继承
  - 只需要重写我们感兴趣的事件
- 方法3：使用内部类（匿名或命名，从适配器继承）
  - 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

Java事件处理机制：三种实现事件处理的方法

【例8-15】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
import java.awt.event.*;    import javax.swing.*;
public class ImplementMouseListener extends JFrame implements
    MouseListener{
    public ImplementMouseListener () {
        super("窗体实现MouseListener接口");
        this.setSize(300,150);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.addMouseListener(this); //为窗口注册鼠标事件监听器
    }
     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){
        this.setTitle("点击坐标为 ("+e.getX()+", "+e.getY()+")");    }
    public static void main(String[] args){
      ImplementMouseListener frm=new ImplementMouseListener ();
      frm.setVisible(true);
    }
}
```

实现了监听接口的监听器对象

Java事件处理机制：三种实现事件处理的方法

```
//方法2：继承MouseAdapter类
import java.awt.event.*;    import javax.swing.*;
class MyAdapter extends MouseAdapter { // 定义鼠标事件监听器
    public void mouseClicked(MouseEvent e) {
            JFrame frm = (JFrame) e.getSource();
            frm.setTitle("点击坐标为 (" + e.getX() + ", " + e.getY() + ")");
    }
}
public class AdapterTester extends JFrame {
    public AdapterTester() {
            this.setSize(300, 150);
            this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            this.addMouseListener(new MyAdapter()); // 为窗口注册鼠标
    事件监听器
    }
    public static void main(String[] args) {
            AdapterTester f = new AdapterTester();
            f.setVisible(true);
    }
}
```

Java事件处理机制：三种实现事件处理的方法

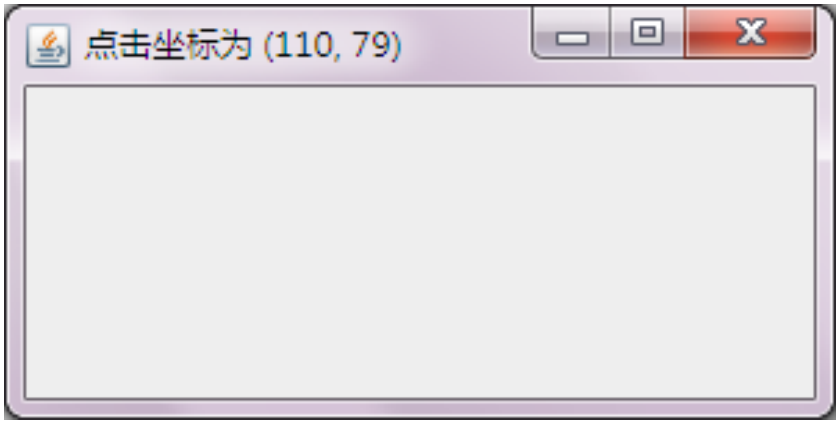
```
//方法3：使用内部类---命名类
import java.awt.event.*;    import javax.swing.*;
public class UseInnerClass extends JFrame{
    public UseInnerClass () {
        this. setSize(300,150);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this. addMouseListener(new MAClass()); //为窗口注册鼠标事件监听器
    }
    class MAClass extends MouseAdapter{ // 内部类，定义监听器
        public void mouseClicked(MouseEvent e){
            setTitle("点击坐标为 ("+e.getX()+", "+e.getY()+")");
        }
    }
    public static void main(String[] args){
            UseInnerClass frm=new UseInnerClass ();
            frm.setVisible(true);     }
}
```

Java事件处理机制：三种实现事件处理的方法

```
//方法3：使用内部类---匿名类
import java.awt.event.*;    import javax.swing.*;
public class UseInnerClass extends JFrame{
    public UseInnerClass () {
        this.setSize(300,150);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e){
                setTitle("点击坐标为 ("+e.getX()+", "+e.getY()+")");
            }
        }); //为窗口定义并注册鼠标事件监听器
    }
    public static void main(String[] args){
            UseInnerClass frm=new UseInnerClass ();
            frm.setVisible(true);
    }
}
```

Java事件处理机制：三种实现事件处理的方法

采用三种不同方法的程序，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



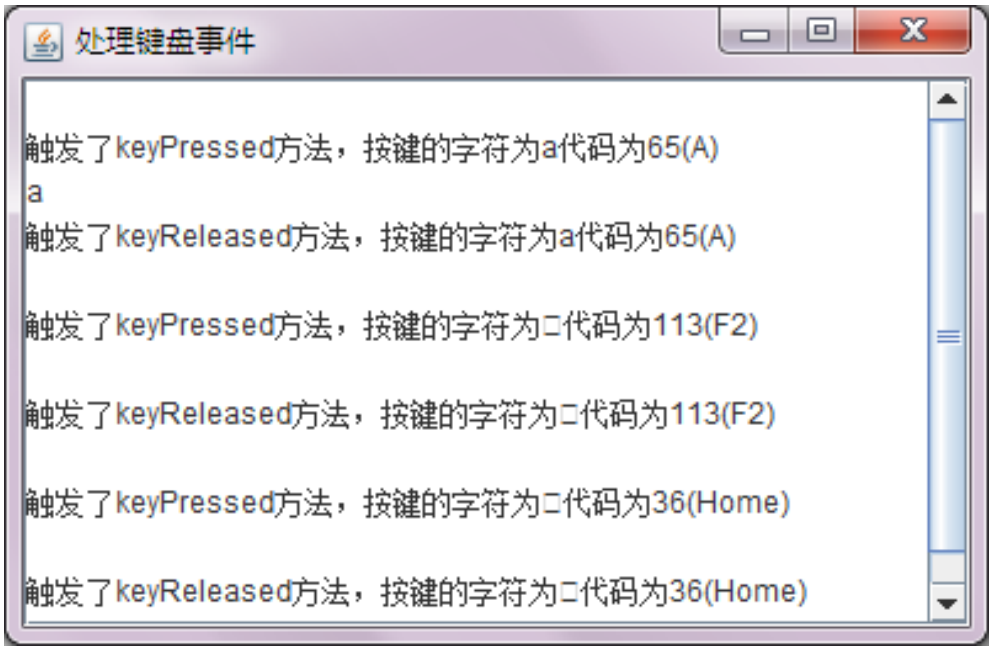
Java事件处理机制：三种实现事件处理的方法

【例8-16】为文本区域添加键盘事件处理方法

```
import java.awt.event.*;       import javax.swing.*;
public class KeyEventTester extends JFrame{
   public KeyEventTester(){
     super("处理键盘事件");
     this.setSize(400,260);
     this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     JTextArea ta=new JTextArea();
      ta.addKeyListener(new KeyAdapter() { //注册键盘事件监听器
         public void keyPressed(KeyEvent e){
            ta.append("\n触发了keyPressed方法，按键的字符为"
               +e.getKeyChar()+"代码为"+e.getKeyCode()
               +"("+KeyEvent.getKeyText(e.getKeyCode())+")\n"); }
         public void keyReleased(KeyEvent e){
            ta.append("\n触发了keyReleased方法，按键的字符为"
              +e.getKeyChar()+"代码为"+e.getKeyCode()
              +"("+KeyEvent.getKeyText(e.getKeyCode())+")\n");}
         });
```

Java事件处理机制：三种实现事件处理的方法

```
        JScrollPane sp=new JScrollPane(ta,
            JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
            JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
        this.add(sp);
    }
    public static void main(String[] args) {
        KeyEventTester frm=new KeyEventTester ();
        frm.setVisible(true);
    }
}
```



Swing原子组件

- Swing原子组件有很多种，与顶层容器和中间容器相比，原子组件用法都比较简单
- 可将其分为三类
  - 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

Swing原子组件：第1类

- **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。通过使用该组件可以轻松地为软件加上一个进度条
- 提示信息
  - 通常不必直接处理**JToolTip**类
  - 通常使用**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

Swing原子组件：第1类之JLabel

- **JLabel** 对象可以显示文本、图像或同时显示二者。标签不对输入事件作出反应。因此，它无法获得键盘焦点
- 可以通过设置垂直和水平对齐方式。默认情况下，标签在其显示区内垂直居中对齐，只显示文本的标签是开始边对齐；而只显示图像的标签则水平居中对齐
- 常用构造方法
  - **JLabel**(String text) 创建带有指定文本的标签
  - **JLabel**(String text, int horizontalAlignment) 创建指定文本和对齐方式的标签。对齐方式有**JLabel.LEFT、CENTER、RIGHT、LEADING 或 TRAILING**
  - **JLabel**(Icon image) 创建带有指定图像的标签
  - **JLabel**(Icon image, int horizontalAlignment)
  - **JLabel**(String text, Icon icon, int horizontalAlignment)

Swing原子组件：第1类之JLabel

- 常用方法
  - void **setText**(String text) 设置标签要显示的文本
  - void **setIcon**(Icon icon) 设置标签要显示的图像
  - void **setVerticalAlignment**(int alignment) 设置标签内容沿 **Y** 轴的对齐方式
  - void **setHorizontalAlignment**(int alignment) 设置标签内容沿 **X** 轴的对齐方式

Swing原子组件：第1类之JLabel

【例8-17】用JLabel显示文本或图像

```
import javax.swing.*;
public class JLabelTester{
   public static void main(String[] args) {
      JFrame frm=new JFrame("JLabel用于显示文本或图像");
      frm.setSize(300,200);
      frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JLabel lbl1=new JLabel("在JLabel中显示文本",JLabel.CENTER);
      JLabel lbl2=new JLabel(new ImageIcon("img/a.jpg"));
      frm.add(lbl1, "North");
      frm.add(lbl2, "South");
      frm.setVisible(true);
   }
}
```

Swing原子组件：第1类之JProgressBar

- 以可视化形式显示某些任务进度的组件。在任务的完成进度中，进度条显示该任务完成的百分比。此百分比通常由一个矩形以可视化形式表示，该矩形开始是空的，随着任务的完成逐渐被填充。此外，进度条可显示此百分比的文本表示形式
- 进度条使用 **BoundedRangeModel** 作为其数据模型，以 **value** 属性表示该任务的“当前”状态，**minimum** 和 **maximum** 属性分别表示开始点和结束点。
- 常用构造方法
  - **JProgressBar()** 显示边框但不带进度字符串的水平进度条。初始值和最小值都为 **0**，最大值为 **100**
  - **JProgressBar**(int orient) 指定方向（**JProgressBar.VERTICAL**、**HORIZONTAL**）
  - **JProgressBar**(int min, int max) 指定最小值和最大值的水平进度条
  - **JProgressBar**(int orient, int min, int max)

Swing原子组件：第1类之JProgressBar

- 常用方法
  - void **setOrientation**(int newOrientation) 设置方向
  - void **setMinimum**(int n) 设置最小值
  - void **setMaximum**(int n) 设置最大值
  - void **setValue**(int n) 设置当前进度值
  - void **setString**(String s) 设置进度字符串的值。默认情况下，此字符串为 **null**，隐含使用简单百分比字符串的内置行为
  - void **setStringPainted**(boolean b) 设置是否显示进度字符串
- **changeEvent事件**
  - 当进度条的当前值改变时，就触发该事件
  - 对应的事件监听器接口为**changeListener**

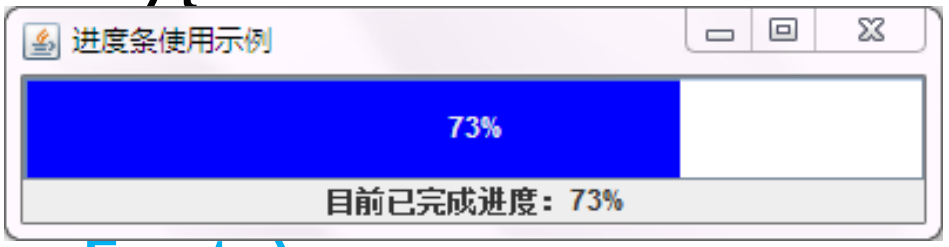
Swing原子组件：第1类之JProgressBar

【例8-18】进度条使用示例

```
import java.awt.*; import javax.swing.*;   import javax.swing.event.*;
public class JProgressBarTester extends JFrame
                                        implements ChangeListener {
   JLabel label;
   JProgressBar pb;
   public JProgressBarTester(){
      super("进度条使用示例");
      this.setSize(400,100);
      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      label=new JLabel("",JLabel.CENTER);
      label.setToolTipText("显示进度信息"); //设置提示信息
      int value=0;
      pb=new JProgressBar(JProgressBar.HORIZONTAL,0,100);
      pb.setForeground(Color.BLUE); // 进度颜色
      pb.setBackground(Color.WHITE); // 背景颜色
```

Swing原子组件：第1类之JProgressBar

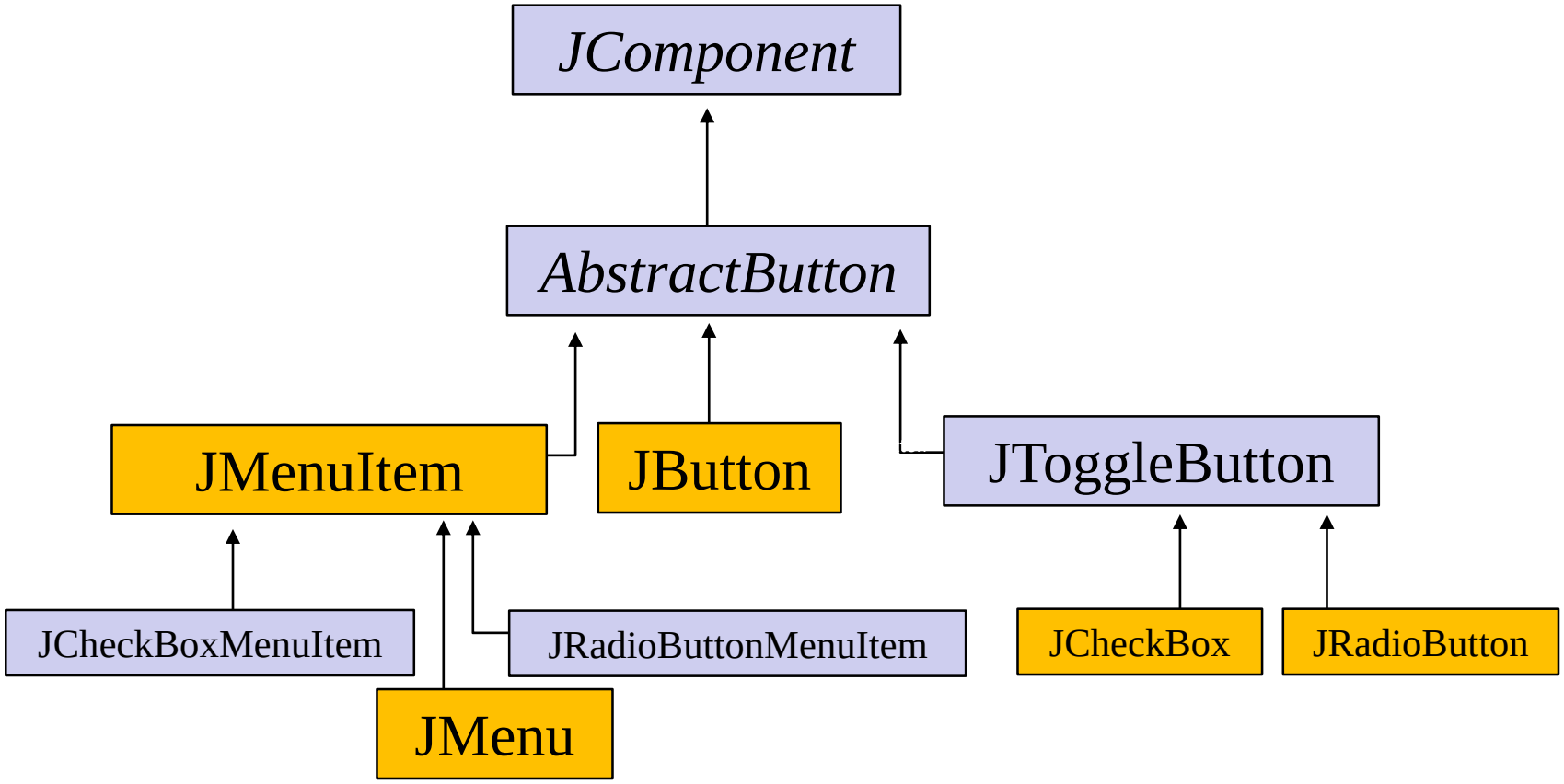
```
    pb.setValue(value);              pb.setStringPainted(true);
    pb.setToolTipText("进度条"); //设置提示信息
    pb.addChangeListener(this); //注册事件监听器
    add(label,"South");  add(pb, "Center");  setVisible(true);
    for(int i=0;i<=100;i++){
        pb.setValue(i); //改变进度条的值，触发ChangeEvent
        try {  //main线程随机休眠1~501毫秒
            Thread.sleep((long) (500*Math.random())+1);
        } catch (InterruptedException e) {
        e.printStackTrace(); }
    }
}
public void stateChanged(ChangeEvent e)
{   label.setText("目前已完成进度："+pb.getValue()+"%");    }
public static void main(String[] args)
{   JProgressBarTester frm=new JProgressBarTester();  }
}
```



Swing原子组件：按钮类

- AbstractButton抽象类是众多按钮类的基类，继承它的类包括
  - JButton
  - JToggleButton——表示有两个选择状态的按钮
    - JCheckBox（复选框）
    - JRadioButton（单选按钮）
  - JMenuItem——菜单项
    - JCheckBoxMenuItem（复选）
    - JRadioButtonMenuItem（单选按钮）
  - JMenu ——菜单

Swing原子组件：按钮类



Swing原子组件：按钮类之JButton

- 命令按钮，可以带有文本或图像或两者
- 常用构造方法
  - **JButton**(String text) 创建一个带文本的按钮
  - **JButton**(Icon icon) 创建一个带图标的按钮
  - **JButton**(String text, Icon icon)
- 常用方法
  - void **setEnabled**(boolean b) 启用或禁用按钮
  - boolean **requestFocusInWindow**() 设置焦点
  - void **setActionCommand**(String actionCommand) 添加动作命令
  - void **addActionListener**(ActionListener l) 注册监听器
- **ActionEvent**事件
  - 单击命令按钮将触发动作事件**ActionEvent**
  - 对应的事件监听器接口**ActionListener**
    - 接口的方法**actionPerformed(ActionEvent e)**

Swing原子组件：按钮类之JButton

【例8-19】命令按钮使用示例

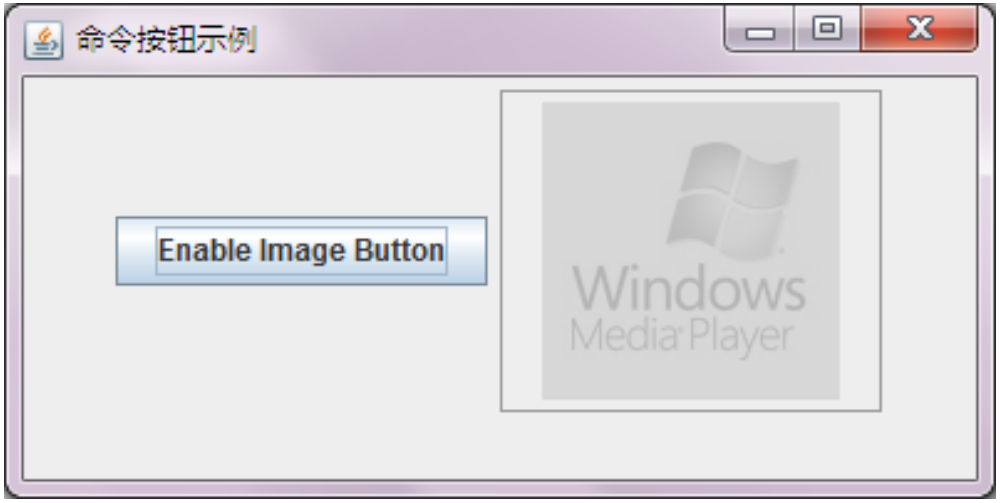
```
import java.awt.*;   import java.awt.event.*;   import javax.swing.*;
public class JButtonTester extends JFrame
                                    implements ActionListener{
   JButton btn1,btn2;
   public JButtonTester(){
         super("命令按钮示例");
         this.setSize(400,200);
         this.setLayout(new FlowLayout());
         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

         btn1=new JButton("Disable Image Button");
         btn1.setActionCommand("aaa");
         btn2=new JButton(new ImageIcon("img/a.jpg"));
         btn2.setActionCommand("bbb");
         btn1.addActionListener(this); btn2.addActionListener(this);
         this.add(btn1);          this.add(btn2);
   }
```

窗体类实现ActionListener接口，故窗体本身就充当监听器

Swing原子组件：按钮类之JButton

```
public void actionPerformed(ActionEvent e){
        String cmd=e.getActionCommand();
        if(cmd.equals("aaa")){
                if(btn2.isEnabled()){
                        btn2.setEnabled(false);
                        btn1.setText("Enable Image Button");
                }else{
                        btn2.setEnabled(true);
                        btn1.setText("Disable Image Button");
                }
        }else if(cmd.equals("bbb"))
                btn1.requestFocusInWindow();//设置焦点
}
public static void main(String[] args) {
        JButtonTester frm=new JButtonTester();
        frm.setVisible(true);
}
}
```

Swing原子组件：按钮类之JButton



【例8-19'】命令按钮使用示例---为各按钮定义独立的监听器

```
public class JButtonTester2 extends JFrame {
    private JButton btn1, btn2;
    public JButtonTester2() {
            …
            btn1.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                            if (btn2.isEnabled()) {
                                    btn2.setEnabled(false);
                                    btn1.setText("Enable Image Button");
                            } else {
                                    btn2.setEnabled(true);
                                    btn1.setText("Disable Image Button");
                            }
                    }
            });
            btn2.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                            btn1.requestFocusInWindow(); // 设置焦点
                    }
            });
           …
    }
    …
}
```

此时，窗体类不用实现ActionListener接口，因此，窗体不充当监听器

Eclipse IDE中添加匿名监听器的小技巧：
输入new加空格后，按 alt+/ 键，从提示列表中双击选择监听器接口或适配器，即可自动生成监听器代码框架。

Swing原子组件：按钮类之JRadioButton

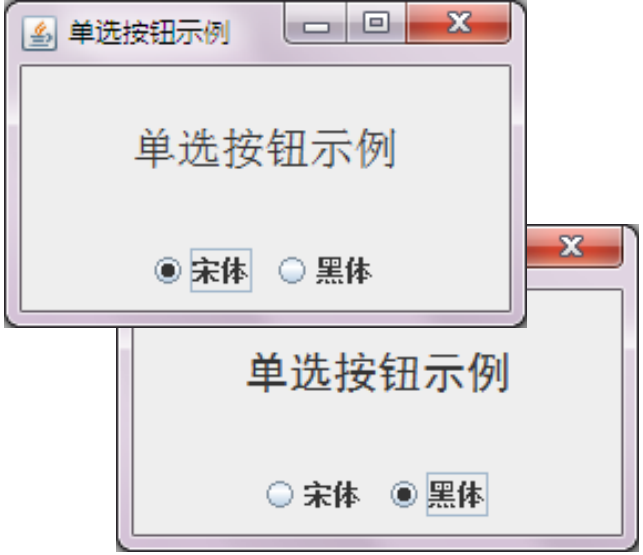
- 有选中和未选中两种状态
- 把多个单选按钮添加到一个按钮组**ButtonGroup**，在任何时候，用户只能选中其中一个按钮
- 常用构造方法
  - **JRadioButton**(String text) 创建一个具有指定文本的默认未选中的单选按钮
  - **JRadioButton**(String text, boolean selected) 创建一个具有指定文本和选择状态的单选按钮
- 常用方法
  - boolean **isSelected**() 返回按钮状态
  - void **setSelected**(boolean b) 设置按钮的状态
  - void **addItemListener**(ItemListener l) 注册选项事件监听器
- **ItemEvent**事件
  - 按钮状态发生变化时将触发选项事件**ItemEvent**
  - 对应的事件监听器接口**ItemListener**
    - 接口的方法**itemStateChanged(ItemEvent e)**

Swing原子组件：按钮类之JRadioButton

【例8-20】单选按钮使用示例：设置文本的字体名称

```
import java.awt.*;   import java.awt.event.*;   import javax.swing.*;
public class JRadioButtonTester extends JFrame
                                        implements ItemListener {
   JLabel label;
   JRadioButton rb1,rb2;
   public JRadioButtonTester(){
         super("单选按钮示例");
         this.setSize(300,200);
         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

         label=new JLabel("单选按钮示例",JLabel.CENTER);
         rb1=new JRadioButton("宋体");
         rb2=new JRadioButton("黑体");
         rb1.addItemListener(this); //注册选项事件监听器
         rb2.addItemListener(this);
```

```
        //单选按钮组，是一个逻辑分组
        ButtonGroup bg=new ButtonGroup();
        bg.add(rb1);  bg.add(rb2);
        JPanel panel=new JPanel();
        panel.add(rb1);    panel.add(rb2);
        this.add(label, "Center");
        this.add(panel, "South");
    }
    public void itemStateChanged(ItemEvent e){
        // 注意，一组单选按钮中由选中A改为选中B，则A、B两个按钮的选项事件ItemEvent都会触发。取消的先触发，选中的后触发
        // 在本例，也就是会两次调用事件处理方法 itemStateChanged
        System.out.println("call itemStateChanged");
        if(e.getStateChange()==ItemEvent.SELECTED){ //如果事件中选项的状态变化是被选中了，即e的事件源是被选中的那个选项按钮
            JRadioButton rb=(JRadioButton)e.getSource();
            label.setFont(new Font(rb.getText(),Font.PLAIN,20));
        }
    }
    public static void main(String[] args) {
        JRadioButtonTester frm=new JRadioButtonTester();
        frm.setVisible(true);
    }
}
```



注意，这里两个事件源共用了一个事件处理方法

【例8-20’】修改上例，组合使用单选按钮与命令按钮设置字体


```
public JRadioButtonTester2(){
        …
        btn=new JButton("设置字体");
        btn.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                JRadioButton rb=null; //引用某个单选按钮
                if(rb1.isSelected())  //rb1选中
                    rb=rb1;
                else if(rb2.isSelected()) //rb2选中
                    rb=rb2;
                if(rb==null) return;
                label.setFont(new Font(rb.getText(),Font.PLAIN,20));
            }
        });
        JPanel panel=new JPanel();
        panel.add(rb1);   panel.add(rb2);   // 按钮组加入面板
        panel.add(btn);

        this.add(label, "Center");   this.add(panel, "South");
}
```

Swing原子组件：按钮类之JCheckBox

- 有选中和未选中两种状态
- 复选框一般用于多选
- 常用构造方法
  - **JCheckBox**(String text) 创建一个具有指定文本的默认未选中的复选框
  - **JCheckBox** (String text, boolean selected) 创建一个具有指定文本和选择状态的复选框
- 常用方法（同**JRadioButton**）
  - boolean **isSelected**() 返回复选框状态
  - void **setSelected**(boolean b) 设置复选框的状态
  - void **addItemListener**(ItemListener l) 注册选项事件监听器
- **ItemEvent**事件
  - 复选框状态发生变化时将触发选项事件**ItemEvent**
  - 对应的事件监听器接口**ItemListener**
    - 接口的方法**itemStateChanged(ItemEvent e)**

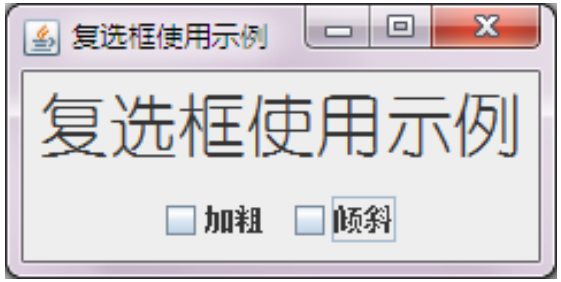
Swing原子组件：按钮类之JCheckBox

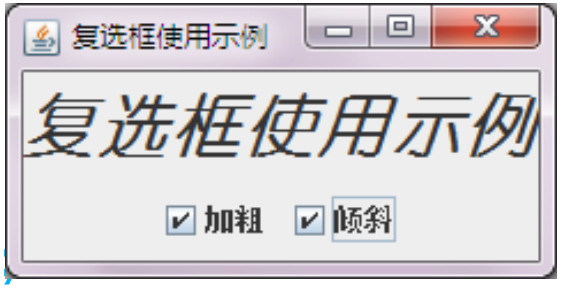
【例8-21】复选框使用示例：设置文本的字体样式

```
import java.awt.*;  import java.awt.event.*;  import javax.swing.*;
public class JCheckBoxTester extends JFrame
                                    implements ItemListener {
   JLabel label;
   JCheckBox cb1,cb2;
   public JCheckBoxTester(){
      super("复选框使用示例");
      this.setSize(240,120);
      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      label=new JLabel("复选框使用示例",JLabel.CENTER);
      label.setFont(new Font("幼圆",Font.PLAIN,30)); //设置字体
      cb1=new JCheckBox("加粗");
      cb2=new JCheckBox("倾斜");
      cb1.addItemListener(this); //注册选项事件监听器
      cb2.addItemListener(this);
```

Swing原子组件：按钮类之JRadioButton

```
        JPanel panel=new JPanel();
        panel.add(cb1);
        panel.add(cb2);
        this.add(label, "Center");
        this.add(panel, "South");
    }
    public void itemStateChanged(ItemEvent e){
        int fontStyle=Font.PLAIN;
        if(cb1.isSelected()) fontStyle+=Font.BOLD;
        if(cb2.isSelected()) fontStyle+=Font.ITALIC;
        Font font=label.getFont();
        label.setFont(new Font(font.getName(),
                               fontStyle, font.getSize()));
    }
    public static void main(String[] args) {
            JCheckBoxTester frm=new JCheckBoxTester();
            frm.setVisible(true);  }
}
```





注意，这里两个事件源也共用了一个事件处理方法

【例8-21’】修改上例，复选框的选项监听器通过匿名类分别定义

```
cb1.addItemListener(new ItemListener() {
    public void itemStateChanged(ItemEvent e) {
        Font font=label.getFont(); //获取标签当前字体
        int fontStyle=font.getStyle();
        if(e.getStateChange()==ItemEvent.SELECTED)
            fontStyle+=Font.BOLD;
        else
            fontStyle-=Font.BOLD;
        label.setFont(new Font(font.getName(),fontStyle,font.getSize()));
    }
});
cb2.addItemListener(new ItemListener() {
    public void itemStateChanged(ItemEvent e) {
        Font font=label.getFont(); //获取标签当前字体
        int fontStyle=font.getStyle();
        if(e.getStateChange()==ItemEvent.SELECTED)
            fontStyle+=Font.ITALIC;
        else
            fontStyle-=Font.ITALIC;
        label.setFont(new Font(font.getName(),fontStyle,font.getSize()));

    }
});
```

此时，窗体类不用实现ItemListener接口，因此，窗体不充当监听器

Swing原子组件：列表类之JComboBox

- 组合框—命令按钮或文本框与下拉列表组合的组件
- 用户可以从下拉列表中选择值，下拉列表在用户请求时显示。
- 用户可以从下拉列表中选择一项，与单选按钮相比，更节省GUI空间
- 默认不可编辑。如果使组合框处于可编辑状态，则组合框将包括用户可在其中键入值的文本框
- 列表中的元素可以是任意类型的对象，界面呈现的是对象的字符串表示（toString()方法返回的值）
- 组合框实际上就对应着一个数据模型ComboBoxModel
- 构造方法
  - JComboBox<E>() 创建具有空对象列表的组合框
  - JComboBox<E>(E[] items) 创建包含指定数组中的元素的组合框。默认情况下，选择数组中的第一项
  - JComboBox<E>(Vector<E> items) 创建包含指定向量中的元素的组合框。默认情况下，选择向量中的第一项
  - JComboBox<E>(ComboBoxModel aModel)：关联自现有的ComboBoxModel

Swing原子组件：列表类之JComboBox

- 常用方法
  - void **addItem**(Object anObject) 向下拉列表添加选项
  - void **insertItemAt**(Object anObject, int index) 在下拉列表的指定位置处插入选项
  - void **removeItemAt**(int anIndex) 移除指定位置处选项
  - void **removeAllItems**() 移除所有选项
  - void **setSelectedItem**(Object anObject) 将组合框显示区域中所选项设置为参数中的对象
  - void **setSelectedIndex**(int anIndex) 选择指定位置处选项
  - Object **getSelectedItem**() 返回所选项（也就是一个对象）
  - int **getSelectedIndex**() 返回所选项的索引
  - void **setEditable**(boolean aFlag) 设置文本框是否可编辑
  - void **setModel**(ComboBoxModel aModel) 设置数据模型
  - void **addItemListener**(ItemListener l) 注册选项事件监听器
- **ItemEvent事件**
  - 选择某个选项时将触发选项事件**ItemEvent**
  - 对应的事件监听器接口**ItemListener**
    - 接口的方法**itemStateChanged(ItemEvent e)**

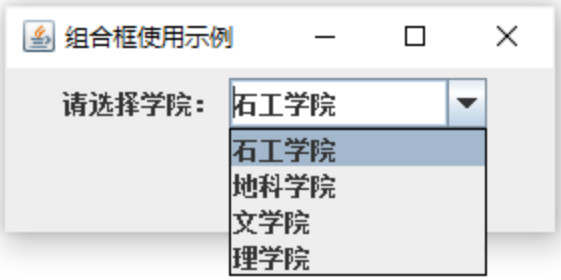
Swing原子组件：列表类之JComboBox

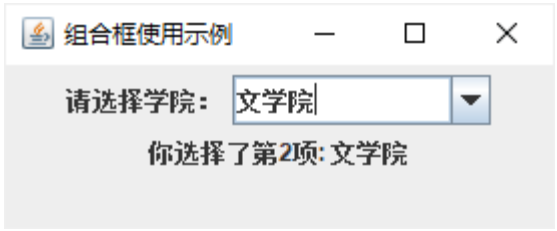
【例8-22】组合框使用示例

```
import java.awt.*;  import java.awt.event.*;  import javax.swing.*;
public class JComboBoxTester extends JFrame   implements ItemListener {
   JLabel lbl2;
   JComboBox<String> cbox;
   public JComboBoxTester(){
         super("组合框使用示例");
         this.setSize(200,120);
         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         this.setLayout(new FlowLayout()); //流式布局
         JLabel lbl1=new JLabel("请选择学院：",JLabel.RIGHT);
         lbl2=new JLabel("",JLabel.LEFT);
         String[] schools={"石工学院","地科学院","文学院","理学院"};
         cbox=new JComboBox<String>(schools);
         //cbox.setEditable(true); // 设置其中的文本框可编辑
         cbox.addItemListener(this); //注册选项事件监听器
         this.add(lbl1);  this.add(cbox);this.add(lbl2);
   }
```

Swing原子组件：列表类之JComboBox

```
    public void itemStateChanged(ItemEvent e){
            if(e.getStateChange()==ItemEvent.SELECTED) {
                System.out.println("call itemStateChanged");
                lbl2.setText("你选择了第"+cbox.getSelectedIndex()
                                        +"项: "+cbox.getSelectedItem());
            }
    }
    public static void main(String[] args) {
            JComboBoxTester frm=new JComboBoxTester();
            frm.setVisible(true);
    }
}
```

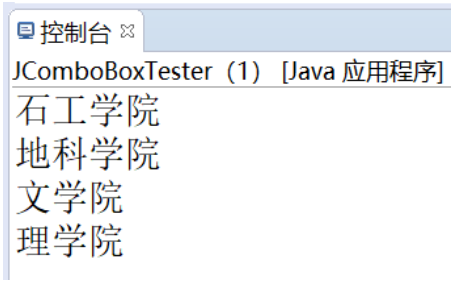




Swing原子组件：列表类之JComboBox

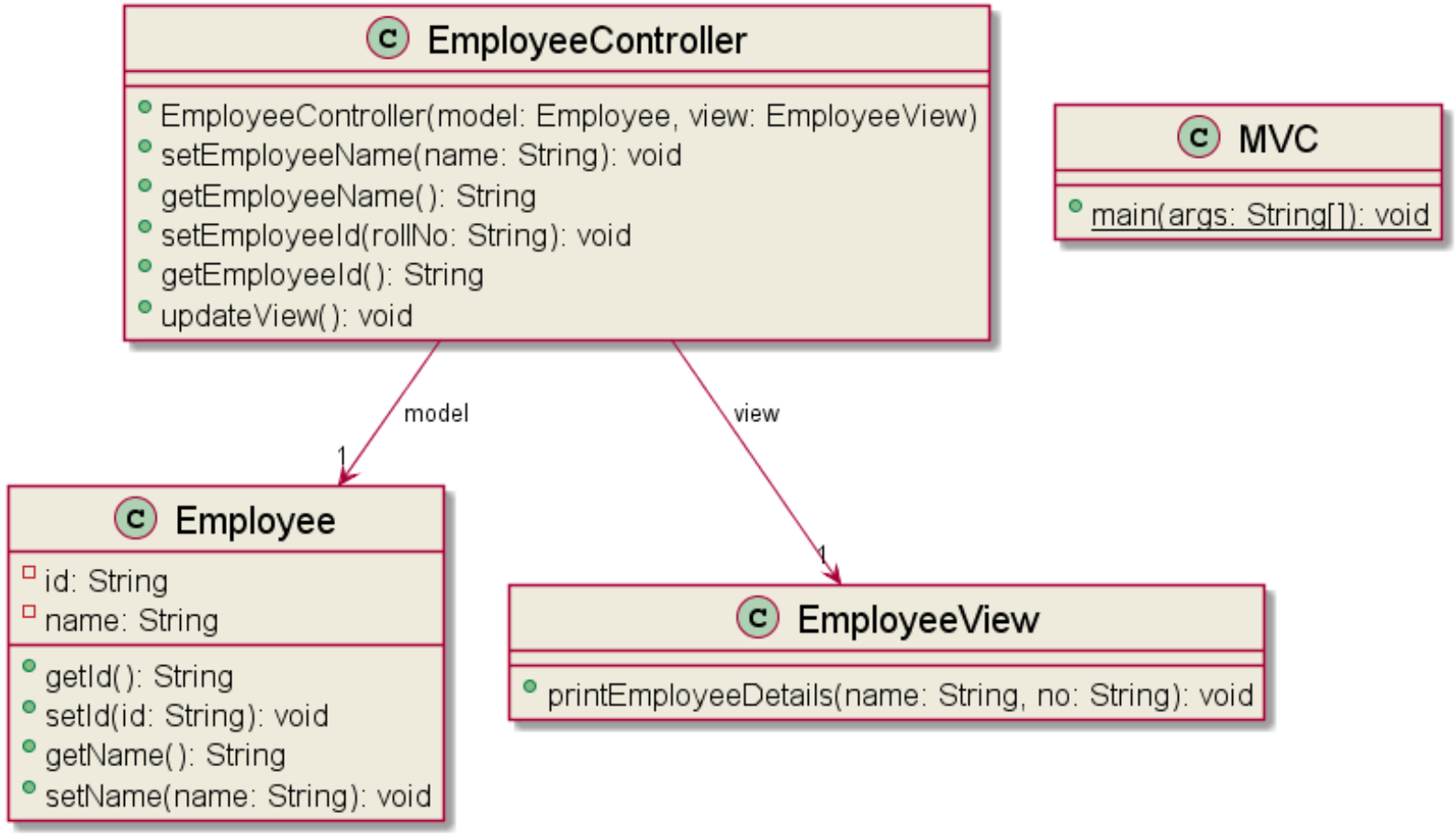
- 组合框与一个数据模型关联，数据模型(Model)与视图(View)是分开设计的，模型改变则界面视图也随之变化，这种模式称为模型-视图-控制器(MVC)模式。
  - **Model**（模型）：表示携带数据的对象。它也可以具有逻辑来更新控制器，如果其数据改变。
  - **View**（视图）：表示模型数据的可视化。通常它有UI逻辑。
  - **Controller**（控制器）：引用模型和视图。它控制数据流进入模型对象，并在数据更改时更新视图。它保持视图和模型分开。
- 上例中，可以获取组合框的数据模型并输出其中的数据：

```
// 获取组合框的数据模
ComboBoxModel<String> dm=cbox.getModel();
for(int i=0;i<dm.getSize();i++) // 遍历数据模型
   System.out.println(dm.getElementAt(i)); 型
```



Swing原子组件：列表类之JComboBox

- 采用MVC模式设计类的简单示例



Swing原子组件：列表类之JComboBox

- 采用MVC模式设计类的简单示例

```
class Employee {// 模型类--用于存储员工数据
    private String id;
    private String name;
    public String getId() { return id; }
    public void setId(String id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
}

class EmployeeView { // 视图类--用于显示员工数据
    public void printEmployeeDetails(String name, String no) {
        System.out.println("Employee: ");
        System.out.println("Name: " + name);
        System.out.println("ID: " + no);
    }
}
```

Swing原子组件：列表类之JComboBox

- 采用MVC模式设计类的简单示例

```
class EmployeeController {// 控制器类--用于访问与显示员工数据
  private Employee model;// 需要控制的模型
  private EmployeeView view;//需要控制的视图
  public EmployeeController(Employee model, EmployeeView view) {
        this.model = model;
        this.view = view;
  }
  public void setEmployeeName(String name) { model.setName(name); } //修改模型
  public String getEmployeeName() { return model.getName(); } //从模型获取数据
  public void setEmployeeId(String rollNo) { model.setId(rollNo); } //修改模型
  public String getEmployeeId() { return model.getId(); } //从模型获取数据
  public void updateView() {//更新视图
        view.printEmployeeDetails(model.getName(), model.getId());
  }
}
```

Swing原子组件：列表类之JComboBox

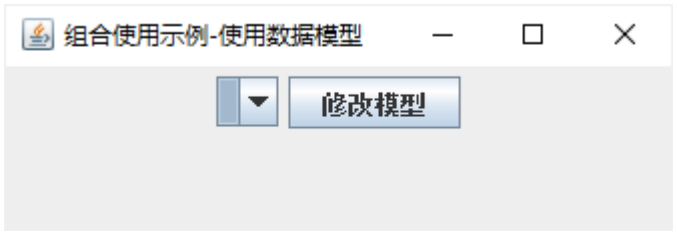
- 采用MVC模式设计类的简单示例

```
public class MVC {
    public static void main(String[] args) {
        Employee model = new Employee(); // 创建模型对象
        model.setName("Tom");
        model.setId("1");
        EmployeeView view = new EmployeeView();//创建视图对象
        // 创建控制器
       EmployeeController controller = new EmployeeController(model, view);
       controller.updateView(); //控制器更新视图
       controller.setEmployeeName("New Name");//控制器修改模型对象
       controller.updateView();//控制器更新视图
    }
}
```

Swing原子组件：列表类之JComboBox

- 给组合框设置数据模型，并改变模型中的数据，视图也随之更新

```
public class JComboBoxTester3 extends JFrame {
    public JComboBoxTester3(){
        ……
        DefaultComboBoxModel<String> dm=new
    DefaultComboBoxModel<String>(); // 创建数据模型
        JComboBox<String> cbox=new JComboBox<String>();
        cbox.setModel(dm); //为组合框关联一个数据模型
        JButton btn=new JButton("修改数据模型");
        btn.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                dm.addElement("新能源学院");
                dm.addElement("海空学院");
                dm.addElement("青岛软件学院");
            }
        });
      this.add(cbox);          this.add(btn);
    }
   ……
}
```



Swing原子组件：列表类之JList

- 用户可以从列表框中选择一项或多项
- 可以设置选择模式（一次可选几项及是否可连续）
- JList不支持自动滚动，当其中有很多选项需要滚动显示时，需要把它放在一个JScrollPane对象里面
- 常用构造方法
  - **JList<E>**() 创建一个空列表框
  - **JList<E>**(E[] listData) 创建包含指定数组中的元素的列表框
  - **JList<E>** (Vector<E> listData) 用指定向量初始化列表框
- 常用方法
  - void **setSelectionMode**(int selectionMode) 设置选择模式
    - 一次只能选择一列表项：
      ListSelectionModel.SINGLE_SELECTION
    - 一次可选择多个连续列表项：
      ListSelectionModel.SINGLE_INTERVAL_SELECTION
    - （默认）任意多选：
      ListSelectionModel.MULTIPLE_INTERVAL_SELECTION

Swing原子组件：列表类之JList

- void **setListData**(Object[] listData) 设置列表数据
- void **setListData**(Vector<E> listData) 设置列表数据
- void **setVisibleRowCount**(int visibleRowCount) 设置在列表中显示的行数
- Object **getSelectedValue**() 返回第一个选择项
- Object[] **getSelectedValues**() 返回所有选择项
- int **getSelectedIndex**() 返回第一个选择项的索引
- int[] **getSelectedIndices**() 返回所有选择项的索引
- List<E> **getSelectedValuesList**() 以**List**返回所有选择项
- void **clearSelection**() 清除选择
- void **addListSelectionListener**(ListSelectionListener l) 注册列表选择事件监听器

Swing原子组件：列表类之JList

- ListSelectionEvent事件
  - 用户选择列表选项时将触发事件ListSelectEvent
  - 对应的事件监听器接口ListSelectionListener
    - 接口的方法valueChanged(ListSelectionEvent e)

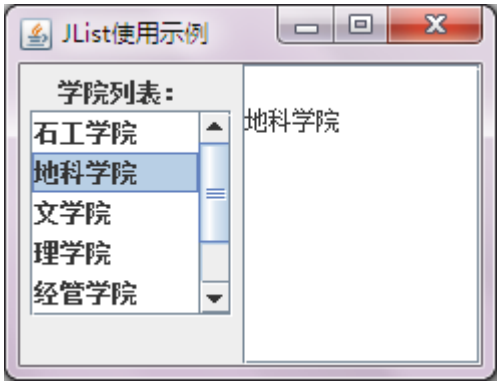
【例8-23】JList使用示例

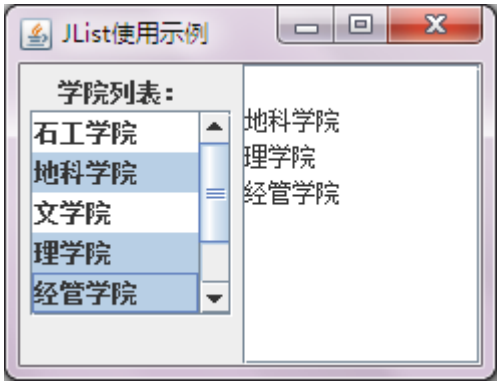
```
import javax.swing.*;
import javax.swing.event.*;
import java.util.*;
public class JListTester extends JFrame
                                implements ListSelectionListener{
   JList<String> list;
   JTextArea ta;
   public JListTester(){
         super("JList使用示例");
         this.setSize(300,200);
         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

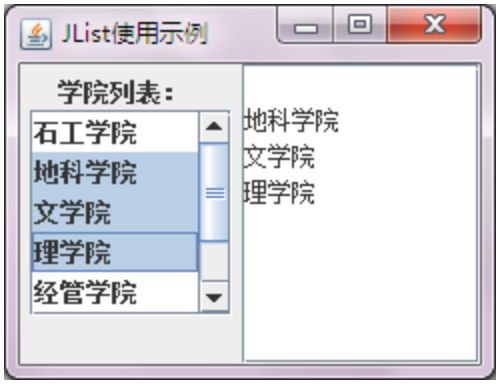
Swing原子组件：列表类之JList

```
JLabel lbl=new JLabel("学院列表：",JLabel.LEFT);
String[] schools={"石工学院","地科学院","文学院","理学院",
                  "经管学院","信控学院","机电学院"};
list=new JList<String>(schools);
list.setVisibleRowCount(5);
list.addListSelectionListener(this); //注册列表选择事件监听器
JScrollPane listSP=new JScrollPane(list);

Box vBox=Box.createVerticalBox();
vBox.add(lbl);vBox.add(listSP);
JPanel panel=new JPanel();
panel.add(vBox);

ta=new JTextArea(5,10);
JScrollPane sp=new JScrollPane(ta);
this.add(panel,"West");    this.add(sp,"Center");
}
```

```
    public void valueChanged(ListSelectionEvent e){
        // 用鼠标选择，则按下触发一次列表选择事件，松开又触发一次
        // 若用键盘方向键选择，则仅仅触发一次列表选择事件
        if(e.getValueIsAdjusting()){
            System.out.println("call valueChanged");
            ta.setText(""); //清空文本区域
            List<String> items=list.getSelectedValuesList();
            for(String str:items)     ta.append("\n"+str);
        }
    }
    public static void main(String[] args) {
            JListTester frm=new JListTester();
            frm.setVisible(true);
    }
}
```

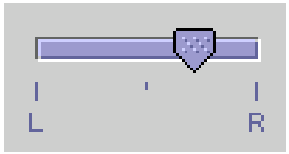






Swing原子组件：连续数值选择

- **Jslider**（滑块）

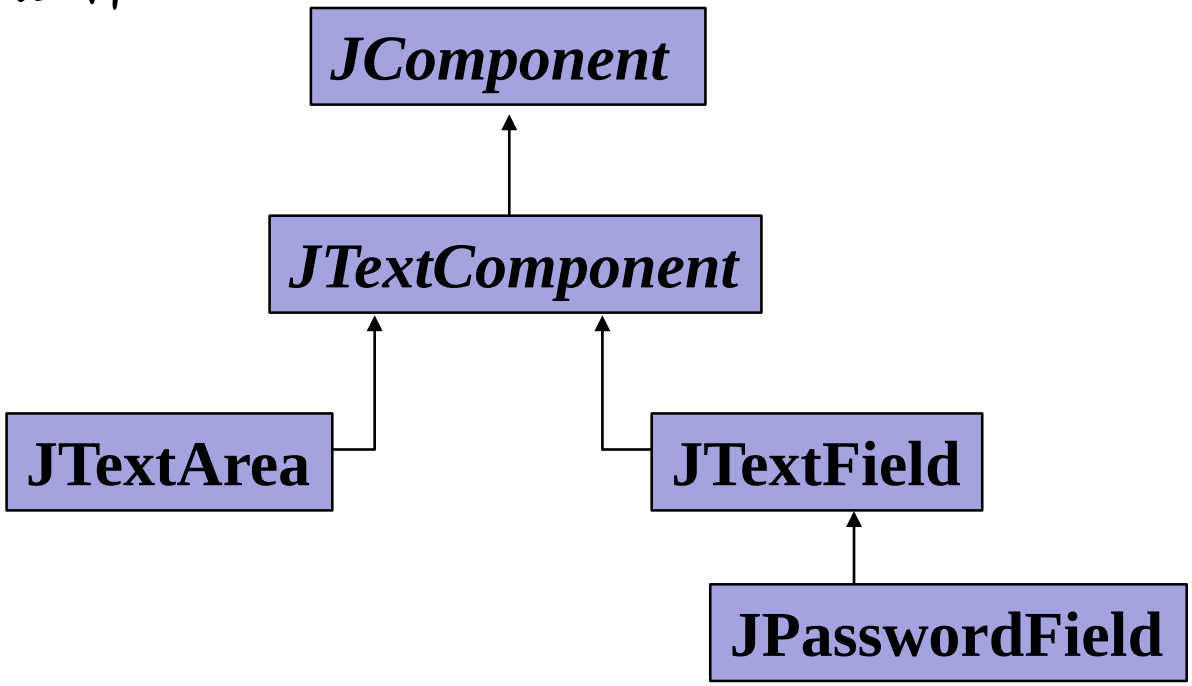


  - 空间大
  - 好像电器用品上机械式的控制杆，可以设置它的最小、最大、初始刻度，还可以设置它的方向，还可以为其标上刻度或文本
  - 在JSlider上移动滑动杆，会产生ChangeEvent事件
- **Jspinner**（微调控制器/数字调节器）

  - 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
  - 改变JSpinner上的值，会产生ChangeEvent事件

Swing原子组件：文本编辑类

- JTextComponent类是文本组件的抽象父类，其主要派生类有
  - 单行文本框JTextField
  - 密码框JPasswordField，用屏蔽字符表示文本
  - 文本区域JTextArea，要滚动显示其中类容往往要用到ScrollPane组件



Swing原子组件：文本编辑类

- JTextField常用构造方法
  - JTextField() 构造一个内容为null的文本框
  - JTextField(String text) 构造一个指定内容的文本框
  - JTextField(int columns) 构造一个内容为null的但指定列宽的文本框
- JPasswordField常用构造方法与JTextField类似
- JTextArea常用构造方法
  - JTextArea () 构造一个内容为null的文本区域
  - JTextArea (String text) 构造一个指定内容的文本区域
  - JTextArea(int rows, int columns) 构造一个内容为null的但指定行数和列数的文本区域
  - JTextArea(String text, int rows, int columns) 构造一个具有指定文本、行数和列数的文本区域

Swing原子组件：文本编辑类

- **JTextComponent常用方法**
  - void **setEditable**(boolean b) 设置是否可编辑
  - String **getText**() 获取文本
  - void **setText**(String t) 设置文本
  - String **getSelectedText**() 获取选定文本
  - int **getSelectionStart**() 获取选定文本的起始位置
  - int **getSelectionEnd**() 获取选定文本的结束位置
  - void **setSelectionStart**(int selectionStart) 将选定起始点设置为指定的位置
  - void **setSelectionEnd**(int selectionEnd)将选定结束点设置为指定的位置
  - void **select**(int selectionStart, int selectionEnd) 选定指定的开始和结束位置之间的文本
  - void **selectAll**() 选定所有文本
  - void **copy**() 将所选文本复制到系统剪贴板
  - void **paste**()将系统剪贴板的内容粘贴到文本组件
  - void **cut**()当前选定的文本粘贴到系统剪贴板，并从文本组件中移除该文本

Swing原子组件：文本编辑类

- **JPasswordField特有的常用方法**
  - void **setEchoChar**(char c) 设置回显字符
  - char **getEchoChar**() 获取回显字符
  - char[] **getPassword**() 替代 ~~String **getText**()~~
- **JTextArea特有的常用方法**
  - void **append**(String str) 添加指定文本到文本区域末尾
  - void **insert**(String str, int pos) 将指定文本插入到的文本区域内指定位置处
  - void **replaceRange**(String str, int start, int end) 用给定的新文本替换从指示的起始位置到结尾位置的文本
  - void **setLineWrap**(boolean wrap) 设置文本区的换行策略。如果设置为 **true**，则当行的长度大于所分配的宽度时，将换行。如果设置为 **false**，则始终不换行。此属性默认为 **false**

Swing原子组件：文本编辑类

- 文本组件的事件
  - 文本组件采用分离模型结构：组件和与组件相关的数据模型，其中数据模型保存了组件状态或数据
  - 文本组件的数据模型是Document接口，负责维护文本组件的状态数据
  - 对文本组件的内容进行增删改等操作时，组件的文档（即数据模型）会引发DocumentEvent事件，对应监听器接口为DocumentListener，其中定义了3个抽象方法
    - void **changedUpdate**(DocumentEvent e) 当Document里的属性或属性集发生改变时触发该方法
    - void **insertUpdate**(DocumentEvent e) 当向Document里插入文本时触发该方法
    - void **removeUpdate**(DocumentEvent e) 当从Document中删除文本时触发该方法
  - 文本组件获取文档的方法是getDocument()
  - 此外，在文本框和密码框中按回车键还可触发ActionEvent事件

Swing原子组件：文本编辑类

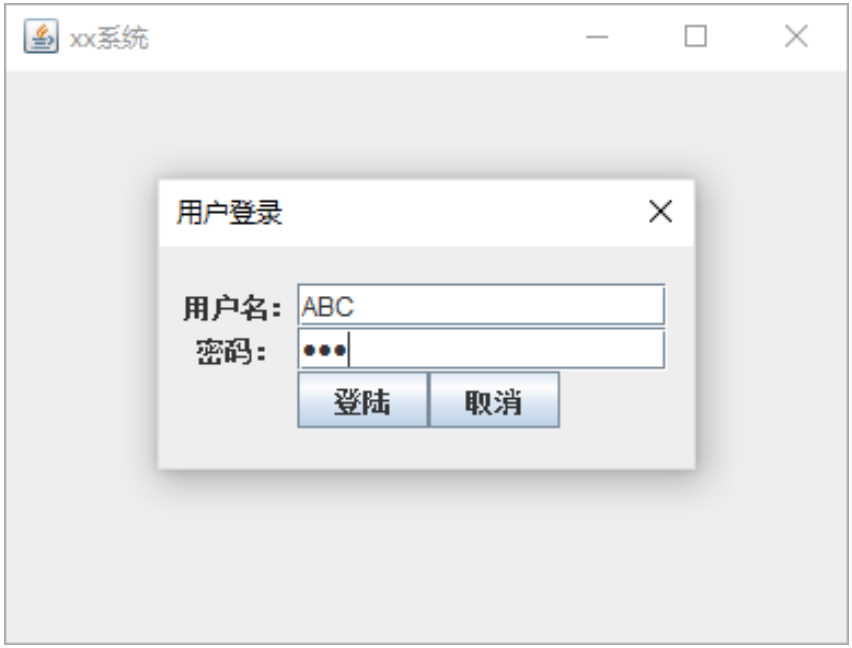
【例8-24】文本框和密码框示例：用户登录界面

① 定义两个窗体类：一个登录对话框和一个主窗体；

② 主窗体打开后，显示登录对话框，账号非法或用户取消则退出应用程序；

③ 定义一个枚举，表示用户登录操作的结果：合法、非法与取消。

```
// UserLoginResult.java
// 用户登录结果取值枚举
public enum UserLoginResult {
    VALID,      //验证合法
    INVALID,    //验证非法
    CANCEL      //用户取消或关闭对话框
}
```



```
// FrmUserLogin.java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class FrmUserLogin  extends JDialog{ // 登录对话框
   UserLoginResult result=UserLoginResult.CANCEL; // 用户登录结果
   public FrmUserLogin(JFrame ower){
          super(ower,"用户登陆");
          this.setSize(260, 140);
          this.setLocationRelativeTo(ower); //窗口置于所依附窗体的中央
          this.setResizable(false);
          this.setDefaultCloseOperation(HIDE_ON_CLOSE);
          GridBagLayout layout=new GridBagLayout(); //网格包布局
          this.setLayout(layout); //给对话框容器设置布局管理器
          JLabel lblUserName=new JLabel("用户名：");
          JLabel lblpwd=new JLabel("密码：");
          JTextField txtUserName=new JTextField(15);
          JTextField txtpwd=new JPasswordField(15);
          JButton btnOK=new JButton("登陆");
          JButton btnCancel=new JButton("取消");
```

```
GridBagConstraints st=new GridBagConstraints();
st.fill=GridBagConstraints.NONE;
st.anchor=GridBagConstraints.CENTER;
add(lblUserName, set(st, 0, 0, 1, 1));
add(txtUserName, set(st, 1, 0, 3, 1));
add(lblpwd, set(st, 0, 1, 1, 1));
add(txtpwd, set(st, 1, 1, 3, 1));
add(btnOK, set(st, 1, 2, 1, 1));
add(btnCancel, set(st, 2, 2, 1, 1));

btnOK.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
          if(txtUserName.getText().trim().equals("ABC") &&
                        txtpwd.getText().equals("123"))
                result=UserLoginResult.VALID;
          else
                result=UserLoginResult.INVALID;
          setVisible(false); //关闭对话框
   }
});
```

```
        btnCancel.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                    result=UserLoginResult.CANCEL;
                    setVisible(false); //关闭对话框
            }
        });

    }

    //修改约束对象的约束条件
    GridBagConstraints set(GridBagConstraints st,
                                                    int x,int y,int w,int h){
        st.gridx=x;
        st.gridy=y;
        st.gridwidth=w;
        st.gridheight=h;
        return st;
    }

}
```

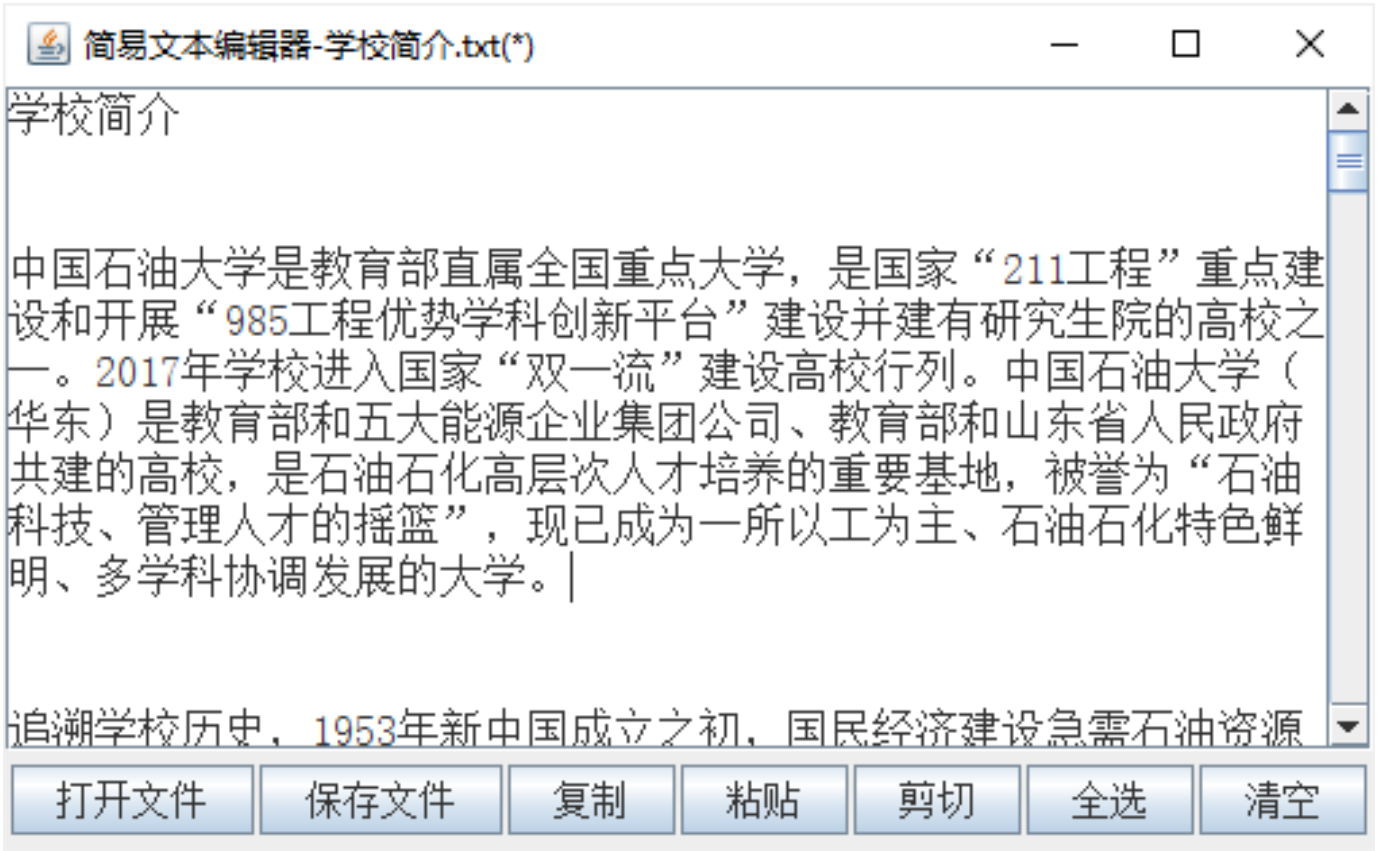
```
// FrmUserMain.java
import java.awt.*;
import javax.swing.*;
public class FrmUserMain extends JFrame{ // 主窗体
   public FrmUserMain(){
         super("xx系统");
         this.setSize(400,300);
         this.setLocationRelativeTo(null); // 窗口置于显示屏的中央
         this.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
   }
   public static void main(String[] args) {
         FrmUserMain frmMain=new FrmUserMain();
         frmMain.setVisible(true); // 显示主窗体
         FrmUserLogin frmLogin=new FrmUserLogin(frmMain);
         frmLogin.setModal(true); //设置为模式对话框
         frmLogin.setVisible(true); // 显示登录对话框
         if(frmLogin.result!=UserLoginResult.VALID){
               if(frmLogin.result==UserLoginResult.INVALID)
                  JOptionPane.showMessageDialog(frmMain, "用户账号错误！",
                                          "提示",JOptionPane.ERROR_MESSAGE);
               System.exit(0); // 退出应用程序
         }
         frmMain.getContentPane().setBackground(Color.GREEN);
   }
}
```

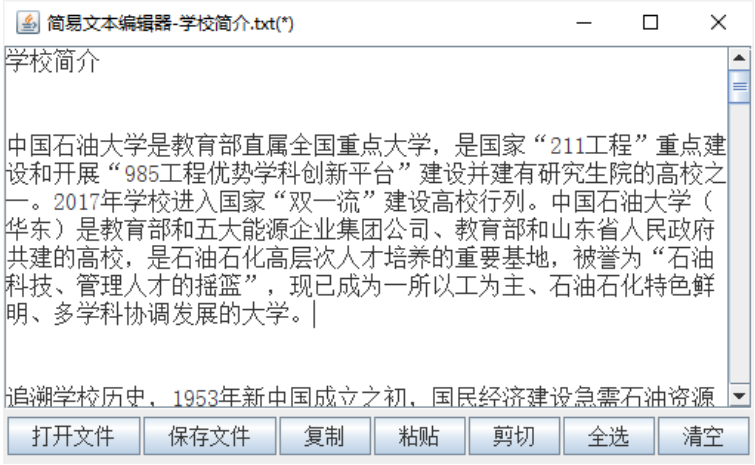
Swing原子组件：文本编辑类

【例8-25】文本区域示例：简易文本编辑器。

① 功能包括：打开文件、保存文件、复制、粘贴、剪切、全选

② 当文本区域中的内容被修改后，在窗体标题中添加修改标志“(*)”






```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import javax.swing.plaf.FontUIResource;
import javax.swing.text.Document;

import java.io.*;
import java.util.Enumeration;
public class JTextAreaTester  extends JFrame
                                    implements ActionListener{
   private JButton btnOpen, btnSave, btnCopy, btnPaste, btnCut,
                                                btnSelectAll, btnClearAll;
   private JTextArea ta;
   private boolean isTextModified; //文本被编辑的标记
   private String file=""; //文件名
   private final static String APP_NAME = "简易文本编辑器";
```

```
public JTextAreaTester(){
    super(APP_NAME );
    this.setSize(500, 300);
    this.setDefaultCloseOperation(EXIT_ON_CLOSE);

    ta=new JTextArea();
     ta.setLineWrap(true); //自动换行
    ta.setFont(new Font("宋体",Font.PLAIN,16));
    JScrollPane sp=new JScrollPane(ta); // 根据需要自动显示滚动条

     btn1=new JButton("打开文件");   btn1.setActionCommand("打开文件");
     btn2=new JButton("保存文件");   btn2.setActionCommand("保存文件");
     btnCopy=new JButton("复制");    btnCopy.setActionCommand("复制");
     btnPaste=new JButton("粘贴");  btnPaste.setActionCommand("粘贴");
     btnCut=new JButton("剪切");     btnCut.setActionCommand("剪切");
     btn3=new JButton("全选");       btn3.setActionCommand("全选");
     btn4=new JButton("清空");       btn4.setActionCommand("清空");

     // 为组件注册动作事件监听器
     btnOpen.addActionListener(this);
    btnSave.addActionListener(this);
    btnPaste.addActionListener(this);
    btnSelectAll.addActionListener(this);
```

```
    JPanel panel=new JPanel(); // 按钮放置在一个面板内
    FlowLayout layout=(FlowLayout) panel.getLayout();
    layout.setHgap(2);
    panel.add(btn1);            panel.add(btn2);
    panel.add(btnCopy);     panel.add(btnPaste);
     panel.add(btnCut);       panel.add(btn3);
     panel.add(btn4);
    this.add(sp, "Center");   this.add(panel, "South");

    // 给文本框的文档添加文档监听器
    Document doc=ta.getDocument();
    doc.addDocumentListener(new DocumentListener() {
            public void changedUpdate(DocumentEvent e) { }
            public void insertUpdate(DocumentEvent e) {
                  setTitle(APP_NAME+"-"+file+"(*)"); //设置修改标记
                isTextModified=true; // 设置修改标记
            }
            public void removeUpdate(DocumentEvent e) {
                setTitle(APP_NAME+"-"+file+"(*)"); //设置修改标记
                isTextModified=true; // 设置修改标记
            }
    });
}
```

```
public void actionPerformed(ActionEvent e){
        String cmd=e.getActionCommand();
        if(cmd.equals("打开文件")) { // 打开文本文件
            StringBuffer sb=new StringBuffer("");
            try {BufferedReader br=new BufferedReader(
                                                    new FileReader("学校简介.txt"));
                  String str=br.readLine();
                  while(str!=null){
                      sb.append(str+"\n");   str=br.readLine();
                  }
                  br.close();
                  ta.setText(sb.toString());
                  this.setTitle(APP_NAME+"-"+file); //窗体标题初始化
                  isTextModified=false; //重置修改标记
            } catch (Exception e1) { ta.setText(e1.getMessage()); }
        }else if(cmd.equals("保存文件")){ // 保存文本
           // TODO: 保存文件的代码
           this.setTitle(APP_NAME+"-"+file); //窗体标题重置
           isTextModified=false; // 保存后修改
        }else if(cmd.equals("复制")){ ta.paste();  }
}
```

```
    // 初始化全局字体
    private static void initGlobalFont(){
        FontUIResource fontUIResource =
                    new FontUIResource(new Font("宋体",Font.PLAIN, 14));
        for (Enumeration<Object> keys = UIManager.getDefaults().keys();
                                                    keys.hasMoreElements();) {
            Object key = keys.nextElement();
            Object value= UIManager.get(key);
            if (value instanceof FontUIResource) {
                UIManager.put(key, fontUIResource);
            }
        }
    }
    public static void main(String[] args) {
            initGlobalFont(); // 初始化全局所有字体
            JTextAreaTester frm=new JTextAreaTester();
            frm.setVisible(true);
    }
}
```

谢谢大家!