



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



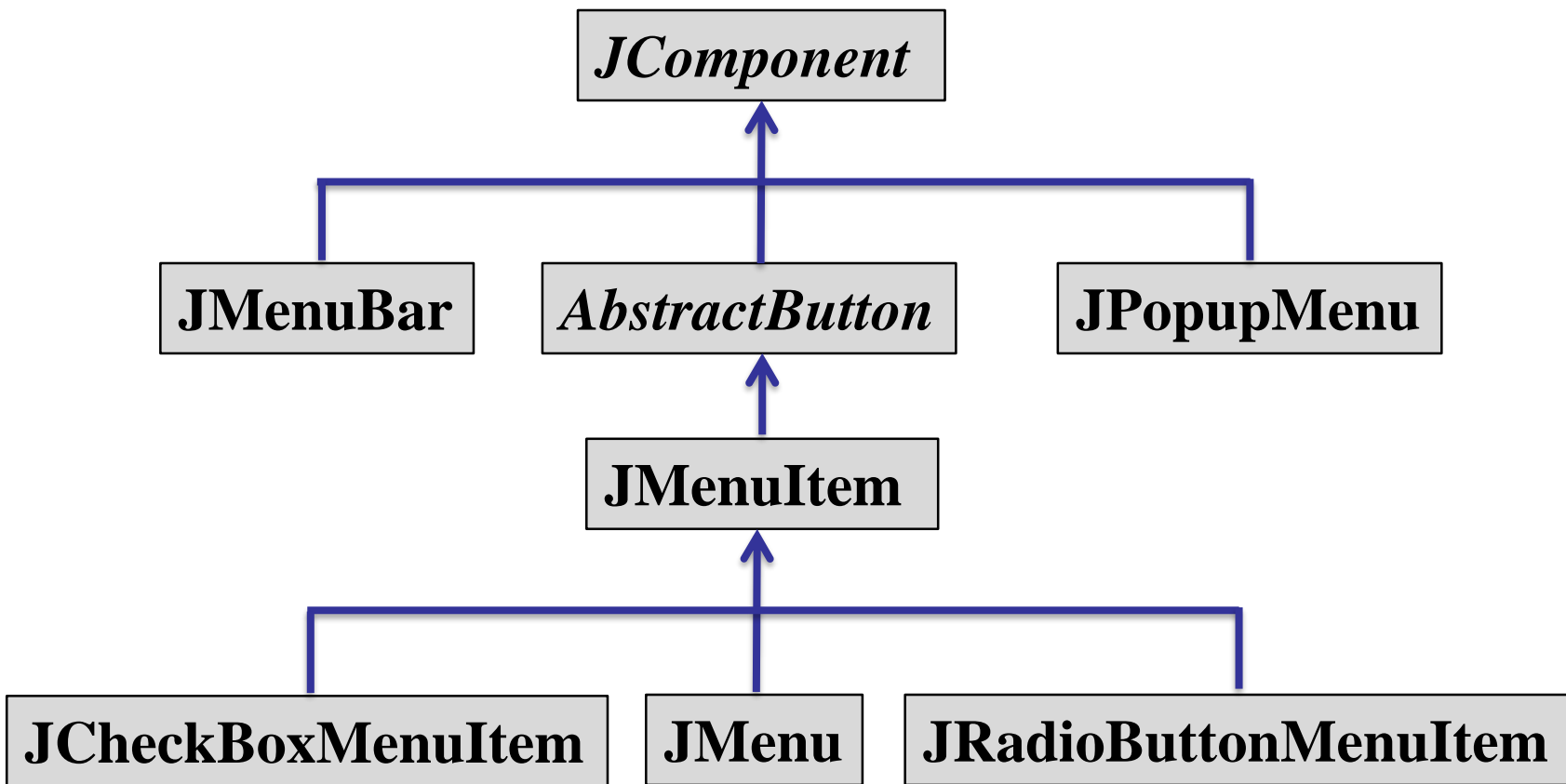
Swing菜单

- 分类
 - 下拉式菜单 (**Menu**)
 - 弹出式菜单 (**PopupMenu**) ,也叫快捷菜单
- 下拉式菜单的三个基本要素
 - 菜单栏 (**JMenuBar**) : 可以看做一个菜单容器, 在其中可以放置任意数量的菜单, 即菜单栏由若干菜单组成
 - 菜单 (**JMenu**) : 可以看做一个菜单项和子菜单的容器, 即每个菜单由若干菜单项和子菜单组成
 - 菜单项 (**JMenuItem**) : 菜单最小单位, 一个菜单项对应一个特定命令, 单击它时, 执行特定的功能。可以看做是一个命令按钮或选项按钮
- 一个弹出式菜单 (快捷菜单) 仅包括菜单项和子菜单, 可以与下拉式菜单中的某个菜单关联



Swing菜单

- 相关菜单类的层次关系





Swing菜单：JMenuBar

- 用于实现菜单栏，由若干菜单构成
- 构造方法
 - **JMenuBar()** 创建一个空的菜单栏
- 常用方法
 - **JMenu add(JMenu c)** 将指定的菜单追加到菜单栏的末尾
 - **int getMenuCount()** 返回菜单栏上的菜单数
 - **JMenu getMenu(int index)** 返回菜单栏中指定位置的菜单



Swing菜单：JMenu

- 用于实现菜单，由若干菜单项和子菜单构成
- 既可作为顶层菜单添加到菜单栏，也可以作为子菜单添加到其他菜单中
- 常用构造方法
 - **JMenu()** 构造一个没有文本的菜单
 - **JMenu(String s)** 构造一个指定文本的菜单
- 常用方法
 - JMenuItem **add**(JMenuItem menuItem) 将某个菜单项追加到此菜单的末尾
 - JMenuItem **add**(String s) 创建具有指定文本的新菜单项，并将其追加到此菜单的末尾
 - Component **add**(Component c) 将某个组件（例如子菜单）追加到此菜单的末尾



Swing菜单：JMenu

- void **addSeparator()** 将新分隔符追加到菜单的末尾
- void **insertSeparator(int index)** 在指定位置插入分隔符
- JMenuItem **insert(JMenuItem mi, int pos)** 在给定位置插入指定的 JMenuItem
- void **insert(String s, int pos)** 在给定位置插入具有指定文本的新菜单项
- void **setMnemonic(int mnemonic)** 设置菜单的键盘助记符（快捷键）。助记符对应键盘上的某个键x（不区分大小写），Alt+x组合时将激活此按钮或菜单。应使用 java.awt.event.KeyEvent 中定义的 VK_XXX 键代码之一指定助记符。
- void **setMnemonic(char mnemonic)** 设置键盘助记符
- void **remove(JMenuItem item)** 移除指定菜单项
- void **remove(int pos)** 移除指定索引处的菜单项
- void **remove(Component c)** 移除组件 c
- void **removeAll()** 移除所有菜单项



Swing菜单: JMenuItem

- 组成菜单的最小单位, 不可再分解
- 一个菜单项对应一个功能命令
- 常用构造方法
 - **JMenuItem()** 创建一个没有文本和图标的菜单项
 - **JMenuItem(String text)** 创建一个指定文本的菜单项
 - **JMenuItem(String text, Icon icon)** 创建一个带有指定文本和图标的菜单项
 - **JMenuItem(String text, int mnemonic)** 创建一个带有指定文本和键盘助记符的 菜单项
- 常用方法
 - void **setIcon(Icon defaultIcon)** 设置图标
 - void **setEnabled(boolean b)** 启用或禁用菜单项
 - void **setMnemonic(int mnemonic)** 设置菜单项的键盘助记符 (快捷键)



Swing菜单: JMenuItem

- **void setAccelerator(KeyStroke keyStroke)** 设置菜单项的加速键。KeyStroke表示键盘上按键操作的类，使用其静态方法getKeyStroke()获得响应的加速键
 - **static KeyStroke getKeyStroke(char keyChar)** 用指定字符作为加速键
 - **static KeyStroke getKeyStroke(int keyCode, int modifiers)** 设置组合键作为加速键
 - **keyCode**的值由KeyEvent类定义的虚拟键常量来指定：
VK_A、**VK_1**、**VK_ENTER**、**VK_F1**、**VK_SPACE**分别表示字母A键、数字1键、回车键、F1键、空格键等
 - **modifiers**指定了修饰键：**0**（无修饰键）、
InputEvent.SHIFT_DOWN_MASK /
CTRL_DOWN_MASK / **ALT_DOWN_MASK**



Swing菜单: JMenuItem

- **ActionEvent事件或ItemEvent事件**
 - 单击菜单项将引发动作事件**ActionEvent**, 对应监听接口为**ActionListener**, 触发其**actionPerformed**方法
 - 对于**JRadioButtonItem**和**JCheckBoxItem**菜单项, 当选项状态改变时, 还会引发**ItemEvent**事件, 对应的监听器接口为**ItemListener**, 触发其**itemStateChanged**方法
- 单选按钮菜单项 **JRadioButtonItem** 和 复选框菜单项 **JCheckBoxItem** 的用法与 **JMenuItem** 的相似
 - 单选按钮菜单项与单选按钮**JRadioButton**一样, 需要在逻辑上放入一个按钮组中, 一次只能有一个菜单项被选中
 - 复选框按钮菜单项与复选框**JCheckBox**一样, 用户选择该菜单时, 该菜单会在选中与未选中之间进行切换



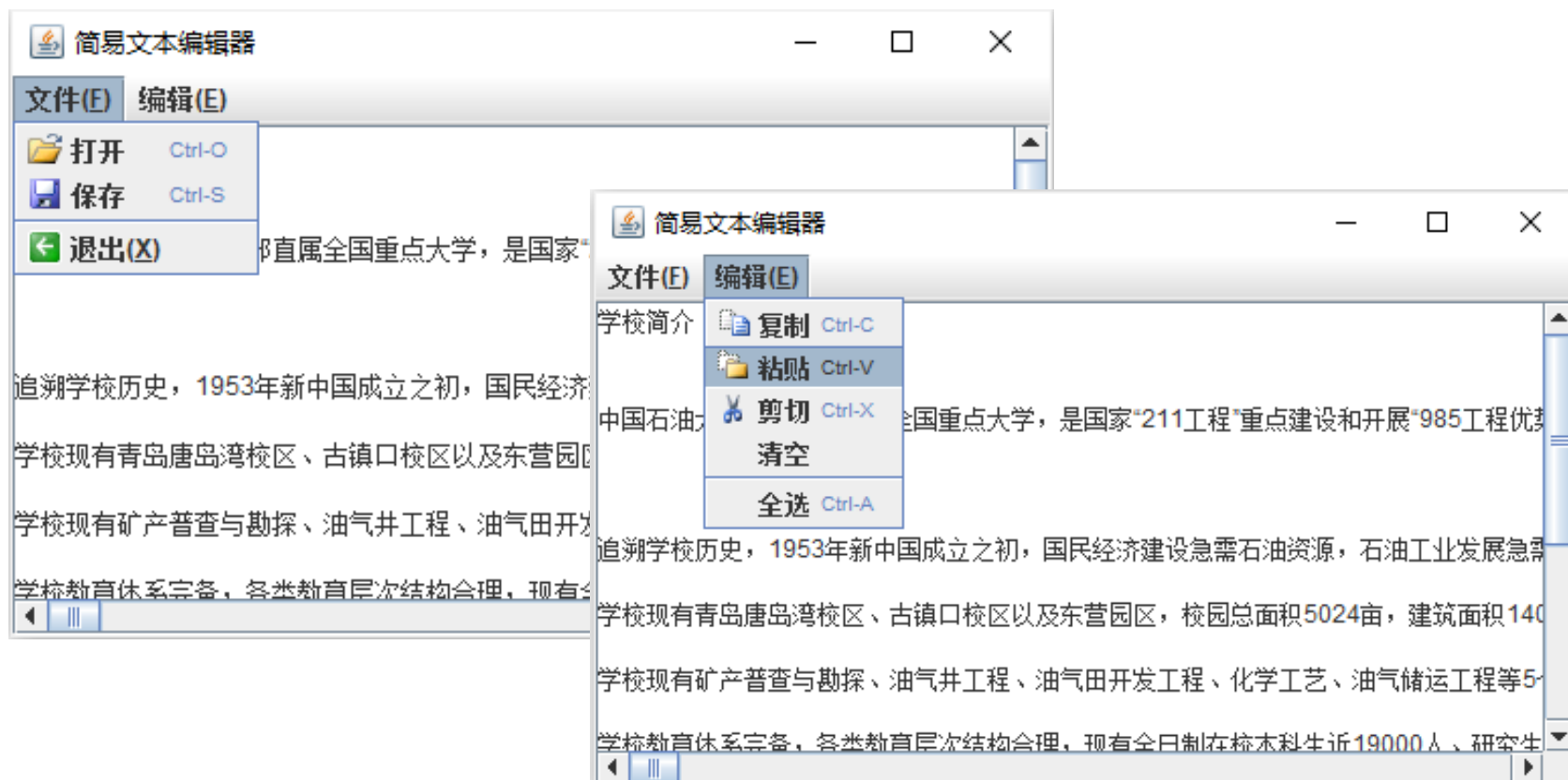
Swing菜单：下拉式菜单的设计步骤

- 创建菜单栏JMenuBar
- 创建多个菜单JMenu,并加入菜单栏中
- 为每个菜单创建其所包含的菜单项JMenuItem或子菜单，并加入该菜单中
- 为每个菜单项添加事件处理程序
- 调用 JFrame 、 JDialog 、 JApplet 等顶层容器的 setMenuBar方法，将菜单栏添加到其上



Swing菜单：下拉式菜单的设计步骤

【例8-25】 将例8-24中的简易文本编辑器的命令按钮改用菜单实现：



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

public class JMenuTester extends JFrame
    implements ActionListener{

    private JMenuBar mb;
    private JMenu fileMenu, editMenu;
    private JMenuItem miOpen, miSave, miExit, miCopy, miPaste, miCut,
        miSelectAll, miClearAll;
    private JTextArea ta;
    public JMenuTester(){
        super("简易文本编辑器");
        this.setSize(500, 300);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        ta=new JTextArea();
        JScrollPane sp=new JScrollPane(ta); // 根据需要自动显示滚动条
        this.add(sp);
    }
}
```

```
mb=new JMenuBar(); // 菜单栏
this.setJMenuBar(mb); //为窗体设置菜单栏
//菜单
fileMenu=new JMenu("文件(F)"); fileMenu.setMnemonic('F');
editMenu=new JMenu("编辑(E)"); editMenu.setMnemonic('E');
//菜单添加到菜单栏
mb.add(fileMenu);
mb.add(editMenu);
// 菜单项
miOpen=new JMenuItem("打开",new ImageIcon("img/open.png"));
miOpen.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O,
                                                InputEvent.CTRL_DOWN_MASK));
miOpen.setActionCommand("打开");
miOpen.addActionListener(this); //为菜单项注册事件监听器

miSave=new JMenuItem("保存",new ImageIcon("img/save.png"));
miSave.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,
                                                InputEvent.CTRL_DOWN_MASK));
miSave.setActionCommand("保存");
miSave.addActionListener(this);
```

.....

//菜单项添加到菜单

fileMenu.add(miOpen);

fileMenu.add(miSave);

fileMenu.addSeparator(); **// 分割线**

fileMenu.add(miExit);

editMenu.add(miCopy);

editMenu.add(miPaste);

editMenu.add(miCut);

editMenu.add(miClearAll);

editMenu.addSeparator(); **// 分割线**

editMenu.add(miSelectAll);

}

public void actionPerformed(ActionEvent e) {

String cmd=e.getActionCommand();

if(cmd.equals("打开")){ **//...**

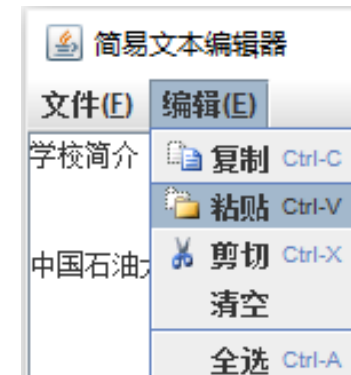
} else if (cmd.equals("保存")) { **//...**

}

}

public static void main(String[] args) { **//...**

}





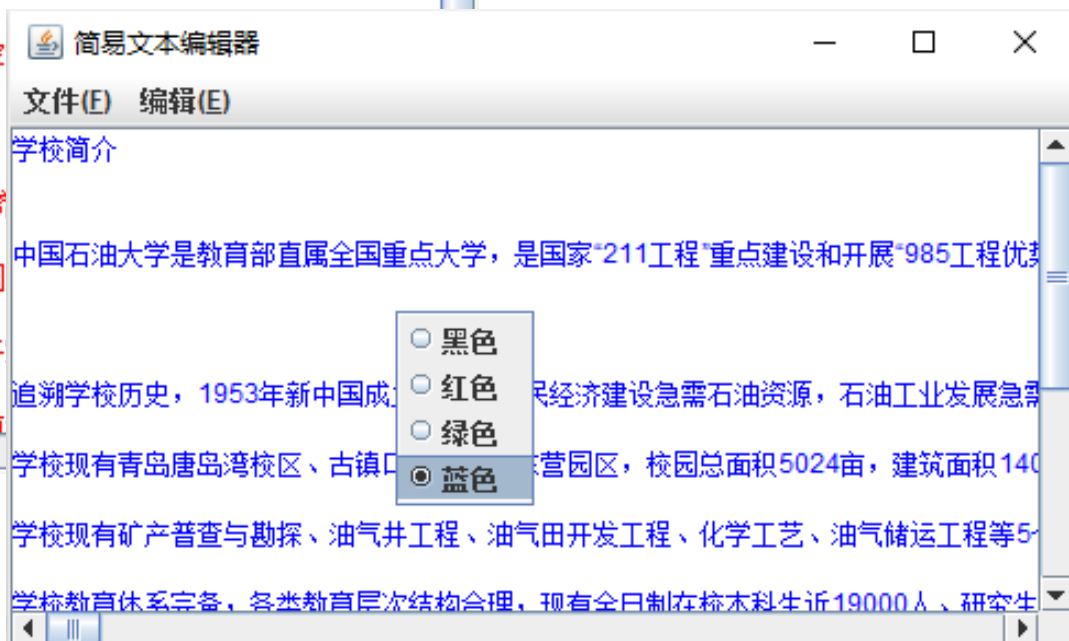
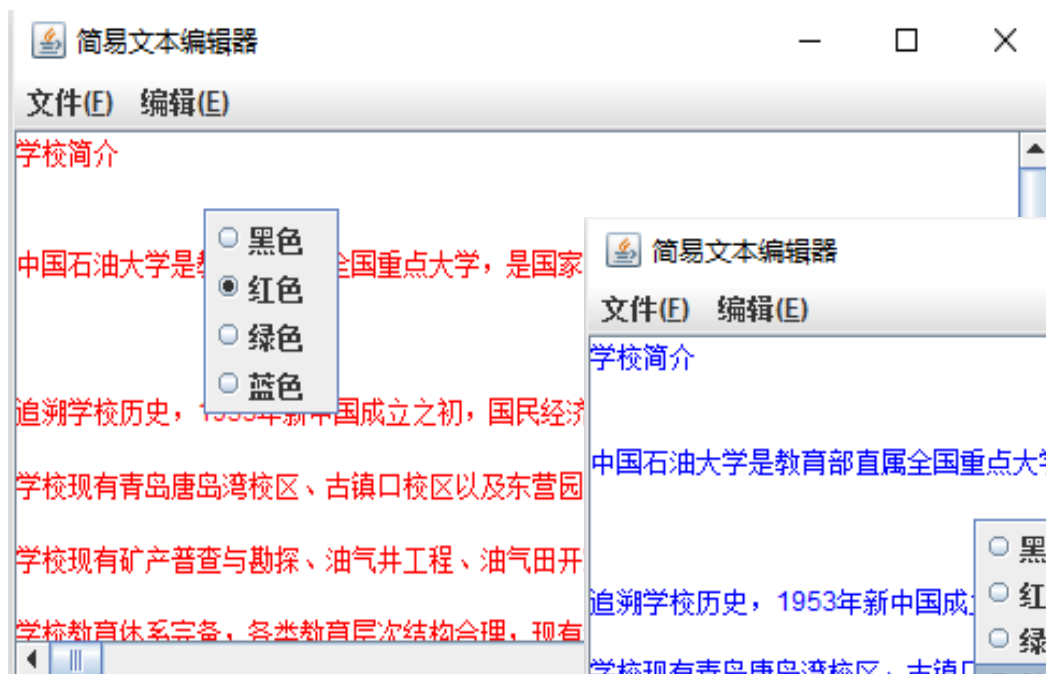
Swing菜单：弹出式菜单

- 利用JPopupMenu组件创建弹出式菜单
- JPopupMenu的常用构造方法
 - JPopupMenu()
- JPopupMenu的常用方法
 - 添加菜单项和分割线的方法，与JMenu相同
 - void **show**(Component invoker, int x, int y) 在组件调用者的坐标空间中的位置 (x, y)显示弹出菜单
 - 该方法一般在鼠标右键释放时调用，即在鼠标事件MouseEvent的MouseReleased方法中调用
 - 更简便的方法是，若需要显示弹出式菜单的组件为c，则调用c.**setComponentPopupMenu**(popup)方法即可



Swing菜单：弹出式菜单

【例8-26】 为例8-25中的简易文本编辑器的文本区域添加上弹出式菜单




```
...
public class JMenuTester extends JFrame implements ActionListener{
    ...
    private JPopupMenu pm;
    private JRadioButtonMenuItem miBlack,miRed,miGreen,miBlue;
    public JMenuTester(){
        .....
        //弹出式菜单
        pm=new JPopupMenu();
        ta.setComponentPopupMenu(pm); //为文本区域设置弹出式菜单
        // 菜单项
        miBlack=new JRadioButtonMenuItem("黑色");
        miRed=new JRadioButtonMenuItem("红色");
        miGreen=new JRadioButtonMenuItem("绿色");
        miBlue=new JRadioButtonMenuItem("蓝色");
        //单选按钮组
        ButtonGroup bg=new ButtonGroup();
        bg.add(miBlack); bg.add(miRed);
        bg.add(miGreen); bg.add(miBlue);
        //将菜单项添加到弹出式菜单
        pm.add(miBlack); pm.add(miRed);
        pm.add(miGreen); pm.add(miBlue);
    }
}
```

单选按钮菜单
项与单选按钮
一样，也需要
加入到按钮组

```
miBlack.addItemListener(new ItemListener() {  
    public void itemStateChanged(ItemEvent e) {  
        if (e.getStateChange() == ItemEvent.SELECTED)  
            ta.setForeground(Color.BLACK);  
    }  
});
```

```
miRed.addItemListener(new ItemListener() {  
    public void itemStateChanged(ItemEvent e) {  
        if (e.getStateChange() == ItemEvent.SELECTED)  
            ta.setForeground(Color.RED);  
    }  
});  
miGreen.addItemListener(...);  
miBlue.addItemListener(...);  
}
```

```
public void actionPerformed(ActionEvent e) { //...}  
public static void main(String[] args) { //...}  
}
```



Swing工具栏

- **JToolBar**类用于实现工具栏
- 也是按钮、文本框等各种控件的容器
- 可以位于窗体的任何一个边框，或者单独成为一个窗体，用户可以拖动它以改变其位置
- 常用构造方法
 - **JToolBar()** 创建新的工具栏，默认的方向为 **JToolBar.HORIZONTAL**，其中的组件水平摆放；
JToolBar(int orientation) 创建具有指定 **orientation** 的新工具栏。**orientation** 不是 **HORIZONTAL** 就是 **VERTICAL**
 - **JToolBar(String name)** 创建一个具有指定 **name** 的新工具栏。名称用作浮动式 (**undocked**) 工具栏的标题。
默认的方向为 **HORIZONTAL**
 - **JToolBar(String name, int orientation)**



Swing工具栏

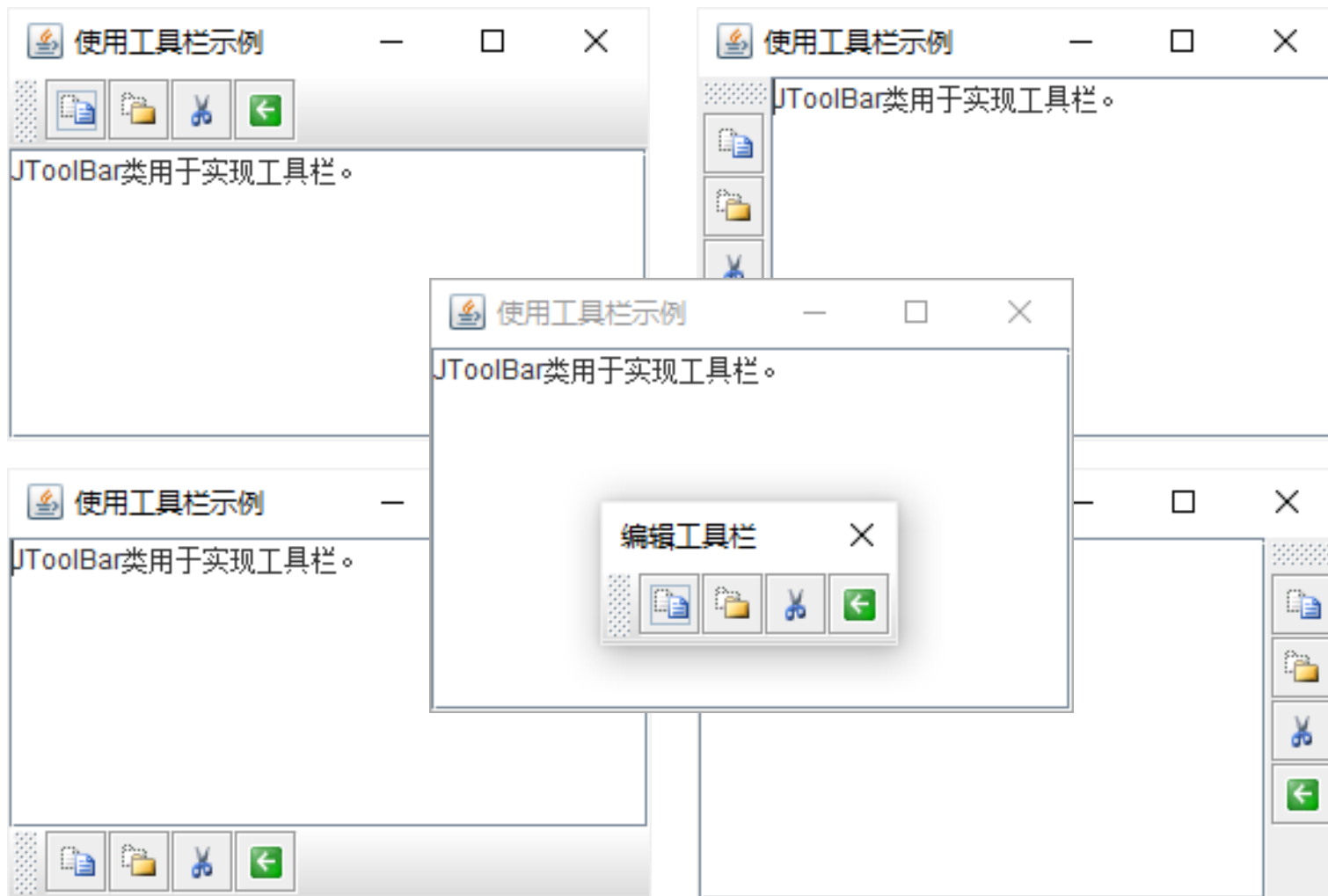
● JToolBar的常用方法

- Component **add**(Component comp) 将指定组件追加到此容器的尾部
- JButton **add**(Action a) 添加一个指派动作的新的 JButton
- void **addSeparator**() 将默认大小的分隔符添加到工具栏的末尾
- void **setFloatable**(boolean b) 设置工具栏是否为浮动状态。通常，可以将浮动工具栏拖动到同一个容器中的不同位置，或者拖动到自己的窗口中
- void **setRollover**(boolean rollover) 设置是否仅当鼠标指针悬停在工具栏按钮上时，才绘制该按钮的边框
- void **setToolTipText**(String text) 设置提示信息



Swing工具栏

【例8-27】为窗体添加工具栏



```
import java.awt.event.*;
import javax.swing.*;
public class JToolBarTester extends JFrame
    implements ActionListener{
    private JButton btnCopy,btnPaste,btnCut,btnExit;
    private JTextArea ta;
    public JToolBarTester(){
        super(" 工具栏应用");
        this.setSize(300, 200);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ta=new JTextArea();
        JScrollPane sp=new JScrollPane(ta);

        // 命令按钮
        btnCopy=new JButton(new ImageIcon("img/copy.png"));
        btnCopy.setToolTipText(" 复制");
        btnCopy.setActionCommand(" 复制");
        btnPaste.addActionListener(this); // 注册事件监听器
        btnPaste=new JButton(new ImageIcon("img/paste.png"));
        btnPaste.setToolTipText(" 粘贴");
        btnPaste.setActionCommand(" 粘贴");
        btnPaste.addActionListener(this); // 注册事件监听器
```

```
btnCut=new JButton(new ImageIcon("img/cut.png"));
btnCut.setToolTipText("剪切"); btnCut.addActionListener(this);
btnExit=new JButton(new ImageIcon("img/exit.png"));
btnExit.setToolTipText("退出"); btnExit.addActionListener(this);
```

```
JToolBar tb=new JToolBar("编辑工具栏"); //工具栏
```

```
//将命令按钮添加到工具栏
```

```
tb.add(btnCopy); tb.add(btnPaste); tb.add(btnCut); tb.add(btnExit);
```

```
this.add(tb, "North"); // 将工具栏添加至窗体标题栏下方
```

```
this.add(sp, "Center");
```

```
}
```

```
public void actionPerformed(ActionEvent e){
```

```
String cmd=e.getActionCommand();
```

```
if(cmd.equals("退出"))
```

```
    System.exit(0);
```

```
else if(cmd.equals("粘贴"))
```

```
    ta.paste();
```

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

```
    JToolBarTester frm=new JToolBarTester();
```

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

```
}
```



Swing工具栏：动作接口 Action

- **Action**可被用来将功能和状态从组件中分离出来。当多个组件执行相同的功能时，例如某个菜单项与工具栏上的某个按钮，可以考虑使用**Action**对象实现该功能
- 一个**Action**对象本身就是一个动作监听器，它不仅提供了动作事件处理，而且还提供了集中处理动作事件触发组件的状态，这些组件包括工具栏按钮、菜单项、按钮和文本框等。
- 一个**Action**可以处理的状态，包括：
 - 文字 (**text**)
 - 图标 (**icon**)
 - 助记符(**mnemonic**)
 - 启用 (**enabled**)
 - 选定 (**selected**) 的状态
 - 工具提示文本 (**toolTipText**)



Swing工具栏：动作接口Action

- 大多数的Swing组件都支持Action对象，可以将某个组件xxx与Action对象关联（附加），这可通过组件的构造方法xxx(Action a) 或调用xxx.setAction(Action a)方法或容器的add(Action a)方法实现。关联后将发生：
 - 更新组件的状态以匹配Action对象的状态。例如，如果操作的文本和图标值被设置，组件的文本和图标设置为这些值
 - 将Action对象注册为组件上的一个动作侦听器
 - 如果Action对象的状态改变，组件的状态更新以匹配Action对象。例如，如果更改了该Action对象的启用状态，则与它相关联的所有组件都将更改它们的启用状态以匹配Action对象



Swing工具栏：动作接口 Action

● 抽象类 **AbstractAction**

- 提供 **Action** 接口的默认实现。它定义了一些标准行为，比如 **Action** 对象属性 (**icon**、**text** 和 **enabled**) 的 **get** 和 **set** 方法。
- 开发人员只需为此抽象类创建子类并定义 **actionPerformed** 方法即可
- 常用构造方法
 - **AbstractAction()** 用默认描述字符串和默认图标定义一个 **Action** 对象
 - **AbstractAction(String name)** 指定描述字符串
 - **AbstractAction(String name, Icon icon)** 指定描述字符串和指定图标



Swing工具栏：动作接口 Action

- 抽象类 **AbstractAction**

- 常用方法

- **void putValue(String key, Object newValue)** 设置与指定键关联的值
 - **Object getValue(String key)** 获取指定键对应的值
 - **void setEnabled(boolean newValue)** 启用或禁用该操作

- 其中的常用的键 (**Key**) 有 (从 **Action** 接口继承)

- **NAME** : 用来存储文本的键
 - **SHORT_DESCRIPTION**: 用来存储用于工具提示文本的键
 - **SMALL_ICON**: 用来存储小型 **Icon** (比如 **ImageIcon**) 的键。该键通常用于菜单, 比如 **JMenuItem**



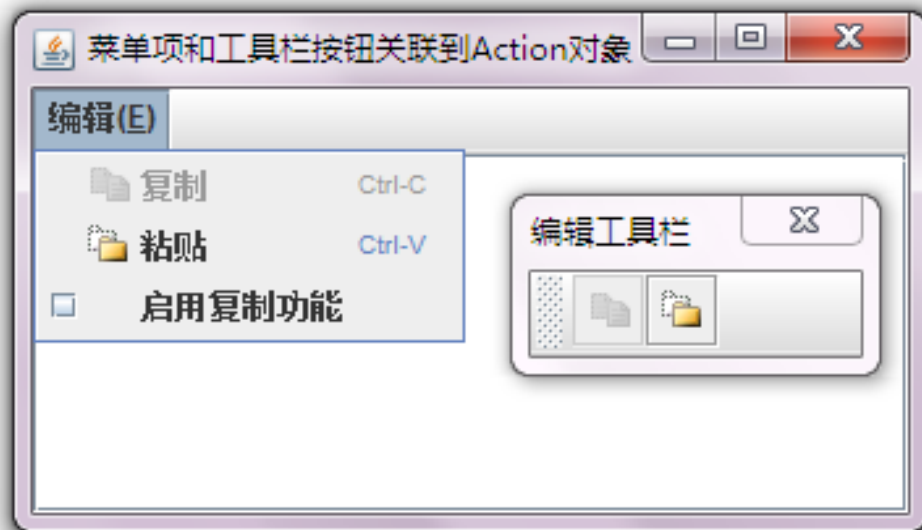
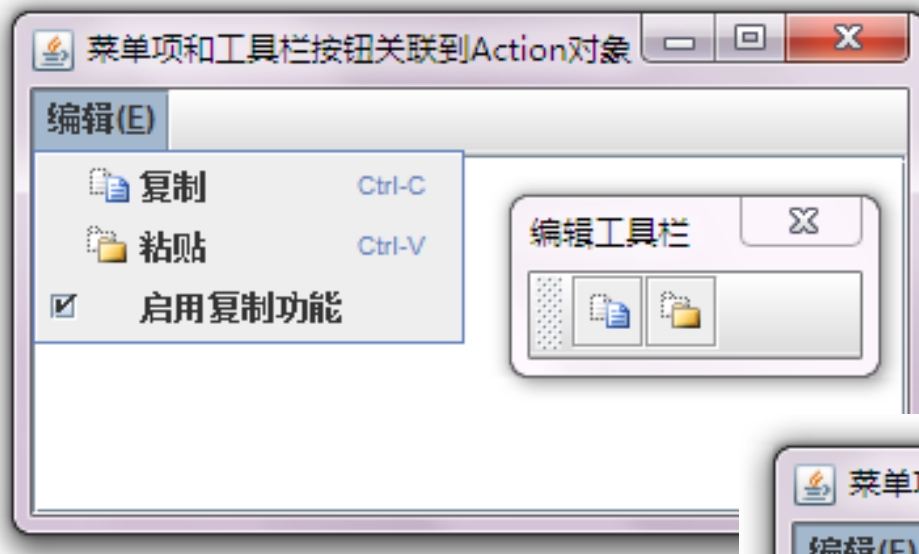
Swing工具栏：动作接口 Action

- **LARGE_ICON_KEY**: 用来存储 Icon 的键。该键通常由按钮（如 JButton 和 JToggleButton）使用。如果将同一个 Action 用于菜单和按钮，通常应同时指定 **SMALL_ICON** 和 **LARGE_ICON_KEY**。菜单将使用 **SMALL_ICON**，按钮将使用 **LARGE_ICON_KEY**。此字段的值以 'Swing' 为前缀，以避免与现有 Action 的可能冲突
- **ACCELERATOR_KEY**: 用来存储将用作动作加速器的 **KeyStroke** 的键
- **MNEMONIC_KEY**: 用来存储助记符的键
- **SELECTED_KEY**: 用来存储对应于选定状态的 **Boolean** 值的键



Swing工具栏：动作接口Action

【例8-28】 利用Action使菜单项和工具栏按钮执行相同的功能和具有同样的状态



```
import java.awt.event.*;
import javax.swing.*;
public class ActionTester extends JFrame{
    private JCheckBoxMenuItem cbMenu;
    private JTextArea ta;
    private Action copyAction,pasteAction;
    public ActionTester(){
        super("菜单项和工具栏按钮关联到Action对象");
        this.setSize(350, 200);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // 两个自定义的Action对象
        copyAction=new CopyAction("复制",
            new ImageIcon("img/copy.png"), "复制",
            KeyStroke.getKeyStroke(
                KeyEvent.VK_C, InputEvent.CTRL_DOWN_MASK));
        pasteAction=new PasteAction("粘贴",
            new ImageIcon("img/paste.png"),"粘贴",
            KeyStroke.getKeyStroke(
                KeyEvent.VK_V, InputEvent.CTRL_DOWN_MASK));
```

//菜单栏

```
JMenuBar mb=new JMenuBar();  
JMenu editMenu=new JMenu("编辑(E)"); editMenu.setMnemonic('E');  
editMenu.add(copyAction); //关联Action对象  
editMenu.add(pasteAction);  
cbMenu=new JCheckBoxMenuItem("启用复制功能",true);  
cbMenu.addItemListener(new ItemListener() { //创建并注册选项监听器  
    public void itemStateChanged(ItemEvent e) {  
        copyAction.setEnabled(!copyAction.isEnabled());  
    }  
});
```

```
JMenuItem copyMI=new JMenuItem(); //菜单项  
copyMI.setAction(copyAction); // 菜单项关联  
Action对象  
editMenu.add(copyMI); // 菜单项添加到菜单
```

```
editMenu.add(cbMenu);  
mb.add(editMenu);  
this.setJMenuBar(mb);  
JToolBar tb=new JToolBar("编辑工具栏"); //工具栏  
tb.add(copyAction); //工具栏单联Action对象  
tb.add(pasteAction);
```

```
JButton btnCopy=new JButton(); // 命名按钮  
btnCopy.setAction(copyAction); // 命令按钮  
关联到Action对象  
tb.add(btnCopy); // 命令按钮添加到工具栏
```

```
ta=new JTextArea();  
JScrollPane sp=new JScrollPane(ta);  
this.add(tb, "North");    this.add(sp, "Center");
```

```
}
```

```

class CopyAction extends AbstractAction{ // 内部类--复制动作
    public CopyAction(String text, ImageIcon icon,
                      String desc,KeyStroke keyStroke){
        super(text,icon);
        putValue(SHORT_DESCRIPTION, desc);
        putValue(ACCELERATOR_KEY, keyStroke);
    }
    public void actionPerformed(ActionEvent e){ ta.copy(); }
}

class PasteAction extends AbstractAction{ // 内部类--粘贴动作
    public PasteAction(String text, ImageIcon icon,
                      String desc,KeyStroke keyStroke){
        super(text,icon);
        putValue(SHORT_DESCRIPTION, desc);
        putValue(ACCELERATOR_KEY, keyStroke);
    }
    public void actionPerformed(ActionEvent e){ ta.paste(); }
}

public static void main(String[] args) {
    ActionTester frm=new ActionTester();
    frm.setVisible(true);
}
}

```




Swing高级组件

- 对话框（均为模式对话框）
 - 标准对话框—由JOptionPane类的静态方法提供
 - 消息对话框—JOptionPane.showMessageDialog方法
 - 确认对话框—JOptionPane.showConfirmDialog方法
 - 输入对话框—JOptionPane.showInternalInputDialog方法
 - 文件选择器---由JFileChooser类提供
 - 打开文件对话框：从中选择一个要处理的文件，获得文件路径
 - 保存文件对话框：从中选择一个文件要保存的路径并输入文件名
 - 颜色选择器---由JColorChooser类的静态方法提供
- 表格组件JTable类
- 树组件JTree类



Swing高级组件：标准对话框

● JOptionPane 类

有助于方便地弹出要求用户提供值或向其发出通知的标准对话框

静态方法名	描述
showConfirmDialog	显示一个确认对话框，询问一个确认问题，如 yes/no/cancel
showMessageDialog	显示一个消息对话框，告知用户某事已发生
showInputDialog	显示一个输入对话框，提示输入字符串
showOptionDialog	上述三项的大统一 (Grand Unification)。



Swing高级组件：标准对话框

- 创建消息对话框的常用方法

static void **showMessageDialog**(Component parent,
Object message, String title, int messageType)

- **parentComponent** — 确定在其中央位置显示对话框的窗口；如果为 null，则显示在屏幕中央
- **message** — 要显示的消息，可以是文本、图标等
- **title** - 对话框的标题字符串
- **messageType** — 要显示的消息类型，为 JOptionPane 中定义的常量：
 - **ERROR_MESSAGE** — 使用 x 图标
 - **INFORMATION_MESSAGE** — 使用 i 图标
 - **WARNING_MESSAGE** — 使用 ! 图标
 - **QUESTION_MESSAGE** — 使用 ? 图标
 - **PLAIN_MESSAGE** — 未使用图标



Swing高级组件：标准对话框

【例8-29】消息对话框示例



```
JOptionPane.showMessageDialog(this,  
    "输入用户名和密码不正确，请重试！",  
    "系统提示",  
    JOptionPane.ERROR_MESSAGE);
```



```
JOptionPane.showMessageDialog(this,  
    "文件已成功保存至磁盘！",  
    "系统提示",  
    JOptionPane.INFORMATION_MESSAGE);
```



```
JOptionPane.showMessageDialog(  
    this, "磁盘可用空间已不足100M,  
    请及时清理空间！", "系统提示",  
    JOptionPane.WARNING_MESSAGE);
```



Swing高级组件：标准对话框

- 创建确认对话框的常用方法

static int **showConfirmDialog**(Component parent,
Object message, String title,
int optionType, int messageType)

- **optionType**— 指定对话框中要显示的按钮组合，为 **OptionPane** 中定义的常量：
 - **YES_NO_OPTION**—显示yes（是）和 no（否） 按钮
 - **YES_NO_CANCEL_OPTION**— 显示yes（是）、no（否）和cancel（取消）按钮
 - **OK_CANCEL_OPTION**— 显示ok(确定)和canel(取消)按钮
- 返回值— 指示用户操作，为 **OptionPane** 中的常量：
YES_OPTION、**NO_OPTION**、**OK_OPTION**、**CANCEL_OPTION**、**CLOSED_OPTION**（用户关闭对话框）
- 通过返回值判断用户操作时，应当与按钮类型一致！



Swing高级组件：标准对话框

【例8-30】 确认对话框示例：退出应用程序确认



```

import java.awt.event.*;      import javax.swing.*;
public class ConfirmDialogTester extends JFrame {
    public ConfirmDialogTester(){
        super("确认对话框示例");
        this.setSize(400, 300);
        this.setDefaultCloseOperation(JFrame.DI_DO_NOTHING_ON_CLOSE);
        this.setVisible(true);
        this.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e){
                JFrame frm=(JFrame) e.getWindow();
                int resultDlg=JOptionPane.showConfirmDialog(frm,
                    "确定要退出应用程序吗？ ", "操作确认",
                    JOptionPane.YES_NO_OPTION,
                    JOptionPane.QUESTION_MESSAGE);
                if(resultDlg==JOptionPane.YES_OPTION){
                    frm.dispose(); //销毁窗体
                    System.exit(0);
                }
            }
        });
    }
    public static void main(String[] args) { new ConfirmDialogTester(); }
}

```



Swing高级组件：文件选择器JFileChooser

- 当需要获取文件信息或读取文件时，首先要指定文件名（包括路径），此时便可利用文件选择器方便地从文件系统中选择所要处理的文件——打开文件对话框
- 当或者当需要将数据保存到文件时，首先要指定文件名（包括路径），此时便可利用文件选择器方便地指定路径并输入文件名——保存文件对话框
- 常用构造方法
 - **JFileChooser()** 构造一个指向用户默认目录的 JFileChooser。此默认目录在 Windows 上通常是“我的文档”，在 Unix 上是用户的主目录
 - **JFileChooser(String dir)** 构造一个由字符串表示的路径的 JFileChooser
 - **JFileChooser(File dir)** 构造一个由 File 对象表示的路径的 JFileChooser



Swing高级组件：文件选择器JFileChooser

● 常用方法

- **int showOpenDialog(Component parent)** 弹出一个 "Open File" 文件选择器对话框。注意，**approve** 按钮上显示的文本由 **L&F** 决定。其返回值
 - **JFileChooser.APPROVE_OPTION**: 用户确定选择某个文件
 - **JFileChooser.CANCEL_OPTION**: 用户取消了操作
 - **JFileChooser.ERROR_OPTION**: 发生错误或者该对话框已被解除
- **int showSaveDialog(Component parent)** 弹出一个 "Save File" 文件选择器对话框
- **void setDialogTitle(String dialogTitle)** 设置文件选择器对话框窗体的标题



Swing高级组件：文件选择器JFileChooser

- `File` `getSelectedFile()` 返回选中的文件
- `File[]` `getSelectedFiles()` 如果将文件选择器设置为允许选择多个文件，则返回选中文件的列表
- `void` `setMultiSelectionEnabled(boolean b)` 设置文件选择器，以允许选择多个文件
- `void` `setFileFilter(FileFilter filter)` 设置当前文件过滤器，以便从用户的视图中过滤文件
- `void` `addChoosableFileFilter(FileFilter filter)` 向用户可选择的文件过滤器列表添加一个过滤器
 - `FileNameExtensionFilter`是抽象类`FileFilter`的一个实现，其构造方法如下：

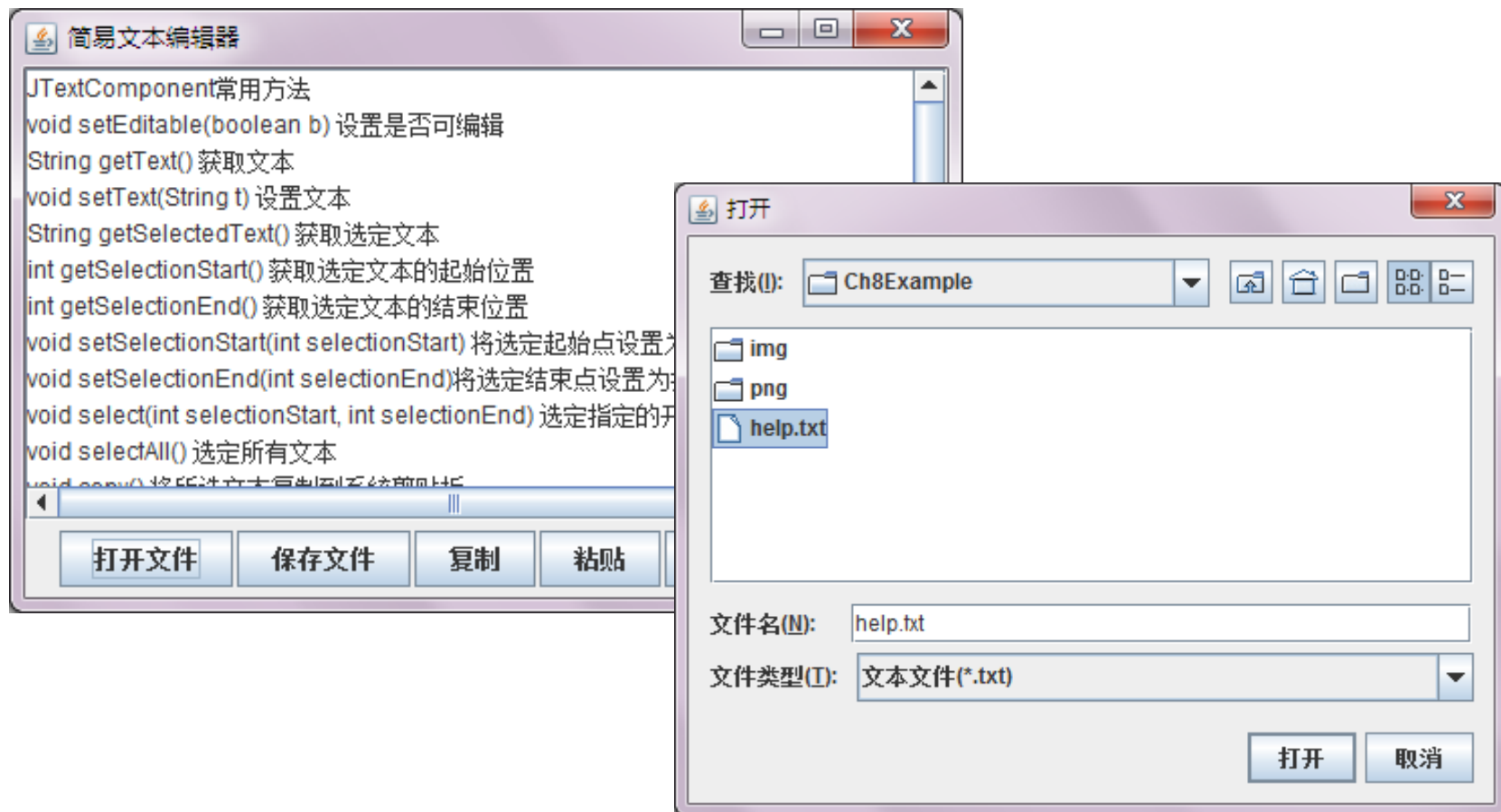
`FileNameExtensionFilter(String description, String... extensions)`

例：`new FileNameExtensionFilter("文本文件", "txt", "htm")`



Swing高级组件：文件选择器JFileChooser

【例8-31】 例8-24中打开文件功能中，加用文件选择器指定文件名及其路径



```
public void actionPerformed(ActionEvent e){
    String cmd=e.getActionCommand();
    if(cmd.equals("打开文件")){ // 打开文本文件
        JFileChooser fc=new JFileChooser(); //文件选择器
        FileNameExtensionFilter filter=
            new FileNameExtensionFilter("文本文件(*.txt)", "txt");
        fc.addChoosableFileFilter(filter); //添加一个过滤器
        filter=new FileNameExtensionFilter(
            "THML文件(*.htm,*.html)", "htm","html");
        fc.addChoosableFileFilter(filter); //添加一个过滤器
        int returnVal = fc.showOpenDialog(this); //显示打开文件对话框
        if(returnVal != JFileChooser.APPROVE_OPTION) //判断用户操作
            return;
        String fileName=fc.getSelectedFile().getAbsolutePath(); //获取
        用户选择的文件
    }
}
```

// 读取文件的代码 此处省略了哦!

```
}else if(cmd.equals("保存文件")){
}else if(cmd.equals("粘贴")){ ta.paste(); }
```

```
}
```



Swing高级组件：颜色选择器JColorChooser

- 提供颜色对话框，便于用户选择一个颜色
- 可以直接用其静态方法**showDialog**用于显示一个颜色对话框

static Color **showDialog**(Component parent,
String title, Color initialColor)

— 参数

- **parent** — 对话框的父 **Component**
- **title** — 对话框的标题
- **initialColor** — 显示颜色选取器时的初始 **Color** 设置

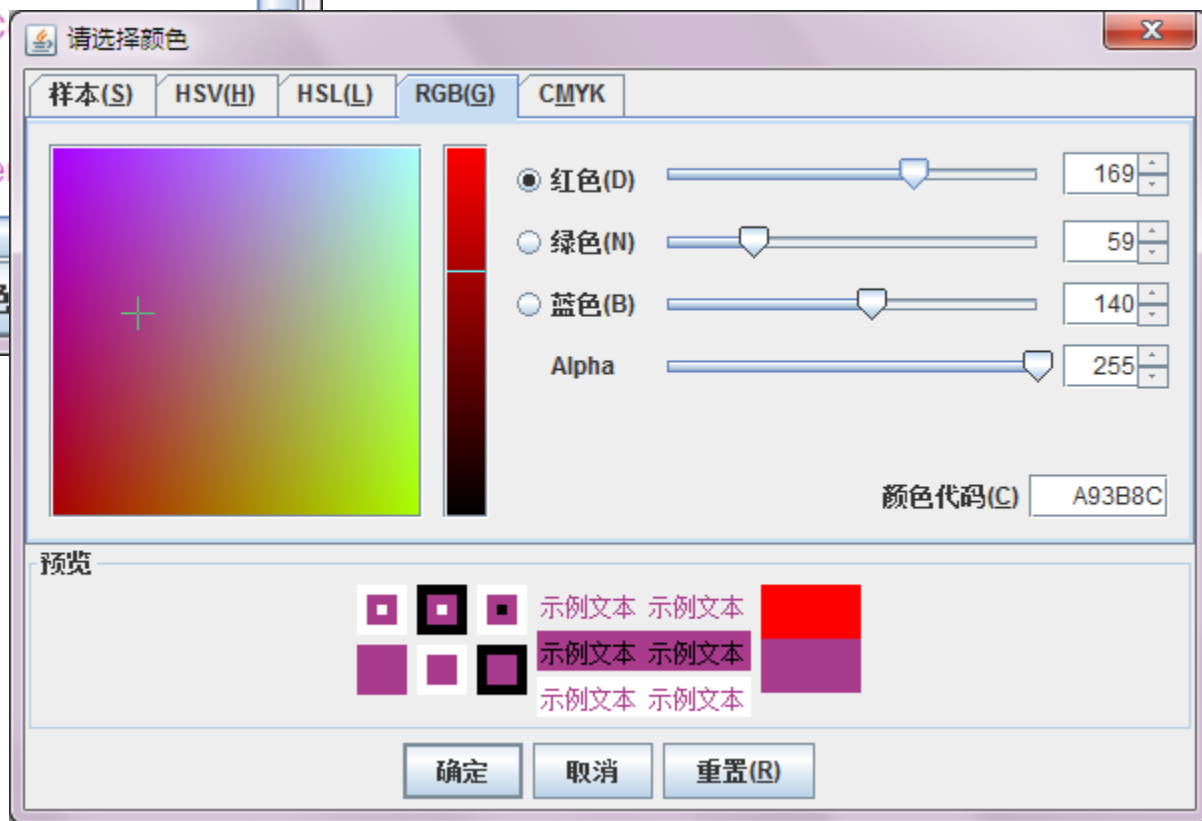
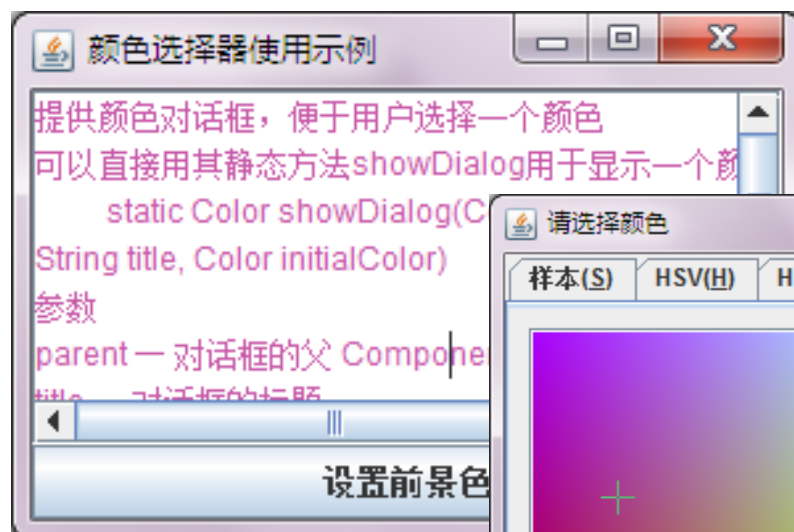
— 返回值

- 所选颜色；如果用户退出，则返回 **null**



Swing高级组件：颜色选择器JColorChooser

【例8-32】 利用JColorChooser显示颜色对话框，以便选取颜色



```

import java.awt.*;    import java.awt.event.*;  import javax.swing.*;
public class JColorChooserTester extends JFrame
                                implements ActionListener{

    private JButton btnColor;
    private JTextArea ta;
    public JColorChooserTester(){      .....      }
    public void actionPerformed(ActionEvent e){
        JButton btn=(JButton)e.getSource();
        if(btn==btnColor){
            Color color = JColorChooser.showDialog(this,
                "请选择颜色", Color.RED); // 颜色选择器
            if(color!=null) // 返回颜色不为null
                ta.setForeground(color);
        }
    }
    public static void main(String[] args) {
        JColorChooserTester frm=new JColorChooserTester();
        frm.setVisible(true);
    }
}

```



Swing高级组件：JTable

- JTable组件用于显示和编辑由行和列构成的二维数据表
- 数据表的维护是由实现TableModel接口的数据模型类来实现的，而JTable是数据模型的一个视图（可以排序、过滤）
- JTable类的常用构造方法
 - JTable(int numRows, int numColumns) 使用DefaultTableModel构造具有numRows行和numColumns列个空单元格的JTable。列名称采用"A"、"B"、"C"等形式
 - JTable(Object[][] rowData, Object[] columnNames) 构造一个JTable来显示二维数组rowData中的值，其列名由一维数组columnNames指定
 - JTable(Vector rowData, Vector columnNames)
 - JTable(TableModel dm) 使用数据模型dm构造一个JTable



Swing高级组件：JTable

● JTable类的常用方法

- void **setModel**([TableModel](#) dm) 将数据模型设置为dm
- void **setRowSelectionAllowed**(boolean rowSelectionAllowed) 设置是否可以选择表中的行
- void **setColumnSelectionAllowed**(boolean columnSelectionAllowed) 设置是否可以选择表中的列
- void **setSelectionMode**(int selectionMode) 将表的选择模式设置为只允许单个选择 (ListSelectionModel.SINGLE_SELECTION)、单个连续间隔选择 (SINGLE_INTERVAL_SELECTION) 或多间隔选择 (MULTIPLE_INTERVAL_SELECTION, 默认值)
- void **selectAll**() 选择表中的所有行、列和单元格
- void **clearSelection**() 取消选中所有已选定的行和列



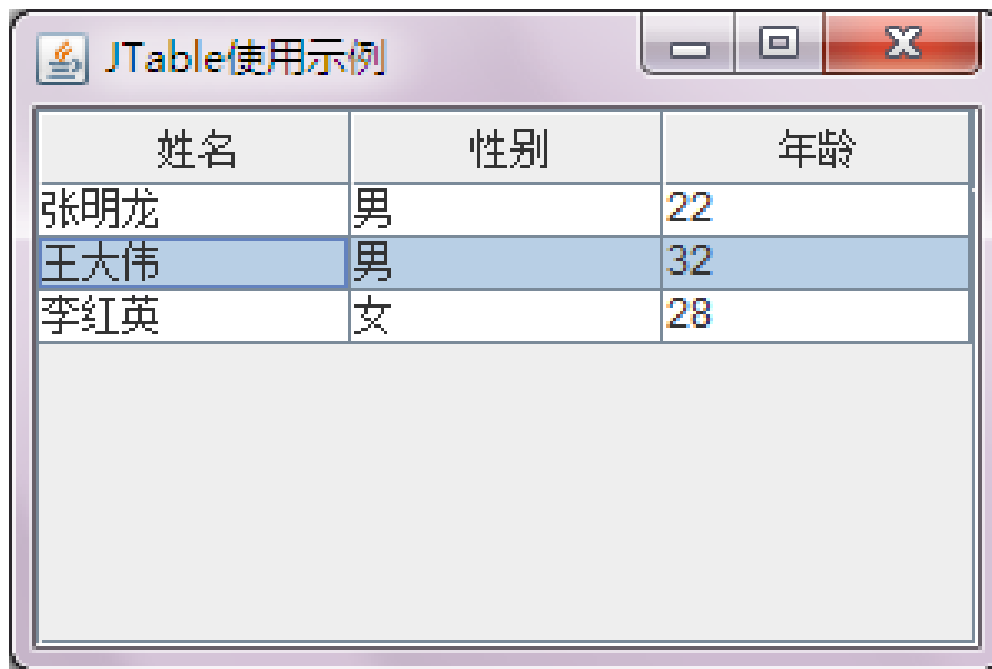
Swing高级组件: JTable

- void **setModel**(TableModel dm) 将数据模型设置为dm
- int **getSelectedRow()** 返回第一个选定行的索引; 如果没有选定的行, 则返回 -1
- int[] **getSelectedRows()** 返回所有选定行的索引
- int **getSelectedRowCount()** 返回选定行数
- int **getSelectedColumn()**
- int[] **getSelectedColumns()**
- int **getSelectedColumnCount()**
- int **getEditingRow()** 返回包含当前被编辑的单元格的行索引。如果没有编辑任何内容, 则返回 -1
- int **getEditingColumn()**



Swing高级组件：JTable

【例8-33】 利用JTable显示二维数组中的数据



姓名	性别	年龄
张明龙	男	22
王大伟	男	32
李红英	女	28

```

import javax.swing.*;
public class JTableTester extends JFrame{
    public JTableTester(){
        super("JTable使用示例");
        this.setSize(300, 200);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        String[] header={"姓名","性别","年龄"}; // 定义表格列标题文本
        Object[][] data={{"张明龙", "男", 22}, // 定义表格数据
                           {"王大伟", "男", 32},
                           {"李红英", "女", 28} };
        JTable table=new JTable(data,header); // 表格
        JScrollPane sp=new JScrollPane(table);

        this.add(sp, "Center");
    }
    public static void main(String[] args) {
        JTableTester frm=new JTableTester();
        frm.setVisible(true);
    }
}

```



Swing高级组件: JTable

● TableModel接口

First Name	Last Name	Sport	# of Years	Vegetarian
Kathy	Smith	Snowboarding	5	<input type="checkbox"/>
John	Doe	Rowing	3	<input checked="" type="checkbox"/>
Sue	Black	Knitting	2	<input type="checkbox"/>
Jane	White	Speed reading	20	<input checked="" type="checkbox"/>



- **TableModel** 接口指定了 **JTable** 用于访问表格式数据模型的方法
- 只要数据模型实现了 **TableModel** 接口, 就可以通过以下两行代码设置 **JTable** 显示该模型:

```

TableModel myData = new MyTableModel();
JTable table = new JTable(myData);
  
```



Swing高级组件：JTable

– TableModel接口规定的方法

void	addTableModelListener (TableModelListener l) 每当数据模型发生更改时，就将一个侦听器添加到被通知的列表中
Class <?>	getColumnClass (int columnIndex) 针对列中所有的单元格值，返回最具体的超类
int	getColumnCount () 返回该模型中的列数
String	getColumnName (int columnIndex) 返回 columnIndex 位置的列的名称
int	getRowCount () 返回该模型中的行数
Object	getValueAt (int rowIndex, int columnIndex) 返回 columnIndex 和 rowIndex 位置的单元格值
boolean	isCellEditable (int rowIndex, int columnIndex) 如果 rowIndex 和 columnIndex 位置的单元格是可编辑的，则返回 true
void	removeTableModelListener (TableModelListener l) 每当数据模型发生更改时，就从被通知的列表中移除一个侦听器。
void	setValueAt (Object aValue, int rowIndex, int columnIndex) 将 columnIndex 和 rowIndex 位置的单元格中的值设置为 aValue



Swing高级组件: JTable

● 抽象类AbstractTableModel

- 为 **TableModel** 接口中的大多数方法提供默认实现。其中，**setValueAt**方法提供了此空实现，因此，如果用户的数据模型是不可编辑的，则不必实现此方法
- 它负责管理侦听器，并为生成 **TableModelEvents** 以及将其调度到侦听器提供方便
- 用户可以自由构造表格数据结构，例如用数组或向量
- 要创建一个具体的 **TableModel** 作为 **AbstractTableModel** 的子类，还需实现三个方法的：
 - `public int getRowCount();`
 - `public int getColumnCount();`
 - `public Object getValueAt(int row, int column);`
- 程序员自己实现**AbstractTableModel** 的子类并不容易，因为还需要存储和管理表格数据



Swing高级组件: JTable

- 非抽象类DefaultTableModel
 - 是抽象类AbstractTableModel 的具体实现
 - 内部采用向量Vector存储和管理表格列名称(protected Vector columnIdentifiers, 包含多个列标识符)和数据(protected Vector dataVector, 包含多个 Object 类型的 Vector)
 - 提供了对表格数据的增删改方法
 - 是JTable的默认数据模型
 - 常用构造方法
 - DefaultTableModel(int rowCount, int columnCount)
 - DefaultTableModel(Object[][] data, Object[] columnNames)
构造一个 DefaultTableModel, 并通过将 data 和 columnNames 传递到 setDataVector 方法来初始化该表
 - DefaultTableModel(Vector data, Vector columnNames) 构造一个 DefaultTableModel, 并通过将 data 和 columnNames 传递到 setDataVector 方法来初始化该表
 - 常用方法

void	addColumn (Object columnName, Object [] columnData) 将一列添加到模型中
void	addColumn (Object columnName, Vector columnData)
void	addRow (Object [] rowData) 添加一行到模型的结尾
void	addRow (Vector rowData)
void	insertRow (int row, Object [] rowData) 在模型中的 row 位置插入一行
void	insertRow (int row, Vector rowData)
void	removeRow (int row) 移除模型中 row 位置的行
void	setColumnIdentifiers (Object [] newIdentifiers) 替换模型中的列标识符
void	setColumnIdentifiers (Vector columnIdentifiers)
void	setDataVector (Object [][] dataVector, Object [] columnIdentifiers) 用数组 dataVector 中的值替换 dataVector 实例变量中的值
void	setDataVector (Vector dataVector, Vector columnIdentifiers)
Object	getValueAt (int row, int column) 返回 row 和 column 处单元格的属性值
void	setValueAt (Object aValue, int row, int column) 设置 column 和 row 处单元格的对象值



Swing高级组件：JTable

- 非抽象类DefaultTableModel

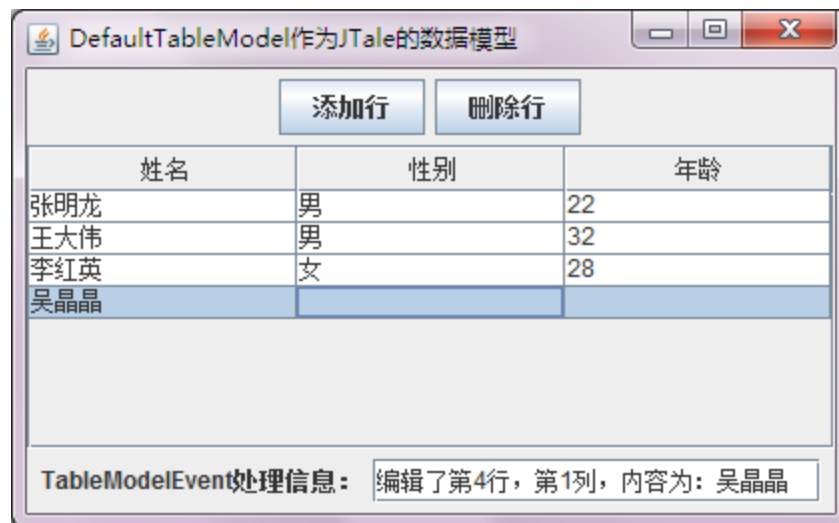
- TableModelEvent事件

- 当对表格内容进行操作时，例如修改表格结构、增加行、删除行、改变单元格的值，都会引发TableModelEvent事件，数模型对应的事件监听接口为TableModelListenser，处理事件的方法为：
`public void tableChanged(TableModelEvent e);`
 - TableModelEvent类的常用方法
 - `int getType()` 返回事件类型，该类型为以下类型之一：
TableModelEvent.INSERT、UPDATE 和 DELETE
 - `int getColumn()` 返回事件的列号
 - `int getFirstRow()` 返回第一个被更改的行号。
HEADER_ROW 表示元数据，即列的名称、类型和顺序
 - `int getLastRow()` 返回最后一个被更改的行号



Swing高级组件: JTable

【例8-34】 利用DefaultTableModel创建人员信息表格，并实现行的添加和删除；同时，为数据模型注册TableModelEvent事件监听器，当用户在表格上添加或删除行以及编辑单元格内容时，对产生TableModelEvent事件进行相应的处理。



```
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.table.*;
public class DefaultTableModelTester extends JFrame
    implements ActionListener,TableModelListener{
    private JButton btnAdd,btnDel;
    private JTable table;
    private DefaultTableModel dm;
    private JTextField txtInfo;
    public DefaultTableModelTester(){
        super("DefaultTableModel作为JTale的数据模型");
        this.setSize(420, 260);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel panel=new JPanel();
        btnAdd=new JButton("添加行"); btnAdd.addActionListener(this);
        btnDel=new JButton("删除行"); btnDel.addActionListener(this);
        panel.add(btnAdd); panel.add(btnDelete);
    }
}
```

```
String[] header={"姓名","性别","年龄"}; // 定义表格列标题文本
Object[][] data={{ "张明龙","男",22},{ "王大伟","男",32},{ "李红英","
女",28}}; // 定义表格数据
dm=new DefaultTableModel(data, header); //数据模型
dm.addTableModelListener(this); //为数据模型注册事件监听器
table=new JTable(dm); // 以dm为数据模型创建JTable
JScrollPane sp=new JScrollPane(table);

JPanel stPanel=new JPanel();
stPanel.add(new JLabel("TableModelEvent处理信息: "));
txtInfo=new JTextField(20);
stPanel.add(txtInfo);

this.add(panel, "North");
this.add(sp, "Center");
this.add(stPanel, "South");

this.setVisible(true);

}
```

//按钮动作事件处理方法

```
public void actionPerformed(ActionEvent e){  
    JButton btn=(JButton)e.getSource();  
    if(btn==btnAdd)  
        dm.addRow(new Object[3]); // 创建数组并作为一行添加到dm  
    else if(btn==btnDel){  
        int rowID=table.getSelectedRow(); //获取被选择的行号  
        if(rowID>0)  
            dm.removeRow(rowID); // 从dm中删除该行  
        else  
            JOptionPane.showMessageDialog(this,  
                "请选择要删除的行!", "系统提示",  
                JOptionPane.WARNING_MESSAGE);  
    }  
}
```

// 表格模型事件处理方法

```
public void tableChanged(TableModelEvent e){
    int rowID=e.getLastRow();
    int columnID=e.getColumn(); //删除或添加行后，返回的列号为-1
    int userOp=e.getType(); //操作类型: INSERT/ UPDATE/ DELETE
    if(columnID== -1)
        switch(userOp){
            case TableModelEvent.INSERT:
                txtInfo.setText("添加了第" +(rowID+1)+ "行");    break;
            case TableModelEvent.DELETE:
                txtInfo.setText("删除了第" +(rowID+1)+ "行");    break;
        }
    else
        txtInfo.setText("编辑了第" +(rowID+1)+ "行，第" +(columnID+1)
            + "列，内容为: " +dm.getValueAt(rowID, columnID));
}

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



Swing高级组件：JTree

- 计算机世界中的树由一序列具有严格父子关系的节点组成。每个节点既可以是上一级节点的子节点，也可以是下一级节点的父节点。因此，同一个节点既可以是父节点也可以是子节点。没有父节点的节点是树的根节点，没有子节点的节点是叶子节点
- **JTree**与**JTable**一样，也是采用分离模型设计，有自己的数据模型（默认为实现了**TreeModel**接口的**DefaultTreeModel**），它是用树形结构分层显示数据模型中数据的视图
- 常用构造方法
 - **JTree**(**Object[] value**) 创建一棵用对象数组中的元素作为子节点的树，根节点不显示
 - **JTree**(**Vector<?> value**)
 - **JTree**(**TreeNode root**) 创建一棵以**root**为根节点的树，并显示根节点
 - **JTree**(**TreeModel newModel**) 使用指定的数据模型创建树，默认显示根节点



Swing高级组件: JTree

● 常用方法

- void **setRootVisible**(boolean rootVisible) 确定根节点是否可见
- void **setEditable**(boolean flag) 确定树是否可编辑
- void **scrollPathToVisible**([TreePath](#) path) 确保路径中所有的路径组件均展开（最后一个路径组件除外）并滚动，以便显示该路径标识的节点。仅当此 **JTree** 包含在 **JScrollPane** 中时才工作
- void **setSelectionModel**([TreeSelectionModel](#) selectionModel) 设置树的选择模型，取值如下
 - **SINGLE_TREE_SELECTION**: 只允许一条路径
 - **CONTIGUOUS_TREE_SELECTION**: 多条连续路径
 - **DISCONTIGUOUS_TREE_SELECTION**: 多条不连续的路径
- [TreePath](#) **getSelectionPath**() 返回首选节点的路径
- [TreePath](#)[] **getSelectionPaths**()
- void **clearSelection**() 清除该选择
- void **addTreeSelectionListener**([TreeSelectionListener](#) tsl) 为 **TreeSelection** 事件添加侦听器

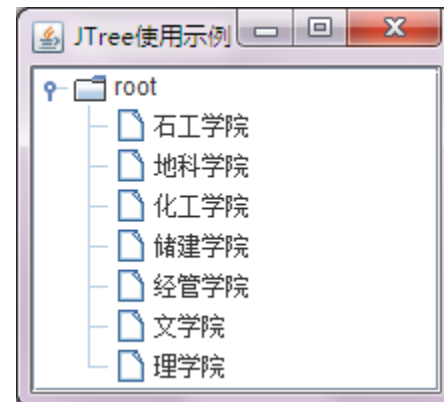


Swing高级组件: JTree

【例8-35】 用JTree显示某高校的下属学院。

```
import javax.swing.*;  import javax.swing.tree.*;
public class JTreeTester extends JFrame{
    public JTreeTester(){
        super("JTree使用示例");
        this.setSize(200, 200);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        String[] schools={"石工学院","地科学院","化工学院",
                           "储建学院","经管学院","文学院","理学院"};

        JTree tree=new JTree(schools); // 用数组元素作为树的子节点
        tree.setRootVisible(true); // 设置显示根节点可见
        JScrollPane sp=new JScrollPane(tree);
        this.add(sp, "Center");
        this.setVisible(true);
    }
    public static void main(String[] args) { new JTreeTester(); }
}
```





Swing高级组件：JTree

- **DefaultMutableTreeNode** 类
 - JTree上的每个节点由实现了**TreeNode**接口的类表示
 - **TreeNode**接口规定了获取树节点信息的方法，例如获取父节点、获取子节点等
 - **MutableTreeNode**接口又扩展了**TreeNode**接口，并提供了更多实用方法，如添加节点、删除节点、设置父节点一棵用对象数组中的元素作为子节点的树，根节点不显示
 - **DefaultMutableTreeNode**类实现了**MutableTreeNode**接口，它是JTree默认的节点类型
 - 程序可以通过**DefaultMutableTreeNode**类创建树的各个节点，并将根节点传递给JTree的构造方法从而在JTree中显示节点



Swing高级组件：JTree

- **DefaultMutableTreeNode**类常用构造方法
 - **DefaultMutableTreeNode(Object userObject)** 创建没有父节点和子节点、但允许有子节点的树节点，并使用指定的用户对象对它进行初始化
 - **DefaultMutableTreeNode(Object userObject, boolean allowsChildren)** 创建节点时指定是否允许有子节点
- **DefaultMutableTreeNode**类常用方法
 - **void add(MutableTreeNode newChild)** 添加子节点。节点newChild将从其父节点移除，并添加到此节点的子数组的结尾，使其成为此节点的子节点
 - **void remove(MutableTreeNode aChild)** 移除子节点
 - **void remove(int childIndex)** 移除指定索引处的子节点
 - **void removeAllChildren()**



Swing高级组件: JTree

- **DefaultMutableTreeNode** 类常用方法
 - TreeNode **getFirstChild()** 获取此节点的第一个子节点
 - TreeNode **getLastChild()** 获取最后一个子节点
 - TreeNode **getChildAt(int index)** 获取此节点的子节点数组中指定索引处的子节点
 - **int getChildCount()** 获取子节点数
 - TreeNode **getParent()** 获取此节点的父节点
 - TreeNode[] **getPath()** 返回从根到达此节点的路径。返回的**TreeNode** 对象组成中的第一个元素是根节点，最后一个元素是此节点
 - TreeNode **getRoot()** 获取包含此节点的树的根
 - DefaultMutableTreeNode **getNextNode()** 获取在此节点的树的前序遍历中此节点之后的节点
 - DefaultMutableTreeNode **getPreviousNode()** 获取在此节点的树的前序遍历中此节点之前的节点
 - String **toString()** 获取此节点的显示文本



Swing高级组件：JTree

- 利用**DefaultMutableTreeNode**创建树节点并在**JTree**中显示的步骤
 - 创建一个**DefaultMutableTreeNode**节点`root`，该节点作为树的根节点
 - 创建其他**DefaultMutableTreeNode**节点`fatheri`，并调用**`root.add(fatheri)`**方法将其添加为`root`的子节点
 - 如果`fatheri`也有子节点`sonj`，则创建**DefaultMutableTreeNode**节点`sonj`，并调用**`fatheri.add(sonj)`**方法将其添加为`fatheri`的子节点
 - 依次类推，创建其他节点并添加到树中
 - 最后，将根节点`root`作为参数传递给**JTree**的构造方法，例如：

JTree tree = new JTree(root);



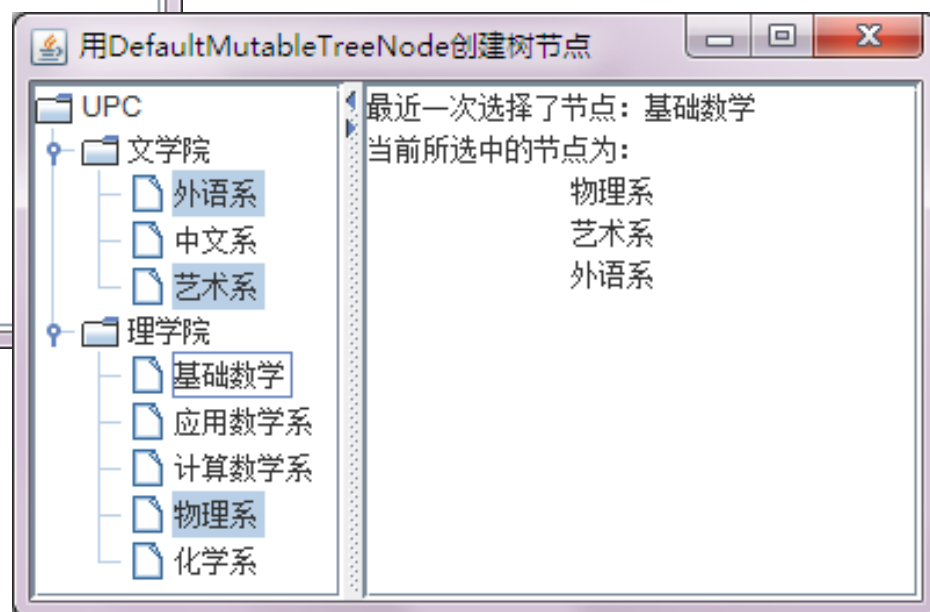
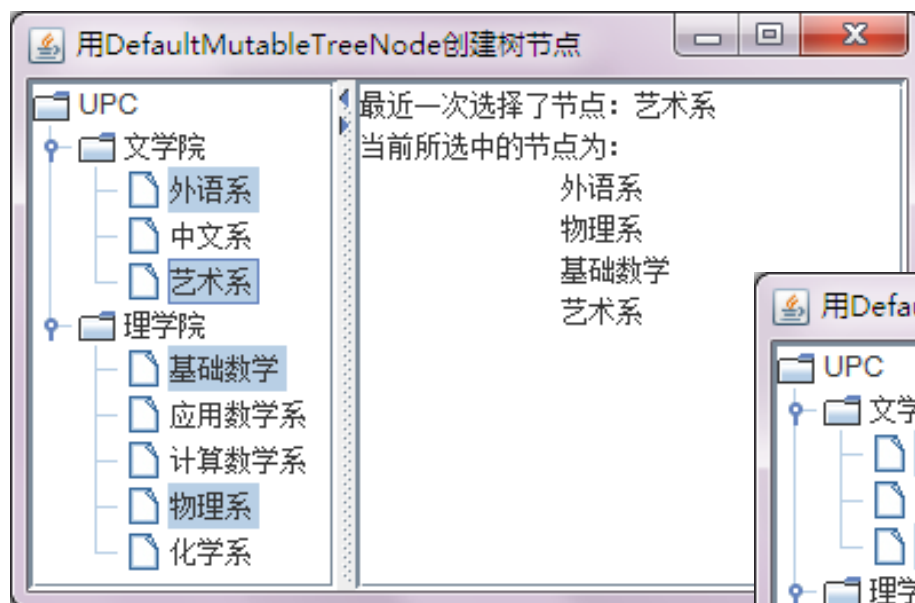
Swing高级组件: JTree

- **TreeSelectionEvent** 事件
 - **JTree**上的节点选择改变时, 将引发该事件
 - 对应的监听器为**TreeSelectionListener**, 其事件处理方法为:
void valueChanged(TreeSelectionEvent e);
 - **TreeSelectionEvent** 类
 - **TreePath getPath()** 方法用于获取选中的第一个节点的路径对象
 - **TreePath**对象由一些列节点所组成, 表示从根节点到指定节点的所有节点元素
 - **Object getLastPathComponent()** 获取路径中的最后一个节点
 - **int getPathCount()** 获取路径上的节点数



Swing高级组件: JTree

【例 8-36】 用 DefaultMutableTreeNode 创建树节点并在 JTree 中显示。




```
import javax.swing.*;
import javax.swing.tree.*;
import javax.swing.event.*;
public class DefaultMutableTreeNodeTester extends JFrame
    implements TreeSelectionListener{

    private JTextArea ta;
    public DefaultMutableTreeNodeTester(){
        super("用DefaultMutableTreeNode创建树节点");
        this.setSize(400, 300);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        String[] schools={"文学院","理学院"};
        String[][] departments={{"外语系","中文系","艺术系"},
                                {"基础数学","应用数学系","计算数学系",
                                "物理系","化学系"}};

        //创建一个节点作为根节点
        DefaultMutableTreeNode root=
            new DefaultMutableTreeNode("UPC");
```

```

DefaultMutableTreeNode father,son;
for(int i=0;i<schools.length;i++){
    father=new DefaultMutableTreeNode(schools[i]);
    root.add(father);
    for(int j=0;j<departments[i].length;j++){
        son=new DefaultMutableTreeNode(departments[i][j]);
        father.add(son);
    }
}
JTree tree=new JTree(root); // 在JTree中显示以root为根节点的树
tree.addTreeSelectionListener(this);
JScrollPane sp=new JScrollPane(tree);

ta=new JTextArea();    JScrollPane sp2=new JScrollPane(ta);
JSplitPane splitPane=new JSplitPane();
splitPane.setOneTouchExpandable(true);
splitPane.setLeftComponent(sp);
splitPane.setRightComponent(sp2);
this.add(splitPane);
this.setVisible(true);
}

```

```

public void valueChanged(TreeSelectionEvent e){
    TreePath treePath=e.getPath();
    DefaultMutableTreeNode node=
        (DefaultMutableTreeNode) treePath.getLastPathComponent();
    ta.setText("最近一次选择了节点: "+node.toString()+"\n");
    ta.append("当前所选中的节点为: \n");
    JTree tree=(JTree) e.getSource();
    TreePath[] paths=tree.getSelectionPaths();
    if(paths!=null){
        for(TreePath p:paths){
            node=(DefaultMutableTreeNode)
                p.getLastPathComponent();
            ta.append("\t"+node.toString()+"\n");
        }
    }
}

```

```

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

```



Swing高级组件：JTabbedPane

- JTabbedPane组件用于实现选项卡面板
- 是一个容器，可用来存放多个带标签的选项卡，每个选项卡又可存放一个组件（通常是一个容器），用户单击每一张选项卡上的标签，便可切换至对应的选项卡
- 选项卡面板一般用作设置配置选项
- CardLayout是该容器默认的布局管理器
- 构造方法
 - JTabbedPane() 创建一个空 TabbedPane，选项卡标签位置为默认的JTabbedPane.Top
 - JTabbedPane(int tabPlacement) 创建一个空 TabbedPane，并指定选项卡标签的位置，其取值为：Top、Bottom、Left或Right
 - JTabbedPane(int tabPlacement, int tabLayoutPolicy) 创建一个空 TabbedPane，并指定选项卡标签的位置和布局策略，其中布局策略可取常量：
 - WRAP_TAB_LAYOUT(默认)：选项卡标签一行放不下时换行
 - SCROLL_TAB_LAYOUT：不换行，而是滚动显示



Swing高级组件：JTabbedPane

● 常用方法

void	addTab (<u>String</u> title, <u>Icon</u> icon, <u>Component</u> component, <u>String</u> tip) 添加一个选项卡，并指定选项卡的 title （标签）和/或 icon 表示的 component （要显示的组件） 和 tip （提示信息），其中任意一个都可以为 null
void	insertTab (<u>String</u> title, <u>Icon</u> icon, <u>Component</u> component, <u>String</u> tip, int index) 在 index 位置插入一个选项卡
void	removeTabAt (int index) 移除 index 位置的选项卡
void	setTabLayoutPolicy (int tabLayoutPolicy) 设置在一次运行中不能放入所有的选项卡时，选项卡窗格使用的对选项卡进行布局安排的策略
void	setTabPlacement (int tabPlacement) 设置此选项卡窗格的选项卡布局
<u>Component</u>	getSelectedComponent () 返回当前选项卡中的组件
int	getSelectedIndex () 返回当前选择的此选项卡窗格的索引



Swing高级组件：JTabbedPane

- **ChangeEvent事件**

- 用户在选项卡面板上选换选项卡时，会引发**ChangeEvent**事件
- 对应的事件监听器接口为**ChangeListener**
- 响应该事件的方法为

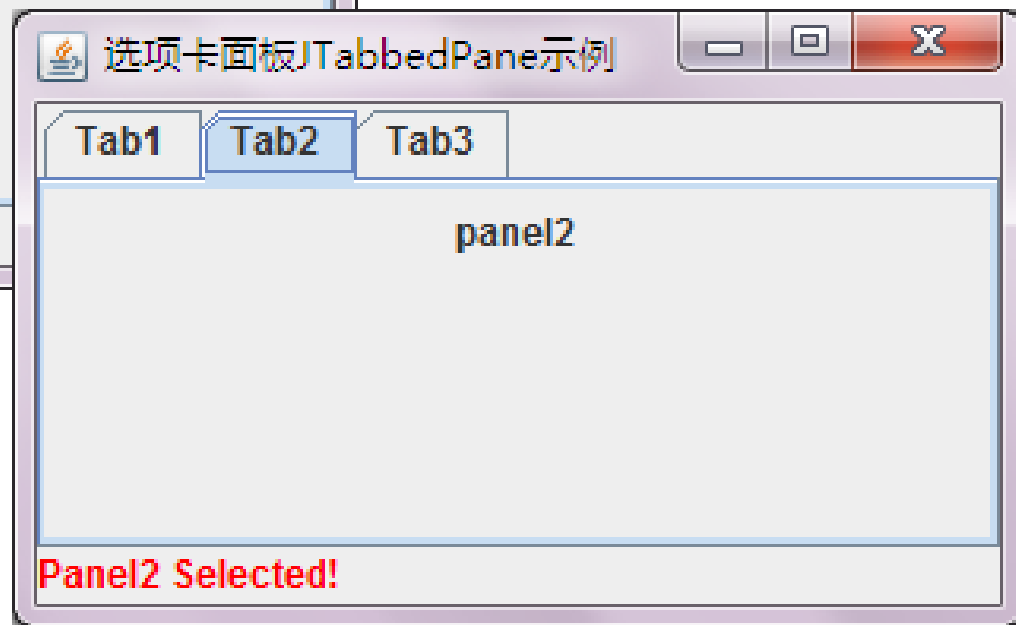
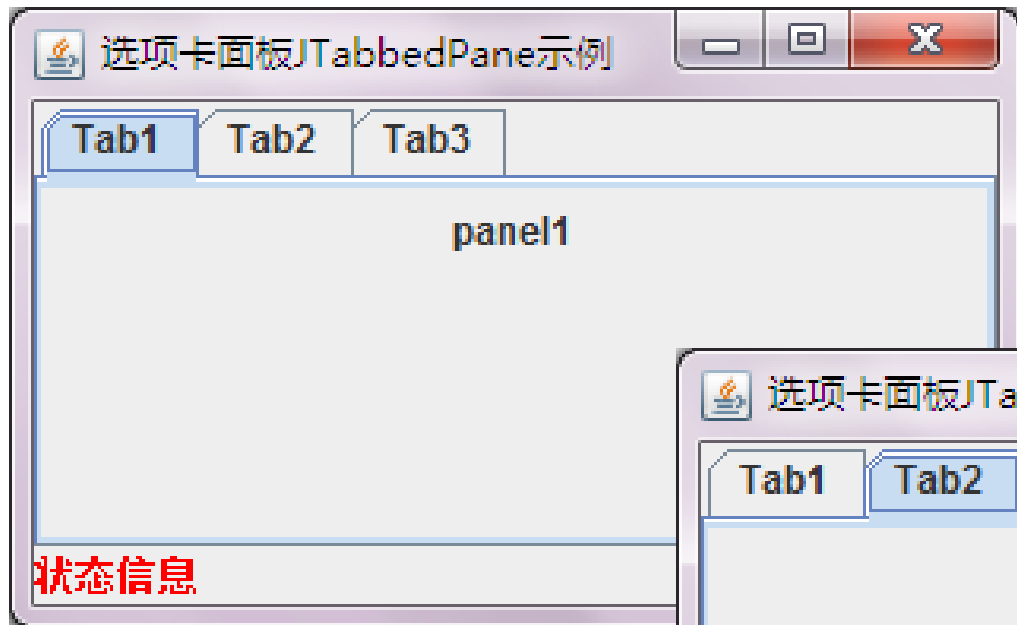
void stateChanged(ChangeEvent e)

- 多数情况下，选项卡面板仅用作为容器，一般无需响应用户的操作，因此不必对它进行事件处理



Swing高级组件: JTabbedPane

【例8-37】选项卡面板示例。



```
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];
        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");  
this.setVisible(true);
```

```
}  
public void stateChanged(ChangeEvent e){  
    JTabbedPane tp=(JTabbedPane) e.getSource();  
    JPanel c=(JPanel) tp.getSelectedComponent();  
    if(c==panels[0])  
        lblInfo.setText("Panel1 Selected!");  
    else if(c==panels[1])  
        lblInfo.setText("Panel2 Selected!");  
    else if(c==panels[2])  
        lblInfo.setText("Panel3 Selected!");  
}
```

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



小 结

1. GUI编程的Swing组件：顶层容器、中间层容器、原子组件
2. 布局管理器用于简化容器中组件的布局，使软件界面看上去更专业更美观
3. 事件处理方法
 - a) 为组件注册事件监听器
 - b) 定义或实现事件监听器或事件适配器，其中包含了响应事件的处理方法
4. Swing组件（包括相关的类、接口）提供了大量的方法，详细的用法需要查阅对应的API文档



习 题

1. P251 习题8.1~8.3大题

2. 上机:

- a) 编写一个简易记事本，界面要求使用菜单栏和工具栏。包括以下功能：
- 文本文件的新建、打开和保存，其中打开和保存文件时要求使用文件选择器
 - 编辑功能：复制、粘贴、剪切、清空、全选
 - 字体颜色设置：要求使用颜色选择器
 - 字体样式设置：正常、粗体、倾斜
 - 字体名称设置：宋体、黑体、幼圆
 - 关于记事本的信息：要求使用一个对话框
 - 设计一个登录面对话框，模拟用户登录（在记事本显示后，弹出该对话框，模拟验证用户合法性，非法用户重试3此后退出系统）
- b) 习题8.4.1，记录数据可以存放在一个对象数组中



谢谢大家!