

# JAVA程序设计

潘微科

感谢：教材《Java大学实用教程》的作者和其他老师提供PowerPoint讲义等资料！  
说明：本课程所使用的所有讲义，都是在以上资料上修改的。

# Outline

- 10.1 AWT组件与Swing组件概述
- 10.2 JFrame窗体
- 10.3 菜单组件
- 10.4 布局设计
- 10.5 中间容器
- 10.6 文本组件
- 10.7 按钮与标签组件
- 10.8 复选框与单选按钮组件
- 10.9 列表组件
- 10.10 表格组件
- 10.11 树组件
- 10.12 进度条组件
- 10.13 组件常用方法
- 10.14 窗口事件
- 10.15 鼠标事件
- 10.16 焦点事件
- 10.17 键盘事件
- 10.18 AWT线程
- 10.19 计时器
- 10.20 MVC设计模式
- 10.21 播放音频
- 10.22 按钮绑定到键盘
- 10.23 对话框
- 10.24 多文档界面
- 10.25 发布应用程序

# 补充

- Java GUI (Graphical User Interfaces) programming
  1. GUI containers (容器): to **hold** and group components
  2. GUI components (组件): to **create** user interfaces
  3. GUI helpers: to **support** GUI components
    - E.g., GUI layout managers (布局): to **arrange** components in a container

# 补充

- AWT (Abstract Windows Toolkit)
  - Fine for developing **simple** GUI, but not for **comprehensive** GUI project
  - **Prone** to platform-specific bugs
  - AWT (GUI) components were **REPLACED (被替换)** by Swing (GUI) components
- Swing components are painted directly on canvases (画布) using Java code, except for components that are subclasses of **java.awt.Window** or **java.awt.Panel**, which must be drawn using **native GUI** on a specific platform.

# 补充

- **Swing GUI containers (容器)**

- java.awt.Container

顶层/底层容器

- javax.swing.JFrame: a top-level container

- javax.swing.JApplet: a top-level container

- javax.swing.JDialog: a top-level container

default: BorderLayout

- javax.swing.JPanel: not a top-level container

default: FlowLayout

中间容器

heavyweight  
lightweight

# 补充

- **Swing GUI components (组件)**

- javax.swing.JMenuBar
- javax.swing.JMenu
- javax.swing.JMenuItem
- javax.swing.JCheckBoxMenuItem
- javax.swing.JRadioButtonMenuItem
- javax.swing.JPopupMenu

} 菜单

- javax.swing.JTextField
- javax.swing.JPasswordField
- javax.swing.JTextArea

} 文本

注：本slide中的内容为教材中的10.3和10.6

# 补充

- **Swing GUI components (组件)**
  - javax.swing.JButton          按钮
  - javax.swing.JCheckBox      复选框
  - javax.swing.JRadioButton   单选按钮
  - javax.swing.JLabel          标签
  - javax.swing.JComboBox      下拉列表
  - javax.swing.JTable          表格
  - javax.swing.JTree          树
  - javax.swing.JProgressBar   进度条

# 补充

- **Swing GUI components (组件)**
  - `javax.swing.JComponent`



# 补充

- **helper**

- java.awt.Graphics
- java.awt.Color
- java.awt.Font
- java.awt.FontMetrics
- java.awt.Dimension
- java.awt.LayoutManager

布局:10.4

- FlowLayout, BorderLayout, GridLayout, CardLayout, BoxLayout, Null

# Outline

- 10.1 AWT组件与Swing组件概述
- 10.2 JFrame窗体
- 10.3 菜单组件
- 10.4 布局设计
- 10.5 中间容器
- 10.6 文本组件
- 10.7 按钮与标签组件
- 10.8 复选框与单选按钮组件
- 10.9 列表组件
- 10.10 表格组件
- 10.11 树组件
- 10.12 进度条组件
- 10.13 组件常用方法
- 10.14 窗口事件
- 10.15 鼠标事件
- 10.16 焦点事件
- 10.17 键盘事件
- 10.18 AWT线程
- 10.19 计时器
- 10.20 MVC设计模式
- 10.21 播放音频
- 10.22 按钮绑定到键盘
- 10.23 对话框
- 10.24 多文档界面
- 10.25 发布应用程序


## Outline


- **10.1 AWT组件与Swing组件概述**
- 10.2 JFrame窗体
- 10.3 菜单组件
- 10.4 布局设计
- 10.5 中间容器
- 10.6 文本组件
- 10.7 按钮与标签组件
- 10.8 复选框与单选按钮组件
- 10.9 列表组件
- 10.10 表格组件
- 10.11 树组件
- 10.12 进度条组件
- 10.13 组件常用方法
- 10.14 窗口事件
- 10.15 鼠标事件
- 10.16 焦点事件
- 10.17 键盘事件
- 10.18 AWT线程
- 10.19 计时器
- 10.20 MVC设计模式
- 10.21 播放音频
- 10.22 按钮绑定到键盘
- 10.23 对话框
- 10.24 多文档界面
- 10.25 发布应用程序

- Java早期进行用户界面设计时，使用java.awt包（package）中提供的类。
- AWT是Abstract Window Toolkit（抽象窗口工具包）的缩写。
- java.awt包中的类创建的组件习惯上称为**重组件（heavyweight components）**。例如，当用java.awt包中的Button类创建一个按钮组件时，都有一个相应的**本地组件**在它工作，即**显示组件和处理组件事件**，该本地组件称为它的**同位体**。

- Java 2（JDK 1.2）推出之后，增加了一个新的**javax.swing**包（package），该包提供了功能更为强大的用来设计GUI界面的类。
- javax.swing包为我们提供了更加丰富、功能强大的组件，称为**Swing组件**，其中**大部分**组件是**轻组件（lightweight components）**，**没有同位体**，而是把**与显示组件有关的许多工作**和**处理组件事件的工作**交给相应的**UI代表**来完成。
- 这些**UI代表**是用**Java语言编写的类**，这些类被增加到Java的运行环境中，因此组件的外观**不依赖平台**，不仅在不同平台上的**外观**是相同的，而且与重组件相比**有更高的性能**。
- 如果Java运行环境低于1.2版本，就不能运行含有Swing组件的程序。

## 10.1 AWT组件与SWING组件概述

- Component - Container
  - JComponent
    - JButton
    - JTextField
    - JTextArea
    - JTree
    - JTable
    - JPanel

轻组件
  - Window
    - Frame - JFrame
    - Dialog - JDialog

重组件

- Java把由**Component**类的子类或间接子类创建的对象称为**组件**；把由**Container**类的子类或间接子类创建的对象称为**容器**。
  - 可以向容器添加组件。**Container**类提供了一个public方法**add()**，一个容器可以调用这个方法将组件添加到该容器中。
  - 调用**removeAll()**方法可以移掉容器中的全部组件，调用**remove(Component com)**方法可以移掉容器中参数指定的组件。
  - 每当容器添加新的组件或移掉组件时，应该让容器调用**validate()**方法，以保证容器中的组件能正确显示出来。
- 容器本身也是一个组件，因此可以把一个容器添加到另一个容器中实现容器的**嵌套**。

- javax.swing包中有4个最重要的类JComponent, JFrame, JApplet和JDialog。
  - **JComponent类的子类都是轻组件**，JComponent类是java.awt包中Container类的子类，因此**所有轻组件也都是容器**。
  - **JFrame, JApplet, JDialog都是重组件**，即它们是有同位体的组件。这样，窗体（JFrame）、小应用程序（Java Applet）、对话框（JDialog）可以和**操作系统**交互信息。轻组件必须要在这些容器中绘制自己，习惯上称这些容器为Swing的**顶层/底层容器**。



## Outline

- 10.1 AWT组件与Swing组件概述
- **10.2 JFrame窗体**
- 10.3 菜单组件
- 10.4 布局设计
- **10.5 中间容器**
- 10.6 文本组件
- 10.7 按钮与标签组件
- 10.8 复选框与单选按钮组件
- 10.9 列表组件
- 10.10 表格组件
- 10.11 树组件
- 10.12 进度条组件
- 10.13 组件常用方法
- 10.14 窗口事件
- 10.15 鼠标事件
- 10.16 焦点事件
- 10.17 键盘事件
- 10.18 AWT线程
- 10.19 计时器
- 10.20 MVC设计模式
- 10.21 播放音频
- 10.22 按钮绑定到键盘
- **10.23 对话框**
- **10.24 多文档界面**
- 10.25 发布应用程序

- javax.swing包中的JFrame类是java.awt包中Frame类的子类。
- JFrame类的常用方法
  - JFrame(): 创建一个无标题的窗体。
  - JFrame(String s): 创建一个标题为s的窗体。
  - public void setBounds(int a, int b, int width, int height): 设置出现在屏幕上时的初始位置为(a, b)，即距屏幕左面a个像素、距屏幕上方b个像素；窗体的宽是width，高是height。
  - public void setSize(int width, int height): 设置窗体的大小，窗体在屏幕出现时默认位置是(0, 0)。
  - public void setVisible(boolean b): 设置窗体是可见还是不可见，窗体默认是不可见的。

- JDK 1.4或之前的版本要求如下：
    - 不可以把组件直接添加到JFrame窗体中。
    - JFrame窗体含有一个称为**内容窗格（content pane，又称内容面板）**的容器，应当把组件添加到内容窗格中（内容窗格也是重容器）。
    - **不能为JFrame窗体设置布局**，而应当为JFrame窗体的**内容窗格**设置布局。**内容窗格**的默认布局是BorderLayout布局。
    - JFrame窗体通过调用方法getContentPane()方法得到它的**内容窗格**。
- 

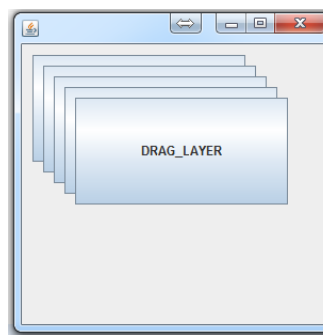
- 1.JPanel面板
- 我们会经常使用JPanel创建一个面板，再向这个面板添加组件，然后把这个面板添加到底层容器或其他中间容器中。
- JPanel面板的默认布局是FlowLayout布局。
- 可以使用JPanel类构造方法JPanel()构造一个面板容器对象。

- 2.JScrollPane滚动窗格
- 我们可以把一个组件放到一个滚动窗格中，然后通过滚动条来观察这个组件。
- 例如，JTextArea不自带滚动条，因此我们就需要把文本区放到一个滚动窗格中。可以使用JScrollPane的构造方法JScrollPane(component com)构造一个滚动窗格。

- **3.JSplitPane**拆分窗格
- 拆分窗格是被分成两部分的容器。拆分窗格有两种类型：水平拆分和垂直拆分。
- 水平拆分窗格用一条拆分线把容器分成左右两部分，左面放一个组件，右面放一个组件，拆分线可以水平移动。
- 垂直拆分窗格由一条拆分线分成上下两部分，上面放一个组件，下面放一个组件，拆分线可以垂直移动。
- 可以使用JSplitPane的构造方法JSplitPane(int a, Component b, Component c)构造一个拆分窗格，参数a取JSplitPane的**静态常量****HORIZONTAL\_SPLIT**或**VERTICAL\_SPLIT**，以决定是水平拆分还是垂直拆分。后两个参数决定要放置的组件。
- 拆分窗格调用setDividerLocation(double position)设置拆分线的位置。

- 4. **JLayeredPane** 分层窗格
- 如果添加到容器中的组件经常需要处理**重叠**问题，就可以考虑将组件添加到JLayeredPane容器。
- JLayeredPane容器将容器分为5层，容器使用add(JComponent com, int **layer**)添加组件com，并指定com所在的层，其中参数layer取值JLayeredPane类中的**类常量**：**DEFAULT\_LAYER**、**PALETTE\_LAYER**、**MODAL\_LAYER**、**POPUP\_LAYER**、**DRAG\_LAYER**。
  - **DEFAULT\_LAYER**是最底层，添加到**DEFAULT\_LAYER**层的组件如果和其它层的组件发生重叠，将被其它组件遮挡。
  - **DRAG\_LAYER**层是最上面的层，如果JLayeredPane中添加了许多组件，当你用鼠标移动一组件时，可以把移动的组件放到**DRAG\_LAYER**层，这样，组件在移动过程中，就不会被其它组件遮挡。
  - 添加到同一层上的组件，如果发生重叠，后添加的会遮挡先前添加的组件。
  - JLayeredPane对象调用public void setLayer(Component com, int layer)可以重新设置组件com所在的层，调用public int getLayer(Component com)可以获取组件com所在的层数。

## 10.5 中间容器



```
import javax.swing.*;
import java.awt.*;
public class Example10_6
{
    public static void main(String args[])
    {
        new WindowLayered();
    }
}
```

### 【例子6: Example10\_6.java】

- 我们在JLayeredPane容器中**添加5个组件**，分别位于不同的层上。

```
class WindowLayered extends JFrame
{
```

```
    WindowLayered()
```

```
    {
```

```
        // ---
```

```
        setBounds(100,100,300,300);
```

```
        setVisible(true);
```

```
        // ---
```

```
        JButton b1 = new JButton("DEFAULT_LAYER");
```

```
        JButton b2 = new JButton("PALETTE_LAYER");
```

```
        JButton b3 = new JButton("MODAL_LAYER");
```

```
        JButton b4 = new JButton("POPUP_LAYER");
```

```
        JButton b5 = new JButton("DRAG_LAYER");
```

```
        // ---
```

```
        b5.setBounds(50,50,200,100);
```

```
        b4.setBounds(40,40,200,100);
```

```
        b3.setBounds(30,30,200,100);
```

```
        b2.setBounds(20,20,200,100);
```

```
        b1.setBounds(10,10,200,100);
```

```
        // ---
```

```
        JLayeredPane pane = new JLayeredPane();
```

```
        pane.setLayout(null);
```

```
        // ---
```

```
        pane.add(b5, JLayeredPane.DRAG_LAYER);
```

```
        pane.add(b4, JLayeredPane.POPUP_LAYER);
```

```
        pane.add(b3, JLayeredPane.MODAL_LAYER);
```

```
        pane.add(b2, JLayeredPane.PALETTE_LAYER);
```

```
        pane.add(b1, JLayeredPane.DEFAULT_LAYER);
```

```
        // ---
```

```
        add(pane, BorderLayout.CENTER);
```

```
        // ---
```

```
        validate();
```

```
        // ---
```

```
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
```

```
    }
```

```
}
```



- 1. **JDialog**类
- JDialog类和JFrame类都是Window类的子类，二者有相似之处也有不同的地方，比如**对话框必须要依赖于某个窗口或组件**，当它所依赖的窗口或组件消失时，对话框也将消失；而当它所依赖的窗口或组件可见时，对话框又会自动恢复。

- 2.对话框的模式
- 对话框分为**无模式（modaless）**和**有模式（modal）**两种。
- **无模式对话框**处于激活状态时，程序仍能激活它所依赖的窗口或组件，它也不**堵塞**线程的执行。
- **有模式对话框**处于激活状态时，只让程序响应对话框内部的事件，程序不能再激活它所依赖的窗口或组件，而且它将**堵塞**当前线程的执行，直到该对话框消失不可见。

## 【例子30: Example10\_30.java】

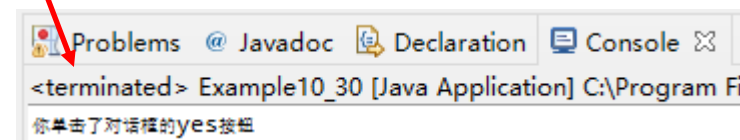
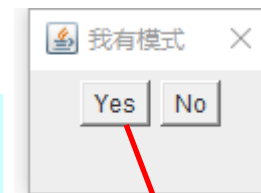
- 当对话框处于激活状态时，**命令行无法输出信息**，当对话框消失时，再根据对话框消失的原因，命令行输出信息：“Button: Yes”或“Button: No”。

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

public class Example10_30
{
    public static void main(String args[])
    {
        MyDialog dialog = new MyDialog(null, "我有模式", true);

        dialog.setVisible(true);

        if(dialog.getMessage() == MyDialog.YES)
        {
            System.out.println("你单击了对话框的yes按钮");
        }
    }
}
```



```
        if(dialog.getMessage() == MyDialog.NO)
        {
            System.out.println("你单击了对话框的No按钮");
        }
        else
        {
            if(dialog.getMessage() == MyDialog.CLOSE)
            {
                System.out.println("你单击了对话框的关闭图标");
            }
        }
    }
    System.exit(0);
}
```

## 10.23 对话框

```
class MyDialog extends JDialog implements ActionListener
{
    static final int YES=1, NO=0, CLOSE=-1;
    int message=10;
    Button yes,no;

    MyDialog(JFrame f, String s, boolean b)
    {
        super(f,s,b);

        // ---
        setLayout(new FlowLayout());
        setBounds(60,60,100,100);

        // ---
        yes = new Button("Yes");
        yes.addActionListener(this);

        // ---
        no = new Button("No");
        no.addActionListener(this);

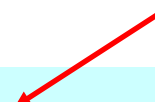
        // ---
        add(yes);

        // ---
        add(no);

        // --- 匿名类
        addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                message = CLOSE;
                setVisible(false);
            }
        });
    }
}
```

```
// ---
public void actionPerformed(ActionEvent e)
{
    if(e.getSource()==yes) // 事件源
    {
        message = YES;
        setVisible(false);
    }
    else if(e.getSource()==no)
    {
        message = NO;
        setVisible(false);
    }
}

// ---
public int getMessage()
{
    return message;
}
}
```



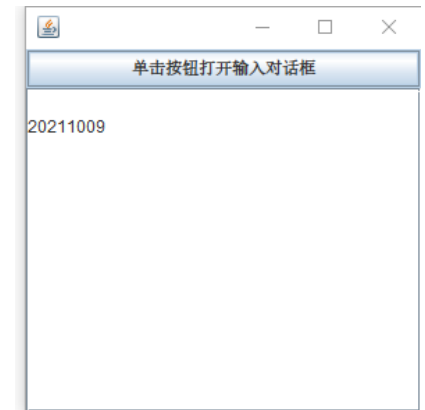
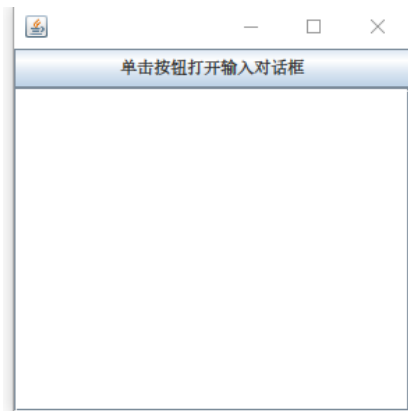
- 3.输入对话框
- javax.swing包中的JOptionPane类的静态方法public static String showInputDialog(Component parentComponent, Object message, String title, int messageType)可以创建一个输入对话框。参数分别是对话框所依赖的组件、对话框上显示的消息、对话框的标题和对话框的外观。

- 4.消息对话框
- javax.swing包中的JOptionPane类的静态方法public static void show**MessageDialog**(Component parentComponent, String message, String title, int messageType)可以创建一个**消息对话框**。参数分别是对话框所依赖的组件、对话框上显示的消息、对话框的标题和对话框的外观。

- 5.确认对话框
- 确认对话框是**有模式对话框**，可以用javax.swing包中的**JOptionPane**类的静态方法`public static int showConfirmDialog(Component parentComponent, Object message, String title, int optionType)`创建一个**确认对话框**。参数分别是对话框所依赖的组件、对话框上显示的消息、对话框的标题和对话框的外观。

### 【例子31: Example10\_31.java】

- 用户在输入对话框中输入数字字符，如果输入的字符中有**非数字字符**，将弹出一个消息对话框，提示用户输入了非法字符，该消息对话框消失后，将清除用户输入的非法字符；如果用户的输入没有非法字符，将弹出一个确认对话框，让用户确认，如果单击确认对话框上的“是(Y)”按钮，就把数字放入文本区。





## 10.23 对话框

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

public class Example10_31
{
    public static void main(String args[])
    {
        new Dwindow();
    }
}

class Dwindow extends JFrame implements ActionListener
{
    JButton inputNumber;
    JTextArea save;
    Pattern p; //模式对象
    Matcher m; //匹配对象

    Dwindow()
    {
        // ---
        inputNumber = new JButton("单击按钮打开输入对话框");
        inputNumber.addActionListener(this);
        add(inputNumber, BorderLayout.NORTH);


        // ---
        save = new JTextArea(12,16);
        add(new JScrollPane(save), BorderLayout.CENTER);

        // ---
        setBounds(60,60,300,300);
        setVisible(true);

        // ---
        p = Pattern.compile("\\D+"); //创建模式对象(含有非数字字符的模式)

        // ---
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

## 10.23 对话框



```
public void actionPerformed(ActionEvent e)
{
    String str = JOptionPane.showInputDialog(null, "请输入数字字符序列", "输入对话框",
JOptionPane.INFORMATION_MESSAGE);

    if(str!=null)
    {
        m = p.matcher(str);
        while(m.find())
        {
            JOptionPane.showMessageDialog(this, "您输入了非法字符", "消息对话框", JOptionPane.WARNING_MESSAGE);
            str = JOptionPane.showInputDialog(null, "请输入数字字符序列");
            m = p.matcher(str);
        }

        // ---
        int n = JOptionPane.showConfirmDialog(this, "确认正确吗?", "确认对话框", JOptionPane.YES_NO_OPTION);
        if(n == JOptionPane.YES_OPTION)
        {
            save.append("\n"+str);
        }
    }
}
```

- 6.颜色对话框
- 可以用javax.swing包中的JColorChooser类的静态方法public static Color showDialog(Component com, String title, Color initialColor)创建一个颜色对话框，其中参数com指定对话框所依赖的组件，title指定对话框的标题，initialColor 指定对话框返回的初始颜色，即对话框消失后返回的默认值。
- 颜色对话框可根据用户在颜色对话框中选择的颜色返回一个颜色对象。

### 【例子32: Example10\_32.java】

- 当用户单击buttonOpen按钮时，弹出一个颜色对话框，然后根据用户选择的颜色来改变按钮showColor的颜色。

## 10.23 对话框

```
class ColorWin extends JFrame implements ActionListener
```

```
{
```

```
    JButton buttonOpen, showColor;
```

```
    ColorWin(String s)
```

```
    {
```

```
        // ---
```

```
        setTitle(s);
```

```
        // ---
```

```
        buttonOpen = new JButton("打开颜色对话框");
```

```
        buttonOpen.addActionListener(this);
```

```
        add(buttonOpen, BorderLayout.NORTH);
```

```
        // ---
```

```
        showColor = new JButton();
```

```
        add(showColor, BorderLayout.CENTER);
```

```
        // ---
```

```
        setBounds(60, 60, 300, 300);
```

```
        setVisible(true);
```

```
        // ---
```

```
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
    }
```

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

```
    {
```

```
        Color newColor = JColorChooser.showDialog(this, "调色板", showColor.getBackground());
```

```
        if(newColor != null)
```

```
        {
```

```
            showColor.setBackground(newColor);
```

```
        }
```

```
    }
```

```
}
```

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

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

```
import javax.swing.*;
```

```
public class Example10_32
```

```
{
```

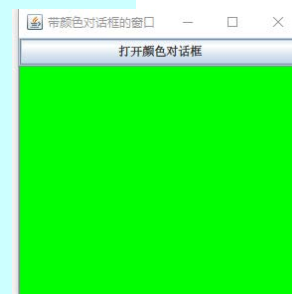
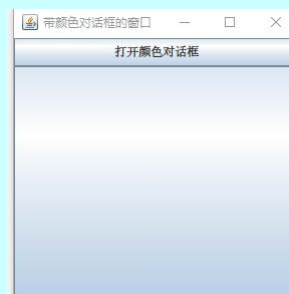
```
    public static void main(String args[])
```

```
    {
```

```
        new ColorWin("带颜色对话框的窗口");
```

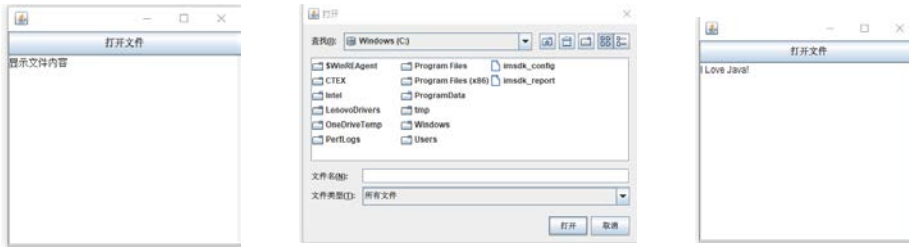
```
    }
```

```
}
```



- 7.文件对话框
- 文件对话框提供从文件系统中进行文件选择的界面。
- **JFileChooser**对象调用下列方法可以使得一个有模式对话框显示在桌面上，该对话框称作文件对话框，文件对话框将在参数指定的组件 `parentComponent` 的正前方显示，如果 `parentComponent` 为 `null`，则在系统桌面的正前方显示。
  - `showDialog(Component parentComponent, String s)`
  - `showOpenDialog(Component parentComponent)`
  - `showSaveDialog(Component parentComponent)`
- 当文件对话框消失后，上述方法返回下面的整型常量之一，返回的值依赖于单击了对话框上的“确认”按钮还是“取消”按钮。
  - `JFileChooser.APPROVE_OPTION`
  - `JFileChooser.CANCEL_OPTION`

## 10.23 对话框



```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import java.io.*;
public class Example10_33
{
    public static void main(String args[])
    {
        new FileWindow();
    }
}
```

### 【例子33: Example10\_33.java】

- 当用户单击“打开文件”按钮，将弹出一个文件对话框，用户可以把选择的文件的内容显示在一个文本区中。

```
class FileWindow extends JFrame implements ActionListener
{
    JButton buttonFile;
    JTextArea text;
    JFileChooser fileChooser;
    FileWindow()
    {
        fileChooser = new JFileChooser("c:/");

        // ---
        buttonFile = new JButton("打开文件");
        buttonFile.addActionListener(this);
        add(buttonFile, BorderLayout.NORTH);

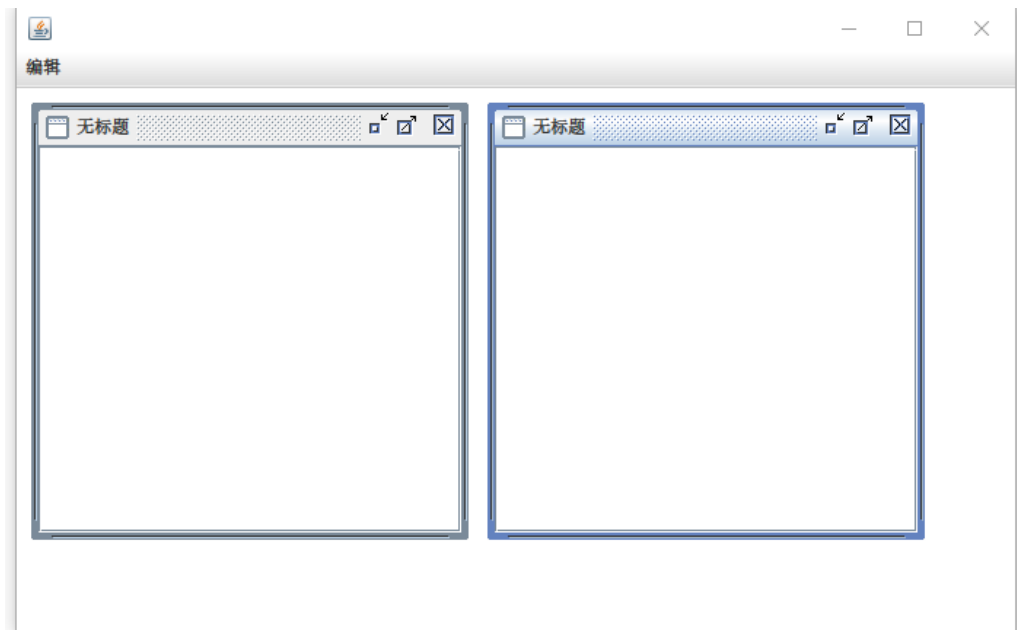
        // ---
        text = new JTextArea("显示文件内容");
        add(new JScrollPane(text), BorderLayout.CENTER);
        // ---
        setBounds(60,60,300,300);
        setVisible(true);
        // ---
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

```
public void actionPerformed(ActionEvent e)
{
    text.setText(null);
    int n = fileChooser.showOpenDialog(null);
    if(n == JFileChooser.APPROVE_OPTION)
    {
        File file = fileChooser.getSelectedFile();
        // ---
        try
        {
            FileReader readfile = new FileReader(file);
            BufferedReader in = new
BufferedReader(readfile);
            String s = null;
            while( (s=in.readLine()) != null )
            {
                text.append(s+"\n");
            }
        }
        catch(IOException ee){}
    }
}
```

- Java实现多文档界面（MDI）常用的方式是在一个JFrame窗体中添加若干个**内部窗体**，内部窗体由J**InternalFrame**类负责创建。这些内部窗体被限制在JFrame窗体中。
- 在使用内部窗体时，需要将**内部窗体**事先添加到J**DesktopPane****桌面窗格中**，一个桌面窗格可以添加若干个**内部窗体**，这些内部窗体将被限制在该桌面窗格中，然后把桌面窗格添加到JFrame窗体即可。
- 桌面窗格使用方法add(JInternalFrame e, int layer)添加内部窗体，并指定内部窗体所在的层次。

### 【例子34: Example10\_34.java】

- 单击JFrame中的“新建”菜单，在窗体中出现一个新的内部窗体，该内部窗体中有一个文本区对象；当单击JFrame中的“复制”菜单时，就将处于活动状态的内部窗体里面的文本区选中的内容复制到系统的剪贴板；当单击JFrame中的“粘贴”菜单时，就将系统剪贴板中的文本内容粘贴到处于活动状态的内部窗体的文本区中。





## Outline

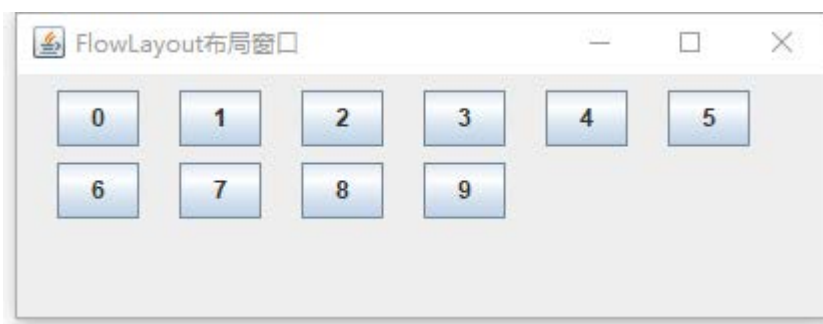
- 10.1 AWT组件与Swing组件概述
- 10.2 JFrame窗体
- 10.3 菜单组件
- **10.4 布局设计**
- 10.5 中间容器
- 10.6 文本组件
- 10.7 按钮与标签组件
- 10.8 复选框与单选按钮组件
- 10.9 列表组件
- 10.10 表格组件
- 10.11 树组件
- 10.12 进度条组件
- 10.13 组件常用方法
- **10.14 窗口事件**
- **10.15 鼠标事件**
- **10.16 焦点事件**
- **10.17 键盘事件**
- 10.18 AWT线程
- 10.19 计时器
- 10.20 MVC设计模式
- 10.21 播放音频
- 10.22 按钮绑定到键盘
- 10.23 对话框
- 10.24 多文档界面
- 10.25 发布应用程序

## 10.4 布局设计

- 希望控制组件在容器中的位置，这就需要学习**布局设计**的知识。我们将分别介绍**java.awt**包中的FlowLayout、BorderLayout、CardLayout、GridLayout布局类和**java.swing.border**包中的BoxLayout布局类。
- 对于JFrame窗体，默认布局是BorderLayout布局。
- 容器可以使用方法**setLayout(布局对象);**来设置自己的布局。

- 1. **FlowLayout**布局
- FlowLayout类创建的对象称做FlowLayout型布局。FlowLayout类的一个常用构造方法如下：
  - FlowLayout()
- 该构造方法可以创建一个**居中对齐**的布局对象。例如：
  - FlowLayout flow = new FlowLayout();
- 如果一个容器con使用这个布局对象，即con.setLayout(flow); 那么，con可以使用Container类提供的add方法将组件**顺序**地添加到容器中，组件按照加入的先后顺序从左向右排列，一行排满之后就转到下一行继续从左至右排列。
- FlowLayout布局对象调用setHgap(int hgap)方法和setVgap(int vgap)方法可以设置布局的**水平间隙**和**垂直间隙**。

## 10.4 布局设计



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

public class Example10_2
{
    public static void main(String args[])
    {
        new WindowFlow("FlowLayout布局窗口");
    }
}
```

### 【例子2: Example10\_2.java】

- JFrame使用FlowLayout布局放置10个组件。

```
class WindowFlow extends JFrame
{
```

```
    JButton b[];
    WindowFlow(String s)
```

```
    {
        // ---
        setTitle(s);
```

```
        // ---
        b = new JButton[10];
```

```
        // ---
        FlowLayout flow = new FlowLayout();
        flow.setAlignment(FlowLayout.LEFT);
        flow.setHgap(20);
        flow.setVgap(8);
        this.setLayout(flow);
```

```
        // ---
        for(int i=0; i<b.length; i++)
        {
            b[i] = new JButton("" + i);
            this.add(b[i]);
```

```
        // ---
        validate();
```

```
        // ---
        setBounds(100, 100, 200, 160);
        setVisible(true);
```

```
        // ---
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
```

```
    }
}
```

- 2. **BorderLayout** 布局
- BorderLayout 布局是 Window 型容器的 **默认布局**，例如 **JFrame** 和 **JDialog** 都是 **Window** 类的间接子类。
- 如果容器使用 BorderLayout 布局，那么容器空间简单地划分为 **东、西、南、北、中五个区域**。每加入一个组件都应该指明把这个组件添加在哪个区域中，区域由 BorderLayout 中的静态常量 EAST, WEST, SOUTH, NORTH, CENTER 表示。
- 添加到某个区域的组件将占据这个区域。 **每个区域只能放置一个组件**，如果向某个已放置了组件的区域再放置一个组件， **那么先前的组件将被后者替换掉**。

### 【例子3: Example10\_3.java】

- 使用BorderLayout布局。

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

public class Example10_3
{
    public static void main(String args[])
    {
        // ---
        JFrame win = new JFrame("窗体");

        // ---
        JButton bSouth = new JButton("南");
        JButton bNorth = new JButton("北");
        JButton bEast = new JButton("东");
        JButton bWest = new JButton("西");
        JTextArea bCenter = new JTextArea("中心");

        // ---
        win.add(bNorth, BorderLayout.NORTH);
        win.add(bSouth, BorderLayout.SOUTH);
        win.add(bEast, BorderLayout.EAST);
        win.add(bWest, BorderLayout.WEST);
        win.add(bCenter, BorderLayout.CENTER);
    }
}
```



```
// ---
win.validate();

// ---
win.setBounds(100,100,300,300);
win.setVisible(true);

// ---
win.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

- 3.**CardLayout**布局
- 使用**CardLayout**的容器可以容纳多个组件，但是实际上同一时刻容器只能从这些组件中选出一个来显示，这个被显示的组件将占据所有的容器空间。
- **JTabbedPane**创建的对象是一个轻容器，称作**选项卡窗格**。选项卡窗格的默认布局是**CardLayout**布局。
- 选项卡窗格可以使用**`add(String text, Component com);`**方法将组件**com**添加到容器当中，并指定和该组件**com**对应的选项卡的文本提示是**text**。

### 【例子4: **Example10\_4.java**】

- 我们在选项卡窗格中添加了5个按钮，并设置了相对应的选项卡的文本提示，然后将选项卡窗格添加到窗体中。

## 10.4 布局设计

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

public class Example10_4
{
    public static void main(String args[])
    {
        new MyWin();
    }
}
```

```
class MyWin extends JFrame
```

```
{
```

```
    JTabbedPane p;
```

```
    Icon icon[];
```

```
    String imageName[] = {"a.jpg", "b.jpg", "c.jpg", "d.jpg", "e.jpg"};
```

```
    public MyWin()
```

```
    {
```

```
        // ---
```

```
        icon = new Icon[imageName.length];
```

```
        for(int i=0; i<icon.length; i++)
```

```
        {
```

```
            icon[i] = new ImageIcon(imageName[i]);
```

```
        }
```

```
        // ---
```

```
        p = new JTabbedPane(JTabbedPane.LEFT);
```

```
        for(int i=0; i<icon.length; i++)
```

```
        {
```

```
            int i2 = i + 1;
```

```
            p.add("观看第" + i2 + "个图片",
```

```
                new JButton(icon[i]));
```

```
        }
```

```
        // ---
```

```
        add(p, BorderLayout.CENTER);
```

```
        // ---
```

```
        p.validate();
```

```
        // ---
```

```
        validate();
```

```
        // ---
```

```
        setBounds(100, 100, 500, 300);
```

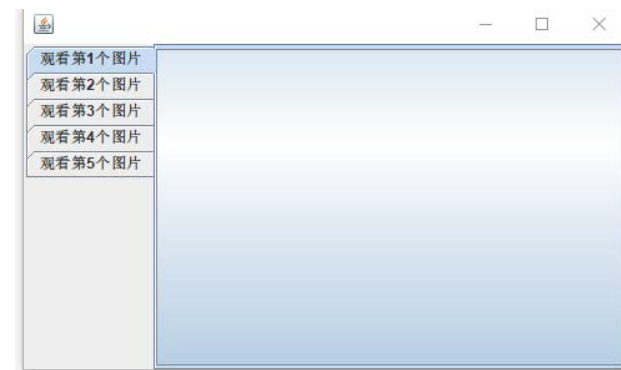
```
        setVisible(true);
```

```
        // ---
```

```
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
```

```
    }
```

```
}
```



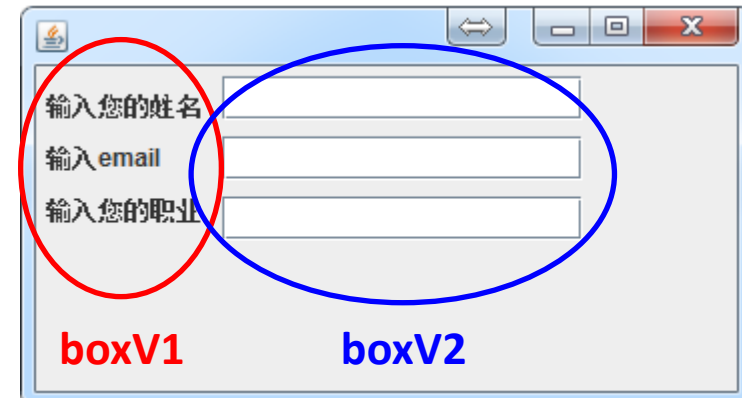


- 4. **GridLayout**布局
- GridLayout是使用较多的布局编辑器，其基本布局策略是把容器划分成若干行乘若干列的网格区域，组件就位于这些划分出来的小格中。
- (1) 使用GridLayout的构造方法GridLayout(int m, int n)创建布局对象，指定划分网格的行数m和列数n，例如：
  - GridLayout grid = new GridLayout(10,8);
- (2) 使用GridLayout布局的容器调用方法add将组件加入容器，组件进入容器的顺序将按照第一行第一个、第一行第二个...第一行最后一个、第二行第一个、...最后一行第一个...最后一行最后一个。

- 5. **BoxLayout**布局
- 用BoxLayout类可以创建一个布局对象，称为**盒式布局**。BoxLayout在java.swing.border包中。java swing包提供了Box类，该类也是Container类的一个子类，创建的容器称作一个盒式容器，盒式容器的默认布局是盒式布局，而且不允许更改盒式容器的布局。因此，在策划程序的布局时，可以利用容器的嵌套，在某个容器中嵌入几个盒式容器，达到布局的目的。使用**盒式布局的容器将组件排列在一行或一列**，这取决于创建盒式布局对象时，指定了行排列还是列排列。

- 在**行型盒式布局**容器中添加的组件的上沿在同一水平线上。在**列型盒式布局**容器中添加的组件的左沿在同一垂直线上。
- 使用Box类的静态方法 `createHorizontalBox()`可以获得一个具有**行型盒式布局**的盒式容器；使用Box类的静态方法 `createVerticalBox()`可以获得一个具有**列型盒式布局**的盒式容器。
- 如果想控制盒式布局容器中组件之间的距离，就需要使用水平支撑或垂直支撑。Box类调用静态方法`createHorizontalStrut(int width)`可以得到一个不可见的水平Strut类型对象，称做**水平支撑**。该水平支撑的宽度是width。Box类调用静态方法`createVerticalStrut(int height)`可以得到一个不可见的垂直Strut类型对象，称做**垂直支撑**。参数height决定垂直支撑的高度。

```
import javax.swing.*;
import java.awt.*;
public class Example10_5
{
    public static void main(String args[])
    {
        new WindowBox();
    }
}
```



## 【例子5: Example10\_5.java】

- 有两个列型盒式容器boxV1和boxV2，以及一个行型盒式容器baseBox。在列型盒式容器的组件之间添加垂直支撑，控制组件之间的距离，将boxV1、boxV2添加到baseBox中，并在它们之间添加了水平支撑。

```
class WindowBox extends JFrame
{
    Box baseBox, boxV1, boxV2;
    WindowBox()
    {
        // ---
        boxV1 = Box.createVerticalBox();
        boxV1.add(new JLabel("输入您的姓名"));
        boxV1.add(Box.createVerticalStrut(8));
        boxV1.add(new JLabel("输入email"));
        boxV1.add(Box.createVerticalStrut(8));
        boxV1.add(new JLabel("输入您的职业"));

        // ---
        boxV2 = Box.createVerticalBox();
        boxV2.add(new JTextField(16));
        boxV2.add(Box.createVerticalStrut(8));
        boxV2.add(new JTextField(16));
        boxV2.add(Box.createVerticalStrut(8));
        boxV2.add(new JTextField(16));

        // ---
        baseBox = Box.createHorizontalBox();
        baseBox.add(boxV1);
        baseBox.add(Box.createHorizontalStrut(10));
        baseBox.add(boxV2);

        // ---
        FlowLayout flow = new FlowLayout();
        flow.setAlignment(FlowLayout.LEFT);
        setLayout(flow);

        this.add(baseBox);
        validate();
        setBounds(120,125,200,200);
        setVisible(true);

        // ---
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }
}
```

- 6.**null**布局
- 我们可以把一个容器的布局设置为**null**布局（空布局）。空布局容器可以**准确地定位**组件在容器中的位置和大小。**setBounds(int a, int b, int width, int height)**方法是所有组件都拥有的一个方法，组件调用该方法可以设置组件本身的大小和在容器中的位置。  
例如，**p**是某个容器，**p.setBounds(null);**会把**p**的布局设置为**空布局**。
- 向空布局的容器**p**添加一个组件**com**需要两个步骤，首先使用**add(com)**方法向容器添加组件，然后组件**com**再调用**setBounds(int a, int b, int width, int height)**方法设置该组件在容器中的位置和本身的大小，组件都是一个**矩形结构**，方法中的参数**a,b**是被添加的组件**com**的左上角在容器中的位置坐标，即该组件距容器左面**a**个像素，距容器上方**b**个像素；**width**和**height**是组件**com**的宽和高。

## Outline

- 10.1 AWT组件与Swing组件概述
- 10.2 JFrame窗体
- **10.3 菜单组件**
- 10.4 布局设计
- 10.5 中间容器
- **10.6 文本组件**
- **10.7 按钮与标签组件**
- **10.8 复选框与单选按钮组件**
- **10.9 列表组件**
- **10.10 表格组件**
- **10.11 树组件**
- **10.12 进度条组件**
- **10.13 组件常用方法**
- 10.14 窗口事件
- 10.15 鼠标事件
- 10.16 焦点事件
- 10.17 键盘事件
- 10.18 AWT线程
- 10.19 计时器
- 10.20 MVC设计模式
- 10.21 播放音频
- 10.22 按钮绑定到键盘
- 10.23 对话框
- 10.24 多文档界面
- 10.25 发布应用程序

## 10.3 菜单组件

- 窗口中的菜单条（menu bar）、菜单（menu）、菜单项（menu item）是我们所熟悉的界面，**菜单**放在**菜单条**里，**菜单项**放在**菜单**里。
- 1.JMenuBar菜单条
- (1) JComponent类的子类JMenuBar是负责创建**菜单条**的，即JMenuBar的一个实例就是一个菜单条。JFrame类有一个将菜单条放置到窗口中的方法`public void setJMenuBar(JMenuBar menubar);`需要注意的是，只能向窗口添加一个菜单条。

- (2) JMenu菜单
- JComponent类的子类JMenu类是负责创建菜单的，JMenu类的主要方法有以下几种：
  - JMenu(String s): 建立一个指定标题的菜单，标题由参数s确定。
  - public void add(Menuitem item): 向菜单增加由参数item指定的菜单项对象。
  - public void add(String s): 向菜单增加指定的选项。
  - public JMenuitem getItem(int n): 得到指定索引处的菜单项。
  - public int getItemCount(): 得到菜单项数目。



- (3) JMenuItem菜单项
- **JMenuItem是JMenu的父类**，该类是负责创建**菜单项**的，即JMenuItem的一个实例就是一个菜单项。菜单项将被放在菜单里。JMenuItem类的主要方法有以下几种：
  - JMenuItem(String s): 构造有标题的菜单项。
  - JMenuItem(String s, Icon icon): 构造有标题和图标的菜单项。
  - public void setEnabled(boolean b): 设置当前菜单项是否可被选择。
  - public String getLabel(): 得到菜单项的名字。

- `public void setAccelerator(KeyStroke keyStroke)`: 为菜单项设置**快捷键**。
- 为了向该方法的参数传递一个KeyStroke对象，可以使用KeyStroke类的类方法：`public static KeyStroke getKeyStroke(char keyChar)`返回一个KeyStroke对象。
- 也可以使用KeyStroke类的静态方法`public static KeyStroke getKeyStroke(int keyCode, int modifiers)`返回一个KeyStroke对象
  - 参数keyCode取值范围：KeyEvent.VK\_A~KeyEvent.VK\_Z
  - 参数modifiers取值范围：InputEvent.**ALT**\_MASK, InputEvent.**CTRL**\_MASK和InputEvent.**SHIFT**\_MASK

- (4) 嵌入子菜单JMenu是JMenuItem的子类，因此菜单项本身也可以是一个菜单，称这样的菜单项为**子菜单**。为了使得菜单项有一个图标，可以用图标类Icon声明一个图标，然后使用其子类ImageIcon类创建一个图标，如：
  - Icon icon = new ImageIcon("open.gif");

### 【例子1: Example10\_1.java】

- 一个含有菜单的窗口。

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

public class Example10_1
{
    public static void main(String args[])
    {
        FirstWindow win = new FirstWindow("一个简单的窗口");
    }
}
```



## 10.3 菜单组件

```
class FirstWindow extends JFrame
{
    JMenuBar menubar;
    JMenu menu;
    JMenuItem item1,item2;

    FirstWindow(String s)
    {
        setTitle(s);

        // ---
        item1 = new JMenuItem("打开", new ImageIcon("open.gif"));
        item2 = new JMenuItem("保存", new ImageIcon("save.gif"));
        item1.setAccelerator(KeyStroke.getKeyStroke('O'));
        item2.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S, InputEvent.CTRL_MASK));

        // ---
        menu = new JMenu("文件");
        menu.add(item1);
        menu.addSeparator();
        menu.add(item2);

        // ---
        menubar = new JMenuBar();
        menubar.add(menu);
        setJMenuBar(menubar);

        // ---
        validate();

        // ---
        setSize(160,170);
        setLocation(120,120);
        setVisible(true);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }
}
```

Deprecated. It is recommended that **CTRL\_DOWN\_MASK** and `getModifiersEx()` be used instead.  
将**CTRL\_MASK**替换为**CTRL\_DOWN\_MASK**即可。



- 1.JTextField文本框
- JTextField创建的一个对象就是一个文本框。用户可以在文本框输入单行的文本。
- JTextField类的主要方法：
  - (1) JTextField(int x): 如果使用这个构造方法创建文本框对象，文本框的可见字符个数由参数x指定。
  - (2) JTextField(String s): 如果使用这个构造方法创建文本框对象，则文本框的初始字符串为s。

- (3) `public void setText(String s)`: 文本框对象调用该方法可以设置文本框中的文本为参数s指定的文本。
- (4) `public String getText()`: 文本框对象调用该方法可以获取文本框中的文本。
- (5) `public void setEditable(boolean b)`: 文本框对象调用该方法可以指定文本框的可编辑性。
- (6) `public void setHorizontalAlignment(int alignment)`: 设文本在文本框中的对齐方式，其中alignment的有效值确定对齐方式。

- 2.JPasswordField
- 密码框可以使用`setEchoChar(char c)`设置回显字符（默认的回显字符是‘\*’）；使用`char[] getPassword()`方法返回密码框中的密码。



- 3.ActionEvent事件
- 学习组件除了需要了解组件的属性和功能外，一个更重要的方面是学习怎样**处理**组件上发生的**界面事件**。当用户在有输入焦点的**文本框（JTextField）**中按回车键、单击按钮、在一个下拉式列表中选择一条**条目**等操作时，都会发生界面事件。程序有时需要对发生的事件作出反应，来实现特定的任务。
- 在学习处理事件时，必须很好地掌握**事件源**、**监视/监听器**和**处理事件的接口**这三个概念。
- (1) 事件源
- **能够产生事件的对象**都可以成为**事件源**，如文本框、按钮、下拉式列表等。也就是说，事件源必须是一个对象，而且这个对象必须是Java认为能够发生事件的对象。





- (2) 监视/监听器。
- 我们需要一个对象对事件源进行**监视/监听**，以便对发生的事件作出处理。事件源通过调用相应的方法将某个**对象**作为自己的监视器。
- 例如，对于文本框，这个方法是**addActionListener(ActionListener listener)**，对于获取了监视器的文本框对象，在文本框（JTextField）获得输入焦点之后，如果用户按回车键，**Java运行系统就自动用ActionEvent类创建一个对象**，即发生了**ActionEvent事件**。



- (3) 处理事件的接口。
- 注意到发生ActionEvent事件的事件源对象获得监视器的方法是：
  - **addActionListener**(ActionListener listener);
- 该方法中的参数是ActionListener类型的接口，因此必须将一个**实现ActionListener接口的类**创建的对象作为传递给该方法的参数，使得该对象成为事件源的监视/监听器。
- 监视/监听器负责调用特定的方法来处理事件，也就是说创建监视/监听器的类必须提供处理事件的特定方法，即实现接口中的方法。
- Java采用**接口回调技术**来处理事件，当事件源发生事件时，接口立刻通知监视/监听器自动调用实现的某个接口方法，该接口方法规定了怎样处理事件的操作。**接口回调这一过程对程序是不可见的，Java在设计组件事件时已经设置好了这一回调过程，程序只需让事件源获得正确的监视/监听器**，即将实现了正确接口的对象的引用传递给方法。

- (4) `ActionEvent`类中的方法
- `ActionEvent`事件对象调用方法`public Object getSource()`可以返回发生`ActionEvent`事件的对象的引用。



### 【例子7: Example10\_7.java】

- 窗口（窗体）中有一个文本框：text的事件监视器由PoliceStation类负责创建。当用户在text中输入字符串回车后，监视器负责在命令行窗口输出该字符串以及它的长度。

```
class MyWindow extends JFrame
{
    JTextField text;
    PoliceStation police;

    MyWindow()
    {
        setLayout(new FlowLayout());

        police = new PoliceStation();

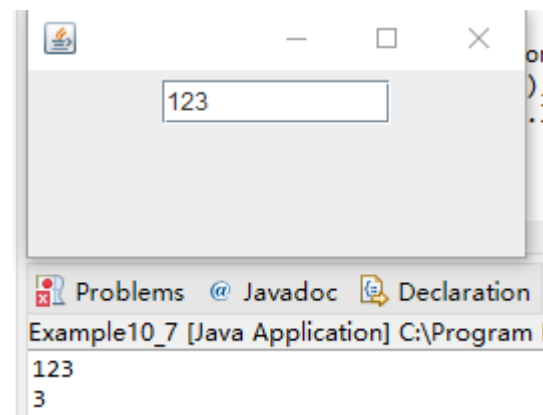
        text = new JTextField(10);
        add(text);
        text.addActionListener(police);
        //text是事件源，police是监视器

        setBounds(100,100,150,150);
        setVisible(true);
        validate();
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }
}
```

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

public class Example10_7
{
    public static void main(String args[])
    {
        MyWindow win = new MyWindow();
    }
}
```

```
class PoliceStation implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        String str = e.getActionCommand();
        System.out.println(str);
        System.out.println(str.length());
    }
}
```





## 【例子8: Example10\_8.java】

- titleText和passwordText有监视器。当在titleText中输入字符串回车后，监视器负责将窗体的标题更改为当前titleText中的文本。当在passwordText中输入密码回车后，监视器负责将密码显示在titleText中。

```
class PoliceWindow extends JFrame implements ActionListener
```

```
{
    JTextField titleText;
    JPasswordField passwordText;

    PoliceWindow()
    {
        titleText = new JTextField(10);
        add(titleText);
        titleText.addActionListener(this);

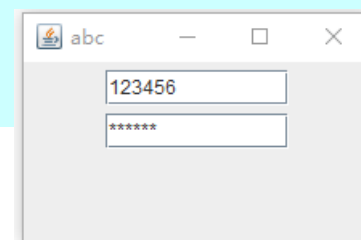
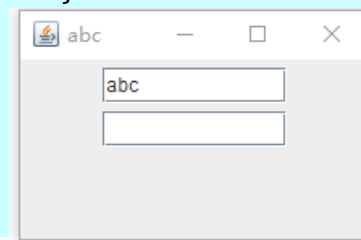
        passwordText = new JPasswordField(10);
        passwordText.setEchoChar('*');
        add(passwordText);
        passwordText.addActionListener(this);

        validate();
        setLayout(new FlowLayout());
        setBounds(100, 100, 150, 150);
        setVisible(true);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }
}
```

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

public class Example10_8
{
    public static void main(String args[])
    {
        PoliceWindow policeWin = new PoliceWindow();
    }
}
```

```
public void actionPerformed(ActionEvent e)
{
    JTextField textSource = (JTextField)e.getSource();
    if(textSource==titleText)
    {
        this.setTitle(titleText.getText());
    }
    else
    {
        if(textSource == passwordText)
        {
            char[] c = passwordText.getPassword();
            titleText.setText(new String(c));
        }
    }
}
```





```
class MathWindow extends JFrame
{
    JTextField inputText, showText;
    MathWindow()
    {
        inputText = new JTextField(10);
        add(inputText);
        showText = new JTextField(10);
        add(showText);
    }
}
```

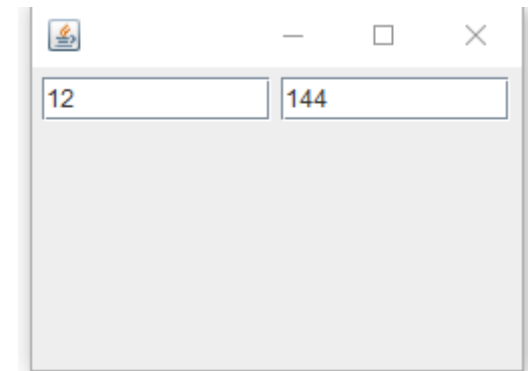
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.math.*;
public class Example10_9
{
    public static void main(String args[])
    {
        MathWindow win=new MathWindow();
    }
}
```

## 【例子9: Example10\_9.java】

- 使用匿名对象作为inputText的监视器，当在inputText中输入一个数字字符串后，监视器负责计算这个数的平方，并将结果放入showText中。

```
inputText.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    { String s = inputText.getText();
      try
      {
          BigInteger n = new BigInteger(s);
          n = n.pow(2);
          showText.setText(n.toString());
      }
      catch(NumberFormatException e2)
      {
          showText.setText("请输入数字字符");
          inputText.setText(null);
      }
    }
});

setLayout(new FlowLayout());
validate();
setBounds(100,100,260,190);
setVisible(true);
setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
}
```



- 4.菜单项上的ActionEvent事件
- 单击某个菜单项可以发生ActionEvent事件。菜单项使用addActionListener(ActionListener listener)方法获得监视器。

- 5. JTextArea 文本区
- (1) 使用 `JTextArea(int rows, int columns)` 构造一个可见行和可见列分别是 `rows`、`columns` 的文本区。
  - 使用 `setLineWrap(boolean b)` 方法决定输入的文本能否在文本区的右边界自动换行；
  - 使用 `setWrapStyleWord(boolean b)` 决定是以单词为界（`b` 取 `true` 时）或以字符为界（`b` 取 `false` 时）进行换行。
  - 使用 `append(String s)` 尾加文本。
  - 使用 `insert(String s, int x)` 方法在文本区的指定位置处插入文本。
  - 使用 `getCaretPosition()` 获取文本区中输入光标的位置。
  - 使用 `copy()` 和 `cut()` 方法将文本区中选中的内容拷贝或剪切到系统的剪贴板。
  - 使用 `paste()` 方法将系统剪贴板上的文本数据粘贴在文本区中。



## 10.6 文本组件

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class Example10_10
{
    public static void main(String args[])
    {
        EditWindow win = new EditWindow("窗口");
    }
}
```

### • 【例子10: Example10\_10.java】

- 单击菜单“编辑”中的相应菜单项将文本区中选中的内容剪切到系统剪贴板或将系统剪贴板的内容粘贴到文本区。

```
class EditWindow extends JFrame implements ActionListener
```

```
{
    JMenuBar menubar;
    JMenu menu;
    JSplitPane splitPane;
    JMenuItem itemCopy, itemCut, itemPaste;
    JTextArea text1, text2;
```

```
EditWindow(String s)
```

```
{
    setTitle(s);
    setSize(260, 270);
    setLocation(120, 120);
    setVisible(true);
```

```
    itemCopy = new JMenuItem("复制");
    itemCut = new JMenuItem("剪切");
    itemPaste = new JMenuItem("粘贴");
```

```
    itemCopy.addActionListener(this);
    itemCut.addActionListener(this);
    itemPaste.addActionListener(this);
```

```
    menu = new JMenu("编辑");
    menu.add(itemCopy);
    menu.add(itemCut);
    menu.add(itemPaste);
```

```
    menubar = new JMenuBar();
    menubar.add(menu);
    setJMenuBar(menubar);
```

```
    text1 = new JTextArea();
    text2 = new JTextArea();
    splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
    text1, text2);
```

```
    splitPane.setDividerLocation(120);
    add(splitPane, BorderLayout.CENTER);
```

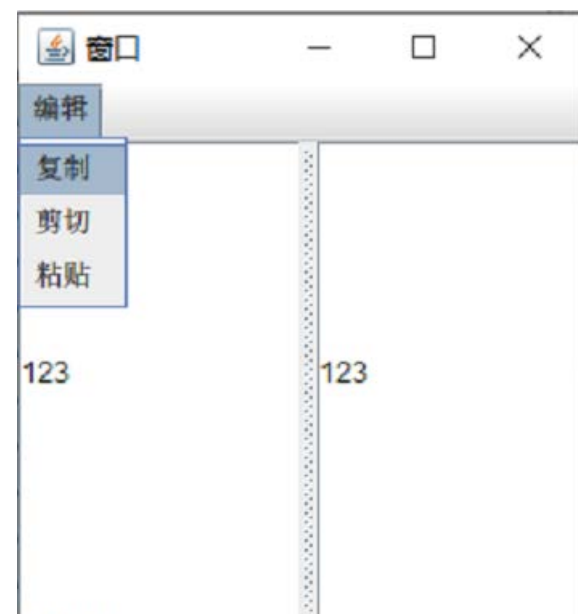
```
    validate();
```

```
    setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
```

```
}
```

## 10.6 文本组件

```
public void actionPerformed(ActionEvent e)
{
    if(e.getSource() == itemCopy)
    {
        text1.copy();
    }
    else
    {
        if(e.getSource() == itemCut)
        {
            text1.cut();
        }
        else
        {
            if(e.getSource() == itemPaste)
            {
                text2.paste();
            }
        }
    }
}
```



- (2) 文本区上的**DocumentEvent**事件
- 文本区可以触发DocumentEvent事件，DocumentEvent类在javax.swing.event包中。用户在文本区组件的UI代表的视图中进行文本编辑操作，使得文本区中的文本内容发生变化，将导致该组件所维护的文档模型中的数据发生变化，从而导致DocumentEvent事件的发生。需要使用**addDocumentListener()**
- 方法向组件维护的文档注册监视器。
- 监视器需实现DocumentListener接口，该接口中有三个方法：
  - public void changedUpdate(DocumentEvent e)
  - public void removeUpdate(DocumentEvent e)
  - public void insertUpdate(DocumentEvent e)
- 文本区调用 getDocument()方法返回维护的文档，该文档是实现了Document接口类的一个实例。

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.regex.*;
import javax.swing.event.*;
public class Example10_11
{
    public static void main(String args[])
    {new PatternWindow();}
}
```

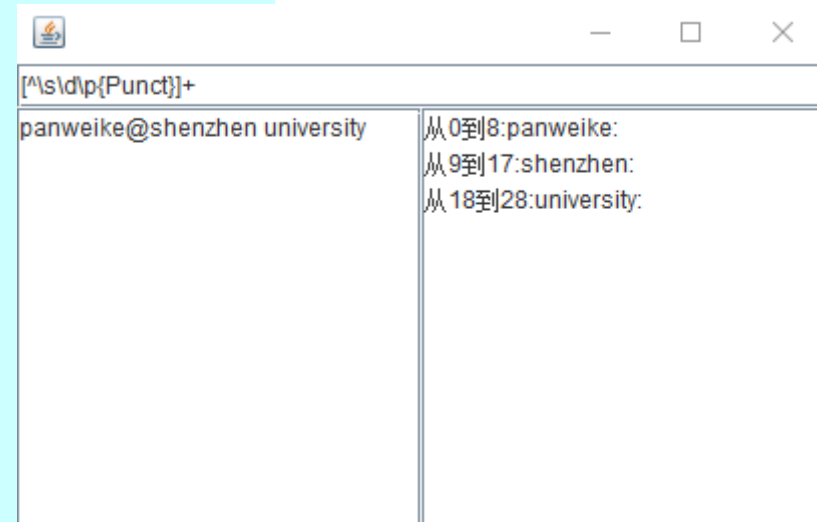
### 【例子11: Example10\_11.java】

- 有两个文本区和一个文本框。当用户在文本区inputText进行编辑操作时，文本区showText将显示第一个文本区中所有和指定模式匹配的字符串。用户可以事先在一个文本框patternText中输入指定的模式，比如，输入：\d+，即通过该模式获得文本区inputText中的全部数字。

```
class PatternWindow extends JFrame implements DocumentListener,
ActionListener
{
    JTextArea inputText,showText;
    JTextField patternText;
    Pattern p; //模式对象
    Matcher m; //匹配对象
    PatternWindow()
    {
        inputText = new JTextArea();
        showText = new JTextArea();

        patternText = new JTextField("[^\\s\\d\\p{Punct}]+");
        patternText.addActionListener(this);
        add(patternText, BorderLayout.NORTH);

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(1, 2));
        panel.add(new JScrollPane(inputText));
        panel.add(new JScrollPane(showText));
        add(panel, BorderLayout.CENTER);
    }
}
```



## 10.6 文本组件

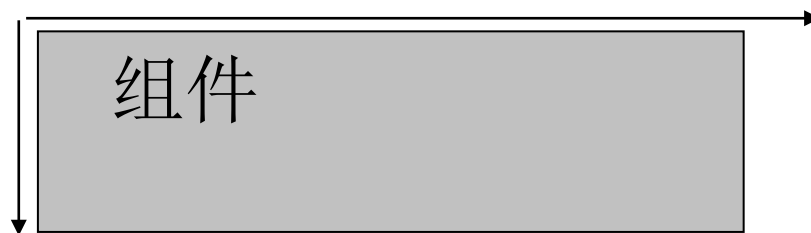
```
validate();
(inputText.getDocument()).addDocumentListener(this); // 向文档注册监视器
setBounds(120, 120, 260, 270);
setVisible(true);
setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
}

public void changedUpdate(DocumentEvent e)
{
    hangdleText();
}
public void removeUpdate(DocumentEvent e)
{
    changedUpdate(e);
}
public void insertUpdate(DocumentEvent e)
{
    changedUpdate(e);
}

public void hangdleText()
{
    showText.setText(null);
    String s = inputText.getText();
    p = Pattern.compile(patternText.getText()); // 初始化模式对象
    m = p.matcher(s);
    while(m.find())
    {
        showText.append("从"+m.start()+"到"+m.end()+"：");
        showText.append(m.group()+"：\n");
    }
}

public void actionPerformed(ActionEvent e)
{
    hangdleText();
}
}
```

- JComponent类是所有组件的父类，这一节介绍JComponent类的几个常用方法。
- **组件都是矩形形状**，组件本身有一个默认的坐标系，组件的左上角的坐标值是(0,0)。如果一个组件的宽是40，高是10，那么，该坐标系中，x坐标的最大值是40；y坐标的最大值是10。如下图所示。



组件上的坐标系

- 1.组件的**颜色**
- `public void setBackground(Color c):` 设置组件的背景色。
- `public void setForeground(Color c):` 设置组件的前景色。
- `public Color getBackground(Color c):` 获取组件的背景色。
- `public Color getForeground(Color c):` 获取组件的前景色。
- 上述方法中都涉及到Color类，Color类是java.awt包中的类，该类创建的对象称为颜色对象。
- 用Color类的构造方法`public Color(int red, int green, int blue)`可以创建一个颜色对象，其中red、green和blue的取值在0到255之间。

- 2.组件透明
- 组件默认是不透明的。 `public void setOpaque(boolean isOpaque)` 设置组件是否不透明，当参数 `isOpaque` 取 `false` 时组件被设置为透明，取值 `true` 时组件被设置为不透明。
- `public boolean isOpaque()` 当组件不透明时该方法返回 `true`，否则返回 `false`。



- 3.组件的边框
- 组件默认的边框是一个黑边的矩形。
- `public void setBorder(Border border)`: 设置组件的边框。
- `public Border getBorder()`: 返回边框。
- 组件调用`setBorder`方法来设置边框，该方法的参数是一个接口，因此必须向该参数传递一个实现接口`Border`类的实例，如果传递一个`null`，组件将取消边框。

- 4.组件的**字体**
- **public void setFont(Font f):** 组件调用该方法设置组件上的字体。例如，文本组件调用该方法可以设置文本组件中的字体。
- **public Font getFont(Font f):** 组件调用该方法获取组件上的字体。上述方法中用到了java.awt包中的Font类，该类创建的对象称为字体对象。
  - Font类的构造方法是**public Font(String name, int style, int size);**使用该构造方法可以创建字体对象。其中，**name**是字体的名字，如果系统不支持字体的名字，将取默认的名字创建字体对象。**style**决定字体的样式，取值是一个整数。在创建字体对象时，应当给出一个合理的字体名字，也就是说，程序所在的计算机系统上有这样的字体名字。如果在创建字体对象时，没有给出一个合理的字体名字，那么该字体在特定平台的字体系统名称为默认名称。

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

public class Example10_19
{
    public static void main(String args[])
    {
        new FontWin();
    }
}
```

### 【例子19: Example10\_19.java】

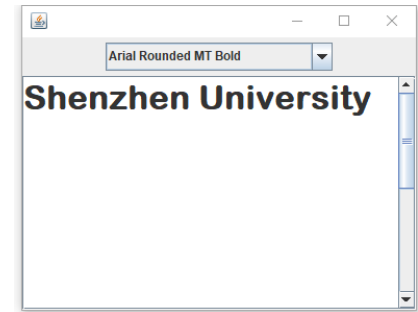
- 我们在一个下拉列表中列出全部可用字体名字，然后在下拉列表中选择字体名字，文本区用这种字体显示特定的文本“Shenzhen University”。

```
class FontWin extends JFrame implements ItemListener
{
    JComboBox listFont;
    JTextArea text;
    FontWin()
    {
        GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
        String fontName[] = ge.getAvailableFontFamilyNames();
        listFont = new JComboBox(fontName);
        listFont.addItemListener(this);

        JPanel pNorth = new JPanel();
        pNorth.add(listFont);
        add(pNorth, BorderLayout.NORTH);

        text = new JTextArea(12,12);
        add(new JScrollPane(text), BorderLayout.CENTER);

        setBounds(100,120,300,300);
        setVisible(true);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }
}
```



```
public void itemStateChanged(ItemEvent e)
{
    String name = (String)listFont.getSelectedItem();
    Font f = new Font(name, Font.BOLD, 32);
    text.setFont(f);
    text.setText("Shenzhen University");
}
```

- 5.组件的**大小**与**位置**
- `public void setSize(int width, int height)`: 组件调用该方法设置组件的大小, 通过参数`width`和`height`指定组件的宽度和高度。
- `public void setLocation(int x, int y)`: 组件调用该方法设置组件在容器中的位置, 参数`x, y`是组件距容器的左边界`x`个像素, 距容器的上边界 `y` 个像素。
- `public void setBounds(int x, int y, int width, int height)`: 组件调用该方法设置组件在容器中的位置和组件的大小。

- **public Dimension getSize():** 组件调用该方法返回一个Dimension对象的引用，该对象实体中含有名字是width和height的成员变量，也就是当前组件的宽度和高度。
- **public Point getLocation(int x, int y):** 组件调用该方法返回一个Point对象的引用，该对象实体中含有名字是x和y的成员变量，就是组件的左上角在容器的坐标系中的x坐标和y坐标。
- **public Rectangle getBounds():** 组件调用该方法返回一个Rectangle对象的引用，该对象实体中含有名字是x、y、width和height的成员变量，分别是当前组件左上角在容器坐标系中的x坐标和y坐标，宽度和高度。

- 6.组件的**激活**与**可见性**
- `public void setEnabled(boolean b)`: 组件调用该方法可以设置组件**是否可被激活**，当参数**b**取值**true**时，组件可以被激活，当参数**b**取值**false**时，组件不可激活。默认情况下，组件是可以被激活的。
- `public void setVisible(boolean)`: 设置组件在该容器中的**可见性**，当参数**b**取值**true**时，组件在容器中可见，当参数**b**取值**false**时，组件在容器中不可见。除了**Window**型组件外，其它类型组件默认是可见的。

- 以下有关组件的内容请同学们自学
  - 10.7 按钮与标签组件
  - 10.8 复选框与单选按钮组件
  - 10.9 列表组件
  - 10.10 表格组件
  - 10.11 树组件
  - 10.12 进度条组件

## Outline

- 10.1 AWT组件与Swing组件概述
- 10.2 JFrame窗体
- 10.3 菜单组件
- 10.4 布局设计
- 10.5 中间容器
- 10.6 文本组件
- 10.7 按钮与标签组件
- 10.8 复选框与单选按钮组件
- 10.9 列表组件
- 10.10 表格组件
- 10.11 树组件
- 10.12 进度条组件
- 10.13 组件常用方法
- **10.14 窗口事件**
- **10.15 鼠标事件**
- **10.16 焦点事件**
- **10.17 键盘事件**
- 10.18 AWT线程
- 10.19 计时器
- 10.20 MVC设计模式
- 10.21 播放音频
- 10.22 按钮绑定到键盘
- 10.23 对话框
- 10.24 多文档界面
- 10.25 发布应用程序





- 1.WindowListener接口
- JFrame类是Window类的子类，Window型对象都能触发**WindowEvent**事件。当一个JFrame窗口被激活、撤销激活、打开、关闭、图标化或撤销图标化时，会引发**窗口事件**，即**WindowEvent**创建一个**窗口事件对象**。
- 窗口使用addWindowListener()方法获得监视器，创建**监视器**对象的类必须实现WindowListener接口，该接口中有7个不同的方法，分别是：
  - public void Window**Activated**(WindowEvent e): 当窗口**从非激活状态到激活状态**时，窗口的监视器调用该方法。
  - public void Window**Deactivated**(WindowEvent e): 当窗口**从激活状态到非激活状态**时，窗口的监视器调用该方法。
  - public void Window**Closing**(WindowEvent e): 窗口**正在被关闭**时，窗口监视器调用该方法。

## 10.14 窗口事件

- `public void WindowClosed(WindowEvent e)`: 当窗口**关闭**时，窗口的监视器调用该方法。
- `public void WindowIconified(WindowEvent e)`: 窗口**图标化**时，窗口的监视器调用该方法。
- `public void WindowDeiconified(WindowEvent e)`: 当窗口**撤销图标化**时，窗口的监视器调用该方法。
- `public void WindowOpened(WindowEvent e)`: 当窗口**打开**时，窗口的监视器调用该方法。

- **WindowEvent**创建的事件对象调用**getWindow()**方法可以获取发生窗口事件的窗口。
- 当单击窗口上的**关闭图标**时，监视器首先调用**WindowClosing()**方法，然后执行窗口初始化时用**setDefaultCloseOperation(int n)**方法设定的关闭操作，最后再执行**WindowClosed()**方法。
  - 如果在**WindowClosing()**方法执行了**System.exit(0);**或**setDefaultCloseOperation(int n)**设定的关闭操作是**EXIT\_ON\_CLOSE**或**DO\_NOTHING\_ON\_CLOSE**，那么监视器就没有机会再调用**WindowClosed()**方法了。
- 当单击窗口的**图标化按钮**时，监视器调用**WindowIconified()**方法后，还将调用**WindowDeactivated()**方法。
- 当**撤销窗口图标化**时，监视器调用**WindowDeiconified()**方法后还会调用**WindowActivated()**方法。

- 2.WindowAdapter适配器 
- 接口中如果有多个方法会给使用者带来诸多不便，因为实现这个接口的类必须实现该接口中的全部方法，否则这个类必须是一个abstract类。为了给编程人员提供方便，对于Java提供的接口，如果其中的方法多于一个，就提供一个相关的称为适配器（adapter）的类，这个适配器是已经实现了相应接口的类，只是相应方法的实现内容为空。
- 例如，Java在提供WindowListener接口的同时，又提供了  WindowAdapter类，**WindowAdapter类实现了WindowListener接口**。因此，可以使用WindowAdapter类的**子类**创建的对象作为监视器，在子类中重写所需要的接口方法即可。

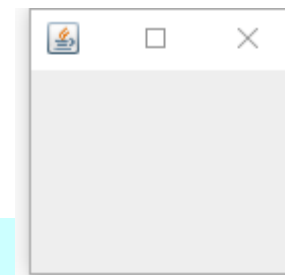
### 【例子20: Example10\_20.java】

- 我们使用WindowAdapter的匿名类（是WindowAdapter的一个子类）做窗口的监视器。


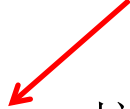
```
import java.awt.event.*;
import javax.swing.*;

public class Example10_20
{
    public static void main(String args[])
    {
        MyWindow10_20 win = new MyWindow10_20();
    }
}
```

```
class MyWindow10_20 extends JFrame
{
    MyWindow10_20()
    {
        addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
        setBounds(100,100,150,150);
        setVisible(true);
        setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    }
}
```



- 1.鼠标事件的触发
- 组件是可以触发鼠标事件的事件源。用户的下列7种操作都可以使得组件触发鼠标事件：
  - 鼠标指针从组件外进入
  - 鼠标指针从组件内退出
  - 鼠标指针停留在组件上时，按下鼠标
  - 鼠标指针停留在组件上时，释放鼠标
  - 鼠标指针停留在组件上时，单击鼠标
  - 在组件上拖动鼠标指针
  - 在组件上移动鼠标指针
- 鼠标事件的类型是**MouseEvent**，即组件触发鼠标事件时，**MouseEvent**类自动创建一个事件对象。

- 2. MouseListener接口与MouseMotionListener接口
- Java使用两个接口来处理鼠标事件
- (1) MouseListener接口
- 如果事件源使用addMouseListener(MouseListener listener)获取监视器，那么用户的下列5种操作可以使得事件源触发鼠标事件：
  - 鼠标指针从组件外进入
  - 鼠标指针从组件内退出
  - 鼠标指针停留在组件上时，按下鼠标
  - 鼠标指针停留在组件上时，释放鼠标
  - 鼠标指针停留在组件上时，单击或连续单击鼠标

- 创建监视器的类必须要实现MouseListener接口，该接口有5个方法：
  - mouseEntered(MouseEvent e): 负责处理鼠标**进入组件**触发的鼠标事件。
  - mouseExited(MouseEvent e): 负责处理鼠标**退出组件**触发的鼠标事件。
  - mousePressed(MouseEvent e): 负责处理鼠标**按下**触发的鼠标事件。
  - mouseReleased(MouseEvent e): 负责处理鼠标**释放**触发的鼠标事件。
  - mouseClicked(MouseEvent e): 负责处理鼠标**单击或连击**触发的鼠标事件。



- (2) `MouseMotionListener`接口
- 如果事件源使用`addMouseMotionListener(MouseMotionListener listener)`获取监视器，那么用户的下列两种操作可使得事件源触发鼠标事件：
  - 在组件上拖动鼠标指针
  - 在组件上移动鼠标指针
- 相应的有两个处理接口的方法
  - `mouseDragged(MouseEvent e)`: 负责处理鼠标**拖动**事件，即在事件源上拖动鼠标时，监视器将自动调用接口中的这个方法对事件做出处理。
  - `mouseMoved(MouseEvent e)`: 负责处理鼠标**移动**事件，即在事件源上移动鼠标时，监视器将自动调用接口中的这个方法对事件做出处理。

- 由于处理鼠标事件的接口中的方法多于一个，Java提供了相应的适配器（adapter）类，分别是**MouseListener**和**MouseMotionListener**，这两个类分别实现了MouseListener接口和MouseMotionListener接口。

- 3.MouseEvent类
- 在处理鼠标事件时，程序经常关心鼠标在当前组件坐标系中的位置，以及触发鼠标事件使用的是鼠标的左键或右键等信息。
- MouseEvent类中有下列几个重要的方法：
  - **getX()**: 鼠标事件调用该方法返回触发当前鼠标事件时，鼠标指针在事件源坐标系中的**x-坐标**。
  - **getY()**: 鼠标事件调用该方法返回触发当前鼠标事件时，鼠标指针在事件源坐标系中的**y-坐标**。
  - **getClickCount()**: 鼠标事件调用该方法返回鼠标被连续点击的**次数**。
  - **getModifiers()**: 鼠标事件调用该方法返回一个整数值，如果是通过鼠标**左键**触发的鼠标事件，该方法返回的值等于InputEvent类中的类常量BUTTON1\_MASK；如果是**右键**返回的是InputEvent类中的类常量BUTTON3\_MASK来表示。
  - **getSource()**: 鼠标事件调用该方法返回触发当前鼠标事件的**事件源**。

## 10.15 鼠标事件

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

public class Example10_21
{
    public static void main(String args[])
    {
        new MouseWindow();
    }
}
```

### 【例子21: Example10\_21.java】

- 分别监视按钮、标签和窗口的内容面板上的鼠标事件（如鼠标点击左键），当发生鼠标事件时，获取鼠标的坐标值，注意，**事件源的坐标系**的左上角是原点。

```
class MouseWindow extends JFrame implements MouseListener
{
    JButton button;
    JTextArea textArea;
    MouseWindow()
    {
        setLayout(new FlowLayout());
        addMouseListener(this);

        button = new JButton("我是按钮");
        button.addMouseListener(this);
        add(button);

        textArea = new JTextArea(8,18);
        add(new JScrollPane(textArea));

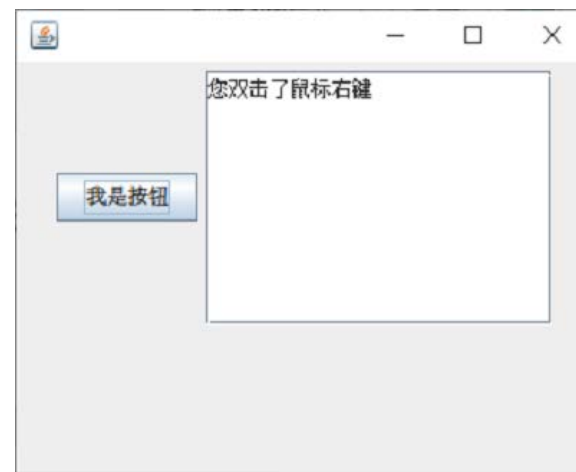
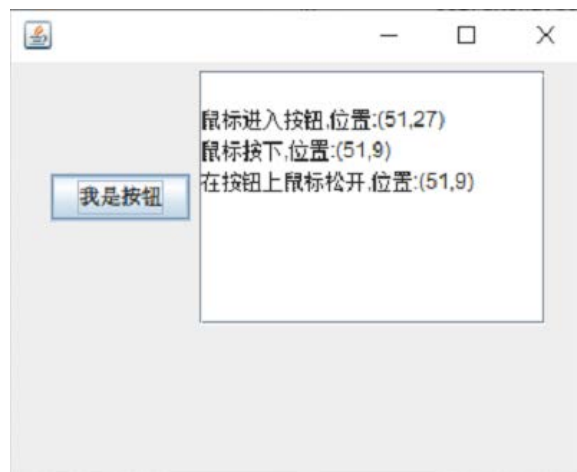
        setBounds(100,100,350,280);
        setVisible(true);
        validate();
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

```
// ---
public void mousePressed(MouseEvent e)
{
    textArea.append("\n鼠标按下,位置:"+"("+e.getX()+","+e.getY()+")");
}

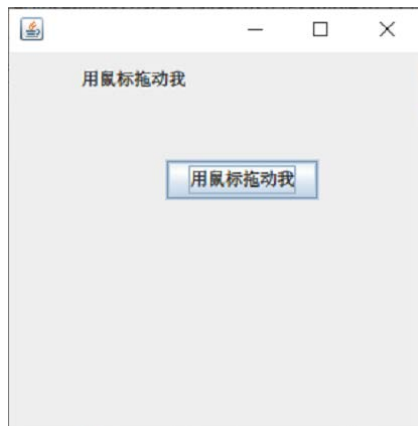
// ---
public void mouseReleased(MouseEvent e)
{
    if(e.getSource()==button)
    {
        textArea.append("\n在按钮上鼠标松开,位置:"+"("+e.getX()+","+e.getY()+")");
    }
}
```

## 10.15 鼠标事件

```
// ---  
public void mouseEntered(MouseEvent e)  
{  
    if(e.getSource()==button)  
    {  
        textArea.append("\n鼠标进入按钮,位置:"+"("+e.getX()+","+e.getY()+")");  
    }  
}  
  
// ---  
public void mouseExited(MouseEvent e){}  
  
// ---  
public void mouseClicked(MouseEvent e)  
{  
    if(e.getModifiers()==InputEvent.BUTTON3_MASK &&  
    e.getClickCount()>=2)  
    {  
        textArea.setText("您双击了鼠标右键");  
    }  
}  
}
```



## 10.15 鼠标事件



```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Example10_22
{
    public static void main(String args[])
    {
        JFrame fr = new JFrame();
        fr.add(new LP(), BorderLayout.CENTER);
        fr.setVisible(true);
        fr.setBounds(12,12,300,300);
        fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        fr.validate();
    }
}
```

- 4.用鼠标拖动组件
- 可以使用坐标变换来实现组件的拖动。当我们用鼠标**拖动**容器中的组件时，可以先获取鼠标指针在组件坐标系中的坐标 $x,y$ ，以及组件的左上角在容器坐标系中的坐标 $a,b$ ；如果在拖动组件时，想让鼠标指针的位置相对于拖动的组件保持静止，那么，组件左上角在容器坐标系中的位置应当是 $a+x-x_0$ 和 $a+y-y_0$ ，其中 $x_0$ 和 $y_0$ 是最初在组件上按下鼠标时鼠标指针在组件坐标系中的位置坐标。

### 【例子22: Example10\_22.java】

- 窗体中添加了一个分层窗格，分层窗格中添加了一些组件。该例子使用MouseListener和MouseMotionListener接口处理鼠标事件。

## 10.15 鼠标事件

```
class LP extends JLayeredPane implements MouseListener, MouseMotionListener
{
    JButton button;
    JLabel label;
    int x,y,a,b,x0,y0;

    LP()
    {
        // ---
        button = new JButton("用鼠标拖动我");
        button.addMouseListener(this);
        button.addMouseMotionListener(this);

        // ---
        label = new JLabel("用鼠标拖动我");
        label.addMouseListener(this);
        label.addMouseMotionListener(this);

        // ---
        setLayout(new FlowLayout());
        add(label ,JLayeredPane.DEFAULT_LAYER);
        add(button,JLayeredPane.DEFAULT_LAYER);
    }

    public void mousePressed(MouseEvent e)
    {
        JComponent com = null;
        com = (JComponent)e.getSource();
        setLayer(com,JLayeredPane.DRAG_LAYER);
        a = com.getBounds().x;
        b = com.getBounds().y;
        x0 = e.getX();
        y0 = e.getY();
    }

    public void mouseReleased(MouseEvent e)
    {
        JComponent com = null;
        com = (JComponent)e.getSource();
        setLayer(com,JLayeredPane.DEFAULT_LAYER);
    }

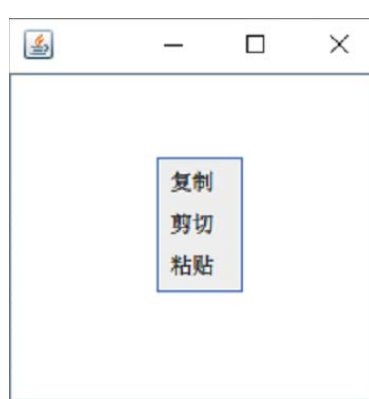
    // ---
    public void mouseEntered(MouseEvent e){}

    // ---
    public void mouseExited(MouseEvent e){}

    // ---
    public void mouseClicked(MouseEvent e){}

    // ---
    public void mouseMoved(MouseEvent e){}

    // ---
    public void mouseDragged(MouseEvent e)
    {
        JComponent com = null;
        if(e.getSource() instanceof JComponent)
        {
            com = (JComponent)e.getSource();
            a = com.getBounds().x;
            b = com.getBounds().y;
            x = e.getX();
            y = e.getY();
            com.setLocation(a+x-x0,b+y-y0);
        }
    }
}
```



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

public class Example10_23
{
    public static void main(String args[])
    {
        new JPopupMenuWindow();
    }
}
```

- 5.弹出式菜单
- 单击鼠标右键出现“弹出式菜单”是用户熟悉和常用的操作。弹出式菜单由JPopupMenu类负责创建，可以用下列构造方法创建弹出式菜单：
  - public JPopupMenu(): 构造无标题弹出式菜单。
  - public JPopupMenu(String label): 构造由参数label指定标题的弹出式菜单。
  - 通过调用public void show(Component invoker, int x, int y)方法设置弹出式菜单在组件invoker上的弹出位置。

### 【例子23: Example10\_23.java】

- 当你在文本区上单击鼠标右键时，在鼠标位置处弹出菜单，用户选择相应的菜单项拷贝、剪贴等操作。



## 10.15 鼠标事件

```
class JPopupMenuWindow extends JFrame implements ActionListener
{
    JPopupMenu menu;
    JMenuItem itemCopy,itemCut,itemPaste;
    JTextArea text;
    JPopupMenuWindow()
    {
        menu = new JPopupMenu();
        itemCopy = new JMenuItem("复制");
        itemCut = new JMenuItem("剪切");
        itemPaste = new JMenuItem("粘贴");
        menu.add(itemCopy);
        menu.add(itemCut);
        menu.add(itemPaste);

        text = new JTextArea();
        text.addMouseListener(new MouseAdapter()
        { public void mousePressed(MouseEvent e)
          {
              if(e.getModifiers()==InputEvent.BUTTON3_MASK)
              {
                  menu.show(text, e.getX(), e.getY() );
              }
          }
        });

        add(new JScrollPane(text), BorderLayout.CENTER);

        itemCopy.addActionListener(this);
        itemCut.addActionListener(this);
        itemPaste.addActionListener(this);

        setBounds(120,100,220,220);
        setVisible(true);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }
}
```

```
public void actionPerformed(ActionEvent e)
{
    if(e.getSource()==itemCopy)
    {
        text.copy();
    }
    else
    {
        if(e.getSource()==itemCut)
        {
            text.cut();
        }
        else
        {
            if(e.getSource()==itemPaste)
            {
                text.paste();
            }
        }
    }
}
```

- 组件可以触发**焦点事件**。组件可以使用`public void addFocusListener(FocusListener listener)`增加焦点事件监视器。
- 创建监视器的类必须要实现`FocusListener`接口，该接口有两个方法：
  - `public void focusGained(FocusEvent e)`
  - `public void focusLost(FocusEvent e)`
- 当组件从无输入焦点变成有输入焦点触发`FocusEvent`事件时，监视器调用类实现的接口方法`focusGained(FocusEvent e)`；当组件从有输入焦点变成无输入焦点触发`FocusEvent`事件时，监视器调用类实现的接口方法`focusLost(FocusEvent e)`。
- 一个组件可以调用`public boolean requestFocusInWindow()`方法**获得输入焦点**。

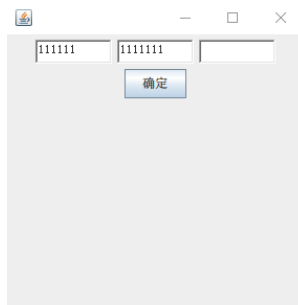
- 当一个组件处于激活状态时，组件可以成为触发KeyEvent事件的事件源。当某个组件处于激活状态时，如果用户敲击了键盘上一个键就会导致这个组件触发KeyEvent事件。
- 1.使用KeyListener接口处理键盘事件
- 组件使用addKeyListener方法获得监视器。监视器是一个对象，创建该对象的类必须实现接口KeyListener。接口KeyListener中有3个方法：
  - public void keyPressed(KeyEvent e)
  - public void keyTyped(KeyEvent e)
  - public void KeyReleased(KeyEvent e)

- 当按下键盘上某个键时，监视器就会发现，然后方法 **keyPressed**(KeyEvent e) 会自动执行，并且 KeyEvent 类自动创建一个对象传递给方法 keyPressed(KeyEvent e) 中的参数 e。
- 方法 keyTyped(KeyEvent e) 是 keyPressed(KeyEvent e) 和 keyReleased(KeyEvent e) 方法的组合。
- 用 KeyEvent 类的 public int get**KeyCode**() 方法可以判断哪个键被按下、敲击或释放，该方法返回一个键码值（如表 10.1 所示）。
- KeyEvent 类的 public char get**KeyChar**() 判断哪个键被按下、敲击或释放，该方法返回键的字符。

表 10.1 键码表

键 码	键	键 码	键
VK_F1-VK_F12	功能键 F1~F12	VK_SEMICOLON	分号
VK_LEFT	向左箭头	VK_PERIOD	.
VK_RIGHT	向右箭头	VK_SLASH	/
VK_UP	向上箭头	VK_BACK_SLASH	\
VK_DOWN	向下箭头	VK_0~VK_9	0~9
VK_KP_UP	小键盘的向上箭头	VK_A~VK_Z	a~z
VK_KP_DOWN	小键盘的向下箭头	VK_OPEN_BRACKET	[
VK_KP_LEFT	小键盘的向左箭头	VK_CLOSE_BRACKET	]
VK_KP_RIGHT	小键盘的向右箭头	VK_UNMPAD0-VK_NUMPAD9	小键盘上的 0~9
VK_END	END	VK_QUOTE	单引号 ‘
VK_HOME	HOME	VK_BACK_QUOTE	单引号 ’
VK_PAGE_DOWN	向后翻页	VK_ALT	Alt
VK_PAGE_UP	向前翻页	VK_CONTROL	Ctrl
VK_PRINTSCREEN	打印屏幕	VK_SHIFT	Shift
VK_SCROLL_LOCK	滚动锁定	VK_ESCAPE	Esc
VK_CAPS_LOCK	大写锁定	VK_NUM_LOCK	数字锁定
VK_TAB	制表符	VK_DELETE	删除
PAUSE	暂停	VK_CANCEL	取消
VK_INSERT	插入	VK_CLEAR	清除
VK_ENTER	回车	VK_BACK_SPACE	退格
VK_SPACE	空格	VK_COMMA	逗号
VK_PAUSE	暂停		

## 10.17 键盘事件



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

public class Example10_24
{
    public static void main(String args[])
    {
        Win win = new Win();
    }
}
```

### 【例子24: Example10\_24.java】

- 通过处理键盘事件来实现软件序列号的输入。当文本框获得输入焦点后，用户敲击键盘将使得当前文本框触发KeyEvent事件，在处理事件时，程序检查文本框中光标的位置，如果光标已经到达指定位置，就将输入焦点转移到下一个文本框。

```
class Win extends JFrame implements KeyListener,
FocusListener
{
    TextField text[] = new TextField[3];
    JButton b;
    Win()
    {
        setLayout(new FlowLayout());
        for(int i=0;i<3;i++)
        {
            text[i] = new TextField(7);
            text[i].addKeyListener(this); // 监视键盘事件
            text[i].addFocusListener(this); // 监视焦点事件
            add(text[i]);
        }

        b = new JButton("确定");
        add(b);

        text[0].requestFocusInWindow();
    }
}
```

```
setBounds(10,10,300,300);
setVisible(true);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

public void keyPressed(KeyEvent e)
{
    TextField t = (TextField) e.getSource();
    if(t.getCaretPosition()>=6)
    {
        t.transferFocus(); // !!!
    }
}

public void keyTyped(KeyEvent e){}
public void keyReleased(KeyEvent e){}
public void focusGained(FocusEvent e)
{
    TextField text = (TextField) e.getSource();
    text.setText(null);
}

public void focusLost(FocusEvent e){}
}
```

## 10.17 键盘事件

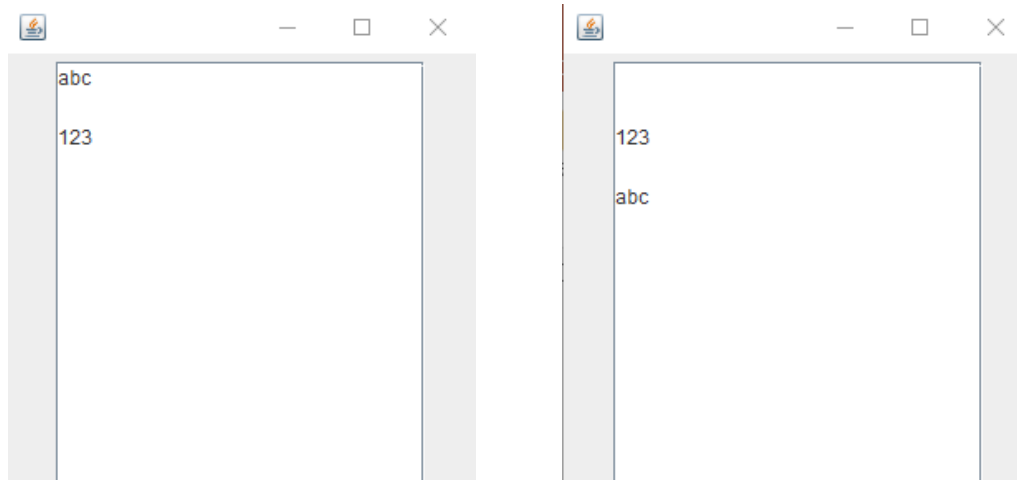
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Example10_25
{
    public static void main(String args[])
    {
        KeyWin win=new KeyWin();
    }
}
```

- 2.处理复合键
- 键盘事件KeyEvent对象调用**getModifiers()**方法，可以返回下列整数值，它们分别是InputEvent类的类常量：ALT\_MASK、CTRL\_MASK、SHIFT\_MASK

### 【例子25: Example10\_25.java】

- 用户通过CTRL+C、CTRL+X和CTRL+V实现文本区内容的拷贝、剪切和粘贴。



## 10.17 键盘事件

```
class KeyWin extends JFrame implements KeyListener
{
    JTextArea text;
    KeyWin()
    {
        setLayout(new FlowLayout());
        text = new JTextArea(30,20);
        text.addKeyListener(this);
        add(new JScrollPane(text), BorderLayout.CENTER);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(10,10,300,300);
        setVisible(true);
    }

    public void keyTyped(KeyEvent e)
    {
        JTextArea te = (JTextArea) e.getSource();
        if(e.getModifiers()==InputEvent.CTRL_MASK&&e.getKeyCode()==KeyEvent.VK_X)
        {
            te.cut();
        }
        else
        {
            if(e.getModifiers()==InputEvent.CTRL_MASK&&e.getKeyCode()==KeyEvent.VK_C)
            {
                te.copy();
            }
            else
            {
                if(e.getModifiers()==InputEvent.CTRL_MASK&&e.getKeyCode()==KeyEvent.VK_V)
                {
                    te.paste();
                }
            }
        }
    }

    public void keyPressed(KeyEvent e){}
    public void keyReleased(KeyEvent e){}
}
```



- 小结：
  - 事件类型： WindowEvent, MouseEvent, FocusEvent, KeyEvent
  - 监视器接口： WindowListener, MouseListener/MouseMotionListener, FocusListener, KeyListener
  - 适配器： WindowAdapter, MouseAdapter/MouseMotionAdapter
- 要求： 理解事件触发和响应的流程（与10.6节中 **ActionEvent**, **DocumentEvent** 的内容类似）； 能查阅各种方法等的细节

## Outline

- 10.1 AWT组件与Swing组件概述
- 10.2 JFrame窗体
- 10.3 菜单组件
- 10.4 布局设计
- 10.5 中间容器
- 10.6 文本组件
- 10.7 按钮与标签组件
- 10.8 复选框与单选按钮组件
- 10.9 列表组件
- 10.10 表格组件
- 10.11 树组件
- 10.12 进度条组件
- 10.13 组件常用方法
- 10.14 窗口事件
- 10.15 鼠标事件
- 10.16 焦点事件
- 10.17 键盘事件
- **10.18 AWT线程**
- **10.19 计时器**
- **10.20 MVC设计模式**
- **10.21 播放音频**
- **10.22 按钮绑定到键盘**
- 10.23 对话框
- 10.24 多文档界面
- **10.25 发布应用程序**

- 1.AWT线程
- 当Java程序包含图形用户界面（GUI）时，Java虚拟机在运行应用程序时会自动启动更多的线程，其中有两个重要的线程：AWT-EventQueue和AWT-Windows。
- AWT-EventQueue线程负责处理GUI事件，AWT-Windows线程负责将窗体或组件绘制到桌面。
- 2.挂起、恢复和终止线程
- 我们可以通过GUI界面事件，即在AWT-EventQueue线程中，通知其它线程开始运行、挂起、恢复或死亡。

- 所谓挂起一个线程就是让线程暂时让出CPU的使用权限，暂时停止执行，挂起一个线程可以使用wait方法。
- 恢复线程就是让曾挂起的线程恢复执行过程，即从曾中断处继续线程的执行。为了恢复线程，可以让主线程或其它线程执行notifyAll()方法，通知挂起的线程继续执行。
- 终止线程就是让线程结束run方法的执行进入死亡状态。

### 【例子26: Example10\_26.java】

- 通过单击“开始”按钮启动线程，该线程负责移动一个红色的标签。通过单击“挂起”按钮暂时中断线程的执行，单击“恢复”按钮恢复线程。通过单击“终止”按钮终止线程。

- 使用Timer类的构造方法：Timer(int a, Object b)创建一个计时器，其中的参数a的单位是毫秒，确定计时器每隔a毫秒“震铃”一次，参数b是计时器的监视器。
- 计时器发生的震铃事件是ActionEvent类型事件。当震铃事件发生时，监视器就会监视到这个事件，监视器就回调ActionListener接口中的actionPerformed方法。

### 【例子27: Example10\_27.java】

- 使用了计时器，使得标签每隔一秒钟显示一个汉字。

- MVC是一种通过三个不同部分构造一个软件或组件的理想办法：
  - 模型（**model**）：用于**存储**数据的对象。
  - 视图（**view**）：为模型提供数据**显示**的对象。
  - 控制器（**controller**）：**处理用户的交互操作**，对用户的操作作出响应；让模型和视图进行必要的交互，即通过视图修改获取模型中的数据；当模型中的数据变化时，让视图更新显示。

### 【例子28: Example10\_28.java】

- 首先编写一个封装三角梯形的类，然后再编写一个窗口。
  - 要求窗口使用三个文本框和一个文本区为三角形对象中的数据提供**视图**，其中三个文本框用来显示和更新梯形对象的上底、下底和高的长度；文本区对象用来显示梯形的面积。
  - 窗口中用一个按钮作为**控制器**，用户单击该按钮后，程序用3个文本框中的数据分别作为梯形的上底、下底和高的长度，并将计算出的三角梯形的面积显示在文本区中。

## 10.21 播放音频

- 假设音频文件hello.au位于应用程序当前目录中，播放音频的步骤如下：
- 1.创建File对象
- `File musicFile=new File("hello.au");`
- 2.获取URI对象
- `URI uri=musicFile.toURI();`
- 3.获取URL对象
- `URL url=uri.toURL();`



- 4. 创建音频对象
- 使用Applet的一个静态的方法（类方法）：`newAudioClip(java.net.URL url)`
- 根据参数url封装的音频获得一个可用于播放的音频对象clip，clip对象可以使用下列方法来处理声音文件：
  - `play()`：开始播放，
  - `loop()`：循环播放，
  - `stop()`：停止播放。

**【例子29: Example10\_29.java】**

- **按钮绑定到键盘**通常被理解为用户直接敲击某个键代替用鼠标单击该按钮所产生的效果。
- 1.AbstractAction类与特殊的监视器
- 如果按钮通过**addActionListener()**方法注册的监视器和程序为按钮的键盘操作指定的监视器是**同一个监视器**，用户直接敲击某个键（按钮的键盘操作）就可代替用鼠标单击该按钮所产生的效果，这也就是人们通常理解的按钮的键盘绑定。

- 抽象类`javax.swing.AbstractAction`类已经实现了`Action`接口，因为大部分应用不需要实现`Action`中的其他方法，因此编写`AbstractAction`类的子类时，只要重写`public void actionPerformed(ActionEvent e)`即可，该方法是`ActionListener`接口中的方法。
- 为按钮的键盘操作指定了监视器后，用户只要敲击相应的键，监视器就执行`actionPerformed()`方法。

- 2.指定监视器的步骤
- 以下假设按钮是`button`，`listener`是`AbstractAction`类的子类的实例。
- (1) 获取输入映射：按钮首先调用`public final InputMap getInputMap(int condition)`方法返回一个`InputMap`对象，其中参数`condition`取值`JComponent`类的下列`static`常量：`WHEN_FOCUSED`，`WHEN_IN_FOCUSED_WINDOW`，`WHEN_ANCESTOR_OF_FOCUSED_COMPONENT`。
- 例如：  
`InputMap inputmap =  
button.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW);`

- (2) 绑定按钮的键盘操作：步骤(1)返回的输入映射首先调用方法 `public void put(KeyStroke keyStroke, Object actionMapKey)` 将敲击键盘上的某键指定为按钮的键盘操作，并为该操作指定一个 `Object` 类型的 **映射关键字**，例如：`inputmap.put(KeyStroke.getKeyStroke("A"), "abcdefg");`
- (3) 为按钮的键盘操作指定监视器：按钮调用方法 `public final ActionMap getActionMap()` 返回一个 `ActionMap` 对象：**`ActionMap actionmap = button.getActionMap();`** 然后，该对象 `actionmap` 调用方法 `public void put(Object key, Action action)` 为按钮的键盘操作指定监视器 (实现单击键盘上的键通知监视器的过程)。例如：`actionmap.put("abcdefg", listener);`

- 可以使用**jar.exe**把一些文件压缩成一个JAR文件，来发布我们的应用程序。可以把Java应用程序中涉及到的类压缩成一个JAR文件，如Tom.jar，然后使用Java解释器(使用参数-jar)执行这个压缩文件：`java -jar Tom.jar`或用鼠标双击该文件，执行这个压缩文件。

- 假设D:\test目录中的应用程序有3个类Example、MyInterbalFrame和Mywindow，其中Example是主类。生成一个Jar文件的步骤如下：
- (1)首先用文本编辑器（如Windows中的记事本）编写一个清单文件  
**moon.mf**  
Manifest-Version: 1.0  
Main-Class: Example  
Created-By: 1.6
- 注：编写清单文件时，在"Manifest-Version: "和"1.0"之间、"Main-Class: "和主类"Example"之间以及"Created-By: "和"1.6"之间必须有且只有一个空格。保存moon.mf到D:\test。

注意：与4.7节中的区别。此处，用了**Main-Class**，而4.7节用**Class**。

- (2)生成JAR文件
- `D:\test\jar cfm Tom.jar moon.mf Example.class MyInterbalFrame.class Mywindow.class`
- 如果目录D:\test下的字节码文件刚好是应用程序需要的全部字节码文件，也可以这样生成JAR文件：`D:\test\jar cfm Tom.jar moon.mf *.class`
- 其中，参数c表示要生成一个新的JAR文件，f表示要生成的JAR文件的名称，m表示文件清单文件的名称。
- 现在就可以将Tom.jar文件复制到任何一个安装了Java运行环境的计算机上，只要用鼠标双击该文件就可以运行该Java应用程序了。



# 小节

- 10.1 AWT组件与Swing组件概述
- 10.2 JFrame窗体
- 10.3 菜单组件
- 10.4 布局设计
- 10.5 中间容器
- 10.6 文本组件
- 10.7 按钮与标签组件
- 10.8 复选框与单选按钮组件
- 10.9 列表组件
- 10.10 表格组件
- 10.11 树组件
- 10.12 进度条组件
- 10.13 组件常用方法
- 10.14 窗口事件
- 10.15 鼠标事件
- 10.16 焦点事件
- 10.17 键盘事件
- 10.18 AWT线程
- 10.19 计时器
- 10.20 MVC设计模式
- 10.21 播放音频
- 10.22 按钮绑定到键盘
- 10.23 对话框
- 10.24 多文档界面
- 10.25 发布应用程序