# GUI组件和用户界面设计

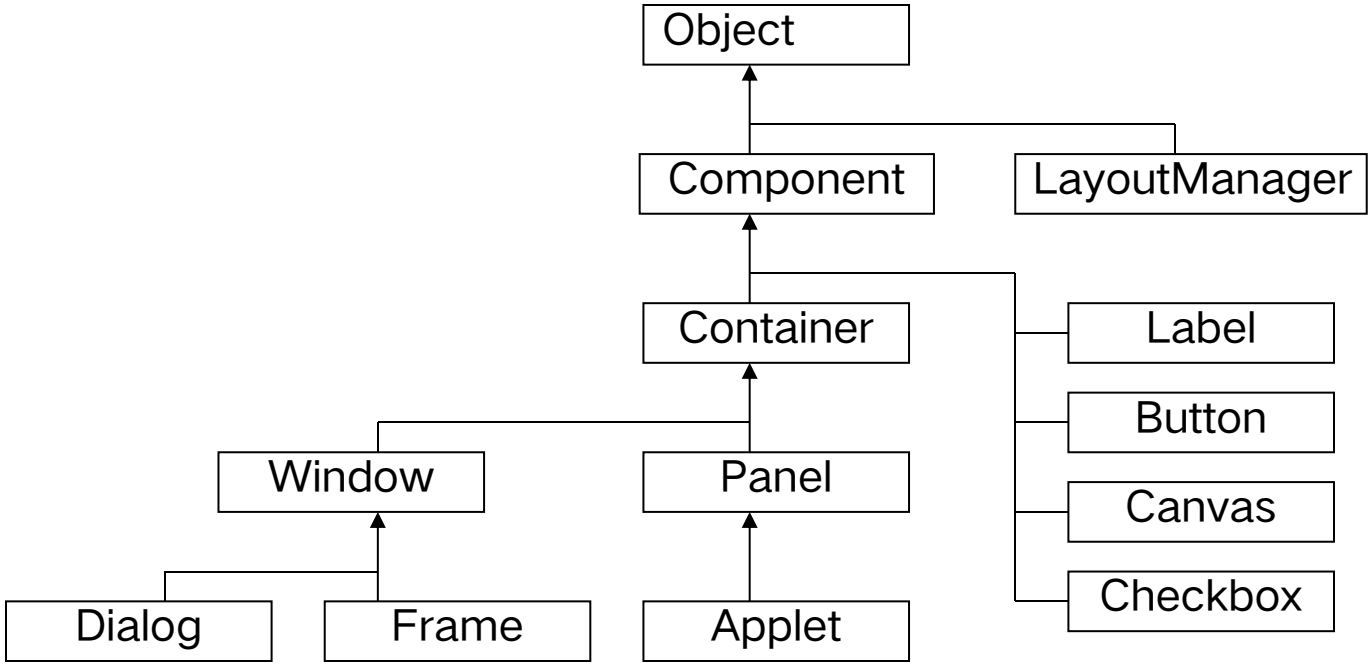
## Swing和AWT组件概述

Java语言中，为了方便图形用户界面(GUI)的开发，设计了专门的类库来生成各种标准图形界面元素和处理图形界面的各种事件。用来生成图形用户界面的类库：

- 早期Java GUI：Java最初的GUI开发主要依赖于AWT(abstract window toolkit,抽象窗口工具集)。java.awt包
- Swing的引入：随着Java的成熟，Swing作为AWT的替代品被引入，提供更好的性能和跨平台一致性。javax.swing包

# 1、 AWT组件介绍

java.awt包提供了基本的Java程序的GUI设计工具，包中的主要类或接口之间的继承关系:



# 1、 AWT组件介绍

(1)组件Component

Component是一个以**图形化**的方式显示在屏幕上并能与用户进行交互的**对象**，例如Button、Label，组件通常被放在**容器**中。

**Component类**是抽象类，定义了所有组件所具有的通用特性和行为，并派生出其他所有的组件。

Compoment类提供的功能：

- 基本的绘画支持。方法repaint()、paint()等用来在屏幕上绘制组件。
- 外形控制。包括字体、颜色等。
- 大小和位置控制。方法有:setSize（）、setLocation()等。
- 图像处理。类Component实现了接口ImageObserver，其中的方法imageUpdate()用来进行图像跟踪。
- 组件的状态控制。例如：setEnable()控制组件是否接收用户的输入，isEnable()，isVisible()、isValid()返回组件的状态。

# 1、 AWT组件介绍

(2)容器Container

容器是Component的**子类**,它具有**组件**的所有性质，同时又具有**容纳**其它组件和容器的功能。

每个容器

- add()方法向容器添加某个组件，
- remove()方法从容器中删除某个组件。

每个容器都与一个**布局管理器**相联，以确定容器内组件的布局方式。

- 容器通过方法setLayout()设置某种布局。

# 1、 AWT组件介绍

(3)布局管理器LayoutManager

**布局管理器**管理组件在容器中的**布局方式**。**布局管理器类**都实现了接口LayoutManager。

Java系统提供的标准布局管理器类：

- FlowLayout
- BorderLayout
- GridLayout
- CardLayout
- BoxLayout
- GridBagLayout

## 2. Swing介绍

Swing 组件在javax.swing包中。其特点:

(1) Swing组件是用100%纯Java代码实现的轻量级组件.

- 没有本地代码，不依赖操作系统的支持，这是它与重量级组件AWT的最大区别。
- Swing比AWT组件具有更强的**实用性和美观性**。

## 2. Swing介绍

### （2） Swing组件的多样化

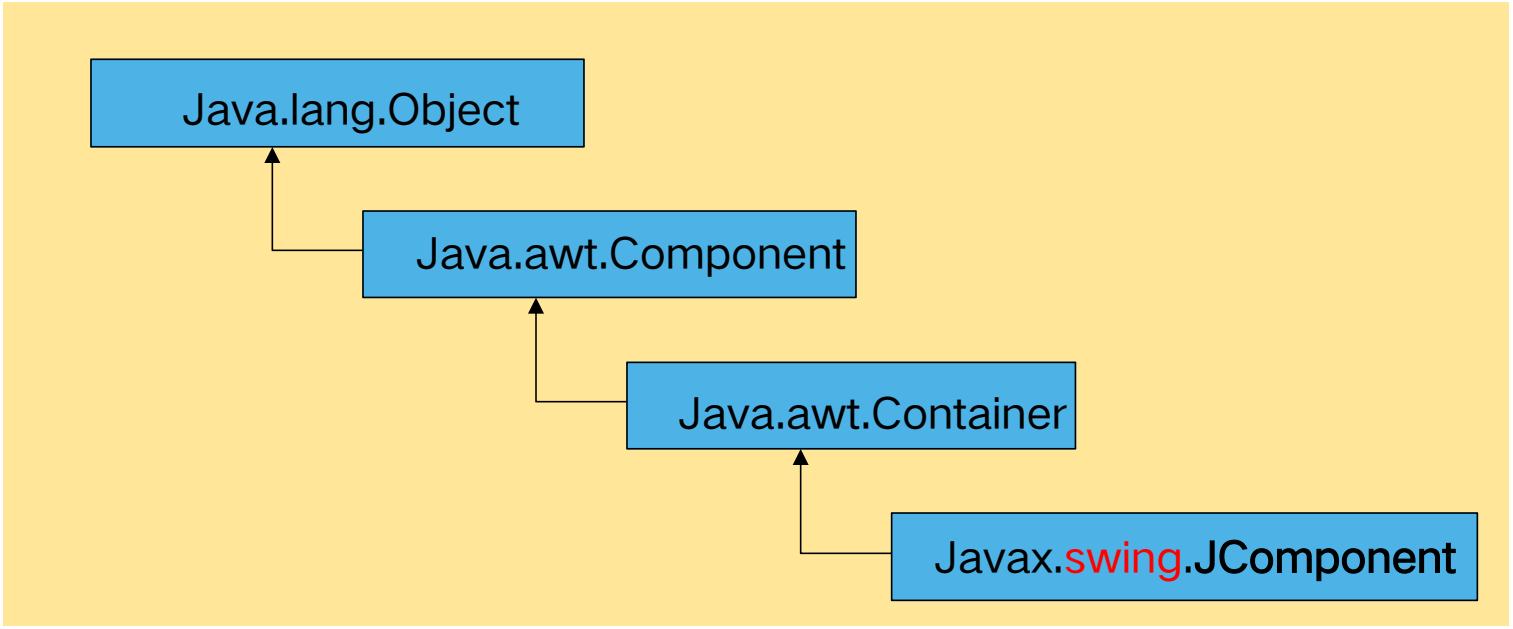
Swing是**AWT的扩展**， **Swing组件以“J”开头**

- 有与AWT类似的按钮（JButton)、标签（JLabel)、复选框（JCheckBox)、菜单（JMenu）等基本组件外，
- 增加了一个丰富的高层组件集合，如表格（JTable）、树（JTree)。
- **大多数Swing组件从JComponent类继承**

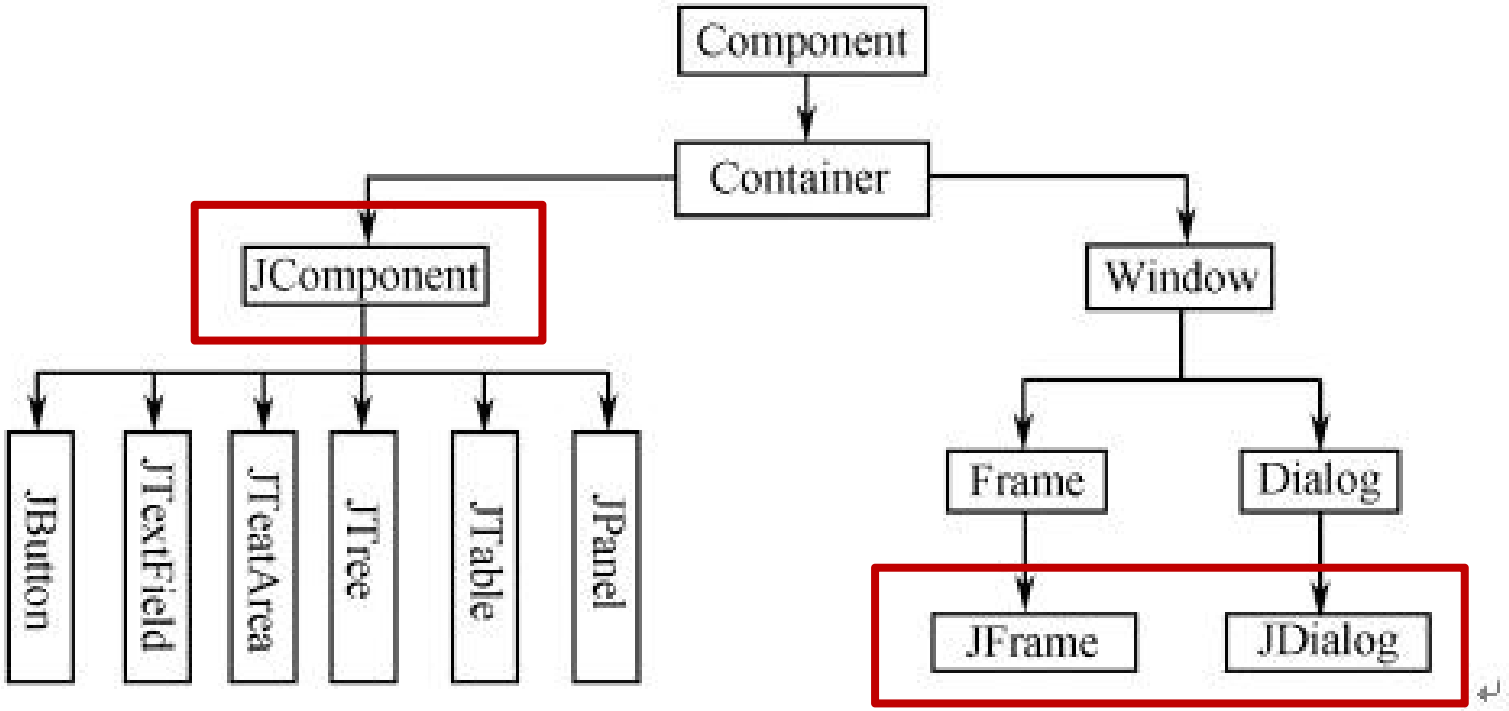
  **JComponent是一个抽象类**:定义所有子类组件的一般方法，如:

  使用setBorder()方法:设置组件外围的边框;

  使用setTooltipText()方法:为组件设置对用户有帮助的提示信息。




Swing 组件的继承的父类




JComponent类的部分子类以及JFrame类和JDialog类

## 3. Swing组件的分类

**1. 顶层容器（Top-Level Containers）：**

JFrame: 表示一个窗口，通常作为应用程序的**主窗口**。

JApplet: 表示一个小的应用程序。

JDialog: 表示对话框，用于显示需要用户输入或确认的窗口。

JWindow: 表示一个无边框的窗口，可以用于显示轻量级的内容或工具提示。

## 3. Swing组件的分类

**2. 普通容器（Standard Containers）：**

**JPanel: 轻量级容器，用于组织其他组件。**

JScrollPane: 滚动窗格，允许用户滚动查看内容。

JSplitPane: 分隔窗格。

JToolBar: 工具栏，通常包含一组工具按钮或控件。

## 3. Swing组件的分类

**3. 特殊容器（Special Containers）：**

JInternalFrame: 内部窗口，可以作为顶级窗口中的一个组件，可以有标题栏、边框和可以调整大小的边缘。

JLayeredPane: 分层窗格，允许组件在不同的层级上显示，从而创建复杂的布局。

JRootPane: 根窗格，是JFrame和JDialog的顶层组件，负责管理标题栏、菜单栏和内容区域。

## 3. Swing组件的分类

**4. 基本控件（Basic Controls）：**

JButton: **按钮控件**，用户可以点击执行操作。

JComboBox: 下拉列表，允许用户从列表中选择一个选项。

JList: 列表控件，显示一列可供选择的项。

JMenu: **菜单项**，用于创建菜单栏中的菜单。

JSlider: 滑动条，允许用户通过滑动选择一个值。

JTextField: **文本字段**，允许用户输入单行文本。

# JFrame窗体

当应用程序需要一个**窗口**时，可使用**JFrame或其子类**创建一个对象。

| 构造方法 | 说明 |
| --- | --- |
| **JFrame()** | 创建一个无标题的窗口。 |
| **JFrame(String s)** | 创建一个标题为s的窗口。 |

| 方法名称 | 说明 |
| --- | --- |
| 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；窗口的宽是width，高是height。 |
| public void setSize(int width,int height) | 设置窗口的大小,在屏幕出现默认位置是(0,0)。 |
| public void setVisible(boolean b) | 设置窗口是可见还是不可见，窗口默认是**不可见**的。 |

```java
public class JFrameTest1 {
    public static void main(String[] args) {
        JFrame frame = new JFrame( title: "测试窗口");
        frame.setSize( width: 300, height: 100);
        frame.setVisible(true);
    }
}
```

# 菜单组件

## 1．JMenuBar菜单条

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：

| 方法名称 | 说明 |
| --- | --- |
| public void setJMenuBar(JMenuBar menubar) | 将菜单条添加到窗口的菜单条区域（注意：只能向窗口添加一个菜单条） |

# 菜单组件

## 2. JMenu菜单

JComponent类的子类JMenu类是负责创建菜单的，即JMenu的一个实例就是一个菜单。JMenu类的主要方法有以下几种：

| 构造方法 | 说明 |
| --- | --- |
| JMenu(String s) | 建立一个指定标题菜单，标题由参数s确定 |

| 方法名称 | 说明 |
| --- | --- |
| public void add(JMenuItem item) | 向菜单增加由参数item指定的菜单选项对象 |
| public void add(String s) | 创建一个新的 JMenuItem，其标签为 s，然后将其添加到菜单中 |
| | |
| | |

# 菜单组件

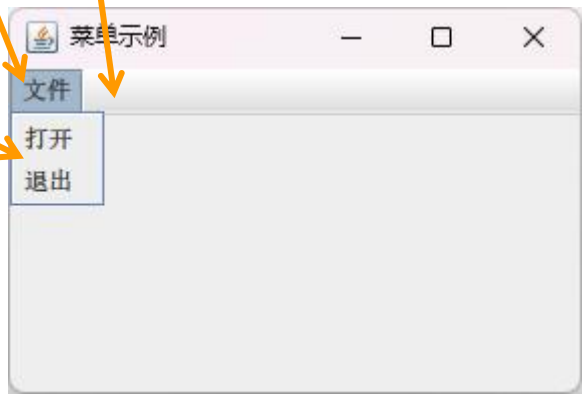
## 3. JMenuItem菜单项

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。菜单项放在菜单里。

| 构造方法 | 说明 |
| --- | --- |
| **JMenuItem(String s)** | 构造有标题的菜单项 |
| **JMenuItem(String text, Icon icon)** | 构造有标题和图标的菜单项 |

| 方法名称 | 说明 |
| --- | --- |
| 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 |

```java
public class MenuExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame( title: "菜单示例");// 创建JFrame对象
        JMenuBar menuBar = new JMenuBar();// 创建菜单条
        JMenu fileMenu = new JMenu( s: "文件");// 创建菜单
        // 创建菜单项
        JMenuItem openItem = new JMenuItem( text: "打开");
        JMenuItem exitItem = new JMenuItem( text: "退出");
        // 将菜单项添加到菜单中
        fileMenu.add(openItem);
        fileMenu.add(exitItem);
        // 将菜单添加到菜单条中
        menuBar.add(fileMenu);
        // 将菜单条设置到JFrame中
        frame.setJMenuBar(menuBar);
        // 设置窗口的一些属性
        frame.setSize( width: 300,  height: 200);
        frame.setVisible(true);
    }
}
```



## 10.4 布局设计

当把组件添加到容器中时，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。

本节将介绍java.awt包中的FlowLayout、BorderLayout、CardLayout、GridLayout布局类和java.swing.border包中的BoxLayout布局类。

**容器**可以使用方法

**setLayout(布局对象);**

来设置自己的布局。

对于JFrame窗口，默认布局是BorderLayout布局。

# 1．FlowLayout布局

| 构造方法 | 说明 |
| --- | --- |
| **FlowLayout()** | 创建一个居中对齐的布局对象 |

```
FlowLayout flow=new FlowLayout();
```

如果一个容器con使用**flow**这个布局对象：

**con.setLayout(flow);**

那么容器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。

| 方法名称 | 说明 |
| --- | --- |
| **setAlignment(int aligin)** | 重新设置布局的对齐方式，其中aligin可以取值FlowLayout.LEFT、FlowLayout.CENTER或FlowLayout.RIGHT |
| **setHgap(int hgap)** | 重新设置布局的水平间隙 |
| **setVgap(int vgap)** | 垂直间隙 |

```java
public class FlowLayoutExample {
    public static void main(String[] args) {
        // 创建JFrame对象
        JFrame frame = new JFrame( title: "FlowLayout示例");
        frame.setSize( width: 300,  height: 100);
        frame.setVisible(true);
        JPanel panel = new JPanel();// 创建JPanel对象作为容器
        frame.add(panel);// 将JPanel添加到JFrame中

        FlowLayout flow = new FlowLayout();// 创建FlowLayout对象
        panel.setLayout(flow);// 将FlowLayout设置给JPanel
        // 添加一些按钮组件到panel中
        panel.add(new javax.swing.JButton( text: "按钮1"));
        panel.add(new javax.swing.JButton( text: "按钮2"));
        panel.add(new javax.swing.JButton( text: "按钮3"));
    }
}
```

## 2．BorderLayout布局

BorderLayout布局将容器划分为五个区域：中心（CENTER）、北（NORTH）、南（SOUTH）、西（WEST）和东（EAST）。

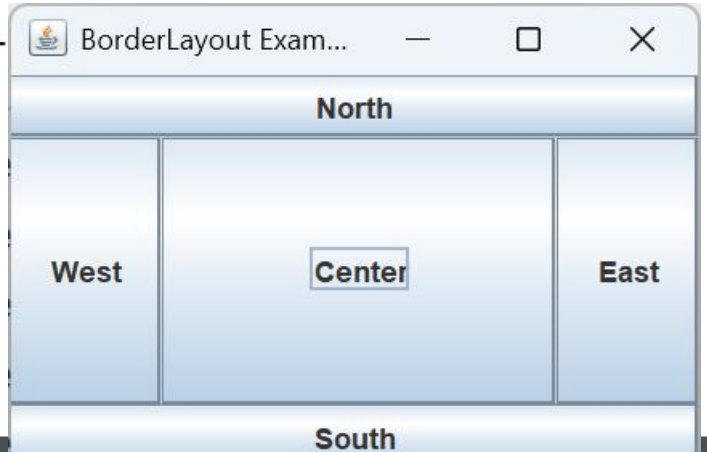
BorderLayout布局是Window容器的默认布局。JFrame、JDialog都是Window类的间接子类，它们的内容面板的默认布局都是BorderLayout布局。

一个使用BorderLayout布局的容器con,可以使用**add()方法**将一个组件b添加到中心区域：

```
con.add(b,BorderLayout.CENTER);
或 con.add(BorderLayour.CENTER,b);
```

```java
public class BorderLayoutExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame( title: "BorderLayout Example");
        frame.setSize( width: 300,  height: 200);
        frame.setVisible(true);

        // 添加组件到不同的区域
        frame.add(new JButton( text: "Center"), BorderLayout.CENTER);
        frame.add(new JButton( text: "North"), BorderLayout.NORTH);
        frame.add(new JButton( text: "South"), BorderLayout.SOUTH);
        frame.add(new JButton( text: "West"), BorderLayout.WEST);
        frame.add(new JButton( text: "East"), BorderLayout.EAST);
    }
}
```

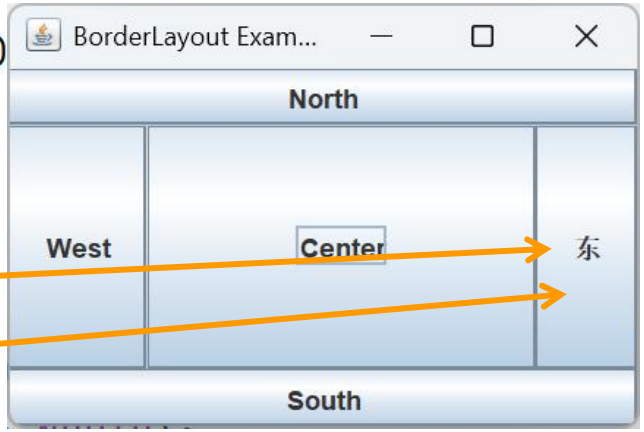


## 2．BorderLayout布局

**注:** 添加到某个区域的组件将占据整个这个区域.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。

**注:** 使用BorderLayout布局的容器最多能添加5个组件，如果容器中需要添加的组件超过5个，就必须使用容器的嵌套或改用其他布局策略。

```
// 添加组件到不同的区域
frame.add(new JButton( text: "Center"), BorderLayout.CENTER);
frame.add(new JButton( text: "North"), BorderLayout.NORTH);
frame.add(new JButton( text: "South"), BorderLayout.SOUTH);
frame.add(new JButton( text: "West"), BorderLayout.WEST);
frame.add(new JButton( text: "East"), BorderLayout.EAST);
frame.add(new JButton( text: "东"), BorderLayout.EAST);
```



## 3．CardLayout 布局

CardLayout容器可以容纳多个组件,但同一时刻容器只能从这些组件中选出一个来显示,就像一叠“扑克牌”每次只能显示最上面的一张一样,这个被显示的组件将占据所有的容器空间。

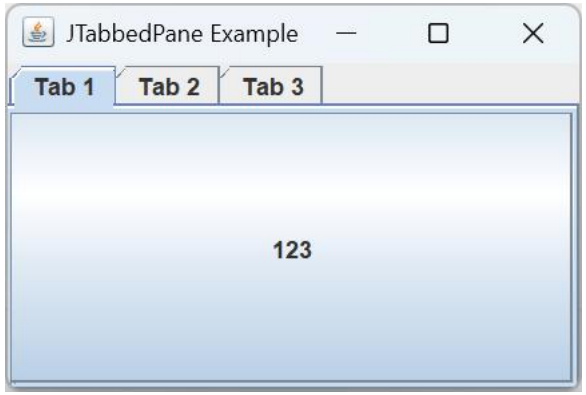
JTabbedPane窗格的默认布局是CardLayout布局，并且自带一些选项卡,这些选项卡与用户添加到JTabbedPane窗格中的组件相对应，单击该选项卡，JTabbedPane窗格将显示对应的组件。

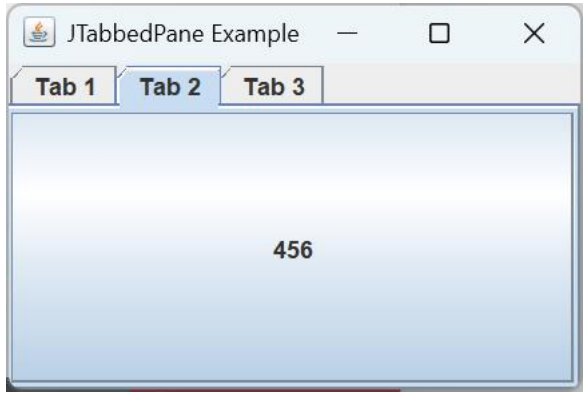
# JTabbedPane

| 构造方法 | 说明 |
| --- | --- |
| public JTabbedPane(int tabPlacement) | 选项卡的位置可以通过构造方法JTabbedPane(int tabPlacement)来指定,该参数的有效值为JTabbedPane.TOP、JTtabbedPane.BOTTOM、JTabbedPane.LEFT 和JTabbedPane.RIGHT |

| 方法名称 | 说明 |
| --- | --- |
| add(String text,Component c); | 将组件c添加到JTabbedPane窗格中，并指定和组件c对应的选项卡的文本提示是text |

```java
public class JTabbedPaneExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame( title: "JTabbedPane Example");
        frame.setVisible(true);
        frame.setSize( width: 300,  height: 200);
        // 创建JTabbedPane实例，选项卡位于顶部
        JTabbedPane p = new JTabbedPane(JTabbedPane.TOP);
        // 添加组件和对应的选项卡
        p.addTab( title: "Tab 1", new JButton( text: "123"));
        p.addTab( title: "Tab 2", new JButton( text: "456"));
        p.addTab( title: "Tab 3", new JButton( text: "789"));
        // 将JTabbedPane添加到窗口中
        frame.add(p);
    }
}
```



## 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若干列的网格区域，组件就位于这些划分出来的小格中。

| 构造方法 | 说明 |
| --- | --- |
| public GridLayout(int m,int n) | 指定划分网格的行数m和列数n |

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从做到右，从上到下。

使用GridLayout布局的容器最多可添加m×n个组件。GridLayout布局中每个网格都是相同大小，并且强制组件与网格的大小相同。

```java
public class GridLayoutExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame( title: "GridLayout Example");
        frame.setVisible(true);
        frame.setSize( width: 300,  height: 200);
        // 创建使用GridLayout的面板
        JPanel panel = new JPanel(new GridLayout( rows: 3,  cols: 2)); // 3行2列
        frame.add(panel);
        // 添加组件到面板
        for (int i = 1; i <= 6; i++) {
            panel.add(new JButton( text: "按钮 " + i));
        }
    }
}
```

## 5．BoxLayout布局

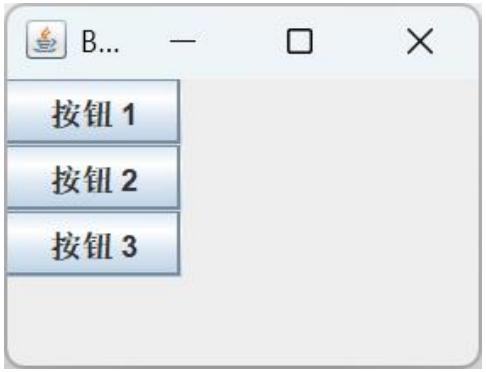
用BoxLayout类可以创建一个布局对象,称为盒式布局。

java swing包提供了Box类,该类也是Container类的一个子类，创建的容器称为一个**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。

| 构造方法 | 说明 |
| --- | --- |
| BoxLayout(Container target，int axis) | 为指定的容器target创建一个盒式布局对象。参数axis的有效值是BoxLayout.X_AXIS代表水平布局 BoxLayout.Y_AXIS代表垂直布局 |

```java
public class BoxLayoutExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame( title: "BoxLayout Example");
        frame.setVisible(true);
        frame.setSize( width: 200,  height: 150);
        // 创建一个面板
        JPanel panel = new JPanel();
        // 使用BoxLayout为面板设置布局，这里设置为垂直布局
        panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
        // 向面板添加组件
        panel.add(new JButton( text: "按钮 1"));
        panel.add(new JButton( text: "按钮 2"));
        panel.add(new JButton( text: "按钮 3"));
        // 将面板添加到框架中
        frame.add(panel);
    }
}
```



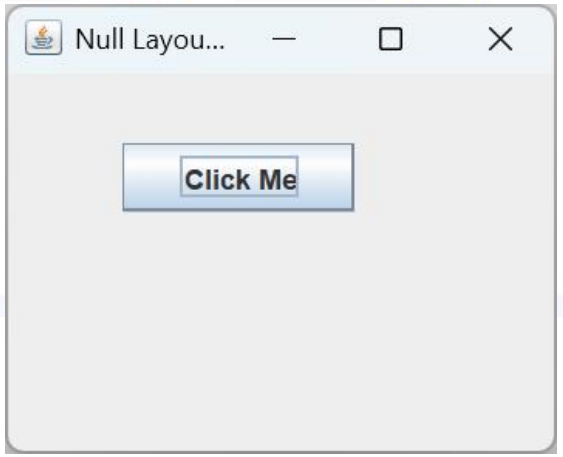
## 6．null布局

在Java Swing中，可以将一个容器的布局管理器设置为null，这样容器就不再使用任何布局管理器来自动排列组件。这种布局通常被称为“绝对定位”或“空布局”。

使用空布局时，您可以使用组件的setBounds方法来精确控制组件在容器中的位置和大小。

| 方法名称 | 说明 |
| --- | --- |
| setBounds(int a,int b,int width,int height) | 组件调用该方法可以设置本身的大小和在容器中的位置 |

```java
public class NullLayoutExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame( title: "Null Layout Example");
        frame.setVisible(true);
        frame.setSize( width: 250,  height: 200);
        // 设置空布局
        frame.setLayout(null);
        // 创建一个按钮组件
        JButton button = new JButton( text: "Click Me");
        // 设置按钮的位置和大小
        button.setBounds( x: 50,  y: 30,  width: 100,  height: 30);
        // 将按钮添加到框架中
        frame.add(button);
    }
}
```



# 10.6 文本组件

JComponent的子类JTextField是专门用来建立文本框的,即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。

| 构造方法 | 说明 |
| --- | --- |
| public JTextField(int x) | 创建文本框对象，可以在文本框中输入若干个字符，文本框的可见字符个数由参数 x指定 |
| public JTextField(String s) | 创建文本框对象，则文本框的初始字符串为s，可以在文本框中输入若干个字符 |

| 方法名称 | 说明 |
| --- | --- |
| public void setText(String s) | 设置文本框中的文本为参数s指定的文本，文本框中先前的文本将被清除 |
| public String getText() | 获取文本框中的文本 |
| public void setEditable(boolean b) | 指定文本框的可编辑性。创建的文本框默认是可编辑的 |

# JTextField

```
public class SimpleJTextField {
    public static void main(String[] args) {
        // 创建 JFrame 实例
        JFrame frame = new JFrame( title: "文本框");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize( width: 300,  height: 100);
        // 创建 JTextField 实例并添加到 JFrame
        JTextField textField = new JTextField("请输入...");
        frame.add(textField);
        // 显示窗口
        frame.setVisible(true);
    }
}
```

# JPasswordField密码框

使用JTextField的子类JPasswordField可以建立一个密码框对象。

```
public class JPasswordFieldExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame( title: "密码框测试");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize( width: 300,  height: 100);
        // 创建 JPasswordField 实例
        JPasswordField pw = new JPasswordField( columns: 20);
        // 将密码框添加到窗口
        frame.add(pw);
        // 显示窗口
        frame.setVisible(true);
    }
}
```

## ActionEvent事件

当用户在有输入焦点的文本框中按回车键、单击按钮、在一个下拉式列表中选择一个条目等操作时，都发生界面事件。程序有时要对发生的事件做出反应，来实现特定的任务。

在学习处理事件时，读者必须很好地掌握**事件源、监视器、处理事件的接口**这三个概念。

JTextField和JPasswordField触发**ActionEvent事件**，通过处理文本框这个具体的组件上的事件，来掌握**处理事件的基本原理**。

# ActionEvent事件_处理事件的基本原理

1)事件源: **能够产生事件的对象**都可以称为**事件源**,如文本框、按钮、下拉式列表等。

2)监视器: 我们需要一个**对象**对**事件源**进行监视,以便对发生的事件做出处理。

**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(ActioListener listener)

对于获取了监视器的文本框对象,在文本框获得输入焦点之后,如果用户按回车键,Java运行系统就自动用ActionEvent类创建了一个对象,即发生了ActionEvent事件。

## ActionEvent事件_处理事件的基本原理

也就是说,**事件源**获得**监视器**之后,**相应的操作**就会导致**事件的发生**,并通知监视器,监视器就会做出**相应的处理**。

3)**处理事件的接口:** 发生ActionEvent事件的事件源对象获得监视器方法是:

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**传递给**该方法的参数**,使得该对象成为事件源的**监视器**。

监视器负责调用特定的方法处理事件,创建监视器的类必须提供处理事件的特定方法,即**实现接口方法**。

# ActionEvent事件_ ActionEvent类中的方法

## 4)ActionEvent类中的方法

| 方法名称 | 说明 |
| --- | --- |
| public Object getSource() | 返回发生ActionEvent事件的对象的引用 |
| public String getActionCommand() | 获取发生ActionEvent事件时，和该事件相关的一个命令字符串 |

对于JTextField返回的是**文本内容**


```
"C:\Program Files\Ja
文本框中的文本: hi
文本框中的文本: hi2
```

## 事件处理模型

**当需要对组件的某种事件进行响应和处理时，程序员必须完成两个步骤：**

1) 为组件注册**实现规定接口**的事件监听器；
2) **实现**事件监听器接口中声明的事件处理**抽象方法**。

例如, button组件的事件响应过程:

```
JButton button=new JButton("Press ");
//(1)为button组件注册事件监听器
button.addActionListener(new ButtonHandler());
```

**鼠标单击**，触发事件
ActionEvent
将调用方法actionPerformed

```
Class  ButtonHandler  implements  ActionListener {//实现事件监听接口的类
  public void actionPerformed( ActionEvent  event )  //(2)实现事件处理方法。
{ …//事件处理代码 }
}
```

# 事件处理模型

## ActionEvent动作事件

ActionEvent:当特定组件（例如Button）的动作（例如点击按钮）发生时，由组件生成此动作事件。

该事件被传递给使用组件的 addActionListener 方法注册的每一个 ActionListener 对象，用以接收这类事件.

能够触发这个事件的动作，主要包括：

(1) 点击按钮

(2) 双击一个列表中的选项；

(3) 选择菜单项；

(4) 在文本框中输入回车。

# 处理事件的接口

```
// 定义 MyActionListener 类，实现 ActionListener 接口
class MyActionListener implements ActionListener {
    // 实现 actionPerformed 方法，这是事件监听器的核心方法
    @Override
    public void actionPerformed(ActionEvent e) {
        // 从事件对象中获取动作命令，对于 JTextField，通常是用户输入的文本
        String command = e.getActionCommand();
        // 打印用户输入的文本到控制台
        System.out.println("文本框中的文本: " + command);
    }
}
```

实现接口方法

发生的事件是ActionEvent，它是由用户在JTextField中按下回车键触发的。

当用户在文本框中输入文本并按下 Enter 键时，MyActionListener 的 actionPerformed 方法将被调用，打印出文本框中的文本。

```java
public class ActionEventExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame( title: "ActionEvent Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize( width: 300,  height: 200);
        // 创建 MyActionListener 类的实例，作为事件监听器
        MyActionListener listener = new MyActionListener();
        // 创建 JTextField 实例，用于输入文本
        JTextField textField = new JTextField();
        // 为文本框添加动作监听器，当文本框中发生动作时将调用监听器的方法
        textField.addActionListener(listener);
        // 将文本框添加到窗口的中心位置
        frame.add(textField, BorderLayout.CENTER);
        // 设置窗口为可见
        frame.setVisible(true);
    }
}
```

监视器

事件源

监视器

## 匿名内部类：

- 本质上是一个隐藏了名字的内部类。
- **作用**：方便创建子类对象，最终目的是为了简化代码编写。

## 格式：

```
new 类名或者接口名() {
    重写方法;
};
```

继承/实现

方法重写

创建对象

```
Employee a = new Employee() {
    public void work() {
    }
};
a. work();
```

## 特点总结：

- 匿名内部类是一个**没有名字**的内部类，同时也代表一个对象。
- 匿名内部类产生的对象类型，相当于是当前new的那个的类型的**子类类型**。

```java
// 创建 JTextField 实例，用于输入文本
JTextField textField = new JTextField();
// 为文本框添加动作监听器，当文本框中发生动作时将调用监听器的方法
textField.addActionListener(new ActionListener(){
    @Override
    public void actionPerformed(ActionEvent e) {
        // 从事件对象中获取动作命令，对于 JTextField，通常是用户输入的文本
        String command = e.getActionCommand();
        // 打印用户输入的文本到控制台
        System.out.println("文本框中的文本: " + command);
    }
});
// 将文本框添加到窗口的中心位置
frame.add(textField, BorderLayout.CENTER);
```

# 以2为底的对数计算器

```
    public static void main(String[] args) {
        LogFrame demo = new LogFrame();
        demo.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        demo.setVisible(true);
        demo.setSize( width: 400,  height: 100);
    }
}
```


```java
public class LogFrame extends JFrame {
    private JTextField textFieldNumber, textFieldResult;
    private JButton buttonCalculate;
    LogFrame() {
        super( title: "以2为底的对数计算器"); // 设置窗口标题
        setLayout(new FlowLayout()); // 设置布局
        add(new JLabel( text: "请输入正数："));
        add(textFieldNumber = new JTextField( columns: 3));
        add(buttonCalculate = new JButton( text: "计算"));
        add(new JLabel( text: " 结果为："));
        add(textFieldResult = new JTextField( columns: 12));
        buttonCalculate.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // 从文本字段获取输入的数
                double number = Double.parseDouble(textFieldNumber.getText());
                // 计算以2为底的对数
                double logBase2 = Math.log(number) / Math.log(2);
                // 将结果显示在文本字段中
                textFieldResult.setText(String.valueOf(logBase2));
            }
        }); // 添加监听器
    }
```

# 复习 字符串操作

```
public class Test0 {
    public static void main(String[] args) {
        String data = "apple,banana,cherry";
        // 按逗号拆分
        String[] fruits = data.split( regex: ",");
        for (String fruit : fruits) {
            System.out.println(fruit);
        }
    }
}
```

# 复习 定义一个方法来返回一个包含偶数的数组

```java
public class EvenNumbersArray {
    // 定义一个方法，返回一个包含偶数的整数数组
    public int[] getEvenNumbersArray(int size) {
        // 如果size不是正数，返回一个空数组
        if (size <= 0) {
            return new int[0];
        }
        // 创建一个整数数组，初始化为偶数
        int[] evenNumbers = new int[size];
        for (int i = 0; i < size; i++) {
            // 从2开始，每次增加2，得到偶数
            evenNumbers[i] = 2 * (i + 1);
        }
        return evenNumbers;
    }
```

# end