JAVA程序员培训-3

讲师：陈伟俊

第十二章

GUI应用程序设计

本章内容

- Java抽象窗口工具集(AWT)
- Java组件和容器
- 常用组件类型
- 布局管理器及其使用

抽象窗口工具集(AWT)

- AWT-- AbstractWindow Toolkit
- GUI -- Graphical User Interface
- AWT中定义了多种类和接口，用于在Java Application/Applet中进行GUI设计
- java程序要显示的GUI组件必须是抽象类Component或MenuComponent的子类

java.awt 包

- java.awt包

  提供了基本的java程序GUI设计工具。

  - Component/MenuComponent
  - Container
  - *LayoutManager*

组件（Component）

- Java的图形用户界面的最基本组成部分是组件，组件是一个可以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，例如一个按钮，一个标签等。
- 组件不能独立地显示出来，必须将组件放在一定的容器中才可以显示出来。

容　器(Container)

- 容器(Container)实际上是Component的子类，因此容器类对象本身也是一个组件，具有组件的所有性质，另外还具有容纳其它组件和容器的功能。
- 容器类对象可使用方法add()添加组件
- 两种主要的容器类型
  - Window: 可自由停泊的顶级窗口
  - Panel:可作为容器容纳其它组件，但不能独立存在，必须被添加到其它容器中(如Window 或 Applet)

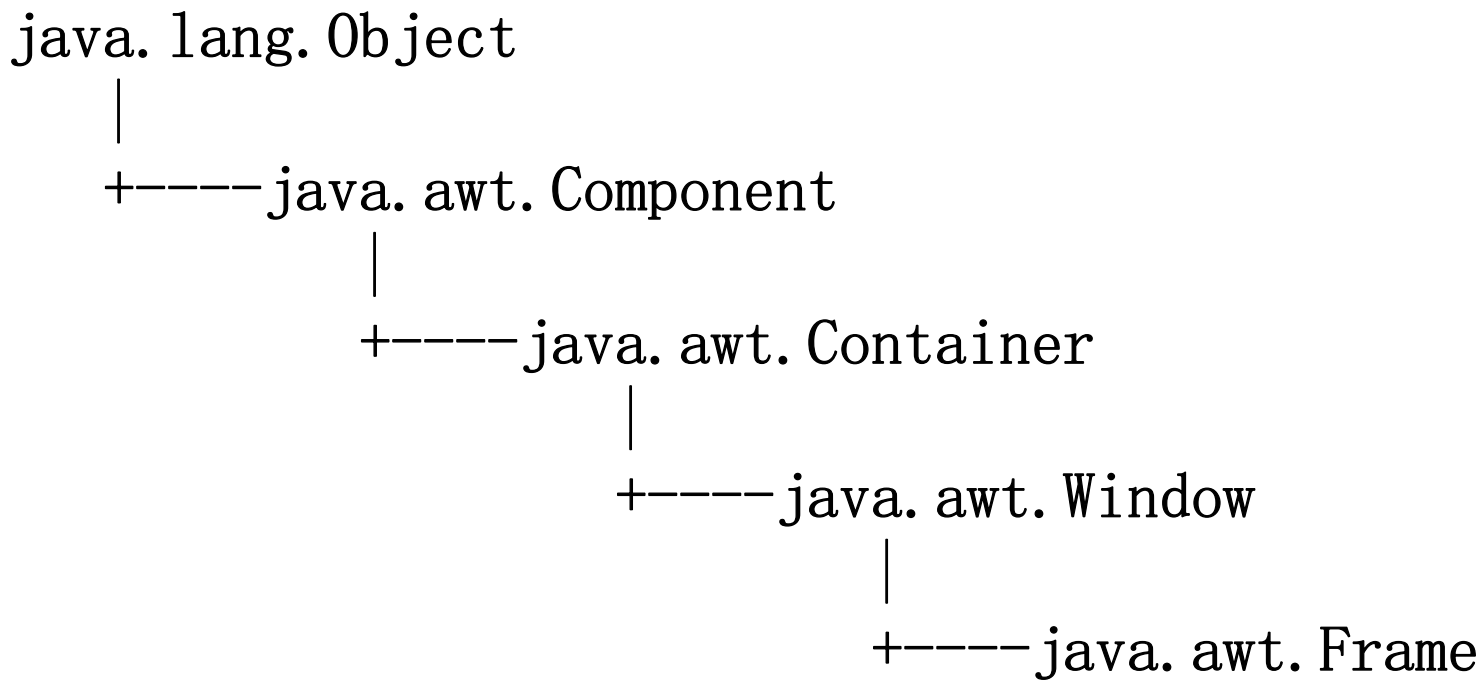
组件定位

- Java组件在容器中的位置和尺寸由布局管理器决定
- 如要人工控制组件在容器中的大小位置，可取消布局管理器，然后使用Component类的下述成员方法：
  - setLocation()
  - setSize()
  - setBounds()

Frame类

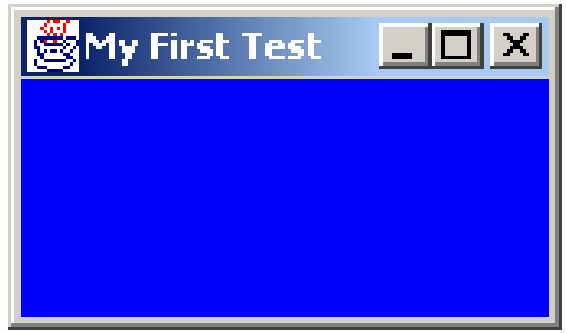
- Frame类是抽象类Window的子类
- Frame对象显示效果是一个“窗口”，带有标题和尺寸重置角标
- 默认初始化为不可见的，可使用setVisible(true)方法使之变为可见
- 默认的布局管理器是BorderLayout
- 可使用setLayout()方法改变其默认布局管理器

Frame类继承层次


java.lang.Object
java.awt.Component
java.awt.Container
java.awt.Window
java.awt.Frame

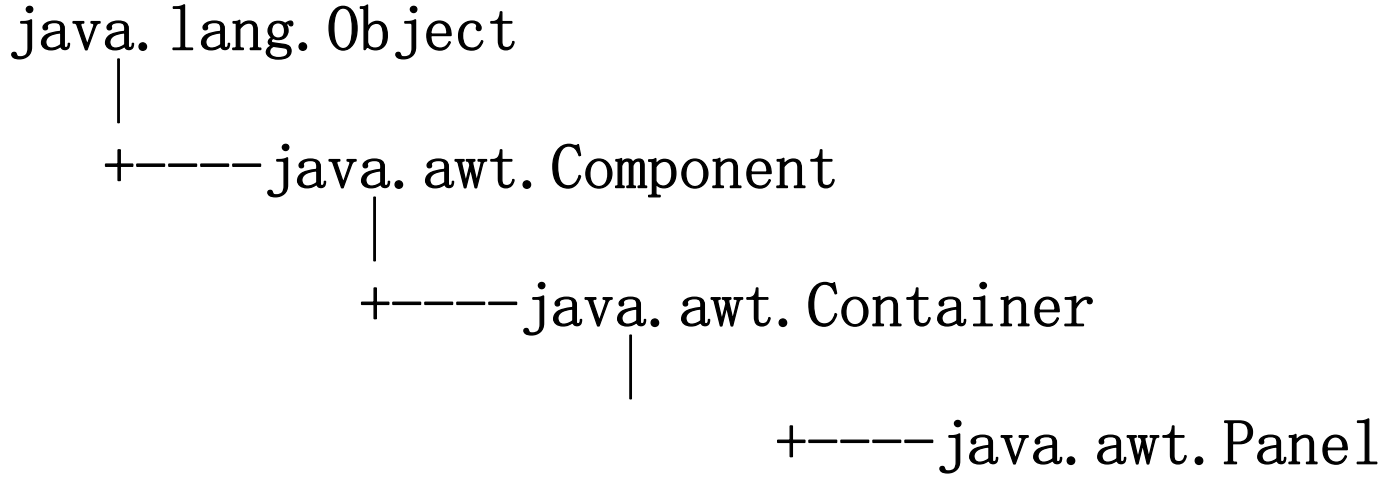
Frame 应用举例

```
import java.awt.*;
    public class TestFrame {
        public static void main( String args[])
    {
          Frame f = new Frame("My First Test");
              f.setSize( 170,100);
        f.setBackground( Color.blue);
        f.setVisible( true)
     }
    }
```



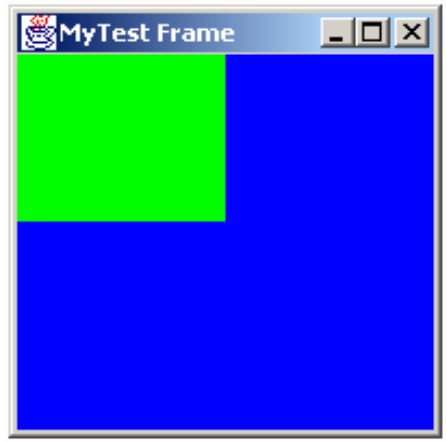
Panel类

- 提供容纳组件的空间
- 可以采用和所在容器不同的布局管理器
- Panel类的继承层次



FrameWithPanel应用举例

```
import java.awt.*;
public class TestFrameWithPanel {
        public static void main(String args[]) {
    Frame f = new Frame("MyTest Frame");
           Panel pan = new Panel();
             f.setSize(200,200);
           f.setBackground(Color.blue);
           f.setLayout(null); // 取消布局管理器
           pan.setSize(100,100);
           pan.setBackground(Color.green);
           f.add(pan);
           f.setVisible(true);
        }
}
```



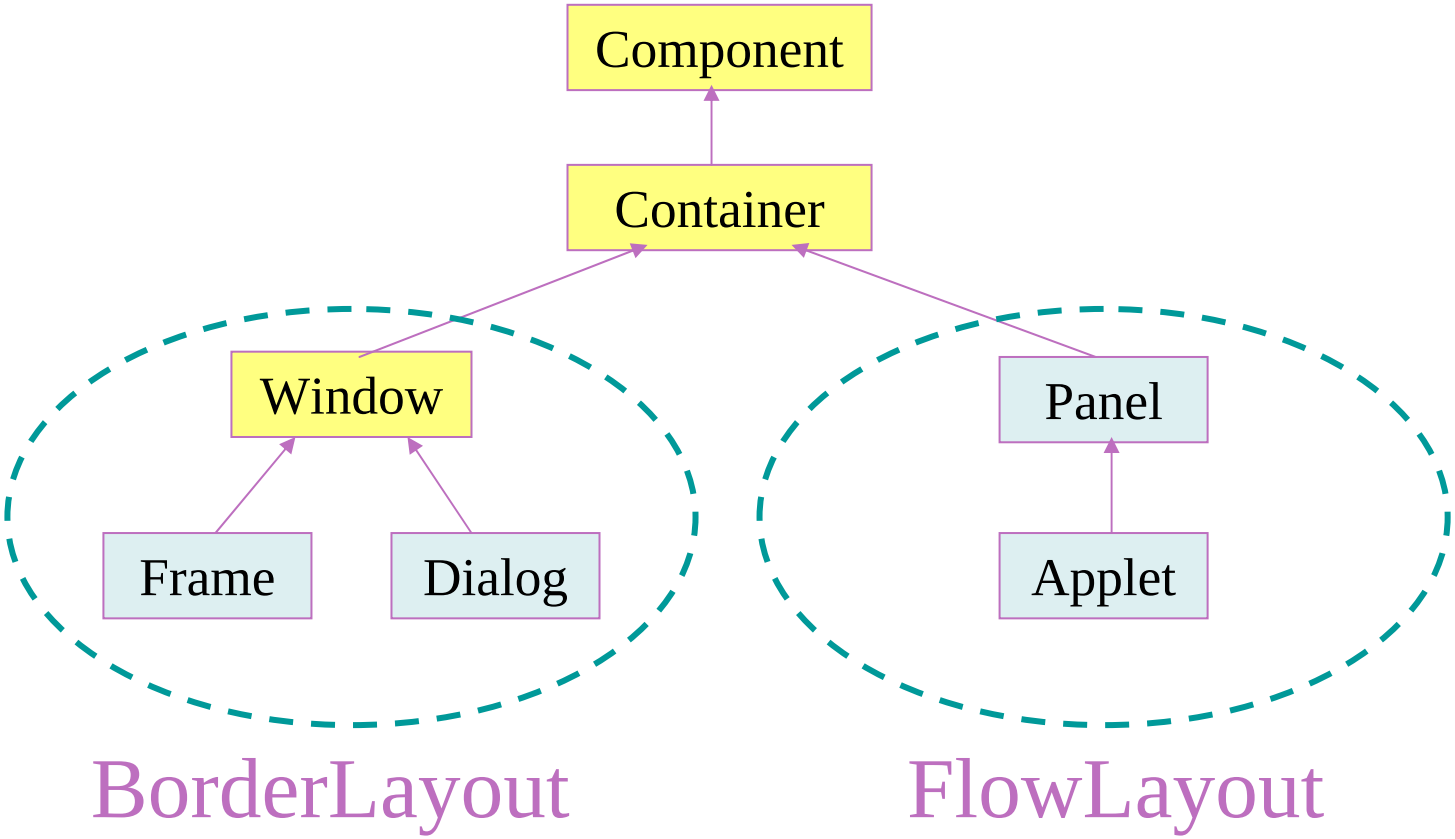
布局管理器

- 为了使我们生成的图形用户界面具有良好的平台无关性，Java语言中，提供了布局管理器这个工具来管理组件在容器中的布局，而不使用直接设置组件位置和大小的方式。
- 每个容器都有一个布局管理器，当容器需要对某个组件进行定位或判断其大小尺寸时，就会调用其对应的布局管理器。

容器布局（Container Layouts）

- FlowLayout
- BorderLayout
- GridLayout
- CardLayout
- GridBagLayout

默认布局管理器

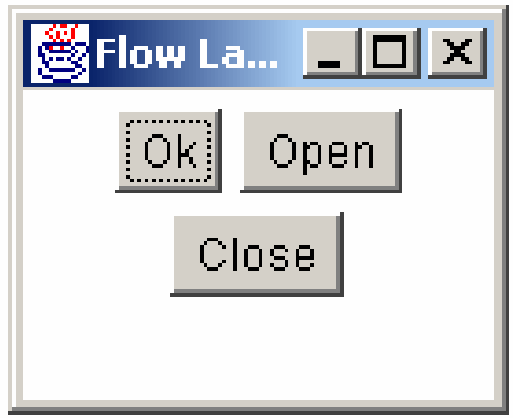

Component
Container
Window
Panel
Frame
Dialog
Applet
BorderLayout
FlowLayout

FlowLayout布局管理器

- FlowLayout是Panel类的默认布局管理器
- FlowLayout布局对组件逐行定位，行内从左到右，一行排满后换行
- 默认对齐方式为居中对齐
- 不改变组件的大小，按组件原有尺寸显示组件 可在构造方法中设置不同的组件间距、行距及对齐方式

FlowLayout 举例

```
import java.awt.*;
public class TestFlowLayout {
        public static void main(String args[]) {
                Frame f = new Frame("Flow Layout");
                  Button button1 = new Button("Ok");
                  Button button2 = new Button("Open");
                  Button button3 = new Button("Close");
                      f.setLayout(new FlowLayout());
                  f.add(button1);
                  f.add(button2);
                  f.add(button3);
                  f.setSize(100,100);
                  f.setVisible(true);
        }
}
```



FlowLayout 的构造方法

- new FlowLayout(FlowLayout.RIGHT,20,40);

  右对齐，组件之间水平间距20个像素，竖直间距40个像素；

- new FlowLayout(FlowLayout.LEFT);

  左对齐，水平和竖直间距为缺省值：5；

- new FlowLayout();

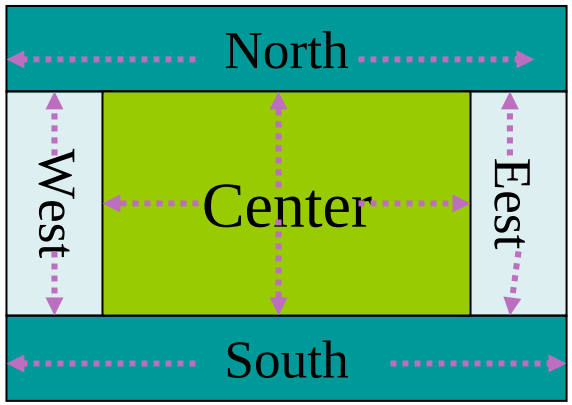
  使用缺省的居中对齐方式，水平和竖直间距为缺省值：5；

BorderLayout 布局管理器

- BorderLayout是Frame类的默认布局管理器
- BorderLayout将整个容器的布局划分成东、西、南、北、中五个区域，组件只能被添加到指定的区域
- 如不指定组件的加入部位，则默认加入到Center区域
- 每个区域只能加入一个组件，如加入多个，则先前加入的组件会被遗弃

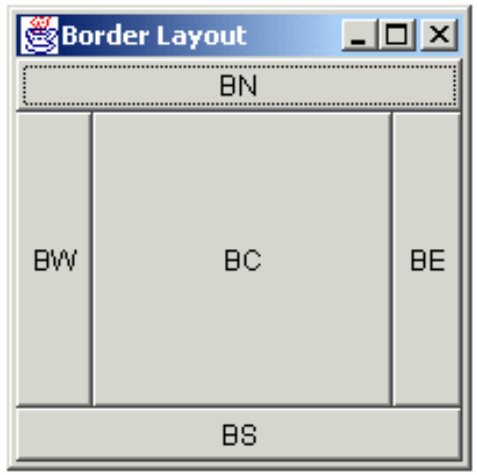
BorderLayout 布局管理器

- BorderLayout型布局容器尺寸缩放原则
  - 北、南两个区域只能在水平方向缩放(宽度可调整)
  - 东、西两个区域只能在垂直方向缩放(高度可调整)
  - 中部可在两个方向上缩放



BorderLayout举例

```
import java.awt.*;
public class TestBorderLayout {
   public static void main(String args[]) {
        Frame f;
        f = new Frame("Border Layout");
        Button bn = new Button("BN");
        Button bs = new Button("BS");
        Button bw = new Button("BW");
        Button be = new Button("BE");
        Button bc = new Button("BC");
        f.add(bn, "North");
        f.add(bs, "South");
        f.add(bw, "West");
        f.add(be, "East");
        f.add(bc, "Center");
        f.setSize(200,200);
        f.setVisible(true);
   }
}
```



GridLayout 布局管理器

- GridLayout型布局管理器将布局划分成规则的矩形网格，每个单元格区域大小相等.
- 组件被添加到每个单元格中，先从左到右添满一行后换行，再从上到下.
- 在GridLayout构造方法中指定分割的行数和列数.
- new GridLayout(3,4);

GridLayout举例

```
import java.awt.*;
public class TestGridLayout {
        public static void main(String args[]) {
            Frame f = new Frame("GridLayout Example");
            Button b1 = new Button("b1");
            Button b2 = new Button("b2");
            Button b3 = new Button("b3");
            Button b4 = new Button("b4");
            Button b5 = new Button("b5");
            Button b6 = new Button("b6");
                f.setLayout (new GridLayout(3,2));
            f.add(b1);          f.add(b2);
            f.add(b3);          f.add(b4);
            f.add(b5);          f.add(b6);
            f.pack();           f.setVisible(true);
        }
}
```

* CardLayout 布局管理器

- CardLayout布局管理器能够帮助用户处理两个以至更多的成员共享同一显示空间，就好象一叠卡片摞在一起。
- 注意：在一张卡片中只能显示一个组件，因此可以使用容器嵌套方法显示多个组件。

//非考试内容，但实际项目开发有时会用

* GridBagLayout 布局管理器

- AWT中最灵活、最复杂的布局管理器，各组件所占空间可以不相同且灵活规定，参见中文参考书;
- 非考试内容，很少用，实际项目开发做界面时可能会用到。

容器的嵌套使用举例

```
import java.awt.*;
public class NestedContainer {
    public static void main(String args[]) {
        Frame f = new Frame("NestedContainer");
        Button b0 = new Button("display Area");
        Panel p = new Panel();
        p.setLayout(new GridLayout(2,2));
        Button b1 = new Button("1");
        Button b2 = new Button("2");
        Button b3 = new Button("3");
        Button b4 = new Button("4");
        p.add(b1);      p.add(b2);
        p.add(b3);      p.add(b4);
                        f.add(b0, "North");
        f.add(p, "Center");
        f.pack();
        f.setVisible(true);
    }
}
```

布局管理器总结(1)

- Frame
  - Frame是一个顶级窗口。
  - Frame的缺省布局管理器为BorderLayout。
- Panel
  - Panel无法单独显示，必须添加到某个容器中。
  - Panel的缺省布局管理器为FlowLayout。
  - 当把Panel作为一个组件添加到某个容器中后，该Panel仍然可以有自己的布局管理器。因此，可以利用Panel使得BorderLayout中某个区域显示多个组件。

布局管理器总结(2)

- 在程序中安排组件的位置和大小时，应注意：
  - 容器中的布局管理器负责各个组件的大小和位置，因此用户无法在这种情况下设置组件的这些属性。如果试图使用Java语言提供的setLocation()，setSize()，setBounds()等方法，则都会被布局管理器覆盖。
  - 如果用户确实需要亲自设置组件大小或位置，则应取消该容器的布局管理器，方法为：

    setLayout(null);

本章结束

第十三章

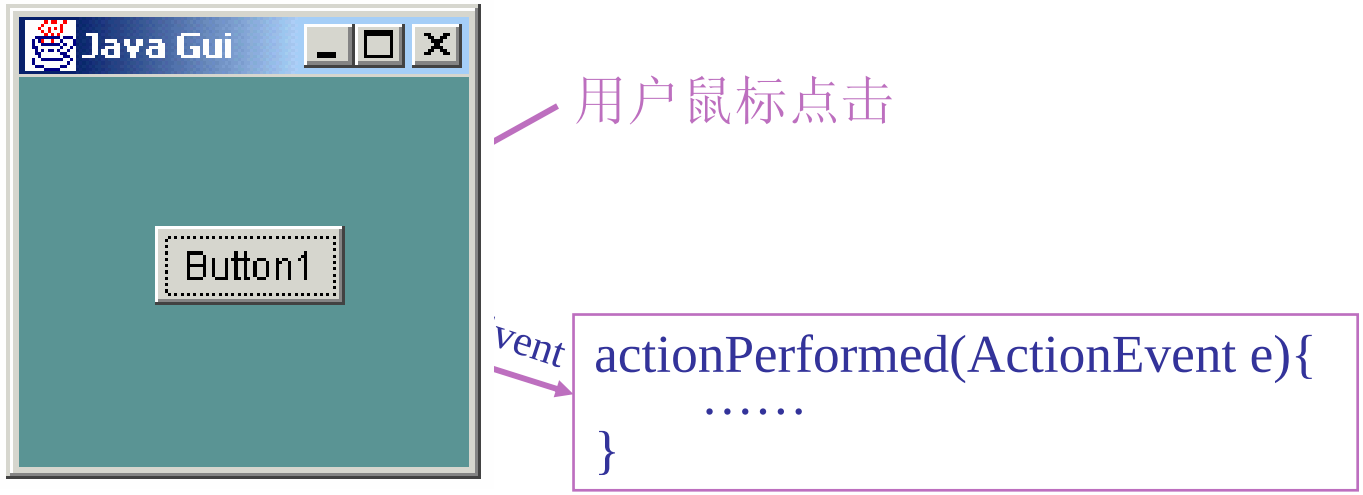
GUI事件处理

本章内容

- Java事件和事件处理
- 事件源、事件监听器、事件类型
- 事件监听器接口和事件适配器
- 内部类在Java事件处理中的应用

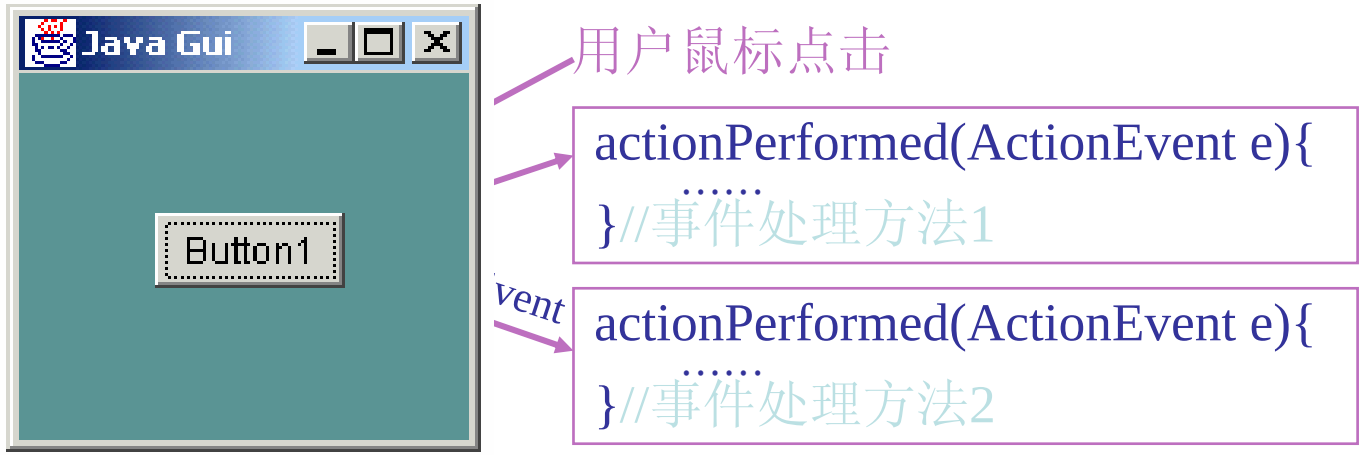
什么是事件?

- 事件(Event) – 一个对象，它描述了发生什么事情
- 事件源(Event source) – 产生事件的组件
- 事件处理方法(Event handler) – 能够接收、解析和处理事件类对象、实现和用户交互的方法



事件处理模型(Delegation Model)

- 一个事件可以被发送到多个不同的处理方法.



- 如果关注某个组件产生的事件 ，则可以在该组件上注册适当的事件处理方法，实际上注册的事件处理者方法所属类型的一个对象----事件监听器

Java事件处理举例(1)

```
import java.awt.*;
import java.awt.event.*;
public class TestActionEvent {
        public static void main(String args[]) {
          Frame f = new Frame("Test");
                Button b = new Button("Press Me!");
                Monitor bh = new Monitor();
                        b.addActionListener(bh);
                f.add(b,BorderLayout.CENTER);
                f.pack();
                f.setVisible(true);
        }
}
class Monitor implements ActionListener {
        public void actionPerformed(ActionEvent e) {
                 System.out.println("a button has been pressed");    }
}
```

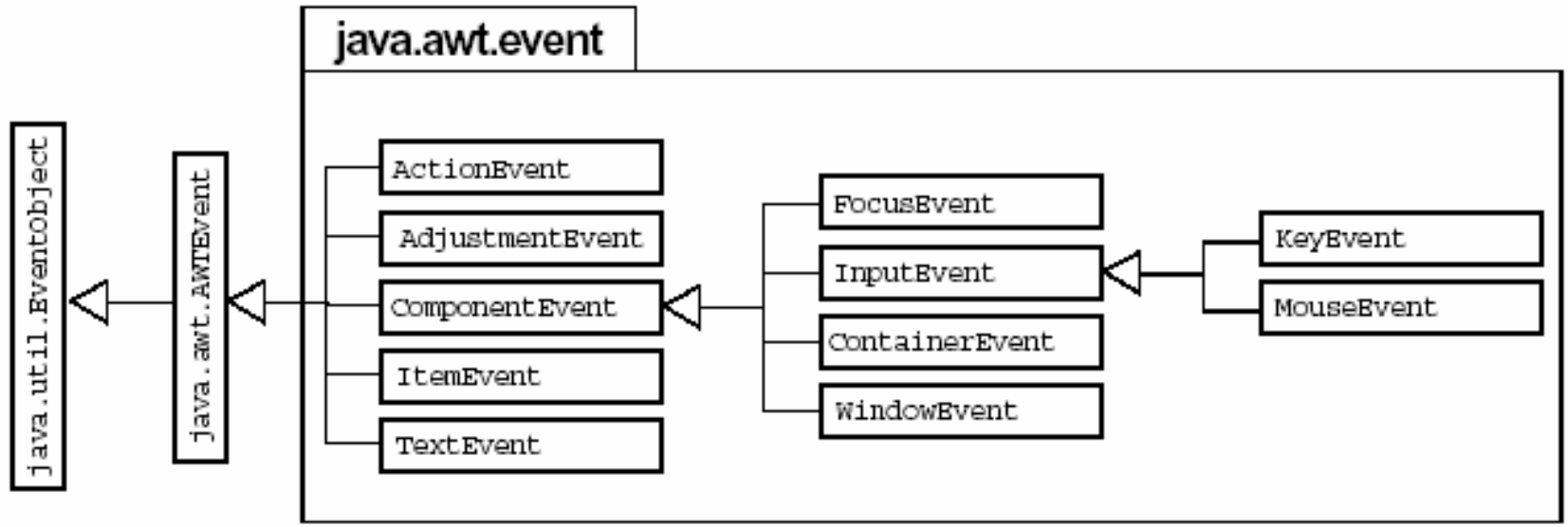
Java事件处理举例(2)

```
import java.awt.*;
import java.awt.event.*;
public class TestActionEvent2 {
        public static void main(String args[]) {
          Frame f = new Frame("Test");
                Button b1 = new Button("Start");
                Button b2 = new Button("Stop");
                Monitor bh2 = new Monitor2();
                        b1.addActionListener(bh);
       b2.addActionListener(bh);
              b2.setActionCommand("game over");
                f.add(b1,"North");       f.add(b2,"Center");
                f.pack();                    f.setVisible(true);
        }
}
class Monitor2 implements ActionListener {
           public void actionPerformed(ActionEvent e) {
                System.out.println("a button has been pressed, the relative
           info                                    is: " + e.getActionCommand());
         }
}
```

事件代理模型综述

- Java GUI设计中，通过注册监听器的方式对所关注的事件源进行监控。
- 注册监听器时应指明该监听器监控(感兴趣)的事件种类。
- 当事件源发生了某种类型的事件时，只触发事先已就该种事件类型注册过的监听器。

Java事件分类


java.awt.event
java.util.EventObject
java.awt.AWTEvent
ActionEvent
AdjustmentEvent
ComponentEvent
ItemEvent
TextEvent
FocusEvent
InputEvent
ContainerEvent
WindowEvent
KeyEvent
MouseEvent

Java GUI事件及相应监听器接口(1)

事件类型	相应监听器接口	监听器接口中的方法
Action	ActionListener)	actionPerformed(ActionEvent)
Item	ItemListener	itemStateChanged(ItemEvent)
Mouse	MouseListener	mousePressed(MouseEvent) mouseReleased(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mouseClicked(MouseEvent)
Mouse Motion	MouseMotionListener	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
Key	KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
Focus	FocusListener	focusGained(FocusEvent) focusLost(FocusEvent)

Java GUI事件及相应监听器接口(2)

事件类型	相应监听器接口	监听器接口中的方法
Adjustment	AdjustmentListener	adjustmentValueChanged (AdjustmentEvent)
Component	ComponentListener	componentMoved(ComponentEvent) componentHidden (ComponentEvent) componentResized(ComponentEvent) componentShown(ComponentEvent)
Window	WindowListener	windowClosing(WindowEvent) windowOpened(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent)
Container	ContainerListener	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
Text	TextListener	textValueChanged(TextEvent)

多重监听器

- 一般情况下，事件源可以产生多种不同类型的事件，因而可以注册(触发)多种不同类型的监听器。
- 一个事件源组件上可以注册多个监听器，针对同一个事件源的同一种事件也可以注册多个监听器，一个监听器可以被注册到多个不同的事件源上。

多重监听器举例(1)

```
import java.awt.*;
import java.awt.event.*;
public class TestMultiListener implements
   MouseMotionListener,MouseListener {
     Frame f = new Frame("多重监听器测试");
     TextField tf = new TextField(30);
     public TestMultiListener(){
        f.add(new Label("请按下鼠标左键并拖动"), "North");
        f.add(tf, "South");
        f.setBackground(new Color(120,175,175));
        f.addMouseMotionListener(this);
        f.addMouseListener(this);
        f.setSize(300, 200);              f.setVisible(true);
     }
     public static void main(String args[]) {
        TestMultiListener t = new TestMultiListener();
     }     //未完，接下页
```

多重监听器举例(2)

```
public void mouseDragged(MouseEvent e) {
        String s = "鼠标拖动到位置 (" + e.getX() + "," + e.getY() +
    ")";
        tf.setText(s);
    }
    public void mouseEntered(MouseEvent e) {
        String s = "鼠标已进入窗体";
        tf.setText(s);
    }
    public void mouseExited(MouseEvent e) {
        String s = "鼠标已移出窗体";
        tf.setText(s);
    }
    // 未使用的MouseMotionListener和MouseListener接口中的方法，也必须实
    现
    public void mouseMoved(MouseEvent e) { }
    public void mousePressed(MouseEvent e) { }
    public void mouseClicked(MouseEvent e) { }
    public void mouseReleased(MouseEvent e) { }
}
```

事件适配器(Event Adapter)

- 为简化编程，针对大多数事件监听器接口定义了相应的实现类----事件适配器类，在适配器类中，实现了相应监听器接口中所有的方法，但不做任何事情。
- 在定义监听器类时就可以继承事件适配器类，并只重写所需要的方法。

  - ComponentAdapter(组件适配器)
  - ContainerAdapter(容器适配器)
  - FocusAdapter(焦点适配器)
  - KeyAdapter(键盘适配器)
  - MouseAdapter(鼠标适配器)
  - MouseMotionAdapter(鼠标运动适配器)
  - WindowAdapter(窗口适配器)

事件适配器类举例

java.awt.event.MouseAdapter

Implemented Interfaces:EventListener, MouseListener

```
{
        ......
        mousePressed(MouseEvent e) {}
        mouseReleased(MouseEvent e)  {}
        mouseEntered(MouseEvent e) {}
        mouseExited(MouseEvent e)  {}
        mouseClicked(MouseEvent e) {}
}
```

事件适配器用法举例

定义一个监听器类，用于处理鼠标点击事件：

```
1     import java.awt.*;
2     import java.awt.event.*;
3
4     public class Monitor3 extends MouseAdapter {
5
6        // 当只需要处理mouseClick事件时，可以继承
7   // 事件适配器类MouseAdapter，以避免实现接口
8   // 时不得不重写接口中定义的所有方法
9
10       public void mouseClicked(MouseEvent e) {
11                // 处理代码.....
12       }
13    }
```

内部类在事件处理中的使用(1)

```
import java.awt.event.*;
public class TestInner {
   Frame f = new Frame("内部类测试");
   TextField tf = new TextField(30);
   public TestInner(){
        f.add(new Label("请按下鼠标左键并拖动"), "North");
        f.add(tf, "South");
        f.setBackground(new Color(120,175,175));
        InnerMonitor im = new InnerMonitor();
        f.addMouseMotionListener(im);
        f.addMouseListener(im);
        f.setSize(300, 200);
        f.setVisible(true);
   }
   public static void main(String args[]) {
        TestMultiListener t = new TestMultiListener();
   } //未完，接下页
```

内部类在事件处理中的使用(2)

```
class InnerMonitor implements MouseMotionListener,MouseListener {
        public void mouseDragged(MouseEvent e) {
            String s = "鼠标位置 (" + e.getX() + "," + e.getY() + ")";
            tf.setText(s);
        }
        public void mouseEntered(MouseEvent e) {
            String s = "鼠标已进入窗体";
            tf.setText(s);
        }
        public void mouseExited(MouseEvent e) {
            String s = "鼠标已移出窗体";
            tf.setText(s);
        }
        public void mouseMoved(MouseEvent e) { }
        public void mousePressed(MouseEvent e) { }
        public void mouseClicked(MouseEvent e) { }
        public void mouseReleased(MouseEvent e) { }
        }//end of inner class InnerMonitor
          }//end of class TestInner
```

本章结束

第十三章

建立GUI应用程序

本章内容

- Component类
- AWT常用组件
- AWT组件可产生的事件
- 为GUI应用程序添加菜单
- 控制GUI显示效果

Component类

- 抽象类Component是所有Java GUI组件的共同父类。
- Component类规定了所有GUI组件的基本特性，该类中定义的方法实现了作为一个GUI部件所应具备的基本功能。
- Component类（及其子类）中常用的属性和对应的操作属性的方法见下页表格

Component及其子类常用属性和方法

属性名称	设置属性的方法	获取属性的方法
背景颜色	void setBackground(Color)	Color getBackground()
边界	void setBounds(Rectangle) void setBounds(int,int,int,int)	Rectangle getBounds()
光标	void setCursor(Cursor)	Cursor getCursor()
拖放目标	void setDropTarget(DropTarget)	DropTarget getDropTarget()
使能	void setEnabled(boolean)	boolean isEnabled()
字体	void setFont(Font)	Font getFont()
前景色	void setForeground(Color)	Color getForeground()
地区	void setLocale(Locale)	Locale getLocale()
位置	void setLocation(Point) void setLocation(int,int)	Pont getLocation() Point getLocationOnScreen( )
部件名称	void setName(String)	String getName()
尺寸	void setSize(Dimension)	Dimension getSize()
可见性	void setVisible(boolean)	boolean getVisible()

常用AWT 组件

组件类型	说　明
Button	可接收点击操作的矩形GUI组件
Canvas	用于绘图的面板
Checkbox	复选框组件
CheckboxMenuItem	复选框菜单项组件
Choice	下拉式列表框，内容不可改变
Component	组件类
Container	容器类
Dialog	对话框组件，顶级窗口、带标题栏
Frame	基本的Java GUI窗口组件
Label	标签类
List	包含内容可变的条目的列表框组件
Menu	菜单组件
MenuItem	菜单项（二级菜单）组件
Panel	基本容器类，不能单独停泊
Scrollbar	滚动条组件
ScrollPane	带水平及垂直滚动条的容器组件
TextArea	多行文本域
TextField	单行文本框
Window	抽象的GUI窗口类，无布局管理器

各种组件可产生的事件

组件类型	Act	Adj	Cmp	Cnt	Foc	Itm	Key	Mou	MM	Text	Win
Button	☆		☆		☆		☆	☆	☆		
Canvas			☆		☆		☆	☆	☆		
Checkbox			☆		☆	☆	☆	☆	☆		
CheckboxMenuItem						☆					
Choice			☆		☆	☆	☆	☆	☆		
Component			☆		☆		☆	☆	☆		
Container			☆	☆	☆		☆	☆	☆		
Dialog			☆	☆	☆		☆	☆	☆		☆
Frame			☆	☆	☆		☆	☆	☆		☆
Label			☆		☆		☆	☆	☆		
List	☆		☆		☆	☆	☆	☆	☆		
MenuItem	☆										
Panel			☆	☆	☆		☆	☆	☆		
Scrollbar		☆	☆		☆		☆	☆	☆		
ScrollPane			☆	☆	☆		☆	☆	☆		
TextArea			☆		☆		☆	☆	☆	☆	
TextField	☆		☆		☆		☆	☆	☆	☆	
Window			☆	☆	☆		☆	☆	☆		☆

如何创建菜单

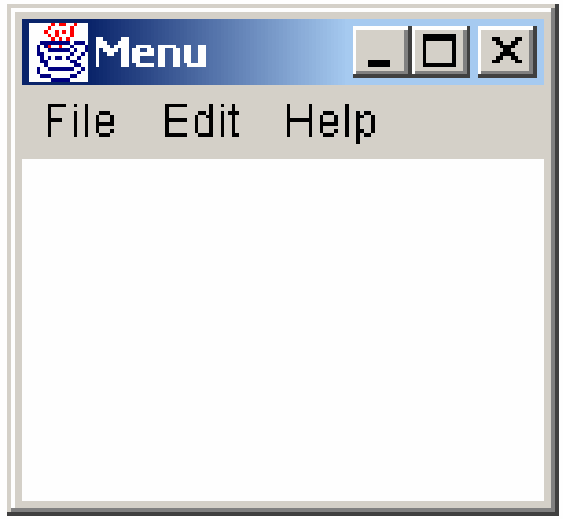
- 首先创建一个MenuBar对象，并将其置于一个可容纳菜单的容器(如Frame对象)中。
- 创建一个或多个Menu对象，并将它们添加到先前创建的MenuBar对象中。
- 创建一个或多个MenuItem对象，再将其加入到各Menu对象中

创建 MenuBar

```
import java.awt.*;
public class TestMenuBar{
        public static void main(String[] args) {
        Frame f = new Frame("MenuBar");
        MenuBar mb = new MenuBar();
        f.setMenuBar(mb);
        f.setSize(200,150);
        f.setVisible(true);
        }
}
```

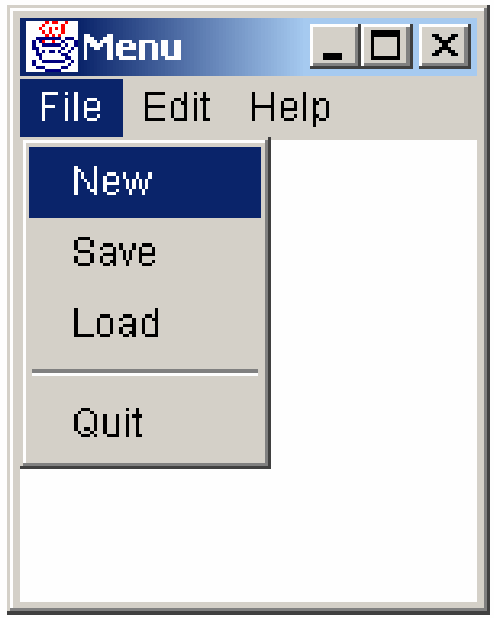
创建 Menu

```
import java.awt.*;
public class TestMenu{
   public static void main(String[] args) {
      Frame f = new Frame("Menu");
      MenuBar mb = new MenuBar();
      f.setMenuBar(mb);
      Menu m1 = new Menu("File");
      Menu m2 = new Menu("Edit");
      Menu m3 = new Menu("Help");
      mb.add(m1);
      mb.add(m2);
      mb.setHelpMenu(m3);
      f.setSize(150,120);
      f.setVisible(true);
    }
}
```



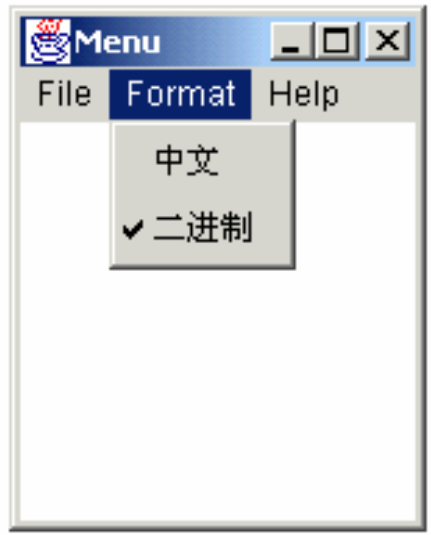
创建 MenuItem

```
import java.awt.*;
    public class TestMenuItem{
      public static void main(String[] args) {
          Frame f = new Frame("Menu");
          MenuBar mb = new MenuBar();
          f.setMenuBar(mb);
          Menu m1 = new Menu("File");
          Menu m2 = new Menu("Edit");
          Menu m3 = new Menu("Help");
          mb.add(m1);              mb.add(m2);
          mb.setHelpMenu(m3);
          MenuItem m11 = new MenuItem("New");
          MenuItem m12 = new MenuItem("Save");
          MenuItem m13 = new MenuItem("Load");
          MenuItem m14 = new MenuItem("Quit");
          m1.add(m11);     m1.add(m12);
          m1.add(m13);     m1.addSeparator();
          m1.add(m14);
          f.setSize(150,170);
          f.setVisible(true);
       }
    }
```



创建 CheckBoxMenuItem

```
import java.awt.*;
public class TestCheckBoxMenuItem{
   public static void main(String[] args) {
        Frame f = new Frame("Menu");
        MenuBar mb = new MenuBar();
        f.setMenuBar(mb);
        Menu m1 = new Menu("File");
        Menu m2 = new Menu("Format");
        Menu m3 = new Menu("Help");
        mb.add(m1);
        mb.add(m2);
        mb.setHelpMenu(m3);
        MenuItem m11 = new MenuItem("中文");
        m2.add(m11);
        CheckboxMenuItem m12 = new
   CheckboxMenuItem("二进制");
        m2.add(m12);
        f.setSize(150,170);
        f.setVisible(true);
   }
}
```



控制显示效果

Color类

Color类将颜色按照sRGB标准格式进行封装，该格式中红、绿、蓝三原色的取值范围都是0~255。

Color类定义了多个构造方法，常用的有：

```
public Color(int r, int g, int b)
public Color(int r, int g, int b, int a)     // a -- 透明度参数
Color  c = new Color(200, 170, 90);
Color  d = new Color(200, 170, 90, 120);
```

在GUI设计中使用Color类

```
Button b = new Button(“Test”);
Color  c = new Color(200, 170, 90);
b.setBackground(c)
```

Swing 工具集简介

- Swing是第二代GUI开发工具集
- AWT采用了与特定平台相关的实现，而绝大多数Swing组件却不是
- Swing是构筑在AWT上层的一组GUI组件的集合，为保证可移植性，它完全用Java语言编写
- 和AWT相比，Swing提供了更完整的组件，引入了许多新的特性和能力

本章结束