第 7 章　Java 中的 GUI 编程

图形用户界面（Graphical User Interface，GUI）使用图形的方式，借助菜单、按钮等标准界面元素和鼠标操作，帮助用户方便地向计算机系统发送指令，启动操作，并将系统运行的结果同样以图形方式显示给用户。图形用户界面操作简单，省去了字符界面用户必须记忆各种命令的麻烦，深受广大用户的喜爱和欢迎。所以学习设计和开发图形用户界面，是应用软件开发人员的必修课。

本章主要介绍 Swing 和 AWT 中基本组件的用法，如何创建简单的用户界面，包括布局管理、用户事件，以及如何创建复杂的用户界面等。

- 图形用户界面概述
- 创建简单的用户界面
- 布局管理
- 用户界面
- 创建复杂的用户界面

7.1　图形用户界面概述

GUI 由若干 GUI 组件（Component）组成，GUI 组件是可见的对象，用户可以通过鼠标或键盘对它进行操作，通过对不同事件的响应，来完成组件与用户之间或组件与组件之间的交互。

用户界面设计在程序设计中有较大的比重，在较为复杂的 Java 应用程序中，用户界面设计是不可缺少的。Java-AWT——抽象窗口工具箱就是专门是了解和掌握 UI 的各主要构件以及布局管理器。

Java 基类（Java Foundation Classes，JFC）是关于 GUI 组件和服务的完整集合，它大大简化了健壮的 Java 应用程序的开发和布署。JFC 作为 Java2SDK 的一个组成部分，主要由 5 个 API 构成：AWT、Swing、Java 2D、Drag and Drop、Accessibility，如图 7.1 所示。

AWT 组件库为各类 Java 应用程序提供了多种 GUI 工具。Swing 提供了一整套用 Java 语言编写的 GUI 组件，以保证可移植性。Java 2D 是一种图形 API，它为 Java 应用程序提供了高级的二维（2D）图形图像处理类的集合。同时，该 API 还包含了一套扩展字体集合。Drag and Drop 技术提供了 Java 和本地应用程序之间的互用性，用来在 Java 应用程序和不支持 Java 技术的应用程序之间交换数据。Accessibility API 提供了一套高级工具，可以辅助开发使用非传

统输入和输出方式的应用程序，它提供了一个辅助技术接口，如：屏幕阅读器、屏幕放大器、听觉文本阅读器（语音处理）等。

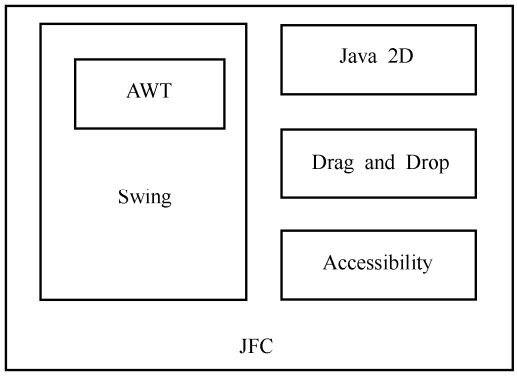


图 7.1　JFC 的组成

Java 早期进行图形用户界面（Graphics User Interface，GUI）设计时，使用 Java.awt 包中提供的类，比如 Button（按钮）、TextField（文本框）等组件类，“AWT”就是 Abstract Windowing Toolkit（抽象窗口工具包）的缩写。Java 2（JDK1.2）增加了一个新的 javax.swing 包，该包提供了功能更为强大的用来设计 GUI 界面的类。

Java 早期的 java.awt 包中的类创建的组件习惯上称作重量组件，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（native）在为它工作（称为它的同位体）。所谓本地组件是指非 Java 语言编写的组件。

AWT 组件的设计原理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本地组件。因此把有同位体的组件称为重量组件，基于重量组件的 GUI 设计有很多不足之处。

比如程序的外观在不同的平台上可能有所不同，而且重量组件的类型也不能满足 GUI 设计的需要，例如，不能把一幅图像添加到 AWT 按钮上或 AWT 标签上，因为 AWT 按钮或标签外观绘制是由本地的同位体来完成的，而同位体可能是用 C++编写的，它的行为是不能被 Java 扩展的。另外，使用 AWT 进行 GUI 设计可能会消耗大量的系统资源。

javax.swing 包提供了更加丰富的、功能强大的组件，称为 Swing 组件，其中大部分组件是轻量组件，没有同位体。Swing 组件的轻组件在设计上和 AWT 完全不同，轻组件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，而且较重量组件而言有更高的性能。

Swing 采用了一种 MVC 的设计方式，即模型－视图－控制（Model-View-Controller），其中模型用来保存内容，视图用来显示内容，控制器用来控制用户输入。

MVC 是现有的编程语言中制作图形用户界面的一种通用的思想，其思路是把数据的内容本身和显示方式分离开，这样就使得数据的显示更加灵活多样。比如，某年级各个班级的学生人数是数据，则显示方式是多种多样的，可以采用柱状图显示，也可以采用饼图显示和采用直接的数据输出。因此在设计的时候，就考虑把数据和显示方式分开，对于实现多种多样的显示是非常有帮助的。

Swing 胜过 AWT 的主要优势在于 MVC 体系结构的普遍使用。在一个 MVC 用户界面中，存在三个通讯对象：模型、视图和控件。模型是指定的逻辑表示法，视图是模型的可视化表示法，而控件则指定了如何处理用户输入。当模型发生改变时，它会通知所有依赖它的视图，视图使用控件指定其相应机制。

javax.swing 包中 JComponent（轻组件）类是 java.awt 包中 Container 类的一个直接子类、Component 类的一个间接子类。javax.swing 包中的 JFame 类和 JDialog 类分别是 java.awt 包中 Frame 类和 Dialog 类的直接子类、Window 类的间接子类（如图 7.2 所示）。

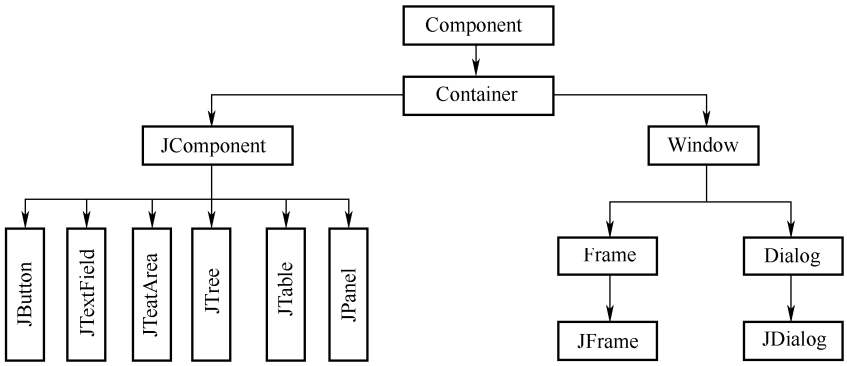


图 7.2　JComponent 类的部分子类以及 JFrame 类和 JDialog 类

在学习 GUI 编程时，必须很好地理解掌握两个概念：容器类（Container）和组件类（Component）。

Java 把由 Component 类的子类或间接子类创建的对象称为一个组件。

Java 把由 Container 的子类或间接子类创建的对象称为一个容器。

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。容器调用 removeAll()方法可以移掉容器中的全部组件；调用 remove（Component c）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：JApplet、JFrame、JDialog 和 JComponent。

JComponent 类的子类都是轻组件，JComponent 类是 java.awt 包中 Container 类的子类，因此所有的轻组件也都是轻容器。JFrame、JApplet、JDialog 都是重组件，即有同位体的组件。窗口（JFrame）、对话框（JDialog）、小应用程序（JApplet）可以和操作系统交互信息，轻组件必须在这些重量容器中绘制自己，习惯上称这些容器为 swing 的底层容器。

下面简单介绍一下 AWT 和 Swing。

1．AWT

AWT（Abstract Windowing Toolkit）是一个独立平台的窗口工具组件集，它依赖于对等组件（Peer），而对等组件是一个本地 GUI 组件，由 AWT 类管理。AWT 的作用是给用户提供基本的界面组件，如按钮、列表框、菜单等。

AWT 组件中，包含与其对等组件的大量实用操作。例如使用 AWT 创建了一个 Menu 类的实例，那么 Java 运行时系统将创建一个菜单对等组件的实例，而由创建的对等组件实际执行菜单的显示与管理。在创建菜单实例时，Solaris JDK 将产生一个 Motif 菜单对等组件，而 Windows JDK 将产生一个 Window 菜单对等组件等。这样，对等组件保护着每个平台的本地外观和感觉（Look and Feel），但是可以在移植时改变它们的大小和位置，因为它们在不同的平台可能会有不同的大小，也许还有不同的行为。而 AWT 类仅是对等组件外围的包装与操作工具，因此，对等组件可以快速产生一个 GUI 工具组件。

AWT 最初只包括与本地对等组件相关联的组件，称为重量组件（Heavyweight Component），这些组件在自己的本地不透明窗口中绘制，在改变其默认行为时，不可以为其扩展子类，此外，它们必须是矩形的，且不能有透明背景。因此，在 AWT 1.1 版本中引入了轻量组件（Lightweight Component）的概念。

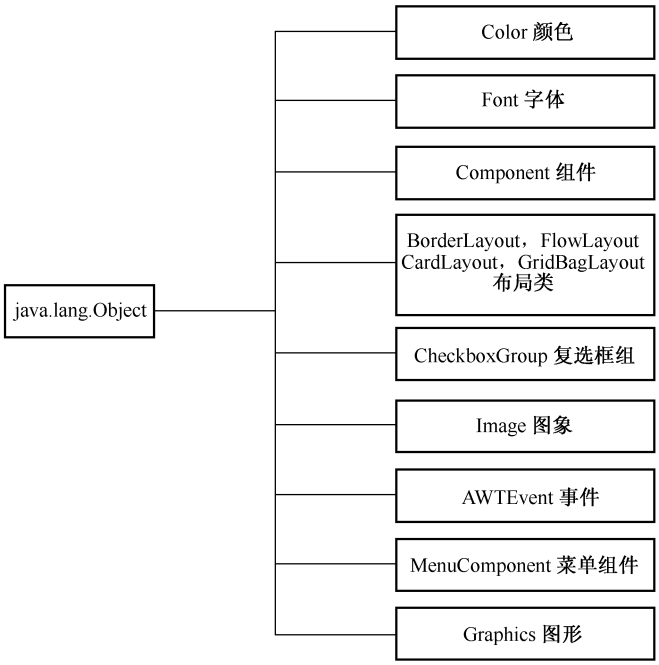
轻量组件没有本地对等组件，它不在本地不透明窗口中绘制，而是在它们的重量容器窗口中绘制，它们直接扩展了 java.awt.Component 类或 java.awt.Container 类。轻量组件不会损失与它们关联的不透明窗口的性能，它们可以有透明的背景及非矩形的外观，但在轻量组件的容器中必须有一个是重量组件，否则无法在窗口内绘制轻量组件。

我们将在本章对 AWT 软件包中的常用组件进行详细讲解。

常用组件（AWT）用于创建 GUI 组件的类包含在 java.awt 软件包中，可以通过下面的语句来引入 java.awt 软件包中的类：

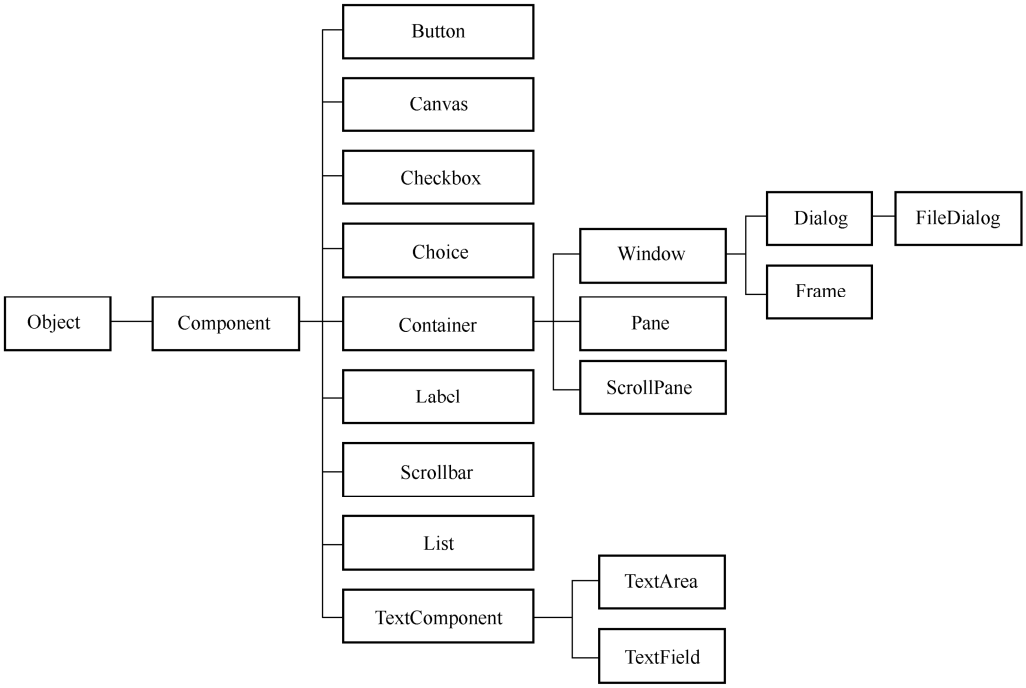
```
import java.awt.*;
```

图 7.3（a）列出了 java.awt 中的各个类，图 7.3（b）列出 AWT 常用图形组件类的继承关系。



（a）java.awt 中的各个类

图 7.3



（b）AWT 常用图形组件类的继承关系

图 7.3（续图）

2. Swing

Swing 是建立在 AWT 基础之上的，用来代替 AWT 中的重量组件，而不是用来替代 AWT 本身。它利用了 AWT 的底层组件，包括图形、颜色、字体、工具包和布局管理器等。它使用 AWT 最好的部分来建立一个新的轻量组件集，而丢弃了 AWT 中有问题的重量组件部分。

Swing 支持可插接观感（Pluggable Look-and-Feel），可插接的观感可使开发人员构建这样的应用程序：这些应用程序可在任何平台上执行，就好像是专门为那个特定平台而开发的一样。在 Microsoft Windows 环境中执行的程序似乎是专为这个环境而开发的；而同样的程序在 UNIX 平台上执行，其行为又似乎是专为 UNIX 环境开发的。

通过引入新特性和丰富的功能，Swing 提供了比 AWT 更全面的组件集合。Swing API 是围绕实现 AWT 各个部分的 API 构建的，这保证了所有早期的 AWT 组件仍然可以使用。

7.2　创建简单用户界面

7.2.1　框架与窗口

框架（Frame）是带标题的顶层层窗口。从类的层次上来看，它属于 Container 类。所以，在每个 Frame 中都可以设置版面，缺省设置是 BorderLayout。Frame 类的构造方法有两种，如表 7-1 所示。

表 7-1 java.awt.Frame 的构造函数与方法（Method）

构造函数	主要功能
Frame()	创建框架
Frame(String title)	创建框架，并以 title 为默认的标题
Method	主要功能
Image getIconImage()	返回窗口最小化时的图标
void setIconImage(Image img)	设置窗口最小化时的图标为 img
int getState()	返回窗口的状态，Frame.Normal 代表一般状态，Frame.ICONIFIED 代表窗口为最小化
MenuBar getMenuBar()	返回窗口使用的菜单对象
void setMwnuBar(MenuBar mb)	设置窗口使用的菜单对象为 mb
void remove(MenuCompoent mb)	删除窗口的菜单对象 mb
String getTitle()	取得窗口的标题
String setTitle(String title)	设置窗口的标题为 title
boolean isResizable()	返回窗口是否可改变大小
void setResizable()	设置窗口是否允许改变大小

需要说明的是，用这两种方法创建的窗口都是非可视窗口，只有作用 Frame 类的父类 Windows 类中的 show 方法后，才能在屏幕上显示出来。设置窗口的大小可以使用 resize 方法。

javax.swing 包中的 JFrame 类是 java.awt 包中 Frame 类的子类，因此 JFrame 类及其子类创建的对象是窗体，JFrame 类或子类创建的对象是重量容器。

（1）尽管 JFrame 窗口也是一个容器，不可以把组件直接添加到 JFame 窗体中。

（2）JFame 窗体含有一个称为内容面板的容器，应当把组件添加到内容面板中（内容面板也是重量容器）。

（3）不能为 JFame 窗体设置布局，而应当为 JFame 窗体的内容面板设置布局。内容面板的默认布局是 BorderLayout 布局。

JFame 窗体通过调用方法 getContentPane()得到其内容面板。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。窗口默认地被系统添加到显示器屏幕上。

窗体有一个基本的结构：窗体的上面是一个很窄的矩形区域，称为菜单条区域，用来放置菜单条。菜单条区域下面的区域用来放置窗体的内容面板（如图 7.4 所示）。

窗体中如果没有添加菜单条，菜单条区域将被内容面板挤占。

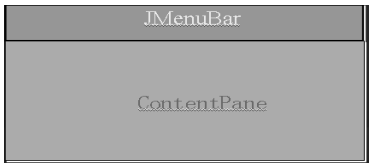


图 7.4 JFame 窗体的基本结构

JFrame 类常用方法：

JFrame()：该构造方法可以创建一个无标题的窗口。

JFrame（String s）：该构造方法可以创建一个标题为 s 的窗口。

public void setBounds（int a,int b,int width,int height）：窗口调用该方法可以设置出现在屏幕上时的初始位置是（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，窗口在屏幕出现的默认位置是（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。

public void setResizable（boolean b）：设置窗口是否可调整大小，窗口默认是可调整大小的。

public void setLocation（int x,int y）：设置窗口出现的坐标，x 为横坐标，y 为纵坐标。

注意：setSize()与 setLocation()联合起来相当于 setBounds()。

public void setDefaultCloseOperation（int operation）：该方法用来设置单击窗体右上角的关闭图标后，程序会做出怎样的处理。

其中的参数 operation 取下列有效值：

- DO_NOTHING_ON_CLOSE：什么也不做。
- HIDE_ON_CLOSE：隐藏当前窗口。
- DISPOSE_ON_CLOSE：隐藏当前窗口，并释放窗体占有的其他资源。
- EXIT_ON_CLOSE：结束窗体所在的应用程序。

窗口构造方法模式如下：

```
Window(String s)
 {
   super(s);
   setSize(260,270);
   setLocation(120,120);
   setVisible(true);
   setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

   JMenuBar menubar=new JMenuBar();
   menubar.add(菜单 1);
   menubar.add(菜单 2);
   ……
   menubar.add(菜单 n);
   setJMenuBar(menubar);

   Container con=getContentPane();
   con.add(组件 1);
   con.add(组件 2);
   ……
   con.add(组件 n);
   con.setLayout(布局策略);
   con.validate();
```

```
    validate();

    组件1.addActionListener(this);
    组件2.addActionListener(this);
    ……
    组件n.addActionListener(this);

    菜单项1.addActionListener(this);
    菜单项2.addActionListener(this);
    ……
    菜单项m.addActionListener(this);
  }
```

7.2.2 按钮（Button）

当用户用鼠标单击按钮时，会激活一个事件。按钮由 Button 类创建，而 Button 类是由 Component 类直接扩展的。表 7-2 列出了 Button 类的构造函数及常用方法。

表 7-2 Button 类的构造函数及常用方法

构造函数	说明
public Button()	创建一个无标签按钮
public Button(String label)	创建一个有标签按钮
常用方法	说明
public String getLabel()	获取按钮标签
public void setLabel(String label)	设置按钮标签

按钮表面上显示的文字称为按钮标签。一个 GUI 可以包含多个按钮，但每个按钮的标签必须是唯一的，多个按钮具有相同的标签是一个逻辑错误，这样会使用户产生混淆。

例 7-1 的程序创建 3 个按钮，并将它们添加到 applet 中，其中两个按钮带有按钮标签。下面的语句用于创建 3 个 Button 引用：

```
private Button button1,button2,button3;
```

每个 Button 引用都通过 new 和一个 Button 构造函数来实例化。下面语句分别对 button1、button 2 和 button 3 进行实例化：

```
button1 = new Button("Click here");
button2 = new Button("Sorry, I do nothing");
button3 = new Button();
```

构造函数 Button 中的字符串参数就是按钮标签，其中对象 Button 3 没有标签。

要想使各对象可见，必须使用 add 方法将它们添加到一个容器中。前面已经介绍过，容器是用于放置多个组件的一个区域，本章主要使用的容器是 applet 和 Frame。使用 Container 类的 add 方法可以将组件安置到容器上。下面的语句实现将创建的三个按钮组件安置到 applet 中：

```
add(button1);
add(button2);
```

```
add(button3);
```

【例 7-1】编写一个程序，功能是创建三个按钮并把它们加入到 applet 中。

```
import java.applet.Applet;
import java.awt.*;

public class MyButtons extends Applet{
private Button button1,button2,button3;
public void init()
{
button1=new Button("Click here");
button2=new Button("Sorry, I do nothing");
button3=new Button();
//添加按钮
add(button1);
add(button2);
add(button3);
}
}
```

先将上述程序保存为 MyButtons.java 文件，再编译生成 MyButtons.class 文件，然后建立一个如下内容的网页文件 test.html，将该 html 文件与 MyButtons.class 文件放在相同的目录下：

```
<html>
<head><title>图形界面测试</title></head>
<body>
<applet code="MyButtons.class" width=300 height=200>
</applet>
</body>
</html>
```

用浏览器打开该 html 文件，或者使用 Java 的小程序查看器 appletviewer，即键入如下命令：

```
appletviewer test.html
```

程序运行的结果如图 7.5 所示。

图 7.5　创建 3 个按钮

7.2.3　标签与文本框

1. 标签（label）

JLabel 类负责创建标签对象，标签用来显示信息，但没有编辑功能。JLabel 类的构造方法：

public Jlabel()：创建没有名字的标签。

public Jlabel(String s)：创建名字是 s 的标签，s 在标签中靠左对齐。

public Jlabel(String s, int alignment)：参数 alignment 决定标签中的文字在标签中的水平对齐方式。

public Jlabel(Icon icon)：创建具有图标 icon 的标签，icon 在标签中靠左对齐。

public Jlabel(String s,Icon icon,int alignment)：创建名字是 s，具有图标 icon 的标签，参数 aligment 决定标签中的文字和图标做为一个整体在标签中的水平对齐方式。

JLabel 类的常用方法：

String getText()：获取标签的名字。

void setText(String s)：设置标签的名字是 s。

Icon getIcon()：获取标签的图标。

void setIcon(Icon icon)：设置标签的图标是 icon。

void setHorizontalTextPosition(int a) 参数 a 确定名字相对于图标的位置，a 的取值是 JLabel.LEFT，JLabel.RIGHT。

void setVerticalTextPosition(int a)：参数 a 确定名字相对于图标的位置，参数 a 取值是 JLabel.BOTTOM，JLabel.TOP。

【例 7-2】编写一个程序，功能是创建两个标签并把它们加入到 applet 中。

```
import java.applet.Applet;
import java.awt.*;
public class MyLabel extends Applet{
private Label label1,label2;
public void init()
{
//create a Label without text
label1=new Label();
//create a Label with a string argument
label2=new Label("Label with text");
//add labels to applet
add(label1);
add(label2);
}
}
```

程序的运行结果如图 7.6 所示。

图 7.6　创建标签示例

Label 对象通过 Label 类及一个 Label 构造函数创建。下面的语句是使用 Label 类创建两个标签引用：

```
private Label label1,label2;
```

对象 label1 使用下面的语句进行创建：

```
label1=new Label();
```

这个标签上未显示任何文本。一般情况下，对于在初始时刻不必显示任何文本的标签通常使用这种类型的标签对象，不过，在后面程序中，仍然可以在该标签上显示信息。标签对象 label2 通过下面语句创建：

```
label2=new Label("Label with text");
```

在这个标签上显示文本字符串“Label with text”。

2. 文本框

单行文本框（TextField）和多行文本框（即文本区，TextArea）是由 TextComponent 类直接继承的，我们先讨论单行文本框。

（1）单行文本框。

单行文本框是一个单行显示区域，可以从键盘上接收用户输入。用户将数据输入文本框，然后按回车键就可以在程序中使用该数据。单行文本框也可以用于显示信息。表 7-3 列出了 TextField 类的构造函数。

表 7-3　TextField 类的构造函数及常用方法

构造函数	说明
public TextField()	创建一个 TextField 对象，默认列宽为 2
public TextField(int cols)	创建一个指定列宽的 TextField 对象
public TextField(String text)	创建一个 TextField 对象，显示字符串 text
public TextField(String text,int cols)	创建一个指定列宽的 TextField 对象，并显示字符串 text
public void setEchoChar(char c)	设置用户输入字符时的回显字符
public void setText(String text)	设置单行文本框的文本内容
public String getText()	获取单行文本框的文本内容

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

【例 7-3】演示单行文本框的构造函数及常用方法的使用。

```
import java.applet.Applet;
import java.awt.*;
public class TextFieldDemo extends Applet{
private TextField textField1,textField2;
public void init(){
//construct TextField with default text
textField1=new TextField("在这里输入用户名");
//construct TextField with 15 elements visible
textField2=new TextField(15);
```

```
//set the echo charactor
textField2.setEchoChar('*');
//add components to applet
add(new Label("用户名: "));
add(textField1);
add(new Label("密码: "));
add(textField2);
}
public void paint(Graphics g){
String s;
s="用户名: "+textField1.getText();
s=s+"密码: "+textField2.getText();
showStatus(s);
}
}
```

程序运行结果如图 7.7 所示。

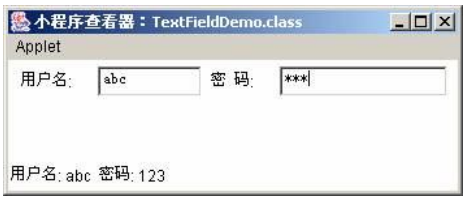


图 7.7　单行文本框示例

通过下列语句：

```
textField1=new TextField("在这里输入用户名");
textField2=new TextField(15);
```

创建了两个 TextField 的对象 textField1 和 textField2，其中 textField1 的列宽为所显示字符串的长度，textField2 的列宽为 15，未显示任何文本。通过下列语句：

```
textField2.setEchoChar('*');
```

设置了 textField2 的掩码字符，每当在该文本框中输入一个字符时，就会显示一个星号(*)。注意，虽然为 textField2 设置了掩码字符，但该文本框的 getText 方法仍然返回实际输入的字符串。

（2）多行文本框即文本区（TextArea）。多行文本框（TextArea）可以显示多行文本，用户在多行文本框中按回车键会使光标移到下一行行首。表 7-4 列出了 TextArea 类的构造函数及常用方法。

表 7-4　TextArea 类的构造函数及常用方法

构造函数	说明
public TextArea()	创建一个 TextArea 对象
public TextArea(int rows,int cols)	创建一个指定行列的 TextArea 对象
public TextArea(String text)	创建一个含有指定文本的 TextArea 对象
public TextArea(String text, int rows,int cols)	创建一个指定行列并含有指定文本的 TextArea 对象

续表

构造函数	说明
Public TextArea(String text,int rows, int cols, int scrollbars)	创建一个指定行列并含有指定文本及指定滚动条类型的 TextArea 对象，scrollbars 可以取以下 4 个值之一： SCROLLBARS_BOTH(默认) SCROLLBARS_VERTICAL_ONLY SCROLLBARS_HORIZONTAL_ONLY SCROLLBARS_NONE
public void append(String s)	在多行文本框尾部添加文本
public void insert(String s,int pos)	在多行文本框指定位置插入文本
public void setText(String text)	设置多行文本框的文本内容
public String getText()	获取多行文本框的文本内容
public String getSelectedText()	获取多行文本框中选中的文本内容
Public void setEditable(boolean b)	设置多行文本框的可编辑状态

【例 7-4】演示 TextArea 类的常用构造方法。其中一个多行文本框不允许用户进行编辑，而另一个则允许用户输入信息，如图 7.8 所示。

```
import java.applet.Applet;
import java.awt.*;
public class TextAreaDemo extends Applet{
private TextArea textArea1,textArea2;
public void init()
{
//creating 2 10*20 TextAreas
textArea1=new TextArea("Read-only Text!",10,20);
textArea2=new TextArea(10,20);
//set textArea1 read-only
textArea1.setEditable(false);
add(textArea1);
add(textArea2);
}
}
```

程序中使用 TextArea 类创建多行文本框 textArea1 和 textArea2。下列语句：

```
textArea1=new TextArea(""Read-only Text!"",10,20);
textArea1=new TextArea(10,20);
```

用于创建两个多行文本框，它们都包括 10 行、20 列，其中，textArea1 初始时显示文本“Read-only Text!”，textArea2 初始时未显示任何文本。下列语句：

```
textArea1.setEditable(false);
```

使多行文本框 textArea1 不可编辑（你仍然可以把光标置于其中，但不能修改它的文本）。

多行文本框总是带有滚动条，仅当多行文本框中的文本超出其可见范围时，滚动条才能滚动。程序员无法直接控制这两个滚动条。

图 7.8　创建多行文本框示例

7.2.4　复选按钮

JCheckBox 复选框

复选框提供两种状态，一种是选中，另一种是未选中，用户通过单击该组件切换状态。

JCheckBox 类常用方法：

public JCheckBox()：创建一个没有名字的复选框。

public JCheckBox(String text)：创建一个名字是 text 的复选框。

public JCheckBox(Icon icon)：创建一个带有默认图标 icon，但没有名字的复选框。

public JCheckBox(String text, Icon icon)：创建一个带有默认图标 icon 和名字 text 的复选框。

public void setIcon(Icon defaultIcon)：设置复选框上的默认图标。

public void setSelectedIcon(Icon selectedIcon)：设置复选框选中状态下的图标。

public boolean isSelected()：如果复选框处于选中状态该方法返回 true，否则返回 false。

如果不对复选框进行初始化设置，默认的初始化设置均为未选中。

例 7-5 的程序是使用复选框的例子，当该程序开始运行时，所有复选框都为非选中状态，如图 7.9 所示。

【例 7-5】编写一个程序，功能是创建 6 个复选框并把它们加入到 applet 中。

```
import java.applet.Applet;
import java.awt.*;
public class MyCheckbox extends Applet{
private String city[]={"桂林","福州","青岛","济南","深圳","大连"};
private Checkbox c[] = new Checkbox[6];
public void init()
{
add(new Label("请选择是省会的城市: "));
//Create six Checkboxes and add them to applet
for(int i=0;i<6;i++){
c[i]=new Checkbox(city[i]);
add(c[i]);
}
add(new Label("这是一个复选框的例子"));
}
}
```

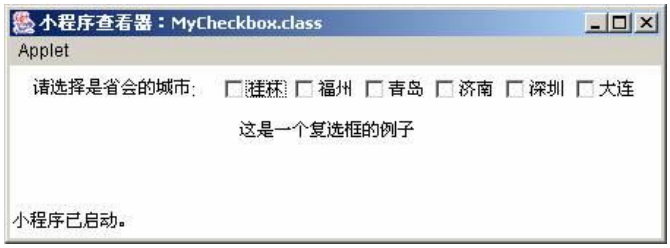


图 7.9　创建复选框按钮

通过下列语句：

```
private Checkbox c[]=new Checkbox[6];
```

声明了 6 个 Checkbox 引用。通过下列语句：

```
for(int i=0;i<6;i++){
c[i]=new Checkbox(city[i]);
add(c[i]);
}
```

创建 6 个 Checkbox 对象，并把它们添加到 applet 中。

7.2.5　单选按钮

JRadioButton 单选按钮

单选按钮和复选框很类似，所不同的是,在若干个复选框中可以同时选中多个，而一组单选按钮同一时刻只能有一个被选中。当创建了若干个单选按钮后，应使用 ButtonGroup 再创建一个按钮组对象，然后利用这个对象把这若干个单选按钮归组。归到同一组的单选按钮每一时刻只能选一个。例如：

```
ButtonGroup group=new ButtonGroup();
JRadioButton button1=new JRadioButton("小学"),
             button2=new JRadioButton("中学");
group.add(button1);
group.add(button2);
```

可以将复选框组合到一起，成为一组单选按钮，在这组按钮中，每次只能选中其中一个（值为 true），其他复选框都处于未选中状态（即值为 false）。单选按钮由 CheckboxGroup 类和 Checkbox 类共同创建。CheckboxGroup 类是从 Object 类直接继承的，因此，CheckboxGroup 对象不能添加到容器中。表 7-5 列出了用于创建单选按钮的构造函数，其常用方法与复选框相同。

表 7-5　CheckboxGroup 类和 Checkbox 类的构造函数

构造函数	说明
public CheckboxGroup()	创建一个 CheckboxGroup 对象
public Checkbox(String label,CheckboxGroup c,Boolean state)	创建一个单选按钮，其标签为 label，状态为 state，该单选按钮将添加到 CheckboxGroup c 中

【例 7-6】演示单选按钮的创建方法，运行结果如图 7.10 所示。

```
import java.applet.Applet;
```

```
import java.awt.*;
public class RadioButton extends Applet{
private String city[]={"北京","上海","西安","重庆","深圳","大连"};
private Checkbox radio[] = new Checkbox[6];
private CheckboxGroup c=new CheckboxGroup();
public void init()
{
add(new Label("请选择中国最大的城市: "));
//Create six radio buttons and add them to applet
for(int i=0;i<6;i++)
{
radio[i]=new Checkbox(city[i],c,false);
add(radio[i]);
}
add(new Label("这是一个单选按钮的例子"));
}
}
```

图 7.10　创建一组单选按钮

下列语句创建一个 CheckboxGroup 引用：

```
private CheckboxGroup c=new CheckboxGroup();
```

下列语句用于对单选按钮进行实例化：

```
radio[i]=new Checkbox(city[i],c,false);
```

构造函数的第一个参数是单选按钮的标签，第二个参数是拥有该单选按钮的 CheckboxGroup，最后一个参数表示单选按钮的初始状态——在一组单选按钮中，只能有一个设置为 true；但是，允许将所有单选按钮的初始值都设为 false。如果在同一个 CheckboxGroup 中将多个单选按钮设为 true，那么只有最后一个设置为 true 的单选按钮是有效的（处于被选中状态）。

7.2.6　下拉列表

列表框（List）用于显示一系列的选项，用户可以从中选择一个或多个选项。

下拉列表框（Choice）与列表框相似，不同的是下拉列表框只能从列表中选择一个选项，当用户单击旁边下拉箭头按钮时，选项列表打开。表 7-6 列出了 List 类及 Choice 类的构造函数及常用方法。

表 7-6　List 类和 Choice 类的构造函数及常用方法

构造函数	说明
public List()	创建一个 List 对象，该对象只允许选中一个选项
public List(int items)	创建一个 List 对象，选项可见行数由 items 指定
public List(int items,boolean ms)	创建一个 List 对象，选项可见行数由 items 指定，是否可以多选由 ms 指定
public Choice()	创建一个 Choice 对象
public void add(String item)	在列表框/下拉列表框最后添加一个选项
public String getItem(int index)	获取列表框/下拉列表框指定索引的选项，索引值从 0 开始
public int getItemCount()	获取列表框/下拉列表框的项目总数
public String[] getItems()	获取列表框的所有选项
public int getSelectedIndex()	获取列表框/下拉列表框中被选定选项的索引，索引值从 0 开始，列表框中若有多个选项被选定，则返回-1
public int[] getSelectedIndexes()	获取列表框中所有被选定选项的索引，索引值从 0 开始
public String getSelectedItem()	获取列表框/下拉列表框中被选定的选项，列表框中若有多个选项被选定，则返回 null
public String[] getSelectedItems()	获取列表框中所有被选定的选项
public void remove(int position)	删除列表框/下拉列表框指定位置的选项
public void remove(String item)	删除列表框/下拉列表框中第一个出现且与参数相同的选项
public void removeAll()	删除列表框/下拉列表框中所有选项
public void select(int index)	选定列表框/下拉列表框中指定索引的选项

【例 7-7】演示列表框和下拉列表框的使用。程序运行结果如图 7.11 所示。

```
import java.applet.Applet;
import java.awt.*;
public class ListAndChoiceDemo extends Applet{
private List cityList;
private Choice cityChoice;
public void init()
{
//create a list with 5 items visible
//allow multiple selections
cityList=new List(5,true);
//add 4 items to the list
cityList.add("桂林");
cityList.add("福州");
cityList.add("济南");
cityList.add("大连");
//create a choice
cityChoice=new Choice();
//add 4 items to the choice
```

```
cityChoice.addItem("北京");
cityChoice.addItem("上海");
cityChoice.addItem("西安");
cityChoice.addItem("重庆");
add(new Label("中国最大的城市: "));
add(cityChoice);
add(new Label("省会城市: "));
add(cityList);
}
}
```

图 7.11　List 类和 Choice 类示例

在实际操作中，若列表框的可见行数少于列表框总选项数，则列表框自动添加竖直滚动条，在允许多选的情况下，可以通过鼠标单击选择/取消选择一个选项。

可以通过列表框/下拉列表框的 add 方法或 addItem 方法向列表框中添加选项。

7.2.7　菜单组件

窗口上方为菜单条，菜单放在菜单条里，菜单项放在菜单里。

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：

```
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)：向菜单增加标题为 s 的菜单项。

public JMenuItem getItem(int n)：得到指定索引处的菜单项，第一菜单项的索引为 0。

public int getItemCount()：得到菜单项数目。

public void setText(String text)：设置菜单的名字为 text。

public String getText()：获取菜单的名子。

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 text, Icon icon)：构造有标题和图标的菜单项。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 mageIcon("dog.gif");
```

JMenuItem(String s)：构造有标题的菜单。

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。也可以使用另一个类方法 public static KeyStroke getKeyStroke(int keyCode, int modifiers)返回一个 KeyStroke 对象。

其中参数 keyCode 取值范围：

KeyEvent.VK_A～KeyEvent.VK_Z ；

modifiers 取值为：

InputEvent.ALT_MASK：Alt 键

InputEvent .CTRL_MASK：Ctrl 键

InputEvent .SHIFT_MASK：Shift 键

4. 嵌入子菜单

JMenu 是 JMenuItem 的子类，因此菜单项本身还可以是一个菜单，称这样的菜单项为子菜单。子菜单的创建很简单，只需创建一个包含若干菜单子项（JMenuItem）的菜单（Jmenu），将这个菜单如同加入菜单项一样将其加入到上一级菜单，作为上一级菜单的菜单项即可。

【例 7-8】创建一个简单的三级菜单。

```
import javax.swing.*;
import java.awt.event.InputEvent;
import java.awt.event.KeyEvent;
class FirstWindow extends JFrame
{
    JMenuBar menubar;
    JMenu menu,item3,itemson3;
    JMenuItem  item1,item2,itemson1,itemson2,red,yellow,blue;
    FirstWindow(String s)
    {
      super(s);
      setSize(160,170);
      setLocation(120,120);
      setVisible(true);
      menubar=new JMenuBar();
      menu=new JMenu("文件");
      item1=new JMenuItem("打开");
      item2=new JMenuItem("保存");
```

```
        item3=new JMenu("工具栏");
        itemson1=new JMenuItem("常用");
        itemson2=new JMenuItem("格式");

        itemson3=new JMenu("颜色");
        red=new JMenuItem("红色");
        yellow=new JMenuItem("黄色");
        blue=new JMenuItem("蓝色");

        itemson3.add(red);
        itemson3.add(yellow);
        itemson3.add(blue);

        item3.add(itemson1);
        item3.add(itemson2);
        item3.add(itemson3);

        menu.add(item1);
        menu.add(item2);
        menu.addSeparator();
        menu.add(item3);
        menubar.add(menu);
        setJMenuBar(menubar);
        validate();
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }
}
public class Example
{
    public static void main(String args[])
    {
      FirstWindow win=new FirstWindow("一个简单的窗口");
    }
}
```

7.3 布局管理

当把组件添加到容器中时，总希望控制组件在容器中的位置，这就需要进行布局设计。java.awt 包中的常用策略有 4 种类型：FlowLayout、BorderLayout、CardLayout、GridLayout 布局类和 java.swing.border 包中有 BoxLayout 布局类。

对于 JFrame 窗口，程序只能将组件添加到它的内容面板中。JFrame 的内容面板是一个 Container 类型的对象，即容器。JFame 窗体通过调用方法 getContentPane()创建该内容面板的对象，内容面板的默认布局是 BorderLayout。

容器可以使用方法 setLayout（布局对象）来设置自己的布局。

7.3.1　流式布局（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：

```
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，每一行中的组件都居中排列，组件之间默认的水平和垂直间隙是 5 个像素。

FlowLayout 布局对象调用　setAlignment(int aligin)方法可以重新设置布局的对齐方式，其中 aligin 可以取值 FlowLayout.LEFT、FlowLayout.CENTER、FlowLayout.RIGHT。

FlowLayout 布局对象调用 setHgap(int hgap)和 setVgap(int vgap)可以重新设置布局的水平间隙和垂直间隙。

【例 7-9】编写一个程序，功能是创建 10 个按钮并按照流布局的方式放置到内容面板中。运行效果如图 7.12 所示。

```
import java.awt.*;
import javax.swing.*;
class WindowFlow extends JFrame
{
   WindowFlow(String s)
   {
     super(s);
     Container  contenetPane=this.getContentPane();
     FlowLayout flow=new FlowLayout();
     flow.setAlignment(FlowLayout.LEFT);
     flow.setHgap(2);
     flow.setVgap(8);
     contenetPane.setLayout(flow);
     for(int i=1;i<=10;i++)
        {
          contenetPane.add(new JButton("i am "+i));
        }
     contenetPane.validate();
     setBounds(100,100,150,120);
     setVisible(true);
     setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
   }
}
class k
```

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

图 7.12　使用 FlowLayou 布局的窗口

7.3.2　边界布局（BorderLayout 布局）

BorderLayout 布局是 Window 型容器的默认布局，例如 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。

BorderLayout 也是一种简单的布局策略，如果一个容器使用这种布局，那么容器空间简单地划分为东、西、南、北、中五个区域，中间的区域最大。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 CENTER、NORTH、WEST、SOUTH、 EAST 表示，例如，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);
```

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掉。

注意，容器每增加或删除组件后都要调用 validate()方法以保证能正确地显示修改后的内容，其实质相当于“刷新”。

【例 7-10】编写一个程序，功能是创建 4 个按钮和一个文本域，按照边界布局的方式放置到内容面板中。运行效果如图 7.13 所示。

```
import javax.swing.*;
import java.awt.*;
class k
{
   public static void main(String args[])
   {
      JFrame win=new JFrame("窗体");
      win.setBounds(100,100,300,300);
      win.setVisible(true);
      JButton bSouth=new JButton("南"),
             bNorth=new JButton("北"),
```

```
            bEast =new JButton("东"),
            bWest =new JButton("西");
        JTextArea  bCenter=new JTextArea("中心");
        Container contenetPane=win.getContentPane();
        contenetPane.add(bNorth,BorderLayout.NORTH);
        contenetPane.add(bSouth,BorderLayout.SOUTH);
        contenetPane.add(bEast,BorderLayout.EAST);
        contenetPane.add(bWest,BorderLayout.WEST);
        contenetPane.add(bCenter,BorderLayout.CENTER);
        contenetPane.validate();
        win.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

图 7.13　使用 BorderLayout 布局的窗口

7.3.3　卡式布局（CardLayout 布局）

使用 CardLayout 的容器可以容纳多个组件，但实际上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 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 卡片式布局。

JTabbedPane 容器自带一些选项卡，这些选项卡和容器中添加的组件相对应，也就是说，单击相应的选项卡时，选项卡窗格将显示对应的组件。选项卡窗格自带的选项卡默认地在该选项卡窗格的顶部，从左向右依次排列，选项卡的顺序和容器添加的组件的顺序相同。

选项卡窗格可以使用 add(String text,Component c);方法将组件 c 添加到容器当中，并指定和该组件 c 对应的选项卡的文本提示是 text。

【例 7-11】编写一个程序，功能是：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的内容面板中。运行效果如图 7.14 所示。

```
import javax.swing.*;
import java.awt.*;
class MyWin extends JFrame
```

```
{
    JTabbedPane p;
    public MyWin()
    {
      setBounds(100,100,500,300);
      setVisible(true);
      p=new JTabbedPane(JTabbedPane.LEFT);
      for(int i=1;i<=5;i++)
      {
        p.add("观看第"+i+"个按钮",new JButton("按钮 "+i));
      }
      p.validate();
      Container  con=getContentPane();
      con.add(p,BorderLayout.CENTER);
      con.validate();
      setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }
}
class k
{
    public static void main(String args[])
    {
       new MyWin();
    }
}
```

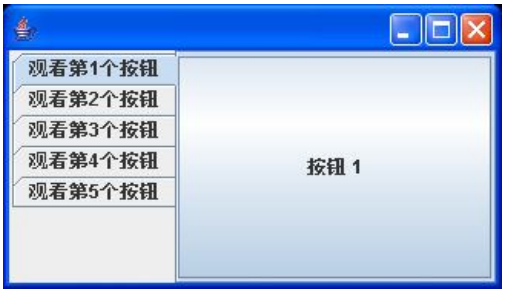


图 7.14　嵌套 JTabedPane 的窗体

7.3.4　网格布局（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。GridLayout 比较灵活，划分多少网格由程序自由控制，而且组件定位也比较精确。

使用 GridLayout 布局编辑器的一般步骤如下：

（1）使用构造方法 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。

（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。

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，这显然很不实用。为了克服此缺点，可以使用容器嵌套。

GridLayout 布局中的每个网格都可以添加容器，而这个容器又可以设置为其他布局策略，如 GridLayout、FlowLayout、CarderLayout 或 BorderLayout 布局等。利用这种嵌套方法，可以设计出符合一定需要的布局。

7.3.5　盒式布局（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，盒式容器的的默认布局是盒式布局，而且不允许更改布局策略。因此，在策划程序的布局时，可以利用容器的嵌套将某个容器嵌入几个盒式容器，达到合理安排盒式布局的目的。即无需显示使用盒式布局，只要使用几个盒式容器就可实现盒式布局。

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，是否指定了是行排列还是列排列。BoxLayout 的构造方法 BoxLayout(Container con,int axis)可以创建一个盒式布局对象，并指定容器 con 使用该布局对象，参数 axis 设置按行或按列排。其可取值 X_AXIS（按行排）或 Y_AXIS（按列排）。与 FlowLayout 布局不同的是，其只有一行或一列，不会出现第二行或第二列。

行型盒式布局容器中添加的组件的上沿在同一水平线上。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 类的类方法 createHorizontalBox()可以获得一个具有行型盒式布局的盒式容器；使用 Box 类的类方法 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 类调用类方法 createHorizontalStrut(int width)可以得到一个不可见的水平 Struct 类型对象，称做水平支撑。该水平支撑的高度为 0，宽度是 width。

Box 类调用类方法 createVertialStrut(int height)可以得到一个不可见的垂直 Struct 类型对象，称做垂直支撑。参数 height 决定垂直支撑的高度，垂直支撑的宽度为 0。

【例 7-12】编写一个程序，功能是：首先创建一个内容容器 con，设置其布局为流式布局；再在其中加入一个具有行盒式布局的盒式容器 baseBox；然后在 baseBox 中加入两个具有列型盒式布局的盒式容器 boxV1、boxV2，并在它们之间加入一个水平支撑对象；最后再分别在 boxV1 和 boxV2 中加入组件，并在组件之间加入垂直支撑对象。其运行效果如图 7.15 所示。

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

      Container con=getContentPane();
      con.setLayout(new FlowLayout());
      con.add(baseBox);
      con.validate();
      setBounds(120,125,200,200);
      setVisible(true);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
public class k
{
    public static void main(String args[])
    {
      new WindowBox();
    }
}
```

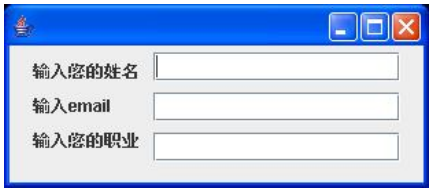


图 7.15　嵌套 Box 容器的窗体

7.3.6　空布局（null 布局）

可以把一个容器的布局设置为 null 布局（空布局）。容器采用空布局，借助方法 setBounds(int a, int b, int width, int height)可以准确地定位组件在容器的位置和大小。

例如，con 是某个容器：

```
con.setLayout(null);
```

把 con 的布局设置为空布局。

向空布局的容器 con 添加一个组件 c 需要两个步骤，首先使用 add（c）方法向容器添加组件，然后组件 c 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。由于组件都是一个矩形结构，所以方法中的参数 a、b 是被添加的组件 c 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；weidth、height 是组件 c 的宽和高。

7.4　用户事件

图形用户界面通过事件机制响应用户和程序的交互。产生事件的组件称为事件源。如当用户单击某个按钮时就会产生动作事件，该按钮就是事件源。要处理产生的事件，需要在特定的方法中编写处理事件的程序。这样，当产生某种事件时就会调用处理这种事件的方法，从而实现用户与程序的交互，这就是图形用户界面事件处理的基本原理。

7.4.1　事件基本概念

与 AWT 有关的所有事件类都由 java.awt.AWTEvent 类派生，它也是 EventObject 类的子类。AWT 事件共有 10 类，可以归为两大类：低级事件和高级事件。

java.util.EventObject 类是所有事件对象的基础父类，所有事件都是由它派生出来的。AWT 的相关事件继承于 java.awt.AWTEvent 类，这些 AWT 事件分为两大类：低级事件和高级事件。低级事件是指基于组件和容器的事件，当一个组件上发生事件，如鼠标的进入、点击、拖放等，或组件的窗口开关等，触发了组件事件。高级事件是基于语义的事件，它可以不和特定的动作相关联，而依赖于触发此事件的类，如在 TextField 中按 Enter 键会触发 ActionEvent 事件，滑动滚动条会触发 AdjustmentEvent 事件，或是选中项目列表的某一条就会触发 ItemEvent 事件。

1. 低级事件

ComponentEvent（组件事件：组件尺寸的变化，移动）

ContainerEvent（容器事件：组件增加，移动）

WindowEvent（窗口事件：关闭窗口，窗口闭合，图标化）

FocusEvent（焦点事件：焦点的获得和丢失）
KeyEvent（键盘事件：键按下、释放）
MouseEvent（鼠标事件：鼠标单击，移动）
2. 高级事件（语义事件）
ActionEvent（动作事件：按钮按下，TextField 中按 Enter 键）
AdjustmentEvent（调节事件：在滚动条上移动滑块以调节数值）
ItemEvent（项目事件：选择项目，不选择“项目改变”）
TextEvent（文本事件，文本对象改变）

7.4.2 焦点事件

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。当组件具有焦点监视器后，如果组件从无输入焦点变成有输入焦点或从有输入焦点变成无输入焦点都会触发 FocusEvent 事件。创建焦点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)。

一个组件调用以下方法可以获得输入焦点：

```
public boolean requestFocusInWindow()
```

调用以下方法可以将焦点移到下一个组件：

```
public void transferFocus()
```

【例 7-13】编写一个程序，功能是：监视器监视组件上的焦点事件，当组件获得焦点时组件的颜色变成蓝色，当失去焦点时，组件的颜色变成红色。运行效果如图 7.16 所示。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class MyWindow extends JFrame implements FocusListener
{
   JTextField text;
   JButton button;
   MyWindow(String s)
   {
     super(s);
     text=new JTextField(10);
     button=new JButton("按钮");
     text.requestFocusInWindow();
     Container con=getContentPane();
     con.setLayout(new FlowLayout());
     con.add(text);
```

```
        con.add(button);
        text.addFocusListener(this);
        button.addFocusListener(this);
        setBounds(100,100,150,150);
        setVisible(true);
        validate();
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      }
    public void focusGained(FocusEvent e)
      {
         JComponent com=(JComponent)e.getSource();
         com.setBackground(Color.blue);
      }
    public void focusLost(FocusEvent e)
      {
         JComponent com=(JComponent)e.getSource();
         com.setBackground(Color.red);
      }
}
class k
{
    public static void main(String args[])
    {
       MyWindow win=new MyWindow("窗口");
    }
}
```

图 7.16　焦点事件的窗体

7.4.3　键盘事件

在 Java1.2 事件模式中，必须要有发生事件的事件源。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，如果用户敲击键盘上一个键就导致这个组件触发 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。该接口包含三个方法：

public void KeyPressed(KeyEvent e)

public void KeyReleased(KeyEvent e)

public void KeyTyped(KeyEvent e)

当按下键盘上某个键时，监视器就会发现，然后方法 KeyPressed 会自动执行，并且 KeyEvent 类自动创建一个键盘事件对象传递给方法 KeyPressed 中的参数 e。方法 KeyTyped 是 Pressedkey 和 KeyReleased 方法的组合，当键被按下又释放时，KeyTyped 方法被调用。

用 KeyEvent 类的 public int getKeyCode()方法，可以判断哪个键被按下、敲击或释放，getKeyCode 方法返回一个键码值。

【例 7-14】编写一个程序，功能是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。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class Win extends JFrame implements KeyListener
{
   JTextField text[]=new JTextField[3];
   Container con;
   Win()
   {
      con=getContentPane();
      con.setLayout(new FlowLayout());
      for(int i=0;i<3;i++)
       {
          text[i]=new JTextField(12);
          text[i].addKeyListener(this);
          con.add(text[i]);
       }
     text[0].requestFocusInWindow();
     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     setBounds(10,10,300,300);
     setVisible(true);
     validate();
   }
   public void keyPressed(KeyEvent e)
   {
      JTextField text=(JTextField)e.getSource();
      if(text.getCaretPosition()>=6)
       {
          text.transferFocus();
       }
   }
   public void keyTyped(KeyEvent e) {}
   public void keyReleased(KeyEvent e) {}
```

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

2. 处理复合键

键盘事件 KeyEvent 对象调用 getModifiers()方法，可以返回下列的整数值，它们分别是 InputEvent 类的类常量：ALT_MASK、CTRL_MASK、SHIFT_MASK。

程序可以根据 getModifiers()方法返回的值处理复合键事件。

例如，对于 KeyEvent 对象 e，当使用 Ctrl+X 时，下面的逻辑表达式为 true：

```
e.getModifiers()==InputEvent.CTRL_MASK&&e.getKeyCode()==KeyEvent.VK_V
```

【例 7-15】编写一个程序，功能是用户通过复合键 CTRL+C、CTRL+X 和 CTRL+V 实现文本区内容的拷贝、剪切和粘贴。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class Win extends JFrame implements KeyListener
{
   JTextArea text;
   Win()
   {
      Container con=getContentPane();
      con.setLayout(new FlowLayout());
      text=new JTextArea(30,20);
      text.addKeyListener(this);
      con.add(new JScrollPane(text),BorderLayout.CENTER);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setBounds(10,10,300,300);
      setVisible(true);
      validate();
   }
    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) {}
}
public class k
{
   public static void main(String args[])
    {
       Win win=new Win();
    }
}
```

7.4.4 鼠标事件

1. 鼠标事件的触发

组件是可以触发鼠标事件的事件源。用户的以下 7 种操作都可以使得组件触发鼠标事件：

- 鼠标指针从组件之外进入。
- 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。
- 在组件上运动鼠标指针。

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 与 MouseMotionListener 接口

Java 分别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，那么用户的以下 5 种操作可使得事件源触发鼠标事件：

- 鼠标指针从组件之外进入。
- 鼠标指针从组件内退出。
- 鼠标指针停留在组件上面时，按下鼠标。
- 鼠标指针停留在组件上面时，释放鼠标。
- 鼠标指针停留在组件上面时，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：

- 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。
- 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。
- 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，那么用户的以下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。
- 在组件上运动鼠标指针。

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。

mouseMoved(MouseEvent e)：负责处理鼠标移动事件。

3．MouseEvent 类

在处理鼠标事件时，程序经常关注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单击的次数。

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，返回的是类常量 BUTTON3_MASK。

getSource()鼠标事件调用该方法返回触发当前鼠标事件的事件源。

【例 7-16】编写一个程序，功能是：分别监视按钮、标签和窗口内容面板上的鼠标事件，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。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class MyWindow extends JFrame implements MouseListener
{
   JButton button;
   JTextArea textArea;
   Container con;
   MyWindow()
   {
     con=getContentPane();
     con.setLayout(new FlowLayout());
     con.addMouseListener(this);
```

```
        button=new JButton("我是按钮");
        button.addMouseListener(this);
        textArea=new JTextArea(8,28);
        con.add(button);
        con.add(new JScrollPane(textArea));
        setBounds(100,100,150,150);
        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 在按钮上鼠标放开,位置:"+"("+e.getX()+","+e.getY()+")");
        }
    }
    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)
    {
        if(e.getSource()==con)
        {
          textArea.append("\n 在鼠标离开容器,位置:"+"("+e.getX()+","+e.getY()+")");
        }
    }
    public void mouseClicked(MouseEvent e)
    {
       if(e.getModifiers()==InputEvent.BUTTON3_MASK&&e.getClickCount()>=2)
        {
          textArea.setText("你双击了鼠标右键");
        }
    }
}
class k
{
```

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

4. 鼠标位置的坐标变换

容器中的一个组件触发鼠标事件后，鼠标事件 e 调用 getX()和 getY()方法可以返回鼠标指针在事件源坐标系中的坐标。有时，程序可能还需要知道鼠标指针在容器坐标系中的坐标，这就需要进行坐标变换。

可以使用 Java.Swing 包中的 SwingUtilities 类的类方法：

```
public static Point convertPoint(Component source, int x, int y,Component destination)
```

根据鼠标指针在当前事件源 source 坐标系中的坐标(x,y)，得到鼠标在容器 destination 坐标系中的坐标，该方法返回一个 Point 对象，该对象再调用 getX()和 getY()方法就可以获取鼠标在容器 destination 坐标系中的坐标。

【例 7-17】编写一个程序，功能是用户使用鼠标拖动按钮来使该组件在容器中的位置发生改变。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class Win extends JFrame implements MouseMotionListener,MouseListener
{
   JButton button[]=new JButton[6];
   JTextField text;
   JLayeredPane layerPane;
   Container con;
   int xr,yr;
   Win()
   {
     layerPane=new JLayeredPane();
     layerPane.setLayout(new FlowLayout());
     con=getContentPane();
     con.add(layerPane,BorderLayout.CENTER);
     for(int i=0;i<button.length;i++)
        {
          button[i]=new JButton("用鼠标拖动我"+i);
          button[i].addMouseMotionListener(this);
          button[i].addMouseListener(this);
          layerPane.add(button[i],JLayeredPane.DEFAULT_LAYER);
        }
     con.addMouseMotionListener(this);
     setBounds(10,10,300,300);
     setVisible(true);
```

```
        validate();
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      }
      public void mouseMoved(MouseEvent e){}
      public void mouseDragged(MouseEvent e)
      {
        JComponent source=(JComponent)e.getSource();
        layerPane.setLayer(source,JLayeredPane.DRAG_LAYER);
        int x=e.getX();
        int y=e.getY();
        java.awt.Point point=SwingUtilities.convertPoint(source,x,y,con);
        int x1=(int)point.getX();          //获取鼠标指针在容器坐标系中的坐标
        int y1=(int)point.getY();
        int w=source.getSize().width;
        int h=source.getSize().height;
        source.setLocation(x1-xr,y1-yr);  //鼠标当前位置减去按钮左上角距离鼠标位置的
水平与竖直距离
      }                                    //即是按钮的当前位置
      }
      public void mouseReleased(MouseEvent e)
      {
        JComponent source=(JComponent)e.getSource();
        layerPane.setLayer(source,JLayeredPane.DEFAULT_LAYER);
      }
      public void mousePressed(MouseEvent e)
      {
       JComponent source=(JComponent)e.getSource();
       xr=e.getX();    //获取鼠标被按下时相对于按钮坐标系的坐标
       yr=e.getY();    //即按钮左上角距离鼠标位置的水平与竖直距离
      }
      public void mouseEntered(MouseEvent e){}
      public void mouseExited(MouseEvent e){}
      public void mouseClicked(MouseEvent e){}
  }
  public class k
  {  public static void main(String args[])
      {
        Win win=new Win();
      }
  }
```

5. 鼠标事件的转移

假如我们正监视一个容器上的鼠标事件，而容器中添加了一些组件，则当在组件上进行单击鼠标、移动鼠标等操作时，容器将不知道这些操作的发生。这时可以使用 javax.swing 包中的 SwingUtilities 类的静态方法 MouseEvent convertMouseEvent(Component source, MouseEvent sourceEvent, Component destination)使 sourceEvent 鼠标事件从 source 事件源转移到

destination 事件源上（声东击西）。

【例 7-18】编写一个程序，功能是：模拟扫雷游戏，当在一个组件上按下鼠标时，同样的事情也就会发生在某些其他组件上。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
class Win extends JFrame implements MouseListener
{
   JLabel b[]=new JLabel[12];
   Icon iconGreen=new ImageIcon("click.gif"),
        iconRed=new ImageIcon("红心.gif");
   Win()
   {
     Container con=getContentPane();
     con.setLayout(new GridLayout(3,4));
     for(int i=0;i<b.length;i++)
       {
        b[i]=new JLabel(iconGreen);
        con.add(b[i]);
        b[i].addMouseListener(this);
        b[i].setBorder(BorderFactory.createBevelBorder(
                    BevelBorder.RAISED,Color.red,Color.white));
       }
     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     setBounds(10,10,380,300);
     setVisible(true);
     validate();
   }
   public void mousePressed(MouseEvent e)
   {
     JLabel label=(JLabel)e.getSource();
     label.setIcon(iconRed);
     int m=(int)(Math.random()*b.length);
     e=SwingUtilities.convertMouseEvent(label,e,b[m]);//转移鼠标事件随机转移到一个组件
        if(e.getSource()==b[m])
          {
             b[m].setIcon(iconRed);
          }
   }
   public void mouseReleased(MouseEvent e)
   {
    for(int i=0;i<b.length;i++)
```

```
            {
               b[i].setIcon(iconGreen);
            }
         }
         public void mouseEntered(MouseEvent e)  {}
         public void mouseExited(MouseEvent e) {}
         public void mouseClicked(MouseEvent e){}
      }
      public class k
      {
         public static void main(String args[])
         {
            Win win=new Win();
         }
      }
```

6. 获取鼠标在系统桌面上的坐标

SDK1.5 在 java.awt 包中新增了一个类 PointerInfo，该类可以帮助程序获取鼠标指针在系统图形设备中的位置坐标。使用 PointerInfo 的类方法 getPointerInfo()可以实例化一个 PointerInfo 对象。例如：

```
PointerInfo pi=MouseInfo.getPointerInfo();
```

PointerInfo 对象调用 Point getLocation();方法返回一个 Point 对象，对象再调用 getX()和 getY()返回鼠标在桌面上的坐标值。需要注意的是，当鼠标运动时，必须及时更新 PointerInfo 对象，因为 PointerInfo 对象中存储的 Point 对象不会自动更新。

7. 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 上的弹出的位置，位置坐标(x,y)按 invoker 的坐标系。

【例 7-19】编写一个程序，功能是：在文本区上单击右键时，在鼠标位置处弹出菜单，用户选择相应的菜单项可以将文本区中选中的内容复制、剪切到系统的剪贴板中或将剪贴板中的文本内容粘贴到文本区。

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
class EditWindow extends JFrame implements ActionListener
{
   JPopupMenu menu;
   JMenuItem  itemCopy,itemCut,itemPaste;
   JTextArea text;
   EditWindow(String s)
```

```
    {
      super(s);
      setSize(260,270);
      setLocation(120,120);
      setVisible(true);
      menu=new JPopupMenu("弹出式菜单");
      itemCopy=new JMenuItem("拷贝");
      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());
                                   }
                              }
                           }
                         );
      setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
      Container con=getContentPane();
      con.add(new JScrollPane(text),BorderLayout.CENTER);
      con.validate();
      validate();
      itemCopy.addActionListener(this);
      itemCut.addActionListener(this);
      itemPaste.addActionListener(this);
    }

   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();
    }
}
public class k
```

```
{
   public static void main(String args[])
   {
     EditWindow win=new EditWindow("窗口");
   }
}
```

7.4.5 关于监听者的总结

JDK1.1 之后 Java 采用的是事件源——事件监听者模型，引发事件的对象称为事件源，而接收并处理事件的对象是事件监听者，无论应用程序还是小程序都采用这一机制。

引入事件处理机制后的编程基本方法如下：

（1）对 java.awt 中组件实现事件处理必须使用 java.awt.event 包，所以在程序开始应加入 import java.awt.event.*语句。

（2）用如下语句设置事件监听者：

事件源.add XXListener

XXListener 代表某种事件监听者。

（3）事件监听者所对应的类实现事件所对应的接口 XXListener，并重写接口中的全部方法。

这样就可以处理图形用户界面中的对应事件了，要删除事件监听者可以使用语句：

事件源.remove XXListener

1. 事件监听器

每类事件都有对应的事件监听器，监听器是接口，根据动作来定义方法。

例如，与键盘事件 KeyEvent 相对应的接口是：

```
public interface KeyListener extends EventListener
{
     public void keyPressed(KeyEvent ev);
     public void keyReleased(KeyEvent ev);
     public void keyTyped(KeyEvent ev);
}
```

注意到在本接口中有 3 个方法，那么 Java 运行时系统何时调用哪个方法？其实根据这 3 个方法的方法名就能够知道应该是什么时候调用哪个方法执行了。当键盘刚按下时，将调用 keyPressed()方法执行，当键盘抬起来时，将调用 keyReleased()方法执行，当键盘敲击一次时，将调用 keyTyped()方法执行。

又例如窗口事件接口：

```
public interface WindowListener extends EventListener
{
     public void windowClosing(WindowEvent e);
     //把退出窗口的语句写在本方法中
     public void windowOpened(WindowEvent e);
     //窗口打开时调用
     public void windowIconified(WindowEvent e);
     //窗口图标化时调用
```

```
    public void windowDeiconified(WindowEvent e);
    //窗口非图标化时调用
    public void windowClosed(WindowEvent e);
    //窗口关闭时调用
    public void windowActivated(WindowEvent e);
    //窗口激活时调用
    public void windowDeactivated(WindowEvent e);
    //窗口非激活时调用
}
```

AWT 的组件类中提供注册和注销监听器的方法：

- 注册监听器：

```
public void add<ListenerType> (<ListenerType>listener);
```

- 注销监听器：

```
public void remove<ListenerType> (<ListenerType>listener);
```

例如 Button 类：

```
public class Button extends Component
{
   ……
   public synchronized void addActionListener(ActionListener l);
   public synchronized void removeActionListener(ActionListener l);
}
```

Java 将所有组件可能发生的事件进行分类，具有共同特征的事件被抽象为一个事件类 AWTEvent，其中包括 ActionEvent 类（动作事件）、MouseEvent 类（鼠标事件）、KeyEvent 类（键盘事件）等。表 7-7 列出了常用 Java 事件类、处理该事件的接口及接口中的方法。

表 7-7　常用 Java 事件类、处理该事件的接口及接口中的方法

事件类/接口名称	接口方法及说明
ActionEvent 动作事件类 ActionListener 接口	actionPerformed(ActionEvent e) 单击按钮、选择菜单项或在文本框中按回车时
AdjustmentEvent 调整事件类 AdjustmentListener 接口	adjustmentValueChanged(AdjustmentEvent e) 当改变滚动条滑块位置时
ComponentEvent 组件事件类 ComponentListener 接口	componentMoved(ComponentEvent e) 组件移动时 componentHidden(ComponentEvent e) 组件隐藏时 componentResized(ComponentEvent e) 组件缩放时 componentShown(ComponentEvent e) 组件显示时
ContainerEvent 容器事件类 ContainerListener 接口	componentAdded(ContainerEvent e) 添加组件时 componentRemoved(ContainerEvent e) 移除组件时
FocusEvent 焦点事件类 FocusListener 接口	focusGained(FocusEvent e) 组件获得焦点时 focusLost(FocusEvent e) 组件失去焦点时
ItemEvent 选择事件类 ItemListener 接口	itemStateChanged(ItemEvent e) 选择复选框、选项框、单击列表框、选中带复选框菜单时

续表

事件类/接口名称	接口方法及说明
KeyEvent 键盘事件类 KeyListener 接口	keyPressed(KeyEvent e) 键按下时 keyReleased(KeyEvent e) 键释放时 keyTyped(KeyEvent e) 击键时
MouseEvent 鼠标事件类 MouseListener 接口	mouseClicked(MouseEvent e) 单击鼠标时 mouseEntered(MouseEvent e) 鼠标进入时 mouseExited(MouseEvent e) 鼠标离开时 mousePressed(MouseEvent e) 鼠标键按下时 mouseReleased(MouseEvent e) 鼠标键释放时
MouseEvent 鼠标移动事件类 MouseMotionListener 接口	mouseDragged(MouseEvent e) 鼠标拖放时 mouseMoved(MouseEvent e) 鼠标移动时
TextEvent 文本事件类 TextListener 接口	textValueChanged(TextEvent e) 文本框、多行文本框内容修改时
WindowEvent 窗口事件类 WindowListener 接口	windowOpened(WindowEvent e) 窗口打开后 windowClosed(WindowEvent e) 窗口关闭后 windowClosing(WindowEvent e) 窗口关闭时 windowActivated(WindowEvent e) 窗口激活时 windowDeactivated(WindowEvent e) 窗口失去焦点时 windowIconified(WindowEvent e) 窗口最小化时 windowDeiconified(WindowEvent e) 最小化窗口还原时

每个事件类都提供下面常用的方法：

（1）public int getID()，返回事件的类型。

（2）public Object getSource()，返回事件源的引用。

当多个事件源触发的事件由一个共同的监听器处理时，我们可以通过 getSource 方法判断当前的事件源是哪一个组件。

2. 事件适配器

为了进行事件处理，需要创建实现 Listener 接口的类，而按 Java 的规定，在实现该接口的类中，必须同时实现接口中所定义的全部方法。在具体程序设计过程中，有可能我们只用到接口中的一个或几个方法。为了方便，Java 为那些声明了多个方法的 Listener 接口提供了一个对应的适配器（Adapter）类，在该类中实现了对应接口的所有方法，只是方法体为空。比如，窗口事件适配器的定义如下：

```
public abstract class WindowAdapter extends Object implements WindowListener{
public void windowOpened(WindowEvent e){}
public void windowClosing(WindowEvent e){}
public void windowClosed(WindowEvent e){}
public void windowIconified(WindowEvent e){}
public void windowDecionified(WindowEvent e){}
public void windowActivated(WindowEvent e){}
public void windowDeactivated(WindowEvent e){}
}
```

由于在接口对应适配器类中实现了接口的所有方法，因此，在创建新类时，可以不实现接口，而是只继承某个适当的适配器，并且仅覆盖所关心的事件处理方法即可。注意，在使用适配器时，一定确保所覆盖的方法书写正确。

表 7-8 列出了接口及对应的适配器类。你只需把接口名称中的 Listener 用 Adapter 代替即为对应适配器的名称。接口 ActionListener、AdjustmentListener、ItemListener、TextListener 均只有一个方法，不需要定义适配器。

表 7-8　接口及对应适配器类

接口名称	适配器名称
ComponentListener	ComponentAdapter
ContainerListener	ContainerAdapter
FocusListener	FocusAdapter
KeyListener	KeyAdapter
MouseListener	MouseAdapter
MouseMotionListener	MouseMotionAdapter
WindowListener	WindowAdapter

在学习处理事件时，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。我们通过处理文本框这个具体组件上的事件，来掌握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）监视器（监听者）。系统需要一个对象对事件源进行监视，以便对发生的事件作出处理。事件源通过调用相应的方法使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 类创建了一个事件响应对象，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，也就是说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。

当事件源发生某事件时，监视器立刻监视到，并使用系统自动调用被实现的某个接口方法，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，是由系统自动完成的。程序只需让事件源获得正确的监视器，即把实现了接口的对象传递给方法 addActionListener(ActionListener listener)即可。

（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。

【例 7-20】编写一个程序，功能是：titleText 和 passwordText 将窗口作为监视器，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。运行效果如图 7.17 所示。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class MyWindow extends JFrame implements ActionListener
{
   JTextField titleText;
   JPasswordField passwordText;
   MyWindow(String s)
   {
     super(s);
     titleText=new JTextField(10);
     passwordText=new JPasswordField (10);
     passwordText.setEchoChar('*');
     titleText.addActionListener(this);
     passwordText.addActionListener(this);
     Container con=getContentPane();
     con.setLayout(new FlowLayout());
     con.add(titleText);
     con.add(passwordText);
     setBounds(100,100,150,150);
     setVisible(true);
     validate();
     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
   public void actionPerformed(ActionEvent e)
   {
     if(e.getSource()==titleText)
      {
        this.setTitle(titleText.getText());
      }
     else if(e.getSource()==passwordText)
      {
        char c[]=passwordText.getPassword();
        titleText.setText(new String(c));
      }
   }
}
class k
{
   public static void main(String args[ ])
```

```
    {
       MyWindow win=new MyWindow("窗口");
    }
}
```

图 7.17　ActionEvent 事件处理

事件源、监视器、处理事件的接口三者之间的关系：

事件源发生事件后，会执行的操作:

（1）被监视器监视到;

（2）自动产生一个 ActionEvent 对象;

（3）在 titleText 中输入文本回车之后，java.awt.envent 中的 ActionEvent 类创建的这一个事件对象，并将它传递给方法 public void actionPerformed(ActionEvent e)中的参数 e，监视器就会知道所发生的事件，并执行接口中的方法 public void actionPerformed(ActionEvent e)对所发生的事件做出处理。

Java 语言类的层次非常分明，因而只支持单继承。为了实现多重继承的能力，Java 用接口来实现，一个类可以实现多个接口，这种机制比多重继承具有更简单、更灵活、更强的功能。在 AWT 中就经常用到声明和实现多个接口。记住无论实现了几个接口，接口中已定义的方法必须一一实现，如果对某事件不感兴趣，可以不具体实现其方法，而用空的方法体来代替。但却必须将所有方法都写上。

7.5　创建复杂用户界面

我们已经知道轻组件都是容器，但仍有一些经常用来添加组件的轻容器，相对于底层重量容器而言，习惯上称这些轻容器为中间容器。面板也是一个容器，可放一组相关的组件。不同的面板可采用不同的布局方式，这样有利于窗口的总体布局。Java Swing 提供了许多功能各异的中间容器，这里只简单介绍 JPanel 面板、JScrollPane 滚动窗格、JSplitPane 拆分窗格和 JLayeredPane 分层窗格。

7.5.1　面板容器（JPanel 面板）

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。

JPanel 面板的构造方法：

public JPanel()：创建一个 FlowLayout 布局的面板。

public JPanel(LayoutManger layout)：创建一个指定布局的面板。

例如：

```
JPanel p1=new JPanel(new BorderLayout());
```

说明：面板的默认布局是 FlowLayout，不同于 Container 的默认。

7.5.2 滚动窗格容器（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 c)构造一个滚动窗格。

它是一个能够自己产生滚动条的容器，通常只包容一个组件，并且根据这个组件的大小自动产生滚动条。比如上面讲 JTextArea 的时候提到：JTextArea 会随用户输入的内容自动扩展大小，很容易打破各组件的布局。但是，如果我们将它包容在一个滚动窗格中，它的扩展就不会直接反映在大小的变化上，而会反映在滚动窗格的滚动条上，也就不会打破各组件的布局。

7.5.3 拆分窗格容器（JSplitPane 拆分窗格）

拆分窗格就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。水平拆分窗口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。

7.5.4 分层窗格容器（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 容器将容器分成 5 个层，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(5 层)、PALETTE_LAYER(4 层)、MODAL_LAYER(3 层)、POPUP_LAYER(2 层)、DRAG_LAYER(1 层，最高层) 。高层中的组件可以将低层中的组件遮挡住。

DRAG_LAYER 层是最上面的层，如果 JLayeredPane 中添加了许多组件，且其中有些组件可用鼠标移动时，可将这些组件放到 DRAG_LAYER 层，这样，组件在移动过程中，就不会被其他组件遮挡。添加到同一层上的组件，如果发生重叠，先添加的会遮挡后添加的组件。

JLayeredPane 对象调用 public void setLayer(Component c,int layer)可以重新设置组件 c 所在的层，调用 public int getLayer(Component c)可以获取组件 c 所在的层数。

【例 7-21】编写一个程序，功能是在 JLayerdPane 容器中添加了 5 个组件，分别位于不同的层上。运行效果如图 7.18 所示。

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

```
class k
{
   public static void main(String args[])
   {
      JFrame win=new JFrame("窗体");
      win.setBounds(100,100,300,300);
      win.setVisible(true);

      JButton b1=new JButton("我在 DEFAULT_LAYER"),
             b2=new JButton("我在 PALETTE_LAYER"),
             b3 =new JButton("我在 MODAL_LAYER"),
             b4 =new JButton("我在 POPUP_LAYER"),
             b5=new JButton("我在 DRAG_LAYER");

      Container contenetPane=win.getContentPane();
      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);

      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);

      b1.setBackground(Color.red);
      b2.setBackground(Color.pink);
      b3.setBackground(Color.green);
      b4.setBackground(Color.orange);
      b5.setBackground(Color.yellow);

      contenetPane.add(pane,BorderLayout.CENTER);
      contenetPane.validate();
      win.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
}
```

图 7.18　使用 JLayeredPane 容器的窗口

本章小结

本章首先介绍 GUI 程序设计的基础知识，其中包括常用的 Swing 组件，如框架、按钮、标签、文本框、下拉列表、菜单等，以及常用的布局设计方法；然后介绍了 GUI 程序设计的一些高级内容，包括事件处理机制、复杂的用户界面设计。本章是 GUI 应用程序设计的基础。通过这部分的学习，希望读者能够熟练运用 Swing 和 AWT 进行图形用户界面的开发，从而达到学以致用的目的。

习题 7

1．设计一个程序如图 7.19 所示。在窗口中实现一个“文件”菜单和一个“系统”菜单的功能，“文件”菜单有“打开”、“保存”、“关闭”等内容，“系统”菜单有“关于”、“退出”等内容，当单击某个菜单时，在文本区域显示提示信息。

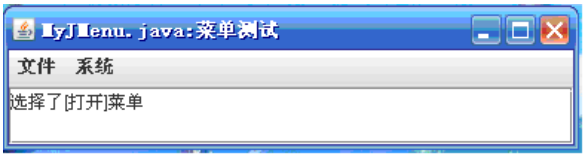


图 7.19　菜单测试

2．设计一个程序，如图 7.20 所示。

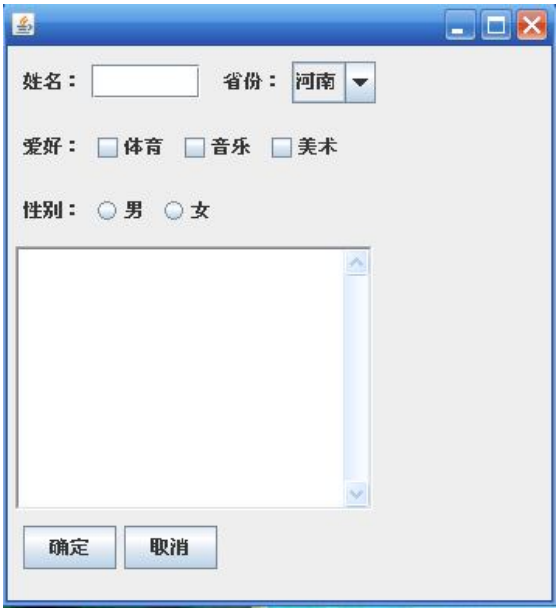


图 7.20　设计窗口样例