



22

GUI Components: Part 2



An actor entering through the door, you've got nothing. But if he enters through the window, you've got a situation.

— Billy Wilder

...the force of events wakes slumberous talents.

— Edward Hoagland

You and I would see more interesting photography if they would stop worrying, and instead, apply horse-sense to the problem of recording the look and feel of their own era.

— Jessie Tarbox Beals



OBJECTIVES

In this chapter you will learn:

- To create and manipulate sliders, menus, pop-up menus and windows.
- To change the look-and-feel of a GUI, using Swing's pluggable look-and-feel.
- To create a multiple-document interface with `JDesktopPane` and `JInternalFrame`.
- To use additional layout managers.



Outline

- 22.1 **Introduction**
- 22.2 JSlider
- 22.3 **Windows: Additional Notes**
- 22.4 **Using Menus with Frames**
- 22.5 JPopupMenu
- 22.6 **Pluggable Look-and-Feel**
- 22.7 JDesktopPane **and** JInternalFrame
- 22.8 JTabbedPane
- 22.9 **Layout Managers: BorderLayout and GridBagLayout**
- 22.10 **Wrap-Up**



22.1 Introduction

- **Pluggable look-and-feel (PLAF)**
 - Swing can customize the look-and-feel of the GUI
 - Motif
 - A popular UNIX look-and-feel
- **Multiple-document interface (MDI)**
 - A main window (the parent window) containing other windows (child windows)
 - Manages several open documents parallel



22.2 JSlider

- **JSlider**
 - Enables the user to select from a range of integer values
 - Inherits from **JComponent**
 - Contains:
 - **Tick marks**
 - Can display major tick marks, minor tick marks and labels for tick marks
 - Are not displayed by default
 - **Thumb**
 - Allows the user to select a value
 - **Snap-to ticks**
 - Cause the thumb to snap to the closest tick mark





Fig. 22.1 | JSlider component with horizontal orientation.



22.2 JSlider (Cont.)

- If a `JSlider` has the focus (is the currently selected GUI component in the user interface)
 - Left/right arrow keys cause the thumb of the `JSlider` to decrease/increase by 1
 - Down/up arrow keys cause the thumb of the `JSlider` to decrease/increase by 1
 - *PgDn* (page down)/*PgUp* (page up) keys cause the thumb of the `JSlider` to decrease/increase by block increments of one-tenth of the range of values
 - *Home/End* keys move the thumb of the `JSlider` to the minimum/maximum value of the `JSlider`



22.2 JSlider (Cont.)

- **Can have either horizontal or vertical orientation**
 - **Minimum value is at the left or bottom end of the JSlider**
 - **Maximum value is at the right or top end of the JSlider**
 - **JSlider method `setInverted` reverses the minimum and maximum value positions**
- **Generate `ChangeEvent`s in response to user interactions**
 - **An object of a class that implements interface `ChangeListener` and declares method `stateChanged` can respond to `ChangeEvent`s**



Look-and-Feel Observation 22.1

If a new GUI component has a minimum width and height (i.e., smaller dimensions would render the component ineffective on the display), override method `getMinimumSize` to return the minimum width and height as an instance of class `Dimension`.



Software Engineering Observation 22.1

For many GUI components, method `getMinimumSize` is implemented to return the result of a call to the component's `getPreferredSize` method.



```
1 // Fig. 22.2: OvalPanel.java
2 // A customized JPanel class.
3 import java.awt.Graphics;
4 import java.awt.Dimension;
5 import javax.swing.JPanel;
6
7 public class OvalPanel extends JPanel
8 {
9     private int diameter = 10; // default diameter of 10
10
11     // draw an oval of the specified diameter
12     public void paintComponent( Graphics g )
13     {
14         super.paintComponent( g );
15
16         g.fillOval( 10, 10, diameter, diameter ); // draw circle
17     } // end method paintComponent
18
19     // validate and set diameter, then repaint
20     public void setDiameter( int newDiameter )
21     {
22         // if diameter invalid, default to 10
23         diameter = ( newDiameter >= 0 ? newDiameter : 10 );
24         repaint(); // repaint panel
25     } // end method setDiameter
26
```

Used as the width and height of the bounding box in which the circle is displayed

Draws a filled circle

Change the circle's **diameter** and **repaint**



```
27 // used by layout manager to determine preferred size
28 public Dimension getPreferredSize()
29 {
30     return new Dimension( 200, 200 );
31 } // end method getPreferredSize
32
33 // used by layout manager to determine minimum size
34 public Dimension getMinimumSize()
35 {
36     return getPreferredSize();
37 } // end method getMinimumSize
38 } // end class OvalPanel
```

Return the preferred width and height of an **OvalPanel**

Return an **OvalPanel**'s minimum width and height



```

1 // Fig. 22.3: SliderFrame.java
2 // Using JSliders to size an oval.
3 import java.awt.BorderLayout;
4 import java.awt.Color;
5 import javax.swing.JFrame;
6 import javax.swing.JSlider;
7 import javax.swing.SwingConstants;
8 import javax.swing.event.ChangeListener;
9 import javax.swing.event.ChangeEvent;
10
11 public class SliderFrame extends JFrame
12 {
13     private JSlider diameterJSlider; // slider to select diameter
14     private OvalPanel myPanel; // panel to draw circle
15
16     // no-argument constructor
17     public SliderFrame()
18     {
19         super( "Slider Demo" );
20
21         myPanel = new OvalPanel(); // create panel to draw circle
22         myPanel.setBackground( Color.YELLOW ); // set background to yellow
23
24         // set up JSlider to control diameter value
25         diameterJSlider =
26             new JSlider( SwingConstants.HORIZONTAL, 0, 200, 10 );
27         diameterJSlider.setMajorTickSpacing( 10 ); // create tick
28         diameterJSlider.setPaintTicks( true ); // paint ticks on slider
29

```

Create `OvalPanel` object `myPanel`

Create `JSlider` object
`diameterSlider` as a horizontal
JSlider with a range of 0-200 and a
initial value of 10

Indicate that each major-tick mark represents 10
values and that the tick marks should be displayed



```
30 // register JSlider event listener
31 diameterJSlider.addChangeListener(
32
33     new ChangeListener() // anonymous inner class
34     {
35         // handle change in slider value
36         public void stateChanged(ChangeEvent e)
37         {
38             myPanel.setDiameter( diameterJSlider.getValue() );
39         } // end method stateChanged
40     } // end anonymous inner class
41 ); // end call to addChangeListener
42
43 add( diameterJSlider, BorderLayout.SOUTH ); //
44 add( myPanel, BorderLayout.CENTER ); // add panel
45 } // end SliderFrame constructor
46 } // end class SliderFrame
```

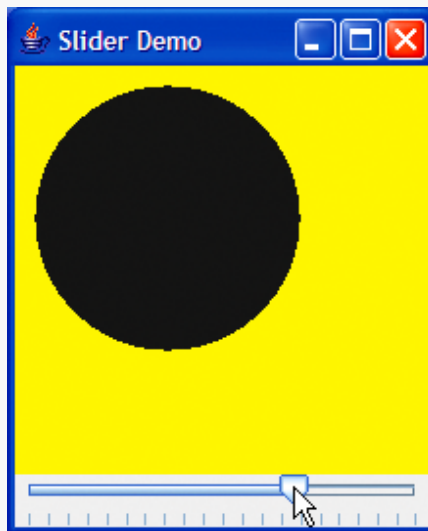
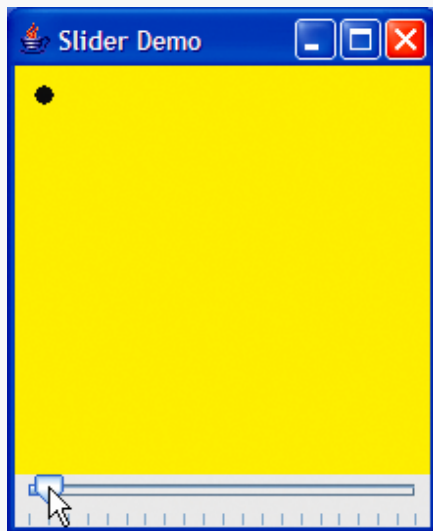
Register a **ChangeListener** to handle **diameterSlider**'s events

Method **stateChanged** is called in response to a user interaction

Call **myPanel**'s **setDiameter** method and pass the current thumb position value returned by **JSlider** method **getValue**



```
1 // Fig. 22.4: SliderDemo.java
2 // Testing SliderFrame.
3 import javax.swing.JFrame;
4
5 public class SliderDemo
6 {
7     public static void main( String args[] )
8     {
9         SliderFrame sliderFrame = new SliderFrame();
10        sliderFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        sliderFrame.setSize( 220, 270 ); // set frame size
12        sliderFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class SliderDemo
```



22.3 Windows: Additional Notes

- **JFrame**

- Is a window with a title bar and a border
- A subclass of `java.awt.Frame`
 - Which is a subclass of `java.awt.Window`
- One of the few Swing GUI components that is not a lightweight GUI component
- Java application windows look like every other window displayed on that platform



Good Programming Practice 22.1

**A windows is an expensive system resource.
Return it to the system when it is no longer
needed.**



22.3 Windows: Additional Notes (Cont.)

- **JFrame method setDefaultCloseOperation**
determines what happens when the user closes the window
 - **DISPOSE_ON_CLOSE**
 - Dispose of the window to return resources to the system
 - **DO_NOTHING_ON_CLOSE**
 - Indicates that the program will determine what to do when the user indicates that the window should close
 - **HIDE_ON_CLOSE**
 - The default
- **JFrame method setVisible**
 - Display the window on the screen
- **JFrame method setLocation**
 - Specify the window's position when it appears on the screen



Common Programming Error 22.1

Forgetting to call method `setVisible` on a window is a runtime logic error—the window is not displayed.



Common Programming Error 22.2

Forgetting to call the `setSize` method on a window is a runtime logic error—only the title bar appears.



22.3 Windows: Additional Notes (Cont.)

- **User manipulation of the window generates window events**
 - Method `addWindowListener` registers event listeners for window events
 - Interface `WindowListener` provides seven window-event-handling methods
 - `windowActivated` – called when the user makes a window the main window
 - `windowClosed` – called after the window is closed
 - `windowClosing` – called when the user initiates closing of the window



22.3 Windows: Additional Notes (Cont.)

- **windowDeactivated** – called when the user makes another window the main window
- **windowDeiconified** – called when the user restores a window from being minimized
- **windowIconified** – called when the user minimizes a window
- **windowOpened** – called when a program first displays a window on the screen



22.4 Using Menus with Frames

- **Menus**

- Allow the user to perform actions without unnecessarily cluttering a GUI with extra components
- Can be attached only to objects of the classes that provide member `setMenuBar`, such as `JFrame` and `JApplet`
- **Class `MenuBar`**
 - Contains the methods necessary to manage a menu bar
- **Class `JMenu`**
 - Contains the methods necessary for managing menus
- **Class `JMenuItem`**
 - Contains the methods necessary to manage menu items
 - Can be used to initiate an action or can be a submenu



Look-and-Feel Observation 22.2

Menus simplify GUIs because components can be hidden within them. These components will only be visible when the user looks for them by selecting the menu.



22.4 Using Menus with Frames (Cont.)

- **Class JCheckBoxMenuItem**
 - **Contains the methods necessary to manage menu items that can be toggled on or off**
- **Class JRadioButtonMenuItem**
 - **Contains the methods necessary to manage menu items that can be toggled on or off like JCheckBoxMenuItems**
 - **When multiple JRadioButtonMenuItems are maintained as part of a ButtonGroup, only one item in the group can be selected at a given time**
- **Mnemonics**
 - **Special characters that can provide quick access to a menu or menu item from the keyboard**



```
1 // Fig. 22.5: MenuFrame.java
2 // Demonstrating menus.
3 import java.awt.Color;
4 import java.awt.Font;
5 import java.awt.BorderLayout;
6 import java.awt.event.ActionListener;
7 import java.awt.event.ActionEvent;
8 import java.awt.event.ItemListener;
9 import java.awt.event.ItemEvent;
10 import javax.swing.JFrame;
11 import javax.swing.JRadioButtonMenuItem;
12 import javax.swing.JCheckBoxMenuItem;
13 import javax.swing.JOptionPane;
14 import javax.swing.JLabel;
15 import javax.swing.SwingConstants;
16 import javax.swing.ButtonGroup;
17 import javax.swing.JMenu;
18 import javax.swing.JMenuItem;
19 import javax.swing.JMenuBar;
20
```



```

21 public class MenuFrame extends JFrame
22 {
23     private final Color colorValues[] =
24         { Color.BLACK, Color.BLUE, Color.RED, Color.GREEN };
25     private JRadioButtonMenuItem colorItems[]; // color menu items
26     private JRadioButtonMenuItem fonts[]; // font menu items
27     private JCheckBoxMenuItem styleItems[]; // font style menu items
28     private JLabel displayJLabel; // displays sample text
29     private ButtonGroup fontButtonGroup; // manages font menu items
30     private ButtonGroup colorButtonGroup; // manages color menu items
31     private int style; // used to create style for font
32
33     // no-argument constructor set up GUI
34     public MenuFrame()
35     {
36         super( "Using JMenus" );
37
38         JMenu fileMenu = new JMenu( "File" ); // create file menu
39         fileMenu.setMnemonic( 'F' ); // set mnemonic to F
40
41         // create About... menu item
42         JMenuItem aboutItem = new JMenuItem( "About..." );
43         aboutItem.setMnemonic( 'A' ); // set mnemonic to A
44         fileMenu.add( aboutItem ); // add about item to file menu
45         aboutItem.addActionListener(
46

```

Create a **JMenu**

Call **JMenu** method
setMnemonic

Add the "About..." **JMenuItem**
to **fileMenu**



```

47 new ActionListener() // anonymous inner class
48 {
49     // display message dialog when user selects About...
50     public void actionPerformed( ActionEvent event )
51     {
52         JOptionPane.showMessageDialog( MenuFrame.this,
53             "This is an example\nof using menus",
54             "About", JOptionPane.PLAIN_MESSAGE );
55     } // end method actionPerformed
56 } // end anonymous inner class
57 ); // end call to addActionListener

```

Create an **ActionListener** to process **aboutItem**'s action event

Display a message dialog box

Create and add menu item **exitItem**

```

59 JMenuItem exitItem = new JMenuItem( "Exit" ); // create exit item
60 exitItem.setMnemonic( 'x' ); // set mnemonic to x
61 fileMenu.add( exitItem ); // add exit item to file menu
62 exitItem.addActionListener(

```

Register an **ActionListener** that terminates the application

```

63
64 new ActionListener() // anonymous inner class
65 {
66     // terminate application when user clicks exitItem
67     public void actionPerformed( ActionEvent event )
68     {
69         system.exit( 0 ); // exit application
70     } // end method actionPerformed
71 } // end anonymous inner class
72 ); // end call to addActionListener

```



```

74 JMenuBar bar = new JMenuBar(); // create menu bar
75 setJMenuBar( bar ); // add menu bar to application
76 bar.add( fileMenu ); // add file menu to menu bar
77
78 JMenu formatMenu = new JMenu( "Format" ); // create format menu
79 formatMenu.setMnemonic( 'r' ); // set mnemonic to r
80
81 // array listing string colors
82 string colors[] = { "Black", "Blue", "Red", "Green" };
83
84 JMenu colorMenu = new JMenu( "color" ); // create color menu
85 colorMenu.setMnemonic( 'c' ); // set mnemonic to c
86
87 // create radiobutton menu items for colors
88 colorItems = new JRadioButtonMenuItem[ colors.length ];
89 colorButtonGroup = new ButtonGroup(); // manages colors
90 itemHandler itemHandler = new ItemHandler(); // handler for
91
92 // create color radio button menu items
93 for ( int count = 0; count < colors.length; count++ )
94 {
95     colorItems[ count ] =
96         new JRadioButtonMenuItem( colors[ count ] ); // create item
97     colorMenu.add( colorItems[ count ] ); // add item to color menu
98     colorButtonGroup.add( colorItems[ count ] ); // add to group
99     colorItems[ count ].addActionListener( itemHandler );
100 } // end for
101

```

Add **fileMenu** to a **JMenuBar** and attach the **JMenuBar** to the application window

Create menu **formatMenu**

Create submenu **colorMenu**

Create **JRadioButtonMenuItem** array **colorItems**

Create a **ButtonGroup** to ensure that only one of the menu items is selected at a time

Add **JRadioButtonMenuItem**s to **colorMenu** and register **ActionListeners**



```

102 colorItems[ 0 ].setSelected( true ); // select first color item
103
104 formatMenu.add( colorMenu ); // add color menu to format menu
105 formatMenu.addSeparator(); // add separator in menu
106
107 // array listing font names
108 String fontNames[] = { "Serif", "Monospaced", "SansSerif" };
109 JMenu fontMenu = new JMenu( "Font" ); // create font menu
110 fontMenu.setMnemonic( 'n' ); // set mnemonic to n
111
112 // create radiobutton menu items for font names
113 fonts = new JRadioButtonMenuItem[ fontNames.length ];
114 fontButtonGroup = new ButtonGroup(); // manages font names
115
116 // create Font radio button menu items
117 for ( int count = 0; count < fonts.length; count++ )
118 {
119     fonts[ count ] = new JRadioButtonMenuItem( fontNames[ count
120     fontMenu.add( fonts[ count ] ); // add font to font menu
121     fontButtonGroup.add( fonts[ count ] ); // add to button group
122     fonts[ count ].addActionListener( itemHandler ); // add handler
123 } // end for
124
125 fonts[ 0 ].setSelected( true ); // select first Font menu item
126 fontMenu.addSeparator(); // add separator bar to font menu
127

```

Invoke **AbstractButton** method **setSelected**

Add **colorMenu** to **formatMenu** and add a horizontal separator line

Create **JRadioButtonMenuItem** array **fonts**

Create a **ButtonGroup** to ensure that only one of the menu items is selected at a time

Add **JRadioButtonMenuItems** to **colorMenu** and register **ActionListeners**

Set default selection and add horizontal separator



```

128 String styleNames[] = { "Bold", "Italic" }; // names of styles
129 styleItems = new JCheckBoxMenuItem[ styleNames.length ];
130 styleHandler styleHandler = new StyleHandler(); // style handler
131
132 // create style checkbox menu items
133 for ( int count = 0; count < styleNames.length; count++ )
134 {
135     styleItems[ count ] =
136         new JCheckBoxMenuItem( styleNames[ count ] ); // for style
137     fontMenu.add( styleItems[ count ] ); // add to font menu
138     styleItems[ count ].addItemListener( styleHandler ); // handler
139 } // end for
140
141 formatMenu.add( fontMenu ); // add Font menu to Format
142 bar.add( formatMenu ); // add Format menu to menu bar
143
144 // set up label to display text
145 displayJLabel = new JLabel( "Sample Text", SwingConstants.CENTER );
146 displayJLabel.setForeground( colorValues[ 0 ] );
147 displayJLabel.setFont( new Font( "Serif", Font.PLAIN, 72 ) );
148
149 getContentPane().setBackground( Color.CYAN ); // set background
150 add( displayJLabel, BorderLayout.CENTER ); // add displayJLabel
151 } // end MenuFrame constructor
152

```

Create JCheckBoxMenuItems

Add fontMenu to formatMenu and
formatMenu to the JMenuBar



```

153 // inner class to handle action events from menu items
154 private class ItemHandler implements ActionListener
155 {
156     // process color and font selections
157     public void actionPerformed( ActionEvent event )
158     {
159         // process color selection
160         for ( int count = 0; count < colorItems.length; count++ )
161         {
162             if ( colorItems[ count ].isSelected() )
163             {
164                 displayJLabel.setForeground( colorValues[ count ] );
165                 break;
166             } // end if
167         } // end for
168
169         // process font selection
170         for ( int count = 0; count < fonts.length; count++ )
171         {
172             if ( event.getSource() == fonts[ count ] )
173             {
174                 displayJLabel.setFont(
175                     new Font( fonts[ count ].getText(), style, 72 ) );
176             } // end if
177         } // end for
178

```

Determine the selected
JRadioButtonMenuItem

getSource method returns a reference
to the **JRadioButtonMenuItem**
that generated the event



```

179     repaint(); // redraw application
180 } // end method actionPerformed
181 } // end class ItemHandler
182
183 // inner class to handle item events from check box menu items
184 private class StyleHandler implements ItemListener
185 {
186     // process font style selections
187     public void itemStateChanged( ItemEvent e )
188     {
189         style = 0; // initialize style
190
191         // check for bold selection
192         if ( styleItems[ 0 ].isSelected() )
193             style += Font.BOLD; // add bold to style
194
195         // check for italic selection
196         if ( styleItems[ 1 ].isSelected() )
197             style += Font.ITALIC; // add italic to style
198
199         displayJLabel.setFont(
200             new Font( displayJLabel.getFont().getName(), style, 72 ) );
201         repaint(); // redraw application
202     } // end method itemStateChanged
203 } // end class StyleHandler
204 } // end class MenuFrame

```

Called if the user selects a
JCheckBoxMenuItem in the **fontMenu**

Determine whether either or both of the
JCheckBoxMenuItems are selected

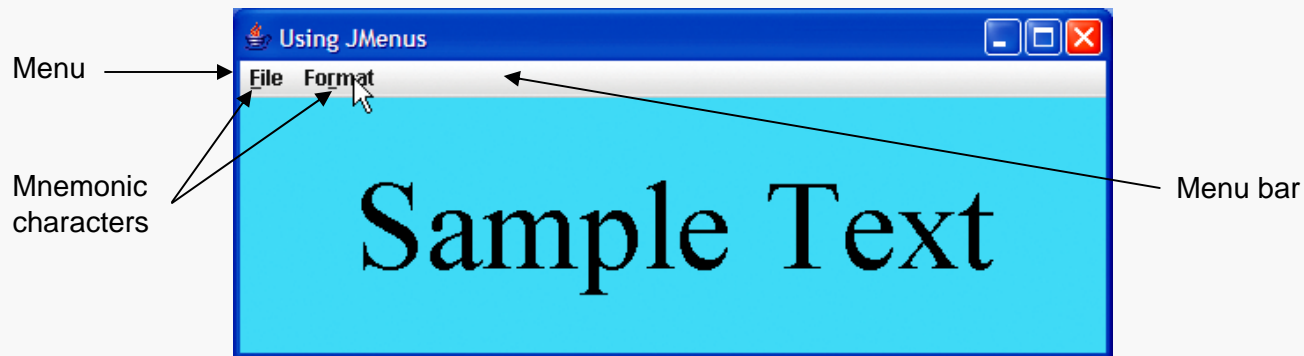


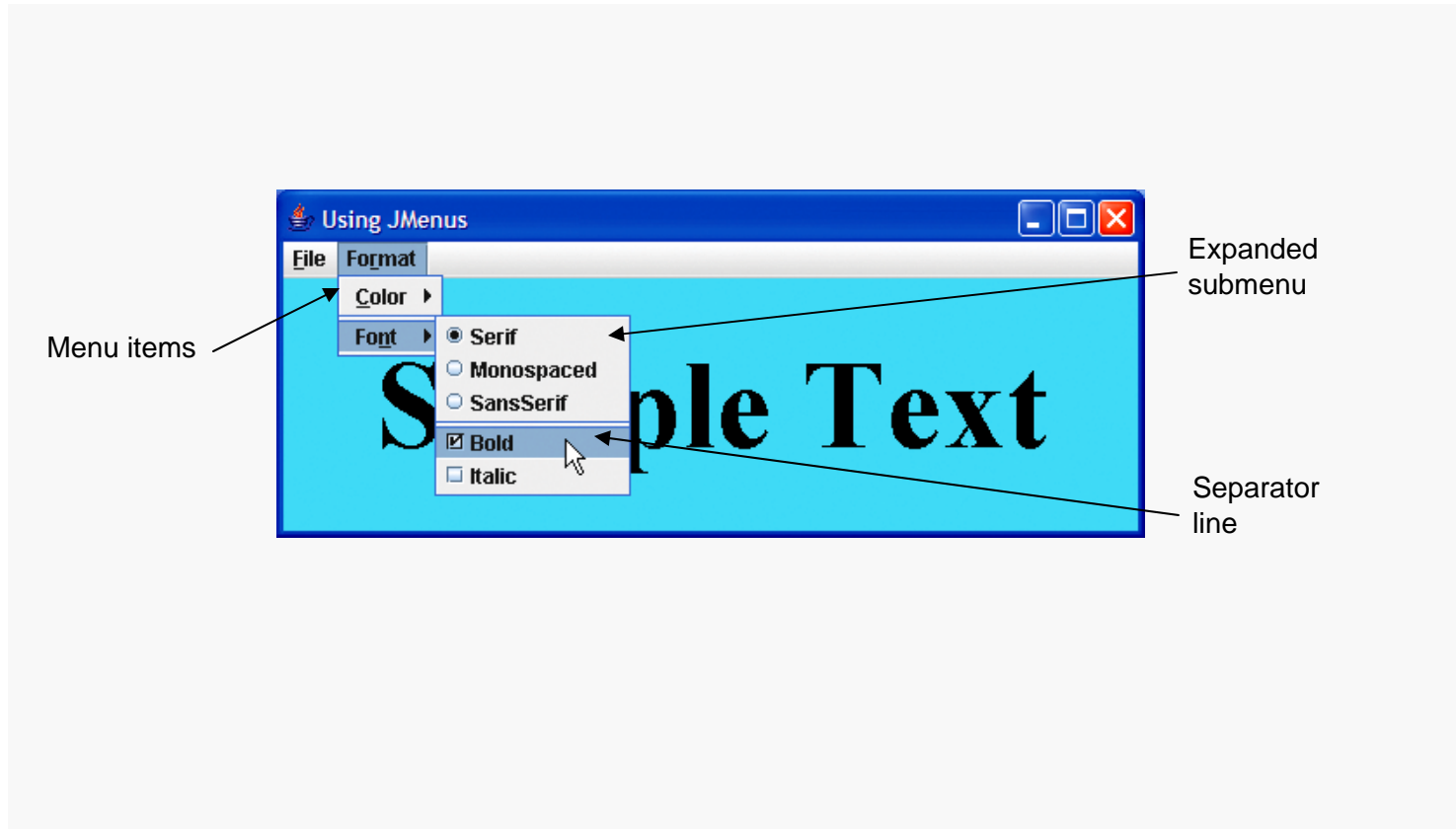
Look-and-Feel Observation 22.3

Mnemonics provide quick access to menu commands and button commands through the keyboard.



```
1 // Fig. 22.6: MenuTest.java
2 // Testing MenuFrame.
3 import javax.swing.JFrame;
4
5 public class MenuTest
6 {
7     public static void main( String args[] )
8     {
9         MenuFrame menuFrame = new MenuFrame(); // create MenuFrame
10        menuFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        menuFrame.setSize( 500, 200 ); // set frame size
12        menuFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class MenuTest
```





Look-and-Feel Observation 22.4

Different mnemonics should be used for each button or menu item. Normally, the first letter in the label on the menu item or button is used as the mnemonic. If several buttons or menu items start with the same letter, choose the next most prominent letter in the name (e.g., x is commonly chosen for a button or menu item called `Exit`).



22.4 Using Menus with Frames (Cont.)

- **showMessageDialog method**
 - **Specifying the parent window helps determine where the dialog box will be displayed**
 - **If specified as `null`, the dialog box appears in the center of the screen**
 - **Otherwise, it appears centered over the specified parent window**
 - **Modal dialog box**
 - **Does not allow any other window in the application to be accessed until the dialog box is dismissed**
 - **Dialog boxes are typically modal**



Common Programming Error 22.3

Forgetting to set the menu bar with JFrame method `setJMenuBar` results in the menu bar not being displayed on the JFrame.



Look-and-Feel Observation 22.5

Menus appear left to right in the order that they are added to a JMenuBar.



Look-and-Feel Observation 22.6

A submenu is created by adding a menu as a menu item in another menu. When the mouse is positioned over a submenu (or the submenu's mnemonic is pressed), the submenu expands to show its menu items.



Look-and-Feel Observation 22.7

Separators can be added to a menu to group menu items logically.



Look-and-Feel Observation 22.8

Any lightweight GUI component (i.e., a component that is a subclass of JComponent) can be added to a JMenu or to a JMenuBar.



22.5 JPopupMenu

- **Context-sensitive pop-up menus**
 - **Provide options that are specific to the component for which the pop-up trigger event was generated**
 - **On most systems, the pop-up trigger event occurs when the user presses and releases the right mouse button**
 - **Created with class JPopupMenu**



Look-and-Feel Observation 22.9

The pop-up trigger event is platform specific. On most platforms that use a mouse with multiple buttons, the pop-up trigger event occurs when the user clicks the right mouse button on a component that supports a pop-up menu.



```
1 // Fig. 22.7: PopupFrame.java
2 // Demonstrating JPopupMenu.
3 import java.awt.Color;
4 import java.awt.event.MouseAdapter;
5 import java.awt.event.MouseEvent;
6 import java.awt.event.ActionListener;
7 import java.awt.event.ActionEvent;
8 import javax.swing.JFrame;
9 import javax.swing.JRadioButtonMenuItem;
10 import javax.swing.JPopupMenu;
11 import javax.swing.ButtonGroup;
12
13 public class PopupFrame extends JFrame
14 {
15     private JRadioButtonMenuItem items[]; // holds items for colors
16     private final Color colorValues[] =
17         { Color.BLUE, Color.YELLOW, Color.RED }; // colors to be used
18     private JPopupMenu popupMenu; // allows user to select color
19
20     // no-argument constructor sets up GUI
21     public PopupFrame()
22     {
23         super( "Using JPopupMenu" );
24
25         ItemHandler handler = new ItemHandler(); // handler for menu items
26         String colors[] = { "Blue", "Yellow", "Red" }; // array of colors
27
```

An instance of class **ItemHandler** will process the item events from the menu items



```

28 ButtonGroup colorGroup = new ButtonGroup(); // manages color items
29 popupMenu = new JPopupMenu(); // create pop-up menu
30 items = new JRadioButtonMenuItem[ 3 ]; // items for selecting color
31
32 // construct menu item, add to popup menu, enable event handling
33 for ( int count = 0; count < items.length; count++ )
34 {
35     items[ count ] = new JRadioButtonMenuItem( colors[ count ] );
36     popupMenu.add( items[ count ] ); // add item to pop-up menu
37     colorGroup.add( items[ count ] ); // add item to button group
38     items[ count ].addActionListener( handler ); // add handler
39 } // end for
40
41 setBackground( Color.WHITE ); // set background to white
42
43 // declare a MouseListener for the window to display pop-up menu
44 addMouseListener(
45     new MouseAdapter() // anonymous inner class
46     {
47         // handle mouse press event
48         public void mousePressed( MouseEvent event )
49         {
50             checkForTriggerEvent( event ); // check for trigger
51         } // end method mousePressed
52     }
53

```

Create a **JPopupMenu** object

Create and add **JRadioButtonMenuItem**
and register **ActionListeners**

Register a **MouseListener** to handle the
mouse events of the application window



```
54 // handle mouse release event
55 public void mouseReleased( MouseEvent event )
56 {
57     checkForTriggerEvent( event ); // check for trigger
58 } // end method mouseReleased
59
60 // determine whether event should trigger popup menu
61 private void checkForTriggerEvent( MouseEvent event )
62 {
63     if ( event.isPopupTrigger() )
64         popupMenu.show(
65             event.getComponent(), event.getX(), event.getY() );
66 } // end method checkForTriggerEvent
67 } // end anonymous inner class
68 ); // end call to addMouseListener
69 } // end PopupFrame constructor
70
```

If the pop-up trigger event occurred, **JPopupMenu** method **show** displays the **JPopupMenu**

Origin component and coordinates arguments determine where the **JPopupMenu** will appear

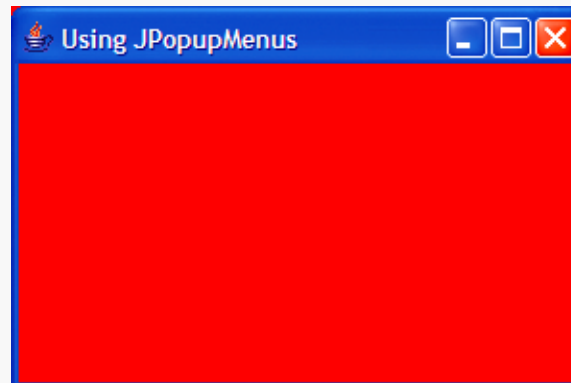
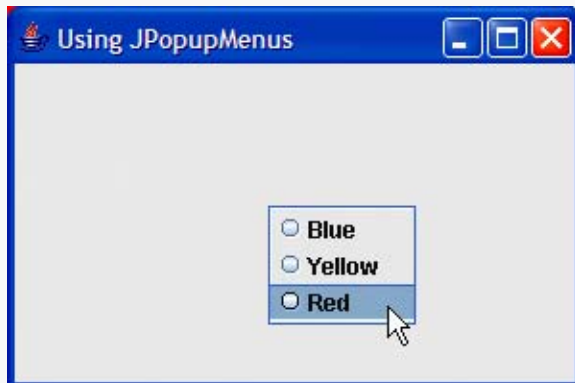


```
71 // private inner class to handle menu item events
72 private class ItemHandler implements ActionListener
73 {
74     // process menu item selections
75     public void actionPerformed((ActionEvent event) )
76     {
77         // determine which menu item was selected
78         for ( int i = 0; i < items.length; i++ )
79         {
80             if ( event.getSource() == items[ i ] )
81             {
82                 getContentPane().setBackground( colorValues[ i ] );
83                 return;
84             } // end if
85         } // end for
86     } // end method actionPerformed
87 } // end private inner class ItemHandler
88 } // end class PopupFrame
```

Determine which **JRadioButtonMenuItem**
the user selected and set the background color



```
1 // Fig. 22.8: PopupTest.java
2 // Testing PopupFrame.
3 import javax.swing.JFrame;
4
5 public class PopupTest
6 {
7     public static void main( String args[] )
8     {
9         PopupFrame popupFrame = new PopupFrame(); // create PopupFrame
10        popupFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        popupFrame.setSize( 300, 200 ); // set frame size
12        popupFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class PopupTest
```



Look-and-Feel Observation 22.10

Displaying a JPopupMenu for the pop-up trigger event of multiple GUI components requires registering mouse-event handlers for each of those GUI components.



22.6 Pluggable Look-and-Feel

- **Java applications' appearances**
 - **A program that uses Java's Abstract Window Toolkit GUI components takes on the look-and-feel of the platform**
 - **Allows users of the application on each platform to use GUI components with which they are already familiar**
 - **Also introduces interesting portability issues**
 - **Swing's lightweight GUI components provide uniform functionality**
 - **Define a uniform cross-platform look-and-feel (known as the metal look-and-feel)**
 - **Also can customize the look-and-feel to appear as a Microsoft Windows-style, Motif-style (UNIX) or Macintosh look-and-feel**



Portability Tip 22.1

GUI components look different on different platforms and may require different amounts of space to display. This could change their layout and alignments.



Portability Tip 22.2

GUI components on different platforms have different default functionality (e.g., some platforms allow a button with the focus to be “pressed” with the space bar, and some do not).



```
1 // Fig. 22.9: LookAndFeelFrame.java
2 // Changing the look and feel.
3 import java.awt.GridLayout;
4 import java.awt.BorderLayout;
5 import java.awt.event.ItemListener;
6 import java.awt.event.ItemEvent;
7 import javax.swing.JFrame;
8 import javax.swing.UIManager;
9 import javax.swing.JRadioButton;
10 import javax.swing.ButtonGroup;
11 import javax.swing.JButton;
12 import javax.swing.JLabel;
13 import javax.swing.JComboBox;
14 import javax.swing.JPanel;
15 import javax.swing.SwingConstants;
16 import javax.swing.SwingUtilities;
17
18 public class LookAndFeelFrame extends JFrame
19 {
20     // string names of look and feels
21     private final String strings[] = { "Metal", "Motif", "Windows" };
22     private UIManager.LookAndFeelInfo looks[]; // look and feels
23     private JRadioButton radio[]; // radiobuttons to select look and feel
24     private ButtonGroup group; // group for radiobuttons
25     private JButton button; // displays look of button
26     private JLabel label; // displays look of label
27     private JComboBox comboBox; // displays look of combo box
28
```



```
29 // set up GUI
30 public LookAndFeelFrame()
31 {
32     super( "Look and Feel Demo" );
33
34     JPanel northPanel = new JPanel(); // create north panel
35     northPanel.setLayout( new GridLayout( 3, 1, 0, 5 ) );
36
37     JLabel label = new JLabel( "This is a Metal look-and-feel",
38                               SwingConstants.CENTER ); // create label
39     northPanel.add( label ); // add label to panel
40
41     JButton button = new JButton( "JButton" ); // create button
42     northPanel.add( button ); // add button to panel
43
44     JComboBox comboBox = new JComboBox( strings ); // create combobox
45     northPanel.add( comboBox ); // add combobox to panel
46
47     // create array for radio buttons
48     radio = new JRadioButton[ strings.length ];
49
50     JPanel southPanel = new JPanel(); // create south panel
51     southPanel.setLayout( new GridLayout( 1, radio.length ) );
52
53     ButtonGroup group = new ButtonGroup(); // button group for look and feels
54     ItemHandler handler = new ItemHandler(); // look and feel handler
55
```



```

56  for ( int count = 0; count < radio.length; count++ )
57  {
58      radio[ count ] = new JRadioButton( strings[ count ] );
59      radio[ count ].addItemListener( handler ); // add handler
60      group.add( radio[ count ] ); // add radiobutton to group
61      southPanel.add( radio[ count ] ); // add radiobutton to panel
62  } // end for
63
64  add( northPanel, BorderLayout.NORTH );
65  add( southPanel, BorderLayout.SOUTH );
66
67  // get installed look-and-feel information
68  looks = UIManager.getInstalledLookAndFeels();
69  radio[ 0 ].setSelected( true ); // set default selection
70 } // end LookAndFeelFrame constructor
71
72 // use UIManager to change look-and-feel of GUI
73 private void changeTheLookAndFeel( int value )
74 {
75     try // change look and feel
76     {
77         // set look and feel for this application
78         UIManager.setLookAndFeel( looks[ value ].getClassName() );
79
80         // update components in this application
81         SwingUtilities.updateComponentTreeUI( this );
82     } // end try

```

Get the array of `UIManager.LookAndFeelInfo` objects that describe each look-and-feel available on your system

Invoke `static` method `setLookAndFeel` to change the look-and-feel

Invoke `static` method `updateComponentTreeUI` to change the look-and-feel of every GUI component attached to the application



```
83     catch ( Exception exception )
84     {
85         exception.printStackTrace();
86     } // end catch
87 } // end method changeTheLookAndFeel
88
89 // private inner class to handle radio button events
90 private class ItemHandler implements ItemListener
91 {
92     // process user's look-and-feel selection
93     public void itemStateChanged( ItemEvent event )
94     {
95         for ( int count = 0; count < radio.length; count++ )
96         {
97             if ( radio[ count ].isSelected() )
98             {
99                 label.setText( String.format( "This is a %s look-and-feel",
100                     strings[ count ] ) );
101                 comboBox.setSelectedIndex( count ); // set combobox index
102                 changeTheLookAndFeel( count ); // change look and feel
103             } // end if
104         } // end for
105     } // end method itemStateChanged
106 } // end private inner class ItemHandler
107 } // end class LookAndFeelFrame
```

Call utility method **changeTheLookAndFeel**

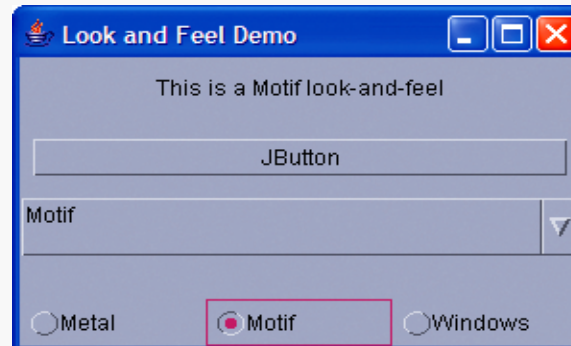
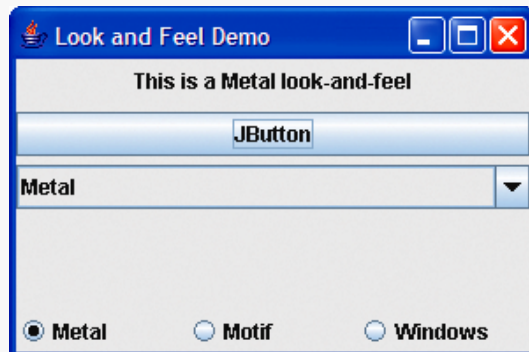


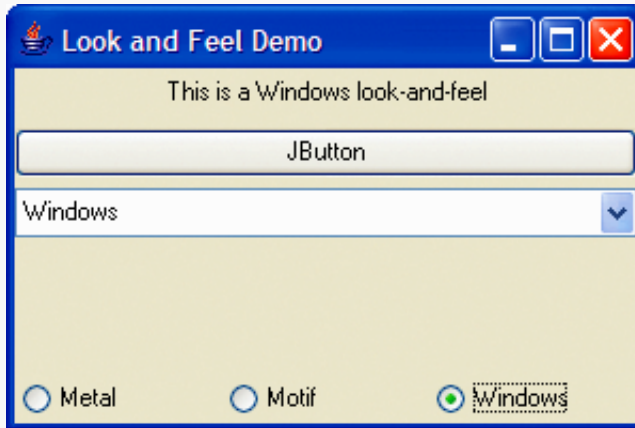
Performance Tip 22.1

Each look-and-feel is represented by a Java class. UIManager method `getInstalledLookAndFeels` does not load each class. Rather, it provides the names of the available look-and-feel classes so that a choice can be made (presumably once at program start-up). This reduces the overhead of having to load all the look-and-feel classes even if the program will not use some of them.



```
1 // Fig. 22.10: LookAndFeelDemo.java
2 // Changing the look and feel.
3 import javax.swing.JFrame;
4
5 public class LookAndFeelDemo
6 {
7     public static void main( String args[] )
8     {
9         LookAndFeelFrame lookAndFeelFrame = new LookAndFeelFrame();
10        lookAndFeelFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        lookAndFeelFrame.setSize( 300, 200 ); // set frame size
12        lookAndFeelFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class LookAndFeelDemo
```





22.7 JDesktopPane and JInternalFrame

- **Multiple-document interface**
 - A main window (called the parent window) contains other windows (called child windows)
 - Manages several open documents that are being processed in parallel
 - Implemented by Swing's JDesktopPane and JInternalFrame



```
1 // Fig. 22.11: DesktopFrame.java
2 // Demonstrating JDesktopPane.
3 import java.awt.BorderLayout;
4 import java.awt.Dimension;
5 import java.awt.Graphics;
6 import java.awt.event.ActionListener;
7 import java.awt.event.ActionEvent;
8 import java.util.Random;
9 import javax.swing.JFrame;
10 import javax.swing.JDesktopPane;
11 import javax.swing.JMenuBar;
12 import javax.swing.JMenu;
13 import javax.swing.JMenuItem;
14 import javax.swing.JInternalFrame;
15 import javax.swing.JPanel;
16 import javax.swing.ImageIcon;
17
18 public class DesktopFrame extends JFrame
19 {
20     private JDesktopPane theDesktop;
21
22     // set up GUI
23     public DesktopFrame()
24     {
25         super( "Using a JDesktopPane" );
26
27         JMenuBar bar = new JMenuBar(); // create menu bar
28         JMenu addMenu = new JMenu( "Add" ); // create Add menu
29         JMenuItem newFrame = new JMenuItem( "Internal Frame" );
30
```

Create a **JMenuBar**, a
JMenu and a **JMenuItem**



```

31 addMenu.add( newFrame ); // add new frame item to Add menu
32 bar.add( addMenu ); // add Add menu to menu bar
33 setJMenuBar( bar ); // set menu bar for this appl

```

Add the **JMenuItem** to the **JMenu** and the **JMenu** to the **JMenuBar** and set the **JMenuBar** for the application window

```

34
35 theDesktop = new JDesktopPane(); // create desktop
36 add( theDesktop ); // add desktop pane to frame

```

The **JDesktopPane** will be used to manage the **JInternalFrame** child windows

```

37
38 // set up listener for newFrame menu item
39 newFrame.addActionListener(

```

```

40
41     new ActionListener() // anonymous inner class
42     {
43         // display new internal window
44         public void actionPerformed( ActionEvent event )
45         {

```

Create a **JInternalFrame** object

```

46         // create internal frame
47         JInternalFrame frame = new JInternalFrame(
48             "Internal Frame", true, true, true, true );

```

```

49
50         MyJPanel panel = new MyJPanel(); // create new panel
51         frame.add( panel, BorderLayout.CENTER ); // add panel
52         frame.pack(); // set internal frame to size of contents

```

Constructor arguments specify title bar string and whether or not the user can resize, close, maximize and minimize the internal frame

Set the size of the child window



```

54         theDesktop.add( frame ); // attach internal frame
55         frame.setVisible( true ); // show internal frame
56     } // end method actionPerformed
57 } // end anonymous inner class
58 ); // end call to addActionListener
59 } // end DesktopFrame constructor
60 } // end class DesktopFrame
61
62 // class to display an ImageIcon on a panel
63 class MyJPanel extends JPanel
64 {
65     private static Random generator = new Random();
66     private ImageIcon picture; // image to be displayed
67     private String[] images = { "yellowflowers.png", "purpleflowers.png",
68         "redflowers.png", "redflowers2.png", "lavenderflowers.png" };
69
70     // load image
71     public MyJPanel()
72     {
73         int randomNumber = generator.nextInt( 5 );
74         picture = new ImageIcon( images[ randomNumber ] ); // set icon
75     } // end MyJPanel constructor
76

```

Add the **JInternalFrame** to **theDesktop**
and display the **JInternalFrame**



```
77 // display ImageIcon on panel
78 public void paintComponent( Graphics g )
79 {
80     super.paintComponent( g );
81     picture.paintIcon( this, g, 0, 0 ); // display icon
82 } // end method paintComponent
83
84 // return image dimensions
85 public Dimension getPreferredSize() ←
86 {
87     return new Dimension( picture.getIconwidth(),
88         picture.getIconHeight() );
89 } // end method getPreferredSize
90 } // end class MyJPanel
```

Specify the panel's preferred size
for use by the **pack** method



```
1 // Fig. 22.12: DesktopTest.java
2 // Demonstrating JDesktopPane.
3 import javax.swing.JFrame;
4
5 public class DesktopTest
6 {
7     public static void main( String args[] )
8     {
9         DesktopFrame desktopFrame = new DesktopFrame();
10        desktopFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        desktopFrame.setSize( 600, 480 ); // set frame size
12        desktopFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class DesktopTest
```

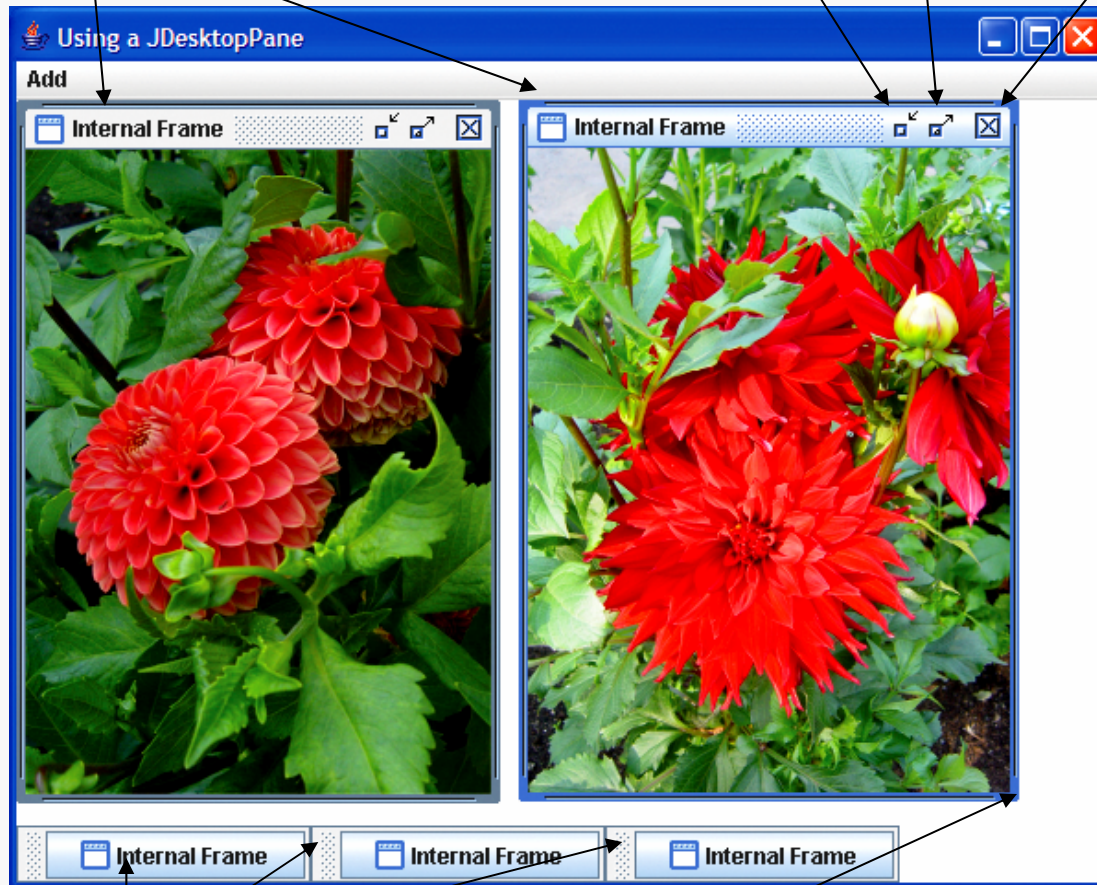


Internal Frames

Minimize

Maximize

Close

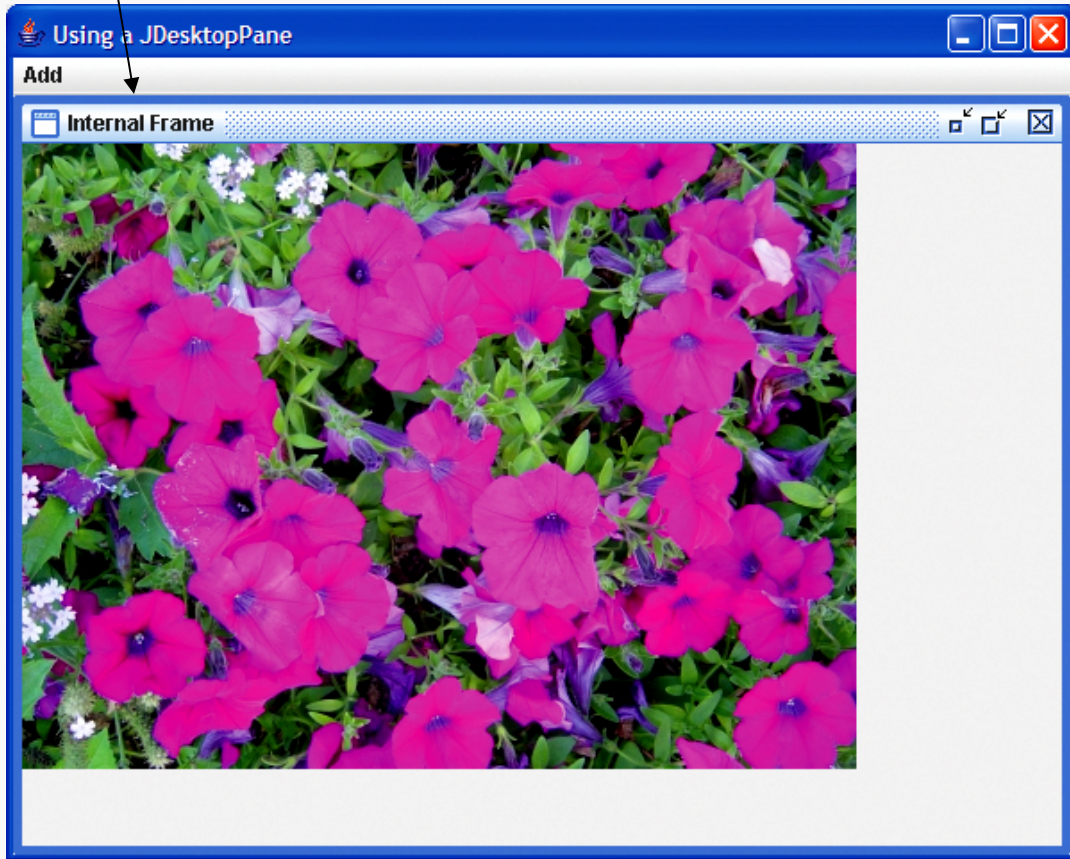


Minimized internal frames

Position the mouse over any corner of a child window to resize the window (if resizing is allowed).



Maximized internal frame



22.8 JTabbedPane

- **JTabbedPane**

- **Arranges GUI components into layers in which only one layer is visible at a time**
 - **When the user clicks a tab, the appropriate layer is displayed**
 - **The tabs can be positioned at top (default), left, right or bottom**
 - **Any component can be placed on a tab**
 - **If the tabs do not fit on one line, they will wrap to form additional lines of tabs**



```

1 // Fig. 22.13: JTabbedPaneFrame.java
2 // Demonstrating JTabbedPane.
3 import java.awt.BorderLayout;
4 import java.awt.Color;
5 import javax.swing.JFrame;
6 import javax.swing.JTabbedPane;
7 import javax.swing.JLabel;
8 import javax.swing.JPanel;
9 import javax.swing.JButton;
10 import javax.swing.SwingConstants;
11
12 public class JTabbedPaneFrame extends JFrame
13 {
14     // set up GUI
15     public JTabbedPaneFrame()
16     {
17         super( "JTabbedPane Demo " );
18
19         JTabbedPane tabbedPane = new JTabbedPane(); // create JTabbedPane
20
21         // set up pane1 and add it to JTabbedPane
22         JLabel label1 = new JLabel( "pane1 one", SwingConstants.CENTER );
23         JPanel pane1 = new JPanel(); // create first pane1
24         pane1.add( label1 ); // add label to pane1
25         tabbedPane.addTab( "Tab One", null, pane1, "First Panel" );
26

```

Create an empty **JTabbedPane** with default settings

Call **JTabbedPane** method **addTab** with arguments that specify the tab's string title, an **Icon** reference to display on the tab, the **COMPONENT** to display when the user clicks on the tab and the tab's tooltip string



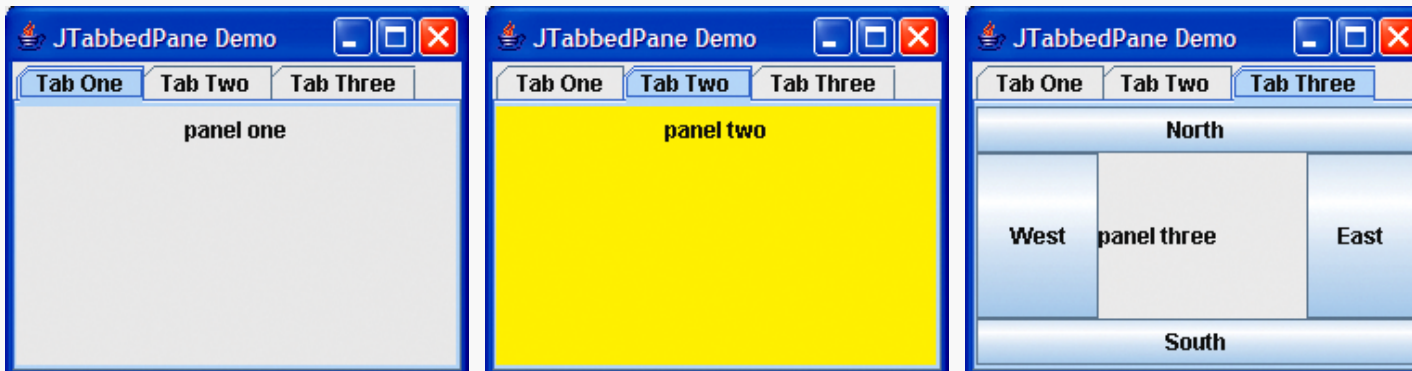
```
27 // set up panel2 and add it to JTabbedPane
28 JLabel label2 = new JLabel( "panel two", SwingConstants.CENTER );
29 JPanel panel2 = new JPanel(); // create second panel
30 panel2.setBackground( Color.YELLOW ); // set background to yellow
31 panel2.add( label2 ); // add label to panel
32 tabbedPane.addTab( "Tab Two", null, panel2, "Second Panel" );
33
34 // set up panel3 and add it to JTabbedPane
35 JLabel label3 = new JLabel( "panel three" );
36 JPanel panel3 = new JPanel(); // create third panel
37 panel3.setLayout( new BorderLayout() ); // use BorderLayout
38 panel3.add( new JButton( "North" ), BorderLayout.NORTH );
39 panel3.add( new JButton( "West" ), BorderLayout.WEST );
40 panel3.add( new JButton( "East" ), BorderLayout.EAST );
41 panel3.add( new JButton( "South" ), BorderLayout.SOUTH );
42 panel3.add( label3, BorderLayout.CENTER );
43 tabbedPane.addTab( "Tab Three", null, panel3, "Third Panel" );
44
45 add( tabbedPane ); // add JTabbedPane to frame
46 } // end JTabbedPaneFrame constructor
47 } // end class JTabbedPaneFrame
```

Add panel2 to tabbedPane

Add panel3 to tabbedPane



```
1 // Fig. 22.14: JTabbedPaneDemo.java
2 // Demonstrating JTabbedPane.
3 import javax.swing.JFrame;
4
5 public class JTabbedPaneDemo
6 {
7     public static void main( String args[] )
8     {
9         JTabbedPaneFrame tabbedPaneFrame = new JTabbedPaneFrame();
10        tabbedPaneFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        tabbedPaneFrame.setSize( 250, 200 ); // set frame size
12        tabbedPaneFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class JTabbedPaneDemo
```



22.9 Layout Managers: BorderLayout and GridBagLayout

- **BoxLayout Layout Manager**
 - Arranges GUI components horizontally along a container's *x*-axis or vertically along its *y*-axis



Layout Manager	Description
BoxLayout	A layout manager that allows GUI components to be arranged left-to-right or top-to-bottom in a container. Class BOX declares a container with BOXLayout as its default layout manager and provides static methods to create a BOX with a horizontal or vertical BoxLayout.
GridBagLayout	A layout manager similar to GridLayout, but unlike it in that components can vary in size and can be added in any order.

Fig. 22.15 | Additional layout managers.



```
1 // Fig. 22.16: BorderLayoutFrame.java
2 // Demonstrating BorderLayout.
3 import java.awt.Dimension;
4 import javax.swing.JFrame;
5 import javax.swing.Box;
6 import javax.swing.JButton;
7 import javax.swing.BoxLayout;
8 import javax.swing.JPanel;
9 import javax.swing.JTabbedPane;
10
11 public class BorderLayoutFrame extends JFrame
12 {
13     // set up GUI
14     public BorderLayoutFrame()
15     {
16         super( "Demonstrating BorderLayout" );
17
18         // create Box containers with BorderLayout
19         Box horizontal1 = Box.createHorizontalBox();
20         Box vertical1 = Box.createVerticalBox();
21         Box horizontal2 = Box.createHorizontalBox();
22         Box vertical2 = Box.createVerticalBox();
23
24         final int SIZE = 3; // number of buttons on each Box
25
26         // add buttons to Box horizontal1
27         for ( int count = 0; count < SIZE; count++ )
28             horizontal1.add( new JButton( "Button " + count ) );
29
```

Create **Box** containers with **static Box** methods **createHorizontalBox** and **createVerticalBox**

Add three **JButtons** to **horizontal1**



```
30 // create strut and add buttons to Box vertical1
31 for ( int count = 0; count < SIZE; count++ )
32 {
33     vertical1.add( Box.createVerticalStrut( 25 ) );
34     vertical1.add( new JButton( "Button " + count ) );
35 } // end for
36
37 // create horizontal glue and add buttons to Box horizontal2
38 for ( int count = 0; count < SIZE; count++ )
39 {
40     horizontal2.add( Box.createHorizontalGlue() );
41     horizontal2.add( new JButton( "Button " + count ) );
42 } // end for
43
44 // create rigid area and add buttons to Box vertical2
45 for ( int count = 0; count < SIZE; count++ )
46 {
47     vertical2.add( Box.createRigidArea( new Dimension( 12, 8 ) ) );
48     vertical2.add( new JButton( "Button " + count ) );
49 } // end for
50
51 // create vertical glue and add buttons to panel
52 JPanel panel = new JPanel();
53 panel.setLayout( new BorderLayout( panel, BorderLayout.Y_AXIS ) );
54
```

Add three vertical struts and three JButtons to **vertical1**

Add horizontal glue and three JButtons to **horizontal2**

Add three rigid areas and three JButtons to **vertical2**

Use **Container** method **setLayout** to set **panel**'s layout to a vertical **BoxLayout**



```
55 for ( int count = 0; count < SIZE; count++ )
56 {
57     panel.add( Box.createGlue() );
58     panel.add( new JButton( "Button " + count ) );
59 } // end for
60
61 // create a JTabbedPane
62 JTabbedPane tabs = new JTabbedPane(
63     JTabbedPane.TOP, JTabbedPane.SCROLL_TAB_LAYOUT );
64
65 // place each container on tabbed pane
66 tabs.addTab( "Horizontal Box", horizontal1 );
67 tabs.addTab( "Vertical Box with Struts", vertical1 );
68 tabs.addTab( "Horizontal Box with Glue", horizontal2 );
69 tabs.addTab( "Vertical Box with Rigid Areas", vertical2 );
70 tabs.addTab( "Vertical Box with Glue", panel );
71
72 add( tabs ); // place tabbed pane on frame
73 } // end BorderLayoutFrame constructor
74 } // end class BorderLayoutFrame
```

Add glue and three **JButtons** to **panel**

Create a **JTabbedPane** where the tabs should scroll if there are too many tabs to fit on one line



22.9 Layout Managers: BorderLayout and GridBagLayout (Cont.)

- **Vertical struts**
 - **Invisible GUI component that has a fixed pixel height**
 - **Used to guarantee a fixed amount of space between GUI components**
 - **static Box method createVerticalStrut**
 - **int argument determines the height of the strut in pixels**
 - **Box also declares method createHorizontalStrut**



22.9 Layout Managers: BorderLayout and GridBagLayout (Cont.)

- **Horizontal glue**

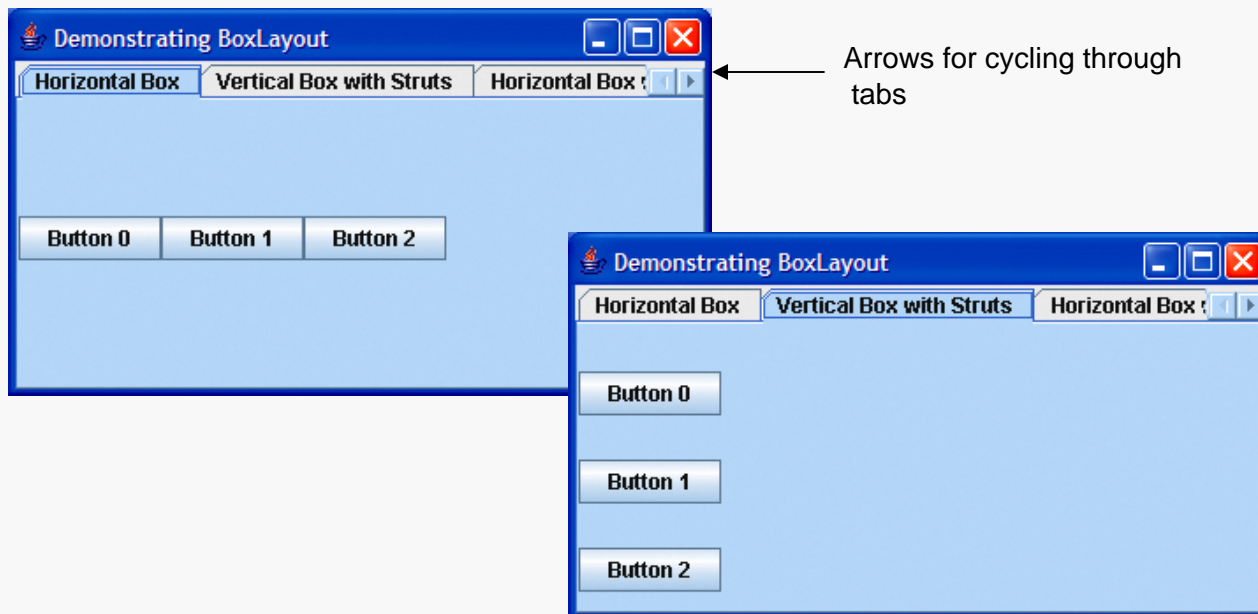
- **An invisible GUI component that occupies additional space between fixed-size GUI components**
 - **When the container is resized, components separated by glue remain the same size, but the glue stretches or contracts to occupy the space between them**
- **static Box methods createHorizontalGlue and createVerticalGlue**

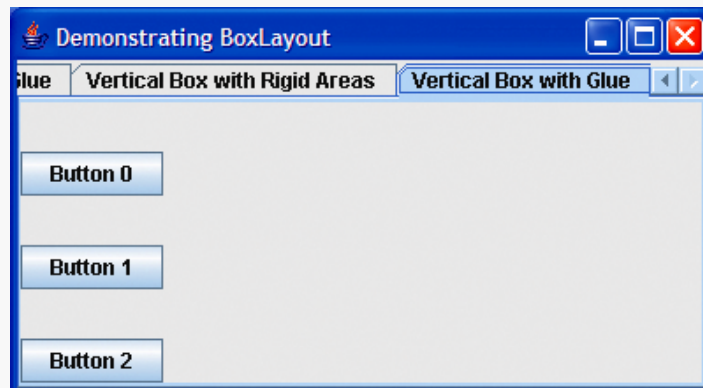
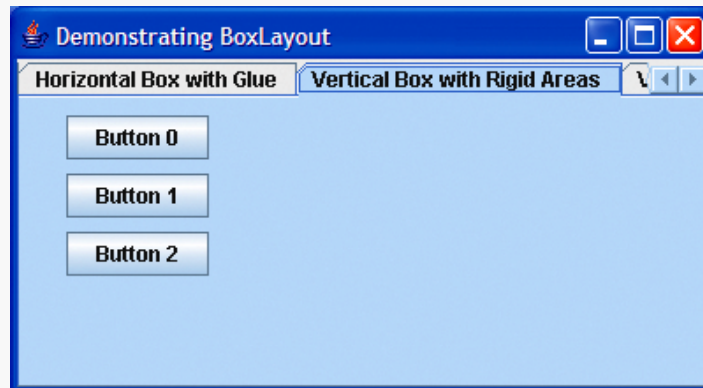
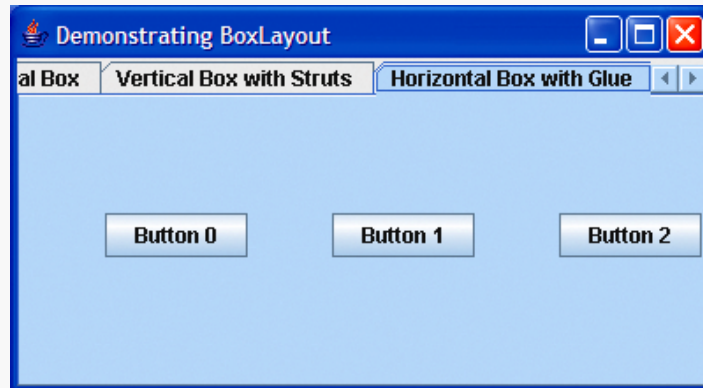


```

1 // Fig. 22.17: BorderLayoutDemo.java
2 // Demonstrating BorderLayout.
3 import javax.swing.JFrame;
4
5 public class BorderLayoutDemo
6 {
7     public static void main( String args[] )
8     {
9         BorderLayoutFrame boxLayoutFrame = new BorderLayoutFrame();
10        boxLayoutFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        boxLayoutFrame.setSize( 400, 220 ); // set frame size
12        boxLayoutFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class BorderLayoutDemo

```





22.9 Layout Managers: BorderLayout and GridBagLayout (Cont.)

- **Rigid areas**
 - **An invisible GUI component that always has a fixed pixel width and height**
 - **Dimension object argument to static Box method `createRigidArea` specifies the area's width and height**



22.9 Layout Managers: BorderLayout and GridBagLayout (Cont.)

- **GridBagLayout Layout Manager**
 - **Similar to GridLayout in that it arranges components in a grid, but more flexible**
 - **The components can vary in size and can be added in any order**
 - **Determining the appearance of the GUI**
 - **Draw the GUI on paper**
 - **Draw a grid over it, dividing the components into rows and columns**
 - **The initial row and column numbers should be 0**
 - **Used by the GridBagLayout layout manager to properly place the components in the grid**



22.9 Layout Managers: BoxLayout and GridBagLayout (Cont.)

- **GridBagConstraints** object
 - Describes how a component is placed in a **GridBagLayout**
 - **anchor** specifies the relative position of the component in an area that it does not fill
 - Constants: **NORTH, NORTHEAST, EAST, SOUTHEAST, SOUTH, SOUTHWEST, WEST, NORTHWEST** and **CENTER** (the default)
 - **fill** defines how the component grows if the area in which it can be displayed is larger than the component
 - Constants: **NONE** (the default), **VERTICAL, HORIZONTAL** and **BOTH**



22.9 Layout Managers: BorderLayout and GridBagLayout (Cont.)

- **gridx** and **gridy** specify where the upper-left corner of the component is placed in the grid
- **gridwidth** and **gridheight** specify the number of columns and rows a component occupies
- **weightx** and **weighty** specify how to distribute extra horizontal and vertical space to grid slots in a **GridBagLayout** when the container is resized
 - A zero value indicates that the grid slot does not grow in that dimension on its own
 - However, if the component spans a column/row containing a component with nonzero weight value, it will grow in the same proportion as the other components in that column/row
 - Use positive nonzero weight values to prevent “huddling”



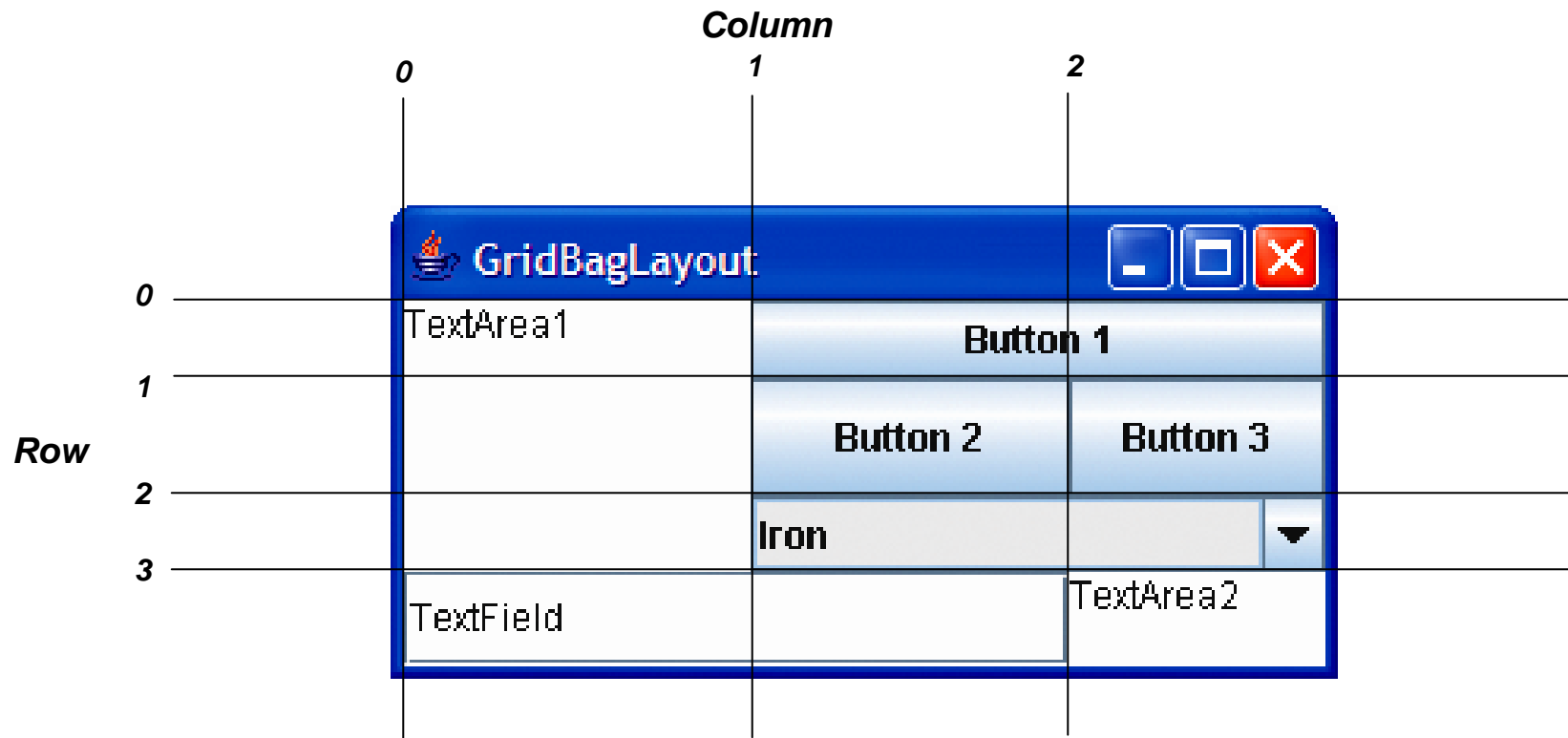


Fig. 22.18 | Designing a GUI that will use GridBagLayout.



GridBagConstraints field	Description
anchor	Specifies the relative position (NORTH, NORTHEAST, EAST, SOUTHEAST, SOUTH, SOUTHWEST, WEST, NORTHWEST, CENTER) of the component in an area that it does not fill.
fill	Resizes the component in specified direction (NONE, HORIZONTAL, VERTICAL, BOTH) when the display area is larger than the component.
gridx	The column in which the component will be placed.
gridy	The row in which the component will be placed.
gridwidth	The number of columns the component occupies.
gridheight	The number of rows the component occupies.
weightx	The amount of extra space to allocate horizontally. The grid slot can become wider when extra space is available.
weighty	The amount of extra space to allocate vertically. The grid slot can become taller when extra space is available.

Fig. 22.19 | GridBagConstraints fields.



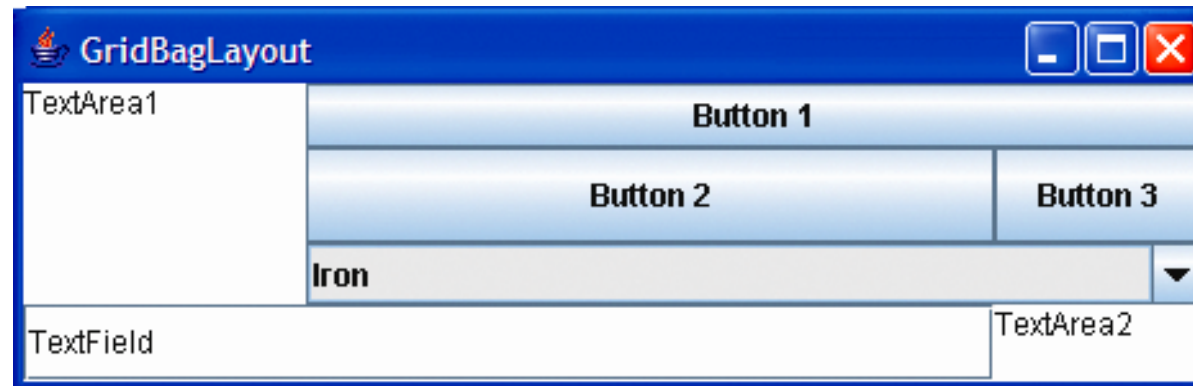
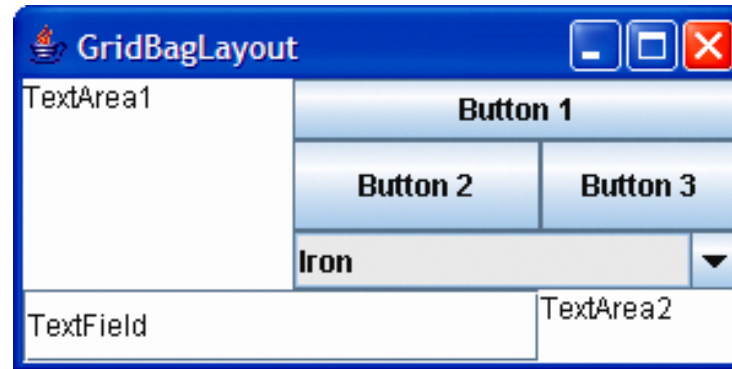


Fig. 22.20 | GridBagLayout with the weights set to zero.



```
1 // Fig. 22.21: GridBagFrame.java
2 // Demonstrating GridBagLayout.
3 import java.awt.GridBagLayout;
4 import java.awt.GridBagConstraints;
5 import java.awt.Component;
6 import javax.swing.JFrame;
7 import javax.swing.JTextArea;
8 import javax.swing.JTextField;
9 import javax.swing.JButton;
10 import javax.swing.JComboBox;
11
12 public class GridBagFrame extends JFrame
13 {
14     private GridBagLayout layout; // layout of this frame
15     private GridBagConstraints constraints; // constraints of this layout
16
17     // set up GUI
18     public GridBagFrame()
19     {
20         super( "GridBagLayout" );
21         layout = new GridBagLayout();
22         setLayout( layout ); // set frame layout
23         constraints = new GridBagConstraints(); // instantiate constraints
24
25         // create GUI components
26         JTextArea textArea1 = new JTextArea( "TextArea1", 5, 10 );
27         JTextArea textArea2 = new JTextArea( "TextArea2", 2, 2 );
28
```

Create a **GridBagLayout** object



Create a
GridBagConstraints
object



```

29 String names[] = { "Iron", "Steel", "Brass" };
30 JComboBox comboBox = new JComboBox( names );
31
32 JTextField textField = new JTextField( "TextField" );
33 JButton button1 = new JButton( "Button 1" );
34 JButton button2 = new JButton( "Button 2" );
35 JButton button3 = new JButton( "Button 3" );
36
37 // weightx and weighty for textArea1 are both 0: the default
38 // anchor for all components is CENTER: the default
39 constraints.fill = GridBagConstraints.BOTH;
40 addComponent( textArea1, 0, 0, 1, 3 );
41
42 // weightx and weighty for button1 are both 0: the default
43 constraints.fill = GridBagConstraints.HORIZONTAL;
44 addComponent( button1, 0, 1, 2, 1 );
45
46 // weightx and weighty for comboBox are both 0: the default
47 // fill is HORIZONTAL
48 addComponent( comboBox, 2, 1, 2, 1 );
49
50 // button2
51 constraints.weightx = 1000; // can grow wider
52 constraints.weighty = 1; // can grow taller
53 constraints.fill = GridBagConstraints.BOTH;
54 addComponent( button2, 1, 1, 1, 1 );
55

```

Cause the **JTextArea** to always fill its entire allocated area

Call utility method **addComponent** with the **JTextArea** object, row, column and numbers of columns and rows to span as arguments

When the window is resized, **button2** will grow



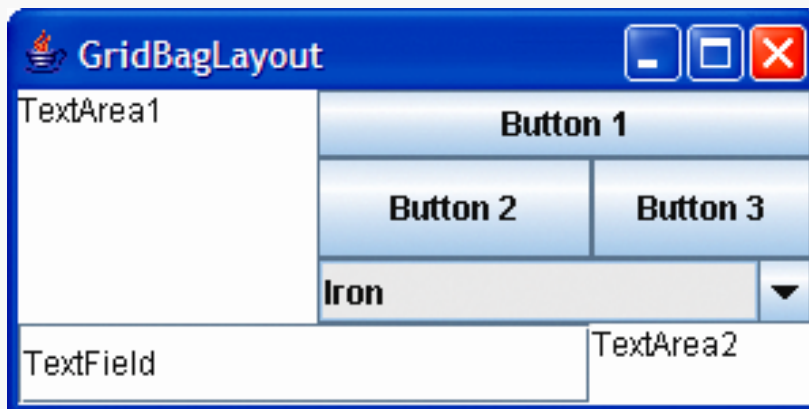
```
56 // fill is BOTH for button3
57 constraints.weightx = 0;
58 constraints.weighty = 0;
59 addComponent( button3, 1, 2, 1, 1 );
60
61 // weightx and weighty for textField are both 0, fill is BOTH
62 addComponent( textField, 3, 0, 2, 1 );
63
64 // weightx and weighty for textArea2 are both 0, fill is BOTH
65 addComponent( textArea2, 3, 2, 1, 1 );
66 } // end GridBagFrame constructor
67
68 // method to set constraints on
69 private void addComponent( Component component,
70     int row, int column, int width, int height )
71 {
72     constraints.gridx = column; // set gridx
73     constraints.gridy = row; // set gridy
74     constraints.gridwidth = width; // set gridwidth
75     constraints.gridheight = height; // set gridheight
76     layout.setConstraints( component, constraints ); // set constraints
77     add( component ); // add component
78 } // end method addComponent
79 } // end class GridBagFrame
```

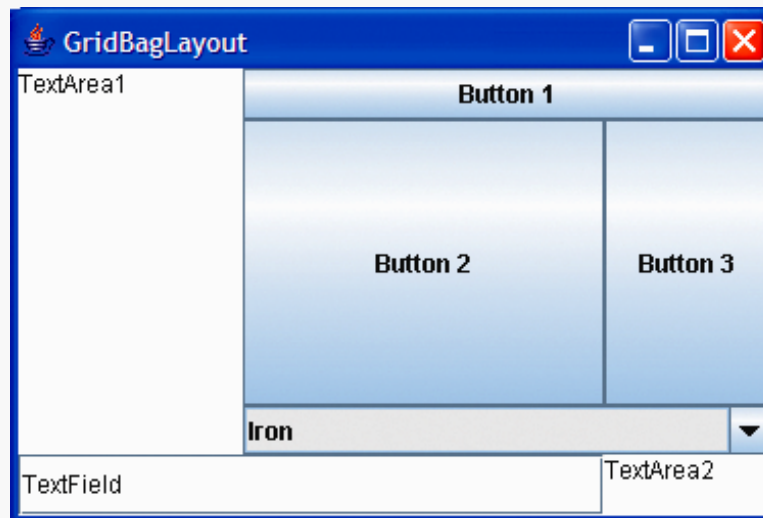
button3 will still grow because of the weight values of button2

Set constraints and add component



```
1 // Fig. 22.22: GridBagDemo.java
2 // Demonstrating GridBagLayout.
3 import javax.swing.JFrame;
4
5 public class GridBagDemo
6 {
7     public static void main( String args[] )
8     {
9         GridBagFrame gridBagFrame = new GridBagFrame();
10        gridBagFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        gridBagFrame.setSize( 300, 150 ); // set frame size
12        gridBagFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class GridBagDemo
```





22.9 Layout Managers: BorderLayout and GridBagLayout (Cont.)

- **GridBagConstraints constants**

- **RELATIVE**

- Specifies that the next-to-last component in a particular row should be placed to the right of the previous component in the row

- **REMAINDER**

- Specifies that a component is the last component in a row

- Components that are not the second-to-last or last component on a row must specify values for **gridwidth** and **gridheight**



```
1 // Fig. 22.23: GridBagFrame2.java
2 // Demonstrating GridBagLayout constants.
3 import java.awt.GridBagLayout;
4 import java.awt.GridBagConstraints;
5 import java.awt.Component;
6 import javax.swing.JFrame;
7 import javax.swing.JComboBox;
8 import javax.swing.JTextField;
9 import javax.swing.JList;
10 import javax.swing.JButton;
11
12 public class GridBagFrame2 extends JFrame
13 {
14     private GridBagLayout layout; // layout of this frame
15     private GridBagConstraints constraints; // constraints of this layout
16
17     // set up GUI
18     public GridBagFrame2()
19     {
20         super( "GridBagLayout" );
21         layout = new GridBagLayout();
22         setLayout( layout ); // set frame layout
23         constraints = new GridBagConstraints(); // instantiate constraints
24
25         // create GUI components
26         String metals[] = { "Copper", "Aluminum", "Silver" };
27         JComboBox comboBox = new JComboBox( metals );
28
29         JTextField textField = new JTextField( "TextField" );
30
```

Create a **GridBagLayout** object



```

31 String fonts[] = { "Serif", "Monospaced" };T
32 JList list = new JList( fonts );
33
34 String names[] = { "zero", "one", "two", "three", "four" };
35 JButton buttons[] = new JButton[ names.length ];
36
37 for ( int count = 0; count < buttons.length; count++ )
38     buttons[ count ] = new JButton( names[ count ] );
39
40 // define GUI component constraints for textField
41 constraints.weightx = 1;
42 constraints.weighty = 1;
43 constraints.fill = GridBagConstraints.BOTH;
44 constraints.gridwidth = GridBagConstraints.REMAINDER;
45 addComponent( textField );
46
47 // buttons[0] -- weightx and weighty are 1: fill is BOTH
48 constraints.gridwidth = 1;
49 addComponent( buttons[ 0 ] );
50
51 // buttons[1] -- weightx and weighty are 1: fill is BOTH
52 constraints.gridwidth = GridBagConstraints.RELATIVE;
53 addComponent( buttons[ 1 ] );
54
55 // buttons[2] -- weightx and weighty are 1: fill is BOTH
56 constraints.gridwidth = GridBagConstraints.REMAINDER;
57 addComponent( buttons[ 2 ] );
58

```

Specify that the **JTextField** is the last component on the line

Specify that the **JButton** is to be placed relative to the previous component

This **JButton** is the last component on the line



```
59 // comboBox -- weightx is 1: fill is BOTH
60 constraints.weighty = 0;
61 constraints.gridwidth = GridBagConstraints.REMAINDER;
62 addComponent( comboBox );
63
64 // buttons[3] -- weightx is 1: fill is BOTH
65 constraints.weighty = 1;
66 constraints.gridwidth = GridBagConstraints.REMAINDER;
67 addComponent( buttons[ 3 ] );
68
69 // buttons[4] -- weightx and weighty are 1: fill is BOTH
70 constraints.gridwidth = GridBagConstraints.RELATIVE;
71 addComponent( buttons[ 4 ] );
72
73 // list -- weightx and weighty are 1: fill is BOTH
74 constraints.gridwidth = GridBagConstraints.REMAINDER;
75 addComponent( list );
76 } // end GridBagFrame2 constructor
77
78 // add a component to the container
79 private void addComponent( Component component )
80 {
81     layout.setConstraints( component, constraints );
82     add( component ); // add component
83 } // end method addComponent
84 } // end class GridBagFrame2
```

The **JComboBox** is the only component on the line

This **JButton** is the only component on the line

This **JButton** is the next-to-last component on the line



```
1 // Fig. 22.24: GridBagDemo2.java
2 // Demonstrating GridBagLayout constants.
3 import javax.swing.JFrame;
4
5 public class GridBagDemo2
6 {
7     public static void main( String args[] )
8     {
9         GridBagFrame2 gridBagFrame = new GridBagFrame2();
10        gridBagFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        gridBagFrame.setSize( 300, 200 ); // set frame size
12        gridBagFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class GridBagDemo2
```

