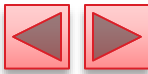
# Chapter 22
# GUI Components: Part 2

Java How to Program, 10/e

© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.

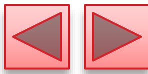


## OBJECTIVES

In this chapter you'll:

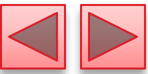
- Create and manipulate sliders, menus, pop-up menus and windows.
- Programatically change the look-and-feel of a GUI, using Swing's pluggable look-and-feel.
- Create a multiple-document interface with `JDesktopPane` and `JInternalFrame`.
- Use additional layout managers `BoxLayout` and `GridBagLayout`.

© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



22.1 Introduction
22.2 `JSlider`
22.3 Understanding Windows in Java
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 `BoxLayout` Layout Manager
22.10 `GridBagLayout` Layout Manager
22.11 Wrap-Up

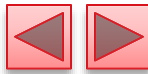
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.1 Introduction

- In this chapter, we cover
  - Additional components and layout managers and lay the groundwork for building more complex GUIs.
  - Sliders for selecting from a range of integer values, then discuss additional details of windows.
  - Swing's pluggable look-and-feel (PLAF).
  - Multiple-document interface (MDI)—a main window (often called the parent window) containing other windows (often called child windows) to manage several open documents in parallel.

© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.

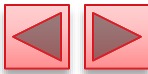


# 22.1 Introduction (Cont.)

***Java SE 8: Implementing Event Listeners with Lambdas***

- Throughout this chapter, we use anonymous inner classes and nested classes to implement event handlers so that the examples can compile and execute with both Java SE 7 and Java SE 8.
- In many of the examples, you could implement the functional event-listener interfaces with Java SE 8 lambdas (as demonstrated in Section 17.9).

© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.2 `JSlider`

- JSliders enable a user to select from a range of integer values.
- Figure 22.1 shows a horizontal `JSlider` with tick marks and the thumb that allows a user to select a value.
- Can be customized to display *major tick marks*, *minor-tick marks* and labels for the tick marks.
- Also support snap-to ticks, which cause the *thumb*, when positioned between two tick marks, to snap to the closest one.

© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.

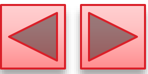
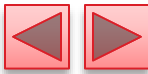


**Fig. 22.1** | `JSlider` component with horizontal orientation.

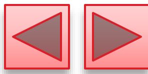
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.2 `JSlider` (cont.)

- The down arrow key and up arrow key also cause the thumb of the `JSlider` to decrease or increase by 1 tick, respectively.
- The *PgDn* (page down) key and *PgUp* (page up) key cause the thumb of the `JSlider` to decrease or increase by block increments of one-tenth of the range of values, respectively.
- The *Home* key moves the thumb to the minimum value of the `JSlider`, and the *End* key moves the thumb to the maximum value of the `JSlider`.

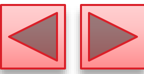
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.2 JSlider (cont.)

- JSliders have either a horizontal orientation or a vertical orientation.
  - For a horizontal JSlider, the minimum value is at the left end of the JSlider and the maximum is at the right end.
  - For a vertical JSlider, the minimum value is at the bottom and the maximum is at the top.
- The minimum and maximum value positions on a JSlider can be reversed by invoking JSlider method setInverted with boolean argument true.

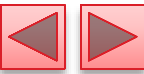
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
 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
10
11     // draw an oval of the specified diameter
12     @Override
13     public void paintComponent(Graphics g)
14     {
15        super.paintComponent(g);
16        g.fillOval(10, 10, diameter, diameter);
17     }
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     }
```

**Fig. 22.2** | JPanel subclass for drawing circles of a specified diameter. (Part 1 of 2.)

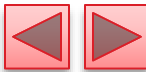
© Copyright 1992–2015 by Pearson Education, Inc. All Rights Reserved.



```
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 final JSlider diameterJSlider; // slider to select diameter
14    private final 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);
23
```

**Fig. 22.3** | `JSlider` value used to determine the diameter of a circle. (Part 1 of 2.)

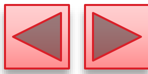
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
24        // set up JSlider to control diameter value
25        diameterJSlider =
26           new JSlider(SwingConstants.HORIZONTAL, 0, 200, 10);
27        diameterJSlider.setMajorTickSpacing(10); // create tick every 10
28        diameterJSlider.setPaintTicks(true); // paint ticks on slider
29
30        // register JSlider event listener
31        diameterJSlider.addChangeListener(
32           new ChangeListener() // anonymous inner class
33           {
34              // handle change in slider value
35              @Override
36              public void stateChanged(ChangeEvent e)
37              {
38                 myPanel.setDiameter(diameterJSlider.getValue());
39              }
40           }
41        );
42
43        add(diameterJSlider, BorderLayout.SOUTH);
44        add(myPanel, BorderLayout.CENTER);
45     }
46  } // end class SliderFrame
```

**Fig. 22.3** | `JSlider` value used to determine the diameter of a circle. (Part 2 of 2.)

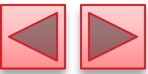
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



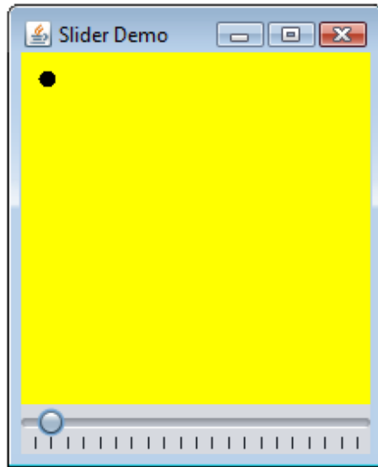
```
1  // Fig. 22.4: SliderDemo.java
2  // Testing SliderFrame.
3  import javax.swing.JFrame;
4
5  public class SliderDemo
6  {
7     public static void main(String[] args)
8     {
9        SliderFrame sliderFrame = new SliderFrame();
10       sliderFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);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

**Fig. 22.4** | Test class for `SliderFrame`. (Part 1 of 2.)

© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



a) Initial GUI with default circle




b) GUI after the user moves the `JSlider`'s thumb to the right

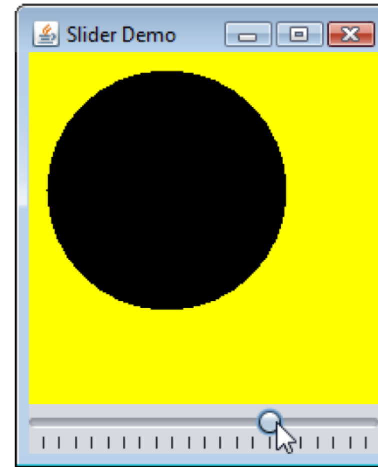
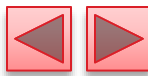



**Fig. 22.4** | Test class for `SliderFrame`. (Part 2 of 2.)

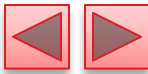
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.2 JSlider (cont.)

- JSlider method setMajorTickSpacing specifies how many values are represented by each major tick mark.
- JSlider method setPaintTicks with a true argument indicates that the tick marks should be displayed (they are not displayed by default).
- JSliders generate ChangeEvents (package javax.swing.event) in response to user interactions.
  - Handled by a ChangeListener (package javax.swing.event) that declares method stateChanged.
- JSlider method getValue returns the current thumb position.

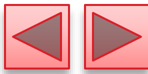
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.3 Understanding Windows in Java

- A `JFrame` *is a* window with a title bar and a border.
- `JFrame` is a subclass of `Frame`, which is a subclass of `Window`.
  - These are heavyweight Swing GUI components.
- A window is provided by the local platform's windowing toolkit.
- By default, when the user closes a `JFrame` window, it is hidden, but you can control this with `JFrame` method setDefaultCloseOperation.
  - Interface WindowConstants (package `javax.swing`), which class `JFrame` implements, declares three constants—`DISPOSE_ON_CLOSE`, `DO_NOTHING_ON_CLOSE` and `HIDE_ON_CLOSE` (the default)—for use with this method.

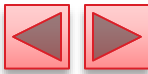
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.3 Windows: Additional Notes (cont.)

- Class `Window` (an indirect superclass of `JFrame`) declares method dispose to return a window's resources to the system.
  - When a `Window` is no longer needed in an application, you should explicitly dispose of it.
  - Can be done by calling the `Window`'s `dispose` method or by calling method `setDefaultCloseOperation` with the argument `WindowConstants.DISPOSE_ON_CLOSE`.
- A window is not displayed until the program invokes the window's `setVisible` method with a `true` argument.
- A window's size should be set with a call to method `setSize`.
- The position of a window when it appears on the screen is specified with method setLocation.

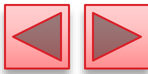
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.3 Windows: Additional Notes (cont.)

- When the user manipulates the window, window events occur.
- Event listeners are registered for window events with `Window` method addWindowListener.
- The WindowListener interface provides seven window-event-handling methods
  - windowActivated (called when user makes a window the active window)
  - windowClosed (called after the window is closed)
  - windowClosing (called when the user initiates closing of the window)
  - windowDeactivated (called when the user makes another window the active window)
  - windowDeiconified (called when a user restores a minimized window)
  - windowIconified (called when window minimized)
  - windowOpened (called when window first displayed)

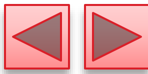
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.4 Using Menus with Frames

- Menus are an integral part of GUIs.
- Allow the user to perform actions without unnecessarily cluttering a GUI with extra components.
- In Swing GUIs, menus can be attached only to objects of the classes that provide method setJMenuBar.
  - Two such classes are `JFrame` and `JApplet`.
- The classes used to declare menus are `JMenuBar`, `JMenu`, `JMenuItem`, `JCheckBoxMenuItem` and class `JRadioButtonMenuItem`.

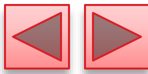
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



## Look-and-Feel Observation 22.1

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

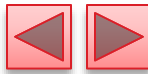
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.4 Using Menus with Frames (cont.)

- Class JMenuBar (a subclass of `JComponent`) managea a menu bar, which is a container for menus.
- Class JMenu (a subclass of `javax.swing.JMenuItem`)—menus.
  - Menus contain menu items and are added to menu bars or to other menus as submenus.
- Class JMenuItem (a subclass of `javax.swing.AbstractButton`)—menu items.
  - A menu item causes an action event when clicked.
  - Can also be a submenu that provides more menu items from which the user can select.

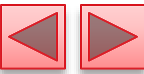
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.4 Using Menus with Frames (cont.)

- Class JCheckBoxMenuItem (a subclass of `javax.swing.JMenuItem`)—menu items that can be toggled on or off.
- Class JRadioButtonMenuItem (a subclass of `javax.swing.JMenuItem`)—menu items that can be toggled on or off like `JCheckBoxMenuItem`s.
  - When multiple `JRadioButtonMenuItem`s are maintained as part of a `ButtonGroup`, only one item in the group can be selected at a given time.
- Mnemonics can provide quick access to a menu or menu item from the keyboard.
  - Can be used with all subclasses of `javax.swing.AbstractButton`.
- `JMenu` method setMnemonic (inherited from class `AbstractButton`) indicates the mnemonic for a menu.

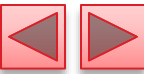
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
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};
```

**Fig. 22.5** | JMenus and mnemonics. (Part 1 of 10.)

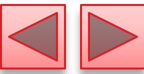
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
25     private final JRadioButtonMenuItem[] colorItems; // color menu items
26     private final JRadioButtonMenuItem[] fonts; // font menu items
27     private final JCheckBoxMenuItem[] styleItems; // font style menu items
28     private final JLabel displayJLabel; // displays sample text
29     private final ButtonGroup fontButtonGroup; // manages font menu items
30     private final 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
```

**Fig. 22.5** | JMenus and mnemonics. (Part 2 of 10.)

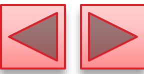
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
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             new ActionListener() // anonymous inner class
47             {
48                // display message dialog when user selects About...
49                @Override
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                }
56             }
57          );
58
```

**Fig. 22.5** | JMenus and mnemonics. (Part 3 of 10.)

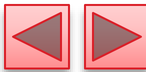
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
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(
63             new ActionListener() // anonymous inner class
64             {
65                // terminate application when user clicks exitItem
66                @Override
67                public void actionPerformed(ActionEvent event)
68                {
69                   System.exit(0); // exit application
70                }
71             }
72          );
73
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
```

**Fig. 22.5** | JMenus and mnemonics. (Part 4 of 10.)

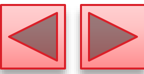
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
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 radio button menu items for colors
88         colorItems = new JRadioButtonMenuItem[colors.length];
89         colorButtonGroup = new ButtonGroup(); // manages colors
90         ItemHandler itemHandler = new ItemHandler(); // handler for colors
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        }
101
102        colorItems[0].setSelected(true); // select first Color item
103
```

**Fig. 22.5** | JMenus and mnemonics. (Part 5 of 10.)

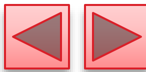
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
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 radio button 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         }
124 
125         fonts[0].setSelected(true); // select first Font menu item
126         fontMenu.addSeparator(); // add separator bar to font menu
127 
```

**Fig. 22.5** | JMenus and mnemonics. (Part 6 of 10.)

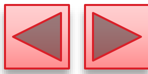
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
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        }
140
141        formatMenu.add(fontMenu); // add Font menu to Format menu
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
```

**Fig. 22.5** | JMenus and mnemonics. (Part 7 of 10.)

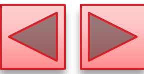
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
152
153    // inner class to handle action events from menu items
154    private class ItemHandler implements ActionListener
155    {
156       // process color and font selections
157       @Override
158       public void actionPerformed(ActionEvent event)
159       {
160          // process color selection
161          for (int count = 0; count < colorItems.length; count++)
162          {
163             if (colorItems[count].isSelected())
164             {
165                displayJLabel.setForeground(colorValues[count]);
166                break;
167             }
168          }
169
```

**Fig. 22.5** | JMenus and mnemonics. (Part 8 of 10.)

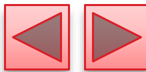
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
170          // process font selection
171          for (int count = 0; count < fonts.length; count++)
172          {
173             if (event.getSource() == fonts[count])
174             {
175                displayJLabel.setFont(
176                   new Font(fonts[count].getText(), style, 72));
177             }
178          }
179
180          repaint(); // redraw application
181       }
182    } // end class ItemHandler
183
184    // inner class to handle item events from checkbox menu items
185    private class StyleHandler implements ItemListener
186    {
187       // process font style selections
188       @Override
189       public void itemStateChanged(ItemEvent e)
190       {
191          String name = displayJLabel.getFont().getName(); // current Font
192          Font font; // new font based on user selections
193
```

**Fig. 22.5** | JMenus and mnemonics. (Part 9 of 10.)

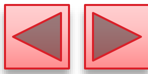
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
194           // determine which items are checked and create Font
195           if (styleItems[0].isSelected() &&
196                styleItems[1].isSelected())
197              font = new Font(name, Font.BOLD + Font.ITALIC, 72);
198           else if (styleItems[0].isSelected())
199              font = new Font(name, Font.BOLD, 72);
200           else if (styleItems[1].isSelected())
201              font = new Font(name, Font.ITALIC, 72);
202           else
203              font = new Font(name, Font.PLAIN, 72);
204
205           displayJLabel.setFont(font);
206           repaint(); // redraw application
207        }
208     }
209  } // end class MenuFrame
```

**Fig. 22.5** | JMenus and mnemonics. (Part 10 of 10.)

© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```java
 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();
10        menuFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11        menuFrame.setSize(500, 200);
12        menuFrame.setVisible(true);
13     }
14  } // end class MenuTest
```

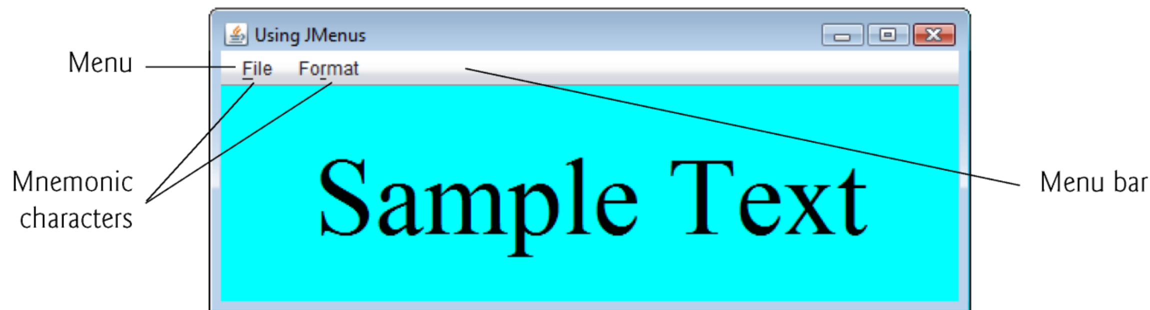
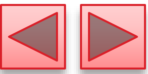



**Fig. 22.6** | Test class for MenuFrame. (Part 1 of 2.)

© Copyright 1992–2015 by Pearson Education, Inc. All Rights Reserved.



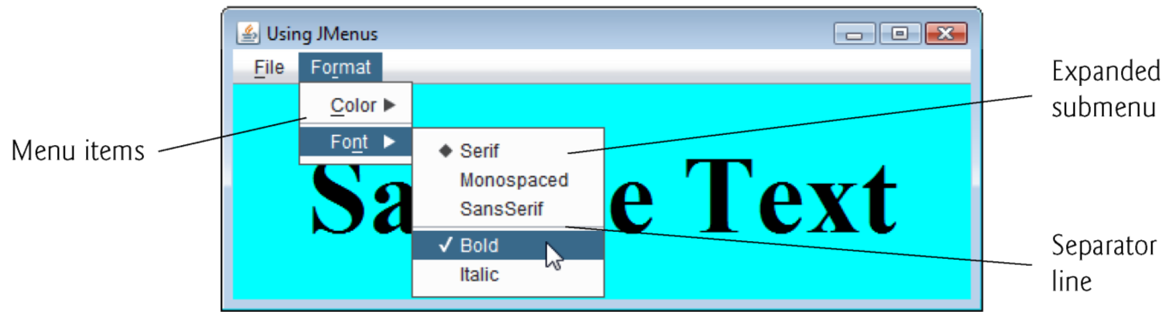


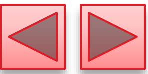

**Fig. 22.6** | Test class for `MenuFrame`. (Part 2 of 2.)

© Copyright 1992–2015 by Pearson Education, Inc. All Rights Reserved.

**Look-and-Feel Observation 22.2**

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

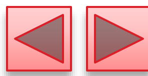
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



## Look-and-Feel Observation 22.3

*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 an* Exit *button or menu item). Mnemonics are case insensitive.*

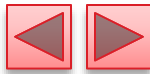
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.4 Using Menus with Frames (cont.)

- In most prior uses of `showMessageDialog`, the first argument was `null`.
  - The first argument specifies the parent window that helps determine where the dialog box will be displayed.
  - If `null`, the dialog box appears in the center of the screen.
  - Otherwise, it appears centered over the specified parent window.
- When using the `this` reference in an inner class, specifying `this` by itself refers to the inner-class object.
  - To reference the outer-class object's `this` reference, qualify `this` with the outer-class name and a dot (`.`).

© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.4 Using Menus with Frames (cont.)

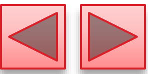
- Dialog boxes are typically modal—does not allow any other window in the application to be accessed until the dialog box is dismissed.
- Class JDialog can be used to create your own modal or nonmodal dialogs.
- `JMenuBar` method add attaches a menu to a `JMenuBar`.
- `AbstractButton` method setSelected selects the specified button.
- `JMenu` method addSeparator adds a horizontal separator line to a menu.
- `AbstractButton` method isSelected determines if a button is selected.

© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.

## Look-and-Feel Observation 22.4

*Menus appear left to right in the order they're added to a* `JMenuBar`*.*

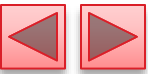
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



## Look-and-Feel Observation 22.5

*A submenu is created by adding a menu as a menu item in another menu.*

© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



## Look-and-Feel Observation 22.6

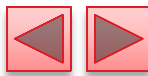
*Separators can be added to a menu to group menu items logically.*

© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.

## Look-and-Feel Observation 22.7

*Any* `JComponent` *can be added to a* `JMenu` *or to a* `JMenuBar`.

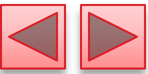
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.5 JPopupMenu

- Context-sensitive pop-up menus are created with class JPopupMenu (a subclass of `JComponent`).
  - Provide options that are specific to the component for which the popup trigger event occurred—on most systems, when the user presses and releases the right mouse button.
- `MouseEvent` method isPopupTrigger returns `true` if the popup trigger event occurred
- `JPopupMenu` method show displays a `JPopupMenu`.
  - The first argument specifies the origin component—helps determine where the `JPopupMenu` will appear on the screen.
  - The last two arguments are the *x*-*y* coordinates (measured from the origin component's upper-left corner) at which the `JPopupMenu` is to appear.

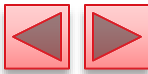
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



## Look-and-Feel Observation 22.8

*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.*

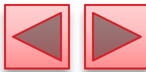
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
 1  // Fig. 22.7: PopupFrame.java
 2  // Demonstrating JPopupMenus.
 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 final JRadioButtonMenuItem[] items; // holds items for colors
16     private final Color[] colorValues =
17        { Color.BLUE, Color.YELLOW, Color.RED }; // colors to be used
18     private final JPopupMenu popupMenu; // allows user to select color
19
20     // no-argument constructor sets up GUI
21     public PopupFrame()
22     {
23        super("Using JPopupMenus");
24
```

**Fig. 22.7** | JPopupMenu for selecting colors. (Part 1 of 5.)

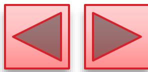
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
25        ItemHandler handler = new ItemHandler(); // handler for menu items
26        String[] colors = { "Blue", "Yellow", "Red" };
27
28        ButtonGroup colorGroup = new ButtonGroup(); // manages color items
29        popupMenu = new JPopupMenu(); // create pop-up menu
30        items = new JRadioButtonMenuItem[colors.length];
31
32        // construct menu item, add to pop-up 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        }
40
41        setBackground(Color.WHITE);
42
```

**Fig. 22.7** | JPopupMenu for selecting colors. (Part 2 of 5.)

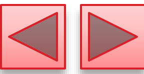
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
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              @Override
49              public void mousePressed(MouseEvent event)
50              {
51                 checkForTriggerEvent(event);
52              }
53
54              // handle mouse release event
55              @Override
56              public void mouseReleased(MouseEvent event)
57              {
58                 checkForTriggerEvent(event);
59              }
60
```

**Fig. 22.7** | JPopupMenu for selecting colors. (Part 3 of 5.)

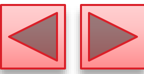
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
61             // determine whether event should trigger pop-up menu
62             private void checkForTriggerEvent(MouseEvent event)
63             {
64                if (event.isPopupTrigger())
65                   popupMenu.show(
66                      event.getComponent(), event.getX(), event.getY());
67             }
68          }
69       );
70    } // end PopupFrame constructor
71
```

**Fig. 22.7** | `JPopupMenu` for selecting colors. (Part 4 of 5.)

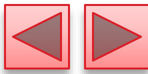
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



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

**Fig. 22.7** | `JPopupMenu` for selecting colors. (Part 5 of 5.)

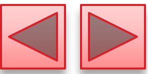
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
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();
10       popupFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11       popupFrame.setSize(300, 200);
12       popupFrame.setVisible(true);
13    }
14 } // end class PopupTest
```

**Fig. 22.8** | Test class for **PopupFrame**. (Part 1 of 2.)

© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



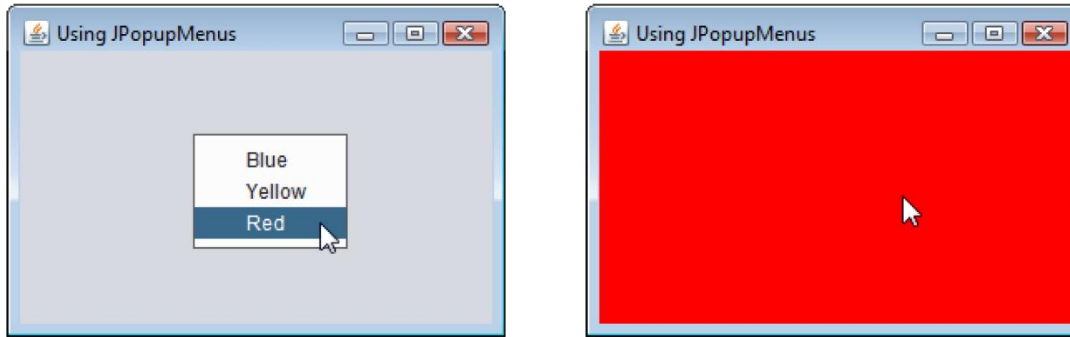
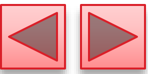



**Fig. 22.8** | Test class for `PopupFrame`. (Part 2 of 2.)

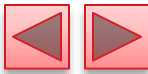
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



## Look-and-Feel Observation 22.9

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

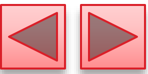
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.6 Pluggable Look-and-Feel

- Java's AWT GUI components (package `java.awt`) take on the look-and-feel of the platform on which the program executes.
  - Introduces interesting portability issues.
- Swing's lightweight GUI components provide uniform functionality across platforms and define a uniform cross-platform look-and-feel.
  - Section 12.2 introduced the *Nimbus* look-and-feel.
  - Earlier versions of Java used the metal look-and-feel, which is still the default.
- Can customize the look-and-feel
  - The installed look-and-feels will vary by platform.

© Copyright 1992–2015 by Pearson Education, Inc. All Rights Reserved.



## Portability Tip 22.1

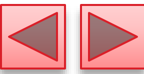
*GUI components often look different on different platforms (fonts, font sizes, component borders, etc.) and might require different amounts of space to display. This could change their layout and alignments.*

© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.

**Portability Tip 22.2**

*GUI components on different platforms have might different default functionality—e.g., not all platforms allow a button with the focus to be "pressed" with the space bar.*

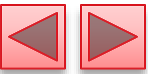
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



## 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.*

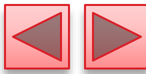
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
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  {
```

**Fig. 22.9** | Look-and-feel of a Swing-based GUI. (Part 1 of 6.)

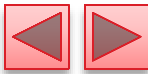
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
20      private final UIManager.LookAndFeelInfo[] looks;
21      private final String[] lookNames; // look-and-feel names
22      private final JRadioButton[] radio; // for selecting look-and-feel
23      private final ButtonGroup group; // group for radio buttons
24      private final JButton button; // displays look of button
25      private final JLabel label; // displays look of label
26      private final JComboBox<String> comboBox; // displays look of combo box
27
28      // set up GUI
29      public LookAndFeelFrame()
30      {
31         super("Look and Feel Demo");
32
33         // get installed look-and-feel information
34         looks = UIManager.getInstalledLookAndFeels();
35         lookNames = new String[looks.length];
36
37         // get names of installed look-and-feels
38         for (int i = 0; i < looks.length; i++)
39            lookNames[i] = looks[i].getName();
40
41         JPanel northPanel = new JPanel();
42         northPanel.setLayout(new GridLayout(3, 1, 0, 5));
43
```

**Fig. 22.9** | Look-and-feel of a Swing-based GUI. (Part 2 of 6.)

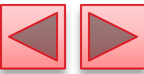
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
44         label = new JLabel("This is a " + lookNames[0] + " look-and-feel",
45            SwingConstants.CENTER);
46         northPanel.add(label);
47
48         button = new JButton("JButton");
49         northPanel.add(button);
50
51         comboBox = new JComboBox<String>(lookNames);
52         northPanel.add(comboBox);
53
54         // create array for radio buttons
55         radio = new JRadioButton[looks.length];
56
57         JPanel southPanel = new JPanel();
58
59         // use a GridLayout with 3 buttons in each row
60         int rows = (int) Math.ceil(radio.length / 3.0);
61         southPanel.setLayout(new GridLayout(rows, 3));
62
63         group = new ButtonGroup(); // button group for look-and-feels
64         ItemHandler handler = new ItemHandler(); // look-and-feel handler
65
```

**Fig. 22.9** | Look-and-feel of a Swing-based GUI. (Part 3 of 6.)

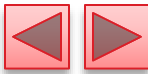
© Copyright 1992–2015 by Pearson Education, Inc. All Rights Reserved.



```
66         for (int count = 0; count < radio.length; count++)
67         {
68            radio[count] = new JRadioButton(lookNames[count]);
69            radio[count].addItemListener(handler); // add handler
70            group.add(radio[count]); // add radio button to group
71            southPanel.add(radio[count]); // add radio button to panel
72         }
73
74         add(northPanel, BorderLayout.NORTH); // add north panel
75         add(southPanel, BorderLayout.SOUTH); // add south panel
76
77         radio[0].setSelected(true); // set default selection
78      } // end LookAndFeelFrame constructor
79
```

**Fig. 22.9** | Look-and-feel of a Swing-based GUI. (Part 4 of 6.)

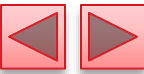
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
80    // use UIManager to change look-and-feel of GUI
81    private void changeTheLookAndFeel(int value)
82    {
83       try // change look-and-feel
84       {
85          // set look-and-feel for this application
86          UIManager.setLookAndFeel(looks[value].getClassName());
87
88          // update components in this application
89          SwingUtilities.updateComponentTreeUI(this);
90       }
91       catch (Exception exception)
92       {
93          exception.printStackTrace();
94       }
95    }
96
```

**Fig. 22.9** | Look-and-feel of a Swing-based GUI. (Part 5 of 6.)

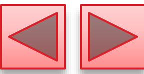
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
97      // private inner class to handle radio button events
98      private class ItemHandler implements ItemListener
99      {
100        // process user's look-and-feel selection
101        @Override
102        public void itemStateChanged(ItemEvent event)
103        {
104           for (int count = 0; count < radio.length; count++)
105           {
106              if (radio[count].isSelected())
107              {
108                 label.setText(String.format(
109                     "This is a %s look-and-feel", lookNames[count]));
110                 comboBox.setSelectedIndex(count); // set combobox index
111                 changeTheLookAndFeel(count); // change look-and-feel
112              }
113           }
114        }
115     } // end private inner class ItemHandler
116  } // end class LookAndFeelFrame
```

**Fig. 22.9** | Look-and-feel of a Swing-based GUI. (Part 6 of 6.)

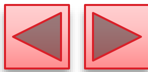
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
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(400, 220);
12       lookAndFeelFrame.setVisible(true);
13    }
14 } // end class LookAndFeelDemo
```

**Fig. 22.10** | Test class for `LookAndFeelFrame`. (Part 1 of 3.)

© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



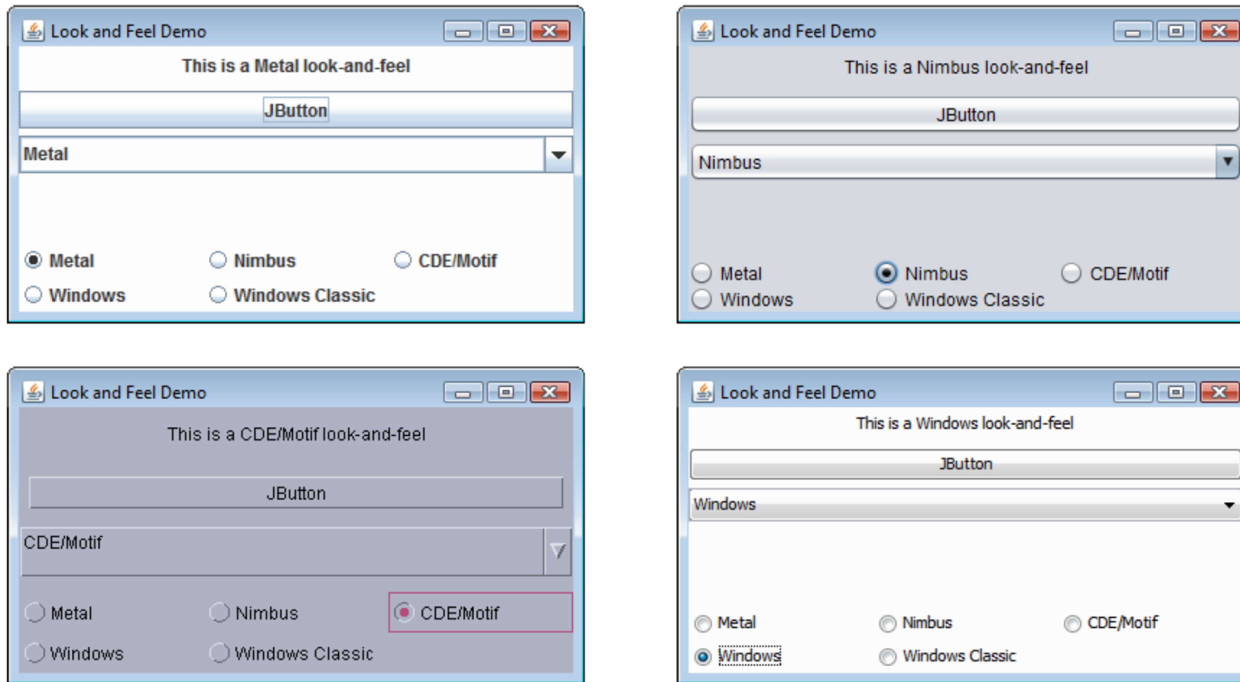
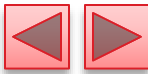



**Fig. 22.10** | Test class for `LookAndFeelFrame`. (Part 2 of 3.)

© Copyright 1992–2015 by Pearson Education, Inc. All Rights Reserved.



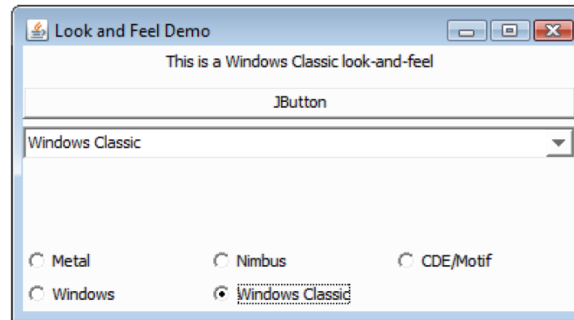
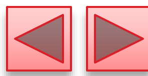



**Fig. 22.10** | Test class for `LookAndFeelFrame`. (Part 3 of 3.)

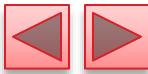
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.6 Pluggable Look-and-Feel (cont.)

- Class UIManager (package `javax.swing`) contains nested class LookAndFeelInfo (a `public static` class) that maintains information about a look-and-feel.
- `UIManager` `static` method getInstalledLookAndFeels gets an array of `UIManager.LookAndFeelInfo` objects that describe each look-and-feel available on your system.
- `UIManager` `static` method setLookAndFeel changes the look-and-feel.
- `UIManager.LookAndFeelInfo` method getClassName determines the name of the look-and-feel class that corresponds to the `UIManager.LookAndFeelInfo` object.
- SwingUtilities `static` method updateComponentTreeUI changes the look-and-feel of every GUI component attached to its argument to the new look-and-feel.

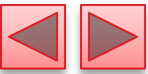
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.7 JDesktopPane and JInternalFrame

- Multiple-document interface (MDI)
  - a main window (called the parent window) containing other windows (called child windows), and is often used to manage several open documents.
- Swing's JDesktopPane and JInternalFrame classes implement multiple-document interfaces.

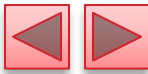
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
 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 final JDesktopPane theDesktop;
21 
```

**Fig. 22.11** | Multiple-document interface. (Part 1 of 5.)

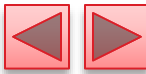
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
22    // set up GUI
23    public DesktopFrame()
24    {
25       super("Using a JDesktopPane");
26
27       JMenuBar bar = new JMenuBar();
28       JMenu addMenu = new JMenu("Add");
29       JMenuItem newFrame = new JMenuItem("Internal Frame");
30
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 application
34
35       theDesktop = new JDesktopPane();
36       add(theDesktop); // add desktop pane to frame
37
```

**Fig. 22.11** | Multiple-document interface. (Part 2 of 5.)

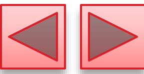
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
38         // set up listener for newFrame menu item
39         newFrame.addActionListener(
40            new ActionListener() // anonymous inner class
41            {
42               // display new internal window
43               @Override
44               public void actionPerformed(ActionEvent event)
45               {
46                  // create internal frame
47                  JInternalFrame frame = new JInternalFrame(
48                     "Internal Frame", true, true, true, true);
49
50                  MyJPanel panel = new MyJPanel();
51                  frame.add(panel, BorderLayout.CENTER);
52                  frame.pack(); // set internal frame to size of contents
53
54                  theDesktop.add(frame); // attach internal frame
55                  frame.setVisible(true); // show internal frame
56               }
57            }
58         );
59      } // end DesktopFrame constructor
60   } // end class DesktopFrame
61
```

**Fig. 22.11** | Multiple-document interface. (Part 3 of 5.)

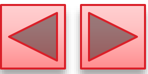
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
62  // class to display an ImageIcon on a panel
63  class MyJPanel extends JPanel
64  {
65     private static final SecureRandom generator = new SecureRandom();
66     private final ImageIcon picture; // image to be displayed
67     private final static String[] images = { "yellowflowers.png",
68        "purpleflowers.png", "redflowers.png", "redflowers2.png",
69        "lavenderflowers.png" };
70
71     // load image
72     public MyJPanel()
73     {
74        int randomNumber = generator.nextInt(images.length);
75        picture = new ImageIcon(images[randomNumber]); // set icon
76     }
77
78     // display imageIcon on panel
79     @Override
80     public void paintComponent(Graphics g)
81     {
82        super.paintComponent(g);
83        picture.paintIcon(this, g, 0, 0); // display icon
84     }
85
```

**Fig. 22.11** | Multiple-document interface. (Part 4 of 5.)

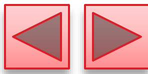
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
86      // return image dimensions
87      public Dimension getPreferredSize()
88      {
89         return new Dimension(picture.getIconWidth(),
90            picture.getIconHeight());
91      }
92   } // end class MyJPanel
```

**Fig. 22.11** | Multiple-document interface. (Part 5 of 5.)

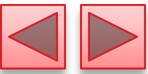
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```java
 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);
12        desktopFrame.setVisible(true);
13     }
14  } // end class DesktopTest
```

**Fig. 22.12** | Test class for DeskTopFrame. (Part 1 of 3.)

© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



Internal frames

Minimize

Maximize

Close

Using a JDesktopPane

Add

Internal Frame

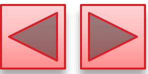
Internal Frame

Minimized internal frames

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

**Fig. 22.12** | Test class for `DeskTopFrame`. (Part 2 of 3.)

© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



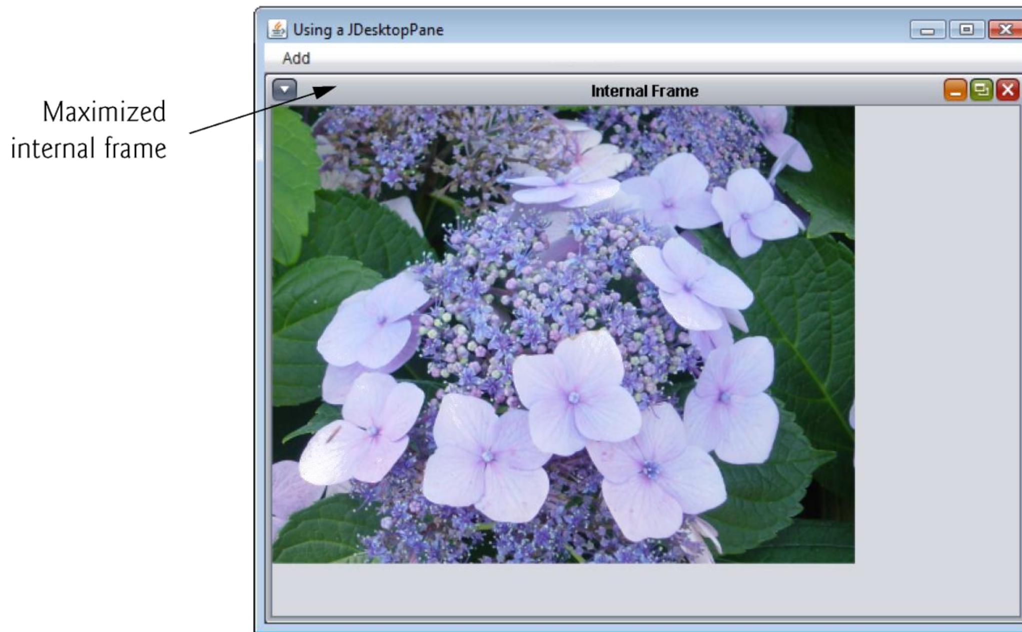
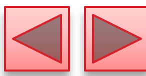



**Fig. 22.12** | Test class for `DeskTopFrame`. (Part 3 of 3.)

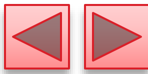
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.7 JDesktopPane and JInternalFrame (cont.)

- The `JInternalFrame` constructor used here takes five arguments
  - a `String` for the title bar of the internal window
  - a `boolean` indicating whether the internal frame can be resized by the user
  - a `boolean` indicating whether the internal frame can be closed by the user
  - a `boolean` indicating whether the internal frame can be maximized by the user
  - a `boolean` indicating whether the internal frame can be minimized by the user.
- For each of the `boolean` arguments, a `true` value indicates that the operation should be allowed (as is the case here).

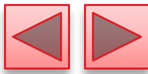
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.7 JDesktopPane and JInternalFrame (cont.)

- A JInternalFrame has a content pane to which GUI components can be attached.
- JInternalFrame method pack sets the size of the child window.
  - Uses the preferred sizes of the components to determine the window's size.
- Classes JInternalFrame and JDesktopPane provide many methods for managing child windows.

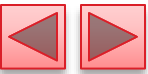
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.8 JTabbedPane

- A JTabbedPane arranges GUI components into layers, of which only one is visible at a time.
- Users access each layer by clicking a tab.
- The tabs appear at the top by default but also can be positioned at the left, right or bottom of the `JTabbedPane`.
- Any component can be placed on a tab.
  - If the component is a container, such as a panel, it can use any layout manager to lay out several components on the tab.
- Class `JTabbedPane` is a subclass of `JComponent`.

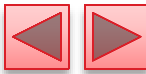
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```java
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
```

**Fig. 22.13** | JTabbedPane used to organize GUI components. (Part 1 of 3.)

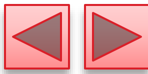
© Copyright 1992-2015 by Pearson
Education, Inc. All Rights Reserved.



```
21         // set up panel1 and add it to JTabbedPane
22         JLabel label1 = new JLabel("panel one", SwingConstants.CENTER);
23         JPanel panel1 = new JPanel();
24         panel1.add(label1);
25         tabbedPane.addTab("Tab One", null, panel1, "First Panel");
26
27         // set up panel2 and add it to JTabbedPane
28         JLabel label2 = new JLabel("panel two", SwingConstants.CENTER);
29         JPanel panel2 = new JPanel();
30         panel2.setBackground(Color.YELLOW);
31         panel2.add(label2);
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();
37         panel3.setLayout(new 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
```

**Fig. 22.13** | `JTabbedPane` used to organize GUI components. (Part 2 of 3.)

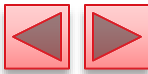
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
45          add(tabbedPane); // add JTabbedPane to frame
46      }
47   } // end class JTabbedPaneFrame
```

**Fig. 22.13** | `JTabbedPane` used to organize GUI components. (Part 3 of 3.)

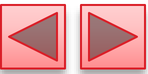
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```java
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);
12       tabbedPaneFrame.setVisible(true);
13    }
14 } // end class JTabbedPaneDemo
```

**Fig. 22.14** | Test class for `JTabbedPaneFrame`. (Part 1 of 2.)

© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



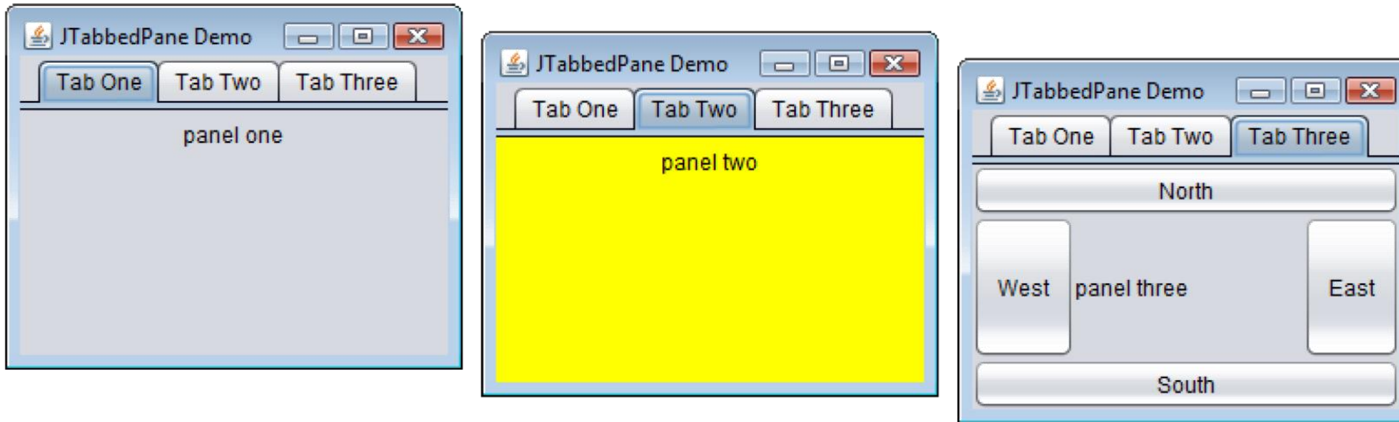
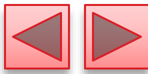



**Fig. 22.14** | Test class for `JTabbedPaneFrame`. (Part 2 of 2.)

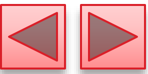
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.8 JTabbedPane (cont.)

- `JTabbedPane` method `addTab` adds a new tab. In the version with four arguments:
  - The first is a `String` that specifies the title of the tab.
  - The second is an `Icon` reference that specifies an icon to display on the tab—can be `null`
  - The third is a `Component` to display when the user clicks the tab.
  - The last is a `String` that specifies the tab's tool tip.

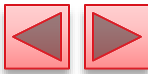
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.9 BoxLayout Manager

- This section presents two additional layout managers (summarized in Fig. 22.15).

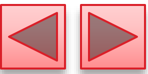
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



| Layout manager | Description |
| --- | --- |
| `BoxLayout` | Allows GUI components to be arranged left-to-right or top-to-bottom in a container. Class `Box` declares a container that uses `BoxLayout` and provides `static` methods to create a `Box` with a horizontal or vertical `BoxLayout`. |
| `GridBagLayout` | Similar to `GridLayout`, but the components can vary in size and can be added in any order. |

**Fig. 22.15** | Additional layout managers.

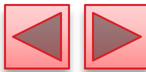
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.9 Layout Managers: `BoxLayout` and `GridBagLayout`

- The `BoxLayout` layout manager (in package `javax.swing`) arranges GUI components horizontally along a container's *x*-axis or vertically along its *y*-axis.

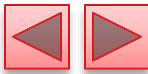
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
1  // Fig. 22.16: BoxLayoutFrame.java
2  // Demonstrating BoxLayout.
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 BoxLayoutFrame extends JFrame
12 {
13    // set up GUI
14    public BoxLayoutFrame()
15    {
16       super("Demonstrating BoxLayout");
17 
18       // create Box containers with BoxLayout
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
```

**Fig. 22.16** | `BoxLayout` layout manager. (Part 1 of 4.)

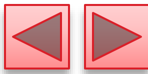
© Copyright 1992–2015 by Pearson Education, Inc. All Rights Reserved.



```
25
26          // add buttons to Box horizontal1
27          for (int count = 0; count < SIZE; count++)
28             horizontal1.add(new JButton("Button " + count));
29
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          }
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          }
43
```

**Fig. 22.16** | BoxLayout layout manager. (Part 2 of 4.)

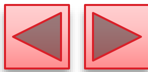
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
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        }
50
51        // create vertical glue and add buttons to panel
52        JPanel panel = new JPanel();
53        panel.setLayoutnew BoxLayout(panel, BoxLayout.Y_AXIS)  ();
54
55        for (int count = 0; count < SIZE; count++)
56        {
57           panel.add(Box.createGlue());
58           panel.add(new JButton("Button " + count));
59        }
60
61        // create a JTabbedPane
62        JTabbedPane tabs = new JTabbedPane(
63           JTabbedPane.TOP, JTabbedPane.SCROLL_TAB_LAYOUT);
64
```

**Fig. 22.16** | BoxLayout layout manager. (Part 3 of 4.)

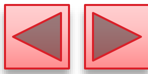
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
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 BoxLayoutFrame constructor
74   } // end class BoxLayoutFrame
```

**Fig. 22.16** | `BoxLayout` layout manager. (Part 4 of 4.)

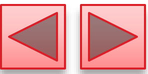
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



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

**Fig. 22.17** | Test class for `BoxLayoutFrame`. (Part 1 of 2.)

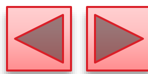
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



Arrows for cycling through tabs

**Fig. 22.17** | Test class for `BoxLayoutFrame`. (Part 2 of 2.)

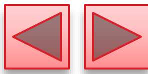
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.9 BoxLayout Manager (cont.)

- `static` `Box` method createVerticalBox returns references to `Box` containers with a vertical `BoxLayout` in which GUI components are arranged top-to-bottom.
- Before adding each button, line 33 adds a vertical strut to the container with .
- A vertical strut is an invisible GUI component that has a fixed pixel height and is used to guarantee a fixed amount of space between GUI components.
  - created with `static` `Box` method createVerticalStrut
  - When the container is resized, the distance between GUI components separated by struts does not change.
- Class `Box` also declares method createHorizontalStrut for horizontal `BoxLayout`s.

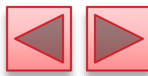
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.9 BoxLayout Manager (cont.)

- Horizontal glue is an invisible GUI component that can be used between fixed-size GUI components to occupy additional space.
  - created with `static` `Box` method createHorizontalGlue
  - When the container is resized, components separated by glue components remain the same size, but the glue stretches or contracts to occupy the space between them.
- Class `Box` also declares method createVerticalGlue for vertical `BoxLayout`s.

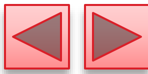
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.9 BoxLayout Manager (cont.)

- A rigid area is an invisible GUI component that always has a fixed pixel width and height.
  - created with `static` `Box` method create-RigidArea
- The `BoxLayout` constructor receives a reference to the container for which it controls the layout and a constant indicating whether the layout is horizontal (BoxLayout.X_AXIS) or vertical (BoxLayout.Y_AXIS).
- `static` `Box` method createGlue creates a component that expands or contracts based on the size of the `Box`.
- JTabbedPane.TOP—tabs should appear at the top of the `JTabbedPane`.
- JTabbedPane.SCROLL_TAB_LAYOUT—tabs should wrap to a new line if there are too many to fit on one line.

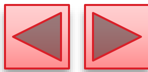
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.10 GridBagLayout Layout Manager

- One of the most powerful predefined layout managers is GridBagLayout (in package `java.awt`).
- Similar to `GridLayout` but much more flexible.
- Components can vary in size and can be added in any order.

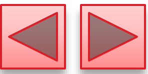
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.10 GridBagLayout Layout Manager (cont.)

- The first step in using `GridBagLayout` is determining the appearance of the GUI.
- Use paper to draw the GUI, then draw a grid over it, dividing the components into rows and columns.
- The initial row and column numbers should be 0, so that the `GridBagLayout` layout manager can use the row and column numbers to properly place the components in the grid.
- Figure 22.18 demonstrates drawing the lines for the rows and columns over a GUI.

© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



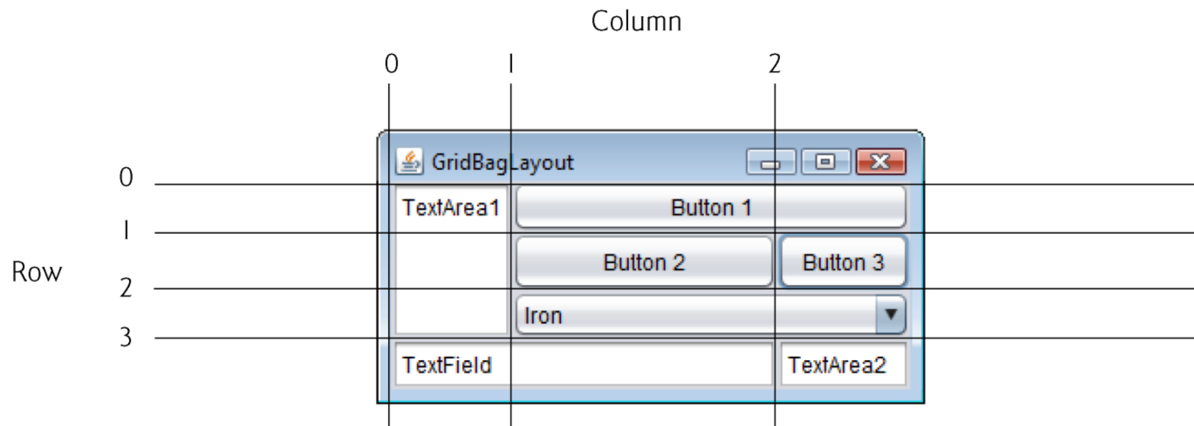
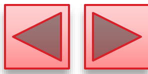



**Fig. 22.18** | Designing a GUI that will use `GridBagLayout`.

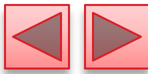
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.10 GridBagLayout Layout Manager (cont.)

- A GridBagConstraints object describes how a component is placed in a `GridBagLayout`.
- Several `GridBagConstraints` fields are summarized in Fig. 22.19.
- `GridBagLayout` method setConstraints takes a `Component` argument and a `GridBagConstraints` argument.

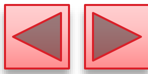
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



| 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 the 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.

© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



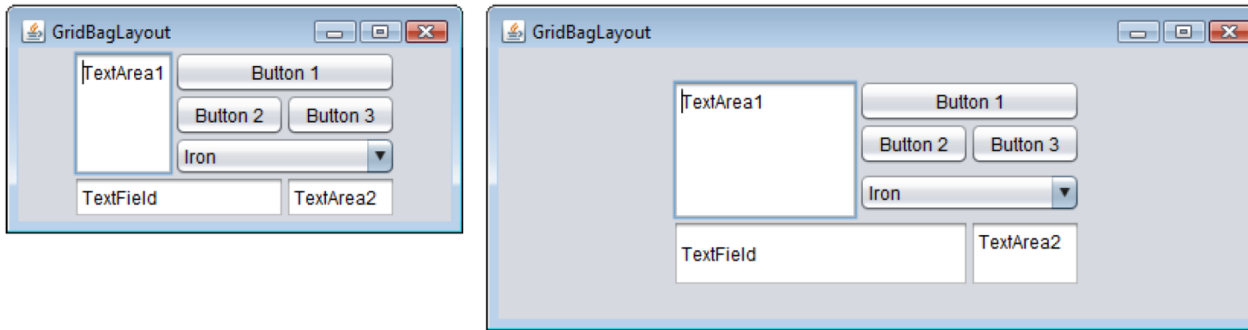
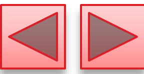



**Fig. 22.20** | `GridBagLayout` with the weights set to zero.

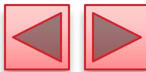
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
 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 final GridBagLayout layout; // layout of this frame
15     private final GridBagConstraints constraints; // layout's constraints
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
```

**Fig. 22.21** | `GridBagLayout` layout manager. (Part 1 of 4.)

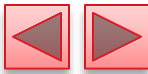
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
25         // create GUI components
26         JTextArea textArea1 = new JTextArea("TextArea1", 5, 10);
27         JTextArea textArea2 = new JTextArea("TextArea2", 2, 2);
28 
29         String[] names = { "Iron", "Steel", "Brass" };
30         JComboBox<String> comboBox = new JComboBox<String>(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);
```

**Fig. 22.21** | GridBagLayout layout manager. (Part 2 of 4.)

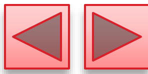
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
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
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
```

**Fig. 22.21** | `GridBagLayout` layout manager. (Part 3 of 4.)

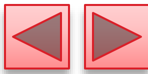
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
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;
73         constraints.gridy = row;
74         constraints.gridwidth = width;
75         constraints.gridheight = height;
76         layout.setConstraints(component, constraints); // set constraints
77         add(component); // add component
78      }
79   } // end class GridBagFrame
```

**Fig. 22.21** | `GridBagLayout` layout manager. (Part 4 of 4.)

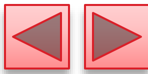
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
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);
12       gridBagFrame.setVisible(true);
13    }
14 } // end class GridBagDemo
```

**Fig. 22.22** | Test class for `GridBagFrame`. (Part 1 of 3.)

© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



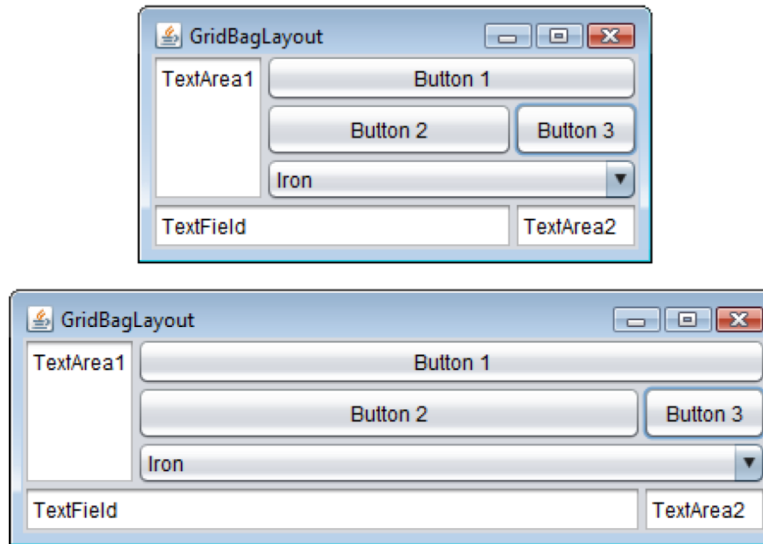
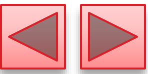



**Fig. 22.22** | Test class for `GridBagFrame`. (Part 2 of 3.)

© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



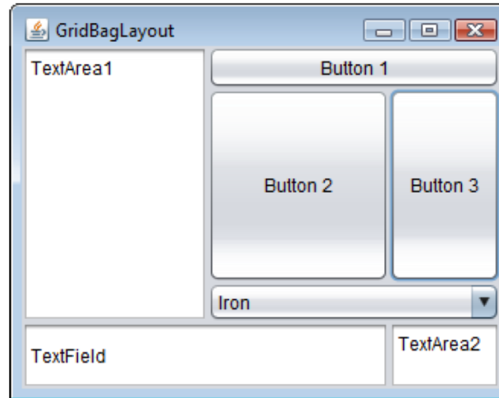
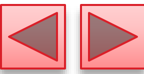



**Fig. 22.22** | Test class for `GridBagFrame`. (Part 3 of 3.)

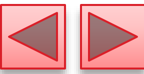
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



# 22.10 GridBagLayout Layout Manager (cont.)

- A variation of GridBagLayout uses GridBagConstraints constants RELATIVE and REMAINDER.
  - 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.

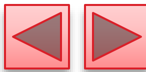
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
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 final GridBagLayout layout; // layout of this frame
15    private final GridBagConstraints constraints; // layout's constraints
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
```

**Fig. 22.23** | GridBagConstraints constants RELATIVE and REMAINDER. (Part 1 of 4.)

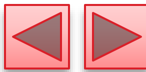
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
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
31        String[] fonts = { "Serif", "Monospaced" };
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
```

**Fig. 22.23** | `GridBagConstraints` constants `RELATIVE` and `REMAINDER`. (Part 2 of 4.)

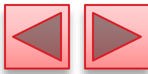
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
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
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
```

**Fig. 22.23** | `GridBagConstraints` constants `RELATIVE` and `REMAINDER`. (Part 3 of 4.)

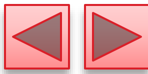
© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



```
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       }
84    } // end class GridBagFrame2
```

**Fig. 22.23** | `GridBagConstraints` constants `RELATIVE` and `REMAINDER`. (Part 4 of 4.)

© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.

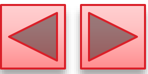


```java
// Fig. 22.24: GridBagDemo2.java
// Demonstrating GridBagLayout constants.
import javax.swing.JFrame;

public class GridBagDemo2
{
   public static void main(String[] args)
   {
      GridBagFrame2 gridBagFrame = new GridBagFrame2();
      gridBagFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      gridBagFrame.setSize(300, 200);
      gridBagFrame.setVisible(true);
   }
} // end class GridBagDemo2
```

**Fig. 22.24** | Test class for GridBagDemo2. (Part 1 of 2.)

© Copyright 1992-2015 by Pearson Education, Inc. All Rights Reserved.



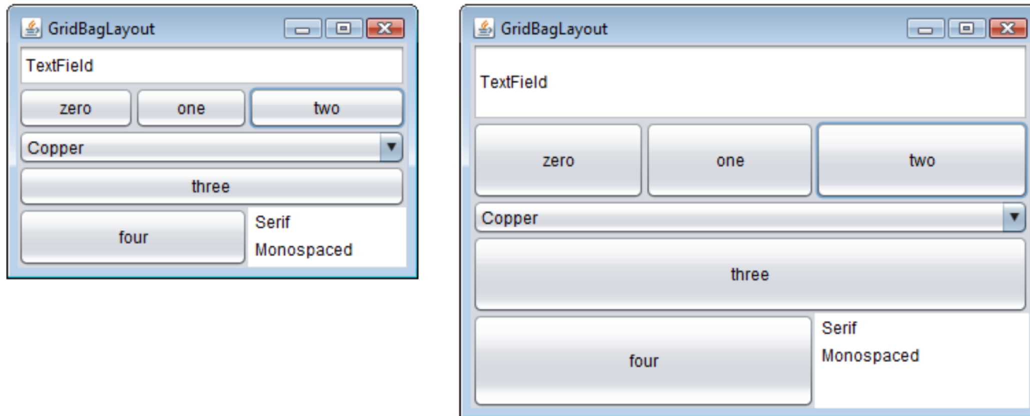



**Fig. 22.24** | Test class for `GridBagDemo2`. (Part 2 of 2.)

© Copyright 1992–2015 by Pearson Education, Inc. All Rights Reserved.