Graphical User Components
Part 2

Outline

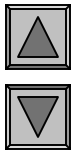
Introduction
`JTextArea`
`JSlider`
Windows: Additional Notes
Using Menus with Frames
Pluggable Look-and-Feel
`JDesktopPane` **and** `JInternalFrame`
`JTabbedPane`

14.1 Introduction

- Advanced GUI components
  - Text areas
  - Sliders
  - Menus
- Multiple Document Interface (MDI)
- Advanced layout managers
  - `BoxLayout`
  - `GridBagLayout`

14.2 JTextArea

- JTextArea
  - Area for manipulating multiple lines of text
  - extends JTextComponent



Outline

TextAreaDemo.java

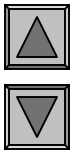
Line 16

Lines 18-24

```
1  // Fig. 14.1: TextAreaDemo.java
2  // Copying selected text from one textarea to another.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class TextAreaDemo extends JFrame {
8     private JTextArea textArea1, textArea2;
9     private JButton copyButton;
10
11    // set up GUI
12    public TextAreaDemo()
13    {
14       super( "TextArea Demo" );
15
16       Box box = Box.createHorizontalBox();
17
18       String string = "This is a demo string to\n" +
19          "illustrate copying text\nfrom one textarea to \n" +
20          "another textarea using an\nexternal event\n";
21
22       // set up textArea1
23       textArea1 = new JTextArea( string, 10, 15 );
24       box.add( new JScrollPane( textArea1 ) );
25
```

Create `Box` container for organizing GUI components

Populate `JTextArea` with `String`, then add to `Box`



Outline

TextAreaDemo.java

Line 36

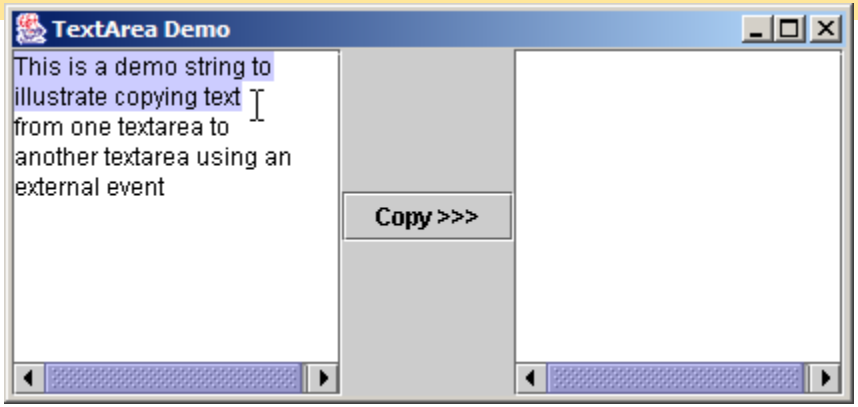
Lines 44-45

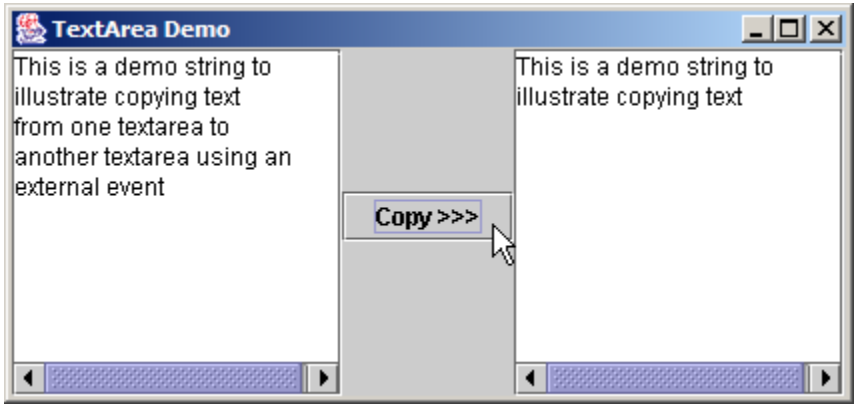
```
26        // set up copyButton
27        copyButton = new JButton( "Copy >>>" );
28        box.add( copyButton );
29        copyButton.addActionListener(
30
31           new ActionListener() {  // anonymous inner class
32
33              // set text in textArea2 to selected text from textArea1
34              public void actionPerformed( ActionEvent event )
35              {
36                 textArea2.setText( textArea1.getSelectedText() );
37              }
38
39           } // end anonymous inner class
40
41        ); // end call to addActionListener
42
43        // set up textArea2
44        textArea2 = new JTextArea( 10, 15 );
45        textArea2.setEditable( false );
46        box.add( new JScrollPane( textArea2 ) );
47
48        // add box to content pane
49        Container container = getContentPane();
50        container.add( box );   // place in BorderLayout.CENTER
51
```

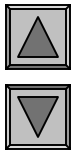
When user presses `JButton`, `textArea1`'s highlighted text is copied into `textArea2`

Instantiate uneditable `JTextArea`

```
52        setSize( 425, 200 );
53        setVisible( true );
54
55     } // end constructor TextAreaDemo
56
57     public static void main( String args[] )
58     {
59        TextAreaDemo application = new TextAreaDemo();
60        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
61     }
62
63  } // end class TextAreaDemo
```







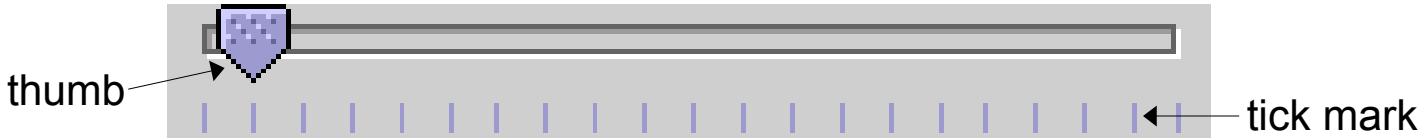
Outline

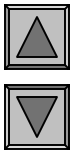
TextAreaDemo.java

14.5 JSlider

- JSlider
  - Enable users to select from range of integer values
  - Several features
    - Tick marks (major and minor)
    - Snap-to ticks
    - Orientation (horizontal and vertical)

Fig. 14.6 `JSlider` component with horizontal orientation





Outline

OvalPanel.java

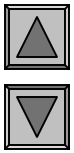
Line 14

Line 18

```
1  // Fig. 14.7: OvalPanel.java
2  // A customized JPanel class.
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class OvalPanel extends JPanel {
7     private int diameter = 10;
8
9     // draw an oval of the specified diameter
10    public void paintComponent( Graphics g )
11    {
12       super.paintComponent( g );
13
14       g.fillOval( 10, 10, diameter, diameter );
15    }
16
17    // validate and set diameter, then repaint
18    public void setDiameter( int newDiameter )
19    {
20       // if diameter invalid, default to 10
21       diameter = ( newDiameter >= 0 ? newDiameter : 10 );
22       repaint();
23    }
24
```

Draw filled oval of `diameter`

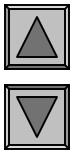
Set `diameter`, then repaint



Outline

OvalPanel.java

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



Outline

SliderDemo.java

Lines 18-19

Lines 22-23

```
1  // Fig. 14.8: SliderDemo.java
2  // Using JSliders to size an oval.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6  import javax.swing.event.*;
7
8  public class SliderDemo extends JFrame {
9     private JSlider diameterSlider;
10    private OvalPanel myPanel;
11
12    // set up GUI
13    public SliderDemo()
14    {
15       super( "Slider Demo" );
16
17       // set up OvalPanel
18       myPanel = new OvalPanel();
19       myPanel.setBackground( Color.YELLOW );
20
21       // set up JSlider to control diameter value
22       diameterSlider =
23          new JSlider( SwingConstants.HORIZONTAL, 0, 200, 10 );
24       diameterSlider.setMajorTickSpacing( 10 );
25       diameterSlider.setPaintTicks( true );
26
```

Instantiate `OvalPanel` object and set background to yellow

Instantiate horizontal `JSlider` object with min. value of 0, max. value of 200 and initial thumb location at 10

```
27        // register JSlider event listener
28        diameterSlider.addChangeListener(
29
30           new ChangeListener() {  // anonymous inner class
31
32              // handle change in slider value
33              public void stateChanged( ChangeEvent e )
34              {
35                 myPanel.setDiameter( diameterSlider.getValue() );
36              }
37
38           } // end anonymous inner class
39
40        ); // end call to addChangeListener
41
42        // attach components to content pane
43        Container container = getContentPane();
44        container.add( diameterSlider, BorderLayout.SOUTH );
45        container.add( myPanel, BorderLayout.CENTER );
46
47        setSize( 220, 270 );
48        setVisible( true );
49
50     } // end constructor SliderDemo
51
```

Register anonymous `ChangeListener` object to handle `JSlider` events

When user accesses `JSlider`, set `OvalPanel`'s `diameter` according to `JSlider` value

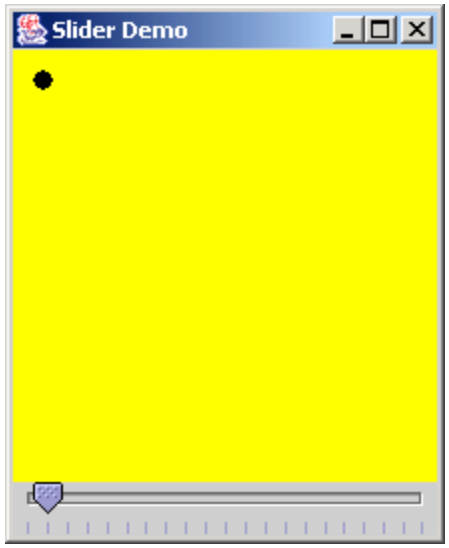
Outline

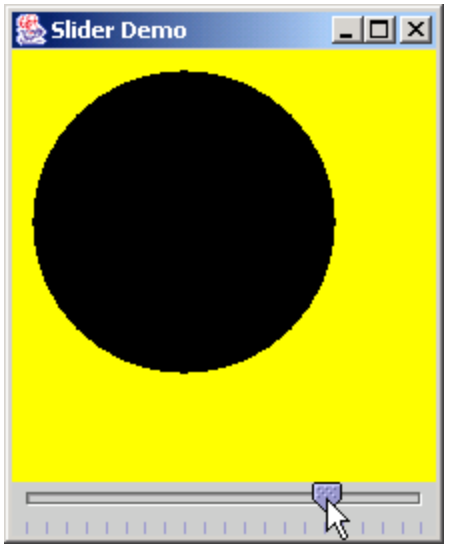
`SliderDemo.java`

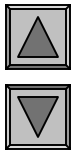
Line 28

Line 35

```
52    public static void main( String args[] )
53    {
54       SliderDemo application = new SliderDemo();
55       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
56    }
57
58 } // end class SliderDemo
```







Outline

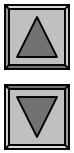
SliderDemo.java

14.6 Windows: Additional Notes

- `JFrame`
  - Windows with *title bar* and *border*
  - Subclass of `java.awt.Frame`
    - Subclass of `java.awt.Window`
  - Heavyweight component
  - Three operations when user closes window
    - `DISPOSE_ON_CLOSE`
    - `DO_NOTHING_ON_CLOSE`
    - `HIDE_ON_CLOSE`

14.7 Using Menus with Frames

- Menus
  - Allows for performing actions with cluttering GUI
  - Contained by menu bar
    - `JMenuBar`
  - Comprised of menu items
    - `JMenuItem`



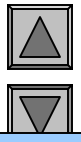
Outline

MenuTest.java

Line 22

```
1  // Fig. 14.9: MenuTest.java
2  // Demonstrating menus
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class MenuTest extends JFrame {
8     private final Color colorValues[] =
9        { Color.BLACK, Color.BLUE, Color.RED, Color.GREEN };
10    private JRadioButtonMenuItem colorItems[], fonts[];
11    private JCheckBoxMenuItem styleItems[];
12    private JLabel displayLabel;
13    private ButtonGroup fontGroup, colorGroup;
14    private int style;
15
16    // set up GUI
17    public MenuTest()
18    {
19       super( "Using JMenus" );
20
21       // set up File menu and its menu items
22       JMenu fileMenu = new JMenu( "File" );
23       fileMenu.setMnemonic( 'F' );
24
```

Instantiate **File** JMenu



Outline

uTest.java

Line 26

Lines 36-38

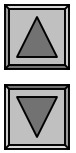
Line 46

```
25       // set up About... menu item
26       JMenuItem aboutItem = new JMenuItem( "About..." );
27       aboutItem.setMnemonic( 'A' );
28       fileMenu.add( aboutItem );
29       aboutItem.addActionListener(
30
31          new ActionListener() {  // anonymous inner class
32
33             // display message dialog when user selects About...
34             public void actionPerformed( ActionEvent event )
35             {
36                JOptionPane.showMessageDialog( MenuTest.this,
37                   "This is an example\nof using menus",
38                   "About", JOptionPane.PLAIN_MESSAGE );
39             }
40
41          } // end anonymous inner class
42
43       ); // end call to addActionListener
44
45       // set up Exit menu item
46       JMenuItem exitItem = new JMenuItem( "Exit" );
47       exitItem.setMnemonic( 'x' );
48       fileMenu.add( exitItem );
49       exitItem.addActionListener(
50
```

Instantiate **About...** `JMenuItem` to be placed in `fileMenu`

When user selects **About...** `JMenuItem`, display message dialog with appropriate text

Instantiate **Exit** `JMenuItem` to be placed in `fileMenu`



Outline

MenuTest.java

Line 56

Line 64

Line 69

```
51          new ActionListener() {  // anonymous inner class
52
53             // terminate application when user clicks exitItem
54             public void actionPerformed( ActionEvent event )
55             {
56                System.exit( 0 );
57             }
58
59          }  // end anonymous inner class
60
61       ); // end call to addActionListener
62
63       // create menu bar and attach it to MenuTest window
64       JMenuBar bar = new JMenuBar();
65       setJMenuBar( bar );
66       bar.add( fileMenu );
67
68       // create Format menu, its submenus and menu items
69       JMenu formatMenu = new JMenu( "Format" );
70       formatMenu.setMnemonic( 'r' );
71
72       // create Color submenu
73       String colors[] = { "Black", "Blue", "Red", "Green" };
74
```

When user selects **Exit** `JMenuItem`, exit system

Instantiate `JMenuBar` to contain `JMenu`s

Instantiate **Format** `JMenu`

```
75        JMenu colorMenu = new JMenu( "Color" );
76        colorMenu.setMnemonic( 'C' );
77
78        colorItems = new JRadioButtonMenuItem[ colors.length ];
79        colorGroup = new ButtonGroup();
80        ItemHandler itemHandler = new ItemHandler();
81
82        // create color radio button menu items
83        for ( int count = 0; count < colors.length; count++ ) {
84           colorItems[ count ] =
85              new JRadioButtonMenuItem( colors[ count ] );
86           colorMenu.add( colorItems[ count ] );
87           colorGroup.add( colorItems[ count ] );
88           colorItems[ count ].addActionListener( itemHandler
89        }
90
91        // select first Color menu item
92        colorItems[ 0 ].setSelected( true );
93
94        // add format menu to menu bar
95        formatMenu.add( colorMenu );
96        formatMenu.addSeparator();
97
98        // create Font submenu
99        String fontNames[] = { "Serif", "Monospaced", "SansSerif" };
100
```

Instantiate **Color** `JMenu` (submenu of **Format** `JMenu`)

Instantiate `JRadioButtonMenuItem`s for **Color** `JMenu` and ensure that only one menu item is selected at a time

Separator places line between `JMenuItem`s

Outline

`MenuTest.java`

Line 75

Lines 78-79

```
101       JMenu fontMenu = new JMenu( "Font" );
102       fontMenu.setMnemonic( 'n' );
103
104       fonts = new JRadioButtonMenuItem[ fontNames.length ];
105       fontGroup = new ButtonGroup();
106
107       // create Font radio button menu items
108       for ( int count = 0; count < fonts.length; count++ ) {
109          fonts[ count ] = new JRadioButtonMenuItem( fontNames[ count ] );
110          fontMenu.add( fonts[ count ] );
111          fontGroup.add( fonts[ count ] );
112          fonts[ count ].addActionListener( itemHandler );
113       }
114
115       // select first Font menu item
116       fonts[ 0 ].setSelected( true );
117
118       fontMenu.addSeparator();
119
120       // set up style menu items
121       String styleNames[] = { "Bold", "Italic" };
122
123       styleItems = new JCheckBoxMenuItem[ styleNames.length ];
124       StyleHandler styleHandler = new StyleHandler();
125
```

Instantiate **Font** `JMenu` (submenu of **Format** `JMenu`)

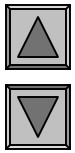
Instantiate `JRadioButtonMenuItem`s for **Font** `JMenu` and ensure that only one menu item is selected at a time

Outline

`MenuTest.java`

Line 101

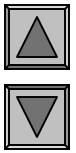
Lines 104-105



Outline

MenuTest.java

```
126       // create style checkbox menu items
127       for ( int count = 0; count < styleNames.length; count++ ) {
128          styleItems[ count ] =
129             new JCheckBoxMenuItem( styleNames[ count ] );
130          fontMenu.add( styleItems[ count ] );
131          styleItems[ count ].addItemListener( styleHandler );
132       }
133
134       // put Font menu in Format menu
135       formatMenu.add( fontMenu );
136
137       // add Format menu to menu bar
138       bar.add( formatMenu );
139
140       // set up label to display text
141       displayLabel = new JLabel( "Sample Text", SwingConstants.CENTER );
142       displayLabel.setForeground( colorValues[ 0 ] );
143       displayLabel.setFont( new Font( "Serif", Font.PLAIN, 72 ) );
144
145       getContentPane().setBackground( Color.CYAN );
146       getContentPane().add( displayLabel, BorderLayout.CENTER );
147
148       setSize( 500, 200 );
149       setVisible( true );
150
151    } // end constructor
152
```



Outline

MenuTest.java

Line 163

Lines 168 and 176

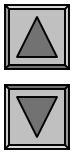
d 177-

```
153    public static void main( String args[] )
154    {
155       MenuTest application = new MenuTest();
156       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
157    }
158
159    // inner class to handle action events from menu items
160    private class ItemHandler implements ActionListener {
161
162       // process color and font selections
163       public void actionPerformed( ActionEvent event )
164       {
165          // process color selection
166          for ( int count = 0; count < colorItems.length; count++ )
167
168             if ( colorItems[ count ].isSelected() ) {
169                displayLabel.setForeground( colorValues[ count ] );
170                break;
171             }
172
173          // process font selection
174          for ( int count = 0; count < fonts.length; count++ )
175
176             if ( event.getSource() == fonts[ count ] ) {
177                displayLabel.setFont(
178                   new Font( fonts[ count ].getText(), style, 72 ) );
179                break;
180             }
```

Invoked when user selects `JMenuItem`

Determine which font or color menu generated event

Set font or color of `JLabel`, respectively



Outline

MenuTest.java

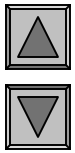
e 192

Lines 197-202

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

Invoked when user selects `JCheckBoxMenuItem`

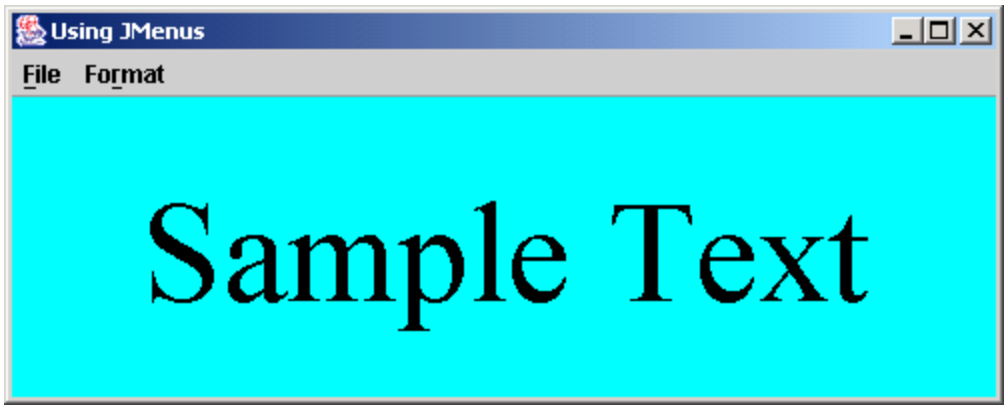
Determine new font style

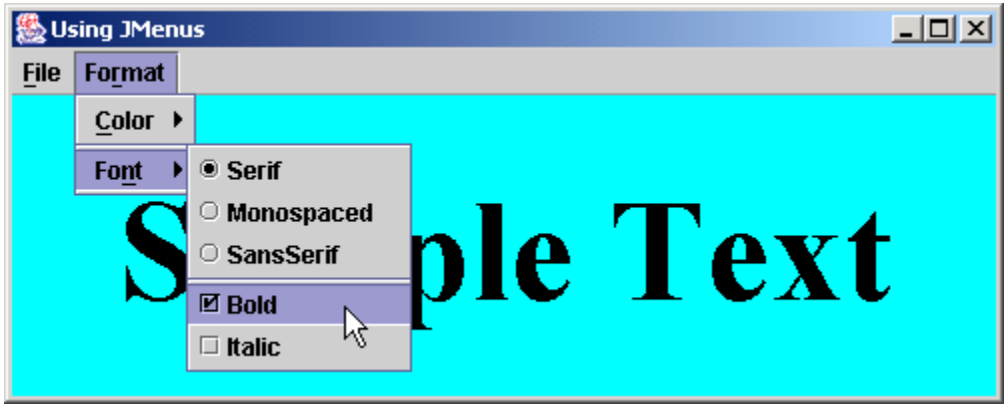


Outline

MenuTest.java

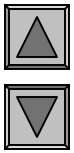
```
206
207          repaint();
208       }
209
210    } // end class StyleHandler
211
212 } // end class MenuTest
```





14.9 Pluggable Look-and-Feel

- Pluggable look-and-feel
  - Change look-and-feel dynamically
    - e.g., Microsoft Windows look-and-feel to Motif look-and-feel
  - Flexible



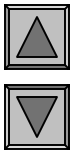
Outline

LookAndFeelDemo.java

Line 9

```
1  // Fig. 14.11: LookAndFeelDemo.java
2  // Changing the look and feel.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class LookAndFeelDemo extends JFrame {
8     private final String strings[] = { "Metal", "Motif", "Windows" };
9     private UIManager.LookAndFeelInfo looks[];
10    private JRadioButton radio[];
11    private ButtonGroup group;
12    private JButton button;
13    private JLabel label;
14    private JComboBox comboBox;
15
16    // set up GUI
17    public LookAndFeelDemo()
18    {
19       super( "Look and Feel Demo" );
20
21       Container container = getContentPane();
22
23       // set up panel for NORTH of BorderLayout
24       JPanel northPanel = new JPanel();
25       northPanel.setLayout( new GridLayout( 3, 1, 0, 5 ) );
26
```

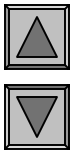
Hold installed look-and-feel information



Outline

LookAndFeelDemo.java

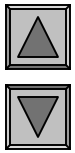
```
27        // set up label for NORTH panel
28        label = new JLabel( "This is a Metal look-and-feel",
29           SwingConstants.CENTER );
30        northPanel.add( label );
31
32        // set up button for NORTH panel
33        button = new JButton( "JButton" );
34        northPanel.add( button );
35
36        // set up combo box for NORTH panel
37        comboBox = new JComboBox( strings );
38        northPanel.add( comboBox );
39
40        // create array for radio buttons
41        radio = new JRadioButton[ strings.length ];
42
43        // set up panel for SOUTH of BorderLayout
44        JPanel southPanel = new JPanel();
45        southPanel.setLayout( new GridLayout( 1, radio.length ) );
46
47        // set up radio buttons for SOUTH panel
48        group = new ButtonGroup();
49        ItemHandler handler = new ItemHandler();
50
```



Outline

LookAndFeelDemo.java

```
51        for ( int count = 0; count < radio.length; count++ ) {
52           radio[ count ] = new JRadioButton( strings[ count ] );
53           radio[ count ].addItemListener( handler );
54           group.add( radio[ count ] );
55           southPanel.add( radio[ count ] );
56        }
57
58        // attach NORTH and SOUTH panels to content pane
59        container.add( northPanel, BorderLayout.NORTH );
60        container.add( southPanel, BorderLayout.SOUTH );
61
62        // get installed look-and-feel information
63        looks = UIManager.getInstalledLookAndFeels();
64
65        setSize( 300, 200 );
66        setVisible( true );
67
68        radio[ 0 ].setSelected( true );
69
70     } // end constructor LookAndFeelDemo
71
72     // use UIManager to change look-and-feel of GUI
73     private void changeTheLookAndFeel( int value )
74     {
```



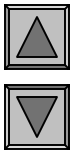
Outline

LookAndFeelDemo.java

```
75        // change look and feel
76        try {
77           UIManager.setLookAndFeel( looks[ value ].getClassName() );
78           SwingUtilities.updateComponentTreeUI( this );
79        }
80
81        // process problems changing look and feel
82        catch ( Exception exception ) {
83           exception.printStackTrace();
84        }
85     }
86
87     public static void main( String args[] )
88     {
89        LookAndFeelDemo application = new LookAndFeelDemo();
90        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
91     }
92
93     // private inner class to handle radio button events
94     private class ItemHandler implements ItemListener {
95
96        // process user's look-and-feel selection
97        public void itemStateChanged( ItemEvent event )
98        {
99           for ( int count = 0; count < radio.length; count++ )
100
```

Change look-and-feel

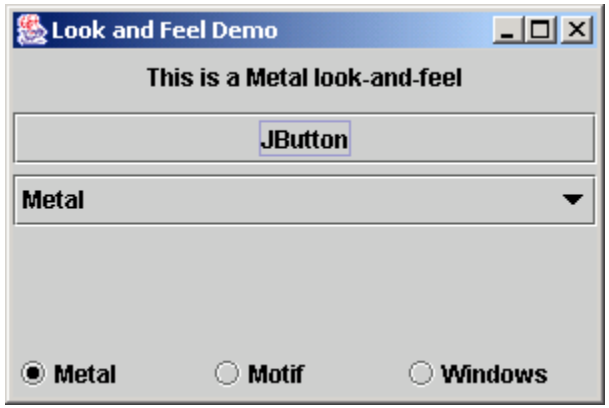
Lines 77-78

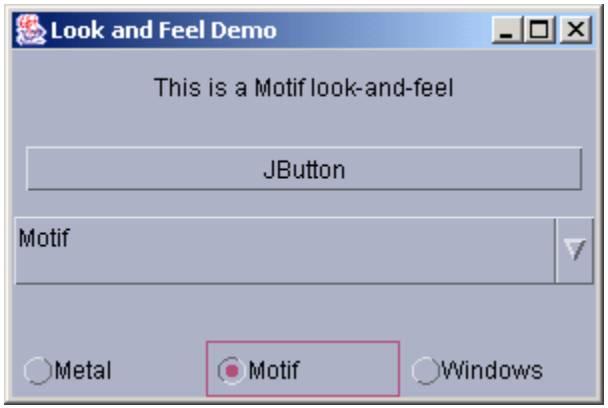


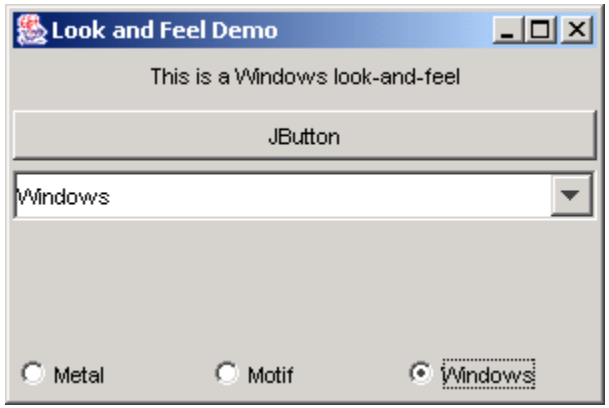
Outline

LookAndFeelDemo.java

```
101            if ( radio[ count ].isSelected() ) {
102               label.setText( "This is a " +
103                  strings[ count ] + " look-and-feel" );
104               comboBox.setSelectedIndex( count );
105               changeTheLookAndFeel( count );
106            }
107         }
108
109      } // end private inner class ItemHandler
110
111   } // end class LookAndFeelDemo
```

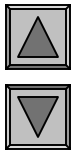






14.10 `JDesktopPane` and `JInternalFrame`

- Multiple document interface
  - Main (parent) window
  - Child windows
  - Switch freely among documents

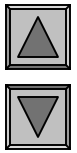


Outline

DesktopTest.java

```
1  // Fig. 14.12: DesktopTest.java
2  // Demonstrating JDesktopPane.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class DesktopTest extends JFrame {
8     private JDesktopPane theDesktop;
9
10    // set up GUI
11    public DesktopTest()
12    {
13       super( "Using a JDesktopPane" );
14
15       // create menu bar, menu and menu item
16       JMenuBar bar = new JMenuBar();
17       JMenu addMenu = new JMenu( "Add" );
18       JMenuItem newFrame = new JMenuItem( "Internal Frame" );
19
20       addMenu.add( newFrame );
21       bar.add( addMenu );
22
23       setJMenuBar( bar );
24
25       // set up desktop
26       theDesktop = new JDesktopPane();
27       getContentPane().add( theDesktop );
```

Manages `JInternalFrame` child windows displayed in `JDesktopPane`



Outline

DesktopTest.java

```
28
29       // set up listener for newFrame menu item
30       newFrame.addActionListener(
31
32          new ActionListener() {  // anonymous inner class
33
34             // display new internal window
35             public void actionPerformed( ActionEvent event ) {
36
37                // create internal frame
38                JInternalFrame frame = new JInternalFrame(
39                   "Internal Frame", true, true, true, true );
40
41                // attach panel to internal frame content pane
42                Container container = frame.getContentPane();
43                MyJPanel panel = new MyJPanel();
44                container.add( panel, BorderLayout.CENTER );
45
46                // set size internal frame to size of its contents
47                frame.pack();
48
49                // attach internal frame to desktop and show it
50                theDesktop.add( frame );
51                frame.setVisible( true );
52             }
53
54          } // end anonymous inner class
```

Handle event when user selects `JMenuItem`

Invoked when user selects `JMenuItem`

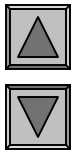
Line 35

Create `JInternalFrame`

`JPanel`s can be added to `JInternalFrame`s

Line 47

Use preferred size for window

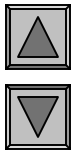


Outline

DesktopTest.java

```
55
56       ); // end call to addActionListener
57
58       setSize( 600, 460 );
59       setVisible( true );
60
61    } // end constructor
62
63    public static void main( String args[] )
64    {
65       DesktopTest application = new DesktopTest();
66       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
67    }
68
69 } // end class DesktopTest
70
71 // class to display an ImageIcon on a panel
72 class MyJPanel extends JPanel {
73    private ImageIcon imageIcon;
74    private String[] images = { "yellowflowers.png", "purpleflowers.png",
75       "redflowers.png", "redflowers2.png", "lavenderflowers.png" };
76
77    // load image
78    public MyJPanel()
79    {
```

```
80          int randomNumber = ( int ) ( Math.random() * 5 );
81          imageIcon = new ImageIcon( images[ randomNumber ] );
82       }
83
84       // display imageIcon on panel
85       public void paintComponent( Graphics g )
86       {
87          // call superclass paintComponent method
88          super.paintComponent( g );
89
90          // display icon
91          imageIcon.paintIcon( this, g, 0, 0 );
92       }
93
94       // return image dimensions
95       public Dimension getPreferredSize()
96       {
97          return new Dimension( imageIcon.getIconWidth(),
98             imageIcon.getIconHeight() );
99       }
100
101 } // end class MyJPanel
```

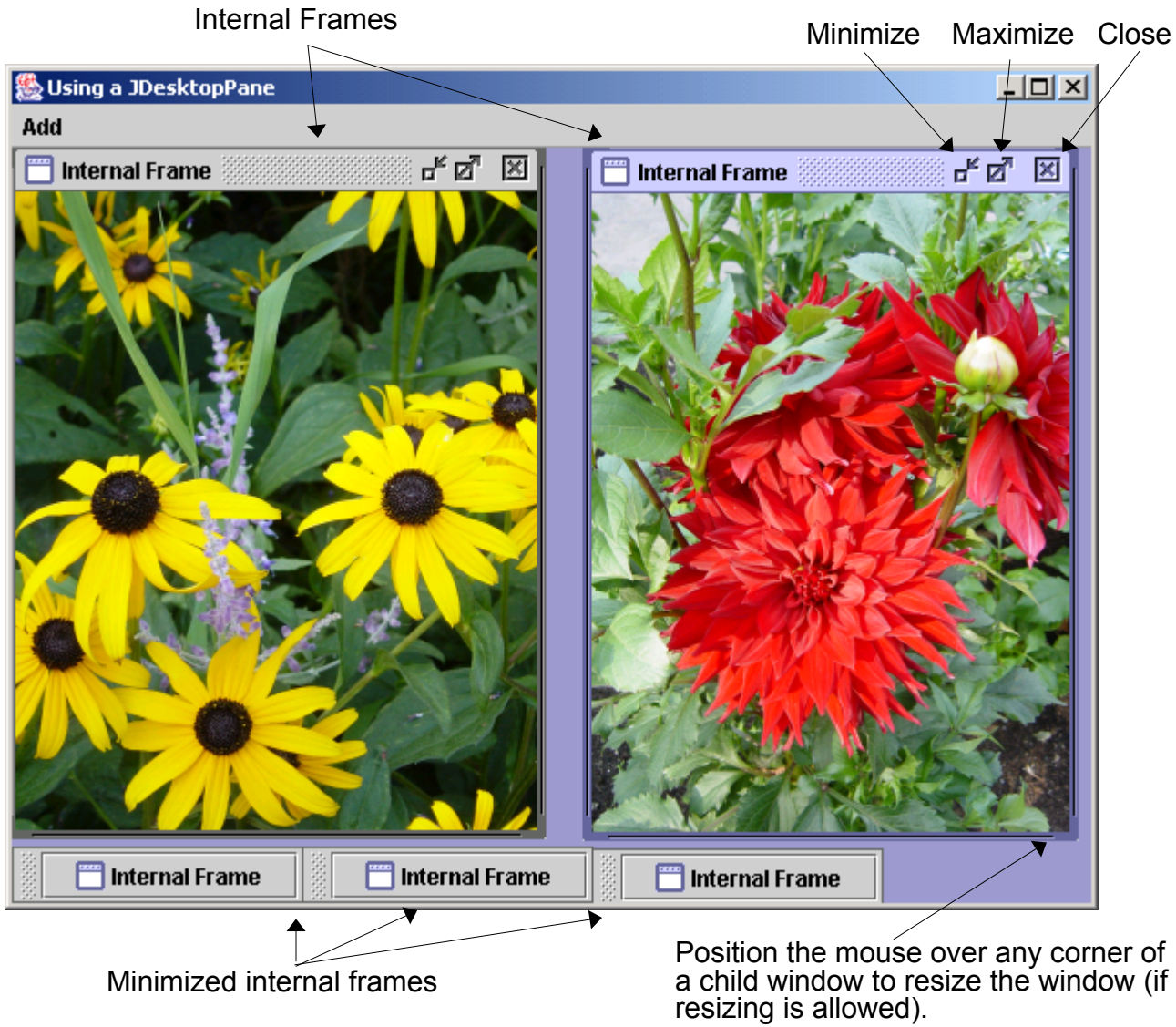


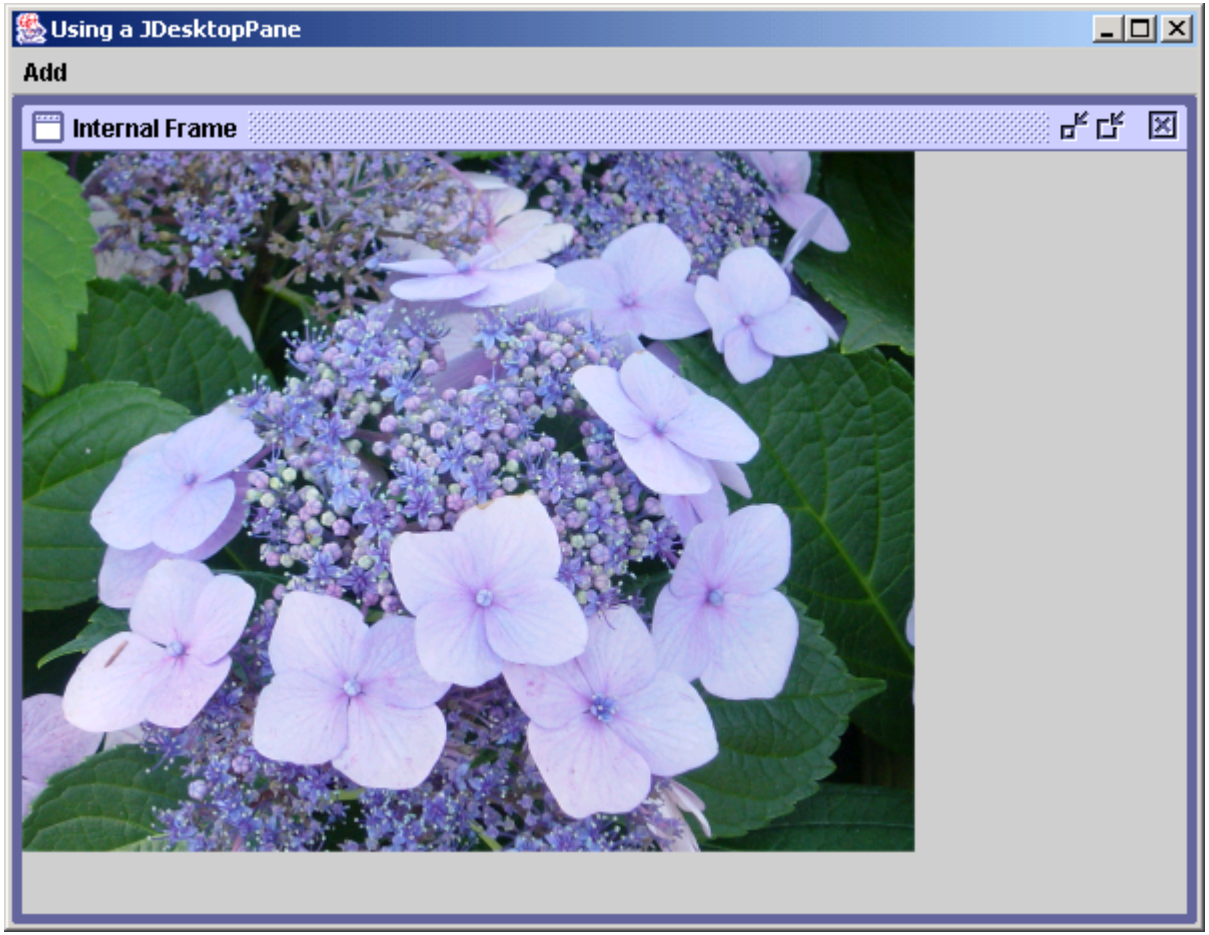
Outline

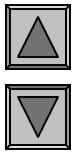
DesktopTest.java

Outline

DesktopTest.java





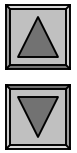


Outline

DesktopTest.java

14.11 JTabbedPane

- Arranges GUI components into layers
  - One layer visible at a time
  - Access each layer via a tab
  - `JTabbedPane`



Outline

JTabbedPaneDemo.java

Line 14

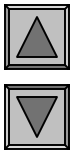
Line20

```
1  // Fig. 14.13: JTabbedPaneDemo.java
2  // Demonstrating JTabbedPane.
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class JTabbedPaneDemo extends JFrame  {
7
8     // set up GUI
9     public JTabbedPaneDemo()
10    {
11       super( "JTabbedPane Demo " );
12
13       // create JTabbedPane
14       JTabbedPane tabbedPane = new JTabbedPane();
15
16       // set up pane11 and add it to JTabbedPane
17       JLabel label1 = new JLabel( "panel one", SwingConstants.CENTER );
18       JPanel panel1 = new JPanel();
19       panel1.add( label1 );
20       tabbedPane.addTab( "Tab One", null, panel1, "First Panel" );
21
22       // set up panel2 and add it to JTabbedPane
23       JLabel label2 = new JLabel( "panel two", SwingConstants.CENTER );
24       JPanel panel2 = new JPanel();
25       panel2.setBackground( Color.YELLOW );
26       panel2.add( label2 );
27       tabbedPane.addTab( "Tab Two", null, panel2, "Second Panel" );
```

Create a JTabbedPane

Add the first panel

Add the second panel



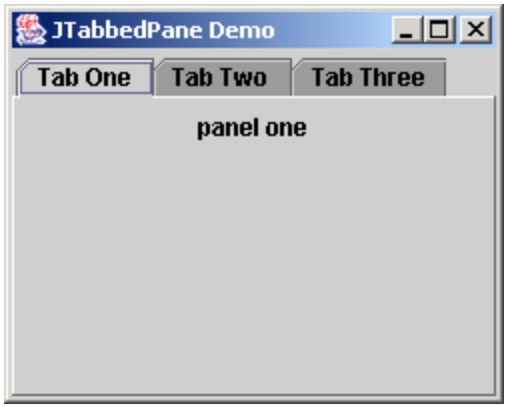
Outline

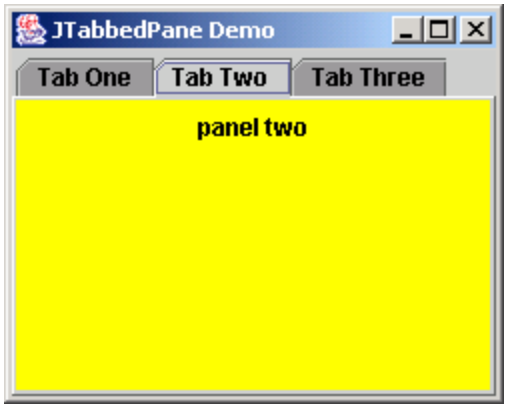
JTabbedPaneDemo.java

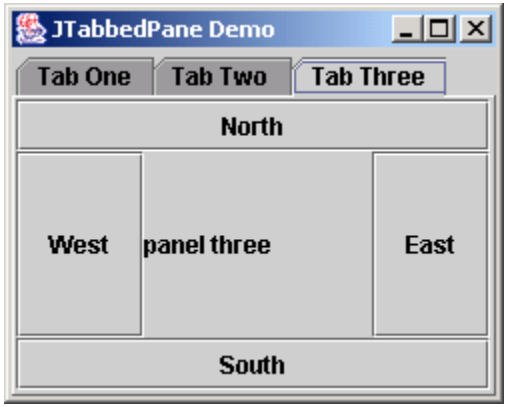
Line 38

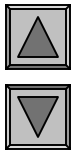
```
28
29       // set up panel3 and add it to JTabbedPane
30       JLabel label3 = new JLabel( "panel three" );
31       JPanel panel3 = new JPanel();
32       panel3.setLayout( new BorderLayout() );
33       panel3.add( new JButton( "North" ), BorderLayout.NORTH );
34       panel3.add( new JButton( "West" ), BorderLayout.WEST );
35       panel3.add( new JButton( "East" ), BorderLayout.EAST );
36       panel3.add( new JButton( "South" ), BorderLayout.SOUTH );
37       panel3.add( label3, BorderLayout.CENTER );
38       tabbedPane.addTab( "Tab Three", null, panel3, "Third Panel" );
39
40       // add JTabbedPane to container
41       getContentPane().add( tabbedPane );
42
43       setSize( 250, 200 );
44       setVisible( true );
45
46    } // end constructor
47
48    public static void main( String args[] )
49    {
50       JTabbedPaneDemo tabbedPaneDemo = new JTabbedPaneDemo();
51       tabbedPaneDemo.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
52    }
53
54 } // end class CardDeck
```

Add the third panel









Outline

`JTabbedPaneDemo.java`