Chapter 14 – Graphical User Components Part 2

Outline

Chapter 14 – Graphical User Components Part 2

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`

```
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 16

Lines 18-24

```java
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`

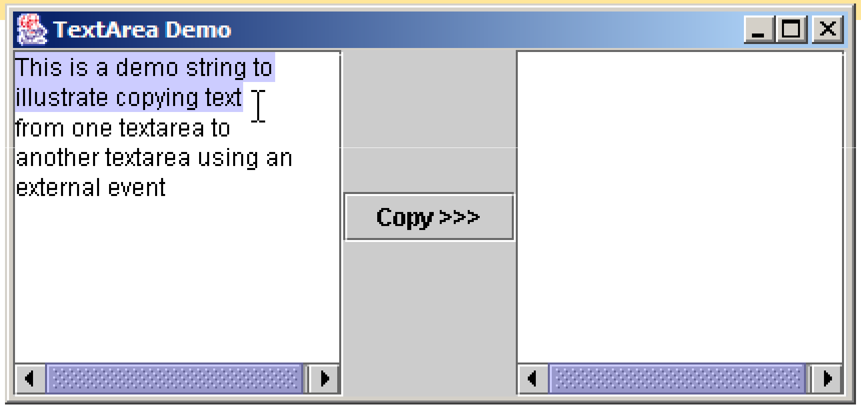
Outline

`TextAreaDemo.java`

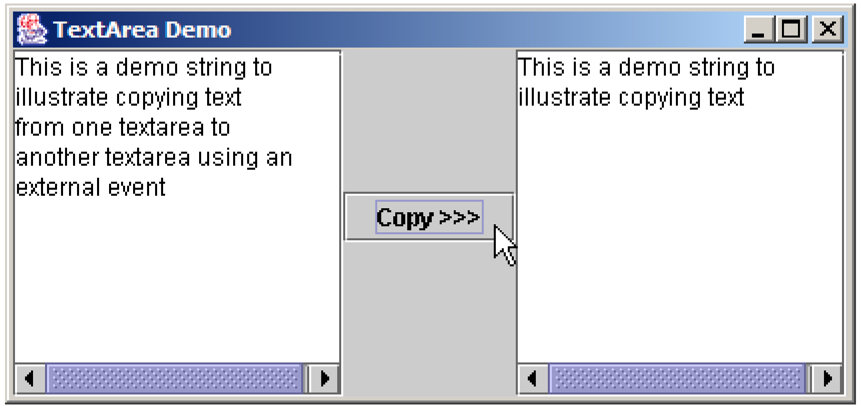
Line 36

Lines 44-45

```
      setSize( 425, 200 );
      setVisible( true );

   } // end constructor TextAreaDemo

   public static void main( String args[] )
   {
      TextAreaDemo application = new TextAreaDemo();
      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
   }

} // end class TextAreaDemo
```



This is a demo string to illustrate copying text from one textarea to another textarea using an external event



This is a demo string to illustrate copying text from one textarea to another textarea using an external event

This is a demo string to illustrate copying text

Outline

TextAreaDemo.java

14.3 Creating a Customized Subclass of `JPanel`

- Extend `JPanel` to create new components
  - Dedicated drawing area
    - Method `paintComponent` of class `JComponent`

Outline

CustomPanel.java

Line 11

Line 25

```
1  // Fig. 14.2: CustomPanel.java
2  // A customized JPanel class.
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class CustomPanel extends JPanel {
7     public final static int CIRCLE = 1, SQUARE = 2;
8     private int shape;
9
10    // use shape to draw an oval or rectangle
11    public void paintComponent( Graphics g )
12    {
13       super.paintComponent( g );
14
15       if ( shape == CIRCLE )
16          g.fillOval( 50, 10, 60, 60 );
17       else if ( shape == SQUARE )
18          g.fillRect( 50, 10, 60, 60 );
19    }
20
21    // set shape value and repaint CustomPanel
22    public void draw( int shapeToDraw )
23    {
24       shape = shapeToDraw;
25       repaint();
26 }
27
28 } // end class CustomPanel
```

Store integer representing shape to draw

Override method `paintComponent` of class `JComponent` to draw oval or rectangle

Method `repaint` calls method `paintComponent`

Outline

`CustomPanelTest.java`

Lines 18-19

```java
1  // Fig. 14.3: CustomPanelTest.java
2  // Using a customized Panel object.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class CustomPanelTest extends JFrame {
8     private JPanel buttonPanel;
9     private CustomPanel myPanel;
10    private JButton circleButton, squareButton;
11
12    // set up GUI
13    public CustomPanelTest()
14    {
15       super( "CustomPanel Test" );
16
17       // create custom drawing area
18       myPanel = new CustomPanel();
19       myPanel.setBackground( Color.GREEN );
20
21       // set up squareButton
22       squareButton = new JButton( "Square" );
23       squareButton.addActionListener(
24
```

Instantiate `CustomPanel` object and set background to green

```java
         new ActionListener() {  // anonymous inner class

            // draw a square
            public void actionPerformed( ActionEvent event )
            {
               myPanel.draw( CustomPanel.SQUARE );
            }

         } // end anonymous inner class

      ); // end call to addActionListener

      circleButton = new JButton( "Circle" );
      circleButton.addActionListener(

         new ActionListener() {  // anonymous inner class

            // draw a circle
            public void actionPerformed( ActionEvent event )
            {
               myPanel.draw( CustomPanel.CIRCLE );
            }

         } // end anonymous inner class

      ); // end call to addActionListener

```

When user presses `squareButton`, draw `square` on `CustomPanel`

When user presses `circleButton`, draw `circle` on `CustomPanel`

Outline

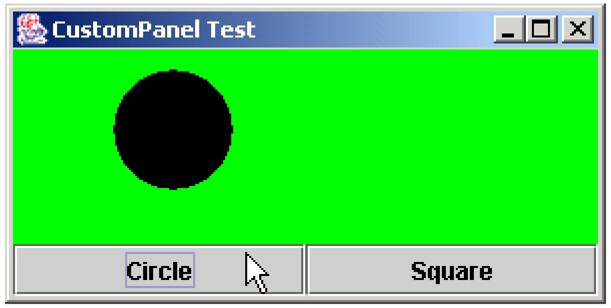
`CustomPanelTest.java`

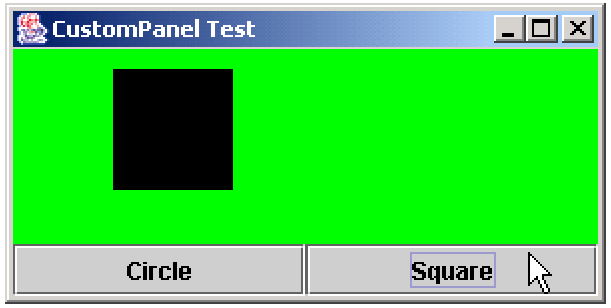
Outline

```
52         // set up panel containing buttons
53         buttonPanel = new JPanel();
54         buttonPanel.setLayout( new GridLayout( 1, 2 ) );
55         buttonPanel.add( circleButton );
56         buttonPanel.add( squareButton );
57
58         // attach button panel & custom drawing area to content pane
59         Container container = getContentPane();
60         container.add( myPanel, BorderLayout.CENTER );
61         container.add( buttonPanel, BorderLayout.SOUTH );
62
63         setSize( 300, 150 );
64         setVisible( true );
65
66      } // end constructor CustomPanelTest
67
68      public static void main( String args[] )
69      {
70         CustomPanelTest application = new CustomPanelTest();
71         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
72      }
73
74   } // end class CustomPanelTest
```

Use `GridLayout` to organize buttons

Line 54





14.4 `JPanel` Subclass that Handles Its Own Events

- `JPanel`
  - Does not support conventional events
    - e.g., events offered by buttons, text areas, etc.
  - Capable of recognizing lower-level events
    - e.g., mouse events, key events, etc.
  - Self-contained panel
    - Listens for its own mouse events

```
1  // Fig. 14.4: SelfContainedPanel.java
2  // A self-contained JPanel class that handles its own mouse events.
3  package com.deitel.jhtp5.ch14;
4
5  import java.awt.*;
6  import java.awt.event.*;
7  import javax.swing.*;
8
9  public class SelfContainedPanel extends JPanel {
10    private int x1, y1, x2, y2;
11
12    // set up mouse event handling for SelfContainedPanel
13    public SelfContainedPanel()
14    {
15       // set up mouse listener
16       addMouseListener(
17
18          new MouseAdapter() {  // anonymous inner class
19
20             // handle mouse press event
21             public void mousePressed( MouseEvent event )
22             {
23                x1 = event.getX();
24                y1 = event.getY();
25             }
26
```

Self-contained `JPanel` listens for `MouseEvents`

Save coordinates where user pressed mouse button

Outline

`SelfContainedPanel.java`

Line 16

Lines 23-24

```
27               // handle mouse release event
28               public void mouseReleased( MouseEvent event )
29               {
30                  x2 = event.getX();
31                  y2 = event.getY();
32                  repaint();
33               }
34
35            } // end anonymous inner class
36
37         ); // end call to addMouseListener
38
39         // set up mouse motion listener
40         addMouseMotionListener(
41
42            new MouseMotionAdapter() {  // anonymous inner class
43
44               // handle mouse drag event
45               public void mouseDragged( MouseEvent event )
46               {
47                  x2 = event.getX();
48                  y2 = event.getY();
49                  repaint();
50               }
51
```

Save coordinates where user released mouse button, then repaint

Self-contained `JPanel` listens for when mouse moves

Save coordinates where user dragged mouse, then repaint

Outline

nedPa

Lines 30-31

Line 40

47-48

```java
52          } // end anonymous inner class
53
54       ); // end call to addMouseMotionListener
55
56    } // end constructor SelfContainedPanel
57
58    // return preferred width and height of SelfContainedPanel
59    public Dimension getPreferredSize()
60    {
61       return new Dimension( 150, 100 );
62    }
63
64    // paint an oval at the specified coordinates
65    public void paintComponent( Graphics g )
66    {
67       super.paintComponent( g );
68
69       g.drawOval( Math.min( x1, x2 ), Math.min( y1, y2 ),
70          Math.abs( x1 - x2 ), Math.abs( y1 - y2 ) );
71    }
72
73 } // end class SelfContainedPanel
```

Draw oval

Outline

SelfContainedPanel.java

Lines 69-70

```
1  // Fig. 14.5: SelfContainedPanelTest.java
2  // Creating a self-contained subclass of JPanel that processes
3  // its own mouse events.
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  import com.deitel.jhtp5.ch14.SelfContainedPanel;
9
10 public class SelfContainedPanelTest extends JFrame {
11    private SelfContainedPanel myPanel;
12
13    // set up GUI and mouse motion event handlers for application window
14    public SelfContainedPanelTest()
15    {
16       // set up a SelfContainedPanel
17       myPanel = new SelfContainedPanel();
18       myPanel.setBackground( Color.YELLOW );
19
20       Container container = getContentPane();
21       container.setLayout( new FlowLayout() );
22       container.add( myPanel );
23
```

Instantiate `SelfContaintedPanel` object and set background to yellow

Outline

`SelfContainedPanelTest.java`

Lines 17-18

```java
24        // set up mouse motion event handling
25        addMouseMotionListener(
26
27           new MouseMotionListener() {  // anonymous inner class
28
29              // handle mouse drag event
30              public void mouseDragged( MouseEvent event )
31              {
32                 setTitle( "Dragging: x=" + event.getX() +
33                    "; y=" + event.getY() );
34              }
35
36              // handle mouse move event
37              public void mouseMoved( MouseEvent event )
38              {
39                 setTitle( "Moving: x=" + event.getX() +
40                    "; y=" + event.getY() );
41              }
42
43           } // end anonymous inner class
44
45        ); // end call to addMouseMotionListener
46
47        setSize( 300, 200 );
48        setVisible( true );
49
50     } // end constructor SelfContainedPanelTest
```

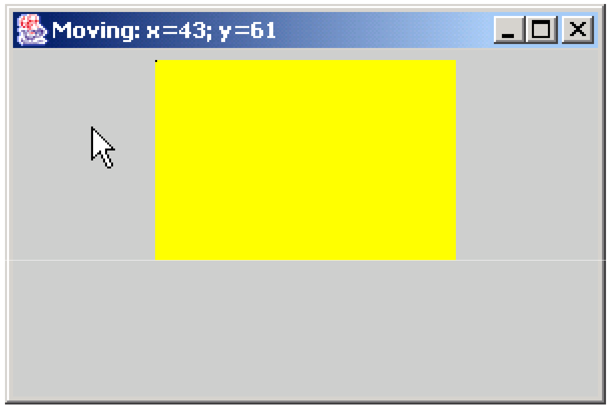
Register anonymous-inner-class object to handle mouse motion events

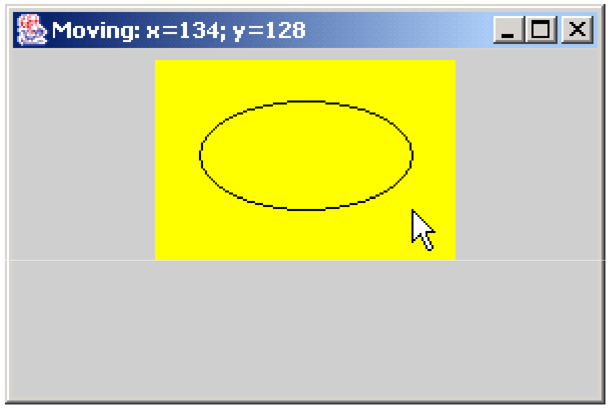
Display `String` in title bar indicating *x-y* coordinate where mouse-motion event occurred

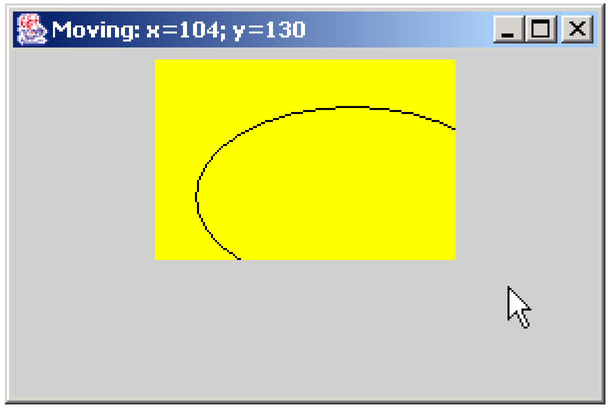
`SelfContainedPanelTest.java`

Line 25

```java

    public static void main( String args[] )
    {
       SelfContainedPanelTest application = new SelfContainedPanelTest();
       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    }

} // end class SelfContainedPanelTest
```







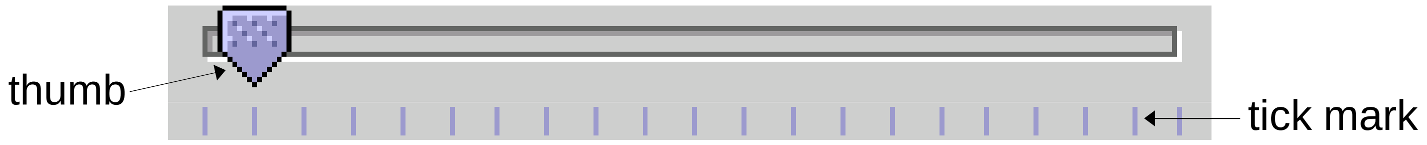
Outline

SelfContainedPanelTest.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



```
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`

Line 14

Line 18

```java
    // used by layout manager to determine preferred size
    public Dimension getPreferredSize()
    {
       return new Dimension( 200, 200 );
    }

    // used by layout manager to determine minimum size
    public Dimension getMinimumSize()
    {
       return getPreferredSize();
    }

 } // end class OvalPanel
```

Outline

`OvalPanel.java`

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`

Outline

SliderDemo.java

Line 28

Line 35

```
27       // register JSlider event listener
28       diameterSlider.addChangeListener(
29 
30          new ChangeListener() {  // anonymous inner cla
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 
```

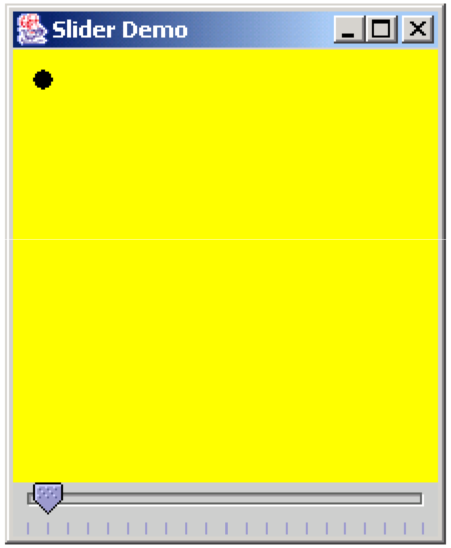
Register anonymous `ChangeListener` object to handle `JSlider` events

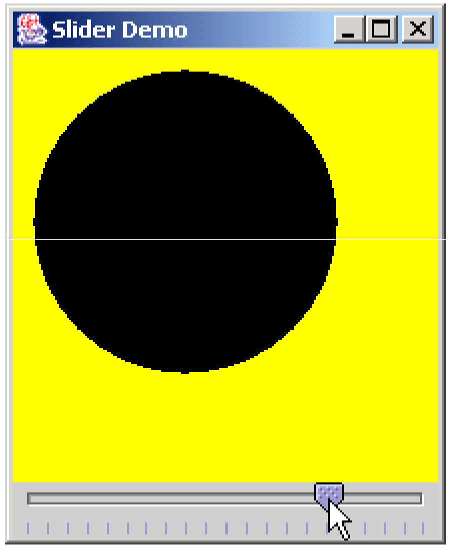
When user accesses `JSlider`, set `OvalPanel`'s `diameter` according to `JSlider` value

```
    public static void main( String args[] )
    {
       SliderDemo application = new SliderDemo();
       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    }

 } // 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`

```
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

`uTest.java`

Line 26

Lines 36-38

Line 46

Outline

MenuTest.java

Line 56

Line 64

Line 69

```java
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 `JMenus`

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

utline

`MenuTest.java`

Line 75

Lines 78-79

```java
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

```java
      // create style checkbox menu items
      for ( int count = 0; count < styleNames.length; count++ ) {
         styleItems[ count ] =
            new JCheckBoxMenuItem( styleNames[ count ] );
         fontMenu.add( styleItems[ count ] );
         styleItems[ count ].addItemListener( styleHandler );
      }

      // put Font menu in Format menu
      formatMenu.add( fontMenu );

      // add Format menu to menu bar
      bar.add( formatMenu );

      // set up label to display text
      displayLabel = new JLabel( "Sample Text", SwingConstants.CENTER );
      displayLabel.setForeground( colorValues[ 0 ] );
      displayLabel.setFont( new Font( "Serif", Font.PLAIN, 72 ) );

      getContentPane().setBackground( Color.CYAN );
      getContentPane().add( displayLabel, BorderLayout.CENTER );

      setSize( 500, 200 );
      setVisible( true );

   } // end constructor

```

```
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`

Line 163

Lines 168 and 176

d 177-

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 ) );
```

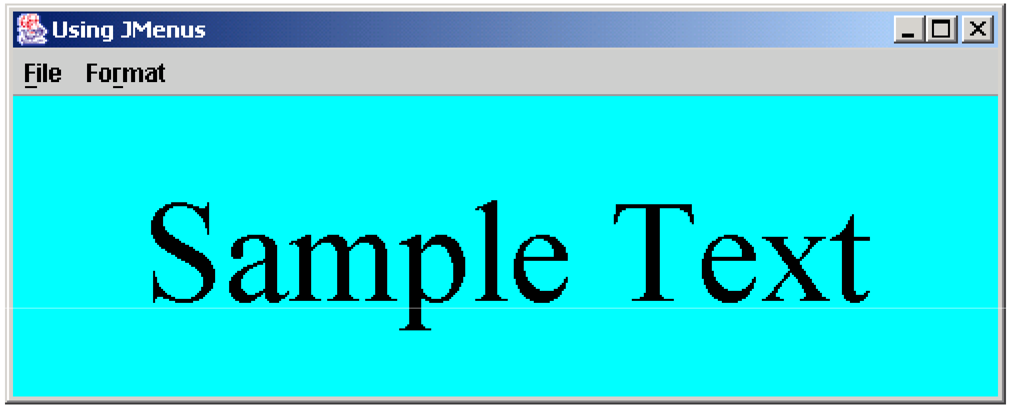
Invoked when user selects `JCheckBoxMenuItem`

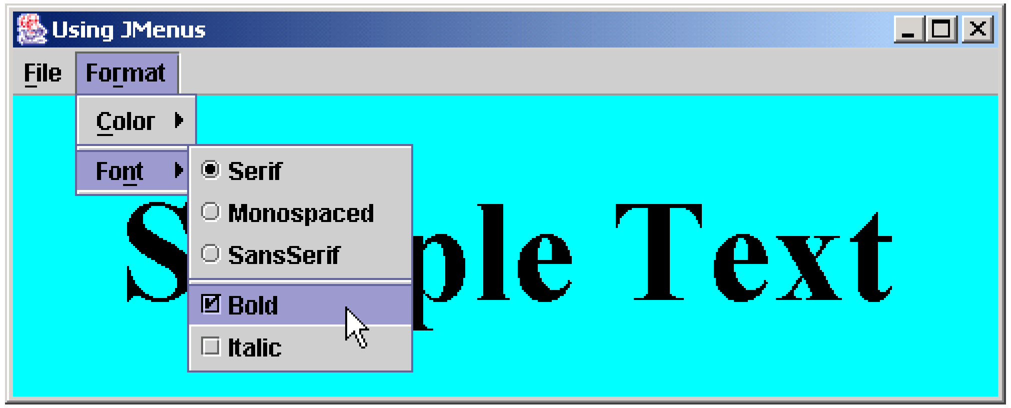
Determine new font style

```

            repaint();
         }

      } // end class StyleHandler

} // end class MenuTest
```

Outline

MenuTest.java





14.8 `JPopupMenu`

- Context-sensitive popup menus
  - `JPopupMenu`
  - Menu generated depending on which component is accessed

```
1  // Fig. 14.10: PopupTest.java
2  // Demonstrating JPopupMenus
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class PopupTest extends JFrame {
8     private JRadioButtonMenuItem items[];
9     private final Color colorValues[] =
10       { Color.BLUE, Color.YELLOW, Color.RED };
11    private JPopupMenu popupMenu;
12
13    // set up GUI
14    public PopupTest()
15    {
16       super( "Using JPopupMenus" );
17
18       ItemHandler handler = new ItemHandler();
19       String colors[] = { "Blue", "Yellow", "Red" };
20
21       // set up popup menu and its items
22       ButtonGroup colorGroup = new ButtonGroup();
23       popupMenu = new JPopupMenu();
24       items = new JRadioButtonMenuItem[ 3 ];
25
```

Instantiate `JPopupMenu` object

Outline

`PopupTest.java`

Line 23

```
26        // construct each menu item and add to popup menu; also
27        // enable event handling for each menu item
28        for ( int count = 0; count < items.length; count++ ) {
29           items[ count ] = new JRadioButtonMenuItem( colors[ count ] );
30           popupMenu.add( items[ count ] );
31           colorGroup.add( items[ count ] );
32           items[ count ].addActionListener( handler );
33        }
34
35        getContentPane().setBackground( Color.WHITE );
36
37        // declare a MouseListener for the window that displays
38        // a JPopupMenu when the popup trigger event occurs
39        addMouseListener(
40
41           new MouseAdapter() {  // anonymous inner class
42
43              // handle mouse press event
44              public void mousePressed( MouseEvent event )
45              {
46                 checkForTriggerEvent( event );
47              }
48
49              // handle mouse release event
50              public void mouseReleased( MouseEvent event )
51              {
52                 checkForTriggerEvent( event );
53              }
```

Create `JRadioButtonMenuItem` objects to add to `JPopupMenu`

Determine whether popup-trigger event occurred when user presses or releases mouse button

Outline

`PopupTest.java`

Lines 46 and 52

PopupTest.java

Lines 59-60

```
54
55             // determine whether event should trigger popup menu
56             private void checkForTriggerEvent( MouseEvent event )
57             {
58                if ( event.isPopupTrigger() )
59                   popupMenu.show(
60                      event.getComponent(), event.getX(), event.getY() );
61             }
62
63          } // end anonymous inner clas
64
65       ); // end call to addMouseListener
66
67       setSize( 300, 200 );
68       setVisible( true );
69
70    } // end constructor PopupTest
71
72    public static void main( String args[] )
73    {
74       PopupTest application = new PopupTest();
75       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
76    }
77
```

Show `JPopupMenu` if popup-trigger occurred

Outline

va

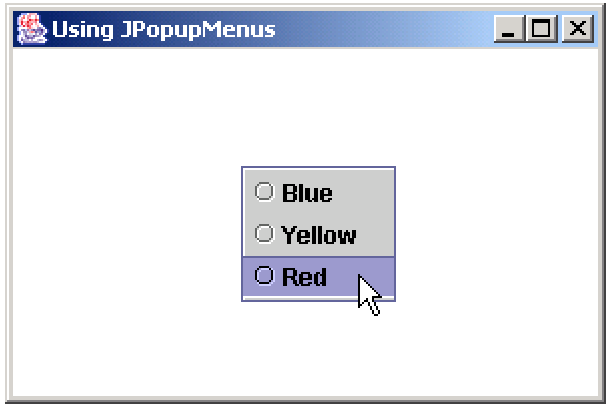
Line 82

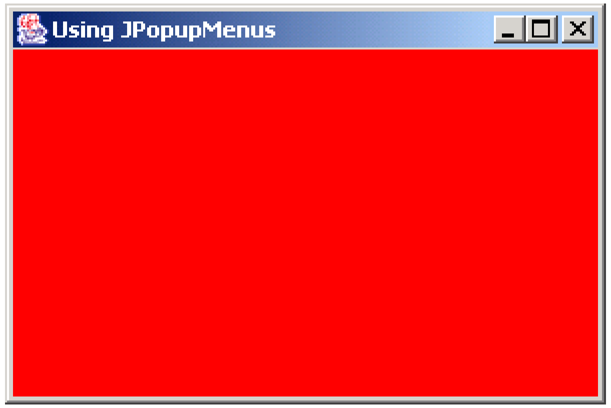
Line 87

```
78     // private inner class to handle menu item events
79     private class ItemHandler implements ActionListener {
80
81        // process menu item selections
82        public void actionPerformed( ActionEvent event )
83        {
84           // determine which menu item was selected
85           for ( int i = 0; i < items.length; i++ )
86              if ( event.getSource() == items[ i ] ) {
87                 getContentPane().setBackground( colorValues[ i ] );
88                 return;
89              }
90        }
91
92     } // end private inner class ItemHandler
93
94  } // end class PopupTest
```

Invoked when user selects `JRadioButtonMenuItem`

Determine which `JRadioButtonMenuItem` was selected, then set window background color





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

```
      // set up label for NORTH panel
      label = new JLabel( "This is a Metal look-and-feel",
         SwingConstants.CENTER );
      northPanel.add( label );

      // set up button for NORTH panel
      button = new JButton( "JButton" );
      northPanel.add( button );

      // set up combo box for NORTH panel
      comboBox = new JComboBox( strings );
      northPanel.add( comboBox );

      // create array for radio buttons
      radio = new JRadioButton[ strings.length ];

      // set up panel for SOUTH of BorderLayout
      JPanel southPanel = new JPanel();
      southPanel.setLayout( new GridLayout( 1, radio.length ) );

      // set up radio buttons for SOUTH panel
      group = new ButtonGroup();
      ItemHandler handler = new ItemHandler();

```

Outline

LookAndFeelDemo
.java

Outline

LookAndFeelDemo
.java

```
      for ( int count = 0; count < radio.length; count++ ) {
         radio[ count ] = new JRadioButton( strings[ count ] );
         radio[ count ].addItemListener( handler );
         group.add( radio[ count ] );
         southPanel.add( radio[ count ] );
      }

      // attach NORTH and SOUTH panels to content pane
      container.add( northPanel, BorderLayout.NORTH );
      container.add( southPanel, BorderLayout.SOUTH );

      // get installed look-and-feel information
      looks = UIManager.getInstalledLookAndFeels();

      setSize( 300, 200 );
      setVisible( true );

      radio[ 0 ].setSelected( true );

   } // end constructor LookAndFeelDemo

   // use UIManager to change look-and-feel of GUI
   private void changeTheLookAndFeel( int value )
   {
```

```
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
```

Outline

LookAndFeelDemo.java

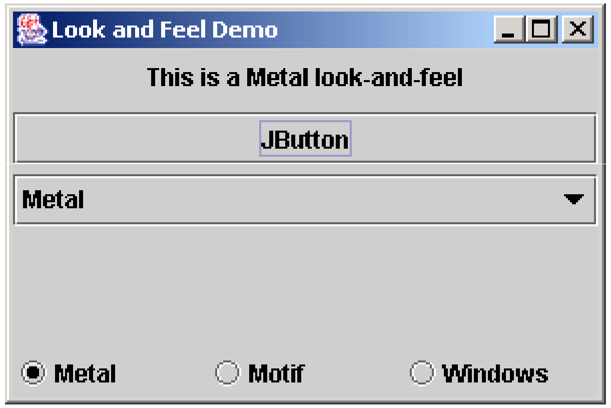
Change look-and-feel

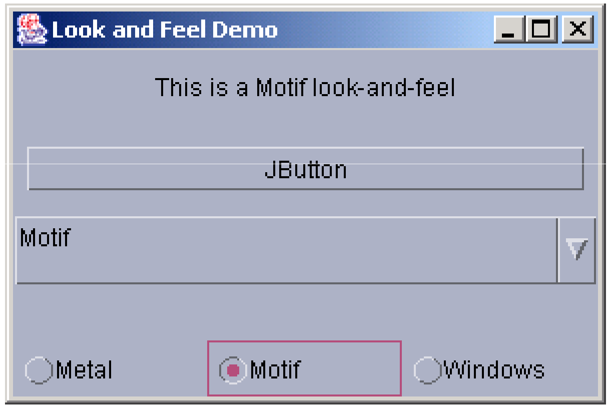
Lines 77-78

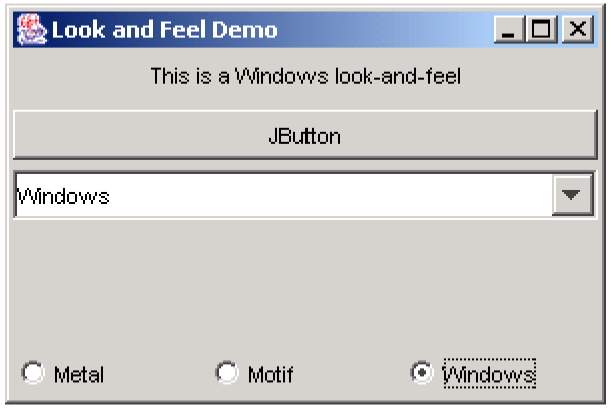
```
            if ( radio[ count ].isSelected() ) {
               label.setText( "This is a " +
                  strings[ count ] + " look-and-feel" );
               comboBox.setSelectedIndex( count );
               changeTheLookAndFeel( count );
            }
      }

   } // end private inner class ItemHandler

} // end class LookAndFeelDemo
```

Outline

LookAndFeelDemo.java







14.10 `JDesktopPane` and `JInternalFrame`

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

Outline

DesktopTest.java

```java
// Fig. 14.12: DesktopTest.java
// Demonstrating JDesktopPane.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class DesktopTest extends JFrame {
   private JDesktopPane theDesktop;

   // set up GUI
   public DesktopTest()
   {
      super( "Using a JDesktopPane" );

      // create menu bar, menu and menu item
      JMenuBar bar = new JMenuBar();
      JMenu addMenu = new JMenu( "Add" );
      JMenuItem newFrame = new JMenuItem( "Internal Frame" );

      addMenu.add( newFrame );
      bar.add( addMenu );

      setJMenuBar( bar );

      // set up desktop
      theDesktop = new JDesktopPane();
      getContentPane().add( theDesktop );
```

Manages `JInternalFrame` child windows displayed in `JDesktopPane`

```
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`

Create `JInternalFrame`

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

Use preferred size for window

Outline

`DesktopTest.java`

Line 35

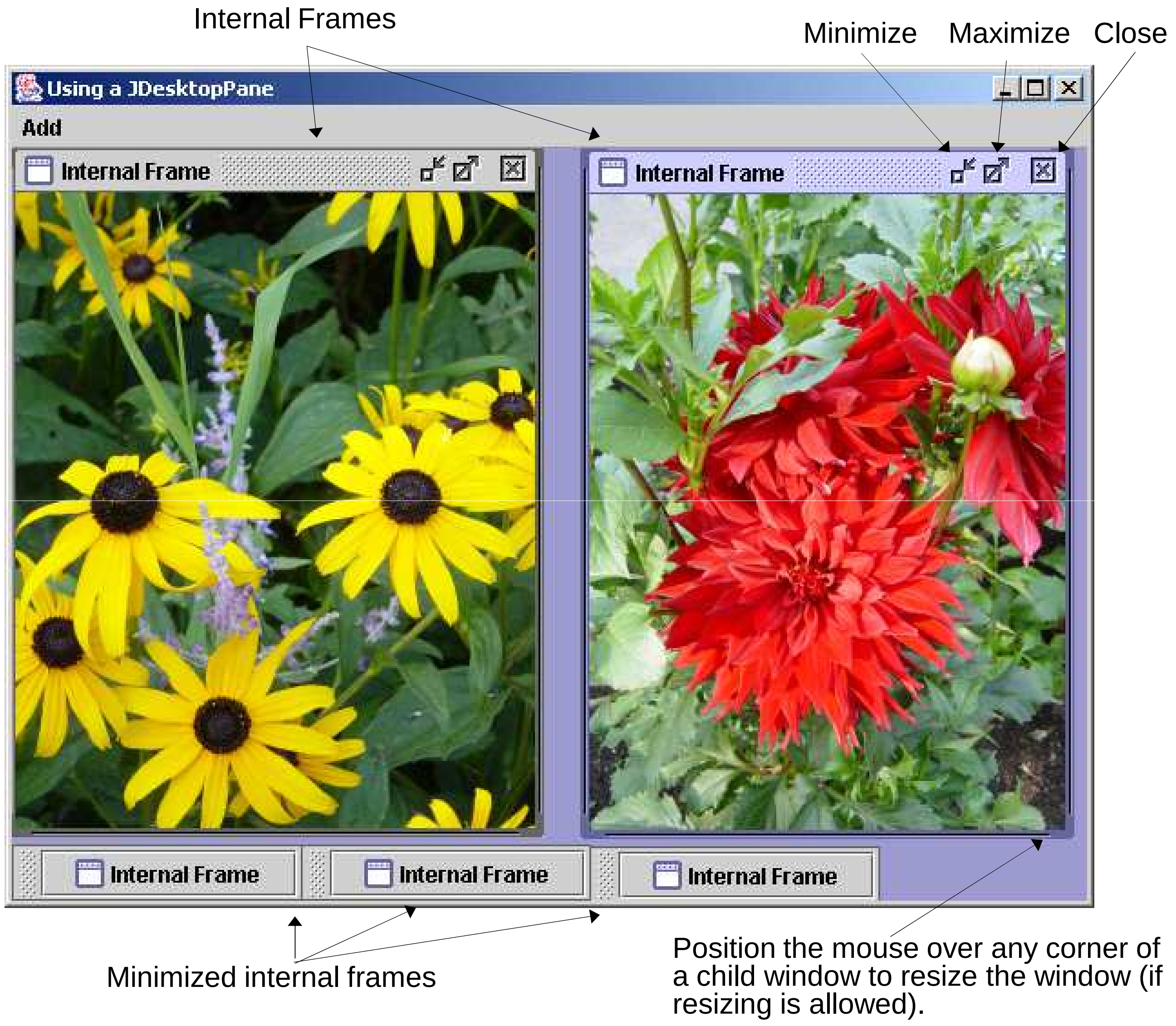
Line 47

```

      ); // end call to addActionListener

      setSize( 600, 460 );
      setVisible( true );

   } // end constructor

   public static void main( String args[] )
   {
      DesktopTest application = new DesktopTest();
      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
   }

} // end class DesktopTest

// class to display an ImageIcon on a panel
class MyJPanel extends JPanel {
   private ImageIcon imageIcon;
   private String[] images = { "yellowflowers.png", "purpleflowers.png",
      "redflowers.png", "redflowers2.png", "lavenderflowers.png" };

   // load image
   public MyJPanel()
   {
```

DesktopTest.java

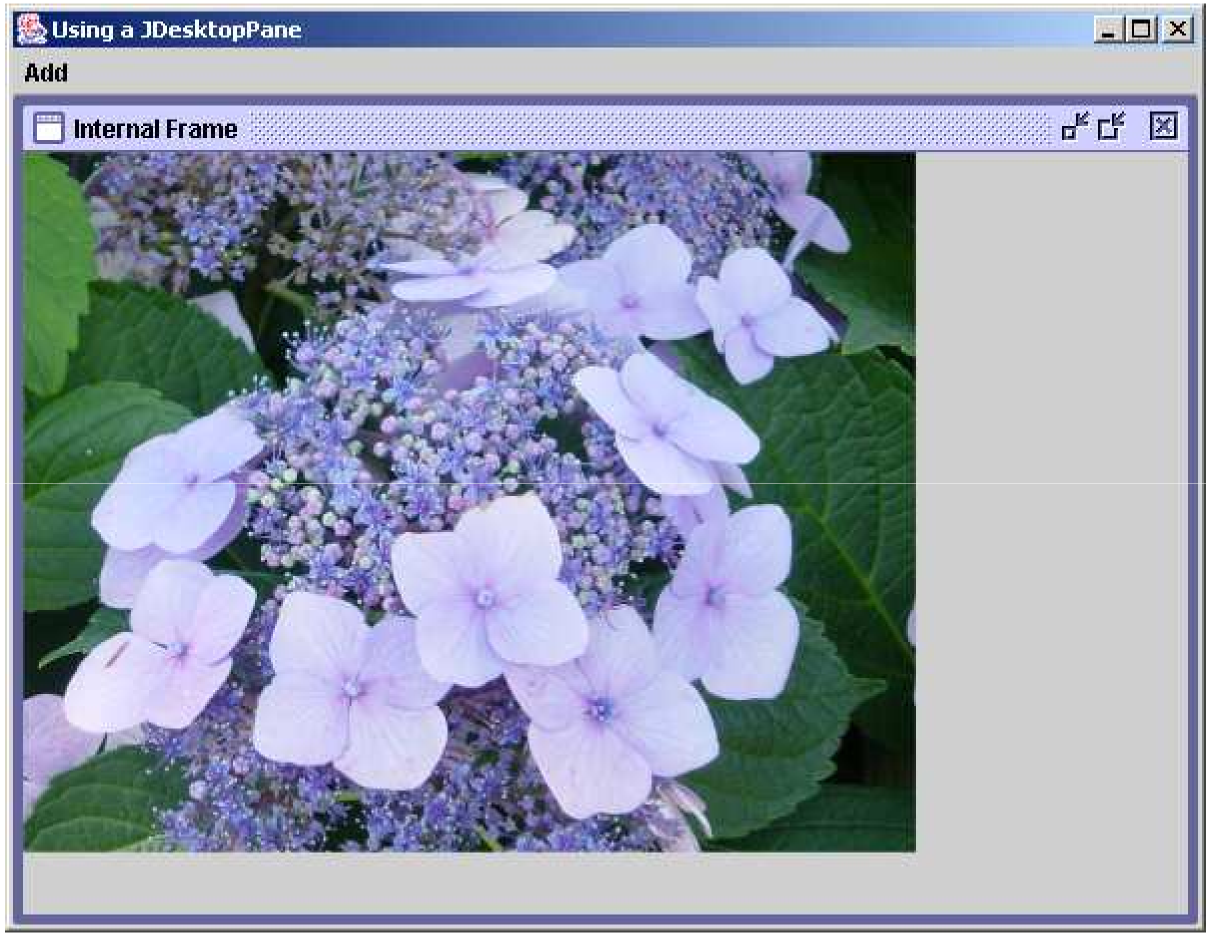
```
         int randomNumber = ( int ) ( Math.random() * 5 );
         imageIcon = new ImageIcon( images[ randomNumber ] );
      }

      // display imageIcon on panel
      public void paintComponent( Graphics g )
      {
         // call superclass paintComponent method
         super.paintComponent( g );

         // display icon
         imageIcon.paintIcon( this, g, 0, 0 );
      }

      // return image dimensions
      public Dimension getPreferredSize()
      {
         return new Dimension( imageIcon.getIconWidth(),
            imageIcon.getIconHeight() );
      }

   } // end class MyJPanel
```

`DesktopTest.java`


Internal Frames
Minimize
Maximize
Close
Using a JDesktopPane
Add
Internal Frame
Internal Frame
Internal Frame
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).

Outline

DesktopTest.java



14.11 `JTabbedPane`

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

```
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 panel1 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 14

Line20

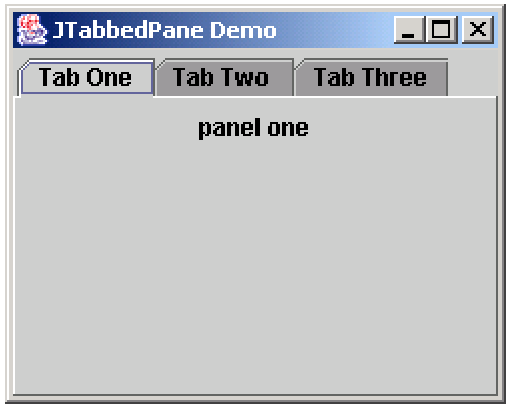
```
28
29         // set up panel3 and add it to JTabbedPane
30         JLabel label3 = new JLabel( "panel three" );
31         JPanel panel3 = new JPanel();
32         panel3.setLayout( new BorderLayout() );
33         panel3.add( new JButton( "North" ), BorderLayout.NORTH );
34         panel3.add( 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
```

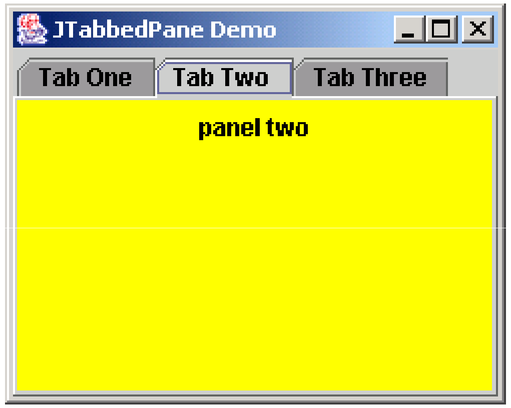
Add the third panel

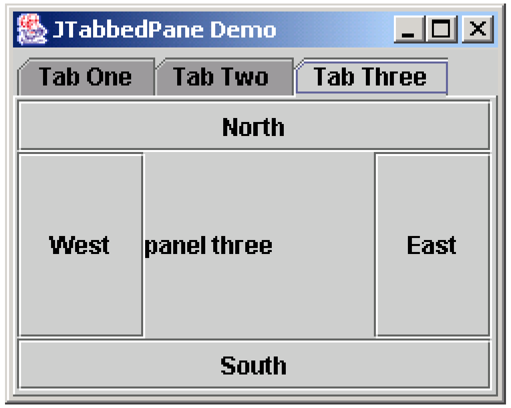
Outline

JTabbedPaneDemo.java

Line 38







`JTabbedPaneDemo.java`

14.12 Layout Managers: `BoxLayout` and `GridBagLayout`

- Layout Managers
  - `BoxLayout`
  - `GridBagLayout`

Fig. 14.14 Additional layout managers

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

`BoxLayout` Layout Manager

- `BoxLayout`
  - Arranges GUI components
    - Horizontally along x-axis
    - Vertically along y-axis

```
1  // Fig. 14.15: BoxLayoutDemo.java
2  // Demonstrating BoxLayout.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class BoxLayoutDemo extends JFrame {
8
9     // set up GUI
10    public BoxLayoutDemo()
11    {
12       super( "Demostrating BoxLayout" );
13
14       // create Box containers with BoxLayout
15       Box horizontal1 = Box.createHorizontalBox();
16       Box vertical1 = Box.createVerticalBox();
17       Box horizontal2 = Box.createHorizontalBox();
18       Box vertical2 = Box.createVerticalBox();
19
20       final int SIZE = 3; // number of buttons on each Box
21
22       // add buttons to Box horizontal1
23       for ( int count = 0; count < SIZE; count++ )
24          horizontal1.add( new JButton( "Button " + count ) );
25
```

Outline

BoxLayoutDemo.java

Lines 15-18

Line 24

Create Boxes

Add three JButtons to horizontal Box

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

Add three `JButton`s to vertical `Box`

Strut guarantees space between components

tDemo.j
ava

Add three `JButton`s to horizontal `Box`

Glue guarantees expandable space between components

Lines 33-36

Add three `JButton`s to vertical `Box`

Rigid area guarantees fixed component size

-42

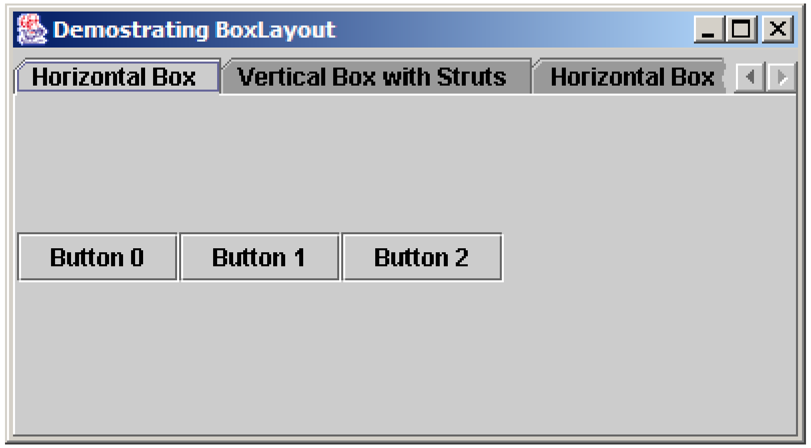
Line 40

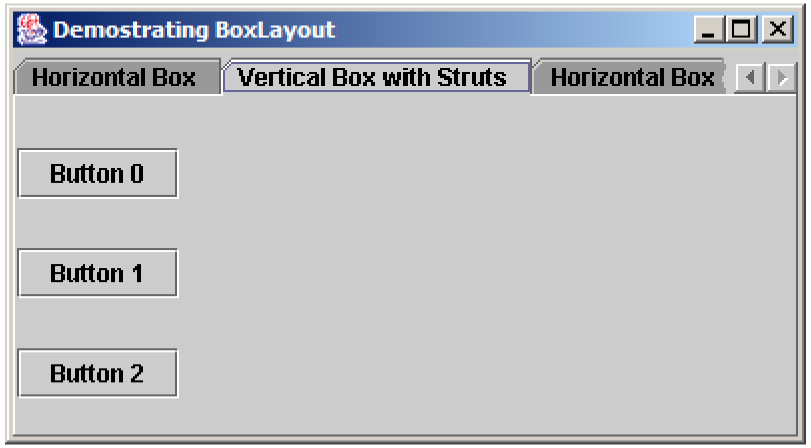
```
53         // create a JTabbedPane
54         JTabbedPane tabs = new JTabbedPane(
55            JTabbedPane.TOP, JTabbedPane.SCROLL_TAB_LAYOUT );
56
57         // place each container on tabbed pane
58         tabs.addTab( "Horizontal Box", horizontal1 );
59         tabs.addTab( "Vertical Box with Struts", vertical1 );
60         tabs.addTab( "Horizontal Box with Glue", horizontal2 );
61         tabs.addTab( "Vertical Box with Rigid Areas", vertical2 );
62         tabs.addTab( "Vertical Box with Glue", panel );
63
64         getContentPane().add( tabs );  // place tabbed pane on content pane
65
66         setSize( 400, 220 );
67         setVisible( true );
68
69      } // end constructor
70
71      public static void main( String args[] )
72      {
73         BoxLayoutDemo application = new BoxLayoutDemo();
74         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
75      }
76
77   } // end class BoxLayoutDemo
```

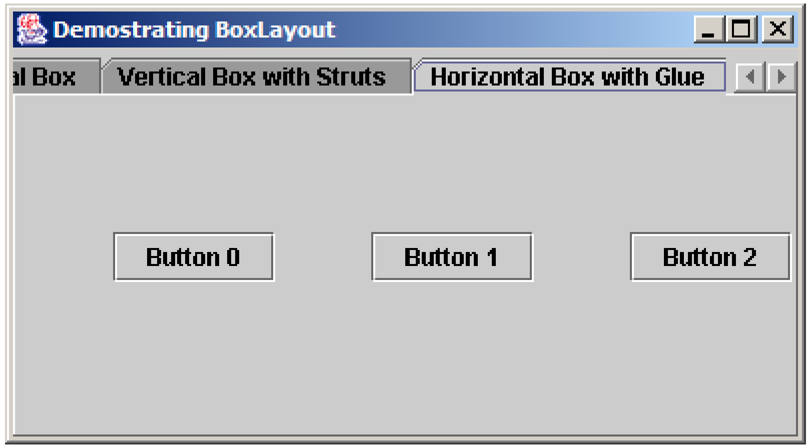
Create a JTabbedPane to hold the Boxes

BoxLayoutDemo.java

Lines 54-55





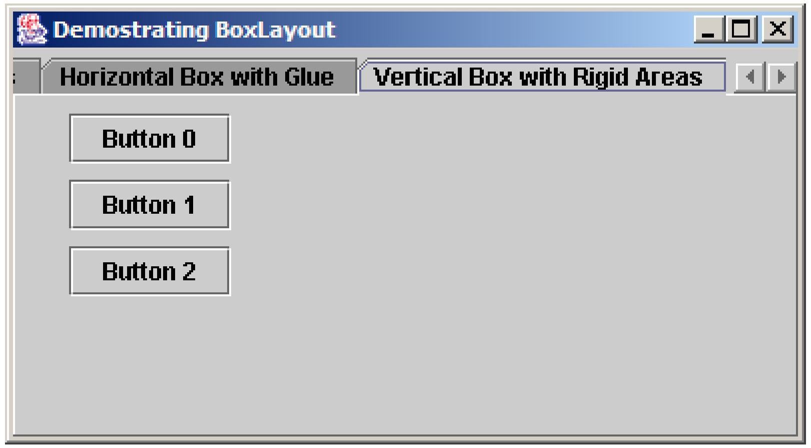


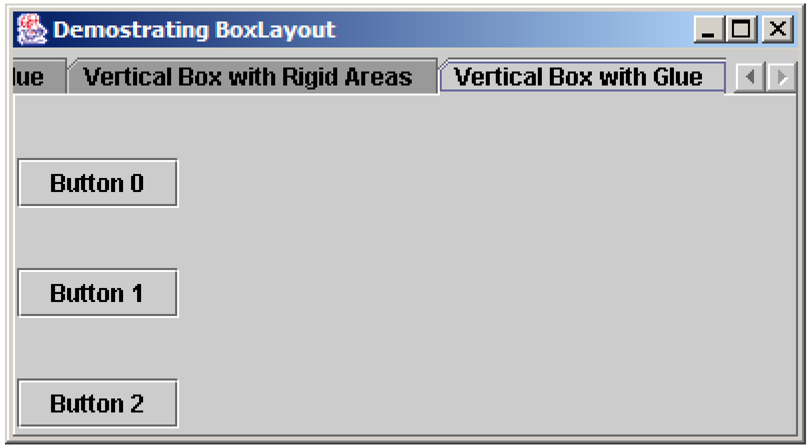
Outline

BoxLayoutDemo.java

Outline

`BoxLayoutDemo.java`





GridBagLayout Layout Manager

- GridBagLayout
  - Flexible GridBagLayout
    - Components can vary in size
    - Components can occupy multiple rows and columns
    - Components can be added in any order
  - Uses GridBagConstraints
    - Specifies how component is placed in GridBagLayout

Fig. 14.16 Designing a GUI that will use `GridBagLayout`

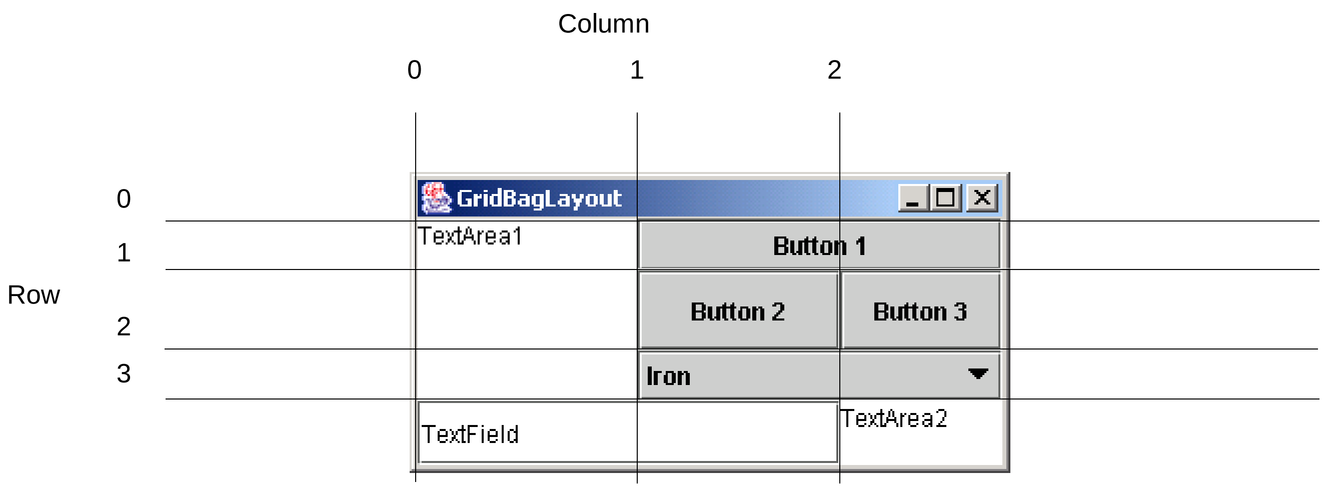
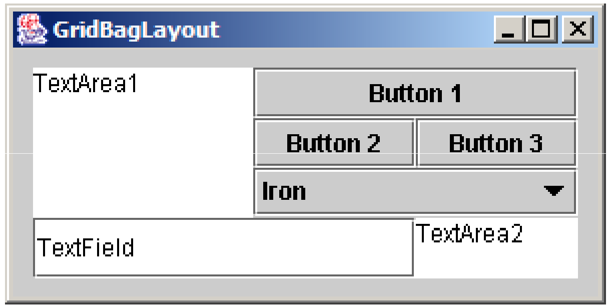


Fig. 14.17 `GridBagConstraints` fields

`GridBagConstraints` field	Description
`fill`	Resize the component in specified direction (`NONE`, `HORIZONTAL`, `VERTICAL`, `BOTH`) when the display area is larger than the component.
`gridx`	The column in which the component will be placed.
`gridy`	The row in which the component will be placed.
`gridwidth`	The number of columns the component occupies.
`gridheight`	The number of rows the component occupies.
`weightx`	The portion of extra space to allocate horizontally. The grid slot can become wider when extra space is available.
`weighty`	The portion of extra space to allocate vertically. The grid slot can become taller when extra space is available.

Fig. 14.18 `GridBagLayout` with the weights set to zero



Outline

GridBagDemo.java

Line 19

Line 22

```
1  // Fig. 14.19: GridBagDemo.java
2  // Demonstrating GridBagLayout.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class GridBagDemo extends JFrame {
8     private Container container;
9     private GridBagLayout layout;
10    private GridBagConstraints constraints;
11
12    // set up GUI
13    public GridBagDemo()
14    {
15       super( "GridBagLayout" );
16
17       container = getContentPane();
18       layout = new GridBagLayout();
19       container.setLayout( layout );
20
21       // instantiate gridbag constraints
22       constraints = new GridBagConstraints();
23
24       // create GUI components
25       JTextArea textArea1 = new JTextArea( "TextArea1", 5, 10 );
26       JTextArea textArea2 = new JTextArea( "TextArea2", 2, 2 );
27
```

Set `GridBagLayout` as layout manager

Used to determine component location and size in grid

Outline

.jav

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

If user resizes `Container`, first `JTextArea` is filled entire allocated area in grid

First `JTextArea` spans one row and three columns

If user resizes `Container`, first `JButton` fills horizontally in grid

First `JButton` spans two rows and one column

Line 51

If user resizes `Container`, second `JButton` fills extra space

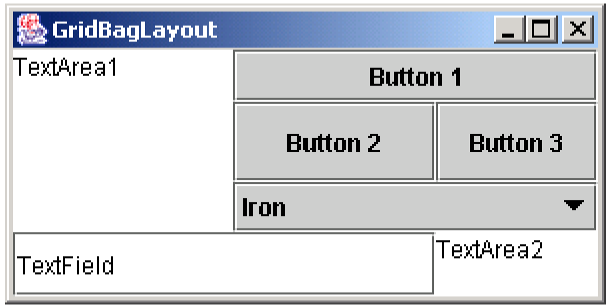
```
      // fill is BOTH for button3
      constraints.weightx = 0;
      constraints.weighty = 0;
      addComponent( button3, 1, 2, 1, 1 );

      // weightx and weighty for textField are both 0, fill is BOTH
      addComponent( textField, 3, 0, 2, 1 );

      // weightx and weighty for textArea2 are both 0, fill is BOTH
      addComponent( textArea2, 3, 2, 1, 1 );

      setSize( 300, 150 );
      setVisible( true );

   } // end constructor GridBagDemo

   // method to set constraints on
   private void addComponent( Component component,
      int row, int column, int width, int height )
   {
      // set gridx and gridy
      constraints.gridx = column;
      constraints.gridy = row;

```

GridBagDemo.java

```
      // set gridwidth and gridheight
      constraints.gridwidth = width;
      constraints.gridheight = height;

      // set constraints and add component
      layout.setConstraints( component, constraints );
      container.add( component );
   }

   public static void main( String args[] )
   {
      GridBagDemo application = new GridBagDemo();
      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
   }

} // end class GridBagDemo
```

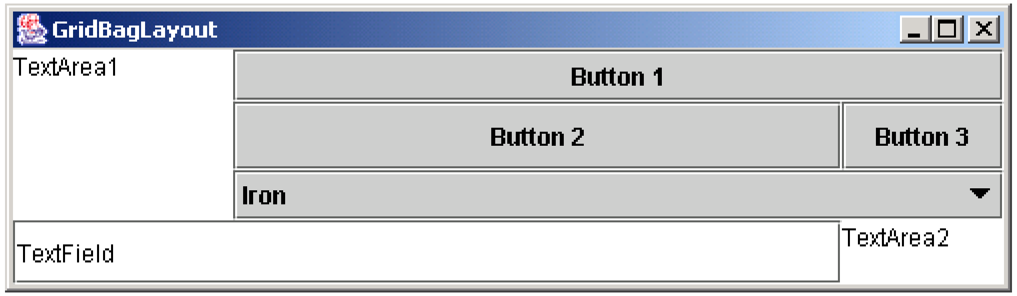
Outline

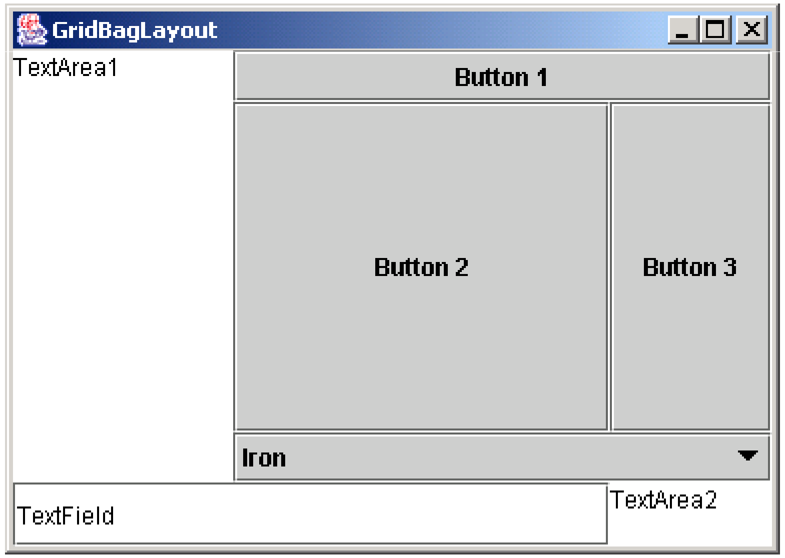
GridBagDemo.jav
a



[Outline](#)

GridBagDemo.java





`GridBagConstraints` **Constants** `RELATIVE` **and** `REMAINDER`

- Constants `RELATIVE` and `REMAINDER`
  - Used in place of variables `gridx` and `gridy`
  - `RELATIVE`
    - Specifies next-to-last component placement in row or column
      - Component should be placed next to one previously added
  - `REMAINDER`
    - Specifies component as last component in row or column

```java
1  // Fig. 14.20: GridBagDemo2.java
2  // Demonstrating GridBagLayout constants.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class GridBagDemo2 extends JFrame {
8     private GridBagLayout layout;
9     private GridBagConstraints constraints;
10    private Container container;
11
12    // set up GUI
13    public GridBagDemo2()
14    {
15       super( "GridBagLayout" );
16
17       container = getContentPane();
18       layout = new GridBagLayout();
19       container.setLayout( layout );
20
21       // instantiate gridbag constraints
22       constraints = new GridBagConstraints();
23
24       // create GUI components
25       String metals[] = { "Copper", "Aluminum", "Silver" };
26       JComboBox comboBox = new JComboBox( metals );
27
```

Set `GridBagLayout` as layout manager

Used to determine component location and size in grid

Outline

`GridBagDemo2.java`

Lines 18-19

Line 22

```
28        JTextField textField = new JTextField( "TextField" );
29
30        String fonts[] = { "Serif", "Monospaced" };
31        JList list = new JList( fonts );
32
33        String names[] = { "zero", "one", "two", "three", "four" };
34        JButton buttons[] = new JButton[ names.length ];
35
36        for ( int count = 0; count < buttons.length; count++ )
37           buttons[ count ] = new JButton( names[ count ] );
38
39        // define GUI component constraints for textField
40        constraints.weightx = 1;
41        constraints.weighty = 1;
42        constraints.fill = GridBagConstraints.BOTH;
43        constraints.gridwidth = GridBagConstraints.REMAINDER;
44        addComponent( textField );
45
46        // buttons[0] -- weightx and weighty are 1: fill is BOTH
47        constraints.gridwidth = 1;
48        addComponent( buttons[ 0 ] );
49
50        // buttons[1] -- weightx and weighty are 1: fill is BOTH
51        constraints.gridwidth = GridBagConstraints.RELATIVE;
52        addComponent( buttons[ 1 ] );
53
```

Specify `textField` as last (only) component in first row

Place `button[0]` as first component in second row

Place `button[1]` right next to `button[0]`

Outline

`GridBagDemo2.java`

Line 43

Line 48

```
54      // buttons[2] -- weightx and weighty are 1: fill is BOTH
55      constraints.gridwidth = GridBagConstraints.REMAINDER;
56      addComponent( buttons[ 2 ] );
57
58      // comboBox -- weightx is 1: fill is BOTH
59      constraints.weighty = 0;
60      constraints.gridwidth = GridBagConstraints.REMAINDER;
61      addComponent( comboBox );
62
63      // buttons[3] -- weightx is 1: fill is BOTH
64      constraints.weighty = 1;
65      constraints.gridwidth = GridBagConstraints.REMAINDER;
66      addComponent( buttons[ 3 ] );
67
68      // buttons[4] -- weightx and weighty are 1: fill is BOTH
69      constraints.gridwidth = GridBagConstraints.RELATIVE;
70      addComponent( buttons[ 4 ] );
71
72      // list -- weightx and weighty are 1: fill is BOTH
73      constraints.gridwidth = GridBagConstraints.REMAINDER;
74      addComponent( list );
75
76      setSize( 300, 200 );
77      setVisible( true );
78
79   }  // end constructor
80
```

Outline

mo2.ja
va

Place `button[2]` right next to `button[1]`

Specify `comboBox` as last (only) component in third row

Line 61

Specify `buttons[3]` as last (only) component in fourth row

Line 70

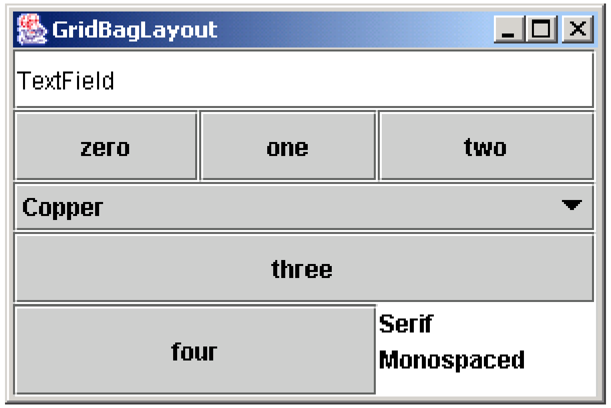
Place `button[4]` as first component in fifth row

Specify `list` as last component in fifth row

```
    // add a Component to the container
    private void addComponent( Component component )
    {
       layout.setConstraints( component, constraints );
       container.add( component );      // add component
    }

    public static void main( String args[] )
    {
       GridBagDemo2 application = new GridBagDemo2();
       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    }

 }  // end class GridBagDemo2
```

Outline

GridBagDemo2.java



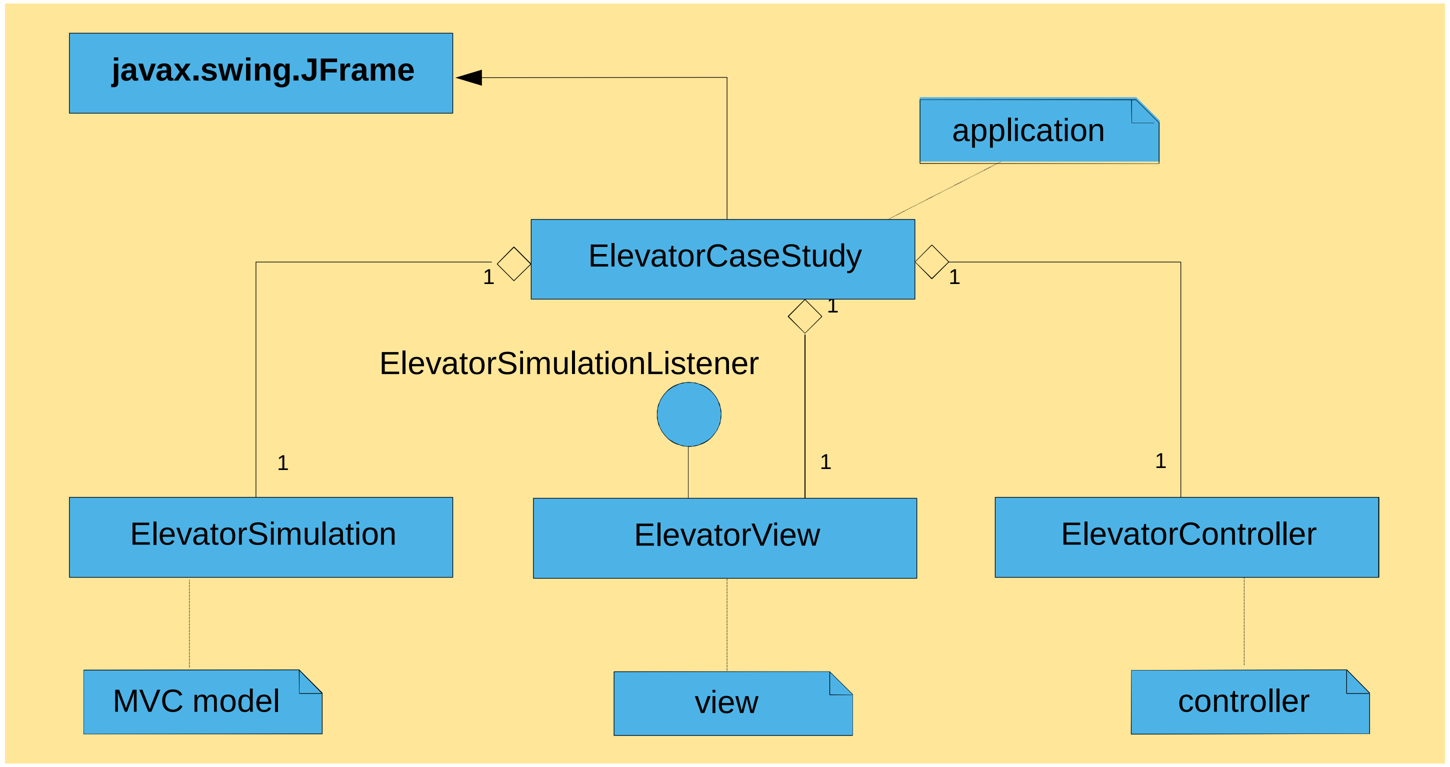
14.13 (Optional Case Study) Thinking About Objects: Model-View-Controller

- Model-View-Controller
  - Architectural pattern for building systems
  - Divide system responsibilities into three parts
    - Model
      - Maintains program data and logic
    - View
      - Visual representation of model
    - Controller
      - Processes user input and modifies model
  - Step by step
    - User uses controller to change data in model
    - Model then informs view of change
    - View changes visual presentation to reflect change

Model-View-Controller Elevator Simulation

- Model-View-Controller in elevator simulation
  - Example
    - User presses First Floor of Second Floor `Jbutton`
      - Controller adds `Person` to model
    - Model notifies view of `Person`'s creation
    - View displays `Person` on `Floor` in response to notification

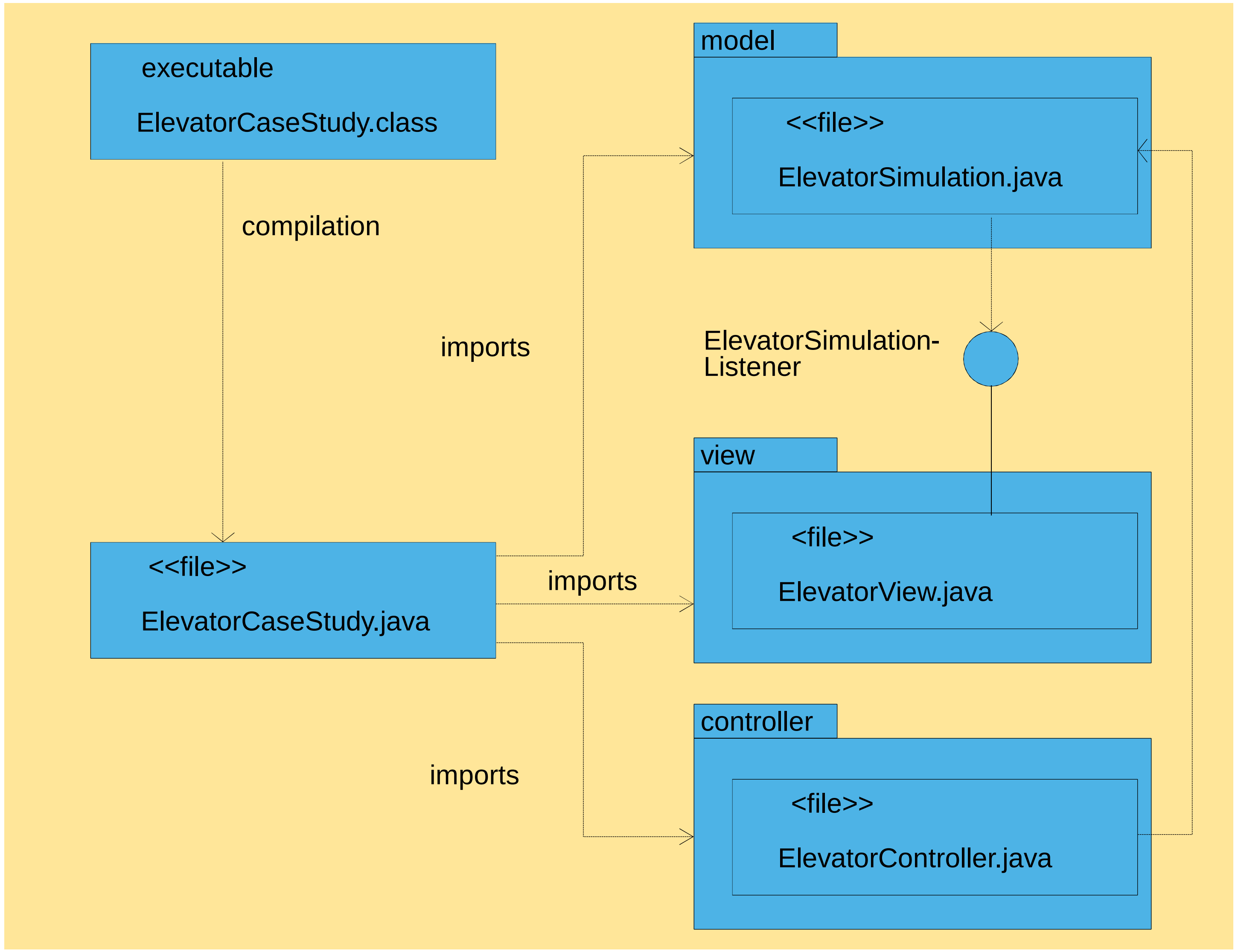
Fig. 14.21 Class diagram of the elevator simulation



14.13 (Optional Case Study) Thinking About Objects: Model-View-Controller

- Component diagram (UML)
  - Models “pieces” (components) used by system
    - e.g., `.class` file, `.java` files, images, packages, etc.
  - Notation
    - Components are represented as “plugs”
    - Packages are represented as “folders”
    - Dotted arrows indicate *dependencies* among components
      - Changing one component requires changing another

Fig. 14.22 Artifacts of the elevator simulation



```java
1  // ElevatorController.java
2  // Controller for Elevator Simulation
3  package com.deitel.jhtp5.elevator.controller;
4
5  import java.awt.*;
6  import java.awt.event.*;
7
8  import javax.swing.*;
9
10 // Deitel packages
11 import com.deitel.jhtp5.elevator.model.*;
12 import com.deitel.jhtp5.elevator.event.*;
13 import com.deitel.jhtp5.elevator.ElevatorConstants;
14
15 public class ElevatorController extends JPanel
16    implements ElevatorConstants {
17
18    // controller contains two JButtons
19    private JButton firstControllerButton;
20    private JButton secondControllerButton;
21
22    // reference to ElevatorSimulation
23    private ElevatorSimulation elevatorSimulation;
24
```

`ElevatorController` GUI for elevator simulation

`JButton`s for creating `Person`s on `Floor`

Outline

`ElevatorController.java`

Line 15

Lines 19-20

```
25    public ElevatorController( ElevatorSimulation simulation )
26    {
27       elevatorSimulation = simulation;
28       setBackground( Color.WHITE );
29
30       // add first button to controller
31       firstControllerButton = new JButton( "First Floor" );
32       add( firstControllerButton );
33
34       // add second button to controller
35       secondControllerButton = new JButton( "Second Floor" );
36       add( secondControllerButton );
37
38       // anonymous inner class registers to receive ActionEvents
39       // from first Controller JButton
40       firstControllerButton.addActionListener(
41          new ActionListener() {
42
43             // invoked when a JButton has been pressed
44             public void actionPerformed( ActionEvent event )
45             {
46                // place Person on first Floor
47                elevatorSimulation.addPerson(
48                   FIRST_FLOOR_NAME );
49
```

Register `JButton`s with separate anonymous `ActionListener`s

Add `Person` to respective `Floor`, depending on `JButton` that user pressed

Outline

`ElevatorController.java`

Line 40

Lines 47-48

Outline

```
50                // disable user input
51                firstControllerButton.setEnabled( false );
52             }
53          } // end anonymous inner class
54       );
55 
56       // anonymous inner class registers to receive ActionEvents
57       // from second Controller JButton
58       secondControllerButton.addActionListener(
59          new ActionListener() {
60 
61             // invoked when a JButton has been pressed
62             public void actionPerformed( ActionEvent event )
63             {
64                // place Person on second Floor
65                elevatorSimulation.addPerson(
66                   SECOND_FLOOR_NAME );
67 
68                // disable user input
69                secondControllerButton.setEnabled( false );
70             }
71          } // end anonymous inner class
72       );
73 
```

`ElevatorController.java`

Register `JButton`s with separate anonymous `ActionListener`s

Lines 51 and 69

Add `Person` to respective `Floor`, depending on `JButton` that user pressed

Disable `JButton` after `Person` is created (so user cannot create more than one `Person` on `Floor`)

Outline

ElevatorControl

Lines 89 and 93

```
74         // anonymous inner class enables user input on Floor if
75         // Person enters Elevator on that Floor
76         elevatorSimulation.addPersonMoveListener(
77            new PersonMoveListener() {
78
79               // invoked when Person has entered Elevator
80               public void personEntered(
81                  PersonMoveEvent event )
82               {
83                  // get Floor of departure
84                  String location =
85                     event.getLocation().getLocationName();
86
87                  // enable first JButton if first Floor departure
88                  if ( location.equals( FIRST_FLOOR_NAME ) )
89                     firstControllerButton.setEnabled( true );
90
91                  // enable second JButton if second Floor
92                  else
93                     secondControllerButton.setEnabled( true );
94
95               } // end method personEntered
96
97               // other methods implementing PersonMoveListener
98               public void personCreated(
99                  PersonMoveEvent event ) {}
100
```

Enable `ElevatorModel` to listener for `PersonMoveEvent`s

Enable `JButton` after `Person` enters `Elevator` (so user can create another `Person`)

```
            public void personArrived(
               PersonMoveEvent event ) {}

            public void personExited(
               PersonMoveEvent event ) {}

            public void personDeparted(
               PersonMoveEvent event ) {}

            public void personPressedButton(
               PersonMoveEvent event ) {}

         } // end anonymous inner class
      );
   } // end ElevatorController constructor
}
```

Outline

ElevatorController.java

```java
// ElevatorConstants.java
// Constants used between ElevatorModel and ElevatorView
package com.deitel.jhtp5.elevator;

public interface ElevatorConstants {

   public static final String FIRST_FLOOR_NAME = "firstFloor";
   public static final String SECOND_FLOOR_NAME = "secondFloor";
   public static final String ELEVATOR_NAME = "elevator";
}
```

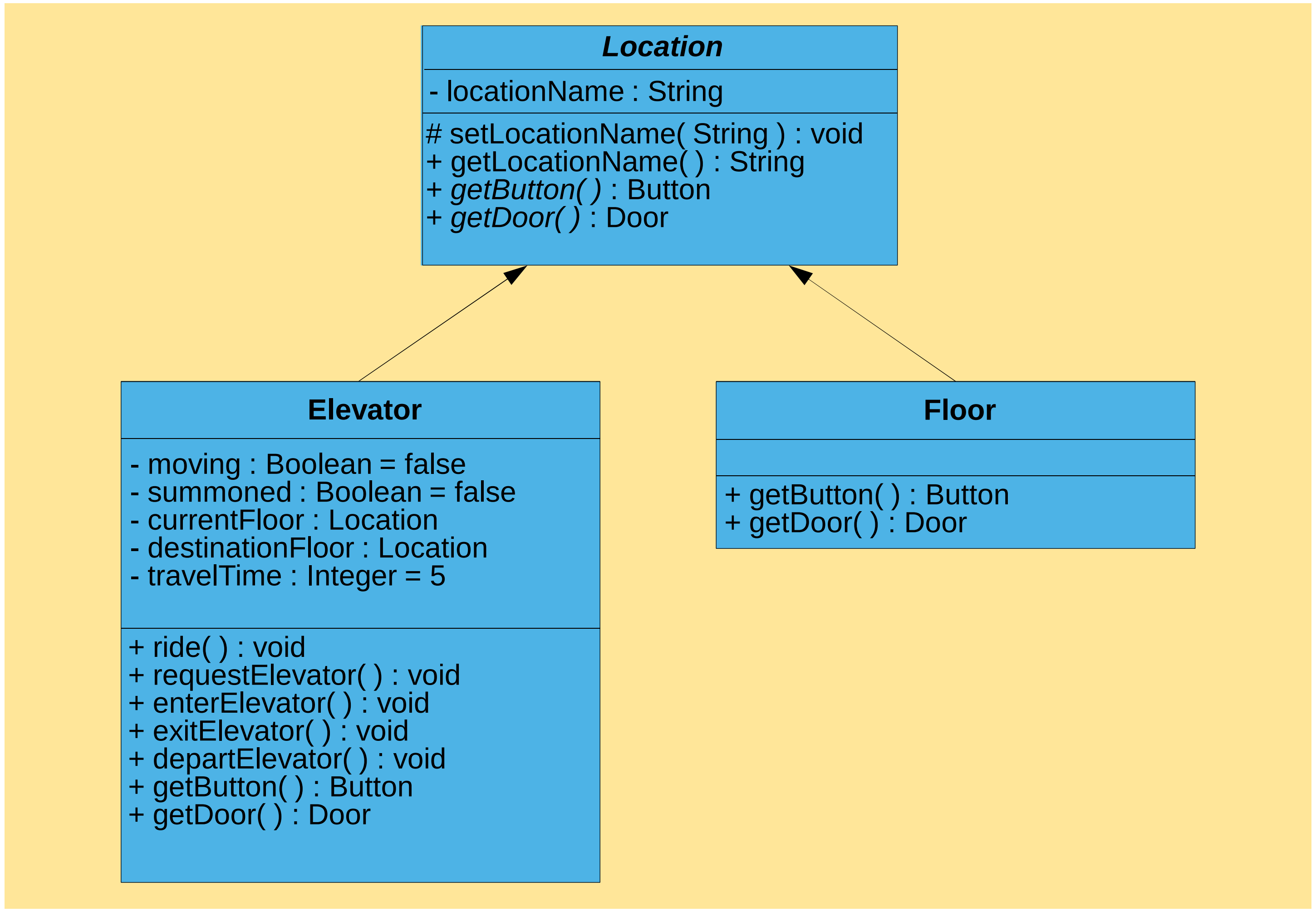
Outline

ElevatorConstants.java

14.13 (Optional Case Study) Thinking About Objects: Model-View-Controller

- Classes `Location`
  - Subclasses `Elevator` and `Floor`
    - Attribute `capacity` no longer needed

Fig. 14.25 Modified class diagram showing generalization of superclass `Location` and subclasses `Elevator` and `Floor`


Location
- locationName : String
# setLocationName(String) : void
+ getLocationName( ) : String
+ *getButton( )* : Button
+ *getDoor( )* : Door
Elevator
- moving : Boolean = false
- summoned : Boolean = false
- currentFloor : Location
- destinationFloor : Location
- travelTime : Integer = 5
+ ride( ) : void
+ requestElevator( ) : void
+ enterElevator( ) : void
+ exitElevator( ) : void
+ departElevator( ) : void
+ getButton( ) : Button
+ getDoor( ) : Door
Floor
+ getButton( ) : Button
+ getDoor( ) : Door

```java
1   // ElevatorCaseStudy.java
2   // Application with Elevator Model, View, and Controller (MVC)
3   package com.deitel.jhtp5.elevator;
4
5   // Java core packages
6   import java.awt.*;
7
8   // Java extension packages
9   import javax.swing.*;
10
11  // Deitel packages
12  import com.deitel.jhtp5.elevator.model.*;
13  import com.deitel.jhtp5.elevator.view.*;
14  import com.deitel.jhtp5.elevator.controller.*;
15
16  public class ElevatorCaseStudy extends JFrame {
17
18     // model, view and controller
19     private ElevatorSimulation model;
20     private ElevatorView view;
21     private ElevatorController controller;
22
23     // constructor instantiates model, view, and controller
24     public ElevatorCaseStudy()
25     {
```

Import packages `model`, `view` and `controller`

`ElevatorCaseStudy` aggregates one instance each of classes `ElevatorSimulation`, `ElevatorView` and `ElevatorController`

Outline

`ElevatorCaseStudy.java`

Lines 12-14

Lines 19-21

```java
      super( "Deitel Elevator Simulation" );

      // instantiate model, view and controller
      model = new ElevatorSimulation();
      view = new ElevatorView();
      controller = new ElevatorController( model );

      // register View for Model events
      model.setElevatorSimulationListener( view );

      // add view and controller to ElevatorCaseStudy
      getContentPane().add( view, BorderLayout.CENTER );
      getContentPane().add( controller, BorderLayout.SOUTH );

   } // end ElevatorCaseStudy constructor

   // main method starts program
   public static void main( String args[] )
   {
      // instantiate ElevatorCaseStudy
      ElevatorCaseStudy simulation = new ElevatorCaseStudy();
      simulation.setDefaultCloseOperation( EXIT_ON_CLOSE );
      simulation.pack();
      simulation.setVisible( true );
   }
}
```

Outline

ElevatorCaseStudy.java

Register `ElevatorSimulation` as listener for `ElevatorView`

Add `ElevatorView` and `ElevatorController` to `ElevatorCaseStudy`

14.14 (Optional) Discovering Design Patterns: Design Patterns Used in Packages `java.awt` and `javax.swing`

- Continue design-patterns discussion
  – Design patterns associated with Java GUI components
    - GUI components take advantage of design patterns

14.14.1 Creational Design Patterns

- Factory Method design pattern
  - Suppose we design system that opens image from file
    - Several image formats exist (e.g., GIF, JPEG, etc.)
      - Each image format has different structure
    - Method `createImage` of class `Component` creates `Image`
    - Two `Image` objects (one for GIF image, one for JPEG image)
    - Method `createImage` uses parameter to determine proper `Image` subclass from which to instantiate `Image` object

      ```
      createImage( "image.gif" );
      ```
      - Returns `Image` object with GIF data

      ```
      createImage( "image.jpg" );
      ```
      - Returns `Image` object with JPEG data
    - Method `createImage` is called a *factory method*
      - Determines subclass to instantiate object at run time

14.14.2 Structural Design Patterns

- Adapter design pattern
  - Used with objects with incompatible interfaces
    - Allows these objects to collaborate with each other
    - Object's interface *adapts* to another object's interface
  - Similar to adapter for plug on electrical device
    - European electrical sockets differ from those in United States
    - American plug will not work with European socket
    - Use *adapter* for plug
  - Class `MouseAdapter`
    - Objects that generate `MouseEvent`s adapts to objects that handle `MouseEvent`s

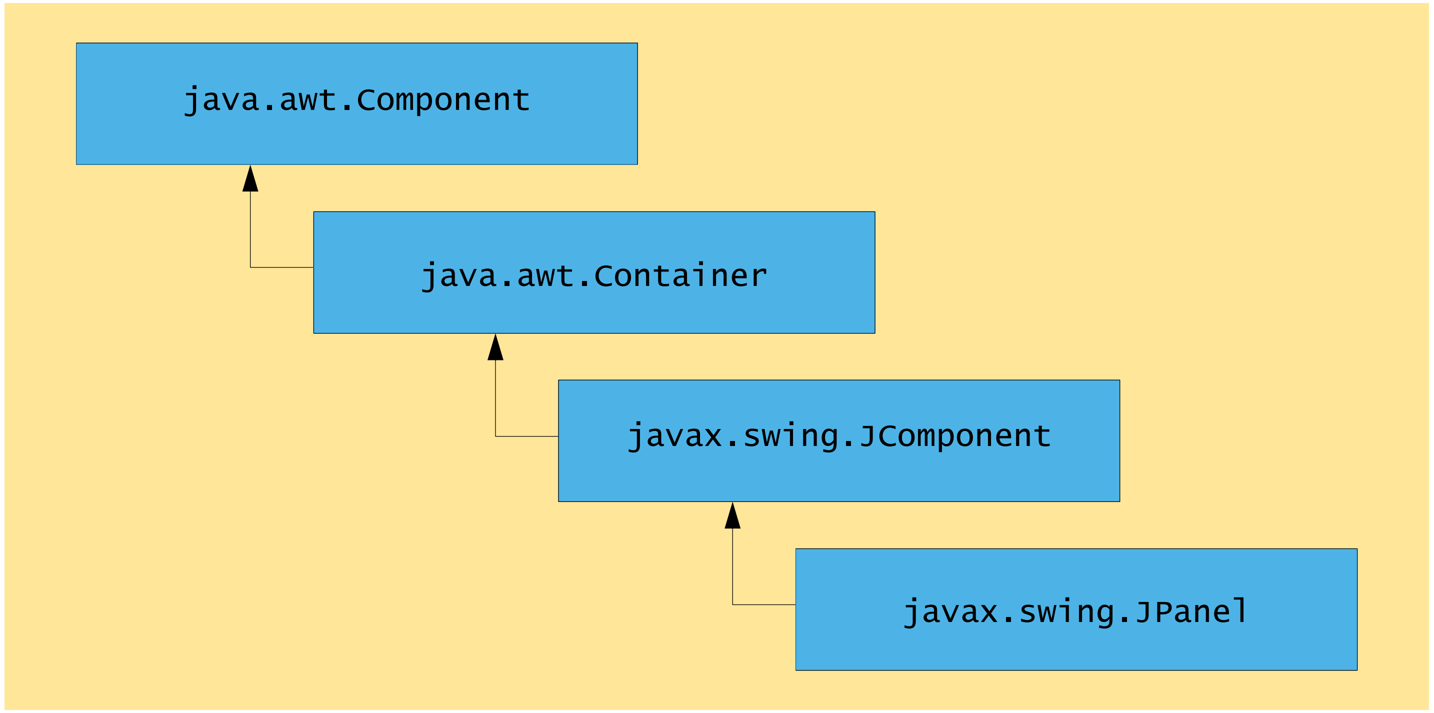
14.14.2 Structural Design Patterns

- Bridge design pattern
  - Design class `Button` for Windows and Macintosh systems
    - Class contains button information (e.g., `String` label)
    - Subclasses `Win32Button` and `MacButton`
      - Contain look-and-feel information
    - Problem with this approach
      - Creating class `ImageButton` (subclass of `Button`)
        - Requires creating `Win32ImageButton` and `MacImageButton`
    - Solution:
      - Separate abstraction (i.e., `Button`) from implementation (i.e., `Win32Button` and `MacButton`)
      - `Button` contains reference (*bridge*) to `ButtonPeer`
        - Handles platform-specific implementations

14.14.2 Structural Design Patterns

- Composite design pattern
  - Organize components into hierarchical structures
    - Each node represents component
    - All nodes implement same interface
      - Polymorphism ensures clients traverse all nodes uniformly
  - Used by Swing components
    - `JPanel` is `JContainer` subclass
    - `JPanel` object can contain GUI component
      - `JPanel` remains unaware of component’s specific type

Fig. 14.27 Inheritance hierarchy for class `JPanel`



14.14.3 Behavioral Design Patterns

- Chain-of-Responsibility design pattern
  - Determine object that handles message at run time
  - Three-line office-phone system
    - First line handles call
    - If first line is busy, second line handles call
    - If second line is busy, third line handles call
  - Message sent through “chain”
    - Each object in chain decides whether to handle message
      - If unable to handle message, that object sends message to next object in chain
  - Method `processEvent` of class `Button`
    - Handles `AWTEvent` or sends to next object

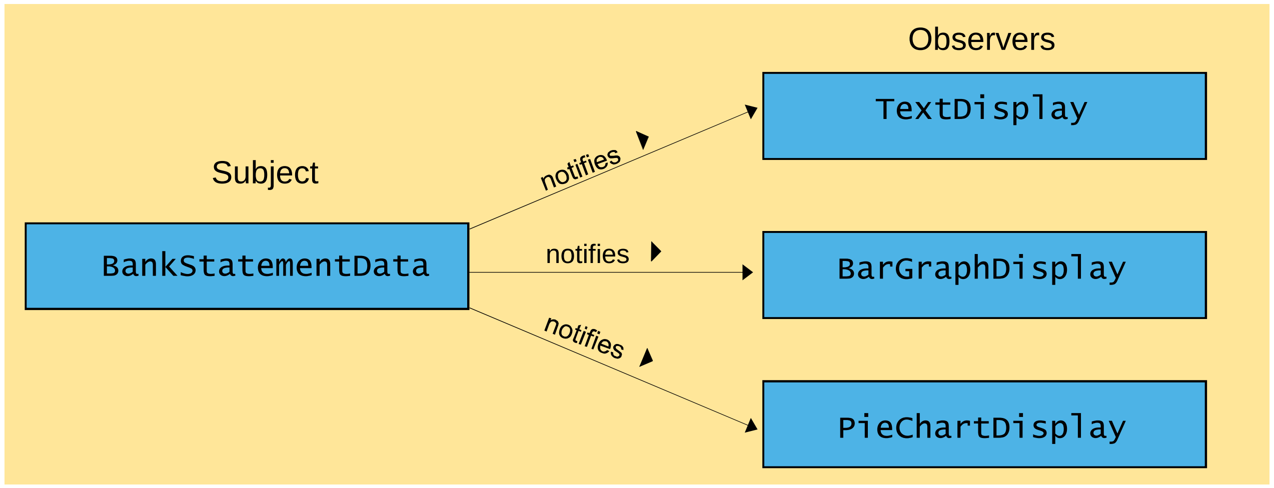
14.14.3 Behavioral Design Patterns

- Command design pattern
  - Applications provide several ways to perform same task
    - **Edit** menu with menu items for cutting and copying text
    - Toolbar and popup menus may offer same feature
  - Encapsulate functionality (*command*) in reusable object
    - e.g., "cut text" functionality
    - Functionality can then be added to menus, toolbars, etc.
    - Developers code functionality only once

14.14.3 Behavioral Design Patterns

- Observer design pattern
  - Design program for viewing bank-account information
    - Class `BankStatementData` store bank-statement data
    - Class `TextDisplay` displays data in text format
    - Class `BarGraphDisplay` displays data in bar-graph format
    - Class `PieChartDisplay` displays data in pie-chart format
    - `BankStatementData` (subject) notifies `Display` classes (observers) to display data when it changes
  - Subject notifies observers when subject changes state
    - Observers act in response to notification
    - Promotes *loose coupling*
  - Used by
    - class `java.util.Observable`
    - class `java.util.Observer`

Fig. 14.28 Basis for the Observer design pattern



14.14.3 Behavioral Design Patterns

- Strategy design pattern
  - Encapsulates algorithm
  - `LayoutManager`s are strategy objects
    - Classes `FlowLayout`, `BorderLayout`, `GridLayout`, etc.
      - Implement interface `LayoutManager`
    - Each class uses method `addLayoutComponent`
      - Each method implementation uses different algorithm
        - `FlowLayout` adds components left-to-right
        - `BorderLayout` adds components in five regions
        - `GridLayout` adds components in specified grid
    - Class `Container` has `LayoutManager` reference
      - Use method `setLayout`
        - Select different layout manager at run time

14.14.3 Behavioral Design Patterns

- Template Method design pattern
  - Objects share single algorithm defined in superclass
  - Consider Fig.14.28
    - Display objects use *same algorithm* to acquire and display data
      - Get statements from `BankStatementData`
      - Parse statements
      - Display statements
    - Create superclass `BankStatementDisplay`
      - Provides methods that comprise algorithm
      - Subclasses override “display” method, because each subclass displays data differently