Graphical User Interface Components

Introduction
Swing Overview
`JLabel`
Event-Handling Model
`JTextField` and `JPasswordField`
    How Event Handling Works
`JButton`
`JCheckBox` and `JRadioButton`
`JComboBox`
`JList`
Multiple-Selection Lists
Mouse Event Handling
Adapter Classes
Keyboard Event Handling
Layout Managers
    `FlowLayout`
    `BorderLayout`
    `GridLayout`

Panels

© 2002 Prentice Hall, Inc. All rights reserved.

Introduction

- Graphical User Interface (GUI)
  - Gives program distinctive "look" and "feel"
  - Provides users with basic level of familiarity
  - Built from GUI components (controls, widgets, etc.)
    - User interacts with GUI component via mouse, keyboard, etc.

© 2002 Prentice Hall, Inc. All rights reserved.

Some basic GUI components.

Component	Description
`JLabel`	An area where uneditable text or icons can be displayed.
`JTextField`	An area in which the user inputs data from the keyboard. The area can also display information.
`JButton`	An area that triggers an event when clicked.
`JCheckBox`	A GUI component that is either selected or not selected.
`JComboBox`	A drop-down list of items from which the user can make a selection by clicking an item in the list or possibly by typing into the box.
`JList`	An area where a list of items is displayed from which the user can make a selection by clicking once on any element in the list. Double-clicking an element in the list generates an action event. Multiple elements can be selected.
`JPanel`	A container in which components can be placed.

Fig. 12.2 Some basic GUI components.

© 2002 Prentice Hall, Inc. All rights reserved.

Swing Overview

- Swing GUI components
  - Package **`javax.swing`**
  - Components originate from AWT (package **`java.awt`**)
  - Contain *look and feel*
    - Appearance and how users interact with program
  - *Lightweight components*
    - Written completely in Java
    - Not tied to the windowing system of the platform

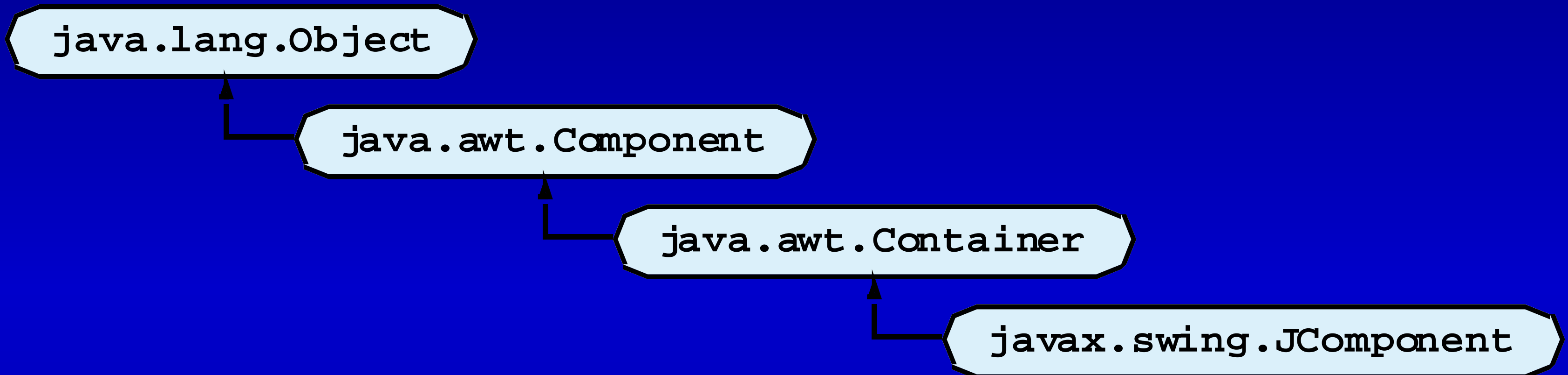
© 2002 Prentice Hall, Inc. All rights reserved.

Swing Overview (cont.)

- Class **`Component`**
  - Contains method **`paint`** for drawing **`Component`** onscreen
- Class **`Container`**
  - Collection of related components
  - Contains method **`add`** for adding components
- Class **`JComponent`**
  - *Pluggable look and feel* for customizing look and feel
  - Shortcut keys (*mnemonics*)
  - Common event-handling capabilities

© 2002 Prentice Hall, Inc. All rights reserved.

Common superclasses of many of the Swing components.



© 2002 Prentice Hall, Inc. All rights reserved.

12.3 `JLabel`

- Label
  - Provide text on GUI
  - Defined with class **`JLabel`**
  - Can display:
    - Single line of read-only text
    - Image
    - Text and image

© 2002 Prentice Hall, Inc. All rights reserved.

```java
1  // Fig. 12.4: LabelTest.java
2  // Demonstrating the JLabel class.
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class LabelTest extends JFrame {
12    private JLabel label1, label2, label3;
13
14    // set up GUI
15    public LabelTest()
16    {
17       super( "Testing JLabel" );
18
19       // get content pane and set its layout
20       Container container = getContentPane();
21       container.setLayout( new FlowLayout() );
22
23       // JLabel constructor with a string argument
24          label1 = new JLabel( "Label with text" );
25       label1.setToolTipText( "This is label1" );
26       container.add( label1 );
27
28       // JLabel constructor with string, Icon and
29       // alignment arguments
30       Icon bug = new ImageIcon( "bug1.gif" );
31       label2 = new JLabel( "Label with text and icon",
32          bug, SwingConstants.LEFT );
33       label2.setToolTipText( "This is label2" );
34       container.add( label2 );
35
```

Declare three **JLabel**s

Create first **JLabel** with text "**Label with text**"

Tool tip is text that appears when user moves cursor over **JLabel**

Create second **JLabel** with text to left of image

Outline

LabelTest.java

Line 12

Line 24

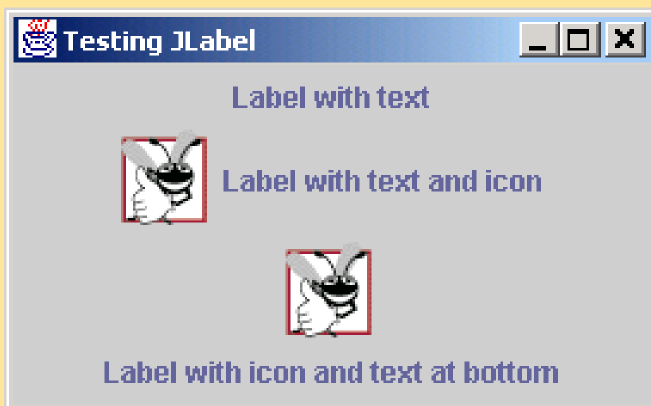
Line 25

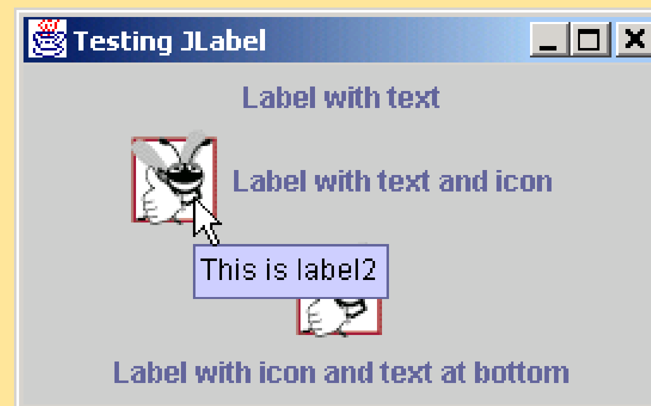
Lines 31-32

© 2002 Prentice Hall, Inc. All rights reserved.

```
36        // JLabel constructor no arguments
37        label3 = new JLabel();
38        label3.setText( "Label with icon and text at bottom" );
39        label3.setIcon( bug );
40        label3.setHorizontalTextPosition( SwingConstants.CENTER );
41        label3.setVerticalTextPosition( SwingConstants.BOTTOM );
42        label3.setToolTipText( "This is label3" );
43        container.add( label3 );
44
45        setSize( 275, 170 );
46        setVisible( true );
47     }
48
49     // execute application
50     public static void main( String args[] )
51     {
52        LabelTest application = new LabelTest();
53
54        application.setDefaultCloseOperation(
55           JFrame.EXIT_ON_CLOSE );
56     }
57
58  }  // end class LabelTest
```

Create third **JLabel** with text below image





© 2002 Prentice Hall, Inc. All rights reserved.

12.4 Event-Handling Model

- GUIs are *event driven*
  - Generate *events* when user interacts with GUI
    - e.g., moving mouse, pressing button, typing in text field, etc.
    - Class **`java.awt.AWTEvent`**

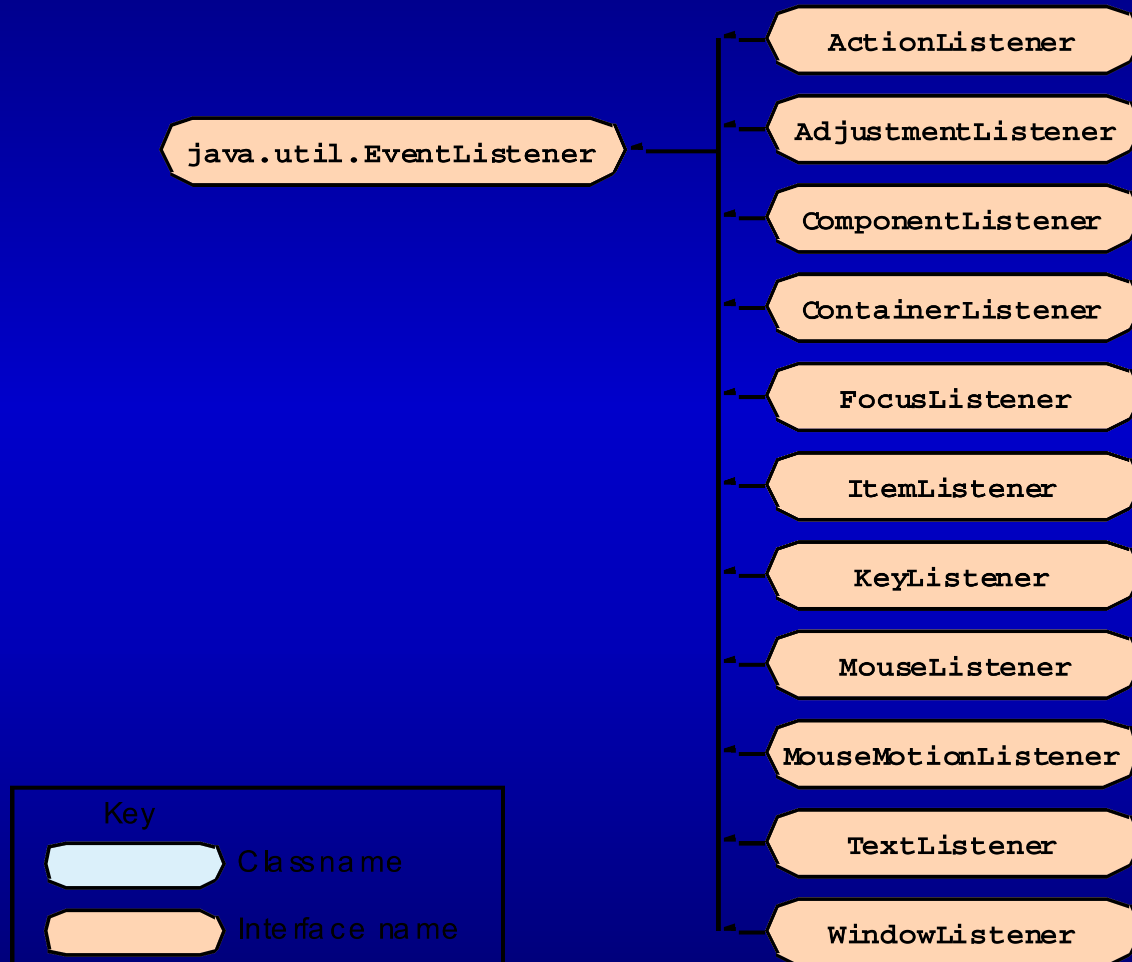
© 2002 Prentice Hall, Inc. All rights reserved.

12.4 Event-Handling Model (cont.)

- Event-handling model
  - Three parts
    - Event source
      - GUI component with which user interacts
    - Event object
      - Encapsulates information about event that occurred
    - Event listener
      - Receives event object when notified, then responds
  - Programmer must perform two tasks
    - Register event listener for event source
    - Implement event-handling method (event handler)

© 2002 Prentice Hall, Inc. All rights reserved.

Fig. 12.6 Event-listener interfaces of package `java.awt.event`



© 2002 Prentice Hall, Inc. All rights reserved.

12.5 `JTextField` and `JPasswordField`

- **`JTextField`**
  - Single-line area in which user can enter text
- **`JPasswordField`**
  - Extends **`JTextField`**
  - Hides characters that user enters

© 2002 Prentice Hall, Inc. All rights reserved.

```
1  // Fig. 12.7: TextFieldTest.java
2  // Demonstrating the JTextField class.
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class TextFieldTest extends JFrame {
12    private JTextField textField1, textField2, textField3;
13    private JPasswordField passwordField;
14
15    // set up GUI
16    public TextFieldTest()
17    {
18       super( "Testing JTextField and JPasswordField" );
19
20       Container container = getContentPane();
21       container.setLayout( new FlowLayout() );
22
23       // construct textfield with default sizing
24       textField1 = new JTextField( 10 );
25       container.add( textField1 );
26
27       // construct textfield with default text
28       textField2 = new JTextField( "Enter text here" );
29       container.add( textField2 );
30
31       // construct textfield with default text and
32       // 20 visible elements and no event handler
33       textField3 = new JTextField( "Uneditable text field",
34       textField3.setEditable( false );
35       container.add( textField3 );
```

Outline

TextFieldTest.java

Lines 12-13

Line 24

Line 28

Declare three **JTextField**s and one **JPasswordField**

First **JTextField** contains empty string

Second **JTextField** contains text "**Enter text here**"

Third **JTextField** contains uneditable text

© 2002 Prentice Hall, Inc. All rights reserved.

```
36
37        // construct textfield with default text
38        passwordField = new JPasswordField( "Hidden text" );
39        container.add( passwordField );
40
41        // register event handlers
42        TextFieldHandler handler = new TextFieldHandler();
43        textField1.addActionListener( handler );
44        textField2.addActionListener( handler );
45        textField3.addActionListener( handler );
46        passwordField.addActionListener( handler );
47
48        setSize( 325, 100 );
49        setVisible( true );
50     }
51
52     // execute application
53     public static void main( String args[] )
54     {
55        TextFieldTest application = new TextFieldTest();
56
57        application.setDefaultCloseOperation(
58           JFrame.EXIT_ON_CLOSE );
59     }
60
61     // private inner class for event handling
62     private class TextFieldHandler implements ActionListener {
63
64        // process text field events
65        public void actionPerformed( ActionEvent event )
66        {
67           String string = "";
68
69           // user pressed Enter in JTextField textField1
70           if ( event.getSource() == textField1 )
```

JPasswordField contains text "**Hidden text**," but text appears as series of asterisks (*)

Line 38

Lines 43-46

Register GUI components with **TextFieldHandler** (register for **ActionEvents**)

Line 65

Every **TextFieldHandler** instance is an **ActionListener**

Method **actionPerformed** invoked when user presses Enter in GUI field

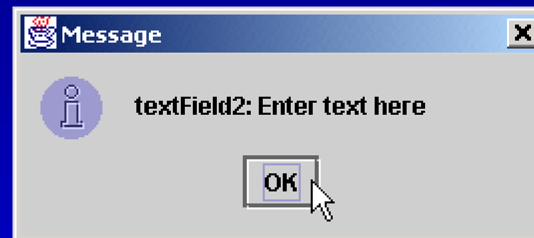
© 2002 Prentice Hall, Inc. All rights reserved.

```
71                string = "textField1: " + event.getActionCommand();
72
73             // user pressed Enter in JTextField textField2
74             else if ( event.getSource() == textField2 )
75                string = "textField2: " + event.getActionCommand();
76
77             // user pressed Enter in JTextField textField3
78             else if ( event.getSource() == textField3 )
79                string = "textField3: " + event.getActionCommand();
80
81             // user pressed Enter in JTextField passwordField
82             else if ( event.getSource() == passwordField ) {
83                JPasswordField pwd =
84                   ( JPasswordField ) event.getSource();
85                string = "passwordField: " +
86                    new String( passwordField.getPassword() );
87             }
88
89             JOptionPane.showMessageDialog( null, string );
90          }
91
92       }  // end private inner class TextFieldHandler
93
94    }  // end class TextFieldTest
```

Outline

TextFieldTest.java

© 2002 Prentice Hall, Inc. All rights reserved.



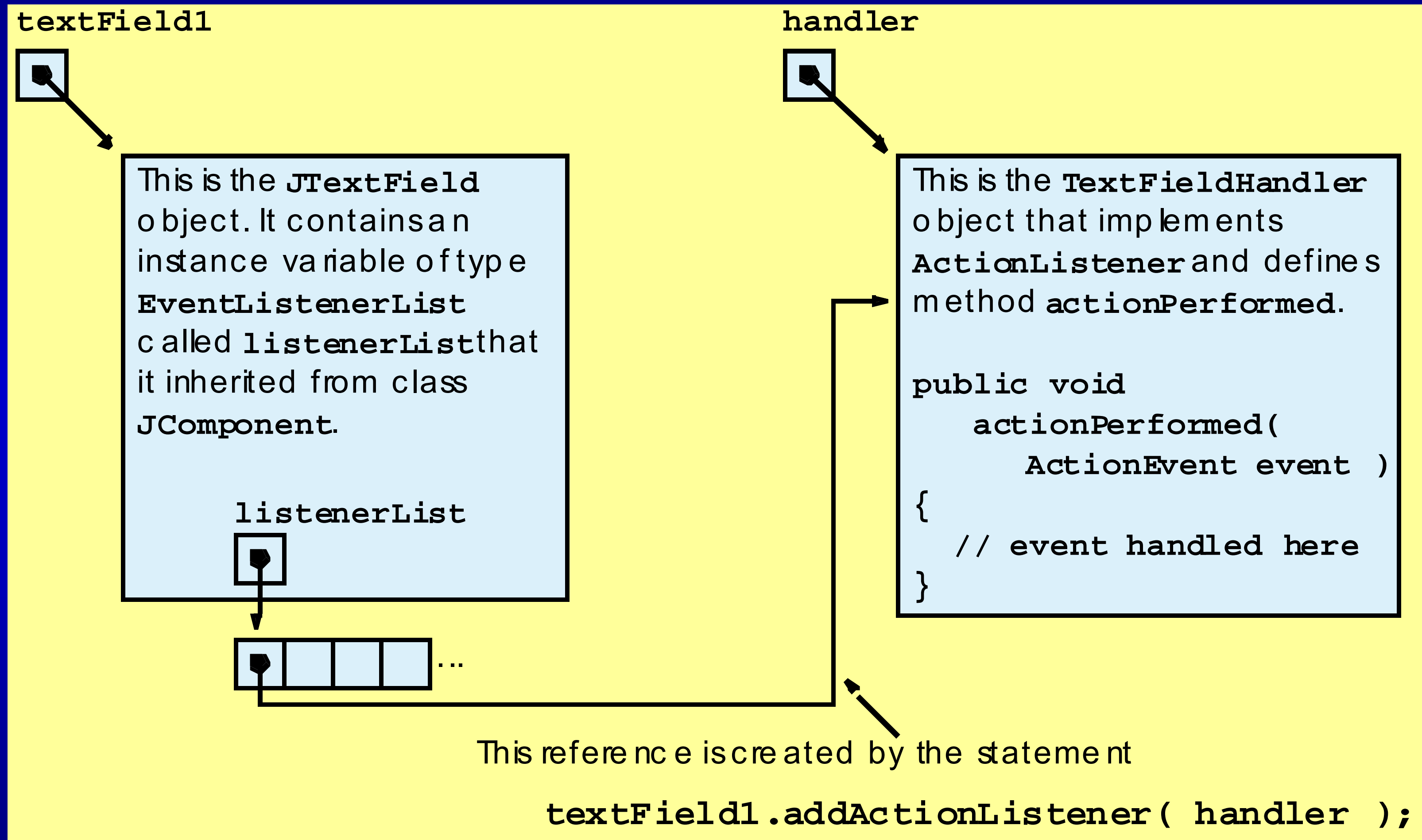
© 2002 Prentice Hall, Inc. All rights reserved.

12.5.1 How Event Handling Works

- Two open questions from Section 12.4
  - How did event handler get registered?
    - Answer:
      - Through component's method **`addActionListener`**
      - Lines 43-46 of **`TextFieldTest.java`**
  - How does component know to call **`actionPerformed`**?
    - Answer:
      - Event is dispatched only to listeners of appropriate type
      - Each event type has corresponding event-listener interface
        - Event ID specifies event type that occurred

© 2002 Prentice Hall, Inc. All rights reserved.

Fig 12.8 Event registration for `JTextField` `textField1`.



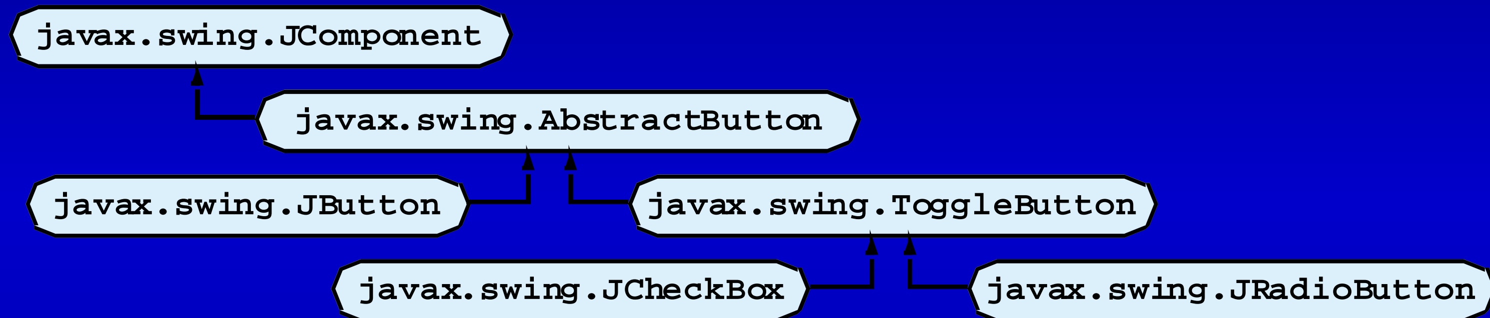
© 2002 Prentice Hall, Inc. All rights reserved.

12.6 `JButton`

- Button
  - Component user clicks to trigger a specific action
  - Several different types
    - Command buttons
    - Check boxes
    - Toggle buttons
    - Radio buttons
  - **`javax.swing.AbstractButton`** subclasses
    - Command buttons are created with class **`JButton`**
      - Generate **`ActionEvent`**s when user clicks button

© 2002 Prentice Hall, Inc. All rights reserved.

Fig. 12.9 The button heirarchy.



© 2002 Prentice Hall, Inc. All rights reserved.

```java
1  // Fig. 12.10: ButtonTest.java
2  // Creating JButtons.
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class ButtonTest extends JFrame {
12    private JButton plainButton, fancyButton;
13
14    // set up GUI
15    public ButtonTest()
16    {
17       super( "Testing Buttons" );
18
19       // get content pane and set its layout
20       Container container = getContentPane();
21       container.setLayout( new FlowLayout() );
22
23       // create buttons
24       plainButton = new JButton( "Plain Button" );
25       container.add( plainButton );
26
27       Icon bug1 = new ImageIcon( "bug1.gif" );
28       Icon bug2 = new ImageIcon( "bug2.gif" );
29       fancyButton = new JButton( "Fancy Button", bug1 );
30       fancyButton.setRolloverIcon( bug2 );
31       container.add( fancyButton );
32
33       // create an instance of inner class ButtonHandler
34       // to use for button event handling
35       ButtonHandler handler = new ButtonHandler();
```

Create two references to **JButton** instances

Instantiate **JButton** with text

Instantiate **JButton** with image and *rollover* image

Instantiate **ButtonHandler** for **JButton** event handling

Outline

ButtonTest.java

Line 12

Line 24

Lines 27-30

Line 35

© 2002 Prentice Hall, Inc. All rights reserved.

```java
36       fancyButton.addActionListener( handler );
37       plainButton.addActionListener( handler );
38
39       setSize( 275, 100 );
40       setVisible( true );
41    }
42
43    // execute application
44    public static void main( String args[] )
45    {
46       ButtonTest application = new ButtonTest();
47
48       application.setDefaultCloseOperation(
49          JFrame.EXIT_ON_CLOSE );
50    }
51
52    // inner class for button event handling
53    private class ButtonHandler implements ActionListener {
54
55       // handle button event
56       public void actionPerformed( ActionEvent event )
57       {
58          JOptionPane.showMessageDialog( null,
59             "You pressed: " + event.getActionCommand() );
60       }
61
62    }  // end private inner class ButtonHandler
63
64 }  // end class ButtonTest
```

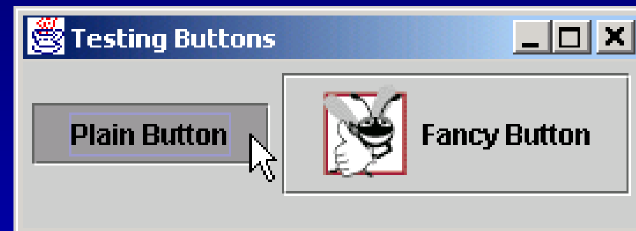
Register **JButton**s to receive events from **ButtonHandler**

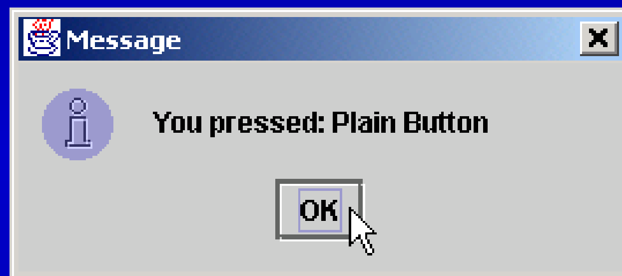
When user clicks **JButton**, **ButtonHandler** invokes method **actionPerformed** of all registered listeners

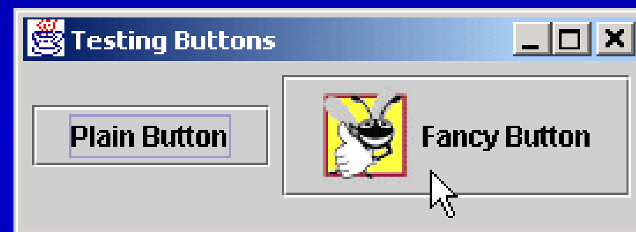
Lines 36-37

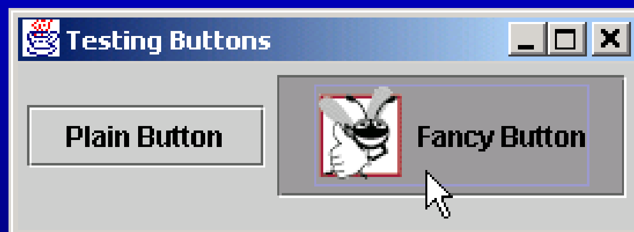
Lines 56-60

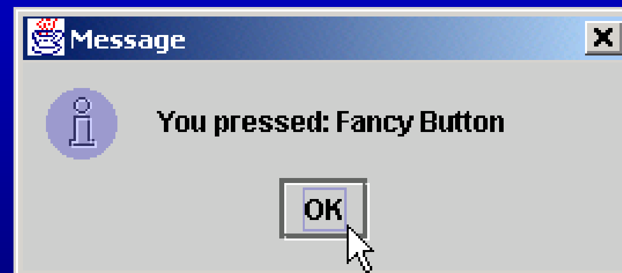
© 2002 Prentice Hall, Inc. All rights reserved.











© 2002 Prentice Hall, Inc. All rights reserved.

12.7 `JCheckBox` and `JRadioButton`

- State buttons
  - On/Off or **`true`**/**`false`** values
  - Java provides three types
    - **`JToggleButton`**
    - **`JCheckBox`**
    - **`JRadioButton`**

© 2002 Prentice Hall, Inc. All rights reserved.

Outline

CheckBoxTest.java

Line 13

Line 27

Lines 31-35

```
1  // Fig. 12.11: CheckBoxTest.java
2  // Creating Checkbox buttons.
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class CheckBoxTest extends JFrame {
12    private JTextField field;
13    private JCheckBox bold, italic;
14
15    // set up GUI
16    public CheckBoxTest()
17    {
18       super( "JCheckBox Test" );
19
20       // get content pane and set its layout
21       Container container = getContentPane();
22       container.setLayout( new FlowLayout() );
23
24       // set up JTextField and set its font
25       field =
26          new JTextField( "Watch the font style change", 20 );
27       field.setFont( new Font( "Serif", Font.PLAIN, 14 ) );
28       container.add( field );
29
30       // create checkbox objects
31       bold = new JCheckBox( "Bold" );
32       container.add( bold );
33
34       italic = new JCheckBox( "Italic" );
35       container.add( italic );
```

Declare two **JCheckBox** instances

Set **JTextField** font to Serif, 14-point plain

Instantiate **JCheckBox**s for bolding and italicizing **JTextField** text, respectively

© 2002 Prentice Hall, Inc. All rights reserved.

```
36
37          // register listeners for JCheckBoxes
38          CheckBoxHandler handler = new CheckBoxHandler();
39          bold.addItemListener( handler );
40          italic.addItemListener( handler );
41
42          setSize( 275, 100 );
43          setVisible( true );
44       }
45
46       // execute application
47       public static void main( String args[] )
48       {
49          CheckBoxTest application = new CheckBoxTest();
50
51          application.setDefaultCloseOperation(
52             JFrame.EXIT_ON_CLOSE );
53       }
54
55       // private inner class for ItemListener event handling
56       private class CheckBoxHandler implements ItemListener {
57          private int valBold = Font.PLAIN;
58          private int valItalic = Font.PLAIN;
59
60          // respond to checkbox events
61          public void itemStateChanged( ItemEvent event )
62          {
63             // process bold checkbox events
64             if ( event.getSource() == bold )
65
66                if ( event.getStateChange() == ItemEvent.SELECTED )
67                   valBold = Font.BOLD;
68                else
69                   valBold = Font.PLAIN;
70
```

Register **JCheckBox**s to receive events from **CheckBoxHandler**

When user selects **JCheckBox**, **CheckBoxHandler** invokes method **itemStateChanged** of all registered listeners

Outline

Line 61

© 2002 Prentice Hall, Inc. All rights reserved.

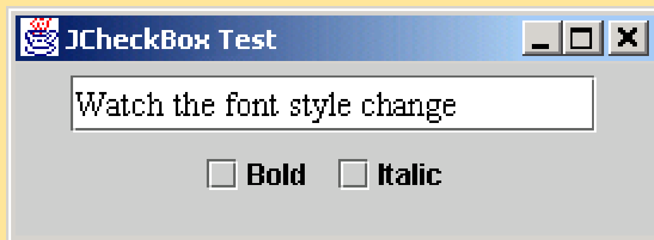
Outline

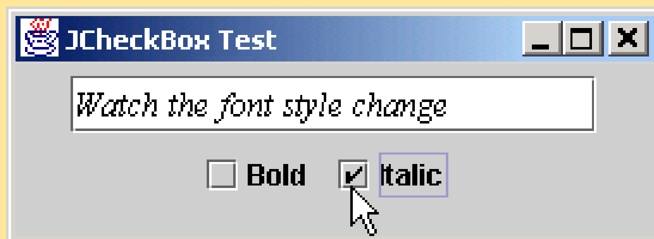
CheckBoxTest.java

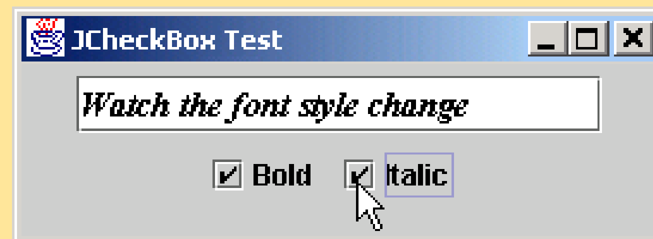
Lines 80-81

```
71          // process italic checkbox events
72          if ( event.getSource() == italic )
73
74             if ( event.getStateChange() == ItemEvent.SELECTED )
75                valItalic = Font.ITALIC;
76             else
77                valItalic = Font.PLAIN;
78
79          // set text field font
80          field.setFont(
81             new Font( "Serif", valBold + valItalic, 14 ) );
82       }
83
84    }  // end private inner class CheckBoxHandler
85
86 }  // end class CheckBoxTest
```

Change **JTextField** font, depending on which **JCheckBox** was selected







© 2002 Prentice Hall, Inc. All rights reserved.

```
1  // Fig. 12.12: RadioButtonTest.java
2  // Creating radio buttons using ButtonGroup and JRadioButton.
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class RadioButtonTest extends JFrame {
12    private JTextField field;
13    private Font plainFont, boldFont, italicFont, boldItalicFont;
14    private JRadioButton plainButton, boldButton, italicButton,
15       boldItalicButton;
16    private ButtonGroup radioGroup;
17
18    // create GUI and fonts
19    public RadioButtonTest()
20    {
21       super( "RadioButton Test" );
22
23       // get content pane and set its layout
24       Container container = getContentPane();
25       container.setLayout( new FlowLayout() );
26
27       // set up JTextField
28       field =
29          new JTextField( "Watch the font style change", 25 );
30       container.add( field );
31
32       // create radio buttons
33       plainButton = new JRadioButton( "Plain", true );
34       container.add( plainButton );
35
```

Outline

RadioButtonTest.java

Lines 14-15

Line 16

Declare four **JRadioButton** instances

JRadioButtons normally appear as a **ButtonGroup**

© 2002 Prentice Hall, Inc. All rights reserved.

```
36       boldButton = new JRadioButton( "Bold", false);
37       container.add( boldButton );
38
39       italicButton = new JRadioButton( "Italic", false );
40       container.add( italicButton );
41
42       boldItalicButton = new JRadioButton(
43          "Bold/Italic", false );
44       container.add( boldItalicButton );
45
46       // register events for JRadioButtons
47       RadioButtonHandler handler = new RadioButtonHandler();
48       plainButton.addItemListener( handler );
49       boldButton.addItemListener( handler );
50       italicButton.addItemListener( handler );
51       boldItalicButton.addItemListener( handler );
52
53       // create logical relationship between JRadioButtons
54       radioGroup = new ButtonGroup();
55       radioGroup.add( plainButton );
56       radioGroup.add( boldButton );
57       radioGroup.add( italicButton );
58       radioGroup.add( boldItalicButton );
59
60       // create font objects
61       plainFont = new Font( "Serif", Font.PLAIN, 14 );
62       boldFont = new Font( "Serif", Font.BOLD, 14 );
63       italicFont = new Font( "Serif", Font.ITALIC, 14 );
64       boldItalicFont =
65          new Font( "Serif", Font.BOLD + Font.ITALIC, 14 );
66       field.setFont( plainFont );
67
68       setSize( 300, 100 );
69       setVisible( true );
70    }
```

Instantiate **JRadioButtons** for manipulating **JTextField** text font

Register **JRadioButtons** to receive events from **RadioButtonHandler**

JRadioButtons belong to **ButtonGroup**

Outline

RadioButtonTest.java

Lines 36-44

Lines 54-58

© 2002 Prentice Hall, Inc. All rights reserved.

```
71
72    // execute application
73    public static void main( String args[] )
74    {
75       RadioButtonTest application = new RadioButtonTest();
76
77       application.setDefaultCloseOperation(
78          JFrame.EXIT_ON_CLOSE );
79    }
80
81    // private inner class to handle radio button events
82    private class RadioButtonHandler implements ItemListener {
83
84       // handle radio button events
85       public void itemStateChanged( ItemEvent event )
86       {
87          // user clicked plainButton
88          if ( event.getSource() == plainButton )
89             field.setFont( plainFont );
90
91          // user clicked boldButton
92          else if ( event.getSource() == boldButton )
93             field.setFont( boldFont );
94
95          // user clicked italicButton
96          else if ( event.getSource() == italicButton )
97             field.setFont( italicFont );
98
99          // user clicked boldItalicButton
100          else if ( event.getSource() == boldItalicButton )
101             field.setFont( boldItalicFont );
102       }
103
104    }  // end private inner class RadioButtonHandler
105
106 }  // end class RadioButtonTest
```

Outline

RadioButtonTest.java

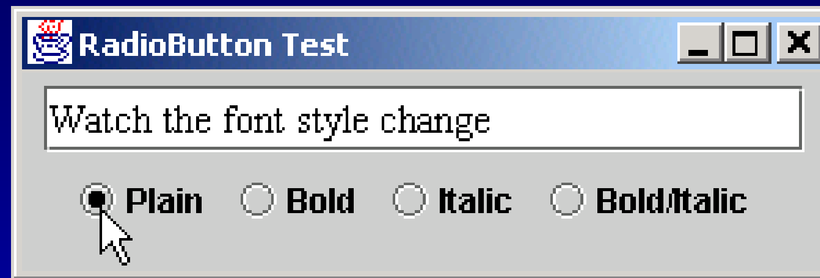
Lines 85-104

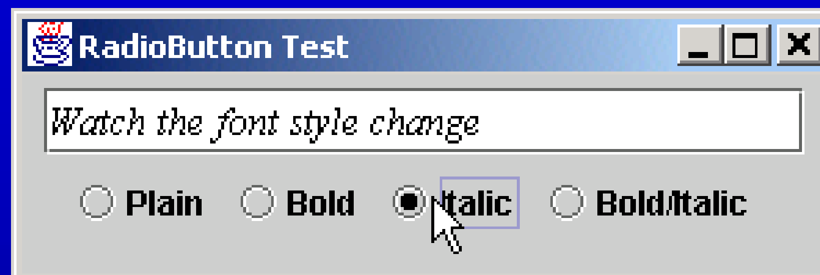
Lines 88-102

When user selects **JRadioButton**, **RadioButtonHandler** invokes method **itemStateChanged** of all registered listeners

Set font corresponding to **JRadioButton** selected

© 2002 Prentice Hall, Inc. All rights reserved.





© 2002 Prentice Hall, Inc. All rights reserved.

12.8 `JComboBox`

- **`JComboBox`**
  - List of items from which user can select
  - Also called a *drop-down list*

© 2002 Prentice Hall, Inc. All rights reserved.

```
1  // Fig. 12.13: ComboBoxTest.java
2  // Using a JComboBox to select an image to display.
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class ComboBoxTest extends JFrame {
12    private JComboBox imagesComboBox;
13    private JLabel label;
14
15    private String names[] =
16       { "bug1.gif", "bug2.gif",  "travelbug.gif", "buganim.gif" };
17    private Icon icons[] = { new ImageIcon( names[ 0 ] ),
18       new ImageIcon( names[ 1 ] ), new ImageIcon( names[ 2 ] ),
19       new ImageIcon( names[ 3 ] ) };
20
21    // set up GUI
22    public ComboBoxTest()
23    {
24       super( "Testing JComboBox" );
25
26       // get content pane and set its layout
27       Container container = getContentPane();
28       container.setLayout( new FlowLayout() );
29
30       // set up JComboBox and register its event handler
31       imagesComboBox = new JComboBox( names );
32       imagesComboBox.setMaximumRowCount( 3 );
33
34       imagesComboBox.addItemListener(
35
```

Instantiate **JComboBox** to show three **String**s from **names** array at a time

Register **JComboBox** to receive events from anonymous **ItemListener**

Outline

ComboBoxTest.java

Lines 31-32

Line 34

© 2002 Prentice Hall, Inc. All rights reserved.

```
36          // anonymous inner class to handle JComboBox events
37          new ItemListener() {
38
39             // handle JComboBox event
40             public void itemStateChanged( ItemEvent event )
41             {
42                // determine whether check box selected
43                if ( event.getStateChange() == ItemEvent.SELECTED )
44                   label.setIcon( icons[
45                      imagesComboBox.getSelectedIndex() ] );
46             }
47
48          }  // end anonymous inner class
49
50       ); // end call to addItemListener
51
52       container.add( imagesComboBox );
53
54       // set up JLabel to display ImageIcons
55       label = new JLabel( icons[ 0 ] );
56       container.add( label );
57
58       setSize( 350, 100 );
59       setVisible( true );
60    }
61
62    // execute application
63    public static void main( String args[] )
64    {
65       ComboBoxTest application = new ComboBoxTest();
66
67       application.setDefaultCloseOperation(
68          JFrame.EXIT_ON_CLOSE );
69    }
70
71 }  // end class ComboBoxTest
```

Outline

ComboBoxTest.java

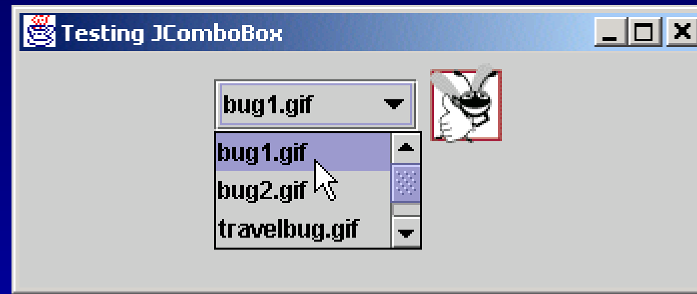
Lines 40-46

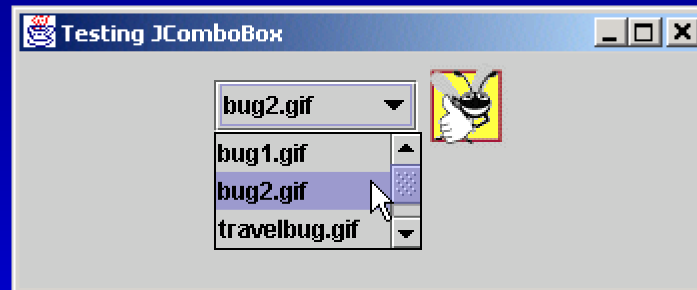
Lines 43-45

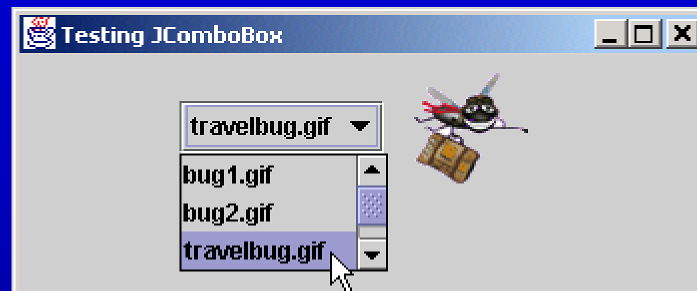
When user selects item in **JComboBox**, **ItemListener** invokes method **itemStateChanged** of all registered listeners

Set appropriate **Icon** depending on user selection

© 2002 Prentice Hall, Inc. All rights reserved.







© 2002 Prentice Hall, Inc. All rights reserved.

12.9 JList

- List
  - Series of items
  - user can select one or more items
  - Single-selection vs. multiple-selection
  - **JList**

© 2002 Prentice Hall, Inc. All rights reserved.

```
1  // Fig. 12.14: ListTest.java
2  // Selecting colors from a JList.
3
4  // Java core packages
5  import java.awt.*;
6
7  // Java extension packages
8  import javax.swing.*;
9  import javax.swing.event.*;
10
11 public class ListTest extends JFrame {
12    private JList colorList;
13    private Container container;
14
15    private String colorNames[] = { "Black", "Blue", "Cyan",
16       "Dark Gray", "Gray", "Green", "Light Gray", "Magenta",
17       "Orange", "Pink", "Red", "White", "Yellow" };
18
19    private Color colors[] = { Color.black, Color.blue,
20       Color.cyan, Color.darkGray, Color.gray, Color.green,
21       Color.lightGray, Color.magenta, Color.orange, Color.pink,
22       Color.red, Color.white, Color.yellow };
23
24    // set up GUI
25    public ListTest()
26    {
27       super( "List Test" );
28
29       // get content pane and set its layout
30       container = getContentPane();
31       container.setLayout( new FlowLayout() );
32
33       // create a list with items in colorNames array
34       colorList = new JList( colorNames );
35       colorList.setVisibleRowCount( 5 );
```

Outline

ListTest.java

Line 34

Use **colorNames** array to populate **JList**

© 2002 Prentice Hall, Inc. All rights reserved.

```
36
37       // do not allow multiple selections
38       colorList.setSelectionMode(
39          ListSelectionModel.SINGLE_SELECTION );
40
41       // add a JScrollPane containing JList to content pane
42       container.add( new JScrollPane( colorList ) );
43
44       // set up event handler
45       colorList.addListSelectionListener(
46
47          // anonymous inner class for list selection events
48          new ListSelectionListener() {
49
50             // handle list selection events
51             public void valueChanged( ListSelectionEvent event )
52             {
53                container.setBackground(
54                   colors[ colorList.getSelectedIndex() ] );
55             }
56
57          }  // end anonymous inner class
58
59       ); // end call to addListSelectionListener
60
61       setSize( 350, 150 );
62       setVisible( true );
63    }
64
65    // execute application
66    public static void main( String args[] )
67    {
68       ListTest application = new ListTest();
69
```

Outline

ListTest.java

Lines 38-39

Lines 53-54

JList allows single selections

Register **JList** to receive events from anonymous **ListSelectionListener**

When user selects item in **JList**, **ListSelectionListener** invokes method **valueChanged** of all registered listeners

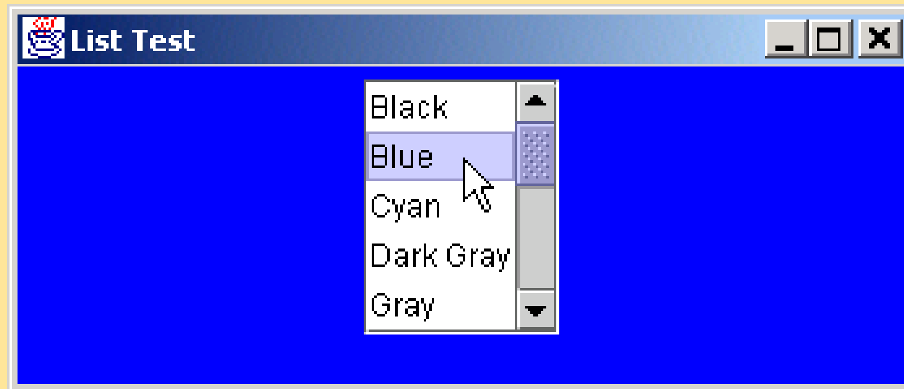
Set appropriate background depending on user selection

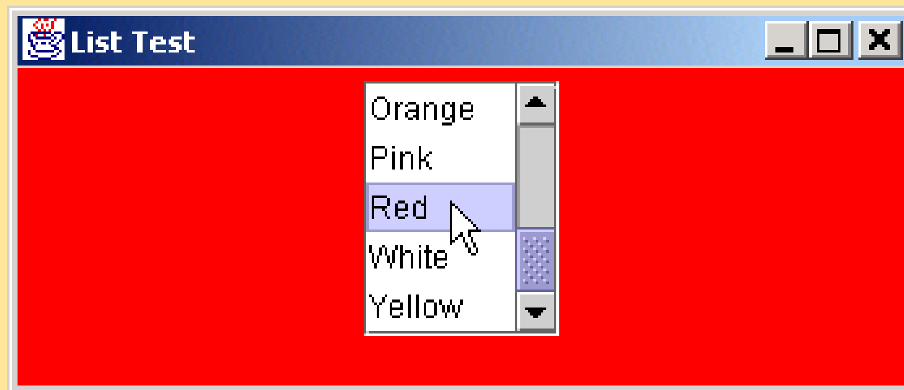
© 2002 Prentice Hall, Inc. All rights reserved.

Outline

ListTest.java

```
70         application.setDefaultCloseOperation(
71            JFrame.EXIT_ON_CLOSE );
72     }
73
74  }  // end class ListTest
```





© 2002 Prentice Hall, Inc. All rights reserved.

12.10 Multiple-Selection Lists

- Multiple-selection list
  - Select many items from **`Jlist`**
  - Allows continuous range selection

© 2002 Prentice Hall, Inc. All rights reserved.

```
 1  // Fig. 12.15: MultipleSelection.java
 2  // Copying items from one List to another.
 3
 4  // Java core packages
 5  import java.awt.*;
 6  import java.awt.event.*;
 7
 8  // Java extension packages
 9  import javax.swing.*;
10
11 public class MultipleSelection extends JFrame {
12    private JList colorList, copyList;
13    private JButton copyButton;
14
15    private String colorNames[] = { "Black", "Blue", "Cyan",
16       "Dark Gray", "Gray", "Green", "Light Gray",
17       "Magenta", "Orange", "Pink", "Red", "White", "Yellow" };
18
19    // set up GUI
20    public MultipleSelection()
21    {
22       super( "Multiple Selection Lists" );
23
24       // get content pane and set its layout
25       Container container = getContentPane();
26       container.setLayout( new FlowLayout() );
27
28       // set up JList colorList
29       colorList = new JList( colorNames );
30       colorList.setVisibleRowCount( 5 );
31       colorList.setFixedCellHeight( 15 );
32       colorList.setSelectionMode(
33          ListSelectionModel.MULTIPLE_INTERVAL_SELECTION );
34       container.add( new JScrollPane( colorList ) );
35
```


Outline

MultipleSelection.java

Line 29

Lines 32-33

© 2002 Prentice Hall, Inc. All rights reserved.

```
36        // create copy button and register its listener
37        copyButton = new JButton( "Copy >>>" );
38
39        copyButton.addActionListener(
40
41           // anonymous inner class for button event
42           new ActionListener() {
43
44              // handle button event
45              public void actionPerformed( ActionEvent event )
46              {
47                 // place selected values in copyList
48                 copyList.setListData(
49                    colorList.getSelectedValues() );
50              }
51
52           }  // end anonymous inner class
53
54        ); // end call to addActionListener
55
56        container.add( copyButton );
57
58        // set up JList copyList
59        copyList = new JList();
60        copyList.setVisibleRowCount( 5 );
61        copyList.setFixedCellWidth( 100 );
62        copyList.setFixedCellHeight( 15 );
63        copyList.setSelectionMode(
64           ListSelectionModel.SINGLE_INTERVAL_SELECTION );
65        container.add( new JScrollPane( copyList ) );
66
67        setSize( 300, 120 );
68        setVisible( true );
69     }
70
```

When user presses **`JButton`**, **`JList`** **`copyList`** adds items that user selected from **`JList colorList`**

`JList colorList` allows single selections

Outline

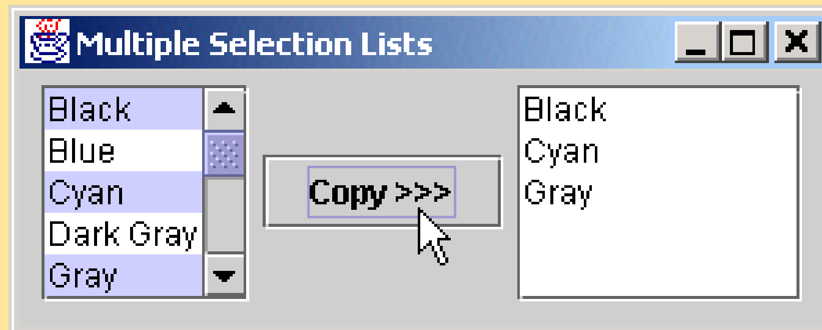
MultipleSelection.java

Lines 48-49

Lines 63-64

© 2002 Prentice Hall, Inc. All rights reserved.

```
71     // execute application
72     public static void main( String args[] )
73     {
74        MultipleSelection application = new MultipleSelection();
75
76        application.setDefaultCloseOperation(
77           JFrame.EXIT_ON_CLOSE );
78     }
79
80  }  // end class MultipleSelection
```



Outline

MultipleSelection.java

© 2002 Prentice Hall, Inc. All rights reserved.

12.11 Mouse Event Handling

- Event-listener interfaces for mouse events
  - **`MouseListener`**
  - **`MouseMotionListener`**
  - Listen for **`MouseEvent`**s

© 2002 Prentice Hall, Inc. All rights reserved.

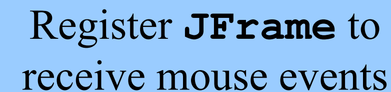
Fig. 12.16 MouseListener and MouseMotionListener interface methods

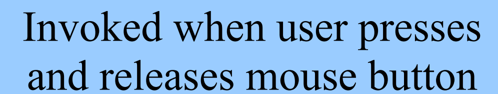
MouseListener and MouseMotionListener interface methods	
Methods of interface ***MouseListener***	
`public void mousePressed( MouseEvent event )`	Called when a mouse button is pressed with the mouse cursor on a component.
`public void mouseClicked( MouseEvent event )`	Called when a mouse button is pressed and released on a component without moving the mouse cursor.
`public void mouseReleased( MouseEvent event )`	Called when a mouse button is released after being pressed. This event is always preceded by a **mousePressed** event.
`public void mouseEntered( MouseEvent event )`	Called when the mouse cursor enters the bounds of a component.
`public void mouseExited( MouseEvent event )`	Called when the mouse cursor leaves the bounds of a component.
Methods of interface ***MouseMotionListener***	
`public void mouseDragged( MouseEvent event )`	Called when the mouse button is pressed with the mouse cursor on a component and the mouse is moved. This event is always preceded by a call to **mousePressed**.
`public void mouseMoved( MouseEvent event )`	Called when the mouse is moved with the mouse cursor on a component.

Fig. 12.16 MouseListener and MouseMotionListener interface methods.

© 2002 Prentice Hall, Inc. All rights reserved.

```java
 1  // Fig. 12.17: MouseTracker.java
 2  // Demonstrating mouse events.
 3
 4  // Java core packages
 5  import java.awt.*;
 6  import java.awt.event.*;
 7
 8  // Java extension packages
 9  import javax.swing.*;
10
11  public class MouseTracker extends JFrame
12     implements MouseListener, MouseMotionListener {
13
14     private JLabel statusBar;
15
16     // set up GUI and register mouse event handlers
17     public MouseTracker()
18     {
19        super( "Demonstrating Mouse Events" );
20
21        statusBar = new JLabel();
22        getContentPane().add( statusBar, BorderLayout.SOUTH );
23
24        // application listens to its own mouse events
25        addMouseListener( this );
26        addMouseMotionListener( this );
27
28        setSize( 275, 100 );
29        setVisible( true );
30     }
31
32     // MouseListener event handlers
33
34     // handle event when mouse released immediately after press
35     public void mouseClicked( MouseEvent event )
```





Outline

MouseTracker.java

Lines 25-26

Line 35

© 2002 Prentice Hall, Inc. All rights reserved.

```
36    {
37       statusBar.setText( "Clicked at [" + event.getX() +
38          ", " + event.getY() + "]" );
39    }
40
41    // handle event when mouse pressed
42    public void mousePressed( MouseEvent event )
43    {
44       statusBar.setText( "Pressed at [" + event.getX() +
45          ", " + event.getY() + "]" );
46    }
47
48    // handle event when mouse released after dragging
49    public void mouseReleased( MouseEvent event )
50    {
51       statusBar.setText( "Released at [" + event.getX() +
52          ", " + event.getY() + "]" );
53    }
54
55    // handle event when mouse enters area
56    public void mouseEntered( MouseEvent event )
57    {
58       JOptionPane.showMessageDialog( null, "Mouse in window" );
59    }
60
61    // handle event when mouse exits area
62    public void mouseExited( MouseEvent event )
63    {
64       statusBar.setText( "Mouse outside window" );
65    }
66
67    // MouseMotionListener event handlers
68
69    // handle event when user drags mouse with button pressed
70    public void mouseDragged( MouseEvent event )
```

Outline

MouseTracker.java

Line 42

Invoked when user presses mouse button

Line 56

Invoked when user releases mouse button after dragging mouse

Line 70

Invoked when mouse cursor enters **JFrame**

Invoked when mouse cursor exits **JFrame**

Invoked when user drags mouse cursor

© 2002 Prentice Hall, Inc. All rights reserved.

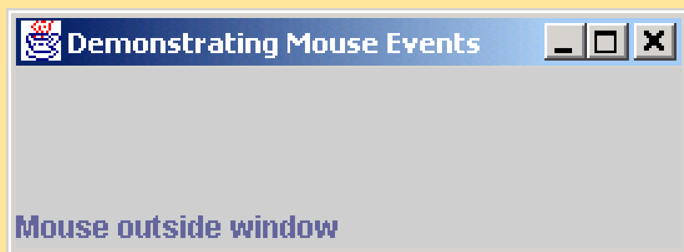
Outline

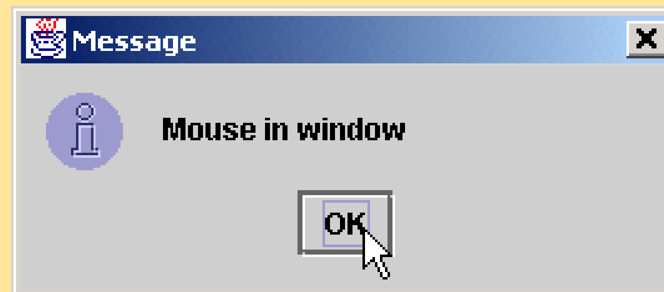
MouseTracker.java

Line 77

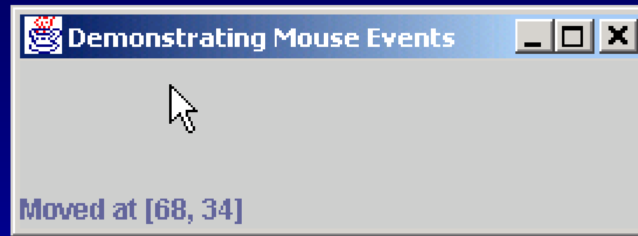
```
71    {
72       statusBar.setText( "Dragged at [" + event.getX() +
73          ", " + event.getY() + "]" );
74    }
75
76    // handle event when user moves mouse
77    public void mouseMoved( MouseEvent event )
78    {
79       statusBar.setText( "Moved at [" + event.getX() +
80          ", " + event.getY() + "]" );
81    }
82
83    // execute application
84    public static void main( String args[] )
85    {
86       MouseTracker application = new MouseTracker();
87
88       application.setDefaultCloseOperation(
89          JFrame.EXIT_ON_CLOSE );
90    }
91
92 }  // end class MouseTracker
```

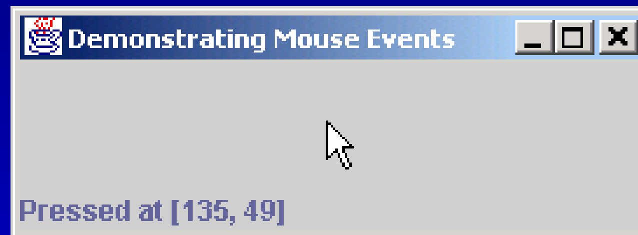
Invoked when user moves mouse cursor

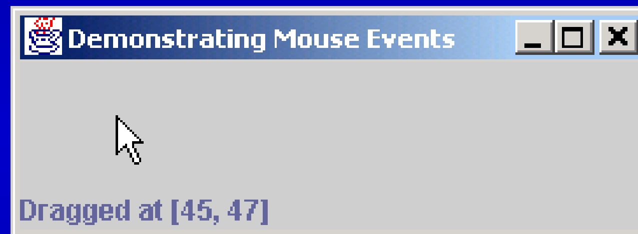


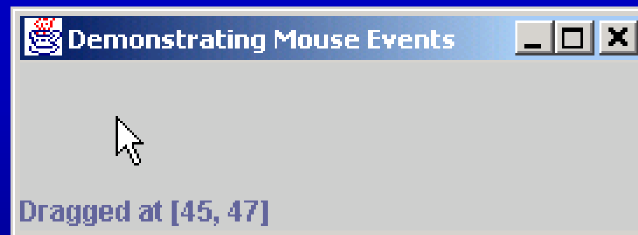


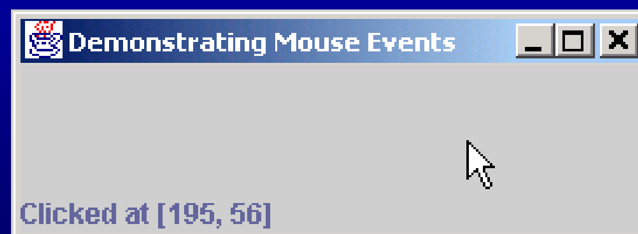
© 2002 Prentice Hall, Inc. All rights reserved.











© 2002 Prentice Hall, Inc. All rights reserved.

12.12 Adapter Classes

- Adapter class
  - Implements interface
  - Provides default implementation of each interface method
  - Used when all methods in interface is not needed

© 2002 Prentice Hall, Inc. All rights reserved.

Fig. 12.18 Event adapter classes and the interfaces they implement.

Event adapter class	Implements interface
`ComponentAdapter`	`ComponentListener`
`ContainerAdapter`	`ContainerListener`
`FocusAdapter`	`FocusListener`
`KeyAdapter`	`KeyListener`
`MouseAdapter`	`MouseListener`
`MouseMotionAdapter`	`MouseMotionListener`
`WindowAdapter`	`WindowListener`

Fig. 12.18 Event adapter classes and the interfaces they implement.

© 2002 Prentice Hall, Inc. All rights reserved.

```
 1 // Fig. 12.19: Painter.java
 2 // Using class MouseMotionAdapter.
 3
 4 // Java core packages
 5 import java.awt.*;
 6 import java.awt.event.*;
 7
 8 // Java extension packages
 9 import javax.swing.*;
10
11 public class Painter extends JFrame {
12    private int xValue = -10, yValue = -10;
13
14    // set up GUI and register mouse event handler
15    public Painter()
16    {
17       super( "A simple paint program" );
18
19       // create a label and place it in SOUTH of BorderLayout
20       getContentPane().add(
21          new Label( "Drag the mouse to draw" ),
22          BorderLayout.SOUTH );
23
24       addMouseMotionListener(
25
26          // anonymous inner class
27          new MouseMotionAdapter() {
28
29             // store drag coordinates and repaint
30             public void mouseDragged( MouseEvent event )
31             {
32                xValue = event.getX();
33                yValue = event.getY();
34                repaint();
35             }
```

Register **MouseMotionListener** to listen for window's mouse-motion events

Override method **mouseDragged**, but not method **mouseMoved**

Store coordinates where mouse was dragged, then repaint **JFrame**

Outline

Painter.java

Line 24

Lines 30-35

Lines 32-34

© 2002 Prentice Hall, Inc. All rights reserved.

Outline

Painter.java

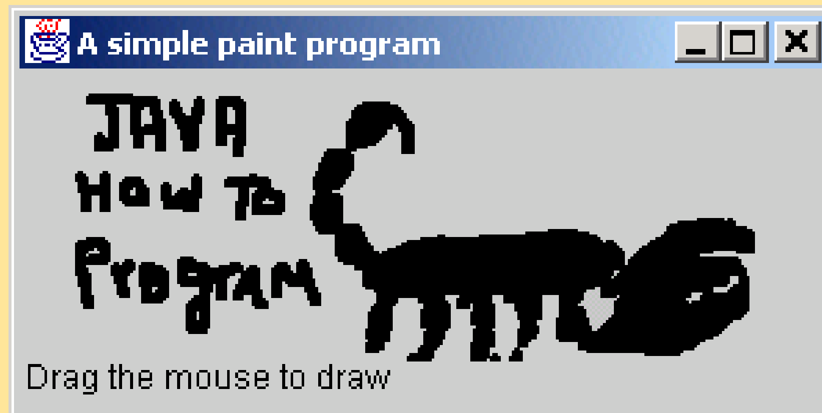
Line 52

```
36
37            }  // end anonymous inner class
38
39         ); // end call to addMouseMotionListener
40
41         setSize( 300, 150 );
42         setVisible( true );
43      }
44
45      // draw oval in a 4-by-4 bounding box at the specified
46      // location on the window
47      public void paint( Graphics g )
48      {
49         // we purposely did not call super.paint( g ) here to
50         // prevent repainting
51
52         g.fillOval( xValue, yValue, 4, 4 );
53      }
54
55      // execute application
56      public static void main( String args[] )
57      {
58         Painter application = new Painter();
59
60         application.addWindowListener(
61
62            // adapter to handle only windowClosing event
63            new WindowAdapter() {
64
65               public void windowClosing( WindowEvent event )
66               {
67                  System.exit( 0 );
68               }
69
```

Draw circle of diameter **4** where user dragged cursor

© 2002 Prentice Hall, Inc. All rights reserved.

```
70             }  // end anonymous inner class
71
72          ); // end call to addWindowListener
73       }
74
75 }  // end class Painter
```



Outline

Painter.java

© 2002 Prentice Hall, Inc. All rights reserved.

Outline

MouseDetails.java

Line 21

```
1  // Fig. 12.20: MouseDetails.java
2  // Demonstrating mouse clicks and
3  // distinguishing between mouse buttons.
4
5  // Java core packages
6  import java.awt.*;
7  import java.awt.event.*;
8
9  // Java extension packages
10 import javax.swing.*;
11
12 public class MouseDetails extends JFrame {
13    private int xPos, yPos;
14
15    // set title bar String, register mouse listener and size
16    // and show window
17    public MouseDetails()
18    {
19       super( "Mouse clicks and buttons" );
20
21       addMouseListener( new MouseClickHandler() );
22
23       setSize( 350, 150 );
24       setVisible( true );
25    }
26
27    // draw String at location where mouse was clicked
28    public void paint( Graphics g )
29    {
30       // call superclass's paint method
31       super.paint( g );
32
33       g.drawString( "Clicked @ [" + xPos + ", " + yPos + "]",
34          xPos, yPos );
35    }
```

Register mouse listener

© 2002 Prentice Hall, Inc. All rights reserved.

```
36
37    // execute application
38    public static void main( String args[] )
39    {
40       MouseDetails application = new MouseDetails();
41
42       application.setDefaultCloseOperation(
43          JFrame.EXIT_ON_CLOSE );
44    }
45
46    // inner class to handle mouse events
47    private class MouseClickHandler extends MouseAdapter {
48
49       // handle mouse click event and determine which mouse
50       // button was pressed
51       public void mouseClicked( MouseEvent event )
52       {
53          xPos = event.getX();
54          yPos = event.getY();
55
56          String title =
57             "Clicked " + event.getClickCount() + " time(s)";
58
59          // right mouse button
60          if ( event.isMetaDown() )
61             title += " with right mouse button";
62
63          // middle mouse button
64          else if ( event.isAltDown() )
65             title += " with center mouse button";
66
67          // left mouse button
68          else
69             title += " with left mouse button";
```

Outline

MouseDetails.java

Line 51

Lines 53-54

Invoke method **mouseClicked** when user clicks mouse

Store mouse-cursor coordinates where mouse was clicked

Determine number of times user has clicked mouse

Determine if user clicked right mouse button

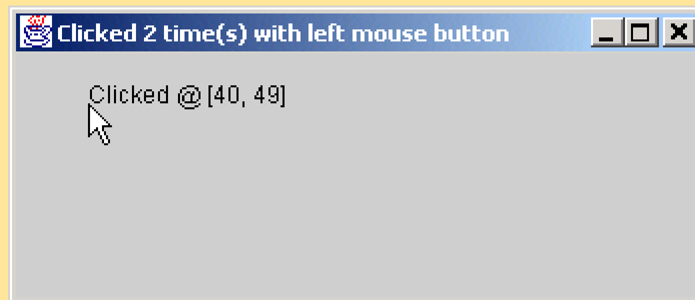
Determine if user clicked middle mouse button

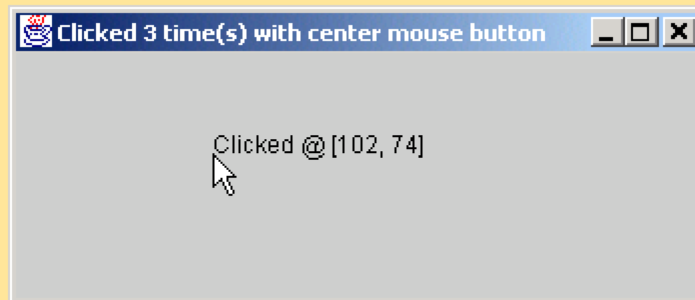
© 2002 Prentice Hall, Inc. All rights reserved.

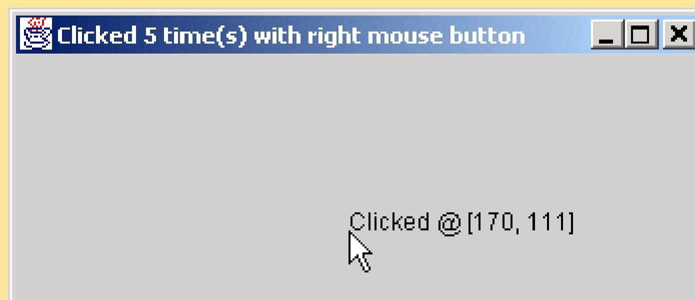
```
70
71          setTitle( title );  // set title bar of window
72          repaint();
73       }
74
75    }  // end private inner class MouseClickHandler
76
77 }  // end class MouseDetails
```

Outline

MouseDetails.java







© 2002 Prentice Hall, Inc. All rights reserved.

Fig. 12.21 `InputEvent` methods that help distinguish among left-, center- and right-mouse-button clicks.

`InputEvent` method	Description
`isMetaDown()`	This method returns **`true`** when the user clicks the right mouse button on a mouse with two or three buttons. To simulate a right-mouse-button click on a one-button mouse, the user can press the *Meta* key on the keyboard and click the mouse button.
`isAltDown()`	This method returns **`true`** when the user clicks the middle mouse button on a mouse with three buttons. To simulate a middle-mouse-button click on a one- or two-button mouse, the user can press the *Alt* key on the keyboard and click the mouse button.

Fig. 12.21 `InputEvent` methods that help distinguish among left-, center- and right-mouse-button clicks.

© 2002 Prentice Hall, Inc. All rights reserved.

12.22 Keyboard Event Handling

- Interface **`KeyListener`**
  - Handles *key events*
    - Generated when keys on keyboard are pressed and released
    - **`KeyEvent`**
      - Contains *virtual key code* that represents key

© 2002 Prentice Hall, Inc. All rights reserved.

```
1  // Fig. 12.22: KeyDemo.java
2  // Demonstrating keystroke events.
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class KeyDemo extends JFrame implements KeyListener {
12    private String line1 = "", line2 = "";
13    private String line3 = "";
14    private JTextArea textArea;
15
16    // set up GUI
17    public KeyDemo()
18    {
19       super( "Demonstrating Keystroke Events" );
20
21       // set up JTextArea
22       textArea = new JTextArea( 10, 15 );
23       textArea.setText( "Press any key on the keyboard..." );
24       textArea.setEnabled( false );
25       getContentPane().add( textArea );
26
27       // allow frame to process Key events
28       addKeyListener( this );
29
30       setSize( 350, 100 );
31       setVisible( true );
32    }
33
34    // handle press of any key
35    public void keyPressed( KeyEvent event )
```

Register **JFrame** for key events

Called when user presses key

Outline

KeyDemo.java

Line 28

Line 35

© 2002 Prentice Hall, Inc. All rights reserved.

```
36     {
37        line1 = "Key pressed: " +
38           event.getKeyText( event.getKeyCode() );
39        setLines2and3( event );
40     }
41
42     // handle release of any key
43     public void keyReleased( KeyEvent event )
44     {
45        line1 = "Key released: " +
46           event.getKeyText( event.getKeyCode() );
47        setLines2and3( event );
48     }
49
50     // handle press of an action key
51     public void keyTyped( KeyEvent event )
52     {
53        line1 = "Key typed: " + event.getKeyChar();
54        setLines2and3( event );
55     }
56
57     // set second and third lines of output
58     private void setLines2and3( KeyEvent event )
59     {
60        line2 = "This key is " +
61           ( event.isActionKey() ? "" : "not " ) +
62           "an action key";
63
64        String temp =
65           event.getKeyModifiersText( event.getModifiers() );
66
67        line3 = "Modifier keys pressed: " +
68           ( temp.equals( "" ) ? "none" : temp );
69
```

Called when user releases key

Return virtual key code

Called when user types key

Determine if *modifier keys* (e.g., *Alt*, *Ctrl*, *Meta* and *Shift*) were used

Outline

KeyDemo.java

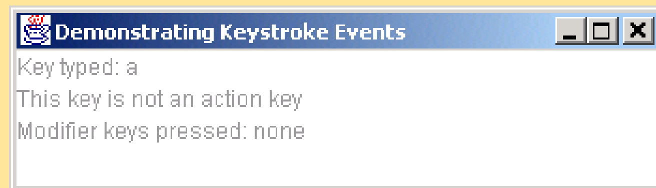
Line 43

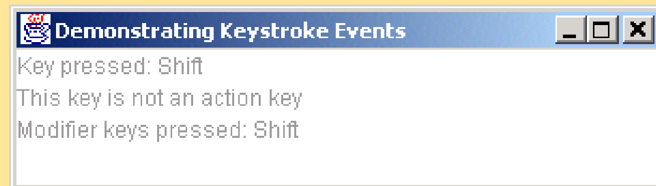
Line 46

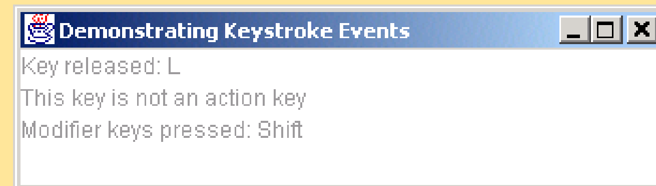
Lines 64-65

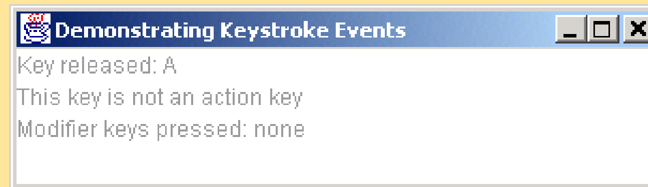
© 2002 Prentice Hall, Inc. All rights reserved.

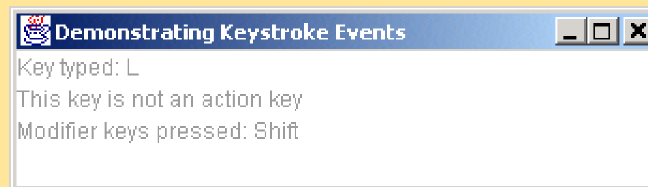
```
70          textArea.setText(
71             line1 + "\n" + line2 + "\n" + line3 + "\n" );
72       }
73 
74       // execute application
75       public static void main( String args[] )
76       {
77          KeyDemo application = new KeyDemo();
78 
79          application.setDefaultCloseOperation(
80             JFrame.EXIT_ON_CLOSE );
81       }
82 
83    }  // end class KeyDemo
```

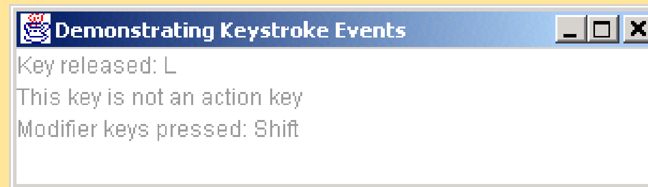


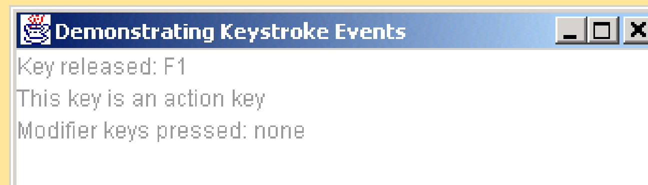












Outline

KeyDemo.java

© 2002 Prentice Hall, Inc. All rights reserved.

12.14 Layout Managers

- Layout managers
  - Provided for arranging GUI components
  - Provide basic layout capabilities
  - Processes layout details
  - Programmer can concentrate on basic “look and feel”
  - Interface **`LayoutManager`**

© 2002 Prentice Hall, Inc. All rights reserved.

Fig. 12.23 Layout managers.

Layout manager	Description
`FlowLayout`	Default for **`java.awt.Applet`**, **`java.awt.Panel`** and **`javax.swing.JPanel`**. Places components sequentially (left to right) in the order they were added. It is also possible to specify the order of the components using the **`Container`** method **`add`** that takes a **`Component`** and an integer index position as arguments.
`BorderLayout`	Default for the content panes of **`JFrame`**s (and other windows) and **`JApplet`**s. Arranges the components into five areas: North, South, East, West and Center.
`GridLayout`	Arranges the components into rows and columns.

Fig. 12.23 Layout managers.

© 2002 Prentice Hall, Inc. All rights reserved.

12.14.1 `FlowLayout`

- **`FlowLayout`**
  - Most basic layout manager
  - GUI components placed in container from left to right

© 2002 Prentice Hall, Inc. All rights reserved.

```
1  // Fig. 12.24: FlowLayoutDemo.java
2  // Demonstrating FlowLayout alignments.
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class FlowLayoutDemo extends JFrame {
12    private JButton leftButton, centerButton, rightButton;
13    private Container container;
14    private FlowLayout layout;
15
16    // set up GUI and register button listeners
17    public FlowLayoutDemo()
18    {
19       super( "FlowLayout Demo" );
20
21       layout = new FlowLayout();
22
23       // get content pane and set its layout
24       container = getContentPane();
25       container.setLayout( layout );
26
27       // set up leftButton and register listener
28       leftButton = new JButton( "Left" );
29
30       leftButton.addActionListener(
31
32          // anonymous inner class
33          new ActionListener() {
34
35             // process leftButton event
```

Outline

FlowLayoutDemo.java

Lines 21-25

© 2002 Prentice Hall, Inc. All rights reserved.

```
36             public void actionPerformed( ActionEvent event )
37             {
38                layout.setAlignment( FlowLayout.LEFT );
39
40                // re-align attached components
41                layout.layoutContainer( container );
42             }
43
44          }  // end anonymous inner class
45
46       ); // end call to addActionListener
47
48       container.add( leftButton );
49
50       // set up centerButton and register listener
51       centerButton = new JButton( "Center" );
52
53       centerButton.addActionListener(
54
55          // anonymous inner class
56          new ActionListener() {
57
58             // process centerButton event
59             public void actionPerformed( ActionEvent event )
60             {
61                layout.setAlignment( FlowLayout.CENTER );
62
63                // re-align attached components
64                layout.layoutContainer( container );
65             }
66          }
67       );
68
69       container.add( centerButton );
70
```


When user presses left **JButton**, left align components

When user presses center **JButton**, center components

© 2002 Prentice Hall, Inc. All rights reserved.

```
71       // set up rightButton and register listener
72       rightButton = new JButton( "Right" );
73
74       rightButton.addActionListener(
75
76          // anonymous inner class
77          new ActionListener() {
78
79             // process rightButton event
80             public void actionPerformed( ActionEvent event )
81             {
82                layout.setAlignment( FlowLayout.RIGHT );
83
84                // re-align attached components
85                layout.layoutContainer( container );
86             }
87          }
88       );
89
90       container.add( rightButton );
91
92       setSize( 300, 75 );
93       setVisible( true );
94    }
95
96    // execute application
97    public static void main( String args[] )
98    {
99       FlowLayoutDemo application = new FlowLayoutDemo();
100
101       application.setDefaultCloseOperation(
102          JFrame.EXIT_ON_CLOSE );
103    }
104
105 }  // end class FlowLayoutDemo
```

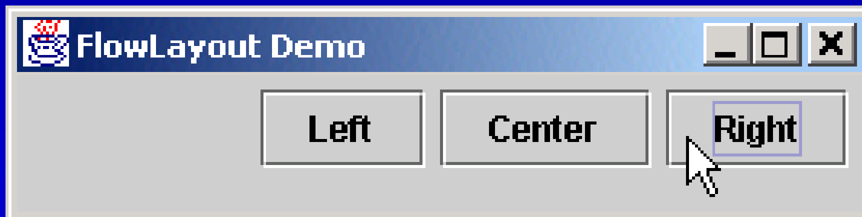
When user presses right **JButton**, right components

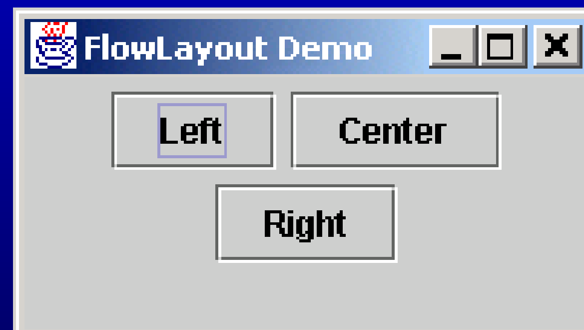
Outline

FlowLayoutDemo.java

Line 82

© 2002 Prentice Hall, Inc. All rights reserved.





© 2002 Prentice Hall, Inc. All rights reserved.

12.14.2 BorderLayout

- **BorderLayout**
  - Up to 5 components can be added
  - Arranges components into five regions
    - **NORTH** (top of container)
    - **SOUTH** (bottom of container)
    - **EAST** (left of container)
    - **WEST** (right of container)
    - **CENTER** (center of container)
  - North, south regions extends horizontally to the sides of the container
  - East, west regions extend vertically between the north and south regions
  - Center region expands to take all the remaining space in the layout

© 2002 Prentice Hall, Inc. All rights reserved.

Outline

BorderLayoutDemo.java

Lines 24-28

```
1  // Fig. 12.25: BorderLayoutDemo.java
2  // Demonstrating BorderLayout.
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class BorderLayoutDemo extends JFrame
12    implements ActionListener {
13
14    private JButton buttons[];
15    private String names[] = { "Hide North", "Hide South",
16       "Hide East", "Hide West", "Hide Center" };
17    private BorderLayout layout;
18
19    // set up GUI and event handling
20    public BorderLayoutDemo()
21    {
22       super( "BorderLayout Demo" );
23
24       layout = new BorderLayout( 5, 5 );
25
26       // get content pane and set its layout
27       Container container = getContentPane();
28       container.setLayout( layout );
29
30       // instantiate button objects
31       buttons = new JButton[ names.length ];
32
33       for ( int count = 0; count < names.length; count++ ) {
34          buttons[ count ] = new JButton( names[ count ] );
35          buttons[ count ].addActionListener( this );
```

Set layout as **BorderLayout** with 5-pixel horizontal and vertical gaps

© 2002 Prentice Hall, Inc. All rights reserved.

Outline

BorderLayoutDemo.java

Lines 54-57

```
36         }
37 
38         // place buttons in BorderLayout; order not important
39         container.add( buttons[ 0 ], BorderLayout.NORTH );
40         container.add( buttons[ 1 ], BorderLayout.SOUTH );
41         container.add( buttons[ 2 ], BorderLayout.EAST );
42         container.add( buttons[ 3 ], BorderLayout.WEST );
43         container.add( buttons[ 4 ], BorderLayout.CENTER );
44 
45         setSize( 300, 200 );
46         setVisible( true );
47      }
48 
49      // handle button events
50      public void actionPerformed( ActionEvent event )
51      {
52         for ( int count = 0; count < buttons.length; count++ )
53 
54            if ( event.getSource() == buttons[ count ] )
55               buttons[ count ].setVisible( false );
56            else
57               buttons[ count ].setVisible( true );
58 
59         // re-layout the content pane
60         layout.layoutContainer( getContentPane() );
61      }
62 
63      // execute application
64      public static void main( String args[] )
65      {
66         BorderLayoutDemo application = new BorderLayoutDemo();
```

Place **JButton**s in regions specified by **BorderLayout**

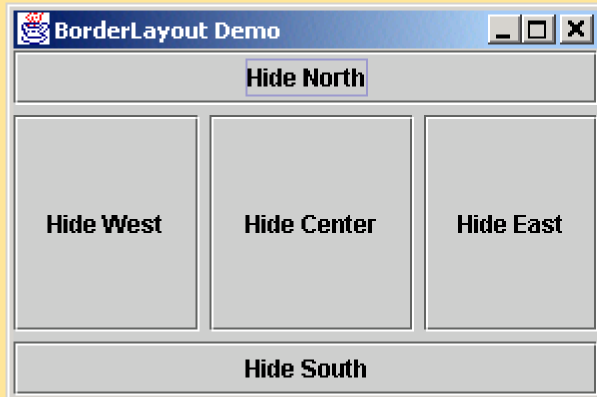
When **JButton**s are "invisible," they are not displayed on screen, and **BorderLayout** rearranges

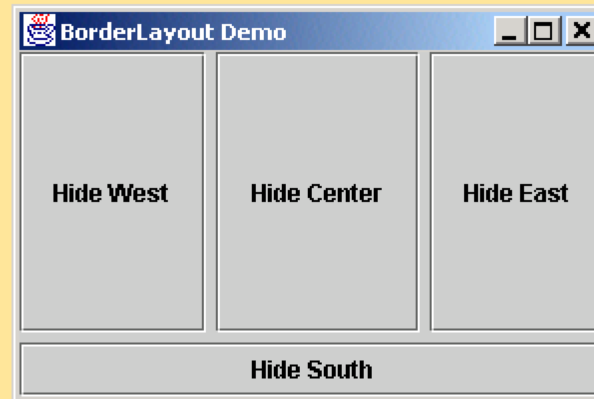
© 2002 Prentice Hall, Inc. All rights reserved.

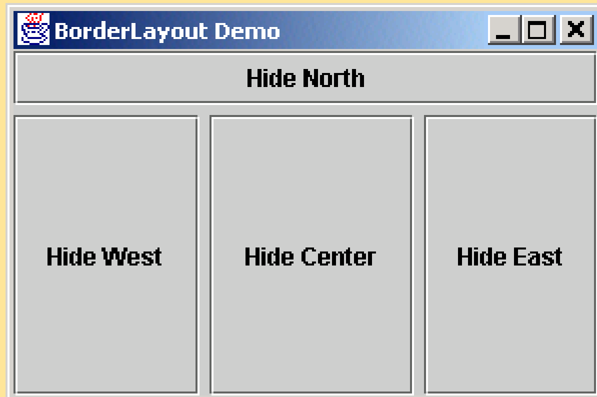
Outline

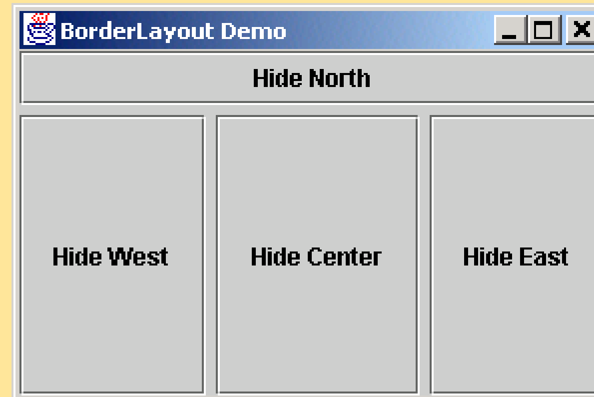
BorderLayoutDemo.java

```
67
68       application.setDefaultCloseOperation(
69          JFrame.EXIT_ON_CLOSE );
70    }
71
72 }  // end class BorderLayoutDemo
```

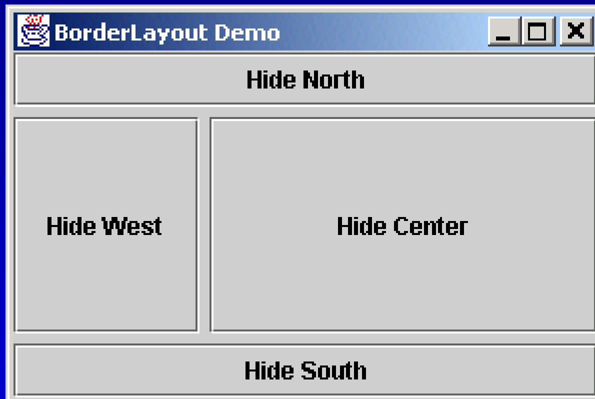


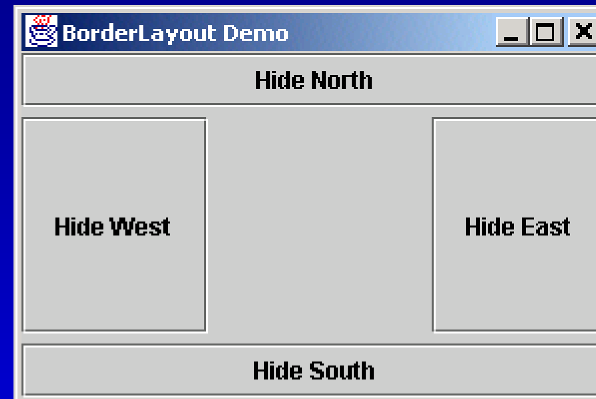






© 2002 Prentice Hall, Inc. All rights reserved.





© 2002 Prentice Hall, Inc. All rights reserved.

12.14.3 GridLayout

- **GridLayout**
  - Divides container into grid of specified row an columns
  - Components are added starting at top-left cell
    - Proceed left-to-fight until row is full
    - Each component has the same width and height
    - Components are added starting at the top, left cell of the grid proceeding left to right until the row is full. Then the next row is filled in the same way

© 2002 Prentice Hall, Inc. All rights reserved.

```
1  // Fig. 12.26: GridLayoutDemo.java
2  // Demonstrating GridLayout.
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class GridLayoutDemo extends JFrame
12    implements ActionListener {
13
14    private JButton buttons[];
15    private String names[] =
16       { "one", "two", "three", "four", "five", "six" };
17    private boolean toggle = true;
18    private Container container;
19    private GridLayout grid1, grid2;
20
21    // set up GUI
22    public GridLayoutDemo()
23    {
24       super( "GridLayout Demo" );
25
26       // set up layouts
27       grid1 = new GridLayout( 2, 3, 5, 5 );
28       grid2 = new GridLayout( 3, 2 );
29
30       // get content pane and set its layout
31       container = getContentPane();
32       container.setLayout( grid1 );
33
34       // create and add buttons
35       buttons = new JButton[ names.length ];
```

Outline

GridLayoutDemo.java

Line 27

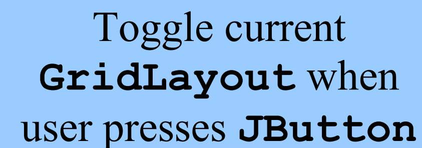
Line 28

Create **GridLayout grid1** with 2 rows and 3 columns

Create **GridLayout grid2** with 3 rows and 2 columns

© 2002 Prentice Hall, Inc. All rights reserved.

```
36
37        for ( int count = 0; count < names.length; count++ ) {
38           buttons[ count ] = new JButton( names[ count ] );
39           buttons[ count ].addActionListener( this );
40           container.add( buttons[ count ] );
41        }
42
43        setSize( 300, 150 );
44        setVisible( true );
45     }
46
47     // handle button events by toggling between layouts
48     public void actionPerformed( ActionEvent event )
49     {
50        if ( toggle )
51           container.setLayout( grid2 );
52        else
53           container.setLayout( grid1 );
54
55        toggle = !toggle;   // set toggle to opposite value
56        container.validate();
57     }
58
59     // execute application
60     public static void main( String args[] )
61     {
62        GridLayoutDemo application = new GridLayoutDemo();
63
64        application.setDefaultCloseOperation(
65           JFrame.EXIT_ON_CLOSE );
66     }
67
68  }  // end class GridLayoutDemo
```



Outline

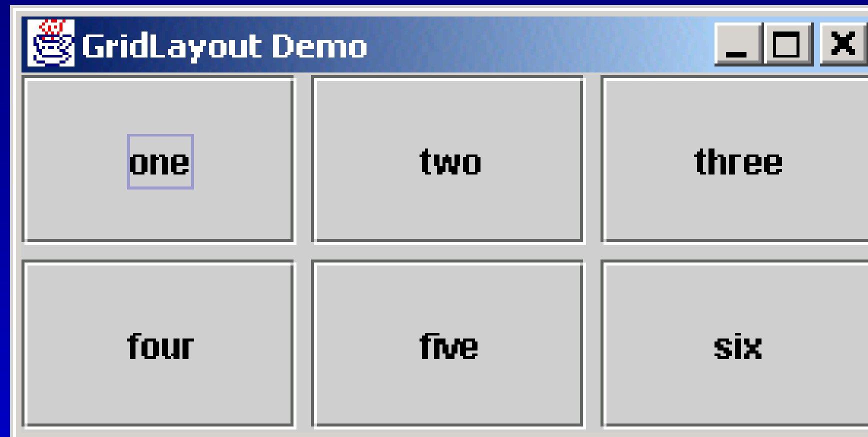
GridLayoutDemo.java

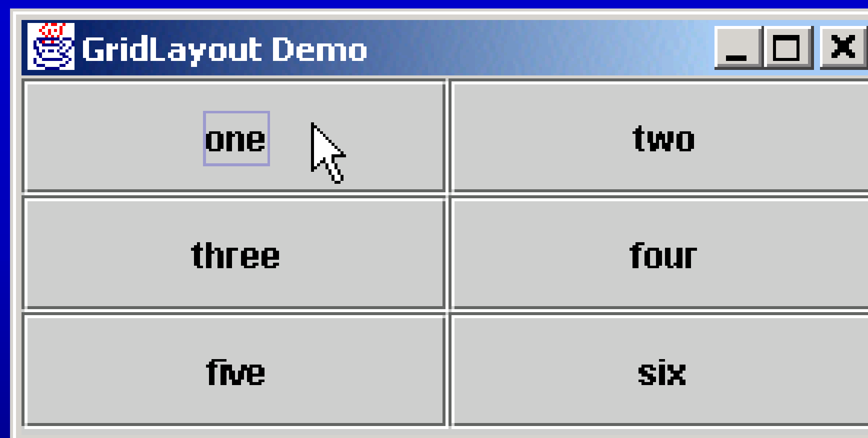
Lines 50-53

Line 56

Validate() recomputes the container's layout based on the current layout manager

© 2002 Prentice Hall, Inc. All rights reserved.





© 2002 Prentice Hall, Inc. All rights reserved.

12.15 Panels

- Panel
  - Complex GUIs require that each component be placed in an exact location
  - Use multiple panels with each panel's components arranged in a specific layout
  - Class **`JPanel`** is **`JComponent`** subclass
  - May have components (and other panels) added to them

© 2002 Prentice Hall, Inc. All rights reserved.

```
1  // Fig. 12.27: PanelDemo.java
2  // Using a JPanel to help lay out components.
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class PanelDemo extends JFrame {
12    private JPanel buttonPanel;
13    private JButton buttons[];
14
15    // set up GUI
16    public PanelDemo()
17    {
18       super( "Panel Demo" );
19
20       // get content pane
21       Container container = getContentPane();
22
23       // create buttons array
24       buttons = new JButton[ 5 ];
25
26       // set up panel and set its layout
27       buttonPanel = new JPanel();
28       buttonPanel.setLayout(
29          new GridLayout( 1, buttons.length ) );
30
31       // create and add buttons
32       for ( int count = 0; count < buttons.length; count++ ) {
33          buttons[ count ] =
34             new JButton( "Button " + ( count + 1 ) );
35          buttonPanel.add( buttons[ count ] );
```

Create **JPanel** to hold **JButtons**

Add **JButtons** to **JPanel**

Outline

PanelDemo.java

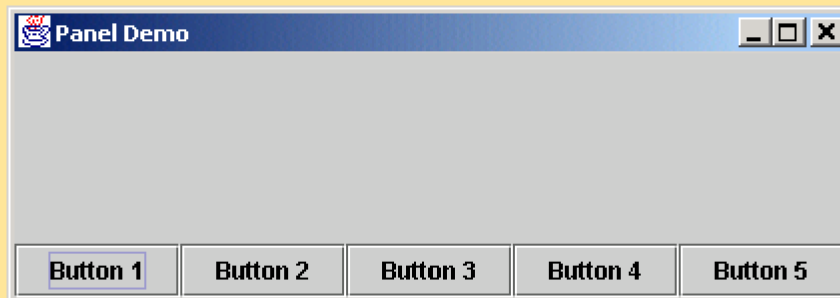
Line 27

Line 35

© 2002 Prentice Hall, Inc. All rights reserved.

```
36          }
37
38          container.add( buttonPanel, BorderLayout.SOUTH );
39
40          setSize( 425, 150 );
41          setVisible( true );
42       }
43
44       // execute application
45       public static void main( String args[] )
46       {
47          PanelDemo application = new PanelDemo();
48
49          application.setDefaultCloseOperation(
50             JFrame.EXIT_ON_CLOSE );
51       }
52
53    }  // end class PanelDemo
```

Add **JPanel** to **SOUTH** region of **Container**



Outline

PanelDemo.java

Line 38

© 2002 Prentice Hall, Inc. All rights reserved.