

11

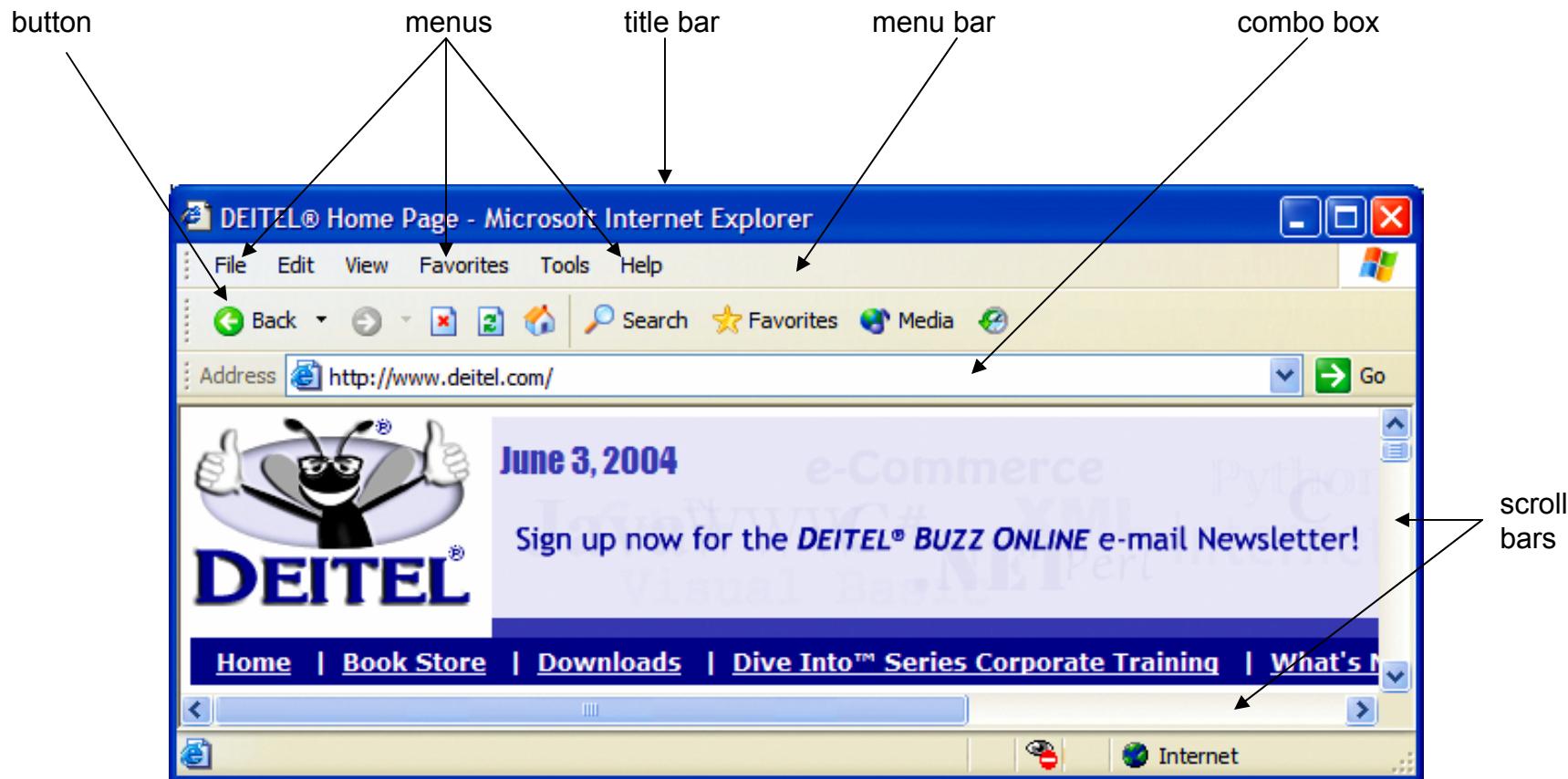
# GUI Components: Part 1



# 11.1 Introduction

- **Graphical user interface (GUI)**
  - Presents a user-friendly mechanism for interacting with an application
  - Often contains title bar, menu bar containing menus, buttons and combo boxes
  - Built from GUI components





**Fig. 11.1 | Internet Explorer window with GUI components.**

# 11.2 Simple GUI-Based Input/Output with JOptionPane

- **Dialog boxes**
  - Used by applications to interact with the user
  - Provided by Java's JOptionPane class
    - Contains input dialogs and message dialogs



```

1 // Fig. 11.2: Addition.java
2 // Addition program that uses JOptionPane for input and output.
3 import javax.swing.JOptionPane; // program uses JOptionPane
4
5 public class Addition
6 {
7     public static void main( String args[] )
8     {
9         // obtain user input from JOptionPane input dialogs
10        String firstNumber =
11            JOptionPane.showInputDialog( "Enter first integer" );
12        String secondNumber =
13            JOptionPane.showInputDialog( "Enter second integer" );
14
15        // convert String inputs to int values for use in a calculation
16        int number1 = Integer.parseInt( firstNumber );
17        int number2 = Integer.parseInt( secondNumber );
18
19        int sum = number1 + number2; // add numbers
20
21        // display result in a JOptionPane message dialog
22        JOptionPane.showMessageDialog( null, "The sum is " + sum,
23            "Sum of Two Integers", JOptionPane.PLAIN_MESSAGE );
24    } // end method main
25 } // end class Addition

```

## Outline

Show input dialog to receive first integer

Addition.java

(1 of 2)

Show input dialog to receive second integer

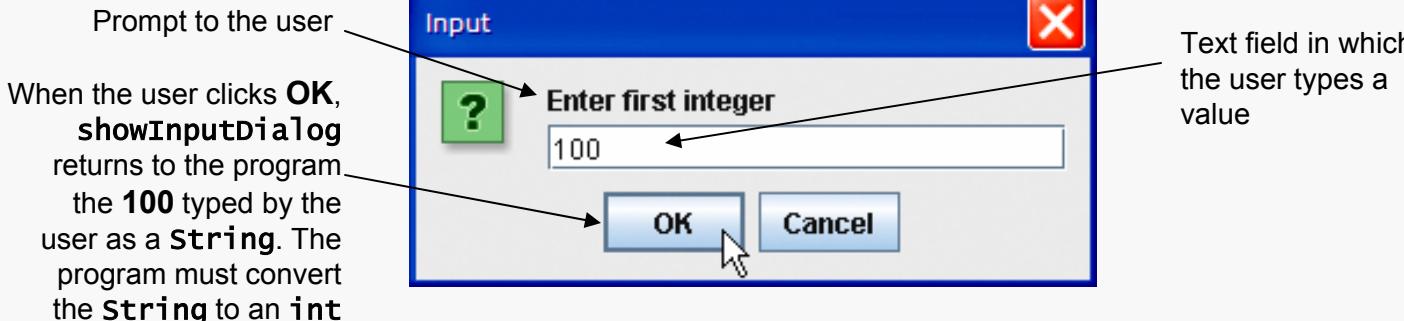
Show message dialog to output sum to user



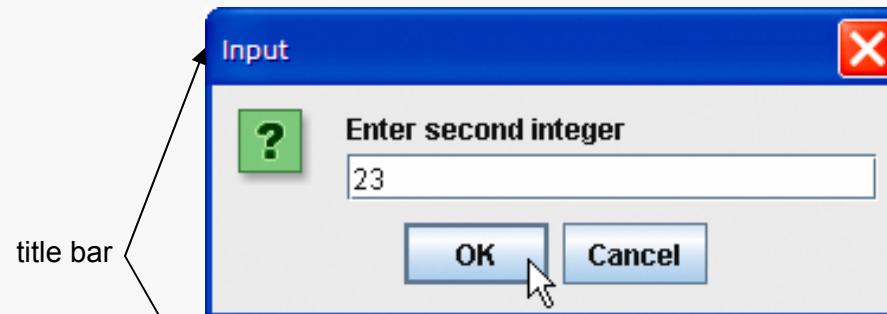
# Outline

## Addition.java (2 of 2)

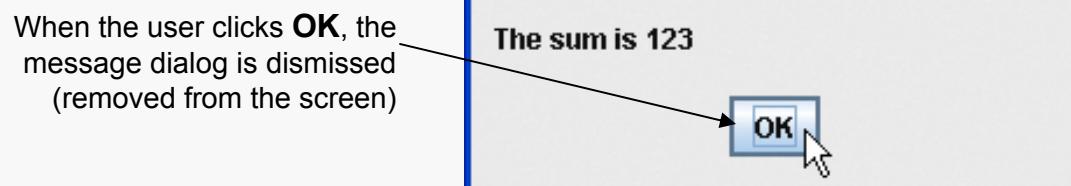
Input dialog displayed by lines 10–11



Input dialog displayed by lines 12–13



Message dialog displayed by lines 22–23



## Look-and-Feel Observation 11.2

---

The prompt in an input dialog typically uses *sentence-style capitalization*—a style that capitalizes only the first letter of the first word in the text unless the word is a proper noun (for example, Deitel).



## Look-and-Feel Observation 11.3

---

The title bar of a window typically uses *book-title capitalization*—a style that capitalizes the first letter of each significant word in the text and does not end with any punctuation (for example, Capitalization in a Book Title).



Message dialog type	Icon	Description
ERROR_MESSAGE		A dialog that indicates an error to the user.
INFORMATION_MESSAGE		A dialog with an informational message to the user.
WARNING_MESSAGE		A dialog warning the user of a potential problem.
QUESTION_MESSAGE		A dialog that poses a question to the user. This dialog normally requires a response, such as clicking a Yes or a No button.
PLAIN_MESSAGE	no icon	A dialog that contains a message, but no icon.

Fig. 11.3 | JOptionPane static constants for message dialogs.



# 11.3 Overview of Swing Components

- **Swing GUI components**
  - Declared in package `javax.swing`
  - Most are pure Java components
  - Part of the Java Foundation Classes (JFC)



Component	Description
JLabel	Displays uneditable text or icons.
JTextField	Enables user to enter data from the keyboard. Can also be used to display editable or uneditable text.
JButton	Triggers an event when clicked with the mouse.
JCheckBox	Specifies an option that can be selected or not selected.
JComboBox	Provides a drop-down list of items from which the user can make a selection by clicking an item or possibly by typing into the box.
JList	Provides a list of items from which the user can make a selection by clicking on any item in the list. Multiple elements can be selected.
JPanel	Provides an area in which components can be placed and organized. Can also be used as a drawing area for graphics.

**Fig. 11.4 | Some basic GUI components.**



# Swing vs. AWT

- **Abstract Window Toolkit (AWT)**
  - Precursor to Swing
  - Declared in package `java.awt`
  - Does not provide consistent, cross-platform look-and-feel



# Portability Tip 11.1

---

**Swing components are implemented in Java, so they are more portable and flexible than the original Java GUI components from package `java.awt`, which were based on the GUI components of the underlying platform. For this reason, Swing GUI components are generally preferred.**

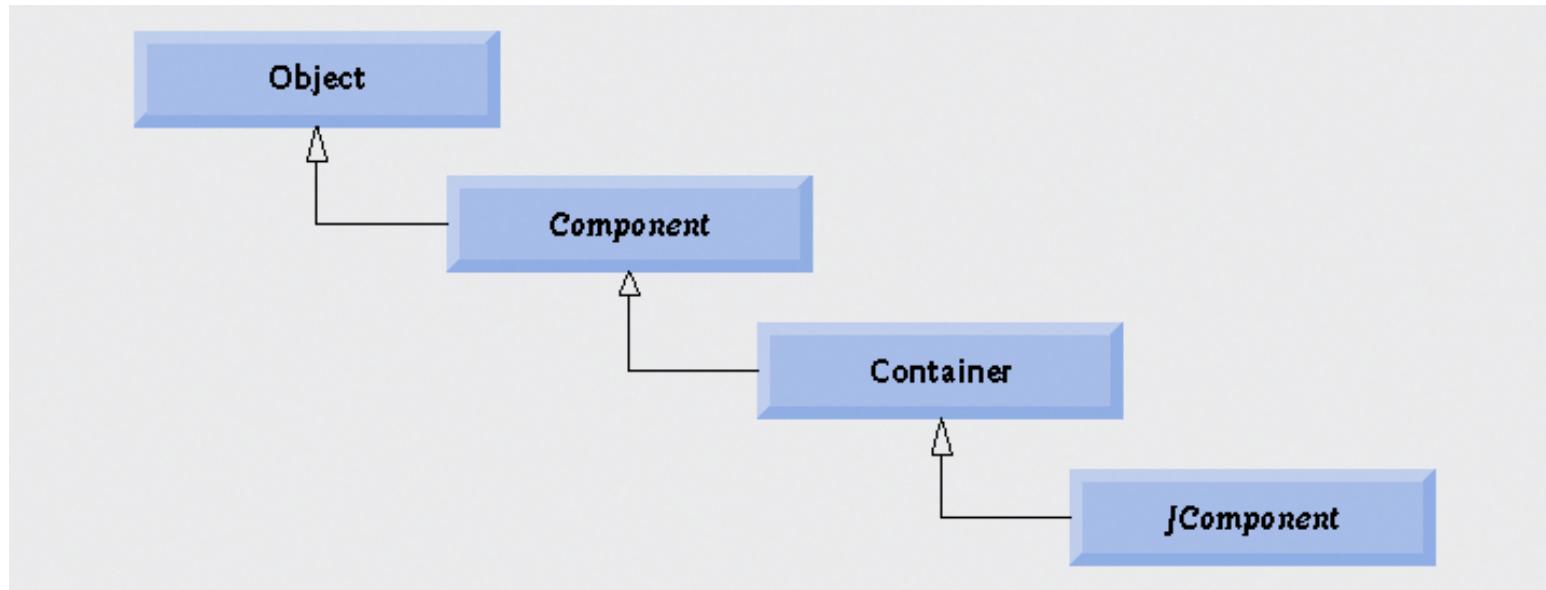


# Lightweight vs. Heavyweight GUI Components

- **Lightweight components**
  - Not tied directly to GUI components supported by underlying platform
- **Heavyweight components**
  - Tied directly to the local platform
  - AWT components
  - Some Swing components



# Swing Object Hierarchy

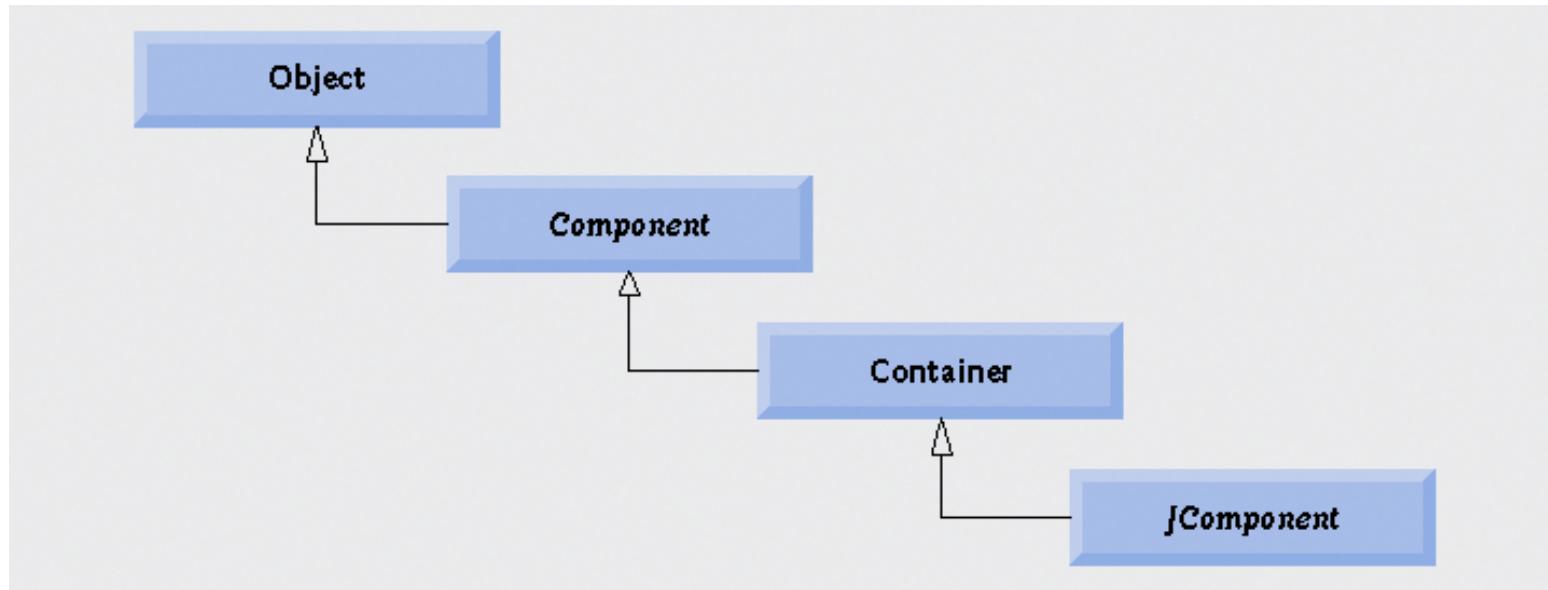


# Superclasses of Swing's Lightweight GUI Components

- **Class Component (package java.awt)**
  - Subclass of Object
  - Declares many behaviors and attributes common to GUI components
- **Class Container (package java.awt)**
  - Subclass of Component
  - Organizes Components
- **Class JComponent (package javax.swing)**
  - Subclass of Container
  - Superclass of all lightweight Swing components



# Swing Object Hierarchy



# Superclasses of Swing's Lightweight GUI Components

- **Common lightweight component features**
  - Pluggable look-and-feel to customize the appearance of components
  - Shortcut keys (called mnemonics)
  - Common event-handling capabilities
  - Brief description of component's purpose (called tool tips)
  - Support for localization



# 11.4 Displaying Text and Images in a Window

- **Class JFrame**

- Most windows are an instance or subclass of this class
- Provides title bar
- Provides buttons to minimize, maximize and close the application



# Labeling GUI Components

- **Label**
  - Text instructions or information stating the purpose of each component
  - Created with class `JLabel`



# Specifying the Layout

- **Laying out containers**
  - Determines where components are placed in the container
  - Done in Java with layout managers
    - One of which is class `FlowLayout`
  - Set with the `setLayout` method of class `JFrame`



```
1 // Fig. 11.6: LabelFrame.java
2 // Demonstrating the JLabel class.
3 import java.awt.FlowLayout; // specifies how components are arranged
4 import javax.swing.JFrame; // provides basic window features
5 import javax.swing.JLabel; // displays text and images
6 import javax.swing.SwingConstants; // common constants used with Swing
7 import javax.swing.Icon; // interface used to manipulate images
8 import javax.swing.ImageIcon; // loads images
9
10 public class LabelFrame extends JFrame
11 {
12     private JLabel label1; // JLabel with just text
13     private JLabel label2; // JLabel constructed with text and icon
14     private JLabel label3; // JLabel with added text and icon
15
16     // LabelFrame constructor adds JLabels to JFrame
17     public LabelFrame()
18     {
19         super( "Testing JLabel" );
20         setLayout( new FlowLayout() ); // set frame layout
21
22         // JLabel constructor with a string argument
23         label1 = new JLabel( "Label with text" );
24         label1.setToolTipText( "This is label1" );
25         add( label1 ); // add label1 to JFrame
26     }
}
```

## Outline

### LabelFrame.java

(1 of 2)



```
27 // JLabel constructor with string, Icon and alignment arguments
28 Icon bug = new ImageIcon( getClass().getResource( "bug1.gif" ) );
29 label2 = new JLabel( "Label with text and icon", bug,
30     SwingConstants.LEFT );
31 label2.setToolTipText( "This is label2" );
32 add( label2 ); // add label2 to JFrame
33
34 label3 = new JLabel(); // JLabel constructor no arguments
35 label3.setText( "Label with icon and text at bottom" );
36 label3.setIcon( bug ); // add icon to JLabel
37 label3.setHorizontalTextPosition( SwingConstants.CENTER );
38 label3.setVerticalTextPosition( SwingConstants.BOTTOM );
39 label3.setToolTipText( "This is label3" );
40 add( label3 ); // add label3 to JFrame
41 } // end LabelFrame constructor
42 } // end class LabelFrame
```

## Outline

LabelFrame.java

(2 of 2)



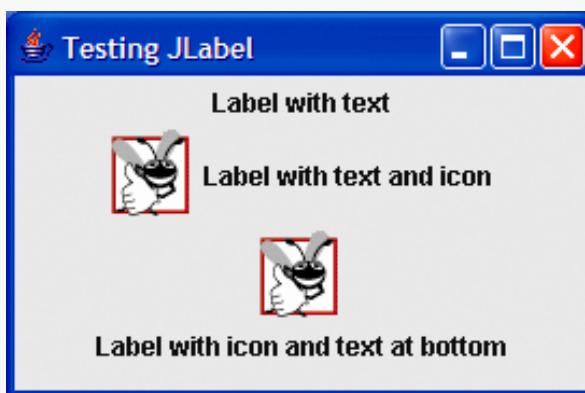
## Outline

### LabelTest.java

```

1 // Fig. 11.7: LabelTest.java
2 // Testing LabelFrame.
3 import javax.swing.JFrame;
4
5 public class LabelTest
6 {
7     public static void main( String args[] )
8     {
9         LabelFrame labelFrame = new LabelFrame(); // create LabelFrame
10        labelFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        labelFrame.setSize( 275, 180 ); // set frame size
12        labelFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class LabelTest

```



# Creating and Attaching JLabel1

- **Method setToolTipText of class JComponent**
  - Specifies the tool tip
- **Method add of class Container**
  - Adds a component to a container



# Common Programming Error 11.1

---

**If you do not explicitly add a GUI component to a container, the GUI component will not be displayed when the container appears on the screen.**



# Look-and-Feel Observation 11.6

---

**Use tool tips to add descriptive text to your GUI components. This text helps the user determine the GUI component's purpose in the user interface.**



# Creating and Attaching JLabel

- **Interface Icon**

- Can be added to a JLabel with the setIcon method
  - Implemented by class ImageIcon

- **Interface SwingConstants**

- Declares a set of common integer constants such as those used to set the alignment of components
  - Can be used with methods setHorizontalAlignment and setVerticalAlignment



# Creating and Attaching JLabels

- Other JLabel methods
  - `getText` and `setText`
    - For setting and retrieving the text of a label
  - `getIcon` and `setIcon`
    - For setting and retrieving the icon displayed in the label
  - `getHorizontalTextPosition` and `setHorizontalTextPosition`
    - For setting and retrieving the horizontal position of the text displayed in the label



Constant	Description
<i>Horizontal-position constants</i>	
<code>SwingConstants.LEFT</code>	Place text on the left.
<code>SwingConstants.CENTER</code>	Place text in the center.
<code>SwingConstants.RIGHT</code>	Place text on the right.
<i>Vertical-position constants</i>	
<code>SwingConstants.TOP</code>	Place text at the top.
<code>SwingConstants.CENTER</code>	Place text in the center.
<code>SwingConstants.BOTTOM</code>	Place text at the bottom.

Fig. 11.8 | Some basic GUI components.



# Creating and Displaying a LabelFrame Window

- Other JFrame methods
  - setDefaultCloseOperation
    - Dictates how the application reacts when the user clicks the close button
  - setSize
    - Specifies the width and height of the window
  - setVisible
    - Determines whether the window is displayed (true) or not (false)



# 11.5 Text Fields and an Introduction to Event Handling with Nested Classes

- **GUIs are event-driven**
  - A user interaction creates an event
    - Common events are clicking a button, typing in a text field, selecting an item from a menu, closing a window and moving the mouse
  - The event causes a call to a method called an event handler



# 11.5 Text Fields and an Introduction to Event Handling with Nested Classes

- **Class JTextField**
  - Superclass of JTextField
    - Superclass of JPasswordField
      - Adds echo character to hide text input in component
  - Allows user to enter text in the component when component has the application's focus



## Outline

### TextFieldFrame.java

(1 of 3)

```

1 // Fig. 11.9: TextFieldFrame.java
2 // Demonstrating the JTextField class.
3 import java.awt.FlowLayout;
4 import java.awt.event.ActionListener;
5 import java.awt.event.ActionEvent;
6 import javax.swing.JFrame;
7 import javax.swing.JTextField;
8 import javax.swing.JPasswordField;
9 import javax.swing.JOptionPane;
10
11 public class TextFieldFrame extends JFrame
12 {
13     private JTextField textField1; // text field with set size
14     private JTextField textField2; // text field constructed with text
15     private JTextField textField3; // text field with text and size
16     private JPasswordField passwordField; // password field with text
17
18     // TextFieldFrame constructor adds JTextFields to JFrame
19     public TextFieldFrame()
20     {
21         super( "Testing JTextField and JPasswordField" );
22         setLayout( new FlowLayout() ); // set frame layout
23
24         // construct textfield with 10 columns
25         textField1 = new JTextField( 10 );
26         add( textField1 ); // add textField1 to JFrame
27

```

Create a new JTextField



## Outline

```

28 // construct textfield with default text
29 textField2 = new JTextField( "Enter text here" );
30 add( textField2 ); // add textField2 to JFrame
31
32 // construct textfield with default text and 21 columns
33 textField3 = new JTextField( "Uneditable text field" , 21 );
34 textField3.setEditable( false ); // disable editing
35 add( textField3 ); // add textField3 to JFrame
36
37 // construct passwordfield with default text
38 passwordField = new JPasswordField( "Hidden text" );
39 add( passwordField ); // add passwordField to JFrame
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 } // end TextFieldFrame constructor
48
49 // private inner class for event handling
50 private class TextFieldHandler implements ActionListener
51 {
52     // process text field events
53     public void actionPerformed( ActionEvent event )
54     {
55         String string = ""; // declare string to display
56

```

TextFieldFrame.java

Create a new JTextField

Create a new uneditable JTextField

Create a new JPasswordField

Create event handler

Register event handler

Create event handler class by implementing the ActionListener interface

Declare actionPerformed method

```

57 // user pressed Enter in JTextField textField1
58 if ( event.getSource() == textField1 )
59     string = String.format( "textField1: %s",
60                             event.getActionCommand() );
61
62 // user pressed Enter in JTextField textField2
63 else if ( event.getSource() == textField2 )
64     string = String.format( "textField2: %s",
65                             event.getActionCommand() );
66
67 // user pressed Enter in JTextField textField3
68 else if ( event.getSource() == textField3 )
69     string = String.format( "textField3: %s",
70                             event.getActionCommand() );
71
72 // user pressed Enter in JTextField passwordField
73 else if ( event.getSource() == passwordField )
74     string = String.format( "passwordField: %s",
75                             new String( passwordField.getPassword() ) );
76
77 // display JTextField content
78 JOptionPane.showMessageDialog( null, string );
79 } // end method actionPerformed
80 } // end private inner class TextFieldHandler
81 } // end class TextFieldFrame

```

Test if the source of the event is the first text field

Get text from text field

Test if the source of the event is the second text field

Get text from text field

Test if the source of the event is the third text field

Get text from text field

Test if the source of the event is the password field

Get password from password field

TextFieldFrame

(3 of 3)



## Outline

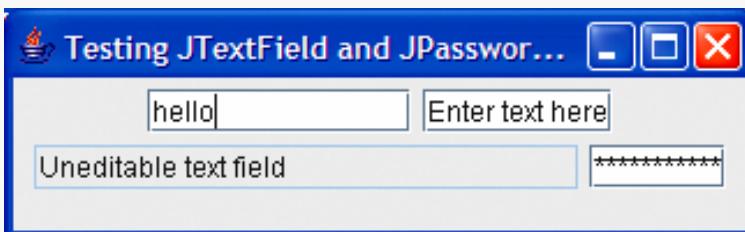
### TextFieldTest.java

(1 of 2)

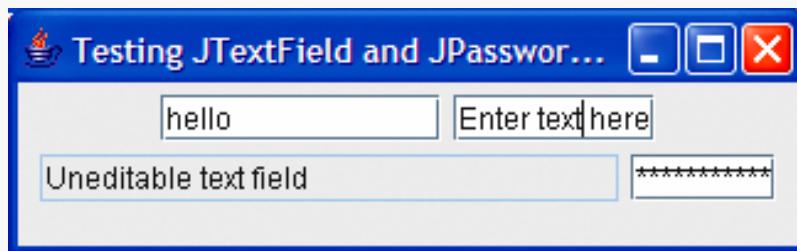
```

1 // Fig. 11.10: TextFieldTest.java
2 // Testing JTextFieldFrame.
3 import javax.swing.JFrame;
4
5 public class TextFieldTest
6 {
7     public static void main( String args[] )
8     {
9         JTextFieldFrame textFieldFrame = new JTextFieldFrame();
10        textFieldFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        textFieldFrame.setSize( 325, 100 ); // set frame size
12        textFieldFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class TextFieldTest

```



## Outline



TextFieldTest  
.java

(2 of 2)



# Steps Required to Set Up Event Handling for a GUI Component

- **Several coding steps are required for an application to respond to events**
  - Create a class for the event handler
  - Implement an appropriate event-listener interface
  - Register the event handler



# Using a Nested Class to Implement an Event Handler

- **Top-level classes**
  - Not declared within another class
- **Nested classes**
  - Declared within another class
  - Non-static nested classes are called inner classes
    - Frequently used for event handling



# Software Engineering Observation 11.2

---

An inner class is allowed to directly access its top-level class's variables and methods, even if they are private.



# Using a Nested Class to Implement an Event Handler

- **JTextFields and JPasswordFields**
  - Pressing enter within either of these fields causes an ActionEvent
    - Processed by objects that implement the ActionListener interface



# Registering the Event Handler for Each Text Field

- **Registering an event handler**
  - Call method `addActionListener` to register an `ActionListener` object
  - `ActionListener` listens for events on the object



# Software Engineering Observation 11.3

---

**The event listener for an event must implement the appropriate event-listener interface.**



# Details of Class `TextFieldHandler's` `actionPerformed` Method

- **Event source**
  - Component from which event originates
  - Can be determined using method `getSource`
  - Text from a `JTextField` can be acquired using `getActionCommand`
  - Text from a `JPasswordField` can be acquired using `getPassword`



# 11.6 Common GUI Event Types and Listener Interfaces

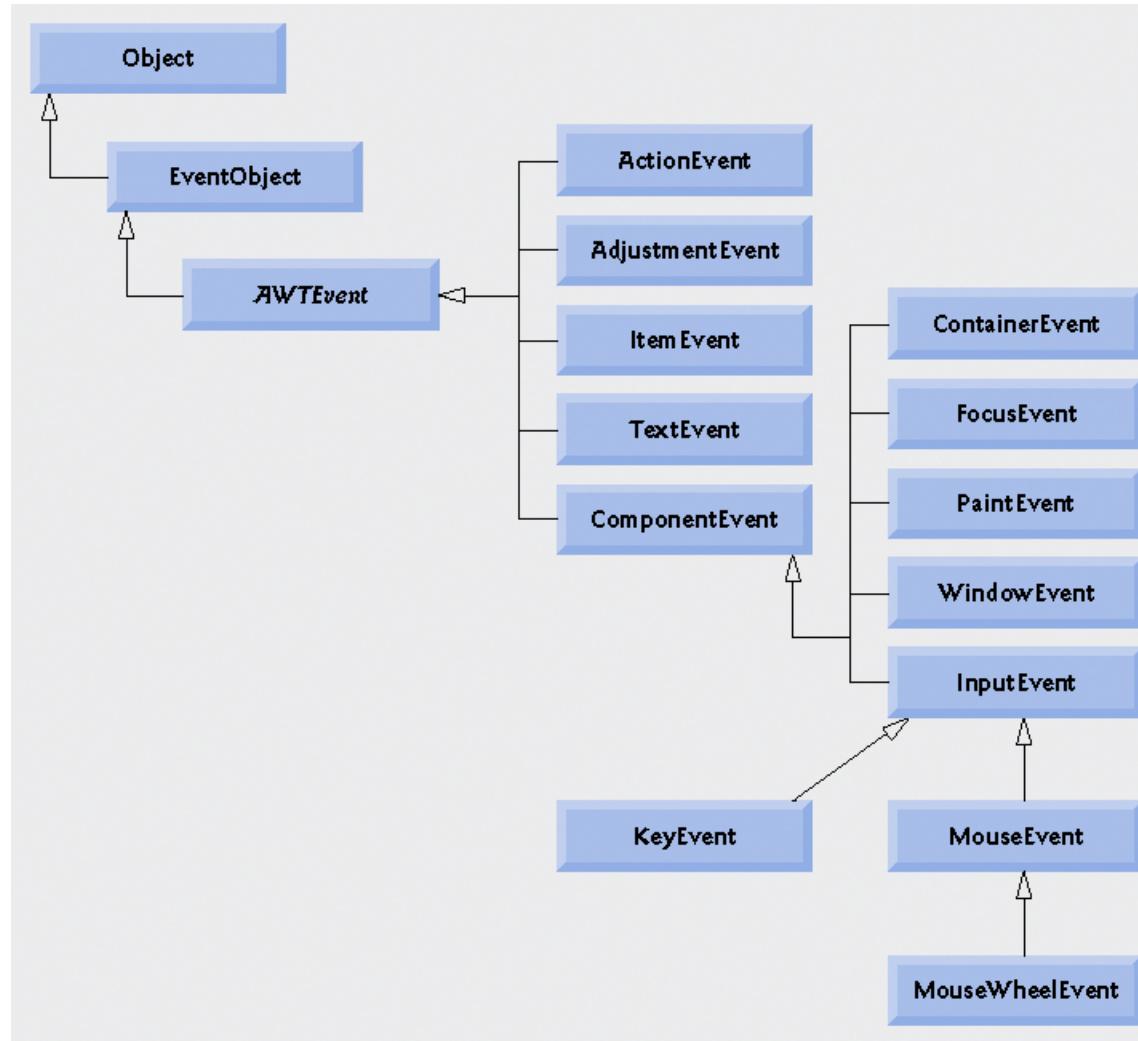
- **Event types**
  - All are subclasses of `AWTEvent`
  - Some declared in package `java.awt.event`
  - Those specific to Swing components declared in `javax.swing.event`



# 11.6 Common GUI Event Types and Listener Interfaces

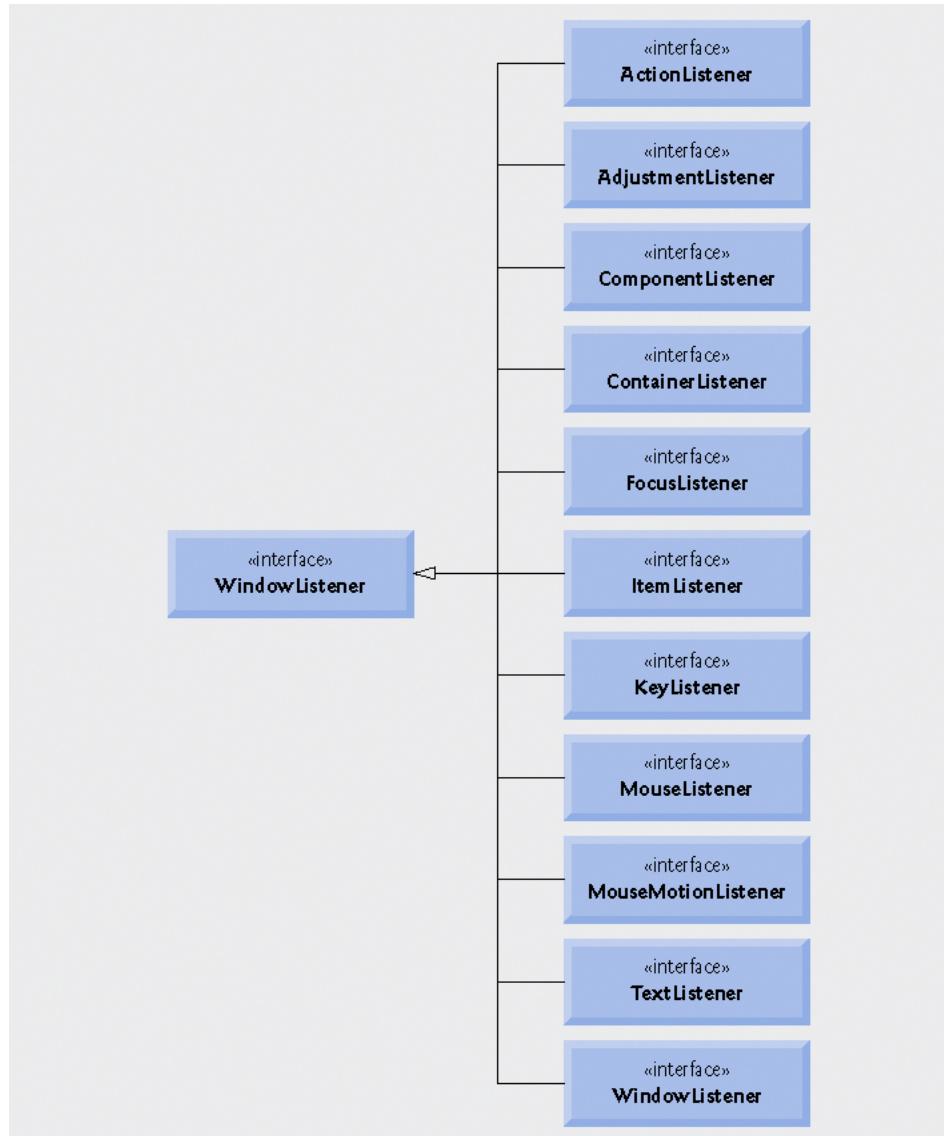
- **Delegation event model**
  - Event source is the component with which user interacts
  - Event object is created and contains information about the event that happened
  - Event listener is notified when an event happens





**Fig. 11.11 | Some event classes of package `java.awt.event`.**





**Fig. 11.12 | Some common event-listener interfaces of package `java.awt.event`.**



# 11.7 How Event Handling Works

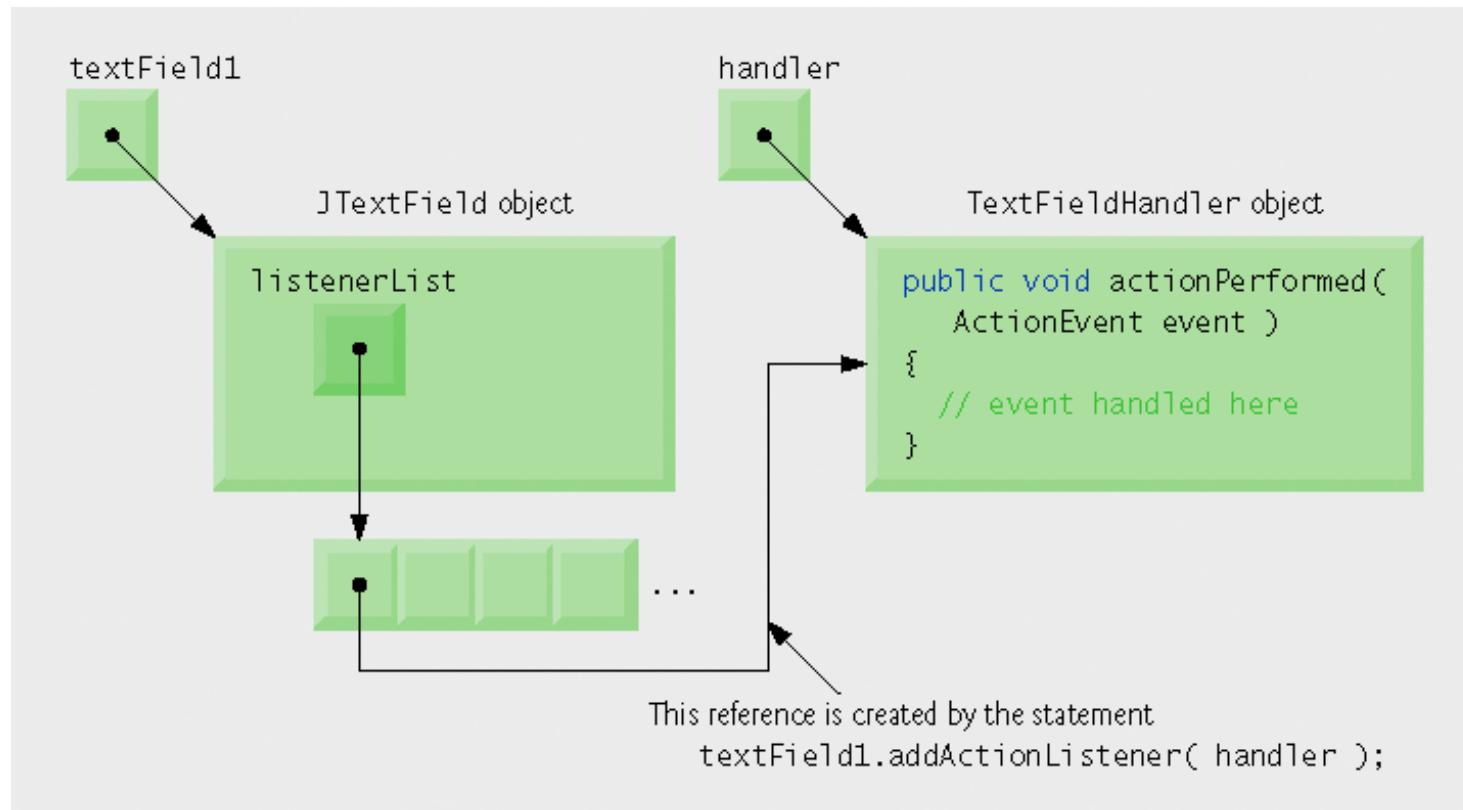
- Remaining questions
  - How did the event handler get registered?
  - How does the GUI component know to call `actionPerformed` rather than some other event-handling method?



# Registering Events

- Every JComponent has instance variable `listenerList`
  - Object of type `EventListenerList`
  - Maintains references to all its registered listeners





**Fig. 11.13 | Event registration for `JTextField` `textField1`.**



# Event-Handler Invocation

- Events are dispatched to only the event listeners that match the event type
  - Events have a unique event ID specifying the event type
- MouseEvents are handled by MouseListeners and MouseMotionsListeners
- KeyEvents are handled by KeyListeners



# 11.8 JButton

- **Button**
  - Component user clicks to trigger a specific action
  - Can be command button, check box, toggle button or radio button
  - Button types are subclasses of class `AbstractButton`



# 11.8 JButton

- **Command button**
  - Generates an `ActionEvent` when it is clicked
  - Created with class `JButton`
  - Text on the face of the button is called **button label**



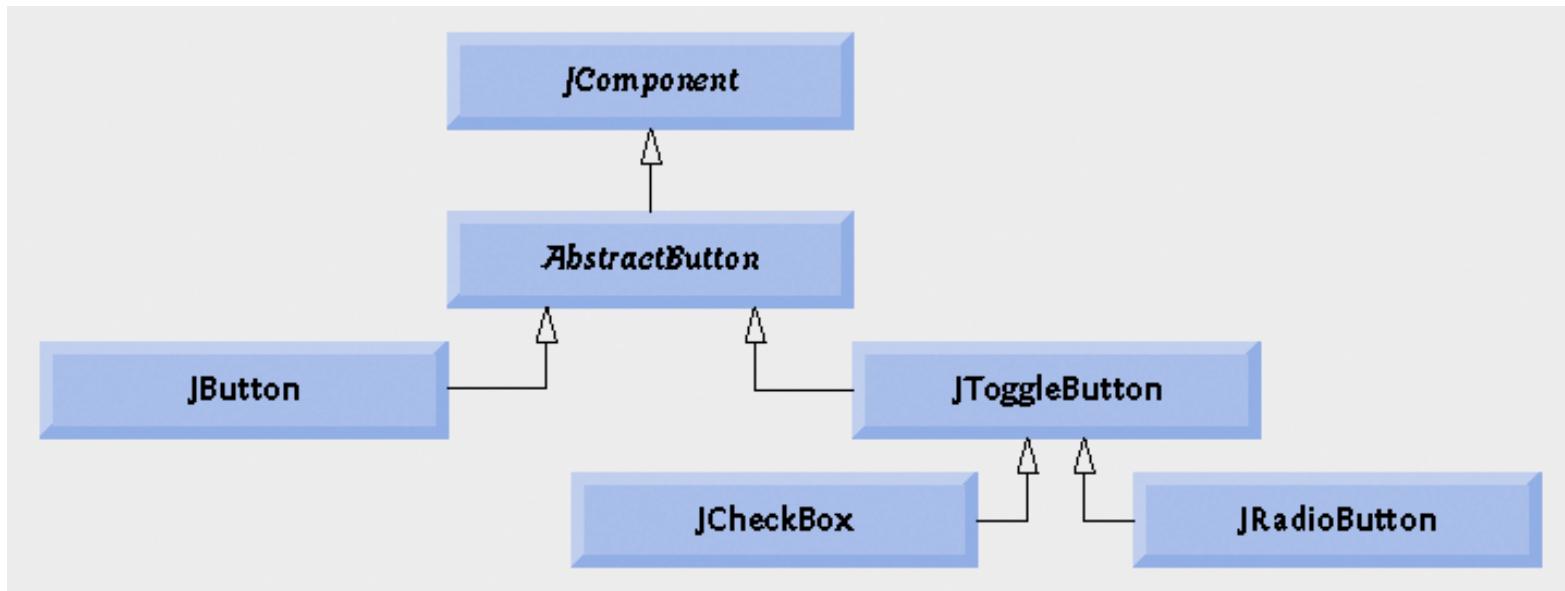


Fig. 11.14 | Swing button hierarchy.



## Outline

### ButtonFrame.java

(1 of 2)

Declare two JButton instance variables

```

1 // Fig. 11.15: ButtonFrame.java
2 // Creating JButtons.
3 import java.awt.FlowLayout;
4 import java.awt.event.ActionListener;
5 import java.awt.event.ActionEvent;
6 import javax.swing.JFrame;
7 import javax.swing.JButton;
8 import javax.swing.Icon;
9 import javax.swing.ImageIcon;
10 import javax.swing.JOptionPane;
11
12 public class ButtonFrame extends JFrame
13 {
14     private JButton plainJButton; // button with just text
15     private JButton fancyJButton; // button with icons
16
17     // ButtonFrame adds JButtons to JFrame
18     public ButtonFrame()
19     {
20         super( "Testing Buttons" );
21         setLayout( new FlowLayout() ); // set frame layout
22
23         plainJButton = new JButton( "Plain Button" ); // button with text
24         add( plainJButton ); // add plainJButton to JFrame
25
26         Icon bug1 = new ImageIcon( getClass().getResource( "bug1.gif" ) );
27         Icon bug2 = new ImageIcon( getClass().getResource( "bug2.gif" ) );
28         fancyJButton = new JButton( "Fancy Button", bug1 ); // set image
29         fancyJButton.setRolloverIcon( bug2 ); // set rollover image
30         add( fancyJButton ); // add fancyJButton to JFrame

```

Create new JButton

Create two ImageIcons

Create new JButton

Set rollover icon for JButton



## Outline

```

31
32 // create new ButtonHandler for button event handling
33 ButtonHandler handler = new ButtonHandler();
34 fancyJButton.addActionListener( handler );
35 plainJButton.addActionListener( handler );
36 } // end ButtonFrame constructor
37
38 // inner class for button event handling
39 private class ButtonHandler implements ActionListener
40 {
41     // handle button event
42     public void actionPerformed( ActionEvent event )
43     {
44         JOptionPane.showMessageDialog( ButtonFrame.this, String.format(
45             "You pressed: %s", event.getActionCommand() ) );
46     } // end method actionPerformed
47 } // end private inner class ButtonHandler
48 } // end class ButtonFrame

```

Create handler for buttons

ButtonFrame.java

Register event handler  
(*z or z*)

Inner class implements  
ActionListener

Access outer class's instance using  
**this** reference

Get text of JButton pressed



## Outline

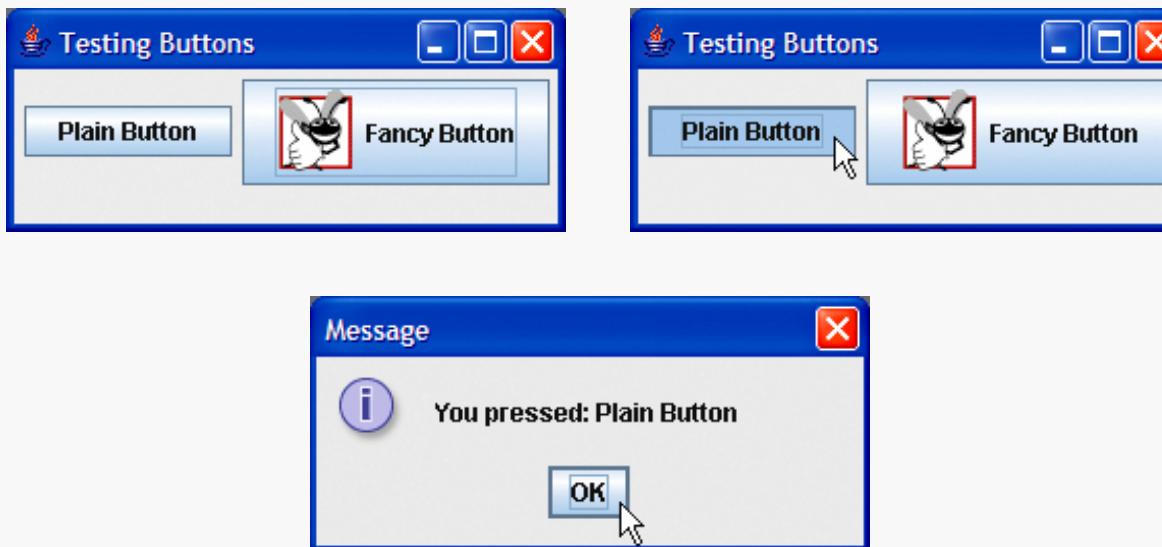
### ButtonTest.java

(1 of 2)

```

1 // Fig. 11.16: ButtonTest.java
2 // Testing ButtonFrame.
3 import javax.swing.JFrame;
4
5 public class ButtonTest
6 {
7     public static void main( String args[] )
8     {
9         ButtonFrame buttonFrame = new ButtonFrame(); // create ButtonFrame
10        buttonFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        buttonFrame.setSize( 275, 110 ); // set frame size
12        buttonFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class ButtonTest

```



## Outline

**ButtonTest.java**

(2 of 2)



# Software Engineering Observation 11.4

---

**When used in an inner class, keyword this refers to the current inner-class object being manipulated. An inner-class method can use its outer-class object's this by preceding this with the outer-class name and a dot, as in ButtonFrame.this.**



# 11.9 Buttons That Maintain State

- **State buttons**
  - Swing contains three types of state buttons
  - `JToggleButton`, `JCheckBox` and `JRadioButton`
  - `JCheckBox` and `JRadioButton` are subclasses of `JToggleButton`



## 11.9.1 JCheckBox

- **JCheckBox**
  - Contains a check box label that appears to right of check box by default
  - Generates an `ItemEvent` when it is clicked
    - `ItemEvents` are handled by an `ItemListener`
    - Passed to method `itemStateChanged`
  - Method `isSelected` returns whether check box is selected (`true`) or not (`false`)



## Outline

### CheckBoxFrame.java

(1 of 3)

Declare two JCheckBox instance variables

```

1 // Fig. 11.17: CheckBoxFrame.java
2 // Creating JCheckBox buttons.
3 import java.awt.FlowLayout;
4 import java.awt.Font;
5 import java.awt.event.ItemListener;
6 import java.awt.event.ItemEvent;
7 import javax.swing.JFrame;
8 import javax.swing.JTextField;
9 import javax.swing.JCheckBox;
10
11 public class CheckBoxFrame extends JFrame
12 {
13     private JTextField textField; // displays text in changing fonts
14     private JCheckBox boldJCheckBox; // to select/deselect bold
15     private JCheckBox italicJCheckBox; // to select/deselect italic
16
17     // CheckBoxFrame constructor adds JCheckboxes to JFrame
18     public CheckBoxFrame()
19     {
20         super( "JCheckBox Test" );
21         setLayout( new FlowLayout() ); // set frame layout
22
23         // set up JTextField and set its font
24         textField = new JTextField( "watch the font style change", 20 );
25         textField.setFont( new Font( "Serif", Font.PLAIN, 14 ) );
26         add( textField ); // add textField to JFrame
27

```

Set font of text field



## Outline

```

28 boldJCheckBox = new JCheckBox( "Bold" ); // create bold checkbox
29 italicJCheckBox = new JCheckBox( "Italic" ); // create italic
30 add( boldJCheckBox ); // add bold checkbox to JFrame
31 add( italicJCheckBox ); // add italic checkbox to JFrame
32

```

Create two JCheckboxes

```

33 // register listeners for JCheckboxes
34 CheckBoxHandler handler = new CheckBoxHandler();
35 boldJCheckBox.addItemListener( handler );
36 italicJCheckBox.addItemListener( handler );
37 } // end CheckBoxFrame constructor
38

```

CheckBoxFrame

Create event handler

Register event handler with  
JCheckboxes

```

39 // private inner class for ItemListener event hand
40 private class CheckBoxHandler implements ItemListener
41 {
42     private int valBold = Font.PLAIN; // controls bold
43     private int valItalic = Font.PLAIN; // controls italic
44

```

Inner class implements  
ItemListener

```

45 // respond to checkbox events
46 public void itemStateChanged( ItemEvent event )
47 {
48     // process bold checkbox events
49     if ( event.getSource() == boldJCheckBox )
50         valBold =
51             boldJCheckBox.isSelected() ? Font.BOLD : Font.PLAIN;
52

```

itemStateChanged method is  
called when a JCheckBox is  
clicked

Test whether JCheckBox is  
selected



## Outline

```
53 // process italic checkbox events
54 if ( event.getSource() == italicJCheckBox )
55     valItalic =
56         italicJCheckBox.isSelected() ? Font.ITALIC : Font.PLAIN;
57
58 // set text field font
59 textField.setFont(
60     new Font( "Serif", valBold + valItalic, 14 ) );
61 } // end method itemStateChanged
62 } // end private inner class CheckBoxHandler
63 } // end class CheckBoxFrame
```

Test source of the event

isSelected method returns whether JCheckBox is selected

(3 of 3)



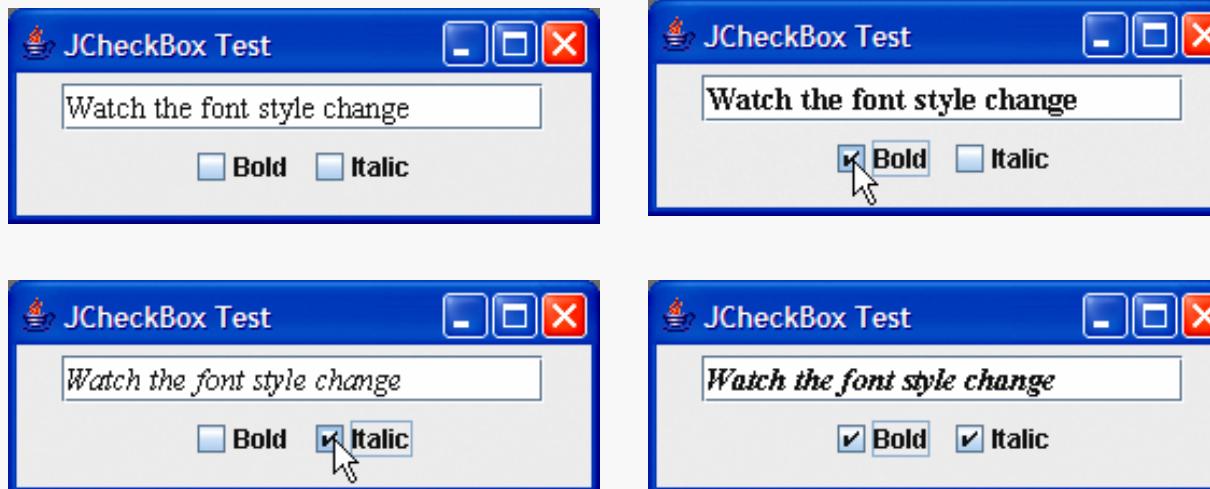
## Outline

### CheckBoxTest.java

```

1 // Fig. 11.18: CheckBoxTest.java
2 // Testing CheckBoxFrame.
3 import javax.swing.JFrame;
4
5 public class CheckBoxTest
6 {
7     public static void main( String args[] )
8     {
9         CheckBoxFrame checkBoxFrame = new CheckBoxFrame();
10        checkBoxFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        checkBoxFrame.setSize( 275, 100 ); // set frame size
12        checkBoxFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class CheckBoxTest

```



## 11.9.2 JRadioButton

- **JRadioButton**
  - Has two states – selected and unselected
  - Normally appear in a group in which only one radio button can be selected at once
    - Group maintained by a **ButtonGroup** object
      - Declares method **add** to add a **JRadioButton** to group
    - Usually represents mutually exclusive options



```

1 // Fig. 11.19: RadioButtonFrame.java
2 // Creating radio buttons using ButtonGroup and JRadioButton.
3 import java.awt.FlowLayout;
4 import java.awt.Font;
5 import java.awt.event.ItemListener;
6 import java.awt.event.ItemEvent;
7 import javax.swing.JFrame;
8 import javax.swing.JTextField;
9 import javax.swing.JRadioButton;
10 import javax.swing.ButtonGroup;
11
12 public class RadioButtonFrame extends JFrame
13 {
14     private JTextField textField; // used to display fo
15     private Font plainFont; // font for plain text
16     private Font boldFont; // font for bold text
17     private Font italicFont; // font for italic text
18     private Font boldItalicFont; // font for bold and italic text
19     private JRadioButton plainJRadioButton; // selects plain text
20     private JRadioButton boldJRadioButton; // selects bold text
21     private JRadioButton italicJRadioButton; // selects italic text
22     private JRadioButton boldItalicJRadioButton; // bold and italic
23     private ButtonGroup radioGroup; // buttongroup to hold radio buttons
24
25     // RadioButtonFrame constructor adds JRadioButtons to JFrame
26     public RadioButtonFrame()
27     {
28         super( "RadioButton Test" );
29         setLayout( new FlowLayout() ); // set frame layout
30

```

Declare four **JRadioButtons**  
and a **ButtonGroup** to manage  
them



## Outline

### RadioButtonFrame.java

(1 of 3)

## Outline

### RadioButtonFrame.java

(2 of 3)

Create the four JRadioButtons

Create the ButtonGroup

Add each JRadioButton to the  
ButtonGroup

```

31 textField = new JTextField( "Watch the font style change", 25 );
32 add( textField ); // add textField to JFrame
33
34 // create radio buttons
35 plainJRadioButton = new JRadioButton( "Plain", true );
36 boldJRadioButton = new JRadioButton( "Bold", false );
37 italicJRadioButton = new JRadioButton( "Italic", false );
38 boldItalicJRadioButton = new JRadioButton( "Bold/Italic", false );
39 add( plainJRadioButton ); // add plain button to JFrame
40 add( boldJRadioButton ); // add bold button to JFrame
41 add( italicJRadioButton ); // add italic button
42 add( boldItalicJRadioButton ); // add bold and italic button
43
44 // create logical relationship between JRadioButtons
45 radioGroup = new ButtonGroup(); // create ButtonGroup
46 radioGroup.add( plainJRadioButton ); // add plain to group
47 radioGroup.add( boldJRadioButton ); // add bold to group
48 radioGroup.add( italicJRadioButton ); // add italic to group
49 radioGroup.add( boldItalicJRadioButton ); // add bold and italic to group
50
51 // create font objects
52 plainFont = new Font( "Serif", Font.PLAIN, 14 );
53 boldFont = new Font( "Serif", Font.BOLD, 14 );
54 italicFont = new Font( "Serif", Font.ITALIC, 14 );
55 boldItalicFont = new Font( "Serif", Font.BOLD + Font.ITALIC, 14 );
56 textField.setFont( plainFont ); // set initial font to plain
57

```



## Outline

```

58 // register events for JRadioButtons
59 plainRadioButton.addItemListener(
60     new RadioButtonHandler( plainFont ) );
61 boldRadioButton.addItemListener(
62     new RadioButtonHandler( boldFont ) );
63 italicRadioButton.addItemListener(
64     new RadioButtonHandler( italicFont ) );
65 boldItalicRadioButton.addItemListener(
66     new RadioButtonHandler( boldItalicFont ) );
67 } // end RadioButtonFrame constructor
68
69 // private inner class to handle radio button events
70 private class RadioButtonHandler implements ItemListener
71 {
72     private Font font; // font associated with this
73
74     public RadioButtonHandler( Font f )
75     {
76         font = f; // set the font of this Listener
77     } // end constructor RadioButtonHandler
78
79     // handle radio button events
80     public void itemStateChanged( ItemEvent event )
81     {
82         textField.setFont( font ); // set font of textField
83     } // end method itemStateChanged
84 } // end private inner class RadioButtonHandler
85 } // end class RadioButtonFrame

```

Register an event handler with each JRadioButton

RadioButtonFrame.java

(3 of 3)

Event handler inner class implements ItemListener

When radio button is selected, the text field's font will be set to the value passed to the constructor



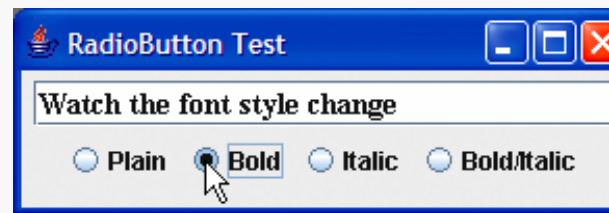
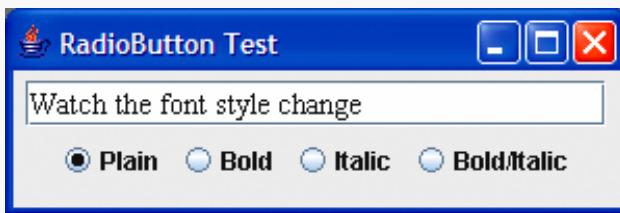
## Outline

### RadioButtonTest.java

```

1 // Fig. 11.20: RadioButtonTest.java
2 // Testing RadioButtonFrame.
3 import javax.swing.JFrame;
4
5 public class RadioButtonTest
6 {
7     public static void main( String args[] )
8     {
9         RadioButtonFrame radioButtonFrame = new RadioButtonFrame();
10        radioButtonFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        radioButtonFrame.setSize( 300, 100 ); // set frame size
12        radioButtonFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class RadioButtonTest

```



## 11.10 JComboBox and Using an Anonymous Inner Class for Event Handling

- **Combo box**
  - Also called a drop-down list
  - Implemented by class JComboBox
  - Each item in the list has an index
  - setMaximumRowCount sets the maximum number of rows shown at once
  - JComboBox provides a scrollbar and up and down arrows to traverse list



## Look-and-Feel Observation 11.11

---

**Set the maximum row count for a JComboBox to a number of rows that prevents the list from expanding outside the bounds of the window in which it is used. This configuration will ensure that the list displays correctly when it is expanded by the user.**



# Using an Anonymous Inner Class for Event Handling

- **Anonymous inner class**
  - Special form of inner class
  - Declared without a name
  - Typically appears inside a method call
  - Has limited access to local variables



## Outline

ComboBoxFrame.java

(1 of 2)

```

1 // Fig. 11.21: ComboBoxFrame.java
2 // Using a JComboBox to select an image to display.
3 import java.awt.FlowLayout;
4 import java.awt.event.ItemListener;
5 import java.awt.event.ItemEvent;
6 import javax.swing.JFrame;
7 import javax.swing.JLabel;
8 import javax.swing.JComboBox;
9 import javax.swing.Icon;
10 import javax.swing.ImageIcon;
11
12 public class ComboBoxFrame extends JFrame
13 {
14     private JComboBox imagesJComboBox; // combobox to hold names of icons
15     private JLabel label; // label to display selected icon
16
17     private String names[] =
18         { "bug1.gif", "bug2.gif", "travelbug.gif", "bug3.gif" };
19     private Icon icons[] = {
20         new ImageIcon( getClass().getResource( names[ 0 ] ) ),
21         new ImageIcon( getClass().getResource( names[ 1 ] ) ),
22         new ImageIcon( getClass().getResource( names[ 2 ] ) ),
23         new ImageIcon( getClass().getResource( names[ 3 ] ) ) };
24
25     // ComboBoxFrame constructor adds JComboBox to JFrame
26     public ComboBoxFrame()
27     {
28         super( "Testing JComboBox" );
29         setLayout( new FlowLayout() ); // set frame layout
30

```

Declare JComboBox instance  
variable



```

31 imagesJComboBox = new JComboBox( names ); // set up JComboBox
32 imagesJComboBox.setMaximumRowCount( 3 ); // display 3 items
33
34 imagesJComboBox.addItemListener(
35     new ItemListener() // anonymous inner class
36     {
37         // handle JComboBox event
38         public void itemStateChanged( ItemEvent event )
39         {
40             // determine whether check box selected
41             if ( event.getStateChange() == ItemEvent.SELECTED )
42                 label.setIcon( icons[
43                     imagesJComboBox.getSelectedIndex() ] );
44         } // end method itemStateChanged
45     } // end anonymous inner class
46 ); // end call to addItemListener
47
48 add( imagesJComboBox ); // add combobox to JFrame
49 label = new JLabel( icons[ 0 ] ); // display first icon
50 add( label ); // add label to JFrame
51 } // end ComboBoxFrame constructor
52 } // end class ComboBoxFrame

```

Create JComboBox and set maximum row count

Create anonymous inner class as the event handler

Declare method itemStateChanged

Test state change of JComboBox

Method getSelectedIndex locates selected item



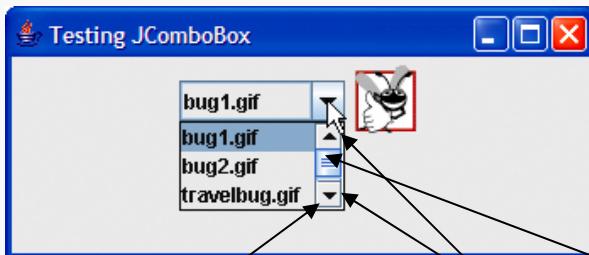
## Outline

### ComboBoxTest.java

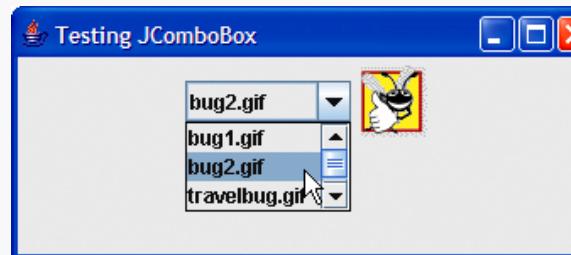
```

1 // Fig. 11.22: ComboBoxTest.java
2 // Testing ComboBoxFrame.
3 import javax.swing.JFrame;
4
5 public class ComboBoxTest
6 {
7     public static void main( String args[] )
8     {
9         ComboBoxFrame comboBoxFrame = new ComboBoxFrame();
10        comboBoxFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        comboBoxFrame.setSize( 350, 150 ); // set frame size
12        comboBoxFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class ComboBoxTest

```



Scrollbar to scroll through the items in the list



scroll box



scroll arrows



# Software Engineering Observation 11.6

---

**Like any other class, when an anonymous inner class implements an interface, the class must implement every method in the interface.**



## 11.11 JList

- **List**
  - Displays a series of items from which the user may select one or more items
  - Implemented by class `JList`
  - Allows for single-selection lists or multiple-selection lists
  - A `ListSelectionEvent` occurs when an item is selected
    - Handled by a `ListSelectionListener` and passed to method `valueChanged`



```

1 // Fig. 11.23: ListFrame.java
2 // Selecting colors from a JList.
3 import java.awt.FlowLayout;
4 import java.awt.Color;
5 import javax.swing.JFrame;
6 import javax.swing.JList;
7 import javax.swing.JScrollPane;
8 import javax.swing.event.ListSelectionListener;
9 import javax.swing.event.ListSelectionEvent;
10 import javax.swing.ListSelectionModel;
11
12 public class ListFrame extends JFrame
13 {
14     private JList colorJList; // List to display colors
15     private final String colorNames[] = { "Black", "Blue", "Cyan",
16         "Dark Gray", "Gray", "Green", "Light Gray", "Magenta",
17         "Orange", "Pink", "Red", "White", "Yellow" };
18     private final Color colors[] = { Color.BLACK, Color.BLUE, Color.CYAN,
19         Color.DARK_GRAY, Color.GRAY, Color.GREEN, Color.LIGHT_GRAY,
20         Color.MAGENTA, Color.ORANGE, Color.PINK, Color.RED, Color.WHITE,
21         Color.YELLOW };
22
23     // ListFrame constructor add JScrollPane containing JList to JFrame
24     public ListFrame()
25     {
26         super( "List Test" );
27         setLayout( new FlowLayout() ); // set frame layout
28

```

## Outline

### ListFrame.java

(1 of 2)

Declare JList instance variable



```

29 colorJList = new JList( colorNames ); // create with colorNames
30 colorJList.setVisibleRowCount( 5 ); // display five rows at once
31
32 // do not allow multiple selections
33 colorJList.setSelectionMode( ListSelectionModel.SINGLE_SELECTION );
34
35 // add a JScrollPane containing JList to frame
36 add( new JScrollPane( colorJList ) );
37
38 colorJList.addListSelectionListener(
39     new ListSelectionListener() // anonymous inner class
40     {
41         // handle list selection events
42         public void valueChanged( ListSelectionEvent event )
43         {
44             getContentPane().setBackground(
45                 colors[ colorJList.getSelectedIndex() ] );
46         } // end method valueChanged
47     } // end anonymous inner class
48 ); // end call to addListSelectionListener
49 } // end ListFrame constructor
50 } // end class ListFrame

```

## Outline

## Create JList

Set selection mode of JList

me.java

(2 of 2)

Add JList to ScrollPane and add to application

Get index of selected item



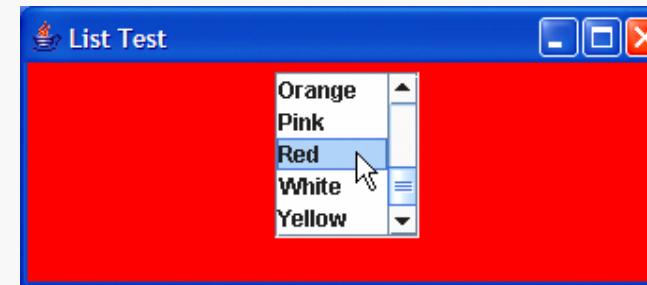
```

1 // Fig. 11.24: ListTest.java
2 // Selecting colors from a JList.
3 import javax.swing.JFrame;
4
5 public class ListTest
6 {
7     public static void main( String args[] )
8     {
9         ListFrame listFrame = new ListFrame(); // create ListFrame
10        listFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        listFrame.setSize( 350, 150 ); // set frame size
12        listFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class ListTest

```

## Outline

### ListTest.java



## 11.12 Multiple-Selection Lists

- **Multiple-selection list**
  - Enables users to select many items
  - Single interval selection allows only a continuous range of items
  - Multiple interval selection allows any set of elements to be selected



```
1 // Fig. 11.25: MultipleSelectionFrame.java
2 // Copying items from one List to another.
3 import java.awt.FlowLayout;
4 import java.awt.event.ActionListener;
5 import java.awt.event.ActionEvent;
6 import javax.swing.JFrame;
7 import javax.swing.JList;
8 import javax.swing.JButton;
9 import javax.swing.JScrollPane;
10 import javax.swing.ListSelectionModel;
11
12 public class MultipleSelectionFrame extends JFrame
13 {
14     private JList colorJList; // list to hold color names
15     private JList copyJList; // list to copy color names into
16     private JButton copyJButton; // button to copy selected names
17     private final String colorNames[] = { "Black", "Blue", "Cyan",
18         "Dark Gray", "Gray", "Green", "Light Gray", "Magenta", "Orange",
19         "Pink", "Red", "White", "Yellow" };
20
21     // MultipleSelectionFrame constructor
22     public MultipleSelectionFrame()
23     {
24         super( "Multiple Selection Lists" );
25         setLayout( new FlowLayout() ); // set frame layout
26     }
```

## Outline

### Multiple SelectionFrame .java

(1 of 3)



## Outline

```

27 colorJList = new JList( colorNames ); // holds names of all colors
28 colorJList.setVisibleRowCount( 5 ); // show five rows
29 colorJList.setSelectionMode(
30     ListSelectionModel.MULTIPLE_INTERVAL_SELECTION );

```

Use a multiple interval selection list

```

31 add( new JScrollPane( colorJList ) ); // add list
32
33 copyJButton = new JButton( "Copy >>" ); // create copy button
34 copyJButton.addActionListener(
35
36     new ActionListener() // anonymous inner class
37     {

```

SelectionFrame.java

(2 of 3)

```

38         // handle button event
39         public void actionPerformed( ActionEvent event )
40         {

```

// place selected values in copyJList

```
41             copyJList.setListData( colorJList.getSelectedValues() );
42         } // end method actionPerformed
43     } // end anonymous inner class
44 } // end call to addActionListener
45
46

```

Use methods **setListData** and  
**getSelectedValues** to copy  
values from one **JList** to the  
other



**Outline**

```
47 add( copyJButton ); // add copy button to JFrame  
48  
49 copyJList = new JList(); // create list to hold  
50 copyJList.setVisibleRowCount( 5 ); // show 5 rows  
51 copyJList.setFixedCellWidth( 100 ); // set width  
52 copyJList.setFixedCellHeight( 15 ); // set height  
53 copyJList.setSelectionMode(  
54     ListSelectionModel.SINGLE_INTERVAL_SELECTION );  
55 add( new JScrollPane( copyJList ) ); // add list with scrollpane  
56 } // end MultipleSelectionFrame constructor  
57 } // end class MultipleSelectionFrame
```

Set cell width for presentation

Set cell height for presentation

Set selection model to single interval selection

Multiple  
onFrame  
.java



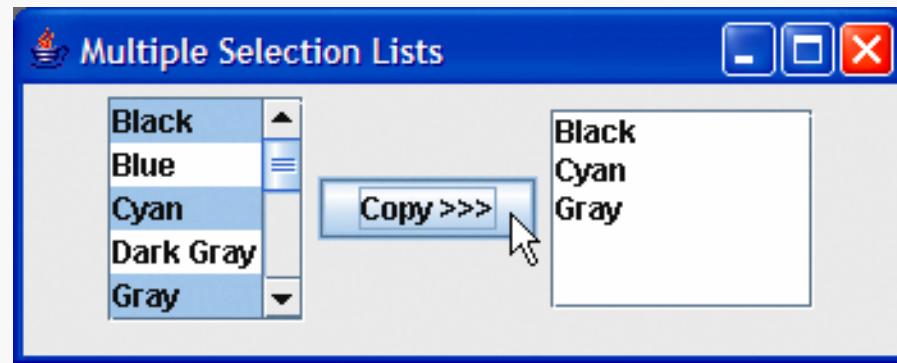
## Outline

### Multiple SelectionTest .java

```

1 // Fig. 11.26: MultipleSelectionTest.java
2 // Testing MultipleSelectionFrame.
3 import javax.swing.JFrame;
4
5 public class MultipleSelectionTest
6 {
7     public static void main( String args[] )
8     {
9         MultipleSelectionFrame multipleSelectionFrame =
10            new MultipleSelectionFrame();
11        multipleSelectionFrame.setDefaultCloseOperation(
12            JFrame.EXIT_ON_CLOSE );
13        multipleSelectionFrame.setSize( 350, 140 ); // set frame size
14        multipleSelectionFrame.setVisible( true ); // display frame
15    } // end main
16 } // end class MultipleSelectionTest

```



## 11.13 Mouse Event Handling

- **Mouse events**
  - Create a `MouseEvent` object
  - Handled by `MouseListener`s and `MouseMotionListener`s
  - `MouseInputListener` combines the two interfaces
  - Interface `MouseWheelListener` declares method `mousewheelMoved` to handle `MousewheelEvents`



## MouseListener and MouseMotionListener interface methods

### *Methods of interface MouseListener*

`public void mousePressed( MouseEvent event )`

Called when a mouse button is pressed while the mouse cursor is on a component.

`public void mouseClicked( MouseEvent event )`

Called when a mouse button is pressed and released while the mouse cursor remains stationary on a component. This event is always preceded by a call to `mousePressed`.

`public void mouseReleased( MouseEvent event )`

Called when a mouse button is released after being pressed. This event is always preceded by a call to `mousePressed` and one or more calls to `mouseDragged`.

`public void mouseEntered( MouseEvent event )`

Called when the mouse cursor enters the bounds of a component.

**Fig. 11.27 | MouseListener and MouseMotionListener interface methods. (Part 1 of 2.)**



## MouseListener and MouseMotionListener interface methods

`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 while the mouse cursor is on a component and the mouse is moved while the mouse button remains pressed. This event is always preceded by a call to `mousePressed`. All drag events are sent to the component on which the user began to drag the mouse.

`public void mouseMoved( MouseEvent event )`

Called when the mouse is moved when the mouse cursor is on a component. All move events are sent to the component over which the mouse is currently positioned.

**Fig. 11.27 | MouseListener and MouseMotionListener interface methods. (Part 2 of 2.)**



## Outline

### MouseTrackerFrame.java

(1 of 4)

```

1 // Fig. 11.28: MouseTrackerFrame.java
2 // Demonstrating mouse events.
3 import java.awt.Color;
4 import java.awt.BorderLayout;
5 import java.awt.event.MouseListener;
6 import java.awt.event.MouseMotionListener;
7 import java.awt.event.MouseEvent;
8 import javax.swing.JFrame;
9 import javax.swing.JLabel;
10 import javax.swing.JPanel;
11
12 public class MouseTrackerFrame extends JFrame
13 {
14     private JPanel mousePanel; // panel in which mouse events will occur
15     private JLabel statusBar; // label that displays event information
16
17     // MouseTrackerFrame constructor sets up GUI and
18     // registers mouse event handlers
19     public MouseTrackerFrame()
20     {
21         super( "Demonstrating Mouse Events" );
22
23         mousePanel = new JPanel(); // create panel
24         mousePanel.setBackground( Color.WHITE ); // set
25         add( mousePanel, BorderLayout.CENTER ); // add
26
27         statusBar = new JLabel( "Mouse outside JPanel" );
28         add( statusBar, BorderLayout.SOUTH ); // add Label to JFrame
29

```

Create JPanel to capture mouse events

Set background to white

Create JLabel and add to application



```
30 // create and register listener for mouse and mouse motion events
```

```
31 MouseHandler handler = new MouseHandler();
```

Create event handler for mouse events

```
32 mousePanel.addMouseListener( handler );
```

```
33 mousePanel.addMouseMotionListener( handler );
```

Register event handler

```
34 } // end MouseTrackerFrame constructor
```

```
35 private class MouseHandler implements MouseListener,
```

```
36     MouseMotionListener
```

```
37 {
```

```
38     // MouseListener event handlers
```

```
39     // handle event when mouse released immediately after press
```

```
40     public void mouseClicked( MouseEvent event )
```

```
41     {
```

Implement mouse listener interfaces

```
42         statusBar.setText( String.format( "Clicked at [%d, %d]",
```

Declare mouseClicked method

```
43             event.getX(), event.getY() ) );
```

```
44     } // end method mouseClicked
```

Find location of mouse click

```
45     // handle event when mouse pressed
```

```
46     public void mousePressed( MouseEvent event )
```

```
47     {
```

Declare mousePressed method

```
48         statusBar.setText( String.format( "Pressed at [%d, %d]",
```

```
49             event.getX(), event.getY() ) );
```

```
50     } // end method mousePressed
```

```
51     // handle event when mouse released after dragging
```

```
52     public void mouseReleased( MouseEvent event )
```

```
53     {
```

Declare mouseReleased method

```
54         statusBar.setText( String.format( "Released at [%d, %d]",
```

```
55             event.getX(), event.getY() ) );
```

```
56     } // end method mouseReleased
```



## Outline

```
60  
61 // handle event when mouse enters area  
62 public void mouseEntered( MouseEvent event )  
63 {  
64     statusBar.setText( String.format( "Mouse entered at [ %d, %d ]" ,  
65         event.getX() , event.getY() ) );  
66     mousePanel.setBackground( Color.GREEN );  
67 } // end method mouseEntered
```

Declare mouseEntered method

MouseTracker  
ava

Set background of JPanel

(3 of 4)

```
68  
69 // handle event when mouse exits area  
70 public void mouseExited( MouseEvent event )  
71 {  
72     statusBar.setText( "Mouse outside JPanel" );  
73     mousePanel.setBackground( Color.WHITE );  
74 } // end method mouseExited
```

Declare mouseExited method

Set background of JPanel



```
76 // MouseMotionListener event handlers  
77 // handle event when user drags mouse with button pressed  
78 public void mouseDragged( MouseEvent event )  
79 {  
80     statusBar.setText( String.format( "Dragged at [%d, %d]",  
81         event.getX(), event.getY() ) );  
82 } // end method mouseDragged  
83  
84 // handle event when user moves mouse  
85 public void mouseMoved( MouseEvent event )  
86 {  
87     statusBar.setText( String.format( "Moved at [%d, %d]",  
88         event.getX(), event.getY() ) );  
89 } // end method mouseMoved  
90 } // end inner class MouseHandler  
91 } // end class MouseTrackerFrame
```

## Outline

Declare mouseDragged method

MouseTracker  
Frame.java

(4 of 4)

Declare mouseMoved method



## Outline

### MouseTrackerFrame.java

(1 of 2)

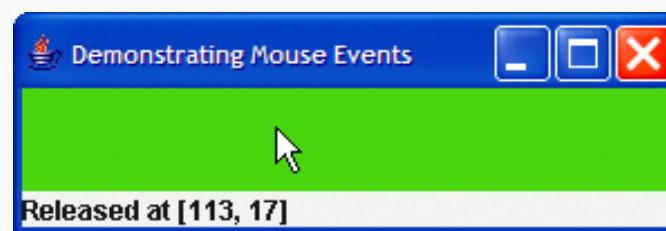
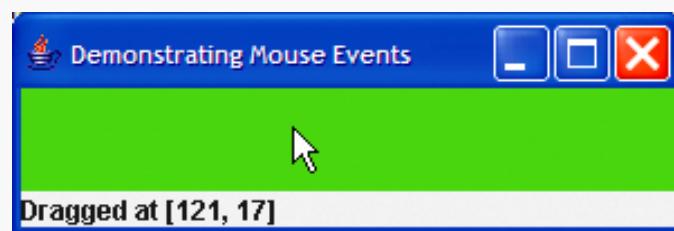
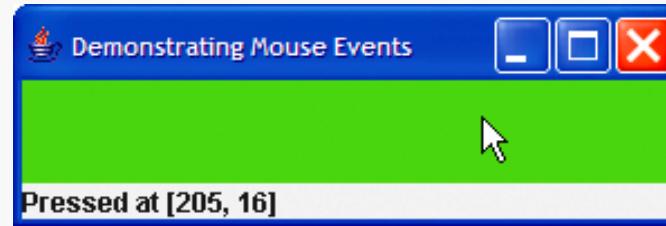
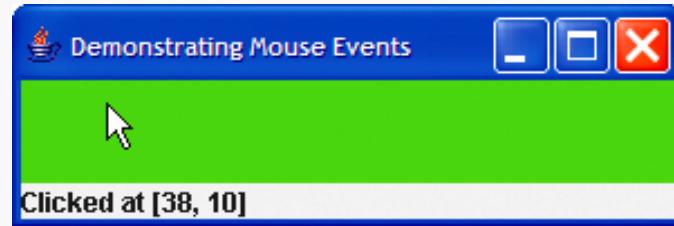
```

1 // Fig. 11.29: MouseTrackerFrame.java
2 // Testing MouseTrackerFrame.
3 import javax.swing.JFrame;
4
5 public class MouseTracker
6 {
7     public static void main( String args[] )
8     {
9         MouseTrackerFrame mouseTrackerFrame = new MouseTrackerFrame();
10        mouseTrackerFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        mouseTrackerFrame.setSize( 300, 100 ); // set frame size
12        mouseTrackerFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class MouseTracker

```



## Outline



MouseTracker  
Frame.java

(2 of 2)



## 11.14 Adapter Classes

- **Adapter class**
  - Implements event listener interface
  - Provides default implementation for all event-handling methods



# Extending MouseAdapter

- **MouseAdapter**
  - **Adapter class for MouseListener and MouseMotionListener interfaces**
  - **Extending class allows you to override only the methods you wish to use**



Event-adapter class in <code>java.awt.event</code>	Implements interface
<code>ComponentAdapter</code>	<code>ComponentListener</code>
<code>ContainerAdapter</code>	<code>ContainerListener</code>
<code>FocusAdapter</code>	<code>FocusListener</code>
<code>KeyAdapter</code>	<code>KeyListener</code>
<code>MouseAdapter</code>	<code>MouseListener</code>
<code>MouseMotionAdapter</code>	<code>MouseMotionListener</code>
<code>windowAdapter</code>	<code>windowListener</code>

**Fig. 11.30 | Event-adapter classes and the interfaces they implement in package `java.awt.event`.**



```
1 // Fig. 11.31: MouseDetailsFrame.java
2 // Demonstrating mouse clicks and distinguishing between mouse buttons.
3 import java.awt.BorderLayout;
4 import java.awt.Graphics;
5 import java.awt.event.MouseAdapter;
6 import java.awt.event.MouseEvent;
7 import javax.swing.JFrame;
8 import javax.swing.JLabel;
9
10 public class MouseDetailsFrame extends JFrame
11 {
12     private String details; // String representing
13     private JLabel statusBar; // JLabel that appears at bottom of window
14
15     // constructor sets title bar string and register mouse listener
16     public MouseDetailsFrame()
17     {
18         super( "Mouse clicks and buttons" );
19
20         statusBar = new JLabel( "Click the mouse" );
21         add( statusBar, BorderLayout.SOUTH );
22         addMouseListener( new MouseClickHandler() ); // add handler
23     } // end MouseDetailsFrame constructor
```

## Outline

### MouseDetailsFrame.java

(1 of 2)

Register event handler



## Outline

```

25 // inner class to handle mouse events
26 private class MouseClickHandler extends MouseAdapter
27 {
28     // handle mouse click event and determine which button was pressed
29     public void mouseClicked( MouseEvent event )
30     {
31         int xPos = event.getX(); // get x position of mouse
32         int yPos = event.getY(); // get y position of mouse
33
34         details = String.format( "Clicked %d time(s)" +
35             event.getClickCount() );
36
37         if ( event.isMetaDown() ) // right mouse button
38             details += " with right mouse button";
39         else if ( event.isAltDown() ) // middle mouse button
40             details += " with center mouse button";
41         else // left mouse button
42             details += " with left mouse button";
43
44         statusBar.setText( details ); // display message in statusBar
45     } // end method mouseClicked
46 } // end private inner class MouseClickHandler
47 } // end class MouseDetailsFrame

```

MouseDetails  
Frame.java

(2 of 2)

Get number of times mouse button  
was clicked

Test for right mouse button

Test for middle mouse button

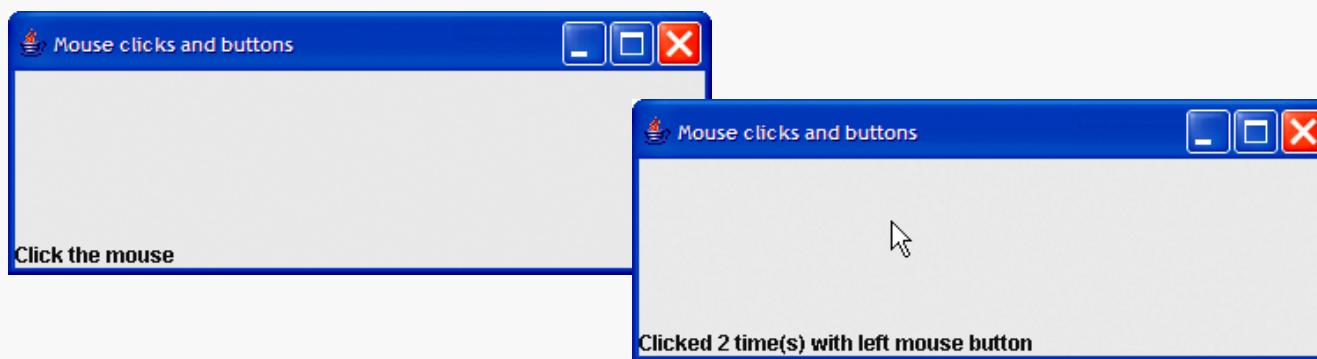


```
1 // Fig. 11.32: MouseDetails.java
2 // Testing MouseDetailsFrame.
3 import javax.swing.JFrame;
4
5 public class MouseDetails
6 {
7     public static void main( String args[] )
8     {
9         MouseDetailsFrame mouseDetailsFrame = new MouseDetailsFrame();
10        mouseDetailsFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        mouseDetailsFrame.setSize( 400, 150 ); // set frame size
12        mouseDetailsFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class MouseDetails
```

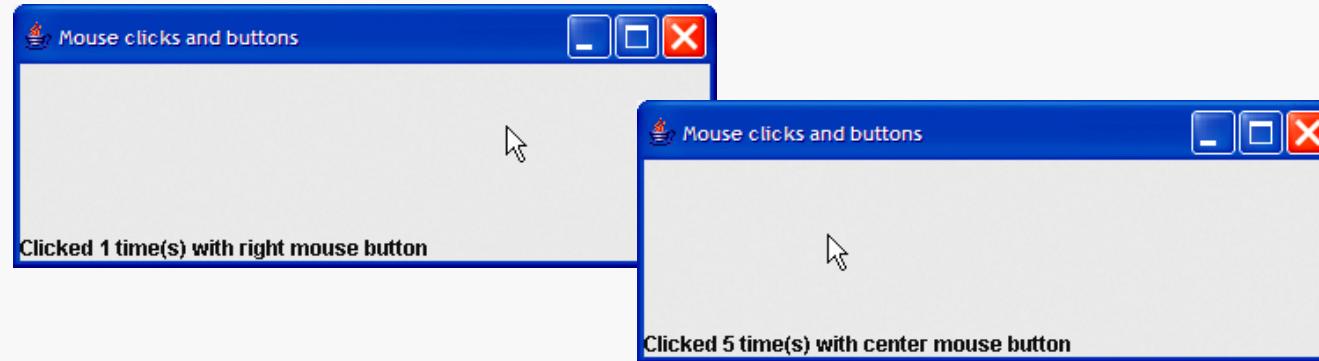
## Outline

### MouseDetails.java

(1 of 2)



## Outline



**MouseDetails  
.java**

(2 of 2)



# 11.15 JPanel Subclass for Drawing with the Mouse

- Overriding class JPanel
  - Provides a dedicated drawing area



## InputEvent method Description

`isMetaDown()`

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 hold down the *Meta* key on the keyboard and click the mouse button.

`isAltDown()`

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 only- or left-mouse button, respectively.

**Fig. 11.33 | InputEvent methods that help distinguish among left-, center- and right-mouse-button clicks.**



# Method paintComponent

- **Method paintComponent**
  - Draws on a Swing component
  - Overriding method allows you to create custom drawings
  - Must call superclass method first when overridden



# Defining the Custom Drawing Area

- **Customized subclass of JPanel**
  - Provides custom drawing area
  - Class **Graphics** is used to draw on Swing components
  - Class **Point** represents an *x-y* coordinate



```
1 // Fig. 11.34: PaintPanel.java
2 // Using class MouseMotionAdapter.
3 import java.awt.Point;
4 import java.awt.Graphics;
5 import java.awt.event.MouseEvent;
6 import java.awt.event.MouseMotionAdapter;
7 import javax.swing.JPanel;
8
9 public class PaintPanel extends JPanel
10 {
11     private int pointCount = 0; // count number of points
12
13     // array of 10000 java.awt.Point references
14     private Point points[] = new Point[ 10000 ];
15
16     // set up GUI and register mouse event handler
17     public PaintPanel()
18     {
19         // handle frame mouse motion event
20         addMouseMotionListener(
21
```

## Outline

### PaintPanel.java

(1 of 2)

Create array of Points



```

22 new MouseMotionAdapter() // anonymous inner
23 {
24     // store drag coordinates and repaint
25     public void mouseDragged( MouseEvent event )
26     {
27         if ( pointCount < points.length )
28         {
29             points[ pointCount ] = event.getPoint(); // find point
30             pointCount++; // increment number of points in array
31             repaint(); // repaint JFrame
32         } // end if
33     } // end method mouseDragged
34 } // end anonymous inner class
35 ); // end call to addMouseMotionListener
36 } // end PaintPanel constructor
37
38 // draw oval in a 4-by-4 bounding box at specified location on window
39 public void paintComponent( Graphics g )
40 {
41     super.paintComponent( g ); // clears drawing area
42
43     // draw all points in array
44     for ( int i = 0; i < pointCount; i++ )
45         g.fillOval( points[ i ].x, points[ i ].y, 4, 4 );
46 } // end method paintComponent
47 } // end class PaintPanel

```

Anonymous inner class for event handling

Override mouseDragged method

(2 of 2)

Get location of mouse cursor

Repaint the JFrame

Get the *x* and *y*-coordinates of the Point



## Look-and-Feel Observation 11.14

---

**Calling repaint for a Swing GUI component indicates that the component should be refreshed on the screen as soon as possible. The background of the GUI component is cleared only if the component is opaque. JComponent method setOpaque can be passed a boolean argument indicating whether the component is opaque (true) or transparent (false).**



## Look-and-Feel Observation 11.15

---

**Drawing on any GUI component is performed with coordinates that are measured from the upper-left corner (0, 0) of that GUI component, not the upper-left corner of the screen.**



## Outline

```
1 // Fig. 11.35: Painter.java
2 // Testing PaintPanel.
3 import java.awt.BorderLayout;
4 import javax.swing.JFrame;
5 import javax.swing.JLabel;
6
7 public class Painter
8 {
9     public static void main( String args[] )
10    {
11        // create JFrame
12        JFrame application = new JFrame( "A simple paint program" );
13
14        PaintPanel paintPanel = new PaintPanel(); // create paint panel
15        application.add( paintPanel, BorderLayout.CENTER ); // in center
16
17        // create a label and place it in SOUTH of Border
18        application.add( new JLabel( "Drag the mouse to draw" ),
19                         BorderLayout.SOUTH );
20
21        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
22        application.setSize( 400, 200 ); // set frame size
23        application.setVisible( true ); // display frame
24    } // end main
25 } // end class Painter
```

Painter.java  
(1 of 2)

Create instance of custom drawing panel



## Outline



**Painter.java**  
(2 of 2)



## 11.16 Key-Event Handling

- **KeyListener interface**
  - For handling KeyEvents
  - Declares methods keyPressed, keyReleased and keyTyped



## Outline

### KeyDemoFrame.java

(1 of 3)

```

1 // Fig. 11.36: KeyDemoFrame.java
2 // Demonstrating keystroke events.
3 import java.awt.Color;
4 import java.awt.event.KeyListener;
5 import java.awt.event.KeyEvent;
6 import javax.swing.JFrame;
7 import javax.swing.JTextArea;
8
9 public class KeyDemoFrame extends JFrame implements KeyListener
10 {
11     private String line1 = ""; // first line of text area
12     private String line2 = ""; // second line of text area
13     private String line3 = ""; // third line of text area
14     private JTextArea textArea; // textarea to display output
15
16     // KeyDemoFrame constructor
17     public KeyDemoFrame()
18     {
19         super( "Demonstrating Keystroke Events" );
20
21         textArea = new JTextArea( 10, 15 ); // set up JTextArea
22         textArea.setText( "Press any key on the keyboard" );
23         textArea.setEnabled( false ); // disable text area
24         textArea.setDisabledTextColor( Color.BLACK );
25         add( textArea ); // add textarea to JFrame
26
27         addKeyListener( this ); // allow frame to process key events
28     } // end KeyDemoFrame constructor
29

```

Implement KeyListener  
interface

Set background color

Register application as event  
handler for itself



## Outline

```

30 // handle press of any key
31 public void keyPressed( KeyEvent event )
32 {
33     line1 = String.format( "Key pressed: %s",
34         event.getKeyText( event.getKeyCode() ) ); // output pressed key
35     setLines2and3( event ); // set output lines two and three
36 } // end method keyPressed
37

```

Declare keyPressed method

Get code of pressed key

```
38 // handle release of any key
```

(2 of 3)

```
39 public void keyReleased( KeyEvent event )
```

Declare keyReleased method

```
40 {
41     line1 = String.format( "Key released: %s",
42         event.getKeyText( event.getKeyCode() ) ); // output released key
43     setLines2and3( event ); // set output lines two and three
44 } // end method keyReleased
45

```

Get code of released key

```
46 // handle press of an action key
```

Declare keyTyped method

```
47 public void keyTyped( KeyEvent event )
```

```
48 {
49     line1 = String.format( "Key typed: %s", event.getKeyChar() );
50     setLines2and3( event ); // set output lines two and three
51 } // end method keyTyped
52

```

Output character typed



## Outline

```
53 // set second and third lines of output
54 private void setLines2and3( KeyEvent event )
55 {
56     Line2 = String.format( "This key is %san action key".
57         ( event.isActionKey() ? "" : "not " ) );
58
59     String temp = event.getKeyModifiersText( event.getModifiers() );
60
61     Line3 = String.format( "Modifier keys pressed: "
62         ( temp.equals( "" ) ? "none" : temp ) );
63         // output modifiers
64
65     textArea.setText( String.format( "%s\n%s\n%s\n",
66         Line1, Line2, Line3 ) );
67 } // end method setLines2and3
68 } // end class KeyDemoFrame
```

( event.isActionKey() ? "" : "not " ) ; Test if it was an action key  
String temp = event.getKeyModifiersText( event.getModifiers() );  
Line3 = String.format( "Modifier keys pressed: " ( temp.equals( "" ) ? "none" : temp ) ); // output modifiers



## Outline

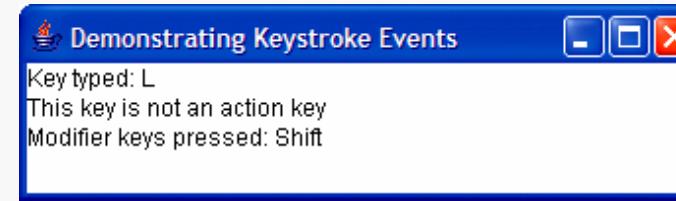
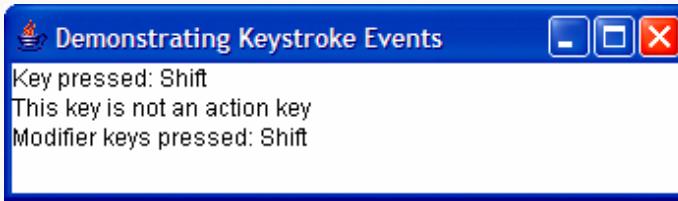
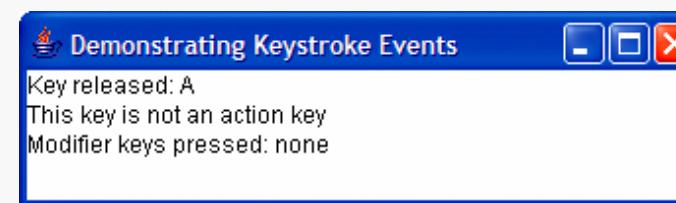
```

1 // Fig. 11.37: KeyDemo.java
2 // Testing KeyDemoFrame.
3 import javax.swing.JFrame;
4
5 public class KeyDemo
6 {
7     public static void main( String args[] )
8     {
9         KeyDemoFrame keyDemoFrame = new KeyDemoFrame();
10        keyDemoFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        keyDemoFrame.setSize( 350, 100 ); // set frame size
12        keyDemoFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class KeyDemo

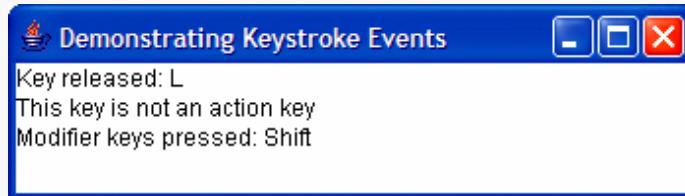
```

### KeyDemo.java

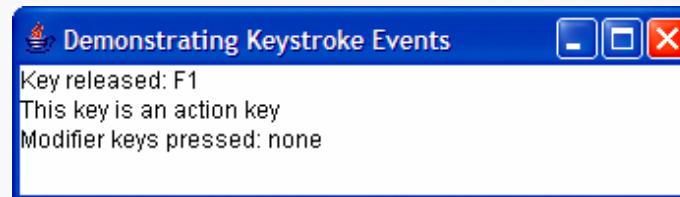
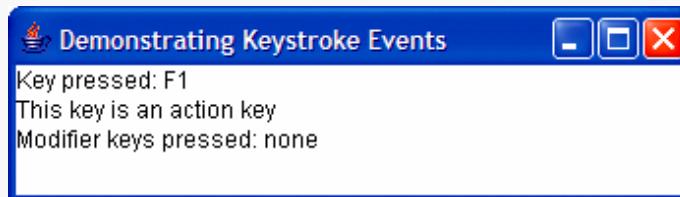
(1 of 2)



## Outline



**KeyDemo.java**  
(1 of 2)



## 11.17 Layout Managers

- **Layout managers**
  - Provided to arrange GUI components in a container
  - Provide basic layout capabilities
  - Implement the interface `LayoutManager`



## 11.17.1 FlowLayout

- **FlowLayout**
  - Simplest layout manager
  - Components are placed left to right in order they are added
  - Components can be left aligned, centered or right aligned



## Layout manager Description

<b>FlowLayout</b>	Default for <code>javax.swing.JPanel</code> . Places components sequentially (left to right) in the order they were added. It is also possible to specify the order of the components by using the <code>Container</code> method <code>add</code> , which takes a <code>Component</code> and an integer index position as arguments.
<b>BorderLayout</b>	Default for <code>JFrames</code> (and other windows). Arranges the components into five areas: <code>NORTH</code> , <code>SOUTH</code> , <code>EAST</code> , <code>WEST</code> and <code>CENTER</code> .
<b>GridLayout</b>	Arranges the components into rows and columns.

**Fig. 11.38 | Layout managers.**



```
1 // Fig. 11.39: FlowLayoutFrame.java
2 // Demonstrating FlowLayout alignments.
3 import java.awt.FlowLayout;
4 import java.awt.Container;
5 import java.awt.event.ActionListener;
6 import java.awt.event.ActionEvent;
7 import javax.swing.JFrame;
8 import javax.swing.JButton;
9
10 public class FlowLayoutFrame extends JFrame
11 {
12     private JButton leftJButton; // button to set alignment left
13     private JButton centerJButton; // button to set alignment center
14     private JButton rightJButton; // button to set alignment right
15     private FlowLayout layout; // layout object
16     private Container container; // container to set layout
17
18     // set up GUI and register button listeners
19     public FlowLayoutFrame()
20     {
21         super( "FlowLayout Demo" );
22
23         layout = new FlowLayout(); // create FlowLayout
24         container = getContentPane(); // get container to layout
25         setLayout( layout ); // set frame layout
26     }
}
```

## Outline

### FlowLayoutFrame.java

(1 of 3)

Create FlowLayout

Set layout of application



## Outline

```
// set up leftJButton and register listener
leftJButton = new JButton( "Left" ); // create Left button
add( leftJButton ); // add Left button to frame
leftJButton.addActionListener(
    new ActionListener() // anonymous inner class
{
    // process leftJButton event
    public void actionPerformed( ActionEvent event )
    {
        layout.setAlignment( FlowLayout.LEFT );
        // realign attached components
        layout.layoutContainer( container );
    } // end method actionPerformed
} // end anonymous inner class
); // end call to addActionListener
```

Add JButton; FlowLayout  
will handle placement

## FlowLayoutFrame .java

(2 of 3)

Set alignment to left

## Adjust layout

```
// set up centerJButton and register listener  
centerJButton = new JButton( "Center" ); // create Center button  
add( centerJButton ); // add Center button to frame
```

Add JButton; FlowLayout  
will handle placement

Set alignment to center



## Outline

```

57         // realign attached components
58         layout.layoutContainer( container );
59     } // end method actionPerformed
60 } // end anonymous inner class
61 ); // end call to addActionListener
62
63 // set up rightJButton and register listener
64 rightJButton = new JButton( "Right" ); // create Right button
65 add( rightJButton ); // add Right button to frame
66 rightJButton.addActionListener(
67
68     new ActionListener() // anonymous inner class
69     {
70
71         // process rightJButton event
72         public void actionPerformed( ActionEvent event )
73
74             layout.setAlignment( FlowLayout.RIGHT );
75
76             // realign attached components
77             layout.layoutContainer( container );
78         } // end method actionPerformed
79     } // end anonymous inner class
80 ); // end call to addActionListener
81 } // end FlowLayoutFrame constructor
82 } // end class FlowLayoutFrame

```

Adjust layout

Add JButton; FlowLayout will handle placement

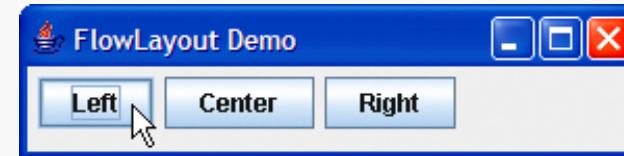
Set alignment to right

Adjust layout

FlowLayoutFrame.java  
(3 of 3)



```
1 // Fig. 11.40: FlowLayoutDemo.java
2 // Testing FlowLayoutFrame.
3 import javax.swing.JFrame;
4
5 public class FlowLayoutDemo
6 {
7     public static void main( String args[] )
8     {
9         FlowLayoutFrame flowLayoutFrame = new FlowLayoutFrame();
10        flowLayoutFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        flowLayoutFrame.setSize( 300, 75 ); // set frame size
12        flowLayoutFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class FlowLayoutDemo
```



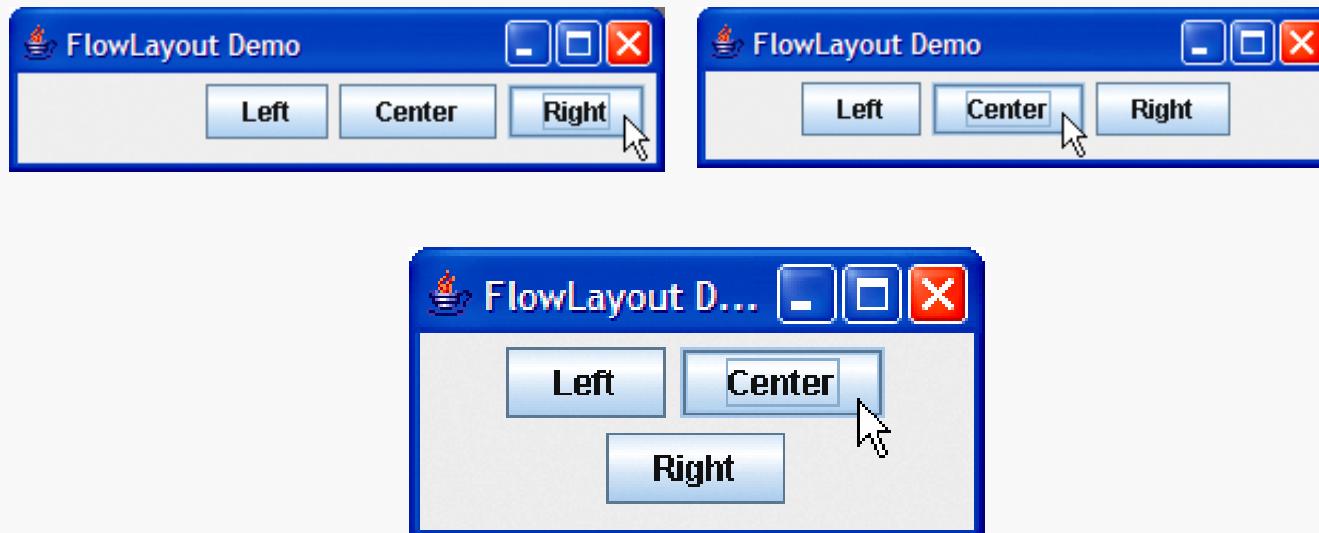
## Outline

### FlowLayoutDemo.java

(1 of 2)



## Outline



FlowLayoutDemo.java

(2 of 2)



## 11.17.2 BorderLayout

- **BorderLayout**
  - Arranges components into five regions – north, south, east, west and center
  - Implements interface LayoutManager2
  - Provides horizontal gap spacing and vertical gap spacing



## Look-and-Feel Observation 11.18

---

**Each container can have only one layout manager.  
Separate containers in the same application can  
use different layout managers.**



## Look-and-Feel Observation 11.19

---

If no region is specified when adding a Component to a BorderLayout, the layout manager assumes that the Component should be added to region BorderLayout.CENTER.



# Common Programming Error 11.6

---

**When more than one component is added to a region in a BorderLayout, only the last component added to that region will be displayed. There is no error that indicates this problem.**



## Outline

### BorderLayout Frame.java

(1 of 2)

```

1 // Fig. 11.41: BorderLayoutFrame.java
2 // Demonstrating BorderLayout.
3 import java.awt.BorderLayout;
4 import java.awt.event.ActionListener;
5 import java.awt.event.ActionEvent;
6 import javax.swing.JFrame;
7 import javax.swing.JButton;
8
9 public class BorderLayoutFrame extends JFrame implements ActionListener
10 {
11     private JButton buttons[]; // array of buttons to hide portions
12     private final String names[] = { "Hide North", "Hide South",
13         "Hide East", "Hide West", "Hide Center" };
14     private BorderLayout layout; // borderlayout object*
15
16     // set up GUI and event handling
17     public BorderLayoutFrame()
18     {
19         super( "BorderLayout Demo" );
20
21         layout = new BorderLayout( 5, 5 ); // 5 pixel gap
22         setLayout( layout ); // set frame layout
23         buttons = new JButton[ names.length ]; // set size of array
24
25         // create JButtons and register listeners for them
26         for ( int count = 0; count < names.length; count++ )
27         {
28             buttons[ count ] = new JButton( names[ count ] );
29             buttons[ count ].addActionListener( this );
30         } // end for

```

Declare BorderLayout instance variable

Create BorderLayout

Set layout

Register event handler



## Outline

### BorderLayout

```

31
32     add( buttons[ 0 ], BorderLayout.NORTH ); // add button to north
33     add( buttons[ 1 ], BorderLayout.SOUTH ); // add button to south
34     add( buttons[ 2 ], BorderLayout.EAST ); // add button to east
35     add( buttons[ 3 ], BorderLayout.WEST ); // add button to west
36     add( buttons[ 4 ], BorderLayout.CENTER ); // add button to center
37 } // end BorderLayoutFrame constructor
38
39 // handle button events
40 public void actionPerformed( ActionEvent event )
41 {
42     // check event source and Layout content pane correspondingly
43     for ( JButton button : buttons )
44     {
45         if ( event.getSource() == button )
46             button.setVisible( false ); // hide button
47         else
48             button.setVisible( true ); // show other buttons
49     } // end for
50
51     layout.layoutContainer( getContentPane() ); // layout content pane
52 } // end method actionPerformed
53 } // end class BorderLayoutFrame

```

Add buttons to application using  
layout manager constants

Make button invisible

Make button visible

Update layout



## Outline

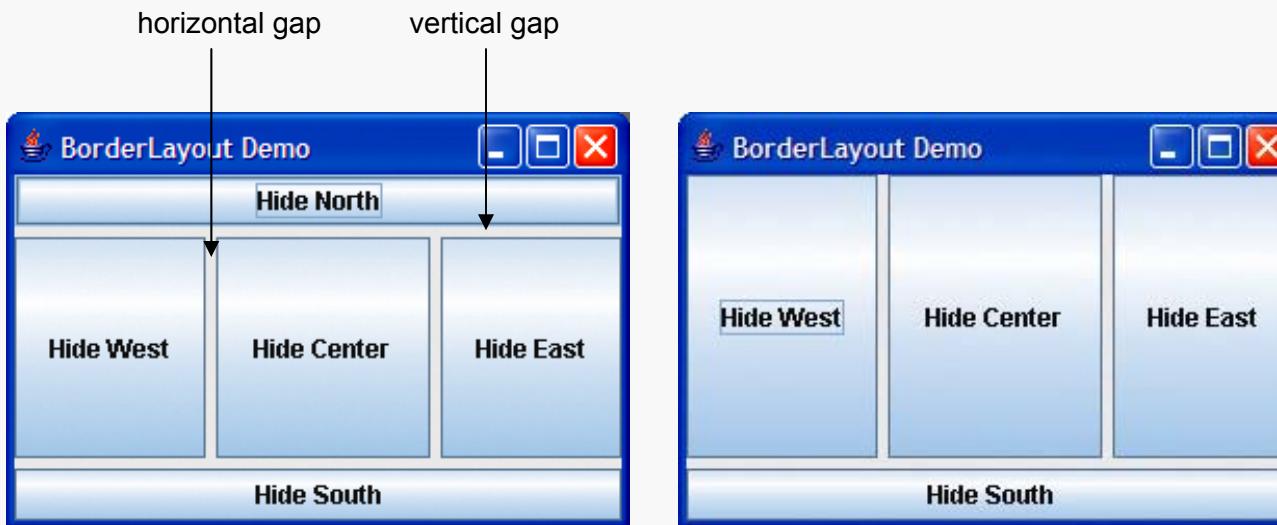
### BorderLayout Demo.java

(1 of 2)

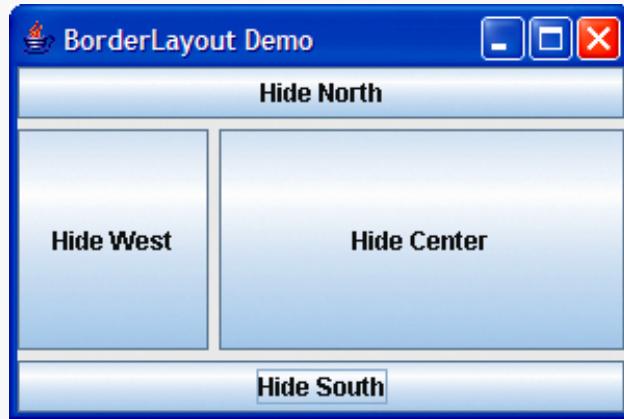
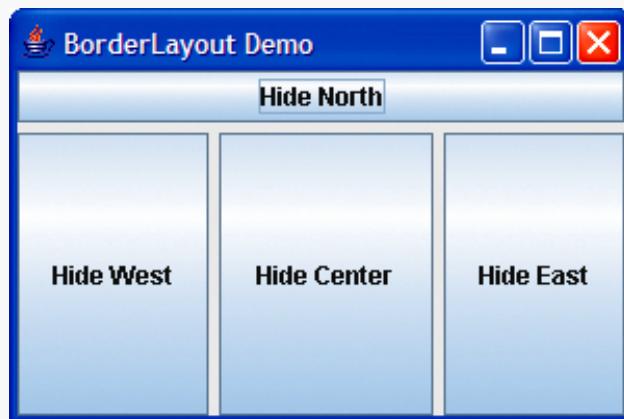
```

1 // Fig. 11.42: BorderLayoutDemo.java
2 // Testing BorderLayoutFrame.
3 import javax.swing.JFrame;
4
5 public class BorderLayoutDemo
6 {
7     public static void main( String args[] )
8     {
9         BorderLayoutFrame borderLayoutFrame = new BorderLayoutFrame();
10        borderLayoutFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        borderLayoutFrame.setSize( 300, 200 ); // set frame size
12        borderLayoutFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class BorderLayoutDemo

```



## Outline



**BorderLayout  
Demo.java**

(2 of 2)



## 11.17.3 GridLayout

- **GridLayout**

- **Divides container into a grid**
- **Every component has the same width and height**



## Outline

### GridLayoutFrame.java

(1 of 2)

```

1 // Fig. 11.43: GridLayoutFrame.java
2 // Demonstrating GridLayout.
3 import java.awt.GridLayout;
4 import java.awt.Container;
5 import java.awt.event.ActionListener;
6 import java.awt.event.ActionEvent;
7 import javax.swing.JFrame;
8 import javax.swing.JButton;
9
10 public class GridLayoutFrame extends JFrame implements ActionListener
11 {
12     private JButton buttons[]; // array of buttons
13     private final String names[] =
14         { "one", "two", "three", "four", "five", "six" };
15     private boolean toggle = true; // toggle between two grid layouts
16     private Container container; // frame container
17     private GridLayout gridLayout1; // first GridLayout
18     private GridLayout gridLayout2; // second GridLayout
19
20     // no-argument constructor
21     public GridLayoutFrame()
22     {
23         super( "GridLayout Demo" );
24         gridLayout1 = new GridLayout( 2, 3, 5, 5 ); // 2 by 3; gaps of 5
25         gridLayout2 = new GridLayout( 3, 2 ); // 3 by 2; no gaps
26         container = getContentPane(); // get content pane
27         setLayout( gridLayout1 ); // set JFrame layout
28         buttons = new JButton[ names.length ]; // create array
29

```

Declare two GridLayout instance variables

Create GridLayout

Set layout



## Outline

```

30   for ( int count = 0; count < names.length; count++ )
31   {
32     buttons[ count ] = new JButton( names[ count ] );
33     buttons[ count ].addActionListener( this ); // register listener
34     add( buttons[ count ] ); // add button to JFrame
35   } // end for
36 } // end GridLayoutFrame constructor
37
38 // handle button events by toggling between layouts
39 public void actionPerformed( ActionEvent event ) { }
40 {
41   if ( toggle )
42     container.setLayout( gridLayout2 ); // set second layout
43   else
44     container.setLayout( gridLayout1 ); // set layout to first
45
46   toggle = !toggle; // set toggle to opposite value
47   container.validate(); // re-layout container
48 } // end method actionPerformed
49 } // end class GridLayoutFrame

```

Add button to JFrame

(2 of 2)

Use second layout

Use first layout

Update layout



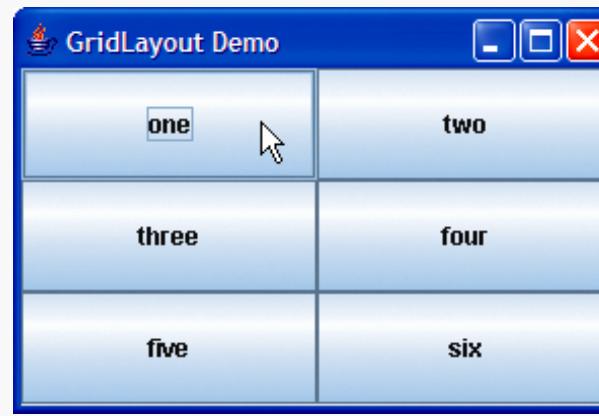
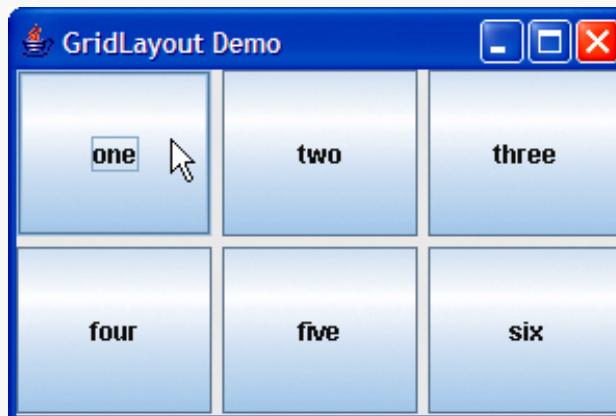
## Outline

### GridLayoutDemo.java

```

1 // Fig. 11.44: GridLayoutDemo.java
2 // Testing GridLayoutFrame.
3 import javax.swing.JFrame;
4
5 public class GridLayoutDemo
6 {
7     public static void main( String args[] )
8     {
9         GridLayoutFrame gridLayoutFrame = new GridLayoutFrame();
10        gridLayoutFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        gridLayoutFrame.setSize( 300, 200 ); // set frame size
12        gridLayoutFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class GridLayoutDemo

```



# 11.18 Using Panels to Manage More Complex Layouts

- Complex GUIs often require multiple panels to arrange their components properly



## Outline

### PanelFrame.java

(1 of 2)

```
1 // Fig. 11.45: PanelFrame.java
2 // Using a JPanel to help lay out components.
3 import java.awt.GridLayout;
4 import java.awt.BorderLayout;
5 import javax.swing.JFrame;
6 import javax.swing.JPanel;
7 import javax.swing.JButton;
8
9 public class PanelFrame extends JFrame
10 {
11     private JPanel buttonJPanel; // panel to hold buttons
12     private JButton buttons[]; // array of buttons
13
14     // no-argument constructor
15     public PanelFrame()
16     {
17         super( "Panel Demo" );
18         buttons = new JButton[ 5 ]; // create buttons array
19         buttonJPanel = new JPanel(); // set up panel
20         buttonJPanel.setLayout( new GridLayout( 1, buttons.length ) );
21     }
22 }
```

Declare a JPanel to hold buttons

Create JPanel

Set layout



```
22 // create and add buttons  
23 for ( int count = 0; count < buttons.length; count++ )  
24 {  
25     buttons[ count ] = new JButton( "Button " + ( count + 1 ) );  
26     buttonJPanel.add( buttons[ count ] ); // add button to panel  
27 } // end for  
28  
29 add( buttonJPanel, BorderLayout.SOUTH ); // add panel to JFrame  
30 } // end PanelFrame constructor  
31 } // end class PanelFrame
```

## Outline

Add button to panel

PanelFrame.java

(2 of 2)

Add panel to application



```
1 // Fig. 11.46: PanelDemo.java
2 // Testing JPanelFrame.
3 import javax.swing.JFrame;
4
5 public class PanelDemo extends JFrame
6 {
7     public static void main( String args[] )
8     {
9         JPanelFrame panelFrame = new JPanelFrame();
10        panelFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        panelFrame.setSize( 450, 200 ); // set frame size
12        panelFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class PanelDemo
```



## Outline

### PanelDemo.java



## 11.19 JTextArea

- **JTextArea**
  - Provides an area for manipulating multiple lines of text
- **Box container**
  - Subclass of Container
  - Uses a BoxLayout layout manager



## Look-and-Feel Observation 11.20

---

To provide line-wrapping functionality for a `JTextArea`, invoke `JTextArea` method `setLineWrap` with a `true` argument.



## Outline

### TextAreaFrame.java

(1 of 2)

```

1 // Fig. 11.47: TextAreaFrame.java
2 // Copying selected text from one textarea to another.
3 import java.awt.event.ActionListener;
4 import java.awt.event.ActionEvent;
5 import javax.swing.Box;
6 import javax.swing.JFrame;
7 import javax.swing.JTextArea;
8 import javax.swing.JButton;
9 import javax.swing.JScrollPane;
10
11 public class TextAreaFrame extends JFrame
12 {
13     private JTextArea textArea1; // displays demo string
14     private JTextArea textArea2; // highlighted text is copied here
15     private JButton copyJButton; // initiates copying of text
16
17     // no-argument constructor
18     public TextAreaFrame()
19     {
20         super( "TextArea Demo" );
21         Box box = Box.createHorizontalBox(); // create box
22         String demo = "This is a demo string to\n" +
23             "illustrate copying text\nfrom one textarea
24             "another textarea using an\nexternal event\n";
25
26         textArea1 = new JTextArea( demo, 10, 15 ); // create textArea1
27         box.add( new JScrollPane( textArea1 ) ); // add scrollpane
28

```

Declare JTextArea instance variables

Create a Box container

Create text area and add to box



```

29 copyJButton = new JButton( "Copy >>" ); // create copy button
30 box.add( copyJButton ); // add copy button to box
31 copyJButton.addActionListener(
32
33     new ActionListener() // anonymous inner class
34     {
35         // set text in textArea2 to selected text from textArea1
36         public void actionPerformed( ActionEvent event )
37         {
38             textArea2.setText( textArea1.getSelectedText() );
39         } // end method actionPerformed
40     } // end anonymous inner class
41 ); // end call to addActionListener
42
43 textArea2 = new JTextArea( 10, 15 ); // create second textarea
44 textArea2.setEditable( false ); // disable editing
45 box.add( new JScrollPane( textArea2 ) ); // add scrollpane
46
47 add( box ); // add box to frame
48 } // end TextAreaFrame constructor
49 } // end class TextAreaFrame

```

## Outline

Add button to box

TextAreaFrame.java

(2 of 2)

Copy selected text from one text area to the other

Create second text area and add it to box

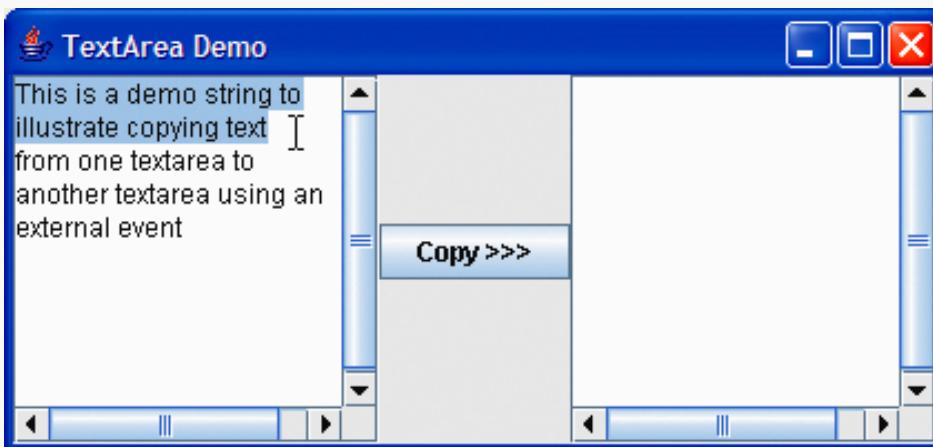


```
1 // Fig. 11.48: TextAreaDemo.java
2 // Copying selected text from one textarea to another.
3 import javax.swing.JFrame;
4
5 public class TextAreaDemo
6 {
7     public static void main( String args[] )
8     {
9         TextAreaFrame textAreaFrame = new TextAreaFrame();
10        textAreaFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        textAreaFrame.setSize( 425, 200 ); // set frame size
12        textAreaFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class TextAreaDemo
```

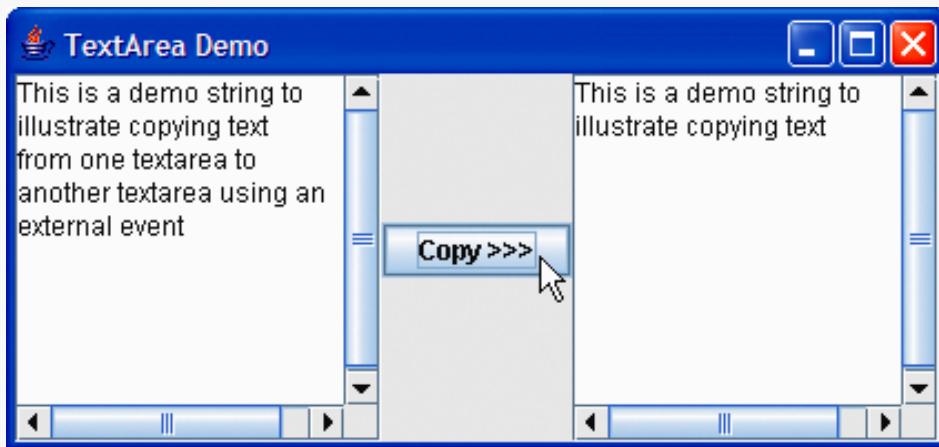
## Outline

### TextAreaDemo.java

(1 of 2)



## Outline



**TextAreaDemo.java**

(2 of 2)



# JScrollPane Scrollbar Policies

- **JScrollPane has scrollbar policies**
  - **Horizontal policies**
    - Always (HORIZONTAL\_SCROLLBAR\_ALWAYS)
    - As needed (HORIZONTAL\_SCROLLBAR\_AS\_NEEDED)
    - Never (HORIZONTAL\_SCROLLBAR\_NEVER)
  - **Vertical policies**
    - Always (VERTICAL\_SCROLLBAR\_ALWAYS)
    - As needed (VERTICAL\_SCROLLBAR\_AS\_NEEDED)
    - Never (VERTICAL\_SCROLLBAR\_NEVER)

