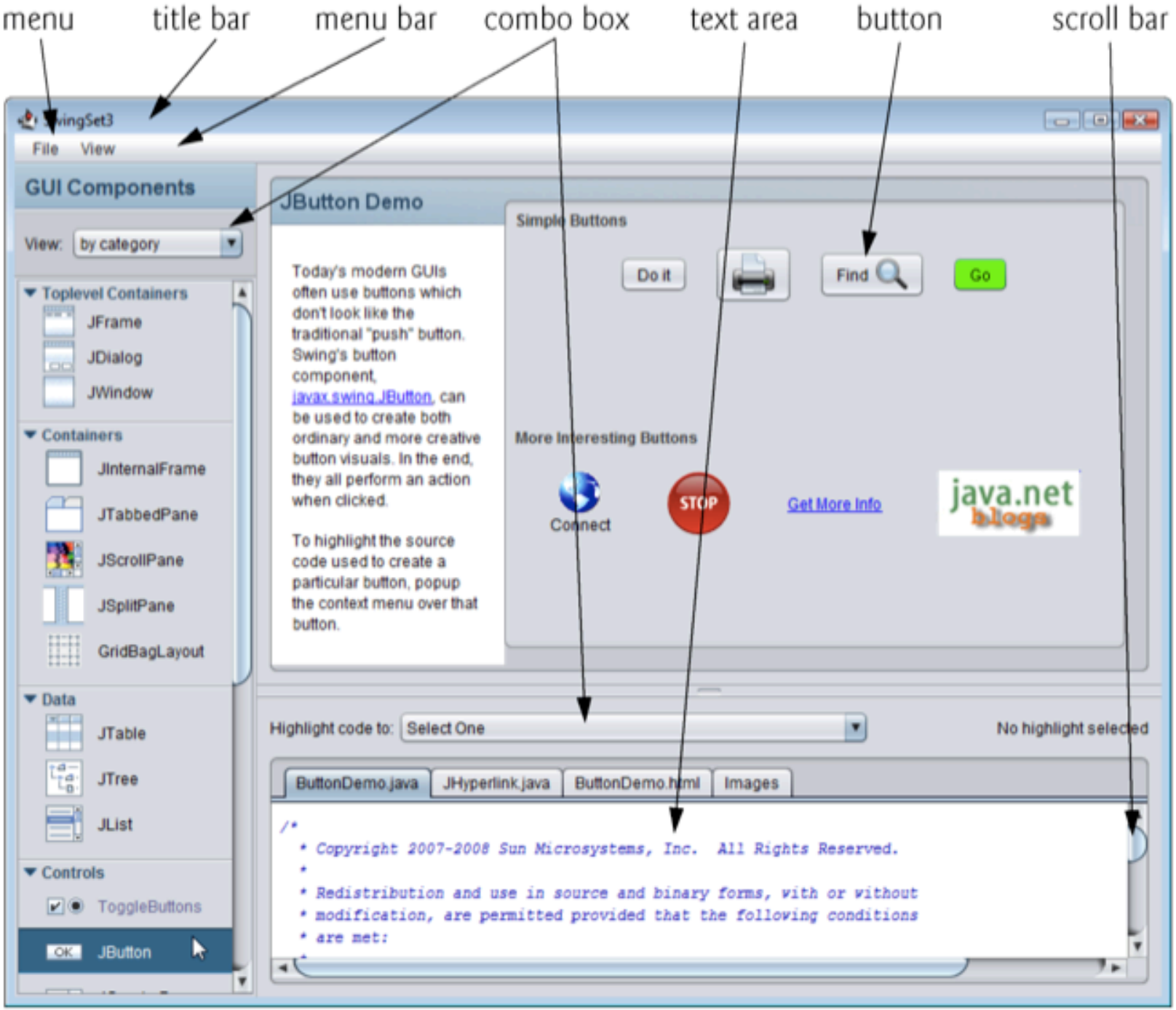
GUI Components: Part 1

Introduction

- A graphical user interface (GUI) presents a user-friendly mechanism for interacting with an application.
  - Pronounced as “GOO-ee”

Introduction (cont.)

- Created from GUI components.
  - Sometimes called controls or widgets—short for window gadgets.
- Interaction via the mouse, the keyboard or another form of input, such as voice recognition.
- IDEs
  - Provide GUI design tools to specify a component's exact size and location in a visual manner by using the mouse.
  - Generates the GUI code for you.
  - Greatly simplifies creating GUIs, but each IDE has different capabilities and generates different code.


menu
title bar
menu bar
combo box
text area
button
scroll bar
SwingSet3
File
View
GUI Components
View:
by category
Toplevel Containers
JFrame
JDialog
JWindow
Containers
JInternalFrame
JTabbedPane
JScrollPane
JSplitPane
GridBagLayout
Data
JTable
JTree
JList
Controls
ToggleButtons
JButton
JButton Demo
Today's modern GUIs often use buttons which don't look like the traditional "push" button. Swing's button component, javax.swing.JButton, can be used to create both ordinary and more creative button visuals. In the end, they all perform an action when clicked.
To highlight the source code used to create a particular button, popup the context menu over that button.
Simple Buttons
Do it
Find
Go
More Interesting Buttons
Connect
STOP
Get More Info
java.net
blogs
Highlight code to:
Select One
No highlight selected
ButtonDemo.java
JHyperlink.java
ButtonDemo.html
Images
/*
* Copyright 2007-2008 Sun Microsystems, Inc. All Rights Reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:

Overview of Swing Components

- Swing GUI components located in package javax.swing.
- Abstract Window Toolkit (AWT) in package java.awt is another set of GUI components in Java.
  - GUI components display differently on each platform.
- Swing GUI components allow you to specify a uniform look-and-feel for your application across all platforms or to use each platform's custom look-and-feel.

Component	Description
`JLabel`	Displays uneditable text and/or icons.
`JTextField`	Typically receives input from the user.
`JButton`	Triggers an event when clicked with the mouse.
`JCheckBox`	Specifies an option that can be selected or not selected.
`JComboBox`	A drop-down list of items from which the user can make a selection.
`JList`	A list of items from which the user can make a selection by clicking on any one of them. Multiple elements can be selected.
`JPanel`	An area in which components can be placed and organized.

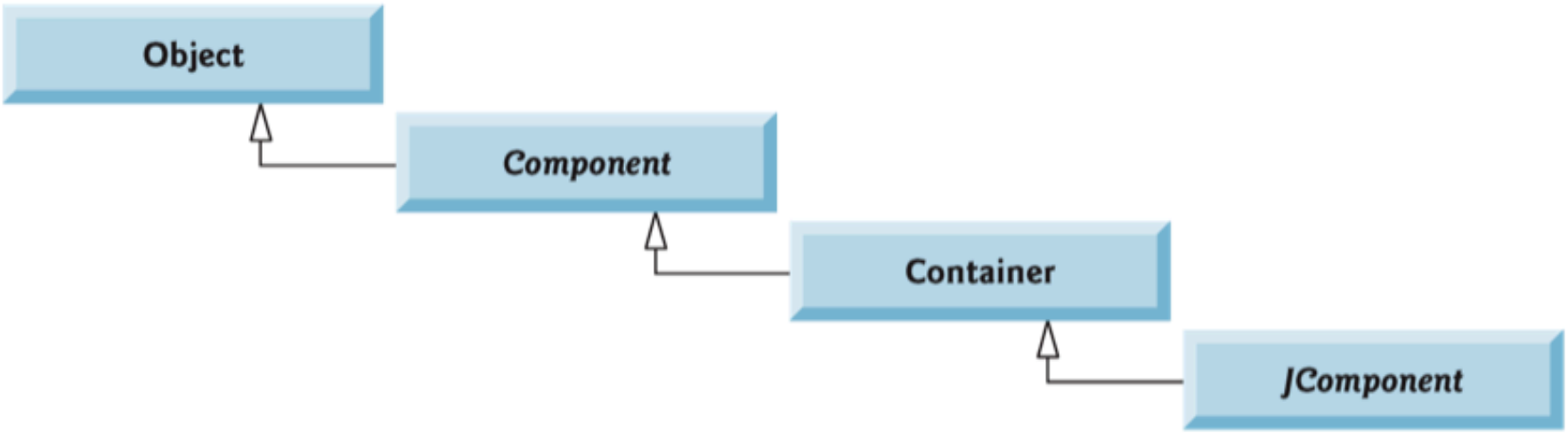
Overview of Swing Components (cont.)

- Most Swing components are not tied to actual GUI components of the underlying platform.
  - Known as lightweight components.

- AWT components are tied to the local platform and are called heavyweight components, because they rely on the local platform's windowing system to determine their functionality and their look-and-feel.

Overview of Swing Components (cont.)

- Class Component (package `java.awt`) declares many of the attributes and behaviors common to the GUI components in packages `java.awt` and `javax.swing`.
- Most GUI components extend class `Component` directly or indirectly.


Object
Component
Container
JComponent

Overview of Swing Components (cont.)

- Class Container (package `java.awt`) is a subclass of `Component`.
- `Component`s are attached to `Container`s so that they can be organized and displayed on the screen.
- Any object that *is a* `Container` can be used to organize other `Component`s in a GUI.
- Because a `Container` *is a* `Component`, you can place `Container`s in other `Container`s to help organize a GUI.

Overview of Swing Components (cont.)

- Class JComponent (package `javax.swing`) is a subclass of `Container`.

- `JComponent` is the superclass of all lightweight Swing components, all of which are also `Container`s.

Displaying Text and Images in a Window

- Most windows that can contain Swing GUI components are instances of class `JFrame` or a subclass of `JFrame`.

- Provides the basic attributes and behaviors of a window
  - a title bar at the top
  - buttons to minimize, maximize and close the window

- Most of our examples will consist of two classes
  - a subclass of `JFrame` that demonstrates new GUI concepts
  - an application class in which `main` creates and displays the application's primary window.

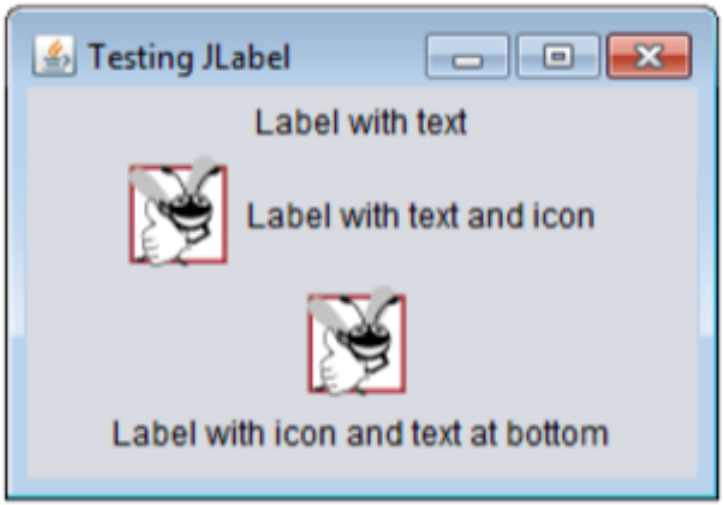
Displaying Text and Images in a Window (cont.)

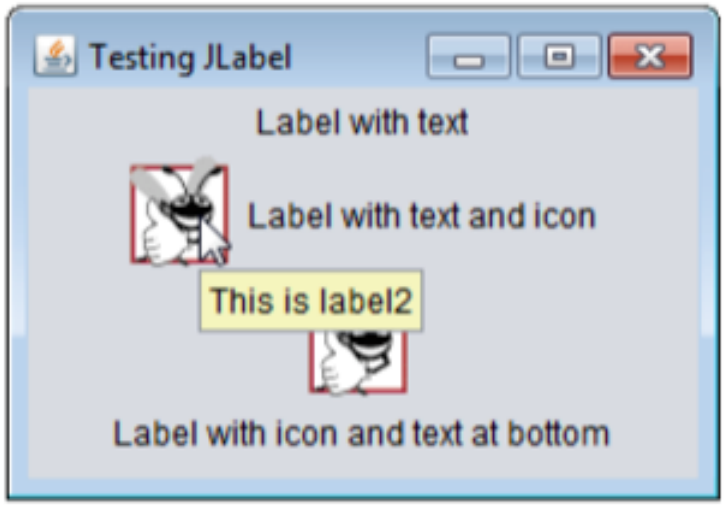
- In a large GUI
  - Difficult to identify the purpose of every component.
  - Provide text stating each component's purpose.
- Such text is known as a label and is created with class `JLabel`—a subclass of `JComponent`.
  - Displays read-only text, an image, or both text and an image.

```java
// Fig. 14.6: LabelFrame.java
// Demonstrating the JLabel class.
import java.awt.FlowLayout; // specifies how components are arranged
import javax.swing.JFrame; // provides basic window features
import javax.swing.JLabel; // displays text and images
import javax.swing.SwingConstants; // common constants used with Swing
import javax.swing.Icon; // interface used to manipulate images
import javax.swing.ImageIcon; // loads images

public class LabelFrame extends JFrame
{
   private JLabel label1; // JLabel with just text
   private JLabel label2; // JLabel constructed with text and icon
   private JLabel label3; // JLabel with added text and icon

   // LabelFrame constructor adds JLabels to JFrame
   public LabelFrame()
   {
      super( "Testing JLabel" );
      setLayout( new FlowLayout() ); // set frame layout

```

```
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
27          // JLabel constructor with string, Icon and alignment arguments
28          Icon bug = new ImageIcon( getClass().getResource(  "bug1.png" ) );
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
```

```java
// Fig. 14.7: LabelTest.java
// Testing LabelFrame.
import javax.swing.JFrame;

public class LabelTest
{
   public static void main( String[] args )
   {
      LabelFrame labelFrame = new LabelFrame(); // create LabelFrame
      labelFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      labelFrame.setSize( 260, 180 ); // set frame size
      labelFrame.setVisible( true ); // display frame
   } // end main
} // end class LabelTest
```


Testing JLabel
Label with text
Label with text and icon
Label with icon and text at bottom


Testing JLabel
Label with text
Label with text and icon
This is label2
Label with icon and text at bottom

Event Handling

- GUIs are event driven.
- When the user interacts with a GUI component, the interaction—known as an event—drives the program to perform a task.
- The code that performs a task in response to an event is called an event handler, and the overall process of responding to events is known as event handling.

Event Handling (cont.)

- Coding steps:
  - Create a class that represents the event handler.
  - Implement an appropriate interface, known as an event-listener interface, in the class from *Step 1.*
  - Indicate that an object of the class from Steps 1 and 2 should be notified when the event occurs. This is known as registering the event handler.

Event Handling (cont.)

- Normally, a component's supported events are described in the Java API documentation for that component's class and its superclasses.

Text Fields and an Introduction to Event Handling with Nested Classes

- When the user types data into a `JTextField` or a `JPasswordField`, then presses *Enter*, an event occurs.
- You can type only in the text field that is "in focus."
- A component receives the focus when the user clicks the component.

Text Fields and an Introduction to Event Handling with Nested Classes (cont.)

- ActionEvent (package `java.awt.event`) occurs.
- Processed by an object that implements the interface ActionListener (package `java.awt.event`).
- To handle `ActionEvent`s, a class must implement interface `ActionListener` and declare method `actionPerformed`.
  - This method specifies the tasks to perform when an `ActionEvent` occurs.

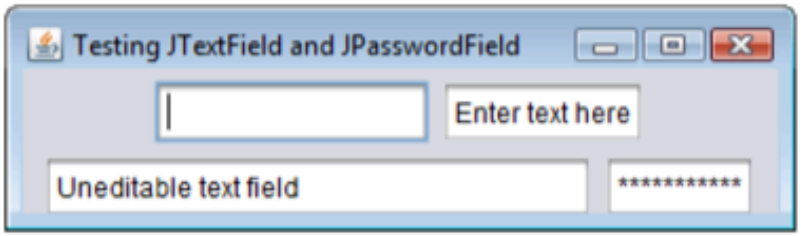
```java
// Fig. 14.9: TextFieldFrame.java
// Demonstrating the JTextField class.
import java.awt.FlowLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JFrame;
import javax.swing.JTextField;
import javax.swing.JPasswordField;
import javax.swing.JOptionPane;

public class TextFieldFrame extends JFrame
{
   private JTextField textField1; // text field with set size
   private JTextField textField2; // text field constructed with text
   private JTextField textField3; // text field with text and size
   private JPasswordField passwordField; // password field with text

```

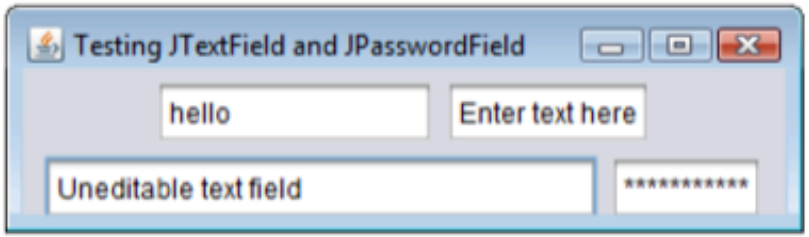
```
18      // TextFieldFrame constructor adds JTextFields to JFrame
19      public TextFieldFrame()
20      {
21         super( "Testing JTextField and JPasswordField" );
22         setLayout( new FlowLayout() ); // set frame layout
23
24         // construct textfield with 10 columns
25         textField1 = new JTextField( 10 );
26         add( textField1 ); // add textField1 to JFrame
27
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
57             // user pressed Enter in JTextField textField1
58             if ( event.getSource() == textField1 )
59                string = String.format( "textField1: %s",
60                   event.getActionCommand() );
61
```

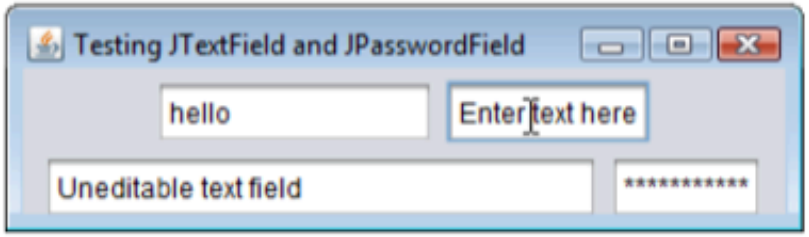
```
62            // user pressed Enter in JTextField textField2
63            else if ( event.getSource() == textField2 )
64               string = String.format( "textField2: %s",
65                  event.getActionCommand() );
66
67            // user pressed Enter in JTextField textField3
68            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                  event.getActionCommand() );
76
77            // display JTextField content
78            JOptionPane.showMessageDialog( null, string );
79         } // end method actionPerformed
80      } // end private inner class TextFieldHandler
81   } // end class TextFieldFrame
```

```java
// Fig. 14.10: TextFieldTest.java
// Testing TextFieldFrame.
import javax.swing.JFrame;

public class TextFieldTest
{
   public static void main( String[] args )
   {
      TextFieldFrame textFieldFrame = new TextFieldFrame();
      textFieldFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      textFieldFrame.setSize( 350, 100 ); // set frame size
      textFieldFrame.setVisible( true ); // display frame
   } // end main
} // end class TextFieldTest
```

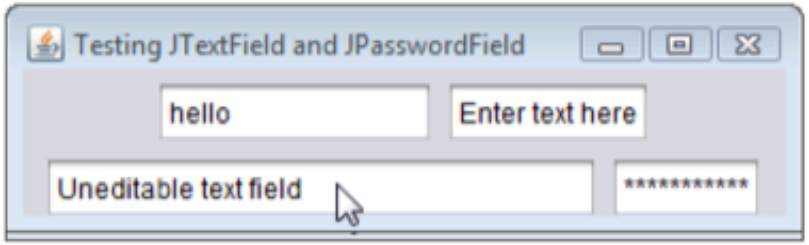


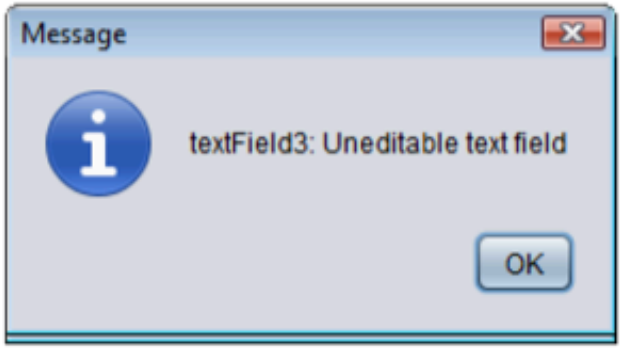

Testing JTextField and JPasswordField
hello
Enter text here
Uneditable text field
****

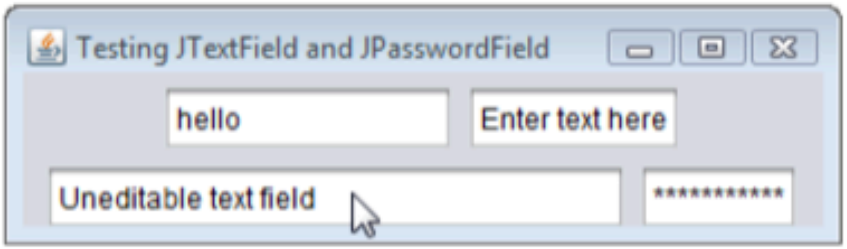

Message
textField1: hello
OK


Testing JTextField and JPasswordField
hello
Enter text here
Uneditable text field
****

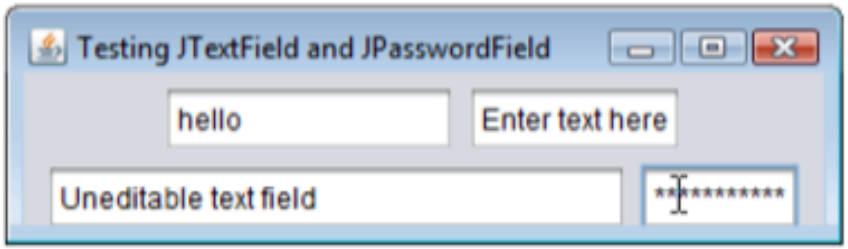

Message
textField2: Enter text here
OK


Testing JTextField and JPasswordField
hello
Enter text here
Uneditable text field
****


Message
textField3: Uneditable text field
OK

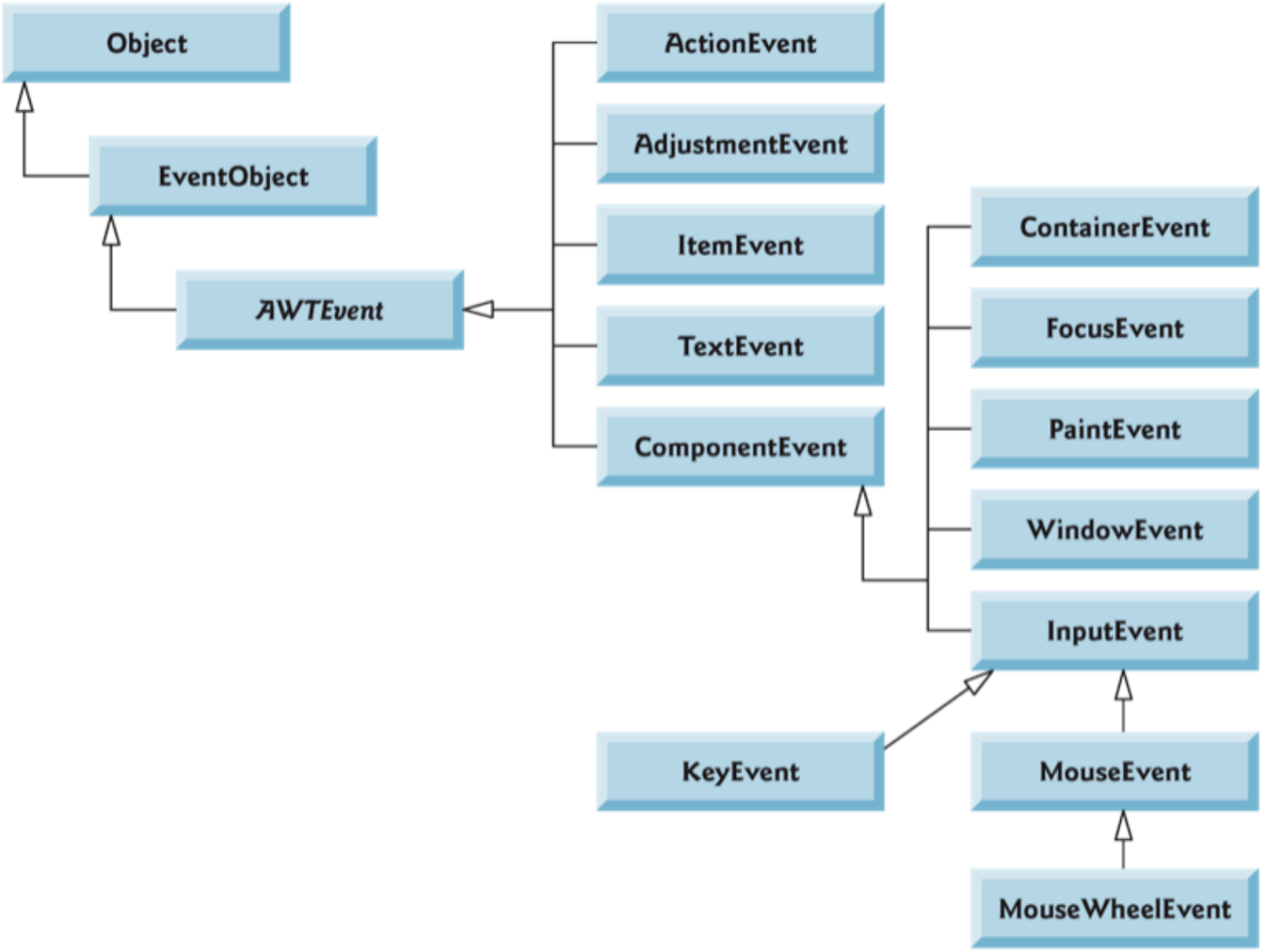

Testing JTextField and JPasswordField
hello
Enter text here
Uneditable text field
****

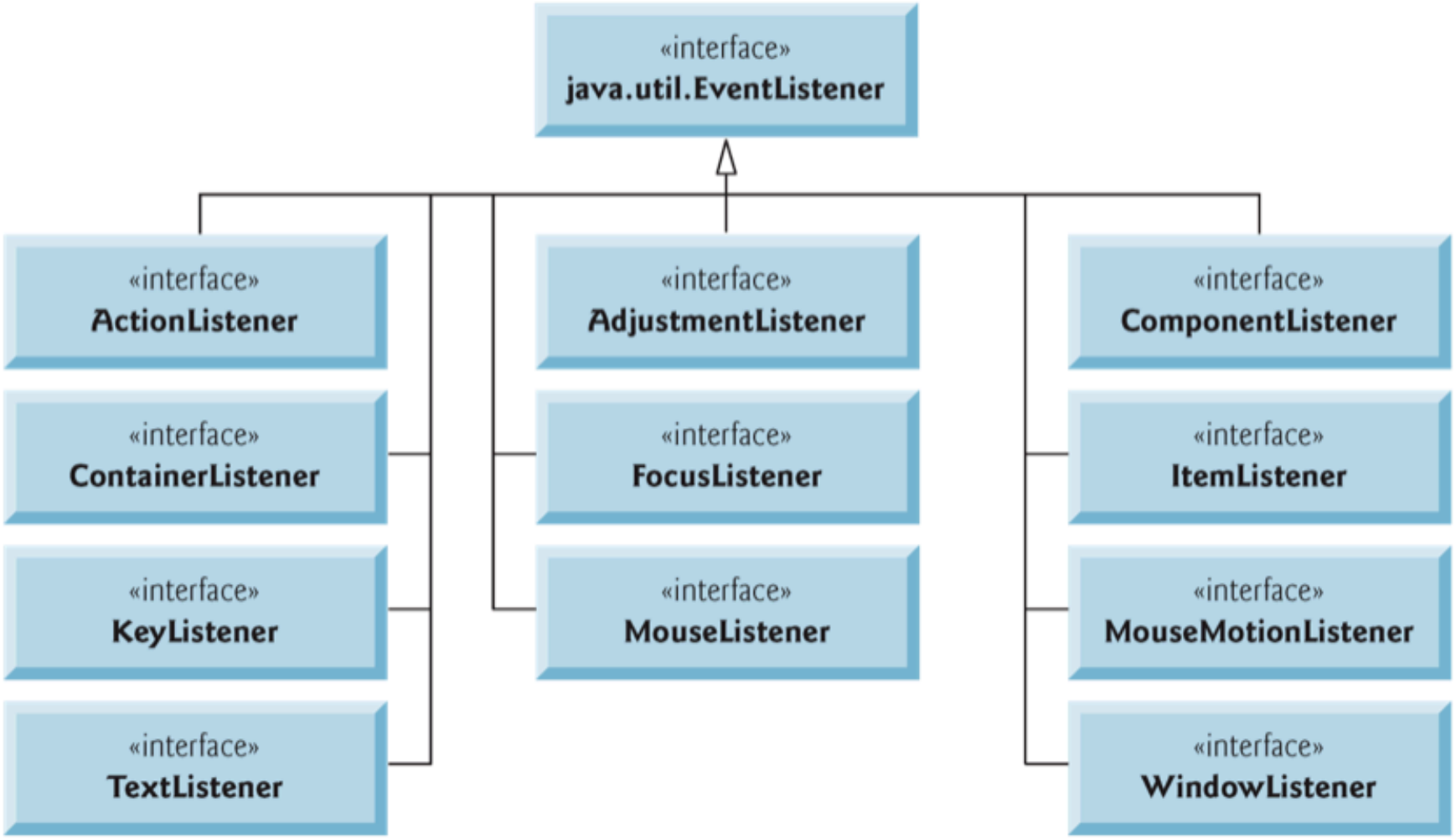

Message
textField3: Uneditable text field
OK


Testing JTextField and JPasswordField
hello
Enter text here
Uneditable text field
****

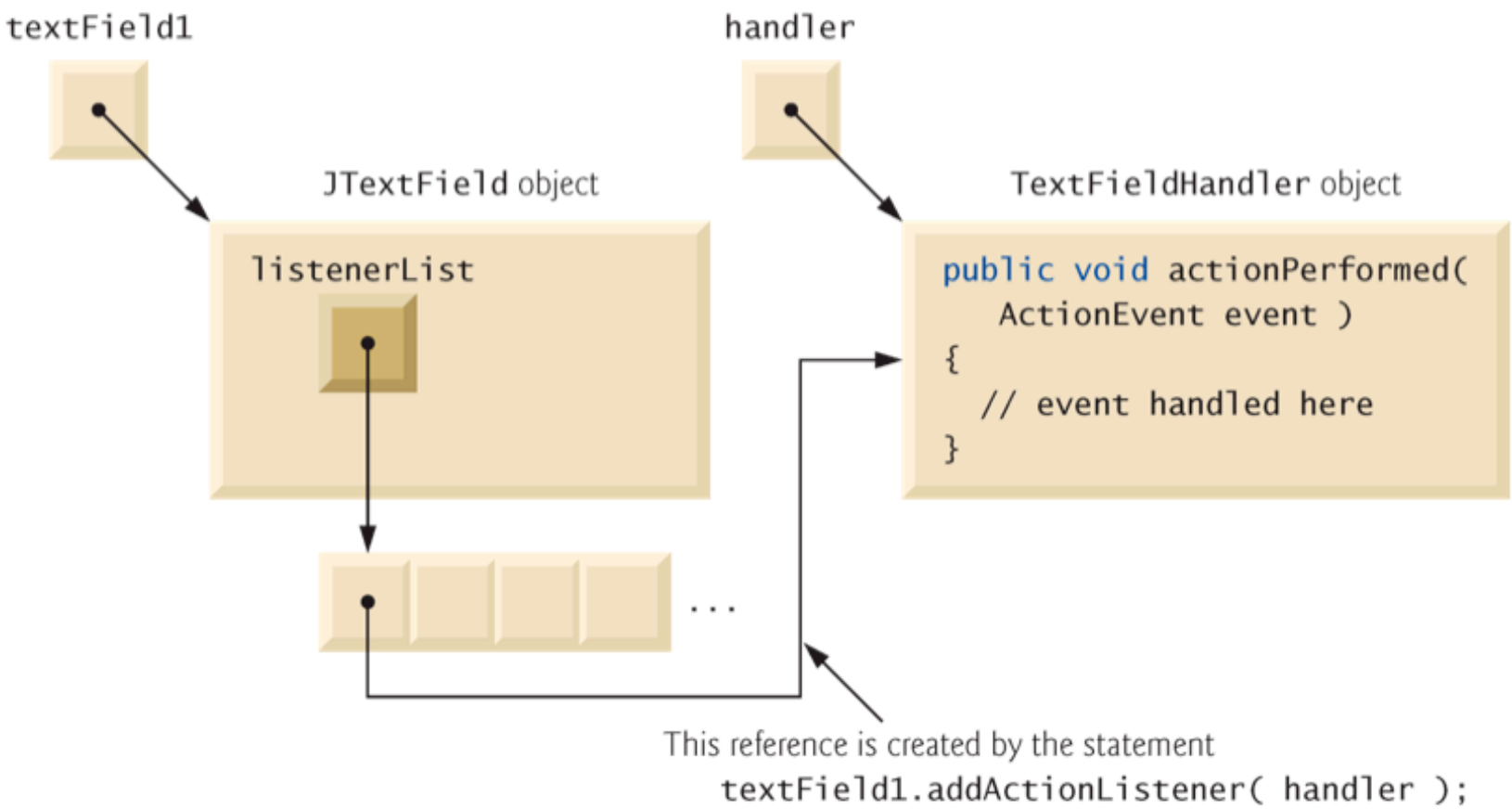

Message
passwordField: Hidden text
OK

Common GUI Event Types and Listener Interfaces



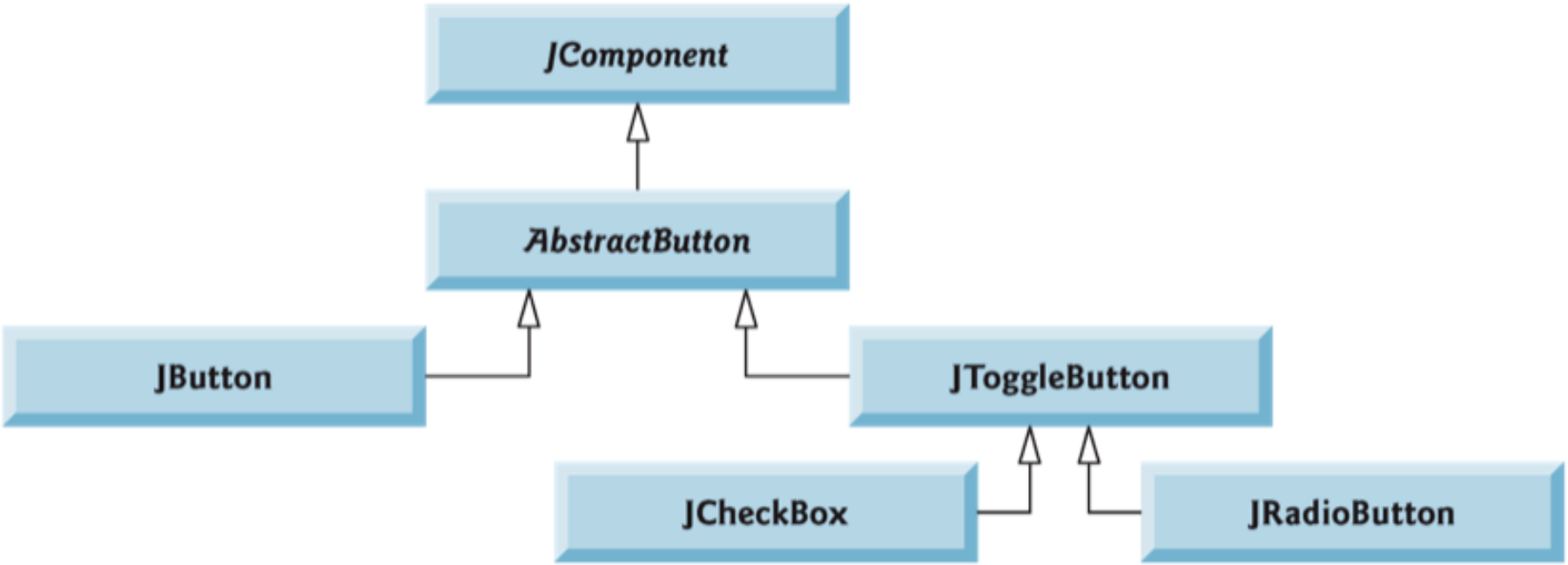

«interface»
java.util.EventListener
«interface»
ActionListener
«interface»
ContainerListener
«interface»
KeyListener
«interface»
TextListener
«interface»
AdjustmentListener
«interface»
FocusListener
«interface»
MouseListener
«interface»
ComponentListener
«interface»
ItemListener
«interface»
MouseMotionListener
«interface»
WindowListener

How Event Handling Works



`JButton`

- A button is a component the user clicks to trigger a specific action.
- Several types of buttons
  - command buttons
  - checkboxes
  - toggle buttons
  - radio buttons
- Button types are subclasses of AbstractButton (package `javax.swing`), which declares the common features of Swing buttons.


JComponent
AbstractButton
JButton
JToggleButton
JCheckBox
JRadioButton

`JButton` (cont.)

- A command button generates an `ActionEvent` when the user clicks it.
- Command buttons are created with class JButton.
- The text on the face of a `JButton` is called a button label.

```java
// Fig. 14.15: ButtonFrame.java
// Creating JButtons.
import java.awt.FlowLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JOptionPane;

public class ButtonFrame extends JFrame
{
   private JButton plainJButton; // button with just text
   private JButton fancyJButton; // button with icons

   // ButtonFrame adds JButtons to JFrame
   public ButtonFrame()
   {
      super( "Testing Buttons" );
      setLayout( new FlowLayout() ); // set frame layout

      plainJButton = new JButton( "Plain Button" ); // button with text
      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
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
```

```java
// Fig. 14.16: ButtonTest.java
// Testing ButtonFrame.
import javax.swing.JFrame;

public class ButtonTest
{
   public static void main( String[] args )
   {
      ButtonFrame buttonFrame = new ButtonFrame(); // create ButtonFrame
      buttonFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      buttonFrame.setSize( 275, 110 ); // set frame size
      buttonFrame.setVisible( true ); // display frame
   } // end main
} // end class ButtonTest
```


Testing Buttons
Plain Button
Fancy Button
Testing Buttons
Plain Button
Fancy Button
Message
You pressed: Plain Button
OK
Testing Buttons
Plain Button
Fancy Button
Testing Buttons
Plain Button
Fancy Button
Message
You pressed: Fancy Button
OK

Buttons That Maintain State

- Three types of state buttons—JToggleButton, JCheckBox and JRadioButton—that have on/off or true/false values.
- Classes `JCheckBox` and `JRadioButton` are subclasses of `JToggleButton`.
- `JRadioButton`s are grouped together and are mutually exclusive—only one in the group can be selected at any time

Buttons That Maintain State

- When the user clicks these buttons, an ItemEvent occurs.
  - Handled by an ItemListener object, which must implement method itemStateChanged.

- An `ItemListener` is registered with method addItemListener.