كلية الهندسة المعلوماتية

برمجة 3
Java Programming

ا. د. علي عمران سليمان

محاضرات الأسبوع السابع والثامن

الفصل الثاني 2023-2024

Outline

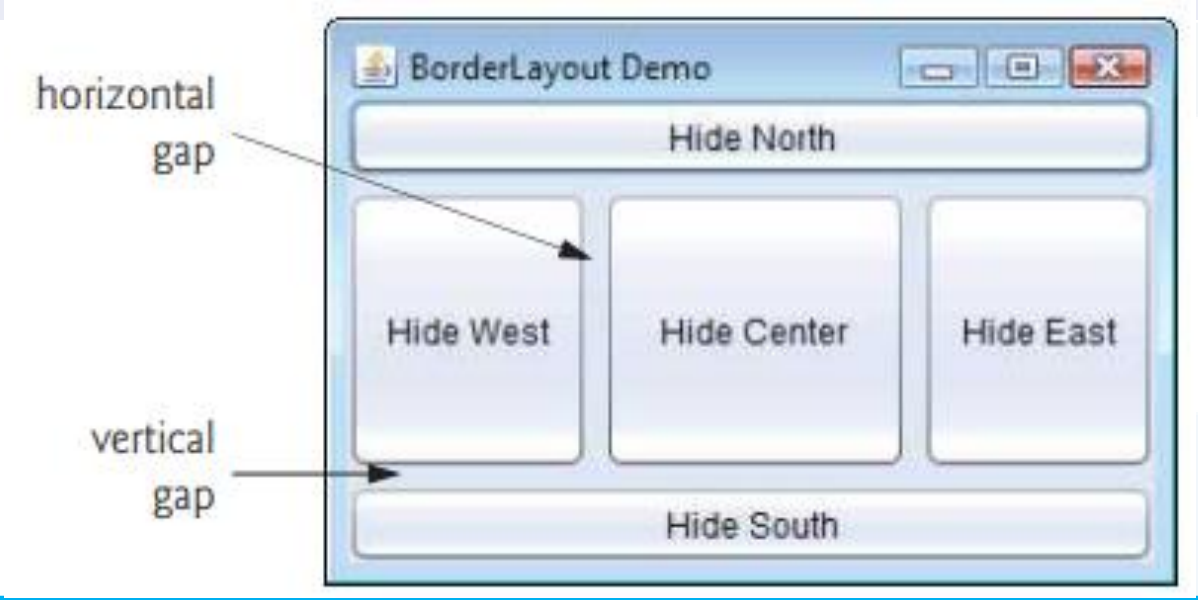
References - Deitel & Deitel, Java How to Program, Pearson; 10th Ed(2015)
- د.علي سليمان، بنى معطيات بلغة JAVA، جامعة تشرين 2013-2014

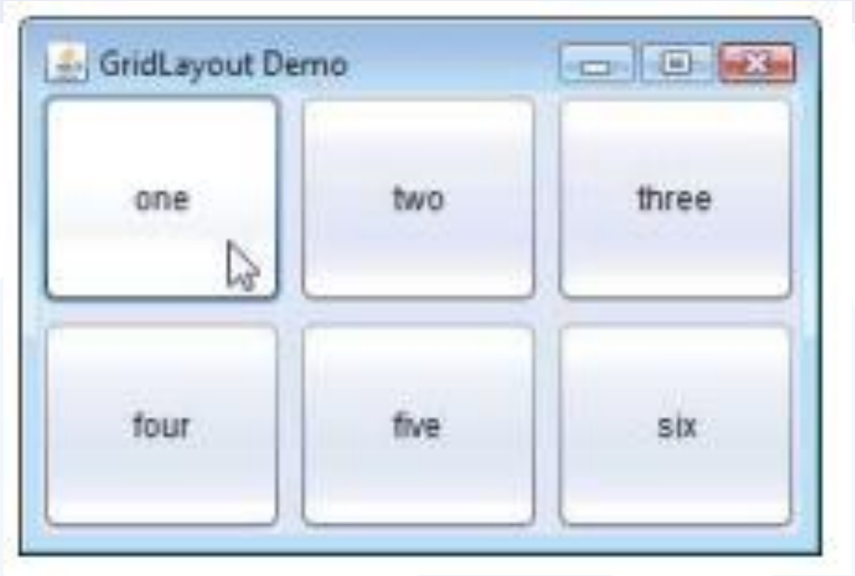
Layout Managers 1

- Absolute positioning
  - By setting a Container's layout to null, setLayout(null)
  - setBounds(x, y, w, h)
- Layout managers
  - Available for arranging GUI components
  - Processes layout details
  - Programmer can concentrate on basic "look and feel"
  - Interface `LayoutManager`
- Visual programming in an IDE

Layout Managers 2

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





Layout Managers

- `FlowLayout`
  - Most basic layout manager
  - GUI components placed in container from left to right
- `BorderLayout`
  - Arranges components into five regions

• `NORTH`	(top of container)	`SOUTH`	(bottom of container)
• `EAST`	(right side)	`WEST`	(left side)
• `CENTER`	(center of container)		

- `GridLayout`
  - Divides container into grid of specified row an columns
  - Components are added starting at top-left cell
    - Proceed left-to-right until row is full

13.4 Event Handling

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

package java.awt.event

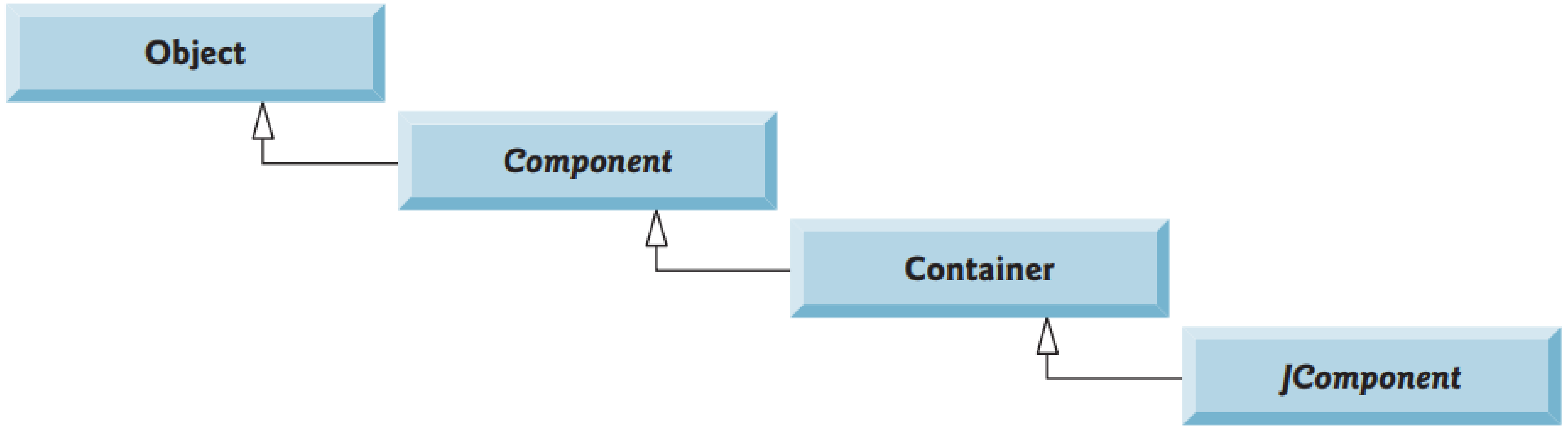


Fig. 12.5 | Common superclasses of the lightweight Swing components.

package java.awt.event

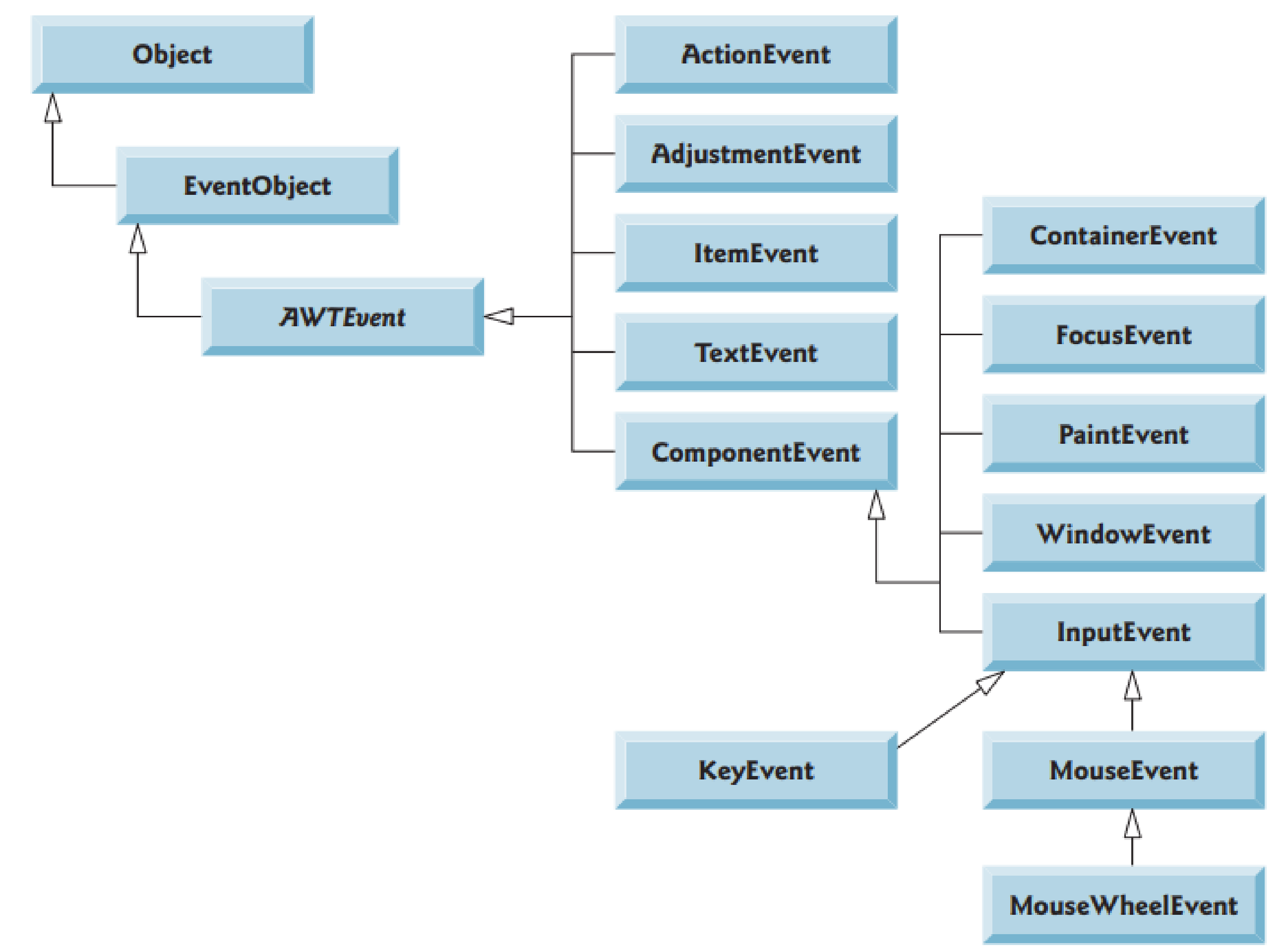


Fig. 12.11 | Some event classes of package `java.awt.event`.

13.4 Event Handling

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

java.awt.event

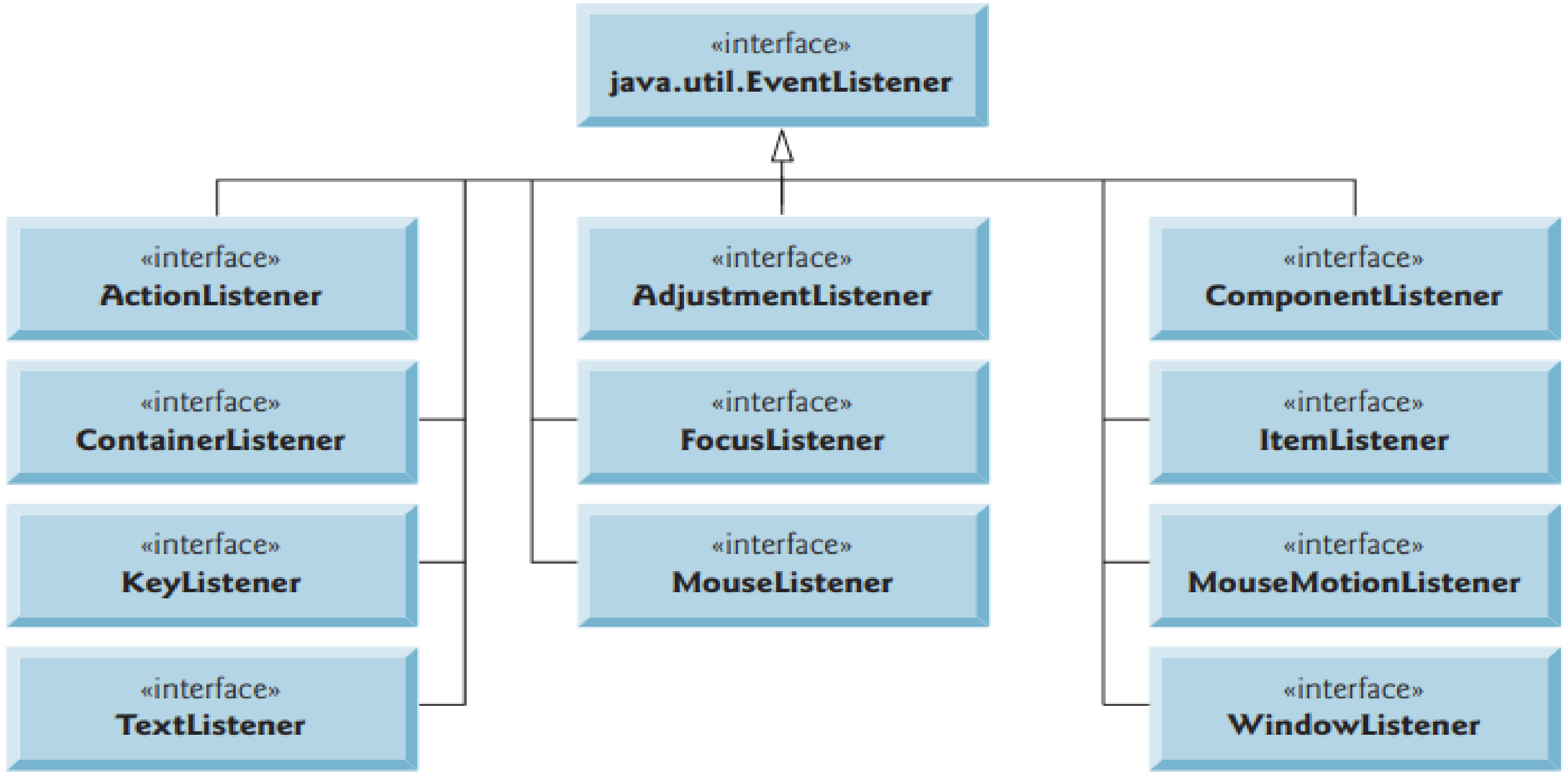


Fig. 12.12 | Some common event-listener interfaces of package `java.awt.event`.

13.5 TextFields

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

13.5 TextFields

```
package ch12GUI;

//Fig. 12.9: TextFieldFrame.java
// JTextFields and JPasswordFields.
 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 final JTextField textField1; // text field with set size
 private final JTextField textField2; // text field with text
 private final JTextField textField3; // text field with text and size
 private final JPasswordField passwordField; // password field with text
 // TextFieldFrame constructor adds JTextFields to JFrame
 public TextFieldFrame()
 { super("Testing JTextField and JPasswordField");
 setLayout(new FlowLayout());
//construct text field with 10 columns
 textField1 = new JTextField(10);
 add(textField1); // add textField1 to JFrame
//construct text field with default text
 textField2 = new JTextField("Enter text here");
  add(textField2); // add textField2 to JFrame
```

Declare three `JTextField`s and one `JPasswordField`

First `JTextField` contains empty string

Second `JTextField` contains text “`Enter text here`”

13.5 TextFields

```
  //construct text field with default text and 21 columns
  textField3 = new JTextField("Uneditable text field", 21);
  textField3.setEditable(false); // disable editing
  add(textField3); // add textField3 to JFrame
  //construct password field with default text
  passwordField = new JPasswordField("Hidden text");
  add(passwordField); // add passwordField to JFrame
  //register event handlers
  TextFieldHandler handler = new TextFieldHandler();
  textField1.addActionListener(handler);
  textField2.addActionListener(handler);
  textField3.addActionListener(handler);
  passwordField.addActionListener(handler);
  }
 //private inner class for event handling
  private class TextFieldHandler implements ActionListener
  {  // process text field events
   @Override
  public void actionPerformed(ActionEvent event)
   {  String string = "";
    // user pressed Enter in JTextField textField1
  if (event.getSource() == textField1)
  string = String.format("textField1: %s", event.getActionCommand());
```

Third `JTextField` contains uneditable text

`JPasswordField` contains text “`Hidden text`,” but text appears as series of asterisks (*)

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

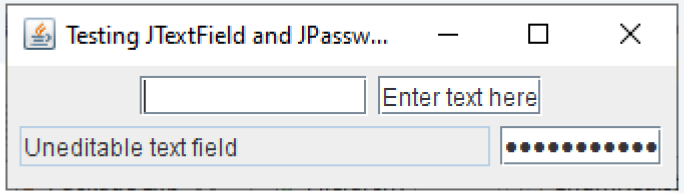
Every `TextFieldHandler` instance is an `ActionListener`

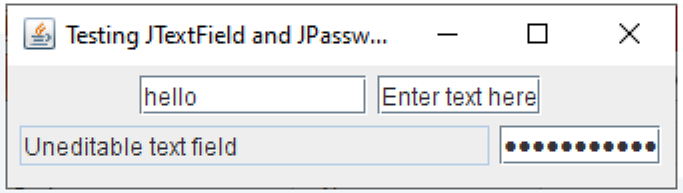
Method `actionPerformed` invoked when user presses Enter in GUI field

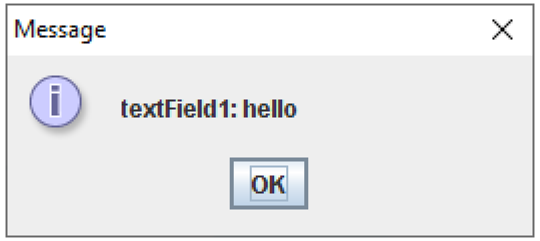
13.5 TextFields

```
// user pressed Enter in JTextField textField2
  else if (event.getSource() == textField2)
 string = String.format("textField2: %s", event.getActionCommand());
// user pressed Enter in JTextField textField3
  else if (event.getSource() == textField3)
 string = String.format("textField3: %s", event.getActionCommand());
// user pressed Enter in JTextField passwordField
  else if (event.getSource() == passwordField)
  string = String.format("passwordField: %s",  event.getActionCommand());
// display JTextField content
  JOptionPane.showMessageDialog(null, string);   }
  } // end private inner class TextFieldHandler
 } // end class TextFieldFrame

//Fig. 12.10: 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);
                   textFieldFrame.setVisible(true);  }
 } // end class TextFieldTest
```

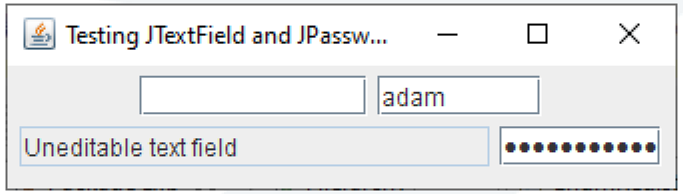
13.5 TextFields

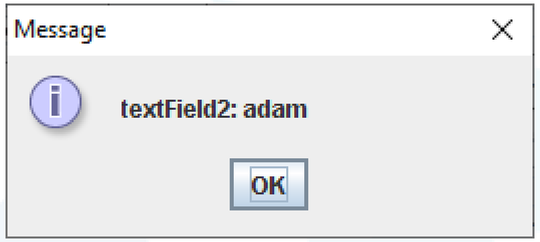




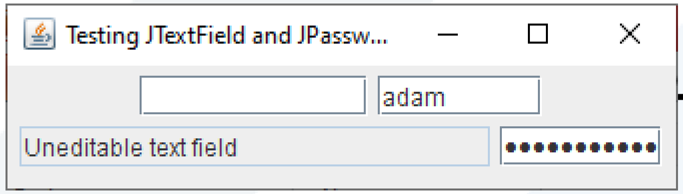


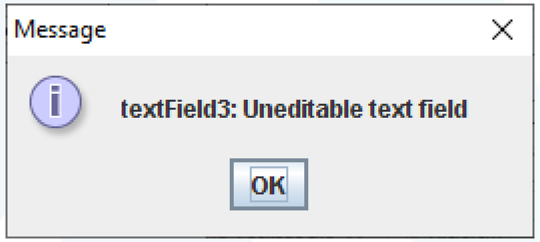
textField1: hello



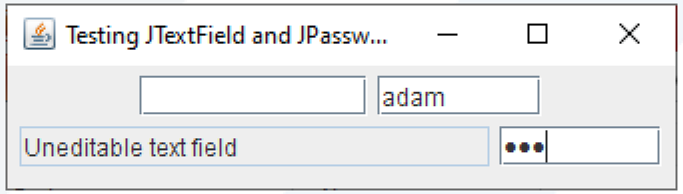


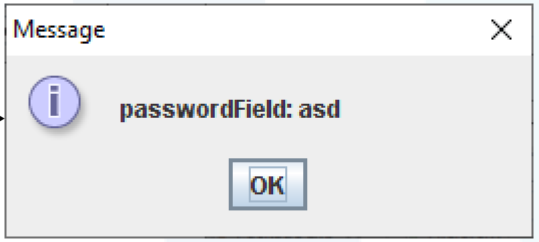
textField2: adam





textField3: Uneditable text field



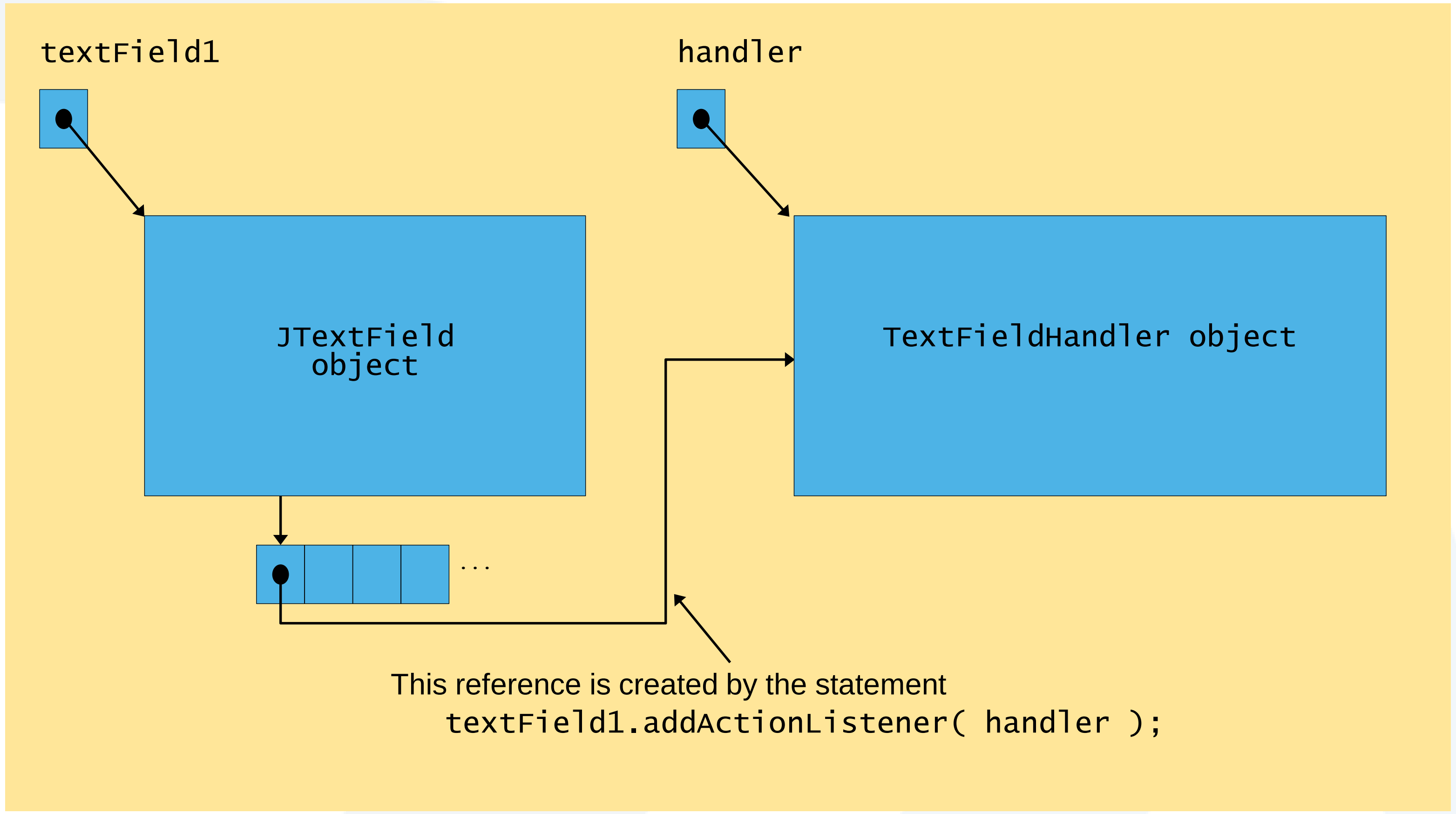


passwordField: asd

13.6 How Event Handling Works

- يتم تسجيل معالج الحدث.
- من خلال طريقة المكون addActionListenerالمضافه على المكونات مثل TextFieldTest.java.
  - يتم إرسال الحدث فقط إلى المستمعين من النوع المناسب .
  - يتم إ ستدعاء الإجراء المناسب للحدث.
  - يحتوي كل نوع حدث على واجهة مستمع الحدث المقابلة .
  - يحتوي كائن الحدث على معرف الحدث نوع الحدث ومصدره وبارامترات اخرى.

Event registration



This reference is created by the statement
```
textField1.addActionListener( handler );
```

Fig. 13.8 Event registration for `JTextField textField1`

13.7 `JButton`

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

Swing button hierarchy

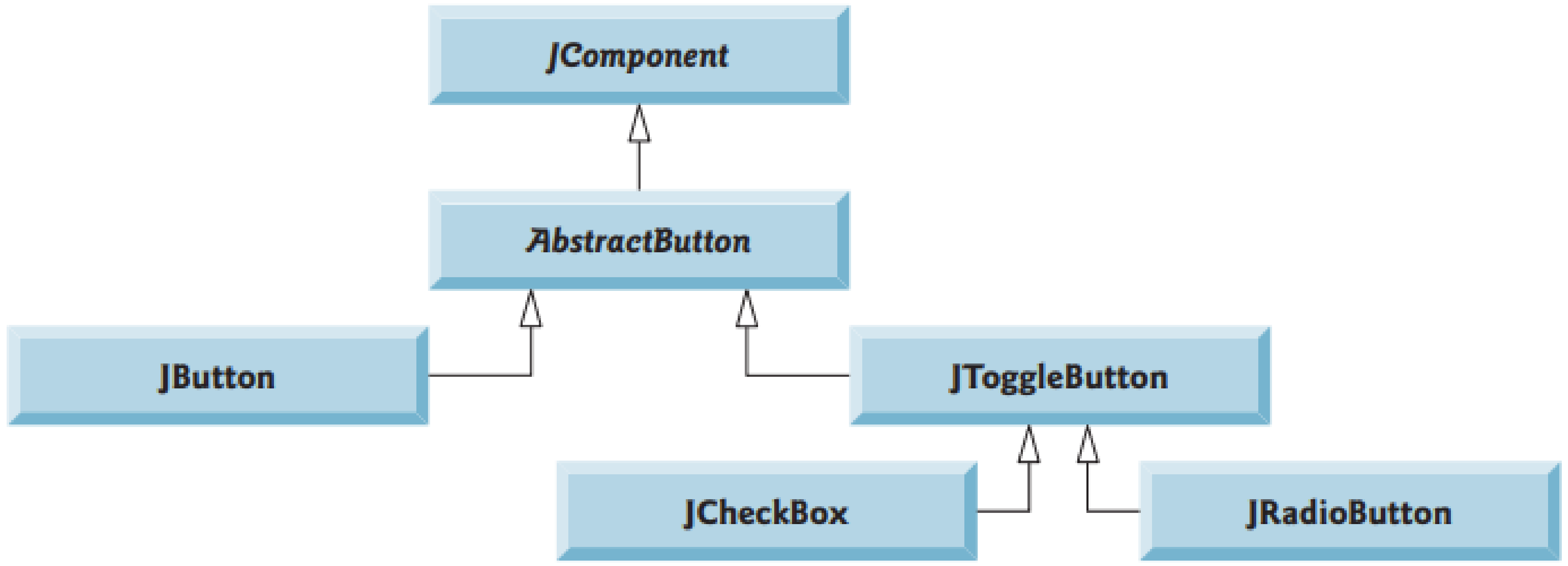


Fig. 12.14 | Swing button hierarchy.

Swing button hierarchy

```
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 final JButton plainJButton; // button with just text
private final JButton fancyJButton; // button with icons
  // ButtonFrame adds JButtons to JFrame
 public ButtonFrame()  { super("Testing Buttons");  setLayout(new FlowLayout());
 plainJButton = new JButton("Plain Button"); // button with text
 add(plainJButton); // add plainJButton to JFrame
Icon bug1 = new ImageIcon(getClass().getResource("bug1.png"));
Icon bug2 = new ImageIcon(getClass().getResource("bug2.png"));
fancyJButton = new JButton("Fancy Button", bug1); // set image
fancyJButton.setRolloverIcon(bug2); // set rollover image
add(fancyJButton); // add fancyJButton to JFrame
```

Create two references to `JButton` instances

Instantiate `JButton` with text

Instantiate `JButton` with image and *rollover* image

Swing button hierarchy

```
 // create new ButtonHandler for button event handling
ButtonHandler handler = new ButtonHandler();
fancyJButton.addActionListener(handler);
plainJButton.addActionListener(handler); }
 // inner class for button event handling
 private class ButtonHandler implements ActionListener
 { // handle button event
 @Override
 public void actionPerformed(ActionEvent event)
 { JOptionPane.showMessageDialog(ButtonFrame.this , String.format("You pressed: 
%s",event.getActionCommand() )); }
   }
 } // end class ButtonFrame
  /* Accessing the this Reference in an Object of a Top-Level Class from an Inner Class When you
  *  execute this application and click one of its buttons, notice that the message dialog that
  * appears is centered over the application's window. This occurs because the call to JOptionPane
  *   method showMessageDialog uses ButtonFrame.this rather than null as the first argument.  */

import javax.swing.JFrame;
 public class ButtonTest   {
```

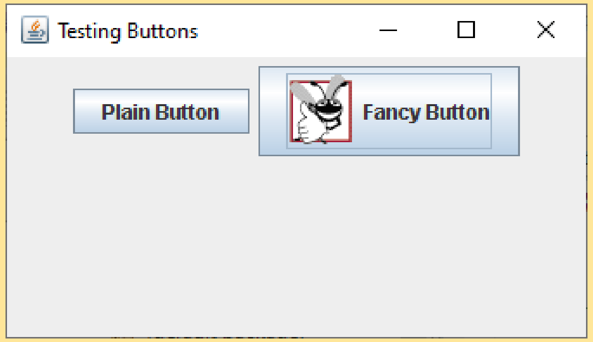
Instantiate `ButtonHandler` for `JButton` event handling

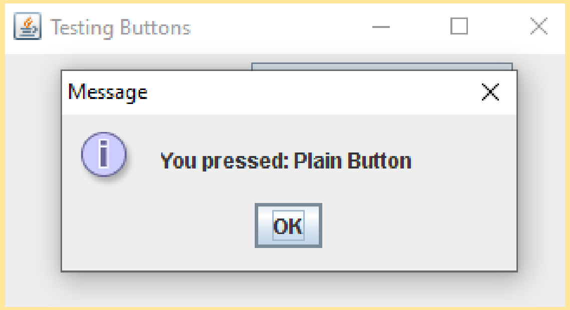
Register `JButton`s to receive events from `ButtonHandler`

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

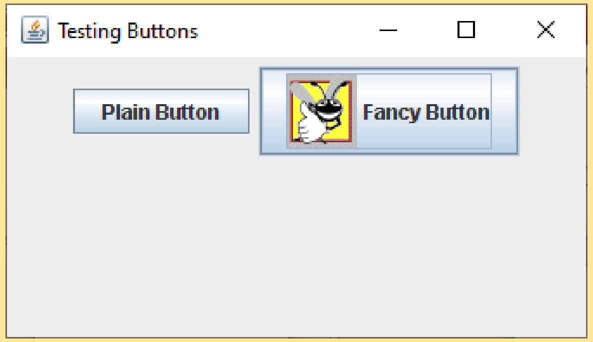
Swing button hierarchy

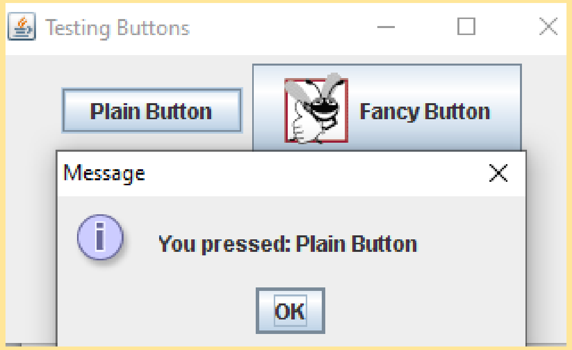
```
public static void main(String[] args)
 {ButtonFrame buttonFrame = new ButtonFrame();
 buttonFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 buttonFrame.setSize(350, 200);
 buttonFrame.setVisible(true);    }
 } // end class ButtonTest
```



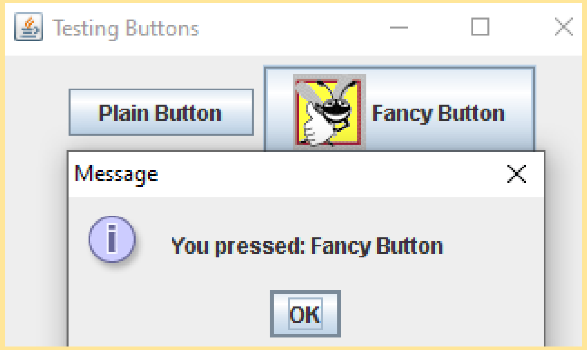


You pressed: Plain Button





You pressed: Plain Button



You pressed: Fancy Button

13.8 `JCheckBox` and `JRadioButton`

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

CheckBoxTest.java 1

```
1  // Fig. 13.11: CheckBoxTest.java
2  // Creating JCheckBox buttons.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class CheckBoxTest extends JFrame {
8     private JTextField field;
9     private JCheckBox bold, italic;
10
11    // set up GUI
12    public CheckBoxTest()
13    {
14       super( "JCheckBox Test" );
15
16       // get content pane and set its layout
17       Container container = getContentPane();
18       container.setLayout( new FlowLayout() );
19
20       // set up JTextField and set its font
21       field = new JTextField( "Watch the font style change", 20 );
22       field.setFont( new Font( "Serif", Font.PLAIN, 14 ) );
23       container.add( field );
24
```

CheckBoxTest.java

Line 9

Line 22

Declare two `JCheckBox` instances

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

CheckBoxTest.java 2

```
25        // create checkbox objects
26        bold = new JCheckBox( "Bold" );
27        container.add( bold );
28
29        italic = new JCheckBox( "Italic" );
30        container.add( italic );
31
32        // register listeners for JCheckBoxes
33        CheckBoxHandler handler = new CheckBoxHandler();
34        bold.addItemListener( handler );
35        italic.addItemListener( handler );
36
37        setSize( 275, 100 );
38        setVisible( true );
39
40     } // end CheckBoxText constructor
41
42     public static void main( String args[] )
43     {
44        CheckBoxTest application = new CheckBoxTest();
45        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
46     }
47
```

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

va

Lines 26 and 29

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

CheckBoxTest.java 3

```java
48    // private inner class for ItemListener event handling
49    private class CheckBoxHandler implements ItemListener {
50       private int valBold = Font.PLAIN;
51       private int valItalic = Font.PLAIN;
52
53       // respond to checkbox events
54       public void itemStateChanged( ItemEvent event )
55       {
56          // process bold checkbox events
57          if ( event.getSource() == bold )
58             valBold = bold.isSelected() ? Font.BOLD : Font.PLAIN;
59
60          // process italic checkbox events
61          if ( event.getSource() == italic )
62             valItalic = italic.isSelected() ? Font.ITALIC
63
64          // set text field font
65          field.setFont( new Font( "Serif", valBold + valItalic, 14 ) );
66
67       } // end method itemStateChanged
68
69    } // end private inner class CheckBoxHandler
70
71 } // end class CheckBoxTest
```

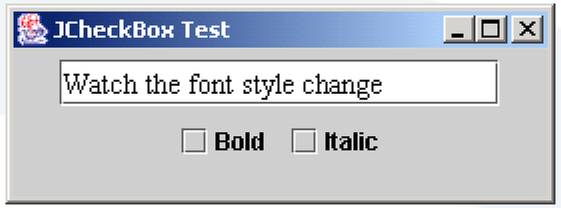
When user selects `JCheckBox`, `CheckBoxHandler` invokes method `itemStateChanges` of all registered listeners

Line 65

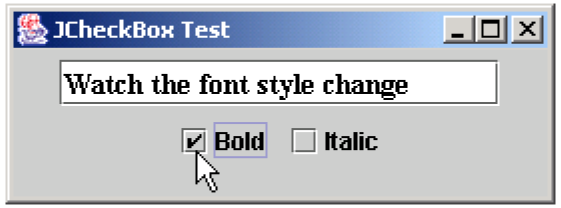
Change `JTextField` font, depending on which `JCheckBox` was selected

CheckBoxTest

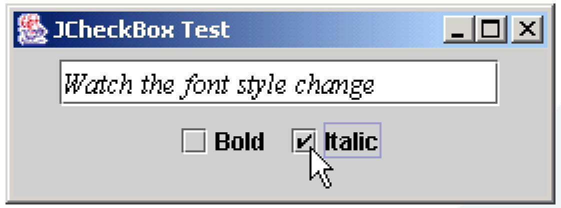
CheckBoxTest.ja
va



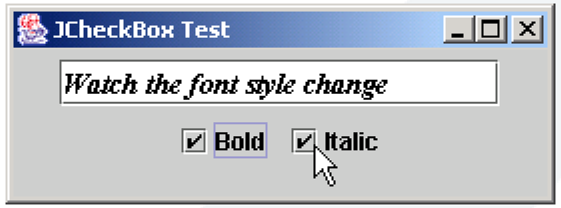
Watch the font style change



Watch the font style change



Watch the font style change



Watch the font style change

RadioButtonTest 1

```java
1   // Fig. 13.12: RadioButtonTest.java
2   // Creating radio buttons using ButtonGroup and JRadioButton.
3   import java.awt.*;
4   import java.awt.event.*;
5   import javax.swing.*;
6
7   public class RadioButtonTest extends JFrame {
8      private JTextField field;
9      private Font plainFont, boldFont, italicFont, boldItalicFont;
10     private JRadioButton plainButton, boldButton, italicButton,
11        boldItalicButton;
12     private ButtonGroup radioGroup;
13
14     // create GUI and fonts
15     public RadioButtonTest()
16     {
17        super( "RadioButton Test" );
18
19        // get content pane and set its layout
20        Container container = getContentPane();
21        container.setLayout( new FlowLayout() );
22
23        // set up JTextField
24        field = new JTextField( "Watch the font style change", 25 );
25        container.add( field );
26
```

RadioButtonTest
.java

Declare four `JRadioButton` instances

Lines 10-11

Line 12

`JRadioButton`s normally appear as a `ButtonGroup`

RadioButtonTest 2

```java
27        // create radio buttons
28        plainButton = new JRadioButton( "Plain", true );
29        container.add( plainButton );
30
31        boldButton = new JRadioButton( "Bold", false );
32        container.add( boldButton );
33
34        italicButton = new JRadioButton( "Italic", false );
35        container.add( italicButton );
36
37        boldItalicButton = new JRadioButton( "Bold/Italic", false );
38        container.add( boldItalicButton );
39
40        // create logical relationship between JRadioButtons
41        radioGroup = new ButtonGroup();
42        radioGroup.add( plainButton );
43        radioGroup.add( boldButton );
44        radioGroup.add( italicButton );
45        radioGroup.add( boldItalicButton );
46
47        // create font objects
48        plainFont = new Font( "Serif", Font.PLAIN, 14 );
49        boldFont = new Font( "Serif", Font.BOLD, 14 );
50        italicFont = new Font( "Serif", Font.ITALIC, 14 );
51        boldItalicFont = new Font( "Serif", Font.BOLD + Font.ITALIC, 14 );
52        field.setFont( plainFont );  // set initial font
```

RadioButtonTest.java

Instantiate JRadioButtons for manipulating JTextField text font

Lines 41-45

JRadioButtons belong to ButtonGroup

RadioButtonTest 3

```
54        // register events for JRadioButtons
55        plainButton.addItemListener( new RadioButtonHandler( plainFont )
);
56        boldButton.addItemListener( new RadioButtonHandler( boldFont ) )
57        italicButton.addItemListener(
58           new RadioButtonHandler( italicFont ) );
59        boldItalicButton.addItemListener(
60           new RadioButtonHandler( boldItalicFont ) );
61
62        setSize( 300, 100 ); setVisible( true );
64
65     } // end RadioButtonTest constructor
66
67     public static void main( String args[] )
68     {
69        RadioButtonTest application = new RadioButtonTest();
70        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
71     }
72
73     // private inner class to handle radio button events
74     private class RadioButtonHandler implements ItemListener {
75        private Font font;
76
77        public RadioButtonHandler( Font f )
78        {           font = f;
80        }
```

Register `JRadioButton`s to receive events from `RadioButtonHandler`

t.java

Lines 55-60

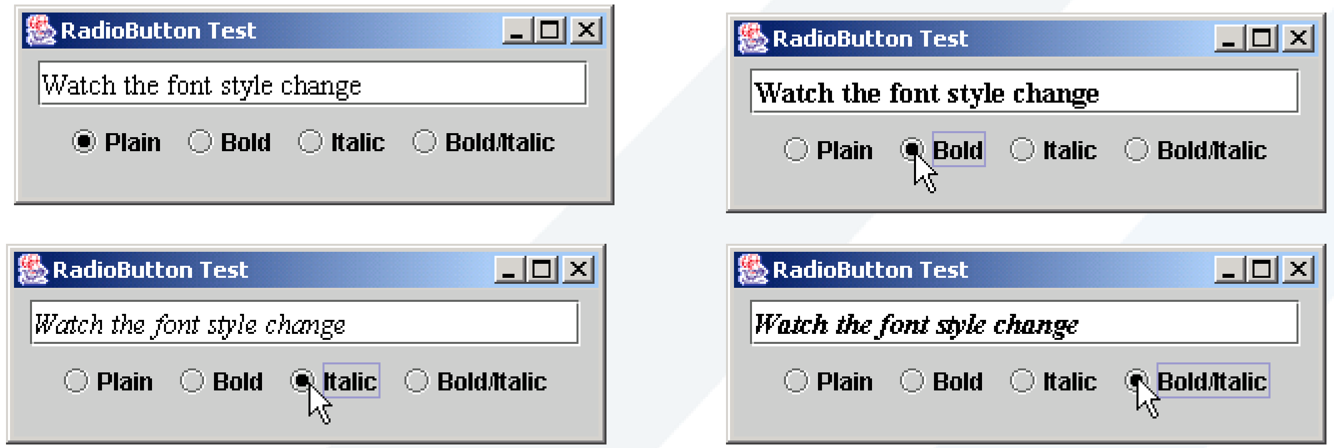
RadioButtonTest 4

```
81
82        // handle radio button events
83        public void itemStateChanged( ItemEvent event )
84        {
85           field.setFont( font );
86        }
87
88     } // end private inner class RadioButtonHandler
89
90  } // end class RadioButtonTest
```

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

Line 83

Set font corresponding to `JRadioButton` selected



RadioButton Test

Watch the font style change

[● Plain] ○ Bold ○ Italic ○ Bold/Italic

RadioButton Test

Watch the font style change

○ Plain [● Bold] ○ Italic ○ Bold/Italic

RadioButton Test

Watch the font style change

○ Plain ○ Bold [● Italic] ○ Bold/Italic

RadioButton Test

Watch the font style change

○ Plain ○ Bold ○ Italic [● Bold/Italic]

ComboBoxTest 1

```java
// Fig. 13.13: ComboBoxTest.java
// Using a JComboBox to select an image to display.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ComboBoxTest extends JFrame {
   private JComboBox imagesComboBox;
   private JLabel label;

   private String names[] =
      { "bug1.gif", "bug2.gif",  "travelbug.gif", "buganim.gif" };
   private Icon icons[] = { new ImageIcon( names[ 0 ] ),
      new ImageIcon( names[ 1 ] ), new ImageIcon( names[ 2 ] ),
      new ImageIcon( names[ 3 ] ) };

   // set up GUI
   public ComboBoxTest()
   {
      super( "Testing JComboBox" );

      // get content pane and set its layout
      Container container = getContentPane();
      container.setLayout( new FlowLayout() );

```

ComboBoxTest
.java

JComboBox

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

ComboBoxTest 2

```
26        // set up JComboBox and register its event handler
27        imagesComboBox = new JComboBox( names );
28        imagesComboBox.setMaximumRowCount( 3 );
29        imagesComboBox.addItemListener(
30
31           new ItemListener() {  // anonymous inner class
32
33              // handle JComboBox event
34              public void itemStateChanged( ItemEvent event )
35              {
36                 // determine whether check box selected
37                 if ( event.getStateChange() == ItemEvent.SELECTED )
38                    label.setIcon( icons[
39                       imagesComboBox.getSelectedIndex() ] );
40              }
41
42           }  // end anonymous inner class
43
44        ); // end call to addItemListener
45
46        container.add( imagesComboBox );
47
48        // set up JLabel to display ImageIcons
49        label = new JLabel( icons[ 0 ] );
50        container.add( label );
51
```

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

ava

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

Line 29

Line 34

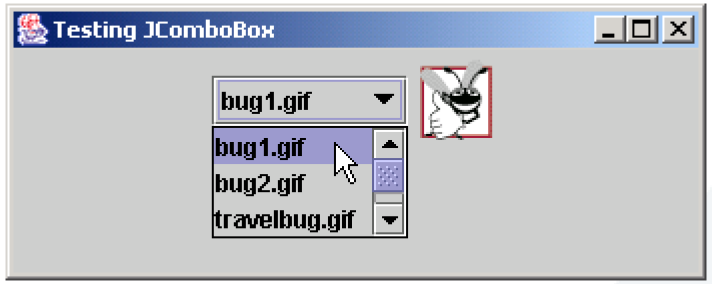
When user selects item in `JComboBox`, `ItemListener` invokes method `itemStateChanged` of all registered listeners

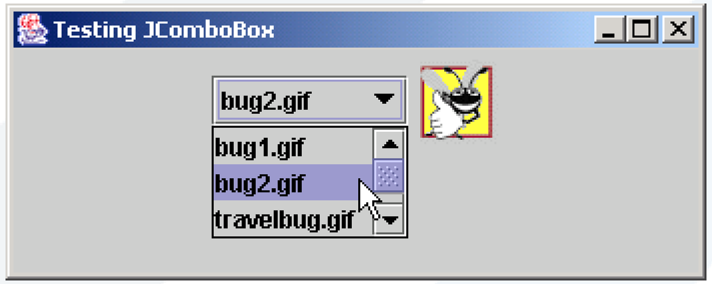
Set appropriate `Icon` depending on user selection

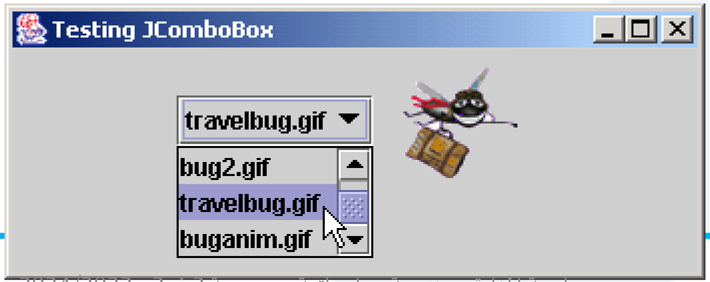
ComboBoxTest 3

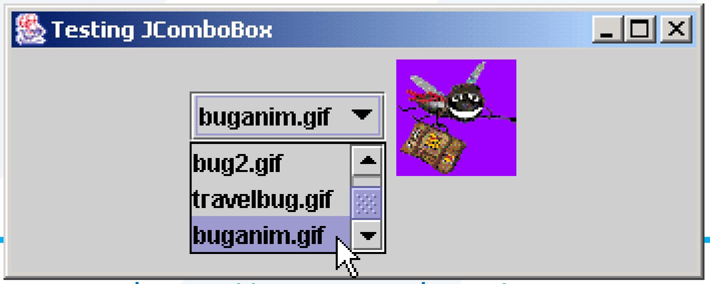
```
        setSize( 350, 100 );
        setVisible( true );

    } // end ComboBoxTest constructor

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

} // end class ComboBoxTest
```

ComboBoxTest.java









CheckBoxFrame 1

```
//Fig. 12.17: CheckBoxFrame.java  JCheckBoxes and item events.
 import java.awt.FlowLayout;
 import java.awt.Font;
 import java.awt.event.ItemListener;
 import java.awt.event.ItemEvent;
 import javax.swing.JFrame;
 import javax.swing.JTextField;
 import javax.swing.JCheckBox;
 public class CheckBoxFrame extends JFrame  {
private final JTextField textField; // displays text in changing fonts
private final JCheckBox boldJCheckBox; // to select/deselect bold
private final JCheckBox italicJCheckBox; // to select/deselect italic
  // CheckBoxFrame constructor adds JCheckBoxes to JFrame
 public CheckBoxFrame() { super("JCheckBox Test");
 setLayout(new FlowLayout());  // set up JTextField and set its font
 textField = new JTextField("Watch the font style change", 20);
 textField.setFont(new Font("Serif", Font.PLAIN, 14));
 add(textField); // add textField to JFrame
```

CheckBoxFrame 2

```
 boldJCheckBox = new JCheckBox("Bold");
 italicJCheckBox = new JCheckBox("Italic");
 add(boldJCheckBox); // add bold checkbox to JFrame
 add(italicJCheckBox); // add italic checkbox to JFrame
// register listeners for JCheckBoxes
CheckBoxHandler handler = new CheckBoxHandler();
boldJCheckBox.addItemListener(handler);
italicJCheckBox.addItemListener(handler);  }
 // private inner class for ItemListener event handling
 private class CheckBoxHandler implements ItemListener
 {// respond to checkbox events
 @Override
 public void itemStateChanged(ItemEvent event)
 { Font font = null; // stores the new Font
 // determine which CheckBoxes are checked and create Font
 if (boldJCheckBox.isSelected() && italicJCheckBox.isSelected() )
 font = new Font("Serif", Font.BOLD + Font.ITALIC, 14);
 else if (boldJCheckBox.isSelected()) font = new Font("Serif", Font.BOLD, 14);
```

CheckBoxFrame 3

```
else if (italicJCheckBox.isSelected())
 font = new Font("Serif", Font.ITALIC, 14);
 else    font = new Font("Serif", Font.PLAIN, 14);
 textField.setFont(font);
       }      }
 } // end class CheckBoxFrame

// Fig. 12.18: CheckBoxTest.java Testing CheckBoxFrame.
 import javax.swing.JFrame;
 public class CheckBoxTest {
 public static void main(String[] args)
 {  CheckBoxFrame checkBoxFrame = new CheckBoxFrame();
 checkBoxFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 checkBoxFrame.setSize(275, 100);
 checkBoxFrame.setVisible(true);
    }
 } // end class CheckBoxTest
```

RadioButtonFrame 1

```
//Fig. 12.19: RadioButtonFrame.java
 // Creating radio buttons using ButtonGroup and JRadioButton.
 import java.awt.FlowLayout;
 import java.awt.Font;
 import java.awt.event.ItemListener;
 import java.awt.event.ItemEvent;
 import javax.swing.JFrame;
 import javax.swing.JTextField;
 import javax.swing.JRadioButton;
 import javax.swing.ButtonGroup;
 public class RadioButtonFrame extends JFrame
 { private final JTextField textField; // used to display font changes
 private final Font plainFont; // font for plain text
 private final Font boldFont; // font for bold text
 private final Font italicFont; // font for italic text
 private final Font boldItalicFont; // font for bold and italic text
 private final JRadioButton plainJRadioButton; // selects plain text
 private final JRadioButton boldJRadioButton; // selects bold text
```

RadioButtonFrame 2

```
private final JRadioButton italicJRadioButton; // selects italic text
 private final JRadioButton boldItalicJRadioButton; // bold and italic
 private final ButtonGroup radioGroup; // holds radio buttons
 // RadioButtonFrame constructor adds JRadioButtons to JFrame
 public RadioButtonFrame() {
 super("RadioButton Test");
 setLayout(new FlowLayout());
 textField = new JTextField("Watch the font style change", 25);
 add(textField); // add textField to JFrame
 // create radio buttons
plainJRadioButton = new JRadioButton("Plain", true);
boldJRadioButton = new JRadioButton("Bold", false);
italicJRadioButton = new JRadioButton("Italic", false);
boldItalicJRadioButton = new JRadioButton("Bold/Italic", false);
 add(plainJRadioButton); // add plain button to JFrame
 add(boldJRadioButton); // add bold button to JFrame
 add(italicJRadioButton); // add italic button to JFrame
 add(boldItalicJRadioButton); // add bold and italic button
```

RadioButtonFrame 3

```
//create logical relationship between JRadioButtons
radioGroup = new ButtonGroup(); // create ButtonGroup
radioGroup.add(plainJRadioButton); // add plain to group
radioGroup.add(boldJRadioButton); // add bold to group
radioGroup.add(italicJRadioButton); // add italic to group
radioGroup.add(boldItalicJRadioButton); // add bold and italic
 // create font objects
 plainFont = new Font("Serif", Font.PLAIN, 14);
 boldFont = new Font("Serif", Font.BOLD, 14);
 italicFont = new Font("Serif", Font.ITALIC, 14);
 boldItalicFont = new Font("Serif", Font.BOLD + Font.ITALIC, 14);
 textField.setFont(plainFont);
//register events for JRadioButtons
plainJRadioButton.addItemListener(new RadioButtonHandler(plainFont));
boldJRadioButton.addItemListener( new RadioButtonHandler(boldFont));
italicJRadioButton.addItemListener(new RadioButtonHandler(italicFont));
boldItalicJRadioButton.addItemListener(new RadioButtonHandler(boldItalicFont));
  }
```

RadioButtonFrame 4

```
// private inner class to handle radio button events
private class RadioButtonHandler implements ItemListener
{  private Font font; // font associated with this listener
public RadioButtonHandler(Font f) { font = f;   }
 // handle radio button events
@Override
public void itemStateChanged(ItemEvent event) { textField.setFont(font);   }
}    } // end class RadioButtonFrame

// Fig. 12.20: RadioButtonTest.java  Testing RadioButtonFrame.
import javax.swing.JFrame;
public class RadioButtonTest
{ public static void main(String[] args)
{ RadioButtonFrame radioButtonFrame = new RadioButtonFrame();
radioButtonFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
radioButtonFrame.setSize(300, 100);
radioButtonFrame.setVisible(true);   }
} // end class RadioButtonTest
```

انتهت محاضرة الأسبوع السابع والثامن