L4j1 - Event Handling in Java

Outline

12.1 Introduction
12.3 `JLabel`
12.4 Event Handling Model
12.6 `JButton`
12.5.1 How Event Handling Works

Event-based Programming

- Main assumption: A widget can "get" an event and call the corresponding processing program
- Main Implementation issues:
- How to pass an event to the widget?
- How to specify the event-processing program that will be called?
- How the event processing program can get details about the event that called it?

3

Event-based Programming: Implementation

- A widget is usually an object of an appropriate class. It inherit all properties of the class
- We can set widget parameters (fields of an object) to adjust it to our need
- Some parameters (fields) may specify programs to process widget events:

  mybutton.setIfPushed (*Function to be called*);

  – That can work in languages like C++ or Lisp where functions can be referred. In Java it is not possible

4

12.4 Java Event Handling Model

- GUIs are event driven
  - Generate events when user interacts with GUI
    - Mouse movements, mouse clicks, typing in a text field, etc.
  - Event information stored in object that extends **`AWTEvent`**
- To process an event
  - Register an event listener
    - Object from a class that implements an event-listener interface (from **`java.awt.event`** or **`javax.swing.event`**)
    - "Listens" for events
  - Implement event handler
    - Method that is called in response to an event
    - Event handling interface has one or more methods that must be defined

12.4 Event Handling Model

- Delegation event model
  - Use of event listeners in event handling
  - Processing of event delegated to particular object
- When an event occurs
  - GUI component notifies its listeners
    - Calls listener's event handling method
- Example:
  - *Enter* pressed in a **`JTextField`**
  - Method **`actionPerformed`** called for registered listener
  - Details in following sections

12.6 `JButton`

- Button
  - Component user clicks to trigger an action
  - Several types of buttons
    - Command buttons, toggle buttons, check boxes, radio buttons
- Command button
  - Generates **`ActionEvent`** when clicked
  - Created with class **`JButton`**
    - Inherits from class **`AbstractButton`**
    - Defines many features of Swing buttons
- **`JButton`**
  - Text on face called button label
  - Each button should have a different label
  - Support display of **`Icon`**s

12.6 JButton

- Methods of class **JButton**
  - Constructors

```
JButton myButton = new JButton( "Label" );
JButton myButton = new JButton( "Label",
  myIcon );
```

  - **setRolloverIcon(myIcon)**
    - Sets image to display when mouse over button
- Class ActionEvent
  - **getActionCommand**
    - Returns label of button that generated event

Outline

1. `import`

1.1 Declarations

2. Initialize buttons and `Icons`

2.2 Register event handler

```
1  // Fig. 12.11: ButtonTest.java
2  // Creating JButtons.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class ButtonTest extends JFrame {
8     private JButton plainButton, fancyButton;
9
10    public ButtonTest()
11    {
12       super( "Testing Buttons" );
13
14       Container c = getContentPane();
15       c.setLayout( new FlowLayout() );
16
17       // create buttons
18       plainButton = new JButton( "Plain Button" );
19       c.add( plainButton );
20
21       Icon bug1 = new ImageIcon( "bug1.gif" );
22       Icon bug2 = new ImageIcon( "bug2.gif" );
23       fancyButton = new JButton( "Fancy Button", bug1 );
24       fancyButton.setRolloverIcon( bug2 );
25       c.add( fancyButton );
26
27       // create an instance of inner class Butto
28       // to use for button event handling
29       ButtonHandler handler = new ButtonHandler();
30       fancyButton.addActionListener( handler );
```

Create **JButtons**. Initialize **fancyButton** with an **ImageIcon**.

erIcon

Set a different icon to appear when the mouse is over the **JButton**.

Outline

3. `main`

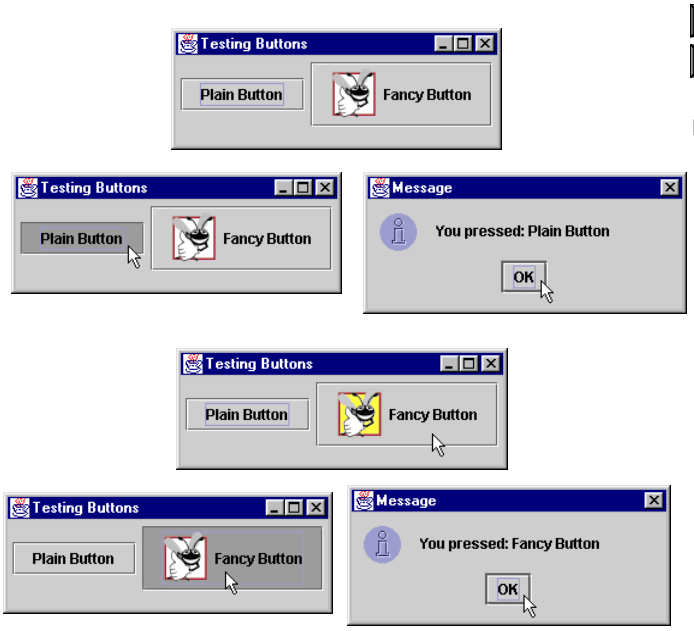
4. Inner class event handler

```
31       plainButton.addActionListener( handler );
32
33       setSize( 275, 100 );
34       show();
35    }
36
37    public static void main( String args[] )
38    {
39       ButtonTest app = new ButtonTest();
40
41       app.addWindowListener(
42          new WindowAdapter() {
43             public void windowClosing( WindowEvent e )
44             {
45                System.exit( 0 );
46             }
47          }
48       );
49    }
50
51    // inner class for button event handling
52    private class ButtonHandler implements ActionListener {
53       public void actionPerformed( ActionEvent e )
54       {
55          JOptionPane.showMessageDialog( null,
56             "You pressed: " + e.getActionCommand() );
57       }
58    }
59 }
```

`getActionCommand` returns label of button that generated event.

Outline

Program Output



12.5.1 How Event Handling Works

- Registering event listeners
  - All **`JComponents`** contain an object of class **`EventListenerList`** called **`listenerList`**
  - When **`text1.addActionListener( handler )`** executes
    - New entry placed into **`listenerList`**
- Handling events
  - When event occurs, has an event ID
    - Component uses this to decide which method to call
    - If **`ActionEvent`**, then **`actionPerformed`** called (in all registered **`ActionListener`**s)