Module 11

GUI Event Handling

Objectives

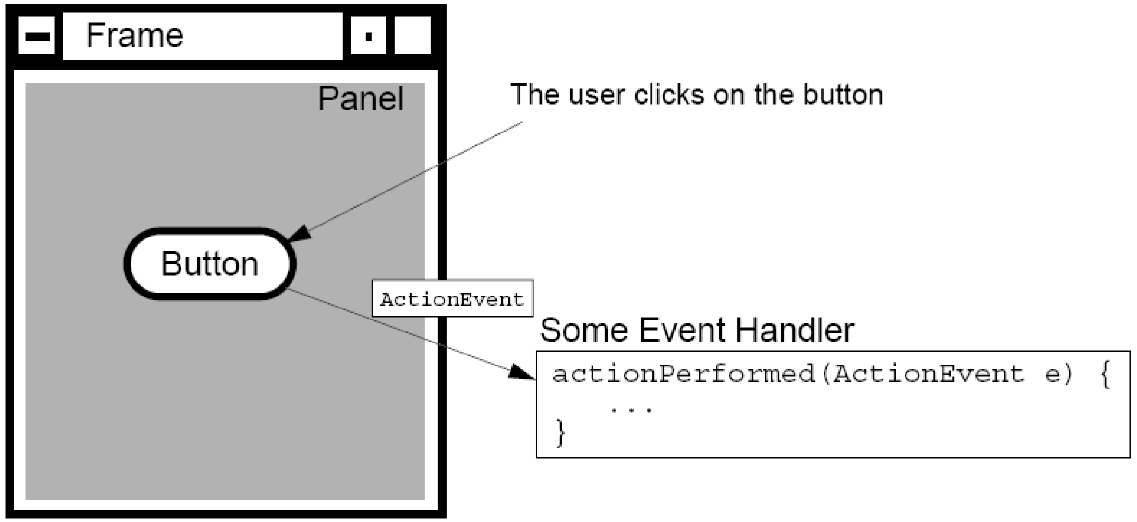
- Define events and event handling
- Write code to handle events that occur in a GUI
- Describe the concept of adapter classes, including how and when to use them
- Determine the user action that originated the event from the event object details
- Identify the appropriate listener interface for a variety of event types
- Create the appropriate event handler methods for a variety of event types
- Understand the use of inner classes and anonymous classes in event handling

Relevance

- What parts of a GUI are required to make it useful?
- How does a graphical program handle a mouse click or any other type of user interaction?

What Is an Event?

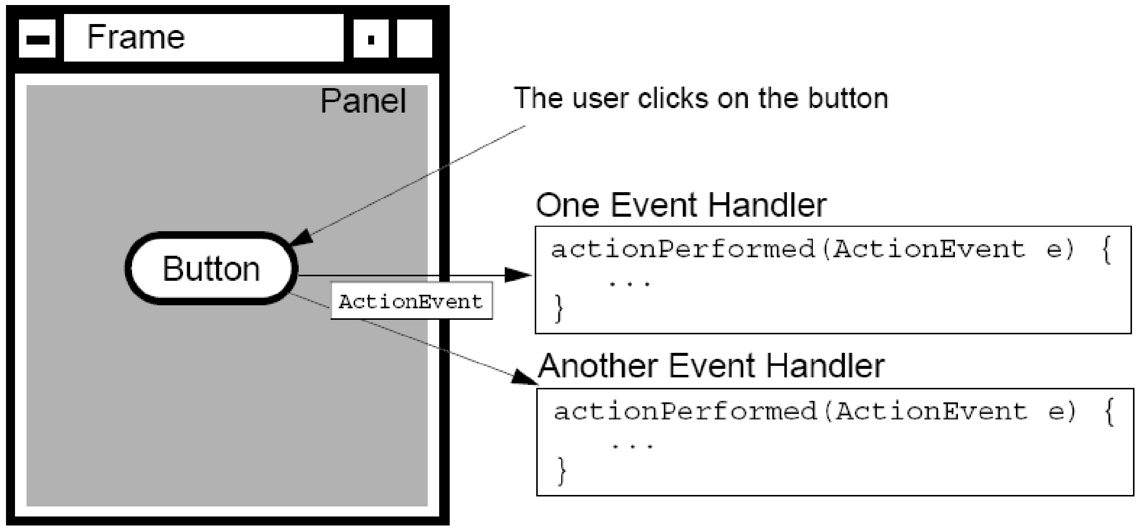
- Events – Objects that describe what happened
- Event sources – The generator of an event
- Event handlers – A method that receives an event object, deciphers it, and processes the user's interaction



```
actionPerformed(ActionEvent e) {
    ...
}
```

Delegation Model

- An event can be sent to many event handlers.



```
actionPerformed(ActionEvent e) {
    ...
}
```

```
actionPerformed(ActionEvent e) {
    ...
}
```

- Event handlers register with components when they are interested in events generated by that component.

Delegation Model

- Client objects (handlers) register with a GUI component that they want to observe.
- GUI components only trigger the handlers for the type of event that has occurred.
- Most components can trigger more than one type of event.
- The delegation model distributes the work among multiple classes.

A Listener Example

```
import java.awt.*;

public class TestButton {
  private Frame f;
  private Button b;

  public TestButton() {
    f = new Frame("Test");
    b = new Button("Press Me!");
    b.setActionCommand("ButtonPressed");
  }

  public void launchFrame() {
    b.addActionListener(new ButtonHandler());
    f.add(b,BorderLayout.CENTER);
    f.pack();
    f.setVisible(true);
  }
```

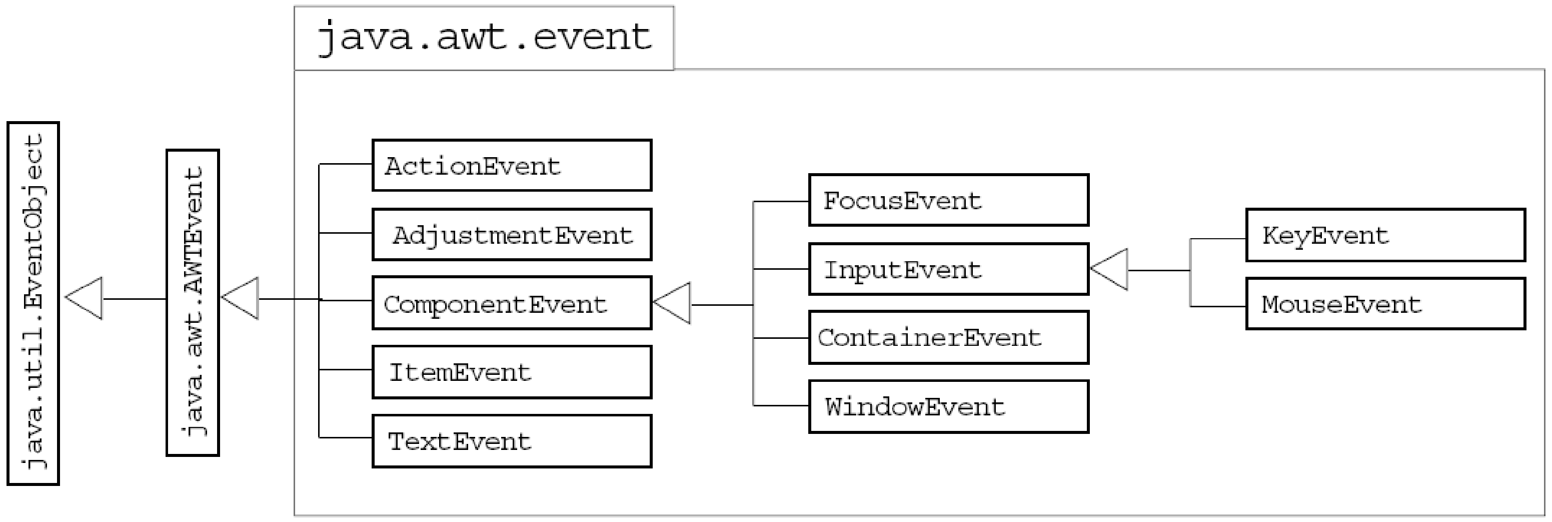
A Listener Example

```

  public static void main(String args[]) {
    TestButton guiApp = new TestButton();
    guiApp.launchFrame();
  }
}
```

Code for the event listener looks like this:

```
import java.awt.event.*;

public class ButtonHandler implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    System.out.println("Action occurred");
    System.out.println("Button’s command is: "
                       + e.getActionCommand());
  }
}
```

Event Categories



Method Categories and Interfaces

Category	Interface Name	Methods
Action	`ActionListener`	`actionPerformed(ActionEvent)`
Item	`ItemListener`	`itemStateChanged(ItemEvent)`

Mouse	`MouseListener`	`mousePressed(MouseEvent)`
		`mouseReleased(MouseEvent)`
		`mouseEntered(MouseEvent)`
		`mouseExited(MouseEvent)`
		`mouseClicked(MouseEvent)`

Mouse motion	`MouseMotionListener`	`mouseDragged(MouseEvent)`
		`mouseMoved(MouseEvent)`

Key	`KeyListener`	`keyPressed(KeyEvent)`
		`keyReleased(KeyEvent)`
		`keyTyped(KeyEvent)`

Method Categories and Interfaces

Category	Interface Name	Methods

Focus	`FocusListener`	`focusGained(FocusEvent)`
		`focusLost(FocusEvent)`

Adjustment	`AdjustmentListener`	`adjustmentValueChanged`
		`(AdjustmentEvent)`

Component	`ComponentListener`	`componentMoved(ComponentEvent)`
		`componentHidden(ComponentEvent)`
		`componentResized(ComponentEvent)`
		`componentShown(ComponentEvent)`

Method Categories and Interfaces

Category	Interface Name	Methods

Window	`WindowListener`	`windowClosing(WindowEvent)`
		`windowOpened(WindowEvent)`
		`windowIconified(WindowEvent)`
		`windowDeiconified(WindowEvent)`
		`windowClosed(WindowEvent)`
		`windowActivated(WindowEvent)`
		`windowDeactivated(WindowEvent)`

Container	`ContainerListener`	`componentAdded(ContainerEvent)`
		`componentRemoved(ContainerEvent)`

Text	`TextListener`	`textValueChanged(TextEvent)`

Complex Example

```
import java.awt.*;
import java.awt.event.*;

public class TwoListener
       implements MouseMotionListener, MouseListener {
  private Frame f;
  private TextField tf;

  public TwoListener() {
    f = new Frame("Two listeners example");
    tf = new TextField(30);
  }
```

Complex Example

```java

  public void launchFrame() {
    Label label = new Label("Click and drag the mouse");
    // Add components to the frame
    f.add(label, BorderLayout.NORTH);
    f.add(tf, BorderLayout.SOUTH);
    // Add this object as a listener
    f.addMouseMotionListener(this);
    f.addMouseListener(this);
    // Size the frame and make it visible
    f.setSize(300, 200);
    f.setVisible(true);
  }
```

Complex Example

```java

    // These are MouseMotionListener events
    public void mouseDragged(MouseEvent e) {
      String s =  "Mouse dragging:  X = " + e.getX()
                  + " Y = " + e.getY();
      tf.setText(s);
    }

    public void mouseEntered(MouseEvent e) {
      String s = "The mouse entered";
      tf.setText(s);
    }

    public void mouseExited(MouseEvent e) {
      String s = "The mouse has left the building";
      tf.setText(s);
    }
```

Complex Example

```java

  // Unused MouseMotionListener method.
  // All methods of a listener must be present in the
  // class even if they are not used.
  public void mouseMoved(MouseEvent e) { }

  // Unused MouseListener methods.
  public void mousePressed(MouseEvent e) { }
  public void mouseClicked(MouseEvent e) { }
  public void mouseReleased(MouseEvent e) { }

  public static void main(String args[]) {
    TwoListener two = new TwoListener();
    two.launchFrame();
  }
}
```

Multiple Listeners

- Multiple listeners cause unrelated parts of a program to react to the same event.
- The handlers of all registered listeners are called when the event occurs.

Event Adapters

- The listener classes that you define can extend adapter classes and override only the methods that you need.
- An example is:

```java
import java.awt.*;
import java.awt.event.*;

public class MouseClickHandler extends MouseAdapter {

  // We just need the mouseClick handler, so we use
  // an adapter to avoid having to write all the
  // event handler methods

  public void mouseClicked(MouseEvent e) {
    // Do stuff with the mouse click...
  }
}
```

Event Handling Using Inner Classes

```
import java.awt.*;
import java.awt.event.*;
public class TestInner {
  private Frame f;
  private TextField tf; // used by inner class

  public TestInner() {
    f = new Frame("Inner classes example");
    tf = new TextField(30);
  }

  class MyMouseMotionListener extends MouseMotionAdapter {
      public void mouseDragged(MouseEvent e) {
        String s = "Mouse dragging:  X = "+ e.getX()
                    + " Y = " + e.getY();
        tf.setText(s);
      }
    }
```

Event Handling Using Inner Classes

```

  public void launchFrame() {
    Label label = new Label("Click and drag the mouse");
    // Add components to the frame
    f.add(label, BorderLayout.NORTH);
    f.add(tf, BorderLayout.SOUTH);
    // Add a listener that uses an Inner class
    f.addMouseMotionListener(new MyMouseMotionListener());
    f.addMouseListener(new MouseClickHandler());
    // Size the frame and make it visible
    f.setSize(300, 200);
    f.setVisible(true);
  }

  public static void main(String args[]) {
    TestInner obj = new TestInner();
    obj.launchFrame();
  }
}
```

Event Handling Using Anonymous Classes

```java
import java.awt.*;
import java.awt.event.*;

public class TestAnonymous {
  private Frame f;
  private TextField tf;

  public TestAnonymous() {
    f = new Frame("Anonymous classes example");
    tf = new TextField(30);
  }

  public static void main(String args[]) {
    TestAnonymous obj = new TestAnonymous();
    obj.launchFrame();
  }

```

Event Handling Using Anonymous Classes

```
  public void launchFrame() {
    Label label = new Label("Click and drag the mouse");
    // Add components to the frame
    f.add(label, BorderLayout.NORTH);
    f.add(tf, BorderLayout.SOUTH);
    // Add a listener that uses an anonymous class
    f.addMouseMotionListener(new MouseMotionAdapter() {
      public void mouseDragged(MouseEvent e) {
        String s = "Mouse dragging:  X = "+ e.getX()
                    + " Y = " + e.getY();
        tf.setText(s);
      }
    }); // <- note the closing parenthesis
    f.addMouseListener(new MouseClickHandler()); // Not shown
    // Size the frame and make it visible
    f.setSize(300, 200);
    f.setVisible(true);
  }
}
```