Java Programming Language SE – 6

Module 13: Handling GUI-Generated Events

Team Emertxe

ORACLE®

Certified Professional

Java SE 6 Programmer

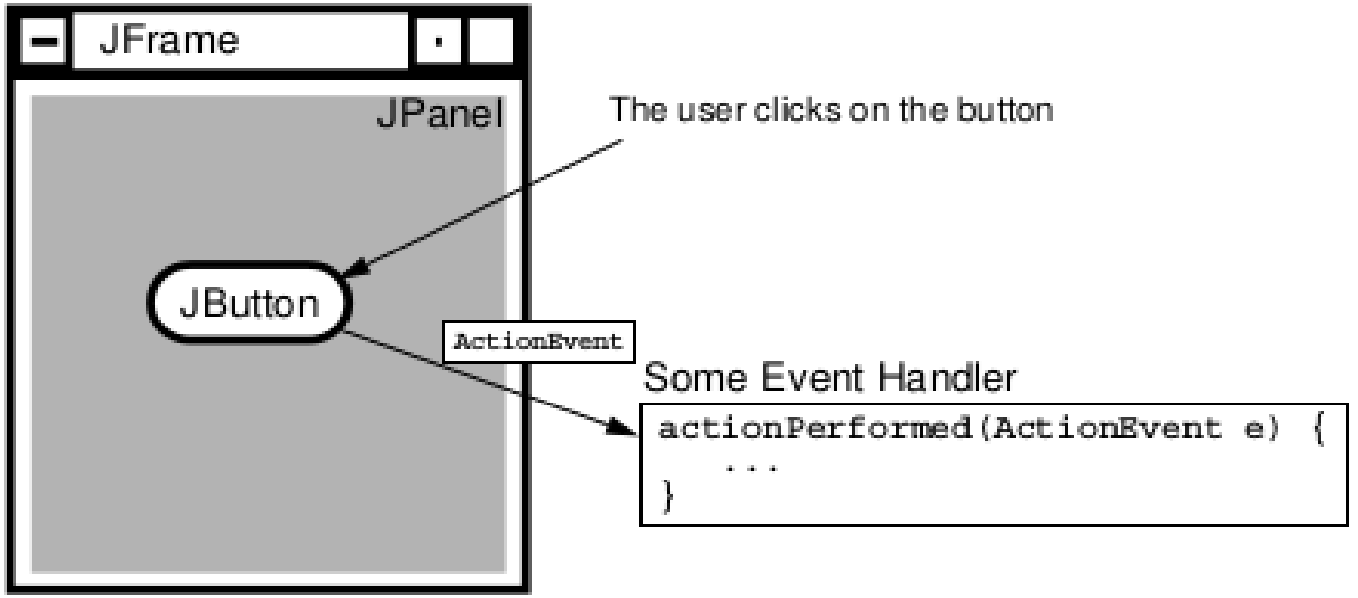
Objectives

- Define events and event handling
- Examine the Java SE event model
- Describe GUI behavior
- Determine the user action that originated an event
- Develop event listeners
- Describe concurrency in Swing-based GUIs and describe the features of the SwingWorker class

EMERTXE

What Is an Event?

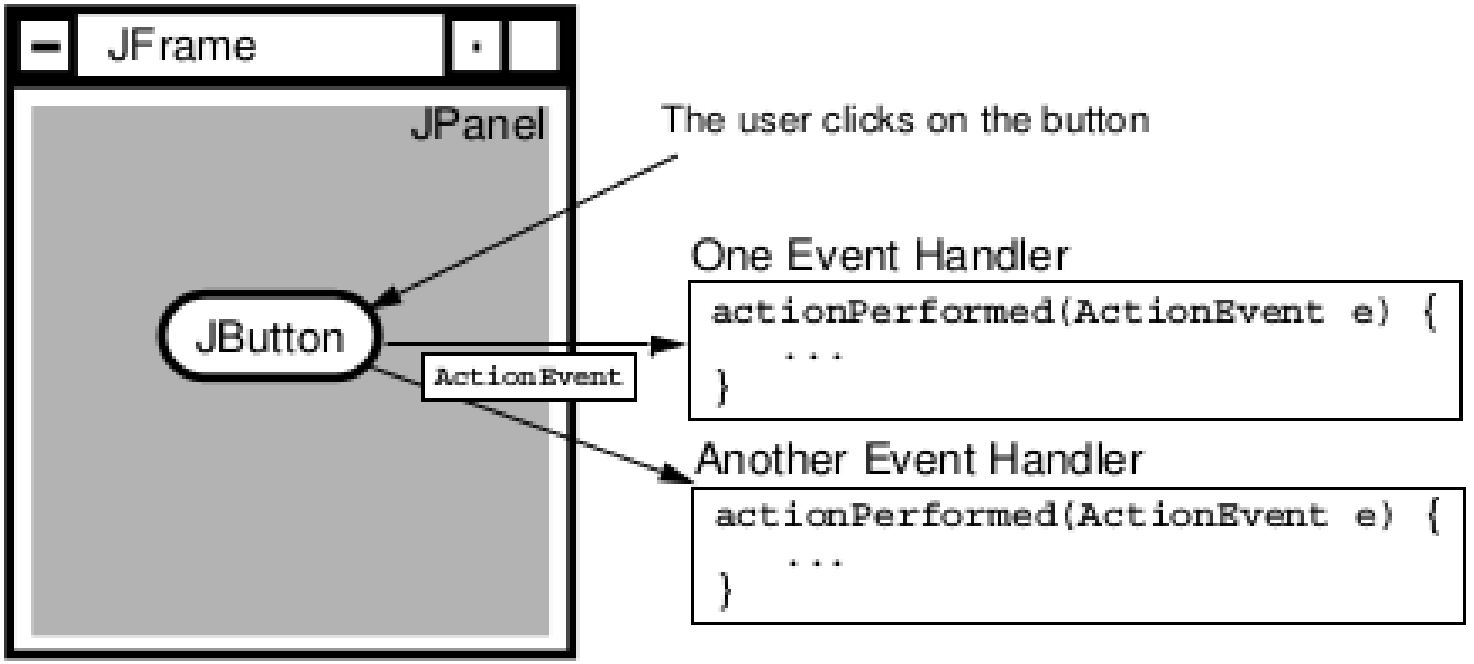
- Events – Objects that describe what happened
- Event sources – The generator of an event
- Event handlers – A method that receives an event object, deciphers it, and processes the user's interaction



EMERTXE

Delegation Model

- An event can be sent to many event handlers.
- 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 trigger only 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

```
public class TestButton {
private JFrame f;
private JButton b;
public TestButton() {
f = new JFrame("Test");
b = new JButton("Press Me!");
b.setActionCommand("ButtonPressed");
}
public void launchFrame() {
b.addActionListener(new ButtonHandler());
f.add(b,BorderLayout.CENTER);
f.pack();
f.setVisible(true);
}
```

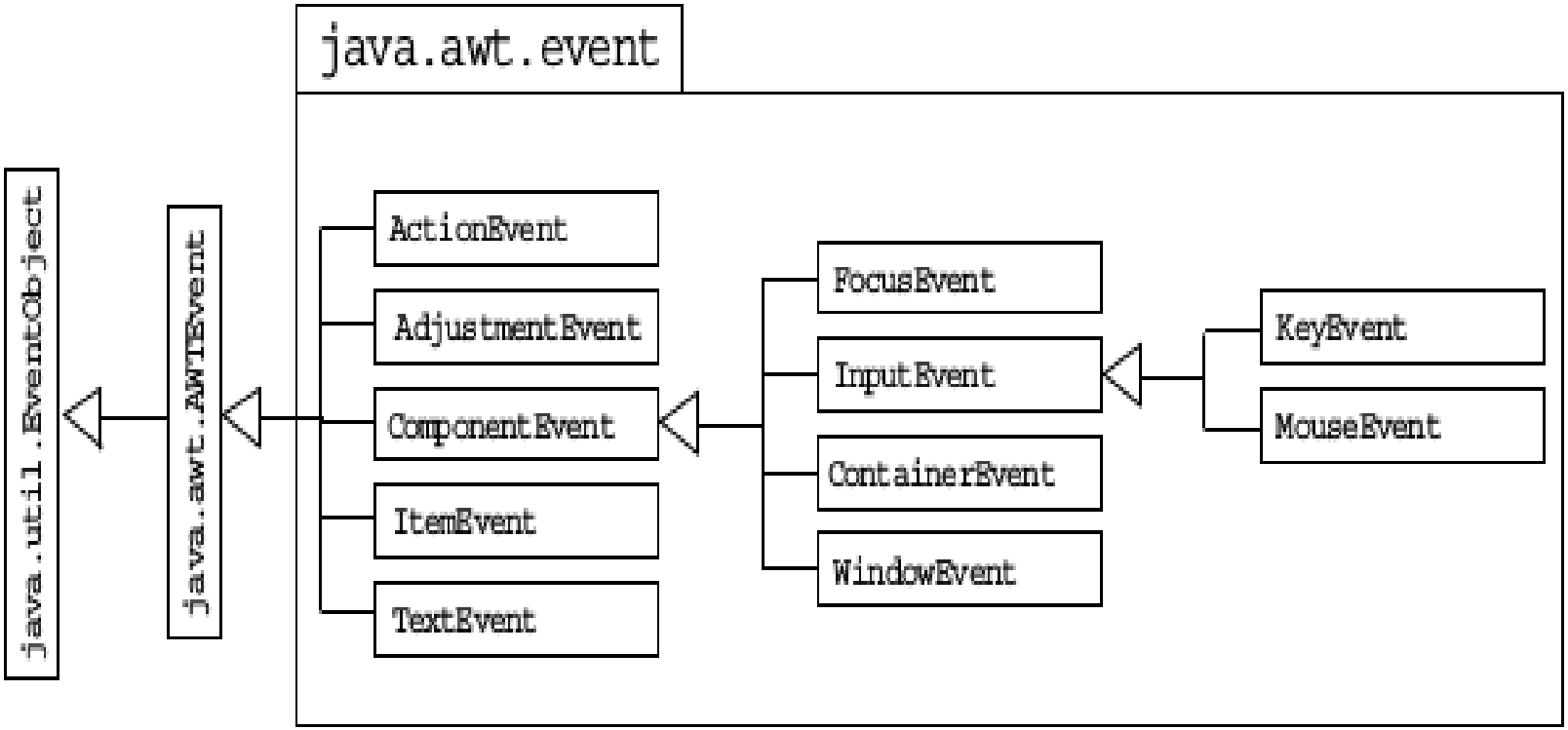
A Listener Example

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

Code for the event listener looks like the following:

```
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)`

EMERTXE

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)`
Window state	`WindowStateListener`	`windowStateChanged(WindowEvent e)`
Window focus	`WindowFocusListener`	`windowGainedFocus(WindowEvent e)` `windowLostFocus(WindowEvent e)`

Method Categories and Interfaces

Category	Interface Name	Methods
Mouse wheel	`MouseWheelListener`	`mouseWheelMoved (MouseWheelEvent e)`
Input methods	`InputMethodListener`	`caretPositionChanged (InputMethodEvent e)` `inputMethodTextChnaged (InputMethodEvent e)`
Hierarchy	`HierarchyListener`	`hierarchyChanged (HierarchyEvent e)`
Hierarchy bounds	`HierarchyBoundsListener`	`ancestorMoved(HierarchyEvent e)` `ancestorResized(HierarchyEvent e)`
AWT	`AWTEventListener`	`eventDispatched(AWTEvent e)`
Text	`TextListener`	`textValueChanged(TextEvent)`

Complex Example

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class TwoListener
implements MouseMotionListener, MouseListener {
private JFrame f;
private JTextField tf;
public TwoListener() {
f = new JFrame("Two listeners example");
tf = new JTextField(30);}
```

Complex Example

```
public void launchFrame() {
JLabel label = new JLabel("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

```
// 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

```
// 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.

Event Adapters Example

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
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.*;
import javax.swing.*;
public class TestInner {
private JFrame f;
private JTextField tf; // used by inner class
public TestInner() {
f = new JFrame("Inner classes example");
tf = new JTextField(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() {
JLabel label = new JLabel("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

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class TestAnonymous {
private JFrame f;
private JTextField tf;
public TestAnonymous() {
f = new JFrame("Anonymous classes example");
tf = new JTextField(30);
}
public static void main(String args[]) {
TestAnonymous obj = new TestAnonymous();
obj.launchFrame();
}
```

Event Handling Using Anonymous Classes

```
public void launchFrame() {
JLabel label = new JLabel("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);
}}
```

Concurrency In Swing

To handle a GUI efficiently, the Swing program needs different threads to:

- Execute the application code (current threads)
- Handle the events that arise from the GUI (event dispatch threads)
- Handle background tasks that might be time consuming (worker threads)

Each task in a worker thread is represented by an instance of javax.swing.SwingWorker.

The SwingWorker Class

The SwingWorker class has methods to service the following requirements:

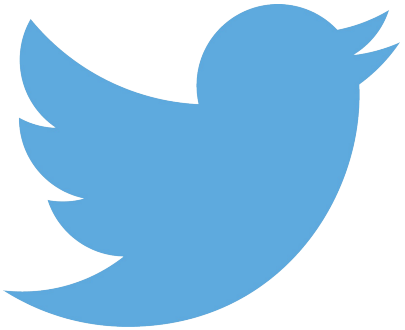
- To provide communication and coordination between worker thread tasks and the tasks on other threads:
  - Properties: state and progress
- To execute simple background tasks:
  - doInBackground method
- To execute tasks that have intermediate results:
  - publish method
- To cancel the background threads:
  - cancel method

Stay connected

About us: Emertxe is India's one of the top IT finishing schools & self learning kits provider. Our primary focus is on Embedded with diversification focus on Java, Oracle and Android areas

Emertxe Information Technologies,
No-1, 9th Cross, 5th Main,
Jayamahal Extension,
Bangalore, Karnataka 560046
T: +91 80 6562 9666
E: training@emertxe.com

https://www.facebook.com/Emertxe



https://twitter.com/EmertxeTweet

https://www.slideshare.net/EmertxeSlides

EMERTXE

Thank You