Objectives

► 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
► Create the appropriate interface and event handler methods for a variety of event types
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
Hierarchical Model (JDK1.0)

- Is based on containment

```java
action()
lostFocus()
mouseExit()
gotFocus()
mouseDown()
mouseMove()
keyDown()
mouseDrag()
mouseUp()
keyUp()
mouseEnter()
```

Frame

Panel

Button

Action event
Advantages

- It is simple and well suited to an object-oriented programming environment.

Disadvantages

- An event can only be handled by the component from which it originated or by one of the containers of the originating component.

- In order to handle events, you must either subclass the component that receives the event or create a handleEvent() method at the base container.
Delegation Model (JDK1.1)

► Events are sent to the component from which the event originated, but it is up to each component to propagate the event to one or more registered classes called listener. Listeners contain event handlers that receive and process the event. In this way, the event handler can be in an object separate from the component. Listeners are classes that implement the EventListener interface.

► Events are objects that are reported only to registered listeners. Every event has corresponding listener interface that mandates which methods must be defined in a class suited to receiving that type of event. The class that implements the interface defines those methods, and can be registered as a listener.

► Events from components that have no registered listeners are not propagated.
Delegation Model

► Client objects (handlers) register with a GUI component 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

► Distributes the work among multiple classes
Event Categories

Java.awt.event

- ActionEvent
- AdjustmentEvent
- ComponentEvent
- ItemEvent
- TextEvent
- KeyEvent
- MouseEvent
- FocusEvent
- InputEvent
- ContainerEvent
- WindowEvent

java.util.EventObject

java.awt.AWTEvent

GUI Event Handling

Java Programming
<table>
<thead>
<tr>
<th>Event Class</th>
<th>Listener Interface</th>
<th>Listener Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionEvent</td>
<td>ActionListener</td>
<td>actionPerformed()</td>
</tr>
<tr>
<td>AdjustmentEvent</td>
<td>AdjustmentListener</td>
<td>adjustmentValueChanged()</td>
</tr>
<tr>
<td>ComponentEvent</td>
<td>ComponentListener</td>
<td>componentHidden()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>componentMoved()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>componentResized()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>componentShown()</td>
</tr>
<tr>
<td>ContainerEvent</td>
<td>ContainerListener</td>
<td>componentAdded()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>componentRemoved()</td>
</tr>
<tr>
<td>FocusEvent</td>
<td>FocusListener</td>
<td>focusGained()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>focusLost()</td>
</tr>
<tr>
<td>ItemEvent</td>
<td>ItemListener</td>
<td>itemStateChanged()</td>
</tr>
<tr>
<td>KeyEvent</td>
<td>KeyListener</td>
<td>keyPressed()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>keyReleased()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>keyTyped()</td>
</tr>
<tr>
<td>Event Type</td>
<td>Event Listener</td>
<td>Methods</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>MouseEvent</td>
<td>MouseListener</td>
<td>mouseClicked(), mouseEntered(), mouseExited(), mousePressed(), mouseReleased()</td>
</tr>
<tr>
<td>MouseMotionEvent</td>
<td>MouseMotionListener</td>
<td>mouseDragged(), mouseMoved()</td>
</tr>
<tr>
<td>TextEvent</td>
<td>TextListener</td>
<td>textValueChanged()</td>
</tr>
<tr>
<td>WindowEvent</td>
<td>WindowListener</td>
<td>windowActivated(), windowClosed(), windowClosing(), windowDeactivated(), windowDeiconified(), windowIconified(), windowOpened()</td>
</tr>
<tr>
<td>Component</td>
<td>Events Generated</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Button</td>
<td>ActionEvent</td>
<td>User clicked on the button</td>
</tr>
<tr>
<td>Checkbox</td>
<td>ItemEvent</td>
<td>User selected or deselected an item</td>
</tr>
<tr>
<td>CheckboxMenuItem</td>
<td>ItemEvent</td>
<td>User selected or deselected an item</td>
</tr>
<tr>
<td>Choice</td>
<td>ItemEvent</td>
<td>User selected or deselected an item</td>
</tr>
<tr>
<td>Component</td>
<td>ComponentEvent</td>
<td>Component moved, resized, hidden, or shown</td>
</tr>
<tr>
<td></td>
<td>FocusEvent</td>
<td>Component gained or lost focus</td>
</tr>
<tr>
<td></td>
<td>KeyEvent</td>
<td>User pressed or released a key</td>
</tr>
</tbody>
</table>
|                    | MouseEvent       | User pressed or released mouse button, mouse entered or exited component, or user moved or dragged mouse. Note: MouseEvent has two corresponding listeners, MouseListener and MouseMotionListener.
<table>
<thead>
<tr>
<th>Container</th>
<th>ContainerEvent</th>
<th>Component added to or removed from container</th>
</tr>
</thead>
<tbody>
<tr>
<td>List</td>
<td>ActionEvent</td>
<td>User double-clicked on list item</td>
</tr>
<tr>
<td></td>
<td>ItemEvent</td>
<td>User selected or deselected an item</td>
</tr>
<tr>
<td>MenuItem</td>
<td>ActionEvent</td>
<td>User selected a menu item</td>
</tr>
<tr>
<td>Scrollbar</td>
<td>AdjustmentEvent</td>
<td>User moved the scrollbar</td>
</tr>
<tr>
<td>TextComponent</td>
<td>TextEvent</td>
<td>User changed text</td>
</tr>
<tr>
<td>TextField</td>
<td>ActionEvent</td>
<td>User finished editing text</td>
</tr>
<tr>
<td>Window</td>
<td>WindowEvent</td>
<td>Window opened, closed, iconified, deiconified, or close requested</td>
</tr>
</tbody>
</table>
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) ;
    }
}
public void launchFrame() {
    Label label = new Label("Click and drag the mouse") ;
    f.add(label, BorderLayout.NORTH) ;
    f.add(tf, BorderLayout.SOUTH) ;
    f.addMouseMotionListener(this) ;
    f.addMouseListener(this) ;
    f.setSize(300,200) ;
    f.setVisible(true) ;
}

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

11

GUI Event Handling
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);
}

// 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();
}
GUI Event Handling

Java Programming
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.

- Example:

```java
import java.awt.*;
import java.awt.event.*;
public class MouseClickHandler extends MouseAdapter {
    public void mouseClicked(MouseEvent e) {
        // do stuff with the mouse click...
    }
}
```
Event Handling Using Anonymous Classes

► You can include an entire class definition within the scope of an expression.

► This approach defines what is called an anonymous inner class and creates the instance all at once.

► For example:

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

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

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

    public void launchFrame() {
        Label label = new Label("Click and drag the mouse");
        f.add(label, BorderLayout.NORTH);
        f.add(tf, BorderLayout.SOUTH);
        f.addMouseMotionListener(new MouseMotionAdapter() {
            public void mouseDragged(MouseEvent e) {
                String s = "Mouse dragging: X= " + e.getX() + "Y= " + e.getY();
                tf.setText(s);
            }
        });
    }
}
```
public static void main(String[] args) {
    TestAnonymous obj = new TestAnonymous();
    obj.launchFrame();
}
```java
import java.awt.*;
import java.awt.event.*;

public class MouseClickHandler extends MouseAdapter {

    private TextField tf;
    public static int count = 0;

    public MouseClickHandler(TextField tf) {
        this.tf = tf;
    }

    public void mouseClicked(MouseEvent e) {
        count++;
        String s = "Mouse has been clicked " + count + " times so far." ;
        tf.setText(s);
    }
}
```
GUI Event Handling

- You can implement event handlers as inner class.
- This allows access to the private data of the outer class.
- For example:
import java.awt.*;
import java.awt.event.*;

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

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

    public void launchFrame() {
        Label label = new Label("Click and drag the mouse");
        f.add(label, BorderLayout.NORTH);
        f.add(tf, BorderLayout.SOUTH);
        f.addMouseMotionListener(new MyMouseMotionListener());
        f.addMouseListener(new MouseClickHandler(tf));
        f.setSize(300, 200);
        f.setVisible(true);
    }
}
class MyMouseMotionListener extends MouseMotionAdapter {
    public void mouseDragged(MouseEvent e) {
        String s = "Mouse dragging: X= " + e.getX() + "Y= " + e.getY() ;
        tf.setText(s) ;
    }

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