Topics

• The Delegation Event Model

• Event Classes

• Event Listeners
  > ActionListener Method
  > MouseListener Methods
  > MouseMotionListener Methods
  > WindowListener Methods

• Steps for Creating GUI Applications with Event Handling
Delegation Event Model
What is Delegation Event Model?

• The Delegation Event Model
  > Model used by Java to handle user interaction with GUI components
  > Describes how your program can respond to user interaction

• Three important players
  > Event Source
  > Event Listener/Handler
  > Event Object
Event Source, Event Listener/Handler

- Event Source
  - GUI component that generates the event
  - Example: button

- Event Listener/Handler
  - Receives and handles events
  - Contains business logic
  - Example: displaying information useful to the user, computing a value
Event Object

- Created when an event occurs (i.e., user interacts with a GUI component)
- Contains all necessary information about the event that has occurred
  > Type of event that has occurred
  > Source of the event
- Represented by an Event class
Event Listener Registration to Event Source in Delegation Event Model
Event Listener Registers to Event Source

• A listener should be registered with a source
• Once registered, listener waits until an event occurs
• When an event occurs
  > An event object created by the event source
  > Event object is fired by the event source to the registered listeners (method of event listener is called with an event object as a parameter)
• Once the listener receives an event object from the source
  > Deciphers the event
  > Processes the event that occurred.
Control Flow of Delegation Event Model

1. Source Registers Listener
2. Fires an Event Object
3. Reacts to the Event
Methods of Event Source Used by Event Listeners for Registration

• Event source registering a listener:
  
  ```java
  void add<Type>Listener(<Type>Listener listenerObj)
  ```

  where,
  
  > `<Type>` depends on the type of event source
    > Can be `Key`, `Mouse`, `Focus`, `Component`, `Action` and others
  > One event source can register several listeners

• Registered listener being unregistered:
  
  ```java
  void remove<Type>Listener(<Type>Listener listenerObj)
  ```
Event Classes
Event Classes

- The *EventObject* class
  - Found in the *java.util* package
- The *AWTEvent* class
  - An immediate subclass of *EventObject*
  - Defined in *java.awt* package
  - Root of all AWT-based events
  - Subclasses follow this naming convention: `<Type>Event`
## Event Classes

<table>
<thead>
<tr>
<th>Event Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComponentEvent</td>
<td>Extends <code>AWTEvent</code>. Instantiated when a component is moved, resized, made visible or hidden.</td>
</tr>
<tr>
<td>InputEvent</td>
<td>Extends <code>ComponentEvent</code>. The abstract root event class for all component-level input event classes.</td>
</tr>
<tr>
<td>ActionEvent</td>
<td>Extends <code>AWTEvent</code>. Instantiated when a button is pressed, a list item is double-clicked, or a menu item is selected.</td>
</tr>
<tr>
<td>ItemEvent</td>
<td>Extends <code>AWTEvent</code>. Instantiated when an item is selected or deselected by the user, such as in a list or a checkbox.</td>
</tr>
<tr>
<td>KeyEvent</td>
<td>Extends <code>InputEvent</code>. Instantiated when a key is pressed, released or typed.</td>
</tr>
<tr>
<td>MouseEvent</td>
<td>Extends <code>InputEvent</code>. Instantiated when a mouse button is pressed, released, or clicked (pressed and released), or when a mouse cursor enters or exits a visible part of a component.</td>
</tr>
<tr>
<td>TextEvent</td>
<td>Extends <code>AWTEvent</code>. Instantiated when the value of a text field or a text area is changed.</td>
</tr>
<tr>
<td>WindowEvent</td>
<td>Extends <code>ComponentEvent</code>. Instantiated when a <code>Window</code> object is opened, closed, activated, deactivated, iconified, deiconified, or when focus is transferred into or out of the window.</td>
</tr>
</tbody>
</table>
Event Listeners
Event Listeners

- Classes that implement the `<Type>Listener` interfaces

<table>
<thead>
<tr>
<th>Event Listeners</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionListener</td>
<td>Receives action events.</td>
</tr>
<tr>
<td>MouseListener</td>
<td>Receives mouse events.</td>
</tr>
<tr>
<td>MouseMotionListener</td>
<td>Receives mouse motion events, which include dragging and moving the mouse.</td>
</tr>
<tr>
<td>WindowListener</td>
<td>Receives window events.</td>
</tr>
</tbody>
</table>
**ActionListener Method**

- Contains exactly one method

<table>
<thead>
<tr>
<th>ActionListener Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void actionPerformed(ActionEvent e)</td>
</tr>
</tbody>
</table>

Contains the handler for the `ActionEvent e` that occurred.
# MouseListener Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void mouseClicked(MouseEvent e)</td>
<td>Contains the handler for the event when the mouse is clicked (i.e., pressed and released).</td>
</tr>
<tr>
<td>public void mouseEntered(MouseEvent e)</td>
<td>Contains the code for handling the case wherein the mouse enters a component.</td>
</tr>
<tr>
<td>public void mouseExited(MouseEvent e)</td>
<td>Contains the code for handling the case wherein the mouse exits a component.</td>
</tr>
<tr>
<td>public void mousePressed(MouseEvent e)</td>
<td>Invoked when the mouse button is pressed on a component.</td>
</tr>
<tr>
<td>public void mouseReleased(MouseEvent e)</td>
<td>Invoked when the mouse button is released on a component.</td>
</tr>
</tbody>
</table>
### MouseMotionListener Methods

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void mouseDragged(MotionEvent e)</td>
<td>Contains the code for handling the case wherein the mouse button is pressed on a component and dragged. Called several times as the mouse is dragged.</td>
</tr>
<tr>
<td>public void mouseMoved(MotionEvent e)</td>
<td>Contains the code for handling the case wherein the mouse cursor is moved onto a component, without the mouse button being pressed. Called multiple times as the mouse is moved.</td>
</tr>
</tbody>
</table>
# WindowListener Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void windowopened(WindowEvent e)</td>
<td>Contains the code for handling the case when the Window object is opened (i.e., made visible for the first time).</td>
</tr>
<tr>
<td>public void windowclosing(WindowEvent e)</td>
<td>Contains the code for handling the case when the user attempts to close Window object from the object's system menu.</td>
</tr>
<tr>
<td>public void windowclosed(WindowEvent e)</td>
<td>Contains the code for handling the case when the Window object was closed after calling dispose (i.e., release of resources used by the source) on the object.</td>
</tr>
<tr>
<td>public void windowactivated(WindowEvent e)</td>
<td>Invoked when a Window object is the active window (i.e., the window in use).</td>
</tr>
<tr>
<td>public void windowdeactivated(WindowEvent e)</td>
<td>Invoked when a Window object is no longer the active window.</td>
</tr>
<tr>
<td>public void windowiconified(WindowEvent e)</td>
<td>Called when a Window object is minimized.</td>
</tr>
<tr>
<td>public void windowdeiconified(WindowEvent e)</td>
<td>Called when a Window object reverts from a minimized to a normal state.</td>
</tr>
</tbody>
</table>
Steps for Creating GUI Application with Event Handling
Steps for Creating GUI Applications with Event Handling

1. Create a GUI class
   > Describes and displays the appearance of your GUI application

2. Create Event Listener class (a class implementing the appropriate listener interface)
   > Override all methods of the appropriate listener interface
   > Describe in each method how you would like the event to be handled
   > May give empty implementations for methods you don't need
Steps for Creating GUI Applications with Event Handling (Continued)

3. Register the listener object with the event source
   > The object is an instantiation of the listener class in step 2
   > Use the `add<Type>Listener` method of the event source
import java.awt.*;
import java.awt.event.*;

public class MouseEventsDemo extends Frame implements MouseListener, MouseMotionListener {

    TextField tf;
    public MouseEventsDemo(String title){
        super(title);
        tf = new TextField(60);
        // Register event listener to the event source
        addMouseListener(this);
    }

    //continued...
// Displays GUI
public void launchFrame() {
    /* Add components to the frame */
    add(tf, BorderLayout.SOUTH);
    setSize(300, 300);
    setVisible(true);
}

// Implement methods of event listener interface
public void mouseClicked(MouseEvent me) {
    String msg = "Mouse clicked."
    tf.setText(msg);
}

//continued...
Mouse Events Example (page #3)

22    public void mouseEntered(MouseEvent me) {
23        String msg = "Mouse entered component.";
24        tf.setText(msg);
25    }
26    public void mouseExited(MouseEvent me) {
27        String msg = "Mouse exited component.";
28        tf.setText(msg);
29    }
30    public void mousePressed(MouseEvent me) {
31        String msg = "Mouse pressed.";
32        tf.setText(msg);
33    }
34    //continued...
public void mouseReleased(MouseEvent me) {
    String msg = "Mouse released.";
    tf.setText(msg);
}

public void mouseDragged(MouseEvent me) {
    String msg = "Mouse dragged at " + me.getX()
    + "," + me.getY();
    tf.setText(msg);
}

//continued...
public void mouseMoved(MouseEvent me) {
    String msg = "Mouse moved at " + me.getX() + "," + me.getY();
    tf.setText(msg);
}

// Main method
public static void main(String args[]) {
    MouseEventsDemo med =
        new MouseEventsDemo("Mouse Events Demo");
    med.launchFrame();
}

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

class CloseFrame extends Frame implements WindowListener {
    Label label;
    CloseFrame(String title) {
        super(title);
        label = new Label("Close the frame.");
        this.addWindowListener(this);
    }
    //continued...
Close Window Example (page #2)

```java
void launchFrame() {
    setSize(300,300);
    setVisible(true);
}

// Implement methods of listener interface
public void windowActivated(WindowEvent e) {
}

public void windowClosed(WindowEvent e) {
}

public void windowClosing(WindowEvent e) {
    setVisible(false);
    System.exit(0);
}

//continued...
```
Close Window Example (page #3)

```java
public void windowDeactivated(WindowEvent e) {
}

public void windowDeiconified(WindowEvent e) {
}

public void windowIconified(WindowEvent e) {
}

public void windowOpened(WindowEvent e) {
}

// Main method
public static void main(String args[]) {
    CloseFrame cf =
        new CloseFrame("Close Window Example");
    cf.launchFrame();
}
```
Adaptor Classes
Adapter Classes

• Why use Adapter classes?
  > Implementing all methods of an interface takes a lot of work
  > Interested in implementing some methods of the interface only

• Adapter classes
  > Built-in in Java
  > Implement all methods of each listener interface with more than one method
  > Implementations of the methods are all empty
Adapter Classes: Close Window Example

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

class CloseFrame extends Frame{
    Label label;
    CFLListener w = new CFLListener(this);

    CloseFrame(String title) {
        super(title);
        label = new Label("Close the frame.");
        this.addWindowListener(w);
    }

    //continued...
```
Adapter Classes: Close Window Example

```java
void launchFrame() {
    setSize(300, 300);
    setVisible(true);
}

public static void main(String args[]) {
    CloseFrame cf = new CloseFrame("Close Window Example");
    cf.launchFrame();
}
//continued...
```
Adapter Classes: Close Window Example

class CFLListener extends WindowAdapter {
    CloseFrame ref;

    CFLListener( CloseFrame ref ){
        this.ref = ref;
    }

    public void windowClosing(WindowEvent e) {
        ref.dispose();
        System.exit(1);
    }
}
Inner Classes

• Class declared within another class

• Why use inner classes?
  > Help simplify your programs
  > Especially in event handling
Inner Classes:
Close Window Example

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

class CloseFrame extends Frame{
    Label label;

    CloseFrame(String title) {
        super(title);
        label = new Label("Close the frame.");
        this.addWindowListener(new CFLListener());
    }

    //continued...
```
void launchFrame() {
    setSize(300, 300);
    setVisible(true);
}

class CFLListener extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
        dispose();
        System.exit(1);
    }
}

//continued...
Inner Classes: Close Window Example

```java
public static void main(String args[]) {
    CloseFrame cf =
        new CloseFrame("Close Window Example");
    cf.launchFrame();
```

30 }
Anonymous Inner Classes

• Unnamed inner classes

• Why use anonymous inner classes?
  > Further simplify your codes
  > Especially in event handling
Anonymous Inner Classes: Close Window Example

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

class CloseFrame extends Frame{
    Label label;
    CloseFrame(String title) {
        super(title);
        label = new Label("Close the frame.");
        this.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                dispose();
                System.exit(1);
            }
        });
    }
```
Anonymous Inner Classes: Close Window Example

```java
void launchFrame() {
    setSize(300, 300);
    setVisible(true);
}

public static void main(String args[]) {
    CloseFrame cf =
        new CloseFrame("Close Window Example");
    cf.launchFrame();
}
```
Thank You!