Module IV
Event Handling

Dr. Zahid Ansari

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

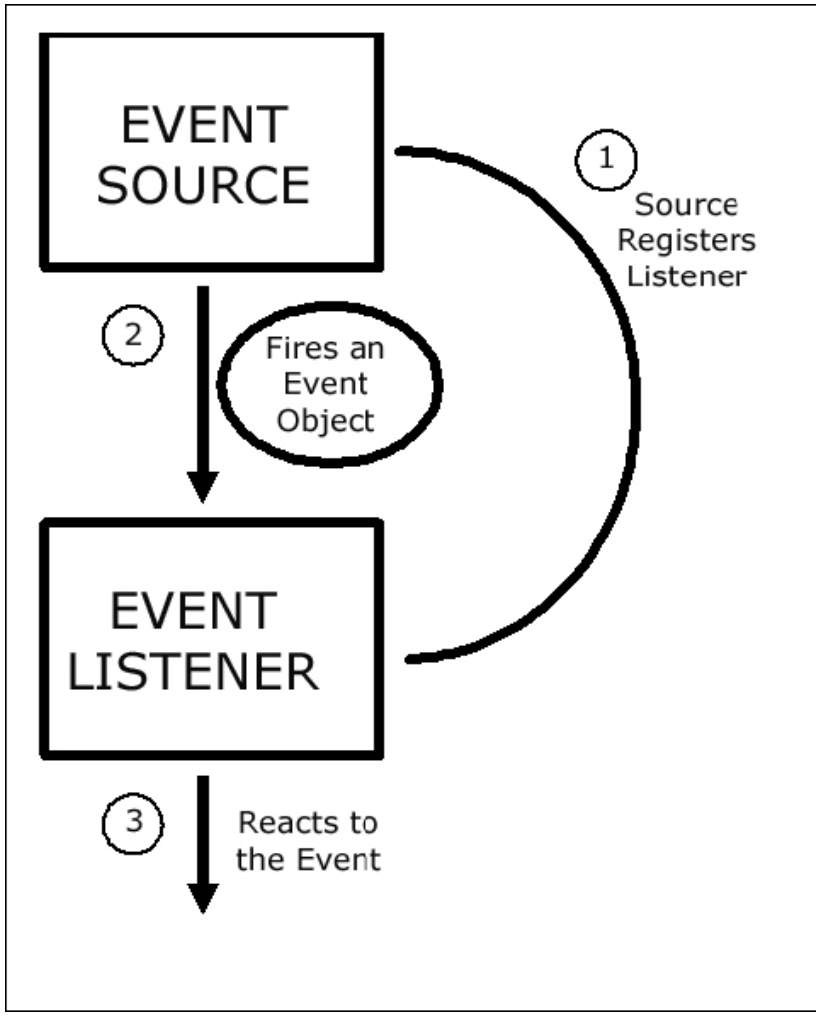
Event Object

- 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 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



Methods of Event Source Used by Event Listeners for Registation

- Event source registering a listener:

void add<Type>Listener(<Type>Listener listnerObj)

- 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:

void remove<Type>Listener(<Type>Listener listnerObj)

Event Classes

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

Event Listeners

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

Event Listeners	***Description***
ActionListener	Receives action events.
MouseListener	Receives mouse events.
MouseMotionListener	Receives mouse motion events, which include dragging and moving the mouse.
WindowListener	Receives window events.

MouseListener Methods

MouseListener Methods
`public void mouseClicked(MouseEvent e)`
Contains the handler for the event when the mouse is clicked (i.e., pressed and released).
`public void mouseEntered(MouseEvent e)`
Contains the code for handling the case wherein the mouse enters a component.
`public void mouseExited(MouseEvent e)`
Contains the code for handling the case wherein the mouse exits a component.
`public void mousePressed(MouseEvent e)`
Invoked when the mouse button is pressed on a component.
`public void mouseReleased(MouseEvent e)`
Invoked when the mouse button is released on a component.

MouseMotionListener Methods

MouseListener Methods
`public void mouseDragged(MouseEvent e)`
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.
`public void mouseMoved(MouseEvent e)`
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.

WindowListener Methods\

WindowListener Methods
`public void windowOpened(WindowEvent e)`
Contains the code for handling the case when the *Window* object is opened (i.e., made visible for the first time).
`public void windowClosing(WindowEvent e)`
Contains the code for handling the case when the user attempts to close *Window* object from the object's system menu.
`public void windowClosed(WindowEvent e)`
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.
`public void windowActivated(WindowEvent e)`
Invoked when a *Window* object is the active window (i.e., the window in use).
`public void windowDeactivated(WindowEvent e)`
Invoked when a *Window* object is no longer the active window.
`public void windowIconified(WindowEvent e)`
Called when a *Window* object is minimized.
`public void windowDeiconified(WindowEvent e)`
Called when a *Window* object reverts from a minimized to a normal state.

Example: Mouse Event Handler

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

public class MouseEvents extends Applet
    implements MouseListener, MouseMotionListener {
    String msg = "";
    int mouseX = 0, mouseY = 0; // coordinates of mouse
    public void init() {
        addMouseListener(this);
        addMouseMotionListener(this);
    }
    // Handle mouse clicked.
    public void mouseClicked(MouseEvent me) {
        // save coordinates
        mouseX = 0;
        mouseY = 10;
        msg = "Mouse clicked.";
        repaint();
    }
```

Example: Mouse Event Handler

```
// Handle mouse entered.
public void mouseEntered(MouseEvent me) {
     // save coordinates
     mouseX = 0;
     mouseY = 10;
     msg = "Mouse entered.";
     repaint();
}

// Handle mouse exited.
public void mouseExited(MouseEvent me) {
       // save coordinates
      mouseX = 0;
      mouseY = 10;
      msg = "Mouse exited.";
     repaint();
}
```

Example: Mouse Event Handler

```
// Handle button pressed.
public void mousePressed(MouseEvent me) {
    // save coordinates
    mouseX = me.getX();
    mouseY = me.getY();
    msg = "Down";
    repaint();
}

// Handle button released.
public void mouseReleased(MouseEvent me) {
    // save coordinates
    mouseX = me.getX();
    mouseY = me.getY();
    msg = "Up";
    repaint();
}
```

Example: Mouse Event Handler

```
// Handle mouse dragged.
public void mouseDragged(MouseEvent me) {
      // save coordinates
      mouseX = me.getX();
      mouseY = me.getY();
      msg = "*";
      showStatus("Dragging mouse at " + mouseX + ", " + mouseY);
      repaint();
}

// Handle mouse moved.
public void mouseMoved(MouseEvent me) {
      // show status
       showStatus("Moving mouse at " + me.getX() + ", " + me.getY());
}

// Display msg in applet window at current X,Y location.
      public void paint(Graphics g) {
      g.drawString(msg, mouseX, mouseY);
  }
}
```

Example: Simple Key Handler

```
// Demonstrate the key event handlers.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
  <applet code="SimpleKey" width=300 height=100>
  </applet>
*/

public class SimpleKey extends Applet
  implements KeyListener {

  String msg = "";
  int X = 10, Y = 20; // output coordinates

  public void init() {
    addKeyListener(this);
  }
```

Example: Simple Key Handler

```
public void keyPressed(KeyEvent ke) {
    showStatus("Key Down");
  }

  public void keyReleased(KeyEvent ke) {
    showStatus("Key Up");
  }

  public void keyTyped(KeyEvent ke) {
    msg += ke.getKeyChar();
    repaint();
  }

  // Display keystrokes.
  public void paint(Graphics g) {
    g.drawString(msg, X, Y);
  }
}
```

Example: Key Event Handler

```
// Demonstrate some virutal key codes.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
  <applet code="KeyEvents" width=300 height=100>
  </applet>
*/

public class KeyEvents extends Applet
  implements KeyListener {

  String msg = "";
  int X = 10, Y = 20; // output coordinates

  public void init() {
    addKeyListener(this);
  }
```

Example: Key Event Handler

```
public void keyPressed(KeyEvent ke) {
    showStatus("Key Down");
    int key = ke.getKeyCode();
    switch(key) {
                    case KeyEvent.VK_F1:
                    msg += "<F1>";  break;
                    case KeyEvent.VK_F2:
                    msg += "<F2>"; break;
                    case KeyEvent.VK_F3:
                    msg += "<F3>"; break;
                    case KeyEvent.VK_PAGE_DOWN:
                    msg += "<PgDn>"; break;
                    case KeyEvent.VK_PAGE_UP:
                    msg += "<PgUp>"; break;
                    case KeyEvent.VK_LEFT:
                     msg += "<Left Arrow>"; break;
                    case KeyEvent.VK_RIGHT:
                    msg += "<Right Arrow>"; break;
    }
    repaint();
 }
```

Example: Key Event Handler

```
 public void keyReleased(KeyEvent ke) {
    showStatus("Key Up");
  }

  public void keyTyped(KeyEvent ke) {
    msg += ke.getKeyChar();
    repaint();
  }

  // Display keystrokes.
  public void paint(Graphics g) {
    g.drawString(msg, X, Y);
  }
}
```

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

Example: Mouse Adapter

```
// Demonstrate an adaptor.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
  <applet code="AdapterDemo" width=300 height=100>
  </applet>
*/

public class AdapterDemo extends Applet {
    public void init() {
        addMouseListener(new MyMouseAdapter(this));
        addMouseMotionListener(new MyMouseMotionAdapter(this));
    }
}
```

Example: Mouse Adapter

```
class MyMouseAdapter extends MouseAdapter {
    AdapterDemo adapterDemo;
    public MyMouseAdapter(AdapterDemo adapterDemo) {
        this.adapterDemo = adapterDemo;
    }
    // Handle mouse clicked.
    public void mouseClicked(MouseEvent me) {
        adapterDemo.showStatus("Mouse clicked");
    }
}
class MyMouseMotionAdapter extends MouseMotionAdapter {
    AdapterDemo adapterDemo;
    public MyMouseMotionAdapter(AdapterDemo adapterDemo) {
        this.adapterDemo = adapterDemo;
    }
    // Handle mouse dragged.
    public void mouseDragged(MouseEvent me) {
        adapterDemo.showStatus("Mouse dragged");
    }
}
```

Inner Classes

- Class declared within another class
- Why use inner classes?
  - Help simplify your programs
  - Especially in event handling

Example: Without Inner Class

```
// This applet does NOT use an inner class.
import java.applet.*;
import java.awt.event.*;

public class MousePressedDemo extends Applet {
    public void init() {
        addMouseListener(new MyMouseAdapter(this));
    }
}

class MyMouseAdapter extends MouseAdapter {
    MousePressedDemo mousePressedDemo;
    public MyMouseAdapter(MousePressedDemo mousePressedDemo) {
        this.mousePressedDemo = mousePressedDemo;
     }
    public void mousePressed(MouseEvent me) {
        mousePressedDemo.showStatus("Mouse Pressed.");
    }
}
```

Example: Using Inner Class

```
// Inner class demo
import java.applet.*;
import java.awt.event.*;
/*
  <applet code="InnerClassDemo" width=200 height=100>
  </applet>
*/

public class InnerClassDemo extends Applet {
    public void init() {
        addMouseListener(new MyMouseAdapter());
    }
    class MyMouseAdapter extends MouseAdapter {
        public void mousePressed(MouseEvent me) {
            showStatus("Mouse Pressed");
        }
    }
}
```

Anonymous Inner Classes

- Unnamed inner classes
- Why use anonymous inner classes?
  - Further simplify your codes
  - Especially in event handling

Example: Anonymous Inner Class

```
// Anonymous inner class demo.
import java.applet.*;
import java.awt.event.*;
/*
  <applet code="AnonymousInnerClassDemo" width=200 height=100>
  </applet>
*/

public class AnonymousInnerClassDemo extends Applet {
    public void init() {
        addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent me) {
                showStatus("Mouse Pressed");
            }
        });
    }
}
```