MODULE 5 - EVENT HANDLING

INTRODUCTION

Event handling is fundamental to Java programming because it is integral to the creation of applets and other types of GUI-based programs. applets are event-driven programs that use a graphical user interface to interact with the user. Furthermore, any program that uses a graphical user interface, such as a Java application written for Windows, is event driven. Thus, you cannot write these types of programs without a solid command of event handling. Events are supported by a number of packages, including **java.util, java.awt,** and **java.awt.event.**

Two Event Handling Mechanisms

1. The old method used in original version of Java (1.0)— Old method, still available, but not in use, and not recommended for new programs.
2. The Delegation Event Model— The modern approach, where the events should be handled by all new programs

The Delegation Event Model

- The modern approach to handling events is based on the ***delegation event model***, which defines standard and consistent mechanisms to generate and process events.
- Here a source generates an event and sends it to one or more listeners.
- The listener simply waits until it receives an event. Once an event is received, the listener processes the event and then returns.
- The advantage of this design is that the application logic that processes events is cleanly separated from the user interface logic that generates those events.
- A user interface element is able to "delegate" the processing of an event to a separate piece of code.
- In the delegation event model, listeners must register with a source in order to receive an event notification. This provides an important benefit: notifications are sent only to listeners that want to receive them.
- This is a more efficient way to handle events than the design used by the old Java 1.0 approach.

Events

- An ***event*** *is an object that describes a state change in a source.*
- It can be generated as a consequence of a person interacting with the elements in a graphical user interface.
- For example, pressing a button, entering a character via the keyboard, selecting an item in a list, and clicking the mouse.
- Events may also occur that are not directly caused by interactions with a user interface.
- For example, an event may be generated when a timer expires, a counter exceeds a value, a software or hardware failure occurs, or an operation is completed.

Event Sources

- A ***source*** *is an object that generates an event. This occurs when the internal state of that object* changes in some way. Sources may generate more than one type of event.
- A source must register listeners in order for the listeners to receive notifications about a specific type of event.
- Each type of event has its own registration method.
- The general form is:

  public void add*TypeListener(TypeListener el)*

- Here, ***Type*** *is the name of the event, and* ***el*** *is a reference to the event listener.*
- For example, the method that registers a keyboard event listener is called **addKeyListener( ). The method** that registers a mouse motion listener is called **addMouseMotionListener( ).**
- Some sources may allow only one listener to register. The general form of such a method is this:

  public void add*TypeListener(TypeListener el)*
  throws java.util.TooManyListenersException

  This is known as *unicasting the event.*
- A source must also provide a method that allows a listener to unregister an interest in a specific type of event. The general form of such a method is this:

  public void remove*TypeListener(TypeListener el)*

  Here, *Type is the name of the event, and el is a reference to the event listener.*

Event Listeners

- A listener is an object that is notified when an event occurs.
- It has two major requirements.
- First, it must have been registered with one or more sources to receive notifications about specific types of events.
- Second, it must implement methods to receive and process these notifications.
- The methods that receive and process events are defined in a set of interfaces found in java.awt.event. For example, the MouseMotionListener interface defines two methods to receive notifications when the mouse is dragged or moved.

Event Classes

- The classes that represent events are at the core of Java's event handling mechanism.
- At the root of the Java event class hierarchy is **EventObject,** which is in **java.util.** It is the superclass for all events. Its one constructor is shown here:

  EventObject(Object *src)*

  Here, ***src*** *is the object that generates this event.*
- **EventObject** contains two methods: **getSource( )** and **toString( ).** The getSource( ) method returns the source of the event. Its general form is shown here:

  Object getSource( )
- As expected, toString( ) returns the string equivalent of the event.
- The class **AWTEvent,** defined within the **java.awt** package, is a subclass of **EventObject.** It is the superclass (either directly or indirectly) of all AWT-based events used by the delegation event model.
- Its **getID( )** method can be used to determine the type of the event. The signature of this method is shown here:

  int getID( )
- It is important to know only that all of the other classes discussed in event handling are subclasses of **AWTEvent.**

To summarize:

- **EventObject** is a superclass of all events.
- **AWTEvent** is a superclass of all AWT events that are handled by the delegation event model.

Main Event Classes in java.awt.event

Event Class	Description
ActionEvent	Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.
AdjustmentEvent	Generated when a scroll bar is manipulated.
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible.
ContainerEvent	Generated when a component is added to or removed from a container.
FocusEvent	Generated when a component gains or loses keyboard focus.
InputEvent	Abstract superclass for all component input event classes.
ItemEvent	Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.
KeyEvent	Generated when input is received from the keyboard.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
MouseWheelEvent	Generated when the mouse wheel is moved.
TextEvent	Generated when the value of a text area or text field is changed.
WindowEvent	Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

The MouseEvent Class

There are eight types of mouse events. The MouseEvent class defines the following integer that can be used to identify them:

MOUSE_CLICKED	The user clicked the mouse.
MOUSE_DRAGGED	The user dragged the mouse.
MOUSE_ENTERED	The mouse entered a component.
MOUSE_EXITED	The mouse exited from a component.
MOUSE_MOVED	The mouse moved.
MOUSE_PRESSED	The mouse was pressed.
MOUSE_RELEASED	The mouse was released.
MOUSE_WHEEL	The mouse wheel was moved.

Constructor

MouseEvent(Component *src, int type, long when, int modifiers,* int *x, int y, int clicks, boolean triggersPopup)*

Two commonly used methods in this class are **getX( ) and getY( ). These return the X and** Y coordinates of the mouse within the component when the event occurred. Their forms are:

int getX( )

int getY( )

you can use the **getPoint( ) method to obtain the coordinates of the mouse.** It is shown here:

Point getPoint( )

The **translatePoint( )** method changes the location of the event. Its form is shown here: void translatePoint(int *x, int y)*

The **getClickCount( )** method obtains the number of mouse clicks for this event. Its signature is shown here:

int getClickCount( )

The **isPopupTrigger( )** method tests if this event causes a pop-up menu to appear on this platform. Its form is shown here:

boolean isPopupTrigger( )

Also available is the **getButton( )** method, shown here:

int getButton( )

It returns a value that represents the button that caused the event. The return value will be one of these constants defined by **MouseEvent:**

NOBUTTON	BUTTON1	BUTTON2	BUTTON3

Event Listener Interfaces

Interface	Description
ActionListener	Defines one method to receive action events.
AdjustmentListener	Defines one method to receive adjustment events.
ComponentListener	Defines four methods to recognize when a component is hidden, moved, resized, or shown.
ContainerListener	Defines two methods to recognize when a component is added to or removed from a container.
FocusListener	Defines two methods to recognize when a component gains or loses keyboard focus.
ItemListener	Defines one method to recognize when the state of an item changes.
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.
MouseListener	Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.
MouseMotionListener	Defines two methods to recognize when the mouse is dragged or moved.
MouseWheelListener	Defines one method to recognize when the mouse wheel is moved.
TextListener	Defines one method to recognize when a text value changes.
WindowFocusListener	Defines two methods to recognize when a window gains or loses input focus.
WindowListener	Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

The MouseListener Interface

This interface defines five methods.

The general forms of these methods are shown here:

void mouseClicked(MouseEvent *me)*

void mouseEntered(MouseEvent *me)*

void mouseExited(MouseEvent *me)*

void mousePressed(MouseEvent *me)*

void mouseReleased(MouseEvent *me)*

The KeyListener Interface

This interface defines three methods.

The general forms of these methods are shown here:

void keyPressed(KeyEvent *ke)*

void keyReleased(KeyEvent *ke)*

void keyTyped(KeyEvent *ke)*

Program to demonstrate handling of mouse events

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="MouseEvents" width=300 height=100>
</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);
               setFont(new Font("Arial",Font.BOLD,20));
     }
```

```
// Handle mouse clicked.
public void mouseClicked(MouseEvent me)
{
     // save coordinates
     mouseX = 0;
     mouseY = 10;
     msg = "Mouse clicked.";
     repaint();
}
// Handle mouse entered.
public void mouseEntered(MouseEvent me)
{
     // save coordinates
     mouseX = 10;
     mouseY = 20;
     msg = "Mouse entered.";
     repaint();
}
// Handle mouse exited.
public void mouseExited(MouseEvent me)
{
     // save coordinates
     mouseX = 10;
     mouseY = 20;
     msg = "Mouse exited.";
     repaint();
}
// 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();
}
// 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);
      }
}
```

Program to handle keyboard events

```
// 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);
      }
      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);
      }
}
```