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.
- Its concept is quite simple:
  a source generates an event and sends it to one or more listeners.
- In this scheme, 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.

Events

- In the delegation model, **an event is an object** that describes a state change in a source.
- An event can be generated as a consequence of a person interacting with the elements in a graphical user interface.
- Some of the activities that cause events to be generated are 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.

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. Here is the general form:

public void addTypeListener (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( ).**

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, such as those found in java.awt.event.

- For example, the MouseMotionListener interface defines two methods to receive notifications when the mouse is dragged or moved.

- Any object may receive and process one or both of these events if it provides an implementation of this interface.

Event Classes

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.

Sources of Events

Event Source	Description
Button	Generates action events when the button is pressed.
Check box	Generates item events when the check box is selected or deselected.
Choice	Generates item events when the choice is changed.
List	Generates action events when an item is double-clicked; generates item events when an item is selected or deselected.
Menu item	Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected.
Scroll bar	Generates adjustment events when the scroll bar is manipulated.
Text components	Generates text events when the user enters a character.
Window	Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Event Listener Interfaces

As explained, the delegation event model has two parts: sources and listeners. As it relates to this chapter, listeners are created by implementing one or more of the interfaces defined by the java.awt.event package.

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 ActionListener Interface

This interface defines the actionPerformed( ) method that is invoked when an action event occurs.

Its general form is shown here:

void actionPerformed(ActionEvent ae)

The ItemListener Interface

This interface defines the itemStateChanged( ) method that is invoked when the state of an item changes. Its general form is shown here:

void itemStateChanged(ItemEvent ie)

The KeyListener Interface

This interface defines three methods.

The keyPressed( ) and keyReleased( ) methods are invoked when a key is pressed and released, respectively.

The keyTyped( ) method is invoked when a character has been entered.

For example, if a user presses and releases the a key, three events are generated in sequence: key pressed, typed, and released.

The general forms of these methods are shown here:

void keyPressed(KeyEvent ke)

void keyReleased(KeyEvent ke)

void keyTyped(KeyEvent ke)

The MouseListener Interface

This interface defines five methods.

If the mouse is pressed and released at the same point, mouseClicked( ) is invoked.

When the mouse enters a component, the mouseEntered( ) method is called.

When it leaves, mouseExited( ) is called. The mousePressed( ) and mouseReleased( ) methods are invoked when the mouse is pressed and released, respectively.

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)

Handling 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);
    }
  }
```

Sample output is shown here: