Chapter -2

Event Handling

What is EVENT?

- Something that takes place
- The outcome

In java, Event are the change in the state of the object

Two event handling mechanisms

- 1.0 Model
- The delegation event model

- The way in which events are handled changed significantly between the original version of Java (1.0) and all subsequent versions of Java, beginning with version 1.1.
- Although the 1.0 method of event handling is still supported, it is not recommended for new programs.
- Also, many of the methods that support the old 1.0 event model have been deprecated.

1.0 Event Model

- An event was propagated up the containment hierarchy until it was handled by a component.
- Upon receiving a user-initiated event, like a mouse click, the system generates an instance of the Event class and passes it along to the program. The program identifies the event's target (i.e., the component in which the event occurred) and asks that component to handle the event. If the target can't handle this event, an attempt is made to find a component that can, and the process repeats.
- This required components to receive events that they did not process, and it wasted valuable time. The delegation event model eliminates this overhead.

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.

- 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.

Events

- In the delegation model, an event is an object that describes a state change in a source.
- Among other causes, 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.
- Many other user operations could also be cited as examples. 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, software or hardware failure occurs, or an operation is completed. You are free to define events that are appropriate for your application.

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.
- 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( )**.

- Some sources may allow more than one listener to register. When an event occurs, all registered listeners are notified and receive a copy of the event object. This is known as multicasting the event.
- In all cases, notifications are sent only to listeners that register to receive them.
- Some sources may allow only one listener to register. When such an event occurs, the registered listener is notified. 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 method is :

  public void **removeTypeListener**(TypeListener el)

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

- For example, to remove a keyboard listener, you would call **removeKeyListener**( ).
- The methods that add or remove listeners are provided by the source that generates events.
- For example, the Component class provides methods to add and remove keyboard and mouse event listeners.

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.

Event Classes

- The classes that represent events are at the core of Java's event handling mechanism.
- Thus, a discussion of event handling must begin with the event classes.
- It is important to understand, however, that Java defines several types of events and that not all event classes can be discussed in this chapter.
- Arguably, the most widely used events at the time of this writing are those defined by the AWT and those defined by Swing.

The KeyEvent Class

- A KeyEvent is generated when keyboard input occurs.
- KeyEvent is a subclass of InputEvent.
- There are three types of key events, which are identified by these integer constants:

  KEY_PRESSED, KEY_RELEASED, and KEY_TYPED.

- The first two events are generated when any key is pressed or released.
- The last event occurs only when a character is generated.
- Remember, not all key presses result in characters. For example, pressing shift does not generate a character.

There are many other integer constants that are defined by KeyEvent.

For example, VK_0 through VK_9 and VK_A through VK_Z define the ASCII equivalents of the numbers and letters.

The **VK** constants specify *virtual key codes*

VK_ALT	VK_DOWN	VK_LEFT	VK_RIGHT
VK_CANCEL	VK_ENTER	VK_PAGE_DOWN	VK_SHIFT
VK_CONTROL	VK_ESCAPE	VK_PAGE_UP	VK_UP

The MouseEvent Class

- This event indicates a mouse action occurred in a component.
- MouseEvent is a subclass of InputEvent.
- There are eight types of mouse events. The MouseEvent class defines the following integer constants 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.

- 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 shown here:

  int getX( )

  int getY( )

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

  Point getPoint( )

  It returns a Point object that contains the X,Y coordinates in its integer members: x and y.

- Also three methods are available that obtain the coordinates of the mouse relative to the screen rather than the component. They are shown here:
  - Point getLocationOnScreen( )
  - int getXOnScreen( )
  - int getYOnScreen( )
- The getLocationOnScreen( ) method returns a Point object that contains both the X and Y coordinate. The other two methods return the indicated coordinate.

TextEvent Class

- TextEvent is generated when character is entered in the text fields or text area.
- Instances of this class describe text events. These are generated by text fields and text areas when characters are entered by a user or program.
- TextEvent defines the integer constant TEXT_VALUE_CHANGED.

WindowEvent Class

- The object of this class represents the change in state of a window.
- This low-level event is generated by a Window object when it is :
  - Opened
  - Closed
  - Activated
  - Deactivated
  - Focus is transferred into or out of the Window.

Some Window events are:

WINDOW_ACTIVATED	**The window was activated.**
WINDOW_CLOSED	The window has been closed.
WINDOW_CLOSING	The user requested that the window be closed.
WINDOW_DEACTIVATED	The window was deactivated.
WINDOW_DEICONIFIED	The window was deiconified.
WINDOW_GAINED_FOCUS	The window gained input focus.
WINDOW_ICONIFIED	The window was iconified.
WINDOW_LOST_FOCUS	The window lost input focus.
WINDOW_OPENED	The window was opened.
WINDOW_STATE_CHANGED	The state of the window changed.

- A commonly used method in this class is getWindow( ). It returns the Window object that generated the event. Its general form is shown here:

  Window getWindow( )

- WindowEvent also defines methods that return the opposite window (when a focus or activation event has occurred), the previous window state, and the current window state. These methods are shown here:

  Window getOppositeWindow( )

  int getOldState( )

  int getNewState( )

Event Listener Interfaces

- As explained, the delegation event model has two parts: sources and listeners.
- Listeners are created by implementing one or more of the interfaces defined by the java.awt.event package.
- When an event occurs, the event source invokes the appropriate method defined by the listener and provides an event object as its argument.
- Table below lists several commonly used listener interfaces and provides a brief description of the methods that they define.
- The following sections examine the specific methods that are contained in each interface.

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

- The Java ActionListener is notified whenever you click on the button or menu item.
- It is notified against ActionEvent.
- The ActionListener interface is found in java.awt.event package.
- It has only one method: actionPerformed(). It is invoked when an action event occurs. Its general form is shown here:

```
void actionPerformed(ActionEvent ae)
```

The KeyListener Interface

- The Java KeyListener is notified whenever you change the state of key.
- It is notified against KeyEvent.
- The KeyListener interface is found in java.awt.event package. It has three methods.
  - void keyPressed(KeyEvent ke)
  - void keyReleased(KeyEvent ke)
  - void keyTyped(KeyEvent ke)
- Example: if a user presses and releases the a key, three events are generated in sequence: key pressed, typed, and released.
- If a user presses and releases the home key, two key events are generated in sequence: key pressed and released.

The MouseListener Interface

- The Java MouseListener is notified whenever you change the state of mouse.
- It is notified against MouseEvent.
- The MouseListener interface is found in java.awt.event package.
- It has five methods:
    - void mouseClicked(MouseEvent me)
    - void mouseEntered(MouseEvent me)
    - void mouseExited(MouseEvent me)
    - void mousePressed(MouseEvent me)
    - void mouseReleased(MouseEvent me)
- 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 WindowListener Interface

- The Java WindowListener is notified whenever you change the state of window.
- It is notified against WindowEvent.
- The WindowListener interface is found in java.awt.event package.
- This interface defines seven methods:
  - void windowActivated(WindowEvent we)
  - void windowClosed(WindowEvent we)
  - void windowClosing(WindowEvent we)
  - void windowDeactivated(WindowEvent we)
  - void windowDeiconified(WindowEvent we)
  - void windowIconified(WindowEvent we)
  - void windowOpened(WindowEvent we)

The **windowActivated**( ) and **windowDeactivated**( ) methods are invoked when a window is activated or deactivated, respectively. If a window is iconified, the windowIconified( ) method is called. When a window is deiconified, the **windowDeiconified**( ) method is called. When a window is opened or closed, the **windowOpened**( ) or **windowClosed**( ) methods are called, respectively. The **windowClosing**( ) method is called when a window is being closed.

ContainerListener Interface

- The interface ContainerListener is used for receiving container events.
- The class that process container events needs to implements this interface.
- Two methods of this interface are:
  - void componentAdded(ContainerEvent e)
  - void componentRemoved(ContainerEvent e)

Adapter Class

- Java provides a special feature, called an adapter class, that can simplify the creation of event handlers in certain situations.
- An adapter class provides an empty implementation of all methods in an event listener interface.
- Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface.
- You can define a new class to act as an event listener by extending one of the adapter classes and implementing only those events in which you are interested.
- For example, the MouseMotionAdapter class has two methods, mouseDragged( ) and mouseMoved( ), which are the methods defined by the MouseMotionListener interface. If you were interested in only mouse drag events, then you could simply extend MouseMotionAdapter and override mouseDragged( ). The empty implementation of mouseMoved( ) would handle the mouse motion events for you.

Adapter Class	Listener Interface
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener and (as of JDK 6) MouseMotionListener and MouseWheelListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener, WindowFocusListener, and WindowStateListener