UNIT-I

Chapter -1

Event Handling

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has the code which is known as event handler that is executed when an event occurs. Java Uses the **Delegation Event Model** to handle the events. This model defines the standard mechanism to generate and handle the events.

There are three participants in event delegation model in Java;
- Event Source – the class which broadcasts the events
- Event Listeners – the classes which receive notifications of events
- Event Object – the class object which describes the event.

Source - The source is an object on which event occurs. Source is responsible for providing information of the occurred event to it's handler. Java provide as with classes for source object.

Listener - It is also known as event handler. Listener is responsible for generating response to an event. From java implementation point of view the listener is also an object. Listener waits until it receives an event. Once the event is received, the listener process the event an then returns.

The benefit of this approach is that the user interface logic is completely separated from the logic that generates the event. The user interface element is able to delegate the processing of an event to the separate piece of code. In this model, Listener needs to be registered with the source object so that the listener can receive the event notification. This is an efficient way of handling the event because the event notifications are sent only to that listener that want to receive them.

Steps involved in event handling

- The User clicks the button and the event is generated.
- Now the object of concerned event class is created automatically and information about the source and the event get populated with in same object.

- Event object is forwarded to the method of registered listener class.
- the method is now get executed and returns.

Event Classes:

Following is the list of commonly used event classes.

S No.	Control & Description
1	**AWTEvent** It is the root event class for all AWT events. This class and its subclasses supercede the original java.awt.Event class.
2	**ActionEvent** The ActionEvent is generated when button is clicked or the item of a list is double clicked.
3	**InputEvent** The InputEvent class is root event class for all component-level input events.
4	**KeyEvent** On entering the character the Key event is generated.
5	**MouseEvent** This event indicates a mouse action occurred in a component.
6	**TextEvent** The object of this class represents the text events.
7	**WindowEvent** The object of this class represents the change in state of a window.
8	**AdjustmentEvent** The object of this class represents the adjustment event emitted by Adjustable objects.
9	**ComponentEvent** The object of this class represents the change in state of a window.
10	**ContainerEvent** The object of this class represents the change in state of a window.
11	**MouseMotionEvent** The object of this class represents the change in state of a window.
12	**PaintEvent** The object of this class represents the change in state of a window.

The Event listener represent the interfaces responsible to handle events. Java provides us various Event listener classes but we will discuss those which are more frequently used. Every method of an event listener method has a single argument as an object which is subclass of EventObject class. For example, mouse event listener methods will accept instance of MouseEvent, where MouseEvent derives from EventObject.

EventListner interface

It is a marker interface which every listener interface has to extend.This class is defined in java.util package.

Class declaration

Following is the declaration for **java.util.EventListener** interface:

```
public interface EventListener
```

Event Listener Interfaces:

Following is the list of commonly used event listeners.

S No.	Control & Description
1	**ActionListener** This interface is used for receiving the action events.
2	**ComponentListener** This interface is used for receiving the component events.
3	**ItemListener** This interface is used for receiving the item events.
4	**KeyListener** This interface is used for receiving the key events.
5	**MouseListener** This interface is used for receiving the mouse events.
6	**TextListener** This interface is used for receiving the text events.
7	**WindowListener** This interface is used for receiving the window events.
8	**AdjustmentListener** This interface is used for receiving the adjusmtent events.
9	**ContainerListener** This interface is used for receiving the container events.
10	**MouseMotionListener** This interface is used for receiving the mouse motion events.
11	**FocusListener** This interface is used for receiving the focus events.

Adapters are abstract classes for receiving various events. The methods in these classes are empty. These classes exist as convenience for creating listener objects.

Layout Manager

Layout means the arrangement of components within the container. In other way we can say that placing the components at a particular position within the container. The task of layouting the controls is done automatically by the Layout Manager.

The layout manager automatically positions all the components within the container. If we do not use layout manager then also the components are positioned by the default layout manager. It is possible to layout the controls by hand but it becomes very difficult because of the following two reasons.

- It is very tedious to handle a large number of controls within the container.
- Often the width and height information of a component is not given when we need to arrange them.

Java provides us with various layout managers to position the controls. The properties like size, shape and arrangement varies from one layout manager to other layout manager. When the size of the applet or the application window changes the size, shape and arrangement of the components also changes in response i.e. the layout managers adapt to the dimensions of applet viewer or the application window.

AWT Layout Manager Classes:

Following is the list of commonly used controls while designed GUI using AWT.

S No.	Layout Manager & Description
1	**BorderLayout** The borderlayout arranges the components to fit in the five regions: east, west, north, south and center.
2	**CardLayout** The CardLayout object treats each component in the container as a
	card. Only one card is visible at a time.
3	**FlowLayout** The FlowLayout is the default layout.It layouts the components in a directional flow.
4	**GridLayout** The GridLayout manages the components in form of a rectangular grid.

Program to implement Key Events

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="Key.class" width=300 height=400>
</applet>
*/
public class Key extends Applet
implements KeyListener
{
        int X=20,Y=30;
        String msg="KeyEvents--->";
        public void init()
        {
                addKeyListener(this);
                setBackground(Color.green);
                setForeground(Color.blue);
        }
        public void keyPressed(KeyEvent k)
        {
                showStatus("KeyDown");

        }
        public void keyReleased(KeyEvent k)
        {
                showStatus("Key Up");
        }
        public void keyTyped(KeyEvent k)
        {
                msg+=k.getKeyChar();
                repaint();
        }
        public void paint(Graphics g)
        {
                g.drawString(msg,X,Y);
        }
}
```

Program to implement Mouse Events

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

/*
<applet code="MouseDemo.class" width=300 height=400>
</applet>
*/
public class MouseDemo extends Applet implements MouseListener ,
MouseMotionListener {
   String msg="";
   int mouseX=0,mouseY=0;
   public void init() {
      addMouseListener(this);
      addMouseMotionListener(this);
   }
    public void mouseClicked(MouseEvent me)
{
 mouseX=me.getX();
 mouseY=me.getY();
 msg="Mouse Clicked.";
 repaint();
}
    public void mouseEntered(MouseEvent me)
    {
mouseX=0;
mouseX=10;
msg="Entered.";
repaint();
    }
    public void mouseExited(MouseEvent me)
    {
mouseX=0;
mouseX=10;
msg="Exited";
repaint();
    }
```

```
    public void mousePressed(MouseEvent me)
    {
mouseX=0;
mouseY=10;
msg="Down.";
repaint();
    }
    public void mouseReleased(MouseEvent me)
    {
mouseX=me.getX();
mouseY=me.getY();
msg="up.";
repaint();
    }
    public void mouseDragged(MouseEvent me)
{
mouseX=me.getX();
mouseY=me.getY();
msg="*";
showStatus("Dragging mouse at "+mouseX+","+mouseY);
repaint();
}

     public void mouseMoved(MouseEvent me)
     {
mouseX=me.getX();
mouseY=me.getY();
showStatus("Mouse is at "+mouseX+","+mouseY);
     }
   public void paint(Graphics g) {
      g.drawString(msg,mouseX,mouseY);
   }
}
```