Unit-V-PART-I

UNIT – V

Event Handling : Events, Event sources, Event Listeners, Event classes, Event listener interface, Handling mouse and keyboard events, Adapter classes, The AWT class hierarchy, AWT controls- labels, buttons, scrollbars, text field, check box, check box groups, choices, handling lists, dialogs, Menubar, layout manager- Flow, Border, Grid, Card

Event Handling

Any program that uses GUI (graphical user interface) such as Java application written for windows, is event driven.

Event describes the change in state of any object.

For Example : Pressing a button, Entering a character in Textbox, Clicking or Dragging a mouse, etc.

An event can be defined as changing the state of an object or behavior by performing actions.

Actions can be a button click, cursor movement, keypress through keyboard or page scrolling, etc.

The java.awt.event package can be used to provide various event classes.

Classification of Events

Foreground Events

Background Events

Types of Events

1. Foreground Events

Foreground events are the events that require user interaction to generate, i.e., foreground events are generated due to interaction by the user on components in Graphic User Interface (GUI).

Interactions are nothing but clicking on a button, scrolling the scroll bar, cursor moments, etc.

2. Background Events

Events that don’t require interactions of users to generate are known as background events.

Examples of these events are operating system failures/interrupts, operation completion, etc.

Event Handling

It is a mechanism to control the events and to decide what should happen after an event occur.

To handle the events, Java follows the Delegation Event model.

Delegation Event model

It has Sources and Listeners.

Delegation Event Model

Source: Events are generated from the source.

There are various sources like buttons, checkboxes, list, menu-item, choice, scrollbar, text components, windows, etc., to generate events.

Listeners: Listeners are used for handling the events generated from the source.

Each of these listeners represents interfaces that are responsible for handling events.

Components of Event Handling

Event handling has three main components,

- **Events :** An event is a change in state of an object.
- **Events Source :** Event source is an object that generates an event.
- **Listeners :** A listener is an object that listens to the event. A listener gets notified when an event occurs.

How Events are handled?

A source generates an Event and send it to one or more listeners registered with the source.

Once event is received by the listener, they process the event and then return.

Events are supported by a number of Java packages, like **java.util**, **java.awt** and **java.awt.event**.

Important Event Classes and Interface

Event Classes	Description	Listener Interface
ActionEvent	generated when button is pressed, menu-item is selected, list-item is double clicked	ActionListener
MouseEvent	generated when mouse is dragged, moved,clicked,pressed or released and also when it enters or exit a component	MouseListener
KeyEvent	generated when input is received from keyboard	KeyListener
ItemEvent	generated when check-box or list item is clicked	ItemListener
TextEvent	generated when value of textarea or textfield is changed	TextListener
MouseWheelEvent	generated when mouse wheel is moved	MouseWheelListener
WindowEvent	generated when window is activated, deactivated, deiconified, iconified, opened or closed	WindowListener
ComponentEvent	generated when component is hidden, moved, resized or set visible	ComponentEventListener
ContainerEvent	generated when component is added or removed from container	ContainerListener
AdjustmentEvent	generated when scroll bar is manipulated	AdjustmentListener
FocusEvent	generated when component gains or loses keyboard focus	FocusListener

Example of Event Handling

```
import java.awt.*;

import java.awt.event.*;

import java.applet.*;

import java.applet.*;
```

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

public class Test extends Applet implements KeyListener
{
	String msg="";
	public void init()
	{
		addKeyListener(this);
	}
	public void keyPressed(KeyEvent k)
	{
		showStatus("KeyPressed");
	}
	public void keyReleased(KeyEvent k)
	{
		showStatus("KeyRealesed");
	}
	public void keyTyped(KeyEvent k)
```

```
        {
                msg = msg+k.getKeyChar();
                repaint();
        }
        public void paint(Graphics g)
        {
                g.drawString(msg, 20, 40);
        }
}
```

HTML code:

```
<applet code="Test" width=300, height=100>
</applet>
```

o/p:-

- Event Listener:-

  Listeners : A listener is an object that listens to the event. A listener gets notified when an event occurs.

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().

actionPerformed() method

The actionPerformed() method is invoked automatically whenever you click on the registered component.

1. **public abstract void** actionPerformed(ActionEvent e);

How to write ActionListener

The common approach is to implement the ActionListener.

If you implement the ActionListener class, you need to follow 3 steps:

1) Implement the ActionListener interface in the class:

1. **public class** ActionListenerExample Implements ActionListener

2) Register the component with the Listener:

```
1. component.addActionListener(instanceOfListenerclass);
```

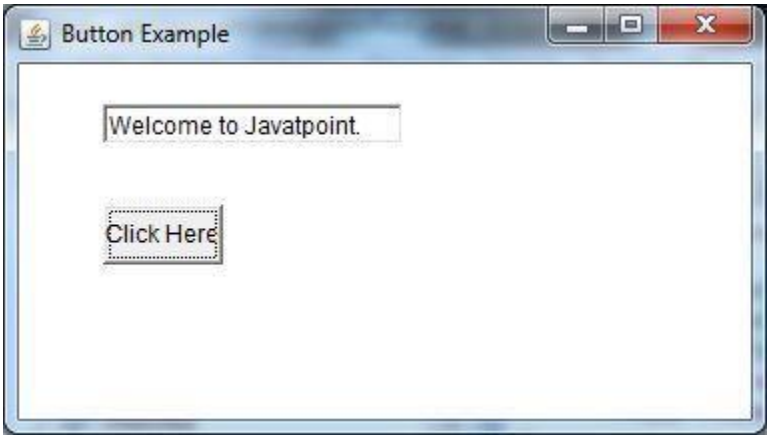
3) Override the actionPerformed() method:

```
1. public void actionPerformed(ActionEvent e)
2. {
3.         //Write the code here
4. }
```

Java ActionListener Example: On Button click

```
1. import java.awt.*;
2. import java.awt.event.*;
3. //1st step
4. public class ActionListenerExample implements ActionListener
5. {
6. public static void main(String[] args)
7.  {
8.     Frame f=new Frame("ActionListener Example");
9.     final TextField tf=new TextField();
10.    tf.setBounds(50,50, 150,20);
11.    Button b=new Button("Click Here");
12.    b.setBounds(50,100,60,30);
13.    //2nd step
```

```
14.     b.addActionListener(this);
15.     f.add(b);f.add(tf);
16.     f.setSize(400,400);
17.     f.setLayout(null);
18.     f.setVisible(true);
19. }
20. //3rd step
21. public void actionPerformed(ActionEvent e)
22. {
23.             tf.setText("Welcome to Javatpoint.");
24. }
25. }
```



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.

Methods of MouseListener interface

The signature of 5 methods found in MouseListener interface are given below:

1. `public abstract void` mouseClicked(MouseEvent e);
2. `public abstract void` mouseEntered(MouseEvent e);

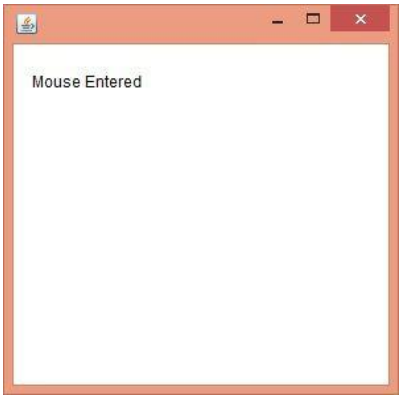
```
3. public abstract void mouseExited(MouseEvent e);
4. public abstract void mousePressed(MouseEvent e);
5. public abstract void mouseReleased(MouseEvent e);
```

Java MouseListener Example

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class MouseListenerExample extends Frame implements MouseListener
4. {
5.     Label l;
6.     MouseListenerExample()
7.     {
8.         addMouseListener(this);
9.         l=new Label();
10.        l.setBounds(20,50,100,20);
11.        add(l);
12.        setSize(300,300);
13.        setLayout(null);
14.        setVisible(true);
15.    }
16.    public void mouseClicked(MouseEvent e)
17.    {
18.        l.setText("Mouse Clicked");
19.    }
20.    public void mouseEntered(MouseEvent e)
21.    {
22.        l.setText("Mouse Entered");
23.    }
24.    public void mouseExited(MouseEvent e)
25.    {
26.        l.setText("Mouse Exited");
27.    }
28.    public void mousePressed(MouseEvent e) {
```

```
29.         l.setText("Mouse Pressed");
30.     }
31.     public void mouseReleased(MouseEvent e)
32.    {
33.         l.setText("Mouse Released");
34.     }
35.   public static void main(String[] args)
36.   {
37.     new MouseListenerExample();
38. }
39. }
```

Output:



MouseMotionListener Interface

The Java MouseMotionListener is notified whenever you move or drag mouse. It is notified against MouseEvent. The MouseMotionListener interface is found in java.awt.event package. It has two methods.

Methods of MouseMotionListener interface

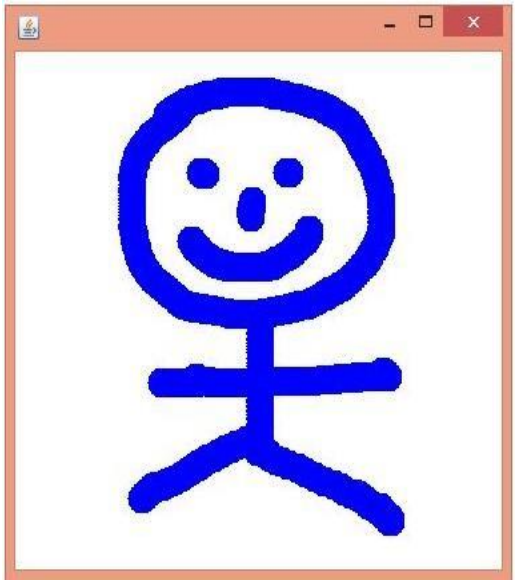
The signature of 2 methods found in MouseMotionListener interface are given below:

1. `public abstract void mouseDragged(MouseEvent e);`
2. `public abstract void mouseMoved(MouseEvent e);`

Java MouseMotionListener Example

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class MouseMotionListenerExample extends Frame implements MouseM
   otionListener{
4.     MouseMotionListenerExample(){
5.         addMouseMotionListener(this);
6.
7.         setSize(300,300);
8.         setLayout(null);
9.         setVisible(true);
10.    }
11.  public void mouseDragged(MouseEvent e)
12. {
13.    Graphics g=getGraphics();
14.    g.setColor(Color.BLUE);
15.    g.fillOval(e.getX(),e.getY(),20,20);
16. }
17. public void mouseMoved(MouseEvent e)
18.  {
19. }
20.
21. public static void main(String[] args)
22.  {
23.    new MouseMotionListenerExample();
24. }
25. }
```

Output:



ItemListener Interface

The Java ItemListener is notified whenever you click on the checkbox. It is notified against ItemEvent. The ItemListener interface is found in java.awt.event package. It has only one method: itemStateChanged().

itemStateChanged() method

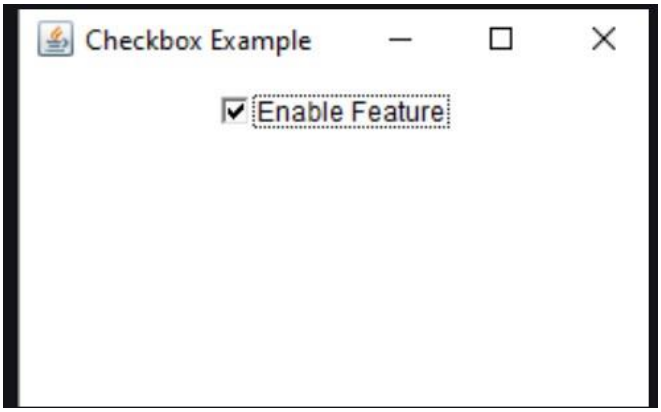
The itemStateChanged() method is invoked automatically whenever you click or unclick on the registered checkbox component.

1. **public abstract void** itemStateChanged(ItemEvent e);

```
// Java AWT Program to demonstrate

// Java ItemListener

import java.awt.*;

import java.awt.event.*;

// Driver Class
```

```
class CheckboxExample
{
        public static void main(String[] args)
       {
                // Create a new frame with a title.
                Frame frame = new Frame("Checkbox Example");
                // Create a checkbox labeled "Enable Feature."
                Checkbox checkbox = new Checkbox("Enable Feature");
                // Create an instance of MyItemListener to handle checkbox events.
                MyItemListener itemListener = new MyItemListener();
                // Register the itemListener with the checkbox to listen for events.
                checkbox.addItemListener(itemListener);
                // Add the checkbox to the frame.
                frame.add(checkbox);
                // Set the frame size and layout.
                frame.setSize(300, 200);
                frame.setLayout(new FlowLayout());
                // Make the frame visible.
                frame.setVisible(true);
        }
}
class MyItemListener implements ItemListener
{
        public void itemStateChanged(ItemEvent e)
{
                if (e.getStateChange() == ItemEvent.SELECTED)
               {
                        // Respond when the checkbox is selected (checked).
                        System.out.println("Feature is enabled.");
```

```
                }
                else
                {
                        // Respond when the checkbox is deselected (unchecked).
                        System.out.println("Feature is disabled.");
                }
        }
}
```



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, and it has three methods.

Interface declaration

Following is the declaration for java.awt.event.KeyListener interface:

public interface KeyListener extends EventListener

Methods of KeyListener interface

The signature of 3 methods found in KeyListener interface are given below:

Sr. no.	Method name	Description
1.	public abstract void keyPressed (KeyEvent e);	It is invoked when a key has been pressed.

2.	public abstract void keyReleased (KeyEvent e);	It is invoked when a key has been released.
3.	public abstract void keyTyped (KeyEvent e);	It is invoked when a key has been typed.

KeyListener Example

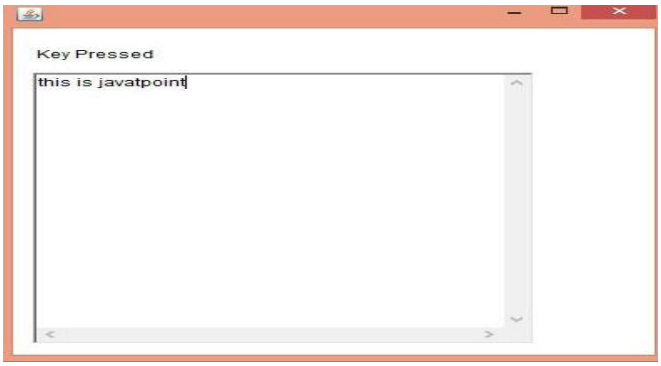
In the following example, we are implementing the methods of the KeyListener interface.

KeyListenerExample.java

```
1. // importing awt libraries
2. import java.awt.*;
3. import java.awt.event.*;
4. public class KeyListenerExample extends Frame implements KeyListener
5.  {
6.       Label l;
7.      TextArea area;
8.  // class constructor
9.      KeyListenerExample()
10.    {
11.        // creating the label
12.        l = new Label();
13. // setting the location of the label in frame
14.        l.setBounds (20, 50, 100, 20);
15. // creating the text area
16.        area = new TextArea();
17. // setting the location of text area
18.        area.setBounds (20, 80, 300, 300);
19. // adding the KeyListener to the text area
20.        area.addKeyListener(this);
21. // adding the label and text area to the frame
```

```
22.         add(l);
23. add(area);
24. // setting the size, layout and visibility of frame
25.         setSize (400, 400);
26.         setLayout (null);
27.         setVisible (true);
28.     }
29. // overriding the keyPressed() method of KeyListener interface where we set the t
    ext of the label when key is pressed
30.     public void keyPressed (KeyEvent e)
31.     {
32.         l.setText ("Key Pressed");
33.     }
34. // overriding the keyReleased() method of KeyListener interface where we set the
    text of the label when key is released
35.     public void keyReleased (KeyEvent e)
36. {
37.         l.setText ("Key Released");
38.     }
39. // overriding the keyTyped() method of KeyListener interface where we set the tex
    t of the label when a key is typed
40.     public void keyTyped (KeyEvent e)
41. {
42.         l.setText ("Key Typed");
43.     }
44.   // main method
45.     public static void main(String[] args)
46. {
47.         new KeyListenerExample();
48.     }
49. }
```

Output:



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. It has three methods.

WindowListener interface declaration

The declaration for java.awt.event.WindowListener interface is shown below:

```
1. public interface WindowListener extends EventListener
```

Methods of WindowListener interface

WindowExample.java

```
1. // importing necessary libraries of awt
2. import java.awt.*;
3. import java.awt.event.WindowEvent;
4. import java.awt.event.WindowListener;
5.
6. // class which inherits Frame class and implements WindowListener interface
7. public class WindowExample extends Frame implements WindowListener {
8.
9. // class constructor
10.     WindowExample() {
```

```
11.
12.     // adding WindowListener to the frame
13.         addWindowListener(this);
14.         // setting the size, layout and visibility of frame
15.         setSize (400, 400);
16.         setLayout (null);
17.         setVisible (true);
18.     }
19. // main method
20. public static void main(String[] args) {
21.     new WindowExample();
22. }
23.
24. // overriding windowActivated() method of WindowListener interface which prints
     the given string when window is set to be active
25. public void windowActivated (WindowEvent arg0) {
26.     System.out.println("activated");
27. }
28.
29. // overriding windowClosed() method of WindowListener interface which prints t
    he given string when window is closed
30. public void windowClosed (WindowEvent arg0) {
31.     System.out.println("closed");
32. }
33.
34. // overriding windowClosing() method of WindowListener interface which prints t
    he given string when we attempt to close window from system menu
35. public void windowClosing (WindowEvent arg0) {
36.     System.out.println("closing");
37.     dispose();
38. }
39.
40. // overriding windowDeactivated() method of WindowListener interface which pri
    nts the given string when window is not active
```

```
41. public void windowDeactivated (WindowEvent arg0) {
42.     System.out.println("deactivated");
43. }
44.
45. // overriding windowDeiconified() method of WindowListener interface which pri
    nts the given string when window is modified from minimized to normal state
46. public void windowDeiconified (WindowEvent arg0) {
47.     System.out.println("deiconified");
48. }
49.
50. // overriding windowIconified() method of WindowListener interface which prints
    the given string when window is modified from normal to minimized state
51. public void windowIconified(WindowEvent arg0) {
52.     System.out.println("iconified");
53. }
54.
55. // overriding windowOpened() method of WindowListener interface which prints t
    he given string when window is first opened
56. public void windowOpened(WindowEvent arg0) {
57.     System.out.println("opened");
58. }
59. }
```

Output:

Adapter Classes

Java adapter classes *provide the default implementation of listener interfaces*.

If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces.

So it *saves code*.

-
- It assists the unrelated classes to work combinedly.
- It provides ways to use classes in different ways.
- It increases the transparency of classes.
- It provides a way to include related patterns in the class.
- It provides a pluggable kit for developing an application.
- It increases the reusability of the class.

ava.awt.event Adapter classes

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener

WindowAdapter Example

In the following example, we are implementing the WindowAdapter class of AWT and one its methods windowClosing() to close the frame window.

AdapterExample.java

```
1.  // importing the necessary libraries
2.  import java.awt.*;
3.  import java.awt.event.*;
4.
5.  public class AdapterExample {
6.  // object of Frame
7.      Frame f;
8.  // class constructor
9.      AdapterExample() {
10. // creating a frame with the title
11.         f = new Frame ("Window Adapter");
12. // adding the WindowListener to the frame
13. // overriding the windowClosing() method
14.         f.addWindowListener (new WindowAdapter() {
15.             public void windowClosing (WindowEvent e) {
16.                 f.dispose();
17.             }
```

```
18.         });
19.           // setting the size, layout and
20.         f.setSize (400, 400);
21.          f.setLayout (null);
22.         f.setVisible (true);
23.     }
24.
25. // main method
26. public static void main(String[] args) {
27.     new AdapterExample();
28. }
29. }
```

Output: