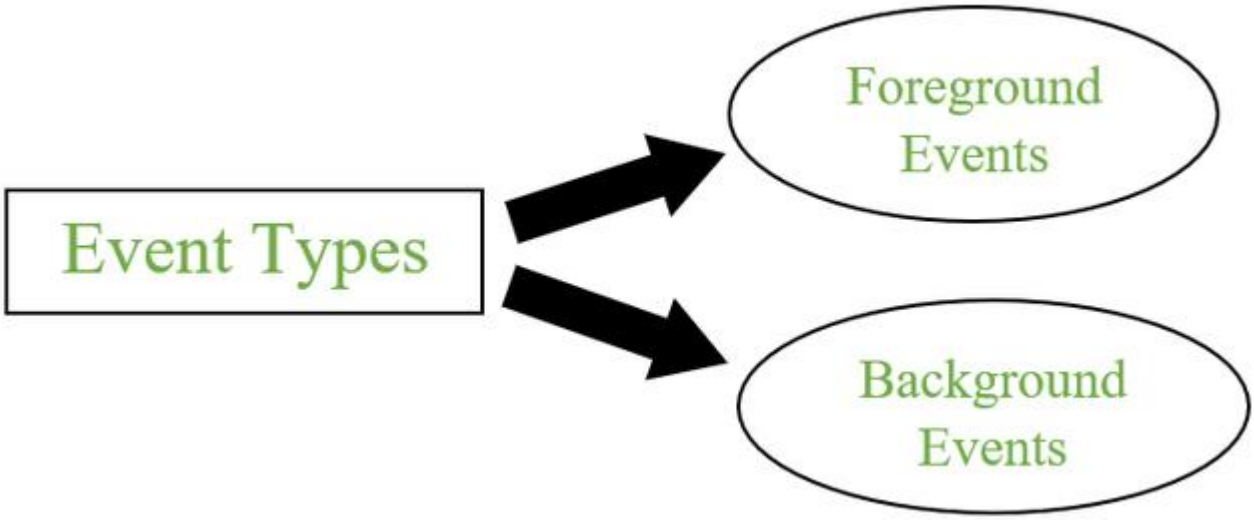
Event Handling in Java

- 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

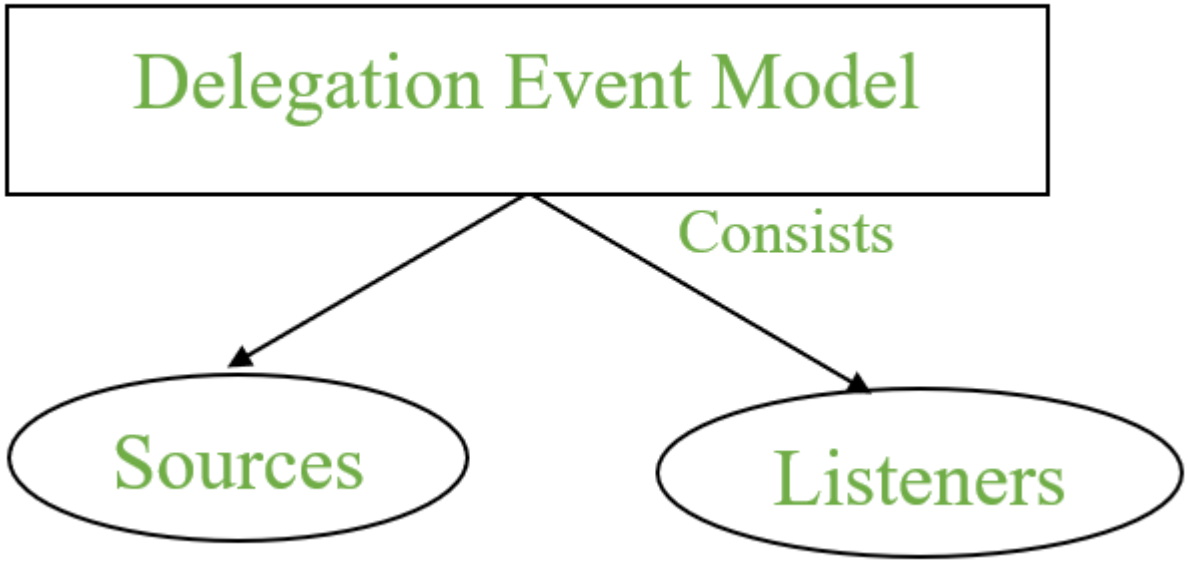
Events that don't require interactions of users to generate are known as background events. Examples of these events are operating system failures/interrupts, operation completion, etc.

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.

To perform Event Handling, we need to register the source with the listener.

Registering the Source With Listener

Different Classes provide different registration methods.

Syntax:

```
addTypeListener()
```

where Type represents the type of event.

Example 1: For **KeyEvent** we use *addKeyListener()* to register.

Example 2:that For **ActionEvent** we use *addActionListener()* to register.

Event Classes in Java

Event Class	Listener Interface	Description
ActionEvent	ActionListener	An event that indicates that a component-defined action occurred like a button click or selecting an item from the menu-item list.
AdjustmentEvent	AdjustmentListener	The adjustment event is emitted by an Adjustable object like Scrollbar.
ComponentEvent	ComponentListener	An event that indicates that a component moved, the size changed or changed its visibility.
ContainerEvent	ContainerListener	When a component is added to a container (or) removed from it, then this event is generated by a container object.
FocusEvent	FocusListener	These are focus-related events, which include focus, focusin, focusout, and blur.
ItemEvent	ItemListener	An event that indicates whether an item was selected or not.
KeyEvent	KeyListener	An event that occurs due to a sequence of keypresses on the keyboard.
MouseEvent	MouseListener & MouseMotionListener	The events that occur due to the user interaction with the mouse (Pointing Device).
MouseWheelEvent	MouseWheelListener	An event that specifies that the mouse wheel was rotated in a component.
TextEvent	TextListener	An event that occurs when an object's text changes.
WindowEvent	WindowListener	An event which indicates whether a

Event Class	Listener Interface	Description
		window has changed its status or not.

Note: *As Interfaces contains abstract methods which need to implemented by the registered class to handle events.*

Different interfaces consists of different methods which are specified below.

Listener Interface	Methods
ActionListener	• actionPerformed()
AdjustmentListener	• adjustmentValueChanged()
ComponentListener	• componentResized() • componentShown() • componentMoved() • componentHidden()
ContainerListener	• componentAdded() • componentRemoved()
FocusListener	• focusGained() • focusLost()
ItemListener	• itemStateChanged()
KeyListener	• keyTyped() • keyPressed() • keyReleased()
MouseListener	• mousePressed() • mouseClicked() • mouseEntered() • mouseExited() • mouseReleased()
MouseMotionListener	• mouseMoved()

Listener Interface	Methods
	• mouseDragged()
MouseWheelListener	• mouseWheelMoved()
TextListener	• textChanged()
WindowListener	• windowActivated() • windowDeactivated() • windowOpened() • windowClosed() • windowClosing() • windowIconified() • windowDeiconified()

Flow of Event Handling

1. User Interaction with a component is required to generate an event.
2. The object of the respective event class is created automatically after event generation, and it holds all information of the event source.
3. The newly created object is passed to the methods of the registered listener.
4. The method executes and returns the result.

Code-Approaches

The three approaches for performing event handling are by placing the event handling code in one of the below-specified places.

1. Within Class
2. Other Class
3. Anonymous Class

Java 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: **public class** ActionListenerExample Implements ActionListener

2) Register the component with the Listener:

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

3) Override the actionPerformed() method:

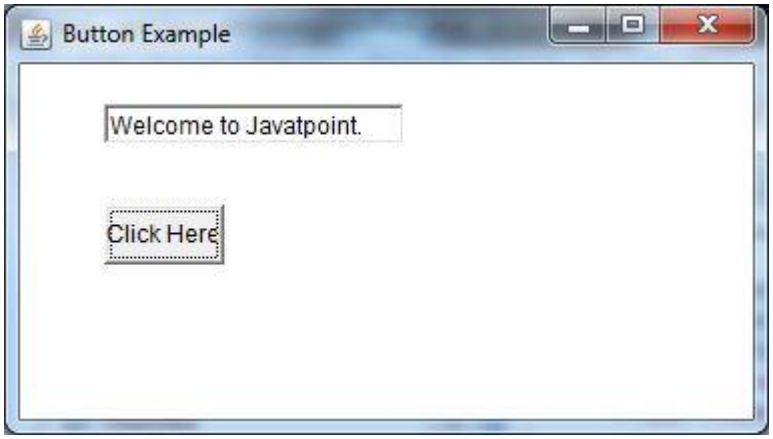
```
1. public void actionPerformed(ActionEvent e){
2.         //Write the code here
3. }
```

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. public static void main(String[] args) {
6.     Frame f=new Frame("ActionListener Example");
7.     final TextField tf=new TextField();
8.     tf.setBounds(50,50, 150,20);
9.     Button b=new Button("Click Here");
10.    b.setBounds(50,100,60,30);
11.    //2nd step
12.    b.addActionListener(this);
13.    f.add(b);f.add(tf);
14.    f.setSize(400,400);
```

```
15.     f.setLayout(null);
16.     f.setVisible(true);
17. }
18. //3rd step
19. public void actionPerformed(ActionEvent e){
20.             tf.setText("Welcome to Javatpoint.");
21. }
22. }
```

Output:



Java ActionListener Example: Using Anonymous class

We can also use the anonymous class to implement the ActionListener. It is the shorthand way, so you do not need to follow the 3 steps:

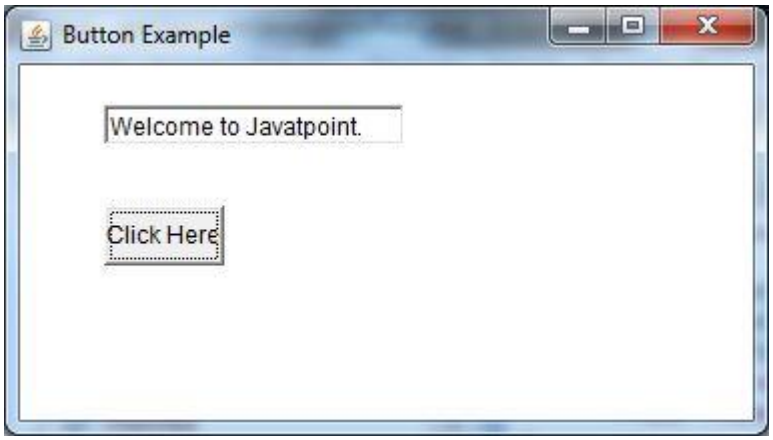
```
1. b.addActionListener(new ActionListener(){
2. public void actionPerformed(ActionEvent e){
3.             tf.setText("Welcome to Javatpoint.");
4. }
5. });
```

Let us see the full code of ActionListener using anonymous class.

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class ActionListenerExample {
4. public static void main(String[] args) {
```

```
5.      Frame f=new Frame("ActionListener Example");
6.      final TextField tf=new TextField();
7.      tf.setBounds(50,50, 150,20);
8.      Button b=new Button("Click Here");
9.      b.setBounds(50,100,60,30);
10.
11.     b.addActionListener(new ActionListener(){
12.     public void actionPerformed(ActionEvent e){
13.             tf.setText("Welcome to Javatpoint.");
14.     }
15.     });
16.     f.add(b);f.add(tf);
17.     f.setSize(400,400);
18.     f.setLayout(null);
19.     f.setVisible(true);
20. }
21. }
```

Output:



Java 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:

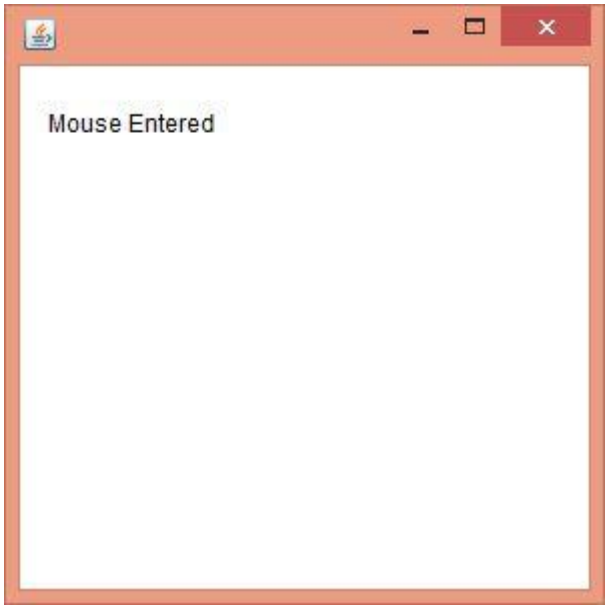
```
1. public abstract void mouseClicked(MouseEvent e);
2. public abstract void mouseEntered(MouseEvent e);
3. public abstract void mouseExited(MouseEvent e);
4. public abstract void mousePressed(MouseEvent e);
5. public abstract void mouseReleased(MouseEvent e);
```

Java MouseListener Example

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class MouseListenerExample extends Frame implements MouseListener{
4.     Label l;
5.     MouseListenerExample(){
6.         addMouseListener(this);
7.
8.         l=new Label();
9.         l.setBounds(20,50,100,20);
10.        add(l);
11.        setSize(300,300);
12.        setLayout(null);
13.        setVisible(true);
14.    }
15.    public void mouseClicked(MouseEvent e) {
16.        l.setText("Mouse Clicked");
17.    }
18.    public void mouseEntered(MouseEvent e) {
19.        l.setText("Mouse Entered");
20.    }
21.    public void mouseExited(MouseEvent e) {
22.        l.setText("Mouse Exited");
23.    }
24.    public void mousePressed(MouseEvent e) {
25.        l.setText("Mouse Pressed");
26.    }
27.    public void mouseReleased(MouseEvent e) {
28.        l.setText("Mouse Released");
29.    }
30. public static void main(String[] args) {
```

```
31.     new MouseListenerExample();
32. }
33. }
```

Output:

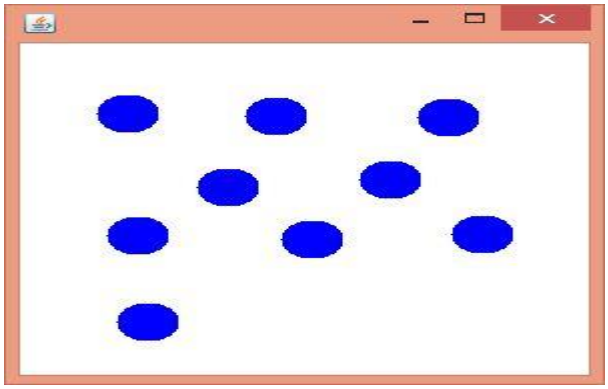


Java MouseListener Example 2

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.  public class MouseListenerExample2 extends Frame implements MouseListener{
4.     MouseListenerExample2(){
5.         addMouseListener(this);
6.
7.         setSize(300,300);
8.         setLayout(null);
9.         setVisible(true);
10.    }
11.    public void mouseClicked(MouseEvent e) {
12.        Graphics g=getGraphics();
13.        g.setColor(Color.BLUE);
14.        g.fillOval(e.getX(),e.getY(),30,30);
15.    }
16.    public void mouseEntered(MouseEvent e) {}
```

```
17.     public void mouseExited(MouseEvent e) {}
18.     public void mousePressed(MouseEvent e) {}
19.     public void mouseReleased(MouseEvent e) {}
20.
21. public static void main(String[] args) {
22.     new MouseListenerExample2();
23. }
24. }
```

Output:



Example to understand AWT Button Control in Java:

```
import java.awt.*;
import java.awt.event.*;
public class ButtonDemo extends Frame
{
Button b1, b2;
ButtonDemo ()
{
b1 = new Button ("OK");
b2 = new Button ("CANCEL");
this.setLayout (null);
b1.setBounds (100, 100, 80, 40);;
b2.setBounds (200, 100, 80, 40);
this.add (b1);
this.add (b2);
this.setVisible (true);
this.setSize (300, 300);
this.setTitle ("button");
```

```
this.addWindowListener (new WindowAdapter ()
{
public void windowClosing (WindowEvent we)
{
System.exit (0);
}
});
}
public static void main (String args[])
{
new ButtonDemo ();
}
}
```

Output: