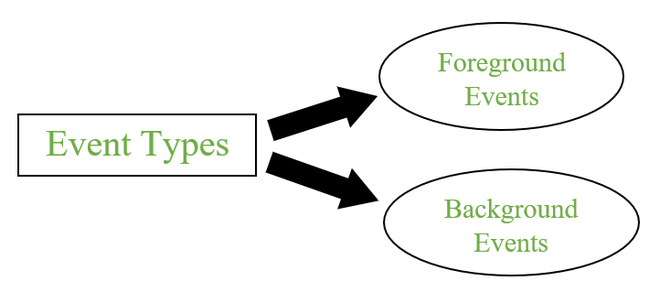
**Event Handling in Java**

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**

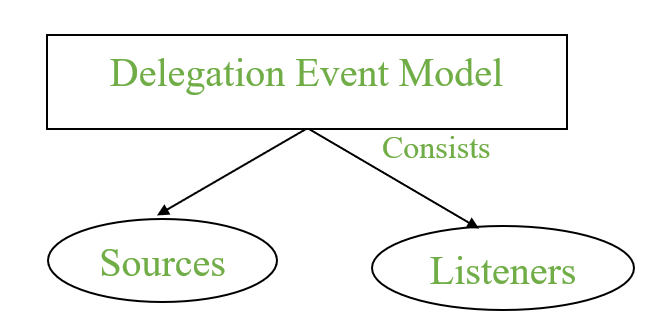
Events that don’t require interactions of users to generate are known as background events. Examples of these events are operating system failures/interrupts, operation completion, etc.

**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.

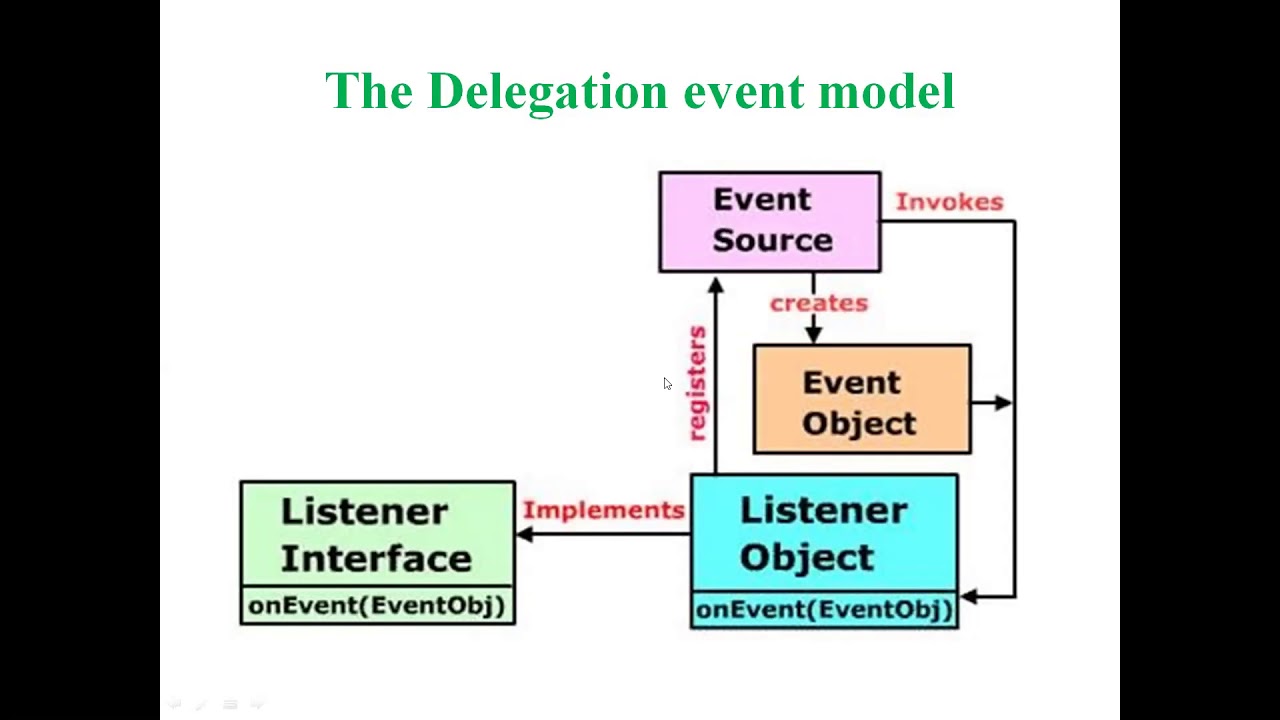
Syntax:

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 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() * 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.

**Handling mouse events-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**

import java.awt.\*;

import java.awt.event.\*;

public class MouseListenerExample extends Frame implements MouseListener{

    Label l;

    MouseListenerExample(){

        addMouseListener(this);

        l=new Label();

        l.setBounds(20,50,100,20);

        add(l);

        setSize(300,300);

        setLayout(null);

        setVisible(true);

    }

    public void mouseClicked(MouseEvent e) {

        l.setText("Mouse Clicked");

    }

    public void mouseEntered(MouseEvent e) {

        l.setText("Mouse Entered");

    }

    public void mouseExited(MouseEvent e) {

        l.setText("Mouse Exited");

    }

    public void mousePressed(MouseEvent e) {

        l.setText("Mouse Pressed");

    }

    public void mouseReleased(MouseEvent e) {

        l.setText("Mouse Released");

    }

public static void main(String[] args) {

    new MouseListenerExample();

}

}

**Handling key events-Java 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. |

import java.awt.\*;

import java.awt.event.\*;

public class KeyListenerExample extends Frame implements KeyListener

{

 Label l;

     TextArea area;

// class constructor

     KeyListenerExample()

{

            // creating the label

         l = new Label();

// setting the location of the label in frame

          l.setBounds (20, 50, 100, 20);

// creating the text area

       area = new TextArea();

// setting the location of text area

      area.setBounds (20, 80, 300, 300);

// adding the KeyListener to the text area

       area.addKeyListener(this);

// adding the label and text area to the frame

       add(l);

add(area);

// setting the size, layout and visibility of frame

         setSize (400, 400);

         setLayout (null);

         setVisible (true);

     }

public void keyPressed (KeyEvent e)

{

         l.setText ("Key Pressed");

     }

public void keyReleased (KeyEvent e)

{

         l.setText ("Key Released");

     }

 public void keyTyped (KeyEvent e)

{

         l.setText ("Key Typed");

   }

   // main method

     public static void main(String[] args)

{

         new KeyListenerExample();

     }

}

**Java Adapter Classes**

Java adapter classes provide the default implementation of listener [interfaces](https://www.javatpoint.com/interface-in-java). 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.

### Pros of using Adapter classes:

* 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.

The adapter classes are found in java.awt.event.The Adapter classes with their corresponding listener interfaces are given below.

## java.awt.event Adapter classes

|  |  |
| --- | --- |
| **Adapter class** | **Listener interface** |
| WindowAdapter | [WindowListener](https://www.javatpoint.com/java-windowlistener) |
| KeyAdapter | [KeyListener](https://www.javatpoint.com/java-keylistener) |
| MouseAdapter | [MouseListener](https://www.javatpoint.com/java-mouselistener) |
| MouseMotionAdapter | [MouseMotionListener](https://www.javatpoint.com/java-mousemotionlistener) |
| FocusAdapter | FocusListener |
| ComponentAdapter | ComponentListener |
| ContainerAdapter | ContainerListener |
| HierarchyBoundsAdapter | HierarchyBoundsListener |

## Java MouseAdapter Example

In the following example, we are implementing the MouseAdapter class. The MouseListener interface is added into the frame to listen the mouse event in the frame.

**MouseAdapterExample.java**

// importing the necessary libraries

**import** java.awt.\*;

**import** java.awt.event.\*;

// class which inherits the MouseAdapter class

**public** **class** MouseAdapterExample **extends** MouseAdapter

{

// object of Frame class

     Frame f;

// class constructor

     MouseAdapterExample()

{

// creating the frame with the title

         f = **new** Frame ("Mouse Adapter");

// adding MouseListener to the Frame

         f.addMouseListener(**this**);

  // setting the size, layout and visibility of the frame

         f.setSize (300, 300);

      f.setLayout (**null**);

         f.setVisible (**true**);

    }

// overriding the mouseClicked() method of the MouseAdapter class

**public** **void** mouseClicked (MouseEvent e)

{

// creating the Graphics object and fetching them from the Frame object using getGraphics() method

  Graphics g = f.getGraphics();

// setting the color of graphics object

         g.setColor (Color.BLUE);

// setting the shape of graphics object

     g.fillOval (e.getX(), e.getY(), 30, 30);

    }

  // main method

**public** **static** **void** main(String[] args)

{

**new** MouseAdapterExample();

}

}

\*A frame, implemented as an instance of the JFrame class, is a window that has decorations such as a border, a title, and supports button components that close or iconify the window.

\*The window iconified event. This event is delivered when the window has been changed from a normal to a minimized state.

\*Graphics is an abstract class provided by Java AWT which is used to draw or paint on the components. It consists of various fields which hold information like components to be painted, font, color, XOR mode, etc., and methods that allow drawing various shapes on the GUI components. Graphics is an abstract class and thus cannot be initialized directly. Objects of its child classes can be obtained in the following two ways.

**Inside paint() or update() method**

## Java KeyAdapter Example

In the following example, we are implementing the KeyAdapter class and its method.

**KeyAdapterExample.java**

// importing the necessary libraries

**import** java.awt.\*;

**import** java.awt.event.\*;

// class which inherits the KeyAdapter class

**public** **class** KeyAdapterExample **extends** KeyAdapter

{

// creating objects of Label, TextArea and Frame

     Label l;

     TextArea area;

     Frame f;

// class constructor

     KeyAdapterExample()

{

// creating the Frame with title

         f = **new** Frame ("Key Adapter");

// creating the Label

         l = **new** Label();

// setting the location of label

         l.setBounds (20, 50, 200, 20);

// creating the text area

         area = **new** TextArea();

// setting the location of text area

         area.setBounds (20, 80, 300, 300);

// adding KeyListener to text area

         area.addKeyListener(**this**);

 // adding the label and text area to frame

         f.add(l);

f.add(area);

// setting the size, layout and visibility of frame

         f.setSize (400, 400);

         f.setLayout (**null**);

         f.setVisible (**true**);

     }

// overriding the keyReleased() method

**public** **void** keyReleased (KeyEvent e)

{

// creating the String object to get the text fromTextArea

         String text = area.getText();

// splitting the String into words

         String words[] = text.split ("\\s");

// setting the label text to print the number of words and characters of given string

         l.setText ("Words: " + words.length + " Characters:" + text.length());

     }

   // main method

**public** **static** **void** main(String[] args) {

**new** KeyAdapterExample();

     }

}