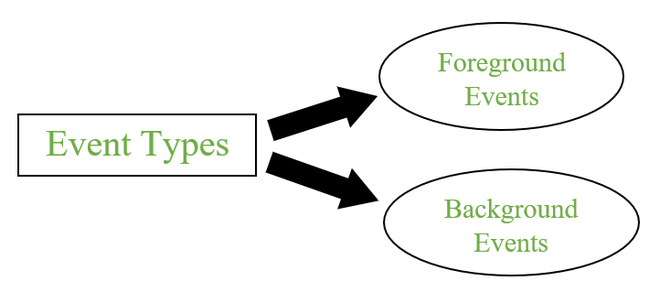
**Event Handling in Java**

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



**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**

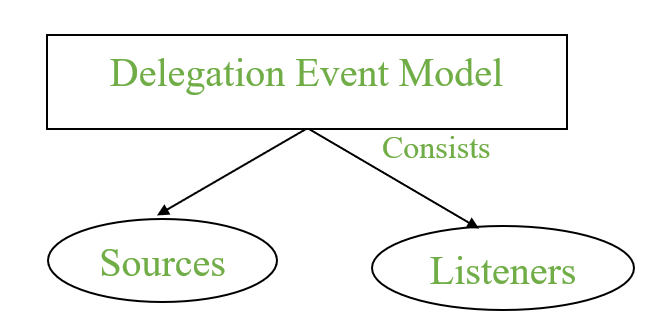
Events that don’t require interactions of users to generate are known as background events. Examples of these events are operating system failures/interrupts, operation completion, etc.

## **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.



* **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. |

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.

### **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

### **Event Handling Within Class**

// Java program to demonstrate the

// event handling within the class

import java.awt.\*;

import java.awt.event.\*;

class action1 extends Frame implements ActionListener {

TextField textField;

action1()

{

// Component Creation

textField = new TextField();

// setBounds method is used to provide

// position and size of the component

textField.setBounds(60, 50, 180, 25);

Button button = new Button("click Here");

button.setBounds(100, 120, 80, 30);

// Registering component with listener

// this refers to current instance

button.addActionListener(this);

// add Components

add(textField);

add(button);

// set visibility

setVisible(true);

}

// implementing method of actionListener

public void actionPerformed(ActionEvent e)

{

// Setting text to field

textField.setText("IT Department!");

}

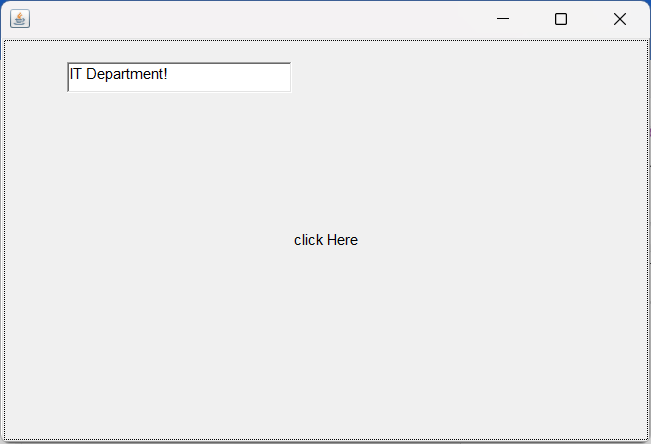
public static void main(String[] args) {

new action1();

}

}

**Output:**



#### Explanation

1. Firstly extend the class with the applet and implement the respective listener.
2. Create Text-Field and Button components.
3. Registered the button component with respective event. i.e. ActionEvent by addActionListener().
4. In the end, implement the abstract method.

### **Event Handling by Other Class**

// Java program to demonstrate the

// event handling by the other class

import java.awt.\*;

import java.awt.event.\*;

class GFG1 extends Frame {

TextField textField;

GFG2()

{

// Component Creation

textField = new TextField();

// setBounds method is used to provide

// position and size of component

textField.setBounds(60, 50, 180, 25);

Button button = new Button("click Here");

button.setBounds(100, 120, 80, 30);

Other other = new Other(this);

// Registering component with listener

// Passing other class as reference

button.addActionListener(other);

// add Components

add(textField);

add(button);

// set visibility

setVisible(true);

}

public static void main(String[] args)

{

new GFG2();

}

}

/// import necessary packages

import java.awt.event.\*;

// implements the listener interface

class Other implements ActionListener {

GFG2 gfgObj;

Other(GFG1 gfgObj)

{

this.gfgObj = gfgObj;

}

public void actionPerformed(ActionEvent e)

{

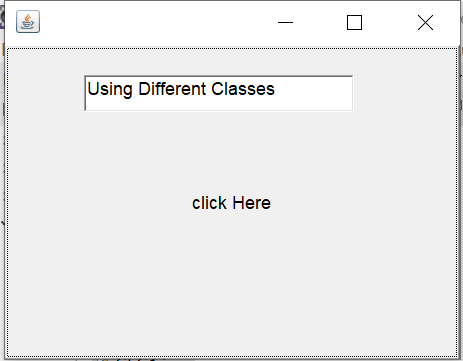
// setting text from different class

gfgObj.textField.setText("Using Different Classes");

}

}

Output:



### **Event Handling By Anonymous Class**

// Java program to demonstrate the

// event handling by the anonymous class

import java.awt.\*;

import java.awt.event.\*;

class GFG3 extends Frame {

TextField textField;

GFG3()

{

// Component Creation

textField = new TextField();

// setBounds method is used to provide

// position and size of component

textField.setBounds(60, 50, 180, 25);

Button button = new Button("click Here");

button.setBounds(100, 120, 80, 30);

// Registering component with listener anonymously

button.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

// Setting text to field

textField.setText("Anonymous");

}

});

// add Components

add(textField);

add(button);

// set visibility

setVisible(true);

}

public static void main(String[] args)

{

new GFG3();

}

}

Output:

