

What is an Event?

Change in the state of an object is known as event i.e. event describes the change in state of source. Events are generated as result of user interaction with the graphical user interface components. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page are the activities that causes an event to happen.

Types of Event

The events can be broadly classified into two categories:

- **Foreground Events** - Those events which require the direct interaction of user. They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page etc.
- **Background Events** - Those events that require the interaction of end user are known as background events. Operating system interrupts, hardware or software failure, timer expires, an operation completion are the example of background events.

What is Event Handling?

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism have 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. Let's have a brief introduction to this model.

The Delegation Event Model has the following key participants namely:

- **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 those 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.

Points to remember about listener

- In order to design a listener class we have to develop some listener interfaces. These Listener interfaces forecast some public abstract callback methods which must be implemented by

the listener class.

- If you do not implement the any if the predefined interfaces then your class can not act as a listener class for a source object.

Callback Methods

These are the methods that are provided by API provider and are defined by the application programmer and invoked by the application developer. Here the callback methods represents an event method. In response to an event java jre will fire callback method. All such callback methods are provided in listener interfaces.

If a component wants some listener will listen to it's events the the source must register itself to the listener.

Event Handling Example

Create the following java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingControlDemo.java

```
package com.tutorialspoint.gui;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class SwingControlDemo {

    private JFrame mainFrame;
    private JLabel headerLabel;
    private JLabel statusLabel;
    private JPanel controlPanel;

    public SwingControlDemo(){
        prepareGUI();
    }

    public static void main(String[] args){
        SwingControlDemo swingControlDemo = new SwingControlDemo();
        swingControlDemo.showEventDemo();
    }

    private void prepareGUI(){
        mainFrame = new JFrame("Java SWING Examples");
        mainFrame.setSize(400,400);
        mainFrame.setLayout(new GridLayout(3, 1));

        headerLabel = new JLabel("",JLabel.CENTER );
        statusLabel = new JLabel("",JLabel.CENTER);

        statusLabel.setSize(350,100);
        mainFrame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent windowEvent){
                System.exit(0);
            }
        });
        controlPanel = new JPanel();
        controlPanel.setLayout(new FlowLayout());

        mainFrame.add(headerLabel);
        mainFrame.add(controlPanel);
        mainFrame.add(statusLabel);
        mainFrame.setVisible(true);
    }

    private void showEventDemo(){
        headerLabel.setText("Control in action: Button");
    }
}
```

```

JButton okButton = new JButton("OK");
JButton submitButton = new JButton("Submit");
JButton cancelButton = new JButton("Cancel");

okButton.setActionCommand("OK");
submitButton.setActionCommand("Submit");
cancelButton.setActionCommand("Cancel");

okButton.addActionListener(new ButtonClickListener());
submitButton.addActionListener(new ButtonClickListener());
cancelButton.addActionListener(new ButtonClickListener());

controlPanel.add(okButton);
controlPanel.add(submitButton);
controlPanel.add(cancelButton);

mainFrame.setVisible(true);
}

private class ButtonClickListener implements ActionListener{
    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();
        if( command.equals( "OK" ) ) {
            statusLabel.setText("Ok Button clicked.");
        }
        else if( command.equals( "Submit" ) ) {
            statusLabel.setText("Submit Button clicked.");
        }
        else {
            statusLabel.setText("Cancel Button clicked.");
        }
    }
}
}

```

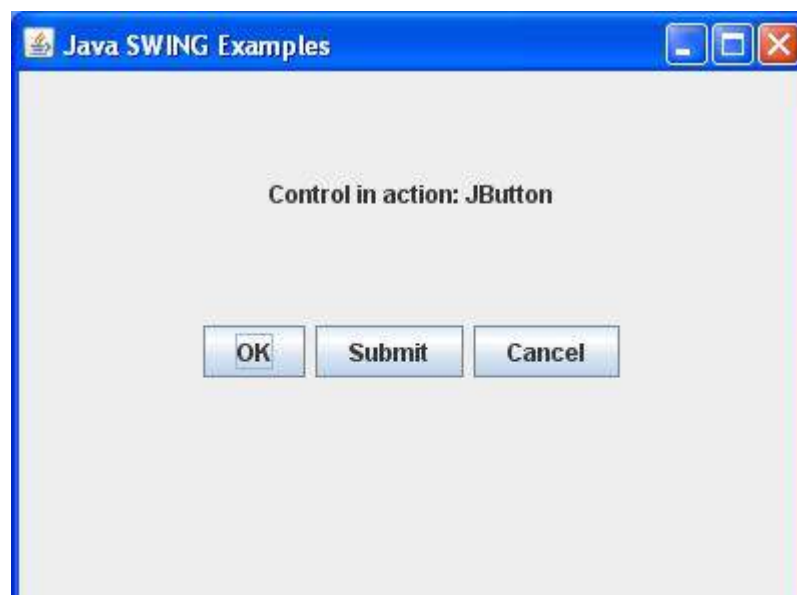
Compile the program using command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\AWT>javac com\tutorialspoint\gui\SwingControlDemo.java
```

If no error comes that means compilation is successful. Run the program using following command.

```
D:\AWT>java com.tutorialspoint.gui.SwingControlDemo
```

Verify the following output



Ok Button clicked.