**The Delegation Event Model**

The modern approach to handling events is predicated on the delegation event model, which defines standard and consistent mechanisms to get and process events. Its concept is sort of simple: a source generates an occasion and sends it to at least one or more listeners. In this scheme, the listener simply waits until it receives an occasion. Once an occasion is received, the listener processes the event then returns.



* Implement the appropriate interface in the listener so that it will receive the type of event desired.
* Implement code to register and unregister (if necessary) the listener as a recipient for the event notifications.

**Advantage of using Delegation Event Model**

The advantage of this design is that the appliance logic that processes events is cleanly separated from the interface logic that generates those events. An interface element is in a position to “delegate” the processing of an occasion to a separate piece of code. In the delegation event model, listeners must register with a source so as to receive an occasional notification. This provides is a crucial benefit: notifications are sent only to listeners that want to receive them. This is a more efficient way to handle events.

***Components of Event Handling***

**1. Events**

An event is an object that describes a phase change during a source. It is often generated as a consequence of an individual interacting with the weather during a graphical interface. Some activities that cause events to be generated are pressing a button, entering a personality via the keyboard, selecting an item during a list, and clicking the mouse. Events can also occur that aren’t directly caused by interactions with an interface. For example, an occasion could also be generated when a timer expires, a counter exceeds a worth, a software or hardware failure occurs, or an operation is completed.

**2. Event Sources**

A source is an object that generates an occasion. This occurs when the interior state of that object changes in how. Sources may generate quite one sort of event. A source must register listeners so as for the listeners to receive notifications a few specific sorts of events. Each sort of event has its own registration method.

**public void addTypeListener(TypeListener el)**

Here, Type is that the name of the event and el may be a regard to the event listener.

**3. Event Listeners**

A listener is an object that’s notified when an occasion occurs. It has two major requirements. First, it registered with one or more sources to receive notifications about specific sorts of events. Second, it implements methods to receive and process these notifications. The methods that receive and process events are defined in interfaces found in **java.awt.event**.



**Steps to handle an event**

**1.Register event source with Listener**

**addXListener(Listener class)**

**2.Override the method of Listener interfcae**

* Whenever the user clicks the button an event is generated.
* Now the object of concerned event class will be automatically created and information about the source and the event gets populated with in the same object.
* Then the event object is forwarded to the method of registered listener class.
* Now the method will get executed and returns.

import java.awt.\*;

import java.awt.event.\*;

class EventHandling extends Frame implements ActionListener

{

TextField textField;

EventHandling ()

{

textField = new TextField ();

textField.setBounds (60, 50, 170, 20);

Button button = new Button ("Show");

button.setBounds (90, 140, 75, 40);

button.addActionListener (this);

add (button);

add (textField);

setSize (250, 250);

setLayout (null);

setVisible (true);

}

public void actionPerformed (ActionEvent e)

{

textField.setText ("Hello World");

}

public static void main (String args[])

{

new EventHandling ();

}

}

Event Handing programs can be written in 3 ways.

* In the same class
* Using Annonymous inner class
* Using separate class

**Using Annonymous inner class**

Annonymous inner class is a class without having any reference name. It is used one time.

The object of interface can be created by annonymous inner class .

**new class/interface(){**

**method(){} };**

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

class testev2 extends JFrame

{

JTextField t;

JButton b;

testev2()

{

t=new JTextField(25);

b=new JButton("click");

add(t);

add(b);

setLayout(new FlowLayout());

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setSize(200,200);

setVisible(true);

b.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent e)

{

t.setText("CIME");

}});

t.addFocusListener(new FocusAdapter(){

public void focusGained(FocusEvent e)

{

t.setBackground(Color.red);

t.setText("t1 got focus");

}

});

}

public static void main(String[] s)

{

new testev2();

}

}

**Using separate class**

**import java.awt.\*;**

**import java.awt.event.\*;**

**import javax.swing.\*;**

**class testev extends JFrame**

**{**

**JTextField t;**

**JButton b;**

**testev()**

**{**

**t=new JTextField(25);**

**b=new JButton("click");**

**add(t);**

**add(b);**

**setLayout(new FlowLayout());**

**setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);**

**setSize(200,200);**

**setVisible(true);**

**clickev action=new clickev(this);**

**b.addActionListener(action);**

**}**

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

**{**

**new testev();**

**}**

**}**

**class clickev implements ActionListener**

**{**

**testev o;**

**clickev(testev o)**

**{**

**this.o=o;**

**}**

**public void actionPerformed(ActionEvent e)**

**{**

**o.t.setText("CIME");**

**}**

**}**

.

.

**KeyListener In Java**

Whenever there is a change in the state of the key, a KeyListener is notified. Just like ActionListener, the KeyListener is also found in the java.awt.event package.

**KeyListener interface provides the following methods:**

public abstract void keyPressed (KeyEvent e);

public abstract void keyReleased(KeyEvent e);

public abstract void keyTyped(KeyEvent e);

We need to implement the above methods to associate the key events with the component. We leave it to the user to implement a KeyListener example using swings in Java.