Event Handling In Java.........

Department of Computer Applications
Chitkara University, Punjab

*

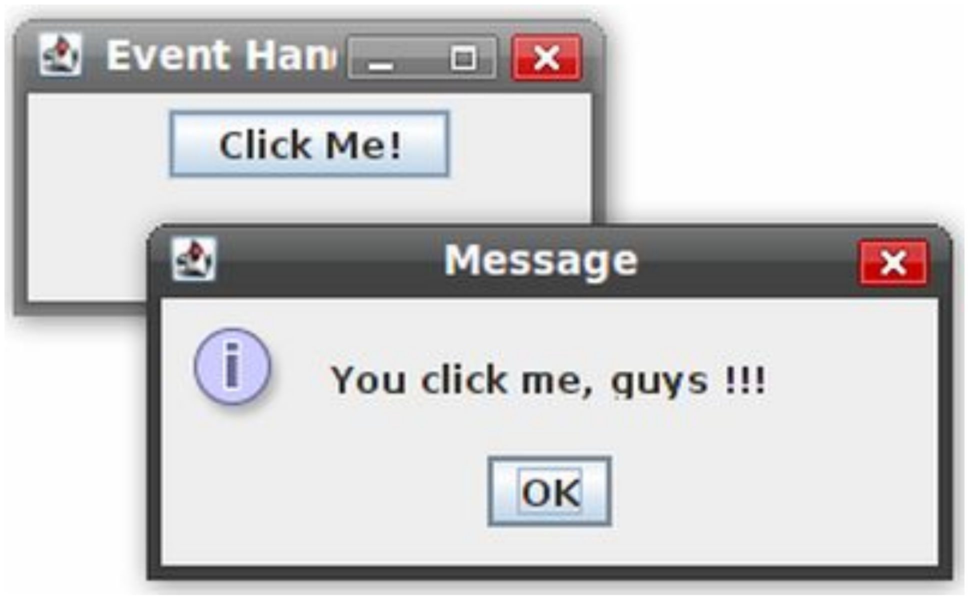
What is Source & Event ?

- **Event:-** When you press a button in your program or Android application the state of the button changes from ‘Unclicked’ to ‘Clicked'. This change in the state of our button is called an Event.

- **Source:-** In Java, nearly everything is an object. The button you press is an object too. Source is the object which generates an event.

What is an Event?

- The following figure clarifies events. When we click on the "click me" button an event is generated; that change event generates another frame that shows our message, that is passed to the program.

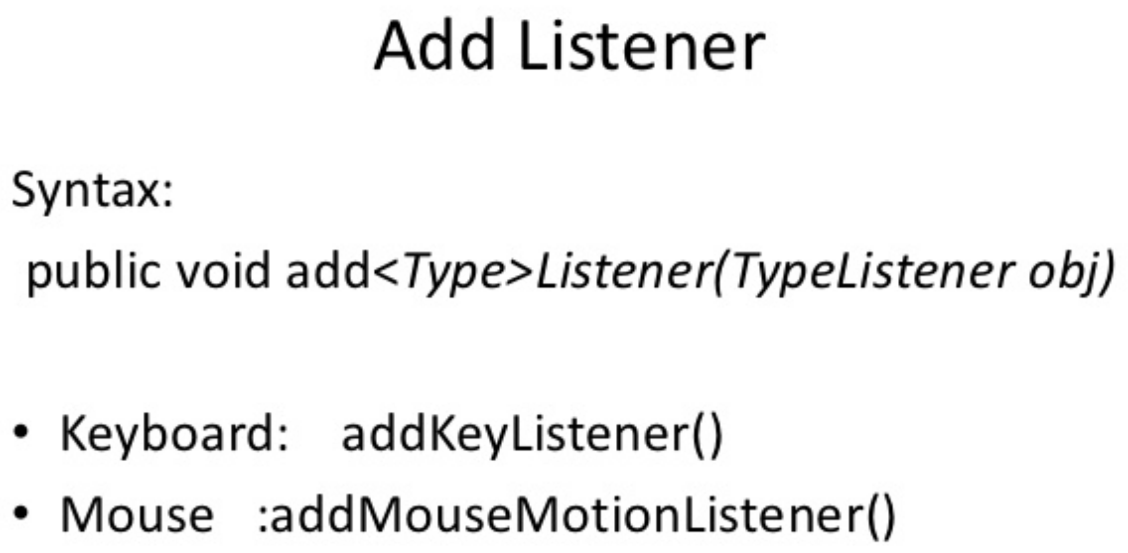


You click me, guys !!!

What is 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.

Conti..



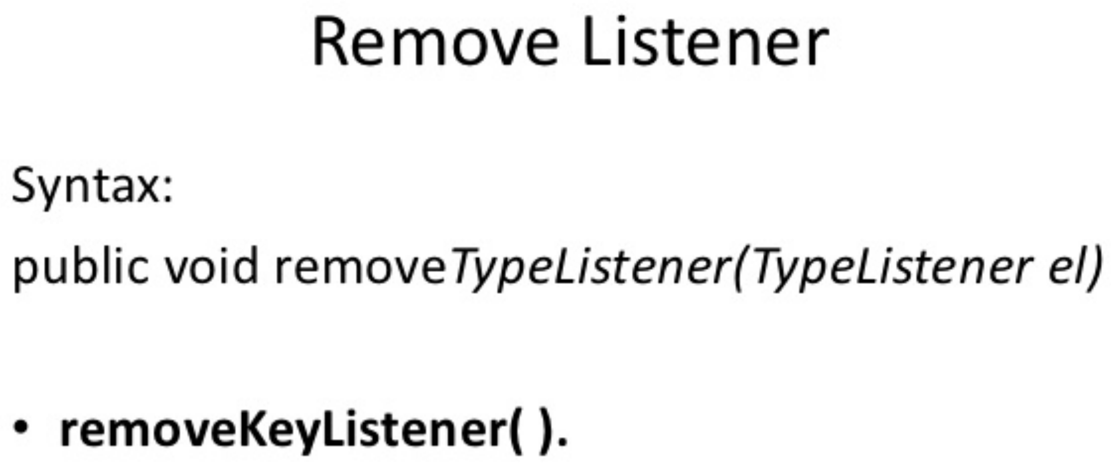
Add Listener

Syntax:

public void add*<Type>Listener(TypeListener obj)*

- Keyboard: addKeyListener()
- Mouse :addMouseMotionListener()

Conti..



Remove Listener

Syntax:

public void remove*TypeListener(TypeListener el)*

- **removeKeyListener( ).**

Delegation Event Model

- We know about Source, Listener, and Event. Now let’s look at the model which joins these 3 entities and make them work in sync.
- The delegation event model is used to accomplish the task. It consists of 2 components Source and listener.
- As soon as the source generates an event it is noticed by the listener and it handles the event at hand.
- The specialty of delegation Event Model is that the GUI component passes the event processing part to a completely separate set of code.

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.

Event Handling

Event Handling provides four types of classes; they are:

- Event Adapters
- Event classes
- Event Sources
- Event Listeners

`

Event Adapters

- Adapters are abstract classes for receiving various events.
- The methods in these classes are empty.
- These classes exists as convenience for creating listener objects.

Event Adapters

Following is the list of commonly used adapters while listening GUI events in AWT.

Sr. No.	Adapter & Description
1. FocusAdapter	An abstract adapter class for receiving focus events.
2. KeyAdapter	An abstract adapter class for receiving key events.
3. MouseAdapter	An abstract adapter class for receiving mouse events.
4. MouseMotionAdapter	An abstract adapter class for receiving mouse motion events.
5. WindowAdapter	An abstract adapter class for receiving window events.

Event classes

- Every event source generates an event and is named by a Java class. An event is generated when something changes within a graphical user interface.

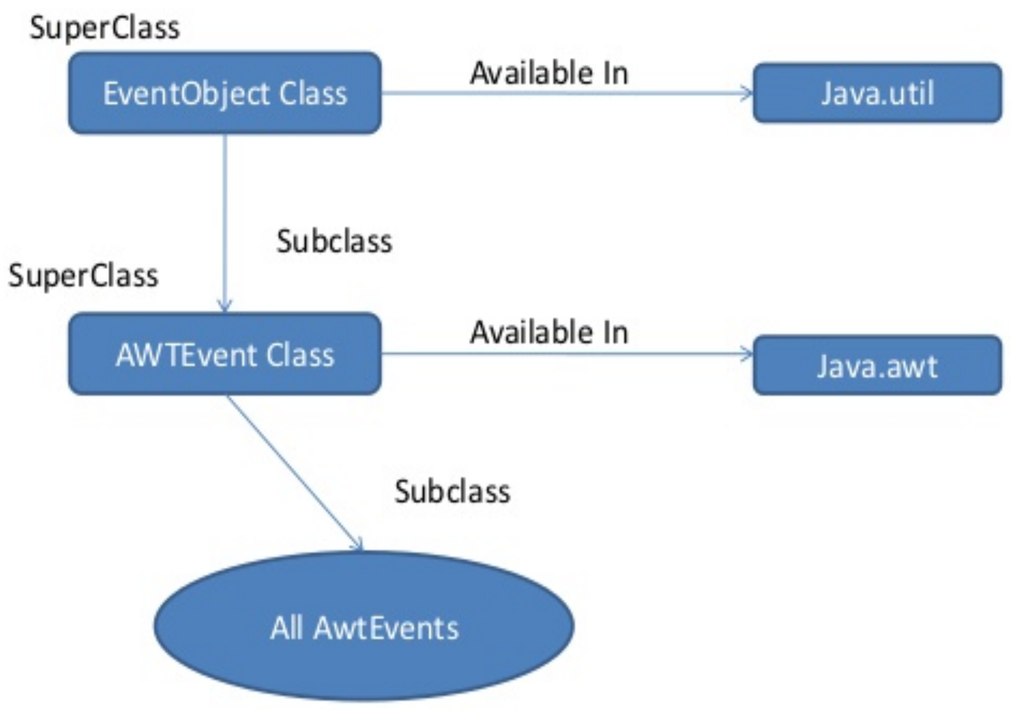
- For example, the event generated by a:

- Button is known as an **ActionEvent** (inbuilt class)
- Checkbox is known as an **ItemEvent** (inbuilt class)

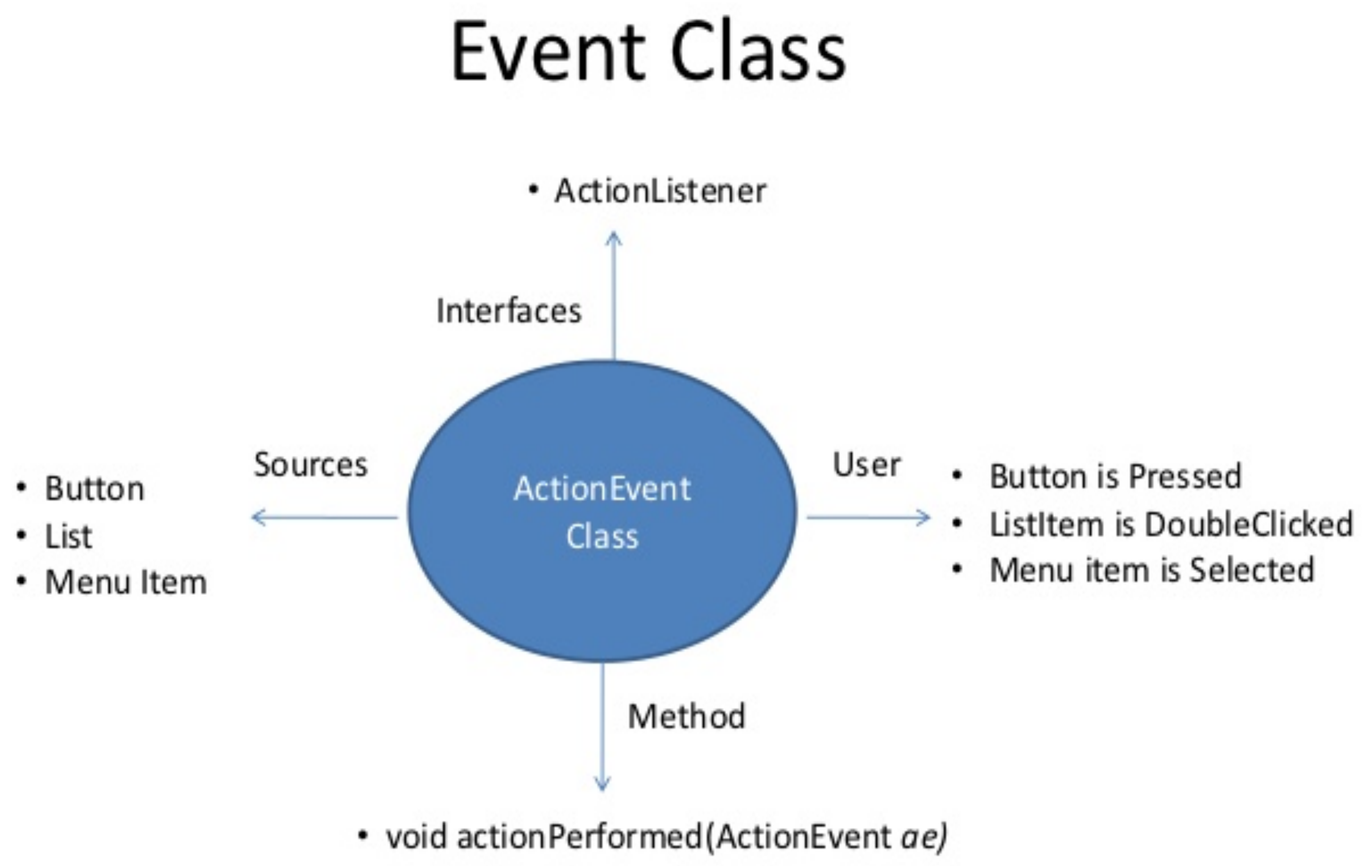
All of the events are listed in the ***java.awt.event*** package.

Event Handling

Event Classes



Event Handling



How to perform event handling in Java ?

The following is required to perform event handling:

- Implement the Listener interface and override its methods
- Register the component with the listener

Event Handling

We can use event handling in:

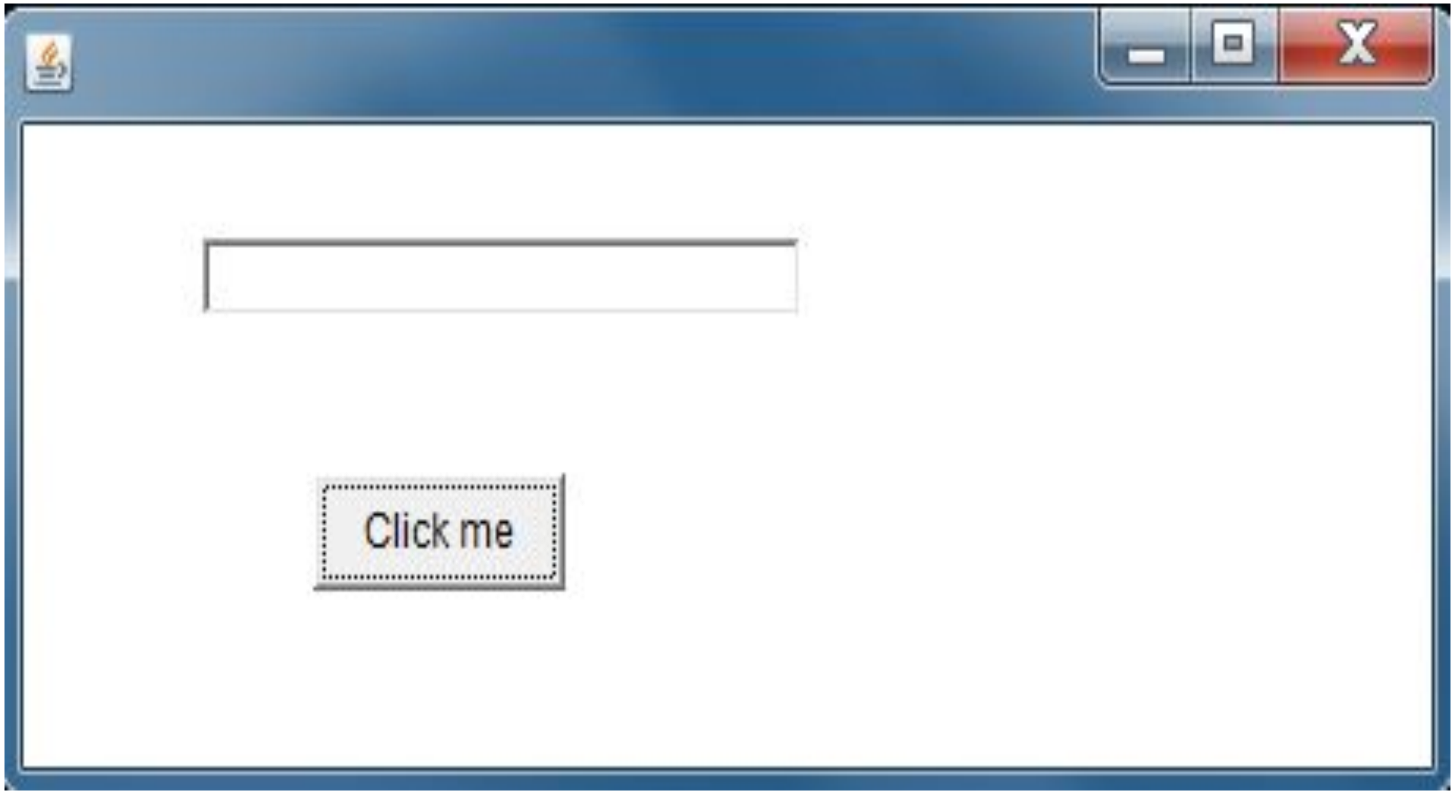
- Within the same class
- Another class
- Anonymously

Event Handling within same class

```
import java.awt.event.*;
import java.awt.*;
class EventActEx1 extends Frame implements
    ActionListener
{
   TextField txtfld;
   EventActEx1() {
      txtfld = new TextField();

      txtfld.setBounds(65, 60, 190, 20);
      Button bt = new Button("Click me");
      bt.setBounds(100, 120, 80, 30);

      bt.addActionListener(this);

      add(bt);
      add(txtfld);
setSize(350, 350);
      setLayout(null);
      setVisible(true);
   }
public void actionPerformed(ActionEvent e) {
      System.out.println("Event done ");
   }
   public static void main(String args[]) {
      new EventActEx1();
   }
}
```

Event handling

Output:



Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Event Listeners are interfaces

- ActionListener
- TextListener
- FocusListener
- MouseListener

- Available in java.awt.event package.

Implementing Listener interface

- Listener Interface include all abstract methods.
- All abstract method must be override by implemented sub class.
- An interface can be implement using *implements* keyword.

Implementing Listener interface

Implementing Listener interface in following ways…

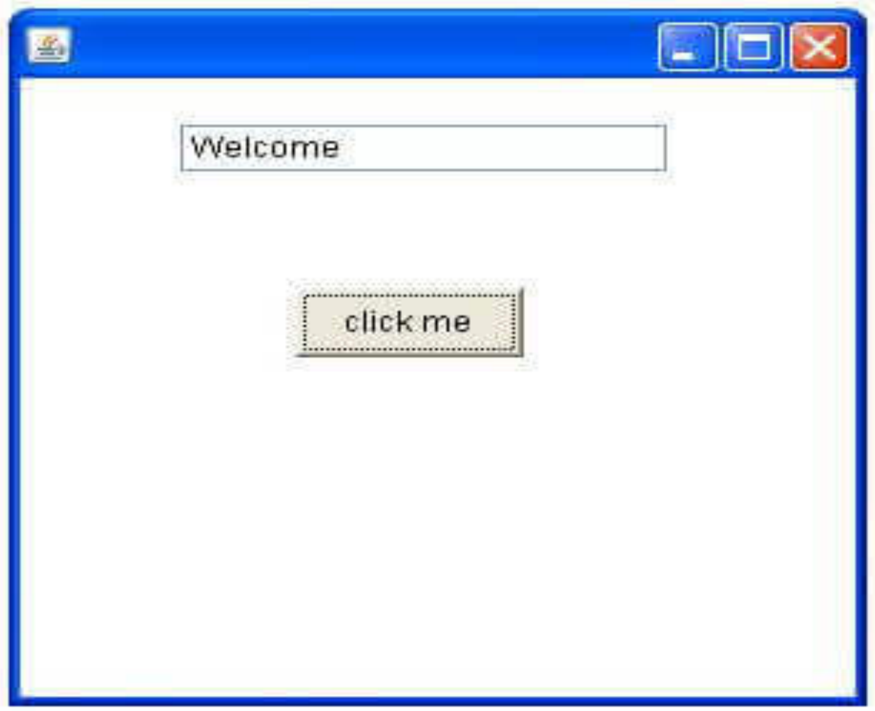
1. Implementation using Frame class
2. Can be implement by another class.
3. Can implement using inner class.
4. Can implement using anonymous class.

Java event handling by implementing ActionListener

```
import java.awt.*;
import java.awt.event.*;
class AEvent extends Frame implements
ActionListener{
TextField tf;
AEvent(){  // default constructor
//create components
tf=new TextField();
// to set the bounds of text Field
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);
//register listener
b.addActionListener(this);
// this refers to current instance(object)
 //add components and set size, layout and visi
   bility
add(b);add(tf); //add component into frame
setSize(300,300);
setLayout(null);
setVisible(true);
}
//override the abstract method of interface
public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}
public static void main(String args[]){
new AEvent();
}    // constructor call
}
```

*

Output



Java event handling by outer class

```
import java.awt.*;
import java.awt.event.*;
class AEvent2 extends Frame{
TextField tf;
AEvent2(){
//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);
//register listener
Outer o=new Outer(this);
b.addActionListener(o);//passing outer clas
s instance
//add components and set size, layout and
visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public static void main(String args[]){
new AEvent2();
}
}
import java.awt.event.*;
class Outer implements ActionListener{
AEvent2 obj;
Outer(AEvent2 obj){
this.obj=obj;
}
public void actionPerformed(ActionEvent e){

obj.tf.setText("welcome");
}
}
```

Java event handling by anonymous class

```
import java.awt.*;
import java.awt.event.*;
class AEvent3 extends Frame{
    TextField tf;
    AEvent3(){
    tf=new TextField();
    tf.setBounds(60,50,170,20);
    Button b=new Button("click me");
    b.setBounds(50,120,80,30);
b.addActionListener(new ActionListener(){
public void actionPerformed(){
System.out.print(“anonymmous
    cls”);
}
});
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public static void main(String args[])
new AEvent3();
}
}
```

*

Thank You