Event Handling Java

In AWT components, we came to know every component (except Panel and Label) generates events when interacted by the user like clicking over a button or pressing enter key in a text field etc. Listeners handle the events. Let us know the style (or design pattern) Java follows to handle the events.

The event handling Java involves four types of classes.

1. Event Sources
2. Event classes
3. Event Listeners
4. Event Adapters

1. Event Sources

Event sources are components, subclasses of **java.awt.Component**, capable to generate events. The event source can be a button, TextField or a Frame etc.

2. Event classes

Almost every event source generates an event and is named by some Java class. For example, the event generated by button is known as **ActionEvent** and that of Checkbox is known as **ItemEvent**. All the events are listed in **java.awt.event** package. Following list gives a few components and their corresponding listeners.

Component	Event it generates
Button, TextField, List, Menu	ActionEvent
Frame	WindowEvent
Checkbox, Choice, List	ItemEvent
Scrollbar	AdjustmentEvent
Mouse (hardware)	MouseEvent
Keyboard (hardware)	KeyEvent

The events generated by hardware components (like **MouseEvent** and **KeyEvent**) are known as **low-level events** and the events generated by software components (like Button, List) are known as **semantic events**.

3. Event Listeners

The events generated by the GUI components are handled by a special group of interfaces known as "**listeners**". Note, Listener is an interface. Every component has its own listener, say, AdjustmentListener handles the events of scrollbar Some listeners handle the events of multiple components. For example, ActionListener handles the events of Button, TextField, List and Menus. Listeners are from **java.awt.event** package.

More description on listeners and list of listeners is available at **Java AWT Listeners**.

4. Event Adapters

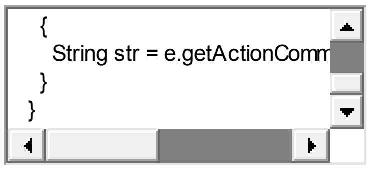
When a listener includes many abstract methods to override, the coding becomes heavy to the programmer. For example, to close the frame, you override seven abstract methods of WindowListener, in which, infact you are using only one method. To avoid this heavy coding, the designers come with another group of classes known as "**adapters**". Adapters are abstract classes defined in **java.awt.event** package. Every listener that has more than one abstract method has got a corresponding adapter class.

More description on adapters and list of adapters is available at **Java AWT Adapters**.

What happens internally at a button click?

We know the events are handled by listeners and ActionListener handles the events of a button. Observe the following skeleton code.

Java



```
  {
   String str = e.getActionComm
  }
 }
```

```
public class ButtonDemo extends Frame implements ActionListener
{
 public ButtonDemo()
 {
  Button btn = new Button("OK");
  btn.addActionListener(this);
  add(btn);
 }
 public void actionPerformed(ActionEvent e)
 {
  String str = e.getActionCommand();
 }
}
```

The above program contains a small part of event handling Java code of a big program, and is used here just for the explanation of internal event handling Java mechanism.

A button object **btn** is created for which events are yet to be linked. For this, the first step is implementing **ActionListener** to the class ButtonDemo. In the second step, we link or register the button **btn** with the ActionListener. For this **addActionListener()** method of Button class is used. The parameter "**this**" refers the ActionListener.

btn.addActionListener(this);

With the above statement, **btn** is linked with the **ActionListener**.

1. When the button **btn** is clicked, the button generates an event called **ActionEvent**. It is the nature of the button taken care by JVM.

2. This **ActionEvent** reaches the **ActionListener** because we registered the button with ActionListener earlier.

3. Now, the question is, what ActionListener does with the ActionEvent object it received?

The ActionListener simply calls **actionPerformed()** method and passes the ActionEvent object to the parameter as follows.

public void actionPerformed(ActionEvent e)

The parameter for the above method comes from ActionListener.

4. Finally the **ActionEvent** object generated by the button **btn** reaches the **e** object of ActionEvent. All this is done by JVM implicitly. For this reason, the **getActionCommand()** method of ActionEvent class knows the label of the button **btn**.

String str = e.getActionCommand();

The **e** represents an object of ActionEvent and the value for the **e** is coming from button **btn**. **str** is nothing but the label of the button OK.