**Event Handling**

**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 and 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.

# **Event and Listener (Java Event Handling)**

|  |
| --- |
| Changing the state of an object is known as an event.  For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling. |

## 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 |

## Registration Methods

For registering the component with the Listener, many classes provide the registration methods. For example:

* **Button**
  + public void addActionListener(ActionListener a){}
* **MenuItem**
  + public void addActionListener(ActionListener a){}
* **TextField**
  + public void addActionListener(ActionListener a){}
  + public void addTextListener(TextListener a){}
* **TextArea**
  + public void addTextListener(TextListener a){}
* **Checkbox**
  + public void addItemListener(ItemListener a){}
* **Choice**
  + public void addItemListener(ItemListener a){}
* **List**
  + public void addActionListener(ActionListener a){}
  + public void addItemListener(ItemListener a){}

### **Java Event Handling Code**

We can put the event handling code into one of the following places:

1. Within class
2. Other class
3. Anonymous class

### **Java event handling by implementing ActionListener**

**import** java.awt.\*;

**import** java.awt.event.\*;

**class** AEvent **extends** Frame **implements** ActionListener

{

TextField tf;

AEvent()

{

tf=**new** TextField();

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

Button b=**new** Button("click me");

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

//register listener

b.addActionListener(**this**);//passing current instance

//add components and set size, layout and visibility

add(b);

add(tf);

setSize(300,300);

setLayout(**null**);

setVisible(**true**);

}

**public** **void** actionPerformed(ActionEvent e)

{

tf.setText("Welcome");

}

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

{

**new** AEvent();

}

}

**public void setBounds(int xaxis, int yaxis, int width, int height);** have been used in the above example that sets the position of the component it may be button, textfield etc.

