JAVA

Event Handling

By :

Subodh Sharma

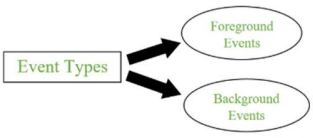
What is an Event?

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.

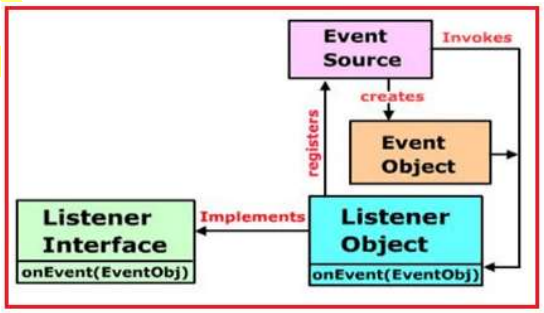


What is Event Handling?

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 *delegation event model* provides a consistent way of generating and processing events. In this model a *source* generates an event and sends it to one or more *listeners*. A listener waits until it receives an event and once it receives an event, processes the event and returns.

Benefit of delegation model : 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. This is an efficient way of handling the event because the event notifications are sent only to those listener that want to receive them.

Listener Registration with Source object :

In this model ,Listener needs to be registered with the source object so that the listener can receive the event notification.

```
public void addTypeListener(TypeListener el)
```

WHERE Type is the name of the event and el is a reference to the event listener.

For example :

Event	Source
Key	keybaord
Mouse	mouse
Action	button, list_item double clicked , menu item clicked
FocusEvent	component gains or looses input

Source	Listener	EventClass	Methods to implement
Keyboard	KeyListener	KeyEvent	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
Button	ActionListener	ActionEvent	actionPerformed(ActionEvent)
Mouse	MouseListener	MouseEvent	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)
MouseMotion	MouseMotionListener	MouseEvent	mouseMoved(MouseEvent) mouseDragged(MouseEvent)
MouseWheel	MouseWheelListener	MouseWheelEvent	mouseMoved(MouseEvent)

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.

Event and Listener (Java Event Handling)

1. Changing the state of an object is known as an event.
2. Event is an instance of an event class.
3. Root class of event classes is java.util.EventObject.
4. We can identify the source object of an event using getSource() method in the EventObject class
5. The subclasses of EventObject deals with specific types of events, such as Action event, window event, mouse 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 Class	Listener Interface	Description
ActionEvent	ActionListener	An event that indicates that a component-defined action occurred like a button click or selecting an item from the menu-item list.
AdjustmentEvent	AdjustmentListener	The adjustment event is emitted by an Adjustable object like Scrollbar.
ComponentEvent	ComponentListener	An event that indicates that a component moved, the size changed or changed its visibility.
ContainerEvent	ContainerListener	When a component is added to a container (or) removed from it, then this event is generated by a container object.
FocusEvent	FocusListener	These are focus-related events, which include focus, focusin, focusout, and blur.
ItemEvent	ItemListener	An event that indicates whether an item was selected or not.
KeyEvent	KeyListener	An event that occurs due to a sequence of keypresses on the keyboard.
MouseEvent	MouseListener & MouseMotionListener	The events that occur due to the user interaction with the mouse (Pointing Device).

Event Class	Listener Interface	Description
MouseWheelEvent	MouseWheelListener	An event that specifies that the mouse wheel was rotated in a component.
TextEvent	TextListener	An event that occurs when an object's text changes.
WindowEvent	WindowListener	An event which indicates whether a window has changed its status or not.

Steps to perform Event Handling

Following steps are required to perform event handling:

1. Register the component with the Listener

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

Example of Button & Choice with event Handling :

```
import java.awt.*;
import java.awt.event.*;
class AwtChoice2 implements ActionListener, ItemListener
{
Frame f;
Choice c;
Button bexit;
TextField tf;
        AwtChoice2()
        {
Font fo = new Font("Times Roman",Font.BOLD,35);
        f= new Frame();
        bexit = new Button("Exit");
        tf = new TextField();
        tf.setFont(fo);
        f.add(bexit);  f.add(tf);
        tf.setBounds(100,200,200,50);
        bexit.setBounds(100,270,75,40);
        c=new Choice();
        c.setBounds(100,100, 200,75);
        c.add("Pratik");  c.add("Anurag");   c.add("Rahul");
c.add("Sparshi");
        c.add("Pooja");
c.select("Rahul");
        f.add(c);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
bexit.addActionListener(this);
c.addItemListener(this);
      }

public void actionPerformed(ActionEvent ae)
{     String str= ae.getActionCommand();
    if(str.equals("Exit"))
        f.dispose();
}
public void itemStateChanged(ItemEvent ie)
{   tf.setText(c.getSelectedItem());
}

public static void main(String args[])
{   AwtChoice2 obj=new AwtChoice2();
}
}
```

Java AWT MenuItem and Menu with Event Handling :

```
class AwtM implements ActionListener
{
Frame fr;
AwtM()
 {
  fr = new Frame("My Example");
  MenuBar mb = new MenuBar();

  Menu me1= new Menu("File");
  MenuItem i11= new MenuItem("New");
  MenuItem i12= new MenuItem("Open");
  MenuItem i13= new MenuItem("Exit");
  me1.add(i11); me1.add(i12); me1.add(i13);
mb.add(me1);

  Menu me2= new Menu("Edit");

  MenuItem i21= new MenuItem("Cut");
  Menu subme= new Menu("Copy");
  MenuItem i23= new MenuItem("Paste");
  MenuItem i31= new MenuItem("Copy1");
  MenuItem i32= new MenuItem("Copy2");

  me2.add(i21); me2.add(subme); me2.add(i23);
mb.add(me2);
subme.add(i31);
subme.add(i32);

fr.setMenuBar(mb);
fr.setSize(400,400);
fr.setLayout(null);
fr.setVisible(true);
i13.addActionListener(this);
i12.addActionListener(this);
i11.addActionListener(this);
 }
```

```
public void actionPerformed(ActionEvent ae)
{
   String str = ae.getActionCommand();
   if(str.equals("Exit"))
     fr.dispose();
  if(str.equals("Open"))
     fr.setBackground(Color.yellow);
  if(str.equals("New"))
     fr.setBackground(Color.white);
}

public static void main(String s[])
  {
 AwtM ob= new AwtM();
  }
}
```

MouseListener and MouseMotionListener in Java

MouseListener and MouseMotionListener is an interface in java.awt.event package . Mouse events are of two types.

a) MouseListener handles the events when the mouse is not in motion.
b) While MouseMotionListener handles the events when mouse is in motion.

There are five types of events that **MouseListener** can generate. There are five abstract functions that represent these five events. **The abstract functions are :**

1. **void mouseReleased(MouseEvent e)** : Mouse key is released
2. **void mouseClicked(MouseEvent e)** : Mouse key is pressed/released
3. **void mouseExited(MouseEvent e)** : Mouse exited the component
4. **void mouseEntered(MouseEvent e)** : Mouse entered the component
5. **void mousepressed(MouseEvent e)** : Mouse key is pressed

There are two types of events that **MouseMotionListener** can generate. There are two abstract functions that represent these five events. **The abstract functions are :**

1. **void mouseDragged(MouseEvent e)** : Invoked when a mouse button is pressed in the component and dragged. Events are passed until the user releases the mouse button.
2. **void mouseMoved(MouseEvent e)** : invoked when the mouse cursor is moved from one point to another within the component, without pressing any mouse buttons.

Class methods

S.N.	Method & Description
1	**int getButton()** Returns which, if any, of the mouse buttons has changed state.
2	**int getClickCount()** Returns the number of mouse clicks associated with this event.
3	**Point getLocationOnScreen()** Returns the absolute x, y position of the event.
4	**static String getMouseModifiersText(int modifiers)** Returns a String describing the modifier keys and mouse buttons that were down during the event, such as "Shift", or "Ctrl+Shift".
5	**Point getPoint()** Returns the x,y position of the event relative to the source component.
6	**int getX()** Returns the horizontal x position of the event relative to the source component.
7	**int getXOnScreen()** Returns the absolute horizontal x position of the event.
8	**int getY()** Returns the vertical y position of the event relative to the source component.
9	**int getYOnScreen()** Returns the absolute vertical y position of the event.
10	**boolean isPopupTrigger() Returns whether or not this mouse event is the popup menu trigger event for the platform.**

11	**String paramString()** Returns a parameter string identifying this event.
12	**void translatePoint(int x, int y)** Translates the event's coordinates to a new position by adding specified x (horizontal) and y (vertical) offsets.

Example :

```
import java.awt.event.*;
import java.awt.*;

class AwtEMouse implements MouseListener, MouseMotionListener
{   AwtEMouse()
    {     Frame fr = new Frame("Mouse Event handling");
          TextField tf = new TextField("textField");
          Label lb = new Label("this is label");
          tf.setBounds(100,100,100,50);
          lb.setBounds(100,200,100,50);
          fr.add(tf);
          fr.add(lb);
          fr.setSize(400,400);
          fr.setLayout(null);
          fr.setVisible(true);
tf.addMouseListener(this);
fr.addMouseMotionListener(this);
  }
//****************** mouseListener******
public void mouseClicked(MouseEvent me)
{
  System.out.println("MouseClicked");
}
public void mouseEntered(MouseEvent me)
{
  System.out.println("MouseEntered");
}
public void mouseExited(MouseEvent me)
{
  System.out.println("MouseExited");
}
public void mousePressed(MouseEvent me)
{
  System.out.println("MousePressed");
}
public void mouseReleased(MouseEvent me)
{
  System.out.println("MouseReleased");
}
```

```
//****************** mouseMotionListener******
public void mouseDragged(MouseEvent me)
{
  System.out.println("Dragged");
}
public void mouseMoved(MouseEvent me)
{
  System.out.println("MouseMoved");
}

public static void main(String s[])
  {
   AwtEMouse obj = new AwtEMouse();
  }
}
```

KeyListener in Java

To handle keyboard events, we use the same general architecture as show in the mouse event. Here we will be implementing the KeyListener interface.

There are three types of events (KEY_PRESSED, KEY_RELEASED and KEY_TYPED) that **KeyListener** can generate. Thus each time the user presses a key, at lease two and often three events are generated.

There are three abstract functions that represent these three events. **The abstract functions are :**
void keyPressed(KeyEvent ke)
void keyReleased(KeyEvent ke)
void keyTyped(KeyEvent ke)

Method Summary

char	**getKeyChar**() Returns the character associated with the key in this event.
int	**getKeyCode**() Returns the integer key-code associated with the key in this event.
static String	**getKeyModifiersText**(int modifiers) Returns a String describing the modifier key(s), such as "Shift", or "Ctrl+Shift".
static String	**getKeyText**(int keyCode) Returns a String describing the keyCode, such as "HOME"
boolean	**isActionKey**() Returns whether or not the key in this event is an "action" key, as defined in Event.java.
String	**paramString**() Returns a parameter string identifying this event.
void	**setKeyChar**(char keyChar) Set the keyChar value to indicate a logical character.
void	**setKeyCode**(int keyCode) Set the keyCode value to indicate a physical key.
void	**setModifiers**(int modifiers) Set the modifiers to indicate additional keys that were held down (shift, ctrl, alt, meta) defined as part of InputEvent.

Example :

```
import java.awt.event.*;
import java.awt.*;
class AwtEKey implements KeyListener
{    Label lb;
     String str="";
     Frame fr;
     TextField tf;
   AwtEKey()
      {
           Font f= new Font("Roman",Font.BOLD,20);
           fr = new Frame("Mouse Event handling");
           tf = new TextField("textField");
           lb = new Label("this is label");
           tf.setFont(f);
           tf.setBounds(100,100,100,50);
           lb.setBounds(100,200,100,50);
                    fr.add(tf);
                    fr.add(lb);
                    fr.setSize(400,400);
                    fr.setLayout(null);
                    fr.setVisible(true);
tf.addKeyListener(this);
  }
//****************** keyListener******
public void keyPressed(KeyEvent ke)
{
 int v= ke.getKeyCode();
if(v==ke.VK_A)
  fr.setBackground(Color.green);
if(v==ke.VK_I)
  fr.setBackground(Color.red);
 }

public void keyReleased(KeyEvent ke){ }

public void keyTyped(KeyEvent ke)
{
char ch=ke.getKeyChar();
str =str+ch;
lb.setText(str);
if (ch==' ')  str="";
if (lb.getText().equals("exit"))
  fr.dispose();
}
public static void main(String s[])
```

```
  {
   AwtEKey obj = new AwtEKey();
  }
}
```