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.

Types of Event

Foreground Events - Those events which require the direct interaction of user.

For example, clicking on a button, moving the mouse, entering a character through keyboard
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.

What is Event Handling?

Event Handling is the mechanism that controls the event and decides what should happen if 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

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.

source must register

A source must register listeners in order for the listeners to receive notifications about a specific type of event. Each type of event has its own registration method.

Syntax:
public void add*Type*Listener (*Type*Listener *el*)

Example:
addKeyListener( ).
addMouseMotionListener( ).

Source may unregister

A source must also provide a method that allows a listener to unregister an interest in a specific type of event.

Syntax:

public void remove*Type*Listener(*Type*Listener *el*)

Example:

removeKeyListener( ).

Events are supported by a number of Java packages, like
java.util, **java.awt** and **java.awt.event**.

How Events are handled?

A source generates an Event and send it to one or more listeners registered with the source. Once event is received by the listener, they process the event and then return.

Important Event Classes and Interface

Event Classes	Description	Listener Interface
ActionEvent	generated when button is pressed, menuitem is selected, list-item is double clicked	ActionListener
MouseEvent	generated when mouse is dragged, moved,clicked,pressed or released and also when it enters or exit a component	MouseListener
KeyEvent	generated when input is received from keyboard	KeyListener
ItemEvent	generated when check-box or list item is clicked	ItemListener
TextEvent	generated when value of textarea or textfield is changed	TextListener
MouseWheelEvent	generated when mouse wheel is moved	MouseWheelListener

WindowEvent	generated when window is activated, deactivated, deiconified, iconified, opened or closed	WindowListener
ComponentEvent	generated when component is hidden, moved, resized or set visible	ComponentEventListener
ContainerEvent	generated when component is added or removed from container	ContainerListener
AdjustmentEvent	generated when scroll bar is manipulated	AdjustmentListener
FocusEvent	generated when component gains or loses keyboard focus	FocusListener

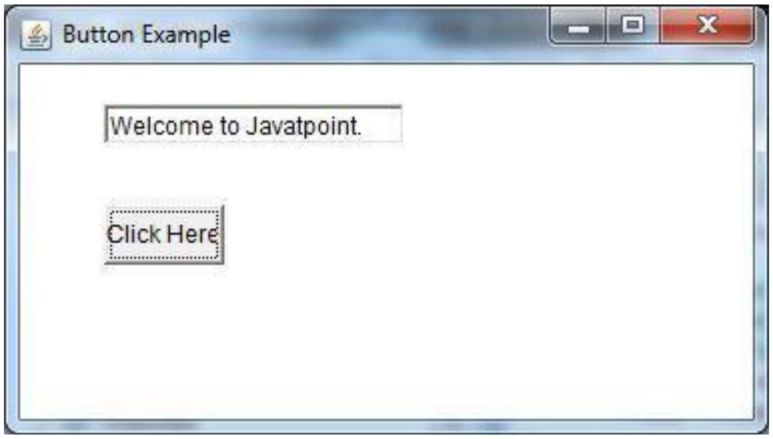
Java ActionListener Interface

- ✓ The Java ActionListener is notified whenever you click on the button or menu item.
- ✓ It is notified against ActionEvent. The ActionListener interface is found in java.awt.event package.
- ✓ It has only one method: actionPerformed().

Java ActionListener Example: On Button click

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

public class ActionListenerExample implements ActionListener{
Frame f;
Button b;
TextField tf;

ActionListenerExample()
{
    f=new      Frame("ActionListener      Example");
tf=new TextField();
   tf.setBounds(50,50, 150,20);

        b=new Button("Click Here");
b.setBounds(50,100,60,30);
  //2nd step
  b.addActionListener(this);
  f.add(b);f.add(tf);
  f.setSize(400,400);
  f.setLayout(null);
  f.setVisible(true);
}
public void actionPerformed(ActionEvent e){
tf.setText("Welcome to Javatpoint.");
}
public  static  void  main(String[]  args)  {
new ActionListenerExample();

} }
```

Output:



Java MouseListener Interface

The Java MouseListener is notified whenever you change the state of mouse. It is notified against MouseEvent.

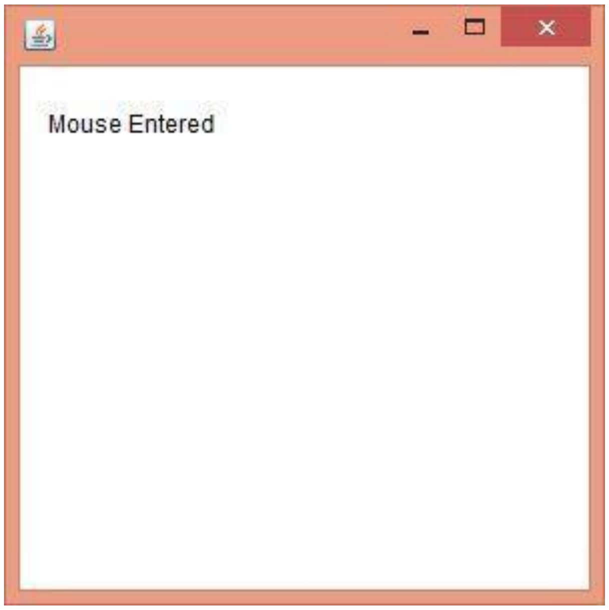
The MouseListener interface is found in java.awt.event package. It has five methods.

```
public abstract void mouseClicked(MouseEvent e);
public abstract void mouseEntered(MouseEvent e);
public abstract void mouseExited(MouseEvent e);   public
abstract void mousePressed(MouseEvent e);
```

Java MouseListener Example-1

```
import java.awt.*;   import
java.awt.event.*;
public class MouseListenerExample extends Frame implements MouseListener{
    Label l;
    MouseListenerExample(){
addMouseListener(this);

        l=new Label();
        l.setBounds(20,50,100,20);
add(l);
        setSize(300,300);
setLayout(null);          setVisible(true);
    }
    public void mouseClicked(MouseEvent e) {
l.setText("Mouse Clicked");
    }
    public void mouseEntered(MouseEvent e) {
l.setText("Mouse Entered");
    }
    public void mouseExited(MouseEvent e) {
l.setText("Mouse Exited");
    }
    public void mousePressed(MouseEvent e) {
l.setText("Mouse Pressed");
    }
    public void mouseReleased(MouseEvent e) {
l.setText("Mouse Released");
    }
public static void main(String[] args) {       new
MouseListenerExample();
}
}
```

Output:

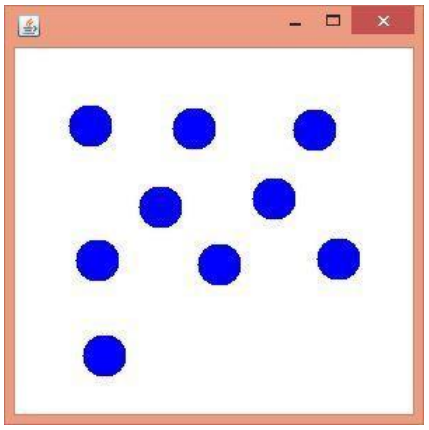


```
Mouse Entered
```

Java MouseListener Example 2

```
import java.awt.*;   import
java.awt.event.*;
public class MouseListenerExample2 extends Frame implements MouseListener{
MouseListenerExample2(){          addMouseListener(this);

       setSize(300,300);
setLayout(null);          setVisible(true);
   }
   public void mouseClicked(MouseEvent e) {
Graphics g=getGraphics();
g.setColor(Color.BLUE);
       g.fillOval(e.getX(),e.getY(),30,30);
}
   public void mouseEntered(MouseEvent e) {}
public void mouseExited(MouseEvent e) {}
public void mousePressed(MouseEvent e) {}
public void mouseReleased(MouseEvent e) {}

public  static  void  main(String[]  args)  {
new MouseListenerExample2();
}
}
```

Output:



Java MouseMotionListener Interface

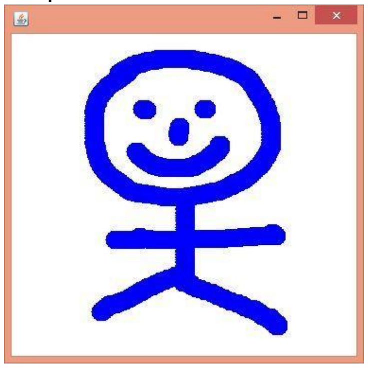
The Java MouseMotionListener is notified whenever you move or drag mouse. It is notified against MouseEvent. The MouseMotionListener interface is found in java.awt.event package. It has two methods.

```
public abstract void mouseDragged(MouseEvent e);
public abstract void mouseMoved(MouseEvent e);
```

Java MouseMotionListener Example

```
import java.awt.*;   import java.awt.event.*;   public class MouseMotionListenerExample
extends Frame implements MouseMotionListener{

   MouseMotionListenerExample(){
addMouseMotionListener(this);

      setSize(300,300);
setLayout(null);          setVisible(true);
   }
public void mouseDragged(MouseEvent e) {
Graphics                          g=getGraphics();
g.setColor(Color.BLUE);
   g.fillOval(e.getX(),e.getY(),20,20);
}
public void mouseMoved(MouseEvent e) {}

public  static  void  main(String[]  args)  {
new MouseMotionListenerExample();
}
}
```

Output:



Java ItemListener Interface

- ✓ The Java ItemListener is notified whenever you click on the checkbox. It is notified against ItemEvent.
- ✓ The ItemListener interface is found in java.awt.event package. It has only one method: itemStateChanged().

itemStateChanged() method

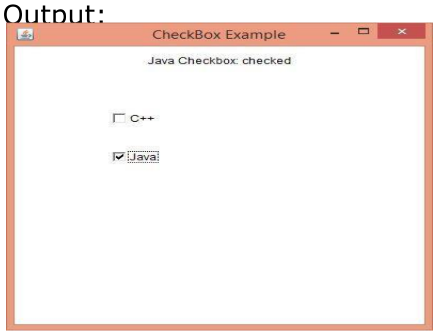
The itemStateChanged() method is invoked automatically whenever you click or unclick on the registered checkbox component.

```
1. public abstract void itemStateChanged(ItemEvent e);
```

Java ItemListener Example

```
import java.awt.*;     import
java.awt.event.*;
public class ItemListenerExample implements ItemListener{
    Checkbox checkBox1,checkBox2;
    Label label;
    ItemListenerExample(){
        Frame f= new Frame("CheckBox Example");
label = new Label();
label.setAlignment(Label.CENTER);
label.setSize(400,100);
        checkBox1 = new Checkbox("C++");
checkBox1.setBounds(100,100, 50,50);
checkBox2 = new Checkbox("Java");
checkBox2.setBounds(100,150, 50,50);
        f.add(checkBox1); f.add(checkBox2); f.add(label);
checkBox1.addItemListener(this);
checkBox2.addItemListener(this);
f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
   public void itemStateChanged(ItemEvent e) {
      if(e.getSource()==checkBox1)
         label.setText("C++ Checkbox: "
         + (e.getStateChange()==1?"checked":"unchecked"));
      if(e.getSource()==checkBox2)
      label.setText("Java Checkbox: "
      + (e.getStateChange()==1?"checked":"unchecked"));
    }
public static void main(String args[])
{
   new ItemListenerExample();
}
}
```

Output:



Java KeyListener Interface

- ✓ The Java KeyListener is notified whenever you change the state of key. It is notified against KeyEvent.
- ✓ The KeyListener interface is found in java.awt.event package. It has three methods.

Methods of KeyListener interface

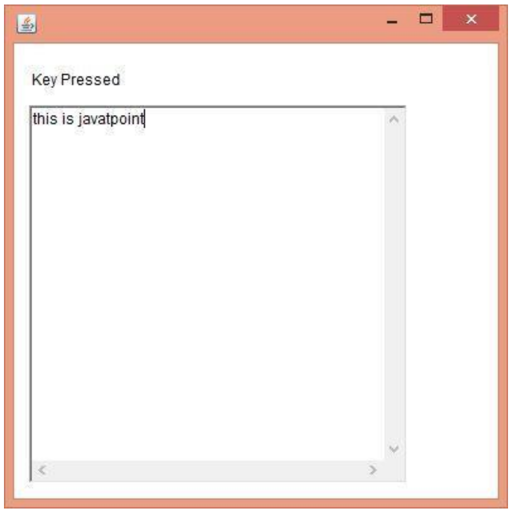
The signature of 3 methods found in KeyListener interface are given below:

```
public abstract void keyPressed(KeyEvent e);
public abstract void keyReleased(KeyEvent e);
public abstract void keyTyped(KeyEvent e);
```

Java KeyListener Example

```
import java.awt.*;   import
java.awt.event.*;
public class KeyListenerExample extends Frame implements KeyListener{
    Label l;
    TextArea area;
    KeyListenerExample(){

        l=new Label();
        l.setBounds(20,50,100,20);
area=new TextArea();
area.setBounds(20,80,300, 300);
area.addKeyListener(this);

        add(l);add(area);
setSize(400,400);
setLayout(null);          setVisible(true);
    }
    public      void      keyPressed(KeyEvent      e)      {
l.setText("Key Pressed");
    }
    public void keyReleased(KeyEvent e) {
l.setText("Key Released");
    }
    public      void      keyTyped(KeyEvent      e)      {
l.setText("Key Typed");
    }

    public static void main(String[] args) {
new KeyListenerExample();
    }
}
```

Output:


Key Pressed

```
this is javatpoint
```

Java Adapter Classes

Java adapter classes *provide the default implementation of listener interfaces*. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it *saves code*.

An adapter class provides the default implementation of all methods in an event listener interface.

The adapter classes are found in **java.awt.event**, **java.awt.dnd** and **javax.swing.event** packages. The Adapter classes with their corresponding listener interfaces are given below.

java.awt.event Adapter classes

Adapter class	**Listener interface**
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener

FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

java.awt.dnd Adapter classes

Adapter class	**Listener interface**
DragSourceAdapter	DragSourceListener
DragTargetAdapter	DragTargetListener

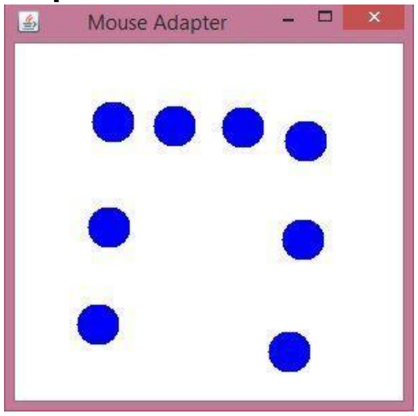
javax.swing.event Adapter classes

Adapter class	**Listener interface**
MouseInputAdapter	MouseInputListener
InternalFrameAdapter	InternalFrameListener

Java MouseAdapter Example

```
import java.awt.*;   import
java.awt.event.*;
public class MouseAdapterExample extends MouseAdapter{
   Frame f;
   MouseAdapterExample(){
f=new Frame("Mouse Adapter");
f.addMouseListener(this);

      f.setSize(300,300);
      f.setLayout(null);
      f.setVisible(true);
}
   public void mouseClicked(MouseEvent e) {
Graphics g=f.getGraphics();
g.setColor(Color.BLUE);
      g.fillOval(e.getX(),e.getY(),30,30);
   }

public static void main(String[] args) {
   new MouseAdapterExample();
}
}
```

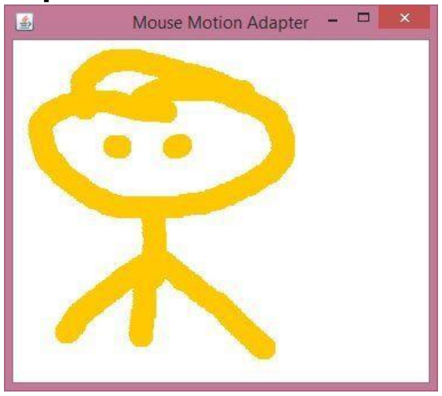
Output:



Java MouseMotionAdapter Example

```
import java.awt.*;   import
java.awt.event.*;
public class MouseMotionAdapterExample extends MouseMotionAdapter{
    Frame f;
    MouseMotionAdapterExample(){
f=new Frame("Mouse Motion Adapter");
f.addMouseMotionListener(this);

        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
}
public void mouseDragged(MouseEvent e) {
Graphics g=f.getGraphics();
g.setColor(Color.ORANGE);
    g.fillOval(e.getX(),e.getY(),20,20);
}  public static void main(String[]
args) {       new
MouseMotionAdapterExample();
}
}
```

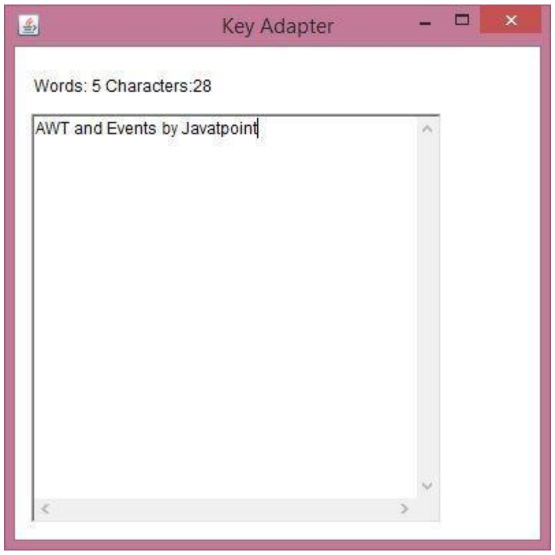
Output:



Java KeyAdapter Example

```
import java.awt.*;   import
java.awt.event.*;
public class KeyAdapterExample extends KeyAdapter{
   Label l;
   TextArea area;
   Frame f;
   KeyAdapterExample(){
f=new    Frame("Key    Adapter");
l=new Label();
      l.setBounds(20,50,200,20);
area=new TextArea();
area.setBounds(20,80,300, 300);
area.addKeyListener(this);

      f.add(l);f.add(area);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
}
   public void keyReleased(KeyEvent e) {
String text=area.getText();        String
words[]=text.split("\\s");
      l.setText("Words: "+words.length+" Characters:"+text.length());
   }

   public static void main(String[] args) {
new KeyAdapterExample();
   }
}
```

Output:



Words: 5 Characters:28

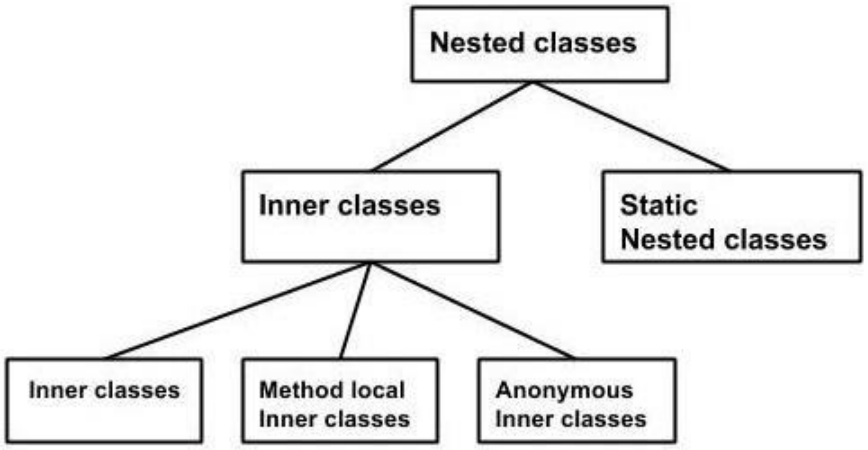
AWT and Events by Javatpoint

Java Inner Classes

Java inner class or nested class is a class which is declared inside the class or interface.

We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.

Additionally, it can access all the members of outer class including private data members and methods.



Syntax of Inner class

```
class Java_Outer_class{
 //code
 class Java_Inner_class{
  //code
 }
}
```

Advantage of java inner classes

There are basically three advantages of inner classes in java. They are as follows:

1) Nested classes represent a special type of relationship that is it can access all the members (data members and methods) of outer class including private.

2) Nested classes are used to develop more readable and maintainable code because it logically group classes and interfaces in one place only. 3) Code Optimization: It requires less code to write.

Types of Nested classes

There are two types of nested classes non-static and static nested classes.The non-static nested classes are also known as inner classes.

- **Non-static nested class (inner class)**
  1. Member inner class
  2. Anonymous inner class
  3. Local inner class ○ **Static nested class**

Type	Description
Member Inner Class	A class created within class and outside method.
Anonymous Inner Class	A class created for implementing interface or extending class. Its name is decided by the java compiler.
Local Inner Class	A class created within method.
Static Nested Class	A static class created within class.
Nested Interface	An interface created within class or interface.

Java Member inner class

A non-static class that is created inside a class but outside a method is called member inner class.

Syntax:

```
class Outer{
 //code
 class Inner{
  //code
 }
}
```

Java Member inner class example

In this example, we are creating msg() method in member inner class that is accessing the private data member of outer class.

```
class TestMemberOuter1{
 private int data=30;
 class Inner{
  void msg(){System.out.println("data is "+data);}
 }
 public static void main(String args[]){
  TestMemberOuter1 obj=new TestMemberOuter1();
  TestMemberOuter1.Inner in=obj.new Inner();
  in.msg();
 }
}
```

Output:

```
data is 30
```

Internal working of Java member inner class

The java compiler creates two class files in case of inner class. The class file name of inner class is "Outer$Inner". If you want to instantiate inner class, you must have to create the instance of outer class. In such case, instance of inner class is created inside the instance of outer class.

Java Anonymous inner class

A class that have no name is known as anonymous inner class in java. It should be used if you have to override method of class or interface. Java Anonymous inner class can be created by two ways:

1. Class (may be abstract or concrete).
2. Interface

Java anonymous inner class example using class

```
abstract class Person{
 abstract void eat();
}
class TestAnonymousInner{
 public static void main(String args[]){
  Person p=new Person(){
  void eat(){System.out.println("nice fruits");}
  };
  p.eat();
 }
}
```

Output:

```
nice fruits
```

Java Local inner class

A class i.e. created inside a method is called local inner class in java. If you want to invoke the methods of local inner class, you must instantiate this class inside the method.

Java local inner class example

```
public class localInner1{
 private int data=30;//instance variable
 void display(){
  class Local{
   void msg(){System.out.println(data);}
  }
  Local l=new Local();
  l.msg();
 }
 public static void main(String args[]){
  localInner1 obj=new localInner1();
  obj.display();
 }
}
```

Output:

```
30
```

Rule: Local variable can't be private, public or protected.

Rules for Java Local Inner class

1) Local inner class cannot be invoked from outside the method.

2) Local inner class cannot access non-final local variable till JDK 1.7. Since JDK 1.8, it is possible to access the non-final local variable in local inner class.

Example of local inner class with local variable

```
class localInner2{
 private int data=30;//instance variable
void display(){
  int value=50;//local variable must be final till jdk 1.7 only
class Local{
   void msg(){System.out.println(value);}
  }
  Local l=new Local();
l.msg();
 }
 public static void main(String args[]){
localInner2   obj=new   localInner2();
obj.display();
 }
}
```