EVENT DELEGATION MODEL

- **Event**
- **Source**
- **Listener**

➢ **What is Delegation Event Model**

In this model an event is generated by a source and is listened by a Listener

➢ **Event**

Change in the state of a source {ex: clicking of a button,mouse click etc}

➢ **Source**

The control which raises an event is known as a source

eg: button,mouse,radiobutton etc

➢ **Listener**

The one which listens to the event raised by a source and acts accordingly is called as a listener

eg: displaying text in a text box when a button is clicked

The Listener has to be registered to the source for performing the action,the method used for registration is

- **addTypeListener(TypeListener el)**

all sources = classes
all listeners=interfaces
all events=classes

- **Examples of some Event classes**
  - **ActionEvent**
  - **ItemEvent**
  - **MouseEvent**
  - **KeyEvent**

- **Examples of some Event Listeners**
  - **ActionListener**
  - **ItemListener**
  - **MouseListener & MouseMotionListener**
  - **KeyListener**

- **Handling Mouse Events**

  To handle mouse events,you must implement the MouseListener and MouseMotionListener Interfaces

- **MouseListener**
  - **mousePressed()**
  - **mouseReleased()**
  - **mouseClicked()**
  - **mouseEntered()**
  - **mouseExited()**

- **MouseMotionListener**
  - **mouseDragged()**
  - **mouseMoved()**

- **KeyHandling Events**

  To handle key events,you must implement KeyListener interface

- **KeyListener methods**
  - **keyPressed()**
  - **keyReleased()**
  - **keyTyped()**

PROGRAM FOR MOUSEHANDLING EVENTS

```
import java.awt.*;
import java.awt .event.*;
import java.applet.*;
/*
<html>
<body>
```

```
<applet code="Mousehandling.class" width=300
height=500>
</applet>
</body>
</html>
*/
public class Mousehandling extends Applet
implements
MouseListener,MouseMotionListener
{
String msg="";
int mousex=0,mousey=0;
public void init()
{
addMouseListener(this);
addMouseMotionListener(this);
}
public void mouseClicked(MouseEvent me)
{
mousex=0;
mousey=10;
msg="mouse clicked";
```

```
repaint();
}
public void mouseEntered(MouseEvent me)
{
mousex=0;
mousey=10;
msg="mouse entered";
repaint();
}
public void mouseExited(MouseEvent me)
{
mousex=0;
mousey=10;
msg="mouse exited";
repaint();
}

public void mousePressed(MouseEvent me)
{
mousex=me.getX();
mousey=me.getY();
msg="down";
```

```
repaint();
}
public void mouseReleased(MouseEvent me)
{
mousex=me.getX();
mousey=me.getY();
msg="UP";
repaint();
}
public void mouseDragged(MouseEvent me)
{
mousex=me.getX();
mousey=me.getY();
msg="*";
showStatus("Dragging mouse at
"+mousex+","+mousey);
repaint();
}
public void mouseMoved(MouseEvent me)
{
 showStatus("Moving mouse at
"+me.getX()+","+me.getY());
```

```
}
public void paint(Graphics g)
{
g.drawString(msg,mousex,mousey);
}
}
```

```
//PROGRAM FOR KEYHANDLING EVENTS
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<html>
<body>
<applet code="keyhandling" width=300
height=500>
</applet>
</body>
</html>*/
public class keyhandling extends Applet
implements KeyListener
```

```
{
 String msg="";
 int x=10,y=20;
 public void init()
 {
 addKeyListener(this);
 }
 public void keyPressed(KeyEvent ke)
 {
  showStatus("key down");
  }

 public void keyReleased(KeyEvent ke)
 {
  showStatus("key up");
  }

 public void keyTyped(KeyEvent ke)
 {
 msg+=ke.getKeyChar();
 repaint();
 }
```

```
    public void paint(Graphics g)
    {
    g.drawString(msg,x,y);
}}
```

Adapter Classes

- Java provides a special feature, called an *adapter class,* that can simplify the creation of event handlers in certain situations.
- An adapter class provides an empty implementation of all methods in an event listener interface.
- Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface.
- You can define a new class to act as an event listener by extending one of the adapter classes and implementing only those events in which you are interested.
- For example, the **MouseMotionAdapter** class has two methods, **mouseDragged( )** and **mouseMoved( )**, which are the methods defined by the **MouseMotionListener** interface. If you were interested in only mouse drag events, then you could simply extend **MouseMotionAdapter** and override **mouseDragged( )**. The empty implementation of **mouseMoved( )** would handle the mouse motion events for you.

Adapter Class	Listener Interface
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener

EXAMPLE PROGRAM

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
/*<applet code="AdapterDemo" width=500 height=500>
</applet>
*/
public class AdapterDemo extends Applet
{
```

```
  String msg="";
   public void init()
{
  addMouseListener(new MyMouseAdapter(this));
  addMouseMotionListener(new MyMouseMotionAdapter(this));
}
}
class MyMouseAdapter extends MouseAdapter
{
  AdapterDemo ad;
public MyMouseAdapter(AdapterDemo ob)
{
ad=ob;
}
public void mouseClicked(MouseEvent me)
{
 ad.showStatus("MouseClicked");
}
```

```
}

class MyMouseMotionAdapter extends MouseMotionAdapter
{
  AdapterDemo ad;
public MyMouseMotionAdapter(AdapterDemo ob)
{
ad=ob;
}
public void mouseDragged(MouseEvent me)
{
 ad.showStatus("MouseDragging");
}
}
```

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();
 }
}
```

Internal working of given code

```
Person p=new Person(){
void eat(){System.out.println("nice fruits");}
```

};

1. A class is created but its name is decided by the compiler which extends the Person class and provides the implementation of the eat() method.
2. An object of Anonymous class is created that is referred by p reference variable of Person type.

Internal class generated by the compiler

```
import java.io.PrintStream;
static class TestAnonymousInner$1 extends Person
{
  TestAnonymousInner$1(){}
  void eat()
   {
     System.out.println("nice fruits");     }  }
```

Java anonymous inner class example using interface

```
interface Eatable{
 void eat();
}
class TestAnnonymousInner1{
 public static void main(String args[]){
 Eatable e=new Eatable(){
```

```
  public void eat(){System.out.println("nice
fruits");}
 };
 e.eat();
 }
}
```