Event Handling

Any change in the state of any object is called event. For Example: Pressing a button, en- tering a character in Textbox, Clicking or dragging a mouse, etc. The three main components in event handling are:

- *Events:* An event is a change in state of an object. For example, mouseClicked, mousePressed.
- *Events Source:* Event source is an object that generates an event. Example: a button, frame, textfield.
- *Listeners:* A listener is an object that listens to the event. A listener gets notified when an event occurs. When listener receives an event, it process it and then return. Listeners are group of interfaces and are defined in java.awt.event package. Each component has its own listener. For example MouseListener handles all MouseEvent.

Some of the event classes and Listener interfaces are listed below.

Event Classes	Generated when	Listener Interfaces
ActionEvent	button is pressed, menu-item is selected, list-item is double clicked	Action Listener
MouseEvent	mouse is dragged, moved, clicked, pressed or released and also when it enters or exit a component	Mouse Listener and Mouse Motion Listener
MouseWheelEvent	mouse wheel is moved	Mouse Wheel Listener
KeyEvent	input is received from keyboard	Key Listener
ItemEvent	check-box or list item is clicked	Item Listener
TextEvent	value of textarea or textfield is changed	Text Listener
AdjustmentEvent	scroll bar is manipulated	Adjustment Listener
WindowEvent	window is activated, deactivated, deico- nified, iconified, opened or closed	Window Listener
ComponentEvent	component is hidden, moved, resized or set visible	Component Listener
ContainerEvent	component is added or removed from container	Container Listener
FocusEvent	component gains or losses keyboard focus	Focus Listener

Java program for handling keyboard events.

Test.java

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

```
import java.awt.*;
//Implementing KeyListener interface to handle keyboard
events public class Test extends Applet implements
KeyListener
{
 String msg="";
 public void
 init()
 {
  addKeyListener(this);            //use keyListener to monitor key events
 }
 public void keyPressed(KeyEvent k)      // invoked when any key is pressed down
 {
  showStatus("KeyPressed");
 }
 public void keyReleased(KeyEvent k)    // invoked when key is released
 {
  showStatus("KeyRealesed");
 }
//keyTyped event is called first followed by key pressed or key released event
 public void keyTyped(KeyEvent k)        //invoked when a textual key is pressed
 {
  msg = msg+k.getKeyChar(); repaint();
 }
 public void paint(Graphics g)
 {
  g.drawString(msg, 20, 40);
 }
}
```

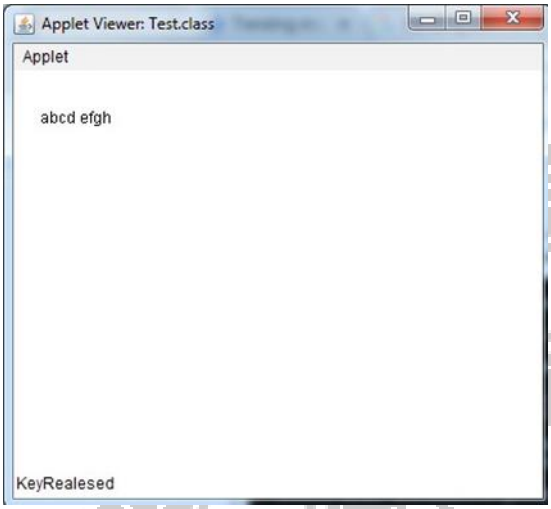
Test1.html

```
<html>
<body>
<applet code="Test.class" width="400" height="300">
```

</applet>

</body>

</html>



Adapter Classes

An adapter class provides the default implementation of all methods in an event listener interface. Adapter classes are very useful when you want to process only few of the events that are handled by a particular event listener interface. For example MouseAdapter provides empty implementation of MouseListener interface. It is useful because very often you do not really use all methods declared by interface, so implementing the interface directly is very lengthy.

- Adapter class is a simple java class that implements an interface with only EMPTY implementation.
- Instead of implementing interface if we extends Adapter class ,we provide implementation only for require method

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 as follows.

Adapter Class	Listener Interface
Window Adapter	Window Listener
Key Adapter	Key Listener
Mouse Adapter	Mouse Listener
Mouse Motion Adapter	Mouse Motion Listener
Focus Adapter	Focus Listener
Component Adapter	Component Listener
Container Adapter	Container Listener

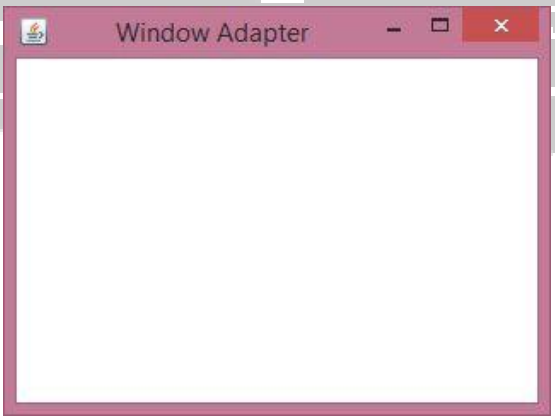
HierarchyBoundsAdapter	HierarchyBoundsListener

Example:

```
import java.awt.*;
import
java.awt.event.*;
public class
  AdapterExample{ Frame f;
  AdapterExample(){
    f=new Frame("Window Adapter");
    f.addWindowListener(new WindowAdapter(){
      public void windowClosing(WindowEvent e) {
        f.dispose();
      }
    });

    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
  }
public static void main(String[] args)
  { new AdapterExample();
}
}
```

Sample Output:



Actions

The Java Action interface and AbstractAction class are terrific ways of

encapsulating be- haviors (logic), especially when an action can be triggered from more than one place in your Java/Swing application.

javax.swing

Interface Action

An Action can be used to separate functionality and state from a component. For example, if you have two or more components that perform the same function, consider using an Action object to implement the function.

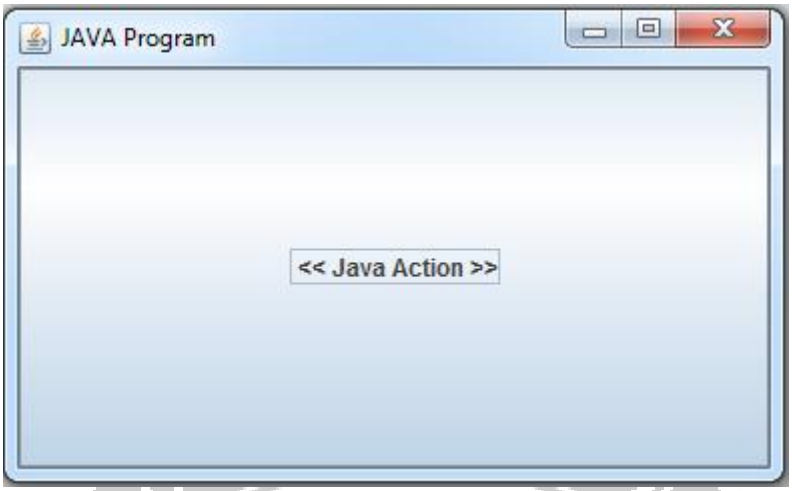
An Action object is an action listener that provides not only action-event handling, but also centralized handling of the state of action-event-firing components such as tool bar buttons, menu items, common buttons, and text fields. The state that an action can handle includes text, icon, mnemonic, enabled, and selected status.

The most common way an action event can be triggered from multiple places in a Java/ Swing application is through the Java menubar (JMenuBar) and toolbar (JToolBar)

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
 public class ButtonAction {
    private static void createAndShowGUI()  {
      JFrame frame1 = new JFrame("JAVA Program");
     frame1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     JButton button = new JButton(" << Java Action >>");
     //Add action listener to button
     button.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e)
        {
           System.out.println("You clicked the button");
        }
     });
      frame1.getContentPane().add(button);
     frame1.pack();
     frame1.setVisible(true);
   }
    public static void main(String[] args) {
      javax.swing.SwingUtilities.invokeLater(new Runnable() {
         public void run() {
           createAndShowGUI();
```

```
        }
} });
```


ROHINI COLLEGE OF ENGINEERING AND TECHNOLOGY
PALKULAM, KANYAKUMARI
OBSERVE OPTIMIZE OUTSPREAD


JAVA Program
<< Java Action >>