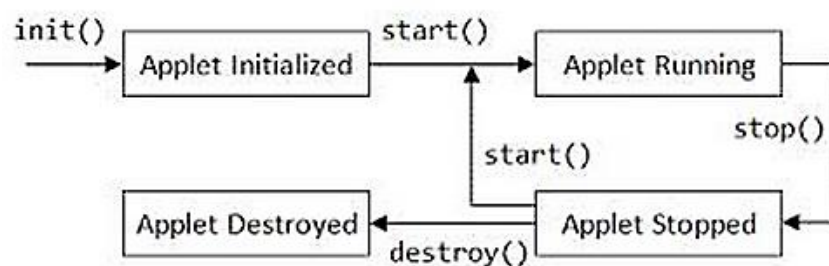


Applet Programming

An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal. There are certain differences between Applet and Java Standalone Application that are described below:

1. An applet is a Java class that extends the `java.applet.Applet` class.
2. A `main()` method is not invoked on an applet, and an applet class will not define `main()`.
3. Applets are designed to be embedded within an HTML page.
4. When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.
5. A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
6. The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
7. Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.

Life Cycle of Applet



Different methods that are used in Applet Life Cycle:

1. **init() Method:** This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.

2. **start() method:** This method is automatically called after the browser calls the init() method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.
3. **stop() method:** This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.
4. **destroy() method:** This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.
5. **paint() method:** Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

Simple Example to Create an Applet Program

To create an Applet program follow the steps:

1. Create a Java file containing Applet Code and Methods described above
2. Create a HTML file and embed the .Class File of the Java file created in the first step
3. Run Applet using either of the following methods
 - Open the HTML file in java enabled web browser
 - Use AppletViewer tool(used only for testing purpose)

//Code of MyFirstApplet.java

```
import java.applet.*;
import java.awt.*;
public class MyFirstApplet extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("MyFirstApplet Program",100,100);
    }
}
```

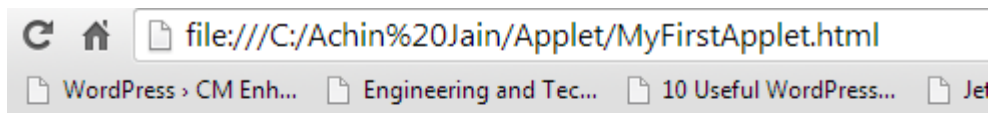
//Code of MyFirstApplet.html

```
<html>
<title>MyFirstApplet</title>
<applet code="MyFirstApplet.class" width=400 height=400>
</applet>
</html>
```

Embedding the .Class File of Java Code in HTML File

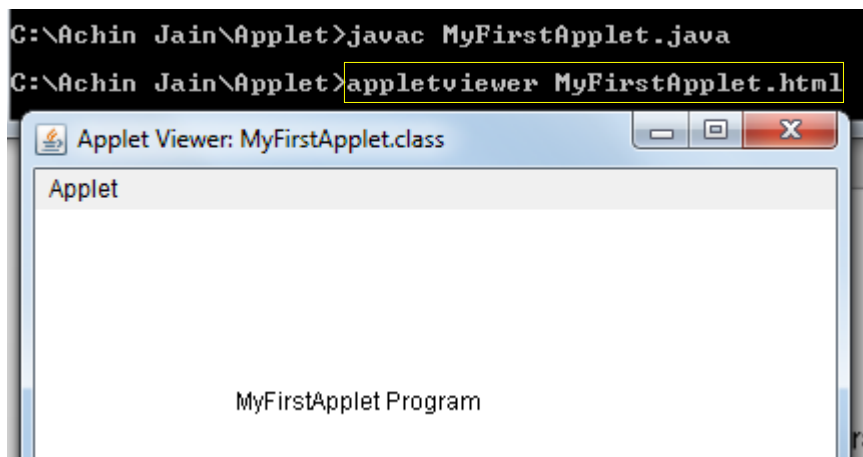
Output

Method – 1: Open the HTML File in Java Enabled Web Browser



MyFirstApplet Program

Method – 2: Use Appletviewer Tool



Passing Parameter to Applets

<PARAM.> tag is used to pass the parameter value from HTML file to Applet code. In the example shown below parameter which is used is named as “*name*” and value of the parameter is initialized as “*Achin Jain*”. Now in the Applet code, you should use same parameter name as “*name*” to fetch the value.

//Code of PARAM Test.html

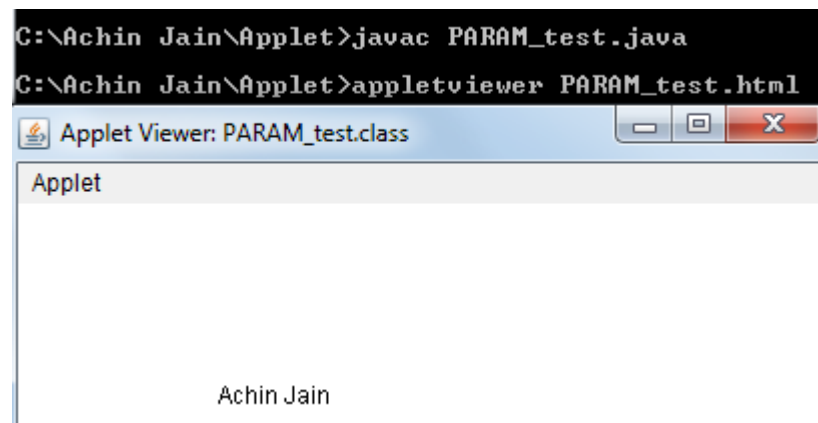
```
<html>
<title>PARAM_test</title>
<applet code="PARAM_test.class" width=400 height=400>
<PARAM NAME="name" VALUE="Achin Jain">
</applet>
</html>
```

//Code of PARAM Test.java

```
import java.applet.*;
import java.awt.*;
public class PARAM_test extends Applet
{
    String str=null;
    public void init()
    {
        str=getParameter("name");
    }
    public void paint(Graphics g)
    {
        g.drawString(str,100,100);
    }
}
```

PARAM Name is used as "*name*" which should be passed as parameter in getParameter() method to fetch the value of the parameter

Output



Getting Input from User

As you know that Applets works in Graphics environment where inputs are treated as Strings. So in order to get input from user, we need to create *TextField* just we used to do in HTML Coding. Below is the example that takes input from user as Number and second TextField displays the Square of that number.

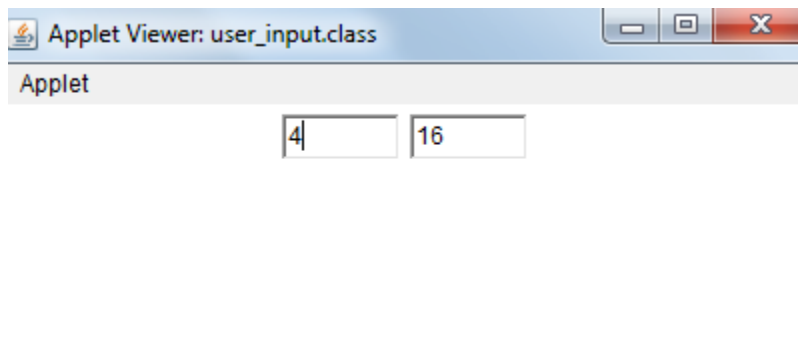
//Code of user_input.java

```
import java.applet.*;
import java.awt.*;
public class user_input extends Applet
{
    TextField t1,t2;
    public void init()
    {
        t1 = new TextField(5);
        t2 = new TextField(5);
        add(t1);
        add(t2);
    }
    public void paint(Graphics g)
    {
        int x=0;
        x=Integer.parseInt(t1.getText());
        t2.setText(String.valueOf(x*x));
    }
}
```

//Code of user_input.html

```
html>
<title>Input from User</title>
<applet code="user_input.class" width=400 height=400>
</applet>
</html>
```

Output



Example to Display Image in Applet

An applet can display images of the format GIF, JPEG, BMP, and others. To display an image within the applet, you use the drawImage() method found in the java.awt.Graphics class.

//Code of Applet image.java

```
import java.applet.*;
import java.awt.*;
public class Applet_image extends Applet
{
    Image img1;
    public void init()
    {
        img1 = getImage(getDocumentBase(), "java.jpg");
    }
    public void paint(Graphics g)
    {
        g.drawImage(img1, 100, 100, this);
    }
}
```

Returns the URL of the document in which applet is embedded

//Code of Applet image.html

```
html>
<title>Display Image in Applet</title>
<applet code="Applet_image.class" width=400 height=400>
</applet>
</html>
```

Output



Example to Play Audio in Applet

An applet can play an audio file represented by the AudioClip interface in the java.applet package. The AudioClip interface has three methods, including:

1. **public void play():** Plays the audio clip one time, from the beginning.
2. **public void loop():** Causes the audio clip to replay continually.
3. **public void stop():** Stops playing the audio clip.

//Code of Applet_audio.java

```
import java.applet.*;
import java.awt.*;
public class Applet_audio extends Applet
{
    AudioClip clip1;
    public void init()
    {
        clip1 = getAudioClip(getDocumentBase(), "au.mp3");
    }
    public void paint(Graphics g)
    {
        clip1.play(); // the clip is restarted from the beginning
        clip1.stop(); // Stops playing this audio clip.
        clip1.loop(); // Starts playing this audio clip in a loop.
    }
}
```

AWT and Graphics Programming

AWT UI Elements

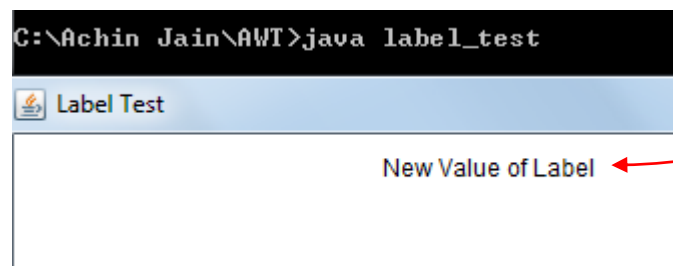
1. Label

Label is a passive control because it does not create any event when accessed by the user. The label control is an object of Label. A label displays a single line of read-only text. However the text can be changed by the application programmer but cannot be changed by the end user in any way.

Example of Label

```
import java.awt.*;
class label_test extends Frame
{
    Frame f;
    Label l;
    label_test()
    {
        f = new Frame("Label Test");
        l = new Label("Hello Java");
        String str = l.getText();//returns the string value associated with label
        l.setText("New Value of Label");//sets the string value of label
        l.setBounds(100,200,300,400);
        f.add(l);//adds the label component to frame
        f.setVisible(true);
        f.setLayout(new FlowLayout());
        f.setSize(500,500);
    }
    public static void main(String args[])
    {
        label_test obj1 = new label_test();
    }
}
```

Output



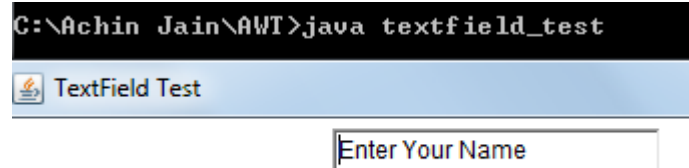
2. TextField

The textfield component allows the user to edit single line of text. When the user types a key in the text field the event is sent to the TextField. The key event may be key pressed, Key released or key typed.

Example

```
import java.awt.*;
class textfield_test extends Frame
{
    Frame f;
    TextField tf;
    textfield_test()
    {
        f = new Frame("TextField Test");
        tf = new TextField(20);
        String str = tf.getText();//returns the string value associated with TextField
        tf.setText("Enter Your Name");//sets the string value of TextField
        tf.setBounds(100,200,300,400);
        f.add(tf);//adds the TextField component to frame
        f.setVisible(true);
        f.setLayout(new FlowLayout());
        f.setSize(500,500);
    }
    public static void main(String args[])
    {
        textfield_test obj1 = new textfield_test();
    }
}
```

Output



3. Button

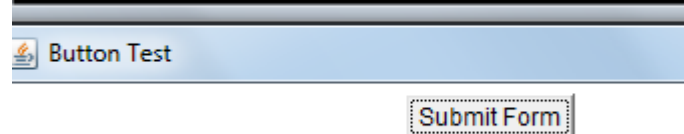
Button is a control component that has a label and generates an event when pressed. When a button is pressed and released, AWT sends an instance of ActionEvent to the button, by calling processEvent on the button.

Example

```
import java.awt.*;
class button_test extends Frame
{
    Frame f;
    Button b1;
    button_test()
    {
        f = new Frame("Button Test");
        b1 = new Button("Submit Form");
        b1.setBounds(100,200,300,400);
        f.add(b1);//adds the Button component to frame
        f.setVisible(true);
        f.setLayout(new FlowLayout());
        f.setSize(500,500);
    }
    public static void main(String args[])
    {
        button_test obj1 = new button_test();
    }
}
```

Output

```
C:\Achin Jain\AWT>javac button_test.java
C:\Achin Jain\AWT>java button_test
```



4. CheckBox

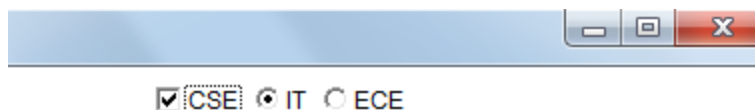
Checkbox control is used to turn an option on (true) or off (false). There is label for each checkbox representing what the checkbox does. The state of a checkbox can be changed by clicking on it. In AWT there is no Radio button class as in Swings, so to create Radio button CheckboxGroup class is used. The CheckboxGroup class is used to group the set of checkbox.

Example

```
import java.awt.*;
class checkbox_test extends Frame
{
    Frame f;
    Checkbox c1,c2,c3;
    CheckboxGroup cbg;
    checkbox_test ()
    {
        f = new Frame("Button Test");
        cbg = new CheckboxGroup();
        c1 = new Checkbox("CSE", null, true);
        c2 = new Checkbox("IT", cbg, true);
        c3 = new Checkbox("ECE", cbg, false);
        f.add(c1); //adds the Button component to frame
        f.add(c2);
        f.add(c3);
        f.setVisible(true);
        f.setLayout(new FlowLayout());
        f.setSize(500,500);
    }
    public static void main(String args[])
    {
        checkbox_test obj1 = new checkbox_test();
    }
}
```

If Checkbox constructor doesn't contains CheckboxGroup(as in 'c1') name as parameter then it will work as normal checkbox where if name is passed as in case of 'c2' and 'c3' they will behave as Radio Buttons.

Output



5. List

The List represents a list of text items. The list can be configured to that user can choose either one item or multiple items.

Example

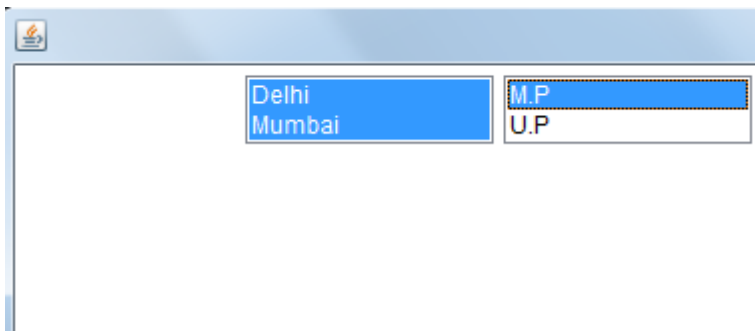
```
import java.awt.*;
class list_test extends Frame
{
    Frame f;
    List city, state;
    list_test()
    {
        f = new Frame();
        city = new List(2, true);
        state = new List(2, false);
        city.add("Delhi");
        city.add("Mumbai");
        state.add("M.P");
        state.add("U.P");
        f.setVisible(true);
        f.setLayout(new FlowLayout());
        f.setSize(500,500);
        f.add(city);
        f.add(state);
    }
    public static void main(String args[])
    {
        list_test obj = new list_test();
    }
}
```

Creates a list with 2 items where multi select is allowed

Creates a list with 2 items where multi select is not allowed

Adding List Items with 'add' method

Output



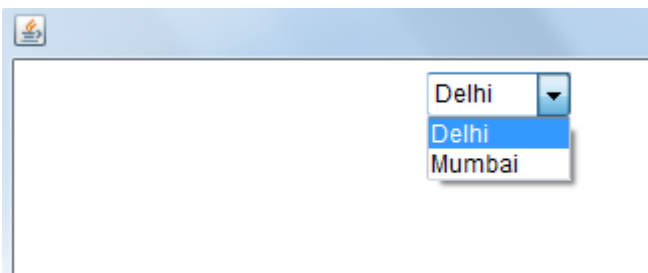
6. Choice

Choice control is used to show pop up menu of choices. Selected choice is shown on the top of the menu. It is same as List the only difference is with the appearance of the control on window.

Example

```
import java.awt.*;
class choice_test extends Frame
{
    Frame f;
    Choice city;
    choice_test()
    {
        f = new Frame();
        city = new Choice();
        city.add("Delhi");
        city.add("Mumbai");
        f.setVisible(true);
        f.setLayout(new FlowLayout());
        f.setSize(500,500);
        f.add(city);
    }
    public static void main(String args[])
    {
        choice_test obj = new choice_test();
    }
}
```

Output



7. TextArea

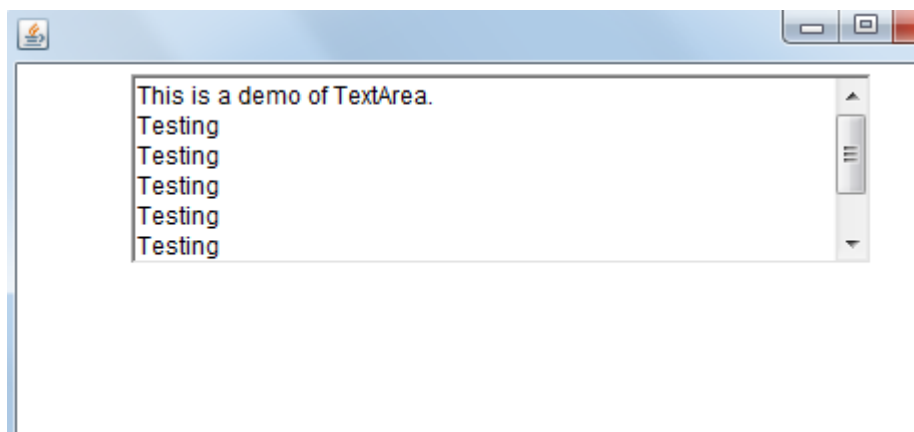
The TextArea control in AWT provide us multiline editor area. The user can type here as much as he wants. When the text in the text area become larger than the viewable area the scroll bar is automatically appears which help us to scroll the text up & down and right & left.

Example

```
import java.awt.*;
class textarea_test extends Frame
{
    Frame f;
    TextArea ta;
    textarea_test()
    {
        f = new Frame();
        ta = new TextArea(5,50);
        f.setVisible(true);
        f.setLayout(new FlowLayout());
        f.setSize(500,500);
        f.add(ta);
    }
    public static void main(String args[])
    {
        textarea_test obj = new textarea_test();
    }
}
```

Definition: TextArea (int rows, int columns)

Output



8. ScrollBar

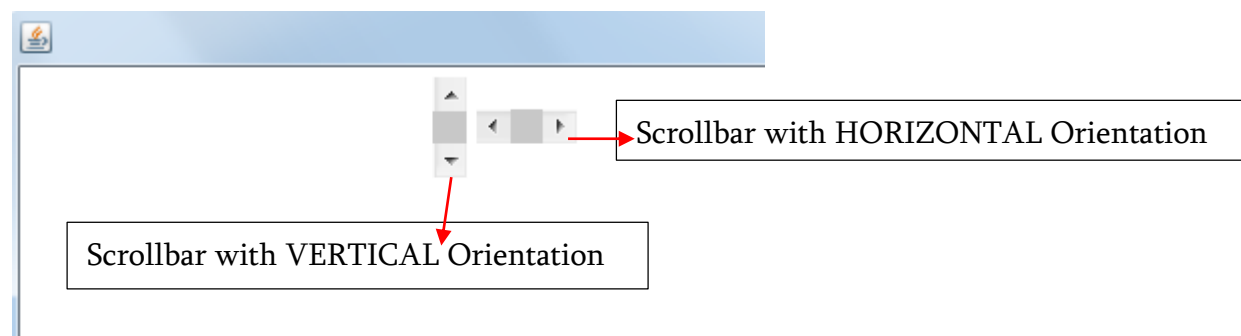
Scrollbar control represents a scroll bar component in order to enable user to select from range of values.

1. **static int HORIZONTAL** --A constant that indicates a horizontal scroll bar.
2. **static int VERTICAL** --A constant that indicates a vertical scroll bar.

```
import java.awt.*;
class scrollbar_test extends Frame
{
    Frame f;
    Scrollbar sb, sb1;
    scrollbar_test()
    {
        f = new Frame();
        sb = new Scrollbar();
        sb.setOrientation(Scrollbar.VERTICAL); //Set Orientation of the Scrollbar
        sb1 = new Scrollbar(Scrollbar.HORIZONTAL);
        f.setVisible(true);
        f.setLayout(new FlowLayout());
        f.setSize(500,500);
        f.add(sb);
        f.add(sb1);
    }
    public static void main(String args[])
    {
        scrollbar_test obj = new scrollbar_test();
    }
}
```

You can also set orientation of scrollbar in construction also.

Output

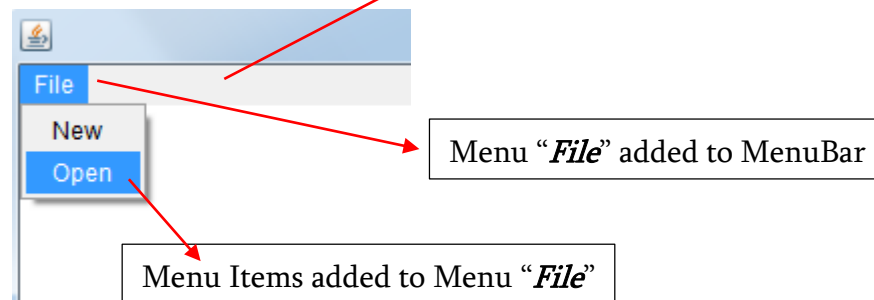


9. Menus in JAVA

Example

```
import java.awt.*;
class menu_test extends Frame
{
    menu_test()
    {
        MenuBar mbar = new MenuBar(); //Creates new MenuBar to hold Menu
        Frame f = new Frame();
        f.setMenuBar(mbar); //Frame is set to use MenuBar(mbar)
        f.setLayout(new FlowLayout());
        f.setVisible(true);
        f.setSize(500,500);
        Menu File = new Menu("File"); //Creates new Menu File with label "File"
        MenuItem i1,i2; //Two menuitems are created to add in Menu created above
        File.add(new MenuItem("New")); //add menuItem i1 to menu "File"
        File.add(new MenuItem("Open")); //add menuItem i2 to menu "File"
        mbar.add(File); //add Menu File to MenuBar
    }
    public static void main(String args[])
    {
        menu_test obj = new menu_test();
    }
}
```

Output



Event Handling

Event is the change in the state of the object or 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.

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. The *Delegation Event Model* has the following key participants namely:

1. **Source** - The source is an object on which event occurs. Source is responsible for providing information of the occurred event to its handler. Java provide as with classes for source object.
2. **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 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.

Event Handling Examples

1. ActionListener Example

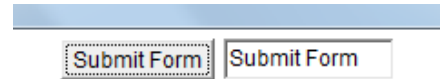
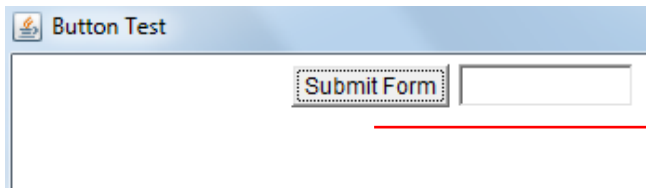
This interface is used to handle the events caused by sources like Buttons, Menu Items, Enter Keys and Double Click of Mouse. When you implements ActionListener Interface following methods needs to be override.

```
public void actionPerformed(ActionEvent ae)
```

Example

```
import java.awt.*;
import java.awt.event.*;
class button_event extends Frame implements ActionListener
{
    Frame f;
    Button b1;
    TextField t1;
    button_event()
    {
        f = new Frame("Button Test");
        b1 = new Button("Submit Form");
        t1 = new TextField(10);
        f.add(b1);
        f.add(t1);
        f.setVisible(true);
        f.setLayout(new FlowLayout());
        f.setSize(500,500);
        b1.addActionListener(this); //Registration of Button with Event Handler
    }
    public void actionPerformed(ActionEvent ae) //Override actionPerformed Method
    {
        String str = ae.getActionCommand(); //This method returns string value of button
        t1.setText(str); //setting the text-field value with button string value
    }
    public static void main(String args[])
    {
        button_event obj1 = new button_event();
    }
}
```

Output



After Button is clicked

2. ItemListener Example

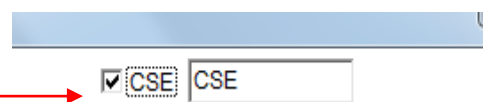
Used for Radio Button, List, Choice and Check Box AWT Controls. The method that needs to be override is as follows:

```
public void itemStateChanged(ItemEvent e)
```

Example

```
class checkbox_test extends Frame implements ItemListener
{
    Frame f;
    Checkbox c1;
    TextField t1;
    checkbox_test ()
    {
        f = new Frame("Button Test");
        c1 = new Checkbox("CSE", null, false);
        t1 = new TextField(10);
        f.add(c1);
        f.add(t1);
        f.setVisible(true);
        f.setLayout(new FlowLayout());
        f.setSize(500,500);
        c1.addItemListener(this); //Registration of Checkbox with ItemListener Interface
    }
    public void itemStateChanged(ItemEvent ie)
    {
        if(c1.getState()) //Returns true if Checkbox 'c1' is selected else returns false
        {
            t1.setText("CSE");
        }
    }
}
```

Output



After CheckBox is selected

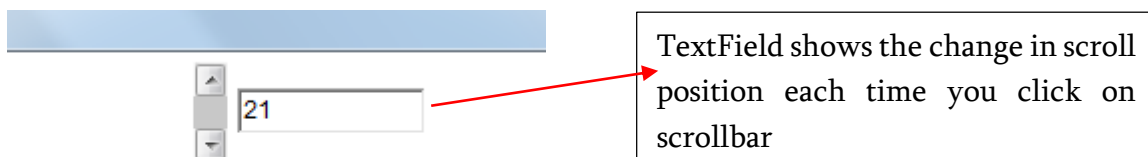
3. AdjustmentListener Example

Used to handle events generated from Scrollbar source. This interface has only method which is: public void adjustmentValueChanged(AdjustmentEvent e)

Example

```
class scrollbar_test extends Frame implements AdjustmentListener
{
    Frame f;
    Scrollbar sb;
    TextField t1;
    scrollbar_test()
    {
        f = new Frame();
        sb = new Scrollbar();
        t1 = new TextField(10);
        f.setVisible(true);
        f.setLayout(new FlowLayout());
        f.setSize(500,500);
        f.add(sb);
        f.add(t1);
        sb.addAdjustmentListener(this); //Registration of Scrollbar object with Interface
    }
    public void adjustmentValueChanged(AdjustmentEvent ae)
    {
        int i = sb.getValue(); //getValue() method returns the current scroll value
        t1.setText(String.valueOf(i));
    }
}
```

Output



4. KeyListener Example

This interface is used to handle the events generated from keys of the keyboard. There are three methods present in the KeyListener Interface and all methods needs to be override.

1. public void keyPressed(KeyEvent ke)
2. public void keyTyped(KeyEvent ke)
3. public void keyReleased(KeyEvent ke)

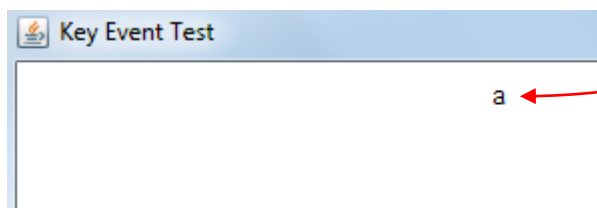
Example

```

class key_event extends Frame implements KeyListener
{
    Label l1;
    Frame f;
    key_event ()
    {
        f = new Frame ("Key Event Test");
        l1 = new Label ();
        f.addKeyListener (this); //Registration of KeyListener with Frame
        f.add(l1);
        f.setVisible (true);
        f.setLayout (new FlowLayout ());
        f.setSize (500,500);
    }
    public void keyTyped(KeyEvent ke) {}
    public void keyReleased(KeyEvent ke) {}
    public void keyPressed(KeyEvent ke)
    {
        char ch = ke.getKeyChar (); //Returns the Pressed Key Char
        l1.setText (String.valueOf (ch));
    }
}
    
```

Blank Definition of the methods despite them are not used. It is necessary since we are using interface

Output



5. MouseMotionListener Example

As the name suggests this interface is used to handle events generated from Mouse Source.

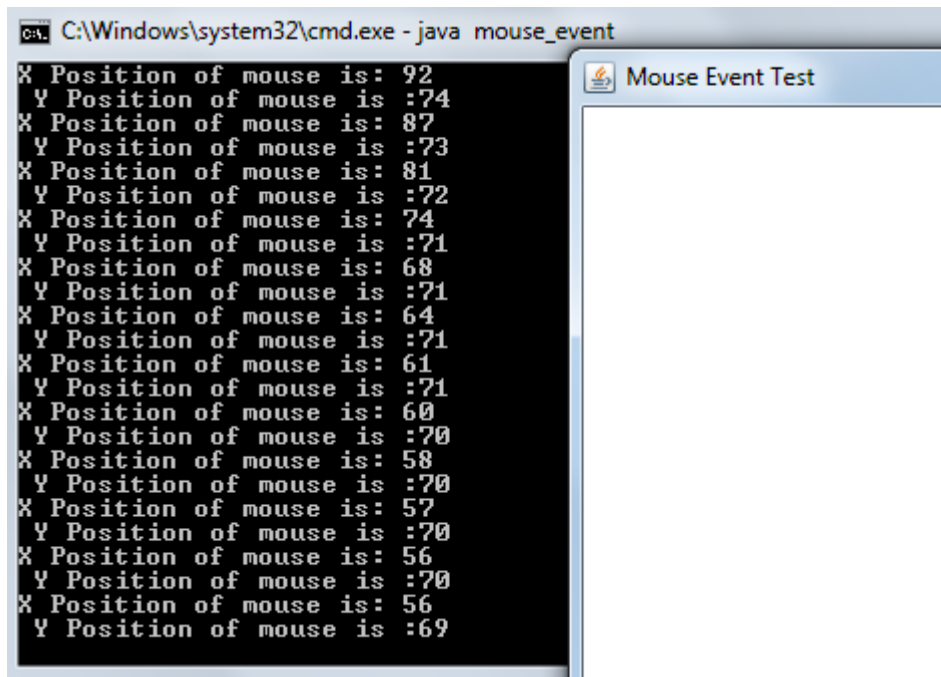
Methods are as follows:

1. public void mouseMoved(MouseEvent me)
2. public void mouseDragged(MouseEvent me)

Example

```
import java.awt.*;
import java.awt.event.*;
class mouse_event extends Frame implements MouseMotionListener
{
    Frame f;
    mouse_event ()
    {
        f = new Frame("Mouse Event Test");
        f.addMouseListener(this); //Registration of MouseMotionListener with Frame
        f.setVisible(true);
        f.setLayout(new FlowLayout());
        f.setSize(500,500);
    }
    public void mouseDragged(MouseEvent me) {} //Empty Definition of the Method
    public void mouseMoved(MouseEvent me)
    {
        int x = me.getX(); // Returns the X coordinate position of mouse
        int y = me.getY(); // Returns the Y coordinate position of mouse
        System.out.println("X Position of mouse is: "+x+"\n Y Position of mouse is :"+y);
    }
    public static void main(String args[])
    {
        mouse_event obj = new mouse_event ();
    }
}
```

Output



```
C:\Windows\system32\cmd.exe - java mouse_event
X Position of mouse is: 92
Y Position of mouse is :74
X Position of mouse is: 87
Y Position of mouse is :73
X Position of mouse is: 81
Y Position of mouse is :72
X Position of mouse is: 74
Y Position of mouse is :71
X Position of mouse is: 68
Y Position of mouse is :71
X Position of mouse is: 64
Y Position of mouse is :71
X Position of mouse is: 61
Y Position of mouse is :71
X Position of mouse is: 60
Y Position of mouse is :70
X Position of mouse is: 58
Y Position of mouse is :70
X Position of mouse is: 57
Y Position of mouse is :70
X Position of mouse is: 56
Y Position of mouse is :70
X Position of mouse is: 56
Y Position of mouse is :69
```

7. MouseListener Example

Methods are as follows:

1. public void mousePressed(MouseEvent me)
2. public void mouseClicked(MouseEvent me)
3. public void mouseReleased(MouseEvent me)
4. public void mouseEntered(MouseEvent me)
5. public void mouseExited(MouseEvent me)

Adapter Classes

Adapter classes are used to simplify the process of event handling in Java. As we know that when we implement any interface all the methods defined in that interface needs to be override in the class, which is not desirable in the case of Event Handling. Adapter classes are useful as they provide empty implementation of all methods in an event listener interface. In this you can define a new class to act as event listener by extending one of the

adapter classes and implementing only those methods that you want to use in your program.

Example of Adapter Class

In the above example of KeyListener and MouseMotionListener we have provided blank definition of all the methods. Below you can see the example of both programs with Adapter classes.

```

class mouse_adapter extends Frame
{
    Frame f;
    mouse_adapter()
    {
        f = new Frame("Mouse Event Test");
        f.addMouseMotionListener(new m1()); //Registration of MouseMotionListener with Adapter Class
        f.setVisible(true);
        f.setLayout(new FlowLayout());
        f.setSize(500,500);
    }
    class m1 extends MouseMotionAdapter
    {
        public void mouseMoved(MouseEvent me)
        {
            int x = me.getX(); // Returns the X coordinate position of mouse
            int y = me.getY(); // Returns the Y coordinate position of mouse
            System.out.println("X Position of mouse is: "+x+"\n Y Position of mouse is :"+y);
        }
    }
    public static void main(String args[])
    {
        mouse_adapter obj = new mouse_adapter();
    }
}

```

Now in the code you can see definition of single method which is required.

Anonymous Inner Class

It is a class that is not assigned any name.

Example

```

class anyony_test extends Frame
{
    Frame f;
    Button b;
    TextField t;
    anyony_test()
    {
        f = new Frame();
        f.setVisible(true);
        f.setLayout(new FlowLayout());
        f.setSize(500,500);
        t = new TextField(10);
        b = new Button("Click here");
        f.add(b);
        f.add(t);
        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                String str = ae.getActionCommand();
                t.setText(str);
            }
        });
    }
}

```

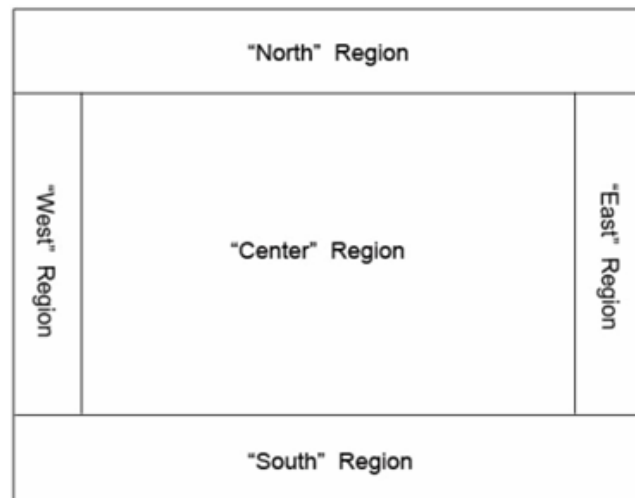
Layout Manager

The Layout manager are used to arrange components in a particular way.

1. Border Layout Manager

BorderLayout arranges the components to fit in the five regions: east, west, north, south and center. Each region is can contain only one component and each component in each

region is identified by the corresponding constant NORTH, SOUTH, EAST, WEST, and CENTER.

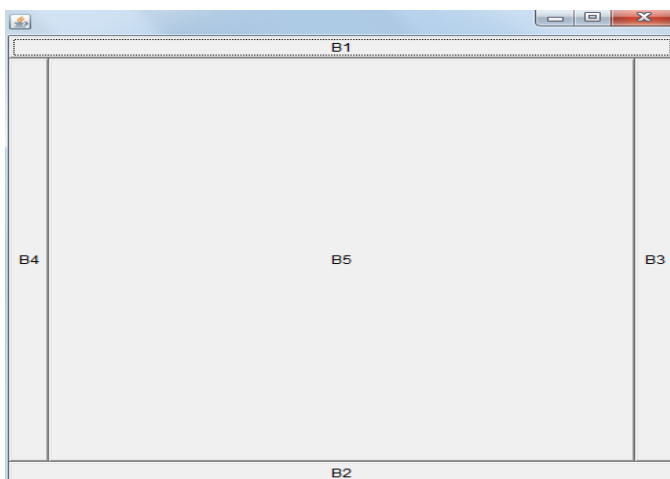


Example

Suppose you have created 5 buttons from b1...b5. To add each button specific to each direction see the following code.

```
f.setLayout(new BorderLayout()); //sets the Layout to BorderLayout
f.add(b1, "North"); //add Button "b1" to North Direction
f.add(b2, "South"); //add Button "b2" to South Direction
f.add(b3, "East"); //add Button "b3" to East Direction
f.add(b4, "West"); //add Button "b4" to West Direction
f.add(b5, "Center"); //add Button "b5" to Center Direction
```

Output



2. Card Layout

The class **CardLayout** arranges each component in the container as a card. Only one card is visible at a time, and the container acts as a stack of cards.

Example

```
class layout_test extends Frame implements ActionListener
{
    Frame f;
    Button b1,b2;
    CardLayout card;
    layout_test()
    {
        f = new Frame();
        card = new CardLayout(40,30);
        b1 = new Button("B1");
        b2 = new Button("B2");
        f.setSize(500,500);
        f.setVisible(true);
        f.setLayout(card); //sets the Layout to CardLayout
        b1.addActionListener(this);
        b2.addActionListener(this);
        f.add("Card1",b1);
        f.add("Card2",b2);
    }
    public void actionPerformed(ActionEvent ae)
    {
        card.next(f);
    }
}
```

Now when you add control to Frame use a string to add controls. For example in this code b1 is added to *card1* and b2 to *card2*.

next() method is used to display next card layout. In this code on click of button this method will evoke

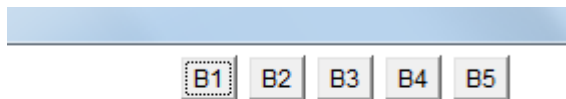
3. Flow Layout

The class **FlowLayout** components in a left-to-right flow. It is the default manager for panels and applets.

Example

```
f.setLayout(new FlowLayout()); //sets the layout to FlowLayout
f.add(b1);
f.add(b2);
f.add(b3);
f.add(b4);
f.add(b5);
```

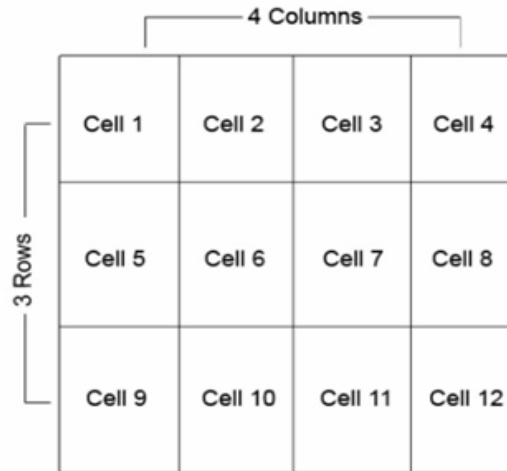
Output



4. Grid Layout

As the name indicates the container is divided into a **grid of cells** dictated by **rows** and **columns**. Each cell accommodates one component. Like with FlowLayout, in GridLayout also, the position need not be mentioned. But remember, in BorderLayout, position is to be mentioned. As the rows and columns are fixed, components addition goes one after another automatically.

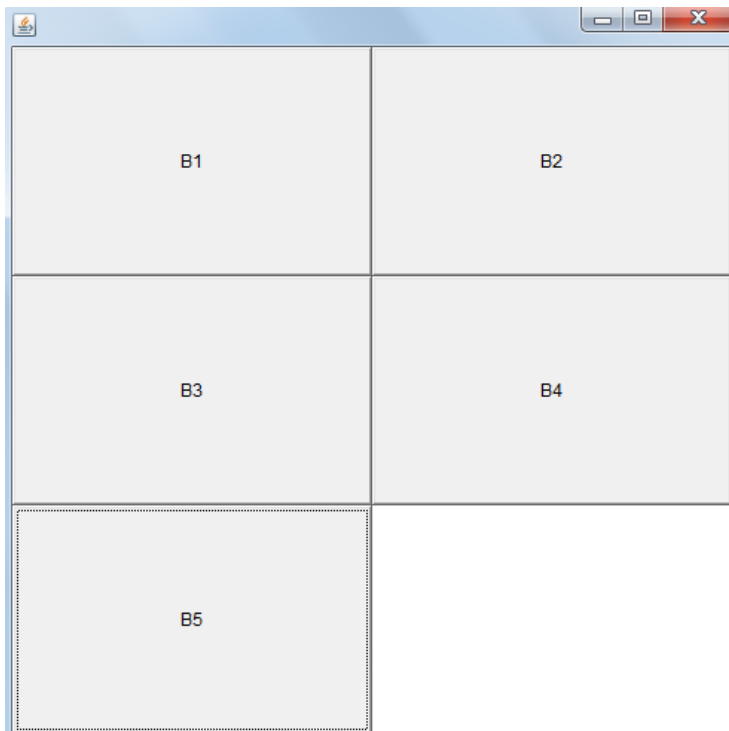
GridLayout of 3 rows and 4 columns (12 cells) is shown below in the figure.



Example

```
f.setLayout(new GridLayout(3,2)); //sets the Layout to GridLayout
f.add(b1);
f.add(b2);
f.add(b3);
f.add(b4);
f.add(b5);
```

Output



Swings

Swing is a part of JFC(Java Foundation Classes) that is used to create GUI application. It is built on the top of AWT and entirely written in java.

Advantages of Swing over AWT

1. Components are platform independent
2. Lightweight
3. Supports PLAF(pluggable look and feel)
4. Supports more powerful and useful components like table, scrollpanes, radiobutton, tabbedPane etc.
5. Follow MVC(Model View Controller) architecture

MVC Architecture of Swing

MVC means Model, View and Controller. Model depicts the state of the component which simple means whether the components is invoking any event or not. View means look and feel of the components on screen. Controller handles the event and controls how the code reacts to user query or action.

Swing Components

Component	Constructor Definition
JLabel	JLabel j1 = new JLabel("Label Name", ImageIcon, Align)
JTextField	JTextField jtf1 = new JTextField(Length)
JButton	JButton jb1 = new JButton("Button Label") – ActionListener is used to handle the events generated
JToggleButton(two states either push or release)	JToggleButton jtb1 = new JToggleButton("On/Off") – ItemListener is used to handle the events generated. Public void itemStateChanged(ItemEvent ie) { if(jtb1.isSelected() -> returns true if button is in push state else return false.

JCheckBox	JCheckBox jcb = new JCheckBox("Checkbox Label", true/false) – ItemListener is used to handle the events
JRadioButton	JRadioButton jrb = new JRadioButton("RadioButton Label", true/fase) ButtonGroup bg = new ButtonGroup(); bg.add(jrb);