

Module 4

Applets

- *Applets* are small applications that are accessed on an Internet server, transported over the Internet, automatically installed, and run as part of a Web document.
- An applet is a Panel that allows interaction with a Java program
- A applet is typically embedded in a Web page and can be run from a browser
- You need special HTML in the Web page to tell the browser about the applet
- You write an applet by extending the class Applet
- Applet is just a class like any other; you can even use it in applications if you want
- When you write an applet, you are only writing *part* of a program
- The browser supplies the main method

Two types of Applets

- Applet-uses AWT to provide These applets use the Abstract Window Toolkit (AWT) to provide the graphic user
- The second type of applets are those based on the Swing class JApplet.----Swing applets use the Swing classes to provide the GUI. Swing offers a richer and often easier-to-use user interface than does the AWT. Thus, Swing-based applets are most popular. However, traditional AWT-based applets are still used, especially when only a very simple user interface is required.

Differences between applets and applications

- Although both the Applets and stand-alone applications are Java programs, there are certain restrictions are imposed on Applets due to security concerns:
 - Applets don't use the main() method, but when they are load, automatically call certain methods (init, start, paint, stop, destroy).
 - They are embedded inside a web page and executed in browsers.
 - They cannot read from or write to the files on local computer.
 - They cannot communicate with other servers on the network.
 - They cannot run any programs from the local computer.
- The above restrictions ensures that an Applet cannot do any damage to the local system.

AWT (Abstract Window Toolkit)

- **AWT** stands for Abstract Window ToolKit and it supports Java GUI programming. It is a portable GUI library for Stand-alone Java applications/applets. The AWT provides the connection between our application and the native GUI .
- First of all, by a heavy-weight, it means the code will take comparatively more time to load and it will consume more System resources. AWT is considered to be heavy-weight because its components are dependent on the underlying Operating System. For instance, When we create an object of **java.awt.Checkbox** class, its underlying Operating System will generate a checkbox for us. This is also the reason, AWT components are platform dependent.

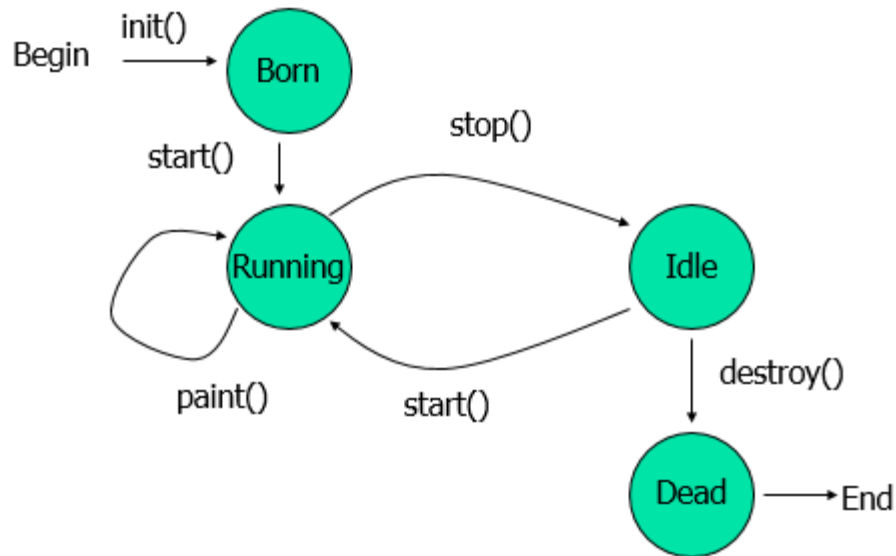
Swings

- **Java Swing** implements a set of GUI components that build on AWT technology and it can provide a **pluggable look and feel**. Java swings are light weight components and they are implemented entirely in the Java programming language.
- **Java Swing** is the advanced and optimized version of **AWT** and it is built on top of AWT. Still, many AWT classes are used in Swing either directly or indirectly.

The genealogy of Applet

```
java.lang.Object
|
+----java.awt.Component
|
+----java.awt.Container
|
+----java.awt.Panel
|
+----java.applet.Applet
```

Applet Life Cycle



public void init ()

- This is the first method to execute
- It is an ideal place to initialize variables
- It is the best place to define the GUI Components (buttons, text fields, scrollbars, etc.), lay them out, and add listeners to them
- Almost every applet you ever write will have an `init()` method

public void start ()

- Not always needed
- Called after `init()`
- Called each time the page is loaded and restarted
- Used mostly in conjunction with `stop()`

public void stop()

- Not always needed
- Called when the browser leaves the page
- Called just before `destroy()`
- Use `stop()` if the applet is doing heavy computation that you don't want to continue when the browser is on some other page
- Used mostly in conjunction with `start()`

public void destroy()

- Called after stop()
- Use to explicitly release system resources (like threads)
- System resources are usually released automatically

public void paint(Graphics g)

- Needed if you do any drawing or painting other than just using standard GUI Components
- Any painting you want to do should be done here, or in a method you call from here
- Painting that you do in other methods may *or may not* happen
- *Never call paint(Graphics), call repaint()*

repaint()

- Call repaint() when you have changed something and want your changes to show up on the screen
- repaint() is a *request*--it might not happen
- When you call repaint(), Java schedules a call to update(Graphics g)

public void update(Graphics g)


- When you call repaint(), Java schedules a call to update(Graphics g)
- Here's what update does:


```
public void update(Graphics g) {  
    // Fills applet with background color, then  
    paint(g);  
}
```

Sample Graphics methods


A Graphics is something you can paint on


`g.drawString("Hello", 20, 20);` Hello

`g.drawRect(x, y, width, height);` 

`g.fillRect(x, y, width, height);` 

`g.drawOval(x, y, width, height);` 

`g.fillOval(x, y, width, height);` 

`g.setColor(Color.red);` 

An Applet Skeleton

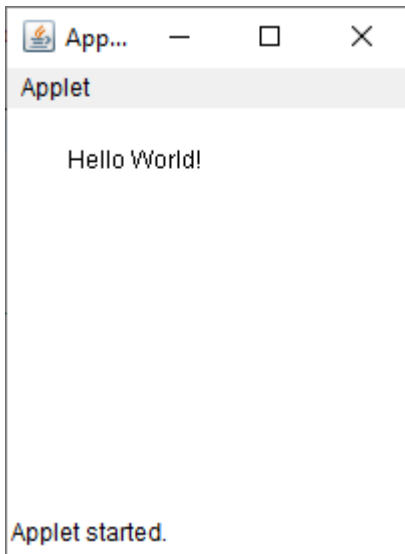
```
import java.awt.*;
import java.applet.*;
/*
<applet code="AppletSkel" width=300 height=100>
</applet>
*/
public class AppletSkel extends Applet {
// Called first.
public void init() {
// initialization
}
/* Called second, after init(). Also called whenever
the applet is restarted. */
public void start() {
// start or resume execution
}
// Called when the applet is stopped.
public void stop() {
// suspends execution
}
/* Called when applet is terminated. This is the last
method executed. */
public void destroy() {
// perform shutdown activities
}
// Called when an applet's window must be restored.
public void paint(Graphics g) {
// redisplay contents of window
}
}
```

// A Simple applet to say Hello

```
import java.applet.Applet;
import java.awt.Graphics;
```

```
public class HelloWorld extends Applet
{
    public void paint( Graphics g )
    {
        g.drawString( "Hello World!", 30, 30 );
    }
}
/*
<applet code="HelloWorld" width=400 height=400>
</applet>
*/
```

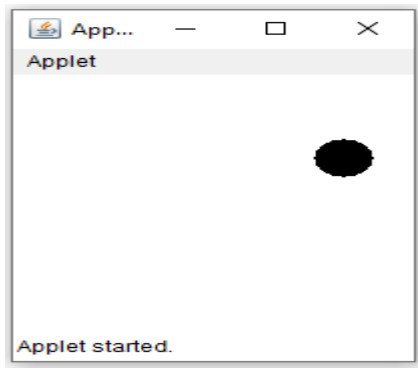
OUTPUT



// An Applet to demonstrate start() method (MOVING BALL)

```
import java.applet.Applet;
import java.awt.Graphics;
public class Test321 extends Applet
{
    int x=50;
    public void start()
    {
        x+=50;
    }
    public void paint(Graphics g)
    {
        g.fillOval(x,50,30,30);
    }
}
```

OUTPUT

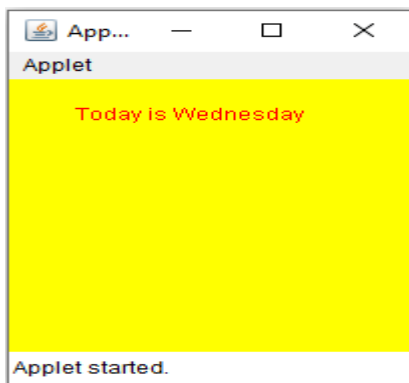


// An applet with foreground and background color set

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

public class applet2 extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString(" Today is Wednesday", 30,30);
        setBackground(Color.yellow);
        setForeground(Color.red);
    }
}
```

OUTPUT



// An applet with DrawingStuff

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

public class DrawingStuff extends Applet
{
    public void paint( Graphics g )
    {
        g.setColor( Color.red );
        g.drawRect( 10, 20, 100, 15 );
    }
}
```

```
g.setColor( Color.pink );
g.fillRect( 240, 160, 40, 110 );

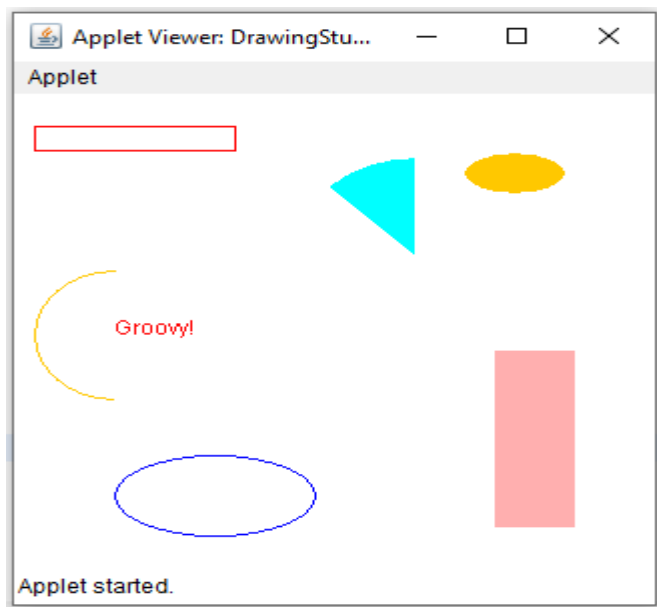
g.setColor( Color.blue );
g.drawOval( 50, 225, 100, 50 );
g.setColor( Color.orange );
g.fillOval( 225, 37, 50, 25 );

g.setColor( Color.orange );
g.drawArc( 10, 110, 80, 80, 90, 180 );
g.setColor( Color.cyan );
g.fillArc( 140, 40, 120, 120, 90, 45 );

g.setColor( Color.red );
g.drawString( "Groovy!", 50, 150 );
}

/*
<applet code=DrawingStuff.class width = 600 height = 600>
</applet>
*/
```

OUTPUT



//Program to develop an Applet that contains Buttons , TextFields and Labels

```
import java.awt.*;
import java.applet.Applet;
public class Buttonawt extends Applet
{
public void init( )
{
    Label l1=new Label( " Enter Your User Name" ); add(l1);
```



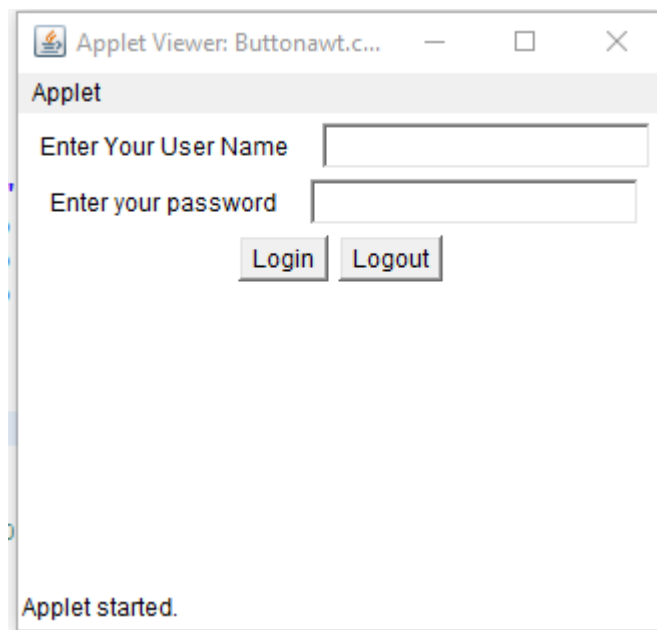
```
TextField tf1=new TextField(20); add(tf1);
Label l2=new Label(" Enter your password"); add(l2);
TextField tf2=new TextField(20); add(tf2);
Button b1=new Button("Login"); add(b1);
Button b2=new Button("Logout");add(b2);

}

}

/*
<applet code="Buttonawt" width=400 height=400> </applet>
*/
```

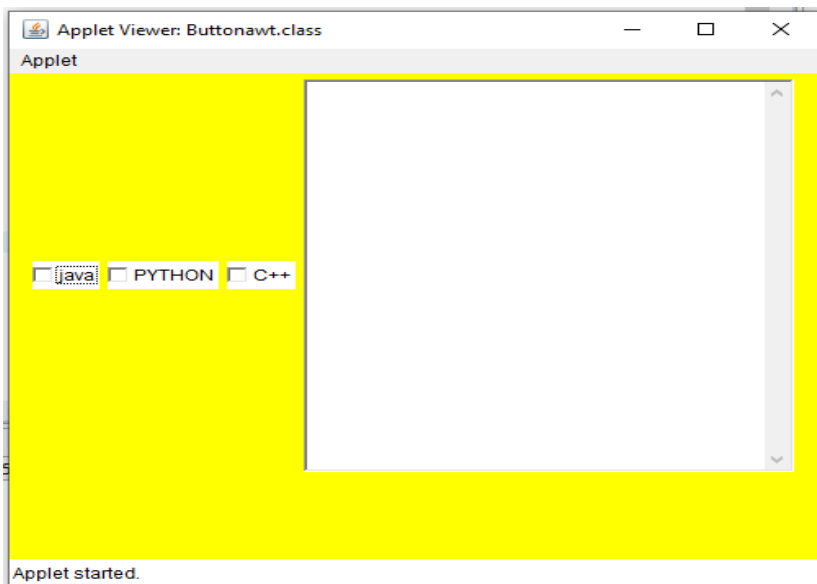
OUTPUT



//Applet containing three checkboxes with RED, BLUE, GREEN and a Text Area

```
import java.awt.*;
import java.applet.Applet;
public class Buttonawt extends Applet
{
    public void init( )
    {
        Checkbox c1=new Checkbox("java"); add(c1);
        Checkbox c2=new Checkbox("PYTHON"); add(c2);
        Checkbox c3=new Checkbox("C++"); add(c3);
        TextArea ta=new TextArea(20,40); add(ta);
    }
}
/*
<applet code="Buttonawt" width=400 height=400> </applet>
*/
```

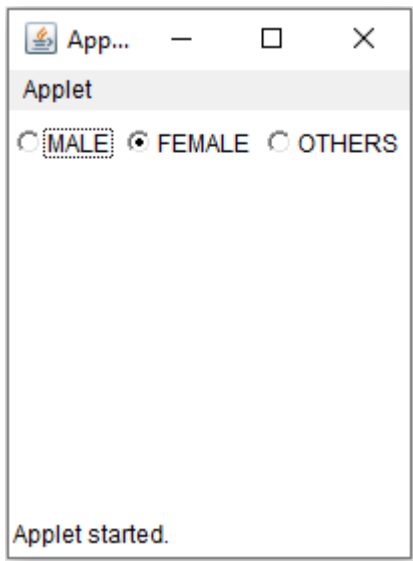
OUTPUT



// Program to develop Applet that contains Radio Buttons

```
import java.awt.*;
import java.applet.Applet;
public class Buttonawt extends Applet
{
    public void init( )
    {
        CheckboxGroup cbg=new CheckboxGroup();
        Checkbox c1=new Checkbox("MALE",cbg,false); add(c1);
        Checkbox c2=new Checkbox("FEMALE",cbg,true); add(c2);
        Checkbox c3=new Checkbox("OTHERS",cbg,false); add(c3);
    }
}
```

OUTPUT

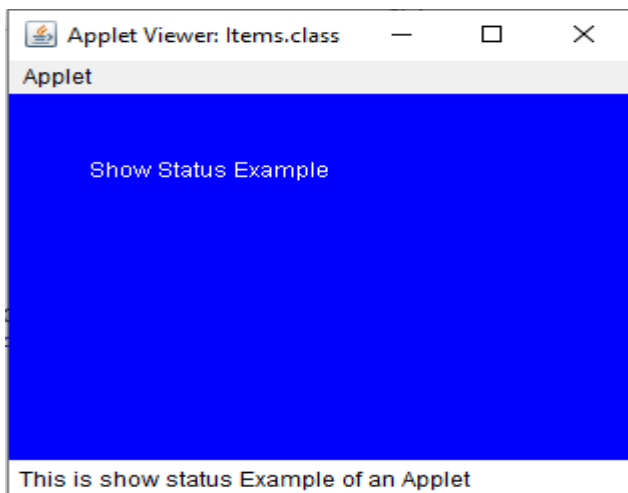


// Set Status Message in Applet Window Example

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

public class Items extends Applet
{
    public void paint(Graphics g)
    {
        setBackground(Color.blue);
        setForeground(Color.white);
        g.drawString("Show Status Example", 40, 50);
        showStatus(" This is show status Example of an Applet");
    }
}
```

OUTPUT

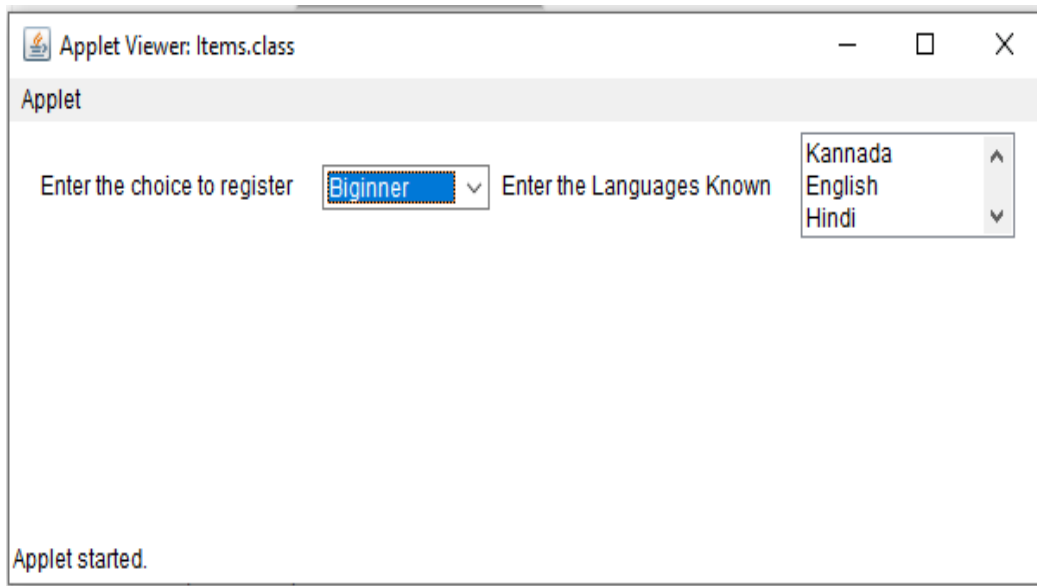


// Applet containing List, Choice

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

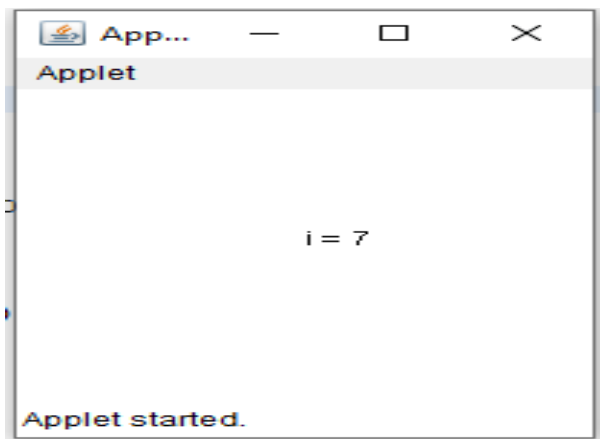
public class Items extends Applet
{
    public void init()
    {
        Label l1=new Label("Enter the choice to register");
        add (l1);
        Choice c=new Choice();
        c.add("Biginner");
        c.add("Intermediate");
        c.add("Advanced");
        add(c);
        Label l2=new Label("Enter the Languages Known");
        add (l2);
        List l=new List(2,true);
        l.add("Kannada");
        l.add("English");
        l.add("Hindi");
        l.add("Telugu");
        l.add("Tamil");
        add(l);
    }
}
```

OUTPUT



```
// Applet to demonstrate repaint method
import java.applet.Applet;
import java.awt.Graphics;
public class AppRepaint extends Applet {
    int i;
    public void paint(Graphics g)
    {
        g.drawString("i = "+i, 100, 100);
        try{
            Thread.sleep(1000);
        } catch (InterruptedException ex){}
        i++;
        repaint();
    }
}
/*
<applet code="AppRepaint" width=500 height=350>
</applet>
*/
```

OUTPUT

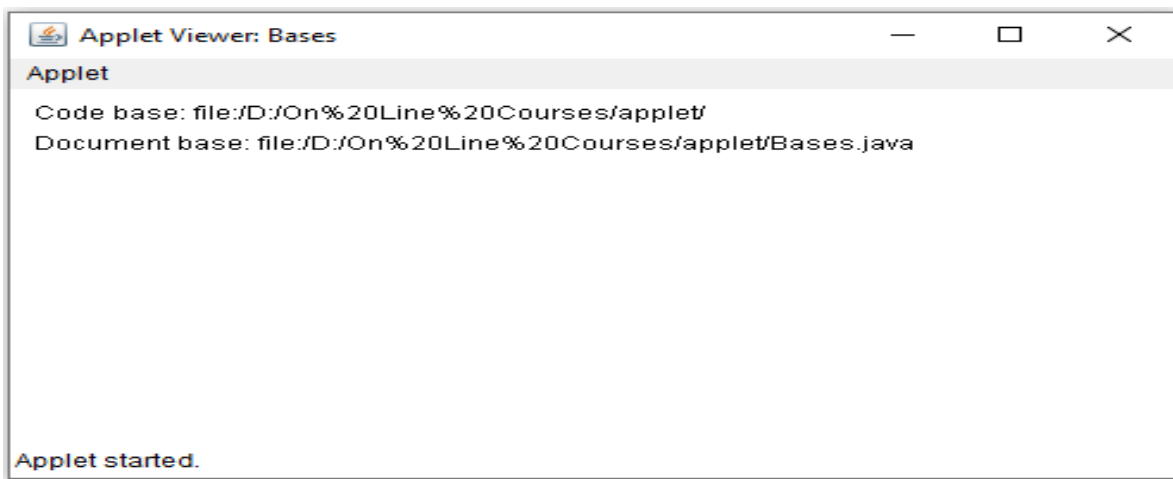


// Applet to demonstrate getDocumentbase() and getCodebase();

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

public class Bases extends Applet
{
    public void paint(Graphics g)
    {
        String msg;
        URL url = getCodeBase(); // get code base
        msg = "Code base: " + url.toString();
        g.drawString(msg, 10, 20);
        url = getDocumentBase(); // get document base
        msg = "Document base: " + url.toString();
        g.drawString(msg, 10, 40);
    }
}
```

OUTPUT



The HTML APPLET tag; Passing parameters to Applets;

<APPLET specifies the beginning of the HTML applet code

CODE="demoxx.class" is the actual name of the applet (usually a 'class' file)

CODEBASE="demos/" is the location of the applet (relative as here, or a full URL)

NAME="smily" the name you want to give to this instance of the applet on this page

WIDTH="100" the physical width of the applet on your page

HEIGHT="50" the physical height of the applet on your page

ALIGN="Top" where to align the applet within its page space (top, bottom, center)

<PARAM specifies a parameter that can be passed to the applet

NAME="name1" the name known internally by the applet in order to receive this parameter

VALUE="000000" the value you want to pass for this parameter

> end of this parameter

<PARAM specifies a parameter that can be passed to the applet (applet specific)

NAME="name2" the name known internally by the applet in order to receive this parameter

VALUE="fffff" the value you want to pass for this parameter

> end of this parameter

</APPLET> specifies the end of the HTML applet code

// Passing parameters to Applets

```
import java.applet.Applet;
import java.awt.Graphics;

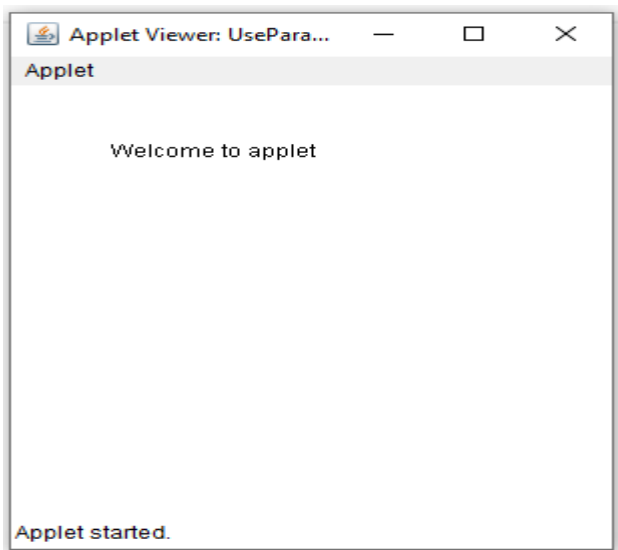
public class UseParam extends Applet{

    public void paint(Graphics g){
        String str=getParameter("msg");
        g.drawString(str,50, 50);
    }

}

/*<applet code="UseParam.class" width="300" height="300">
<param name="msg" value="Welcome to applet">
    */
```

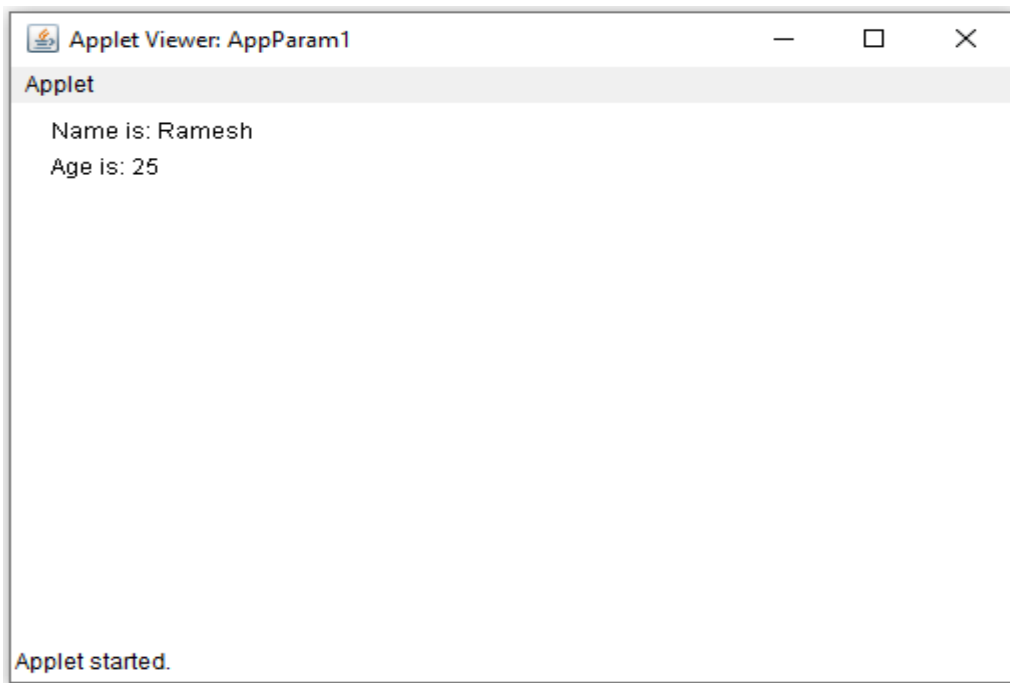
OUTPUT



```
import java.awt.*;
import java.applet.*;
public class AppParam1 extends Applet
{
    String n;
    String a;
    public void init()
    {
        n = getParameter("name");
        a = getParameter("age");
    }
    public void paint(Graphics g)
    {
        g.drawString("Name is: " + n, 20, 20);
        g.drawString("Age is: " + a, 20, 40);
    }
}
```

```
/*
    <applet code="AppParam1" height="300" width="500">
        <param name="name" value="Ramesh" />
        <param name="age" value="25" />
    </applet>
*/
```

OUTPUT



//Example to display Image in Applet

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

public class App_Image extends Applet {
    Image img;
    public void init() {
        img=getImage(getDocumentBase(),"java.jpg");
    }
    public void paint(Graphics g)
    {
        g.drawImage(img,100,100,this);
    }
}

/*
<applet code="App_Image" width=400 height=400> </applet>
*/
```


OUTPUT



//Example to play Audio in Applet

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

public class App Audio extends Applet {
    AudioClip clip;
    public void init(){
        clip=getAudioClip(getDocumentBase(), "Audio2.wav");
    }
    public void paint(Graphics g)
    {
        clip.play();
        //clip.stop();
        clip.loop();
    }
}

/*
<applet code="App_Audio" width=400 height=400> </applet>
*/
```

//Design an Applet display the Sum of Two Numbers and set status message in Applet window

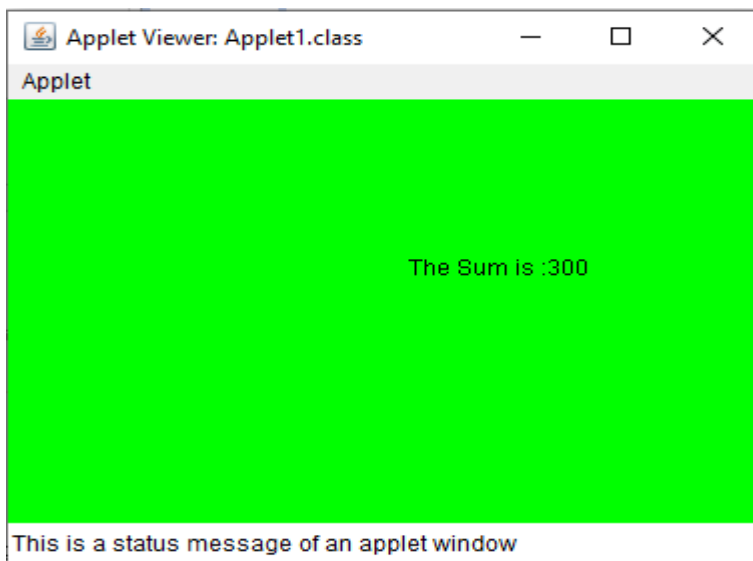
```
import java.awt.*;
import java.applet.*;
/* <applet code = Applet1.class width= 200 height=200> </applet> */

public class Applet1 extends Applet
{

    public void paint(Graphics g)
    {
        setBackground(Color.green);
        int a=100;
        int b=200;
        int sum = a+b;
        String s = "The Sum is :" + String.valueOf(sum);
        g.drawString( s, 200,100);
        showStatus("This is a status message of an applet window");

    } }
```

OUTPUT



//Draw Smiley in an Applet example.

```
import java.awt.*;
import java.applet.*;
/* <applet code = Smiley.class width= 200 height=200> </applet> */

import java.awt.*;
import java.applet.*;

public class Smiley extends Applet{

    public void paint(Graphics g){

        Font f = new Font("Helvetica", Font.BOLD,20);
        g.setFont(f);
        g.drawString("Keep Smiling!!!", 50, 30);
        g.drawOval(60, 60, 200, 200);
        g.fillOval(90, 120, 50, 20);
        g.fillOval(190, 120, 50, 20);
        g.drawLine(165, 125, 165, 175);
        g.drawArc(110, 130, 95, 95, 0, -180);

    }
}
```

OUTPUT



Programs on Frame

// Program to include Scrollbars in a Frame

Constructors of Scrollbar

Constructor	Description
<code>public Scrollbar(int orientation, int value, int extent, int min, int max)</code>	Creates a Scrollbar with the specified <i>orientation</i> , <i>value</i> , <i>extent</i> , <i>minimum</i> , and <i>maximum</i> , where -

orientation - This parameter specifies the Scrollbar to be a horizontal or a vertical Scrollbar.

value - This parameter specifies the starting position of the knob of Scrollbar on the track of a Scrollbar.

extent - This parameter specified the width of the knob of Scrollbar.

min - This parameter specifies the minimum width of the track on which Scrollbar moves.

max - This parameter specifies the maximum width of the track.

```
import java.awt.*;
import javax.swing.*;

public class ScrollEg1
{
    Frame frame;
    Label label1;

    ScrollEg1 ()
    {
        frame = new Frame("Scrollbar");

        label1 = new Label("Displaying a horizontal and vertical Scrollbar in the Frame", Label.CENTER);

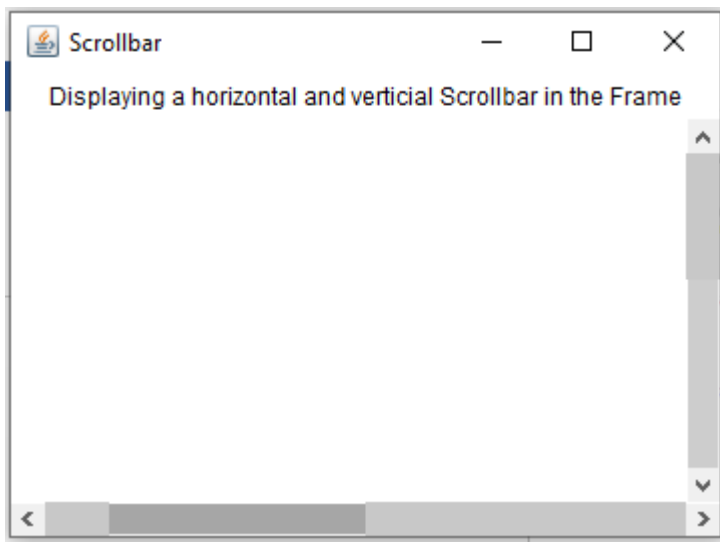
        Scrollbar scrollB1 = new Scrollbar(Scrollbar.HORIZONTAL, 10, 40, 0, 100);
        Scrollbar scrollB2 = new Scrollbar(Scrollbar.VERTICAL, 10, 60, 0, 100);

        frame.add(label1, BorderLayout.NORTH);
        frame.add(scrollB2, BorderLayout.EAST);
        frame.add(scrollB1, BorderLayout.SOUTH);

        frame.setSize(370, 270);
        frame.setVisible(true);
    }

    public static void main(String[] ar) {
        new ScrollEg1 ();
    }
}
```

OUTPUT

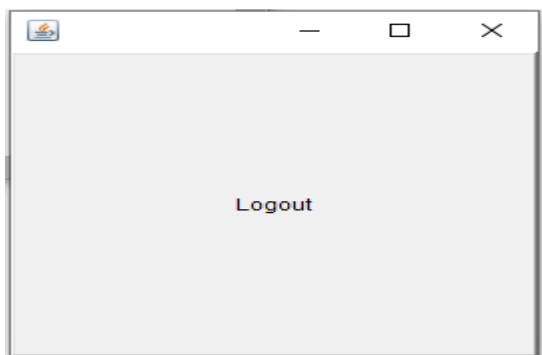


//Since Frame follows Border Layout by default, the second button is placed on the first button

```
import java.awt.*;
public class Frame1Button extends Frame {
    Frame1Button()
    {
        setVisible(true);
        setBounds(10,10,400,400);
        Button b1=new Button("Login");
        add(b1);
        Button b2=new Button("Logout");
        add(b2);
    }
    public static void main(String[] args) {
        Frame1Button ob=new Frame1Button();setDefaultCloseOperation(ob.EXIT_ON_CLOSE);

    }
}
```

OUTPUT



Layout Manager

Layout means the arrangement of components within the container. The layout manager automatically positions all the components within the container. If we do not use layout manager then also the components are positioned by the default layout manager. `LayoutManager` is an interface that is implemented by all the classes of layout managers.

BorderLayout : `BorderLayout()`: creates a border layout but with no gaps between the components.

```
import java.awt.*;

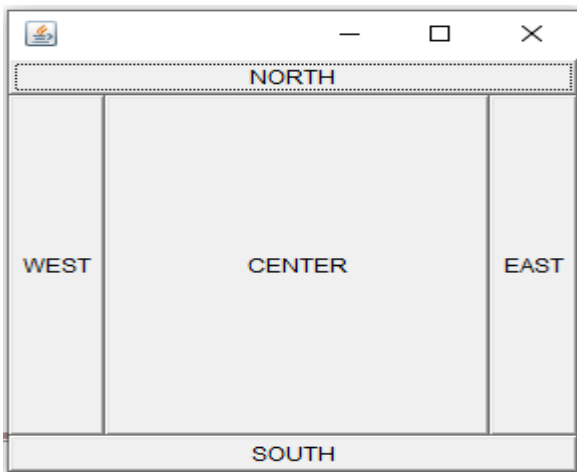
public class Border {
    Frame f;
    Border() {
        f=new Frame();

        Button b1=new Button("NORTH");
        Button b2=new Button("SOUTH");
        Button b3=new Button("EAST");
        Button b4=new Button("WEST");
        Button b5=new Button("CENTER");

        f.add(b1, BorderLayout.NORTH);
        f.add(b2, BorderLayout.SOUTH);
        f.add(b3, BorderLayout.EAST);
        f.add(b4, BorderLayout.WEST);
        f.add(b5, BorderLayout.CENTER);

        f.setSize(300,300);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new Border();
    }
}
```

OUTPUT



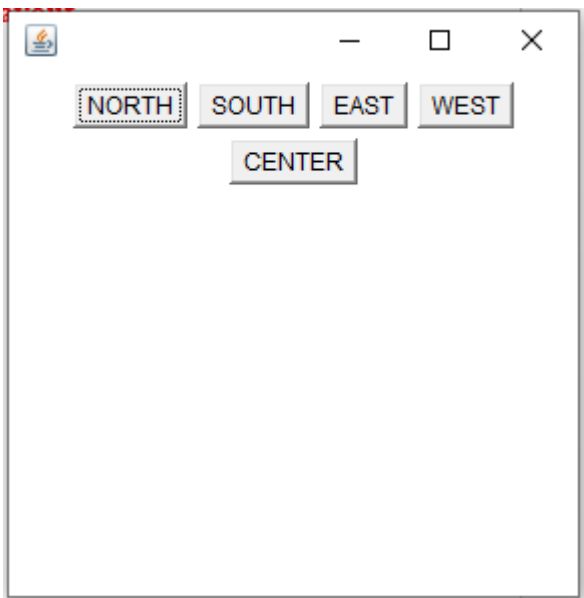
// program to change the Layout of Frame from Border Layout to Flow Layout

```
import java.awt.*;

public class Border {
    Frame f;
    Border() {
        f=new Frame();
        f.setLayout(new FlowLayout());
        Button b1=new Button("NORTH");
        Button b2=new Button("SOUTH");
        Button b3=new Button("EAST");
        Button b4=new Button("WEST");
        Button b5=new Button("CENTER");

        f.add(b1);      f.add(b2);      f.add(b3);
        f.add(b4);      f.add(b5);
        f.setSize(300,300);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new Border();
    }
}
```

OUTPUT

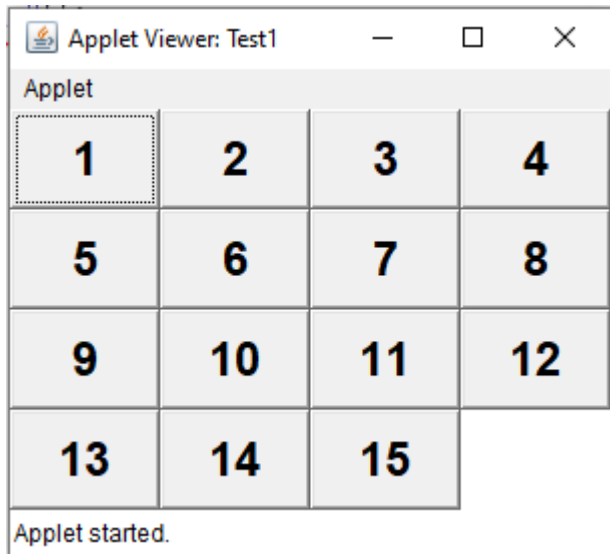


// Program to demonstrate GridLayout

```
//Demonstrate GridLayout
import java.awt.*;
import java.applet.*;
/*
<applet code="Test1" width=300 height=200>
</applet>
*/
public class Test1 extends Applet {
    static final int n = 4;
    public void init() {
        setLayout(new GridLayout(n, n)); // setLayout(new GridLayout(n, n,10,10));

        setFont(new Font("SansSerif", Font.BOLD, 24));
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < n; j++) {
                int k = i * n + j;
                if(k > 0)
                    add(new Button(" " + k));
            }
        }
    }
}
```

OUTPUT



Java CardLayout

The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

Constructors of CardLayout class

1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.
2. **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

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

import javax.swing.*;

public class Test1 extends JFrame implements ActionListener{
    CardLayout card;
    JButton b1,b2,b3;
    Container c;
    Test1(){

        c=getContentPane();
        card=new CardLayout(40,30);
        //create CardLayout object with 40 hor space and 30 ver space
        c.setLayout(card);

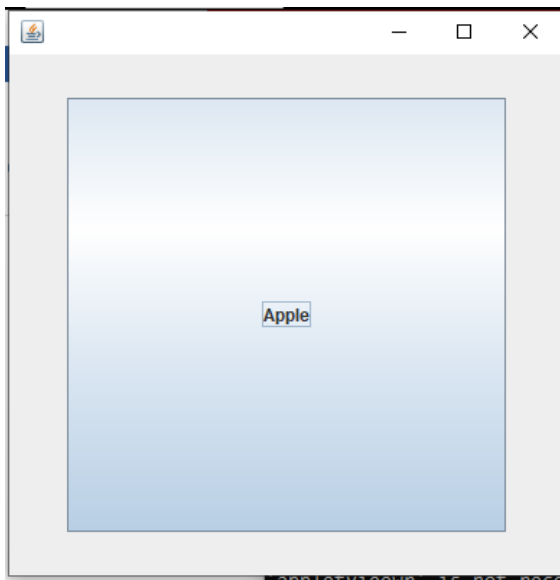
        b1=new JButton("Apple");
        b2=new JButton("Boy");
        b3=new JButton("Cat");
        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);

        c.add("a",b1);c.add("b",b2);c.add("c",b3);

    }
    public void actionPerformed(ActionEvent e) {
        card.next(c);
    }

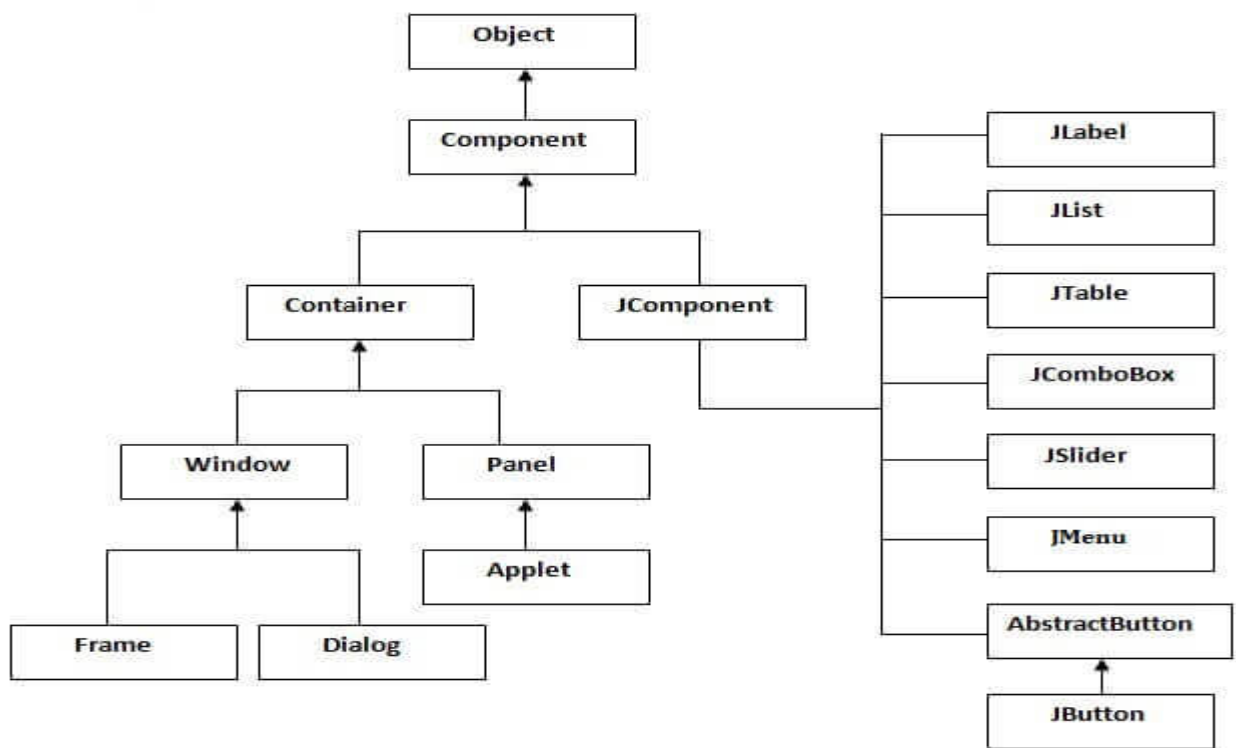
    public static void main(String[] args) {
        Test1 cl=new Test1();
        cl.setSize(400,400);
        cl.setVisible(true);
        cl.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```

OUTPUT



MORE on JAVA Swings

Hierarchy of Java Swing classes



Java Swing is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java. AWT components are referred to as **heavyweight**. AWT translates its visual components into platform-specific equivalents (peers). Swing was introduced in 1997 to fix the problems with AWT.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

MVC - Model-View-Controller.

- One component architecture is MVC - Model-View-Controller.
- The **model** corresponds to the state information associated with the component.
- The **view** determines how the component is displayed on the screen.
- The **controller** determines how the component responds to the user.
- Swing uses a modified version of MVC called "Model-Delegate". In this model the view (look) and controller (feel) are combined into a "delegate".
- Because of the Model-Delegate architecture, the look and feel can be changed without affecting how the component is used in a program.

Components and Containers

- A component is an independent visual control: a button, a slider, a label, ...
- A container holds a group of components.
- In order to display a component, it must be placed in a container.
- A container is also a component and can be contained in other containers.
- Swing components are derived from the **JComponent** class. The only exceptions are the four top-level containers: JFrame, JApplet, JWindow, and JDialog.
- All the Swing components are represented by classes in the javax.swing package.
- All the component classes start with **J**: JLabel, JButton, JScrollbar, ...

Top-level Container Panes

- Each top-level component defines a collection of "panes". The top-level pane is **JRootPane**.
- JRootPane manages the other panes and can add a menu bar.
- There are three panes in JRootPane: 1) the glass pane, 2) the content pane, 3) the layered pane.
- The content pane is the container used for visual components. The content pane is an opaque instance of JPanel.

Two Key Swing Features

- **Lightweight components** : Swing components are lightweight as they are written entirely in Java and do not depend on native peers (platform specific code resources). Rather, they use simple drawing primitives to render themselves on the screen. The look and the feel of the component is not controlled by the underlying operating system but by Swing itself.
- **Pluggable look and feel** : Swing has the capability to support several look and feels. As the look and feel of components is controlled by Swing rather than by operating system, the feel of components can also be changed. The look and feel of a component can be separated from the logic of the component. Thus, it is possible to "plug in" a new look and feel for any given component without affecting the rest of the code.

Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

No.	Java AWT	Java Swing
1)	AWT components are platform-dependent .	Java swing components are platform-independent .
2)	AWT components are heavyweight .	Swing components are lightweight .
3)	AWT doesn't support pluggable look and feel .	Swing supports pluggable look and feel .
4)	AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT doesn't follows MVC (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing follows MVC .

//Program to add different types of Buttons in JApplet a) JButton b)RadioButton c) JCheckbox
d) JToggleButton

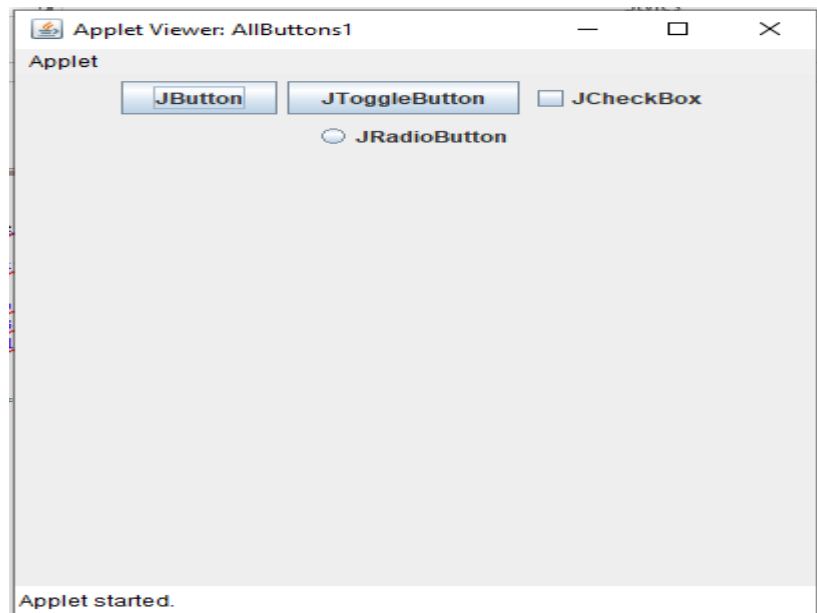
```
import java.awt.*;
import javax.swing.*;

public class AllButtons1 extends JApplet {

    public void init() {
```

```
Container cp = getContentPane();
cp.setLayout(new FlowLayout());
JButton jb = new JButton("JButton");
cp.add(jb);
cp.add(new JToggleButton("JToggleButton"));
cp.add(new JCheckBox("JCheckBox"));
cp.add(new JRadioButton("JRadioButton"));
}
//<applet code=AllButtons1 width=400 height=400></applet>
```

OUTPUT



//Program demonstrate to add JLabel and JTextField to JFrame

```
import java.awt.*;
import javax.swing.*;
public class TFLabel extends JFrame {
    TFLabel() {
        setTitle("JLabel and JTextField");
        JLabel l1=new JLabel("Enter Your Name");
        add(l1);
        JTextField tf=new JTextField(20);
        add(tf);
        setLayout(new FlowLayout());
        setSize(350,275);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public static void main(String args[]) {
        new TFLabel();
    }
}
```

OUTPUT



// Program to demonstrate JComboBox

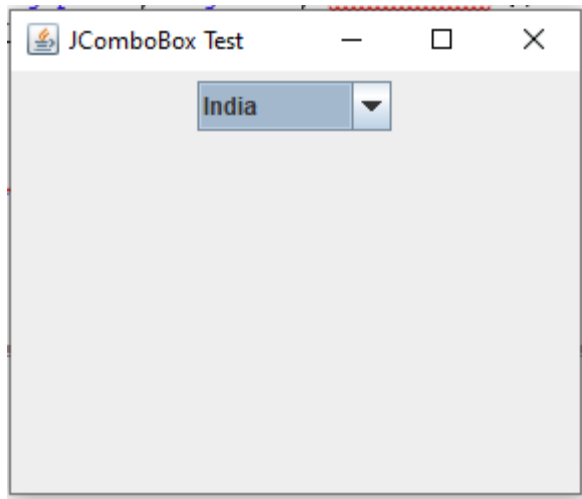
A **JComboBox** is a **component** that **displays** a drop-down list and **gives** users options that we can **select** one and only one item at a time .

JComboBox

- A **JComboBox** can be **editable** or **read-only**.
- An **ActionListener**, **ChangeListener** or **ItemListener** interfaces can be used to handle the user actions on a **JComboBox**.
- A **getSelectedItem()** method can be used to get the selected or entered item from a combo box.

```
import java.awt.*;
import javax.swing.*;
public class JComboBoxTest extends JFrame {
    JComboBoxTest() {
        setTitle("JComboBox Test");
        String country[] = {"India", "Aus", "Singapore", "England", "Newzealand"};
        JComboBox jcb = new JComboBox(country);
        setLayout(new FlowLayout());
        add(jcb);
        setSize(300, 250);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public static void main(String[] args) {
        new JComboBoxTest();
    }
}
```

OUTPUT

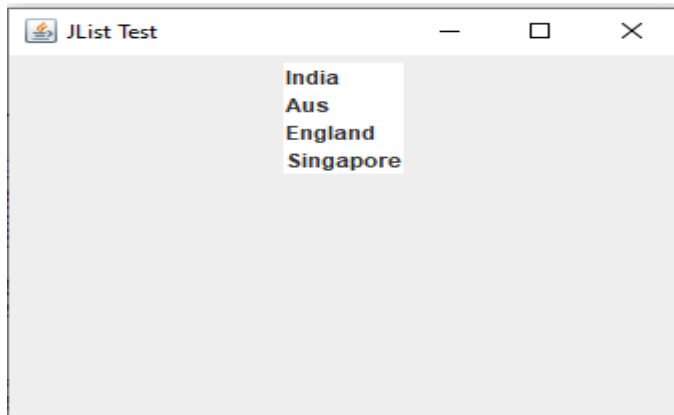


// Program to demonstrate JList

- A **JList** is a component that allows the user to choose either a **single selection** or **multiple selections**.
- A **JList** class itself does not support scrollbar. In order to add scrollbar, we have to use **JScrollPane** class together with the **JList** class. The **JScrollPane** then manages a scrollbar automatically.
- A **getSelectedIndex()** method returns the index of the first selected item or -1 if no items are selected
- A **getSelectedValue()** returns the first selected item or null if no items are selected.

```
import java.awt.*;
import javax.swing.*;
public class JListTest extends JFrame {
    JListTest() {
        setTitle("JList Test");
        DefaultListModel dlm = new DefaultListModel();
        dlm.addElement("India");
        dlm.addElement("Aus");
        dlm.addElement("England");
        dlm.addElement("Singapore");
        JList list = new JList();
        list.setModel(dlm);
        setLayout(new FlowLayout());
        add(list);
        setSize(350,275);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public static void main(String args[]) {
        new JListTest();
    }
}
```

OUTPUT



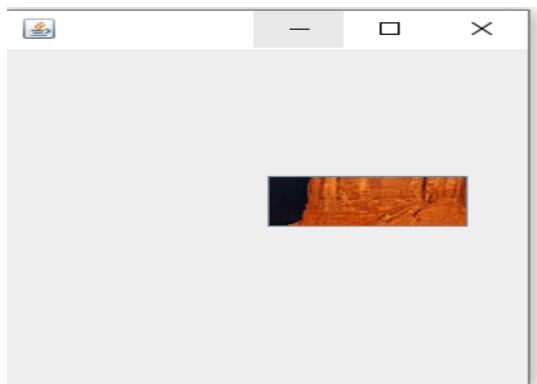
//Program add an Image on JButton

```
// example for displaying image on Button
import java.awt.event.*;
import javax.swing.*;
public class ImageIcon1{
    ImageIcon1(){
        JFrame f=new JFrame();

        JButton b=new JButton(new ImageIcon("a.jpg"));
        b.setBounds(130,100,100, 40);
        f.add(b);
        f.setSize(300,400);
        f.setLayout(null);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {
        new ImageIcon1();
    }
}
```

OUTPUT



Java Swing JTable

The JTable class is a part of Java Swing Package and is generally used to display or edit two-dimensional data that is having both rows and columns. It is similar to a spreadsheet. This arranges data in a tabular form.

Constructors in JTable:

JTable(): A table is created with empty cells.

JTable(int rows, int cols): Creates a table of size rows * cols.

```
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTable;

public class JTableExamples {
    // frame
    JFrame f;
    // Table
    JTable j;

    // Constructor
    JTableExamples()
    {
        // Frame initialization
        f = new JFrame();

        // Frame Title
        f.setTitle("JTable Example");

        // Data to be displayed in the JTable
        String[][] data = {
            { "Kumar", "4031", "CSE" },
            { "Anand", "6014", "ISE" }
        };

        // Column Names
        String[] columnNames = { "Name", "Roll Number", "Department" };

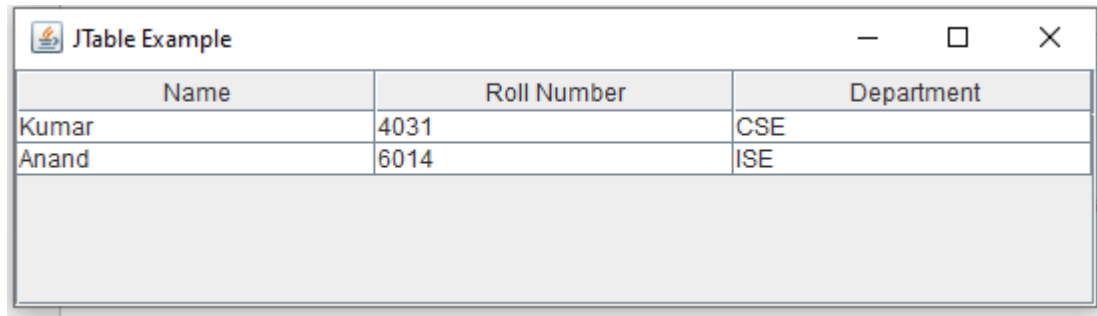
        // Initializing the JTable
        j = new JTable(data, columnNames);
        j.setBounds(30, 40, 200, 300);

        // adding it to JScrollPane
        JScrollPane sp = new JScrollPane(j);
        f.add(sp);
        // Frame Size
        f.setSize(500, 200);
        // Frame Visible = true
        f.setVisible(true);
    }

    // Driver method
    public static void main(String[] args)
    {
        new JTableExamples();
    }
}
```

```
}  
}
```

OUTPUT



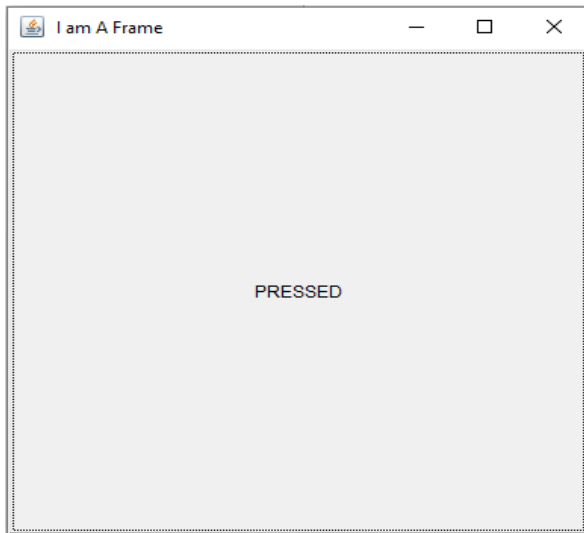
Name	Roll Number	Department
Kumar	4031	CSE
Anand	6014	ISE

EVENT HANDLING

Create a Frame with a button having caption PRESS ME, once the button is clicked, the caption changes to PRESSED.

```
import java.awt.*;  
import java.awt.event.*;  
  
public class Action1Frame extends Frame implements ActionListener  
{  
    Button b1;  
  
    Action1Frame()  
    {  
        super(" I am A Frame");  
        b1=new Button(" press me");  
        add(b1);  
        b1.addActionListener(this);  
        setBounds(10,10,400,400);  
        setVisible(true);  
    }  
  
    public void actionPerformed(ActionEvent ae)  
    {  
        if (ae.getSource()==b1)  
            b1.setLabel("PRESSED");  
    }  
  
    public static void main(String a[])  
    {  
        new Action1Frame();  
    }  
}  
/*  
<applet code=Action1.class width=400 height=400>  
</applet>  
*/
```

OUTPUT



Create a swing applet that has two buttons named alpha and beta. When either of the buttons pressed, it should display “Alpha is pressed” or : Beta is pressed”

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

public class AlphaBeta extends JApplet implements ActionListener{
    JButton rb1,rb2;
    JLabel l;
    public void init()
    {
        rb1=new JButton("Alpha");

        rb2=new JButton("Beta");

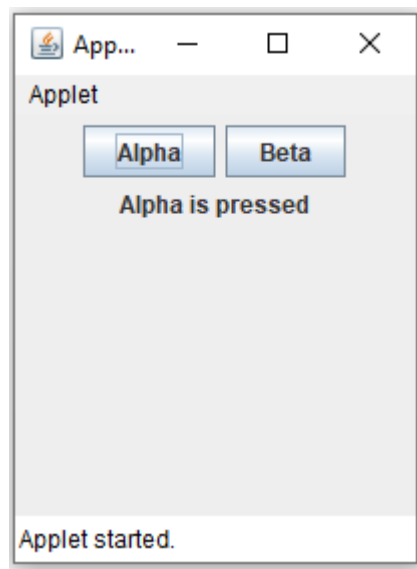
        l=new JLabel();
        setLayout(new FlowLayout());

        getContentPane().add(rb1);getContentPane().add(rb2);   getContentPane().add(l);
        rb1.addActionListener(this);
        rb2.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e){
        if(e.getSource()==rb1){
            l.setText("Alpha is pressed");
        }
        if(e.getSource()==rb2){
            l.setText("Beta is pressed");
        }
    }
}

//<applet code=AlphaBeta width=500 height=500> </applet>
```

OUTPUT



Create an Applet with three text fields and a button. Input integer values in first two Text Fields. On Click of add button, the sum should be displayed in the third Text Field.

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

/*Coding of HTML File <applet code = add.class width= 200 height=200> </applet> */

public class add extends Applet implements ActionListener
{
    Button b1;
    TextField tf1;
    TextField tf2;
    TextField tf3;

    public void init( )
    {
        tf1=new TextField(20);
        tf2=new TextField(20);
        tf3=new TextField(20);
        b1=new Button(" add ");
        add(b1); add(tf1);add(tf2);add(tf3);
        b1.addActionListener(this);
    }

    public void actionPerformed(ActionEvent ae)
    {
        String s1=tf1.getText();
        int a=Integer.parseInt(s1);

        String s2=tf2.getText();
        int b=Integer.parseInt(s2);
        int c=a+b;
        tf3.setText(Integer.toString(c));
    }
}
```

```
    }  
}  
/*  
<applet code=add.class width=400 height=400>  
</applet>  
  
*/
```

OUTPUT

