UNIT-5

Applets

There are two types of Java Programs.

1) Standalone application programs
2) Applet programs

Standalone application program is a program that is run on our system using java runtime environment. Applet is a small java program that can be executed by a java compatible web browser or appletviewer. Applet is a graphical user interface used to execute forms.

Applet is a special type of program that is embedded into a webpage to generate the dynamic content. It runs inside the web browser and works at client side. Applet is embedded in a HTML page using the APPLET or OBJECT tag and hosted on a web server.

All Applets are sub-classes (either directly or indirectly) of *java.applet.Applet* class. Applets are not stand-alone programs. Instead, they run within either a web browser or an appletviewer. JDK provides a standard applet viewer tool called appletviewer.

In general, execution of an applet does not begin at main() method. Output on applet window is not performed by *System.out.println()*. Rather it is handled with various AWT(Abstract Window Toolkit) methods, such as *drawString()*.

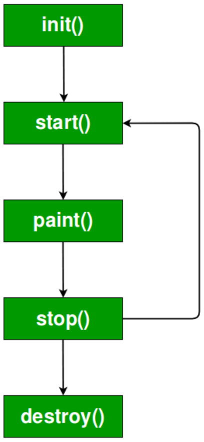
Differences between Application programs and Applet programs:

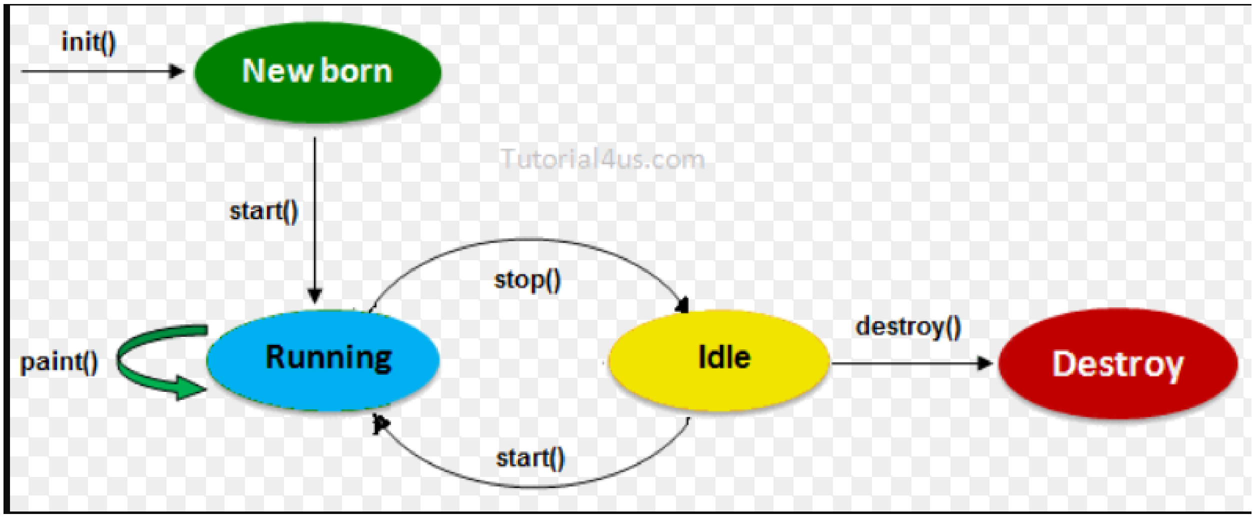
Application Programs	**Applet Programs**
Standalone application program is a program that is run on our system using java runtime environment.	Applet is a small java program that can be executed by a java compatible web browser or appletviewer.
It requires a main method() for its execution.	It does not require a main method() for its execution.
Created by writing the program inside the main() method.	Created by extending the Applet class
Executed using Java Runtime Environment	Executed either by java compatible web browser or appletviewer tool.

Advantages of Applet:

- Applets are used to make the web site more dynamic.
- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many platforms, including Linux, Windows, MacOs etc.

Java Applet Life Cycle:





Note: init() and destroy() methods are executed only once for the entire life cycle.

Lifecycle methods for Applet:

The java.applet.Applet class provides 5 life cycle methods.

java.applet.Applet class

For creating any applet java.applet.Applet class must be inherited. It provides 5 life cycle methods of applet.

1. **public void init():** The **init( )** method is the first method to be called and is used to initialize the Applet. It is invoked only once.
2. **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet. This is running state of Applet.

Note:- **init( )** is called once i.e. when the first time an applet is loaded whereas **start( )** is called each time an applet's HTML document is displayed onscreen. So, if a user leaves a web page and comes back, the applet resumes execution at **start( )**.

3. **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.
   The **paint()** method is called each time an AWT-based applet's output must be redrawn. **paint()** is also called when the applet begins execution. Whatever the cause, whenever the applet must redraw its output, **paint()** is called. The **paint()** method has one parameter of type Graphics. This parameter will contain the graphics context, which describes the graphics environment in which the applet is running.

   Note: Graphics class is present in java.awt package. So import java.awt.*;

4. **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.The **stop( )** method is called when a web browser leaves the HTML document containing the applet.
5. **public void destroy():** is used to destroy the Applet. It is invoked only once.The **destroy( )** method is called when the environment determines that your applet needs to be removed completely from memory.

Applet is a **predefined class** and it has the following life cycle methods

```
Class Applet
{
  public void init()
  {
            showStatus("applet initialized");
   }
  public void start()
  {
            showStatus("applet started");
   }
  public void stop()
  {

            showStatus("applet stopped");
   }
  public void destroy()
  {
            showStatus("applet destroyed");
   }
  public void paint(Graphics g)
  {
    // null implementation;
   }

}
```

There are **two** standard ways in which you can run an applet :

1. Executing the applet within a Java-compatible web browser.
2. Using an applet viewer, such as the standard tool, appletviewer. An applet viewer executes your applet in a window. This is generally the fastest and easiest way to test your applet.

There are two ways to run an applet thru appletviewer

(i) By appletViewer tool (along with java program).

(ii) By html file (java program + html file)

(i) By appletviewer tool

Program 1

// AppletEx2.java

```
import java.applet.Applet;
import java.awt.Graphics;
/*<applet code="AppletEx2.class" width="500" height="500">
</applet> */
public class AppletEx2 extends Applet
{
        //Overriding paint() method
        public void paint(Graphics g)
        {
                g.drawString("This is my first applet”,100,150);
        }
}
```

Note: Comments can be written anywhere in the program. But best practice is to write before class

Compile Program: javac AppletEx2.java

Execute Program: appletviewer AppletEx2.java

(ii) By html file (java program + html file separately)

Program 2

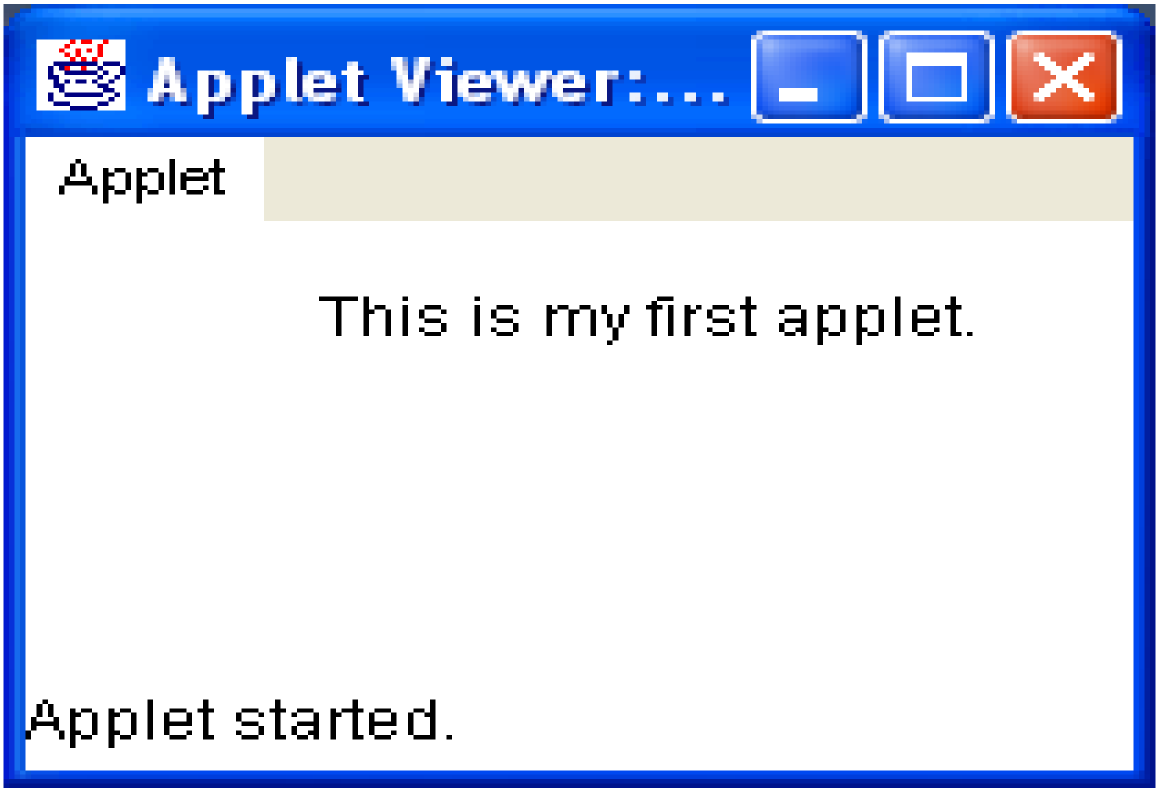
```
//AppletEx1.java
import java.applet.Applet;
import java.awt.Graphics;
public class AppletEx1 extends Applet
{
        //Overriding paint() method
        public void paint(Graphics g)
        {
                g.drawString("This is my first applet ",100,150);
        }
}
```

Compile program: javac AppletEx1.java

//myapplet.html

```
<html>
<body>
<applet code="AppletEx1.class" width="300" height="300">
</applet>
</body>
</html>
```

Execute program: appletviewer myapplet.html



This is my first applet.

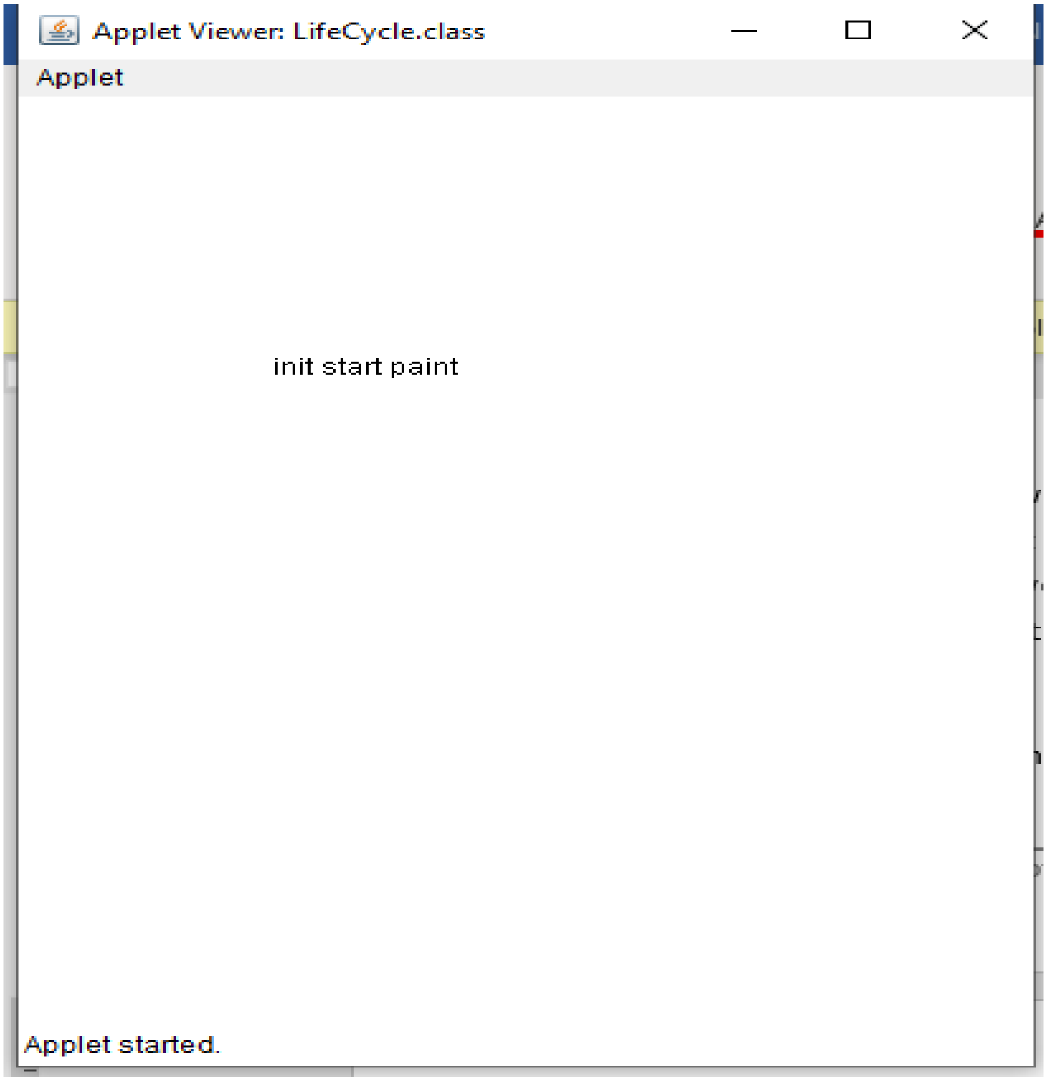
Applet started.

Write applet program to demonstrate applet life cycle methods.

Program 3

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

/*<applet code="LifeCycle.class" height=500 width=400></applet >*/

public class LifeCycle extends Applet
{
        String S="";



        //Overriding all methods of Applet LifeCycle
        public void init()
        {
                 S=S+"init ";
        }
        public void start()
        {
                S=S+"start ";;
        }
        public void paint (Graphics g)
        {
                S=S+"paint ";
                g.drawString(S,100,150);
```

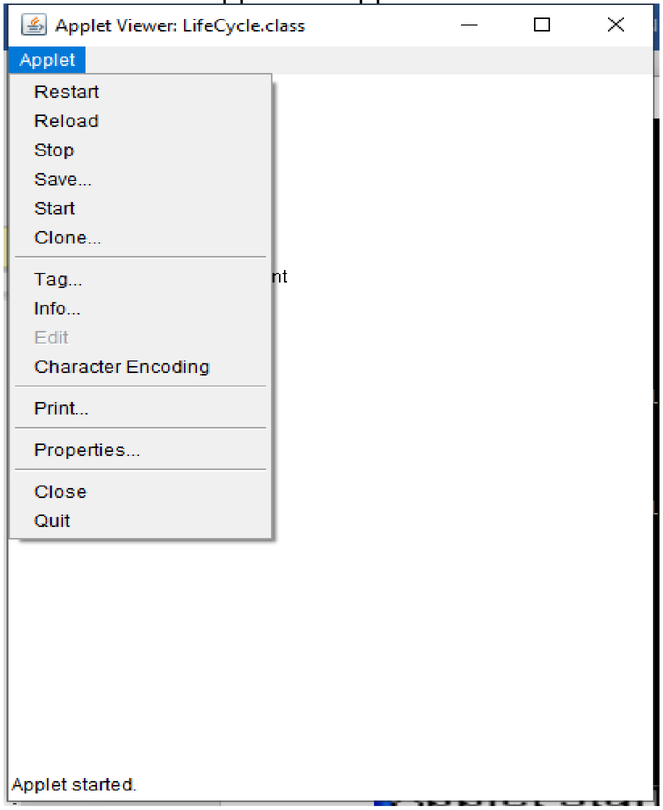
```
    }
    public void stop()
    {
        S=S+"stop ";
    }
    public void destroy()
    {
        S=S+"destroy ";
    }
}
```



```
init start paint
```

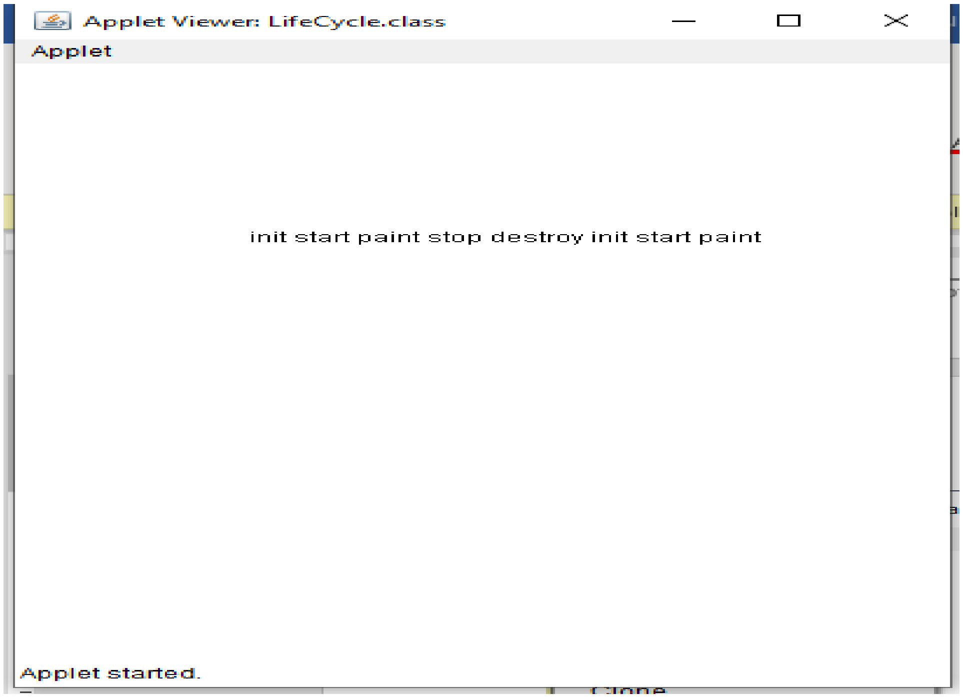
```
Applet started.
```

Note: Click on Applet on Applet window



Applet started.

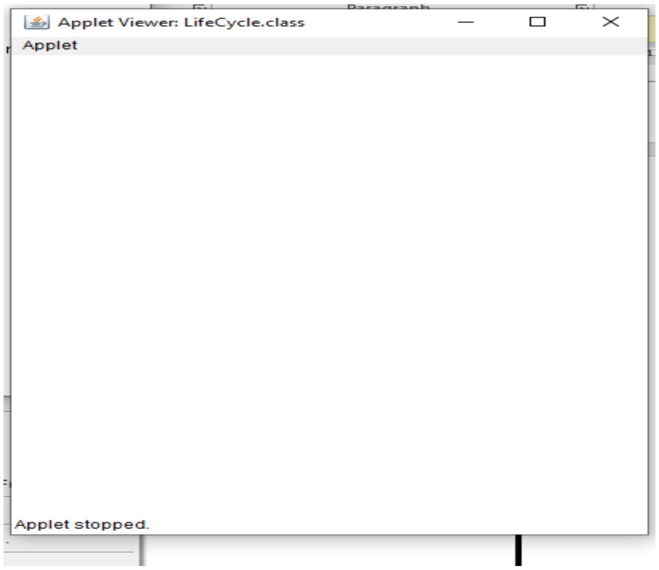
Note: After clicking on Restart we get the following output



```
init start paint stop destroy init start paint
```

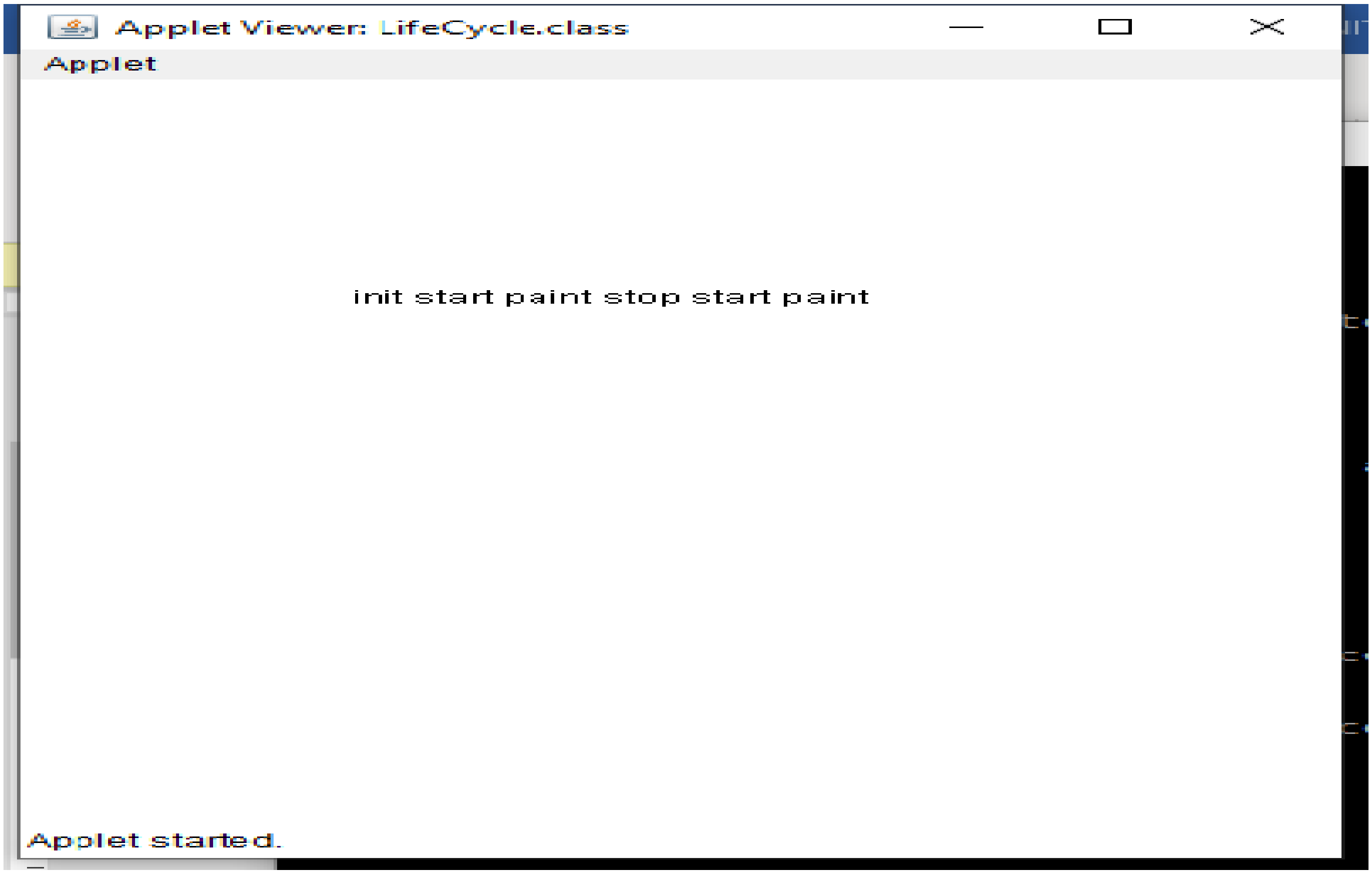
Applet started.

Note: After clicking on Stop we get the following output. The applet enters idle state.



```
Applet stopped.
```

Note: to enter into the running state from the idle state click on the start then the following applet window is created.



```
init start paint stop start paint
```

```
Applet started.
```

Graphics class methods:-

Graphics is a predefined class which is available in awt package. Methods of graphics are

1)void drawString(string str, int x, int y)

2) void drawLine(int start_x, int start_y,int end_x ,int end_y)

3) void drawRect(int start_x, int start_y, int width,int height)

4) void fillRect(int start_x, int start_y, int width,int height)

5) void drawRoundRect(int start_x, int start_y, int width, int height , int x_Diameter, int y_ Diameter)

6) void fillRoundRect(int start_x, int start_y, int width, int height , int x_Diameter, int y_ Diameter)

7) void drawOval(int start_x, int start_y, int width,int height)

8) void fillOval(int start_x, int start_y,int width ,int height)

9) void drawArc(int start_x, int start_y,int width ,int height,int startAngle,int sweepAngle)

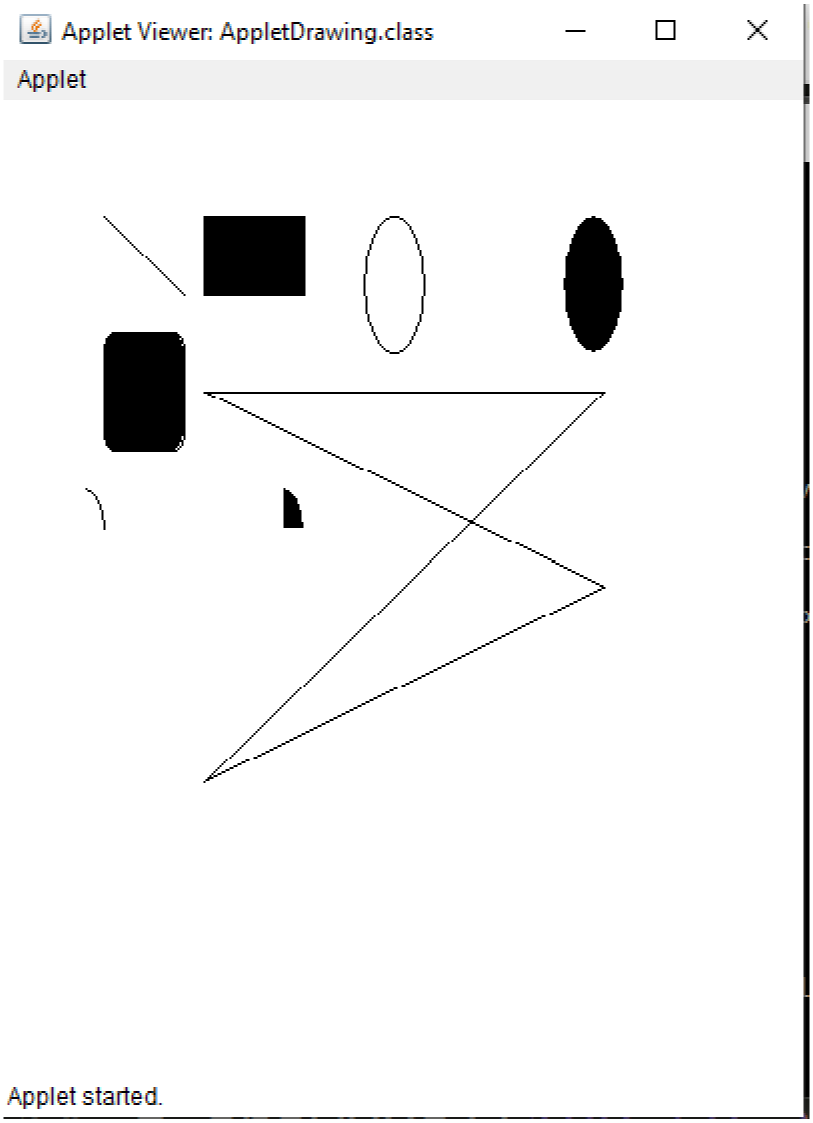
9) void fillArc(int start_x, int start_y, int width,int height,int startAngle,int sweepAngle)

10)void drawPolygon(int x[],int y[], int no_of_points)

Note: In drawPolygon, x and y are integer arrays which hold (x,y) points. The first point and the last point should be same. For example to draw a square we need four points but while using drawPolygon method we require 5 points. As the starting point and ending point are same.

Program 4:

```
import java.awt.*;

import java.applet.*;

/*<applet code="AppletDrawing.class" height=500 width=400></applet >*/

public class AppletDrawing extends Applet

{

public void paint (Graphics g)

 {

        g.drawLine(50,60,90,100);

        g.drawRect(100,60,50,40);

        g.fillRect(100,60,50,40);

        g.drawRoundRect(50,120,40,60,10,20);

        g.fillRoundRect(50,120,40,60,10,20);

        g.drawOval(180,60,30,70);
        g.fillOval(280,60,30,70);
        g.drawArc(30,200,20,40,0,90);
        g.fillArc(130,200,20,40,0,90);
        int x[]={100,300,100,300,100};
        int y[]={150,150,350,250,150};
        g.drawPolygon(x,y,5);
}
}
```



Applet started.

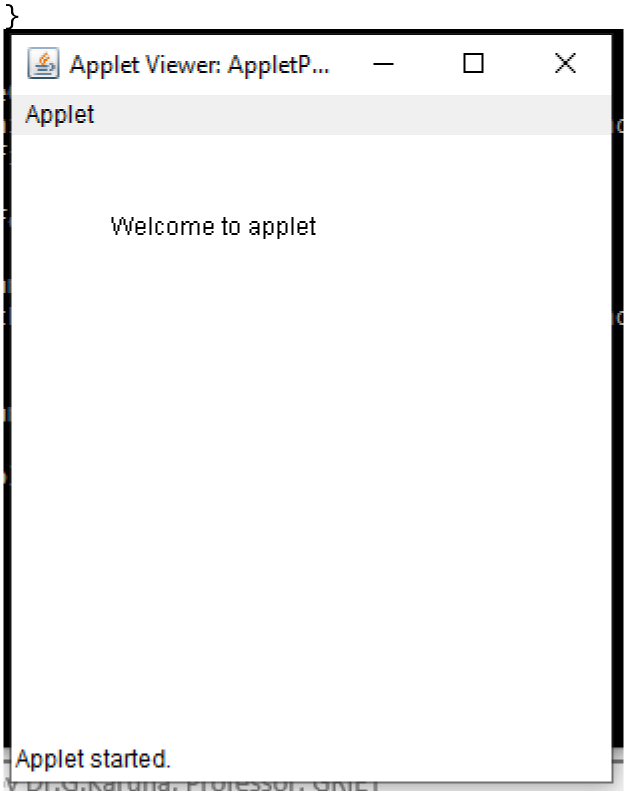
Passing parameters to applets:

We can get any information from the HTML file as a parameter. For this purpose, Applet class provides a method named getParameter().

Syntax: **public** String getParameter(String parameterName)

Program 5:

```
// Passing parameters to Applets
import java.applet.Applet;
import java.awt.Graphics;
/* <applet code="AppletParam.class" width="300" height="300">
<param name="msg" value="Welcome to applet">
</applet>    */
public class AppletParam extends Applet
{
        public void paint(Graphics g)
        {
                String str=getParameter("msg");
                g.drawString(str,50, 50);
        }
}
```

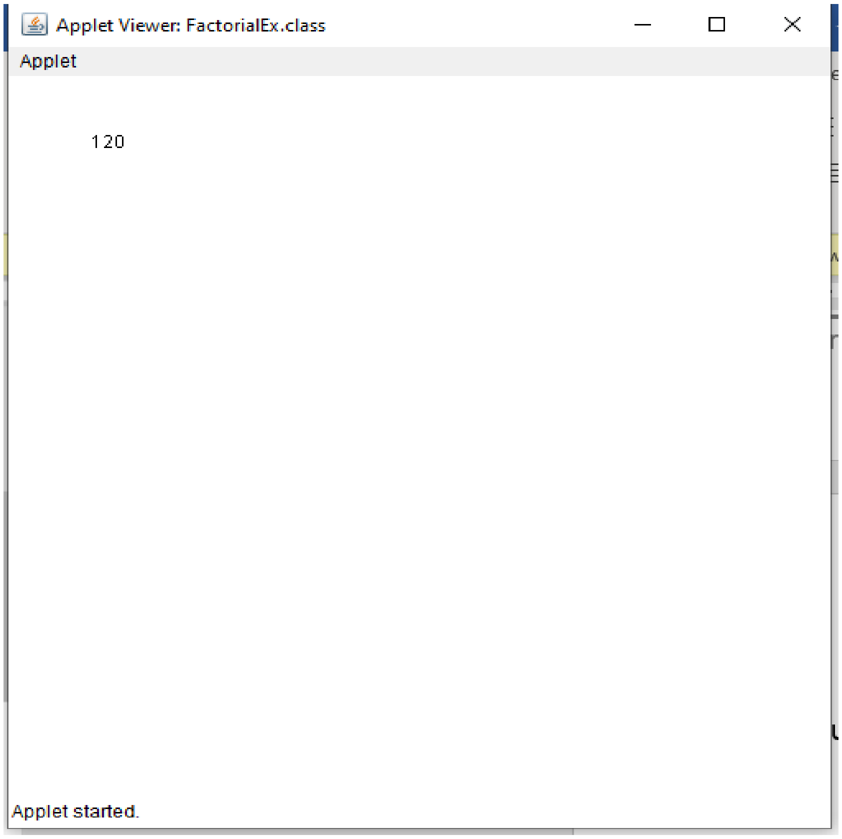


```
Welcome to applet
```

Applet started.

Program 6:

```
// Passing parameters to Applets (Factorial Program)
import java.applet.*;
import java.awt.*;
/* <applet code="FactorialEx6.class" width="500" height="500">
<param name="x" value=5></param>
</applet>    */
public class FactorialEx6 extends Applet
{
        public void paint(Graphics g)
        {
                String str=getParameter("x");
                int y=Integer.parseInt(str);
                int fact=1;
                for(int i=1;i<=y;i++)
                        fact=fact*i;
                g.drawString(fact+" ",50,50);
        }
}
```



```
120
```

Applet started.

Swings

Java Swing is used to create window-based applications or graphical user interface. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely **written in java**. Unlike AWT, Java Swing provides platform-independent and lightweight components. It is build on top of AWT API and acts as a replacement of AWT API, since it has almost every control corresponding to AWT controls. Swing component follows a Model-View-Controller architecture, where **model** represents data, **view** represents presentation and **controller** acts as an interface between model and view

AWT controls like Button, TextField and so on are developed in different languages that are not platform independent. Swing components are developed using Java. so they are written as JButton, JTextField and so on.

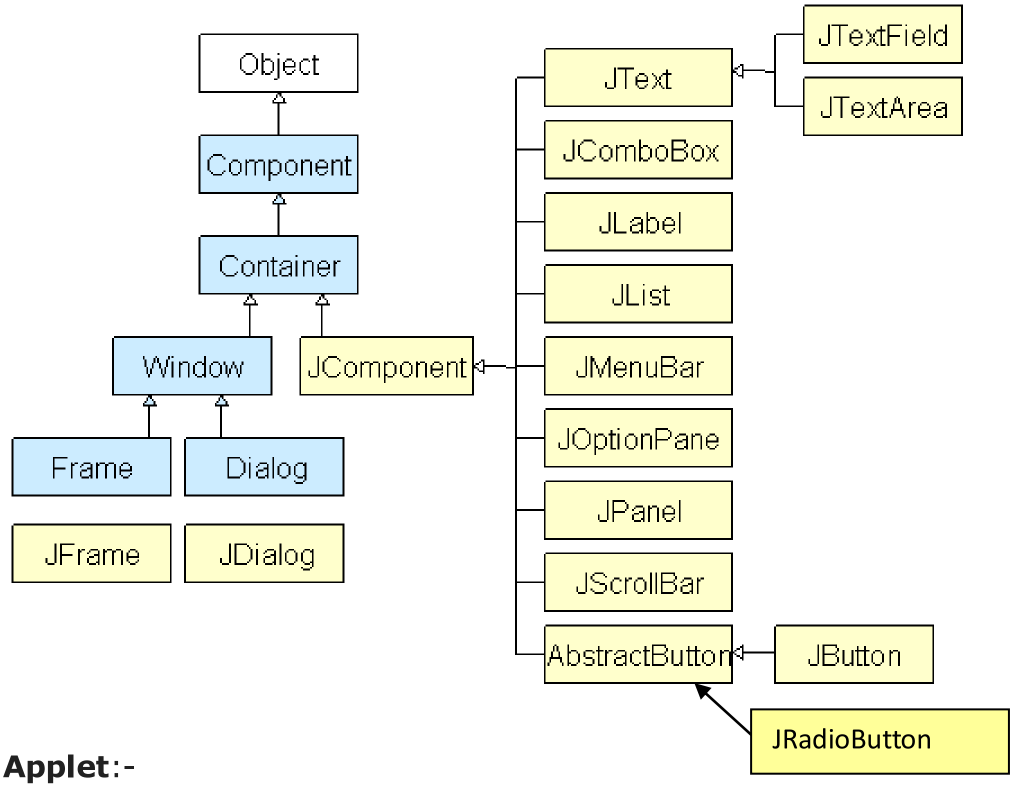
The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, etc.

Difference between AWT and Swing:-

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).	Swing **follows MVC**.

Exploring Swing:

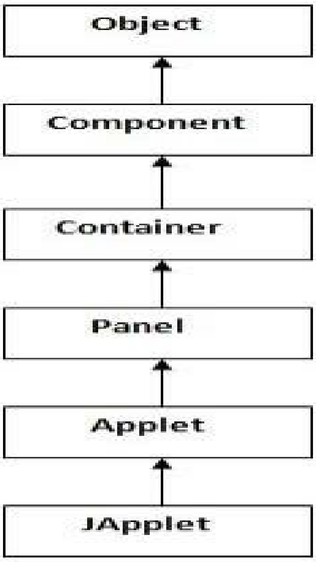
The hierarchy of JAVA SWING class



JApplet:-

JApplet is a class which extends **Applet** class. JApplet class has all the controls of swing. The JApplet class extends the Applet class. The javax.swing.**JApplet** class defines the core functionality of an **applet**. (**java**.awt.**Applet** is the older, AWT- based form)

Hierarchy of Applet



JComponent

The JComponent class is the base class of all Swing components except top-level containers. Swing components whose names begin with "J" are descendants of the JComponent class. For example, JButton, JScrollPane, JPanel, JTable etc. But, JFrame and JDialog don't inherit JComponent class because they are the child of top-level containers.

The JComponent class extends the Container class which itself extends Component. The Container class has support for adding components to the container.

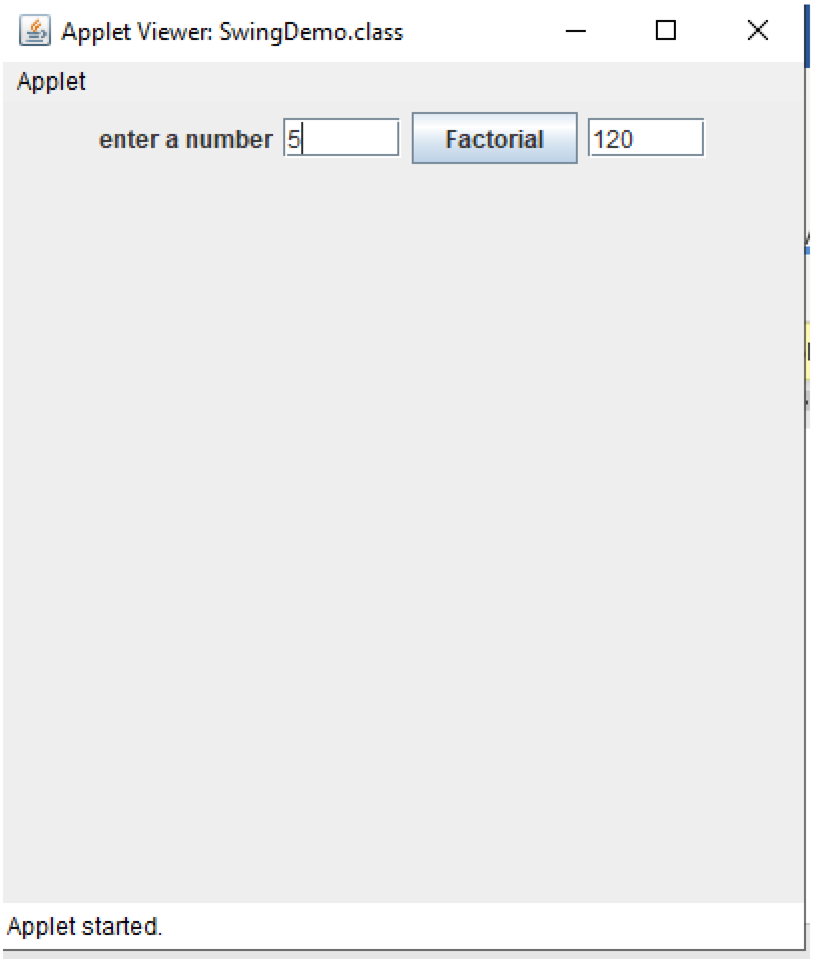
Constructor:

JComponent()

// Java Swing Program

```
import javax.swing.*;

import java.awt.*;

 import java.awt.event.*;

/*<applet code="SwingDemo.class" height=400 width=400> </applet>*/

 public class SwingDemo extends JApplet implements ActionListener

{

JButton b1;

JLabel l1;

JTextField t1,t2;
```

```
 public void init()

{

// These two statements are to be added for all swing programs

Container c=getContentPane();

c.setLayout(new FlowLayout());// here FlowLayout is not default layout

b1=new JButton("Factorial");

t1=new JTextField("",5);

t2=new JTextField("",5);

l1=new JLabel("enter a number");

// instead of adding awt controls to Applet Window in Applet programs,

// for  swing programs we have to add to container

c.add(l1);

c.add(t1);

c.add(b1);

c.add(t2);

b1.addActionListener(this);

}

public void actionPerformed(ActionEvent ae)

{

String str=t1.getText();

int num=Integer.parseInt(str);

int fact=1;

for(int i=1;i<=num;i++)

{

fact=fact*i;
}
t2.setText(fact+"");
}
}
```



enter a number 5

120

Applet started.

2. RadioButton:

Constructor:

JRadioButton(String str)

Method:

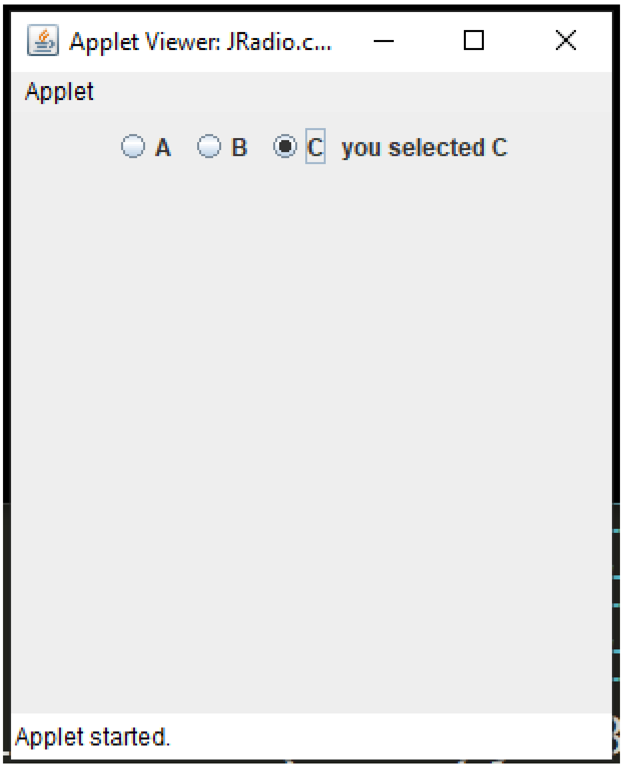
void add(AbstractButton ab)

Note: After creating the JRadioButton we need to Group these Buttons. For this we use ButtonGroup class

ButtonGroup()

// Program on Radiobuttons using java swing

```
import javax.swing.*;

import java.awt.event.*;

 import java.awt.*;

/*<applet code="JRadioButtonDemo.class" height=300 width=300>
</applet>*/

public class JRadioButtonDemo extends JApplet implements ActionListener

{
JLabel jl;
 public void init()
 {
  Container c=getContentPane();

  c.setLayout(new FlowLayout());

  JRadioButton b1=new JRadioButton("A");

  JRadioButton b2=new JRadioButton("B");

  JRadioButton b3=new JRadioButton("C");

  ButtonGroup bg=new ButtonGroup(); //adding JRadioButtons to ButtonGroup

  bg.add(b1);

  bg.add(b2);
  bg.add(b3);
  b1.addActionListener(this);

  b2.addActionListener(this);

  b3.addActionListener(this);

  c.add(b1);
  c.add(b2);
  c.add(b3);
  jl=new JLabel("select one");
  c.add(jl);
 }
public void actionPerformed(ActionEvent ae)
 {
  jl.setText("you selected "+ae.getActionCommand());
 }
}
```



○ A ○ B ● C you selected C

Applet started.

3. JComboBox:// in Applets it is Choice

JComboBox:

Swing provides a combo box through the JComboBox class. A combo box normally displays one entry ,but it will also display a drop-down list that allows a user to select a different entry.

Constructor:

JComboBox(Object[] items)

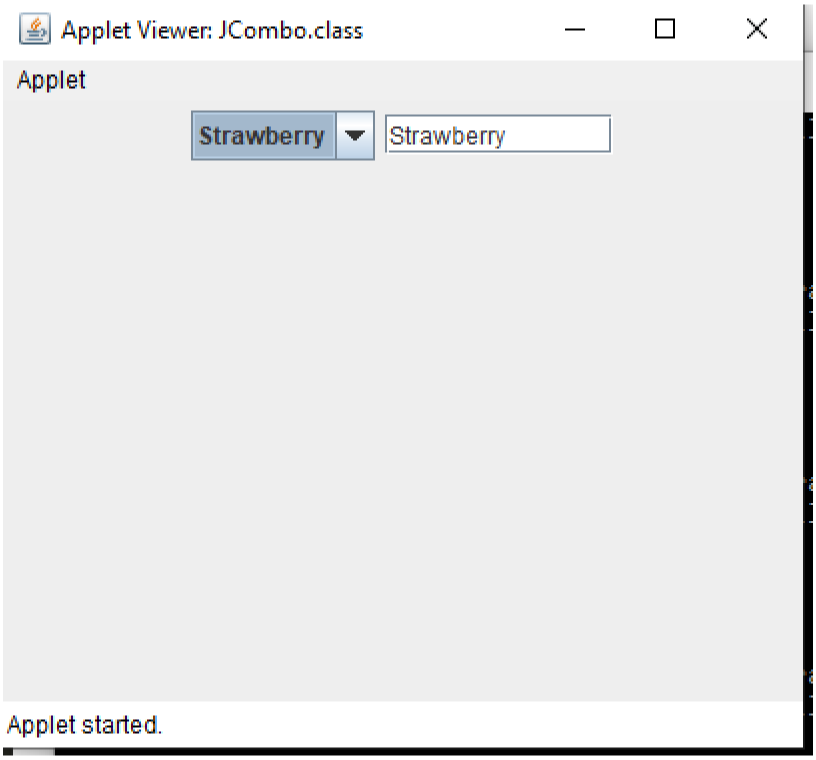
Methods:

void addItem(Object obj**)**

Object getSelectedItem()

// Demonstration on JCombobox

```
import javax.swing.*;

import java.awt.*;

import java.awt.event.*;

/*<applet code="JCombo.class" height=300 width=400></applet>*/

public class JCombo extends JApplet implements ItemListener

{

  JComboBox jcb;

  JTextField tf;

public void init()

{

Container c=getContentPane();

c.setLayout(new FlowLayout());

jcb=new JComboBox();

 jcb.addItem("Vanilla");

jcb.addItem("Choclate");

 jcb.addItem("Strawberry");

c.add(jcb);

tf=new JTextField(" ",10);

c.add(tf);

jcb.addItemListener(this);

}
public void itemStateChanged(ItemEvent ie)

{

tf.setText(ie.getItem()+"");

}

}
```



4. JTabbedPane:

JTabbedPane encapsulates a tabbed pane. JTabbedPane defines by using below constructor.

JTabbedPane()

Methods:

void addTab(String name, Component comp)

Procedure:

The general procedure to use a tabbed pane is outlined here:

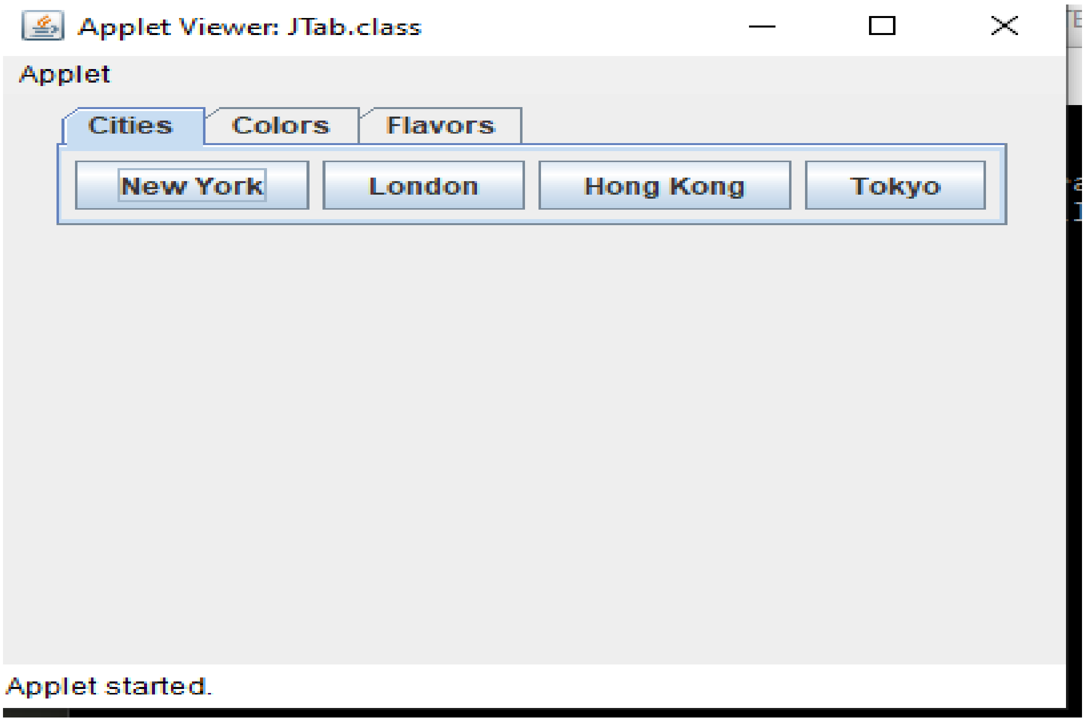
1) create an instance of JTabbedPane.

2) Add each tab by calling addTab()

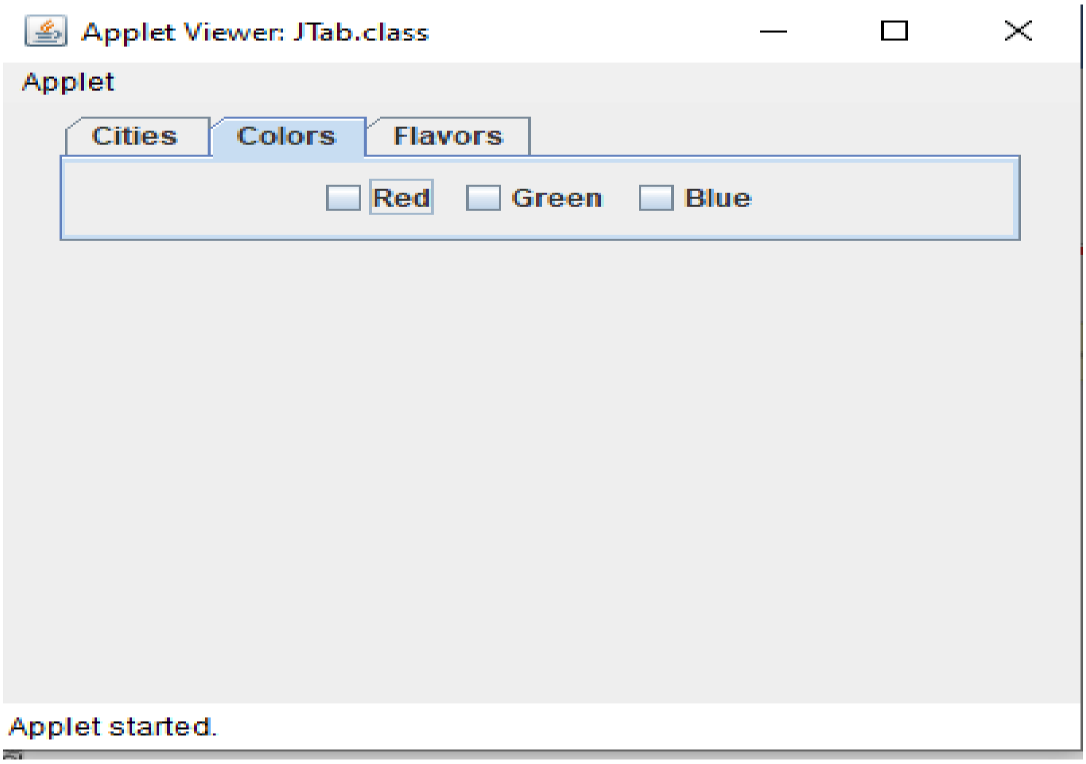
3) Add the tabbed pane to the content pane.

```
// Demonstration of JTabbedPane
```

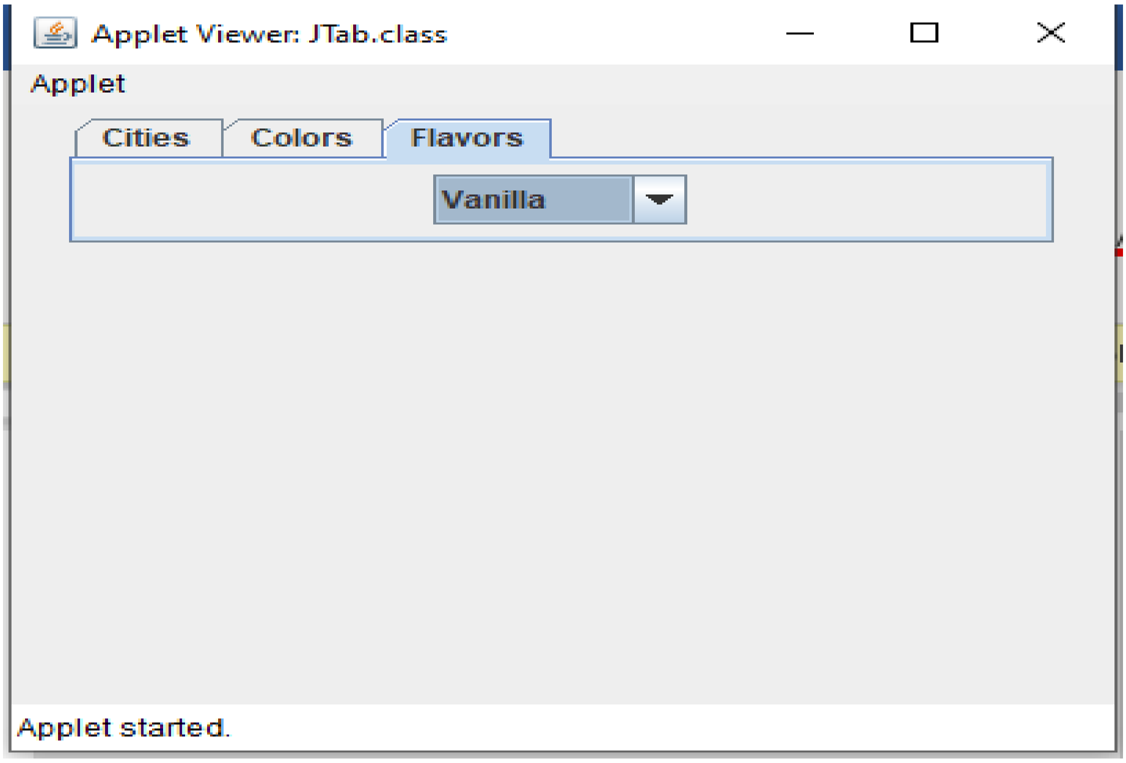
```
 import javax.swing.*;

import java.awt.*;

/*<applet code="JTabbedPaneDemo.class" height=300 width=400>
</applet>*/
public class JTabbedPaneDemo extends JApplet
{
 public void init()
 {
 Container c=getContentPane();
  c.setLayout(new FlowLayout());
  JTabbedPane jtp=new JTabbedPane();
   jtp.addTab("Cities", new CitiesPanel());
  //when the addTab() method is invoked it goes to CitiesPanel() constructor
  // and the Cities New York,London, so on are added to the Cities Tab
   jtp.addTab("Colors", new ColorsPanel());
   jtp.addTab("Flavors", new FlavorsPanel());
   c.add(jtp);
 }
}
 class CitiesPanel extends JPanel
 {
  public CitiesPanel()
  {
  JButton b1=new JButton("New York");
  add(b1);
  JButton b2=new JButton("London");
  add(b2);
  JButton b3=new JButton("Hong Kong");
  add(b3);
 JButton b4=new JButton("Tokyo");
 add(b4);
 }
}
class ColorsPanel extends JPanel
 {
 public ColorsPanel()
 {
 JCheckBox cb1=new JCheckBox("Red");
 JCheckBox cb2=new JCheckBox("Green");
 JCheckBox cb3=new JCheckBox("Blue");
 add(cb1);
 add(cb2);
 add(cb3);
 }
 }
class FlavorsPanel extends JPanel
 {
 public FlavorsPanel()
 {
 JComboBox jcb=new JComboBox();
  jcb.addItem("Vanilla");
 jcb.addItem("Choclate");
  jcb.addItem("Strawberry");
  add(jcb);
 }
 }
```



Applet started.



Applet started.



JScrollPane:

Constructor:

JScrollPane(Component comp)

Procedure

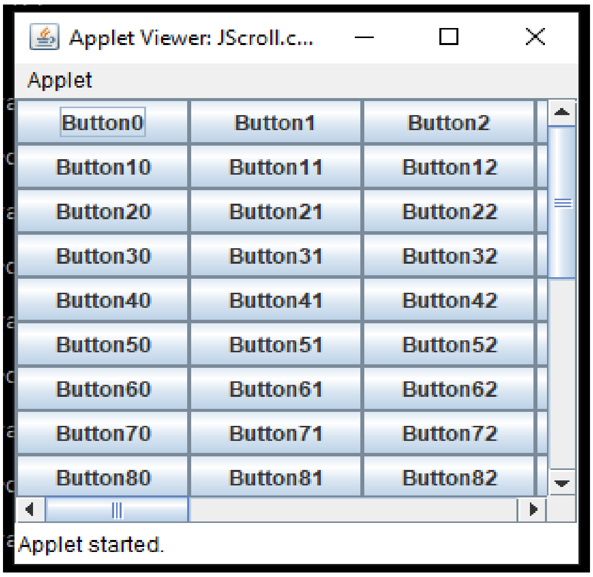
1. Create the component to be scrolled. (ex:-Panel)
2. Create an instance of JScrollPane, pass the component(Panel) to be scrolled.
3. Add the scroll pane to the content pane (Container).

```
// Demonstration of JSCrollPane
import javax.swing.*;
import java.awt.*;
 /*<applet code="JScrollPaneDemo.class" height=250
 width=300></applet>*/

 public class JScrollPaneDemo extends JApplet //Creating an Applet

 {

  public void init()

  {

   Container c=getContentPane();
c.setLayout(new BorderLayout());
JPanel jp=new JPanel(); //1. Create the component to be scrolled.
jp.setLayout(new GridLayout(20,20));
for(int i=0;i<200;i++)
 {
  jp.add(new JButton("Button"+i));
 }
 // 2. Create an instance of JScrollPane, pass the component to be
 //scrolled.
  JScrollPane jsp=new JScrollPane(jp);//
 //3. Add the scroll pane to the content pane.
c.add(jsp, BorderLayout.CENTER);
}
}
```

Note 1: Create the JPanel add the Buttons to the JPanel. Add the JScrollPane to the JPanel. Then add the JScrollPane to the ContentPane.

Note 2: If you add less number of Buttons to the JPanel then the Scroll Bars are not visible even if you create JScrollPane. So, add more number of Buttons to visualize the JScrollPane.

Note 3: Vertical ScrollBar is automatically created if we add more number of rows and horizontal ScroolBar is automatically created if we add more number of columns



Applet started.

JTree

A tree is a component that presents a hierarchical view of data. Trees are implemented in swing by the JTree class.

Constructors:

JTree(TreeNode tn)

The Tree node interface declares methods that obtain information about a tree node.

The mutable TreeNode interface extends TreeNode. It declares methods that can insert and remove child nodes or change the parent node. The DefaultMutableTreeNode class implements the MutableTreeNode interface. It represents a node in a tree.

one of its constructors is shown here:

DefaultMutableTreeNode(Object obj)

To create a hierarchy of tree nodes, the add() method of DefaultMutableTreeNode can be used.

Its signature is shown here: void add(MutableTreeNode child)

here are the steps to follow to use of JTree.

1.Create an instance of JTree.

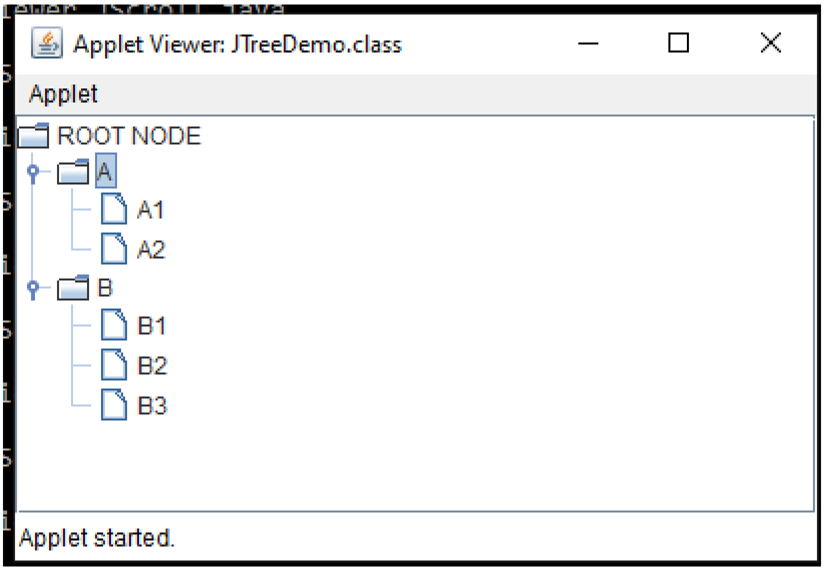
2.Create JScrollPane and pass the tree as the object to be scrolled.

(Add JScrollpane to the Tree)

3.Add the JScrollpane to the content pane.

// Demonstration on Trees

```
import javax.swing.*;

import java.awt.*;

import javax.swing.tree.*;
/*<applet code="JTreeDemo.class" height=200
width=400></applet>*/

public class JTreeDemo extends JApplet

{
 public void init()
 {
  Container c=getContentPane();
  c.setLayout(new BorderLayout());
  DefaultMutableTreeNode top=new DefaultMutableTreeNode("ROOT
  NODE"); DefaultMutableTreeNode a=new DefaultMutableTreeNode("A");
  top.add(a);
  DefaultMutableTreeNode a1=new DefaultMutableTreeNode("A1");
  DefaultMutableTreeNode a2=new DefaultMutableTreeNode("A2");
  a.add(a1);
  a.add(a2);
  DefaultMutableTreeNode b=new DefaultMutableTreeNode("B");
  top.add(b);
  DefaultMutableTreeNode b1=new DefaultMutableTreeNode("B1");
  DefaultMutableTreeNode b2=new DefaultMutableTreeNode("B2");
  DefaultMutableTreeNode b3=new DefaultMutableTreeNode("B3");
   b.add(b1);
  b.add(b2);
  b.add(b3);
  //add the root node to the JTree
  JTree tree=new JTree(top); // 1. Create an instance of JTree.
  JScrollPane jsp=new JScrollPane(tree); // 2. Adding scroll pane to tree
  c.add(jsp, BorderLayout.NORTH); //3. adding scroll pane to container
}
}
```



Note: If u create more number of nodes then the JScrollPane is visible

JTable

JTable is a component that displays rows and columns of data. JTable supplies several constructors.

Constructor:

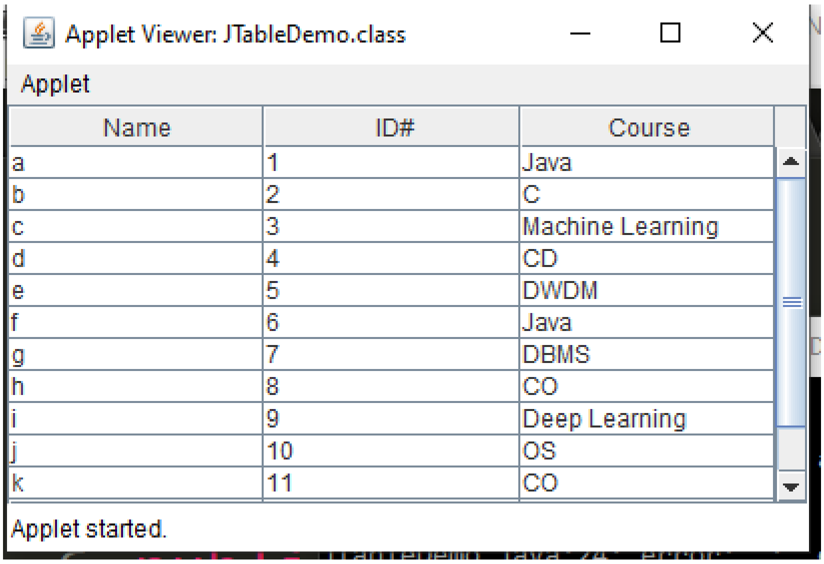
JTable(Object data[][],Object colHead[])

Here are the steps required to set up a simple JTable that can be used to display the data.

1. Create an instance of JTable.
2. Create JScrollPane and pass the table as the object to be scrolled.(Add the table to the scrollpane)
3. Add the scroll pane to the content pane.

// Demonstration on JTable

```
import javax.swing.*;

 import java.awt.*;

/*<applet code="JTableDemo.class" height=200
width=400></applet>*/

public class JTableDemo extends JApplet

{

 public void init()
{
Container c=getContentPane();
c.setLayout(new BorderLayout());

String[] colHeads={"Name","ID#","Course"};
Object[][] data={
{"a","1","Java"},
{"b","2","C"},
{"c","3","Machine Learning"},
{"d","4","CD"},
{"e","5","DWDM"},
{"f","6","Java"},
{"g","7","DBMS"},
{"h","8","CO"},
{"i","9","Deep Learning"},
{"j","10","OS"},
{"k","11","CO"},
{"l","12","FLAT"},
{"m","13","DAA"}
  };
JTable table=new JTable(data,colHeads);//1. Create an instance of JTable.
JScrollPane jsp=new JScrollPane(table);//2. Add the table to the scrollpane
c.add(jsp, BorderLayout.CENTER);//3. Add the scroll pane to the content pane.

}
}
```



Name	ID#	Course
a	1	Java
b	2	C
c	3	Machine Learning
d	4	CD
e	5	DWDM
f	6	Java
g	7	DBMS
h	8	CO
i	9	Deep Learning
j	10	OS
k	11	CO

Applet started.

Applet	**JApplet**
Label	JLabel
Button	JButton
TextField	JTextField
TextArea	JTextArea
Checkbox	JCheckbox
CheckboxGroup	JButtonGroup
Choice	JCombobox
List	JList
	JTabbedPane
	JScrollPane
	JTable
	JTree

JFrame

The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class. JFrame works like the main window where components like labels, buttons, textfields are added to create a GUI.

Unlike Frame, JFrame has the option to hide or close the window with the help of setDefaultCloseOperation(int) method.

Constructors

onstructor	escription
JFrame()	It constructs a new frame that is initially invisible.
JFrame(GraphicsConfiguration gc)	It creates a Frame in the specified GraphicsConfiguration of a screen device and a blank title.
JFrame(String title)	It creates a new, initially invisible Frame with the specified title.
JFrame(String title, GraphicsConfiguration gc)	It creates a JFrame with the specified title and the specified GraphicsConfiguration of a screen device.

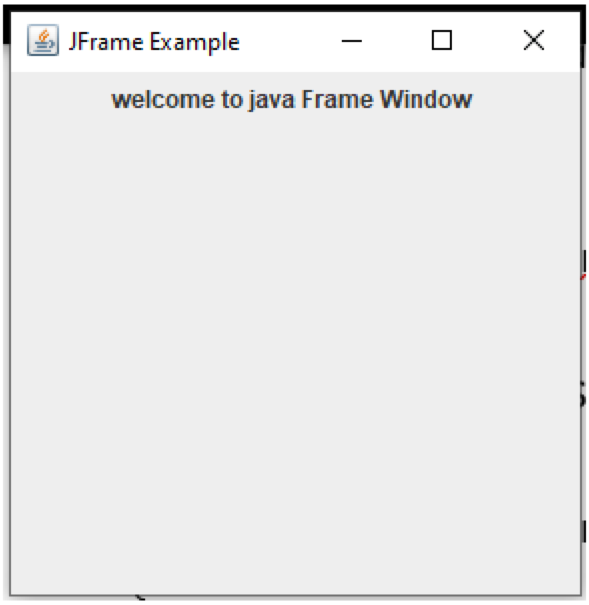
Note: Without creating applet tag we can create a window which is a Frame.

JFrame

```
import java.awt.*;
import javax.swing.*;
public class FrameEx11
{
FrameEx11()
{
  JFrame fm=new JFrame("JFrame Example");
 //Creating a frame with Title.
  JLabel lb = new JLabel("welcome to java Frame Window ");
//Creating a label
  fm.setLayout(new FlowLayout(FlowLayout.CENTER));
  fm.add(lb);         //adding label to the frame.
  fm.setSize(300, 300);              //setting frame size.
  fm.setVisible(true);           //set frame visibilty true.
  fm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
//For closing the Frame
}
public static void main(String args[])
{
  FrameEx11 ta = new FrameEx11();
}
}
```

Compilation: javac FrameEx11.java

Exectuion: java FrameEx11



welcome to java Frame Window