Chapter 2

Swing(10M)

- 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.

- Swing is a set of classes that provides **more powerful and flexible components than are possible with the AWT**.

- In addition to the familiar components, such as buttons, check boxes, and labels, Swing supplies **several exciting additions, including tabbed panes, scroll panes, trees, and tables.**

- Even familiar components such as **buttons have more capabilities in Swing**. For example, **a button may have both an image and a text string associated with it**.

- Also,the image can be changed as the state of the button changes.

- Unlike AWT components, Swing components are not implemented by platform-specific code. Instead, they are **written entirely in Java** and, therefore, **are platform-**

independent. The term ***lightweight*** is used to describe such elements.

- The Swing-related classes are contained in **javax.swing**

- **Java Swing is implemented entirely in the Java programming language.**

***Why AWT components are heavy-weight while Swing components are light-weight in Java?**

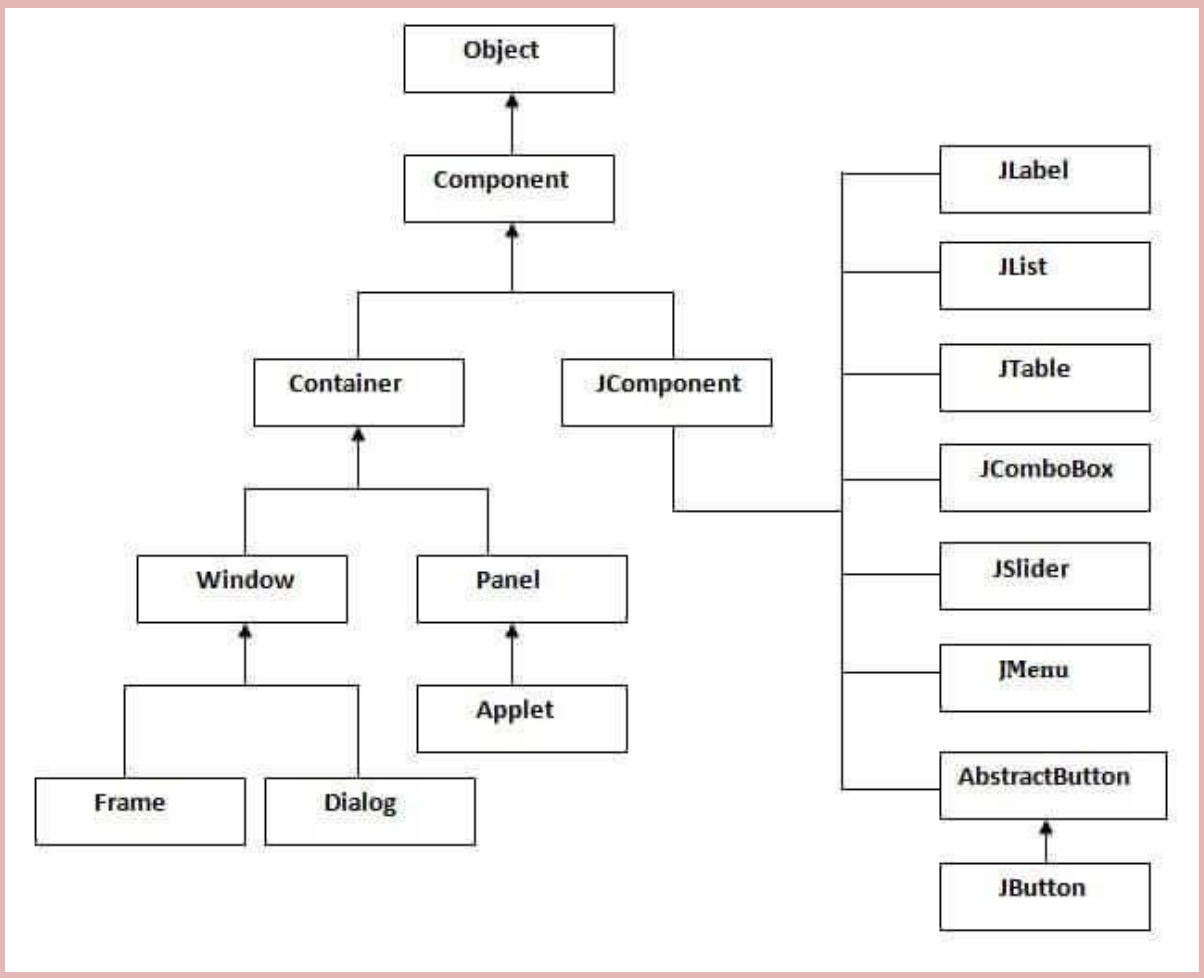
- First of all, by **a heavy-weight, it means the code will take comparatively more time to load and it will consume more System resources**. **AWT is considered to be heavy-weight because its components are dependent on the underlying Operating System**. For instance, When we create an object of java.awt.Checkbox class, its underlying Operating System will generate a checkbox for us. This is also the reason, AWT components are platform dependent.

what is JFC in swing

JFC is short for Java Foundation Classes, which encompass a group of features for building graphical user interfaces (GUIs) and adding rich graphics functionality and interactivity to Java applications

The hierarchy of java swing API is given below

.



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**.

- **What is JFC**

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

A **container** has several layers in it. You can think of a layer as a transparent film that overlays the **container**. In Java Swing, the layer that is used to hold objects is called the **content pane**.

Objects are added to the **content pane** layer of the **container**. The **getContentPane() method** retrieves the **content pane** layer so that you can add an **object** to it.

getContentPane().setBackground(Color.YELLOW);

JApplet

- **JApplet** is rich with functionality that is not found in Applet. For example, JApplet supports various "panes," such as the **contentpane, the glass pane, and the root pane**
  - The content pane can be obtained via the method shown here: **Container getContentPane( )**

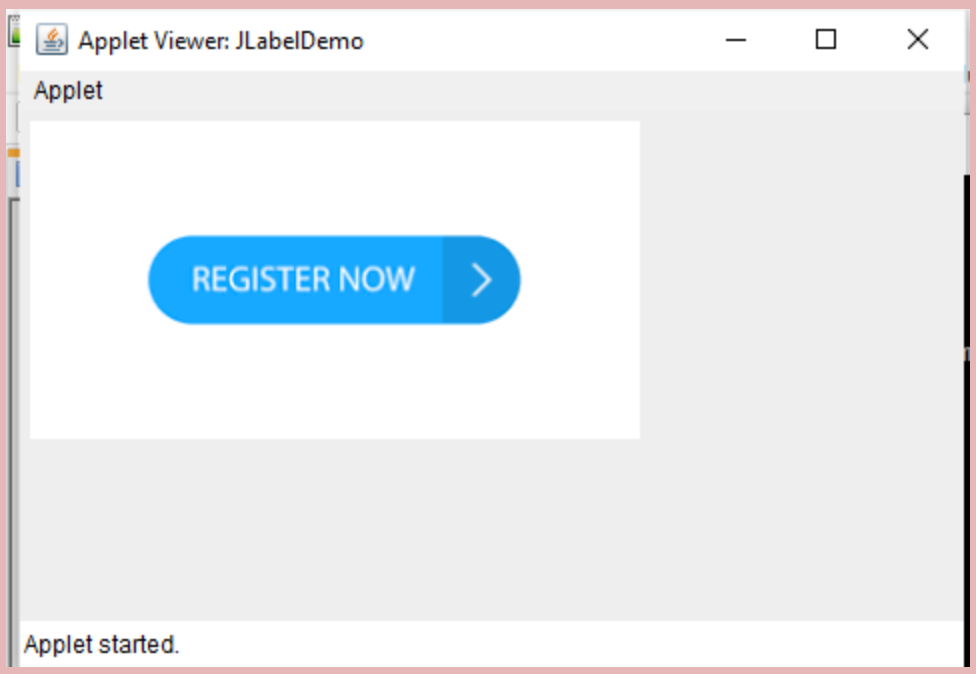
Swing Components

1)JLABEL

- JLabel class, which extends **JComponent**.

```
import java.awt.*;
import javax.swing.*;
/*
<applet code="JLabelDemo" width=250 height=150>
</applet>
```

```
*/
public class JLabelDemo extends JApplet
{
public void init()
{
// Get content pane
Container  contentPane = getContentPane(); //background to display components
contentPane.setLayout(new FlowLayout());

// Create an icon
ImageIcon img = new ImageIcon("register.jpg");
// Create a label
JLabel j = new JLabel("HELLO", img,JLabel.LEFT);
// Add label to the content pane
contentPane.add(j);
}
}
```



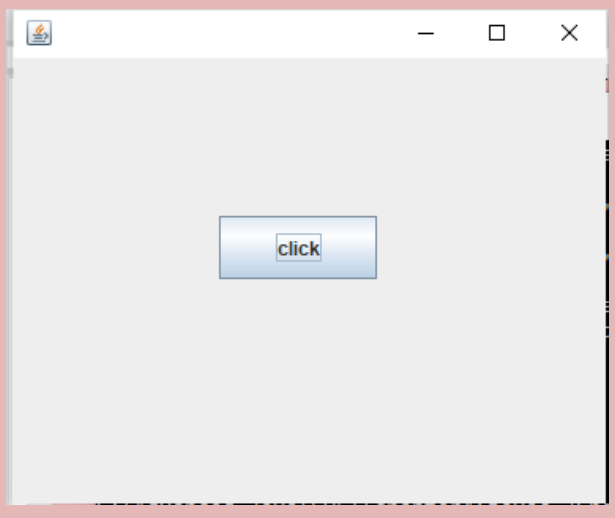
2)JBUTTON

- The **JButton** class provides the functionality of a push button
- Swing **buttons** are subclasses of the **AbstractButton** class, which extends **JComponent**.

*JButton program using frame

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

public class swingframe extends JFrame
{

swingframe()
{
```

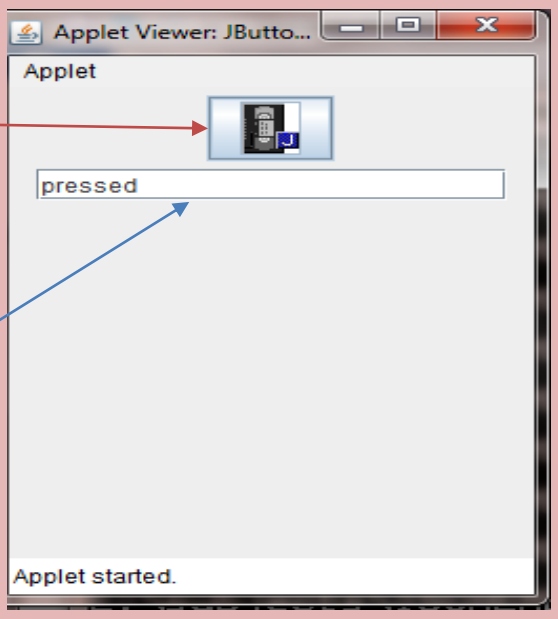
```
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());

JButton b=new JButton("click"); //creating instance of JButton
b.setBounds(130,100,100, 40);

contentPane.add(b); //adding button in JFrame
}

public static void main(String[] args)
{
swingframe s=new swingframe();
s.setSize(400,500);//400 width and 500 height
s.setLayout(null);//using no layout managers
s.setVisible(true);//making the frame visible
}
}
```



Button using Event Handling

```
import java.awt.*;

import java.awt.event.*;

import javax.swing.*;

/*

<applet code="JButtonDemo" width=250 height=300>

</applet>*/

public class JButtonDemo extends JApplet implements ActionListener

 {

JTextField t1;

public void init()

{

// Get content pane

Container contentPane = getContentPane();

contentPane.setLayout(new FlowLayout());


// Add buttons to content pane

ImageIcon img= new ImageIcon("jpgIcon.jpg");

JButton jb = new JButton(img);
```

```
t1 = new JTextField(20);

jb.setActionCommand("pressed");
jb.addActionListener(this);

contentPane.add(jb);
contentPane.add(t1);
}
public void actionPerformed(ActionEvent ae)
 {
t1.setText(ae.getActionCommand());
}
}
```



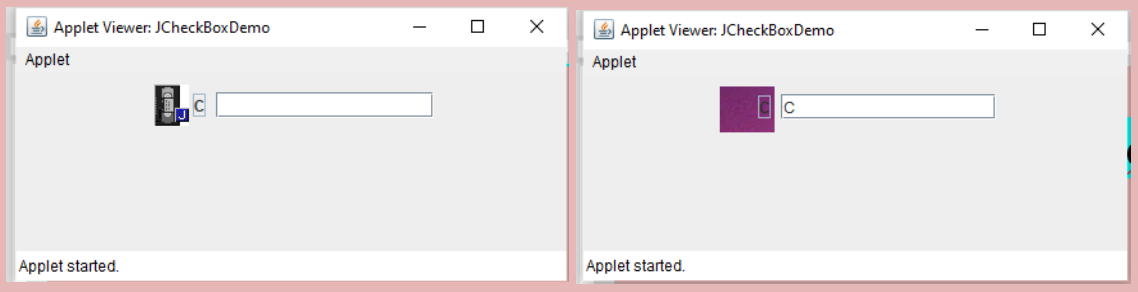
3)JCHECKBOX

- The **JCheckBox** class, which provides the functionality of a check box, is a concrete implementation of **AbstractButton**. Its immediate superclass is **JToggleButton**, which provides support for **two-state buttons**.

```
import java.awt.*;

import javax.swing.*;

/*

<applet code="JCheckBoxDemo" width=400 height=50>

</applet>

*/

public class JCheckBoxDemo extends JApplet

 {

JTextField jtf;

public void init()

 {

// Get content pane

Container contentPane = getContentPane();
```

```
contentPane.setLayout(new FlowLayout());

// Create icons
ImageIcon img1 = new ImageIcon("jpgIcon.jpg");
ImageIcon img2= new ImageIcon("Winter.jpg");
ImageIcon img3 = new ImageIcon("Sunset.jpg");

// Add check boxes to the content pane
JCheckBox cb = new JCheckBox("C",img1);
contentPane.add(cb);

cb.setRolloverIcon(img2);
cb.setSelectedIcon(img3);

jtf = new JTextField(15);
contentPane.add(jtf);
}
```

```
public void itemStateChanged(ItemEvent ie)
 {
JCheckBox cb = (JCheckBox)ie.getItem();
jtf.setText(cb.getText());
}
}
```

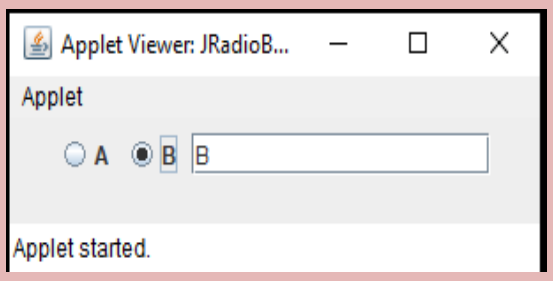


4)JRADIOBUTTON

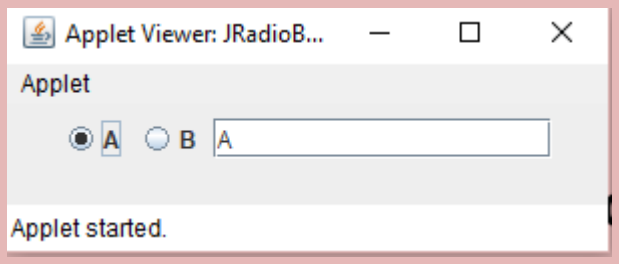
- Radio buttons are supported by the **JRadioButton** class, which is a concrete implementation of **AbstractButton**. Its immediate superclass is **JToggleButton**, which provides support for two-state buttons

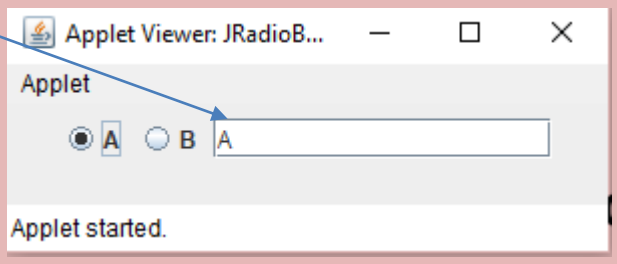
```
import java.awt.*;
import javax.swing.*;
/*
```

```
<applet code="JRadioButtonDemo" width=300 height=50>
</applet>
*/
public class JRadioButtonDemo extends JApplet implements
ActionListener
{
public void init()
{
JTextField tf;
// Get content pane
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());

// Add radio buttons to content pane
JRadioButton b1 = new JRadioButton("A");
contentPane.add(b1);

JRadioButton b2 = new JRadioButton("B");
contentPane.add(b2);
```



```
// Define a button group

ButtonGroup bg = new ButtonGroup();
bg.add(b1);
bg.add(b2);

tf = new JTextField(15);
contentPane.add(tf);
}
public void actionPerformed(ActionEvent ae)
{
tf.setText(ae.getActionCommand());
}
}
```



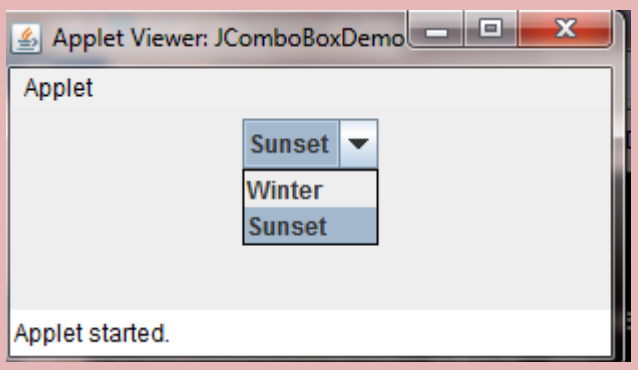


5)JCOMBOBOX

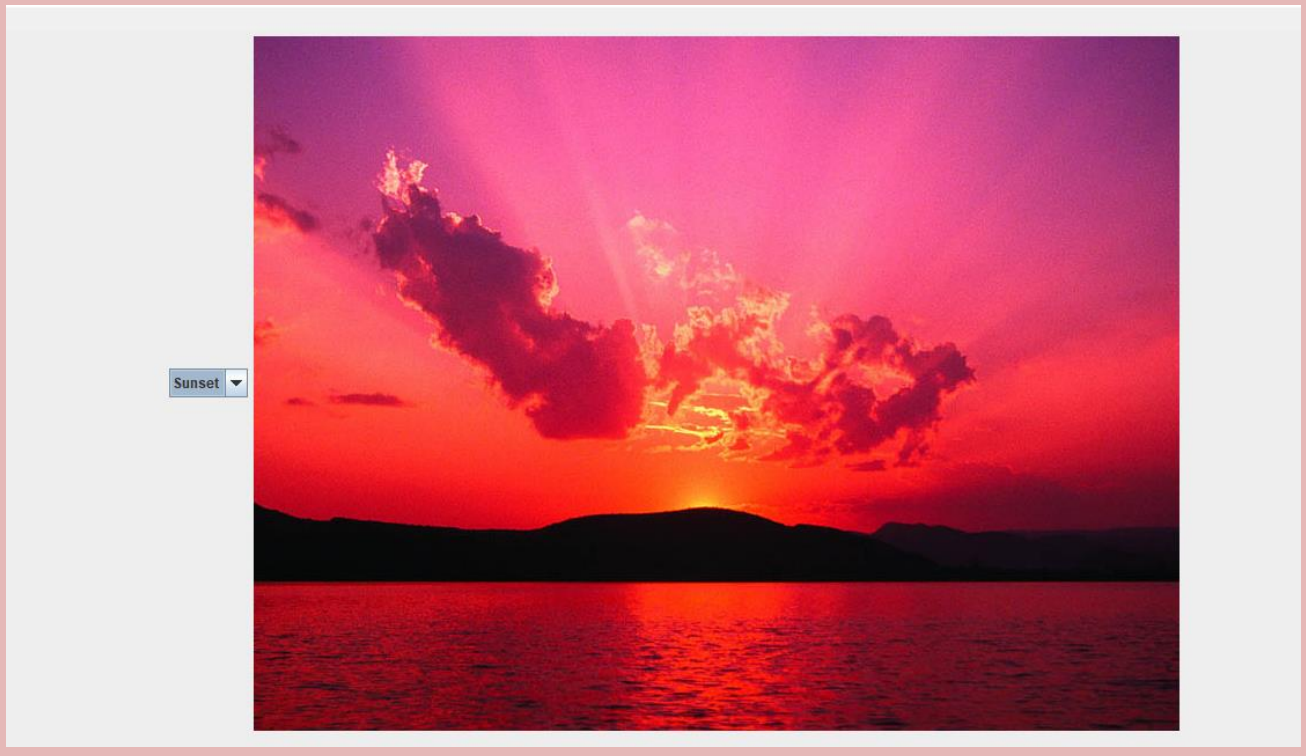
- Swing provides a *combo box* (a combination of a text field and a drop-down list)through the **JComboBox** class, which extends **JComponent**
- Items are added to the list of choices via the **addItem()** method

```
import java.awt.*;

import java.awt.event.*;

import javax.swing.*;

/*

<applet code="JComboBoxDemo" width=300 height=100>

</applet>

*/

public class JComboBoxDemo extends JApplet

implements ItemListener

{

JLabel jl;

ImageIcon Winter,Sunset;


public void init()
```

```
{
// Get content pane
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());

// Create a combo box and add it
JComboBox jc = new JComboBox();
jc.addItem("Winter");
jc.addItem("Sunset");

jc.addItemListener(this);
contentPane.add(jc);

// Create label
jl = new JLabel();
contentPane.add(jl);
}
```



```
public void itemStateChanged(ItemEvent ie)
{
String s = (String)ie.getItem();
ImageIcon i=new ImageIcon(s + ".jpg");
jl.setIcon(i);
}
}
```



6) JScrollPane

- Scroll panes are implemented in Swing by the **JScrollPane** class, which extends **JComponent**.

Constant	Description
HORIZONTAL_SCROLLBAR_ALWAYS	Always provide horizontal scroll bar
HORIZONTAL_SCROLLBAR_AS_NEEDED	Provide horizontal scroll bar,if needed
VERTICAL_SCROLLBAR_ALWAYS	Always provide vertical scroll bar
VERTICAL_SCROLLBAR_AS_NEEDED	Provide vertical scroll bar,if needed

Here are the steps that you should follow to use a scroll pane in an applet:

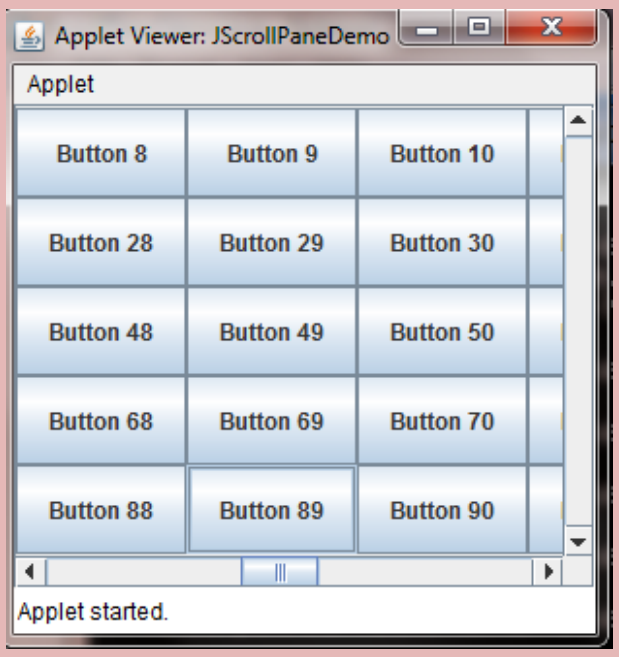
1. Create a panel object.

2. Create a JScrollPane object. (The arguments to the constructor specify the component and the VALUES for vertical and horizontal scroll bars.)

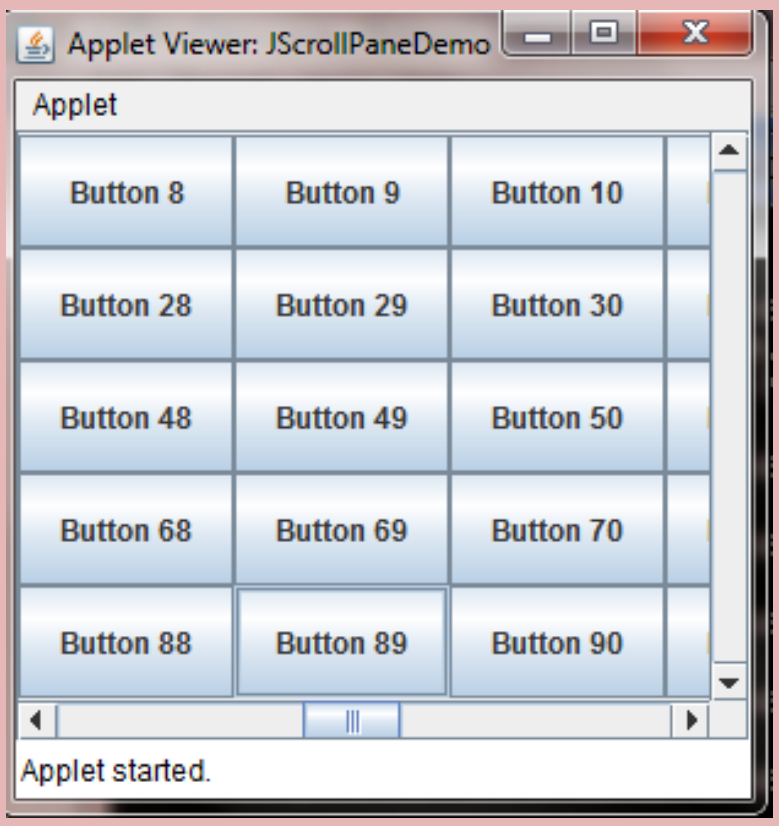
3. Add the scroll pane to the content pane of the applet.

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

```
public class JScrollPaneDemo extends JApplet
{
public void init()
{
// Get content pane
Container contentPane = getContentPane();

JPanel jp = new JPanel();   //step1-create panel object
jp.setLayout(new GridLayout(5,5));

int b = 0;
for(int i = 0; i <10; i++)
{
for(int j = 0; j <10; j++)
{
jp.add(new JButton("Button " + b));
b++;
}
```



```
}
// Add panel to a scroll pane
int v = ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
int h = ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS;

JScrollPane jsp = new JScrollPane(jp, v, h);
// step 2 -Create a JScrollPane object

// Add scroll pane to the content pane
contentPane.add(jsp);
//Step 3. Add the scroll pane to the contentpane
}
}
```



7)JTabbedPane

- Tabbed panes are encapsulated by the **JTabbedPane** class, which extends **JComponent**. We will use its default constructor. Tabs are defined via the following method:
  void **addTab**(String *str*, Component *comp*)

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

1. Create a JTabbedPane object.

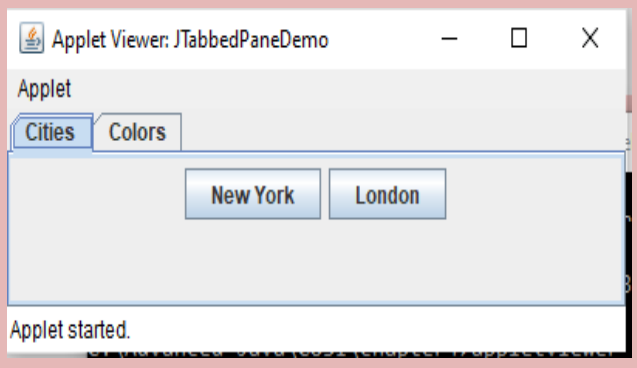
2. Call addTab( ) to add a tab to the pane. (The arguments to this method define

the title of the tab and the component it contains.)

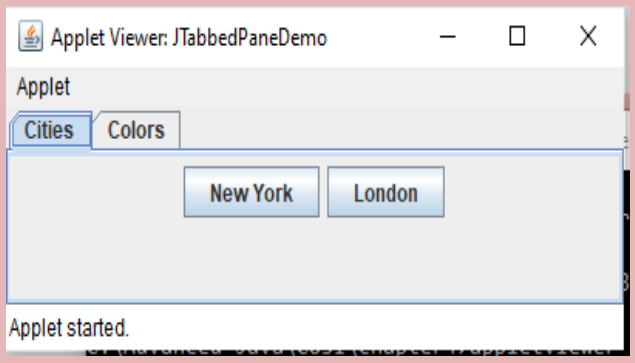
3. Repeat step 2 for each tab.

4. Add the tabbed pane to the content pane of the applet.

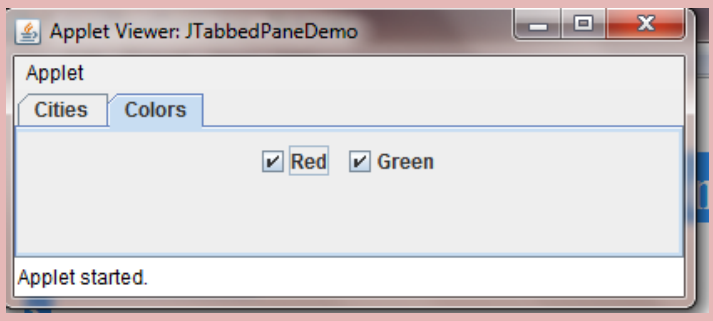
```
import javax.swing.*;
/*
<applet code="JTabbedPaneDemo" width=400 height=100>
</applet>
*/

public class JTabbedPaneDemo extends JApplet
{
public void init()
{
JTabbedPane jtp = new JTabbedPane();

jtp.addTab("Cities", new CitiesPanel());
```



```
jtp.addTab("Colors", new ColorsPanel());

getContentPane().add(jtp);
}
}

class CitiesPanel extends JPanel
{
public CitiesPanel()
{
JButton b1 = new JButton("New York");
add(b1);
JButton b2 = new JButton("London");
add(b2);
}
}
```



```
class ColorsPanel extends JPanel
{
public ColorsPanel()
{
JCheckBox cb1 = new JCheckBox("Red");
add(cb1);
JCheckBox cb2 = new JCheckBox("Green");
add(cb2);
}
}
```



8)JTable

- Tables are implemented by the **JTable** class, which extends **JComponent**.
- One of its constructors is shown here:
  JTable(Object *data*[ ][ ], Object *colHeads*[ ])

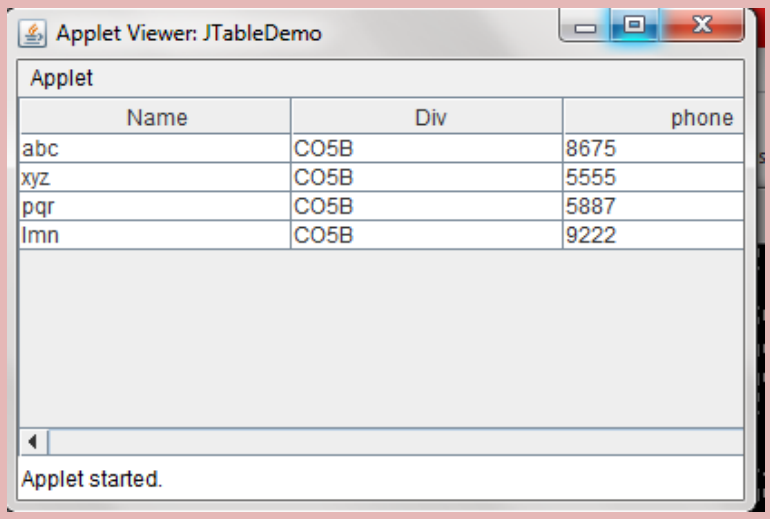
- Here, *data* is a two-dimensional array of the information to be presented, and *colHeads* is a one-dimensional array with the column headings.

Here are the steps for using a table in an applet:

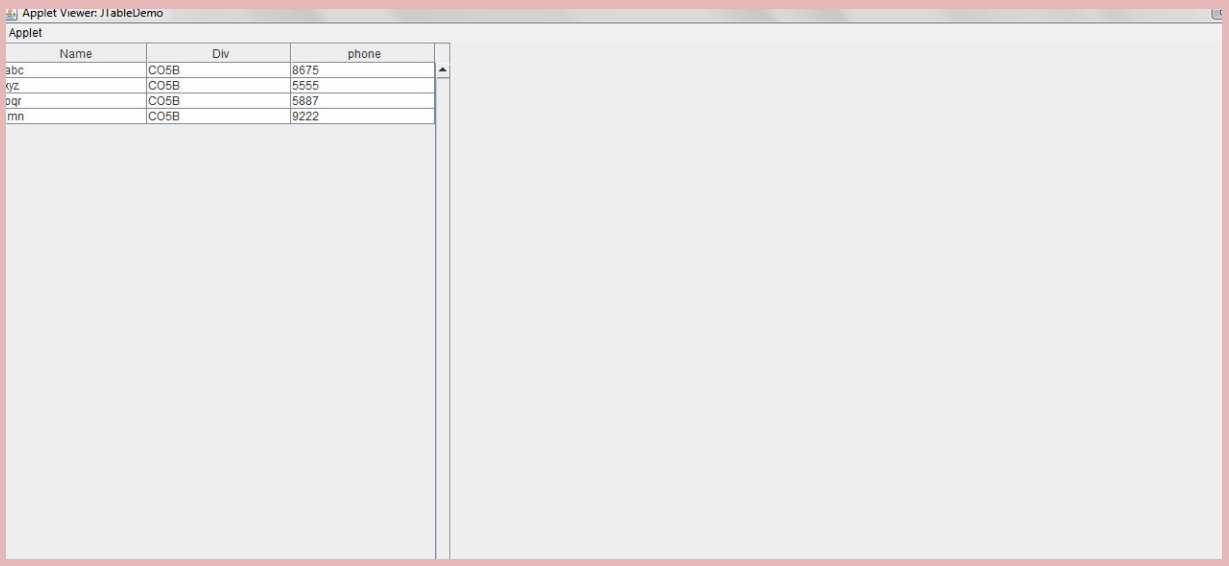
1. Create a **JTable** object.
2. Create a **JScrollPane** object. (The arguments to the constructor specify the table
and the values for vertical and horizontal scroll bars.)
3. Add the table to the scroll pane.
4. Add the scroll pane to the content pane of the applet.

```
import java.awt.*;

import javax.swing.*;


/*

<applet code="JTableDemo" width=400 height=200>

</applet>

*/

public class JTableDemo extends JApplet

{

public void init()

 {

// Get content pane

Container contentPane = getContentPane();
```

```
contentPane.setLayout(new BorderLayout());

// Initialize column headings
String colheads[] = { "Name", "Div", "phone" };

// Initialize data
 Object data[][] = {
{ "abc", "CO5B", "8675" },
{ "xyz", "CO5B", "5555" },
{ "pqr", "CO5B", "5887" },
{ "lmn", "CO5B", "9222" }
};

// Create the table
JTable table = new JTable(data, colheads);
int v = ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
int h = ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;
JScrollPane jsp = new JScrollPane(table, v, h);
```



```
// Add scroll pane to the content pane
contentPane.add(jsp, BorderLayout.WEST);


}

}
```



9)JTREE

- Trees are implemented in Swing by the **JTree** class, which extends **JComponent**.
- Some of its constructors are shown here:

JTree(Hashtable *ht*)
JTree(Object *obj*[ ])
JTree(TreeNode *tn*)

JTree(Vector *v*)

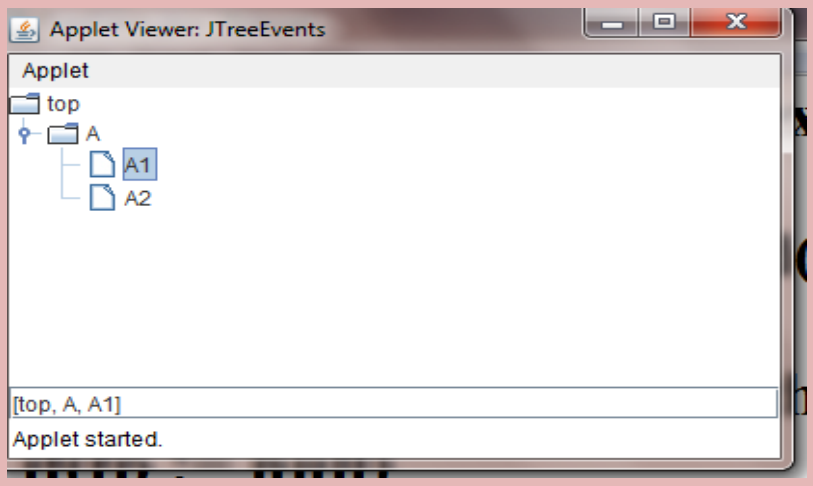
- The **getPathForLocation( )** method is used to translate a mouse click on a specific point of the tree to a tree path.
- The **DefaultMutableTreeNode** class implements the **MutableTreeNode** interface.It represents a node in a tree

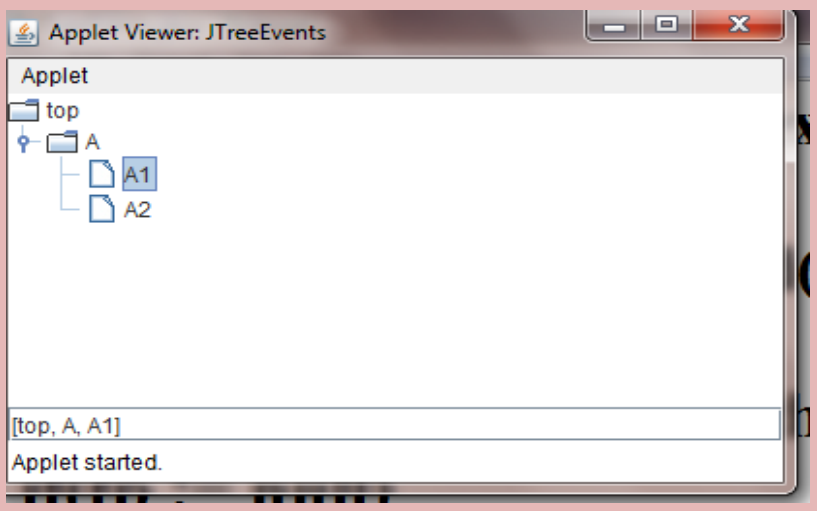
Here are the steps that you should follow to use a tree in an applet
1. Create a **JTree** object.
2. Create a **JScrollPane** object. (The arguments to the constructor specify the tree
and the policies for vertical and horizontal scroll bars.)
3. Add the tree to the scroll pane.
4. Add the scroll pane to the content pane of the applet.

```
import java.awt.*;

import java.awt.event.*;

import javax.swing.*;

import javax.swing.tree.*;

/*

<applet code="JTreeEvents" width=400 height=200>

</applet>

*/

public class JTreeEvents extends JApplet
```

```
{
JTree tree;
JTextField jtf;
public void init()
{
// Get content pane
Container contentPane = getContentPane();
contentPane.setLayout(new BorderLayout());

// Create top node of tree
DefaultMutableTreeNode top = new DefaultMutableTreeNode("top");

// Create subtree of "A"
DefaultMutableTreeNode  A = new DefaultMutableTreeNode("A");
top.add(A);

DefaultMutableTreeNode a1 = new DefaultMutableTreeNode("A1");
A.add(a1);
```

```
DefaultMutableTreeNode a2 = new DefaultMutableTreeNode("A2");
A.add(a2);
// Create tree
tree = new JTree(top);
contentPane.add(tree);
jtf = new JTextField(20);
contentPane.add(jtf, BorderLayout.SOUTH);
tree.addMouseListener(new MyMouseAdapter());
}
class MyMouseAdapter extends MouseAdapter
{
public void mouseClicked(MouseEvent me)
{
TreePath tp = tree.getPathForLocation(me.getX(), me.getY());
if(tp != null)
jtf.setText(tp.toString());
else
jtf.setText("");
```





```
}

}

}
```

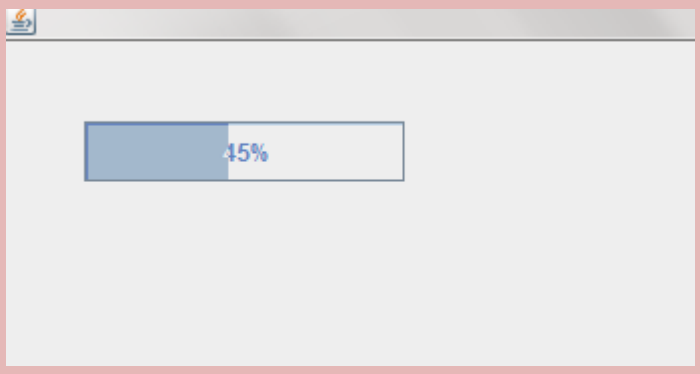
JProgressBar

JProgressBar is a part of Java Swing package**. JProgressBar visually displays the progress of some specified task**. **JProgressBar shows the percentage of completion of specified task.**The progress bar fills up as the task reaches it completion. **In addition to show the percentage of completion of task, it can also display some text** .

```
import javax.swing.*;

class ProgressBarExample extends JFrame

{

JProgressBar jb;

int i=0,num=0;


ProgressBarExample()

{

jb=new JProgressBar(0,2000);
```

```
    jb.setBounds(40,40,160,30);
    jb.setValue(0);
    jb.setStringPainted(true);
    add(jb);
    setSize(250,150);
    setLayout(null);
    }

    public void iterate()
{
    while(i<=2000){
      jb.setValue(i);
      i=i+20;
      try{Thread.sleep(150);}catch(Exception e){}
    }
    }
```

```
 public static void main(String[] args)
{
      ProgressBarExample m=new ProgressBarExample();
      m.setVisible(true);
      m.iterate();
   }
   }
```



Tool tip

You can create a tool tip for any JComponent with **setToolTipText()** method. This method is used to set up a tool tip for the component.

For example, to add tool tip to PasswordField, you need to add only one line of code:

```
import javax.swing.*;

public class ToolTipExample
{
      public static void main(String[] args)
{
       JFrame f=new JFrame("Password Field Example");

       //Creating PasswordField and label
       JPasswordField value = new JPasswordField();
       value.setBounds(100,100,100,30);

       value.setToolTipText("Enter your Password");
       JLabel l1=new JLabel("Password:");
       l1.setBounds(20,100, 80,30);
       //Adding components to frame
       f.add(value);  f.add(l1);
       f.setSize(300,300);
```

```
    f.setLayout(null);
    f.setVisible(true);
}
}
```