UNIT – VIII

Applets – Concepts of Applets, differences between applets and applications, life cycle of an applet, types of applets, creating applets, passing parameters to applets.
Swing – Introduction, limitations of AWT, MVC architecture, components, containers, exploring swing- JApplet, JFrame and JComponent, Icons and Labels, text fields, buttons – The JButton class, Check boxes, Radio buttons, Combo boxes, Tabbed Panes, Scroll Panes, Trees, and Tables.

APPLETS

- ✓ An applet is a program that comes from server into a client and gets executed at client side and displays the result.
- ✓ An applet represents byte code embedded in a html page. (applet = bytecode + html) and run with the help of Java enabled browsers such as Internet Explorer.
- ✓ An applet is a Java program that runs in a browser. Unlike Java applications applets do not have a main () method.
- ✓ To create applet we can use java.applet.Applet or javax.swing.JApplet class. All applets inherit the super class 'Applet'. An Applet class contains several methods that helps to control the execution of an applet.

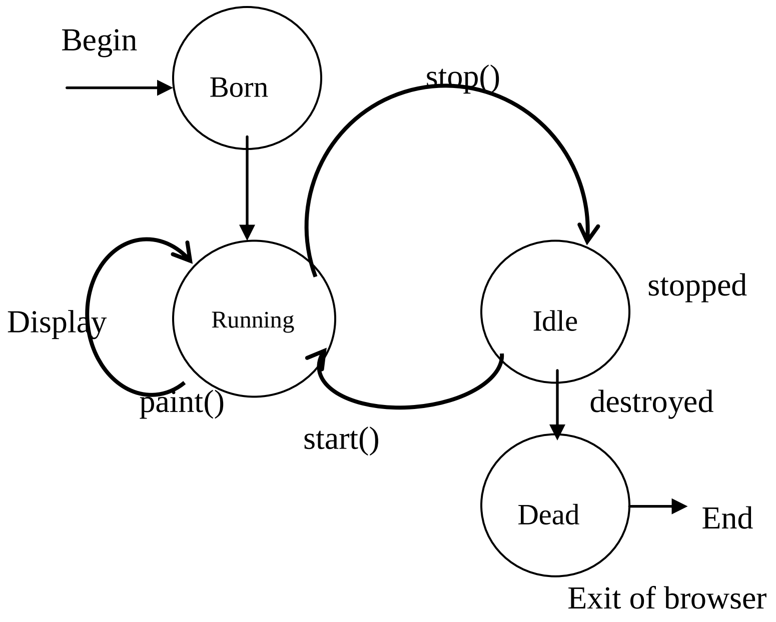
Advantages:
· Applets provide dynamic nature for a webpage.
· Applets are used in developing games and animations.

DEFFERENCES BETWEEN APPLETS AND APPLICATIONS

- ✓ Applets are the small programs while applications are larger programs.
- ✓ Applets don't have the main method while in an application execution starts with the main method.
- ✓ Applets can run in our browser's window or in an appletviewer. To run the applet in an appletviewer will be an advantage for debugging.
- ✓ Applets are designed for the client site programming purpose while the applications don't have such type of criteria.
- ✓ Applet are the powerful tools because it covers half of the java language picture. Java applets are the best way of creating the programs in java. There are a less number of java programmers that have the hands on experience on java applications. This is not the deficiency of java applications but the global utilization of internet. It doesn't mean that the java applications don't have the place. Both (Applets and the java applications) have the same importance at their own places.
- ✓ Applications are also the platform independent as well as byte oriented just like the applets.
- ✓ Applets are designed just for handling the client site problems. while the java applications are designed to work with the client as well as server.

- ✓ Applications are designed to exists in a secure area. while the applets are typically used.
- ✓ Applications and applets have much of the similarity such as both have most of the same features and share the same resources. Applets are created by extending the java.applet.Applet class while the java applications start execution from the main method.
- ✓ Applications are not too small to embed into a html page so that the user can view the application in your browser. On the other hand applet have the accessibility criteria of the resources. The key feature is that while they have so many differences but both can perform the same purpose.

LIFE CYCLE OF AN APPLET



· Let the Applet class extends Applet or JApplet class.

Initialization:
o public void init (): This method is used for initializing variables, parameters to create components. This method is executed only once at the time of applet loaded into memory.

```
public void init()
{
//initialization
}
```

Runnning:
o public void start (): After init() method is executed, the start method is executed automatically. Start method is executed as long as applet gains focus. In this method code

related to opening files and connecting to database and retrieving the data and processing the data is written.

Idle:

o public void stop (): This mehtod is executed when the applet loses focus. Code related to closing the files and database, stopping threads and performing clean up operations are written in this stop method.

Dead/Destroyed:

o public void destroy (): This method is executed only once when the applet is terminated from the memory.

Executing above methods in that sequence is called applet life cycle. We can also use public void paint (Graphics g) in applets.

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
}
}
```

After writing an applet, an applet is compiled in the same way as Java application but running of an applet is different. There are two ways to run an applet.

· Executing an applet within a Java compatible web browser.
· Executing an applet using ‘appletviewer’. This executes the applet in a window.

To execute an applet using web browser, we must write a small HTML file which contains the appropriate ‘APPLET’ tag. <APPLET> tag is useful to embed an applet into an HTML page. It has the following form:

<APPLET CODE=”name of the applet class file” CODEBASE=”path of the applet class file” HEIGHT = maximum height of applet in pixels WIDTH = maximum width of applet in pixels ALIGN = alignment (LEFT, RIGHT, MIDDLE, TOP, BOTTOM)
ALT = alternate text to be displayed>
<PARAM NAME = parameter name VALUE = its value>
</APPLET>

The <PARAM> tag useful to define a variable (parameter) and its value inside the HTML page which can be passed to the applet. The applet can access the parameter value using getParameter () method, as:

String value = getParameter (“pname”);

Where pname is the parameter name and its value is retrieved. The HTML file must be saved with .html extension. After creating this file, open the Java compatible browser (Internet Explorer) and then load this file by specifying the complete path, then Applet program will get executed.

In order to execute applet program with an applet viewer, simply include a comment at the head of Java Source code file that contains the ‘APPLET’ tag.Thus, our code is documented with a prototype of the necessary HTML statements and we can test out compiled applet by starting the appletviewer with the Java file as:

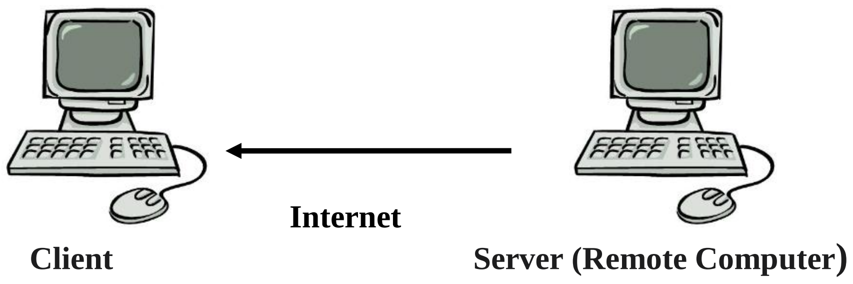
appletviewer programname.java

TYPES OF APPLETS

Applets are of two types:
1. Local Applets
2. Remote Applets

Local Applets: An applet developed locally and stored in a local system is called local applets. So local system does not require internet. We can write our own applets and embed them into the web pages.

Remote Applets: The applet that is downloaded from a remote computer system and embed it into a web page. The internet should be present in the system to download the applet and run it. To download the applet we must know the applet address on web known as Uniform Resource Locator(URL) and must be specified in the applets HTML document as the value of CODEBASE.



CREATING AN APPLET

The APPLET tag is used to start an applet from both an HTML document and from an applet viewer. The syntax for the standard APPLET tag is

< APPLET
[CODEBASE = *codebaseURL*]
CODE = *appletFile*
[ALT = *alternateText*]
[NAME = *appletInstanceName*]
WIDTH = *pixels* HEIGHT = *pixels*
[ALIGN = *alignment*]
[VSPACE = *pixels*] [HSPACE = *pixels*]
>
[< PARAM NAME = *AttributeName* VALUE = *AttributeValue*>]
[< PARAM NAME = *AttributeName2* VALUE = *AttributeValue*>]
. . .
[*HTML Displayed in the absence of Java*]
</APPLET>

Eg1:

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

public class SimpleApplet extends Applet
{
public void paint(Graphics g)
{
g.drawString("A simple applet", 20, 20);
}
}
```

This applet begins with two import statements. The first imports the Abstract Window Tool Kit AWT classes. Applets interact with the user through the AWT, not through the console based IO classes.

The second import the applet package, which contains the applet class. Every applet that we create must be a subclass of an Applet.

The class SimpleApplet must be declared as public, because it will be accessed by code that is outside the program.

The paint() method is declared by the AWT and must be overridden by the applet. It has one parameter of type Graphics.

The drawString() is a method of Graphics class, this method outputs a string beginning at the specified X,Y location.

Compiling and Running Applets:

Compiling in the same way that we have been compiling programs. Running simple applets involves a different process. There are two ways in which we can run an Applet.

Executing the applet within a Java Compatible web browser.
Using an applet viewer such as the standard SDK tool, appletviewerr. An applet viewer excecutes applets in a window.

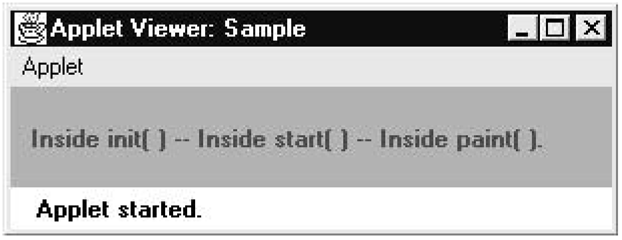
To execute an applet in a web browser, write HTML text file that contains APPLET tag.

```
<applet code = "SimpleApplet.class" width=200 height = 60>
</applet>
```

save as RunApp.html
C:\appletviewer RunApp.html

Eg2:

```
import java.awt.*;
import java.applet.*;
/*
<applet code="Sample" width=300 height=50>
</applet> */
public class Sample extends Applet{
String msg;
// set the foreground and background colors.
public void init() {
setBackground(Color.cyan);
setForeground(Color.red);
msg = "Inside init( ) --";
}
// Initialize the string to be displayed.
public void start() {
msg += " Inside start( ) --";
}
// Display msg in applet window.
public void paint(Graphics g) {
msg += " Inside paint( ).";
g.drawString(msg, 10, 30);
}
}
```

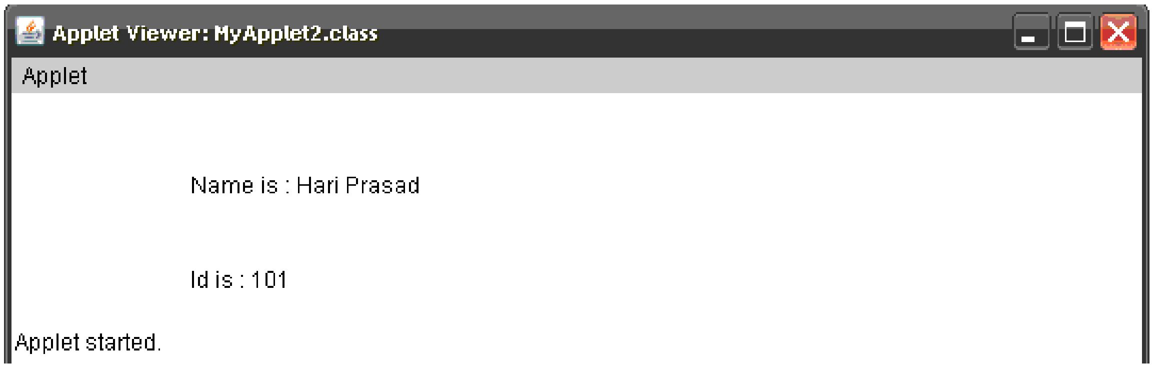


```
Inside init( ) -- Inside start( ) -- Inside paint( ).
Applet started.
```

PASSING PARAMETERS TO AN APPLET

Write a program to pass employ name and id number to an applet.

```
/* <applet code="MyApplet2.class" width = 600 height= 450>
<param name = "t1" value="Hari Prasad">
<param name = "t2" value ="101">
</applet> */
import java.applet.*;
import java.awt.*;
public class MyApplet2 extends Applet
{ String n;
String id;
public void init()
{ n = getParameter("t1");
id = getParameter("t2");
}
public void paint(Graphics g)
{ g.drawString("Name is : " + n, 100,100);
g.drawString("Id is : "+ id, 100,150);
}
}
```



```
Name is : Hari Prasad

Id is : 101
```

Applet started.

SWINGS

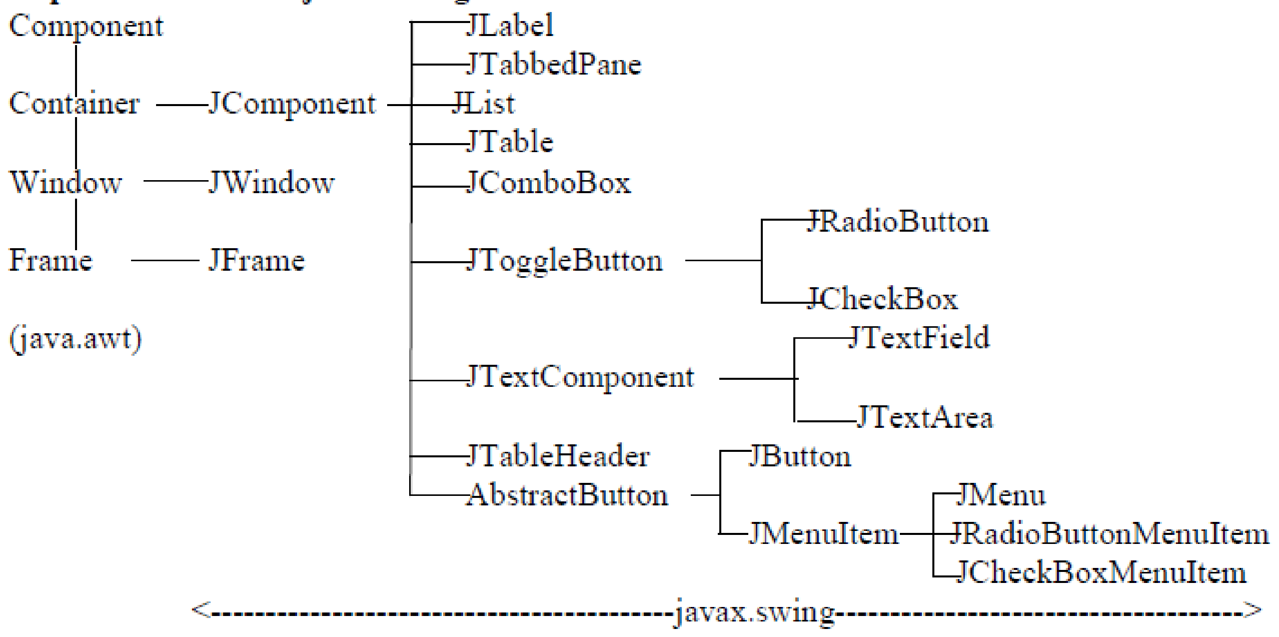
- ✓ Swing is a set of classes that provides more powerful and flexible components than are possible with the AWT.
- ✓ In addition to the familiar components, such as buttons, check boxes, and labels, Swing supplies several exciting additions, including tabbed panes, scroll panes, trees, and tables. Even 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 component classes are:

Class	**Description**
AbstractButton	Abstract superclass for Swing buttons.
ButtonGroup	Encapsulates a mutually exclusive set of buttons.
ImageIcon	Encapsulates an icon.
JApplet	The Swing version of **Applet**.
JButton	The Swing push button class.
JCheckBox	The Swing check box class.
JComboBox	Encapsulates a combo box (an combination of a

	drop-down list and text field).
JLabel	The Swing version of a label.
JRadioButton	The Swing version of a radio button.
JScrollPane	Encapsulates a scrollable window.
JTabbedPane	Encapsulates a tabbed window.
JTable	Encapsulates a table-based control.
JTextField	The Swing version of a text field.
JTree	Encapsulates a tree-based control.

Important classes of javax.swing:



JApplet

Fundamental to Swing is the **JApplet** class, which extends **Applet**. Applets that use Swing must be subclasses of **JApplet**. **JApplet** is rich with functionality that is not found in **Applet**.

The content pane can be obtained via the method shown here:
Container getContentPane( )

The **add( )** method of **Container** can be used to add a component to a content pane. Its form is shown here:

void add(*comp*)

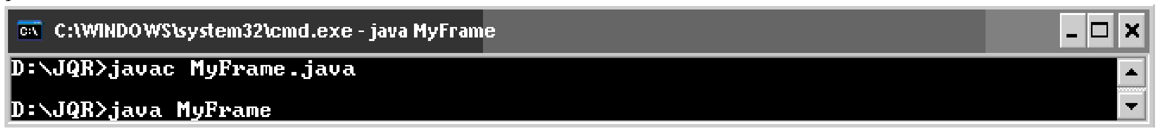
Here, *comp* is the component to be added to the content pane.

JFrame

Create an object to JFrame: JFrame ob = new JFrame ("title");
(or)
Create a class as subclass to JFrame class: MyFrame extends JFrame
Create an object to that class : MyFrame ob = new MyFrame ();

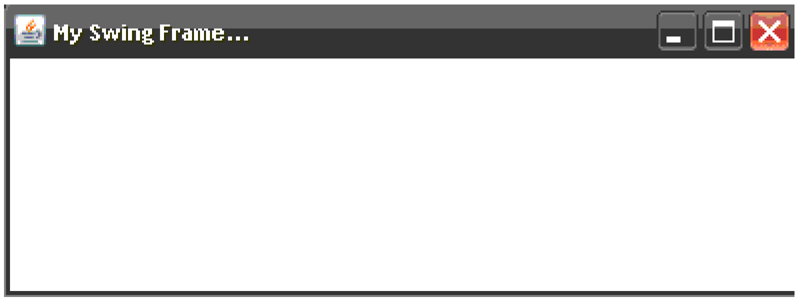
Program 1: Write a program to create a frame by creating an object to JFrame class.

```
//A swing Frame
import javax.swing.*;
class MyFrame
{
public static void main (String agrs[])
{ JFrame jf = new JFrame ("My Swing Frame...");
jf.setSize (400,200);
jf.setVisible (true);
jf.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
}
}
```



```
D:\JQR>javac MyFrame.java

D:\JQR>java MyFrame
```

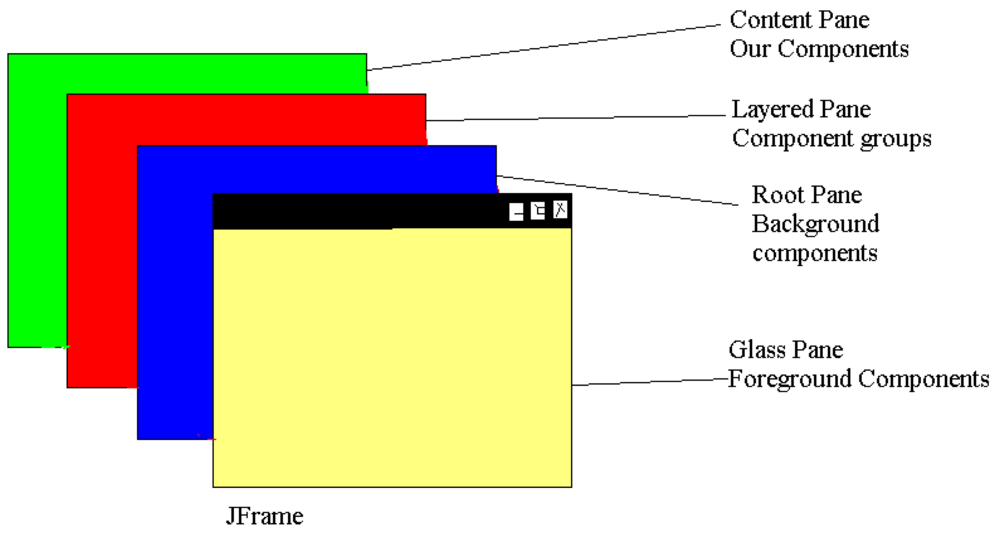


Note: To close the frame, we can take the help of getDefaultCloseOperation () method of JFrame class, as shown here:

getDefaultCloseOperation (constant);

where the constant can be any one of the following:
· JFrame.EXIT_ON_CLOSE: This closes the application upon clicking on close button.
· JFrame.DISPOSE_ON_CLOSE: This disposes the present frame which is visible on the screen. The JVM may also terminate.
· JFrame.DO_NOTHING_ON_CLOSE: This will not perform any operation upon clicking on close button.
· JFrame.HIDE_ON_CLOSE: This hides the frame upon clicking on close button.

Window Panes: In swings the components are attached to the window panes only. A window pane represents a free area of a window where some text or components can be displayed. For example, we can create a frame using JFrame class in javax.swing which contains a free area inside it, this free area is called 'window pane'. Four types of window panes are available in javax.swing package.

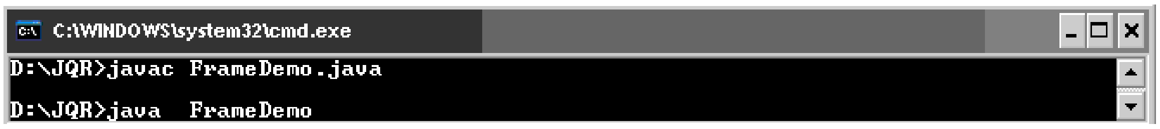


· **Glass Pane:** This is the first pane and is very close to the monitors screen. Any components to be displayed in the foreground are attached to this glass pane. To reach this glass pane, we use getGlassPane () method of JFrame class.
· **Root Pane:** This pane is below the glass pane. Any components to be displayed in the background are displayed in this pane. Root pane and glass pane are used in animations also.
For example, suppose we want to display a flying aeroplane in the sky. The aeroplane can be displayed as a .gif or .jpg file in the glass pane where as the blue sky can be displayed in the root pane in the background. To reach this root pane, we use getRootPane () method of JFrame class.
· **Layered Pane:** This pane lies below the root pane. When we want to take several components as a group, we attach them in the layered pane. We can reach this pane by calling getLayeredPane () method of JFrame class.
· **Content Pane:** This is the bottom most pane of all. Individual components are attached to this pane. To reach this pane, we can call getContentPane () method of JFrame class.
Displaying Text in the Frame: paintComponent (Graphics g) method of JPanel class is used to paint the portion of a component in swing. We should override this method in our class. In the following example, we are writing our class MyPanel as a subclass to JPanel and override the painComponent () method.

Wrire a program to display text in the frame.

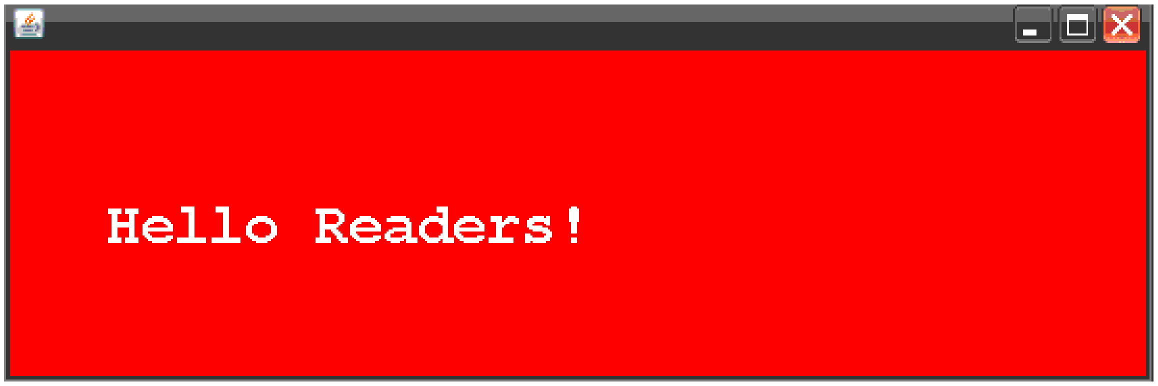
```
import javax.swing.*;
import java.awt.*;
class MyPanel extends JPanel
```

```
{ public void paintComponent (Graphics g)
{ super.paintComponent (g); //call JPanel's method
setBackground (Color.red);
g.setColor (Color.white);
g.setFont (new Font("Courier New",Font.BOLD,30));
g.drawString ("Hello Readers!", 50, 100);
}
}
class FrameDemo extends JFrame
{ FrameDemo ()
{
Container c = getContentPane ();
MyPanel mp = new MyPanel ();
c.add (mp);
setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
}
public static void main(String args[])
{ FrameDemo ob = new FrameDemo ();
ob.setSize (600, 200);
ob.setVisible (true);
}
}
```



```
D:\JQR>javac FrameDemo.java

D:\JQR>java  FrameDemo
```



JComponent

JComponent class of javax.swing package is the subclass of the Component class of java.awt. So, whatever the methods are available in Component class are also available to JComponent. This is the reason why almost all the methods of AWT are useful in swing also. Additional methods found in JComponent are also applicable for the components created in swing.

When a component is created, to dislay it in the frame, we should not attach it to the frame directly as we did in AWT. On the other hand, the component should be attached to a window pane.
· To add the component to the content pane, we can write, as:
c.add (component); where c represents the content pane which is represented by Container object.

· Similarly, to remove the component we can use remove () method as:
c.remove (component);

· To remove all the components from the content pane we can use removeAll () method as: c.removeAll ();

· When components are to be displayed in the frame, we should first set a layout which arranges the components in a particular manner in the frame as:
c.setLayout (new FlowLayout ());
The following methods of JComponent class are very useful while handling the components:

· To set some background color to the component, we can use setBackground () method, as: component.setBackground (Color.yellow);

· To set the foreground color to the component, we can use setForeground () method, as:
component.setForeground (Color.red);

· To set some font for the text displayed on the component, we can use setFont () method. We should pass Font class object to this method, as: component.setFont (Font obj);
where, Font class object can be created as:
Font obj = new Font (“fontname”, style, size);

· To set the tool tip text, we can use setToolTipText () method, as:
component.setToolTipText (“This is for help”);

· To set a mnemonic, we can use setMnemonic () method, as:
component.setMnemonic (‘c’);

· To enable or disable a component, we can use setEnabled () method, as:
component.setEnabled (true);

· To make a component to be visible or invisible, we can use setVisible () method, as:
component.setVisible (true);

· To know the current height of the component, use getHeight () method, as:
component.getHeight ();

· To know the current width of the component, use getWidth () method, as:

component.getHeight ();

· To know the current x coordinate of the component
component.getX ()
· To know the current y coordinate of the component
component.getY ()

· To set the location of the component in the frame, we can use setBounds () method, as:
component.setBounds (x, y, width, height);

ICONS AND LABELS

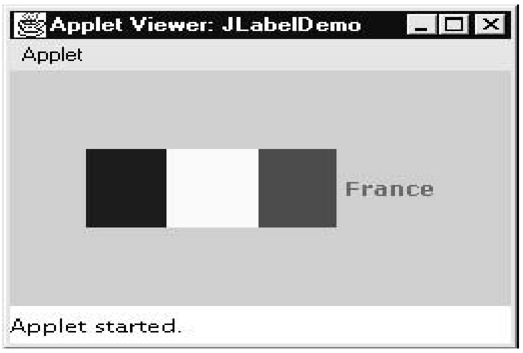
In Swing, icons are encapsulated by the **ImageIcon** class, which paints an icon from an image. Two of its constructors are shown here:

ImageIcon(String *filename*)
ImageIcon(URL *url*)

Swing labels are instances of the **JLabel** class, which extends **JComponent**. It can display text and/or an icon. Some of its constructors are shown here:

JLabel(Icon *i*)
Label(String *s*)
JLabel(String *s*, Icon *i*, int *align*)

```
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();
// Create an icon
ImageIcon ii = new ImageIcon("france.gif");
// Create a label
JLabel jl = new JLabel("France", ii, JLabel.CENTER);
// Add label to the content pane
contentPane.add(jl);
}
}
```

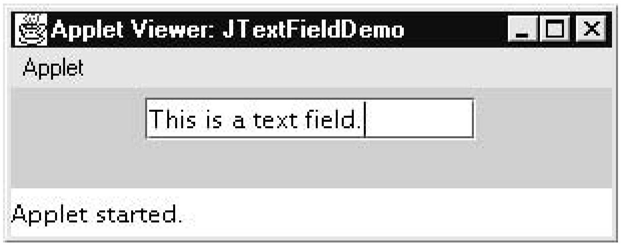


TEXT FIELDS

The Swing text field is encapsulated by the JTextComponent class, which extends JComponent. It provides functionality that is common to Swing text components. One of its subclasses is **JTextField**, which allows you to edit one line of text. Some of its constructors are shown here:

JTextField( )
JTextField(int *cols*)
JTextField(String *s*, int *cols*)
JTextField(String *s*)

```
import java.awt.*;
import javax.swing.*;
/*
<applet code="JTextFieldDemo" width=300 height=50>
</applet>
*/
public class JTextFieldDemo extends JApplet {
JTextField jtf;
public void init() {
// Get content pane
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());
// Add text field to content pane
jtf = new JTextField(15);
contentPane.add(jtf);
}
}
```



BUTTONS

Swing buttons provide features that are not found in the **Button** class defined by the AWT. For example, you can associate an icon with a Swing button. Swing buttons are subclasses of the **AbstractButton** class, which extends **JComponent**. **AbstractButton** contains many methods that allow you to control the behavior of buttons, check boxes, and radio buttons.

The JButton Class

The **JButton** class provides the functionality of a push button. **JButton** allows an icon, a string, or both to be associated with the push button. Some of its constructors are shown here:

· To create a JButton with text: JButton b = new JButton (“OK”);
· To create a JButton with image: JButton b = new JButton (ImageIcon ii);
· To create a JButton with text & image: JButton b = new JButton (“OK”, ImageIcon ii);
It is possible to create components in swing with images on it. The image is specified by ImageIcon class object.

```
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 jtf;
public void init() {
// Get content pane
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());
// Add buttons to content pane
ImageIcon france = new ImageIcon("france.gif");
JButton jb = new JButton(france);
jb.setActionCommand("France");
jb.addActionListener(this);
contentPane.add(jb);
ImageIcon germany = new ImageIcon("germany.gif");
jb = new JButton(germany);
jb.setActionCommand("Germany");
jb.addActionListener(this);
contentPane.add(jb);
ImageIcon italy = new ImageIcon("italy.gif");
jb = new JButton(italy);
jb.setActionCommand("Italy");
jb.addActionListener(this);
contentPane.add(jb);
ImageIcon japan = new ImageIcon("japan.gif");
jb = new JButton(japan);
jb.setActionCommand("Japan");
jb.addActionListener(this);
contentPane.add(jb);
// Add text field to content pane
jtf = new JTextField(15);
contentPane.add(jtf);
}
public void actionPerformed(ActionEvent ae) {
jtf.setText(ae.getActionCommand());
}
}
```

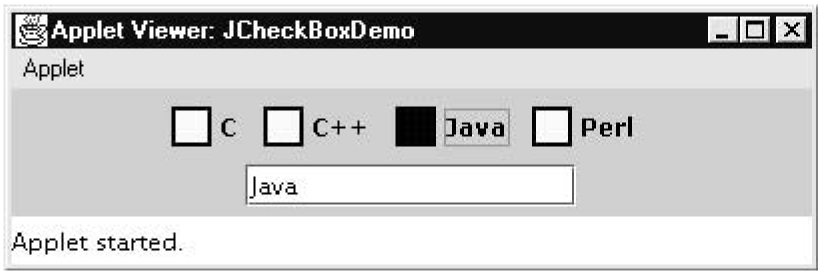
CHECK BOXES

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. Some of its constructors are shown here:

```
JCheckBox(Icon i)
JCheckBox(Icon i, boolean state)
JCheckBox(String s)
JCheckBox(String s, boolean state)
JCheckBox(String s, Icon i)
JCheckBox(String s, Icon i, boolean state)
```

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="JCheckBoxDemo" width=400 height=50>
```

```
</applet>
*/
public class JCheckBoxDemo extends JApplet
implements ItemListener {
JTextField jtf;
public void init() {
// Get content pane
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());
// Create icons
ImageIcon normal = new ImageIcon("normal.gif");
ImageIcon rollover = new ImageIcon("rollover.gif");
ImageIcon selected = new ImageIcon("selected.gif");
// Add check boxes to the content pane
JCheckBox cb = new JCheckBox("C", normal);
cb.setRolloverIcon(rollover);
cb.setSelectedIcon(selected);
cb.addItemListener(this);
contentPane.add(cb);
cb = new JCheckBox("C++", normal);
cb.setRolloverIcon(rollover);
cb.setSelectedIcon(selected);
cb.addItemListener(this);
contentPane.add(cb);
cb = new JCheckBox("Java", normal);
cb.setRolloverIcon(rollover);
cb.setSelectedIcon(selected);
cb.addItemListener(this);
contentPane.add(cb);
cb = new JCheckBox("Perl", normal);
cb.setRolloverIcon(rollover);
cb.setSelectedIcon(selected);
cb.addItemListener(this);
contentPane.add(cb);
// Add text field to the content pane
jtf = new JTextField(15);
contentPane.add(jtf);
}
public void itemStateChanged(ItemEvent ie) {
JCheckBox cb = (JCheckBox)ie.getItem();
jtf.setText(cb.getText());
}
}
```



```
Java
```

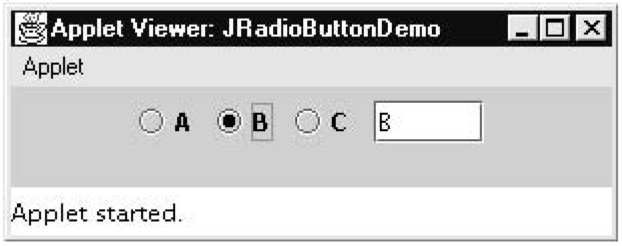
Applet started.

RADIO BUTTONS

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. Some of its constructors are shown here:

JRadioButton(Icon *i*)
JRadioButton(Icon *i*, boolean *state*)
JRadioButton(String *s*)
JRadioButton(String *s*, boolean *state*)
JRadioButton(String *s*, Icon *i*)
JRadioButton(String *s*, Icon *i*, boolean *state*)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="JRadioButtonDemo" width=300 height=50>
</applet>
*/
public class JRadioButtonDemo extends JApplet
implements ActionListener {
JTextField tf;
public void init() {
// Get content pane
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());
// Add radio buttons to content pane
JRadioButton b1 = new JRadioButton("A");
b1.addActionListener(this);
contentPane.add(b1);
JRadioButton b2 = new JRadioButton("B");
b2.addActionListener(this);
contentPane.add(b2);
JRadioButton b3 = new JRadioButton("C");
b3.addActionListener(this);
contentPane.add(b3);
// Define a button group
ButtonGroup bg = new ButtonGroup();
bg.add(b1);
bg.add(b2);
bg.add(b3);
// Create a text field and add it
// to the content pane
tf = new JTextField(5);
contentPane.add(tf);
}
public void actionPerformed(ActionEvent ae) {
tf.setText(ae.getActionCommand());
}
}
```



```
Applet started.
```

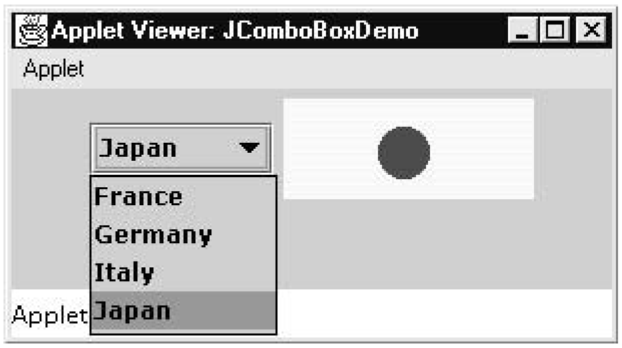
COMBO BOXES

Swing provides a combo box (a combination of a text field and a drop-down list) through the JComboBox class, which extends JComponent. A combo box normally displays one entry. However, it can also display a drop-down list that allows a user to select a different entry. You can also type your selection into the text field. Two of JComboBox's constructors are shown here:

```
JComboBox( )
JComboBox(Vector v)
```

```
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 france, germany, italy, japan;
public void init() {
// Get content pane
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());
// Create a combo box and add it
// to the panel
JComboBox jc = new JComboBox();
jc.addItem("France");
jc.addItem("Germany");
jc.addItem("Italy");
jc.addItem("Japan");
jc.addItemListener(this);
contentPane.add(jc);
// Create label
jl = new JLabel(new ImageIcon("france.gif"));
contentPane.add(jl);
}
public void itemStateChanged(ItemEvent ie) {
String s = (String)ie.getItem();
jl.setIcon(new ImageIcon(s + ".gif"));
}
}
```



TABBED PANES

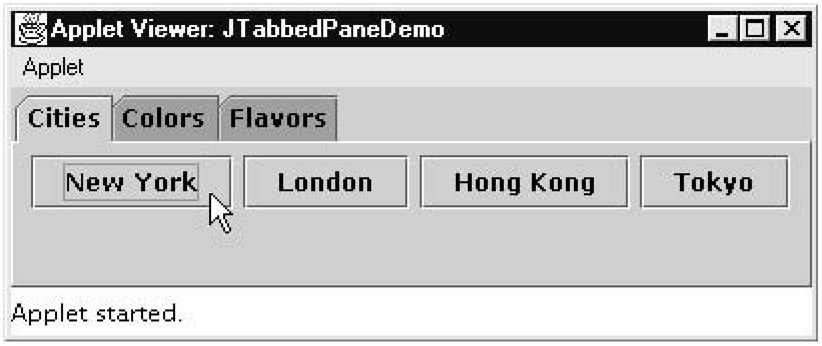
A *tabbed pane* is a component that appears as a group of folders in a file cabinet. Each folder has a title. When a user selects a folder, its contents become visible. Only one of the folders may be selected at a time. Tabbed panes are commonly used for setting configuration options.
Tabbed panes are 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*)

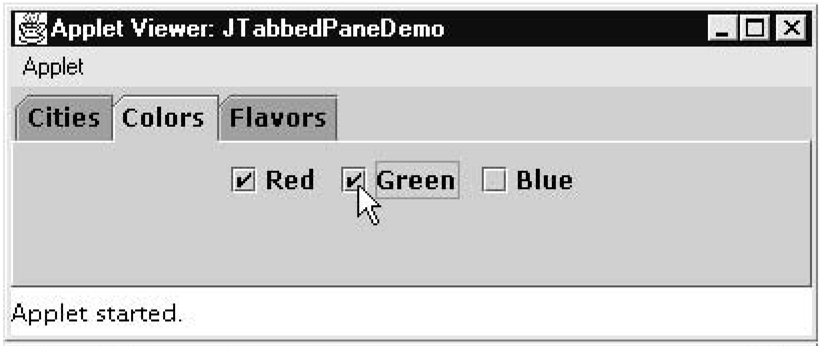
Here, *str* is the title for the tab, and *comp* is the component that should be added to the tab. Typically, a **JPanel** or a subclass of it is added.

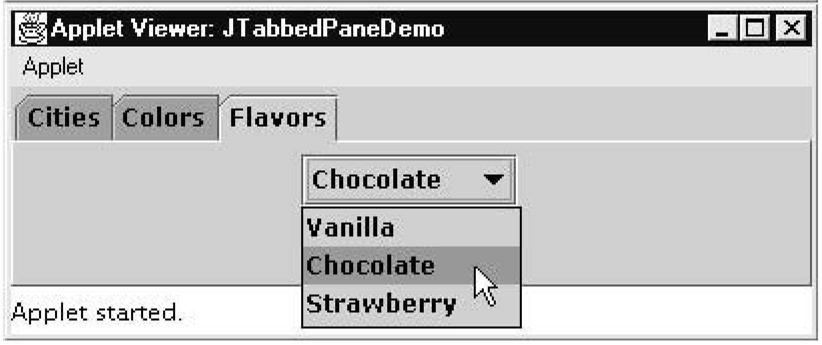
The general procedure to use a tabbed pane in an applet is outlined here:

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());
jtp.addTab("Flavors", new FlavorsPanel());
getContentPane().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");
add(cb1);
JCheckBox cb2 = new JCheckBox("Green");
add(cb2);
JCheckBox cb3 = new JCheckBox("Blue");
add(cb3);
}
}
class FlavorsPanel extends JPanel {
public FlavorsPanel() {
JComboBox jcb = new JComboBox();
jcb.addItem("Vanilla");
jcb.addItem("Chocolate");
jcb.addItem("Strawberry");
add(jcb);
}
}
```







SCROLL PANES

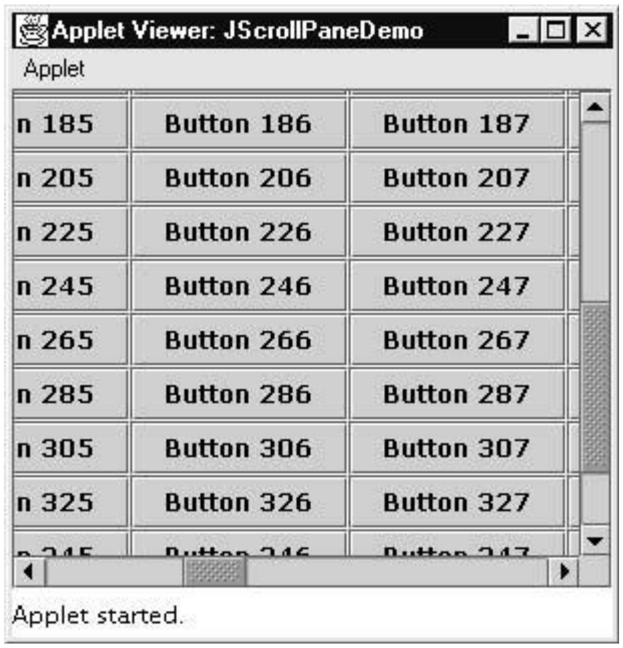
A *scroll pane* is a component that presents a rectangular area in which a component may be viewed. Horizontal and/or vertical scroll bars may be provided if necessary. Scroll panes are implemented in Swing by the **JScrollPane** class, which extends **JComponent**. Some of its constructors are shown here:

JScrollPane(Component *comp*)
JScrollPane(int *vsb*, int *hsb*)
JScrollPane(Component *comp*, int *vsb*, int *hsb*)

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

1. Create a **JComponent** object.
2. Create a **JScrollPane** object. (The arguments to the constructor specify the component and the policies 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();
contentPane.setLayout(new BorderLayout());
// Add 400 buttons to a panel
JPanel jp = new JPanel();
jp.setLayout(new GridLayout(20, 20));
int b = 0;
for(int i = 0; i < 20; i++) {
for(int j = 0; j < 20; 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_AS_NEEDED;
JScrollPane jsp = new JScrollPane(jp, v, h);
// Add scroll pane to the content pane
contentPane.add(jsp, BorderLayout.CENTER);
}
}
```



TREES

A *tree* is a component that presents a hierarchical view of data. A user has the ability to expand or collapse individual subtrees in this display. Trees are implemented in Swing by the **JTree** class, which extends **JComponent**. Some of its constructors are shown here:

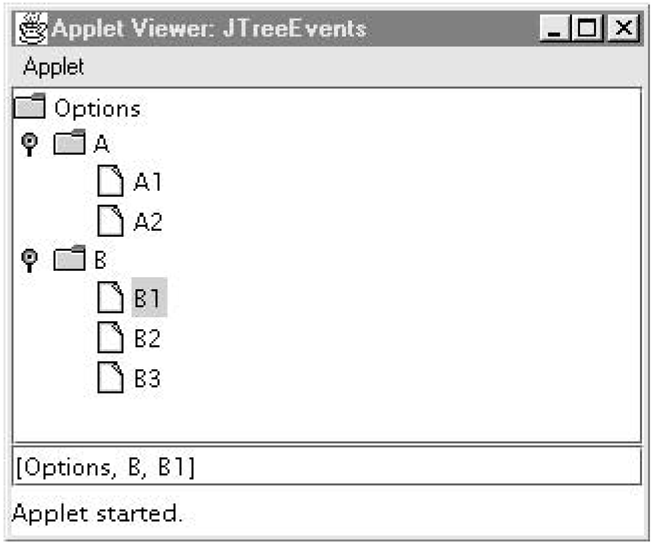
JTree(Hashtable *ht*)
JTree(Object *obj*[ ])
JTree(TreeNode *tn*)
JTree(Vector *v*)

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();
// Set layout manager
contentPane.setLayout(new BorderLayout());
// Create top node of tree
DefaultMutableTreeNode top = new DefaultMutableTreeNode("Options");
// 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 subtree of "B"
DefaultMutableTreeNode b = new DefaultMutableTreeNode("B");
top.add(b);
DefaultMutableTreeNode b1 = new DefaultMutableTreeNode("B1");
b.add(b1);
DefaultMutableTreeNode b2 = new DefaultMutableTreeNode("B2");
b.add(b2);
DefaultMutableTreeNode b3 = new DefaultMutableTreeNode("B3");
b.add(b3);
// Create tree
tree = new JTree(top);
// Add tree to a scroll pane
int v = ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
int h = ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;
JScrollPane jsp = new JScrollPane(tree, v, h);
// Add scroll pane to the content pane
contentPane.add(jsp, BorderLayout.CENTER);
// Add text field to applet
jtf = new JTextField("", 20);
contentPane.add(jtf, BorderLayout.SOUTH);
// Anonymous inner class to handle mouse clicks
tree.addMouseListener(new MouseAdapter() {
public void mouseClicked(MouseEvent me) {
doMouseClicked(me);
}
});
}
void doMouseClicked(MouseEvent me) {
TreePath tp = tree.getPathForLocation(me.getX(), me.getY());
if(tp != null)
jtf.setText(tp.toString());
else
jtf.setText("");
}
}
```



```
[Options, B, B1]
```

Applet started.

TABLES

A *table* is a component that displays rows and columns of data. You can drag the cursor on column boundaries to resize columns. You can also drag a column to a new position. Tables are implemented by the **JTable** class, which extends **JComponent**. One of its constructors is shown here:

JTable(Object *data*[ ][ ], Object *colHeads*[ ])

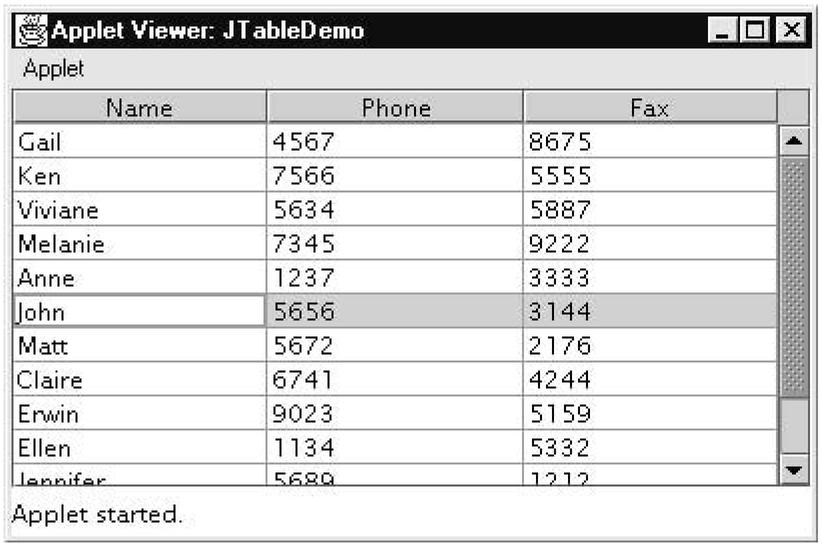
Here, *data* is a two-dimensional array of the information to be presented, and *colHeads* is a one-dimensional array with the column headings.

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 policies 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();
// Set layout manager
contentPane.setLayout(new BorderLayout());
// Initialize column headings
final String[] colHeads = { "Name", "Phone", "Fax" };
// Initialize data
final Object[][] data = {
{ "Gail", "4567", "8675" },
{ "Ken", "7566", "5555" },
{ "Viviane", "5634", "5887" },
{ "Melanie", "7345", "9222" },
{ "Anne", "1237", "3333" },
{ "John", "5656", "3144" },
{ "Matt", "5672", "2176" },
{ "Claire", "6741", "4244" },
{ "Erwin", "9023", "5159" },
{ "Ellen", "1134", "5332" },
{ "Jennifer", "5689", "1212" },
{ "Ed", "9030", "1313" },
{ "Helen", "6751", "1415" }
};
// Create the table
JTable table = new JTable(data, colHeads);
// Add table to a scroll pane
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.CENTER);
}
}
```



Name	Phone	Fax
Gail	4567	8675
Ken	7566	5555
Viviane	5634	5887
Melanie	7345	9222
Anne	1237	3333
John	5656	3144
Matt	5672	2176
Claire	6741	4244
Erwin	9023	5159
Ellen	1134	5332
Jennifer	5689	1212

Applet started.