MODULE 5: APPLETS & SWINGS

Syllabus:

The Applet Class: Two types of Applets; Applet basics; Applet Architecture; An Applet skeleton; Simple Applet display methods; Requesting repainting; Using the Status Window; The HTML APPLET tag; Passing parameters to Applets; getDocumentbase() and getCodebase(); ApletContext and showDocument(); The AudioClip Interface; The AppletStub Interface; Output o the Console.

Beautiful thought: *"You have to grow from the inside out. None can teach you, none can make you spiritual. There is no other teacher but your own soul."* — *Swami Vivekananda*

APPLET

The Applet Introduction:

- ✓ Applets are small Java program/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 program written in the Java programming language that can be included in an HTML page, much in the same way an image is included in a page. When you use a Java technology enabled browser to view a page that contains an applet, the applet's code is transferred to your system and executed by the browser's Java Virtual Machine (JVM)

Two Types of Applets

There are two varieties of applets. They are

1. Based on the Applet class: **Applet**
2. Based on the Swing Class Applet: **JApplet**

1. Based on the Applet class.

- ➢ These Applet uses the Abstract Window Toolkit(AWT) to provide the graphical user interface.
- ➢ This type of applet has been widely available since java was first created.

2. Based on the Swing Class Applet.

- ➢ This applet uses the swing class to provide GUI.
- ➢ Swing offers a rich and easier to use interface than AWT.

- Swing based applets are the most popular in practice.

Applet Basics

- ✓ The reason people are excited about Java as more than just another OOP language is because it allows them to write interactive applets on the web. Hello World isn't a very interactive program, but let's look at a webbed version.

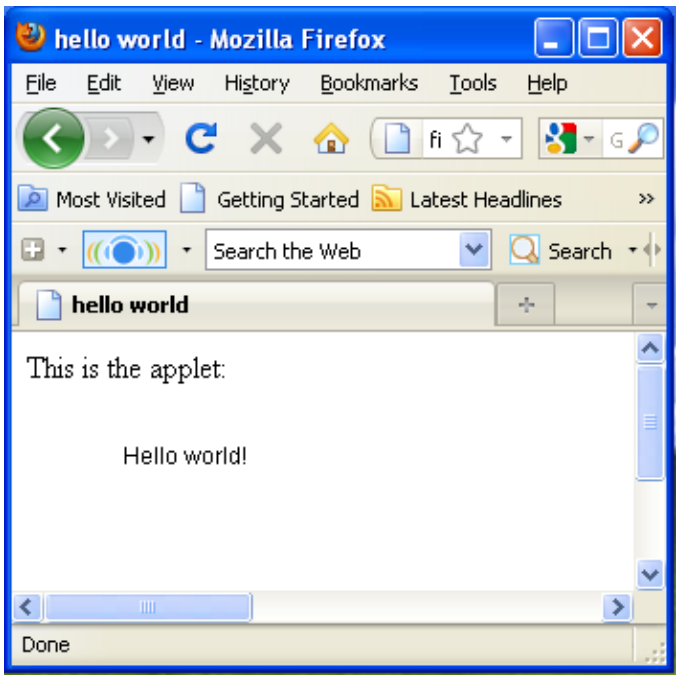
```
import java.applet.Applet;
import java.awt.Graphics;


public class HelloWorldApplet extends Applet
{
     public void paint(Graphics g)
     {
          g.drawString("Hello world!", 50, 25);
     }
 }
```

OUTPUT

- ✓ The applet version of HelloWorld is a little more complicated than the HelloWorld application, and it will take a little more effort to run it as well.
- ✓ First type in the source code and save it into file called HelloWorldApplet.java. Compile this file in the usual way. If all is well a file called HelloWorldApplet.class will be created. Now you need to create an HTML file that will include your applet. The following simple HTML file will do.

```
<html>
<head>
<title> hello world </title>
</head>

<body>
This is the applet:<P>
<applet code="HelloWorldApplet" width="150" height="50">
</applet>
</body>
</html>
```

OUTPUT



- ✓ Save this file as HelloWorldApplet.html in the same directory as the HelloWorldApplet.class file. When you have done that, load the HTML file into a Java enabled browser and see the output in the browser window.
- ✓ If the applet compiled without error and produced a HelloWorldApplet.class file, and yet you don't see the string "Hello World" in your browser chances are that the .class file is in the wrong place. Make sure HelloWorldApplet.class is in the same directory as HelloWorldApplet.html. Also make sure that your browsers support Java or that the Java plugin has been installed. Not all browsers support Java out of the box.

The Applet Class

- ✓ An applet is a small program that is intended not to be run on its own, but rather to be embedded inside another application.
- ✓ The **Applet class** must be the super class of any applet that is to be embedded in a Web page or viewed by the Java Applet Viewer. The Applet class provides a standard interface between applets and their environment.

Method	Summary
Void **destroy**()	Called by the browser or applet viewer to inform this applet that it is being reclaimed and that it should destroy any resources that it has allocated.
AccessibleContext **getAccessibleContext**()	Gets the AccessibleContext associated with this Applet.
AppletContext **getAppletContext**()	Determines this applet's context, which allows the applet to query and affect the environment in which it runs.
String **getAppletInfo**()	Returns information about this applet.
AudioClip **getAudioClip**(URL url)	Returns the AudioClip object specified by the URL argument.
AudioClip **getAudioClip**(URL url,	Returns the AudioClip object specified by the URL and name arguments.

String name)	
URL **getCodeBase**()	Gets the base URL.
URL **getDocumentBase**()	Returns an absolute URL naming the directory of the document in which the applet is embedded.
Image **getImage**(URL url)	Returns an Image object that can then be painted on the screen.
Image **getImage**(URL url, String name)	Returns an Image object that can then be painted on the screen.
Locale **getLocale**()	Gets the Locale for the applet, if it has been set.
String **getParameter**(String name)	Returns the value of the named parameter in the HTML tag.
String[][] **getParameterInfo**()	Returns information about the parameters than are understood by this applet.
Void **init**()	Called by the browser or applet viewer to inform this applet that it has been loaded into the system.
Boolean **isActive**()	Determines if this applet is active.
static AudioClip **newAudioClip**(URL url)	Get an audio clip from the given URL.
Void **play**(URL url)	Plays the audio clip at the specified absolute URL.
Void **play**(URL url, String name)	Plays the audio clip given the URL and a specifier that is relative to it.
Void **resize**(Dimension d)	Requests that this applet be resized.
Void **resize**(int width, int height)	Requests that this applet be resized.
Void **setStub**(AppletStub stub)	Sets this applet's stub.
Void	Requests that the argument string be displayed in the

showStatus(String msg)	"status window".
void **start**()	Called by the browser or applet viewer to inform this applet that it should start its execution.
void **stop**()	Called by the browser or applet viewer to inform this applet that it should stop its execution.

Applet Architecture

✓ An applet is a window-based program, its architecture different from the console-based programs. There are two key concepts to understand the architecture they are

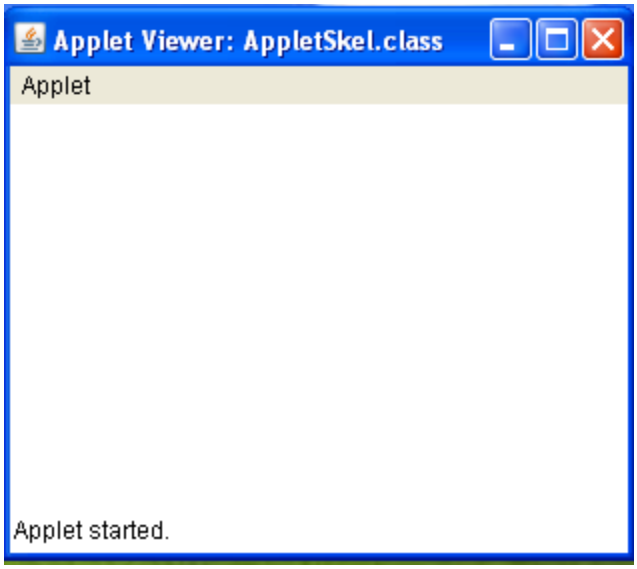
1. **Applets are Event driven**
   - An applet waits until an event occurs.
   - The *AWT* notifies the applet about an event by calling event handler that has been provided by the applet.The applet takes appropriate action and then quickly return control to *AWT*
   - All *Swing* components descend from the AWT Container class

2. **User initiates interaction with an Applet (*and not the other way around*)**

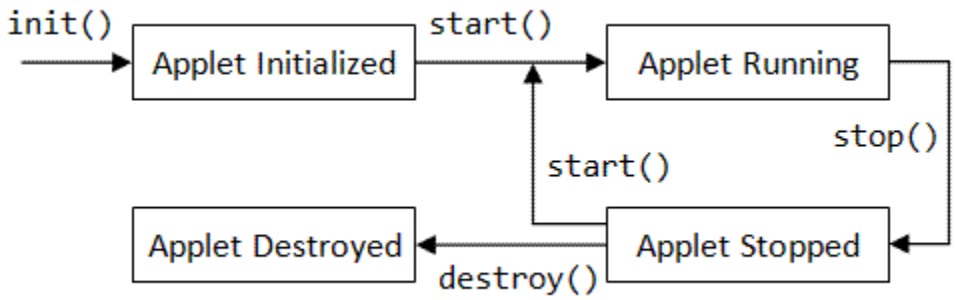
An Applet Skelton

```
// An Applet skeleton.
import java.awt.*;
import javax.swing.*;
/*
<applet code="AppletSkel" width=300 height=100>
</applet>
*/

public class AppletSkel extends JApplet
{
    // 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
    }
}
```

OUTPUT



Applet Initialization and Termination/Applet Life Cycle



It is important to understand the order in which the various methods shown in the skeleton are called. **When an applet begins**, the AWT calls the following **initialization methods**, in this sequence:

1. **init( )**
2. **start( )**
3. **paint( )**

When an applet is terminated, the following sequence of method calls takes place:

1. **stop( )**
2. **destroy( )**

1. init( )

The **init( )** method is the first method to be called. This is where you should initialize variables. This method is called only once during the run time of your applet.

2. start( )

The **start( )** method is called after **init( )**. It is also called to restart an applet after it has been stopped. Whereas **init( )** is called once—the first time an applet is loaded **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. paint( )

The **paint( )** method is called each time your 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. This context is used whenever output to the applet is required.

4. stop( )

The **stop( )** method is called when a web browser leaves the HTML document containing the applet when it goes to another page, for example. When **stop( )** is called, the applet is probably running. You should use **stop( )** to suspend threads

that don't need to run when the applet is not visible. You can restart them when **start( )** is called if the user returns to the page.

5. destroy( )

The **destroy( )** method is called when the environment determines that your applet needs to be removed completely from memory. At this point, you should free up any resources the applet may be using. The **stop( )** method is always called before **destroy()**.

Simple Applet display methods

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

public class HelloWorldApplet extends Applet
{
        public void paint(Graphics  g)
        {
                g.drawString("Hello world!", 50, 25);
        }
}
```

- ✓ Consider the above program to output a string to an applet, use drawString() this is a member of the Graphics class, this drawstring is called from within either update() or paint() as shown in the above program example .The general form of is

  drawString(String msg,int x, int y)

- ✓ The **msg** indicates that string to be output beginning **at x,y**. in java window the upper-left corner location is 0,0.the drawstring() method will not recognize newline character.

- ✓ To set the background color of an applet window use **setBackground**() and to set the foreground color for example the color in which text is shown use **setForeground**().these methods are defined by Component and they have the following general forms

void setBackground(Color newColor) ,

void setForeground(Color newColor)

The **newColor** specifies that new color. The class Color defines the constant shown below that can be used to specify colors.

```
Color.black     Color.lightGray   Color.yellow
Color.blue      Color.magenta     Color.red
Color.cyan      Color.orange      Color.white
Color.darkGray  Color.pink        Color.gray     Color.green
```

Example:

setBackround(Color.cyan);

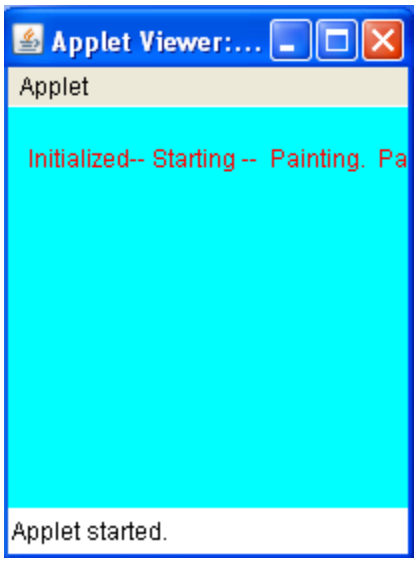
setForeground(Color.red)

Example Program:

```
/* This Applet sets the foreground and background colors and out puts a string. */
import java.applet.*;
import java.awt.*;

public class Simple extends Applet
{
        String msg;

  // set the foreground and background colors.
        public void init()
        {
           setBackground(Color.cyan);
           setForeground(Color.red);
            msg = "Initialized--";
        }
```

```
  // Add to the string to be displayed.
  public void start()
  {
        msg += " Starting --";
  }

  // Display the msg in the applet window.
  public void paint(Graphics g)
  {
        msg += "  Painting.";
        g.drawString(msg, 10, 30);
   }
}
```

OUTPUT:



Requesting repainting;

- ✓ The **repaint()** method is defined by the AWT. It causes the AWT run time system to call to your applet's update() method, which in its default implementation, calls paint(). Again for example if a part of your applet

needs to output a string, it can store this string in a variable and then call repaint(). Inside paint(), you can output the string using drawstring().

The repaint method has four forms.

```
void repaint()

void repaint(int left, int top, int width, int height)

void repaint(long maxDelay)

void repaint(long maxDelay, int x, int y, int width, int height)
```

void repaint()

This causes the entire window to be repainted

void repaint(int left, int top, int width, int height)

This specifies a region that will be repainted. the integers left, top, width and height are in pixels. You save time by specifying a region to repaint instead of the whole window.

void repaint(long maxDelay)
void repaint(long maxDelay, int x, int y, int width, int height)

Calling repaint() is essentially a request that your applet be repainted sometime soon. However, if your system is slow or busy, update() might not be called immediately. This gives rise to a problem of update() being called sporadically. If your task requires consistent update time, like in animation, then use the above two forms of repaint(). Here, the maxDelay() is the maximum number of milliseconds that can elaspe before update() is called.

Using the Status Window

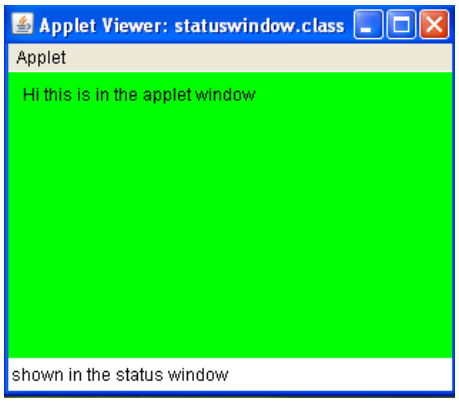
- ✓ If the user has chosen to show the Status Bar in their browser then messages can be put there from an applet.

  The **showStatus()** method would do it for this applet, if the applet was running in a browser.

Example

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

public class statuswindow extends Applet
{
        public void init()
        {
                setBackground(Color.green);

        }
        public void paint(Graphics g)
        {
        g.drawString("Hi this is in the applet window",10,20);
        showStatus("shown in the status window");
        }
}
```

OUTPUT



The HTML APPLET Tag

- ✓ The APPLET tag is used to start an applet from both an HTML document and from an applet viewer.
- ✓ An applet viewer will execute each APPLET tag that it finds in a separate window, while web browsers like Netscape Navigator, Internet Explorer, and HotJava will allow many applets on a single page.

The syntax for the standard APPLET tag is shown here. Bracketed items are optional.

```
< 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>
```

CODEBASE: CODEBASE is an optional attribute that specifies the base URL of the applet code, which is the directory that will be searched for the applet's executable class file (specified by the CODE tag).

CODE: CODE is a required attribute that gives the name of the file containing your applet's compiled **.class** file. This file is relative to the code base URL of the applet, which is the directory that the HTML file was in or the directory indicated by CODEBASE if set.

ALT The ALT tag is an optional attribute used to specify a short text message that should be displayed if the browser understands the APPLET tag but can't currently run Java applets. This is distinct from the alternate HTML you provide for browsers that don't support applets.

WIDTH AND HEIGHT: WIDTH and HEIGHT are required attributes that give the size (in pixels) of the applet display area.

ALIGN: ALIGN is an optional attribute that specifies the alignment of the applet. This attribute is treated the same as the HTML IMG tag with these possible values: LEFT, RIGHT, TOP, BOTTOM, MIDDLE, BASELINE, TEXTTOP, ABSMIDDLE, and ABSBOTTOM.

VSPACE AND HSPACE: These attributes are optional. VSPACE specifies the space, in pixels, above and below the applet. HSPACE specifies the space, in pixels, on each side of the applet. They're treated the same as the IMG tag's VSPACE and HSPACE attributes.

PARAM NAME AND VALUE: The PARAM tag allows you to specify appletspecific arguments in an HTML page. Applets access their attributes with the **getParameter( )** method.

Passing parameters to Applets;

- ✓ Parameters are passed to applets in **NAME=VALUE pairs in <PARAM> tags** between the opening and closing APPLET tags. Inside the applet, you read the values passed through the PARAM tags with the **getParameter() method** of the java.applet.Applet class.

The program below demonstrates this with a generic string drawing applet. The applet parameter "Message" is the string to be drawn.

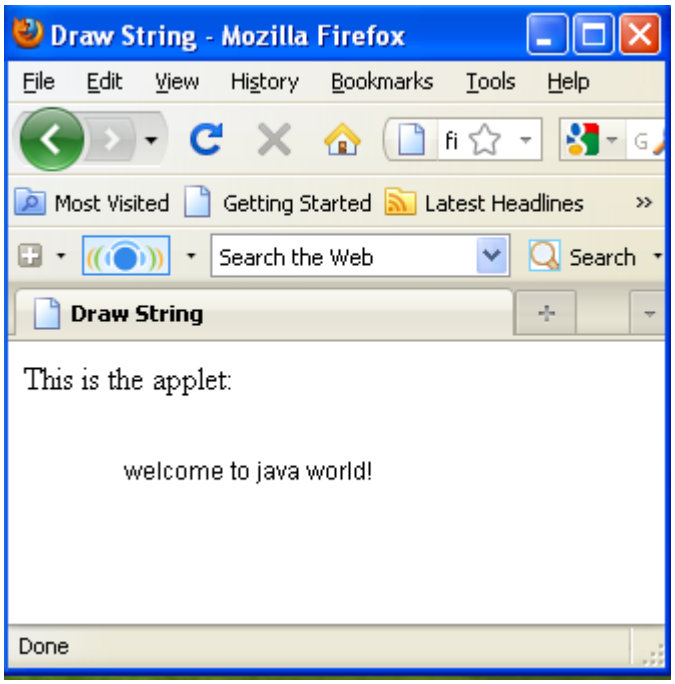
Example:

```
import java.applet.*;
import java.awt.*;
public class DrawStringApplet extends Applet
{
        private String defaultMessage = "Hello!";

        public void paint(Graphics g)
        {
            String inputFromPage = this.getParameter("Message");
             if (inputFromPage == null)

            inputFromPage = defaultMessage;

            g.drawString(inputFromPage, 50, 25);
         }
}
```

You also need an HTML file that references your applet. The following simple HTML file will do:

```
 <HTML>
<HEAD>
<TITLE> Draw String </TITLE>
</HEAD>

<BODY>
This is the applet:<P>
<APPLET code="DrawStringApplet" width="300" height="50">
<PARAM name="Message" value="welcome to java world!">
This page will be very boring if your
browser doesn't understand Java.
</APPLET>
</BODY>
</HTML>
```

OUTPUT



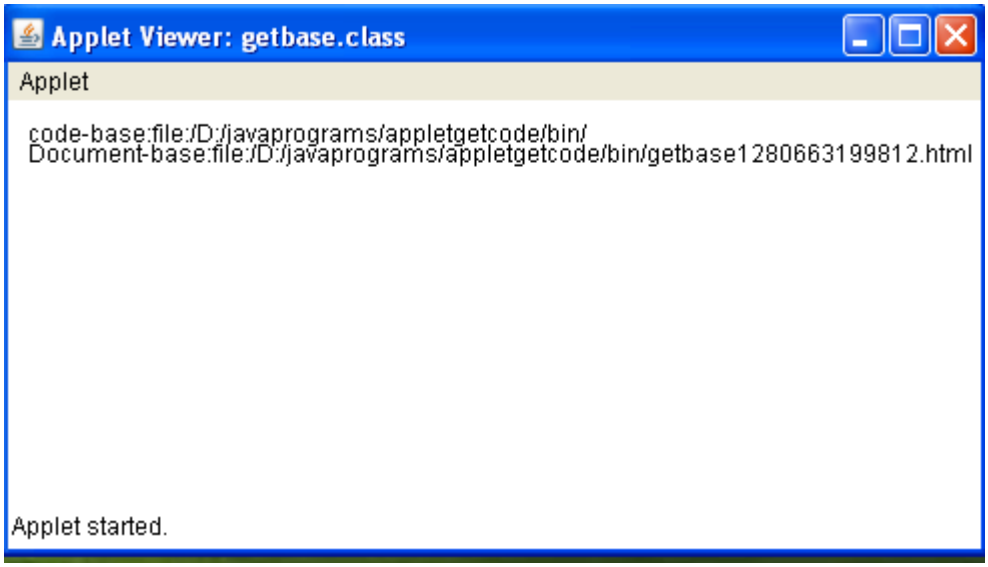
- ✓ You pass getParameter() a string that names the parameter you want. This string should match the name of a PARAM element in the HTML page. getParameter() returns the value of the parameter.
- ✓ All values are passed as strings. If you want to get another type like an integer, then you'll need to pass it as a string and convert it to the type you really want.
- ✓ The PARAM element is also straightforward. It occurs between <APPLET> and </APPLET>. It has two attributes of its own, NAME and VALUE. NAME identifies which PARAM this is. VALUE is the string value of the PARAM. Both should be enclosed in double quote marks if they contain white space.

getDocumentbase() and getCodebase()

- ✓ We sometimes need to load media and Text with the help of Applets. We have the facility to load the data from the directory which holds the HTML file which started the applet and the directory from which the applet's class loaded. These directories are returned in the form of URL by **getDocumnetBase( ) and getCodeBase( ) methods.**

```
import java.awt.*;
import java.applet.*;
import java.net.*;
public class getbase extends Applet
{
        public void paint(Graphics g)
        {
                String message;
           URL url=getCodeBase();
                message="code-base:"+url.toString();
                g.drawString(message,10,20);
                url=getDocumentBase();
                message="Document-base:"+url.toString();
                g.drawString(message,10,30);

        }
}
```

OUTPUT



AppletContext and showDocument()

- ✓ AppletContext is an interface which helps us to get the required information from the environment in which the applet is running and getting executed.
- ✓ This information is derived by **getAppletContext( )** method which is defined by Applet. Once we get the information with the above mentioned method, we can easily bring another document into view by calling **showDocument( )** method. The basic functionality of this method is that it returns no value and never throw any exception even if it fails hence needed to be implemented with utmost care and caution.

There are two showDocument( ) methods.

1. The **method showDocument(URL)** displays the document at the specified URL.

2. The **method showDocument(URL, where)** displays the specified document at the specified location within the browser window.

```
import java.awt.*;
import java.applet.*;
import java.net.*;
public class contextdoc extends Applet
{
        public void start()
        {
                AppletContext ac=getAppletContext();
                URL url=getCodeBase();
                try
                {
                        ac.showDocument(new URL(url+"demo.html"));
                }
                catch(MalformedURLException e)
                {
                        showStatus("URL not found");

                }

        }
}
```

The AudioClip Interface;

- ✓ The AudioClip interface is a simple abstraction for playing a sound clip. Multiple AudioClip items can be playing at the same time, and the resulting sound is mixed together to produce a composite.

It contains three methods:

- ✓ Play(): Starts playing this audio clip.
- ✓ Stop(): Stops playing this audio clip.
- ✓ Loop():Starts playing this audio clip in a loop.

After you have loaded an audio clip using **getAudioClip()**, you can use these methods to play it.

The AppletStub Interface

- ✓ The AppletStub interface provides a way to get information from the run-time browser environment.

Output to the Console

- ✓ The output to an applet window must be accomplished through GUI based methods such as **drawstring()** as illustrated in the above example applet programs, it is still also possible to use console output in your applet for debugging purpose.
- ✓ In an applet program where you call a method such as **System.out.println()** the output is not sent to your applet window instead it appears either in the console session or in the java console that is available in some br **JAVA**

Questions

1. List applet initialization and termination method? Write a java applet that set the background color cyan and foreground color red and output a string message "A simple Applet"?
   (Jan 2013) 4 Marks
2. What are applets? Explain the different stages in the life cycle of applet? (Dec 2011)08Marks
3. How to embed applet inside the html page? Explain with an example program.
4. Explain getCodeBase() and getDocumentBase() methods.
5. Write a note on:
   a. showStatus().
   b. AppletContext and showDocument()
   c. AudioClip interface
6. Exaplin HTML Applet Tag attributes
7. With an example program explain how to pass parameters to Applet.
8. Explain Applet Skelton.

MODULE 5: Applets and Swings

Syllabus:

The Applet Class: Two types of Applets; Applet basics; Applet Architecture; An Applet skeleton; Simple Applet display methods; Requesting repainting; Using the Status Window; The HTML APPLET tag; Passing parameters to Applets; getDocumentbase() and getCodebase(); ApletContext and showDocument(); The AudioClip Interface; The AppletStub Interface; Output o the Console

Swings: The origins of Swing; Two key Swing features; Components and Containers; The Swing Packages; A simple Swing Application; Create a Swing Applet; Jlabel and ImageIcon; JTextField;The Swing Buttons; JTabbedpane; JScrollPane; JList; JComboBox; JTable.

Beautiful thought: "What you do today can improve all your tomorrows". - Ralph Marston

Introduction

Swing contains a set of classes that provides more powerful and flexible GUI components than those of **AWT**. **Swing** provides the look and feel of modern Java GUI. Swing library is an official Java GUI tool kit released by Sun Microsystems. It is used to create graphical user interface with Java.

Swing is a set of program component s for **Java** programmers that provide the ability to create graphical user interface (GUI) components, such as buttons and scroll bars, that are independent of the windowing system for specific operating system . **Swing** components are used with the **Java** Foundation Classes (JFC).

The Origins of Swing

The original Java GUI subsystem was the Abstract Window Toolkit (AWT). AWT translates it visual components into platform-specific equivalents (peers).

Under AWT, the look and feel of a component was defined by the platform. AWT components are referred to as **heavyweight**.

Swing was introduced in 1997 to fix the problems with AWT.

Swing offers following key features:

1. Platform Independent
2. Customizable
3. Extensible
4. Configurable
5. Lightweight

- ✓ Swing components are **lightweight** and don't rely on peers.
- ✓ Swing supports a pluggable look and feel.
- ✓ Swing is built on AWT.

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 applications create a containment-hierarchy with a single top-level container.

Components

Swing components are derived from the **JComponent** class. The only exceptions are the four top-level containers: JFrame, JApplet, JWindow, and JDialog.

JComponent inherits AWT classes Container and Component.

All the Swing components are represented by classes in the javax.swing package.

All the component classes start with **J**: JLabel, JButton, JScrollbar, …

Containers

There are two types of containers:

1) Top-level which do not inherit JComponent, and

2) Lightweight containers that do inherit JComponent.

Lightweight components are often used to organize groups of components. Containers can contain other containers.

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 instance of JPanel.

The Swing Packages:

Swing is a very large subsystem and makes use of many packages. These are the packages used by Swing that are defined by Java SE 6.

The main package is **javax.swing**. This package must be imported into any program that uses Swing. It contains the classes that implement the basic Swing components, such as push buttons, labels, and check boxes.

Some of the Swing Packages are:

javax.swing javax.swing.border javax.swing.colorchooser javax.swing.event javax.swing.filechooser javax.swing.plaf javax.swing.plaf.basic javax.swing.plaf.metal javax.swing.plaf.multi	javax.swing.plaf.synth javax.swing.table javax.swing.text javax.swing.text.html javax.swing.text.html.parser javax.swing.text.rtf javax.swing.tree javax.swing.undo

A simple Swing Application;

There are two ways to create a frame:

By creating the object of Frame class (association)
By extending Frame class (inheritance)

We can write the code of swing inside the main(), constructor or any other method.

By creating the object of Frame class

```
import javax.swing.*;
public class FirstSwing
{
	public static void main(String[] args)
	{
		JFrame f=new JFrame(" MyApp");
			//creating instance of JFrame and title of the frame is MyApp.
		JButton b=new JButton("click");
		   //creating instance of JButton  and name of the button is click.
		b.setBounds(130,100,100, 40);     //x axis, y axis, width, height
		f.add(b);                          //adding button in JFrame
		f.setSize(400,500);                //400 width and 500 height
		f.setLayout(null);                  //using no layout managers
		f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		f.setVisible(true);                //making the frame visible
	}
}
```

Output:

Explanation:

- ✓ The program begins by importing javax.swing. As mentioned, this package contains the components and models defined by Swing.
- ✓ For example, javax.swing defines classes that implement labels, buttons, text controls, and menus. It will be included in all programs that use Swing. Next,
- ✓ the program declares the FirstSwing class
- ✓ It begins by creating a JFrame, using this line of code:

```
JFrame f = new JFrame("My App");
```

- ✓ This creates a container called f that defines a rectangular window complete with a title bar; close, minimize, maximize, and restore buttons; and a

system menu. Thus, it creates a standard, top-level window. The title of the window is passed to the constructor.

Next, the window is sized using this statement:

```
f.setSize(400,500);
```

- ✓ The setSize( ) method (which is inherited by JFrame from the AWT class Component) sets the dimensions of the window, which are specified in pixels. in this example, the width of the window is set to 400 and the height is set to 500.
- ✓ By default, when a top-level window is closed (such as when the user clicks the close box), the window is removed from the screen, but the application is not terminated.
- ✓ If want the entire application to terminate when its top-level window is closed. There are a couple of ways to achieve this. The easiest way is to call setDefaultCloseOperation( ), as the program does:

```
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Swing by inheritance

- ✓ We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

```
import javax.swing.*;
public class MySwing extends  JFram   //inheriting JFrame
{
   JFrame f;
   MySwing()
   {
        JButton b=new JButton("click");//create
        button b.setBounds(130,100,100,  40);

        add(b);//adding button on
        frame setSize(400,500);
        setLayout(null);
        setVisible(true);
   }
   public static void main(String[] args)
   {
       new MySwing();
   }
}
```

Jlabel, JTextField and JPassword

- ✓ **JLabel** is Swing's easiest-to-use component. It creates a label and was introduced in the preceding chapter. Here, we will look at JLabel a bit more closely.
- ✓ JLabel can be used to display text and/or an icon. It is a passive component in that it does not respond to user input. JLabel defines several constructors. Here are three of them:

JLabel(Icon icon)

JLabel(String str)

JLabel(String str, Icon icon, int align)

- **JTextField** is the simplest Swing text component. It is also probably its most widely used text component. JTextField allows you to edit one line of text. It is derived from JTextComponent, which provides the basic functionality common to Swing text components.

Three of JTextField's constructors are shown here:

JTextField(int cols)

JTextField(String str, int cols)

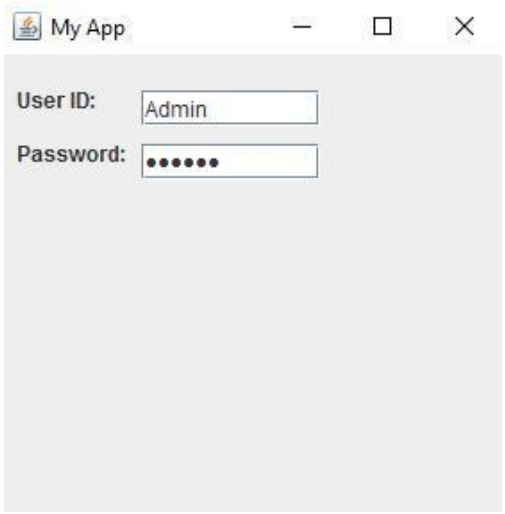
JTextField(String str)

- Here, str is the string to be initially presented, and cols is the number of columns in the text field. If no string is specified, the text field is initially empty. If the number of columns is not specified, the text field is sized to fit the specified string.

- ***JPasswordField*** is a lightweight component that allows the editing of a single line of text where the view indicates something was typed, but does not show the original characters.

```
import javax.swing.*;
public class JTextFieldPgm
{

     public static void main(String[] args)
     {

          JFrame f=new JFrame("My App");

          JLabel namelabel= new JLabel("User ID: ");
          namelabel.setBounds(10, 20, 70, 10);
```

```
      JLabel passwordLabel = new JLabel("Password: ");
         passwordLabel.setBounds(10, 50, 70, 10);

               JTextField userText = new JTextField();
               userText.setBounds(80, 20, 100, 20);

JPasswordField  passwordText = new JPasswordField();
               passwordText.setBounds(80, 50, 100, 20);

               f.add(namelabel);
               f.add( passwordLabel);
               f.add(userText);
               f.add(passwordText);

               f.setSize(300, 300);
               f.setLayout(null);
               f.setVisible(true);

     }

}
```



ImageIcon with JLabel

- JLabel can be used to display text and/or an icon. It is a passive component in that it does not respond to user input. JLabel defines several constructors. Here are three of them:

  JLabel(Icon icon)

  JLabel(String str)

  JLabel(String str, Icon icon, int align)

- Here, str and icon are the text and icon used for the label.
- The align argument specifies the horizontal alignment of the text and/or icon within the dimensions of the label. It must be one of the following values: LEFT, RIGHT, CENTER, LEADING, or TRAILING.
- These constants are defined in the Swing Constants interface, along with several others used by the Swing classes. Notice that icons are specified by objects of type Icon, which is an interface defined by Swing.
- The easiest way to obtain an icon is to use the ImageIcon class. ImageIcon implements Icon and encapsulates an image. Thus, an object of type ImageIcon can be passed as an argument to the Icon parameter of JLabel's constructor.

  ImageIcon(String filename)

- It obtains the image in the file named filename. The icon and text associated with the label can be obtained by the following methods:

  Icon getIcon( )

  String getText( )

The icon and text associated with a label can be set by these methods:

void setIcon(Icon icon)

void setText(String str)

✓ Here, **icon and str** are the icon and text, respectively. Therefore, using setText( ) it is possible to change the text inside a label during program execution.

```
import javax.swing.*;

public class PgmImageIcon
{

     public static void main(String[] args)
     {

          JFrame jf=new
          JFrame("Image Icon");
          jf.setLayout(null);

           Icon icon = new ImageIcon("a.jpg");
  JLabel label1 = new JLabel("Welocme to    SVIT",icon,JLabel.RIGHT);
         label1.setBounds(20, 30,
         267, 200); jf.add(label1);


         jf.setSize(
         300,400);
         jf.setVisib
         le(true);
     }

}
```

The Swing Buttons:

There are four types of Swing Button

1. JButton
2. JRadioButton
3. JCheckBox
4. JComboBox

JButton class provides functionality of a button. A JButton is the Swing equivalent of a Button in AWT. It is used to provide an interface equivalent of a common button.

JButton class has three constuctors,

JButton(Icon *ic*)

JButton(String *str*)

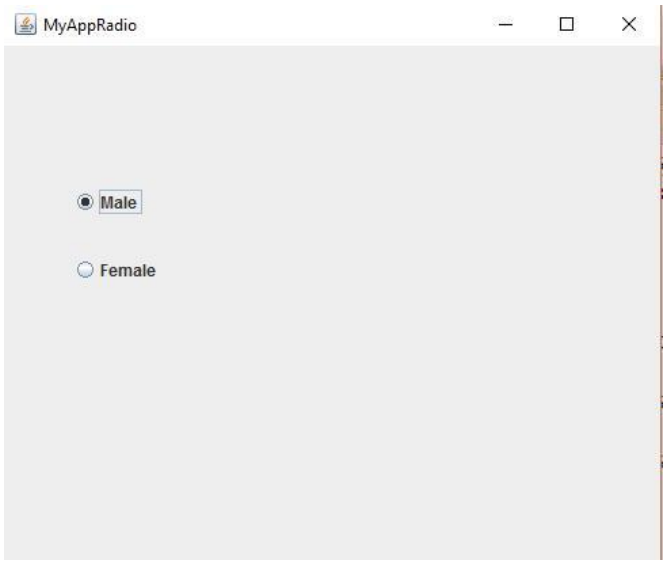
JButton(String *str*, Icon *ic*)

```
import javax.swing.*;

class FirstSwing
{
     public static void main(String args[])
     {
          JFrame jf=new JFrame("My App");

          JButton jb=new JButton("Next");
          jb.setBounds(30, 100, 100, 50);

          JButton jb1=new JButton("Prev");
          jb1.setBounds(30, 200, 100, 50);

          jf.add(jb);
          jf.add(jb1);

          jf.setSize(300, 600);
          jf.setLayout(null);
          jf.setVisible(true);

     }
}
```

A **JRadioButton** is the swing equivalent of a RadioButton in AWT. It is used to represent multiple option single selection elements in a form. This is performed by grouping the JRadio buttons using a ButtonGroup component. The ButtonGroup class can be used to group multiple buttons so that at a time only one button can be selected.

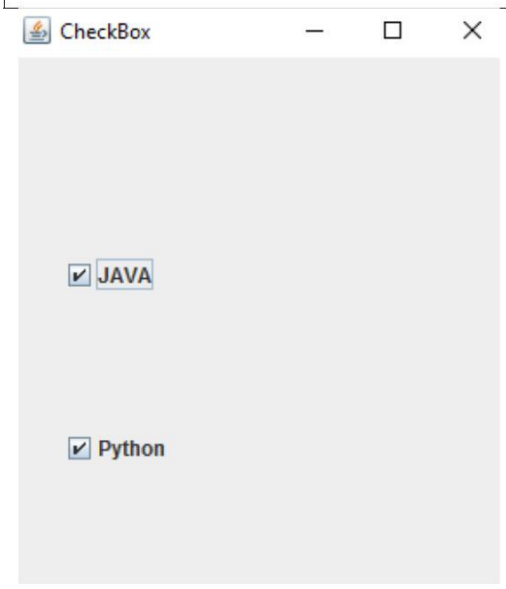
```
import javax.swing.*;

import javax.swing.*;
public class RadioButton1
{
     public static void main(String args[])
     {

          JFrame f=new JFrame("MyAppRadio");

     JRadioButton r1=new JRadioButton("Male ");

     JRadioButton r2=new JRadioButton("Female");

          r1.setBounds(50, 100, 70, 30);
```

```
        r2.setBounds(50,150,70,30);

        ButtonGroup bg=new
        ButtonGroup(); bg.add(r1);
        bg.add(r2);


        f.add(r1)
        ;
        f.add(r2)
        ;

        f.setSize(500,500);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```



A **JCheckBox** is the Swing equivalent of the Checkbox component in AWT. This is sometimes called a ticker box, and is used to represent multiple option selections in a form.

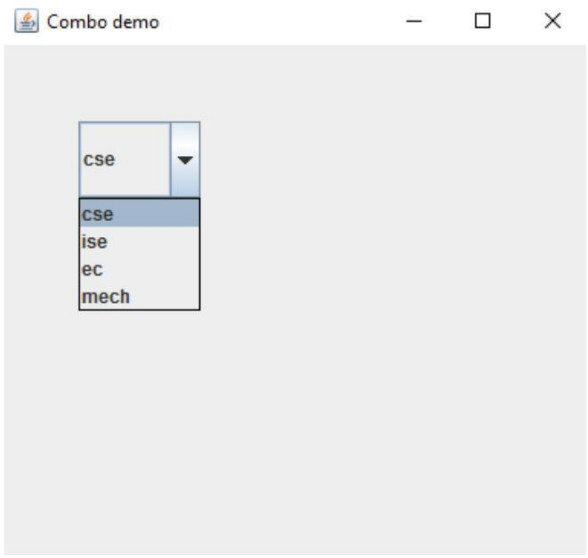
```
import javax.swing.*;

class FirstSwing
{
     public static void main(String args[])
     {
          JFrame jf=new JFrame("CheckBox");

          JCheckBox jb=new JCheckBox("JAVA");
          jb.setBounds(30, 100, 100, 50);

          JCheckBox jb1=new JCheckBox("Python");
          jb1.setBounds(30, 200, 100, 50);

          jf.add(jb);
          jf.add(jb1);

          jf.setSize(300, 600);
          jf.setLayout(null);
          jf.setVisible(true);

     }
}
```



JComboBox

The JComboBox class is used to create the combobox (drop-down list). At a time only one item can be selected from the item list.

```
import java.awt.*;
import javax.swing.*;
public class Comboexample
{
        public static void main(String[] args)
        {
                JFrame f=new JFrame("Combo demo");
                String Branch[]={"cse","ise","ec","mech"};
                JComboBox jc=new JComboBox(Branch);
                jc.setBounds(50,50,80,50);
                f.add(jc);
                f.setSize(400, 400);
                f.setLayout(null);
                f.setVisible(true);
        }
}
```



JTable and JScrollPane:

The JTable class is used to display the data on two dimensional tables of cells.

Commonly used Constructors of JTable class:

JTable(): creates a table with empty cells.

JTable(Object[ ][ ] rows, Object[ ] columns): creates a table with the specified data.

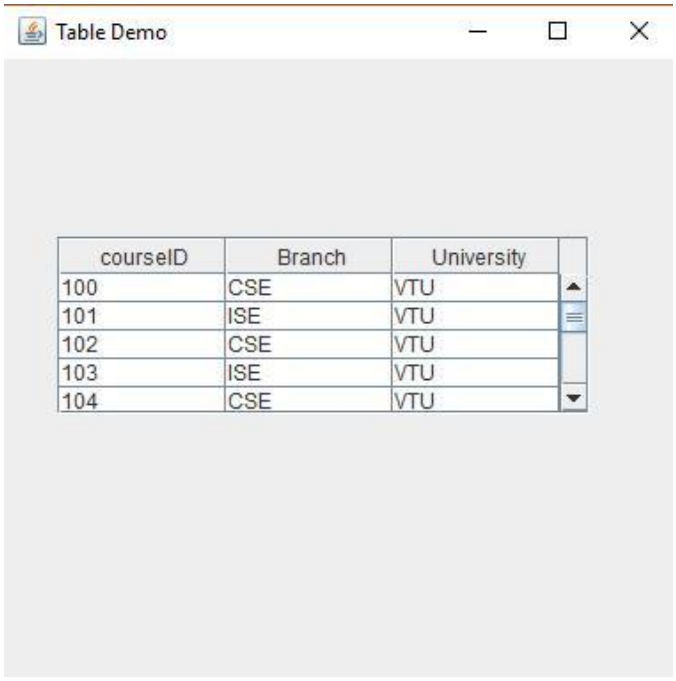
- ✓ JScrollPane is a lightweight container that automatically handles the scrolling of another component.
- ✓ The component being scrolled can either be an individual component, such as a table, or a group of components contained within another lightweight container, such as a JPanel.
- ✓ In either case, if the object being scrolled is larger than the viewable area, horizontal and/or vertical scroll bars are automatically provided, and the

component can be scrolled through the pane. Because JScrollPane automates scrolling, it usually eliminates the need to manage individual scroll bars.

- ✓ The viewable area of a scroll pane is called the viewport.
- ✓ It is a window in which the component being scrolled is displayed.
- ✓ Thus, the view port displays the visible portion of the component being scrolled. The scroll bars scroll the component through the viewport.
- ✓ In its default behavior, a JScrollPane will dynamically add or remove a scroll bar as needed. For example, if the component is taller than the viewport, a vertical scroll bar is added. If the component will completely fit within the viewport, the scroll bars are removed.

```
import javax.swing.*;
public class TableExample1
{
	public static void main(String[] args)
	{
		// TODO Auto-generated method stub
		JFrame f=new JFrame("Table Demo");

		String data[][]={
					{"100","CSE","VTU"}
					,
					{"101","ISE","VTU"}
					,
					{"102","CSE","VTU"}
					,
					{"103","ISE","VTU"}
					,
					{"105","ISE","VTU"}
					,
					{"106","ISE","VTU"}
				};
	String column[]={"courseID","Branch","University"};

		JTable jt=new JTable(data,column);
```

```
            JScrollPane js=new JScrollPane(jt);
            js.setBounds(30,100,300,100);
            f.add(js);

             f.setSize(300,400);
             f.setLayout(null);
             f.setVisible(true);

      }
}
```



JTabbedpane

- ✓ JTabbedPane encapsulates a tabbed pane. It manages a set of components by linking them with tabs.
- ✓ Selecting a tab causes the component associated with that tab to come to the forefront. Tabbed panes are very common in the modern GUI.

- ✓ Given the complex nature of a tabbed pane, they are surprisingly easy to create and use. JTabbedPane defines three constructors. We will use its default constructor, which creates an empty control with the tabs positioned across the top of the pane.
- ✓ The other two constructors let you specify the location of the tabs, which can be along any of the four sides.
- ✓ JTabbedPane uses the SingleSelectionModel model. Tabs are added by calling **addTab( ) method.** Here is one of its forms:

  void addTab(String name, Component comp)

- ✓ Here, **name** is the name for the tab, and **comp** is the component that should be added to the tab. Often, the component added to a tab is a JPanel that contains a group of related components. This technique allows a tab to hold a set of components.

```
import javax.swing.*;


public class MainClass

{

  public static void main(String[] a)

  {

    JFrame f = new JFrame("JTab");



    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    f.add(new JTabbedPaneDemo());
```

```
    f.setSize(500, 500);
    f.setVisible(true);
  }
}
class JTabbedPaneDemo extends JPanel
{
                JTabbedPaneDemo()
                {
                        makeGUI();
                }
               void makeGUI()
                {
                        JTabbedPane jtp = new JTabbedPane();
                        jtp.addTab("Cities", new CitiesPanel());
                        jtp.addTab("Colors", new ColorsPanel());
                        jtp.addTab("Flavors", new FlavorsPanel());
                        add(jtp);
                }
}
```

```
class CitiesPanel extends JPanel
{
  public CitiesPanel()
  {
    JButton b1 = new JButton("NewYork");
    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);
  }
}
```

JList:

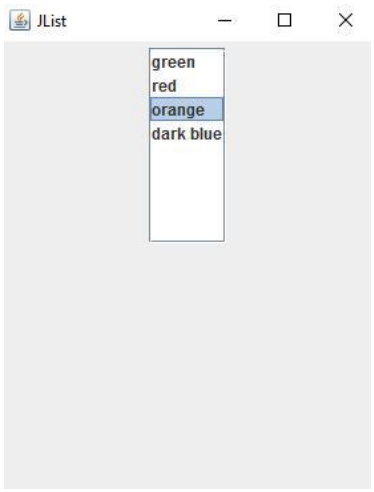
- ✓ In Swing, the basic list class is called JList.
- ✓ It supports the selection of one or more items from a list.
- ✓ Although the list often consists of strings, it is possible to create a list of just about any object that can be displayed.
- ✓ JList is so widely used in Java that it is highly unlikely that you have not seen one before.

JList provides several constructors. The one used here is

JList(Object[ ] items)

- ✓ This creates a JList that contains the items in the array specified by items.
- ✓ JList is based on two models. The first is ListModel. This interface defines how access to the list data is achieved.
- ✓ The second model is the ListSelectionModel interface, which defines methods that determine what list item or items are selected.

```
import java.awt.FlowLayout;

import javax.swing.*;

public class JListPgm
{
  public static void main(String[] args)
  {

    JFrame frame = new JFrame("JList");

String[] selections = { "green", "red", "orange", "dark blue"
    }; JList list = new JList(selections);

    list.setSelectedIndex(1);

    frame.add(new JScrollPane(list));

    frame.setSize(300, 400);
    frame.setLayout(new FlowLayout());
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    frame.setVisible(true);
  }

}
```



	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**.

Questions

1. Write a swing applet program to demonstrate with two JButtons named India and Srilanka. When either of button pressed, it should display respective label with its icon. Refer the image icon
   "india.gif" and "srilanka.gif". set the initial label is "press the button" **(Jan 2015)10marks**
2. Explain JscrollPane with an example. **(Jan 2015) 5marks**
3. Explain IComboBox with an example. **(Jan 2015) 5marks**
4. Name & Explain the different types of Swing Buttons with syntax.

   (Jan 2014) 10 Marks
5. Write the steps to create J-table.write a program to create a table with column heading "fname,lname,age" and insert at least five records in the table and display. (Jan 2014) 10 Marks

6. Differentiate between AWT and Swings? (Jan 2013) 05 Marks

7. Explain the MVC architecture of swings?(Jan 2013)10 Marks

8. Describe the different types of swing button? ?(Jan 2013)10 Marks

9. What is a swing ? explain the components and containers in the swings (Dec 2011)08Marks

10. Explain the following with an example for each

i)JTextField class ii)JButton class iii)JComboBox Class

(Dec 2011)12Marks

11. Explain the following swing buttons.

A. JButton **B**. JToggleButton

C. ChekBoxes **D**. Radio Buttons

12. Explain the concept of JComboBox and JTable. (Jan-2010)

13. Write a program which displays the contents of an array in the tabular format.