Introduction of Java-Applet

Applet is a java program that can be embedded into HTML pages. Applets transported over the internet from one computer to another and run using Applet Viewer or can run on any java enables web browsers (Mozilla, internet explorer etc) using JVM. When a user views an HTML page that contains an Applet, the code for the applet is downloaded to the user's machine. An Applet is designed to run remotely in the client browser, so there are some restrictions on it. The applet can't access system resources on the local computer. Applets are used to make the web site more dynamic and entertaining. An Applet is a Java class that extends the java.applet.Applet class. The Applet class provides the standard interface between the applet and the browser environment.

We generally use web browsers to run applets. It's not always mandatory to open a Web browser for running an Applet. The other way to run an Applet is through **Java applet viewer**. **Applet viewer** is a command line program to run Java applets. It is included in the SDK. It helps you to test an **applet** before you run it in a browser. This is a tool to test for Java applets. The working of Applet viewer is a bit different from a Web browser, though they are logically same. The Applet viewer runs on the HTML documentation, and uses embedded applet tags. The difference in using the applet viewer and the web browser to run the applet is that the applet viewer only deals with the applet code not the HTML cod i.e. it doesn't display HTML code. So we should test our program in applet viewer and web browser to confirm its working.

Definition: "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 provides a standard interface between applets and their environment."

Four definitions of applet:

- A small application
- A secure program that runs inside a web browser
- A subclass of java.applet.Applet
- An instance of a subclass of java.applet.Applet

Advantages of Applet:

1. It provides a GUI. Facilitates graphics, animation and multimedia.
2. Provides a facility for frames and enables an event handling.
3. Avoids a risk and provides a secure two-way interaction between web pages.
4. It is simple to make it work on Linux, Windows and Mac OS i.e. to make it cross platform.
5. Applets can work all the version of Java Plug-in.
6. An applet runs in a sandbox, so the user does not need to trust the code, so it can work without security approval.
7. Applets are supported by most web browsers.
8. Applets are cached in most web browsers, so will be quick to load when returning to a web page.
9. User can also have full access to the machine if user allows.
10. It can improve with use: after a first applet is run, the JVM is already running and starts quickly, benefitting regular users of Java but the JVM will need to restart each time the browser starts fresh.
11. It can run at a comparable (but generally slower) speed to other compiled languages such as C++, but many times faster than JavaScript.

12. It can move the work from the server to the client, making a web solution more scalable with the number of users/clients.

Disadvantages of Java Applet:

1. Java plug-in is required to run applet.
2. Java applet requires JVM so first time it takes significant startup time.
3. If applet is not already cached in the machine, it will be downloaded from internet and will take time.
4. It's difficult to design and build good user interface in applets compared to HTML technology.
5. Applets may require a specific JRE. This is discouraged.
6. Some organizations only allow software installed by the administrators. As a result, many users cannot view applets by default.

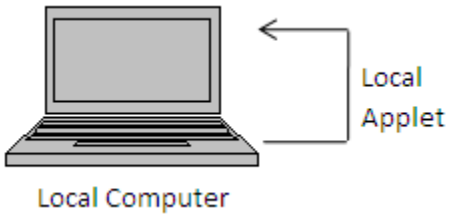
Types of Applets

Local Applets:

- An applet developed locally and stored in a local system is known as a **local applet.**
- When a web page is trying to find a local applet, it does not need to use the internet and therefore the local system does not require the internet connection. It simply searches the directories in the local system; locates and loads the specified applet.
- A local applet is specified by a path name and a file name.

Example of local applet:

```
<applet
   codebase="solitaire"
   code="Solitaire.class"
   width=120
   height=120>
</applet>
```



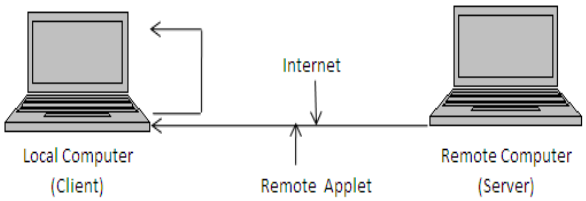
Loading Local Applets

Remote Applets:

- A **remote applet** is that which is developed by someone else and stored on a remote computer connected to the internet. If our system is connected to the internet, we can download the remote applet into our system via at the internet and run it.
- Your browser must, of course, be connected to the Internet at the time it needs to display the remote applet.
- For locate and load a remote applet, we must know the applet's address on the web. This address is known as Uniform Resource Location (URL).

Example:

```
<applet
 codebase="http://www.myconnect.com/applets/"
   code="Solitaire.class"
   width=120
   height=120>
</applet>
```



Loading a Remote Applet

Differentiate between Applets and Application

Applets	Application
Applets run in web pages.	Applications run on stand-alone systems.
Applets are not full featured application programs.	Applications are full featured programs.
Applets are the small programs.	Applications are larger programs.
Applet starts execution with its init method.	Application starts execution with its main method
Applets cannot run independently. They are run from inside a web page using a special feature known as HTML tag. Designed for use on the world wide web (www).	Java application generally refers to an application that is designed standalone use.
Parameters to the applet are given in the HTML file.	Parameters to the application are given at the command prompt (for example args[0], args[1], args[2] and so on).
In applet, there are no main() method that is executed continuously throughout the life of an Applet.	In an application, the program starts at the main() method. The main() method runs throughout the application.
Applets cannot read from or write to the files in the local computer.	Applications read from or write to the files in the local computer.
Applets cannot communicate with other servers on the network.	An application communicates other servers on the network.
Applets cannot run any program from the local computer.	Application runs any program from the local computer.
Applet must run within a GUI.	Application can run with or without GUI
Java applets require an external viewer program, so to use an applet; you need web browser or an applet viewer.	Java application does not require an external viewer program. This means that you can execute a java application directly using the java interpreter.
Applet cannot access the local file system and resources.	Application can access the local file system and resources.
Creating and running an Applet is complex.	Creating and running an application is easy.
Author of the applets is not known.	Author of the application is known.
Applets are event driven.	Applications are control driven.
Applets are designed just for handling the client site problems.	Java applications are designed to work with the client as well as server.
While the applets are typically used.	Applications are designed to exist in a secure area.
An Applet is subjected to more stringent security restrictions in terms of file and network access.	Whereas an application can have free reign over these resources.

Creating an Executable Applet

Executable applet is nothing but the .class file of the applet, which is obtained by compiling the source code of the applet. Compiling the applet is exactly the same as compiling an application. Therefore, we can use the java compiler to compile the applet.
Steps required for compiling the applet are:

1. Create applet and save into file with java extension for example. TestApplet.java
2. Compile applet file by javac TestApplet.java
3. The compiled output file is called .class file (TestApplet.class)
4. If any error message is received, then we must check for errors, correct them and compile the applet again.

Steps involved in developing and testing in applets

- Building an applet code (.java file)
- Creating an executable applet (.class file)
- Designing a web page using HTML tags
- Preparing <APPLET> tag
- Incorporating <APPLET> tag into the web page
- Creating HTML file
- Testing the applet code

Running the Applet using Applet viewer:

1. Open notepad and type the following source code and save it into file name “Hellojava.java”

```
import java.awt.*;
import java.applet.*;
public class Hellojava extends Applet
{
        public void paint (Graphics g)
{
g.drawString("Hello Java",10,100);
}
}
/*<applet code="Hellojava.class" width=300 height=200>
</applet>*/
```

2. Use the java compiler to compile the applet “Hellojava.java” files.

```
C:\jdk> javac Hellojava.java
```

3. After compilation “Hellojava.class” file will be created. Executable applet is nothing but the .class file of the applet, which is obtained by compiling the source code of the applet. If any error message is received, then check the errors, correct them and compile the applet again.
4. The Appletviewer is a program which provides a java run time environment for applets. We can use it to run our applet as follows: **appletviewer HelloJava.java**
6. Output of applet is:

Adding Applet to the HTML file:

Steps to add an applet in HTML document

1. Insert an <APPLET> tag at an appropriate place in the web page i.e. in the body section of HTML file.
2. Specify the name of the applet's .class file.
3. If the .class file is not in the current directory then use the codebase parameter to specify:-
   a. the relative path if file is on the local system, or
   b. the uniform resource locator(URL) of the directory containing the file if it is on a remote computer.
4. Specify the space required for display of the applet in terms of width and height in pixels.
5. Add any user-defined parameters using <param> tags
6. Add alternate HTML text to be displayed when a non-java browser is used.
7. Close the applet declaration with the </APPLET> tag.

Open notepad and type the following source code and save it into file name "Hellojava.java"

```
import java.awt.*;
import java.applet.*;
public class Hellojava extends Applet
{
        public void paint (Graphics g)
{
g.drawString("Hello Java",10,100);
} }
```

Use the java compiler to compile the applet "Hellojava.java" file.

C:\jdk> javac Hellojava.java

After compilation "Hellojava.class" file will be created. Executable applet is nothing but the .class file of the applet, which is obtained by compiling the source code of the applet. If any error message is received, then check the errors, correct them and compile the applet again.

We must have the following files in our current directory.

- Hellojava.java
- Hellojava.class
- HelloJava.html

If we use a java enabled web browser, we will be able to see the entire web page containing the applet. We have included a pair of <APPLET..> and </APPLET> tags in the HTML body section. The <APPLET…> tag supplies the name of the applet to be loaded and tells the browser how much space the applet requires. The <APPLET> tag given below specifies the minimum requirements to place the HelloJava applet on a web page. The display area for the applet output as 300 pixels width and 200 pixels height. CENTER tags are used to display area in the center of the screen.

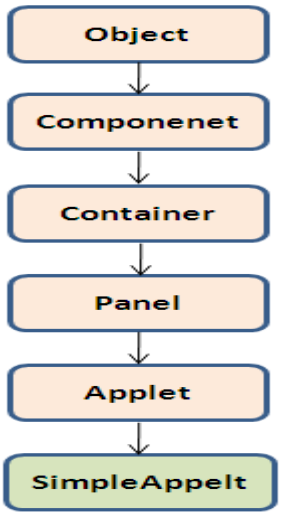
<APPLET CODE = hellojava.class WIDTH = 400 HEIGHT = 200 > </APPLET>

Example: Adding applet to HTML file:
Create Hellojava.html file with following code:

```
<HTML>
<! This page includes welcome title in the title bar and displays a welcome message. Then it specifies the applet to be loaded and executed.
>
<HEAD> <TITLE> Welcome to Java Applet </TITLE> </HEAD>
<BODY> <CENTER> <H1> Welcome to the world of Applets </H1> </CENTER> <BR>
<CENTER>
<APPLET CODE=HelloJava.class WIDTH = 400 HEIGHT = 200 > </APPLET>
</CENTER>
</BODY>
</HTML>
```

Applet Structure and Elements:

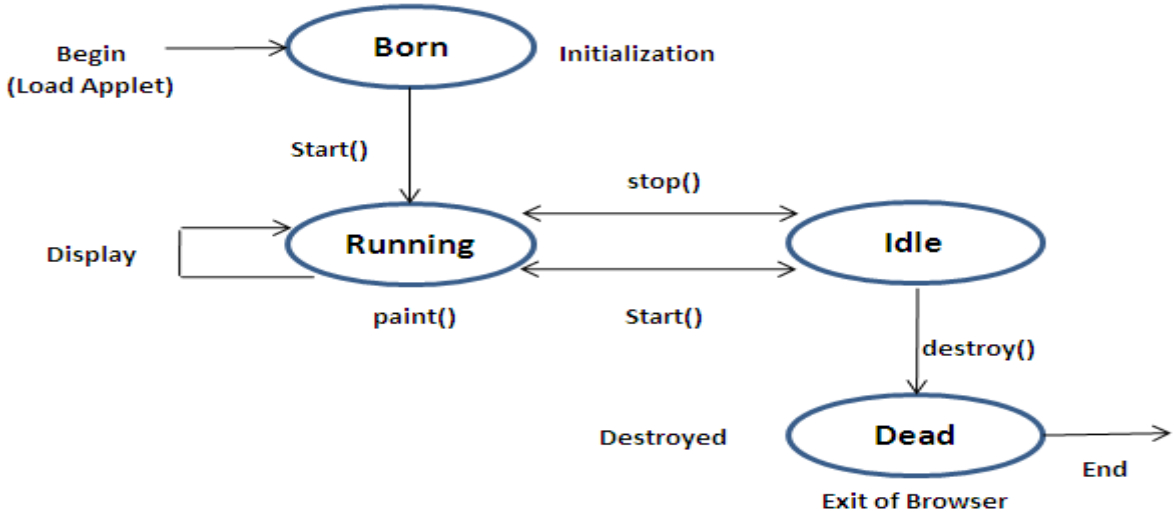
The Java API Applet class provides what you need to design the appearance and manage the behavior of an Applet. This class provides a graphical user interface (GUI) component called a Panel and a number of methods. To create an Applet, you extend (or subclass) the Applet class and implement the appearance and behavior you want. The applet's appearance is created by drawing onto the Panel or by attaching other GUI components such as push buttons, scrollbars, or text areas to the Panel. The applet's behavior is defined by implementing the methods. The `SimpleApplet` class extends `Applet` class, which extends the `Panel` class, which extends the `Container` class. The `Container` class extends `Object`, which is the parent of all Java API classes. The `Applet` class provides the `init`, `start`, `stop`, `destroy`, and `paint` methods.



Chain of classes inherited by Applet Class

Applet Life Cycle:

An Applet has a life cycle, which describes how it starts, how it operates and how it ends. The life cycle consists of four methods: init(), start(), stop() and destroy().



Applet Life Cycle

Initialization State (The init() method):

The life cycle of an Applet is begin on that time when the applet is first loaded into the browser and called the init() method. The init() method is called only one time in the life cycle on an Applet. The init() method is basically called to read the "PARAM" tag in the html file. The init () method retrieve the passed parameter through the "PARAM" tag of html file using get Parameter() method All the initialization such as initialization of variables and the objects like image, sound file are loaded in the init () method .After the initialization of the init() method user can interact with the Applet and mostly applet contains the init() method.

We may do following thing if required.

- Create objects needed by the applet
- Set up initial values
- Load images or fonts
- Set up colors

```
public void init()
{
……
}
```

Running State (The start() method): The start method of an Applet is called after the initialization method init(). This method may be called multiples time when the Applet needs to be started or restarted. For Example if the user wants to return to the Applet, in this situation the start() method of an Applet will be called by the web browser and the user will be back on the applet. In the start method user can interact within the applet.

```
public void start()
{
……..
……..
}
```

Idle (The Stop() method): An applet becomes idle when it is stopped from running. The stop() method stops the applet and makes it invisible. Stopping occurs automatically when we leave the page containing the currently running applet. We can also do so by calling the stop() method explicitly. The stop() method can be called multiple times in the life cycle of applet like the start () method or should be called at least one time. For example the stop() method is called by the web browser on that time When the user leaves one applet to go another applet and the start() method is called on that time when the user wants to go back into the first program or Applet.

```
public void stop()
{
……..
……..
}
```

Dead State (The destroy() method): The destroy() method is called to terminate an Applet. an Applet is said to be dead when it is removed from memory. This occurs automatically by invoking the destroy() method when we quit the browser. It is useful for clean-up actions, such as releasing memory after the applet is removed, killing off threads and closing network/database connections. Thus this method releases all the resources that were initialized during an applet's initialization.

```
public void destroy()
{
……..
……..
}
```

Display State (The paint() method): The paint() method is used for applet display on the screen. The display includes text, images, graphics and background. This happens immediately after the applet enters into the running state. Almost every applet will have a paint() method and can be called several times during an applet's life cycle. The paint() method is called whenever a window is required to paint or repaint the applet.

```
public void paint(Graphics g)
{
……..
……..
}
```

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.

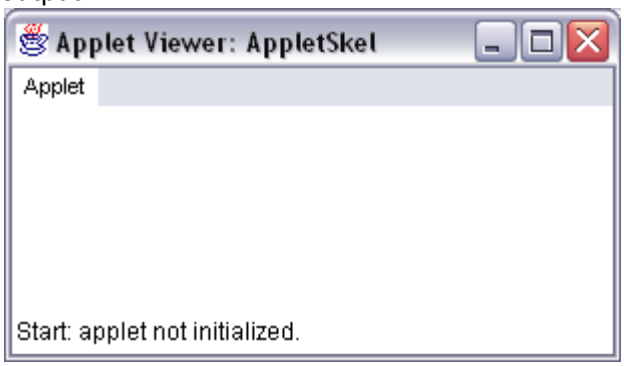
An Applet Skeleton:

Applets override a set of methods that provides the basic mechanism by which the browser or applet viewer interfaces to the applet and controls its execution. Four of these methods - init (), start (), stop () and destroy ()-are defined by Applet. Paint() method is defined by the AWT component class. Default implementation for all of these methods is provided. Applets do not need to override those methods they do not use. But simple applets will not need to define all of them. These five methods can be assembled into the skeleton as shown below:

```
// 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
}
}
```

The skeleton does not do anything; it can be compiled and run. When run, it generates the following output.



Applet Tag:

The Applet tag of html is used to start an applet either from a web browser or an applet viewer. The HTML tag allows a java applet to be embedded in an HTML document. We used the <APPLET> tag to create a space of the required size and then display the applet output in that space. A number of optional attributes can be used to control an applet's appearance.

The complete syntax of the <APPLET> tag is given below are:

```
<APPLET
        [ CODEBASE= codebase_URL]
        CODE = AppletFileName.class
        [ ALT = alternate_text]
        [ NAME = applet_instance_name]
        WIDTH = pixels
        HEIGHT = pixels
        [ ALIGN = alignment]
        [ VSPACE = pixels]
        [HSPACE = pixels ]
>
[ < PARAM NAME = name1 VALUE = value1> ]
[ < PARAM NAME = name2 VALUE = value2> ]
…….
…….
[Text to be displayed in the absence of java]
</APPLET>
```

List of Attributes and their Meanings are:

Attributes	Meaning
CODE = AppletFileName.class	Specifies the name of the applet class to be loaded. I.e. the name of the already compiled .class file in which the executable java byte code for the applet is stored. This attributes must be specified. The extension in the filename is optional.
CODEBASE= codebase_URL (Optional)	Specifies the URL of the directory that contains the applet class file.
WIDTH = pixels HEIGHT = pixels	These attributes specify the width and height (size in pixels) of the space on the HTML page that will be reserved for the applet.
NAME = applet_instance_name (Optional)	It is an **optional** attribute .It is used to specify a name for the applet instance. Applet must be named in order for other applets on the same page to find them by name and communicate with them. To obtain an applet by name use getApplet() method which is defined by the AppletContext interface.
ALIGN = alignment (Optional)	It is an **optional** attribute. It specifies the alignment of the applet. This attribute is treated the same as the HTML Img tag. Possible values for alignment are: top,

Attributes	Meaning
	bottom, left, right, middle, absmiddle, absbottom, texttop, and baseline
HSPACE = pixels (Optional)	Used only when align is set to left or right, this attributes specifies the amount of horizontal blank space the browser should leave surrounding the applet.
VSPACE = pixels (Optional)	Used only when some vertical alignment is specified with the align attribute (top, bottom etc) VSPACE specifies the amount of vertical blank space the browser should leave surrounding the applet.
ALT = alternate_text (Optional)	It is an optional attribute .It is used to specify a short text message that should be displayed if the browser recognizes the APPLET tag but can't currently run java applets. This is distinct from the alternate HTML we provide for browsers that don't support applets.

Passing Parameters to Applets:

The applet tag helps to define the parameters that are to be passed to the applets. We can supply user defined parameters to an applet using <PARAM….> Tag. Each <PARAM…> tag has a name attribute such as color, and a value attribute such as red. Inside the applet code the applet can refer to that parameter by name to find its value. E.g. we can change the color of the text displayed to red by an applet by using <PARAM…..> tag as follows:-

```
<APPLET….>
<PARAM = color VALUE=”red”>
</APPLET>
```

We can change the text to be displayed by an applet by supplying new text to the applet through a <PARAM..> tag

```
<PARAM NAME=text VALUE=”Hello Java”>
```

Example:

Code for java program

```
import java.applet.*;
import java.awt.*;
 public class appletParameter extends Applet
{
 private String strDefault = "Hello! Java Applet.";
 public void paint(Graphics g) {
 String strParameter = this.getParameter("Message");
 if (strParameter == null)
 strParameter = strDefault;
 g.drawString(strParameter, 50, 25);
 }
}
```

Code for HTML Program

```
<HTML>
<HEAD>
<TITLE>Passing Parameter in Java
Applet</TITLE>
</HEAD>
<BODY>
This is the applet:<P>
<APPLET code="appletParameter.class"
width="800" height="100">
<PARAM name="message" value="Welcome
in Passing parameter in java applet
example.">
</APPLET>
</BODY>
</HTML>
```

Compile the program: javac appletParameter.java
Output after running the program : To run the program using appletviewer, go to command prompt and type appletviewer appletParameter.html Appletviewer will run the applet for you and and it should show output like **Welcome in Passing parameter in java applet example.** Alternatively you can also run this example from your java enabled browser.

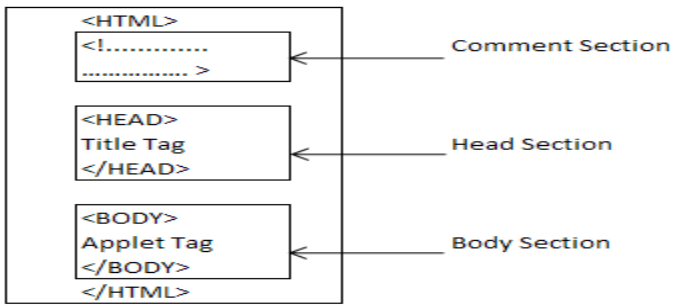
Aligning the Display:
We can align the output of the applet using the ALIGN attribute. This attribute can have one of the nine values: LEFT, RIGHT, TOP, TEXT TOP, MIDDLE, ABSMIDDLE, BASELINE, BOTTOM, ABSBOTTOM.

```
<HTML>
<HEAD> <TITLE> here is an applet </TITLE> </HEAD>
<BODY>
<APPLET
CODE=HelloJava.class
WIDTH = 400
HEIGHT = 200
ALIGN = RIGHT >
</APPLET> </BODY></HTML>
```

Designing a Web Page

Web page is basically design using text and HTML tag that can be interpreted by web browser or an applet viewer. A Web page is also known as HTML page. Web pages are stored using the file extension .html such as Test.html. Web pages are opened using the <HTML> tag and closed using </HTML> tag. HTML tag is divided into three sections:

- **Comment Section:** This section contains comments about the web pages. A comment line begins with a <! And ends with a >. Web browser will ignore the text enclosed between them. The comments are optional and can be included anywhere in the web page.
- **Head Section:** Head Section is starting with <HEAD> tag and ending with </HEAD> tag. This section contains a title for the web pages. For example
  <HEAD> <TITLE> Welcome to applet programming </TITLE> </HEAD>
  The TITLE tag is used to give the title for the web page, starting with <TITLE> and ending with </TITLE> tag. The head section is also optional.
- **Body Section:** This section contains the entire information about the web page and its behavior. The <body> tag is used to start the <BODY> element and the </BODY> tag ends it. It is used to divide a web page within one or more sections. The body of the document contains all that can be seen when the user loads the page.



Structure of HTML

HTML Tags:

Tag	Function
<HTML> …. </HTML>	Signifies the beginning and end of HTML file.
<HEAD> …. </HEAD>	This tag may include details about the web page.
<TITLE> …. </TITLE>	The text contained in it will appear in the title bar of the browser.
<BODY> …. </BODY>	This tag contains the main text of the web page. It is the place where the <APPLET> tag is declared.
<H1> …. </H1>	Header tag. Used to display headings. <H1> creates the largest font header.
<H6> …. </H6>	Created the smallest font header.
<CENTER> …. </CENTER>	Places the text contained in it at the center of the page.
<APPLET ….>	<APPLET..> tag declares the applet details as its attributes.
<APPLET> …. </APPLET>	May hold optionally user defined parameters using <PARAM> tag.
<PARAM ….>	Supplies user defined parameters.
<B> … <B>	Text between these tags will be displayed in bold type.
 	Lines break tag.
<P>	Paragraph tag.
<IMG …>	This tag declares attributes of an image to be displayed.
<HR>	Draws a horizontal rule.
<A…> </A>	Anchor tag used to add hyperlinks.
<FONT…> … </FONT>	We can change the color and size of the text that line in between <FONT> and </FONT> tags using color an size attributes.
<!.....>	Any text starting with a <! Mark and ending with a > mark is ignored by web browser. This tag is used as comment.

Adding controls to applet

Displaying Numerical Values:

We can display numerical values by first converting them into strings and then using the drawstring() method of Graphics class. We can do this easily by calling the ValueOf() method of String class.

Example:

Code for java program

```
import java.awt.*;
import java.applet.*;
public class NumValues extends Applet
{
public void paint (Graphics g)
{
int value1=10;
int value2=20;
int sum=value1 + value2;
String s=”sum” + String.valueOf(sum);
g.drawString(s, 100, 100);
}
}
```

Code for HTML program

```
<html>
<applet
code = NumValues.class
width=300
height=300 >
</applet>
</html>
```

Output:

Getting input from the User:

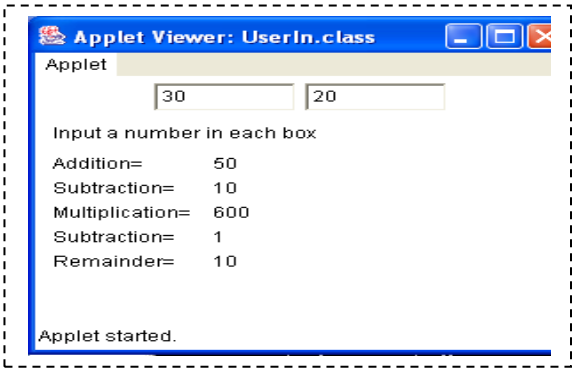
Applets treat inputs as text strings. We must first create an area of the screen in which user can type and edit input items. We can do this by using the TextField class of the applet package.

Example:

```
import java.awt.*;
import java.applet.*;
import java.lang.*;
public class UserIn extends Applet
{
TextField text1, text2;
public void init()
{
        text1=new TextField(8);
        text2=new TextField(8);
        add(text1);
        add(text2);
        text1.setText("0");
        text2.setText("0");
}
public void paint (Graphics g)
{
        int x=0, y=0, a=0;
        String s1, s2, s;
        g.drawString("Input a number in each box",10,50);
        try
        {
        s1=text1.getText();
        x=Integer.parseInt(s1);
        s2=text2.getText();
        y=Integer.parseInt(s2);
        }
        catch (Exception ex) { }
        a=x + y;
        s=String.valueOf(a);
        g.drawString("Addition=",10,75);
        g.drawString(s,100,75);
        a=x - y;
        s=String.valueOf(a);
```

```
        g.drawString("Subtraction=",10,95);
        g.drawString(s,100,95);
        a=x * y;
        s=String.valueOf(a);
        g.drawString("Multiplication=",10,115);
        g.drawString(s,100,115);
        a=x / y;
        s=String.valueOf(a);
        g.drawString("Subtraction=",10,135);
        g.drawString(s,100,135);
        a=x % y;
        s=String.valueOf(a);
        g.drawString("Remainder=",10,155);
        g.drawString(s,100,155);

}
public boolean action(Event event, Object object)
{
        repaint();
        return true;
}
}
```

output:



Run the applet UserIn using following steps:

- Type and save the program (.java file)
- Compile the applet (.class file)
- Write a HTML document (.html file)

```
<html>
<applet  code=UserIn.class  width=300 height=200 >
</applet>
</html>
```

- Use the appletviewer to display the result

Update() method:

Update() method is called when applet request to redraw a portion of window. By default **update()** fills the drawable area of a Component with its background color, and then sends **paint()** to the object. Thus, *flicker* that comes from redrawing the background over and over, can sometimes be fixed by overriding **update().**

```
Public void update(Graphics g)
{
……
Redraw window
…….
}
Public void paint(Graphics g)
{
Update(g);
}
```

Repaint() method:

Applets writes to its window only when its update () or paint () method is called by the AWT. Whenever applet needs to update the information displayed in its window it simply calls repaint () method. The repaint () method is defined by the AWT. It causes the AWT run-time system to execute a call to your Applet's update() method, which calls paint() method as default implementation. Therefore, to output to applet window first store the output then call the repaint() method.

The repaint() method has following four forms:

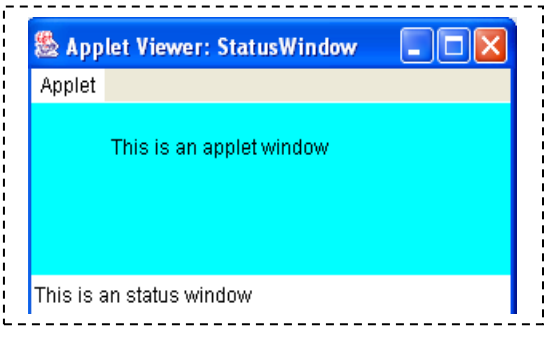
Repaint() mehods	Description
void repaint()	It causes the entire window to be repainted.
void repaint(int left,int top,int width,int height)	This method specifies the region that will be repainted. The upper-left co-ordinates are specified in left and top. The width and height also passed in last two parameters.
void repaint(long maxDelay)	Here maxDelay is used to provide the delay. It is a maximum number of milliseconds that can elapse before paint() method is called.
void repaint(long maxDelay,int x,int y,int width,int height)	Here also maxDelay is used to provide the delay in milliseconds. The dimension of portion of the window to be repainted is given in top, left, width and height parameter in pixels.

Status window:

An applet can also output a message to the status window of the browser or applet viewer on which it is running. To do so call showStatus() with the string that we want displayed. The status window is a good place to give the user feedback about what is occurring in the applet, suggest options, or possibly report some types of errors. The status window also makes an excellent debugging aid because it gives us an easy way to output information about our applet.

Program to display status window:

```
import java.awt.*;
import java.applet.*;
 /*  <applet  code="StatusWindow"  width=300 height=100>
    </applet>  */
public class   StatusWindow  extends Applet
{
public void init()
{
setBackground (Color.cyan);
}
public void paint (Graphics g)
{
g.drawString("This is an applet window",50,30);
showStatus("This is an status window");
}
}
```



getDocumentBase() and getCodeBase():

getDocumentBase():It allows the applet to load data from the directory holding the HTML file that started the applet .
getCodeBase(): It allows the applet to load the directory from which the applet's class file was loaded. These directories are returned as URL objects by getDocumentBase() and getCodeBase() .
They can be concatenated with a string that names the file we want to load. showDocument() method is used to actually load another file.

Program for use of getDocumentBase() and getCodeBase() method.

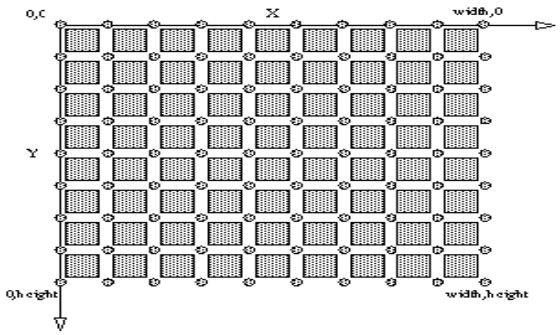
```
import java.awt.*;
import java.applet.*;
import java.net.*;
/*<applet  code="Bases"  width=300 height=100>
</applet>  */
public class  Bases extends Applet
{
public void paint (Graphics g)
{
String msg;
URL url =getCodeBase();
msg="Code base:"+url.toString();
g.drawString (msg,10,20);
url=  getDocumentBase() ;
msg ="Document base:"+url.toString();
g.drawString (msg,10,40);
}
}
```

Java – Graphics Programming

The Applet of an appletviewer is a browser on which the images are displayed. Every applet has its own area of the screen known as **canvas,** where it creates its display. The size of an applet's space is decided by the attributes of the <APPLET…> tag. The applet uses a technique of raster scan and is a screen resolved into the pixel format to view the images. A java applet draws graphical image inside its space using the coordinate system. Java's coordinate system has the origin (0,0) in the upper left corner. Positive x values are to the right and positive y values are to the bottom. The values of coordinates x and y are in pixels.

The Graphics Coordinate System:

The graphics coordinate system is anchored in the upper left-hand corner of a component, with coordinates increasing down and to the right, as depicted in Graphics Coordinate System. Coordinates lie between pixels of the output device. Operations that draw outlines of shapes, such as `Graphics.drawRect()`, traverse a coordinate path with a pen that hangs beneath and to the right of the path. The size of the pen is always one pixel wide and one pixel high.



Graphics co-ordinate System

Circles represent coordinates, and square represent pixels. Coordinates lie between pixels.

The Graphics Class

The Graphics class contains drawing methods. A Graphics object is passed to your paintComponent(Graphics g). The graphics toolbox within the Abstract Windowing Toolkit (or AWT) makes it possible for a Java programmer to draw simple geometric shapes, lines, print text, and position images within the borders of a component, such as a frame, panel, or canvas. In addition to performing graphical operations within a component, each Graphics also keeps track of the following graphical properties:

- The color used for drawing and filling shapes
- The font used for rendering text
- A clipping rectangle
- A graphical mode (XOR or Paint)
- A translation origin for rendering and clipping coordinates

Table shows the most commonly used drawing methods in the Graphics class.

Methods	Description
clearRect()	Erases a rectangular area of the canvas.
copyArea()	Copies a rectangular area of the canvas to another area.
drawArc()	Draws a hollow arc.
drawLine()	Draws a straight line.
drawOval()	Draws a hollow oval.
drawPolygon()	Draws a hollow polygon.
drawRect()	Draws a hollow rectangle.
drawRoundRect()	Draws a hollow rectangle with rounded corners.
drawstring()	Displays a text string.
fillArc()	Draws a filled arc.
fillOval()	Draws a filled oval.
fillPolygon()	Draws a filled polygon.
fillRect()	Draws a filled rectangle.
fillRoundRect()	Draws a filled rectangle with rounded corners.
getColor()	Retrieves the current drawing color.
getFont()	Retrieves the currently used font.
getFontMetrics()	Retrieves information about the current font.
setColor()	Sets the drawing color.
setFont()	Sets the font.

Graphics Methods:

1. Display Text: drawstring() method:

The drawString() method, takes as parameters an instance of the String class containing the text to be drawn, and two integer values specifying the coordinates where the text should start.

Syntax:drawstring("text",x,y);

Example:

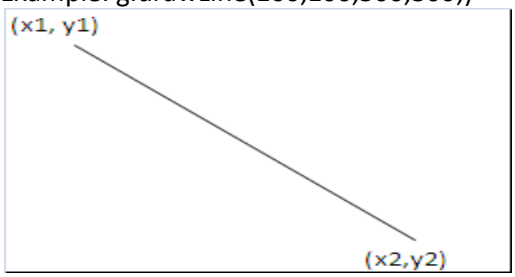
```
import java.awt.*;
import java.applet.*;
public class stringd extends Applet
{
public void paint(Graphics g)
{
g.drawString("Java Programming", 50, 50);
}
}
/*<applet code="stringd.class" width=200 height=100>
</applet>
*/
```

2. Line drawing:

The **drawLine()** method is used to draw line which take two pair of coordinates, (x1,y1) and (x2,y2) as arguments and draws a line between them. The graphics object g is passed to paint() method.

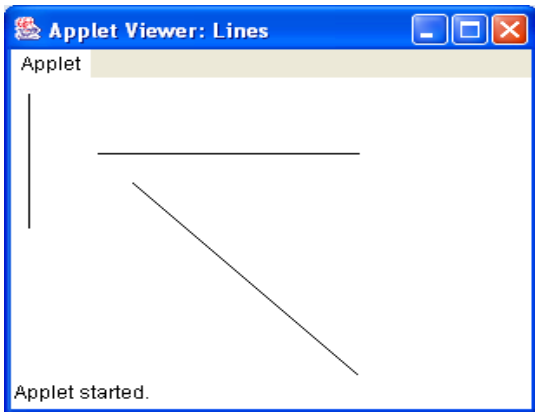
Syntax: g.drawLine(x1,y1,x2,y2);

Example: g.drawLine(100,100,300,300;)



Program to drawing lines

```
import java.awt.*;
import java.applet.*;
/* <applet code="Lines" width=300 height=200>  </applet> */
public class Lines extends Applet
{
public void paint(Graphics g)
 {
g.drawLine(10, 10, 10, 100);
g.drawLine(50, 50, 200, 50);
g.drawLine(70, 70, 250, 250);
}
}
```



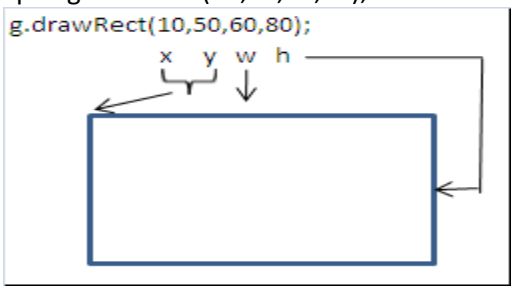
3. Drawing Rectangle:

The **drawRect()** method draws the Rectangle.

Syntax: drawRect(x,y,w,h);

It takes four arguments: first two x and y coordinates represents top left corner of rectangle and remaining two represents the width and height of rectangle in pixels.

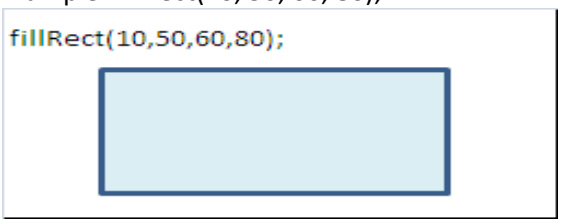
Example: g.drawRect(10,50,60,80);



fillRect(): The method draws a filled rectangle from origin (x,y) and width w and height h.

Syntax: fillRect(x,y,w,h);

Example: fillRect(10, 50, 60, 80);



clearRect(): This function clears the drawn rectangle from the screen.

Syntax: clearRect(x,y,w,h);

Example:

```
public class clear extends Applet
{
Public void paint(Graphics g)
{
```

```
g.drawRect(10,15,100,100);
g.fillRect(150,15,100,100);
g.clearRect(10,15,102,102);
}
}
```

Drawing of 3-D rectangle:

draw3DRect(): This function draws a view of 3-D rectangle at position(x,y) with width w, height h and Boolean variable to indicate raised rectangle.

Syntax: draw3DRect(x,y,w,h,b);

Example: draw3DRect(10,10,100,100,true);

fill3DRect(): This function draws a view of 3-D filled rectangle at position (x,y) with width w, height h and Boolean variable to indicate raised rectangle.

Syntax: fill3DRect(x,y,w,h,b);

Example: fill3DRect(10,10,100,100,true);

4. Drawing of Rounded Rectangle:

The **drawRoundRect()** and **fillRoundRect()** methods are used to draw and fill rounded rectangle. The rounded rectangle is a rectangle having corners being rounded. The rounded rectangle also produces an ellipse as well as circle. Incase if the width and height of rectangle is same then a founded square is produced.

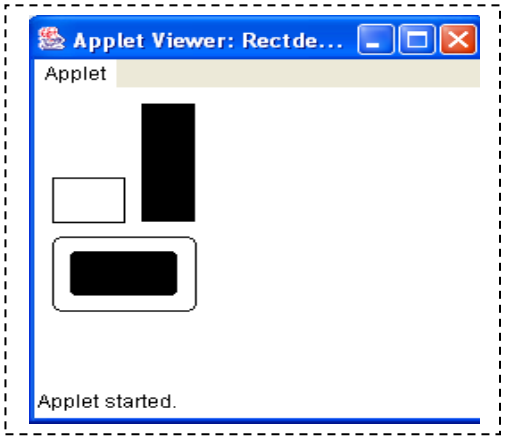
Syntax: drawRoundRect(x,y,w,h,ac,ah);

First four parameters are similar to drawRect() method. The last two ac and ah represents the width and height of angle of corners. The extra parameters indicates how much of corners will be rounded.

Example: drawRoundRect(10, 100, 80, 50, 10, 10);
fillRoundRect(20, 110, 60, 30, 5, 5);

Program for Drawing Lines and Rectangle

```
import java.awt.*;
import java.applet.*;
public class Rectdemo extends Applet
{
public void paint(Graphics g)
{
g.drawRect(10, 60, 40, 30);
g.fillRect(60, 10, 30, 80);
g.drawRoundRect(10, 100, 80, 50, 10, 10);
g.fillRoundRect(20, 110, 60, 30, 5, 5);
}
}
/*<applet code="Rectdemo.class" width=250 height=200 >
</applet>*/
```



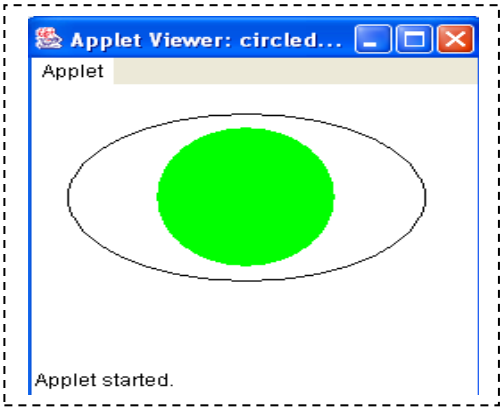
5. Circles and Ellipses:

The Graphics class does not have any method for circles or ellipses. The **drawOval()** method can be used to draw a circle or an ellipse. The drawOval() method takes four arguments; the first two represent the

top left corner of the imaginary rectangle and the other two represent the width and height of the oval itself. If the width and height are the same the oval becomes a circle.
The **fillOval()** method draws a filled circle or ellipse at the specified location with width w and height h.
Syntax:
drawOval(x,y,w,h);
fillOval(x,y,w,h);

Program to print oval and filled circle

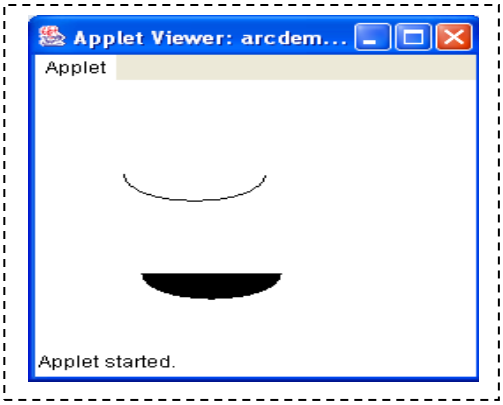
```
import java.awt.*;
import java.applet.*;
public class circledemo extends Applet
{
public void paint(Graphics g)
{
g.drawOval(20, 20, 200, 120);
g.setColor(Color.green);
g.fillOval(70, 30, 100, 100);        // this is a circle
}
}
/*<applet code="circledemo.class" width=250 height=200>
</applet>
*/
```



6. Drawing Arcs:
The **drawArc()** method is used to draw arcs which takes six arguments, the first four are the same as the arguments for drawOval() method and the last two represent the starting angle of the arc and the number of degrees (sweep angle) around the arc.
Syntax: drawArc(x,y,w,h,sa,aa);
Example: drawArc(60,125,80,40,180,180);
Similarly the **fillArc()** method draws a filled arc at location x,y of height h, width w, start angle s and arc angle aa.
Syntax: fillArc(x,y,w,h,sa,aa);
Example: g.fillArc(60, 125, 80, 40, 180, 180);

Program to print arc and fill arc

```
import java.awt.*;
import java.applet.*;
public class arcdemo extends Applet
{
public void paint(Graphics g)
{
g.drawArc(50, 50, 80, 40, 180, 180);
g.fillArc(60, 125, 80, 40, 180, 180);
}
}
/*<applet code="arcdemo.class" width=250 height=200>
</applet>
*/
```



7. Drawing Polygons:

A polygon is a closed path or circuit which is made by joining line segments. In polygon, each line segment intersects exactly two others line segment i.e. the end of the first line is the beginning of the second line, the end of the second is the beginning of the third and so on. A polygon may be considered a set of lines connected together.

drawPolygon() method of Graphics class takes three arguments.

- An array of integers containing x coordinates
- An array of integers containing y coordinates
- An integer for the total number of points.

Syntax: drawPolygon(xp,yp,n)

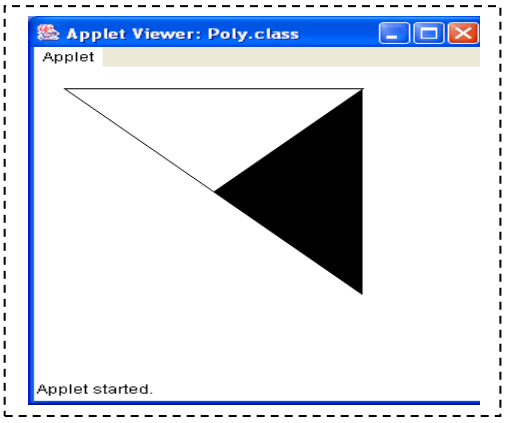
This function draws polygon by joining x and y co-ordinates of the 'n' number of points stored in the xp and yp array.

fillPolygon() function draws a fill polygon by joining x and y coordinates of n points stored in xp, yp array.

Syntax: fillPolygon(xp,yp,n)

Program to draw polygon and fill polygon

```
import java.awt.*;
import java.applet.*;
public class Poly extends Applet
{
int x1[]={20, 120, 220, 20};
int y1[]={20, 120, 20, 20};
int n1=4;
int x2[]={120, 220, 220, 120};
int y2[]={120, 20, 220, 120};
int n2=4;
public void paint(Graphics g)
{
g.drawPolygon(x1, y1, n1);
g.fillPolygon(x2, y2, n2);
}
}
/*<applet code="Poly.class" width=300 height=300>
</applet>
*/
```



Using control loops in applets:

Probably the most often-used loop in programming is the `for` loop, which instructs a program to perform a block of code a specified number of times. All three sections of the `for` loop, which are separated by semicolons. The initialization section of the `for` statement is used to initialize the loop-control variable that controls the action. The condition section represents a Boolean condition that should be equal to true for the loop to continue execution. Finally, the increment, which is the third part of the statement, is an expression describing how to increment the control variable. The statement after the `for` statement is executed each time the loop's conditional expression is found to be true.

Syntax:

for(initialization; condition_check; increment/decrement)
{ …….}
Example:

Applet Program to enter any String in the text box and display string 10 times.

```
import java.awt.*;
import java.applet.*;
public class Appletloop extends Applet
{
TextField textField1;
public void init()
{
textField1 = new TextField(20);
add(textField1);
textField1.setText("Java Programming");
}

public void paint(Graphics g)
{
g.drawString("Enter a name above.", 70, 45);
String s = textField1.getText();
for (int x=0; x<10; ++x)
g.drawString(s, 80, x * 15 + 70);
}
public boolean action(Event event, Object arg)
{
repaint();
return true;
}
}

/* <applet code=Appletloop.class height=200 width=200>
 </applet> */
```

Output:

Drawing Bar Charts:

Applets can be designed to display bar charts, which are commonly used in analysis of data.
The table shows the result analysis of the subject during the period 2001 to 2004. These values may be placed in a HTML file as PARAM attributes and then used in an applet for displaying a bar chart.
The method getParameter() is used to fetch the data values from values from the HTML file.

Year	2001	2002	2003	2004
% result of subject	96	85	70	98

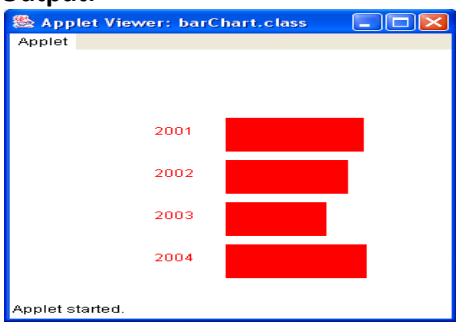
Program to draw bar chart of result analysis

```
import java.awt.*;
import java.applet.*;
public class barChart extends Applet
{
String label[];
int  values[];
int n=0;
public void init()
{
try
{
n=Integer.parseInt(getParameter("cols"));
label=new String[n];
values=new int[n];
label[0]=getParameter("label1");
label[1]=getParameter("label2");
label[2]=getParameter("label3");
label[3]=getParameter("label4");
values[0]=Integer.parseInt(getParameter("values1"));
values[1]=Integer.parseInt(getParameter("values2"));
values[2]=Integer.parseInt(getParameter("values3"));
values[3]=Integer.parseInt(getParameter("values4"));
}
catch(NumberFormatException e){  }
}
public void paint(Graphics g)
{
g.setColor(Color.red);
for(int i=0;i<n;i++)
{
g.drawString(label[i],100,100+i*50);
g.fillRect(150,80+i*50,values[i],40);
}
}
}
```

Code for html file

```
<html>
<applet code="barChart.class" height=700 width=700>
        <param name="cols" value="4">

        <param name="label1" value="2001">
        <param name="label2" value="2002">
        <param name="label3" value="2003">
        <param name="label4" value="2004">

        <param name="values1" value="96">
        <param name="values2" value="85">
        <param name="values3" value="70">
        <param name="values4" value="98">
   </applet>
</html>
```

Output:



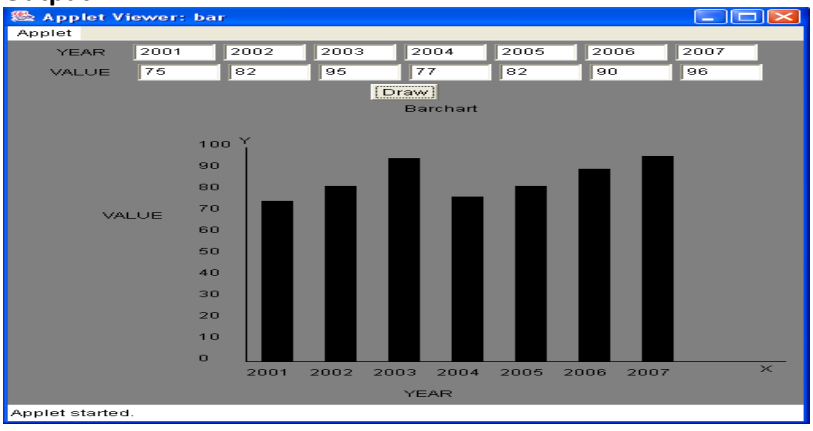
Program to display vertical bar chart

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.lang.*;
/*
<applet code="bar" width=450 height=700></applet>*/
public class bar extends Applet implements ActionListener
{
Label l1,l2;
Button b1;
TextField t1,t2,t3,t4,t5,t6,t7;
TextField tf1,tf2,tf3,tf4,tf5,tf6,tf7;
public void init()
{
setBackground(Color.gray);
l1=new Label("YEAR");
l2=new Label("VALUE");
t1=new TextField(4);
t2=new TextField(4);
```

```
t3=new TextField(4);
t4=new TextField(4);
t5=new TextField(4);
t6=new TextField(4);
t7=new TextField(4);

tf1=new TextField(4);
tf2=new TextField(4);
tf3=new TextField(4);
tf4=new TextField(4);
tf5=new TextField(4);
tf6=new TextField(4);
tf7=new TextField(4);

b1=new Button("Draw");
add(l1);
add(t1); add(t2); add(t3); add(t4); add(t5); add(t6); add(t7);
add(l2);
add(tf1);add(tf2);add(tf3);add(tf4);add(tf5);add(tf6);add(tf7);
add(b1);
tf1.reshape(100,200,20,30);
b1.addActionListener(this);
}
public void actionPerformed(ActionEvent ae)
{
repaint();
}
public void paint(Graphics g)
{
String year[]=new String[7];
String unit[]={"0","10","20","30","40","50","60","70","80","90","100"};
int val=160,val1=0,yval=0;
int value[]=new int[7];

year[0]=t1.getText();
value[0]=Integer.parseInt(tf1.getText());

year[1]=t2.getText();
value[1]=Integer.parseInt(tf2.getText());

year[2]=t3.getText();
value[2]=Integer.parseInt(tf3.getText());

year[3]=t4.getText();
value[3]=Integer.parseInt(tf4.getText());
```

```
year[4]=t5.getText();
value[4]=Integer.parseInt(tf5.getText());

year[5]=t6.getText();
value[5]=Integer.parseInt(tf6.getText());
g.setColor(Color.green);

year[6]=t7.getText();
value[6]=Integer.parseInt(tf7.getText());

g.setColor(Color.black);
g.drawString("Barchart",250,100);
g.drawLine(150,450,490,450);
g.drawLine(150,150,150,450);
yval=450;
for(int i=0;i<=10;i++)
{
g.drawString(unit[i],120,yval);
yval-=30;
}
val1=0;
for (int i=0;i<7;i++)
{
val1=value[i]*3;
value[i]=450-(value[i]*3);
g.fillRect(val,value[i],20,val1);
g.drawString(year[i],val-10,470);
String s=value[i]+"%";
g.drawString("YEAR",250,500);
g.drawString("X",475,465);
g.drawString("VALUE",60,250);
g.drawString("Y",146,145);
val=val+40;
} } }
```

Output:



Line Graph

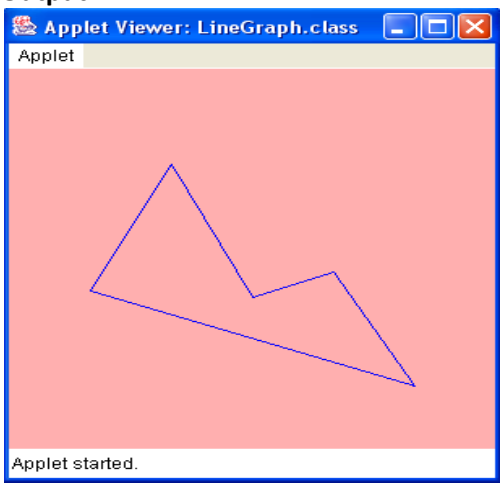
A line graph represents the relationship between two variables graphically. We can implement this relationship using the drawPolygon() method of the Graphics class. Drawing line graph for the following table.

X	50	100	150	200	250
Y	175	75	180	160	250

Applet program to draw line graph

```
/*<applet code="LineGraph.class" width="300" height="300"> </applet>*/
import java.applet.*;
import java.awt.*;
public class LineGraph extends Applet
{
int x[]={50,100,150,200,250};
int y[]={175,75,180,160,250};
public void paint(Graphics g)
{
setBackground(Color.pink);
g.setColor(Color.blue);
g.drawPolygon(x,y,5);
}
}
```

Output:



The class color:

Colors are represented in java as instances of the class color. A color object is created with a specific RGB value. It can then be used to set the color for painting by passing it as parameter to the setColor() method of the current graphics object. Java supports color in a portable, device-independent fashion. The AWT color system allows us to specify any color that we want. It then finds the best match for that color, given the limits of the display hardware currently executing our program or applet. Thus, our code does not need to be concerned with the differences in the way color is supported by various hardware

devices. You can fine tune your colors by controlling the RGB (red/green/blue) density values. Each of these 3 colors has 256 shades. It is possible to "mix" a new shade of color by selecting an integer from 0 to 255 and passing it to a Color constructor. The value 0 indicates the absence of a color in the mixture, and the value 255 indicates the maximum saturation of that color. The color black has RGB(0,0,0) and the color white has RGB(255,255,255). There are 256 * 256 *256 = 2^{24} possible colors in this scheme. Color is encapsulated by the Color class. **Color** defines several constants (for example, **Color.black**) to specify a number of common colors. You can also create your own colors, using one of the color constructors:

- Color (int Red, int Green, int Blue)= Creates a color with the designated combination of red, green, and blue (each 0-255).
- Color (float Red, float Green, float Blue)= Creates a color with the designated combination of red, green, and blue (each 0-1).
- Color (int rgb)= Creates a color with the designated combination of red, green, and blue (each 0-255).

Example:

```
Public void paint(Graphics g)
{
//create new color object “brown” using the color(int red, int green, int blue) constructor of the color class
color brown=new color(107, 69,38);
g.setColor(brown);
g.drawString(“Brown”,50,50);
}
```

Colors are completely specified by RGB values:
Red=(255,0,0), green=(0,255,0), blue=(0,0,255), yellow(red+green)=(255,255,0), magenta(red+blue)=(255,0,255), cyan(green+blue)=(0,255,255), white(red+green+blue)=(255,255,255), grey=(128,128,128) etc.

Setting Background and Foreground color:

setBackground() and getBackground(): You can set/get the background color of any chunk or an applet with the help of *setBackground(**Color** color) andgetBackground() method*. SystemColor class provides access to desktop colors.
Example:

```
setBackground(Color.red);
setBackground(Color.blue)
```

similarly you can set/get the foreground color of an applet using setForeground() and getForeground() methods. Best way to set the background and foreground colors is in the init() method. You can change these colors if necessary during the execution of Applet. The default foreground color is black and background color is light gray.
setBackground( ) method: It is used to set background color of an applet window.
Syntax: void setBackground(Color newcolor)
setForeground( ) method: It is used to set foreground color of an applet window.
Syntax: void setForeground(Color newcolor)

getBackground( ) method: It is used to get the current background color of an applet window.

Syntax: Color getBackground( )

getForeground( ) method: It is used to get the current foreground color of an applet window.

Syntax: Color getForeground( )

The following constants defined in color class are:

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

Fonts class:

The java.awt.Font class is used to create Font objects to set the font for drawing text, labels, text fields, buttons, etc. In Java a font is an instance of a typeface that specifies the following information:

- The typeface used to display the font
- Special effects, such as italics or bolding
- The size of the font

There are three *logical/generic font names*. Java will select a font in the system that matches the general characteristics of the logical font.

serif	This text is in a serif font. Often used for blocks of text (eg, Times).
sansserif	This text is in a SansSerif font. Often used for titles (eg, Arial or Helvetica).
monospaced	This text is in a Monospaced font, often used for computer text (eg, Courier).

You can also get a list of the *system fonts* on the host computer.

Constructor: `Font f = new Font(name, style, size);`

String name	int style	int size
"serif"	Font.PLAIN	Integer point size -- typically in range 10-48.
"sansserif"	Font.BOLD	
"monospaced"	Font.ITALIC	
or a system font.	Font.BOLD+Font.ITALIC	

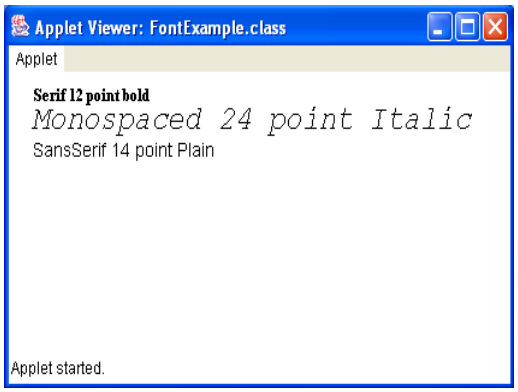
Example:

Font font = new Font("Arial",Font.BOLD,30);

Program demonstrate use of font.

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Font;
public class FontExample extends Applet
{
```

```
private Font font1, font2, font3;
public void init()
{
font1=new Font("Serif",Font.BOLD,12);
font2=new Font("Monospaced",Font.ITALIC,24);
font3=new Font("SansSerif",Font.PLAIN,14);
}
public void paint(Graphics g)
{
g.setFont(font1);
g.drawString("Serif 12 point bold",20,20);
g.setFont(font2);
g.drawString("Monospaced 24 point Italic",20,40);
g.setFont(font3);
g.drawString("SansSerif 14 point Plain",20,60);
}
}
/*<applet code="FontExample.class" height=200 width=200>
</applet> */
```



Field Summary

field	Description
Static Int bold	The bold style constant.
Static Int italic	The italicized style constant.
Static Int plain	The plain style constant.
Protected float pointSize	The point size of this font in float.
Protected String name	The logical name of this font, as passed to the constructor.
Protected int size	The point size of this font, rounded to integer.
Protected int style	The style of the font, as passed to the constructor.

Methods of font class

Method	Description
String getFamily()	Returns the family name of the font (for example, Helvetica).
Font getFont(String nm)	Returns a font from the system properties list.
Font getFont(String nm, Font font)	Returns the specified font from the system properties list.
String getName()	Returns the logical name of the font.
Int getSize()	Returns the point size of the font, rounded to integer.
Int getStyle()	Returns the style of the font.
Boolean isBold()	Indicates whether the font's style is bold.
Boolean isItalic()	Indicates whether the font's style is italic.
Boolean isPlain()	Indicates whether the font's style is plain.

Graphics Environment (java.awt.GraphicsEnvironment)

This class specifies the graphics environment. The resources in this environment might be local or on a remote machine. The graphics environment consists of a number of GraphicsDevice objects and Font objects. GraphicsDevice objects are typically screens or printers and are the destination of Graphics2D drawing methods. Each GraphicsDevice has a number of GraphicsConfiguration objects associated with it. These specify the different configurations in which the GraphicsDevice can be used.

Constructor: GraphicsEnvironment();

Methods of GraphicsEnvironment class

Method	Description
Abstract Font[] getAllFonts()	Returns an array containing a one-point size instance of all fonts available in this environment.
Abstract String[] getAvailableFontFamilyNames()	Returns an array containing the names of all font families available in this environment.
Abstract GraphicsDevice getDefaultScreenDevice()	Returns the default screen graphics device.
Static GraphicEnvironment getLocalGraphicsEnvironment()	Returns the local graphics environment.
Abstract GraphicsDevice[] getScreenDevices()	Returns an array of all of the screen devices.

Program to display all available font family

```
import java.awt.Font;
import java.awt.GraphicsEnvironment;
public class MainClass
{
public static void main(String[] args) throws Exception
{
Font[] fonts  = GraphicsEnvironment.getLocalGraphicsEnvironment().getAllFonts();
for (int i = 0; i < fonts.length; i++)
 {
System.out.print(fonts[i].getFontName() + " : ");
System.out.print(fonts[i].getFamily() + " : ");
System.out.print(fonts[i].getName());
```

```
System.out.println();
}
}
}
```

Drawing Images on the Applet

The AWT image class and the java.awt.image package support to display images and manipulate images on applet. There are three stages with images:

- Creating an image
- Loading an image
- Display an image

The following steps need to be implemented in order to display an image on the applet.

- Create an image object
- Load the image object with the image file, which must be displayed.
- Display the image object on the applet

Creating an image object:

An area in the memory can be reserved to hold an instance of an image object.
Syntax: Image <memvar name>
Example: Image img;
It creates an area in memory called img, which hold an instance of an image object.

Loading the image object with the image file

The image must be load in to memory space 'img'
The applet class provides a method called getImage() to load an image into an instance variable.
Syntax: img=getImage(<URL object>, <filename>);
The applet has two methods that can be used to create a base URL without providing a specific address.

- **getDocumentBase():** This method returns a URL that represent the folder containing the web page presenting the applet. For example if page is located at http://www.yahoo.com/images/, getDocumentBase() method returns a URL pointing to that path.
- **getCodeBase():** This method returns a URL object that represent the folder where the applets main class file is located.

Example:

Image img=getImage(getCodeBase(), "happy.gif");
Happy.img resides in the same directory where applet.class file is present.

Display the image object on the applet:

After image is loaded into an image object, the image can be displayed on the applet using the drawImage() method of the Graphics class. The drawImage method can be called within the paint() method.
Two way to display image:
1. To display image as its actual size
Syntax: drawImage(image_object, x, y, this);
2. To display the image at a different size

Syntax: drawImage(image_object, x, y, width, height, this);
The width and height arguments describe the width and height in pixels that the image should occupy when displayed.

Program to display image on applet

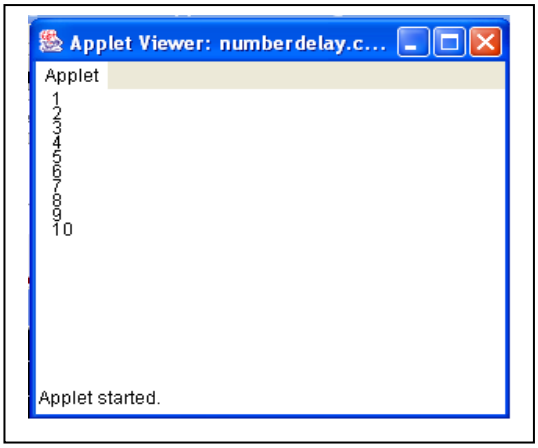
```
import java.awt.*;
import java.applet.*;
public class imagedemo extends Applet
{
Image img;
public void init()
{
img=getImage(getCodeBase(),"computer.jpg");
}
public void paint(Graphics g)
{
g.drawImage(img,10,20, this);
}
}

/*<applet code=imagedemo.class height=300 width=300>
</applet> */
```

Output:

Solved Programs

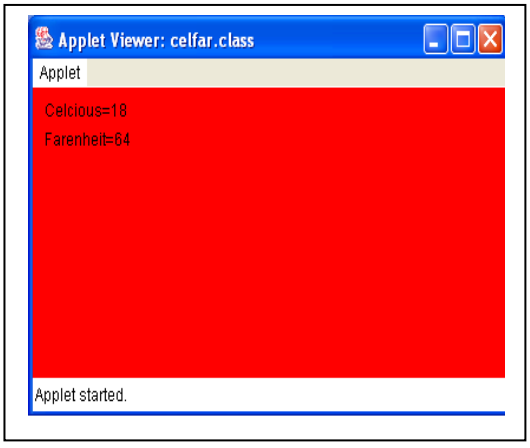
1. Program to display numbers from 1 to 10 on applet such that each number will be display after delay of 100 ms.

```
import java.lang.*;
import java.awt.*;
import java.applet.*;
public class numberdelay extends Applet implements Runnable
{
String msg;
int i,j;
Thread t=null;
public void init()
{
j=0;
msg=null;
}

public void start()
{
t=new Thread(this);
t.start();
}

public void run()
{
repaint();
}
public void paint(Graphics g)
{
for(int i=1;i<11;i++)
{
j=j+10;
try
{
msg=String.valueOf(i);
Thread.sleep(100);
}
catch(Exception e)
{
}
g.drawString(msg,10,j);
if(i==10)
j=0;
} } }
/*  <applet code=numberdelay.class height=200 width=300></applet>
*/
```



2. Program to accept the value of temperature in celcious using "param" tag and display the temperature in Fahrenheit on applet with red background.

```
/*
<applet code=celfar.class height=200 width=400>
<param name=Celcious value=18> </applet>
*/

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

public class celfar extends Applet
{
String cel,far;
int c,f;
public void init()
{ setBackground(Color.red);
}

public void start()
{
cel=getParameter("Celcious");
try
{
if(cel!=null)
c=Integer.parseInt(cel);
else
c=0;
f=c*9/5+32;
far=String.valueOf(f);
}

catch(Exception e)
{
}
}

public void paint(Graphics g)
{
g.drawString("Celcious="+cel,10,20);
g.drawString("Farenheit="+far,10,40);
}
}
```



3. Write a program to create an applet for displaying different shapes.

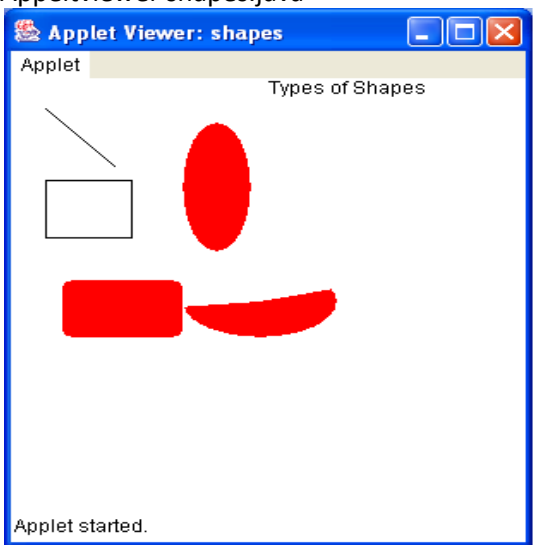
Program to display various shapes on Applet.

```
import java.awt.*;
import java.applet.*;
public class shapes extends Applet
{
public void paint(Graphics g)
{
g.setFont(new Font("Arial",10, 12));
g.drawString("Types of Shapes",150,10);
g.drawLine(20,20,60,60);
g.drawRect(20,70,50,40);
g.setColor(Color.red);
g.fillOval(100,30,40,90);
g.fillRoundRect(30,140,70,40,10,10);
g.fillArc(100,130,90,50,190,190);
}
}

/*<applet code=shapes height=300 width=300>
</applet>
*/
```

Output:

Javac shapes.java

Appeltviewer shapes.java



4. Write a Program to draw a Bar chart for the table given below which shows annual result analysis of a school from period 2001-2005. These values may be placed in a HTML file as <param> attributes and then used in Applet for displaying bar chart
OR
Write a program to draw a bar chart for plotting students passing percentage in last 5 years.

Year	2001	2002	2003	2004	2005
Result %	80	90	100	100	98

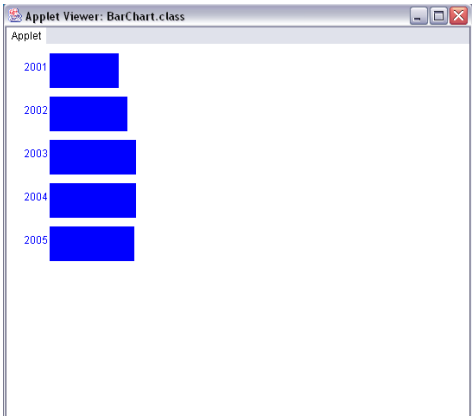
Java Source Code

```
import java.awt.*;
import java.applet.*;
public class BarChart extends Applet
{
int n = 0;
String label [ ];
int value [ ];
public void init()
{
try
{
n = Integer.parseInt(getParameter ("columns"));
label = new String [n];
value = new int[n];
label [0] =getParameter ("label1");
label [1] =getParameter ("label2");
label [2] =getParameter ("label3");
label [3] =getParameter ("label4");
label [4] =getParameter ("label5");
value [0] = Integer.parseInt(getParameter ("c1"));
value [1] = Integer.parseInt(getParameter ("c2"));
value [2] = Integer.parseInt(getParameter ("c3"));
value [3] = Integer.parseInt(getParameter ("c4"));
value [4] = Integer.parseInt(getParameter("c5"));
}
catch (NumberFormatException e)
{          }
}
public void paint(Graphics g)
{
for(int i = 0;i<n;i++)
{
g.setColor(Color.blue);
g.drawString(label [i], 20,i*50+30);
g.fillRect(50,i*50+10,value[i],40);
}
}
}
```

HTML Sourc Code :

```
<html>
<applet code ="BarChart.class" height = 100
width = 100>
<PARAM NAME ="columns" VALUE ="5">
<PARAM NAME ="c1" VALUE ="80">
<PARAM NAME ="c2" VALUE ="90">
<PARAM NAME ="c3" VALUE ="100">
<PARAM NAME ="c4" VALUE ="100">
<PARAM NAME ="c5" VALUE ="98">

<PARAM NAME ="label1" VALUE ="2001">
<PARAM NAME ="label2" VALUE ="2002">
<PARAM NAME ="label3" VALUE ="2003">
<PARAM NAME ="label4" VALUE ="2004">
<PARAM NAME ="label5" VALUE ="2005">
</applet>
</html>
```

Output:



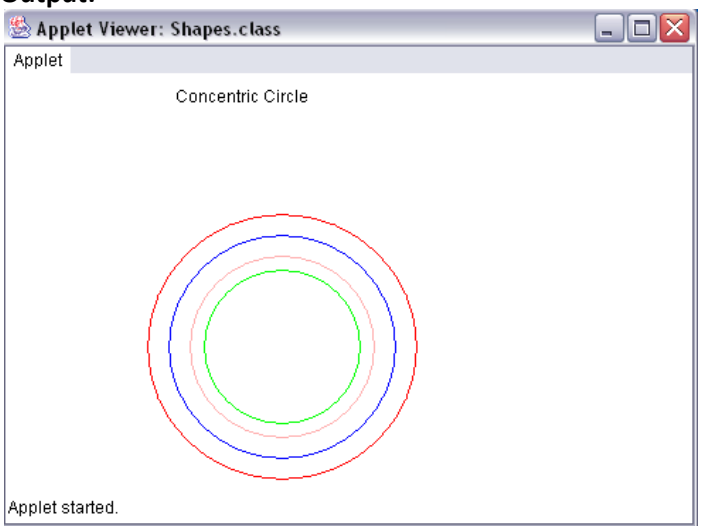
5. Program to display a string "Concentric Circles" using font "Times New Roman", size as 14 and style as bold+italic and display four concentric circles with different colors on the Applet.

```
import java.awt.*;
import java.applet.*;
public class Shapes extends Applet
{
public void paint(Graphics g)
{
g.setFont("Times New Roman",int BOLD+ITALIC,int 14);
g.drawString("Concentric Circle",120,20);
g.setColor(Color.red);
g.drawOval(100,100,190,190);
g.setColor(Color.blue);
g.drawOval(115,115,160,160);
g.setColor(Color.pink);
g.drawOval(130,130,130,130);
g.setColor(Color.green);
g.drawOval(140,140,110,110);
}
}
```

Html source

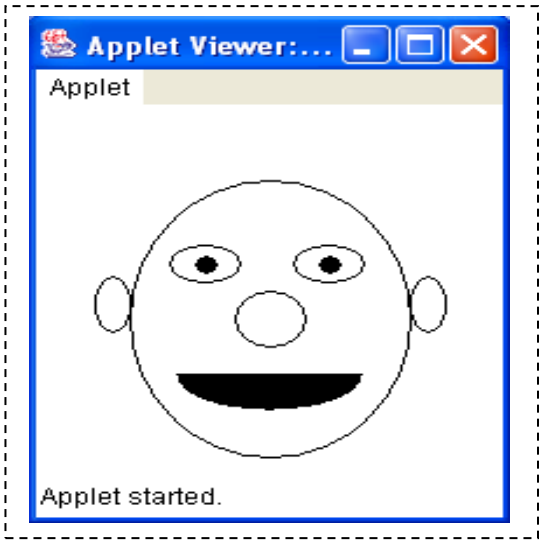
```
<html>
<applet code="Shapes.class" height=300 width = 200>
</applet>
</html>
```

Output:



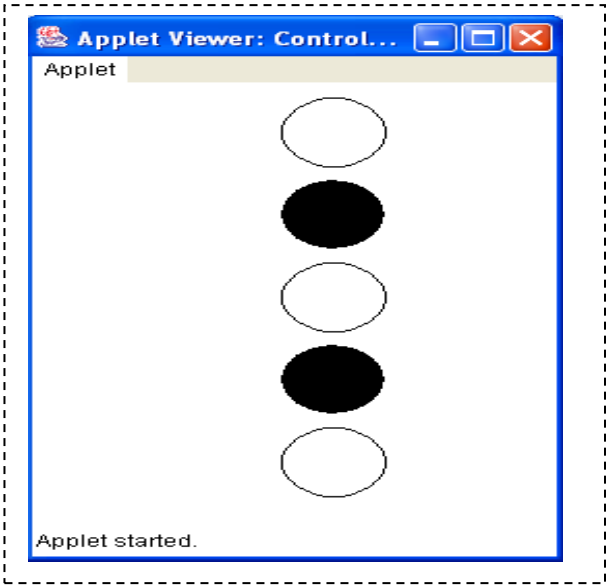
6. Program for drawing a human face

```
import java.awt.*;
import java.applet.*;
public class Face extends Applet
{
public void paint(Graphics g)
{
g.drawOval(40, 40, 120, 150);    //Head
g.drawOval(57, 75, 30, 20);      //Left Eye
g.drawOval(110, 75, 30, 20);     // Right Eye
g.fillOval(68, 81, 10, 10);      //Pupil(left)
g.fillOval(121, 81, 10, 10);     //pupil(right)
g.drawOval(85, 100, 30, 30);     //Nose
g.fillArc(60, 125, 80, 40, 180, 180);        //Mouth
g.drawOval(25, 92, 15, 30);      //left ear
g.drawOval(160, 92, 15, 30);     //right ear
}
}

/*
<applet code=Face.class height=200 width=200>
 </applet>
*/
```



7. Program Using Control Loops in Applets

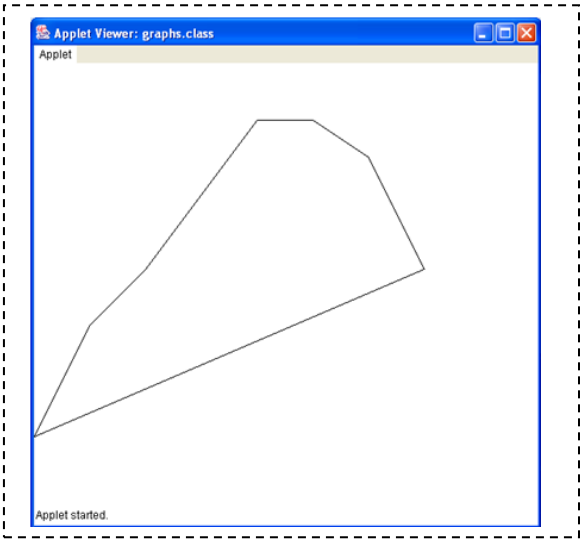
```
import java.awt.*;
import java.applet.*;
public class ControlLoop extends Applet
{
public void paint(Graphics g)
{
for (int i=0; i<=4; i++)
{
if (i%2 == 0)
g.drawOval(120, i*60+10, 50, 50);
else
g.fillOval(120, i*60+10, 50, 50);

}
}
}
/*
<applet code=ControlLoop.class height=200 width=200>
 </applet>
*/
```



8. Program to draw line graphs from the following table of values

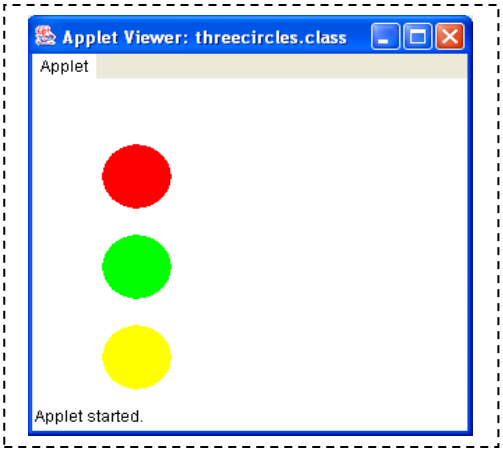
X	0	60	120	180	240	300	360	400
Y	400	280	220	140	60	60	100	220

```
import java.awt.*;
import java.applet.*;
public class graphs extends Applet
{
int x[]={0,60,120,180,240,300,360,420};
int y[]={400,280,220,140,60,60,100,220};
int n=x.length;
public void paint(Graphics g)
{
g.drawPolygon(x,y,n);
}
}
/*
<applet code=graphs.class height=150 width=100>
 </applet>
*/
```



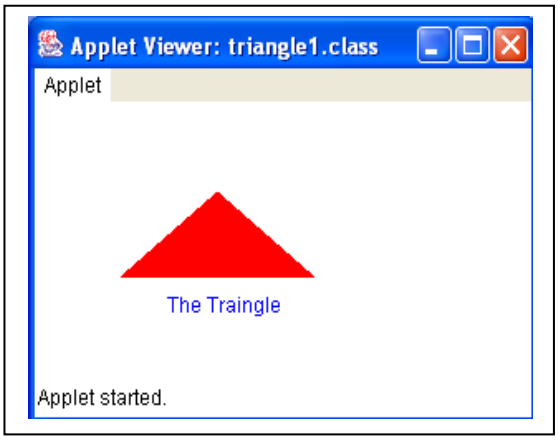
9. Design an applet which displays three circles one below the other and fill them with red, green and yellow color respectively.

```
import java.awt.*;
import java.applet.*;
public class threecircles extends Applet
{
public void paint(Graphics g)
{
g.setColor(Color.red);
g.fillOval(50,50,50,50);
g.setColor(Color.green);
g.fillOval(50,120,50,50);
g.setColor(Color.yellow);
g.fillOval(50,190,50,50);
}
}
/*
<applet code=threecircles.class height=200 width=80>
 </applet>
*/
```



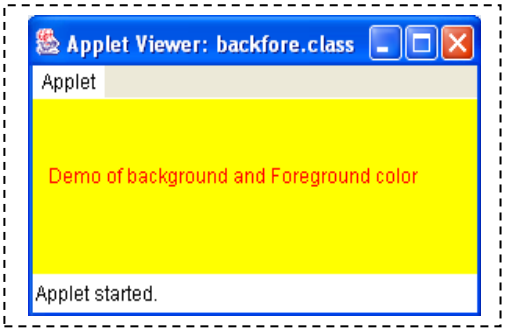
10. Design an applet which displays a triangle filled with red color and a message as 'The Traingle' in blue below it.

```
import java.awt.*;
import java.applet.*;
public class triangle1 extends Applet
{
int x1[]={110, 170, 50, 110};
int y1[]={50, 100, 100, 50};
int n1=4;

public void paint(Graphics g)
{
g.setColor(Color.red);
g.fillPolygon(x1, y1, n1);
g.setColor(Color.blue);
g.drawString("The Traingle",80,120);

}
}
/*<applet code="triangle1.class" width=300 height=300>
</applet>
*/
```



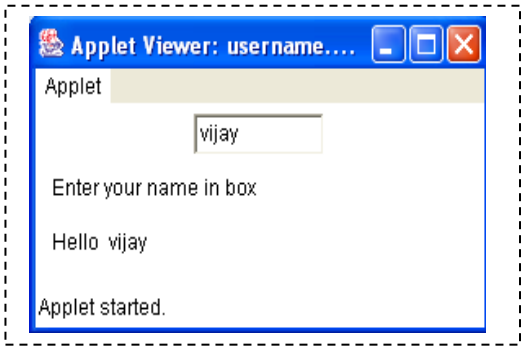
11. Write a program to set background and foreground color using applet.

```
import java.awt.*;
import java.applet.*;
public class backfore extends Applet
{
public void paint(Graphics g)
{
setBackground(Color.yellow);

g.setColor(Color.red);
g.drawString("Demo of background and Foreground color",10,50);
}
}
/*<applet code="backfore.class" width=100 height=100>
</applet>
*/
```



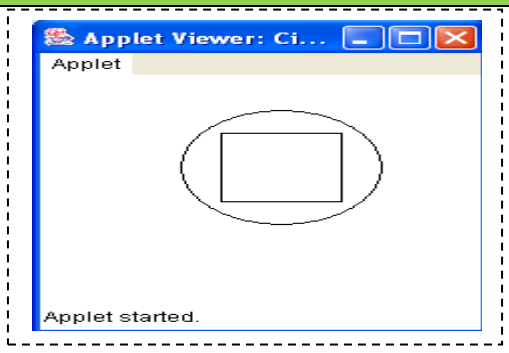
12. Write an applet to accept username from text box in the form and print "Hello <username>".

```
import java.awt.*;
import java.applet.*;
import java.lang.*;
public class username extends Applet
{
TextField text1;
public void init()
{
text1=new TextField(8);
add(text1);
}
public void paint (Graphics g)
{
String s1,s2;
g.drawString("Enter your name in box",10,50);
s1=text1.getText();
s2="Hello  "+s1;
g.drawString(s2,10,80);
}
public boolean action(Event event, Object object)
{
repaint();
return true;
} }
/* <applet code=username.class          width=100 height=100 >
</applet> */
```



13. Write the applets to draw square inside a circle.

```
import java.awt.*;
import java.applet.*;
public class CirSqr extends Applet
{
public void paint(Graphics g)
{
g.drawOval(70,30,100,100);
g.drawRect(90,50,60,60);
}
}
/*<applet code="CirSqr.class" height=200 width=200>
</applet> */
```



14. Write an applet to accept a username in the form of parameter and print 'Hello <username>'.

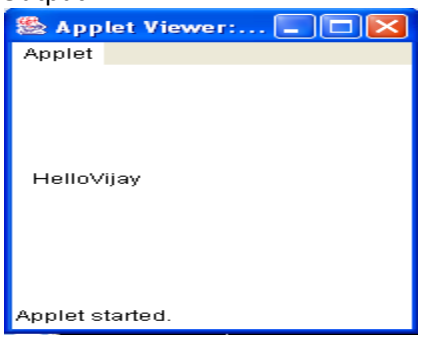
Java code

```
import java.awt.*;
import java.applet.*;
import java.lang.*;
public class hellojava1 extends Applet

{
String str;
public void init()
{
str=getParameter("string");
if(str==null)
str="Java";
str="Hello"+str;
}
public void paint (Graphics g)
{
g.drawString(str,10,100);
}
}
```

Html code

```
<html>
<body>
<applet code=hellojava1.class width=200
height=200>
<param name="string" value="Vijay">
</applet>
</body>
</html>
```

Output:



15. Create an applet which accepts user name as a parameter for html page and displays numbers of character from it.

```
java.awt.*;
import java.applet.*;
/* <applet code=Dispchar.class width=400 height=300>
<param name=UserName value=MSBTE>
</applet> */
public class Dispchar extends Applet
{
public void paint(Graphics g)
{
String user;
int count;
user = getParameter("UserName");
count = user.length();
g.drawString("UserName: " + user + " Total number of characters: " + count,10,20);
}
}
```

16. Implement a program to display the checker board

```
import java.awt.*;
import java.applet.*;
public class chess extends Applet
{
public void paint(Graphics g)
{
int r,c,x,y;
for(r=0;r<=8;r++)
{
for(c=0;c<=8;c++)
{
x=c*20;
y=r*20;
if((r%2)==(c%2))
{
g.setColor(Color.black);
}
else
{
g.setColor(Color.white);
}
g.fillRect(x,y,20,20);
}
}
}
}

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

Output: