Chapter - 15

Applets

Applets

What is an Applet?

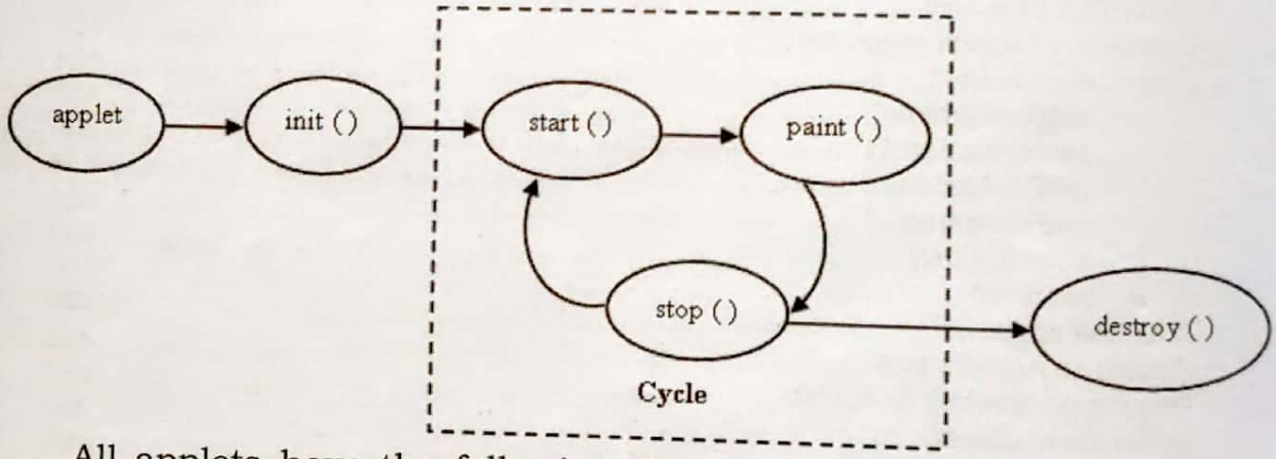
According to Sun "An applet is a small program that is intended not to be run on its own, but rather to be embedded inside another application. The Applet class provides a standard interface between applets and their environment."

Definitions of applet:

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

An applet is a small program that executes on an internet server, transport in the net faster and executes as a part of web document. Applets are used to develop web applications. Applets are secured programming i.e. applets executes only in the browser (**Internet Explorer**, **Netscape Navigator**, **Safari**, etc.) and can't access any resources of the computer such as files on hard disk, data in memory, hardware of the computer. Therefore computer is safe from virus and etc.

The Applet Life Cycle or Five Stages of an Applet



All applets have the following five methods and these methods are executed in the same sequence.

```
public void init();
public void start();
public void paint();
public void stop();
public void destroy();
```

Applets should have these methods because their superclasses, **java.applet.Applet**, has **init()**, **start()**, **stop()**, **destroy()** methods and **java.awt.Component**, has **paint()** method. In the superclass, these are simply do-nothing methods. For example,

```
public void init() {}
public void start() {}
public void paint() {}
public void stop() {}
public void destroy() {}
```

Subclasses may override these methods to accomplish certain tasks at certain times.

init(), start(), paint(), stop(), and destroy() methods.

init(): The init() method is called exactly once in an applet's life, when the applet is first loaded. It's normally used to read PARAM tags, start downloading any other images or media files you need, and set up the user interface. Most applets have init() methods.

For example, the **init()** method is a good place to read parameters that were passed to the applet via <PARAM> tags because it's called exactly once when the applet starts up. However, they do not have to override them. Since they're declared in the superclass, the Web browser can invoke them when it needs to without knowing in advance whether the method is implemented in the superclass or the subclass. This is a good example of polymorphism.

start() : The **start()** method is called at least once in an applet's life, when the applet is started or restarted. In some cases it may be called more than once. Many applets you write will not have explicit **start()**methods and will merely inherit one from their superclass. A **start()** method is often used to start any threads the applet will need while it runs.

paint() : The **paint()** is defined in AWT Component class and not present in Applet class. The paint() method is called at least once in an applet's life, when ever applet's output must be redrawn this method is called. This is the actual method which prints the output in the applet because this method contains an object of Graphics class which can be used to draw the drawings in the applet.

stop() : The stop() method is called at least once in an applet's life, when the browser leaves the page in which the applet is embedded. The applet's **start()** method will be called if at some later point the browser returns to the page containing the applet. In some cases the **stop()** method may be called multiple times in an applet's life. Many applets you write will not have explicit stop()methods and will merely inherit one from their superclass. Your applet should use the stop() method to pause any running threads. When your applet is stopped, it *should* not use any CPU cycles.

destroy() : The destroy() method is called exactly once in an applet's life, just before the browser unloads the applet. This method is generally used to perform any final clean-up. For example, an applet that stores state on the server might send some data back to the server before it's terminated. many applets will not have explicit **destroy()** methods and just inherit one from their superclass.

For example, in a video applet, the **init()** method might draw the controls and start loading the video file. The **start()** method would wait until the file was loaded, and then start playing it. The **stop()** method would pause the video, but not rewind it. If the **start()** method were called again, the video would pick up where it left off; it would not start over from the beginning. However, if **destroy()** were called and then init(), the video would start over from the beginning.

In the JDK's appletviewer window, selecting the **Restart** menu item calls **stop()** and then **start()**. Selecting the **Reload** menu item calls **stop(), destroy(),** and **init()**, in that order.

The applet **start()** and **stop()** methods are not related to the similarly named methods in the java.lang.Thread class.

Your own code may occasionally invoke **start()** and **stop()**. For example, it's customary to stop playing an animation when the user clicks the mouse in the applet and restart it when they click the mouse again.

Your own code can also invoke **init()** and **destroy()**, but this is normally a bad idea. Only the environment should call **init()** and **destroy().**

The Applet class

The Applet class of **java.applet** package contain all necessary methods for the execution of applet program. It contains the life cycle of an applet according to which the applet executes. The **Applet** class contains methods to **start**, **stop**, **destroy** etc. of an applet.

Any user-defined applet class can be defined by inheriting from **Applet** class because it the superclass which contains the life cycle of an applet. If life cycle inherits into user-defined applet class then it executes like an applet otherwise does not.

The **Applet** class extends from **Panel** and its hierarchy is shown below.

```
java.lang.Object
  |
  +----java.awt.Component
        |
        +----java.awt.Container
              |
              +----java.awt.Panel
                    |
                    +----java.applet.Applet
```

The following table contains methods of an **Applet** class

Methods	**Meaning**
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, String name)	Returns the AudioClip object specified by the URL and name arguments.
URL **getCodeBase**()	Gets the base URL.
URL **getDocumentBase**()	Gets the URL of the document in which this 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 that 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 **showStatus**(String msg)	Requests that the argument string be displayed in the "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.

The paint() method

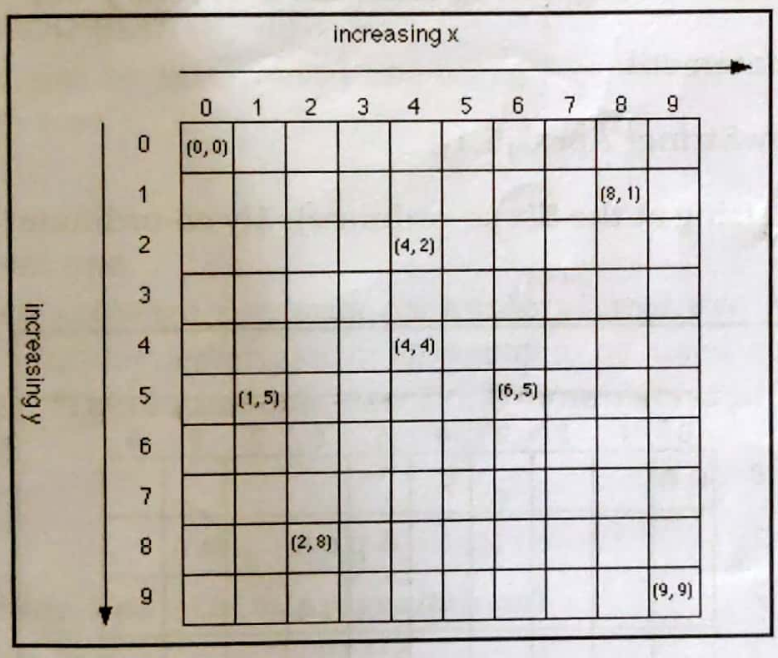
The **paint()** method of AWT **Component** class can be used to draw any kind of drawings in the applet such as drawing text, simple geometric shapes(lines, rectangles, etc), images etc. The signature of the paint() method is

```
public void paint(Graphics g)
```

The **paint()** method contains a parameter of **Graphics** class. The **Graphics** class is the abstract base class for all graphics contexts that allow an application to draw text, simple geometric shapes(lines, rectangles, etc).

The Coordinate System

Java uses the standard, two-dimensional, computer graphics coordinate system. The first visible pixel in the upper left-hand corner of the applet canvas is (0, 0). Coordinates increase to the right and down.



Graphics Objects

In Java all drawing takes place via a **Graphics** object. This is an instance of the class **java.awt.Graphics**.

Initially the Graphics object you use will be the one passed as an argument to an applet's **paint()** method. Everything you learn today about drawing in an applet transfers directly to drawing in other objects like **Panels**, **Frames**, **Buttons**, **Canvas** and more.

Each **Graphics** object has its own coordinate system, and all the methods of **Graphics** including those for drawing **Strings**, **lines**, **rectangles**, **circles**, **polygons** and more. Drawing in Java starts with particular Graphics object. You get access to the Graphics object through the **paint(Graphics g)** method of your applet. Each draw method call will look like **g.drawString("Hello World", 0, 50)** where g is the particular **Graphics** object with which you're drawing.

For convenience's sake in this lecture the variable **g** will always refer to a preexisting object of the **Graphics** class. As with any other method you are free to use some other name for the particular **Graphics** context, myGraphics or appletGraphics perhaps.

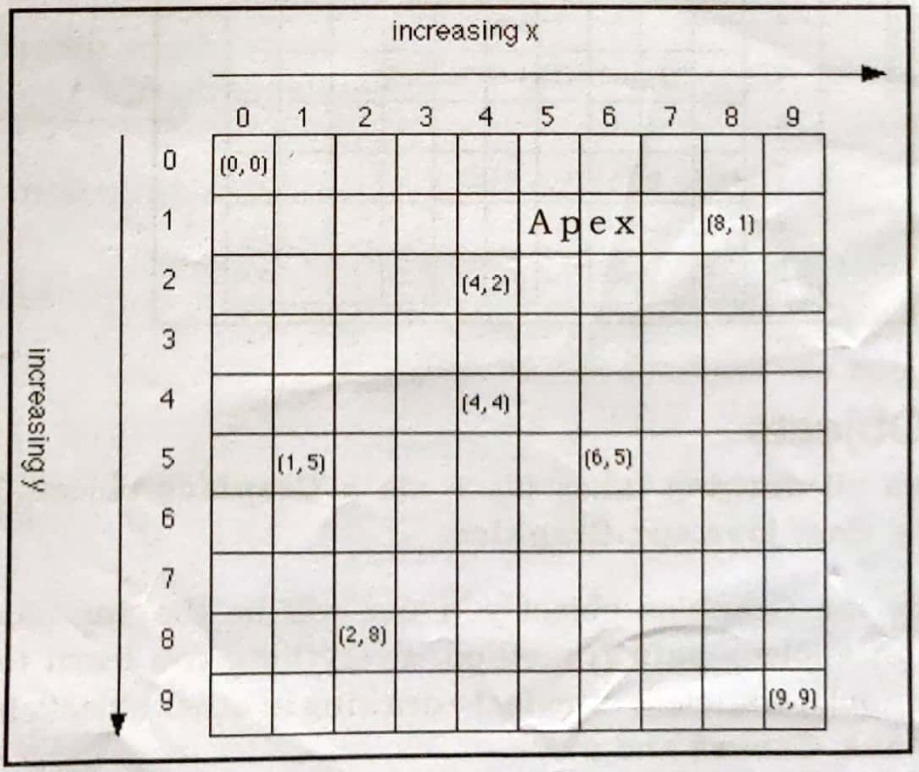
The Graphics class contain **drawString()** method which prints the text as the specified location on the applet. The signature of the **drawString()** method is

public void drawString(String text, int x-co, int y-co)

For example, the statement

g.drawString("Apex",5,1);

prints **Apex** string at the **5**(x co-ordinate), **1**(y co-ordinate) of the applet as shown bellow



The APPLET Element

Applet programs can not be executed using **java** command because applet program does not contain main() method. Applet programs can be executed using html program. The html program can execute applet using applet tag (<Applet> ... </Applet>). The simple structure of html file to run applet program is:

```
<applet code="appletclassname" width="widthofapplet"
height="heightofapplet">

</applet>
```

APPLET elements use **CODE** attribute to run the applet class file. The **CODE** attribute tells the browser where to look for the compiled .class file. It is relative to the location of the source document. **WIDTH** and **HEIGHT** attributes are used to specify width and height of an applet. These numbers are specified in pixels and are required.

Execution of Applet programs

Applet program can be executed in two ways.

1. Appletviewer.exe
2. Browser

1. Appletviewer.exe

The Java software contains an **appletviewer.exe** which acts like a mini browser. Appletviewer.exe program can be used to test the applet execution faster. There are some functions of applets that does not execute in appletviewer.

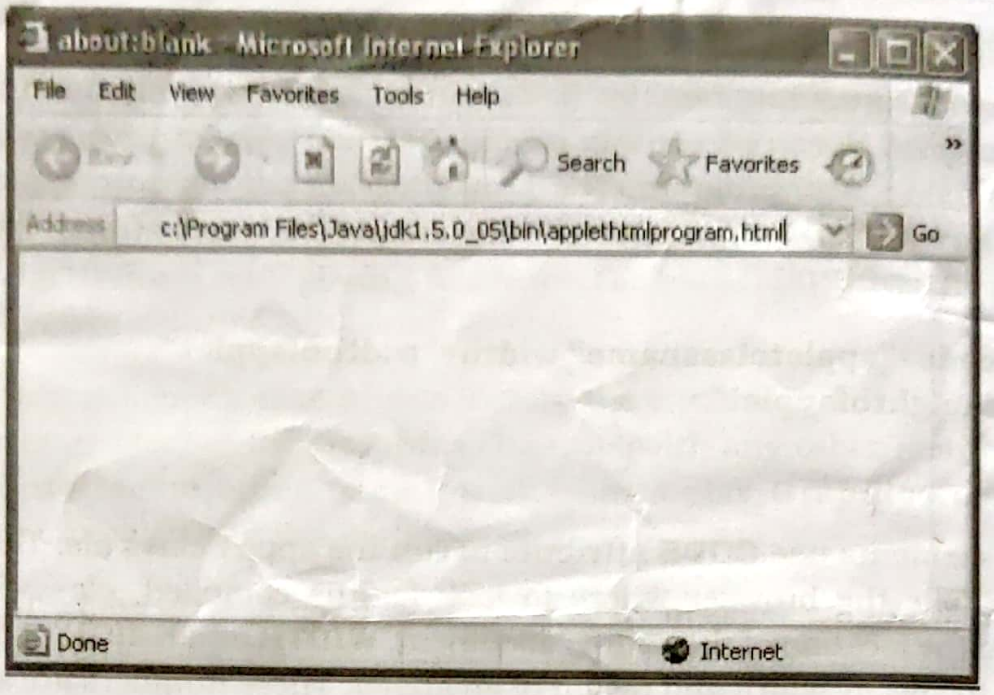
Example:

```
Bin>appletviewer applethtmlprogram.html
```

2. Browser

Applet programs actually executes in browsers such as **Internet Explorer**, **Netscape Navigator**, **Mouzilla**, **Safari** etc,. Applet program executes in the browser a part of browser document. Give the path of applet program html file in the browser for execution.

Example :



A first program on Applet

Bin>edit applet1.java

```
import java.awt.*;
import java.applet.*;
public class applet1 extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("This is sample Applet", 100, 100);
    }
}
```

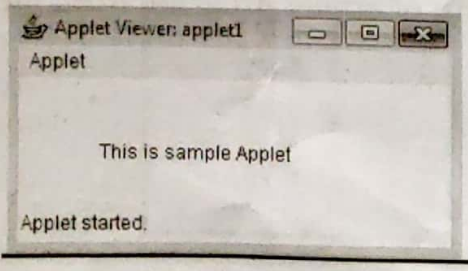
Bin>javac applet1.java

Bin>edit applet1.html

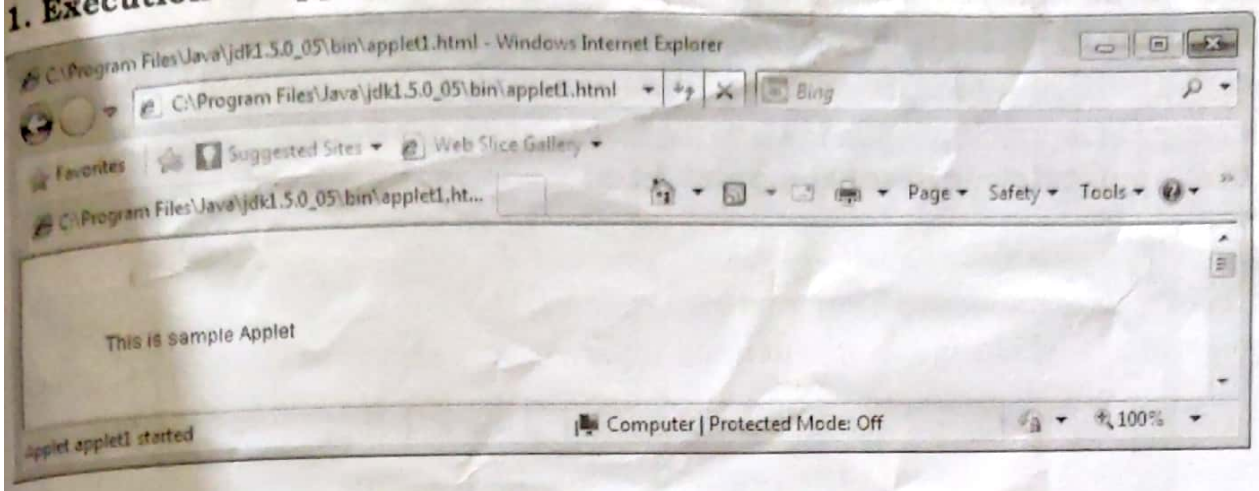
```
<applet code="applet1" width=400 height=400>
</applet>
```

1. Execution of applet using appletviewer.exe

Bin>appletviewer applet1.html



1. Execution of applet using Browser



A program to demonstrate life cycle of an applet

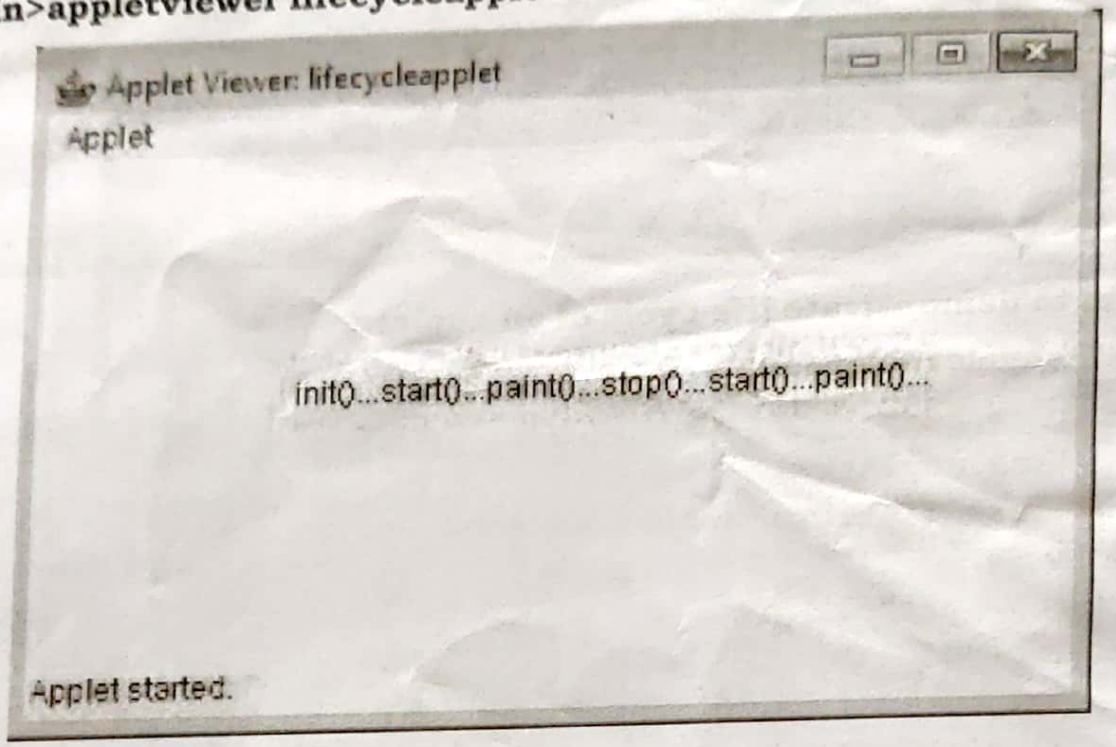
Bin>edit lifecycleapplet.java

```
import java.awt.*;
import java.applet.*;
public class lifecycleapplet extends Applet
{
    String str="";
    public void init()
    {
        str="init()...";
    }
    public void start()
    {
        str=str+"start()...";
    }
    public void paint(Grahics g)
    {
        str=str+"paint()...";
        g.drawString(str,100,100);
    }
    public void stop()
    {
        str=str+"stop()...";
    }
    public void destroy()
    {
        str=str+"destroy()...";
    }
}
```

Bin>javac lifecycleapplet.java
Bin>edit lifecycleapplet.html

```
<applet code="lifecycleapplet" width=400 height=400>
</applet>
```

Bin>appletviewer lifecycleapplet.html



Explanation:

When the above applet program is executed using appletviewer, the first method **init()** is executed where **str** is stored with "**init()...**" and then **start()** method executes where **str** is appended with "**start()...**" therefore now **str** contains "**init()...start()...**". After **start()** method the **paint()** method executes where **str** is appended with "**paint()...**" and the **drawString()** method prints the **str** value on the applet as "**init()...start()...paint()...**". Minimize and restore the appletviewer window, when minimized the **stop()** method is executed where **str** is appended with "**stop()**" and when restored, **start()** and **paint()** methods executes and appends the "**start()...**" and "**paint()...**" respectively and print the output as shown above. Finally, close the applet by clicking on the close button. Before terminating the applet program the **stop()** and **destroy()** methods executes where it appends "**stop()...**" and "**destroy()...**" to the **str** and **Sytem.print(str);** method prints the **str** value on the console as

```
init()...start()...paint()...stop()...start()...paint()...stop()...destroy()...
```

Class Component

A ***component*** is an object having a graphical representation that can be displayed on the screen and that can interact with the user. Examples of components are the **buttons**, **checkboxes**, and **scrollbars** of a typical graphical user interface.

The **Component** class is the abstract superclass of the nonmenu-related Abstract Window Toolkit components. Class Component can also be extended directly to create a lightweight component. A lightweight component is a component that is not associated with a native opaque window.

The following table shows methods of **Component** class.

Methods	Meaning
void **addComponentListener** (ComponentListener l)	Adds the specified component listener to receive component events from this component.
void **addFocusListener** (FocusListener l)	Adds the specified focus listener to receive focus events from this component when this component gains input focus.
void **addKeyListener** (KeyListener l)	Adds the specified key listener to receive key events from this component.
void **addMouseListener** (MouseListener l)	Adds the specified mouse listener to receive mouse events from this component.
void **addMouseMotionListener** (MouseMotionListener l)	Adds the specified mouse motion listener to receive mouse motion events from this component.
void **addMouseWheelListener** (MouseWheelListener l)	Adds the specified mouse wheel listener to receive mouse wheel events from this component.
Image **createImage**(ImageProducer producer)	Creates an image from the specified image producer.
Image **createImage**(int width, int height)	Creates an off-screen drawable image to be used for double buffering.
Color **getBackground**()	Gets the background color of this component.

Rectangle **getBounds**()	Gets the bounds of this component in the form of a Rectangle object.
Rectangle **getBounds**(Rectangle rv)	Stores the bounds of this component into "return value" **rv** and return **rv**.
Component **getComponentAt**(int x, int y)	Determines if this component or one of its immediate subcomponents contains the (*x*, *y*) location, and if so, returns the containing component.
Component **getComponentAt**(Point p)	Returns the component or subcomponent that contains the specified point.
Font **getFont**()	Gets the font of this component.
FontMetrics **getFontMetrics**(Font font)	Gets the font metrics for the specified font.
Color **getForeground**()	Gets the foreground color of this component.
Graphics **getGraphics**()	Creates a graphics context for this component.
int **getHeight**()	Returns the current height of this component.
String **getName**()	Gets the name of the component.
Container **getParent**()	Gets the parent of this component.
Dimension **getSize**()	Returns the size of this component in the form of a Dimension object.
int **getWidth**()	Returns the current width of this component.
int **getX**()	Returns the current x coordinate of the components origin.
int **getY**()	Returns the current y coordinate of the components origin.
boolean **isVisible**()	Determines whether this component should be visible when its parent is visible.
void **print**(Graphics g)	Prints this component.
void **repaint**()	Repaints this component by calling paint()

void **repaint**(int x, int y, int width, int height)	Repaints the specified rectangle of this component.
void **repaint**(long tm)	Repaints the component.
void **repaint**(long tm, int x, int y, int width, int height)	Repaints the specified rectangle of this component within tm milliseconds.
void **requestFocus**()	Requests that this Component get the input focus, and that this Component's top-level ancestor become the focused Window.
void **setBackground**(Color c)	Sets the background color of this component.
void **setBounds**(int x, int y, int width, int height)	Moves and resizes this component.
void **setBounds**(Rectangle r)	Moves and resizes this component to conform to the new bounding rectangle r.
void **setEnabled**(boolean b)	Enables or disables this component, depending on the value of the parameter b.
void **setFont**(Font f)	Sets the font of this component.
void **setForeground**(Color c)	Sets the foreground color of this component.
void **setName**(String name)	Sets the name of the component to the specified string.
void **setSize**(Dimension d)	Resizes this component so that it has width d.width and height d.height.
void **setSize**(int width, int height)	Resizes this component so that it has width width and height height.
void **setVisible**(boolean b)	Shows or hides this component depending on the value of parameter b.
String **toString**()	Returns a string representation of this component and its values.
void **update**(Graphics g)	Updates this component.

setBackground() method

The **setBackground()** method is defined by **Component** class. This method sets the background color of an applet. The signature of this method is

setBackground(Color c)

Class Color

The **Color** class contains public static final variables each represent a color. The variables of **Color** class are:

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

These variables also present in Capital letters such as **Color.BLACK**, **Color.MAGENTA**, and so on.

setForeground()

The setForeground() method is defined by Component class. This method sets the foreground color of an applet i.e. text color. The signature of this method is

setForeground(Color c)

getBackground()

The getBackground() method is defined by Component class. This method returns back ground color of an applet. The signature of this method is..

Color getBackgound()

getForeground()

The getForeground() method is defined by **Component** class. This method returns fore ground color of an applet. The signature of this method is..

Color getForeground()

showStatus()

This method id defined by **Applet** class. This method shows messages on the status bar of the window of browser or appletviewer where applet is running. The signature of this method is:

public void showStatus(String msg);

A program to demonstrate setBackground(), setForeground(), getBackground(), getForeground() and showStatus() methods.

Bin>edit color1.java

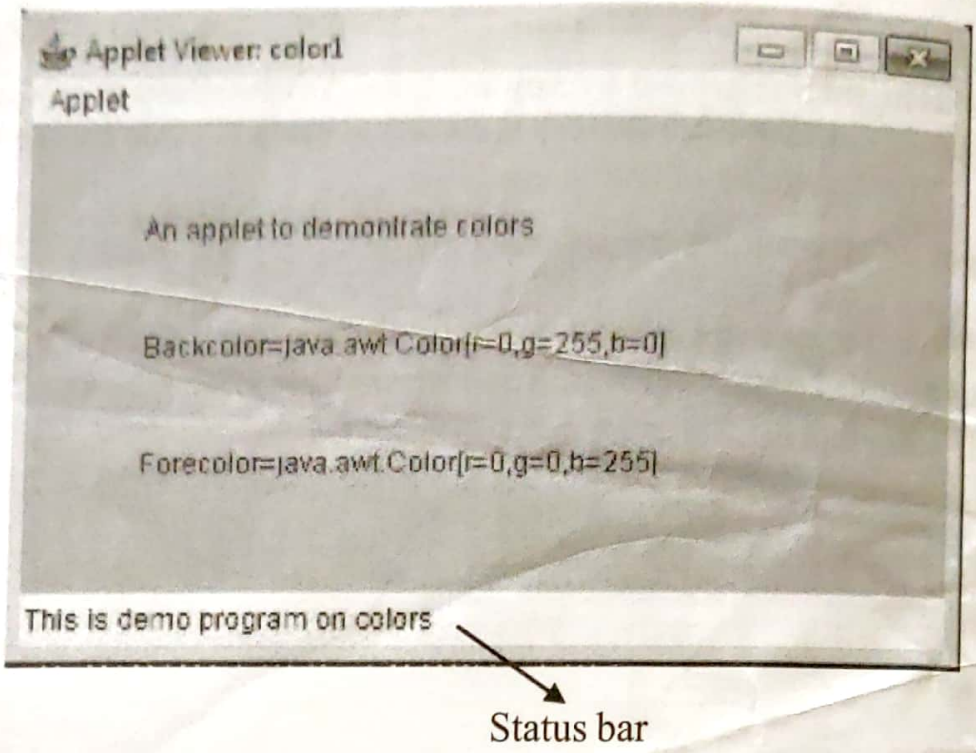
```
import java.awt.*;
import java.applet.*;
public class color1 extends Applet
{
    public void init()
    {
        setBackground(Color.green);
        setForeground(Color.BLUE);
    }
    public void paint(Graphics g)
    {
        g.drawString("An applet to demontrate
                                    colors.",50,50);
        Color c;
        c=getBackground();
        g.drawString("Backcolor="+c.toString(),50,100);
        c=getForeground();
        g.drawString("Forecolor="+c.toString(),50,150);
        showStatus("This is demo program on colors");
    }
}
```

Bin>javac color1.java

Bin>edit color1.html

```
<applet code="color1" width=400 height=400>
</applet>
```

Bin>appletviewer color1.html



repaint()

The repaint() method is defined in **Component** class. This method causes a call to the component's **paint()** method which do the drawings in the applet. The repaint method is overloaded and their signatures are:

Methods	Meaning
void **repaint**()	Repaints this component by calling paint()
void **repaint**(int x, int y, int width, int height)	Repaints the specified rectangle of this component.
void **repaint**(long tm)	Repaints the component with in tm milliseconds.
void **repaint**(long tm, int x, int y, int width, int height)	Repaints the specified rectangle of this component within tm milliseconds.

A program on Banner applet

Banner is a text that scrolls the text in the same place. In this program, we display the text "**Apex Computers**" as banner.

The banner program contains a thread that executes for every 200 milliseconds which displays the banner.

Bin>edit banner1.java

```
import java.applet.*;
import java.awt.*;
public class banner1 extends Applet implements Runnable
{
    Thread t;
    String str="APEX COMPUTERS";
    boolean flag;
    public void init()
    {
        setBackground(Color.yellow);
        setForeground(Color.red);
    }
    public void start()
    {
        t=new Thread(this,"Banner");
        flag=true;
        t.start();
    }
    public void run()
    {
        try
        {
            while(flag)
            {
                repaint();
                Thread.sleep(200);
                char ch=str.charAt(0);
                str=str.substring(1,str.length());
                str=str+ch;
                if(flag==false)
                    break;
            }
        }
        catch(InterruptedException e)
        {
            System.out.print("\nError:"+e);
            showStatus("Error : "+e);
        }
    }
```

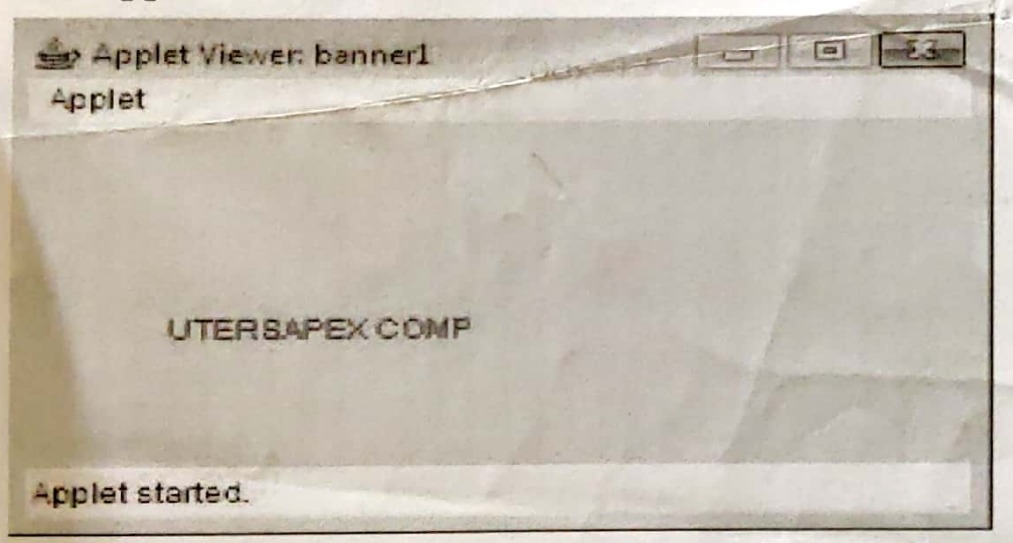
```
    public void paint(Graphics g)
    {
        g.drawString(str,50,100);
    }
    public void stop()
    {
        flag=false;
    }
}
```

Bin>javac banner1.java

Bin>edit banner1.html

```
<applet code="banner1" width=400 height=400>
</applet>
```

Bin>appletviewer banner1.html



Finding an Applet's Size

When running inside a web browser the size of an applet is set by the height and width attributes and cannot be changed by the applet. Many applets need to know their own size. After all we don't want to draw outside the lines.

We can get the size of an applet by using **getSize()** method of **java.awt.Component**. The **getSize()** method returns a **java.awt.Dimension** object. A **Dimension** object has two public int fields, **height** and **width**. Below is a simple applet that prints its own dimensions.

Bin>edit appletsize.java

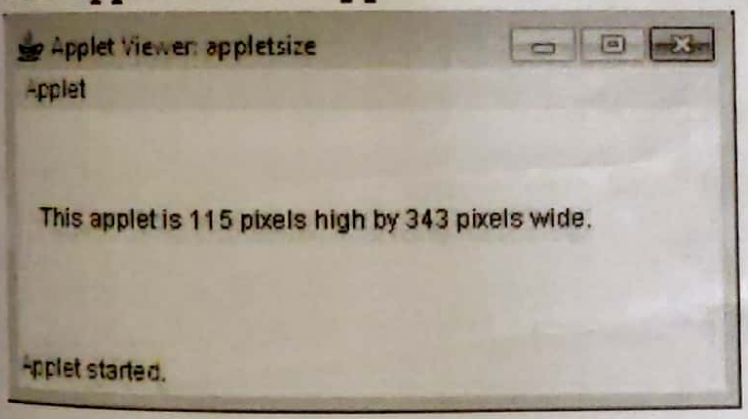
```
import java.applet.*;
import java.awt.*;
public class apletsize extends Applet
{
    public void paint(Graphics g)
    {
        Dimension appletSize = this.getSize();
        int appletHeight = appletSize.height;
        int appletWidth = appletSize.width;
        g.drawString("This applet is " + appletHeight +
                " pixels high by " + appletWidth +
                " pixels wide.", 15, appletHeight/2);
    }
}
```

Bin>javac appletsize.java

Bin>edit appletsize.html

```
<applet code="appletsize" width=400 height=200>
</applet>
```

Bin>appletviewer appletsize.html



Passing Parameters to Applets

It is possible to pass parameters to applets at runtime. Parameters are passed to applets from html programs because applets executes from html programs. We can use the **PARAM** tag to pass parameter name and its value from with in the element of **APPLET**. The general structure of **PARAM** tab is:

```
<PARAM name="paraname1" value="paravalue1">
<PARAM name="paraname2" value="paravalue2">
                :                    :
<PARAM name="paranameN" value="paravalueN">
```

To an applet any number of parameters can be passed provided parameter names should differ.

In the applet program, parameter values can be obtained using the **getParameter()** method of Applet class. The signature of **getParameter()** method is:

public String getParameter(String param_name)

If the parameter name in the **getParameter()** method is not found then it returns **null**.

A program to demonstrate passing parameters to applets

In this program, we pass two parameters such as **name** and **age** of the user with different values to applet and the applet prints whether the user is eligible for fashion show or not.

Bin>edit para1.java

```
import java.awt.*;
import java.applet.*;
public class para1 extends Applet
{
    public void paint(Graphics g)
    {
          String name;
          int age;
          name=getParameter("name");
          age=Integer.parseInt(getParameter("age"));
          if(age>=21)
             g.drawString("You are eligible for
                fashion show.",50,50);
          else
             g.drawString("You are not eligible for
                fashion show try after few years.",50,50);
             g.drawString("Your name=" +name,50,70);
             g.drawString("Your age=" + age,50,90);
    }
}
```

Bin>javac para1.java

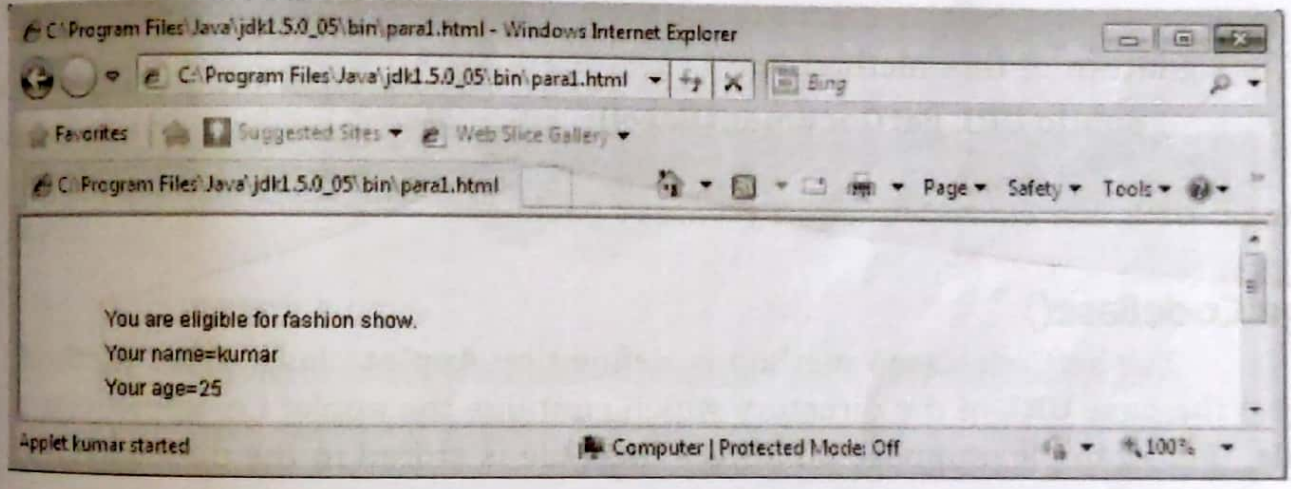
Bin>edit para1.html

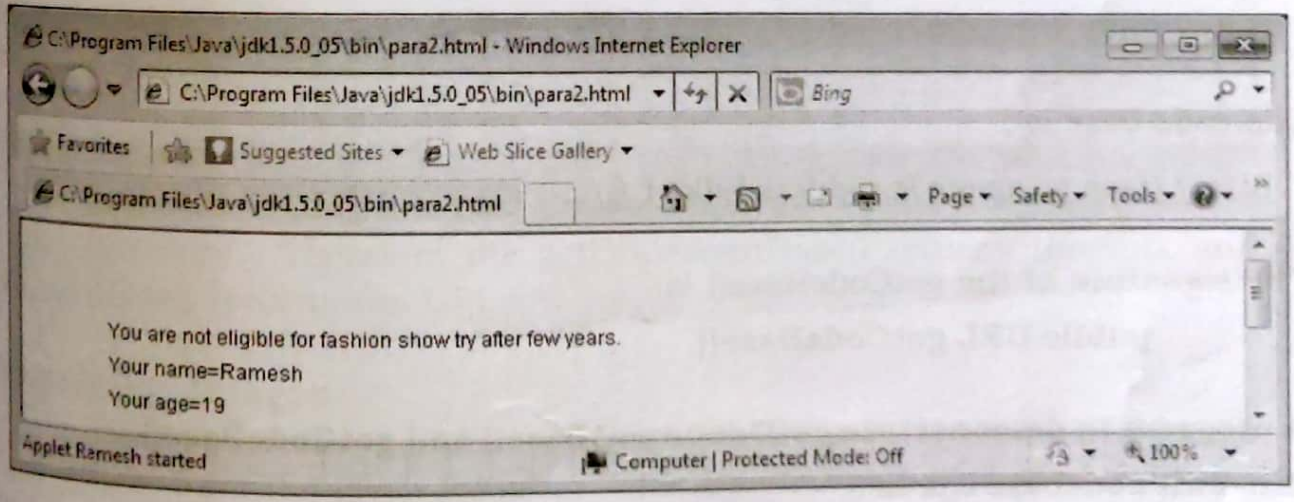
```
<applet code="para1" width=400 height=200>
<param name=name value="kumar">
<param name=age value="25">
</applet>
```

Bin>edit para2.html

```
<applet code="para1" width=400 height=200>
<param name=name value="Ramesh">
<param name=age value="19">
</applet>
```

Execute **para1.html** and **para2.html** in two different windows of browser(Internet Explorer). For example.





To a single applet program we can pass parameters from different html programs i.e. multiple html programs can share the same applet program. This method of passing parameters to applets is useful when a single applet works for different values.

Of course we are free to change **name** and **age** values of our choice. We only need to change the HTML, not the Java source code. PARAMs let us customize applets without changing or recompiling the code.

getDocumentBase()

The getDocumentBase() method is defined by **Applet** class. The method gets the **URL** of the document(html file) in which this applet is embedded. For example, suppose an applet is contained within the document:

http://java.sun.com/products/jdk/1.6/index.html

The document base is:

http://java.sun.com/products/jdk/1.6/index.html

The signature of this method is:

public URL getDocumentBase()

Note: URL is a class defined in **java.net** package.

getCodeBase()

The getCodeBase() method is defined by **Applet** class. This method gets the base **URL** of the directory which contains the applet i.e. applet class file. For example, suppose an applet class file is stored in the directory:

http://java.sun.com/products/jdk/1.6/

The code base is :

http://java.sun.com/products/jdk/1.6/

The signature of the getCodeBase() is:

public URL getCodeBase()

A program to demonstrate getDocumentBase() and getCodeBase().

Bin>edit codebase1.java

```
import java.awt.*;
import java.applet.*;
import java.net.*;
public class codebase1 extends Applet
{
```

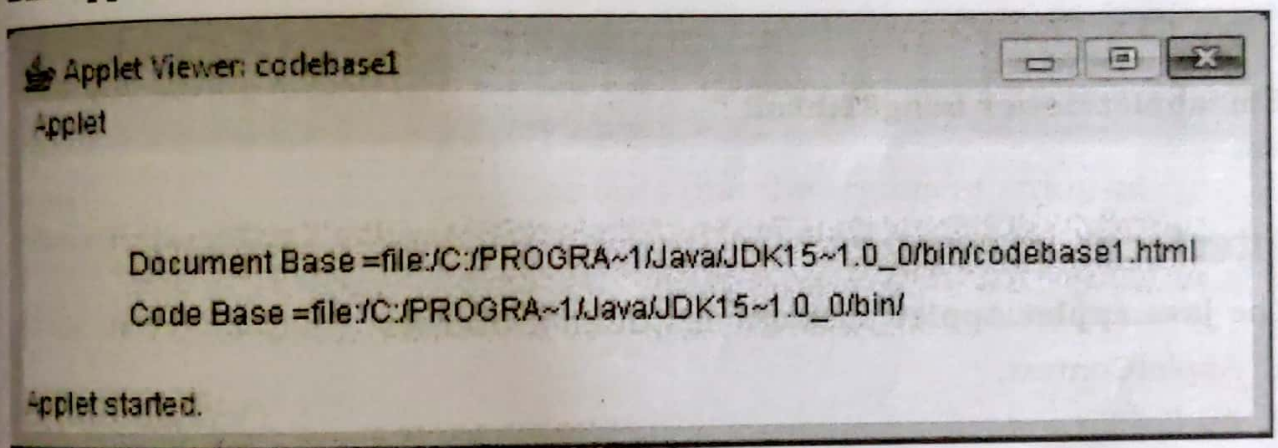
```
    public void paint(Graphics g)
    {
          URL url;
          url=getDocumentBase();
          g.drawString("Document Base="+url.toString() ,50,50);
          url=getCodeBase();
          g.drawString("Code Base =" + url ,50,70);
    }
}
```

Bin>javac codebase1.java

Bin>edit codebase1.html

```
<applet code="codebase1" width=500 height=200>
</applet>
```

Bin>appletviewer codebase1



Explanation:

In the above program, both html program and applet program are stored in bin directory. Therefore the getDocumentBase() returns the URL and getCodeBase() returns the URL

Dispalying Images

The **Applet** class contain **getImage()** method which returns an **Image** class object from the given location. The **drawImage()** method of Graphics class can be used to draw the image on the applet.

A program to draw an image on the applet.

Bin>edit image1.java

```
import java.awt.*;
import java.applet.*;
import java.net.*;
public class image1 extends Applet
{
    public void paint(Graphics g)
    {
        Image i=getImage(getCodeBase(),"car.jpg");
        g.drawImage(i,50,50,this);
    }
}
```

Bin>javac image1.java

Bin>copy con image1.html

```
<applet code="image1" width=400 height=400>
</applet>
```

Bin>appletviewer image1.html

Interfaces in Applet

The **java.applet.Applet** package has three interfaces

1. AppletContext,
2. AudioClip and
3. AppletStub

1. Interface AppletContext

In applets, there are some situations where we want to transfer the control from one applet to another when we work with multiple applets. This can be achieved by using **showDocument()** method of **AppletContext** interface. This interface corresponds to an applet's execution environment which can be used to execute other applets or web documents(Ex: .html. .jsp etc.,).

The methods in this interface can be used by an applet to obtain information about its execution environment.

The following table shows methods in the **AppletContext** interface.

Method	Meaning
Applet **getApplet**(String name)	Finds and returns the applet in the document represented by this applet context with the given name.
AudioClip **getAudioClip**(URL url)	Creates an audio clip.
Image **getImage**(URL url)	Returns an Image object that can then be painted on the screen.
InputStream **getStream**(String key)	Returns the stream to which specified key is associated within this applet context.
void **showDocument**(URL url)	Requests that the browser or applet viewer show the Web page indicated by the url argument.
void **showDocument**(URL url, String target)	Requests that the browser or applet viewer show the Web page indicated by the url argument.
void **showStatus**(String status)	Requests that the argument string be displayed in the "status window".

Some of the methods of **AppletContext** interface are described below.

getAudioClip()

This method of **AppletContext** interface creates an audio clip and returns an instance of the AutdioClip interface.

AudioClip getAudioClip(URL url)

The url in the method represents an absolute **URL** giving the location of the audio clip.

getImage()

This method of **AppletContext** interface returns an Image object that can then be painted on the screen.

This method returns whether image exist or not. When the applet attempts to draw the image on the screen, the data will be loaded. The graphics primitives that draw the image will incrementally paint on the screen.

Image getImage(URL url)

The url argument that is passed as an argument must specify an absolute **URL** of an image.

getApplet()

This method of **AppletContext** interface finds and returns the applet in the document represented by this applet context with the given name. The name can be set in the HTML tag by setting the **name** attribute.

Applet getApplet(String name)

The **name** argument represent the name of an applet that is set in the HTML file.

showDocument()

This method of AppletContext interface requests the browser or appletviewer to show the Web page indicated by the url argument. The browser or appletviewer determines which window or frame to display the Web page. This method does not work in non browsers such as appletviewer.

void **showDocument**(URL url)

The url argument that is passed as an argument must specify an absolute **URL** giving the location of the document. In case **URL** is not a valid one then it throws **MalformedURLException**.

showDocument()

This method of **AppletContext** interface requests the browser or appletviewer to show the Web page indicated by the url argument. The target argument indicates in which HTML frame the document is to be displayed. This method does not work in non browsers such as appletviewer.

void showDocument(URL url,String target)

The target argument is interpreted as follows:

Target Argument	Description
"_self"	Show in the window and frame that contain the applet.
"_parent"	Show in the applet's parent frame. If the applet's frame has no parent frame, acts the same as "_self".
"_top"	Show in the top-level frame of the applet's window. If the applet's frame is the top-level frame, acts the same as "_self".

"_blank"	Show in a new, unnamed top-level window.
name	Show in the frame or window named *name*. If a target named *name* does not already exist, a new top-level window with the specified name is created, and the document is shown there.

showStatus()

This method of **AppletContext** interface requests that the argument string be displayed in the "status window". Many browsers and appletviewers provide such a window, where the application can inform users of its current state.

void **showStatus**(String status)

The status argument is a string to display in the status window.

The getAppletContext() of an Applet

This method is defined by Applet class. This method returns the context of currently executing applet using which a html file can be executed.

public AppletContext getAppletContext()

A program to show an html file from an applet using showDocument() of AppletContext interface.

Bin>edit runhtml.java

```
import java.awt.*;
import java.applet.*;
import java.net.*;
public class runhtml extends Applet
{
    public void start()
    {
        AppletContext ac;
        ac=getAppletContext();
        URL url=getCodeBase();
        try
        {
            ac.showDocument(new URL(url+
                    "banner1.html"),"_blank");

        }
```

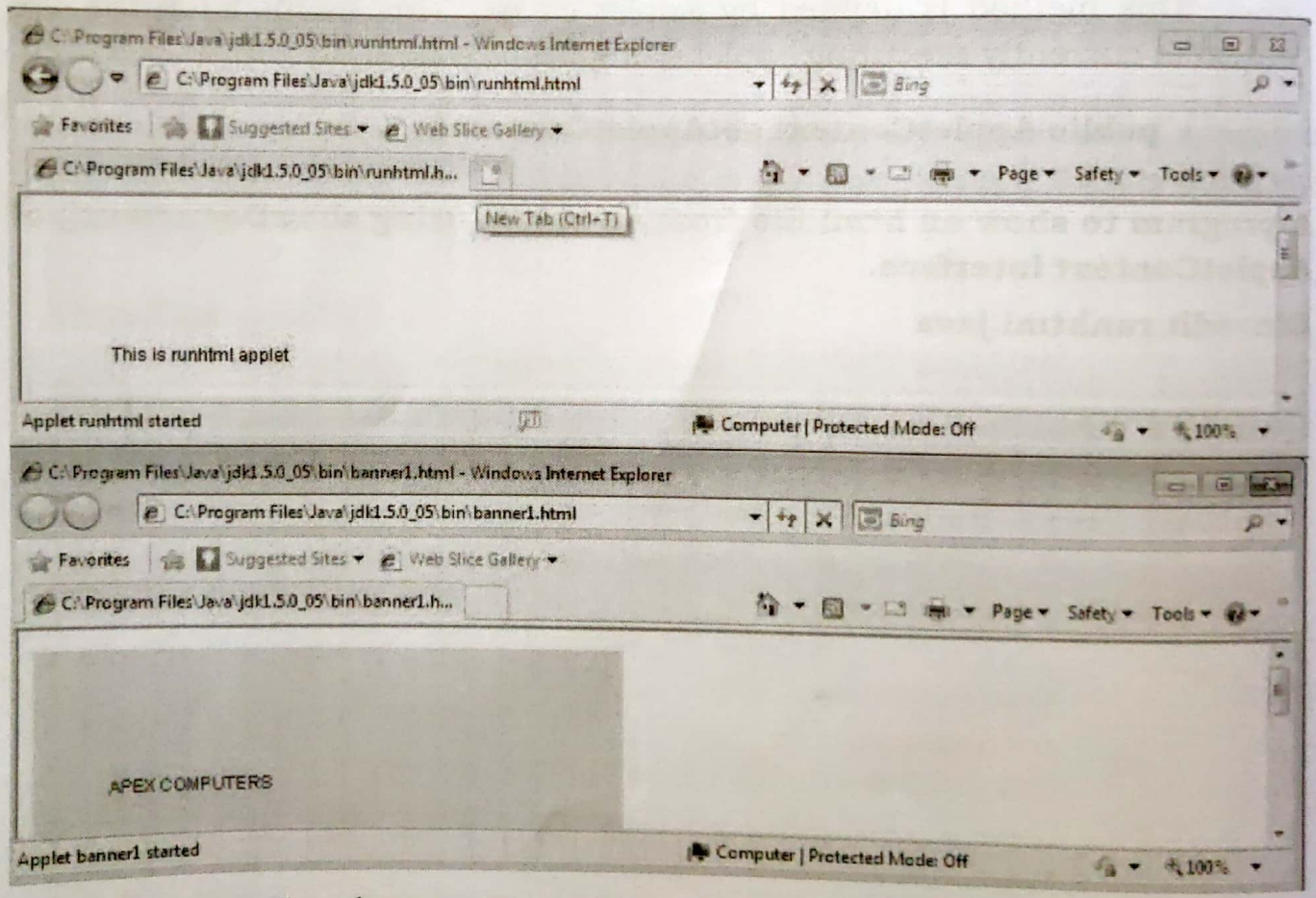
```
            catch(MalformedURLException e)
            {
                  System.out.print(e);
            }
      }
      public void paint(Graphics g)
      {
            g.drawString("This is runhtml applet",50,100);
      }
}
```

Bin>javac runhtml.java

Bin>edit runhtml.html

```
<applet code="runhtml" width=400 height=400>
</applet>
```

Execute runhtml.html in the browser and we the output as shown below



Note: Execute the above program by replacing the code in try block with the following and identify the difference in the output.

ac.showDocument(new URL(url+"banner1.html"));

1. Interface AudioClip

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.

Method	Meaning
void **loop**()	Starts playing this audio clip in a loop.
void **play**()	Starts playing this audio clip.
void **stop**()	Stops playing this audio clip.

A program to play an audio file using AudioClip interface

Bin>edit appletaudio1.java

```
import java.awt.*;
import java.applet.*;
import java.net.*;
public class appletaudio1 extends Applet
{
    AudioClip music;
    public void paint(Graphics g)
    {
          g.drawString("Audio playing...",100,100);
          URL url=getCodeBase();
          music=getAudioClip(url,"spacemusic.au");
          music.play();
    }
    public void stop()
    {
          music.stop();
    }
}
```

Bin>javac appletaudio1.java

Bin>edit appletaudio1.html

```
<applet code="appletaudio1" width=400 height=400>
</applet>
```

Bin>appletviewer appletaudio1.html

Note: Copy **.au** format audio file into the Bin directory and execute the program. Applets plays **.au** format files.

2. Interface AppletStub

When an applet is first created, an applet stub is attached to it using the applet's **setStub()** method. This stub serves as the interface between the applet and the browser environment or applet viewer environment in which the application is running.