APPLETS

CLASSIFICATION OF JAVA PROGRAMS

- Java programs can be broadly classified into two:
  - **Applications**
  - **Applets**

Applications

- Standalone general-purpose java programs that can be run by using the virtual machine installed on the local system**.**
- A java application starts its execution from the main() method
- They are usually large programs which are designed to work under secure areas

APPLETS INTRODUCTION

- Applets are java programs that are invoked from a Web page which can be used to provide interactive features to web applications.

- They can capture mouse movements and also have controls like buttons or check boxes.

- In response to the user action an applet can change the provided graphic content which makes applets well suitable for demonstration, visualization and teaching.

- Applets are also used to create online game collections
- Java applets are executed by a program called applet engine that exists inside the Web browser

APPLETS INTRODUCTION

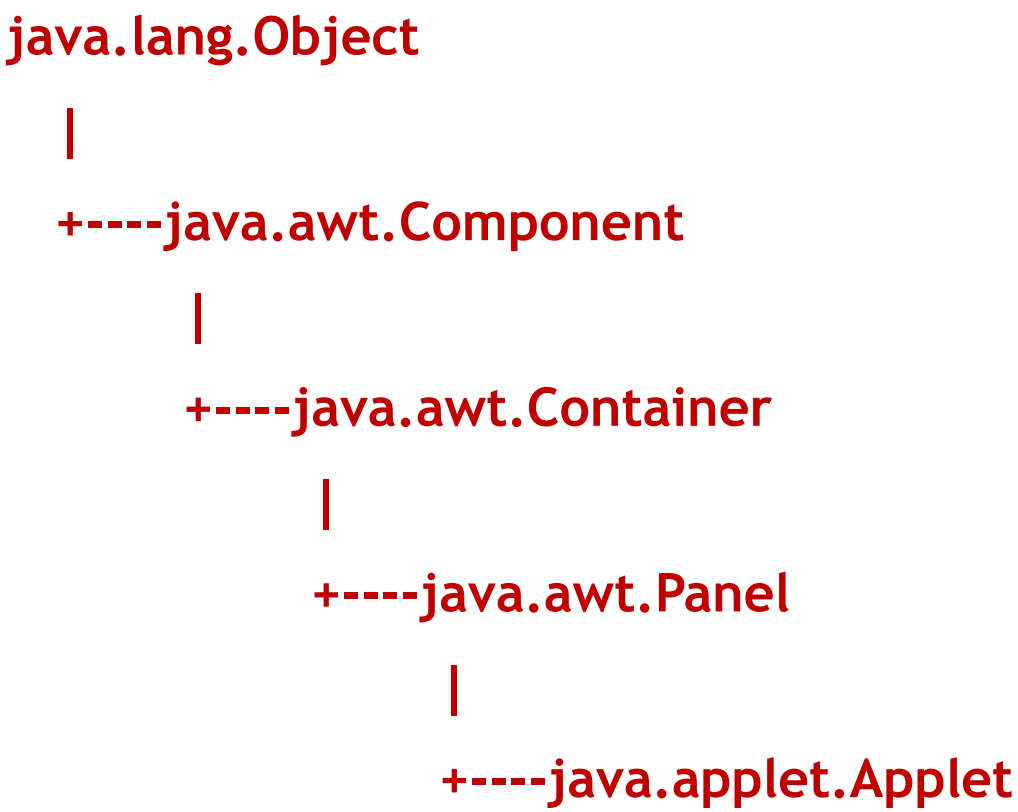
- A reference to an applet is embedded in a Web page using a special HTML tag
- When a user, using a Java-enabled browser, loads a Web page with an applet in it, the browser downloads that applet from a Web server and executes it on the local system.
- **Several restrictions are imposed on applets:**
  - Applets cannot read or write files on the Web user's disk.
  - Applets cannot make a network connection to a computer other than the one from which the Web page is served
  - Applets cannot run any programs on the Web user's system

The restricted environment in which applets are allowed to run is called the **applet Sandbox**

CREATING APPLETS

- All applets are subclasses of **Applet** class which belongs to the package **java.applet**
- Class **Applet** provides all the basic functionality of an applet, the ability to create a window, refresh it, draw on it and destroy it.
- Applets have many different activities that correspond to various major events in the life cycle of the applet
- Each activity has a corresponding method which will be overridden in the applet subclass according to the requirement.
- The path for execution or way to execute all these methods is referred as **applet life cycle**.
- These methods are automatically invoked when we are executing an applet

APPLET CLASS HIERARCHY



CREATING APPLETS

- The applet class always has a signature like this:

```
import java.applet.Applet;
public class myClass extends Applet
{
    ...
}
```

- Java requires that your applet subclass be declared **public**

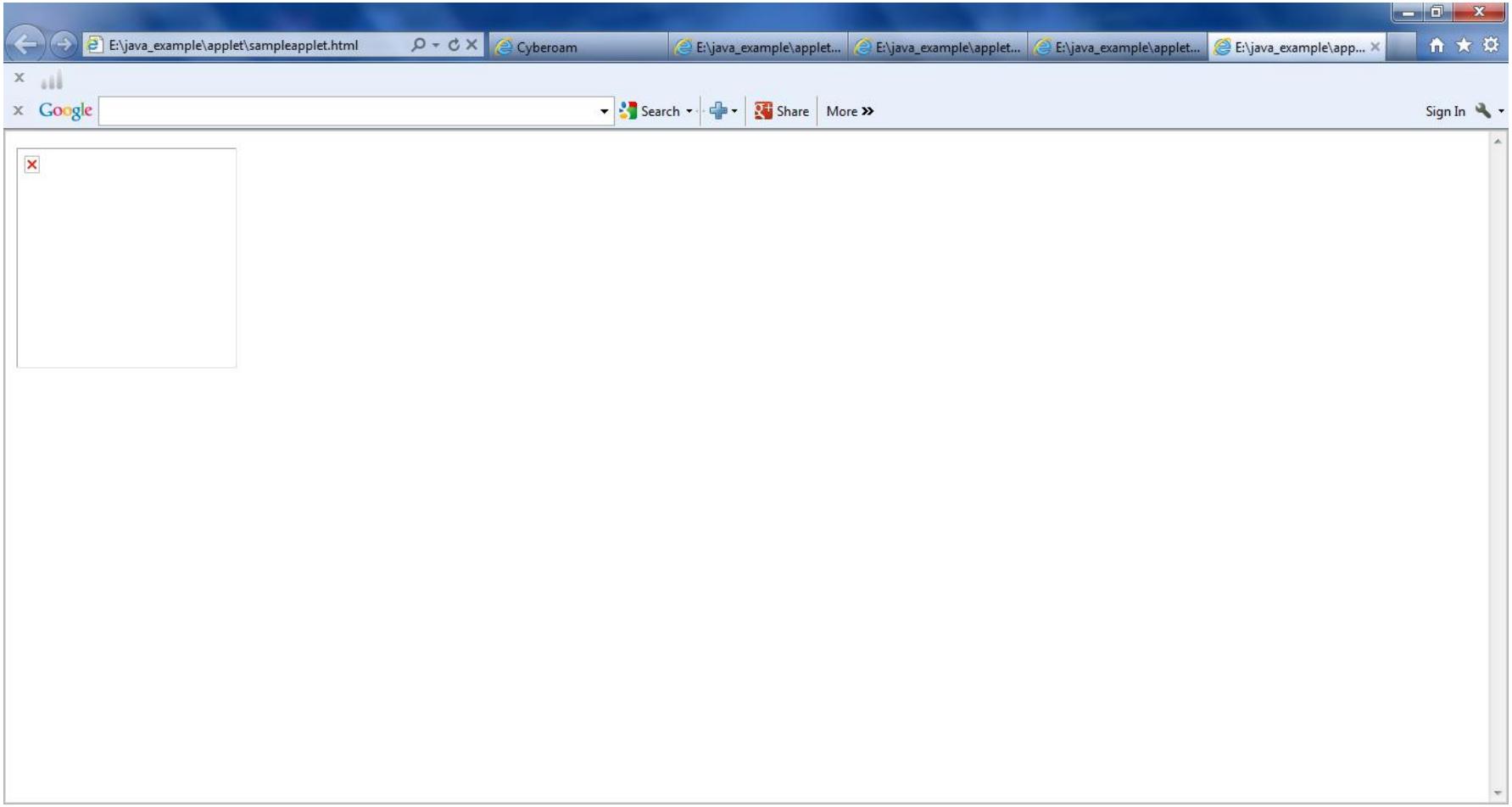
EXECUTING APPLETS

- There are two ways in which an applet can be executed
  - **Using web browser**
  - **Using applet viewer**

- Once an applet has been compiled, the name of the class file obtained is included in an **HTML** file using the **APPLET** tag
- The applet will be executed by a Java-enabled web browser when it encounters the **APPLET** tag within the HTML file.
  Eg: ***<applet code="myClass.class" width=200 height=200>***

  </applet>

EXECUTING APPLETS



EXECUTING APPLETS

- **AppletViewer** is a standalone command-line program provided by sun Microsystems.
- **AppletViewer** is generally used by developers for testing their applets before deploying them to a website.
- To execute an applet in **applet viewer,** simply include a comment at the head of your Java source code file that contains the **APPLET** tag
- This way, your code is documented with the necessary HTML statements needed by your applet.

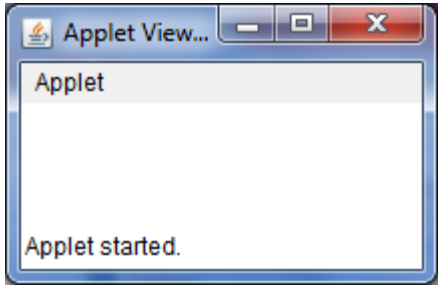
Eg: **import java.applet.Applet;**

```
/*
        <applet code="myClass" width=200 height=60> </applet>
  */
public class myClass extends Applet
 {

 }
```

EXECUTING APPLETS

- The **APPLET** tag that will run an applet called **myClass** in a window that is 200 pixels wide and 60 pixels high.
- The compiled applet can be tested by starting the applet viewer with your Java source code file specified as the target



MAJOR APPLET ACTIVITIES

- There are five different methods which need to be defined in your applet

  They are:

  - **init()**
  - **start()**
  - **stop()**
  - **destroy()**
  - **paint()**

  Each method corresponds to a major activity that applet must undertake during its life cycle.

  Initialization, Starting, Stopping, Destruction & ***Painting***

APPLET METHODS

init()

- The **init()** method is the first method to be called on loading the applet.

- Typical activities during applet initialization include ***reading and parsing parameters to the applet, creating any helper objects used by the applet or loading images or fonts.***

- This method is called **only once** during the run time of your applet

APPLET METHODS

start()

- Starts the execution of applet
- It is also called to restart an applet after it has been stopped
- When a user minimizes a web page to move to another page ,the execution of **start()** is stopped, where as on maximizing the web page resumes the execution of this method

stop( )

- The **stop( )** method is called whenever the user moves away from the HTML page that contains applet
- When **stop()** is called, the applet is probably running. **stop()** is used to suspend the execution of applet, so that it doesn't take system resources when the user is not viewing the page or has quit the browser
- **start()** is called if the user returns to the page.

APPLET METHODS

- The default implementations of **start()** and **stop()** methods provided by the Applet class do nothing
- These methods are overridden in complex applets which employ multiple Threads of execution
- Eg: ***Applets which display animations, play sounds or manipulate images***

destroy()

- Used to remove applet completely from the memory of the computer
- The **stop()** method is always called before **destroy().**
- In normal case this method will not be overridden

APPLET METHODS

- **paint()** is used to do something on the screen, be it text, a line, a coloured background, or an image
- **paint()** will be invoked many thousands of times during the life-time of an applet, typically in response to some change in the applet's environment.
- Following are some situations in which paint() can be invoked:
  - When the applet is created, after init() and start() have been called.
  - Whenever the browser window is resized, moved etc.
  - Whenever an obscured portion of the applet's drawing area is re-exposed (due to another window being moved)

APPLET METHODS

- Whenever the user triggers an Event which should cause a change in the appearance of the applet (e.g., a mouse click or drag, or a key press.)
- When an animation wants to change the image in the applet's window.
- We need to override **paint()** in our applet depending upon the requirement

Synatx:

public void paint(Graphics g)

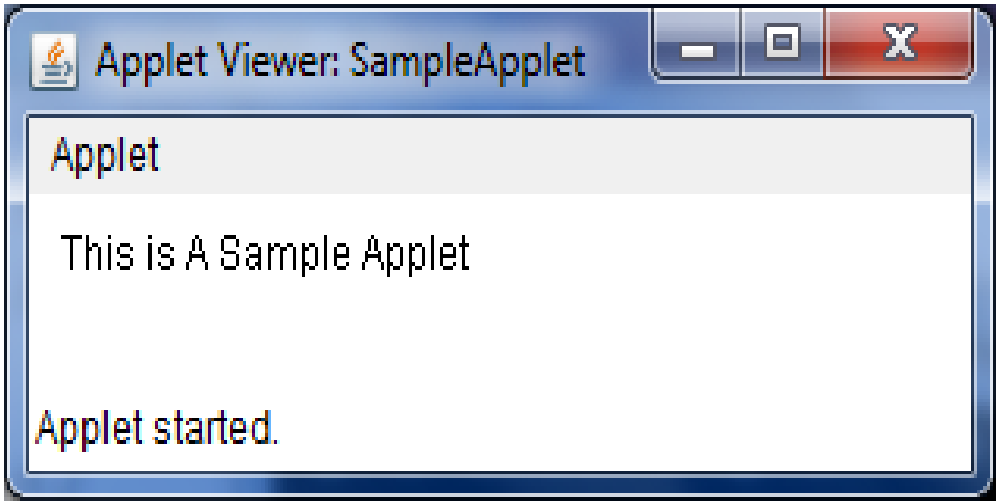
accepts a **Graphics** object as argument which describes the properties of the surface the applet is being requested to paint.

This object is created and passed to paint by the browser.

import java.awt.Graphics statement should be specified at the beginning of file containing applet code

APPLET EXAMPLE

```
import java.awt.*;
/* <applet code="SampleApplet" width=200 height=60> </applet>
 */
public class SampleApplet extends java.applet.Applet
 {
    public void paint(Graphics g)
     {
        g.drawString("This is A Sample Applet",10,60),;
       }
}
```



update() and repaint()

- The repaint() method is used whenever an applet needs to update information in its window
- AWT defines repaint() method which causes the AWT run-time system to execute update() method of an applet.
- The update() method performs the following task:
  - **It clears the screen area.**
  - **It sets the background and foreground colors**
  - **It invokes the paint() method and passes it the same graphics object it received**

  Syntax:

  public void update(Graphics g)

- User applications usually relies on indirect calls to the **update()** method through the repaint() method to update the components

repaint()

- The following are 4 forms of repaint() method:

  - **void repaint( )**
  - **void repaint(int left, int top, int width, int height)**
  - **void repaint(long maxDelay)**
  - **void repaint(long maxDelay, int x, int y, int width, int height)**

DIFFERENCE B/W APPLETS AND APPLICATION

- Applications require main method to start its execution of a java program. An applet on other hand doesn't require main method

- When an applet starts, it inherits or overrides life cycle methods such as **init(), start(),stop()** and **destroy().**During the execution of an applet, these methods are called several times. In contrast application has no life cycle methods

- An applet can be embedded in HTML pages where as an application has no support for HTML.

DIFFERENCE B/W APPLETS AND APPLICATION

- Applets can be used to create static and dynamic web pages where as applications are used to create stand alone application

- An application can run with or without graphical user interface where as an applet must run with a graphical user interface

- An applet runs under the control of web browser where as an application needs java virtual machine for its execution.

APPLET TAG

<APPLET> is a special extension to HTML for including applets in Web pages

- **Syntax:**

< APPLET

[CODEBASE = codebaseURL]
CODE = appletFile
[ALT = alternateText]
[NAME = appletInstanceName]
WIDTH = pixels HEIGHT = pixels [ALIGN = alignment]
[VSPACE = pixels]
[HSPACE=pixels] **>**

[< PARAM NAME = AttributeName VALUE =AttributeValue>]
[< PARAM NAME = AttributeName2 VALUE = AttributeValue>]

</APPLET>

APPLET TAG

- **CODE-** name of the class file that contains the applet
- **CODEBASE:** an optional attribute which specifies the directory that will be searched for the applet's executable class file
- **ALT:** Alternate text to be displayed in case browser does not support applet
- **NAME:** Defines a unique name for the applet
- **WIDTH AND HEIGHT:** WIDTH and HEIGHT are required attributes that give the size (in pixels) of the applet display area
- **ALIGN:** Defines the text alignment around the applet
- **VSPACE and HSPACE:** OPTIONAL attributes specify the number of pixels above and below the applet (VSPACE) and on each side of the applet (HSPACE)
- **<PARAM NAME = *appletAttribute1* VALUE = value> -**allows you to specify applet specific arguments in an HTML page

METHODS OF THE APPLET CLASS

- Apart from the methods that are used in the applet's life cycle, Applet class defines several other useful methods
- **void showStatus(String message):**outputs a message on the status window/bar of the browser or appletviewer where applet is running
- **URL getDocumentBase():**returns the URL of the HTML file in which applet is embedded
- **URL getCodeBase():** returns the URL of the class file that contains the applet.
- **String getAppletInfo():**Returns a string that describes the applet
- **String getParameter(String *paramName*)**:Returns the parameter associated with *paramName.*.

METHODS OF THE APPLET CLASS: EXAMPLE

```
import java.awt.*;
import java.applet.*;
import java.net.*;
/*
 <applet code="AppletMethods"  width=300 height=50>
      </applet>
*/
public class AppletMethods extends Applet
{
    // Display code and document bases.
    public void paint(Graphics g)
    {
                showStatus("This is a status message of an applet window");
                String msg;
                URL url = getCodeBase(); // get code base
                msg = "Code base: " + url.toString();
                g.drawString(msg, 10, 20);
```

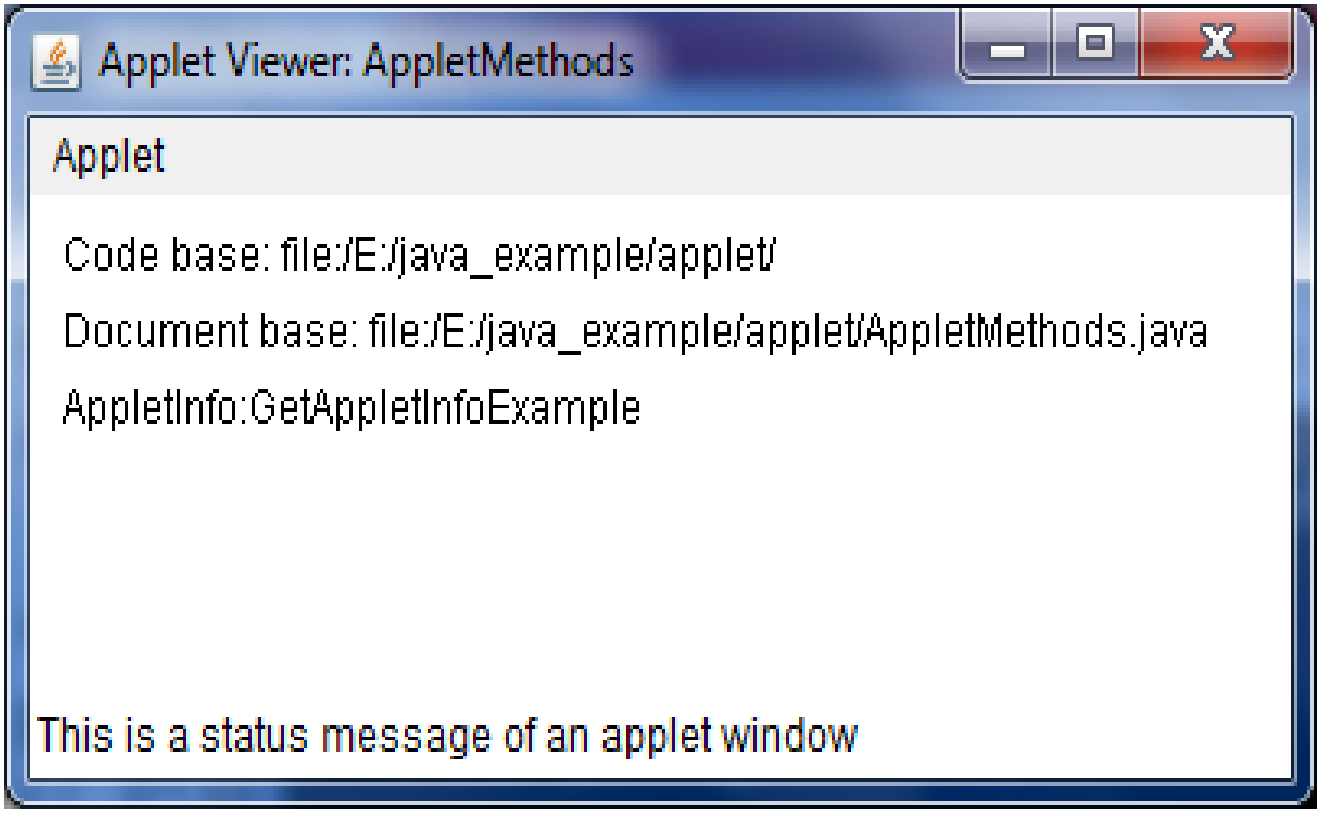
METHODS OF THE APPLET CLASS:EXAMPLE

```
    url = getDocumentBase(); // get document base
    msg = "Document base: " + url.toString();
    g.drawString(msg, 10, 40);
    msg="AppletInfo:"+getAppletInfo();//get getAppletInfo
    g.drawString(msg,10,60);
}
public String getAppletInfo()//overriding getAppletInfo
{
            String info = "";
            info = info + "GetAppletInfoExample";
            return info;
        }
}
```

METHODS OF THE APPLET CLASS:EXAMPLE

E:\java_example\applet>javac AppletMethods.java

E:\java_example\applet>appletviewer AppletMethods.java



PASSING PARAMETERS TO APPLET

- It is possible to pass parameters to an applet using the **<APPLET>** tag.
- **getParameter()** method is used to retrieve these parameters in applet code
- It returns the value of the specified parameter in the form of a String object.

EXAMPLE:

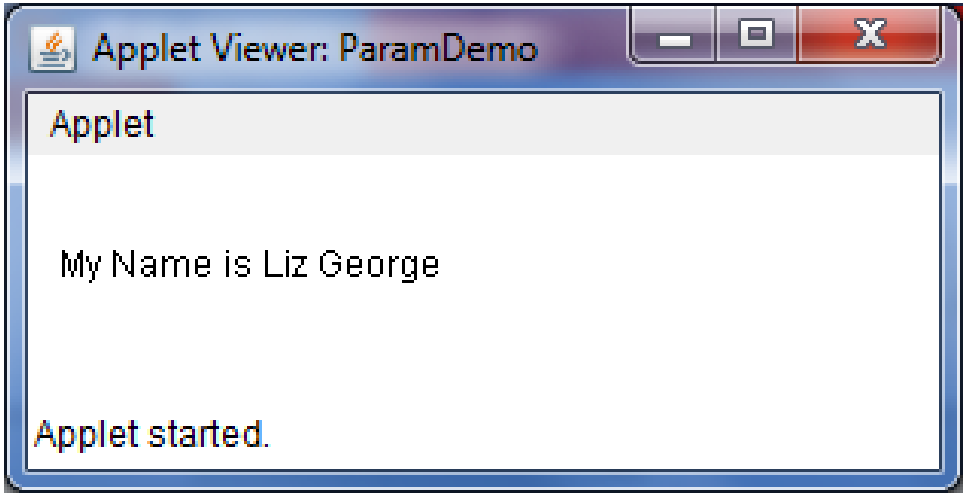
```
import java.awt.*;
import java.applet.*;


/*<applet code="ParamDemo"  width=300  height=80>
<param name=FirstName  value="liz">
<param name=LastName  value=George>
</applet>
*/
```

PASSING PARAMETERS TO APPLET

```
public class ParamDemo extends Applet
{
    String str;
  public void init()
  {
      str="My Name is "+getParameter("FirstName")+" "+
     getParameter("LastName");
 }
 public void paint(Graphics g)
 {
  g.drawString(str,10,40);

 }
}
```



GRAPHICS IN APPLET

- One of the most important features of Java is its ability to draw graphics.

- The graphics co-ordinate system in Java starts from point (0,0) in the top left-hand corner of the window and increases to the left and down and co-ordinates are specified in pixels.

- We can write Java applets that can draw lines, figures of different shapes, images, and text in different fonts, styles, and colours.

GRAPHICS CLASS

- You can draw strings, lines, rectangles, ovals, arcs, polygons, and polylines, using the methods in the **Graphics** class which belongs to **java.awt** package.
- **void drawString(String text, int x, int y):**displays a string at a given position.
- **void drawLine(int x1,int y1,intx2,inty2):** draws a line from the point(x1,y1) to the point (x2,y2);

- **void drawRect(int x, int y, int width, int height)**
- **void fillRect(int x, int y, int width, int height)** These methods draw an outline and filled rectangle whose upper left corner is specified by (x,y) and having the width and height given as arguments

GRAPHICS CLASS

- **void drawOval(int x,int y, int width,int height)**
- **void fillOval(int x,int y, int width,int height) :**These methods draw oval shapes, ie,ellipses and circles bound by an invisible rectangle whose upper-left corner is (x,y) and whose width and height are specified by arguments

- **void drawPolygon(int *x*[ ], int *y*[ ], int *numPoints*):**
- **void fillPolygon(int *x*[ ], int *y*[ ], int *numPoints*):** draws outline and filled polygon whose vertices are defined by corresponding elements of x and y arrays. The **numPoints** specify number of elements in the Polygon

- **void setColor (Color c) :**Sets this graphics context's current color to the specified color.

COLOR CLASS

- Color is a predefined class present in **java.awt** package. This class facilitates the user to make use of different colors provided by this class in the applet created.
- Color class provides 13 pre-defined colours, where each color is a constant specified by this class.
- They can be accessed by using the syntax: Color.CONSTANT_NAME
  where **CONSTANT_NAME** is one of:

  BLACK ,BLUE, CYAN,DARK_GRAY,GRAY, GREEN,LIGHT_GRAY, RED, MAGENTA,ORANGE,PINK, WHITE, YELLOW
- It is possible to generate different colours, apart from using the above said constants with the help of constructor of the Color class, which takes 3 integer values with-in the range 0-255(RGB)
- **Color(int *red*, int *green*, int *blue*);//constrcutor**

  Ex: `Color brown = new Color(192, 128, 64);`

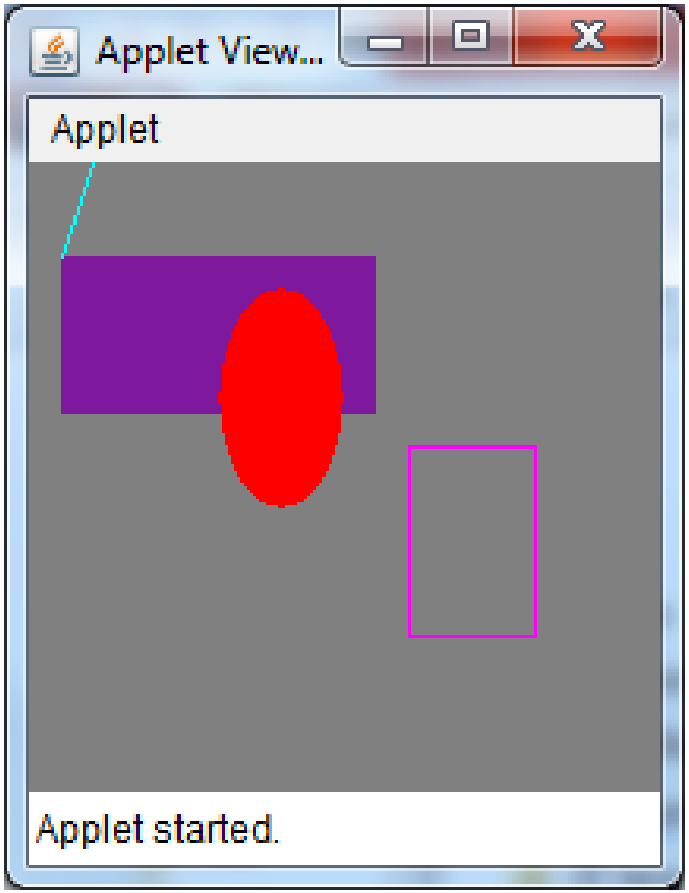
COLOR CLASS

- **void setBackground(Color c):**Used to set the background color
- **void setForeground(Color c)**:Used to set the foreground color (the color in which text is shown)

```
import java.awt.*;
import java.applet.*;
 /*<applet width=200 height=200 code="ColorTest">  </applet> */
public class ColorTest extends Applet {
public void paint(Graphics g)
  {       setBackground(Color.GRAY);
                        Color cl=new Color(126,25,157);
                        g.setColor(cl);
                            g.fillRect(10, 30, 100, 50);
```

COLOR CLASS

```
 g.setColor(Color.MAGENTA);
      g.drawRect(120,90,40,60);
g.setColor(Color.CYAN);
          g.drawLine(20, 0, 10, 30);
g.setColor(Color.RED);
          g.fillOval(60, 40, 40, 70);
} }
```



FONT CLASS

- The java.awt package contains a **Font** class which provides the mechanism for setting the attributes of the font used
- **Constructor:**

  Font(String ftype, int fstyle, int fsize);

ftype -refers to the name of the font

fstyle-refers to the style of the font, which is a constant defined in Font class which takes values like **BOLD,ITALIC,PLAIN** etc

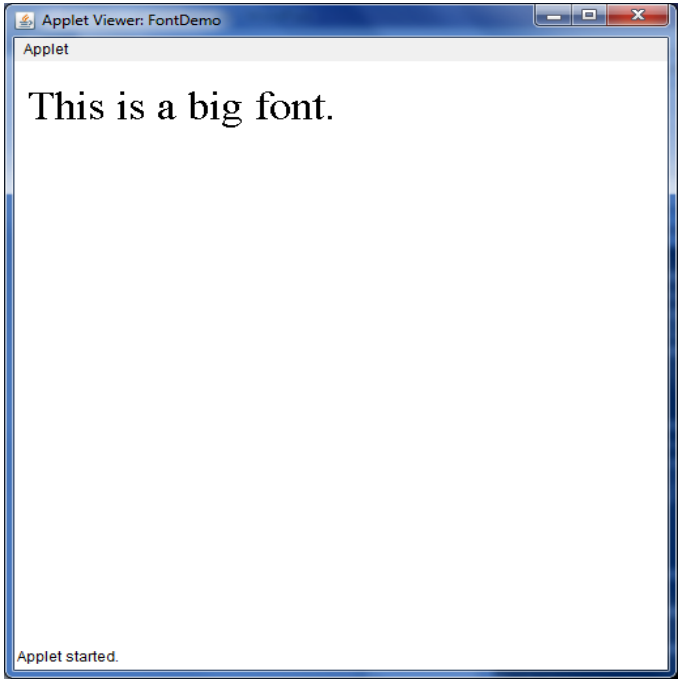
fsize-refers to the size of the font

EXAMPLE:

```
import java.awt.*;
import java.applet.*;
/*<applet width=500 height=500 code="FontDemo">
   </applet>
*/
```

FONT CLASS

```
public class FontDemo extends Applet
{           public void paint(Graphics g)
            {
                Font f = new Font("TimesRoman", Font.PLAIN, 34);
                g.setFont(f);
               g.drawString("This is a big font.", 10, 50);
            }
}
```



ADDING IMAGES TO AN APPLET

- Applets have built in support for images, including the ability to decode and display image format files.
- The applet's **getImage()** method returns an object of type Image which you can display in the paint method using the Graphics method **drawImage()**

- Syntax:

Image getImage(URL *url* , String imageName)

boolean drawImage(Image *imgObj*, int *left*, int *top*, ImageObserver *imgOb*)

- Where *imgObj*, is your image object, *left* and *top* represent the position to draw the top-left corner of the image.
- An ***imageobserver*** is an object that will monitor an image while it loads

ADDING IMAGES TO AN APPLET

EXAMPLE:

```
import java.awt.*;
import java.applet.*;
public class GetImg extends Applet
{
    Image buimg;
     public void init() {
      buimg = getImage(getCodeBase(),"coin.jpg");
    }
     public void paint(Graphics g) {
      g.drawImage(buimg, 60,60,this);
   }
}
```

ADDING IMAGES TO AN APPLET

Consider the statement:

g.drawImage(buimg, 60,60,this);

- It allows us to draw the image object that we loaded above to the (x,y) position (60,60) and sets the image observer as **this.**

- The Java keyword this refers to -"this instance of our Applet class". So our applet is concerned with notification of the image loading.

PLAYING SOUND IN AN APPLET

- Applets also have support for playing sound. The easiest way to play sounds in applets is by using the play() method
- Syntax:

void play(URL url, String clipName)) ;

- If an audio clip is found at the location specified by **url** with the name specified by **clipName**, the clip is played
- For a more complicated use of sounds getAudioClip() method can be used.
- Syntax:

AudioClip getAudioClip(URL url, String name)

returns an object of the java.applet.AudioClip that encapsulates the audio clip found at the location specified by url

We can then use the **play(), loop()** or **stop()** methods of this object

PLAYING SOUND IN AN APPLET

```
import java.applet.*;
import java.awt.*;
public class GetAudio extends Applet
 {
    public void paint(Graphics g)
    {
          g.drawString("Playing music",20,20);
          play(getCodeBase(),"wwind.wav");
   }
 }
```

PLAYING SOUND IN AN APPLET

```
import java.applet.Applet;
import java.applet.AudioClip;
/*<applet code=GetAudClip width=300 height=300>
 </applet>*/
public class GetAudClip extends Applet {
  public void init() {
    // Load audio clip
    AudioClip ac = getAudioClip(getCodeBase(),"wwind.wav");

    // Play audio continuously
    ac.loop();
  }
}
```