Chapter 16

APPLETS

In this chapter, you will learn about applets. Applets are a second kind of program supported by Java language. Applets are programs that travel across a network as bytecodes, load automatically in the local machine and are executed by web browsers. The architecture of an applet, how to create and embed them in an HTML document are discussed in this chapter.

16.1 Applet Basics

Applets are a second kind of programs that Java supports. Applets are programs that can be downloaded from a foreign machine through a network and executed in a local machine. Applets are not controlled by the local operating system. Applets are split into small packets called bytecodes and travel across the network. These bytecodes are reassembled by the Java Virtual Machine (JVM) in the receiving machine and are executed by the browser. Applets cannot access the local resources like hard disk and floppy. Applets cannot be executed directly. An applet can only be an element of an HTML page. Therefore, an applet can be executed only by executing an HTML page. These HTML pages, as usual, are executed by web browsers. For testing purpose, the **applet** element of an HTML page can be executed by **appletviewer** provided in the JDK.

Applets do not use **main()** method and **System.out.println()** method. Applets use only graphics methods for output. They generate window-based output. Therefore, they need graphics support. To draw graphics output, they make use of Java's Abstract Window Tool kit(AWT). Applets are defined in Applet class. Therefore, to write an applet program, the java.applet and java.awt packages are to be imported. The Applet class hierarchy is shown in fig.16.1. The **JApplet** is defined in **Swing**.

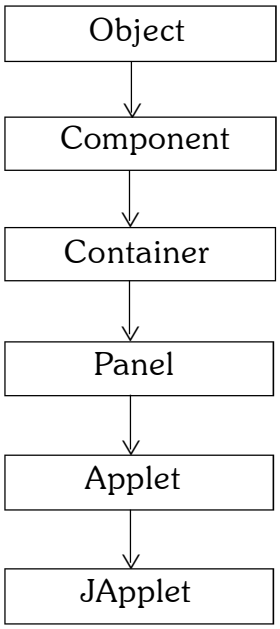


Fig.16.1 Class Hierarchy of Applet Class

A simple applet program is given in program 16.1.

Program 16.1

```
// This is a simple applet.

import java.applet.*;
import java.awt.*;
public class Myapplet
    extends Applet
    {
    public void paint(Graphics g)
        {
        g.drawString("Welcome to Applet World", 20, 40);
        }
    }
```

Compile this program as has been done for application program as below:

C>javac Myapplet.java

Then create test1.htm file with the following applet element. In the simplest form, the applet element is:

<applet code = Myapplet width = 200 height = 150 >
<\applet>

Then execute the htm file with **appletviewer** as :

C>appletviewer test1.htm

The output of this program is given below:

Fig.16.2 Output Screen for Program 16.1 When Executed With Aappletviewer

The same test1.htm viewed through the Internet Explorer is given below:

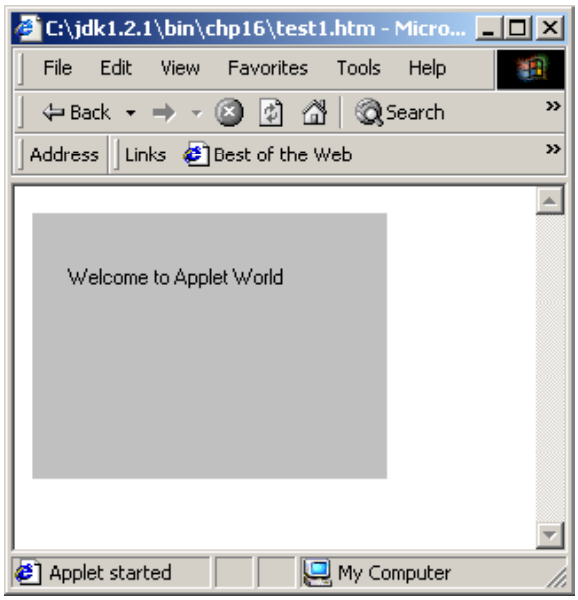


Fig.16.3 Output Screen of Program 16.1 When Viewed With Internet Explorer

In the above program, the applet window has a width of 200 pixel and height of 150 pixel. The String "Welcome to Applet World" is displayed at the x,y co-ordinate 20,40 of the display area.

Instead of creating two programs, an applet program and another htm file, to execute the applet, the applet element of htm file can be embedded within the applet program itself as a comment. By this method, the htm and applet program can be combined and executed as a single program. The combined applet is given in program 16.2.

Program 16.2

```
import java.applet.*;
import java.awt.*;
/*
<applet code = applet1 width = 200 height = 150>
</applet>
*/
// This is a simple applet
public class applet1
    extends Applet
    {
    public void paint(Graphics g)
        {
         g.drawString("Welcome to Applet World", 20, 40);
        }
    }
```

First compile the program 16.2 using **javac** compiler. Then execute the program using **appletviewer** and specifying the applet's source file.

C>appletviewer applet1.java

The result of the above program 16.2 is the same as that of program 16.1.

Applets rely for input and output on AWT. All user interactivity with the applet is event-driven, such as mouse click. Therefore, all elements that compose an applet can be understood only after knowing the details of event handling, which is discussed in chapter 18. In the following sections more basics of applet are given.

Applets can only be a part of HTML page and are executed by web browsers. However, during the development stage, applets can be executed directly using appletviewer.

16.2 Methods of Building an Applet

Applets are created, executed, stopped and destroyed by appropriate methods provided in **Applet** class. All these activities are carried out by the following methods. These methods are called by AWT automatically. Appropriate methods can be overridden to meet the users requirement.

a) **init()**
This method is used to initialize the variables of the applet. This is the method called first. This is called only once.

b) **start()**
This method is called automatically after the **init()** method. It is also called whenever a user returns to the page containing the applet after going to a different page. **start()** method can be called repeatedly while the **init()** method is called only once, when the applet is loaded. It is in this method that a thread is restarted.

c) **paint()**
This method is called automatically after the **start()** method. The user screen is drawn using the **paint()** method. This method has an argument of type **Graphics** defined in AWT. The **paint()** method is called when the applet begins execution.

d) **stop()**
This method is called automatically when a user leaves the page containing the applet. Hence, it can be called repeatedly. This helps the suspension of the activity when the user is not using the applet. Otherwise, it will slow down the system performance.

e) **destroy()**
This method is called automatically when the browser ends the activity. This method helps to release the resources used by the applet. A user has to put all the codes required to wind up the final activities when the applet is removed from the memory of the system. The **stop()** method is called automatically before **destroy()** method.

Program 16.3 gives an illustration to write an init, start, stop and destroy methods. The strings mesg1 to mesg4 are not assigned initial values. They are assigned string literals when the respective methods are executed. The string mesg3 is assigned a text inside the **destroy()** method. Statements inside **stop** and **destroy** methods will be executed only when the applet is stopped and destroyed. This can be done by restarting the applet.

Program 16.3

```
// This program illustrates the basic methods of applet.
import java.applet.*;
import java.awt.*;
/*
<applet code = Allmethod  width = 250 height = 200>
</applet>
*/
public class Allmethod
    extends Applet
    {
    String mesg1, mesg2, mesg3, mesg4;
    public void init()
        {
        mesg1 = “message from init  method”;
        }
    public void start()
        {
        mesg2 = “message from start method”;
        }
    public void stop()
        {
        mesg3 = “message from stop method”;
        }
    public void destroy()
        {
        mesg4 = “system is destroying your applet”;
        }
    public void paint(Graphics gp)
        {
        gp.drawString(“Demo for basic methods of
                          applet”, 20, 40);
        if (mesg1 != null)
            gp.drawString(mesg1, 20, 80);
        if (mesg2 != null)
            gp.drawString(mesg2, 20, 100);
        if (mesg3 != null)
            gp.drawString(mesg3, 20, 120);
        if (mesg4 != null)
            gp.drawString(mesg4, 20, 140);
        }
    }
```

The applet when executed gives the following output screen:

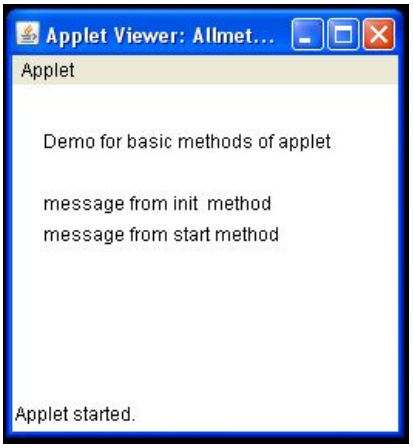


Fig.16.4 Output Screen for Program 16.3

In the output window, select Applet and click Restart. This will destroy the applet and start again. As a result, mesg3 and mesg4 are assigned values and displayed in the new screen as shown below:

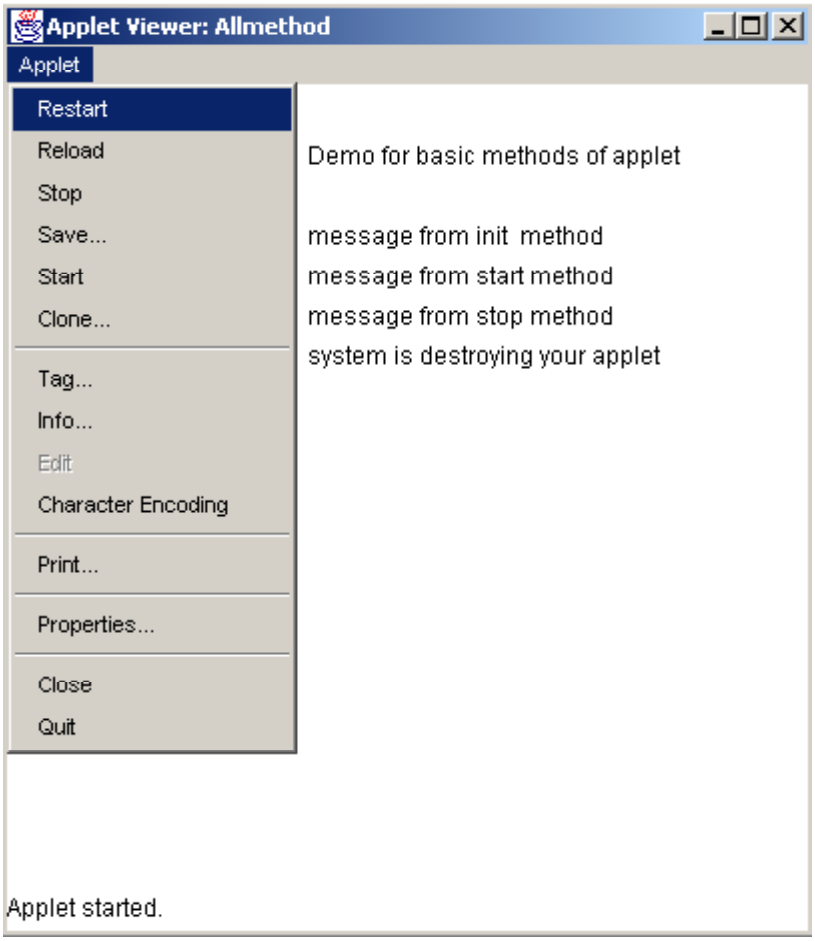


Fig.16.5 Output Screen for Program 16.3 After Restart

The use of **init()** method to set background and foreground colors of a window is given in program16.4:

Program 16.4

```
// This program illustrates the init method.
import java.applet.*;
import java.awt.*;
/*
<applet code = Appinit  width = 200 height = 200>
</applet>
*/
public class Appinit
    extends Applet
    {
    public void init()
        {
        setBackground(Color.green);
        setForeground(Color.red);
        }
    public void paint(Graphics gp)
        {
        gp.drawString("init method illustration", 20, 40);
        }
    }
```

The above program gives the following output screen. The window area is in green background and the foreground text is in red color.

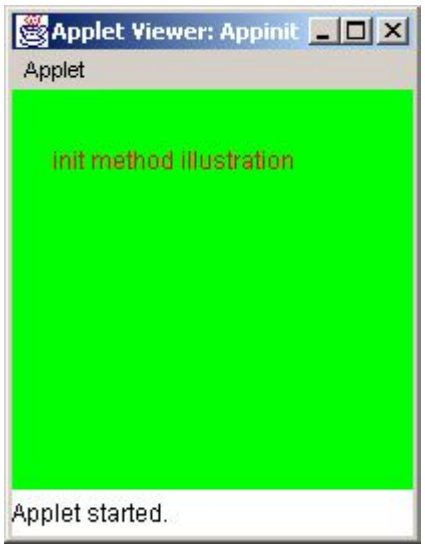


Fig.16.6 Output Screen for Program 16.4

16.3 Some General Methods of Applet

Some of the general methods defined in **Applet** class are given in table 16.1.

Table 16.1 Some General Methods Defined in Applet Class

Method	Purpose of the Method
1. String getAppletInfo()	Returns the string that describes the applet
2. URL getCodeBase()	Returns the URL associated with the applet
3. URL getDocumentBase()	Returns the URL of the HTML page that contains the applet
4. String getParameter(String pname)	Returns the string associated with the parameter pname
5. String[][] getParameterInfo()	Returns a string table that is defined in the applet
6. boolean isActive()	Returns true if the applet has been started; returns false if the applet has been stopped
7. void resize(int width, int height)	Resizes the window according to width and height
8. void showStatus(String s)	Displays the String s in the status window of the browser

16.4 Displaying Text in Status Bar

Text in the status bar of the browser window can be displayed using the **showStatus()** method. Program 16.5 shows the use of this method and **isActive** method.

Program 16.5

```
// This program illustrates the use of isActive and
// showStatus methods.
/*
<applet code = StatusBar width = 300 height = 100>
</applet>
*/
import java.applet.*;
import java.awt.*;
public class StatusBar
```

```
extends Applet
{
public String state()
    {
    if (isActive())
        return " applet is alive";
    else
        return "applet is dead";
    }
public void paint(Graphics gp)
    {
    gp.drawString(" status bar demo", 80, 40);
    gp.drawString(state(), 80, 60);
    showStatus("Text  for the status bar");
    }
}
```

The output screen for the above program is given below:

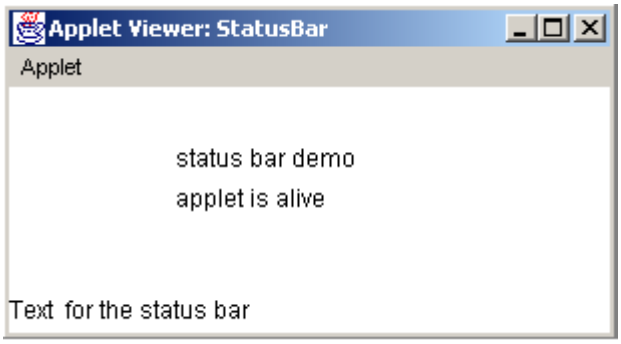


Fig.16.7 Output Screen for Program 16.5

16.5 Embedding Applet Information

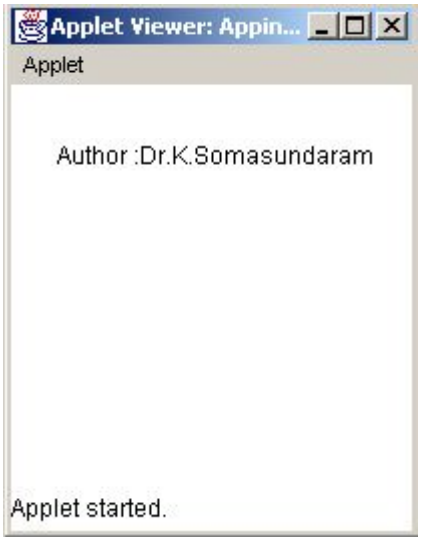
Information or description about the applet can be defined in the **getAppletInfo** method and can be used in the user window. This method along with **resize** is given in program 16.6.

Program 16.6

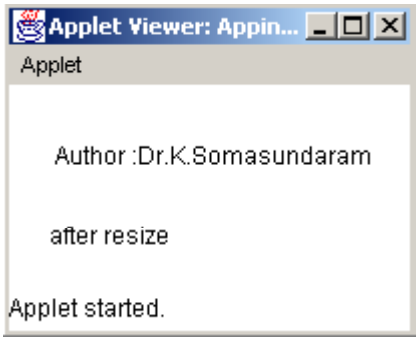
```
// This program illustrates the getAppletInfo method.
import java.applet.*;
import java.awt.*;
/*
<applet code = Appinfo width = 200 height = 200>
</applet>
*/
public class Appinfo
    extends Applet
    {
```

```
public String getAppletInfo()
    {
    return " Author :Dr.K.Somasundaram";
    }
public void paint(Graphics gp)
    {
    gp.drawString(getAppletInfo(), 20, 40);
   try
        {
        Thread.currentThread().sleep(3000);
        }
    catch (InterruptedException e)
        {
        ;
        }
    resize(200, 100);
    gp.drawString("after resize", 20, 80);
    }
}
```

The output screen for the above program before and after resize are given below:



(a)



(b)

Fig.16.8 Output Screen for Program 16.6.
(a) Before Resize b) After Resize

16.6 The HTML Applet Tag

An applet can be an element of an HTML page. The applet statement itself has its own structure and is called applet tag. The individual components define behaviors and properties of an applet. The general structure of an applet tag is given below. Those written with [...] are optional and the other entries are mandatory for an applet.

```
<APPLET
[CODEBASE = codebase URL]
CODE = appletName
[archive = archivefile]
[ALT = alternateText]
[NAME = appletInstanceName]
WIDTH = pixels HEIGHT = pixels
[ALIGN = alignment]
[USPACE = pixels]
[HSPACE = pixels]
>
[ <PARAM NAME = attributeName VALUE = value>]
[ <PARAM NAME = attributeName VALUE = value>]
 .
 .
[text to be displayed in the absence of non-java browser]
</APPLET>
```

CODEBASE

It defines the URL that contains the applet code. If CODEBASE is not defined, the URL of the HTML document is used.

CODE

It specifies the name of the class file of the applet. It is a mandatory requirement. This file is relative to the code base URL of the HTML document or as specified by CODEBASE.

ARCHIVE

It specifies the archive (JAR) file name. It can be used when a group of files are needed for applet execution.

ALT

It specifies the text that is to be displayed if the browser could understand the applet tag but could not execute the applet. It is an optional attribute.

NAME

It specifies the name for the applet instance. Names help one applet to find another applet on the same page to communicate with them. It is an optional attribute.

WIDTH and HEIGHT

This specifies the width and height of the applet display area. It is given in pixels and they are integer numbers.

ALIGN

It specifies the alignment of the applet in the web page. The predefined values are **LEFT**, **RIGHT**, **TOP**, **BOTTOM**, **MIDDLE**, **BASELINE**, **TEXTTOP**, **ABSMIDDLE** and **ABSBOTTOM**.

PARAMNAME and VALUE

Parameters can be defined, given a name and a value. **PARAM** declares it is a parameter, **NAME** declares the name of the parameter and **VALUE** specifies the value that the variable defined by **NAME** can take. The value can be passed into the applet using **getParameter()** method. It is an optional attribute.

VSPACE and HSPACE

VPSACE specifies the space above and below the applet in pixels and HSPACE specifies the space on the left and right side of the applet in pixels.

Using HTML applet tag, applet codes stored in a remote URL server can be brought to local machines and executed using web browsers.

16.7 Reading Parameters into Applets

Applets can read data defined in the applet tag using **PARAM** attribute. It is one of the ways to bring data into the applet. The **getParameter()** returns a String type. Therefore, if numeric values are defined, they are to be converted from String to numeric types. Program 16.7 shows the use of **getParameter()** method.

Program 16.7

```
// This program illustrates the getParameter() method.
import java.applet.*;
import java.awt.*;
/*
<applet code = Param width = 250 height = 200>
<param name = author value = "Dr.K.Somasundram">
<param name = age value = "50">
<param name = designation value = "Professor">
<param name = institute value = "Gandhigram Rural
                                        Institute">
</applet>
*/
```

```
public class Param extends Applet
    {
    public void paint(Graphics gp)
      {
      String au = getParameter("author");
      String ag = getParameter("age");
      int balance = 62 - Integer.parseInt(ag);
      String desg = getParameter("designation");
      String inst = getParameter("institute");
      gp.drawString("Author : " + au, 20, 40);
      gp.drawString("Age : " + ag, 20, 60);
      gp.drawString("Designation: " + desg, 20, 80);
      gp.drawString("Affliation : " + inst, 20, 100);
      gp.drawString("Will retire after : " + balance +
                                " years", 20, 120);
      showStatus("Parameter method");
      }
    }
```

The output screen for the above program is given below:

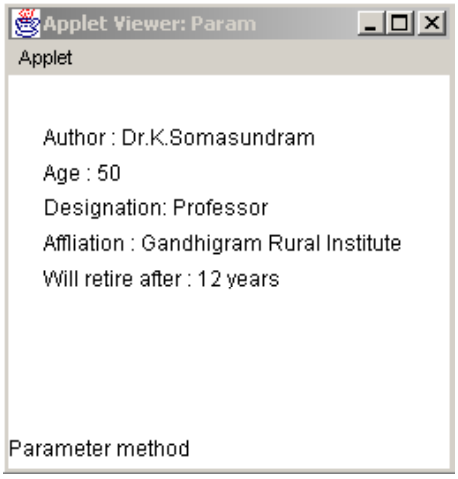


Fig.16.9 Output Screen for Program 16.7

16.8 Colors in Applet

The background and foreground colors of an applet can be set using the values defined in **Color** class. The color values defined in **Color** class are :

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

The methods used for setting the foreground and background colors are defined in **Component** and are given below:

i) void setBackground(Color colorvalue)
ii) void setForeground(Color colorvalue)

where, the colorvalue specifies color values defined in the **Color** class. Program 16.4 given earlier illustrates the above two methods.

It is also possible to detect the colors used in an applet window. The methods are :

i) Color getBackground()
ii) Color getForeground()

16.9 Getting Documentbase and Codebase

The **URL** that contains the applet code is called codebase. The **URL** that contains the HTML page is called documentbase. These two **URLs** can be obtained using **getCodeBase()** and **getDocumentBase()** methods as defined in table 16.1.

The program 16.8 illustrates the use of the above two methods:

Program 16.8

```
// This program illustrates the getDocumentBase()
// and getCodeBase() methods.
import java.applet.*;
import java.awt.*;
import java.net.*; // needed to obtain URL values
/*
<applet code = Docbase width = 350 height = 100>
</applet>
*/
public class Docbase
    extends Applet
    {
    public String getAppletInfo()
        {
        return " Author :Dr.K.Somasundaram";
        }
    public void paint(Graphics gp)
      {
      URL docb = getDocumentBase();
      URL codb = getCodeBase();
      gp.drawString("Documentbase : " + docb, 20, 40);
      gp.drawString("Codebase        : " + codb, 20, 60);
      showStatus(getAppletInfo());
      }
    }
```

The above program gives the following screen output:

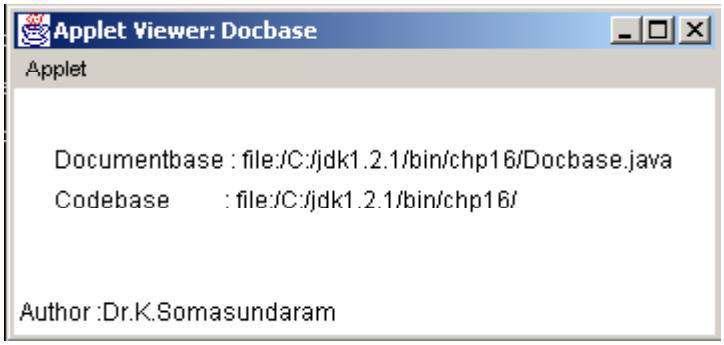


Fig.16.10 Output Screen for Program 16.8

16.10 Interfaces in Applet

Applet package has three interfaces **AppletContext**, **AppletStub** and **AudioClip**. The **AppletContext** interface contains methods which can be used to get information about the applet's environment. The **AppletStub** contains methods which can serve as interface between browsers. The **AudioClip** interface has methods to control audio clips.

16.11 Multimedia in Applet

The multimedia elements sound and image can be brought into the applets. Methods defined in Applet class to handle sound, image and related methods are given in table 16.2.

Table 16.2 Multimedia Methods Defined in Applet class

Method	Purpose of the Method
1. AudioClip getAudioClip (URL url)	Returns the AudioClip object located at the site url
2. AudioClip getAudioClip (URL url, String fname)	Returns the AudioClip object located at the site url with name fname
3. static final AudioClip newAudioClip(URL, url)	Returns an AudioClip specified at the location url
4. void play(URL url)	Plays an AudioClip found at the location url
5. void play(URL url, String clipName)	Plays the audio clip clipName found at the location url
6. Image getImage(URL url)	Returns an Image object found at the location url
7. Image getImage(URL url, String imageName)	Returns an Image object imageName found at the location url

16.11.1 Playing Audio Clips

Java applet supports audio clips of formats au, aiff, midi, wav and rmf. Audio clips can be played directly by calling the **play()** method and specifying the URL and the audio clip name. Program 16.9 shows the use of **play()** method to play an audio clip.

Program 16.9

```
// This program illustrates the play() method.
import java.applet.*;
import java.awt.*;
import java.net.*; // needed to obtain URL values
/*
<applet code = Appaudio1 width = 200 height = 100>
</applet>
*/
public class Appaudio1
    extends Applet
    {
    public String getAppletInfo()

        {
        return " Author :Dr.K.Somasundaram";
        }
    public void paint(Graphics gp)
      {
      gp.drawString("Playing music ", 20, 40);
      URL codb = getCodeBase();
      play(codb, "shore.aif");
      showStatus(getAppletInfo());
      }
    }
```

The output of the above program is given below:

Fig.16.11 Output Screen for Program 16.9

The AudioClip interface has three methods to control the audio clip. They are:

i) void play()
Starts playing this audio clip

ii) void loop()
Starts playing this audio clip in loop

iii) void stop()
Stops playing this audio clip

Multiple audio clips can be played at the same time. The resulting sound is a mixture of all the clips. The following program 16.10 plays one audio clip using the **loop()** and **getAudioClip()** methods.

Program 16.10

```
// This program illustrates the getAudioClip()  and loop() //
methods.
import java.applet.*;
import java.awt.*;
import java.net.*; // needed to obtain URL values
/*
<applet code = Appaudio2 width = 200 height = 100>
</applet>
*/
public class Appaudio2
    extends Applet
    {
    URL codb;
    AudioClip music;
    public void init()
      {
      codb = getCodeBase();
      music = getAudioClip(codb, "bird.au");
      music.loop();
      }
    public void stop()
        {
        music.stop();
        }
    public void paint(Graphics gp)
      {
      gp.drawString("Playing music ", 20, 40);
      showStatus("getAudioClip() and loop() methods");
      }
  }
```

The output screen for the above program is given below:

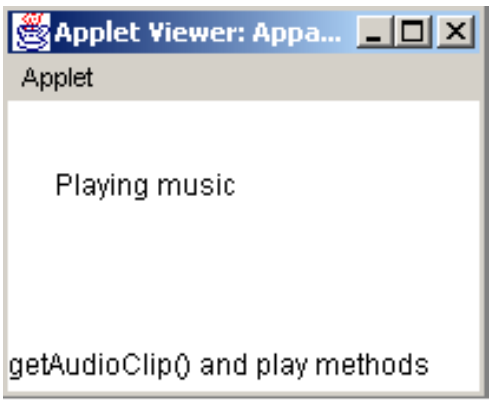


Fig.16.12 Output Screen for Program 16.10

16.11.2 Images in Applet

Still images from a specified **URL** can be obtained using **getImage()** method. Images of format JPEG and GIF are supported for images. The images can be drawn inside an applet using **drawImage()** method defined in **Graphics** class. The following program 16.11 shows how to bring in a still image in JPG format.

Program 16.11

```
// This program illustrates the getImage() method.
import java.applet.*;
import java.awt.*;
import java.net.*; // needed to obtain URL values
/*
<applet code = Appimage2 width = 400 height = 400>
</applet>
*/
public class Appimage2 extends Applet
    {
    URL codb;
    Image picture;
    public void init()
      {
      codb = getCodeBase();
      picture = getImage(codb, "magesh.jpg");
      }
    public void start() { }
    public void paint(Graphics gp)
      {
       gp.drawImage(picture, 10, 10, this);
       showStatus("   S.Magesh");
      }
    }
```

The above program gives the following output:

Fig.16.13 Output Screen for Program 16.11

Animated GIF images can also be handled in the applet. Program 16.12 shows a way to handle an animated GIF image.

Program 16.12

```
// This program illustrates the getImage() method.
import java.applet.*;
import java.awt.*;
import java.net.*; // needed to obtain URL values
/*
<applet code = Appimage1 width = 200 height = 150>
</applet>
*/
public class Appimage1
    extends Applet
    {
    URL codb;
    AudioClip music;
    Image picture1;
    public void init()
      {
      setBackground(Color.green);
      codb = getCodeBase();
      picture1 = getImage(codb, “sample1.gif”);
      music = getAudioClip(codb, “instrument.mid”);
      music.loop();
      }
    public void paint(Graphics gp)
```

```
    {
    gp.drawImage(picture1, 10, 10, this);
    showStatus("getImage() and drawImage methods");
    }
}
```

The above program gives the following output:

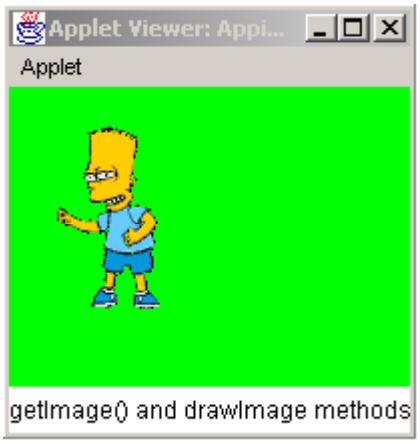


Fig.16.14 Output Screen for Program 16.12

Multimedia elements sound and image can be included in applets and controlled.

16.11.3 Applet Showing Other HTML Pages

One applet can bring another valid HTML document to view using **showDocument()** method. This method is defined in the **AppletContext** interface. There are two forms of this method. They are:

i) void showDocument(URL url)
Shows the document specified by the URL; this method does not work with appletviewer. It works only with browsers.

ii) void showDocument(URL url, String location)
Shows the document specified by the URL at the location specified by the String; does not work with appletviewer; needs a browser

The predefined locations on the browser window are:

"_self" - show the document in the current frame
"_parent" - show the document in the parent window
"_top" - show the document in the topmost frame
"_blank" - show the document in a new top level window

Other String - show the document in the frame with that name; if no such named frame exists, show it in a frame with that name.

Program 16.13 shows the use of **showDocument** method.

Program 16.13

```
// This program illustrates the showDocument() method.
import java.applet.*;
import java.awt.*;
import java.net.*; // needed to obtain URL values
public class Showdoc
    extends Applet
    {
    URL codb;
    AppletContext ac;
    Image image1;
    public void init()
      {
      ac = getAppletContext();
      codb = getCodeBase();
      image1 = getImage(codb, “sample1.gif”);
      try
            {
            ac.showDocument(new URL(codb +
                     “Appaudio2.htm”), “_blank”);
            }
        catch (MalformedURLException e)
            {
            showStatus(“URL not found”);
            }
        }
    public void paint(Graphics gp)
        {
        gp.drawImage(image1, 0, 0, this);
        }
    }
```

Write the following HTML document with a file name showdoc.htm :

```
<html>
<head>
<h1> Demo for ShowDocument method</h1>
</head>
<body>
<applet code = "Showdoc.class" width = 300 height = 300>
</applet>
</body>
</html>
```

Then view the showdoc.htm using a web browser. The screen output using Internet Explorer is given below:

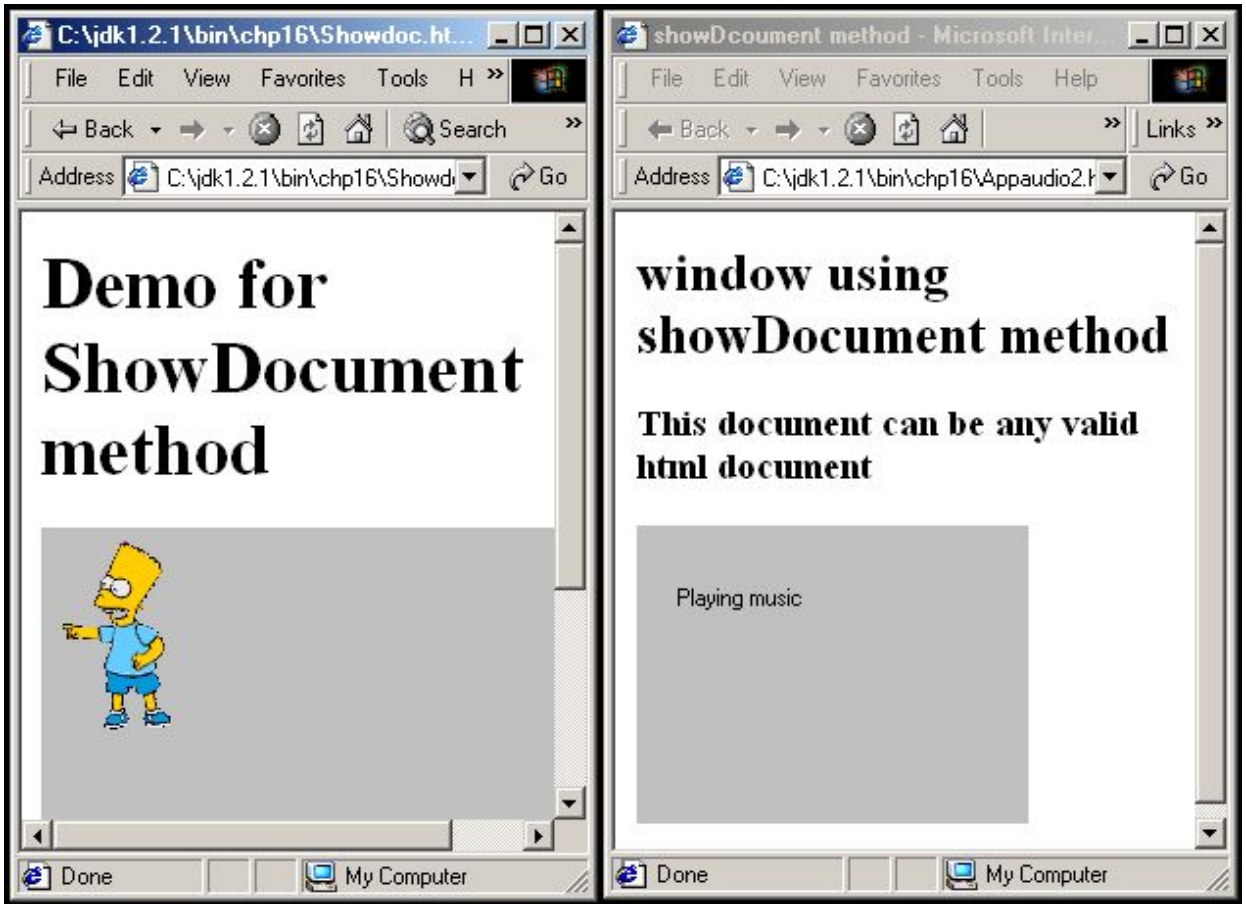


Fig.16.15 Output Screens for Program 16.13 First is the parent window and the second is the result of showDocument() method.

After reading this chapter, you should have learned the following:

- Applets are programs that can be accessed from a remote server and executed in a local machine, using a web browser.
- Appelts can only be a part of HTML page.
- Multimedia elements like text, sound and image can be handled in an applet.
- Applets are designed for network application.

In the next chapter, you will learn about graphics.

Worked Out Problem-16

Problem 16.1w

Write an applet to display the date and time. The time is to be shown like a digital clock.

Program 16.1w

```
/* ------------------------------------------------------------
 This program displays the date and time like a real clock.

 somasundaramk@yahoo.com
------------------------------------------------------------ */

import java.util.*;
import java.applet.*;
import java.awt.*;
/*
<applet code = Prob161  width = 480 height = 200>
</applet>
*/

public class Prob161
    extends Applet
    {
    String date;
    String time;
    int month, day, year;
    Font ft;
    int msec, sec, min, hr;
    GregorianCalendar calendr;
    public void init()
          {
          setBackground(Color.green);
          setForeground(Color.black);
          //set font size and type
          ft = new Font("Courier", Font.PLAIN, 70);
          setFont(ft);
          }
    public void paint(Graphics gp)
          {
          // creating current date
          calendr = new GregorianCalendar();
          day = calendr.get(Calendar.DATE);
          month = calendr.get(Calendar.MONTH);
          month += 1; // January is 0 hence this conversion
          year = calendr.get(Calendar.YEAR);
          date = " " + day + "-" + month + "-" + year;
          msec = calendr.get(Calendar.MILLISECOND);
```

```
        sec = calendr.get(Calendar.SECOND);
        min = calendr.get(Calendar.MINUTE);
        hr = calendr.get(Calendar.HOUR);
        time = " " + hr + ":" + min + ":" + sec;
        gp.drawRect(5, 5, 470, 190);
        gp.drawString(date, 20, 80);
        gp.drawString(time, 20, 170);
        try
        {
          Thread.currentThread().sleep(500);
          }
      catch (InterruptedException ie)
          {
          ;
          }
     repaint();
     showStatus("Clock Display");
      }
}
```

The above program gives the following output:

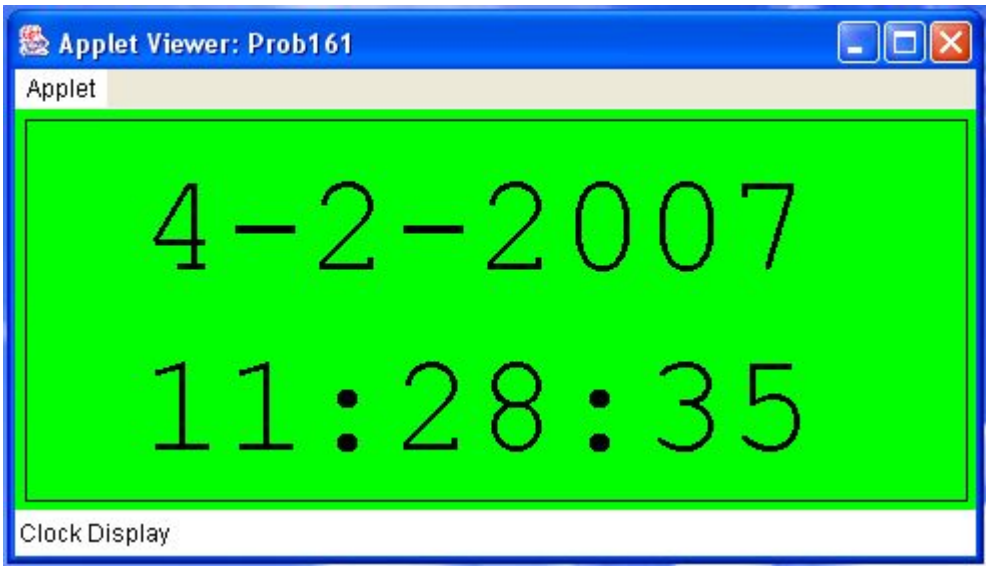


Fig.16.16 Output Screen for the Program 16.1w

* * * * * *

Exercise-16

I. Fill in the following:

16.1. Java language supports two types of programs _____ and _____ .

16.2. Applets travel in the network as _____ .

16.3. The applet is constructed and executed by _____ .

16.4. Applet is defined in _____ class.

16.5. Applets need methods defined in _____ for their output.

16.6. An applet does not need _____ method as in the application program.

16.7. An applet can be executed only by placing the _____ tag in _____ page.

16.8. Any variable in an applet can be initialized using _____ method.

16.9. _____ method is called automatically before the destroy() method is called.

16.10. _____ method removes the applet from the memory.

16.11. The URL of the HTML page that contains the applet code is called _____ .

16.12. The URL that contains the applet class is called _____ .

II. Write applets for the following problems:

16.13. Write an applet that displays your address on the screen. Run it using a browser (after writing an HTML tag) and appletviewer.

16.14. Write an applet for the problem in question 16.13 by taking the address values from parameter.

16.15. Write an applet to play an audio clip in wav format.

16.16. Write an applet to get an image using getImage() method. Draw the image and also play an audio clip in a loop.

* * * * * *

Chapter 17

GRAPHICS

In this chapter, you will learn about graphics and methods used for drawing lines and regular shapes like rectangle, oval, polygon and polyline. Some basic methods used for handling fonts are also given.

Java provides a variety of tools for designing graphics and graphical user interface. All such tools are provided in Abstract Window Toolkit(AWT) package. This package contains a large number of classes. The Swing classes, which have better capability than AWT, are discussed in Chapter 19. In this chapter, we will see the graphics methods to generate shapes, different fonts and colors. Generally, a window is created first and graphic objects are drawn on it. One way to create a window is by using an applet. An applet creates a window on which graphics objects can be drawn. Applet window is a closable one, i.e. by clicking the close icon(x) on the window, the window can be closed. Another way to create a window is by using the frame created in **Frame** class. A frame window is not closable by clicking the close icon(x) on the window. An appropriate method is to be used to close it, which we will see in the next chapter. **Graphics** tools are available in **java.awt.Graphics** class. The applet is obtained from **java.applet.Applet** class. The frame window is obtained from **java.awt.Frame** class. Since **Graphics** class is an abstract class, a graphics context is obtained indirectly.

One way to get **Graphics** context is by passing **Graphics** as a parameter to the **paint()** or **update()** methods. In the examples to follow, applet windows are used to draw the graphic objects. A few examples are also given using Frame window.

To draw graphics objects, a window is needed. One way to obtain a window is by creating an applet which automatically creates **Panel** type window. Alternatively, a Frame Window can also be used.

17.1 Drawing Lines

Lines are drawn using the following method:

public void drawLine**(**int x_1, int y_1, int x_2, int y_2)
draws a line in the current color, from the point (x_1, y_1) to (x_2, y_2)

The following program 17.1 shows the **drawLine()** method:

Program 17.1

```
// This program illustrates the drawLine() method.

/*
<applet code = Line  width = 300 height =300 >
</applet>
*/

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

public class Line
    extends Applet
    {

  public void paint(Graphics gp)
      {
      int x1 = 10, y1 = 25;
      int x2 = 200, y2 = 230;
      gp.drawLine(x1, y1, x2, y2);
      gp.drawString("Line", 20, 270);
      }
    }
```

The above program gives the following output:

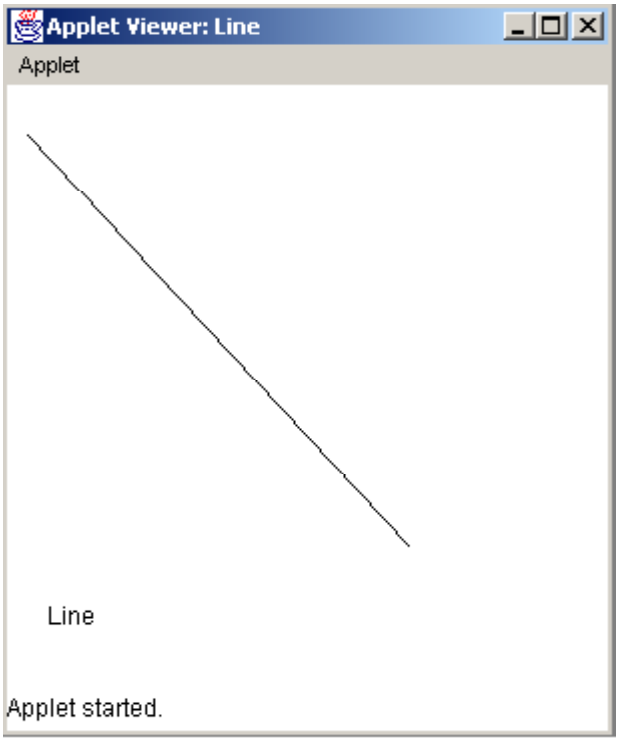


Fig.17.1 Output Screen for Program 17.1

17.2 Drawing Rectangles

The parameters specifying a rectangle is given in fig 17.2

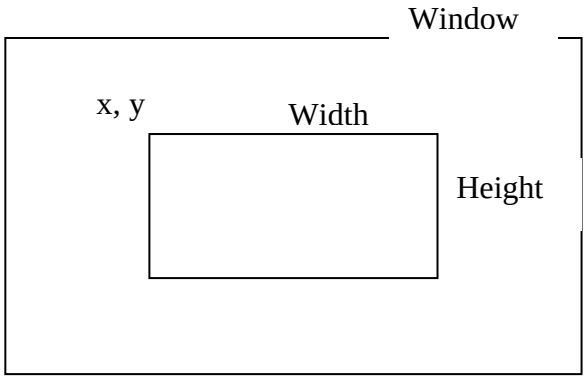


Fig.17.2 Parameters Specifying a Rectangle

Rectangles of different styles can be drawn using the following methods:

public void drawRect(int x, int y, int width, int height)

draws a rectangle with top left corner at (x, y) with the specified width and height in the current color.

public void drawRoundRect(int x, int y, int width, int height, int arcw, int arch)

Draws a round cornered rectangle with top left corner at (x, y) with the specified width and height in the current color; arcw is the horizontal diameter of the arc at the four corners and arch is the vertical diameter of the arc.

public void draw3DRect(int x, int y, int width, int height, boolean raised)
Draws a 3-d rectangle with top left corner at (x,y) with the specified width and height in the current color; if the raised is true, the rectangle appears to be raised above the surface, otherwise it is sunk into the surface.

public void fillRect(int x, int y, int width, int height)
Draws a filled rectangle

public void fillRoundRect(int x, int y, int width, int height, int arcw, int arch)
Draws a filled round cornered rectangle

public void fill3DRect(int x, int y, int width, int height, boolean raised)
Draws a filled 3D rectangle

public void clearRect(int x, int y, int width, int height)
Clears a rectangular area with top left corner at (x,y) with the specified width and height; the cleared area is filled with the background color.

The following program 17.2 shows the use of various rectangle drawing methods:

Program 17.2

```
// This program illustrates the drawRect() method.
//  Graphics objects are drawn on the Applet window.
/*
<applet code = Rectangle  width =300 height =350 >
</applet>
*/
import java.applet.Applet;
import java.awt.Graphics;
public class Rectangle
    extends Applet
    {
    public void paint(Graphics gp)
      {
      int x = 10, y = 25;
      int width = 100, height = 130;
      gp.drawRect(x, y, width, height);
      gp.drawString("Rectangle", 10, 170);
      gp.fillRect(160, 25, 100, 130);
      gp.drawString("Filled Rectangle", 160, 170);
      gp.drawRoundRect(10, 180, 100, 130, 20, 20);
      gp.drawString("Rounded Rectangle", 10, 330);
      gp.fill3DRect(160, 180, 100, 130, false);
      gp.drawString("Filled 3D Rectangle", 160, 330);
      showStatus("Applet with rectangles");
      }
   }
```

The above program gives the following output:

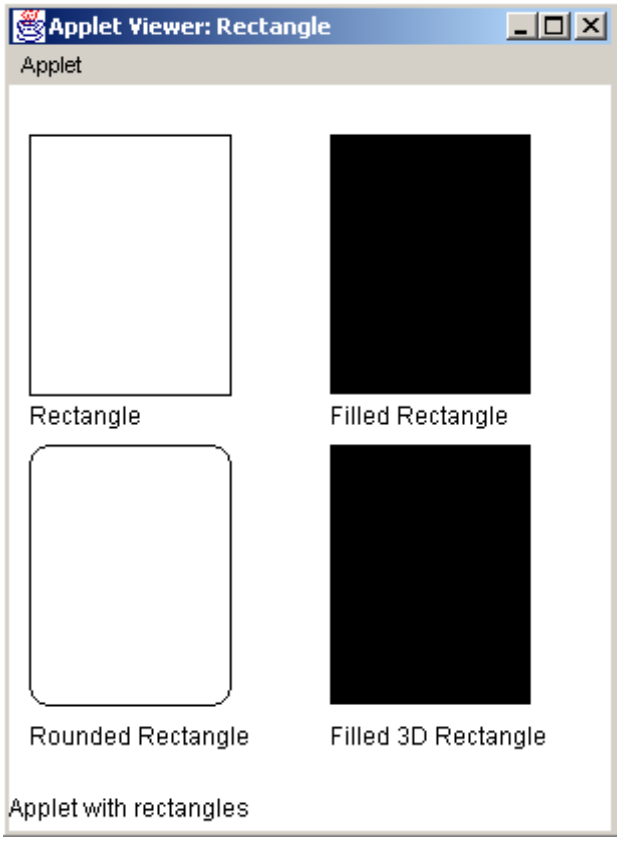


Fig.17.3 Output Screen for Program 17.2

These rectangles can also be drawn on a frame window. To create a frame window, the **Frame** class is used. A frame window is created by inheriting the **Frame** in a class and creating an instance of that class. The **paint()** method is used to obtain a **Graphics** context and call drawing methods. The following program 17.3 shows the use of a frame window to draw the graphics objects. This is an application program.

Program 17.3

```
// This program illustrates the drawRect() method.
// The graphics objects are drawn on the Frame window.

import java.awt.Graphics;
import java.awt.Frame;
class Myframe
    extends Frame
    {
 Myframe(String title)
        {
        super(title);
        setSize(300, 350);
        }
```

```
    public void paint(Graphics gp)
        {
        int x1 = 10, y1 = 25;
        int x2 = 100, y2 = 130;
        gp.drawRect(x1, y1, x2, y2);
        gp.drawString("Rectangle", 10, 170);
        gp.fillRect(160, 25, 100, 130);
        gp.drawString("Filled Rectangle", 160, 170);
        gp.drawRoundRect(10, 180, 100, 130, 20, 20);
        gp.drawString("Rounded Rectangle", 10, 330);
        gp.fill3DRect(160, 180, 100, 130, false);
        gp.drawString("Filled 3D Rectangle", 160, 330);
        }
  }

public class FRectangle
    {
      public static void main(String args [])
        {
        Myframe mframe = new Myframe("Frame with
                                          Rectangles");
        mframe.show();
        }
    }
```

The above program gives the following output:

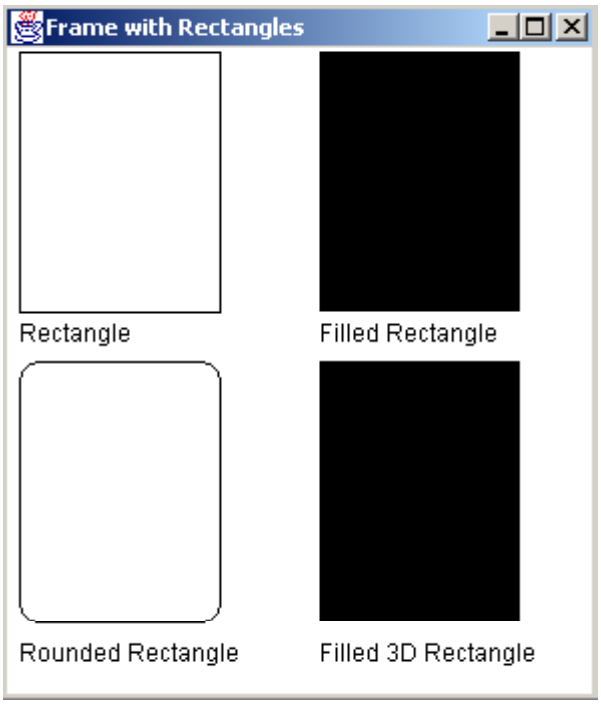


Fig.17.4 Output Screen for Program 17.3

Graphics methods are called on Graphics object. The graphics object is obtained by calling the **paint()** method with Graphics object as parameter.

17.3 Drawing Ovals and Circles

Ovals (ellipse) can be drawn using the following methods. Special case of oval becomes circle. Methods used for drawing ovals are:

public void drawOval(int x, int y, int width, int height)

Draws an oval bounded by a rectangle with its top left corner at (x, y) with the specified width and height

public void fillOval(int x, int y, int width, int height)

Draws a filled oval

The following program 17.4 illustrates the use of **drawOval** method to draw ellipses and circles:

Program 17.4

```
// This program illustrates the drawOval() method.
/*
<applet code = Oval  width =300 height =350 >
</applet> */
import java.applet.Applet;
import java.awt.Graphics;
public class Oval
    extends Applet
    {
    public void paint(Graphics gp)
        {
        int x = 10, y = 25;
        int width = 100, height = 130;
        gp.drawOval(x, y, width, height);
        gp.drawString("Oval", 10, 170);
        gp.fillOval(140, 25, 160, 100);
        gp.drawString("Filled Oval", 160, 170);
        gp.drawOval(10, 200, 100, 100);
        gp.drawString("Circle", 10, 330);
        gp.fillOval(160, 200, 100, 100);
        gp.drawString("Filled Circle", 160, 330);
        showStatus("Applet with ovals and circles");
        }
    }
```

The above program gives the following output:

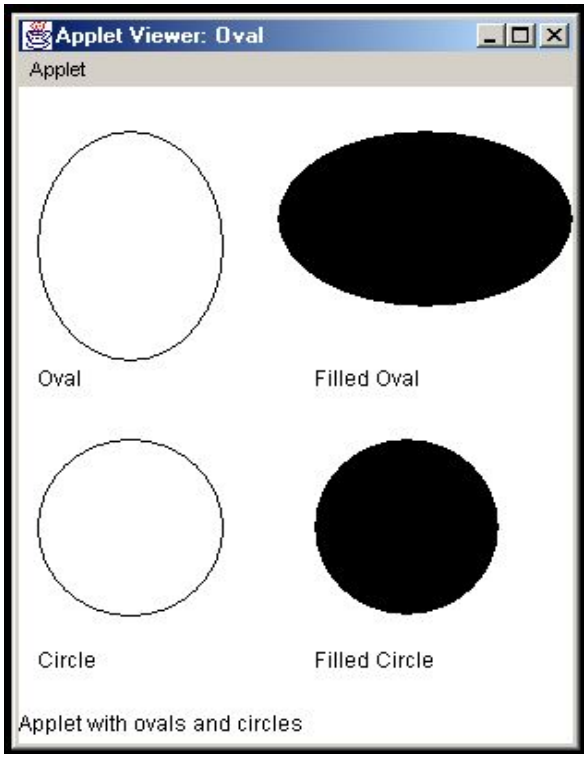


Fig.17.5 Output Screen for Program 17.4

17.4 Drawing Arcs

The co-ordinates to draw an arc are specified as in fig. 17.6.

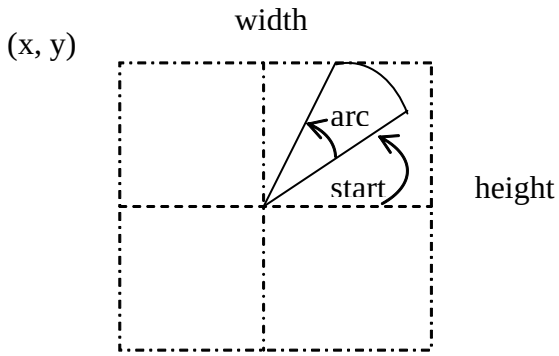


Fig.17.6 Parameters Specifying an Arc

Arcs are drawn using the following methods:

public void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)
Draws an arc bounded by a rectangle having its top left corner at (x, y) with specified width and height; the arc starts at startAngle and sweeps an angle arcAngle.

public void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)
Draws a filled arc

The following program 17.5 shows the use of **drawArc()** method.

Program 17.5

```
// This program illustrates the drawArc() method.
/*
<applet code = Arc  width =300 height =350 >
</applet>
*/
import java.applet.Applet;
import java.awt.Graphics;
public class Arc
    extends Applet
    {
    public void paint(Graphics gp)
        {
        int x = 10, y = 25;
        int width = 100, height = 130;
        gp.drawArc(x, y, width, height, 10, 110);
        gp.drawString("Arc", 10, 130);
        gp.fillArc(160, 25, 100, 130, 10, 110);
        gp.drawString("Filled Arc", 160, 130);
        gp.drawArc(10, 180, 100, 130, 20, 260);
        gp.drawString("Abtuse Arc", 10, 330);
        gp.fillArc(160, 180, 100, 130, 20, 260);
        gp.drawString("Filled Arc", 160, 330);
        showStatus("Applet with Arcs");
        }
    }
```

The above program gives the following output:

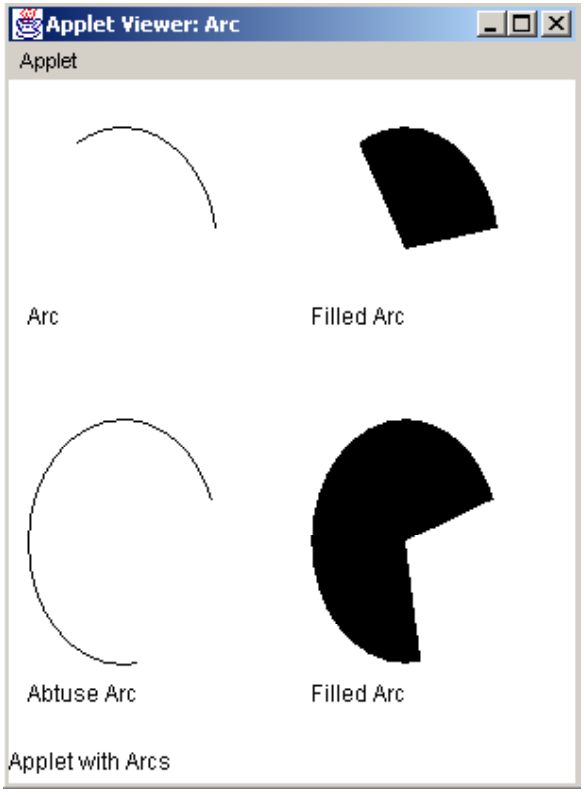


Fig.17.7 Output Screen for Program 17.5

17.5 Drawing Polygons

Polygons with many corners are drawn using the following methods. Each method takes two **int** type arrays, one for all the x co-ordinates and another for all the y co-ordinates, that forms the polygon. All polygons drawn using these methods are closed.

public void drawPolygon(int xPoints[], int yPoints[], int nPoints)

Draws a closed polygon defined by the arrays xPoints and yPoints; nPoints represent the number of (x,y) pairs. Usually, the starting and ending x and y points are to be the same. If the user does not give the ending point as the same as starting point, this method closes the line at the starting point.

public void fillPolygon(int xPoints[], int yPoints[], int nPoints)

Draws a filled closed polygon specified by the arrays xPoints and yPoints and nPoints

public void drawPolygon(Polygon p)

Draws the outline of a polygon defined by the Polygon object P

The following program 17.6 illustrates the use of two of the polygon drawing methods:

Program 17.6

```
// This program illustrates the drawPolygon() method.
/*
<applet code = Polygon  width =300 height =350 >
</applet>
*/
import java.applet.Applet;
import java.awt.Graphics;
public class Polygon
    extends Applet
    {
    public void paint(Graphics gp)
        {
        int x [] = {110,60,10,10,60,110,160,210,210,160,
                                110,110};
        int y [] = {60,10,60,160,210,160,210,160,60,10,
                                60,160};
        int fx [] ={110,90,90,110,130,150,150,130,110};
        int fy [] = {250,270,290,310,310,290,270,250,250};
        int n = 12;
        int fn = 9;
        gp.drawPolygon(x, y, n);
        gp.drawString("Polygon", 90, 210);
```

```
        gp.fillPolygon(fx, fy, fn);
        gp.drawString("Filled Polygon", 160, 300);
        showStatus("Applet with Polygons");
        }
}
```

The above program gives the following output:

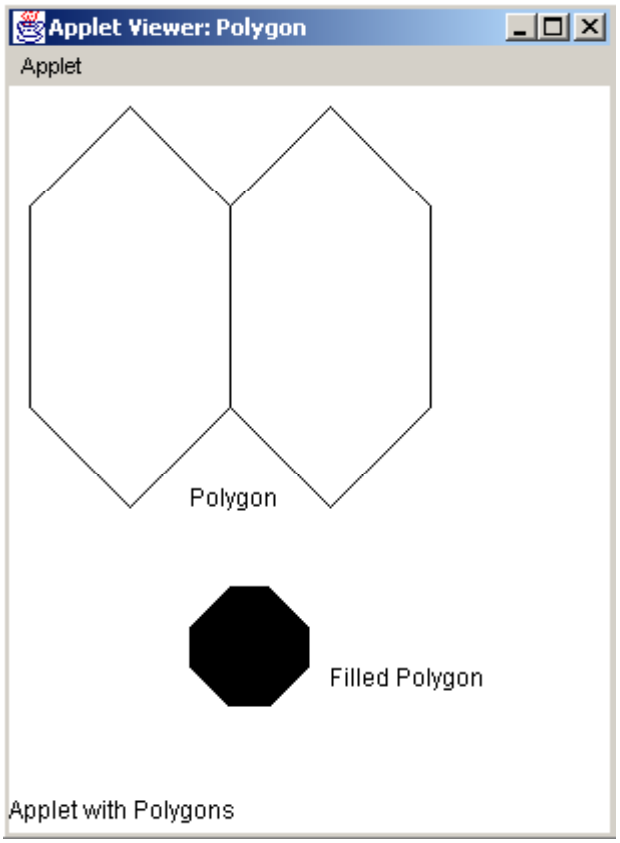


Fig.17.8 Output Screen for Program 17.6

17.6 Drawing Polyline

The **drawPolygon()** method always draw a closed polygon. There is no way to draw an open-ended polygon. This is achieved through a **drawPolyline()** method. This method draws a series of connected lines. The method used for drawing such a polyline is :

public void drawPolyline(int xPoints[], int yPoints[], int nPoints)

Draws a sequence of connected lines, specified by the arrays xPoints and yPoints; each pair (x,y) gives one point. The number of points for the polyline is nPoints.

The following program 17.7 shows the use of **drawPolyline()** method.

Program 17.7

```
// This program illustrates the drawPolyline() method.
/*
<applet code = Polyline  width =300 height =250 >
</applet>
*/
import java.applet.Applet;
import java.awt.Graphics;
public class Polyline
    extends Applet
    {
    public void paint(Graphics gp)
        {
        int x [] ={60,10,10,60,110,160,210,210,160};
        int y [] ={10,60,160,210,160,210,160,60,10};
        int n = 9;
        gp.drawPolyline(x, y, n);
        gp.drawString("Polyline", 90, 210);
        showStatus("Applet with Polylines");
        }
    }
```

The above program gives the following output:

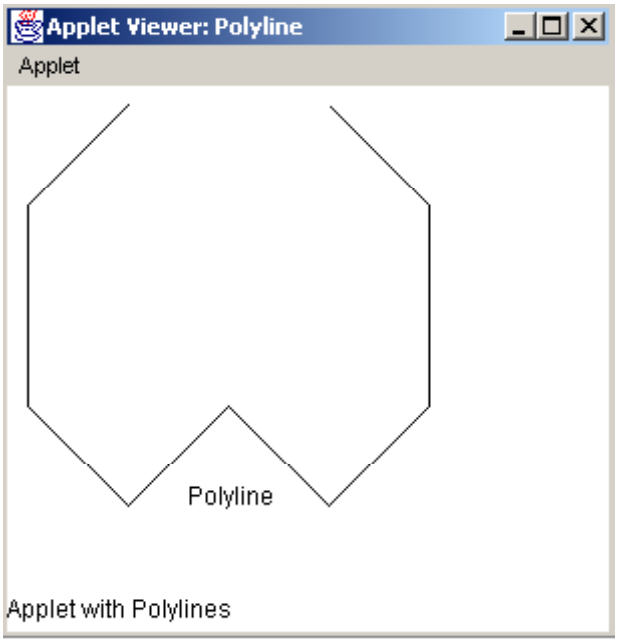


Fig.17.9 Output Screen for Program 17.7

The **drawPolygon()** method always draws a closed polygon, while the **drawPolyline()** method draws an open ended polygon.

17.7 Creating a Graphics Clip

Once a window is created, its origin is at the point (0,0). For certain applications, you may need to work in a clip inside the original window. This clip may need to have its own origin for drawing objects inside the clip. For this purpose, a **create()** method is available for creating a new Graphics clip. The syntax of this method is:

public Graphics create(int x, int y, int width, int height)

Creates a new Graphics clip window having its origin at (x,y) with the specified width and height; the origin for the new clip is (x,y) with reference to the original window. The origin for the clip window created is (0,0). Within this clip, all co-ordinates can be specified relative to this point. Transfer of origin can also be done through **translate()** method.

The following program 17.8, illustrates the use of **create()** method to create a new clip window:

Program 17.8

```
// This program illustrates the create() method.
/*
<applet code = Cre  width =200 height =220 >
</applet>
*/
import java.applet.Applet;
import java.awt.Graphics;
public class Cre
    extends Applet
    {
    Graphics gr;

    public void paint(Graphics gp)
        {
        int x = 10, y = 10;
        int width = 50, height = 80;
        int xnew = 100, ynew = 50, widthnew = 200,
                    heightnew = 200;
        gp.drawRect(x, y, width, height);
        gp.drawString("Rectangle", 10, 120);
        // creates a new clip
        gr = gp.create(xnew, ynew, widthnew, heightnew);
        gr.drawRect(x, y, width, height);
        gr.drawString("Rectangle", 10, 120);
        showStatus("Rectangle in new Graphics window");
        }
    }
```

The above program gives the following output:

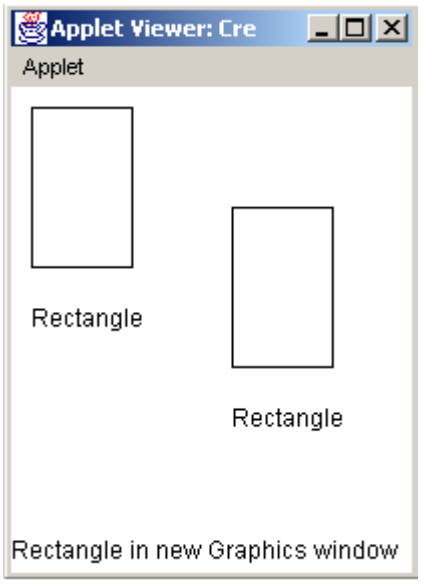


Fig.17.10 Output Screen for Program 17.8

17.8 Colors in Graphics

In the previous chapter, we have seen that colors are handled in **Color** class. In addition to color constants that set fixed colors, methods are available to set arbitrary colors as desired by users. The colors of the window can also be obtained and analyzed. Any color can be generated by suitable combination of the primary colors, red, green and blue. Hence, color generated by this way is called RGB color. Colors can also be represented by hue, saturation and brightness. Color produced by this method is called HSB color.

17.8.1 Constructors for Color Class

Colors can be set using the constructors of **Color** class. Some of the constructors are given below:

public Color(int r, int g, int b)

Creates an RGB color with the specified r(ed), g(reen) and b(lue) values; r, g and b can take value in the range 0 to 255. Alpha value is 255.

public Color(int r, int g, int b, int a)

Creates an RGB color with the specified r(ed) g(reen) b(lue) and a(lpha) values; r, g, b and a can take values in the range 0 to 255.

public Color(int rgb)

Creates an RGB color specified in the combined rgb value; this color consists of red component specified in bits 16 to 23, green component in bits 8 to 15 and blue component in bits 0 to 7. The default alpha value is 255.

public Color(float r, float g, float b)

Creates an RGB color with the specified r(ed), g(reen) and b(lue) values; r, g and b can take values in the range 0.0 to 1.0. The alpha value is 255.

public Color(float r, float g, float b, float a)

Creates an RGB color with the specified r(ed), g(reen), b(lue) and a(lpha) values; r, g, b and a can take values in the range 0.0 to 1.0.

The following program 17.9 shows the use of a few constructor methods of **Color** class:

Program 17.9

```
// This program illustrates the use of color constructor
// methods.
/*
<applet code = Colorcon1  width =200 height =100 >
</applet>
*/
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;
public class Colorcon1
    extends Applet
    {
    Color bgclr, fgclr;
    public void init()
        {
        bgclr = new Color(10, 150, 250);
        fgclr = new Color(0.95f, 0.1f, 0.1f);
        setBackground(bgclr);
        setForeground(fgclr);
        }
    public void paint(Graphics gp)
        {
        gp.drawString("Colorful constructors", 20, 40);
        showStatus("Color constructor");
        }
    }
```

The above program gives the following output:

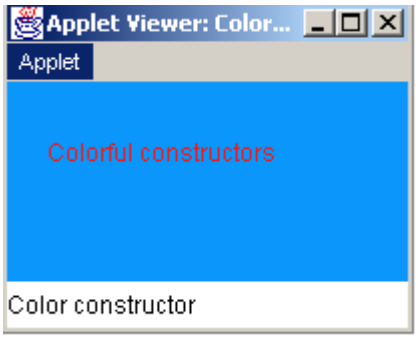


Fig.17.11 Output Screen for Program 17.9

17.8.2 Color Methods

Several methods are available in **Color** class to obtain, modify and to convert from one color model to another color model. Some of the methods are given in table 17.1.

Table 17.1 Some of the Methods Defined in Color Class

Method	Purpose of the Method
1. public int getRed()	Returns an int representing the red component of RGB color in the range 0 to 255
2. public int getGreen()	Returns an int representing the green component of RGB color in the range 0 to 255
3. public int getBlue()	Returns an int representing the blue component of RGB color in the range 0 to 255
4. public int getAlpha()	Returns an int representing the alpha component of RGB color in the range 0 to 255
5. public int getRGB()	Returns an int representing the combined RGB color
6. public Color brighter()	Returns a color which is a brighter version of this RGB color
7. public Color darker()	Returns a color which is a darker version of this RGB color
8. public String toString()	Returns a string representation of this RGB color

9.	public static int HSBtoRGB(int h,int s,int b)	Returns an int after converting the HSB color specified by h, s, b to its equivalent RGB color
10.	public static float[] RGBtoHSB(int r,int g,int b, float[] hsbvals)	Converts the RGB color specified by r, g, b to its equivalent components of HSB; returns a float array if the hsbvals argument is null, otherwise put the values into the array hsbvals
11.	public static Color getHSBColor(float h, float s, float b)	Returns an HSB color as specified by the h, s, b values; h, s and b takes value in the range 0.0 to 1.0.

The following program 17.10 illustrates some of the methods of **Color** class:

Program 17.10

```
// This program illustrates the use of color  methods.
/*
<applet code = Colormeth  width =350 height =200 >
</applet>
*/
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;
public class Colormeth
    extends Applet
   {
    String forecolor, backcolor;
    int rgbval;
    String rgb;
    Color bgclr, fgclr;
    int fgb, bgr;
    public void init()
        {
        bgclr = Color.getHSBColor(0.25f, 0.85f, 0.45f);
        fgclr = new Color(0.85f, 0.2f, 0.1f);
        setBackground(bgclr);
        setForeground(fgclr);
        }
    public void paint(Graphics gp)
        {
        gp.drawString("Color methods", 20, 40);
        forecolor = fgclr.toString();
        backcolor = bgclr.toString();
        rgbval = fgclr.getRGB();
```

```
        rgb = Integer.toBinaryString(rgbval);
        fgb = fgclr.getBlue();
        bgr = bgclr.getRed();
        gp.drawString("F.g. color : " + forecolor, 20, 60);
        gp.drawString("B.g. color : " + backcolor, 20, 80);
        gp.drawString("F.g. - RGB Value : " + rgb,20, 100);
        gp.drawString("F.g. - Blue Value : " + fgb, 20, 120);
        gp.drawString("B.g. - Red Value : " + bgr, 20, 140);
        showStatus(" in every 3 sec foreground color
                     changes to darker");
        try
            {
            Thread.currentThread().sleep(3000);
             fgclr = fgclr.darker();
             setForeground(fgclr);
             }
         catch (InterruptedException e)
             {
             ;
             }
         }
    }
```

The above program gives the following outputs:

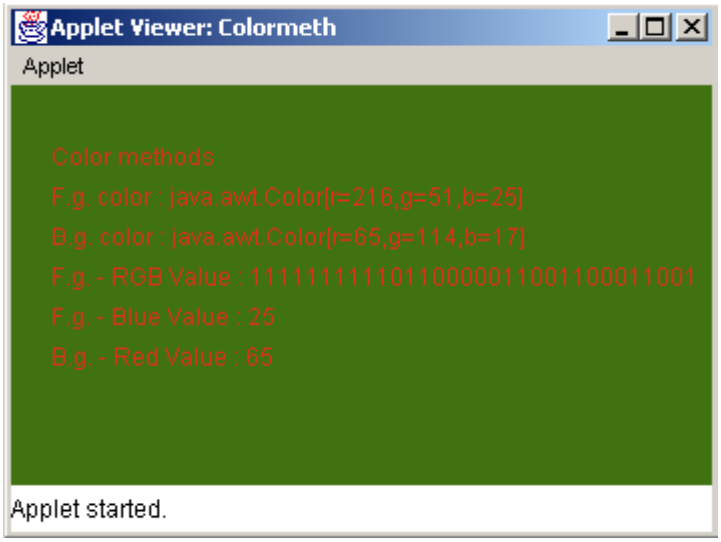


Fig.17.12(a) Output Screen for the Program 17.10 When Started

The output after a few seconds will give the following output:

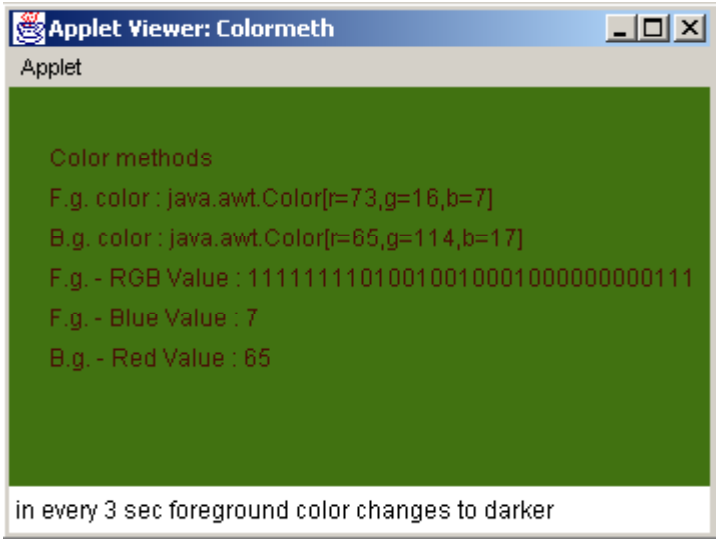


Fig.17.12 (b) Output Screen After A Few Seconds After the Start of Program 17.10

17.9 Setting Paint Modes

The paint mode decides the way the graphics objects are drawn on the window. Generally, when a graphics object is drawn, it overwrites the existing object at that location. If it is needed that all objects drawn at a place be visualized, then XOR mode of paint is to be selected. In this method, when drawing a second graphics object overlapping with an existing graphics object, the first object lying below the second can also be seen. In this method, when pixels of two objects are same at a point, the color, set in the XOR paint mode, is used at those points. This helps to contrast both objects. This is done by the following method in **Graphics** class:

public void setXORMode(Color clr)

The color clr is used when pixels of two graphics object at that location are the same. When two colors are different, an unpredictable color is used at that point. To return to the overwrite mode, the following method is used:

public void setPaintMode()

The following program 17.11 illustrates the use of the paint modes:

Program 17.11

```
// This program illustrates the setPaintMode() and
//  setXORMode() methods.
/*
<applet code = Paintmode  width =300 height =350 >
</applet>
*/
import java.applet.Applet;
import java.awt.Graphics;
```

```
import java.awt.Color;
public class Paintmode
    extends Applet
    {
    public void paint(Graphics gp)
        {
        int x = 10, y = 25;
        int width = 100, height = 130;
        gp.drawRect(x, y, width, height);
        gp.drawString("Rectangle", 10, 170);
        gp.fillRect(80, 45, 70, 80);
        gp.drawString("drawn in PaintMode", 160, 100);
        gp.drawString("Filled Rectangle", 160, 170);
        gp.drawRoundRect(10, 180, 100, 130, 20, 20);
        gp.drawString("Rounded Rectangle", 10, 330);
        gp.setXORMode(Color.white);
        gp.fillRect(60, 140, 100, 130);
        gp.setPaintMode();
        gp.drawString("drawn in XORMode", 150, 280);
        }
    }
```

The above program gives the following output:

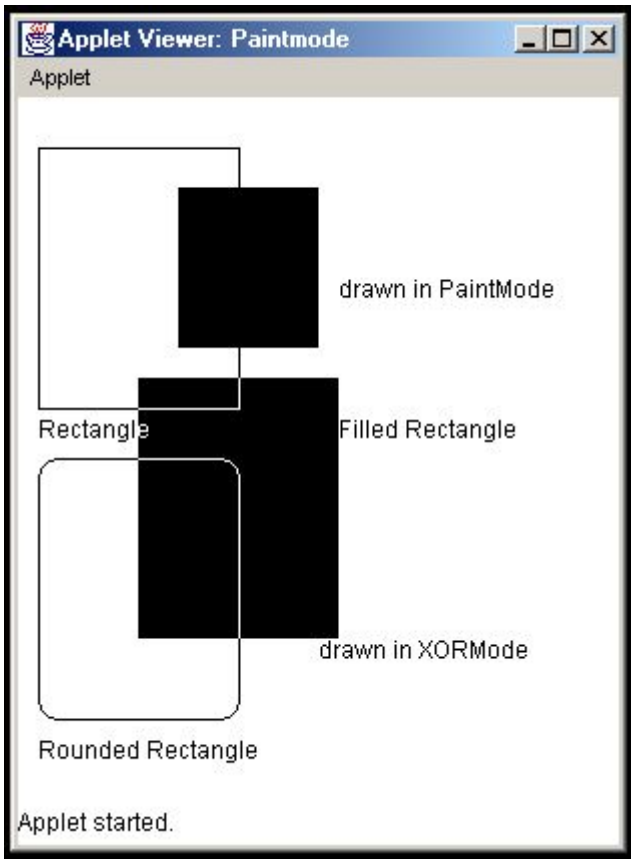


Fig.17.13 Output Screen for Program 17.11

To visualize overlapping graphics, use XORMode.

17.10 Fonts in Graphics

The fonts of text for display can be set according to the need of the user. Fonts loaded in the local computer system can be used for display. Java provides a variety of tools to set new fonts, manipulate their structure, size, style, etc. But only limited features that are needed for ordinary use are discussed in this section. We will see how to set new fonts family, fonts, size and style. To set new fonts, methods in **Graphics** class are used. The methods to study the fonts currently used are available in **Font** class.

17.10.1 Determining Fonts Available in the System

The fonts family available in the local computer system can be obtained using the following methods defined in **GraphicsEnvironment** class of java.awt package. The constructors in this class are :

abstract String[] getAvailableFontFamilyNames()
Returns a String array containing the names of all font families available in the current Graphics Environment

static GraphicsEnvironment getLocalGraphicsEnvironment()
Returns the local GraphicsEnvironment

To make use of the first method given above, one needs a **GraphicsEnvironment** object. To create a **GraphicsEnvironment** object, the second method defined above is used. In the following program 17.12, methods to obtain the font family names are illustrated:

Program 17.12

```
// This program finds the font family available in this
// system.
/*
<applet code = Fontfamily  width =275 height =225 >
</applet>
*/
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.GraphicsEnvironment;
public class Fontfamily
    extends Applet
    {
```

```
String fonts[];
public void paint(Graphics gp)
  {
  GraphicsEnvironment ge =
      GraphicsEnvironment.getLocalGraphicsEnvironment();
  fonts = ge.getAvailableFontFamilyNames();
  showStatus("Fonts Family in this System");
  int size = fonts.length;
  gp.drawRect(10, 5, 240, 20);
  gp.drawRect(10, 30, 180, 15);
  gp.drawString("The no of fonts family in this
            system : " + (size + 1), 20, 20);
  gp.drawString("The first ten font families
            are ", 20, 40);
  gp.drawRect(10, 45, 170, 160);
  for (int i = 0; i < 10; i++)
        gp.drawString(fonts[i], 20, (60 + i * 15));
    }
}
```

The above program gives the following output:

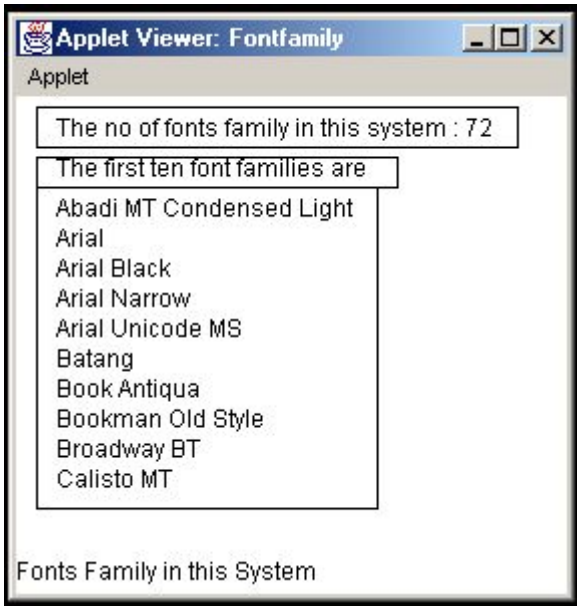


Fig.17.14 Output Screen for Program 17.12

17.10.2 Setting Fonts

The **Font** class has methods to create fonts for drawing text on the screen. The constructor to create new font in **Font** class is:

Font (String name, int style, int size)

where name - is the name of the font family,

style takes either the int value or the constants given below,

Font.PLAIN

Font.BOLD
Font.ITALIC
Font.ITALIC + Font.BOLD

size - is the font size.

The logical family names defined in the JDK are:

SansSerif
Serif
MonoSpaced
Dialog
DialogInput

These font names are mapped to the fonts actually available in the local machine.

Some of the methods defined in **Font** class are given in table 17.2.

Table 17.2 Some of the Methods Defined in Font Class

	Method	Purpose of the Method
1.	public String getFamily()	Returns the family name of this font
2.	public String getName()	Returns the logical name of this font
3.	public String getFontName()	Returns the font face name of this font Example : Serief.BOLD
4.	public int getStyle()	Returns the style of the font PLAIN, BOLD, ITALIC, BOLD+ITALIC
5.	public int getSize()	Returns an int representing the size of the font
6.	public boolean isPlain()	Returns true, if this font has a PLAIN style, otherwise false
7.	public boolean isBold()	Returns true if this font has a BOLD style, otherwise false
8.	public boolean isItalic()	Returns true if this font has a ITALIC style, otherwise false

The following method is defined in **Graphics** class:

public void setFont(Font font)

Sets this graphics context's font to the specified font

The following program 17.13 illustrates some of the methods defined in **Font** class:

Program 17.13

```
// This program illustrates the use of getFont() method
// in Graphics class.
// setFont() method is from Graphics class
// somasundaramk@yahoo.com
/*
<applet code = Fontmeth  width =350 height =350 >
</applet>
*/
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Font;
public class Fontmeth
    extends Applet
    {
    public void paint(Graphics gp)
       {
        Font fnt = new Font("SansSerif", Font.BOLD, 20);
        gp.setFont(fnt);
        String fname = fnt.getName();
        String fontname = fnt.getFontName();
        int fsize = fnt.getSize();
        int style = fnt.getStyle();
        gp.drawString("Font family used : " + fname, 20, 20);
        gp.drawString("Font used : " + fontname, 20, 40);
        gp.drawString("Font  size used  : " + fsize, 20, 60);
        gp.drawString("Font style : " + style, 20, 80);
        fnt = new Font("Serif", Font.ITALIC, 15);
        gp.setFont(fnt);
        fname = fnt.getName();
        fontname = fnt.getFontName();
        fsize = fnt.getSize();
        style = fnt.getStyle();
        gp.drawString("Font family used  : " + fname,
                                              20, 100);
        gp.drawString("Font used  : " + fontname,
                                              20, 120);
        gp.drawString("Font  size used  : " + fsize,
                                              20, 140);
        gp.drawString("Font style : " + style, 20, 160);
        fnt = new Font("Dialog", Font.ITALIC +
                                          Font.BOLD, 25);
        gp.setFont(fnt);
        fname = fnt.getName();
        fontname = fnt.getFontName();
        fsize = fnt.getSize();
        style = fnt.getStyle();
        gp.drawString("Font family used  : " + fname,
                                              20, 200);
```

```
        gp.drawString("Font used  : " + fontname, 20, 230);
        gp.drawString("Font  size used  : " + fsize,
                                                20, 260);
        gp.drawString("Font style : " + style, 20, 290);
      }
   }
```

The above program gives the following output:

Fig.17.15 Output Screen for Program 17.13

After reading this chapter, you should have learned the following:

- Graphics methods are defined in Graphics class in AWT package.
- Graphics objects can be drawn only on a window.
- There are several methods to draw regular shapes.
- Background and foreground colors can be set in Graphics method.
- Font size and type can be handled in Graphics class.

In the next chapter, you will learn about event handling.

Worked Out Problems–17

Problem 17.1w

Write a Java program to draw lines on a frame by pressing the mouse and dragging to another point like a pen tool in a paint tool.

Program 17.1w

```
/* ------------------------------------------------------------
 This program acts like a pen tool in a  paint software.

 somasundaramk@yahoo.com
------------------------------------------------------------- */

import java.util.*;
import java.awt.*;
import java.awt.event.*;
class Drawframe
    extends Frame
    {
    int mx1 = 0, my1 = 0, mx2 = 20, my2 = 50;
    Vector px = new Vector();
    Vector py = new Vector();
    int pointn = 0;
    int pcount = 0;
    Integer intg;
    public Drawframe()
       {
        px.insertElementAt(new Integer(mx1), pointn);

        py.insertElementAt(new Integer(my1), pointn);
       ++pointn;
        px.insertElementAt(new Integer(mx2), pointn);
        py.insertElementAt(new Integer(my2), pointn);
        addMouseListener(new Madapter(this));
        addWindowListener(new Wadapter(this));
       }
    public void paint(Graphics gp)
            {
            gp.drawString("Pen tool demo", 20, 40);
            pcount = px.size();
            for (int i = 0; i < pcount - 1; i++)
            {
            int tx1 = ((Integer)px.elementAt(i)).intValue();
            int tx2 = ((Integer)px.elementAt(i + 1)).intValue();
            int ty1 = ((Integer)py.elementAt(i)).intValue();
            int ty2 = ((Integer)py.elementAt(i + 1)).intValue();
            gp.drawLine(tx1, ty1, tx2, ty2);
            }
```

```
        }
    }
class Wadapter
    extends WindowAdapter
    {
    Drawframe df;
    Wadapter(Drawframe dfram)
        {
        df = dfram;
        }
    public void windowClosing(WindowEvent we)
        {
        System.exit(0);
        }
    }
class Madapter
    extends MouseAdapter
    {
     Drawframe df;
     Madapter(Drawframe dfram)
        {
        df = dfram;
        }
    public void mousePressed(MouseEvent me)
        {
        int x1 = me.getX();
        int y1 = me.getY();
        df.pointn++;
        (df.px).insertElementAt(new Integer(x1), df.pointn);
         (df.py).insertElementAt(new Integer(y1), df.pointn);
        }
    public void mouseDragged(MouseEvent me)
        {
        int x2 = me.getX();
        int y2 = me.getY();
        df.pointn++;
       (df.px).insertElementAt(new Integer(x2), df.pointn);
       (df.py).insertElementAt(new Integer(y2), df.pointn);
        df.repaint();
        }
    public void mouseReleased(MouseEvent me)
        {
        int x2 = me.getX();
        int y2 = me.getY();
        df.pointn++;
       (df.px).insertElementAt(new Integer(x2), df.pointn);
       (df.py).insertElementAt(new Integer(y2), df.pointn);

        df.repaint();
        }
```

```
    }
class Prob171
    {
    public static void main(String args [])
            {
            Drawframe frm = new Drawframe();
            frm.setSize(300, 250);
            frm.setTitle("Pen Tool Demonstration");
            frm.setVisible(true);
            }
    }
```

The above program gives the following output:

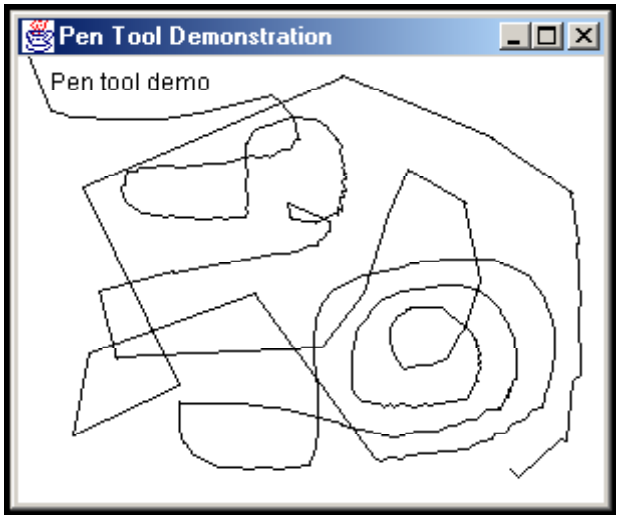


Fig.17.16 Output Screen for Program 17.1w

Problem 17.2w

Write a Java program to produce a rotating color disc:

Program 17.2w

```
/* ------------------------------------------------------------
 This program makes a rotating color disc.

 somasundaramk@yahoo.com

<applet code = Prob172  width =350 height =200 >
</applet>
------------------------------------------------------------*/
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;
```

```
public class Prob172
    extends Applet
    {
     String forecolor, backcolor;
     int rgbval;
     String rgb;
     Color bgclr, fgclr;
     int fgb, bgr;
     Color color [] = {Color.cyan,Color.red,Color.blue,
                          Color.white,Color.orange,Color.pink
        };
    public void init()
        {
        setBackground(Color.green);
        }
    public void paint(Graphics gp)
          {
          int colorn = color.length;
          gp.drawString(" Rotating Color Disc", 100, 20);
          int index;
          while (true)
            {
            for (int i = 0; i < colorn; i++)
               {
               index = Math.abs(colorn - i);
               if (index > 5)
                    index = 0;
               gp.setColor(color[index]);
               gp.fillArc(20, 20, 150, 150, 0, 60);
               ++index;
               if (index > 5)
                    index = 0;
               gp.setColor(color[index]);
               gp.fillArc(20, 20, 150, 150, 60, 60);
               ++index;
               if (index > 5)
                    index = 0;
               gp.setColor(color[index]);
               gp.fillArc(20, 20, 150, 150, 120, 60);
               ++index;
               if (index > 5)
                    index = 0;
               gp.setColor(color[index]);
               gp.fillArc(20, 20, 150, 150, 180, 60);
               ++index;
               if (index > 5)
                    index = 0;
               gp.setColor(color[index]);
               gp.fillArc(20, 20, 150, 150, 240, 60);
               ++index;
```

```
                if (index > 5)
                     index = 0;
                gp.setColor(color[index]);
                gp.fillArc(20, 20, 150, 150, 300, 60);
                try
                     {
                     Thread.sleep(200);
                     }
                 catch (InterruptedException ie)
                     {
                     ;
                     }
                 }
            }
        }
    }
```

The above program gives the following output:

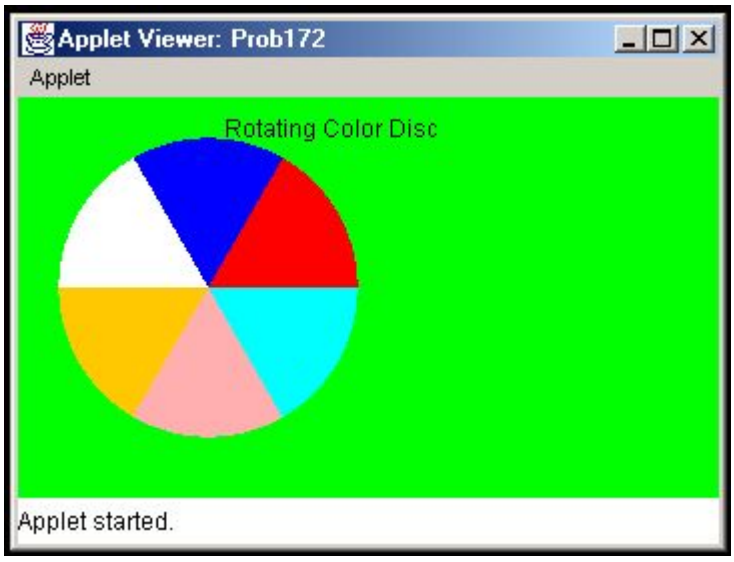


Fig.17.17 Output Screen for Program 17.2w

Exercise–17

I. Fill in the blanks

17.1. Graphics object can only be drawn on _____ .

17.2. A window can be created using _____ class or _____ class.

17.3. A _____ window is closable while _____ window is not closable.

17.4. The drawPolygon method always draws a _____ polygon.

17.5. To draw an open-ended polygon the _____ method can be used.

17.6. To visualize two graphics drawn at the same location _____ is to be used.

17.7. The terms BOLD, PLAIN, ITALIC specifies the _____ of a font.

II. Write Java program for the following problems:

17.8. Write a program to draw a filled rectangle and expand continuously.

17.9. Write a program to draw a one-handed clock.

17.10. Write a green rectangle and print your address in red color on it.

17.11. Create a font of your choice and print your name in three font styles, each one having a different font size.

17.12. Write a program to simulate a bouncing rubber ball on a hard floor. Show at least 5 bounces.

17.13. Write a program to draw a word in font size 6, expanding to 96 font size and collapsing to initial size repeatedly.

17.14. Draw a polygon with 8 corners.

* * * * * *