long lastModified	Returns time stamp of invoking file in millisec elapsed since Jan 1, 1970.
boolean setLastModified (long sec)	Sets the time stamp of invoking object to that specified by "sec".
boolean setReadOnly( )	Sets the invoking file to read only.
void deleteOnExit( )	Removes the file associated with the invoking object when the Java Virtual Machine terminates.

11.3.1 File Name Filter Interface

To limit the number of files returned by the list ( ) method we need to use the second form of list ( ).

```
String [ ] list (FilenameFilter fobj)
```

In this form, FileNameFilter defines only a single method accept().

```
boolean accept (File dir, String filename)
```

A sample code:

```
import java.io. *
public class try implements FilenameFilter
{
  String s;
  public try (String s)
  {
      this.s = "." + s;
  }
  public boolean accept (File dir, String name)
  {
      return name. endsWith (s);
  }
}
class dirfilter
{
  public static void main (String args[ ])
  {
      String dname = "/java/bin";
      FilenameFilter only = new try ("class");
      String s[ ] = f1.list (only);
      for (int i = 0; i <s.length; i++)
      System.out.println (s[i]);
  }
  }
```

11.3.2 FileInputStream and FileOutputStream

FileInputStream: This class creates an InputStream that can be used to read bytes from a file. The constructor used, takes a file name of File object and opens the file for reading.

- FileInputStream(String filepath)
- FileInputStream(File fileobj)

It throws FileNotFoundException.

FileOutputStreams: This class creates an OutputStream that can be used to write bytes to a file. The constructors open an existing file or create a new file for writing data.

- FileOutputStream(String Filepath)
- FileOutputSteam(File Fileobj)
- FileOutputStream(String filepath, boolean append)

FileInputStream overrides six of the methods in abstract class InputStream. The mark( ) and reset( ) are not overriden.

FileOutputStream will create the file before opening it for output when you create the object. In case you attempt to open a read only file, an IOException will be thrown.

Note that the class DataInputStream extends FileInputStream implements the interface DataInput. Therefore, the DataInputStream class implements the methods and described in DataInput in addition to using the methods of InputStream class. The DataInput interface contains following methods to read different data types:

- `readShort( ), readDouble( ), readInt( ), readLong( ),`
- `readLine( ), readLong( ), readChar( ), readFloat( ),`
- `readBoolean( ), readUTF( )`

Similarly, DataOutputStream implements the interface DataOutput and therefore, implements the following methods contained in DataOutput interface:

```
WriteShort( ), WriteChar( ),
WriteFloat( ), WriteLong( ),
WriteDouble( ), WriteInt( ), WriteBoolean( ),
WriteUTF( ).
```

Example: FileInputStream and FileOutputStream

```
import java.io.*;
class fileio
{
  public static void main (String args[ ]) throws Exception
  {
  int size;
  InputStream f = new FileInputStream ("a.dat");
  OutputStream f0 = new FileOutputStream ("b.dat");
  OutputStream f1 = new FileOutputStream ("b1.dat");
  OutputStream f2 = new FileOutputStream ("b2.dat");
  size = f.available( );
  int n = size/4;
  for (int i = 0; i < n; i++)
```

```
                {
                  System.out.print ((char)f.read( ));
          }
                byte b[ ] = new byte [n];
                f.read(b);
                System.out.println (new String (b, 0, n));
                f.skip (n);
                f.read (b, n/2, n/2);
            System.out.println (new String (b, 0, b.length) );
```

11.3.3 Copying Files

```
public class Copy {
  public static void main(String[] args) throws IOException {
    File inputFile = new File("farrago.txt");
    File outputFile = new File("outagain.txt");

    FileReader in = new FileReader(inputFile);
    FileWriter out = new FileWriter(outputFile);
    int c;
    while ((c = in.read()) != -1)
      out.write(c);

    in.close();
    out.close();
  }
}
```

Check Your Progress

Fill in the blanks:

1. To obtain a .... based stream that is attached to the console, you wrap System.in in a Buffered Reader object to create a character stream.
2. .... method returns true if the file has an absolute path and false if path is relative.
3. Java provides a number of .... and .... that allow you to read and write files.

11.4 LET US SUM UP

Java program performs Input/Output through streams. A stream is an abstraction that either produces or consumes information. It is linked to a physical device by the Java Input/Output systems. Java implements streams within class hierarchies defined in java.io package. In Java, the only way to perform console input is to use a byte stream and an older code that uses this approach persists. The preferred method of reading console input for Java2 is to use a character oriented stream. Console output is most easily accomplished using print( ) and println( ). These methods are defined by the class PrintStream. Even though System.out is a byte stream, using it for simple program output is still acceptable. Java provides a number of classes and methods that allow you to read and write files. In

Java all files are byte oriented and Java provides methods to read and write bytes from and to a file. However, Java allows you to wrap a byte oriented file stream within a character based object.

11.5 KEYWORDS

Input Reader: It is the stream that is linked to the instance of BufferedReader that is being created.

Reader: It is an abstract class, one of its concrete subclasses is InputStream-Reader, which converts bytes to characters.

FileInputStream: This class creates an InputStream that can be used to read bytes from a file.

FileOutputStreams: This class creates an OutputStream that can be used to write bytes to a file.

11.6 QUESTIONS FOR DISCUSSION

1. Explain reading character with example.
2. Differentiate between reading and writing files.
3. What are uses of Fileinputstream and Fileoutputstream?
4. What is writing console output?

Check Your Progress: Model Answers

1. Character
2. Boolean isAbsolute( )
3. Classes and methods

11.7 SUGGESTED READINGS

E. Balaguruswami, *Programming with Java*, Tata McGraw-Hill.

Davis, Stephen R., *Learn Java Now*, Microsoft Press.

Naughton, Patrick, *The Java Hand Book*, OSborne McGraw- Hill.

Sams.net, *Java unleased*.

Herbert Schildt, *The Complete Reference Java 2*, Tata McGraw-Hill.

LESSON 12

APPLETS

CONTENTS

12.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Define applets and various applets like Chkr, cs, de etc.
- Describe graphics and text files

12.1 INTRODUCTION

Applets are small applications that are accessed from an Internet server, transported over the Internet, automatically installed and run as part of a web document. After an applet arrives on the client it has limited access to resources, so that it can produce an arbitrary multimedia user interface and run complex computations without introducing the role of viruses or breaching data integrity.

12.2 APPLETS

The applet begins with two import statements, the first imports the Abstract Windows Toolkit (AWT) classes. Applets interact with the user through the AWT, not through the console based Input/Output classes. The second import statement imports the applet package which contains the class Applet. Every applet you create must be a subclass of Applet. The next line in the program declares the class simpleApplet. This class must be declared as public because it will be acessed by code that is outside the program.

Example:

```
import  java.awt.*;
import java.applet. *;
public class simpleApplet extends Applet
{
  public void paint (Graphics g)
  {
      g.drawString ("A simple Applet", 20, 20);
  }
}
```

Inside simpleApplet, paint( ) is declared. This method is defined by the AWT and must be overriden by the applet. paint( ) is called each time the applet must redisplay its output. The situation can occur for several reasons. For example, the window in which the applet is running can be overwritten by another window and then uncovered. paint( ) method has one parameter of type Graphics. This parameter contains the graphics context which describes the graphics environment in which the applet is running.

Inside paint( ) is a call to drawString( ), which is a member of the Graphics class. This method outputs a string beginning at the specified X, Y location.

```
    void drawString (String msg, intX, intY)
```

X and Y is the location on the screen where the drawString will display the message.

To view the output, a Java compatible web browser is needed. A small HTML file is needed where inside the applet tag the Java's class file is passed.

```
        <applet code = "simpleApplet" width = 200 height = 60> </applet>
```

Then we can also use the command C:\> appletviewer app.html

12.2.1 Life Cycle of Applet

You are already aware that applets are small applications that are accessed on an Internet server, transported over the Internet, automatically installed, and run as part of a web document. After an applet arrives on the client, it has limited access to resources, so that it can produce an arbitrary multimedia user interface and run complex computations without introducing the risk of viruses or breaching data integrity.

After you enter the source code for simple Applet, compile in the same way that you have been compiling programs. However, running simple applets involves a different process. In fact, these are two ways in which you can run an applet:

(a) Executing the applet within a Java-compatible Web browser, such as Netscape Navigator.

(b) Using an applet viewer, such as the standard JDK tool, appletviewer. The appletviewer executes your applet in a window. This is generally the fastest and the easiest way to test your applet.

```
          Import java.awt.*;
          /*
          <applet code = "Simple Applet" width = 200 height = 60>
          </applet>
          */
          public class Simple Applet extends Applet {
   public void pait (Graphics 9) {
               9.drawstring ("A simple Applet", 20, 20)
            }
          }
```

12.2.2 Creating Applets

All applets are subclasses of **Applet**. Thus all applets must import **java.applet.** Applets must also import java.awt. (AWT stands for 'Abstract Window Toolkit') Since all applets run in a window, it is necessary to include support for that window. Applets are not executed by the console-based Java run-time interpreter. Rather, they are executed by either a web browser or an applet viewer.

Applets are event driven. An applet resembles a set of interrupt service routines. It waits until an event occurs. The AWT notifies the applet about an event by calling an event handler that has been provided by the applet. Once this happens, the applet must take appropriate action and then quickly return control to the AWT. This is a crucial part.

For the most part, your applet should not enter a "mode" of operation in which it maintains control for an extended period. Instead, it must perform specific actions in response to events and then return control to the AWT run-time system. For an applet to perform a repetitive task on its own, you must start an additional thread of execution.

12.2.3 Adding Applets to HTML Files

To execute an applet in a web browser, you need to write a short HTML text file that contains the appropriate APPLET tag. Here is the HTML file that executes SimpleApplet:

```
<applet code = "SimpleApplet" width = 200 height = 60>
</applet>
```

The width and height attributes specify the dimensions of the display area used by the applet. After you create this file, you can execute your browser and then load this file, while causes SimpleApplet to be executed.

To execute SimpleApplet with an applet viewer, you may also execute the HTML file called RunApp.html (say) on the command line.

```
C:\>appletviewer RunApp.html
```

However, a more convenient method exists that you can use to speed up testing. Simply include a comment at the head of your Java source code file that contains the Applet tag. By doing so, your code is documented with a prototype of the necessary HTML statements, and you can test your compiled applet merely by starting the applet viewer with your Java source code file.

12.2.4 Running an Applet

To execute an applet with an apple viewer, you may also execute the HTML file in which it is enclosed, e.g..

```
C:\>appletviewer RunApp.html
```

Execute the applet the applet viewer, specifying the name of your applet's source file. The applet viewer will encounter the applet tag within the comment and execute your applet.

12.2.5 Applet Initialization and Termination

It is important understand the order in which the various methods shows in the skeleton are called. When an applet begin. The AWT calls the following methods, in this sequence:

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

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

1. stop()
2. destroy ()

Let's look more closely at these methods.

init( )

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

start( )

The start( ) method is called after init( ). It is also called to restart an applet after it has been stopped. Whereas init( ) is called once—the first time an applet is loaded—start( ) is called each time an applet's HTML document is displayed onscreen. So, if a user leaves a web page and comes back, the applet resumes execution at start( ).

paint( )

The paint( ) method is called each time your applet's output must be redrawn. This situation can occur for several reasons. For example, the window in which the applet is running may be overwritten by another window and then uncovered. Or the applet window may be minimized and then restored paint( ) is also called when the applet begins execution. Whatever the cause, whenever the applet must redraw its output, paint( ) is called. The paint( ) method has one parameter of type Graphics. This parameter will contain the graphic context, which describes the graphics environment in which the applet is running. This context is used whenever output to the applet is required.

stop( )

The stop( ) method is called when a web browser leaves the HTML document containing the applet—when it goes to another page, for example. When stop( ) is called, the applet is probably running. You should use stop( ) to suspend threads that don't need to run when the applet is not visible. You can restart them when start( ) is called if the user returns to the page.

destroy( )

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

12.2.6 Passing Parameters to Applets

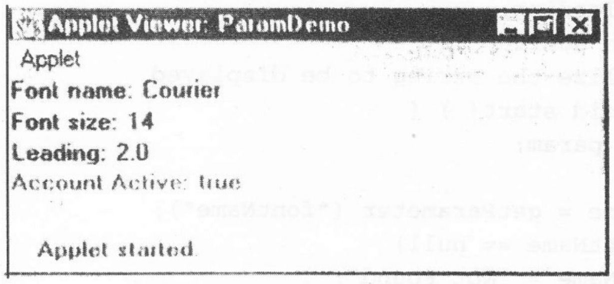
The Applet tag is HTML allows you to pass parameters to your applet. To retrieve a parameter, use the **getparameter( )** method. It returns the value of the specified parameter in the form of a String object. Thus, for numeric and Boolean values, you will need to convert their string representations into Internet formats. Here is an example the demonstrates passing parameters:

```
//Use Parameters
import java.awt.*;
import java.applet.*;
/*
<applet code = "paramDemo" width = 300 height = 80>
<param name = fontName value = Courier>
<param name = fontSize value = 14>
<param name = leading value = 2>
<param name = account Enabled value = true>
</applet>
*/
public class ParamDemo extends Applet {
  string fontName;
  int fontSize;
```

```
    float leading;
    Boolean active;
  //Initialize the string to be displayed.
  Public void start( ) {
    String param;

    fontName = getParameter ("fontName");
    if (fontName == null)
      fontName = "Not Found";
    param = getParameter ("fontSize");
    try {
      if (param ! = null) // if not found
fontSize = Integer.parameter (param);
      else
        fontSize = 0;
    } catch (NumberFormatException e) {
      fontSize = - 1;
    }
    param = getParameter ("leading");
    try {
    if (param ! = null) //if not found
      leading = Float.value01(param).float Value( );
    else
      leading = - 1;
    }
  param = getparameter ("account Enabled");
    if (param != Null)
    active = Boolean.value of (param).booleanvalue();
  }
  // Display parameters.
  public void paint (Graphics g) {
      g.drawstring ("Font name: " + fontName, 0, 10);
      g.drawstring ("Font size: " + fontsize, 0, 26);
      g.drawstring ("Leading: " + leading, 0, 42);
      g.drawstring ("Account Active: " + active, 0, 58);
    }
  }
```

Sample output from this program is shown in Figure 12.1.



```
Font name: Courier
Font size: 14
Leading: 2.0
Account Active: true
```

Figure 12.1

As the program shows, you should be test the return values from getParameter( ). If a parameter isn't available, getParameter( ) will return null. Also, conversions to numeric types must be attempted in a try statement that catches NumberFormatException. Uncaught exceptions should never occur within an applet.

12.2.7 Drawing Image on the Applet

One application of Java is to use active images and animation to provide a graphical means of navigating the web that is more interesting than the underlined blue word used by hypertext. To allow you applet to transfer control to another URL, you must use the showDocument( ) method defined by the AppletContext interface.

AppletContext is interface that lets you get information from the applet's execution environment. The methods defined by AppletContext shown in Table 12.1. The context of the currently executing applet is obtained by a call to the getAppletContext( ) method defined by Applet.

Table 12.1: The Abstract Methods Defined by the AppletContext Interface

Method	Description
Applet getApplet(String *appletName*)	Returns the aplet specified by *appletName* if it is within the current context. Otherwise, null is returned.
Enumeration getApplets( )	Returns an enumeration that contains all of the applets within the current applet context.
AudioClip get AudioClip(URL *url*)	Returns an AudioClip object that encapsulates the audio clip found at the loacation specified by *url*.
Image getImage(URL *url*)	Returns an Image object that encapsulated the image found at the location specified by *url*.
Void show Document (URL *url*)	Brings the document at the URL specified by *url* into view. This method may not be supported by applet viewers.
void showDocument(URL *url*, String where)	Brings the document at the URL specified by *url* into view. This method may not be supported by applet viewers. The placement of the documents is specified where as described in the text.
void showStatus (String *str*)	Displays *str* in the status window.

Within an applet, once you have obtained the applet's context, you can bring another document into view by calling **shownDocument( )**. This method has no return value and throws no exception if it fails, so use it carefully. There are two showDocument ( ) methods. The method **showDocument(URL)** displays the document at the specified URL. The method **show**

Document(URL, where) displays the specified document at the specified location within the browser window. Valid arguments for where are "_self" (show in current frame), "_parent" (show in browser window). You can also specify a name, which causes the document to be shown in a new browser window by that name.

The following applet demonstrates **AppletContext showDocument( )**. Upon execution, if obtains the current applet context and uses that context to transfer control to a file called Test.html. This file must be in the same directory as the applet. Test.html can contain any hypertext that you like.

```
            /* Using an applet context, get CodeBase( ),
               and showDocument ( ) to display an HTML file,
            /*
            import java.awt.*;
            import java.applet,*;
            import java.net.*;
            /*
            <applet code = "ACDemo" width = 300 height = 50>
     </applet>
             */
             public class ACDemo
             extends Applet {
              public void start( )    {
                AppletContext ac = getAppletContext( );
                URL url = getCodeBase( ); // get url of this applet
               try {
                 ac.showDocument(New URL (url + "Test.html"));
               } catch (MalformedURLException e)  {
                 showsStatus ("URL not found");
               }
              }
            }
```

12.3 VARIOUS APPLETS

1. ***Chkr:*** It is the applet which displays the password checker role. It includes how to check the constructs using label-checking rules. It is a procedure, which accepts a database of passwords, which includes a password, a user name, and returns a boolean indicating whether the string is the right password for that user.
2. ***Font:*** Text can be displayed in a wide variety of ways in Java. Multiple fonts can be created in java that allows you to display text in different ways. In Java, a font is defined as an instance of a typeface that specifies the following information:
   (i) The typeface used to display the font
   (ii) The size of the font
   (iii) Special effects, such as italics or bolding

We have shown below the line of code, when the font is created:

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

The above line of code creates a font where the typeface is "Arial". The font will be 30 points tall. The font will also be bold.

Once the font has been created, it must be handed by the "Graphics" object that is to be used for text display. You can create multiple fonts and select which font to use by calling the "setFont" method on the "Graphics" object. This is shown as below.

```
g.setFont(font);
```

On the selection of the required font, any text that is displayed with "drawString" will be displayed using the new font.

```
g.drawString("Hello",10,y);
```

3. *Ga:* It is an optimizing algorithm(genetic algorithm) that is obtained after the evolution of organisms. Java program is developed based on hybrid algorithm using GA for rapid solution of traveling salesperson problem.

4. *Lbg:* There is a well-known algorithm, namely LBG algorithm [Linde, Buzo and Gray, 1980], which is used for clustering a set of L training vectors into a set of M codebook vectors.

5. The LBG algorithm designs an M-vector codebook in stages. Firstly, it by designs a 1-vector codebook, then it uses a splitting technique to initialize the search for a 2-vector codebook, and then it continues the splitting process until the desired M-vector codebook is obtained.

6. *Rc:* This java applet is used to show the transient behavior that occurs when the capacitor is being charged and discharged.

   Rrc: RRC that is radio remote control software is the latest development in radio control that makes use of the most advanced technologies to be found in computer networks today . Most users access RRC via Internet Explorer. The software incorporates a wide variety of tools, Java-based web browsers and relational database management systems as well as client/server architectures. This radio control software allows configurations and settings such as frequencyand modulation. Errors, for example, can be detected rapidly through continuous monitoring.

7. *Sp:* The ScrollPane(sp) is very handy for graphical Components.

8. *Common.html file:* An applet is a program that you embed right into your web page. Placing an applet in your page is like putting a little window in the middle of the page, and one of these Java programs is running in that window. Applets themselves are not written in HTML. They are separate programs written in java language. No special web server is needed to put applets in the web page. You can use the same server that serves your web pages.

   If you know how to put pictures in your web page, you will find that putting applets in your page is very similar (though not identical). The applet program (called a "class" file) is a separate file from your HTML file. You use the tags <APPLET ...> and <PARAM ...> to place the applet into your web page, much the same way you use <IMG ...> to place a picture in your page.

12.4 GRAPHICS AND TEXT

12.4.1 To draw Lines

To draw line we start from one coordinate(x , y) and end with (x1,y1) coordinate.

Method: drawLine()

Syntax: drawLine(int x, int y , int x1 , int y1);

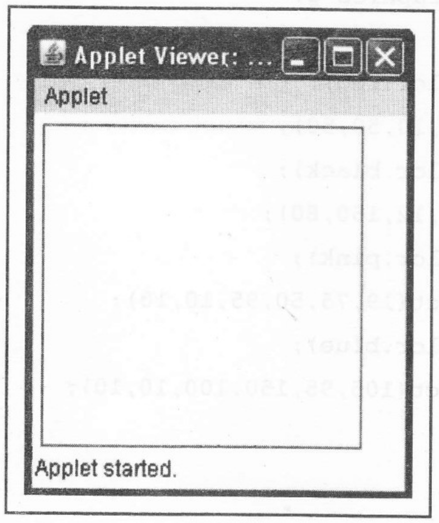
Example:

```
import java.applet.*;
import java.awt.*;
public class LineExample extends Applet
{
        public void init()
        {
                resize(300,500); // to set the size of the display window
        }
        public void paint(Graphics g)
        {
                g.setColor(Color.red);
                g.drawLine(5,5,180,5);
                g.drawLine(180,5,180,180);
                g.drawLine(180,180,5,180);
                g.drawLine(5,5,5,180);
        }
}/* <applet code="LineExample.class" width=200 height=200>
</applet>
*/
```

Outpurt:

```
F:\mypackage\Use\myapplet>javac LineExample.java

F:\mypackage\Use\myapplet>appletviewer LineExample.java
```



12.4.2 To draw Rectangles

To draw hollow rectangle of the specified width and height starting from particular x and y position (i.e., starting point of the rectangle).

Method :	drawRect()
Syntax:	drawRect(int x , int y, int width , int height);

To draw solid rectangle of the specified width and height starting from particular x and y position (i.e., starting point of the rectangle).

Method:	fillRect()
Syntax:	fillRect(int x , int y, int width , int height);

To draw solid rectangle of the specified width and height starting from particular x and y position with rounded corners.

Method:	drawRoundRect()
Syntax:	drawRoundRect(int x, int y , int width , int height , int arcwidth , int archeight);

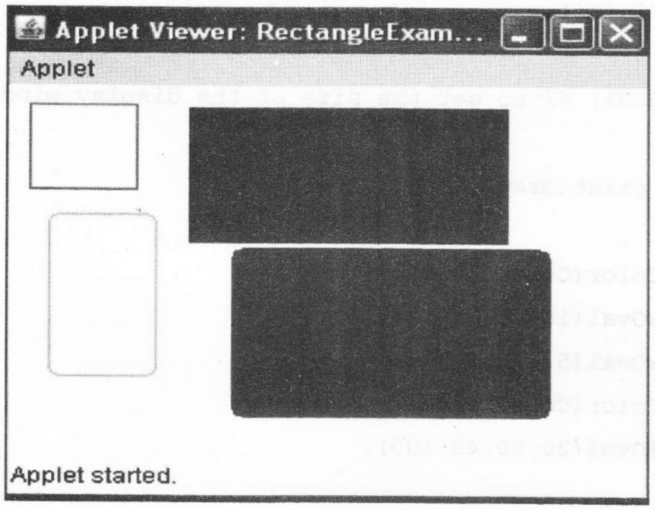
Example:

```
import java.applet.*;
import java.awt.*;
public class RectangleExample extends Applet
{
        public void init()
        {
        resize(300,500); // to set the size of the display window
        }
        public void paint(Graphics g)
        {
                g.setColor(Color.red);
                g.drawRect(10,10,50,50);
                g.setColor(Color.black);
                g.fillRect(85,12,150,80);
                g.setColor(Color.pink);
                g.drawRoundRect(19,75,50,95,10,10);
                g.setColor(Color.blue);
                g.fillRoundRect(105,95,150,100,10,10);
        }
}
```

```
/* <applet code="RectangleExample.class" width=100 height=100>
</applet>*/
```

Output:

```
F:\mypackage\Use\myapplet>javac RectangleExample.java
F:\mypackage\Use\myapplet>appletviewer RectangleExample.java
```



12.4.3 To draw Circle and Ellipses

To draw an ellipse or circle we use the following methods:

Method: drawOval()

Syntax: drawOval(int x , int y , int width , int height);

This method draws a circle or an ellipse such that it fits within the rectangle specified by the x, y, width and height arguments. The center of the rectangle whose origin is (x, y) in this graphics context's coordinate system and whose size is specified by the width and height argument as shown in Figure 12.2.

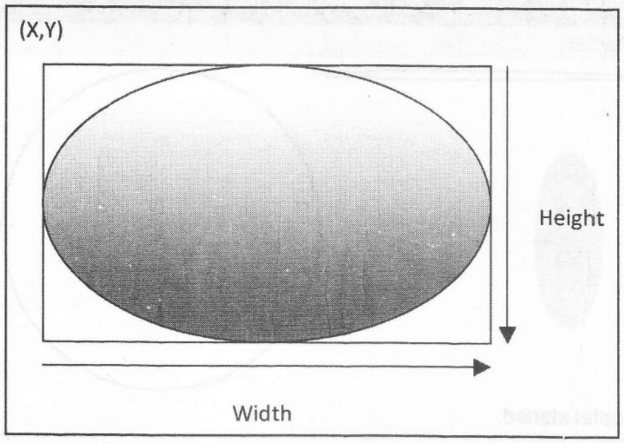


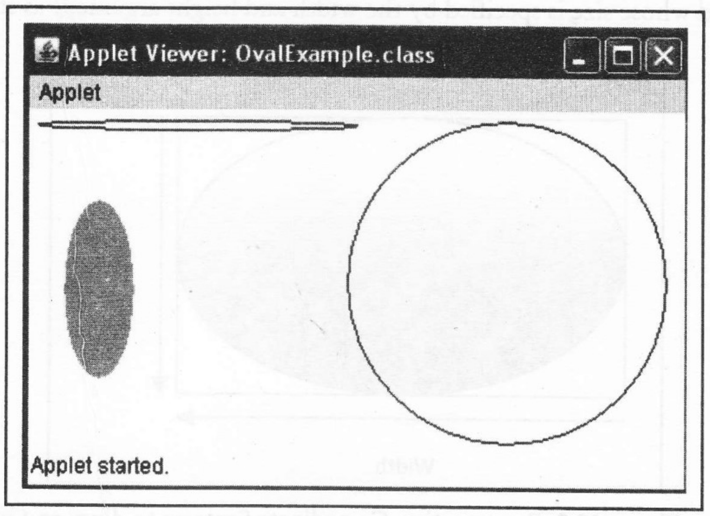
Figure 12.2: Representing Co-ordinatr Systems to draw an Ovral

Example:

```
import java.applet.*;
import java.awt.*;
public class OvalExample extends Applet
{
        public void init()
        {
        resize(300,500); // to set the size of the display window
        }
        public void paint(Graphics g)
        {
                g.setColor(Color.blue);
                g.drawOval(180,5,180,180);
                g.drawOval(5,5,180,5);
                g.setColor(Color.pink);
                g.fillOval(20,50,40;100);
        }
}
/* <applet code="OvalExample.class" width=200 height=200>
</applet>
```

Output:

```
F:\mypackage\Use\myapplet>javac OvalExample.java
F:\mypackage\Use\myapplet>appletviewer OvalExample.java
```



12.4.4 To draw Arc

An arc is a part of an oval, we can think of an oval as a series of arcs that are connected together in an orderly manner.

Method: drawArc()

Syntax: drawArc(int x, int y, int width , int height , int start angle , int angle of sweep);

The first four are the same as the argument of drawOval() method and last two represents the starting angle and number of degree (sweep angle) around the arc.

In drawing arcs, java actually formulates the arcs as an oval and then draws only a part of it as dictated by the last two arguments. Java consider the 3 O' clock position as zero degree position and degrees increase in anticlockwise direction as shown in Figure 12.3.

Suppose we have to draw an arc from 9 O' clock to 3 O' clock then starting angle will be 1800 and angle of sweep will also be 1800 as shown in Figure 12.3.

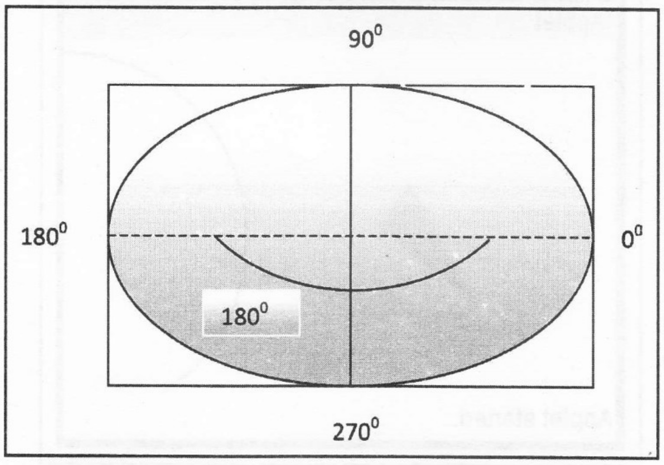


Figure 12.3: Arc as a part of Oval

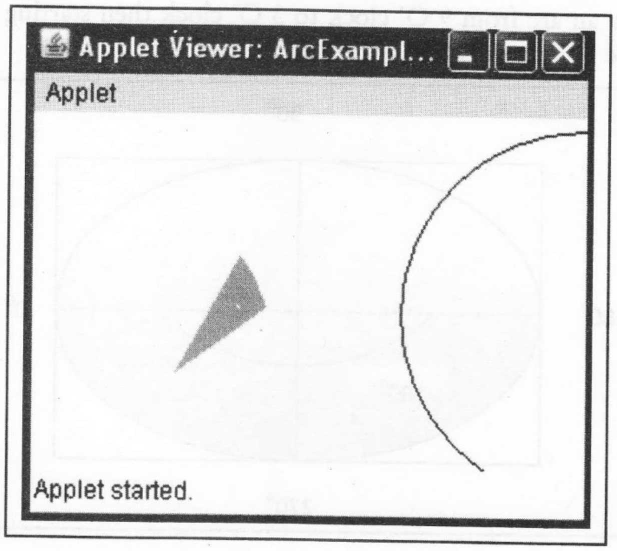
Example:

```
import java.applet.*;
import java.awt.*;
public class ArcExample extends Applet
        {
        public void init()
        {
        resize(300,500); // to set the size of the display window
        }
public void paint(Graphics g)
        {
                g.setColor(Color.blue);
```

```
            g.drawArc(180,5,180,180,90,180);
            g.setColor(Color.pink);
            g.fillArc(20,50,100,150,25,25);
      }
}
/* <applet code="ArcExample.class" width=200 height=200>
</applet>
*/
```

Output:

```
F:\mypackage\Use\myapplet>javac ArcExample.java
F:\mypackage\Use\myapplet>appletviewer ArcExample.java
```



12.4.5 To draw Polygon

The Graphics drawPolygon method can be used to draw outline polygons, and the Graphics fillPolygon method can be used to draw filled polygons.

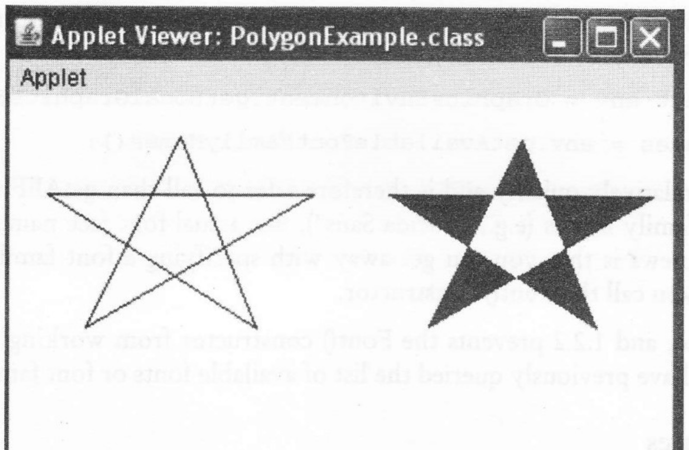
Method:	drawPolygon()
Syntax:	drawPolygon(int[] XArray, int[] YArray, int N);
XArray:	Array of the horizontal locations of the vertices of the polygon, in pixels.
YArray:	Array of the vertical locations of the vertices of the polygon, in pixels.

Example:

```
import java.applet.Applet;
import java.awt.*;
public class PolygonExample extends Applet
{
public void paint (Graphics g)
{
int Cursor;
int[] XArray = {20, 160, 40, 90, 130};
int[] YArray = {50, 50, 120, 20, 120};
g.drawPolygon (XArray, YArray, 5);
Cursor = 0;
while (Cursor < 5)
{
XArray [Cursor] = XArray [Cursor] + 180;
Cursor = Cursor + 1;
 }
g.fillPolygon (XArray, YArray, 5);
}
}
/* <applet code="PolygonExample.class" width=350 height=200>
</applet>
```

Output:

```
F:\mypackage\Use\myapplet>javac PolygonExample.java
F:\mypackage\Use\myapplet>appletviewer PolygonExample.java
```



12.5 FONTS AND TEXT

As we've seen, fonts are represented in AWT by the java.awt.Font class. While you can continue to use fonts in Java 1.2 exactly as you did in Java 1.1, Java 2D has added a number of powerful new features related to fonts and text rendering that you may want to take advantage of.

12.5.1 Available Fonts

Java 1.0 and Java 1.1 support only a small set of fonts, specified by logical font names. Although these logical fonts are guaranteed to be available on every platform, they are not guaranteed to look the same on every platform. In addition, the lack of variety severely limits the design choices available to developers.

Java 1.2 allows an application to use any font installed on the native system and refer to that font by its physical font name, instead of a logical font name. A physical font name is the actual name of a font, such as "Century Gothic" or "Lucida Sans Bold." To request a specific font, simply pass its physical name to the Font() constructor. The Font() constructor always returns a valid Font object, even if the font you have requested does not exist. If you need to check whether you got the font you requested, call the getFontName() method of the returned font.

If you want to be sure that a font exists on the host system before attempting to use it, you should first query the system to find out what fonts are installed. You can do this with methods of the java.awt.GraphicsEnvironment object. The code looks like this:

```
GraphicsEnvironment env = GraphicsEnvironment.getLocalGraphicsEnvironment();
Font[] allfonts = env.getAllFonts();
```

The getAllFonts() method returns an array of Font objects that represents all of the fonts installed on the system. Each Font object in this array represents a font that is one point high, so you have to scale the font (using deriveFont() as explained shortly) before using it. Also, in the initial release of Java 1.2 at least, the getAllFonts() method can take prohibitively long to return (65 seconds on my Windows 95 system).

Another GraphicsEnvironment method, getAvailableFontFamilyNames(), returns an array of font family names instead of an array of Font objects:

```
GraphicsEnvironment env = GraphicsEnvironment.getLocalGraphicsEnvironment();
String[] familynames = env.getAvailableFontFamilyNames();
```

This method returns relatively quickly and is therefore safer to call than getAllFonts(). Note that this method returns font family names (e.g., "Lucida Sans"), not actual font face names (e.g., "Lucida Sans Oblique"). The good news is that you can get away with specifying a font family name instead of a font face name when you call the Font() constructor.

A bug in Java 1.2, 1.2d, and 1.2.2 prevents the Font() constructor from working with any nonlogical font name unless you have previously queried the list of available fonts or font family names.

12.5.2 Font Attributes

In Java 1.2, the Font class has a new constructor that is passed a java.util.Map object that contains a set of font attributes. These attributes specify the desired characteristics of the font; the Font() constructor tries to return a Font that matches the attributes. Typically, you use a java.util.Hashtable or java.util.Hashmap to hold your attribute values. The attribute names or keys are constants defined in java.awt.font.TextAttribute. The important constants are FAMILY, SIZE, WEIGHT, and POSTURE. The TextAttribute class also defines commonly used values for the WEIGHT and POSTURE attributes.

12.5.3 Transforming Fonts

The Font class defines several deriveFont() methods that allow you to use a Font object to create related Font objects. deriveFont() is typically used to return a new Font object that represents an existing font at a different size or in a different style. For example:

```
GraphicsEnvironment env = GraphicsEnvironment.getLocalGraphicsEnvironment();

String[] familynames = env.getAvailableFontFamilyNames();

Font regularFont = new Font("Century Schoolbook", Font.PLAIN, 12);

Font bigFont = regularFont.deriveFont(18.0f);

Font boldFont = regularFont.deriveFont(Font.BOLD);

Font bigBoldFont = regularFont.deriveFont(Font.BOLD, 24.0f);
```

When you are passing a point size to deriveFont(), be sure to explicitly specify a float value, such as the 18.0f constant in the preceding code, so that you do not inadvertently call the version of deriveFont() that takes an integer-style constant.

You can also derive a transformed version of a Font object by passing in an arbitrary java.awt.geom.AffineTransform object. This technique allows you to arbitrarily rotate or skew any font, as we'll discuss later in the chapter.

12.5.4 Text-Rendering Hints

The java.awt.RenderingHints class defines two hints that apply particularly to text drawing. The first controls antialiasing. Antialiasing is a technique used to make the jagged edges of shapes, such as the glyphs of a font, look smoother. It is implemented using translucent colors and compositing: when the edge of a shape only partially covers a pixel, the color used to draw that pixel is given an alpha-transparency value that corresponds to the amount of coverage. If a fully covered pixel is drawn with an opaque color, a pixel that is only one-quarter covered is drawn with an alpha value of .25. As you can imagine, antialiasing can be computationally intensive. However, the smoothing effect it achieves is significant and is particularly useful when drawing small amounts of text at large point sizes.

The first text-related rendering hint simply requests antialiasing for text. If you want text to be antialiased, set the KEY_TEXT_ANTIALIASING hint to VALUE_ TEXT_ANTIALIAS_ON. There is also a more general hint, KEY_ANTIALIASING. Java 2D defines a separate hint for text so that you can choose independently whether to request antialiasing for text and other graphics.

12.5.5 Measuring Text and Fonts

Sometimes you need to obtain measurement information about a font or measure text before you can draw text. For example, to horizontally center a string of text, you must be able to figure out how wide it is. To correctly draw multiple lines of text, you need to be able to query the baseline position and the interline spacing for the font. In Java 1.0 and Java 1.1, you obtained this information with the FontMetrics class (as described near the beginning of the chapter).

Java 2D provides another way to measure the width of a string of text. The Font class defines several getStringBounds() methods that return the width and height of a specified string as a Rectangle2D object. These methods allow widths to be returned as floating-point numbers instead of integers and are therefore more accurate than the stringWidth() method of FontMetrics. Each variant of getStringBounds() allows you to specify a string of text in a different way. What these methods have in common, however, is that they must all be passed a FontRenderContext object. This object contains information needed to accurately measure text. It includes information about whether antialiasing and fractional metrics are being used, for example. You can obtain an appropriate FontRenderContext by calling the getFontRenderContext() method of a Graphics2D object.

12.5.6 Advanced Text Drawing

The easiest way to display text in an application is to use a Swing component such as a JLabel, JTextField, JTextArea, or JEditorPane. Sometimes, however, you have to draw text explicitly, such as when you are implementing a custom Swing component.

The easiest way to draw text is with the drawString() method of Graphics or Graphics2D. drawString() is actually a more complex method than you might think. It works by first taking the characters of a string and converting them to a list of glyphs in a font. There is not always a one-to-one correspondence between characters and glyphs, however, and font encodings usually do not match the Unicode encoding used for characters. Next, the method must obtain the measurements of each glyph in the list of glyphs and position it individually. Only after these steps can the method actually perform the requested string drawing operation.

12.5.7 setPaintMode

```
public abstract void setPaintMode()
```

Sets the paint mode of this graphics context to overwrite the destination with this graphics context's current color. This sets the logical pixel operation function to the paint or overwrite mode. All subsequent rendering operations will overwrite the destination with the current color.

Check Your Progress

Fill in the blanks:

1. Applets are .... driven.
2. The .... class defines two hints that apply particularly to text drawing.
3. To execute an applet in a ...., you need to write a short HTML text file that contains the appropriate APPLET tag.
4. The Applet tag is HTML allows you to pass .... to your applet.

12.6 LET US SUM UP

An applet arrives on the client it has limited access to resources, so that it can produce an arbitrary multimedia user interface and run complex computations without introducing the role of viruses or breaching data integrity. After you enter the source code for simple Applet, compile in the same way that you have been compiling programs. Applets are event driven. An applet resembles a set of interrupt service routines. It waits until an event occurs. The AWT notifies the applet about an event by calling an event handler that has been provided by the applet. The Applet tag is HTML allows you to pass parameters to your applet. To retrieve a parameter, use the **getparameter( )** method. Java allows an application to use any font installed on the native system and refer to that font by its physical font name, instead of a logical font name. A physical font name is the actual name of a font, such as "Century Gothic" or "Lucida Sans Bold."

12.7 KEYWORD

Applets: The small applications that are accessed from an Internet server, transported over the Internet, automatically installed and run as part of a web document.

12.8 QUESTIONS FOR DISCUSSION

1. Explain the lifecycle of Applets.
2. What are the following steps included in applets initializing and termination?
3. Write a program for drawing lines and polygons.
4. Write a code for passing parameters to the applets.

Check Your Progress: Model Answers

1. Event
2. java.awt.RenderingHints
3. web browser
4. parameters

12.9 SUGGESTED READINGS

E. Balaguruswami, *Programming with Java*, Tata McGraw-Hill.

Davis, Stephen R., *Learn Java Now*, Microsoft Press.

Naughton, Patrick, *The Java Hand Book*, OSborne McGraw- Hill.

Sams.net, *Java unleased.*

Herbert Schildt, *The Complete Reference Java 2*, Tata McGraw-Hill.