UNIT V

GRAPHICS PROGRAMMING, INPUT / OUTPUT FILES IN JAVA

GRAPHICS PROGRAMMING

Graphics is one of the most important features of Java. Java applets can be written to draw lines, arcs, figures, images and text in different fonts and styles. Different colors can also be incorporated in display.

THE GRAPHICS CLASS

The graphics class defines a number of drawing functions, Each shape can be drawn edge-only or filled. To draw shapes on the screen, we may call one of the methods available in the graphics class. The most commonly used drawing methods included in the graphics class are listed below. To draw a shape, we only need to use the appropriate method with the required arguments.

DRAWING METHODS OF THE GRAPHICS CLASS

Sr.No	**Method**	**Description**
1.	clearRect()	Erase a rectangular area of the canvas.
2.	copyAre()	Copies a rectangular area of the canvas to another area
3.	drawArc()	Draws a hollow arc
4.	drawLine()	Draws a straight line
5	drawOval()	Draws a hollow oval
6	drawPolygon()	Draws a hollow polygon
7	drawRect()	Draws a hollow rectangle
8.	drawRoundRect()	Draws a hollow rectangle with rounded corners
9.	drawString()	Display a text string
10.	FillArc()	Draws a filled arc
11.	fillOval()	Draws a filled Oval
12.	fillPolygon()	Draws a filled Polygon
13.	fillRect()	Draws a filled rectangle
14.	fillRoundRect()	Draws a filled rectangle with rounded corners
15.	getColor()	Retrieves the current drawing color
16.	getFont()	Retrieves the currently used font
17.	getFontMetrics()	Retrieves the information about the current font
18.	setColor()	Sets the drawing color
19.	setFont()	Sets the font

LINES AND RECTANGLES

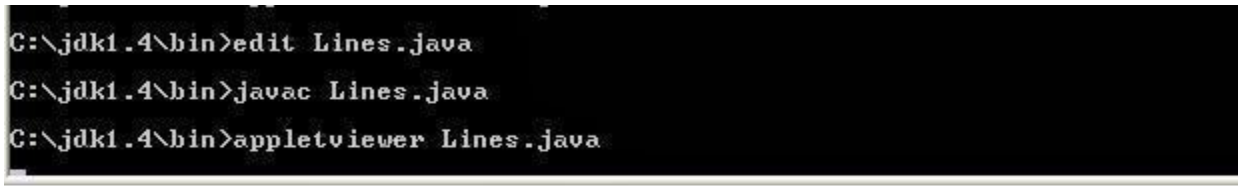
Lines are drawn by means of the drawLine() method.

Syntax
void drawLine(int startX, int startY, int endX, int endY)

drawLine() displays a line in the current drawing color that begins at (start X, start Y) and ends at (endX, end Y).

```
//Drawing Lines
import java.awt.*;
import java.applet.*;
/*
<applet code="Lines" width=300 Height=250>
</applet>
*/
public class Lines extends Applet
{
        public void paint(Graphics g)
        {
                g.drawLine(0,0,100,100);
                g.drawLine(0,100,100,0);
                g.drawLine(40,25,250,180);
                g.drawLine(5,290,80,19);
        }
}
```

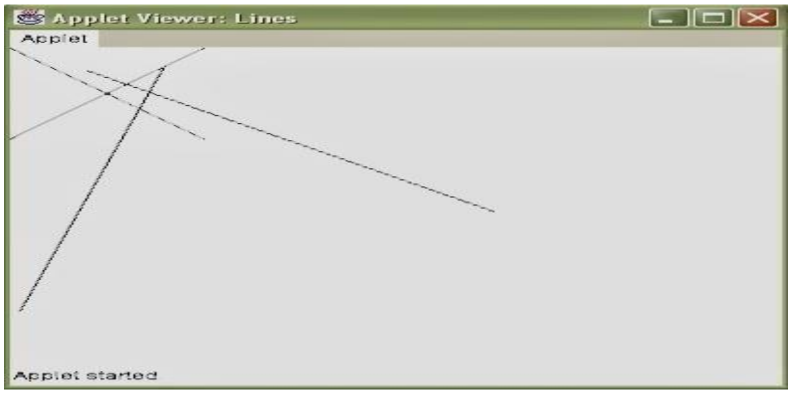
After this you can comiple your java applet program as shown below:



```
C:\jdk1.4\bin>edit Lines.java

C:\jdk1.4\bin>javac Lines.java

C:\jdk1.4\bin>appletviewer Lines.java
```



"Output of Lines.class"

RECTANGLE

The drawRect() and fillRect() methods display an outlined and filled rectangle, respectively.

Syntax
void drawRect(int top, int left, int width, int height) void
fillRect(int top, int left, int width, int height)

The upper-left corner of the rectangle is at(top,left). The dimensions of the rectangle are specified by width and height.

Use drawRoundRect() or fillRoundRect() to draw a rounded rectangle. A rounded rectangle has rounded corners.

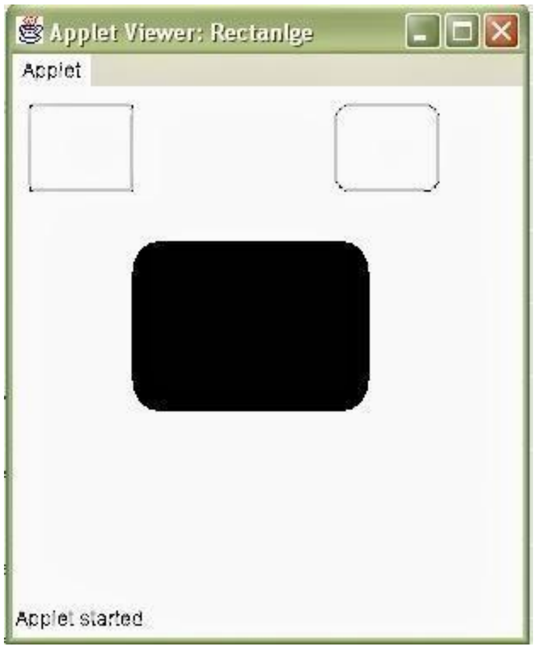
Syntax
void drawRoundRect(int top, int left, int width, int height int Xdiam, int YDiam) void
fillRoundRect(int top, int left, int width, int height int Xdiam, int YDiam)

The upper-left corner of the rounded rectangle is at (top,left). The dimensions of the rectangle are specified by width and height. The diameter of the ribdubg are along the X axis are specified by x Diam. The diameter of the rounding are along the Y axis is specified by Y Diam.

```
import java.awt.*;
import java.applet.*;
/*
<applet code="Rectanlge" width=300 Height=300>
</applet>
*/
public class Rectanlge extends Applet
{
    public void paint(Graphics g)
    {
        g.drawRect(10,10,60,50);
        g.fillRect(100,100,100,0);
        g.drawRoundRect(190,10,60,50,15,15);
        g.fillRoundRect(70,90,140,100,30,40);
    }
}
```

After this you can comiple your java applet program as shown below:

```
C:\Program Files\Java\jdk1.7.0\bin>javac Rectanlge.java C:\Program
Files\Java\jdk1.7.0\bin>appletviewer Rectanlge.java
```



"Output of Rectangle.class"

CIRCLES AND ELLIPSES

The Graphics class does not contain any method for circles or ellipses. To draw an ellipse, use drawOval(). To fill an ellipse, use fillOval().

Syntax
void drawOval(int top, int left, int width, int height)
void fillOval(int top, int left, int width, int height)

The ellipse is drawn within a bounding rectangle whose upper-left corner is specified by (top,left) and whose width and height are specified by width and height. To draw a circle, specify a square as the bounding rectangle i.e get height = width.

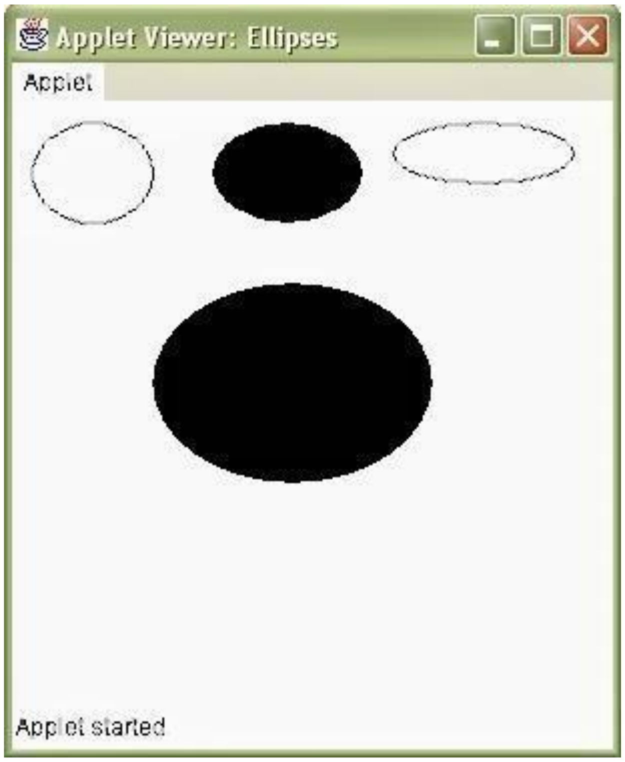
The following program draws serveral ellipses:

```
import java.awt.*;
import java.applet.*;
/*
<applet code="Ellipses" width=300 Height=300>
</applet>
*/
public class Ellipses extends Applet
{
    public void paint(Graphics g)
    {
        g.drawOval(10,10,60,50);
        g.fillOval(100,10,75,50);
        g.drawOval(190,10,90,30);
```

```
            g.fillOval(70,90,140,100);
        }
    }
```

After this you can comiple your java applet program as shown below:

C:\Program Files\Java\jdk1.7.0\bin>javac Ellipses.java C:\Program Files\Java\jdk1.7.0\bin>appletviewer Ellipses.java



"Output of Ellipses.class"

DRAWING ARCS

An arc is a part of oval. Arcs can be drawn with draw Arc() and fillArc() methods.

Syntax
void drawArc(int top, int left, int width, int height, int startAngle, int sweetAngle)
void fillArc(int top, int left, int width, int height, int startAngle, int sweetAngle)

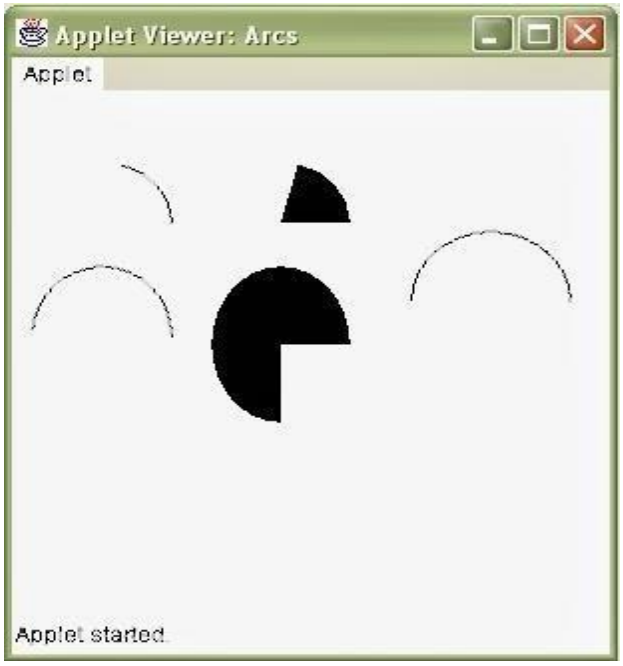
The arc is bounded by the rectangle whose upper-left corner is specified by (top,left) and whose width and height are specified by width and height. The arc is drawn from startAngle through the angular distance specified by sweepAngle. Angles are specified in degree. '0º' is on the horzontal, at the 30' clock's position. The arc is drawn conterclockwise if sweepAngle is positive, and clockwise if sweetAngle is negative.

The following applet draws several arcs:

```
import java.awt.*;
import java.applet.*;
/*
<applet code="Arcs" width=300 Height=300>
</applet>
*/
public class Arcs extends Applet
{
      public void paint(Graphics g)
      {
            g.drawArc(10,40,70,70,0,75);
            g.fillArc(100,40,70,70,0,75);
            g.drawArc(10,100,70,80,0,175);
            g.fillArc(100,100,70,90,0,270);
            g.drawArc(200,80,80,80,0,180);
      }
}
```

After this you can comiple your java applet program as shown below:

```
C:\Program Files\Java\jdk1.7.0\bin>javac Arcs.java C:\Program
Files\Java\jdk1.7.0\bin>appletviewer Arcs.java
```



"Output of Arcs.class"

DRAWING POLYGONS

Polygons are shapes with many sides. It may be considered a set of lines connected together. The end of the first line is the beginning of the second line, the end of the second line is the beginning of the third line, and so on. Use drawPolygon() and fillPolygon() to draw arbitrarily shaped figures.

Syntax

void drawPolygon(int ,[] , int y[], int numPointer)
void fillPolygon(int ,[] , int y[], int numPointer)

The polygon's endpoints are specified by the coordinate pairs contained within the x and y arrays. The number of points defined by x and y is specified by numPoints.

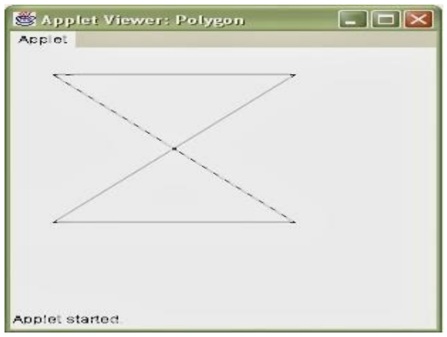
It is obvious that x and y arrays should be of the same size and we must repeat the first point at the end of the array for closing the polygon.

The following appplet draws an hourglass shape: import

```
java.awt.*;
import java.applet.*;
/*
<applet code="Polygon" width=300 Height=300>
</applet>
*/
public class Polygon extends Applet
{
    public void paint(Graphics g)
    {
        int   xpoints[]={30,200,30,200,30};
        int   ypoints[]={30,30,200,200,30};
        int num=5;
        g.drawPolygon(xpoints,ypoints,num);
    }
}
```

After this you can comiple your java applet program as shown below:

```
C:\Program Files\Java\jdk1.7.0\bin>javac Polygon.java
C:\Program Files\Java\jdk1.7.0\bin>appletviewer
Polygon.java
```



JAVA AWT (ABSTRACT WINDOW TOOLKIT)

In Today's World the Most important Interface is the Graphical Interface which provides the Facilities for the user to create their Applications which contains all the Graphical elements. So that JAVA provides various GUI Classes and their Methods those are organized in the form of Packages. The Most important Packages are the Java's AWT Packaged Which Contains 63 Classes and 14 Interface which provides various Functions for a user to create his Application

There are 63 Classes and 14 interfaces provided by the AWT Packages which Provides facility for a user to create his Application. And AWT contains various Controls those Plays a Vital Role in the Graphical Application For Example Labels, Command Buttons, Text Boxes, Checkboxes and Radio Buttons and Lists etc. So that AWT provides all these Components in the Form of Classes. So For using any class first you have to import AWT Packages and then create the object of that Control which you want.

For creating any type of user component, first create the component and then add it to the panel or applet for adding a component to an applet the add method of container class is used.

Introducing Abstract Window Toolkit

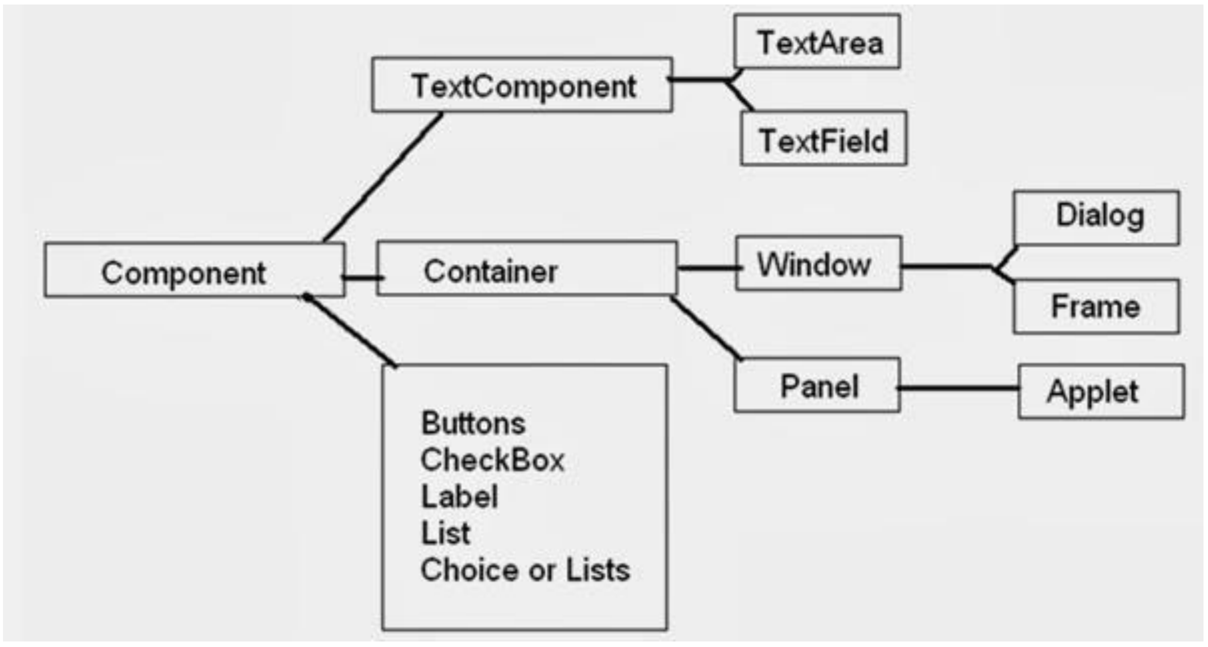
Window Fundamentals

Java Abstract window tool kit package is used for displaying the data within a GUI Environment. Features of AW T Package are as Followings:

1. It provides us a set of user interface components including windows buttons text fields scrolling list etc.
2. It provides us the way to laying out these above components.
3. It provides to create the events upon these components.

The main purpose for using the AWT is using for all the components displaying on the screen. Awt defines all the windows according to a class hierarchy those are useful at a specific level or we can say arranged according to their functionality.

The most commonly used interface is the panels those are used by applets and those are derived from frame which creates a standard window. The hierarchy of this Awt is:-



As you see this Figure this Hierarchical view display all the classes and also their Sub Classes. All the Classes are Contained as a Multi-Level inheritance , As You see that Component Class contains a Panel Class which again Contains a Applet Class so that in the AWT Package are Stored in the form of Multi-values inheritance.

Component Class

This is the super class which defines all the methods for developing the methods those are related with the components and the methods those are related with the Actual Position of the Control and this also provides the various methods to Setup the Sized, Colors either this may a Background or a Foreground Color. It includes all the components those are displayed on the screen and it also defines various types of public methods for managing events such as mouse and keyboard window etc. the Various Methods those are Contained in the Component Class are :-

1. **Void SetSize(int Width, int Height)** : this Method is used to Set the Size by taking the height and width of a Control for Example height of Textbox and width of Textbox. With this Method we Can set height and width of any Control.
2. **Void SetLocation (int x,int y)**: Used to Set the location on the Applet and this will take the Coordinates in the Form of x and y.
3. **Void setBounds(int x,int y,int width,int height)** : this Method will also set the Acual Loction in the Applet by Specifying the X and y Coordinate and also Specify the Width and Height of the Control.
4. **Void setForeground(Color c)** : this Method is used to set the Font Color or Text Color of a Control.
5. **Void setBackground(Color c)** : This Method is used to set the Background Color of a Control whether this may a Control or any Window Like an Applet,Panel etc.

Container

It is subclass of components which Inherits the various Properties from that Class and it provides to laying out the component , that are contains by using the various layout managers. Means how the Components will be displayed in the applet and in which Manner the Components will be displayed in the window. or Position of the Controls without setting the X and y Axis is determined by the Layout Manager and. The Container Provides very important Methods to add or Remove the Components from the Applet or from any other Window. The various Methods those are Contained in the Container class are :

1. **Void add (object o)** :this Method id used to Add a Control in the Applet of in any Container.
2. **Void remove(object o)** : This is used if a user wants to Remove an Control from a Container For Example if a user wants to Remove the Control from a Applet then he may uses this Method.
3. **void removeAll()** : This Method doesn't take any argument and this will remove all the Controls from the Container.

Panel

It is a subclass of container and it is the super class of Applet. It is a window that does not have a title bar menu bar or border. Generally a Panel is used for displaying Multiple Windows in a Single Window and a Panel may have any Layout. And for Adding any Control in the Panel we have to use the Add Method and there are also some important Methods those are Provided by the Panel Class those are Explained below :-

1. **Panel()** : This is the Name of Class For Using you have to Create the object first.
2. **Void setLocation** : Method is used to set the Location of a control in the Panel.
3. **Void setSize()** : This will takes two Arguments and those Arguments will Specify the Width and Height of the Panel.
4. **Void setBounds** : This Method is used to set the Bounds of an Control in the Panel For Example X and y Axis and Width and Height of a Control.

Window

Window is also a Sub Class of a Container and window class creates a top level window. These are not directly created, for this the subclass of window name frame is used and dialogs are used . This is used for displaying a Sub Windows from a Main Window and the various Methods of Window are as follows:-

1. **Void Pack()** : This Method is used for adjusting the Size of Window according to the number of Controls in the Container or in the Applet or in the other window.
2. **Void show()** : This Method is used for displaying a Window.
3. **Void dispose()** : This Method is used for de-allocate the Memory Space which is Consumed by the Controls of the Window.

MANAGING INPUT OUTPUT FILES IN JAVA

The java.io package contains nearly every class you might ever need to perform input and output I/O in Java.

All these streams represent an input source and an output destination.

The stream in the java.io package supports many data such as primitives, Object, localized characters, etc.

Stream A stream can be defined as a sequence of data. there are two kinds of Streams

InPutStream: The InputStream is used to read data from a source.

OutPutStream: the OutputStream is used for writing data to a destination.

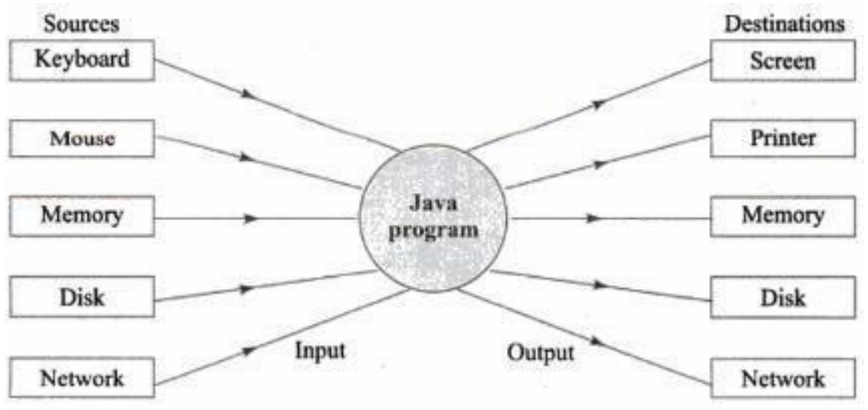
Java provides strong but flexible support for I/O related to Files and networks but this tutorial covers very basic functionality related to streams and I/O. We would see most commonly used example one by one:

Byte Streams Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams but the most frequently used classes are , FileInputStream and FileOutputStream.

Following is an example which makes use of these two classes to copy an input file into an output file:

Stream – Stream is nothing but flow of data, which is used to transfer data from one place to another place

The Java I/O (Input/Output) package java.io contains a group of interfaces and classes



STREAM CLASSES

The java.io package contains a large number of stream classes that provide capabilities for processing all types of data. These classes may be categorized into two groups based on the data type on which they operate.

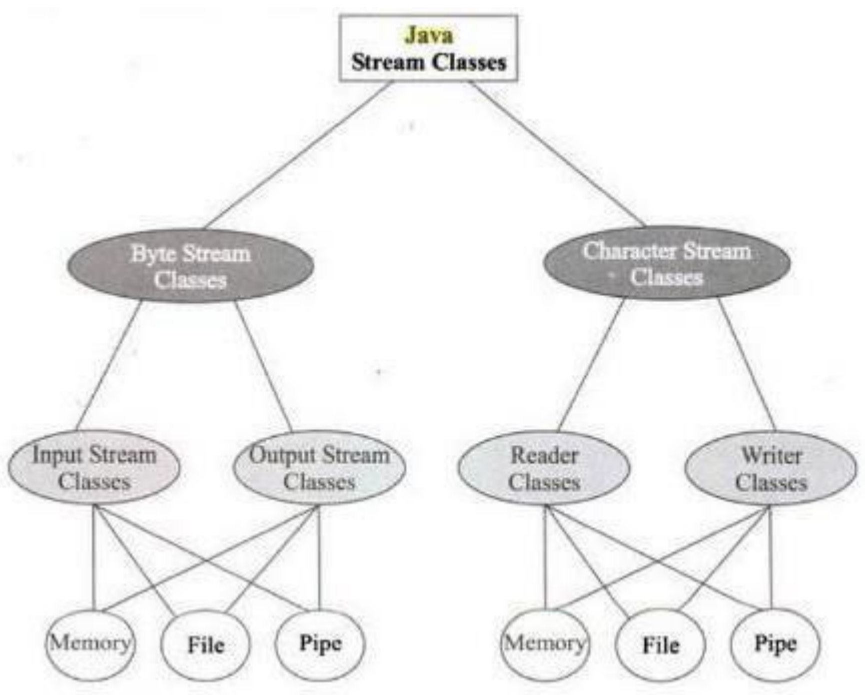
Types of Streams

1. **Byte stream classes**
2. **Character stream classes**

Byte Stream : It provides a convenient means for handling input and output of byte.

Character Stream : It provides a convenient means for handling input and output of characters.

Character stream uses Unicode and therefore can be internationalized.

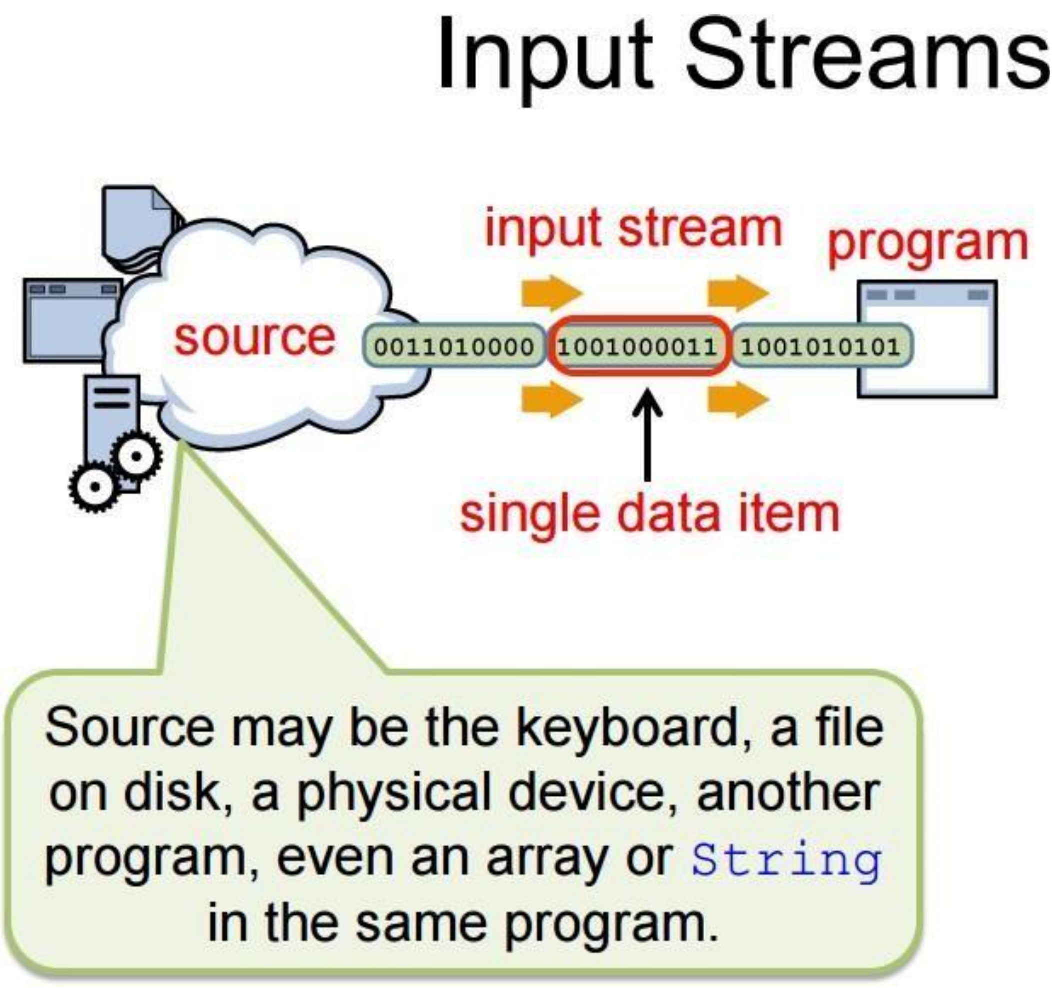


Byte Stream Classes

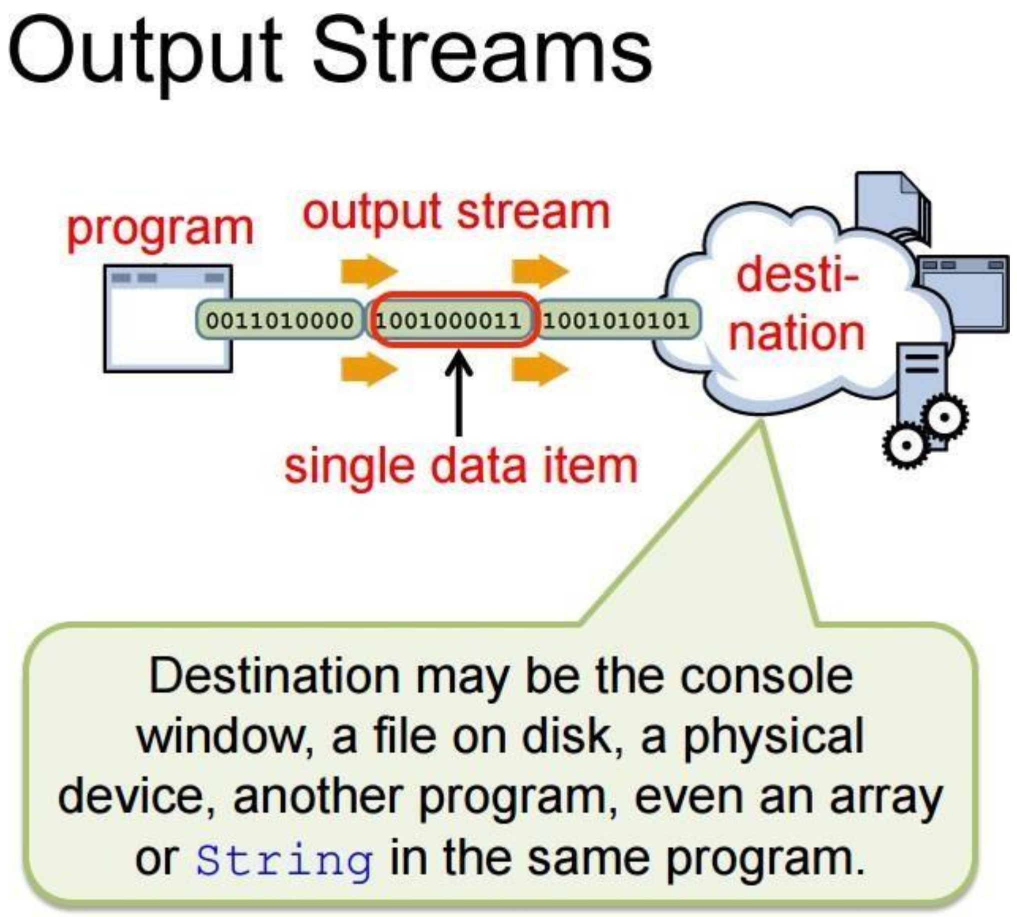
Byte stream classes have been designed to provide functional features for creating and manipulating streams and files for reading and writing bytes. Java provides two kinds of byte stream classes: **input stream classes** and **output stream classes**.

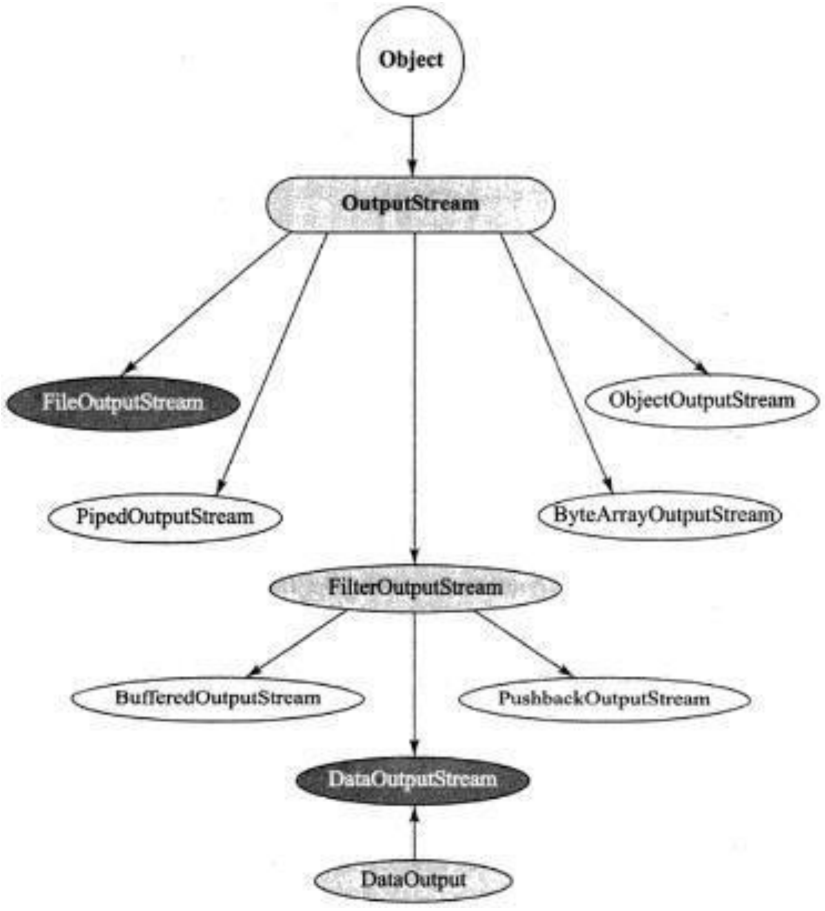
Input Stream Classes

Input stream classes that are used to read bytes include a super class known as **Inputstream** and a number of subclasses for supporting various input-related functions. The super class InputStream is an **abstract** class, and, therefore, we cannot create instances of this class. Rather, we must use the subclasses that inherit from this class.


Input Streams
input stream
program
source
0011010000
1001000011
1001010101
single data item
Source may be the keyboard, a file on disk, a physical device, another program, even an array or String in the same program.

Output Stream Classes
Output stream classes are derived from the base class **Outputstream** like InputStream, the OutputStream is an **abstract** class and therefore we cannot instantiate it. The several subclasses of the OutputStream can be used for performing the output operations.


Output Streams
program
output stream
0011010000
1001000011
1001010101
desti-
nation
single data item
Destination may be the console window, a file on disk, a physical device, another program, even an array or String in the same program.

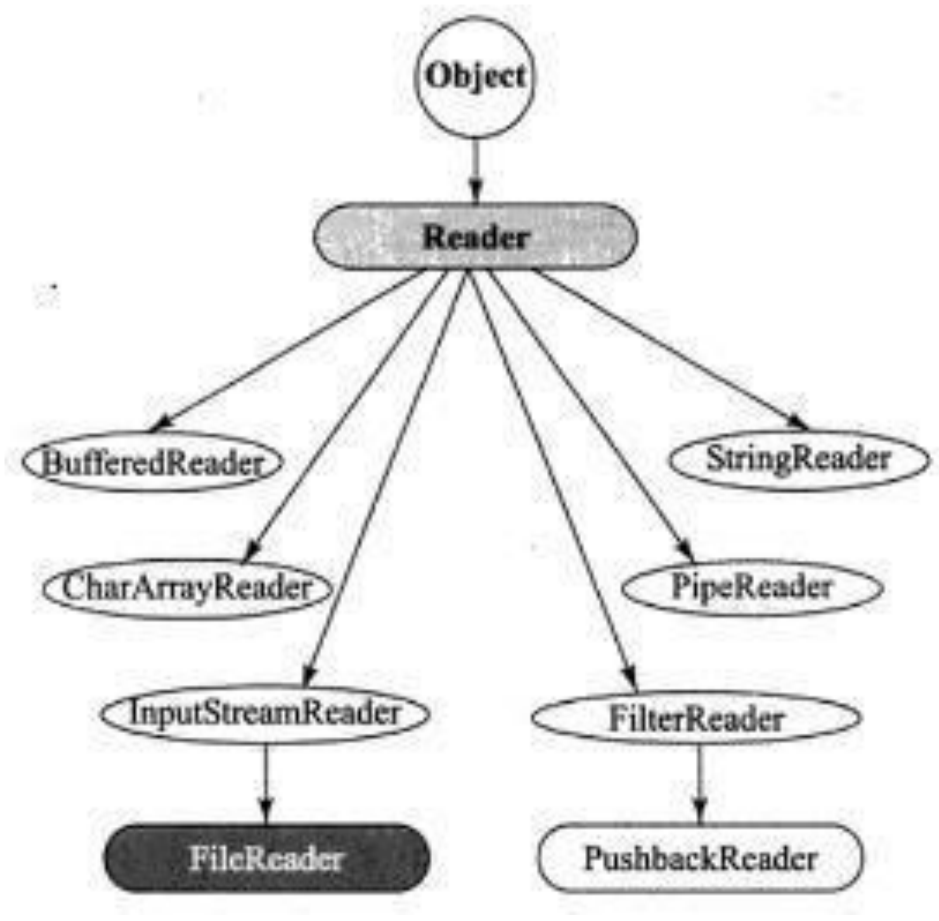

Object
OutputStream
FileOutputStream
PipedOutputStream
FilterOutputStream
ByteArrayOutputStream
ObjectOutputStream
BufferedOutputStream
PushbackOutputStream
DataOutputStream
DataOutput

Character Streams-

Java Byte streams are used to perform input and output of 8-bit bytes, where as Java Character streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams but the most frequently used classes are , FileReader and FileWriter.. Though internally FileReader uses FileInputStream and FileWriter uses FileOutputStream but here major difference is that FileReader reads two bytes at a time and FileWriter writes two bytes at a time.

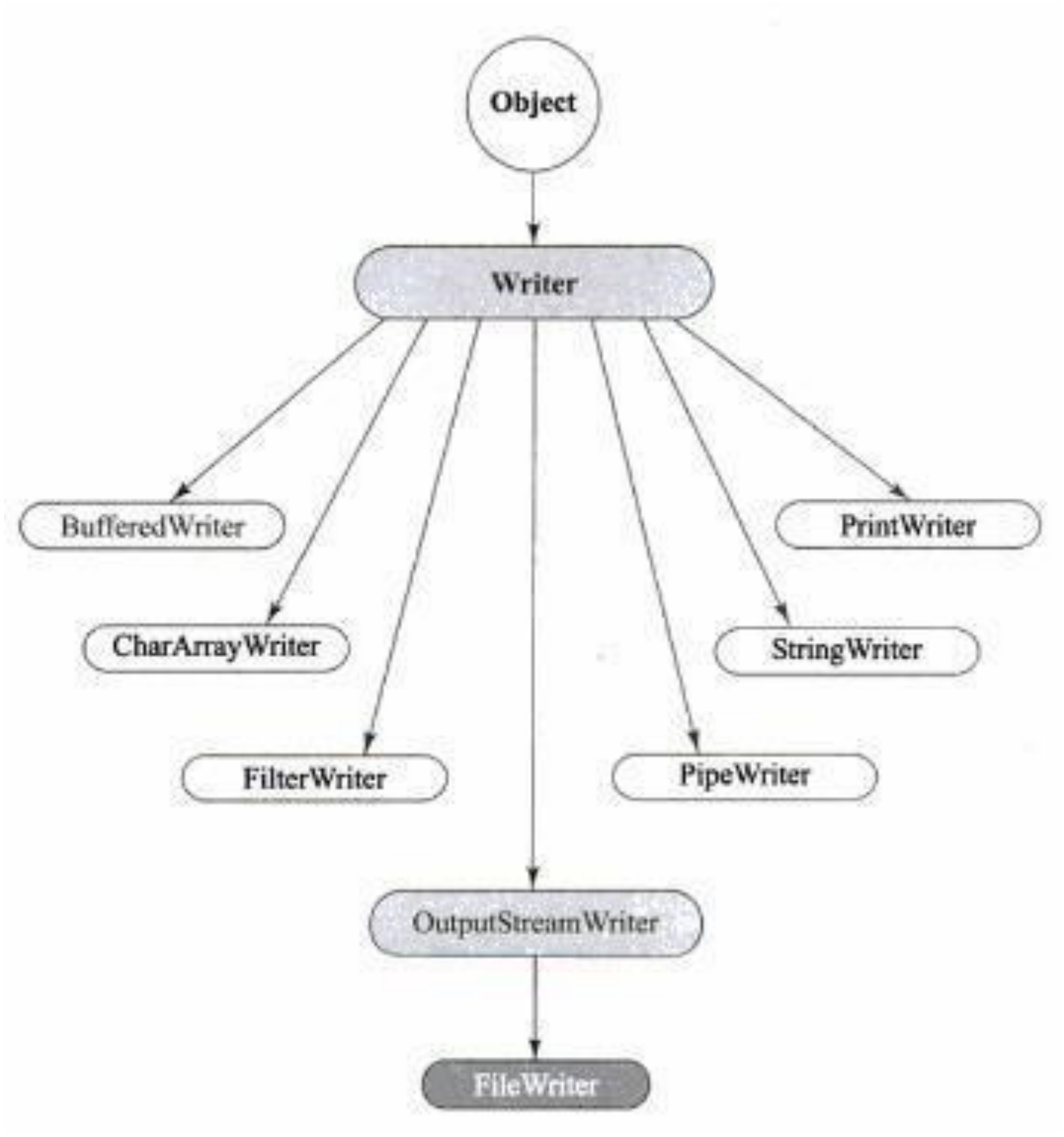
Reader Stream Classes

Reader stream classes that are used to read characters include a super class known as **Reader** and a number of subclasses for supporting various input-related functions. Reader stream classes are functionally very similar to the input stream classes, except input streams use bytes as their fundamental unit of information, while reader streams use characters. The Reader class contains methods that are identical to those available in the InputStream class, except Reader is designed to handle characters. Therefore, reader classes can perform all the functions implemented by the input stream classes.



Writer Stream Classes

Like output stream classes, the writer stream classes are designed to perform all output operations on files. Only difference is that while output stream classes are designed to write bytes, the writer stream are designed to write character. The **Writer** class is an **abstract** class which acts as a base class for all the other writer stream classes. This base class provides support for all output operations by defining methods that are identical to those in Outputstream class.



Program to copy bytes from one file to another.

```
import java.io.*;
class ByteRead
{
	public static void main(String args[])
	{
		System.out.println("Process Started");
		FileInputStream fis=null;
		FileOutputStream fos=null;
		try
		{
			fis=new FileInputStream("in.dat");
			fos=new FileOutputStream("out.dat");
			byte data;
			while((data=(byte)fis.read())!=-1)
			{
				fos.write(data);
			}
		}

		catch(IOException e)
		{
			 System.out.println(e);
		}


		finally
		{
			try
			{
				fis.close();
				fos.close();
			}
			catch(IOException e)
			{
			}
		}
		System.out.println("Process Completed - Data is copied successfully");
	}
}
//Save it as in.dat in bin folder
IPL Team List-2016
1. RCB
2. MI
3. SH
4. DD
5. KIXP
6. RPS
7. GL
8. KKR

C:\Program Files\Java\jdk1.7.0\bin>javac ByteRead.java
C:\Program Files\Java\jdk1.7.0\bin>java ByteRead
Process Started
Process Completed - Data is copied successfully
```

Open **Out.dat file**

```
IPL Team List-2016
1. RCB
2. MI
3. SH
4. DD
5. KIXP
6. RPS
7. GL
8. KKR
```