Java means DURGA SOFT..

CORE JAVA

Material

India's No.1 Software Training Institute

DURGASOFT

www.durgasoft.com Ph: 9246212143 ,8096969696

Java .io package

The data stored in computer memory in two ways.

1) ***Temporary storage.***

   RAM is temporary storage whenever were executing java program that memory is created when program completes memory destroyed. This type of memory is called volatile memory.

2) ***Permanent storage.***

   When we write a program and save it in hard disk that type of memory is called permanent storage it is also known as non-volatile memory.

When we work with stored files we need to fallow following task.

1) Determine whether the file is there or not.
2) Open a file.
3) Read the data from the file.
4) Writing information to file.
5) Closing file.

Stream :-

Stream is a channel it support continiuos flow of data from one palce to another place
Java.io is a package which contains number of classes by using that classes we are able to send the data from one place to another place.

In java language we are transferring the data in the form of two ways:-

1. Byte format
2. Character format

Stream/channel:-

It is acting as medium by using steam or channel we are able to send particular data from one place to the another place.

Streams are two types:-

1. Byte oriented streams.(supports byte formatted data to transfer)
2. Character oriented stream.(supports character formatted data to transfer)

Byte oriented streams:-

Java.io.FileInputStream

byte channel:-

1)FileInputStream

public native int read() throws java.io.IOException;

public void close() throws java.io.IOException;

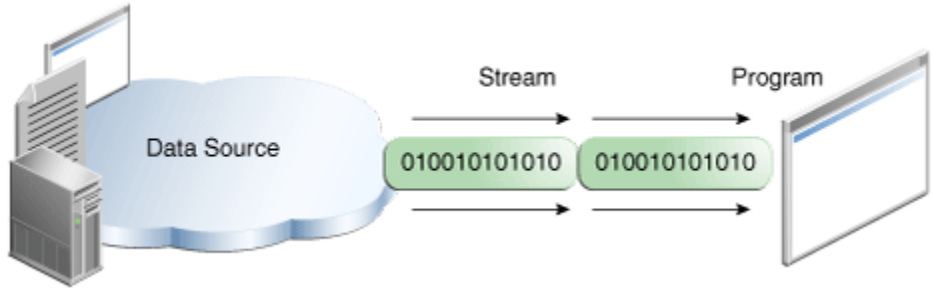
2)FileOutputStream

public native void write(int) throws java.io.IOException;

public void close() throws java.io.IOException;

To read the data from the destination file to the java application we have to use FileInputSream class.

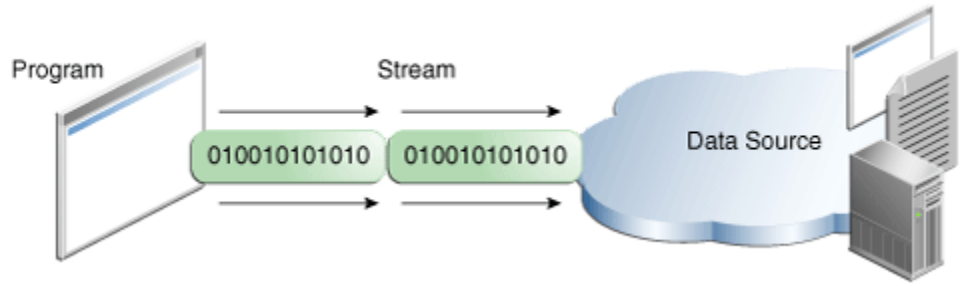
To read the data from the .txt file we have to read() method.



Java.io.FileOutputStream:-

To write the data to the destination file we have to use the FileOutputStream.

To write the data to the destination file we have to use write() method.



Ex:- it will supports one character at a time.

```
import java.io.*;
class Test
{       public static void main(String[] args)throws Exception
        {       //Byte oriented channel
                FileInputStreamfis = new FileInputStream("abc.txt");//read data from source file
                FileOutputStreamfos = new FileOutputStream("xyz.txt");//write data to target file
                int c;
                while((c=fis.read())!=-1)               //read and checking operations
                {       System.out.print((char)c);      //printing data of the file
                        fos.write(c);                   //writing data to target file
```

```
            }
            System.out.println("read() & write operatoins are completed");
        //stream closing operations
        fis.close();
        fos.close();
        }
}
```


Example :-

```
import java.io.*;
class Test
{       public static void main(String[] args)throws Exception
        {       FileReaderfr = new FileReader("abc.txt");//read data from source file
                FileWriterfw = new FileWriter("xyz.txt");//write data to target file
                int c;
                while((c=fr.read())!=-1)                //read and checking operations
                {       System.out.print((char)c);      //printing data of the file
                        fw.write(c);                    //writing data to target file
                }
                System.out.println("read() & write operatoins are completed");
        //stream closing operations
        fr.close();
        fw.close();
        }
}
```

Line oriented I/O:-

Character oriented streams supports single character and line oriented streams supports single line data.

BufferedReader:- to read the data line by line format and we have to use readLine() to read the data.

PrintWriter :- to write the data line by line format and we have to use println() to write the data.

Example :-

```
import java.io.*;
```

```
class Test
{       staticBufferedReaderbr;
        staticPrintWriter pw;
        public static void main(String[] args)
        {       try{
                br=new BufferedReader(new FileReader("get.txt"));
                pw=newPrintWriter(new FileWriter("set.txt"));
                String line;
                while ((line=br.readLine())!=null)//reading & checking
                {       System.out.println(line);          //printing data of file
                        pw.println(line);          //writing  data to target file
                }
                //close the streams
                br.close();
                pw.close();
                }
                catch(IOExceptionio)
                {       System.out.println("getting IOException");
                }
        }
}
```

Buffered Streams:-

Up to we are working with non buffered streams these are providing less performance because these are interact with the hard disk, network.
Now we have to work with Buffered Streams
BufferedInputStream read the data from memory area known as Buffer.

We are having four buffered Stream classes

1. BufferedInputStream
2. BufferedOutputStream
3. BufferedReader
4. BufferedWriter

Ex:-

```
import java.io.*;
class Test
{       staticBufferedReaderbr;
        staticBufferedWriterbw;
        public static void main(String[] args)
        {       try{
                br=new BufferedReader(new FileReader("Test1.java"));
                bw=new BufferedWriter(new FileWriter("States.java"));
                String str;
                while ((str=br.readLine())!=null)
                {       bw.write(str);
                }
                br.close();
                bw.close();
                }
                catch(Exception e)
                {       System.out.println("getting Exception");
                }
        }
}
```

Ex:-

```
import java.io.*;
class Test
{       staticBufferedInputStreambis;
        staticBufferedOutputStreambos;
        public static void main(String[] args)
        {       try{
                bis=new BufferedInputStream(new FileInputStream("abc.txt"));
                bos=new BufferedOutputStream(new FileOutputStream("xyz.txt"));
                intstr;
                while ((str=bis.read())!=-1)
                {       bos.write(str);
                }
                bis.close();
                bos.close();
                }
                catch(Exception e)
                {       System.out.println(e);
                        System.out.println("getting Exception");
                }
        }
}
```

Ex:-

```
import java.io.*;
class Test
```

```
{       public static void main(String[] args) throws IOException
        {       BufferedReaderbr=new BufferedReader(new FileReader("abc.txt"));
                String str;
                        while ((str=br.readLine())!=null)
                        {       System.out.println(str);
                        }

        }
}
```

Serialization:-

The process of saving an object to a file (or) the process of sending an object across the network is called serialization.

But strictly speaking the process of converting the object from java supported form to the network supported form of file supported form.

To do the serialization we required fallowing classes

1. FileOutputStream
2. ObjectOutputStream

Deserialization:-

The process of reading the object from file supported form or network supported form to the java supported form is called deserialization.

We can achieve the deserialization by using fallowing classes.

1. FileInputStream
2. ObjectInputStream

```
import java.io.*;
classEmp implements Serializable
{       inteid;
        String ename;
        Emp(inteid,Stringename)
        {this.eid=eid;
        this.ename=ename;
        }
        public static void main(String[] args)throws Exception
        {
Emp e = new Emp(111,"ratan");

//serialization [write the object to file]
FileOutputStreamfos = new FileOutputStream("xxxx.txt");
ObjectOutputStreamoos = new ObjectOutputStream(fos);
oos.writeObject(e);
System.out.println("serialization completed");

//deserialization  [read object form text file]
FileInputStreamfis = new FileInputStream("xxxx.txt");
ObjectInputStreamois = new ObjectInputStream(fis);
Emp e1 = (Emp)ois.readObject();//returns Object
```

```
System.out.println(e1.eid+"----"+e1.ename);
System.out.println("de serialization completed");
        }
}
```

Transient Modifiers

- Transient modifier is the modifier applicable for only variables and we can't apply for methodsand classes.
- At the time of serialization, if we don't want to save the values of a particular variable to meetsecurity constraints then we should go for transient modifier.
- At the time of serialization JVM ignores the original value of transient variable and default valuewill be serialized

```
import java.io.*;
classEmp implements Serializable
{       transientinteid;
        transient String ename;
}
0----null
```