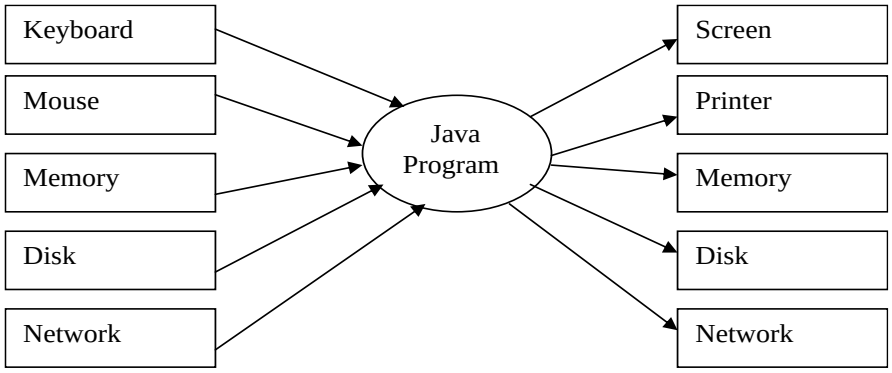
CHAPTER 7 : INPUT OUTPUT

In file processing, input refers to the flow of data into a program and output means the flow of data out of the program.

Java uses the concept of streams to represent the ordered sequence of data. A stream presents a uniform, easy-to-use, object-oriented interface between the program and the input/output devices. A stream in java is a path along which data flows. It has a source and a destination and it supports unidirectional movement of data.



The java.io package contains a large number of stream classes that provides capabilities for processing all types of data. These classes may be organized into two groups depending upon the data type they operate.

- Byte Stream Classes
- Character Stream Classes

Java perfoms IO operation in terms of bytes & characters. The java.io package has given 2 abstract class called: InputStream & OutputStream for byte oriented IO operation, similarly 2 more **abstract** classes called: Reader & Writer are used for character oriented IO operation.

Stream : A sequence of bytes of undetermined length is known as as a stream.

Java streams are classified into two basic types

- InputStream
- OutputStream

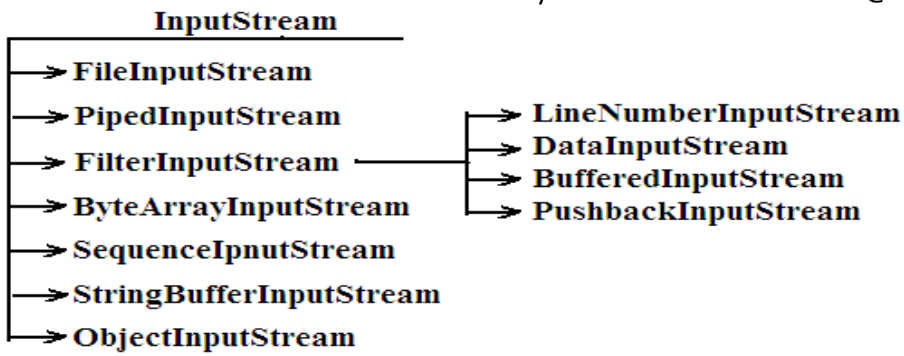
InputStream extract (i.e reads) data from the source (file) and sends it to the program.
OutputStream takes data from the program and sends(i.e writes) it to the destination (file).

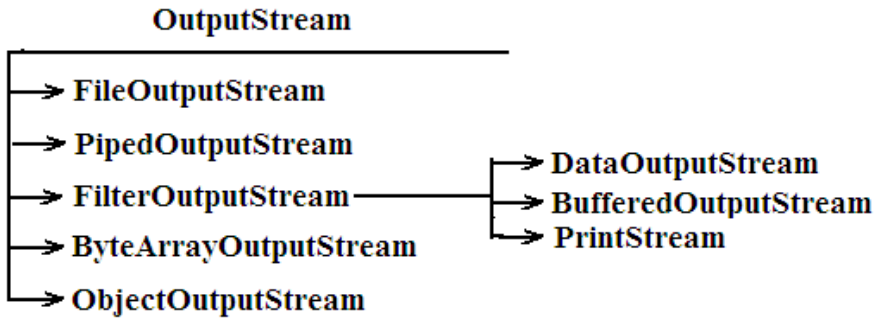
InputStream : When a java program reads a sequence of bytes from the external source then it is represented by InputStream & it's derived class. The InputStream class is also used to represent the byte oriented input devices for the java program, such as keyboard, scanner, etc

OutputStream : When a java program writes a sequence of bytes to external source then it is represented by OutputStream & it's derived class. The OutputStream class is also used to represent the byte oriented output devices for the java program, such as monitor, printer, etc

Reader : When a java program reads a sequence of characters from external source then it is represented by Reader & it's derived class.

Writer : When a java program writes a sequence of characters to the external source then it is represented by Writer & it's derived class.





Reading Methods of InputStream class:

1) abstract int read() : It will read a byte of data from the external source but returns as an integer.
2) int read(byte[] b) : It will read some bytes from the external source & stores them in the byte array b, returns the no. of bytes successfully read.
3) int read(byte[] b, int offset, int length) : It will read some bytes from the input stream & stores in byte array(as specified by 1st parameter) starting from offset(as specified by 2nd parameter) upto the length(as specified by 3rd parameter) of the byte array, and returns the no. of bytes successfully read.

Reading Methods of Reader class:

1) int read() : It will read a characters of data from the reader, but returns as in integer.
2) int read(char[] c) : It will read some characters from the external source and stores them into the char array c, returns the no. of characters successfully read.
3) abstract int read(char[] b, int offset, int length) : It will read some characters from the reader & stores in character array(as specified by 1st parameter) starting from offset(as specified by 2nd parameter) upto the length(as specified by 3rd parameter) of the character array, returns the no. of characters successfully read.

Writing Methods of OutputStream class:

1) abstract void write(int b) : It will write the specified byte to the external source.
2) void write(byte[] b) : It will write all bytes from this array(as specified by 1st parameter) to the external source.
3) void write(byte[] b, int offset, int length) : It will write a portion of byte array(as specified by 1st parameter) from the offset(as specified by 2nd parameter) upto the length(as specified by 3rd parameter) of the array to the external source.

Writing Methods of Writer class:

1) abstract void write(int c) : It will write the specified character to the external source.
2) void write(char[] c) : It will write all characters from this array to the external source.
3) void write(char[] c, int offset, int length) : It will write a portion of character array(as specified by 1st parameter) from the offset(as specified by 2nd parameter) upto the length(as specified by 3rd parameter) of the array to the external source.
4) void write(String str) : Writes entire string to the external source.
5) void write(String str, int off, int len) : Writes a portion of a string to the external source.

Other methods of IO :

1) void close() : Closes this input stream and releases any system resources associated with the stream.
   When we create an object of any class of IO package, then the object of InputStream/Reader behaves like a read pointer, and object of OutputStream/Writer behaves like write pointer by which we can perform the reading and writing operation on an external source(resource) respectively. As soon as the object is created then the IO operation will not occur automatically, the IO operation will start only when we invoke the methods of InputStream/OutputStream/Reader/Writer classes, once the read/write operation will complete then we must release the pointer, so that other classes of IO package may access the resources immediately. To release the pointer we've to use the close() method.
2) void flush() : When we write some data to the external device from the java program, we should guarantee that the data must be written to the external source nothing should remain in buffer, it can be done by using flush() method.
3) int available() : When we create an object of IO package to read data from external source(i.e. object of any derived class of InputStream or Reader), then by using available() method we can get the number of bytes left to be read.
4) long skip(long n) : This must be used by object of InputStream/Reader classes, and this method makes the read pointer to skips over and discards n(as specified by the parameter) bytes of from the current position of pointer.

Exceptions in IO : Generally most of the exception that may be generated in java.io package belongs to checked exception category.

1) EOFException : It will occur when we try to read beyond the end of file(eof) character. The integer value of eof is -1 in java/C/C++.
2) FileNotFoundException : When we try to read from a file that does not exist.
3) IOException : It will occur when the IO operation failed may be due to incorrect data.

File class : This class is avaialable in java.io package. This class is used to refer/access a file/ subdirectory from the java program, it is never used to perform io operation on afile, instead it is used to provide information about a file, such as what is the size of the file, when the file last modified, wheather the file is a read only, etc by its predefined method.

Constructors of File class :-

1) File(String filename/ dirname/ file&dirname)
2) File(String parent, String filename)
3) File(File parent, String filename)

ex:

Let us store a.txt under d:\demo folder and create object of File class.

File f1= new File("a.txt"); -->f1 object is created by 1st constructor. It can be used by the java program which is stored in same demo folder.

File f2= new File("d:\\demo"); -->f2 object is created by 1st constructor. It will refer to demo folder. It can be used by the java program which may be stored in any localtion. The single slash(\) is used as escape character in a String, hence we've to provide double slash(\\), to specify the separator in a path, it is called platform independent path separator in java.

File f3= new File("d:\\demo\\a.txt"); -->f3 object is created by 1st constructor. It will refer to a.txt file. It can be used by the java program which may be stored in any localtion.

File f4= new File("d:\\demo", "a.txt"); -->f4 object is created by 2nd constructor. It will refer to a.txt file. It can be used by the java program which may be stored in any location.

File f5= new File(f2, "a.txt"); -->f5 object is created by 3rd constructor. It will refer to a.txt file. It can be used by the java program which may be stored in any location.

```
import   java.io.*;
class  Listing {
 public static void main(String args[]) {
   File f1= new File("d:\\");
   String na[]= f1.list();
   System.out.println("total : "+na.length);
   System.out.println("--------------");

   System.out.println("All "+ args[0] +" file of d:");
```

```
int cnt=0;
   for(int i=0;i<na.length;i++)
     if(na[i].endsWith(args[0])) {
         cnt++;
         System.out.println(na[i]);
     }
   System.out.println("total java files : "+cnt);
}};
```

The list() method of File class return the name of the all file and folders of a particular direcotory,if the object of File class refers to a file, then it'll generate NullPointerException.

```
/* Find  .java/.class file in d:\ and its sub directory */
import   java.io.*;
class  Listing1 {
 static  String  s1;
 static  int cnt=0;
 static void  find(File  ob){
   String na[]= ob.list();
   for(int i=0;i<na.length;i++) {
     try{
       File f2=new  File(ob, na[i]);
        if(f2.isDirectory())
            find(f2);
```

```
          if(na[i].endsWith(s1)) {
              cnt++;
              System.out.println(na[i]);
           }
       }catch(Exception e){   }
  }
 }
 public static void main(String args[]) {
    s1=args[0];
   File f1= new File("d:\\");
   find(f1);
   System.out.println("total "+s1+" files : "+cnt);
}};
```

FileInputStream : This class is used to read bytes from a file to the java program.
FileInputStream(String name) : Creates an object of this class by opening a connection to an actual file and the java program, as the file name given in the parameter as a String.
FileInputStream(File ob) : Creates an object of this class by opening a connection to an actual file and the java program, as the file name given by the parameter as an object of File class.

The object created by using the above constructor will behave like a read pointer, by which a java program will read bytes from the file.

FileReader : This class is used to read characters from a file to the java program.

FileReader(String name) : Creates an object of this class by opening a connection to an actual and the java program, as the file name given in the parameter as a String.
FileReader(File ob) : Creates an object of this class by opening a connection to an actual file and the java program, as the file name given in the parameter as an object of File class.

The object created by using the above constructor will behave like a read pointer, by which a java program will read characters from the file. If the file is not available to read then FileNotFoundException will occur.

FileOutputStream : This class is used to write bytes to a file from the java program.

FileOutputStream(String name) : Creates an object of this class to write bytes to the file from the java program, as the file name is specified by the parameter as a String.
FileOutputStream(File file) : Creates an object of this class to write bytes to the file from the java program, as the file name is specified by the parameter as an object of File class.

If the file is not available for above constructors then a new file is created, if file is available then pervious content of the file will be lost by current writing operation.

FileOutputStream(String name, boolean append) : Creates an object of this class to write bytes to the file as specified by the 1st parameter as a String.
FileOutputStream(File ob, boolean append) : Creates an object of this class to write bytes to the file with the file object as specified by the 1st parameter as an object of File class.

When the 2nd parameter is false then these two constructor will behave like one argument constructor. But if the 2nd parameter is true and the file is not available then a new file is created, if file is available then current data will be appended at the end of previous data.

FileWriter : This class is used to write characters to a file.

FileWriter(String name) :
FileWriter(File file) :
FileWriter(String name, boolean append) :
FileWriter(File file, boolean append) :

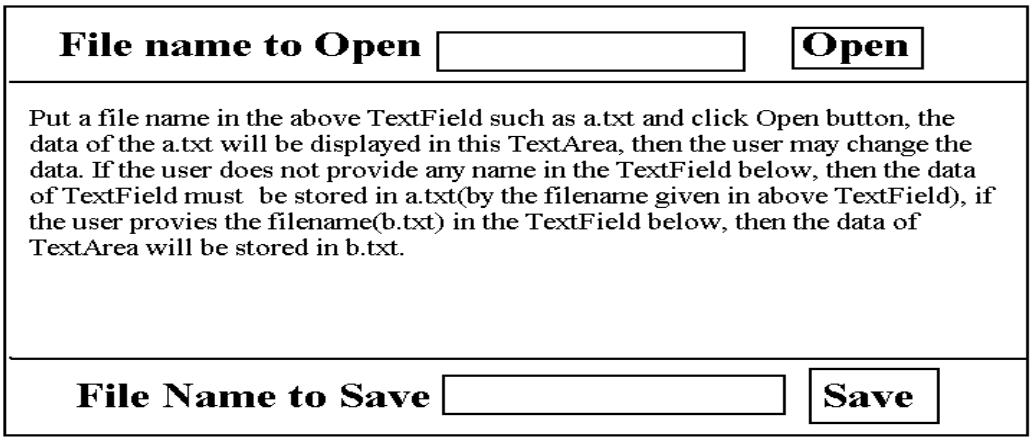
These constructors are similar to the constructor of FileOutputStream class, but these constructor will write charactes to a file.

FILE IO:

```
import java.io.*;
public class FileCopy  {
    public static void main(String args[])
throws Exception       {
  FileInputStream fin=new
         FileInputStream("a.txt");
   FileOutputStream fout= new
          FileOutputStream("b.txt");
    int c=0;
    while( (c=fin.read()) != -1)           {
        System.out.print((char)c);
         fout.write(c);
     }
     fin.close();
     fout.close();
  }};
```

```
import  java.io.*;
class  Copy1  {
  public static void main(String ar[]) throws Exception {
  FileInputStream fin = new FileInputStream("a.txt");
  FileOutputStream fout = new
FileOutputStream("b.txt", true);
  byte b[] = new  byte[fin.available()];
/* available() return the number of bytes to be read.
  File f1=new File("a.txt");
  byte b[] = new  byte[(int)f1.length()];
 */
 int d=fin.read(b); /* reading all data to b*/
  System.out.println(new String(b));
  fout.write(b);/* writing byte array b*/
  fin.close();
  fout.close();
}};
```

Q.



Q. WAP to copy the contents aa.txt & bb.txt into cc.txt as given below

aa.txt	bb.txt	cc.txt
AAAAA	11111	A1A1A1A1A1

BBBBB CCCCC DDDDD	22222 33333 44444 55555	B2B2B2B2B2 C3C3C3C3C3 D4D4D4D4D4 55555

Converting a byte array to String : String p = new String(b); where b is byte array.	**Converting a String to byte array :** byte b[] = s.getBytes(); where s is a String object.
Converting a char array to String : String p = new String(c); where c is char array.	**Converting a String to char array :** char c[] = s.toCharArray(); where s is a String object.

NOTE : There are some name which contain more than one word like Meghanada (Megha+nada) , Bishnupriya (Bishnu+priya), similarly most of the classes of IO package contains more than one word. We have came across the following classes :

File	InputStream
File	OutputStream
File	Reader
File	Writer
This part specifies the extenal source on which IO operation will be performed.	**This part specifies the type of IO operation, i.e. byte or character oriented IO operation.**

InputStreamReader class : This class will read byte from byte oriented input devices(because 1st part of this class is InputStream) such as keyboard and scanner, etc, but it returns a character (because 2nd part of this class is Reader) to the java program. Hence this class behaves like a broker to the java program.

InputStreamReader(InputStream ob) : The constructor has InputStream as a parameter, hence it may accept System.in(keyboard), FileInputStream, etc, so that it will read a byte from byte oriented input devices.

This class has a method called read() which will read a byte from underlying input device so that it will convert the byte into character by using the default encoding scheme of the operating system, and returns the character to the java program.

BufferedReader class : This class behaves like a temporary buffer for the java program. It will accept any subclass of Reader Hierarchy so that it will read a character from the subclass of Reader and stores it in the Buffer so as to provide for the efficient reading of characters, arrays, and lines.. This class has a method called readLine() which will return a line from the buffer as a String to the java program, only when it will get a new line character from the buffer, because this class uses new line as a delimeter for this method. It is the only class under Reader hierarchy that returns a line or string from the buffer to the java program by the

public String readLine( ) throws IOException

BufferedReader (Reader ob):- It will accept any subclass of Reader Hierarchy so that it will read a character from the subclass of Reader and stores it in the Buffer.

BufferedReader(Reader in, int size) :– This constructor accept a character input stream as the 1st parameter, and the no. of bytes to be read is given in 2nd parameter.

In this example we are creating an object of InputStreamReader that reads data from keyboard (i.e.System.in). This data is in the form of bytes. To convert the byte form data into character form we hava to wrap it into InputStreamReader and finally we convert the InputStream intoBufferedReader. Since we have used readLine() method to read data, we can say we are reading a line or String from keyboard.

// Reading char from keyboard	**Reading strings/lines from Keyboard**

```
import  java.io.*;
class    Read {
public static void main(String args[] )
throws Exception {
InputStreamReader   is  =   new
InputStreamReader( System.in );
System.out.println("enter data , '*' to
exit");
  int  c  =  is.read();
  while( c!=  '*' )   {
  System.out.print( (char) c );
  c  =  is.read();
}
is.close();
}};
```

```
import java.io.*;
class Reading  {
   public static void main(String args[]) throws  Exception{
InputStreamReader   is  = new
InputStreamReader(System.in);
BufferedReader   br  = new   BufferedReader(is);
        System.out.println("Enter data , type 'exit' to
terminate");
         String str = br.readLine();
         while(! str.equalsIgnoreCase("exit") ) {
                System.out.println(str);
                str = br.readLine();
         }
          br.close();
          is.close();
  }};
```

Q Write a program to read some data from a file in terms of String/Line by using BufferedRreader class.
Q. Create user defined Scanner class by using BufferedReader.

Text -file vs. Binary file:-

A file in which data is available in human readable form and the data can be edited by text editors(such as notepad, wordpad, etc) is called as Text file. It contains delimeters like space, new line, tab, EOF, etc.

A file in which data is not available in human readable form and the data cannot be edited by text editor is called Binary file. It does not contain any delimeter like space, new line, tab, EOF, etc. (ex. .gif/.jpg/.mp3/.dll/.ini/.obj/.exe/.class etc)

Character IO vs. Byte IO: -

When java performs IO operation in terms of byte by using InputStream & OutputStream classes, then it convert the data into byte by using the default encoding scheme of the system. Hence IO operation in terms of byte is faster, but it is platform dependent.

When java performs IO operation in terms of character by using Reader & writer classes, then it converts the data into byte & vice versa by using java encoding scheme. Hence IO operation in terms of character is slower but it is platform independent.

Reading & writing predefined datatype value by DataInputStream & DataOutputStream class:-

Whenever we are writing some data by the FileOutputStream or FileWriter class, then the individual byte representation of the character or digit that present in the data will be written to the external source i.e. to a file. When we open the file to read the data, we will get the same data. This mode of writing the data is known as text mode of writing and the data is known as Text data.

The DataOutputStream class allows us to write predefined data type value from the java program to the external source i.e. to a file in binary format. If we write an integer value from the java program to a file by DataOutputStream class within its writeInt() method, then the DataOutputStream class will collect 4 byte representation of integer value according to java and these 4 bytes will be given to FileOutputStream for writing in a file. When we open the file to read the value then we will get some other value, because notepad or any other text editor has no idea to represent continuous 4 byte into integer variable, rather they will represent individual byte into ASCII value & to display the character at the generated ASCII value. This mode of writing the data is called binary mode of writing & the data is called binary data. Similarly the DataInputStream class is used to read predefined datatype values from a file with the help of FileInputStream class.

Reading & Writing predefined data type Values :	**//Console Reading :**
import java.io.*; class DataTest {	import java.io.*; class ConsWr {

```
 public static void main(String args[]) throws Exception {
          DataOutputStream dout= new DataOutputStream(
new FileOutputStream("abc.txt"));
          BufferedReader br= new BufferedReader(new
InputStreamReader(System.in));
          System.out.println("Enter an int");
          int x=Integer.parseInt(br.readLine());
          dout.writeInt(x);
          dout.writeFloat(12.35f);
          dout.writeUTF("Ram Kumar");
          dout.flush();
          dout.close();
          DataInputStream din =new DataInputStream(new
FileInputStream("abc.txt"));
        System.out.println("from file :"+din.readInt()+"  : " +
din.readFloat() +" : " +din.readUTF());
         din.close();
}};
```

```
public static void main(String args[])
throws Exception {
Console c= System.console();
System.out.println("Enter name");
String na= c.readLine();
System.out.println("Enter password");
char  z[]= c.readPassword();
String pw = new String(z);
System.out.println("name "+na);
System.out.println("password "+pw);
 }}
```

Console class : The System class has a method called console() which will return an object of Console class only if there is a console(dos prompt) is available to the java program. If there is no dos prompt available to the java program(such as Applet, Servlet, etc) then it will return null.

The readLine() method of Console class is used to read some data from the dos prompt and returns it to the java program as a String.

The readPassword() method of Console class is used to read some data from the dos prompt without displaying the data on the dos prompt. It will return the data as a character array to the java program.

Serialization :

```
import java.io.*;
class MySerial  implements Serializable    {
      int x;    float y;     transient  String pass;
      MySerial(int a, float b, String c)  {     x=a;       y=b;      pass=c;    }
      public String toString()   {      return x + " - "+ y+" - "+pass;      }
};
 class Serial  {
    public static void main(String args[]) throws Exception      {
          MySerial ms =  new MySerial(10,52.5f,"Ram");
          System.out.println("Before : writting \n" + ms);
          ObjectOutputStream oos= new ObjectOutputStream(new FileOutputStream("Ser.txt"));
          oos.writeObject(ms);      oos.close();
          //Reading
          ObjectInputStream ois= new ObjectInputStream(new FileInputStream("Ser.txt"));
          MySerial ms2= (MySerial)ois.readObject();
          System.out.println("After reading "+ms2);
          ois.close();
}};
```

Serialization:- The ObjectOutputStream & ObjectInputStream classes are used to read and write an object in a file respectively.

Whenever we like to write an object in a file, then the state of object should be written in a file. The state of an object means the value of data members of the object.

The ObjectOutputStream class will never write only the state of an object in a file, because the ObjectInputStream class will not able to recreate the object by reading only the state of an object because it needs extra information such as the name of the class, name of data member to recreate the object.

A class whose object we like to write in a file must inherit from Serializable interface, because this interface will add extra information such as the name of data member & class into the state of an object.

The Serialiable, Remote, Cloneable interfaces has no method at all. These interface are known as Marker interface & Tagging interface, because they either modifies the state of an object or provides extra information into the state of an object.

The process of writing the state of an object along with extra information from the java program to external source is known as Serialization & the object is said to be serialized.

The process of recreating the object by reading its state along with extra information from the external source in the java program is called Deserialization and the object is said to be deserialized.

The serialisable interface provides persistency to the state of an object which is required when we transmit the object from the local jvm to external source or to other jvm,

PrintWriter & PrintStream classes(Writing line/String from java program) :- As BufferReader class is used to read a String from the external source to the java program similarly these 2 classes are used to write a String or line from the java program to the external sources by println() method, because println( ) has been overloaded in these two classes.

PrintStream (OutputStream ob, boolean true) : This constructor will accept any byte oriented O/P device in its parameter so that the String given in printIn( ) will be written to the external source in terms of byte.
PrintWriter (OutputStream ob, boolean true)
PrintWriter (Writer ob, boolean true) : The above 2 constructors will accept any byte oriented device or subclasses of OutputStream or subclasses of Writer so that the string given in println( ) will be converted either into byte or character depending upon the type of O/P device given in the constructor.

When the 2nd parameter of the above methods is true then it guarantees that nothing will remain in the buffer but everything will be written to the external source. It is similar to flush() method.

Writing a line/string from java :

```
import  java.io.*;
class  Writing {
public static void   main(String args[])  throws Exception
{
  //Writing  a String/line to monitor
PrintStream ps =new PrintStream(System.out, true);
  ps.println("11111111  1111  11 11111");
  ps.println("22222222");
  PrintWriter pw = new PrintWriter(System.out, true);
  pw.println("aaaaa aaa");
  pw.println("bbbbb  bbb");
  ps.close();
  pw.close();
  //Writing  a String/line to file
  PrintStream ps1= new   PrintStream (new
FileOutputStream("a.txt"));
    ps1.println("999 9999");
    ps1.println("8888 888");
    ps1.close();
PrintWriter pw1 = new PrintWriter(new
FileWriter("b.txt"),  true);
     pw1.println("777 7777");
     pw1.println("666 66 666");
     pw1.close();
 PrintWriter pw2 = new  PrintWriter(new
FileOutputStream("c.txt"), true);
   pw2.println("555555");
   pw2.println("4444444");
  pw2.close();
 }}
```

Comparing characters :

```
import   java.io.*;
class  PBRead {
public static void main(String args[]) throws
Exception {
  String z ="if  ( x == 4 ) x = 4t";
  char cr[]=z.toCharArray();
 CharArrayReader  ca = new
CharArrayReader(cr);
 PushbackReader pb = new
PushbackReader(ca);
  int d;
  while( (d=pb.read()) != -1) {
     switch(d) {
         case '=' :
             if(  (d=pb.read()) ==  '=' )
                 System.out.print(".eq.");
             else {
                 System.out.print("->");
                 pb.unread( d );
             }
             break;
   default :
       System.out.print((char)d);
    }}
}};
```

PushbackReader class:- This class is used to be read a character from any subclasses of Reader class. This class has a method called unread(), which will move the pointer of PushbackReader class back to one character, as if this class has not read a character.

RandomAccessFile class:- This class is used to perform both reading & writing operation of predefined data type value in binary format on a file.

RandomAccessFile(String filename, String mode):- The constructor of this class will accept a file name as the 1st parameter, and the mode of IO operation in the 2nd parameter. The mode may be r-read only, w-write only, s-for appending.

The seek(int n) of this class is used to move the pointer of RandomAccessFile to the n+1 byte, where n is the integer value given in the parameter of this method.

Counting Number of Word, Character, Line by StreamTokenizer :

```
import  java.io.*;
class TokenDemo{
    static int words = 0;
    static int lines = 0;
    static int chars = 0;
    public static void  main(String args[])throws
Exception{
        FileReader fr = new FileReader("a.txt");
        StreamTokenizer st=new
StreamTokenizer(fr);
        st.resetSyntax();
        st.wordChars(33,255);
        st.whitespaceChars(0,' ');
        st.eolIsSignificant(true);
        while(st.nextToken() !=st.TT_EOF){
              switch(st.ttype) {
                case StreamTokenizer.TT_EOF:
                     lines++;
                     break;
                case StreamTokenizer.TT_EOL:
                     lines++;
                     chars++;
                     break;
                case StreamTokenizer.TT_WORD:
                     words++;
                default:
                     chars+=st.sval.length();
                     break;
        }
    }
System.out.println("total lines "+lines+",  words
"+words+", chars "+chars);
}}
```

Random Access File :

```
import   java.io.*;
import   java.util.Scanner;
class  RafDemo {
public static void main(String args[]) throws
Exception {
Scanner s=new Scanner(System.in);
RandomAccessFile raf= new
RandomAccessFile("dd.dat", "rw");
System.out.println("how many rec u want to enter");
int x=s.nextInt();
for( int  i=1; i<=x; i++){
     System.out.println("enter name of "+i);
     String  n = s.next();
     System.out.println("enter roll of "+i);
     int  r = s.nextInt();
     System.out.println("enter mark of "+i);
     float m = s.nextFloat();
     raf.writeUTF(n);
     raf.writeInt(r);
     raf.writeFloat(m);
  }
raf.seek(0);
long  size= raf.length();
int  len =  (int)(size/x); /*len is the length of a
record*/
//reading dat from file
for( int  i=0; i<5; i++){
    System.out.println("enter ur line no.");
    int  y = s.nextInt();
    raf.seek((y-1)*len);
    System.out.println(raf.readUTF()+ " : "
+raf.readInt() + " : " + raf.readFloat());
 }
raf.close();
}};
```

StreamTokenizer class :- This class is used to parse the data to find out the no of word, character, line available in the data, generally it will read the data as a Token (word) from the input device. The I/P device may be any subclass of Reader or InputStream.

StreamTokenizer (Reader ob) / StreamTokenizer (InputStream ob) :- This class keeps information about character, new line, in a syntax table to analysis different tokens.

resetSyntax( )- Will initialize the syntax table with default value. The syntax table is like an index table that stores ascii value of indivisual delimeters like space, new line, eof, etc.

wordChars ( ) – It will accept an integer range, so that any character consisting ASCII value within this range is considered as a character.

whiteSpaceChars ( )- It will accept a range so that any character within this range is considered as WhiteSpaceCharas.

eolIsSignificant ( ) – If we provide true this method, then the StreamTokenizer class will Consider the new line character as a new line, otherwise it will consider the new line character as a simple character.

nextToken( )- When the StreamTokenizer class will read a word or token then this method will return a non-zero integer value but when there will be no more token available in the data then it will return-1 as the value of EOF.

ttype – This datamember of StreamTokenizer class returns the type of token read by this class i.e. whether the token is a space or new line or word etc.

sval – This datamember will hold the actual Token if the token is String type then the token is available in sval. If the token is a number then it is available in nval data member of this class.

Java Scanner to get input from console

```
import java.util.Scanner;
class ScanTest{
 public static void main(String args[]){
   Scanner sc=new Scanner(System.in);
   System.out.println("Enter your rollno");
   int rollno=sc.nextInt();
   System.out.println("Enter your name");
   String name=sc.next();
   System.out.println("Enter your fee");
   double fee=sc.nextDouble();
   System.out.println("Rollno:"+rollno+"
name:"+name+" fee:"+fee);
   sc.close();
}}
```

Java Scanner Example with delimiter

Let's see the example of Scanner class with delimiter. The \s represents whitespace.

```
import java.util.*;
public class ScanTest2{
public static void main(String args[]){
    String input = "10 tea 20 coffee 30 tea buiscuits";
    Scanner s = new Scanner(input).useDelimiter("\\s");
    System.out.println(s.nextInt());
    System.out.println(s.next());
    System.out.println(s.nextInt());
    System.out.println(s.next());
    s.close();
}}
```

PipedInputStream and PipedOutputStream classes

The PipedInputStream and PipedOutputStream classes can be used to read and write data simultaneously. Both streams are connected with each other using the connect() method of the PipedOutputStream class.

Here, we have created two threads t1 and t2. The t1 thread writes the data using the PipedOutputStream object and the t2 thread reads the data from that pipe using the PipedInputStream object. Both the piped stream object are connected with each other.

```
import java.io.*;
class PipedWR{
public static void main(String args[])throws Exception{
final PipedOutputStream pout=new PipedOutputStream();
final PipedInputStream pin=new PipedInputStream();

pout.connect(pin);//connecting the streams

//creating one thread t1 which writes the data
Thread t1=new Thread(){
   public void run(){
      for(int i=65;i<=90;i++){
         try{
            pout.write(i);
            Thread.sleep(1000);
         }catch(Exception e){}
```

```
    }
}};

//creating another thread t2 which reads the data
Thread t2=new Thread(){
   public void run(){
      try{
         for(int i=65;i<=90;i++)
               System.out.println(pin.read());
      }catch(Exception e){}
   }
};

//starting both threads
t1.start();
t2.start();
}}
```