FILE HANDLING

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 follow 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 continuous flow of data from one place to another place.

Java.io is a package that contains number of classes. By using that classes, we are able to send the data from one place to another place.

```
java
|------→io
        |--→FileInputStream(class)
        |--→FileOutputStream(class)
        |--→FileReader(class)
        |--→FileWriter(class)
        |--→Serializable(interface)
```

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 stream or channel we are able to send particular data from one place to the another place.
Streams are two types:-
1. Byte oriented stream.(supports byte formatted data to transfer)
2. Character oriented stream.(supports character formatted data to transfer)

Byte oriented streams:-

Java.io.FileInputStream

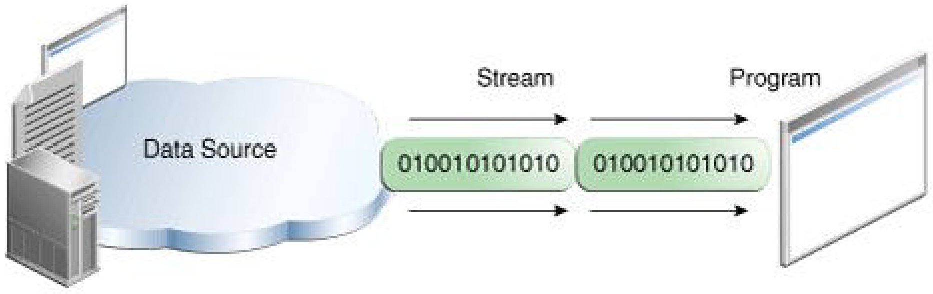
byte channel:-

1)**FileInputStream**

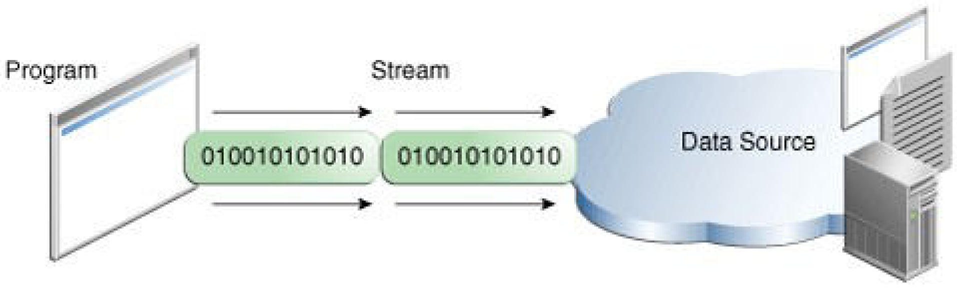
```
public native int read() throws java.io.IOException;
public void close() throws java.io.IOException;
```

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 use read() method.



Java.io.FileOutputStream:-
To write the data to the destination file, we have to use the FileOutputStream class.
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
        FileInputStream fis = new FileInputStream("abc.txt");//read data from source file
        FileOutputStream fos = 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");
        fis.close();                                  //stream closing operations
        fos.close();
        }
}
```

File:

File f=new File("GameFile.txt");

➔This line first checks whether GameFile.txt file is already available (or) not.

➔If it is already available then "f" simply refers that file.

➔If it is not already available then it won't create any physical file, just creates a java File object represents name of the file.

Example:

```
import java.io.*;
class  FileDemo
{
        public static void main(String[] args) throws IOException
        {
                File f=new File("cricket.txt");                //no file yet
                System.out.println(f.exists());                //false
                f.createNewFile();                  //make a file,  "cricket.txt" which is assigned to f
                System.out.println(f.exists());     //true
        }
}
output :
1st run :
false
true

2nd run :
true
true
```

A java File object can represent a directory also.

Example:

```
import java.io.*;
class  FileDemo
{
        public static void main(String[ ] args) throws IOException
        {
                File f=new File("cricket123");                 //no directory yet
                System.out.println(f.exists());                //false
                f.mkdir();                                     //create an actual directory
                System.out.println(f.exists());                //true
        }
}
```

Note: in UNIX everything is a file, java "file IO" is based on UNIX operating system hence in java also we can represent both files and directories by File object only.

File class constructors:

1. File f=new File(String name);
   Creates a java File object to represent name of the file or directory in current working directory.
2. File f=new File(String subdirname , String name);
   Creates a java File object to represent name of the file or directory present in specified sub directory.(some other location)

3. File f=new File(File subdir, String name);

Requirement: Write code to create a file named with demo.txt in current working directory.

Program:
```
import java.io.*;
class  FileDemo
{
		public static void main(String[] args)throws IOException
		{
				File f=new File("demo.txt");
				f.createNewFile();
		}
}
```

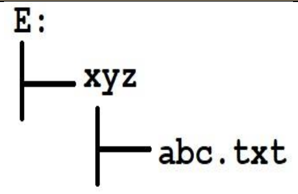
Requirement: Write code to create a directory named with Ramu123 in current working directory and create a file named with abc.txt in that directory.

Program:
```
import java.io.*;
class  FileDemo
{
		public static void main(String[] args) throws IOException
		{
				File f1=new File("Ramu123"); //create an object
				f1.mkdir( );                 //create an actual directory
				File f2=new File("Ramu123","abc.txt");
				f2.createNewFile( );         //make a file, “Ramu123” which is assigned to f2
		}
}
```

Requirement: Write code to create a file named with abc.txt present in E:\xyz folder.

Program:
```
import java.io.*;
class  FileDemo
{
		public static void main(String[] args) throws IOException
		{
				File f=new File("E:\\xyz","abc.txt");
				f.createNewFile();
		}

}
```



Import methods of File class:

Methods	Description
boolean exists( );	Returns true if the specified file or directory available.
boolean createNewFile( );	First This method will check whether the specified file is already available or not. If it is already available then this method simply returns false without creating any physical file. If this file is not already available then it will create a new file and returns true
boolean mkdir( );	First This method will check whether the directory is already available or not. If it is already available then this method simply returns false without creating any directory. If this directory is not already available then it will create a new directory and returns true
boolean isFile( );	Returns true if the specified File object represents a physical file.
String[ ] list( );	It returns the names of all files and subdirectories present in the specified directory.
long length();	Returns the no of characters present in the file.
boolean isDirectory( );	Returns true if the File object represents a directory.
boolean delete( );	To delete a file or directory.

Requirement: Write a program to display the names of all files and directories present in c:\\ramu_classes.

```
import java.io.File;
import java.io.IOException;

public class  FileDemo
{
		public static void main(String[] args)throws IOException
		{
				int count=0;
				File f=new File("c:\\ramu_classes");
				String[ ] s=f.list( );			//list all files and directories in this directory

				for(String s1:s)			//for each string s1 in s
				{
						count++;
						System.out.println(s1);
				}

				System.out.println("total number : "+count);
		}
}
```

Requirement: Write a program to display only file names

```
import java.io.*;
class  FileDemo
{
		public static void main(String[] args) throws IOException
		{
				int count=0;
				File f=new File("c:\\ramu_classes");
				String[ ] s=f.list( );

				for(String s1:s)
				{
						File f1=new File(f,s1);
						if(f1.isFile( ))
						{
								count++;
								System.out.println(s1);
						}
				}
				System.out.println("total number : "+count);

		}
}
```

Requirement: Write a program to display only directory names

```
import java.io.*;
class  FileDemo
{
		public static void main(String[ ] args) throws IOException
		{
				int count=0;
				File f=new File("c:\\ramu_classes");
				String[ ] s=f.list( );

				for(String s1:s)
				{
						File f1=new File(f,s1);
						if(f1.isDirectory())
						{
								count++;
								System.out.println(s1);
						}
				}
				System.out.println("total number : "+count);

		}
}
```

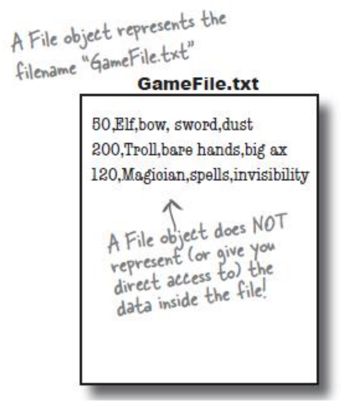
Important points:

➔**A file object represents the name and path of a file or directory on disk.**

➔**For example: /Users/Krishna/Data/GameFile.txt**

➔**But it does NOT represent, or give us access to, the data in the file!**

Molly
Sherry Davis
2360 Fox Drive
Port Hueneme, CA
An address is NOT the same as the actual house! A File object is like a street address... it ~~represents~~ the name and location of a particular file, but it isn't the file itself.


A File object represents the filename "GameFile.txt"
GameFile.txt
A File object does NOT represent (or give you direct access to) the data inside the file!

```
50,Elf,bow, sword,dust
200,Troll,bare hands,big ax
120,Magician,spells,invisibility
```

FileReader:

We can use FileReader object to read character data from the file.

Constructors:

1. **FileReader fr=new FileReader(String file);**
2. **FileReader fr=new FileReader (File f);**

Methods:

1) int read( );
It attempts to read next character from the file and return its ASCII value. If the next character is not available, then we will get -1.

```
int i=fr.read( );
System.out.println((char)i);
```

As this method returns ASCII value, while printing we have to perform type casting.

2) int read(char[ ] ch);
It attempts to read enough characters from the file into char[ ] array and returns the no of characters copied from the file into char[] array.

```
File f=new File("abc.txt");
Char[ ] ch=new Char[(int)f.length( )];
```

3) void close( );

It is used to close the FileReader class.

```
Approach 1:
import java.io.*;
class FileReaderDemo
{
		public static void main(String[] args)throws IOException
		{
				FileReader fr=new FileReader("cricket.txt");				//just an object
				int i=fr.read( );		//more amount of data
				while(i != -1)
				{
						System.out.print((char)i);		//typecasting mandatory
						i=fr.read();
				}
		}
}
Output:
```

```
Charan
Software solutions
ABC

Approach 2:
import java.io.*;
class FileReaderDemo
{
		public static void main(String[] args) throws IOException
		{
			File f=new File("cricket.txt");			//just an object
			FileReader fr=new FileReader(f);		//create a FileReader object
			char[ ] ch=new char[(int)f.length( )];	//small amount of data
			fr.read(ch);				//read the whole file!
			for(char ch1:ch)				//print the array
			{
				System.out.print(ch1);
			}
		}
}
Output:
XYZ
Software solutions.
```

<u>Usage of *FileWriter* and *FileReader* is not recommended because :</u>

1. While writing data by FileWriter compulsory we should insert line separator(\n) manually which is a bigger headache to the programmer.
2. While reading data by FileReader we have to read character by character instead of line by line which is not convenient to the programmer.
3. To overcome these limitations we should go for BufferedWriter and BufferedReader concepts.

FileWriter:

By using FileWriter object, we can write character data to the file.

Constructors:

FileWriter fw=new FileWriter(String fname);

➔ It creates a new file. It gets file name in string.

FileWriter fw=new FileWriter(File f);

➔ It creates a new file. It gets file name in File object.

The above 2 constructors meant for overriding.

Instead of overriding, if we want append operation then we should go for the following 2 constructors.

FileWriter fw=new FileWriter(String file, boolean append);
FileWriter fw=new FileWriter(File file, boolean append);

If the specified physical file is not already available, then these constructors will create that file.

Methods:

Method	Description
write(int ch);	To write a single character to the file.
write(char[ ] ch);	To write an array of characters to the file.
write(String text);	To write a String to the file.
flush( );	To give the guarantee the total data include last character also written to the file.
close( );	To close the stream.

Example:
```
import java.io.*;
class FileWriterDemo
{
		public static void main(String[ ] args) throws IOException
		{
				FileWriter fw=new FileWriter("cricket.txt",true); //create a FileWriter object
```

```
                fw.write(99);                   //adding a single character
                fw.write("charan\nsoftware solutions");   //write characters to the file
                fw.write("\n");
                char[ ] ch={'a','b','c'};
                fw.write(ch);
                fw.write("\n");
                fw.flush( );                    //flush before closing
                fw.close( );                    //close file when done always.
        }
}
Output:
charan
software solutions
abc
```

Note :

- The main problem with FileWriter is "we have to insert line separator manually", which is difficult to the programmer. ('\n')
- And even line separator varying from system to system.

BufferedWriter:

By using BufferedWriter object we can write character data to the file.

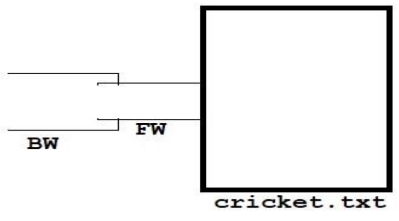
Constructors:

BufferedWriter bw=new BufferedWriter(writer w);
BufferedWriter bw=new BufferedWriter(writer w, int buffersize);

Note: BufferedWriter can't communicate directly with the file it can communicate via some writer object.

Which of the following declarations are valid?

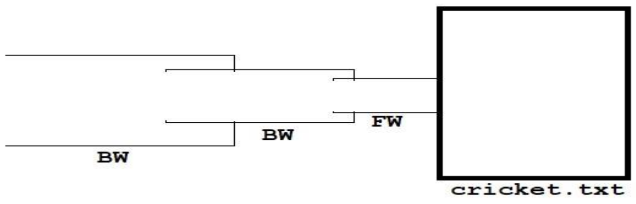
1. BufferedWriter bw=new BufferedWriter("cricket.txt"); (invalid)
2. BufferedWriter bw=new BufferedWriter (new File("cricket.txt")); (invalid)
3. BufferedWriter bw=new BufferedWriter (new FileWriter("cricket.txt")); (valid)



BUFFERING

4)BufferedWriter bw=new BufferedWriter(new BufferedWriter(new FileWriter("cricket.txt"))); (valid)

→ Two level buffering



TWO LEVEL BUFFERING

Methods:

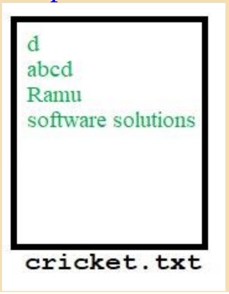
1. write(int ch);	//To write a single character to the file
2. write(char[ ] ch);	//To write n array of characters
3. write(String s);	//To write string to the file
4. flush( );	
5. close( );	
6. newline( );	//To insert a line seperator

When compared with FileWriter which of the following capability is available extra as method form in BufferedWriter.

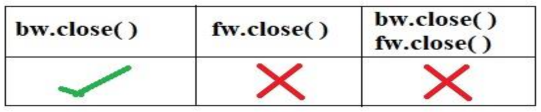
1. Writing data to the file.
2. Closing the file.
3. Flushing the file.
4. Inserting new line character.

Ans : 4

Example:
```
import java.io.*;
class BufferedWriterDemo
{
		public static void main(String[] args)throws IOException
		{
				FileWriter fw=new FileWriter("cricket.txt");
				BufferedWriter bw=new BufferedWriter(fw);
				bw.write(100);
				bw.newLine();
				char[ ] ch={'a','b','c','d'};
				bw.write(ch);
				bw.newLine();
				bw.write("Ramu");
				bw.newLine();
				bw.write("software solutions");
				bw.flush();
				bw.close();
				}
}
```
Output:


```
d
abcd
Ramu
software solutions
```
cricket.txt

Note : Whenever we are closing BufferedWriter, automatically internal Filewriter will be closed and we are not required to close explicitly.



bw.close( )	fw.close( )	bw.close( ) fw.close( )
✓	✗	✗

BufferedReader:

We can use BufferedReader to read character data from the file.
The main advantage of BufferedReader when compared with FileReader is we can read data line by line in addition to character by character.

Constructors:

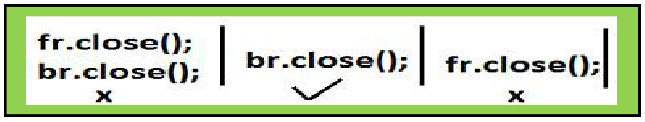
BufferedReader br=new BufferedReader(Reader r);
BufferedReader br=new BufferedReader(Reader r, int buffersize);

Note: BufferedReader can't communicate directly with the File and it can communicate via some Reader object.

Methods:

1. int read(); //to read a single character
2. int read(char[] ch); //to read n array of characters
3. void close();
4. **String readLine();**
   It attempts to read next line and return it , from the File. if the next line is not available then this method returns null.

Example:
```
import java.io.*;
class BufferedReaderDemo
{
        public static void main(String[] args) throws IOException
        {
                FileReader fr=new FileReader("cricket.txt");
                BufferedReader br=new BufferedReader(fr);
                String line=br.readLine( );
                while(line!=null)
                {
                        System.out.println(line);
                        line=br.readLine( );
                }
                br.close( );
        }
}
```
Note: Whenever we are closing BufferedReader, automatically underlying FileReader will be closed, it is not required to close explicitly. Even this rule is applicable for BufferedWriter also.



```
fr.close();  | br.close(); | fr.close();
br.close();  |             |
   x         |     ✓       |     x
```

The most enhanced Reader to read character data from the file is BufferedReader.

PrintWriter:

- This is the most enhanced Writer to write character data to the file.
- The main advantage of PrintWriter over FileWriter and BufferedWriter is
  ➔By using FileWriter and BufferedWriter we can write only character data to the File but
  ➔By using PrintWriter we can write any type of data to the File.

Constructors:

1) PrintWriter pw=new PrintWriter(String fname);
2) PrintWriter pw=new PrintWriter(File f);
3) PrintWriter pw=new PrintWriter(Writer w);

Note: PrintWriter can communicate directly with the File and can communicate via some Writer object also.

Methods:

1. write(int ch);
2. write (char[] ch);
3. write(String s);

4. flush();
5. close();

6. print(char ch);
7. print (int i);
8. print (double d);
9. print (boolean b);
10. print (String s);

11. println(char ch);
12. println (int i);
13. println(double d);
14. println(boolean b);
15. println(String s);

Example:

```
import java.io.*;
class PrintWriterDemo {
  public static void main(String[] args)throws IOException
              {
                           FileWriter fw=new FileWriter("cricket.txt");
                           PrintWriter out=new PrintWriter(fw);
```

```
                    out.write(100);
                    out.println(100);
                    out.println(true);
                    out.println('c');
                    out.println("SaiCharan");
                    out.flush();
                    out.close();
          }
}
Output:
d100
true
c
SaiCharan
```

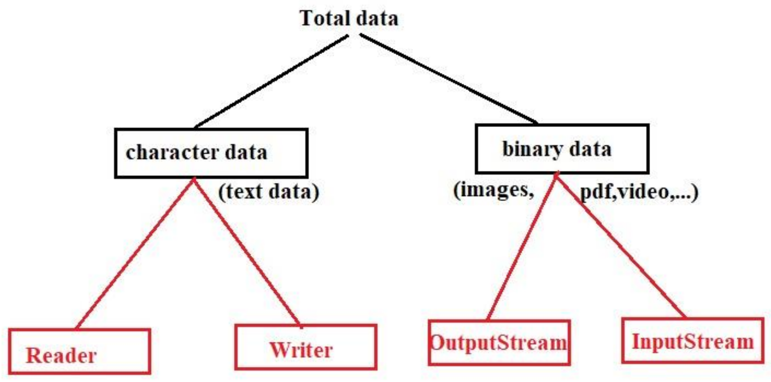
What is the difference between write(100) and print(100)?
In the case of write(100) the corresponding character **"d"** will be added to the File but in the case of print(100) **"100" value** will be added directly to the File.

Note 1:

1. The most enhanced Reader to read character data from the File is **BufferedReader**.
2. The most enhanced Writer to write character data to the File is **PrintWriter.**

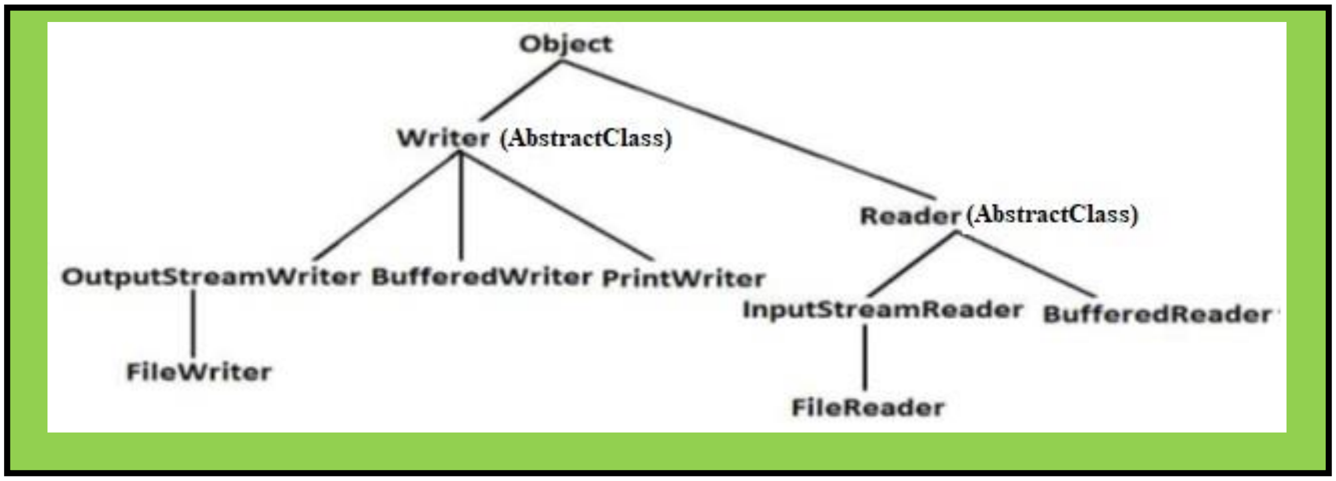
Note 2:

1. we can use **Readers and Writers** to handle **character data**.
   we can use **InputStreams and OutputStreams** to handle **binary data** (like images, audio files, video files etc).



2. We can use OutputStream to **write binary data** to the File and
   we can use InputStream to **read binary data** from the File.

<u>Diagram</u>:


Object
Writer (AbstractClass)
Reader (AbstractClass)
OutputStreamWriter
BufferedWriter
PrintWriter
InputStreamReader
BufferedReader
FileWriter
FileReader