Input/output using Streams in java

From today we will start to learn java.io package where stream classes are defined in order to take input and give output to Files. (Secondary Storage).

Being a java programmer, all students must have the basic concept of File input/output.

The java.io package contents are:-

1. File
2. FileWriter
3. FileReader
4. BufferedWriter
5. BufferedReader
6. PrintWriter

In 1995 — Java 1.0 version came
File I/O — Java 1.2 version came
File I/O — Java 1.5 " "
File I/O

At that time C was popular Database was not popular

If we required to store less amount of data then we should recommended to use File I/O: If we require large amount of data then we must have an option of Database Management Systems link using JDBC.(Java Database Connectivity)

① File

```
File f = new File("data.txt");
```

The above statement actually does not create any physical file - "data.txt".
We are just creating java File object to represent "data.txt" file name.
We can check this by the following statement:-

```
System.out.println(f.exists()); //false
```

To create a physical file:-

```
f.createNewFile();
```

To check we can give statement:-

```
System.out.println(f.exists()); //true.
```

Example:- TestFile1.java

```
import java.io.*;
class TestFile1 {
public static void main(String args[]) throws IOException
{
    File f = new File("data.txt");
    System.out.println(f.exists()); //false
    f.createNewFile();
    System.out.println(f.exists()); //true.
 }
}
```

If you run the above program 1st time
it will show:- false
true.

But 2nd time if you run then it will give
output :- true
true

Because when you run it 2nd time,
file "data.txt" is present previously hence
1st time - 'true'
f.createNewFile(); is not going to
create a new file because file data.txt
is already present after running 1st time.

Java File I/O is based on Unix
operating system. In Unix/Linux, everything
is File (even directories also).
Hence File object is used to represent
directory also.

```
File f = new File("ssc123");
System.out.println(f.exists()); // false
```

To Create a directory —

```
f.mkdir(); //make directory
System.out.println(f.exists()); // true
```

Example :- import java.io.*; $\boxed{\text{TestDir.java}}$

```
class TestDir
{
public static void main(String args[]) throws Exception
{
        File f = new File("ssc123");
      System.out.println(f.exists());
      f.mkdir();
  System.out.println(f.exists());
  }
}
```

When we execute the code

first time :- $\boxed{\begin{matrix}\text{false}\\ \text{true}\end{matrix}}$

after that :- always — $\boxed{\begin{matrix}\text{true}\\ \text{true}\end{matrix}}$

Because at first time directory with name ssc123 gets created.

$$\begin{tikzcd}
\text{boolean f.createNewFile();} \arrow[r] \arrow[dr] & \text{true - if file not present then created} \\
 & \text{false - if file already available does not creates.}
\end{tikzcd}$$

$$\begin{tikzcd}
\text{boolean f.mkdir();} \arrow[r] \arrow[dr] & \text{true} & \text{if directory is not available then created} \arrow[l] \\
 & \text{false} & \text{if directory is present then not created.} \arrow[l]
\end{tikzcd}$$

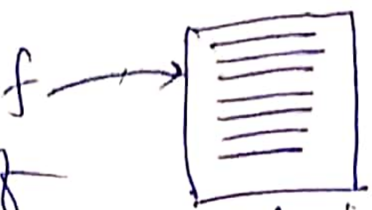
$$\begin{tikzcd}
 & \boxed{\text{File}} \\
f \arrow[ur] \arrow[dr] & \\
 & \text{Directory}
\end{tikzcd}$$

$$\begin{tikzcd}
\text{f.isFile()} \arrow[r] \arrow[dr] & \text{true} \\
 & \text{false}
\end{tikzcd}$$

$$\begin{tikzcd}
\text{f.isDirectory()} \arrow[r] \arrow[dr] & \text{true} \\
 & \text{false}
\end{tikzcd}$$

String [] s = f.list(); → list of all files and directories present in a directory.

Let us assume, f is pointing to a file



f.length() is the method which gets no. of characters in the file, return type is long.

long l = f.length();

Delete the file and directory:-

boolean f.delete();

Example:- To display the names of all files & directories under any specified directory - D:\Java

```
import java.io.*;
class TestFile2
{ public static void main(String args[]) throws Exception
  {
    int count=0;
    File f =new File("D:\\Java");
    String[]s = f.list();
    for (String str : s)
    {
      count++;
      System.out.println(str);
    }
   System.out.println("The total number:" +count);
  }
}
```

O/P :- javac TestFile2.java ↵
java TestFile2 ↵

List of all files & directories given from D:\Java

Now, we want ~~to~~ display only File names :—

```
int count=0;
String [] s = f.list();
 for (String str : s)
 {        File f1 = new File(f, str);
            if (f1.isFile())
            {
              count++;
            System.out.println(str);
            }
        System.out.println("Total no. of files:"+ count);
  }
```

Now, we want to display only directory Names :—

```
  int count=0;
  String s[] = f.list();
   for(String str : s)
    {       File f1 = new File(f, str);

            if (f1.isDirectory())
            {
              count++;
            System.out.println(str);
            }
  System.out.println("Total no. of directories:"+ count);
  }
```

In next lecture we will learn FileWriter

FileWriter

Constructors :-

① FileWriter fw = new FileWriter(String fname); } overwrite

② FileWriter fw = new FileWriter(File f); }

③ FileWriter fw = new FileWriter(String fname, boolean append); } append
true

④ FileWriter fw = new FileWriter(File f, boolean append);

Methods of FileWriter →

① write (int ch);
To write a single character to the file.

② write(char [] ch); — To write an array to the file.

③ write (String s); — To write String to the file.

④ ~~write~~ flush(); - It is recommended to call flush() after ~~every~~ all read and write operation to finished. It give the guarantee that total data including last character written properly.

⑤ close() :- finally close () has to be executed to close all objects (pointers).

Let us convert the above all functions in a single Program :- FileWriterTest.java

```
import java.io.FileWriter;
import java.io.IOException;
class FileWriterTest{
public static void main(String args[]) throws IOException
{
        FileWriter fw = new FileWriter("abc.txt");
        fw.write(100); //adding a single character P
        fw.write("rogramming \n in \n Java");
        fw.write("\n");
        char [] ch = {'a', 'b', 'c'};
        fw.write(ch);
        fw.write("\n");
        fw.flush();
        fw.close();
    }
}
```

→ In the above program FileWriter always over-writes existing data.

→ Instead of over-writing if we want to append we have to create object as:—

FileWriter fw = new FileWriter("abc.txt", true);

Note:- The problem with FileWriter is we have to insert line separator (\n) manually, which is varied from System to System. It is sometimes difficult for programmers.

FileReader We can use FileReader to read character data from the File.

Constructors:—

① FileReader fr = new FileReader(String fname);

② FileReader fr = new FileReader (File f);

Method:—

① int read(); — attempts to read next character from the file and returns its Unicode value.

→ If there is no next character then it will ~~print~~ get -1.

→ As this method - read() returns unicode ~~cha~~ value of type int hence at the time of printing it should be typecasted to character as:—

int read() → unicode value of character

```
System.out.println((char) i);
```

```
FileReader fr = new FileReader("abc.txt");
int i = fr.read();
while (i != -1) {
  System.out.println((char) i);
    i = fr.read();
  }
```

Methods continued :— FileReader :-

② int read(char [] ch);

③ void close();

Example:- FileReaderTest1.java

```
   import java.io.*;
  class FileReaderTest1
  {
  public static void main(String [] args) throws
                                   IOException
  {     FileReader fr = new FileReader("abc.txt");
        int i = fr.read();
   System.out.println("Contents of file abc.txt");
   while(i != -1)
    {    System.out.print((char) i);
           i = fr.read();
  } }    } fr.close();
```

Now, we will try to use 2nd-read(char [] ch) method.
In this method return type is int

int read(char [] ch);
↓
No. of characters copied from file into array

Note:- long size array is not allowed in java hence f.length() returning long should be typecasted into int.

f.length() returns (long) size of the file object f, which contains any file.

Let us use both approaches into a single program—

```
import java.io.*;
class FileReaderTest2
{ public static void main(String []args) throws
                                        IOException
  {
File f = new File("abc.txt");
FileReader fr = new FileReader(f);
char [] ch = new char[(int) f.length()];
fr.read(ch);
for(char ch1 : ch)
  {
   System.out.print(ch1);
  }
System.out.println("****************");
FileReader fr1 = new FileReader("abc.txt");
int i = fr1.read();
while ( i != -1)
{
System.out.print((char) i);
i = fr1.read();
}
} //close of main function
} //close class FileReaderTest2
```

→ In above two approaches to read contents from file — 2nd approache is highly recommended.

→ In first approach — if the contents of file is bigger than the int range, then in case of typecasting, there will be loss of data found. Hence first approach is not recommended, 2nd approach is suitable.

Reading and comparing data line-by-line can be possible by — BufferedReader class.

[FileReader — read data character by character
BufferedReader — read data line by line.]

The main problem with FileReader is that one can read character by character by using read() method. Hence BufferedReader is used to read contents line by line — which is useful for programmers.

Problems with FileReader and FileWriter —

① \n character has different results on different systems.

② character by character reading or writing is not recommended to use for programmers

Hence BufferedReader and BufferedWriter is recommended to use to solve this problem.

BufferedWriter — To write character data to the File.

BufferedWriter bw = new BufferedWriter

Constructors:-

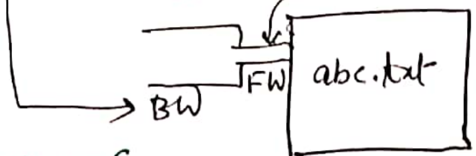
1. BufferedWriter bw = new BufferedWriter(~~[illegible]~~ Writer w);

BufferedWriter can only communicate with Writer object but not directly with the filename.

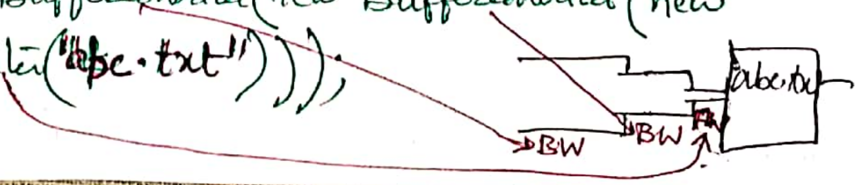
2. BufferedWriter bw = new BufferedWriter(Writer w, int buffersize);

Eg:- Both valid Statements:-

BufferedWriter bw = new BufferedWriter(new FileWriter("abc.txt"));



BufferedWriter bw = new BufferedWriter(new BufferedWriter(new FileWriter("abc.txt")));



abc.txt
FW
BW
BW