Day 1: Java IO-streams (Byte Oriented, Character Oriented classes) and File class

File class in Java:

The **File** class belongs to **java.io** package.

This class is used to know the characteristics of something present in the hard-disk.

This **java.io.File** class has many functionalities using which we can identify the features of file or folder/directory present in the given location.

Example:

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

The above line will not create a new physical file.

If the file "abc.txt" is already exist in the current folder then f will point the already existing file. if the file is not present the f will represent simply name of the file.

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

    System.out.println(f); //abc.txt

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

    f.createNewFile(); // it will create a abc.txt file in the current location

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

    File f2=new File("d://myfiles//abc.txt");

    f2.createNewFile();// if d://myfiles location is not there
                       //then it will throw an exception
```

Note:- if the file is already there, it instead of creating a new file, it will points to the existing file only.

A file object can be used for a folder(directory) also.

Example:

```
    File f=new File("d://myfiles2");

    f.mkdir(); //to create folder

    System.out.println(f);//  d:\myfiles2

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

File class constructors:

```
File f = new File(String fname); //we can provide relative of absolute path.

File f=new File(String subdir,String fname);

File f =new File(File subdir,String fname);
```

Some of the methods of the File class:

Example: creating a new file if it is not present:

```
import java.io.*;
public class Main {
    public static void main(String[] args) {

        try {
            File file = new File("a1.txt");
            if (file.createNewFile()) {
                System.out.println("New File is created!");
            } else {
                System.out.println("File already exists.");
            }
        } catch (IOException e) {
            e.printStackTrace();
```

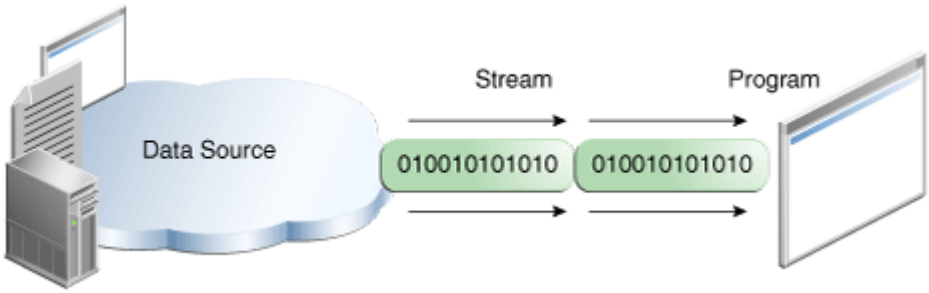
```
        }

    }
}
```

IO-Stream:

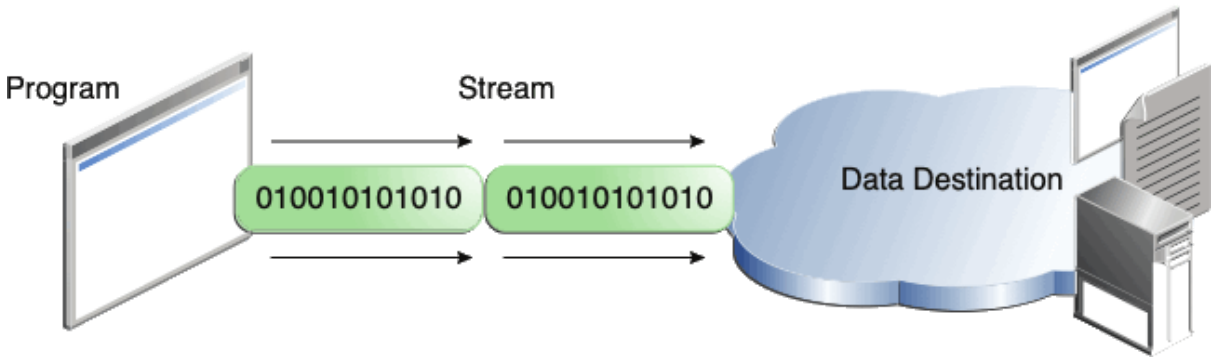
An *I/O Stream* represents an input source or an output destination. A stream can represent many different kinds of sources and destinations, including disk files, devices, other programs, and memory arrays.

It is basically a continuous flow of data from one place to another, here the flow of data is in between the program and the peripherals(devices, other programs, etc.)

Reading data into a program from the peripherals:



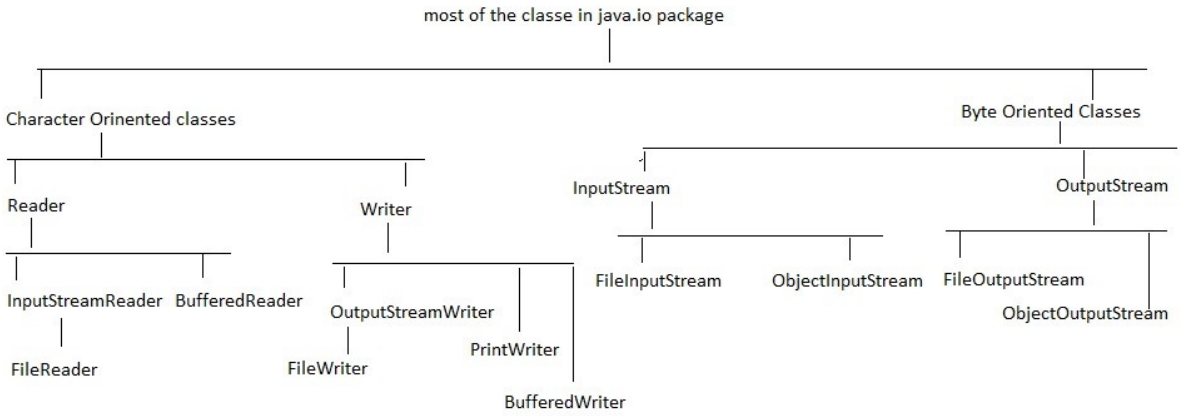
Writing data from the program into the peripherals:



For data transfer to be done, first of all the data should be represented to its low level equivalent
either in the form of **bytes(8 bit)** or in the form of **character(16 bit)**.

Depending upon the representation of the data most of the classes in the **java.io** package is classified into two categories:

1. **Byte oriented classes**
2. **Character oriented classes**



FileWriter class:

Java FileWriter class is used to write character-oriented data to a file. It is character-oriented class which is used for file handling in java.

.

Note:- the main limitation of the FileWriter class is, while writing the data in a file we need to insert the line separator manually("\n") or ("\r\n"), depending upon different Operating System.

Example:

```
import java.io.*;
public class Main {

  public static void main(String[] args) throws IOException{

        FileWriter fw = new FileWriter("d://abc.txt");

        fw.write(100);//d will be added
        fw.write("ramesh\nwelcome");
        fw.write("\n");
        fw.write("india");
        fw.write("\n");
        char ch[]={'a','b','c'};
        fw.write(ch);

        fw.flush();
        fw.close();

       System.out.println("done");

    }
  }


Output:
The above program will create a new file abc.txt inside the D:// drive and write
dramesh
welcome
india
abc
```

FileReader class:

We can use this class to read the data from the file in the form of Character.

Constructor of FileReader class:

- **FileReader(String file);** // It gets filename in string. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.
- **FileReader(File file);** // It gets filename in File instance. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.

Methods of FileReader class:

- **int read();** // it attempts to read the next character from the file and returns its value(unicode). if the next character is not available then it will return -1. (EOF)

  Example:

  abc ⇒ 97 98 99 -1

- **int read(char[] ch);** // it attempts to read the enough character from the file into the character array and returns the number of characters copied.

Example1: reading from the file

```
import java.io.*;
public class Main {

    public static void main(String[] args) throws IOException {

        FileReader fr=new FileReader("d://abc.txt");

        int i=fr.read();

        while( i != -1) {
            System.out.print((char) i);
            i = fr.read();
        }
    }
}
```

Example2: reading from the file (another approach)

```
import java.io.*;
public class Main {

    public static void main(String[] args) throws IOException {
```

```
        File f=new File("d://abc.txt");

        FileReader fr=new FileReader(f);

        char[] chr=new char[(int)f.length()];

        fr.read(chr);

        for(char c:chr){
            System.out.print(c);
        }
    }
}
```

Note: the main limitation of the FileReader is we have to read character by character which is not convenient to the programmer and it is not recommended with respect to performance also.

To overcome the problem of FileWriter and FileReader we should use the **BufferedReader** and the **BufferedWriter** class.

BufferedWriter class:

The BufferedWriter class is used to writer the character data to the file. It makes the performance fast compare to the FileWriter class.

Compare to the FileWriter class this BufferedWriter class provides **newLine()** method support to insert a new line into the file. and this **newLine()** method is independent to different Operating System.

Note: The BufferedWriter class never communicate directly with the file. it communicate with the file via some other Writer object.

Example:

```
BufferedWriter bw = new BufferedWriter(Writer writer);

ex:

BufferedWriter bw = new BufferedWriter("abc.txt"); //error

BufferedWriter bw = new BufferedWriter(new File("abc.txt")); //error

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

Example:

```
import java.io.*;
public class Main {

    public static void main(String[] args) throws IOException {

        FileWriter fw = new FileWriter("d://abc.txt");

        BufferedWriter bw = new BufferedWriter(fw);

        bw.write(100);//d will be added
        bw.newLine(); //inserting a new line
        bw.write("ramesh");
        bw.newLine();
        bw.write("india");
        bw.newLine();
        char ch[]={'a','b','c'};
        bw.write(ch);

        bw.flush();
        bw.close();

        System.out.println("done");

    }
}
```

BufferedReader class:

With the help of BufferedReader class we can read character data from the file.

The main advantage of BufferedReader over the FileReader is we can read the data line by line instead of character by character.

So, internally it gives better performance.

Note:-BufferedReader also can't communicate directly with the file, it will communicate with file via some Reader object.

Example:

```
BufferedReadr br=new BufferedReader(Reader r);

ex:
```

```
BufferedReader br=new BufferedReader(new FileReader("abc.txt"));
```

Methods of BufferedReader class:

int read();

int read(char ch[]);

void close();

String readLine();//it attempts to read the next line from the destination. if the next line is not available then we will return a null value.

Example:

```
import java.io.*;
public class Main {

    public static void main(String[] args) throws IOException {

        FileReader fr=new FileReader("d://abc.txt");

        BufferedReader br=new BufferedReader(fr);
        br.lines.forEach(System.out::println);
        //or
        String line=br.readLine();

        while(line != null){
            System.out.println(line);
            line=br.readLine();
        }
        br.close();
    }
}
```

PrintWriter class:

- It is the child class of Writer class.
- It is the most enhanced writer to write any type of data to the file.
- By using the **FileWriter** and the **BufferedWriter** classes we can write only character data to the file, we can not write the primitive data to the file directly

(like boolean, int, float, etc.) but with the help of the **PrintWriter** class we can write any type of primitive data to the file

Note: PrintWriter class can communicate directly to the file and even via some writer object also.

```
PrintWriter pw=new PrintWriter(String fname);

PrintWriter pw=new PrintWriter(File f);

PrintWriter pw=new PrintWriter(Writer w);
```

Methods of the PrintWriter class:

1. All the methods of Writer class(like write(int c), write(String), etc.)
2. print(int i);

   print(double d);

   print(boolean b);

   print(String s);
3. println(int i);

   println(String s);

   etc.

Note: difference between write(100) and print(100) is write(100) method will write corresponding character 'd' whereas print(100) method will print the int value to the file.

Note: The most enhanced reader is the BufferedReader class and the most enhanced Writer is the PrintWriter class.

```
import java.io.*;
public class Main {

    public static void main(String[] args) throws IOException {
```

```
        //FileWriter fw=new FileWriter("abc.txt",true);
        //PrintWriter out=new PrintWriter(fw);

        //or

        PrintWriter out=new PrintWriter("d://abc.txt");

        out.write(100);//d will be added
        out.println(100);
        out.println(true);
        out.println('c');
        out.println("amit");

        out.flush();
        out.close();
        System.out.println("done..");
    }
}
```

Note: As we can use Reader and Writer class to handle character data, we can use InputStream and OutputStream class to handle binary data like images, audio, video, a class Object state, etc.

Example: copy the image from one location to another location (make sure that img.jpg file is there in the d:// drive)

```
import java.io.*;
public class Main {

    public static void main(String[] args) throws IOException {

        FileInputStream fis=new FileInputStream("d://img.jpg");

        FileOutputStream fos=new FileOutputStream("d://img1.jpg");

        int i=fis.read();

        while(i != -1){
            fos.write(i);
            i=fis.read();
        }

        fos.flush();
        fos.close();
        fis.close();

        System.out.println("success.......");
```

```
    }
}
```

If we use FileReader and FileWriter class for the above application instead of FileInputStream and FileOutputStream then the image will not be copied properly because image file is a byte oriented file not a character oriented file.