File I/O

In Java, we can read data from files and also write data in files.

We do this using **streams**. Java has many input and output streams that are used to read and write data. Same as a continuous flow of water is called water stream, in the same way input and output flow of data is called stream.

Stream

Java provides many input and output stream classes which are used to read and write.

Streams are of two types.

- **Byte Stream**
- **Character Stream**

Let's look at the two streams one by one.

Byte Stream

It is used in the input and output of byte.

We do this with the help of different Byte stream classes. Two most commonly used Byte stream classes are **FileInputStream** and **FileOutputStream**. Some of the Byte stream classes are listed below.

Byte Stream	**Description**
BufferedInputStream	handles buffered input stream
BufferedOutputStream	handles buffered output stream
FileInputStream	used to read from a file
FileOutputStream	used to write to a file
InputStream	Abstract class that describe input stream
OutputStream	Abstract class that describe output stream

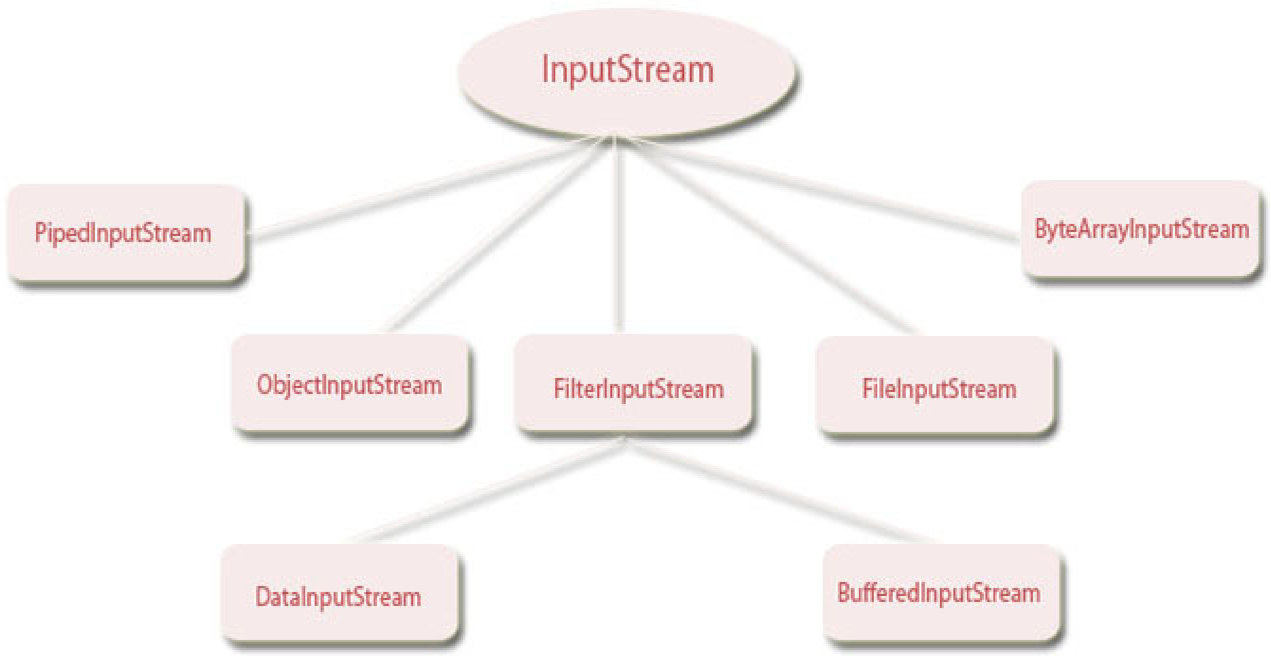
Byte Stream Classes are in divided in two groups -

- **InputStream Classes** - These classes are subclasses of an abstract class, InputStream and they are used to read bytes from a source(file, memory or console).

- **OutputStream Classes** - These classes are subclasses of an abstract class, OutputStream and they are used to write bytes to a destination(file, memory or console).

InputStream

InputStream class is a base class of all the classes that are used to read bytes from a file, memory or console. InputStream is an abstract class and hence we can't create its object but we can use its subclasses for reading bytes from the input stream. We will discuss subclasses of InputStream in the next few articles with examples.



Hierarchy of InputStream classes

Methods of InputStream class.

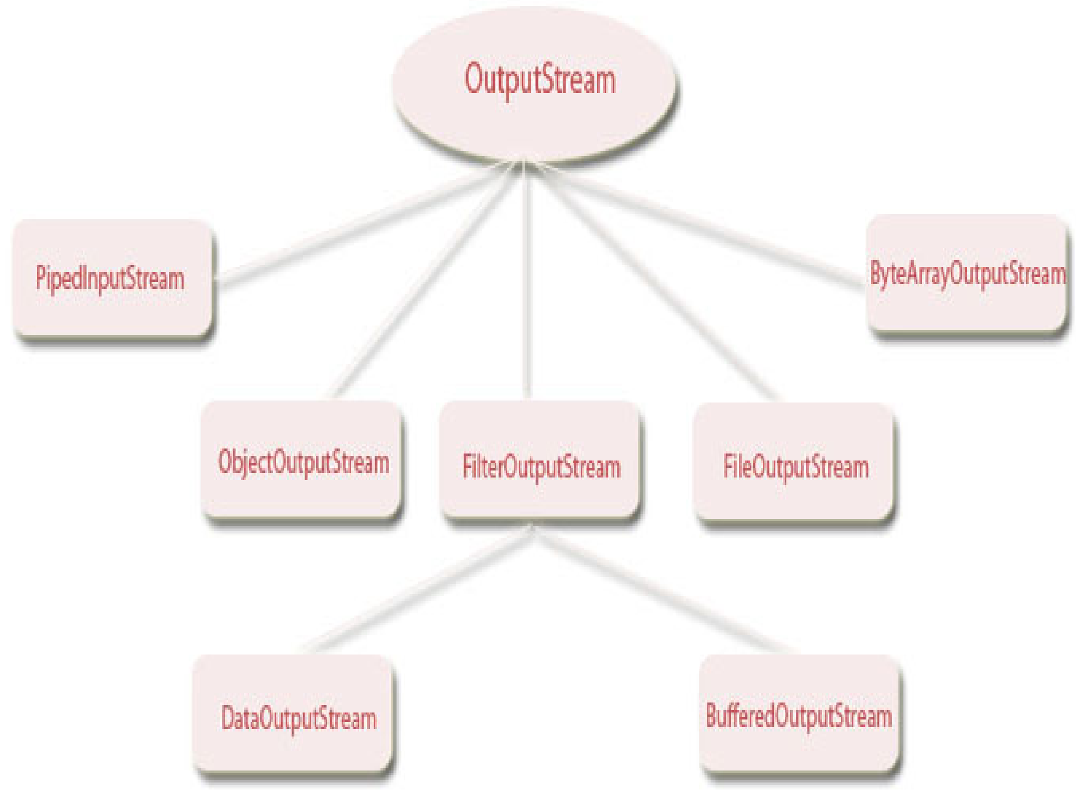
These methods are inherited by InputStream subclasses.

Methods	Description
in available()	This method returns the number of bytes that can be read from the input stream.
abstract int read()	This method reads the next byte out of the input stream.
int read(byte[] b)	This method reads a chunk of bytes from the input stream and store them in its byte array, b.

close()	This method closes this output stream and also frees any resources connected with this output stream.

OutputStream

OutputStream class is a base class of all the classes that are used to write bytes to a file, memory or console. OutputStream is an abstract class and hence we can't create its object but we can use its subclasses for writing bytes to the output stream. In the diagram below we have shown the hierarchy of OutputStream class and some of its important subclasses that are used to write bytes.



Hierarchy of OutputStream classes

Methods of OutputStream class.

Methods of OutputStream class provide support for writing bytes to the output stream. As this is an abstract class. Hence, some undefined abstract methods are defined in the subclasses of OutputStream.

Methods	**Description**

flush()	This method flushes the output steam by forcing out buffered bytes to be written out.
abstract write(int c)	This method writes byte(contained in an int) to the output stream.
write(byte[] b)	This method writes a whole byte array(b) to the output.
close()	This method closes this output stream and also frees any resources connected with this output stream.

Therefore, we need to include java.io package in our program in order to use the stream classes. To include it, we need to write the following code in the beginning of our program.

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

Note that here we wrote * because we want to include all the classes of java.util package.

Taking input from keyboard

To take input from a user, we use **BufferedReader** class by creating an object of it. For that, we have to write the following code.

```
BufferedReader b = new BufferedReader(new InputStreamReader(System.in));
```

Now, let's understand this code word by word.

BufferedReader - This is a class that is used for taking character input.

b - object of BufferedReader class

InputStreamReader - It converts bytes to characters.

System.in - It is input stream. User inputs are read from this.

Thus, we are taking user input from **System.in** which is converted from bytes to characters by the class **InputStreamReader**. This value is stored in the object b of the class **BufferedReader**.

This was a simple explanation of how to input the data entered by the user. Now we will see how to read and write that data.

Reading data

Once we have taken input from the user, we need to read the data. Let's see how to read data.

Reading characters

To read characters, **read()** method is used with the object of the **BufferedReader** class.

Since read function returns an integer value, we need to convert it to character by typecasting it.

Example: Reading characters from the keyboard

```
class Test
{
 public static void main( String args[])
 {
  BufferedReader b = new Bufferedreader(new InputstreamReader(System.in));
  char ch = (char)b.read();
 }
}
```

Reading strings

We use **readLine**() method with the object of the **BufferedReader** class.

```
class Test
{
 public static void main( String args[])
 {
  BufferedReader b = new Bufferedreader(new InputstreamReader(System.in));
  String s = b.readLine();
 }
}
```

Do not forget to include java.util package in the beginning of your program.

That's all we have to do to read data from a user.

Reading and writing in a file

Till now, we have been reading the data entered by a user using the keyboard. Now, we will see how to read and write data in a file.

Writing data in a file

```
class Test
{
 public static void main( String args[])
 {
  FileOutputStream fo=new FileOutputStream("prog.txt");
  String s1="Welcome to Java File handling";
  byte b1[]=s1.getBytes();        //converting string into byte array
  fo.write(b1);
  fo.close();
 }
}
```

Here, **byte b1[ ]=s1.getBytes();** is converting string(character array) into byte array.

Then by writing **fo.write(b1);,** we are writing the data in a file named prog.txt because fo is the object of the **FileOutputStream** class.

Reading data from a file

```
class Test1
{
 public static void main( String args[])
 {
  FileInputStream fi=new FileInputStream("prog.txt");
    int n=0;
    while((n=fi.read())!=-1)
    {
       System.out.println((char)n);
    }
    fin.close();
 }
}
```

In the above code, by writing FileInputStream fi=new FileInputStream("prog.txt");, we are creating an object fi of the class FileInputStream. Thus, the data of the file gets stored in the object fi.

We are assigning n = fi.read() i.e. we are assigning the characters in the value of fi to the integer variable n. Note that we chose here an integer variable because the function read() returns an integer value.

while((n=fi.read())!=-1) - This while loop will continue till all the characters in the value of fi have been read.