Java stream

* A stream is a sequence of data. In Java, a stream is composed of bytes. Stream does not store elements. It simply conveys elements from a source such as a data structure, an array, or an I/O channel, through a pipeline of computational operations. It's called a stream because it is like a stream of water that continues to flow. Java uses the concept of a stream to make I/O operation fast.
* **Java I/O** (Input and Output) is used *to process the input* and *produce the output*
* Java uses the concept of a stream to make I/O operation fast. The **java.io** package contains all the classes required for input and output operations.
* Java I/O classes divided by **input(**file, an array, peripheral device or socket.**)** and **output** being **byte based**(**Byte Stream**) and **Character based (Character Stream)**

**For byte based (byte stream):** Byte based Stream is defined by using two abstract classes at the top of hierarchy, they are **InputStream** and **OutputStream** as shown in below image. These two abstract classes have several concrete classes that handle various devices such as disk files, network connection etc.

* **InPutStream** − The InputStream is used to read data **byte by byte** from a source file
* **OutPutStream** − The OutputStream is used for write data to a **file byte by byte** to a destination.

**For Character based (Character Stream)** Character stream is also defined by using two abstract class at the top of hierarchy, they are **Reader and Writer.** These two abstract classes have several concrete classes that handle unicode character.

* **Reader -** The Reader is used to read data **character by character**
* **Writer -** is used to write data to a file **character by character**

**Byte based classes (byte stream)**

**InputStream Class**. It is the super-class of all classes representing an input stream of bytes.

1111465.png

**OutputStream class**. It is the super-class of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

**java-outputstream.png**

# Character based (Character Stream)

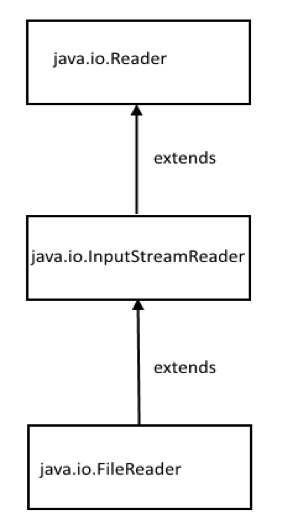
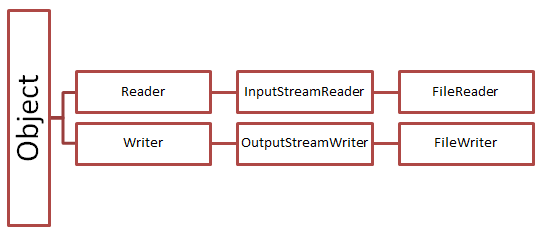
# Reader Class

[Java](https://www.javatpoint.com/java-tutorial) Reader class is for reading character [streams](https://www.javatpoint.com/java-8-stream)

Some of the implementation  classes  are  [BufferedReader](https://www.javatpoint.com/java-bufferedreader-class),  [CharArrayReader](https://www.javatpoint.com/java-chararrayreader-class),  [FilterReader](https://www.javatpoint.com/java-filterreader-class),  [InputStreamReader](https://www.javatpoint.com/Input-from-keyboard-by-InputStreamReader), PipedReader and [StringReader](https://www.javatpoint.com/java-stringreader-class)

# Writer Class

It is a class for writing to character streams. Some of the implementation classes  are **FileWriter, PrintWriter, and BufferWriter**

****

File Handling using Some Important Character Stream Classes

**FileReader** class is used to read data from the file. **FileReader** class inherits from the InputStreamReader class. The FileReader class provides a method **read()** to read data/stream from a file character by character. It returns a character in ASCII form.

**BufferedReader** class is used to read the text from a character-based input stream. It can be used to read data line by line by readLine() method. readLine() return a line of text It makes the performance fast. It inherits [Reader](https://www.javatpoint.com/java-reader-class) [class](https://www.javatpoint.com/object-and-class-in-java).

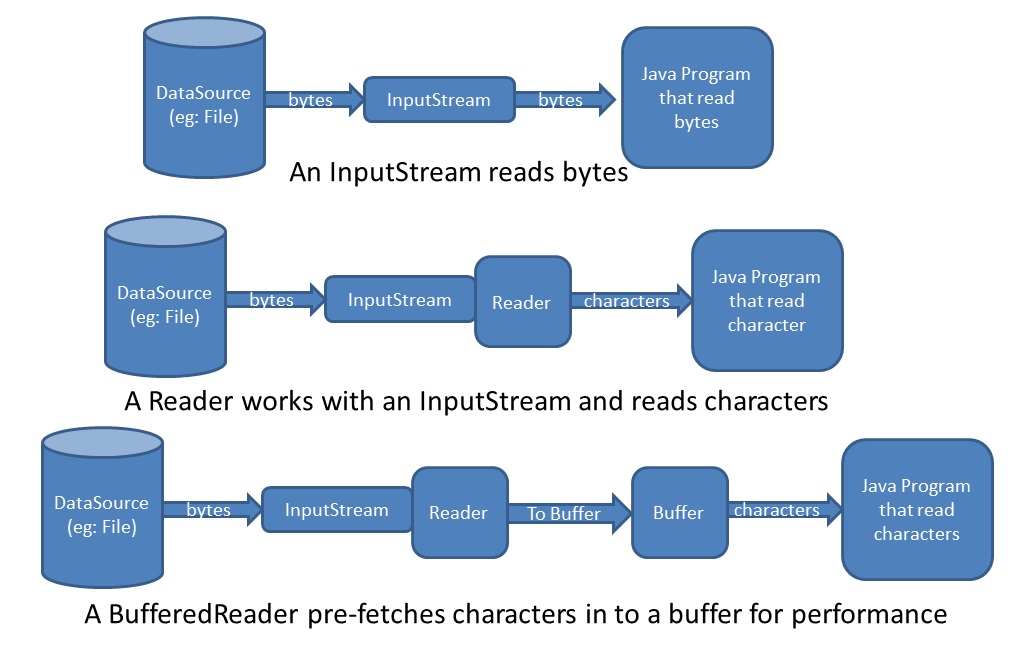
A buffer is essentially a block of memory into which you can write data

**FileWriter** class is used to write character-oriented data to a [file](https://www.javatpoint.com/java-file-class). It is character-oriented class which is used for file handling in [java](https://www.javatpoint.com/java-tutorial). In FileWriter Class we don't need to convert string into byte [array](https://www.javatpoint.com/array-in-java) because it provides method to write string directly.

The FileWriter class provides a method **write()** to write data to a file character by character.

**PrintWriter** class is the implementation of [Writer](https://www.javatpoint.com/java-writer-class) class. It is used to print the formatted representation of data to the text-output stream. For instance writing int, long and other Primitive Data formatted as text rather than as their bytes Values. The Java PrintWriter is useful if you are generating reports (or similar) where you have to mix text and numbers

**BufferedWriter** class is used to provide buffering for Writer instances. It makes the performance fast. It inherits [Writer](https://www.javatpoint.com/java-writer-class) class. The buffering characters are used for providing the efficient writing of single [arrays](https://www.javatpoint.com/array-in-java), characters, and [strings](https://www.javatpoint.com/java-string).



**FileReader** Example

Constructors of FileReader class

|  |  |
| --- | --- |
| FileReader(String file) | It gets filename in [string](https://www.javatpoint.com/java-string). It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException. |

## Methods of FileReader class

|  |  |
| --- | --- |
| **Method** | **Description** |
| read() | read a single character. It is used to return a character in ASCII form. It returns -1 at the end of file. |
| void close() | It is used to close the FileReader class. |

In below example, we are reading the data from the text file **file.txt** using Java FileReader class. But we will get only one character at a time

|  |
| --- |
| **public** **static** **void** main(String[] args) {  **try** {  FileReader fr=**new** FileReader("C:\\Users\\PSAdmin\\TestingFolder\\file.txt");  System.***out***.println(fr.read());  int data = fr.read();  System.***out***.println( (**char**)data ); // typeCasting: because return a character in ASCII form  fr.close  }  **catch** (Exception e) {  System.out.println(“input file is not available”)  }  } |

Let put “System.***out***.println(fr.read())” line in whileloop to read all character from the **file**

|  |
| --- |
| **public** **static** **void** main(String[] args) {  // **TODO** Auto-generated method stub  **try** {  FileReader fr=**new** FileReader("C:\\Users\\PSAdmin\\Downloads\\TestingFolder\\file.txt");    **int** i;  **while**((i=fr.read())!=-1) /\* read() reading a single character.  return a character in ASCII form. It returns -1 at the end of file. \*/  {  System.out.print(i);  System.out.print((char)i);  fr.close();  }  }  **catch** (Exception e) {  System.***out***.println("Cannot read the file");  }   1. } |

**FileWriter Example**

### Constructor of FileWriter class

|  |  |
| --- | --- |
| FileWriter(String fileName) | Creates a FileWriter object using specified fileName. It throws an IOException if the named file exists. |
| FileWriter(String fileName, boolean append): | Creates a FileWriter object using specified fileName with a boolean indicating whether or not to **append** the data written. If the second argument is true, then the data will be written to the end of the file rather than the beginning. It throws an IOException if the named file exists |

Method of Reader Class

|  |  |
| --- | --- |
| Write(int c) | We can send ASCII character, it will convert into single character and writes a single character to a file |
| Write(String) | It writes a [string](https://www.javatpoint.com/java-string). |

**In below example, we are writing the data to the text file**file.txt**using Java FileWriter class. But we can write only one character at a time**

|  |
| --- |
| **public** **static** **void** main(String[] args) {  **try** {  FileWriter w = **new** FileWriter("C:\\Users\\PSAdmin\\Downloads\\TestingFolder\\output.txt");  String content = "I love my country";  //write() methods allow you to write character or strings to a file.  w.write(content);  w.write(65); // ASCI to Char  w.write(66);  w.write(67);  System.***out***.println("Done");  } **catch** (IOException e) {  e.printStackTrace();  }  } |

**Let's look at the following example that reads data from a file and writes the same to another file using FileReader and FileWriter classes.**

|  |
| --- |
| **public** **static** **void** main(String args[]) **throws** IOException {  FileReader in = **null**;  FileWriter out = **null**;  **try** {  in = **new** FileReader("C:\\Users\\PSAdmin\\Downloads\\TestingFolder\\testingFile.txt");  out = **new** FileWriter("C:\\Users\\PSAdmin\\Downloads\\TestingFolder\\sampleOutput.txt");  i**nt** c;  **while** ((c = in.read()) != -1) {  out.write(c);  }  System.***out***.println("Reading and Writing in a file is done!!!");  }  **catch**(Exception e) {  System.***out***.println(e);  }  **finally** {  **if** (in != **null**) {  in.close();  }  **if** (out != **null**) {  out.close();  }  }  } |

**FileReader, BufferedReader, PrintWriter Example**

**Let’s Modify the above examples to use Line-oriented I/O. to do this, we can use two classes which are BufferedReader and PrintWriter**

The PrintWriter class provides various methods that allow us to print data to the output.

### print() Method

* print() - prints the specified data to the writer
* println() - prints the data to the writer along with a new line character at the end

**Read and Write Example - Read one Line at a Time**

|  |
| --- |
| **public** **static** **void** main(String[] args) {  // **TODO** Auto-generated method stub  **try** {  FileReader fr=**new** FileReader("C:\\Users\\PSAdmin\\Downloads\\TestingFolder\\testingFile.txt");  BufferedReader br=**new** BufferedReader(fr);  PrintWriter w = **new** PrintWriter("C:\\Users\\PSAdmin\\Downloads\\TestingFolder\\output.txt");  String st;  **while**((st = br.readLine()) != **null** )  {  w.println(st);  }  w.close();  }  **catch**(FileNotFoundException e)  {  System.out.println(“you got exception. Input File is not found”)  }  **catch**(IOException i)  {  System.out.println(“you got exception. outFile is not found”)  }} |

**PrintWriter and BufferWriter Example**

**Goto 🡪 Advanced Core Java - Basic IO** 🡪 **Slider number 22**

**Goto for bytes 🡪https://web.stanford.edu/class/cs101/bits-bytes.html**

**File Handling using Byte Stream**

In java, we can use a byte stream to handle files. The byte stream has the following built-in classes to perform various operations on a file.

* **FileInputStream** - It is a built-in class in java that allows reading data from a file. This class has implemented based on the byte stream. The FileInputStream class provides a method **read()** to read data from a file byte by byte.
* **FileOutputStream** - It is a built-in class in java that allows writing data to a file. This class has implemented based on the byte stream. The FileOutputStream class provides a method **write()** to write data to a file byte by byte.

Let's look at the following example program that reads data from a file and writes the same to another file using FileInputStream and FileOutputStream classes.

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** IOException {  FileInputStream in = **null**;  FileOutputStream out = **null**;  **try** {  in = **new** FileInputStream("C:\\Users\\PSAdmin\\Downloads\\TestingFolder\\testingFile.txt");  out = **new** FileOutputStream("C:\\Users\\PSAdmin\\Downloads\\TestingFolder\\sampleOutput.txt");  i**nt** c;  **while** ((c = in.read()) != -1) { // read byte by byte  out.write(c); // write byte by byte}  System.***out***.println("Reading and Writing has been success!!!");  }  **catch**(Exception e){  System.***out***.println(e);  }**finally** {  **if** (in != **null**) {  in.close();  }  **if** (out != **null**) {  out.close();  } } } |

For more details visit below link

http://tutorials.jenkov.com/java-io/file.html