 **XIO: IO FILE handling**

1. **What is the purpose of File class?**
2. Java File class represents the files and directory pathnames in an abstract manner. This class is used for creation of files and directories, file searching, file deletion etc.

File file = new File("fileWrite2.txt"); // create a File object AND

// open "fileWrite2.txt"

FileReader fr = new FileReader(file); // create a FileReader to get

// data from 'file'

BufferedReader br = new BufferedReader(fr); // create a BufferReader to

// get its data from a Reader

String data = br.readLine(); // read some data

BufferedReader in = new BufferedReader(new FileReader("foo.in"));

File file = new File("foo");

Always creates a File object, and then does one of two things:

1. If "foo" does NOT exist, no actual file is created.
2. If "foo" does exist, the new File object refers to the existing file.

Notice that File file = new File("foo"); NEVER creates an actual file.

There are two ways to create a file:

1. Invoke the createNewFile() method on a File object. For example:

File file = new File("foo"); // no file yet

file.createNewFile(); // make a file, "foo" which

// is assigned to 'file'

1. Create a Writer or a Stream. Specifically, create a FileWriter, a PrintWriter, or a FileOutputStream. Whenever you create an instance of one of these classes, you automatically create a file, unless one already exists, for instance

File file = new File("foo"); // no file yet

FileWriter fw = new FileWriter(file); // make a FileWriter object AND

// make a file, "foo" to which

// 'file' is assigned, AND assign

// 'fw' to the FileWriter

**File I/O**

* The classes you need to understand in java.io are File, FileReader, BufferedReader, FileWriter, BufferedWriter, PrintWriter, and Console.
* A new File object doesn't mean there's a new file on your hard drive.
* File objects can represent either a file or a directory.
* The File class lets you manage (add, rename, and delete) files and directories.
* The methods createNewFile() and mkdir() add entries to your file system.
* FileWriter and FileReader are low-level I/O classes. You can use them to write and read files, but they should usually be wrapped.
* Classes in java.io are designed to be "chained" or "wrapped." (This is a common use of the decorator design pattern.)
* It's very common to "wrap" a BufferedReader around a FileReader or a BufferedWriter around a FileWriter, to get access to higher-level (more convenient) methods.
* PrintWriters can be used to wrap other Writers, but as of Java 5 they can be built directly from Files or Strings.

**File Streaming** File streaming is carried out by FileInputStream object instance in Java.

// read byte by byte and store into this variable

int byt;

while((byt = fileInputStream.read()) != -1) {

fileOutputStream.write(byt);

}

This objects actually reads a byte(8-bit) at a time and writes it to the given file. It's good when you are working with raw data file, such as images or audio (same but uses AudioInputStream object), etc. but for the text files, it is inconvenient or slower, because it waste the time looping though. you also need to provide a text file character set otherwise whatever you write you won't see in the file, you would just see some random text and signs.

**File Reading (basically streaming with character)** FileReader is used for file reading, so the operation is as,

int c;

while ( (c = fileReader.read()) != -1) { // some logic }

instead of fetching a byte each time, it fetches 2 bytes a time. But still runs the same number of times as the characters in your file. But it is better than byte steaming such as 'FileInputStream' when you're dealing with files. But the both instances use an integer variable to store the value coming from an input file. Byte stream stores 8 bits into a variable and FileReader stores 16 bits, but here, you don't have to specify file's character set and some few more properties, it automatically does it for you, and also supports internationalization. but it's still slower.

**Buffering streams** Now file buffering is faster than file stream reading. It's the same concept as you see on youtube. youtube buffers some amount of video before it actually starts playing it, so you can have flawless video watching experience.

So the file buffering uses BufferedReader object instance, and you need to specify which stream you want to buffer, in this case it's a file so you need to pass a FileReader object.

BufferedReader br = new BufferedReader( new FileReader("example.txt") );

now here, bufferedReader buffers the file means it keeps reading a file one by one character, until it hits '\n', '\r\n' (new line) character and stores fetched characters into a string (in short fetches the lines and stores it into buffer).

String line;

while ((line = br.readLine()) != null) {

printWriter.println(line);

}

now, you can see instead of storing each bytes or 2bytes value in a int variable, it stores the whole line into a memory and works on it. But buffer reader cannot work alone, because it just buffers so you need to specify which stream you want to buffer? It could be a file stream as you saw earlier or it could be a console, as

BufferedReader br = new BufferedReader(System.in);

then it will scan a whole line from the console. that's why you use bufferedReader.readLine() in java to collect the input from a console.

1. **What is the difference between the Reader/Writer class hierarchy and the InputStream/ OutputStream class hierarchy?**
2. First thing you need to understand is

* What is **STREAM?**

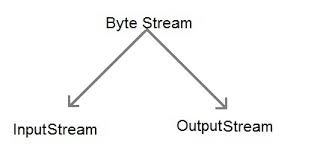
A stream can be defined as a sequence of data. The Input Stream is used to read data from a source and the Output Stream is used for writing data to a destination.

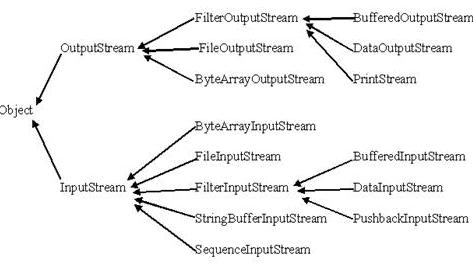
* *Next is type of streams*

We have byte stream and character stream.

**Byte Stream**

Classes we have in Input Stream and output stream



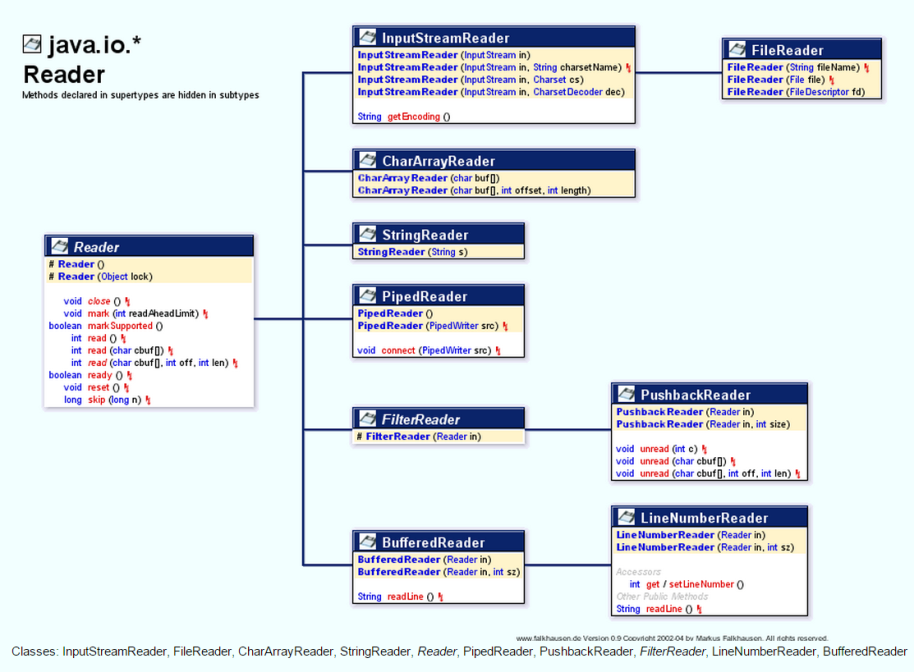


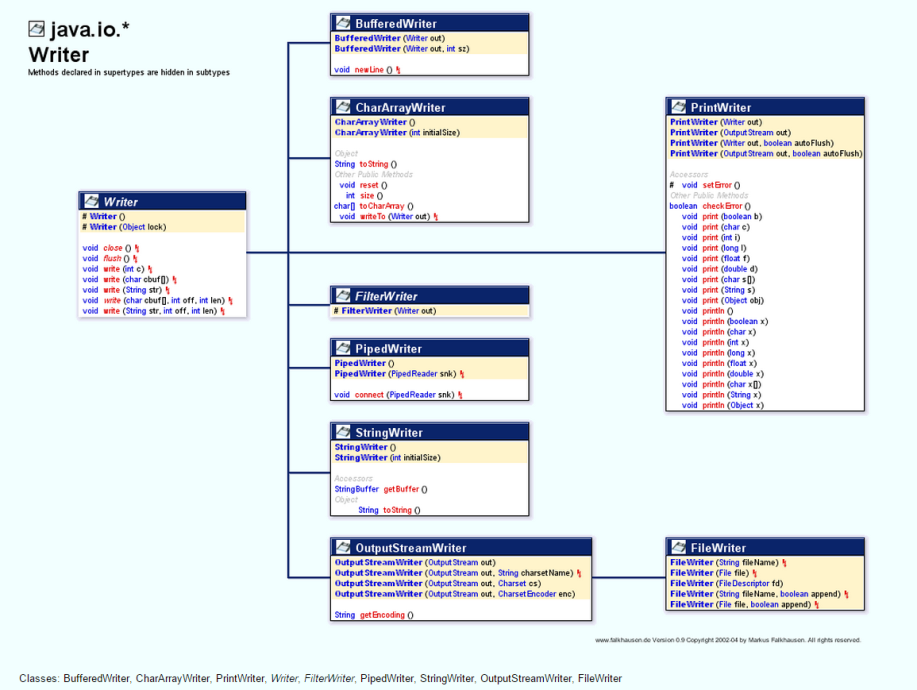
As the name suggests in **simple terms** input stream is used to input the data and output stream is used to output the data

Java **byte streams** are used to perform input and output of **8-bit bytes.** Though there are many classes related to byte streams but the most frequently used classes are, **FileInputStream** and **FileOutputStream**. Also Java **Byte streams** are used to perform input and output of 8-bit bytes.

**Character Stream**

Java **Character streams** are used to perform input and output for **16-bit unicode**. Though there are many classes related to character streams but the most frequently used classes are, **FileReader** and **FileWriter**. Though internally FileReader uses FileInputStream and FileWriter uses FileOutputStream but here major difference is that FileReader reads two bytes at a time and fileWriter writes two bytes at a time.





**For reference**

1. [What is InputStream & Output Stream? Why do we use them and when do we use each of them?](http://stackoverflow.com/questions/1830698/what-is-inputstream-output-stream-why-do-we-use-them-and-when-do-we-use-each)
2. [java DataOutputStream getOutputStream() getInputStream()](http://stackoverflow.com/questions/4913559/java-dataoutputstream-getoutputstream-getinputstream)

**Example for getInputStream and getOutputStream**

1. <http://zerioh.tripod.com/ressources/sockets.html>

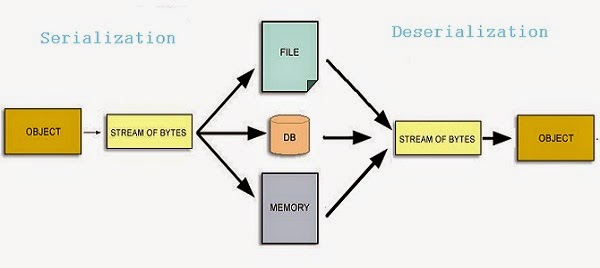
**New Link** <http://docs.oracle.com/javase/tutorial/essential/io/buffers.html>

1. **What is the use of Serialization**
2. As written above serialization will translate the Object state to Byte Streams. This Byte stream can be used for different purpose.

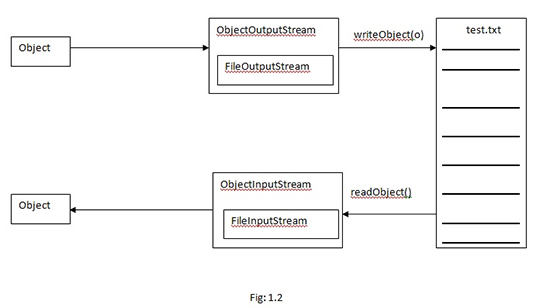
* Write to Disk
* Store in Memory
* Sent byte stream to other platform over network
* Save byte stream in DB(As BLOB)

1. **What is Serialization and deserialization?**
2. Serialization is the process of writing the state of an object to a byte stream. Deserialization is the process of restoring these objects.

**As byte stream create is platform neutral hence once objects created in one system can be deserialized in other platform. 🡪 candidate for webservice**



* The classes you need to understand are all in the java.io package; they include: ObjectOutputStream and ObjectInputStream primarily, and FileOutputStream and FileInputStream because you will use them to create the low-level streams that the ObjectXxxStream classes will use.
* A class must implement Serializable before its objects can be serialized.
* The ObjectOutputStream.writeObject() method serializes objects, and the ObjectInputStream.readObject() method deserializes objects.
* If you mark an instance variable transient, it will not be serialized even though the rest of the object's state will be.
* You can supplement a class's automatic serialization process by implementing the writeObject() and readObject() methods. If you do this, embedding calls to defaultWriteObject() and defaultReadObject(), respectively, will handle the part of serialization that happens normally.
* If a superclass implements Serializable, then its subclasses do automatically.
* If Super class is not serializable then when sub class is de serialized then super class’s default constructor will be invoked. Hence all variable will get default value and reference will be null.



1. **What is a transient variable?**
2. A transient variable is a variable that may not be serialized during Serialization and which is initialized by its default value during de-serialization.

### Question 1) What is the difference between Serializable and Externalizable interface in Java?

This is most frequently asked question in Java serialization interview. Here is my version Externalizable provides us writeExternal() and readExternal() method which gives us flexibility to control java serialization mechanism instead of relying on Java's default serialization. Correct implementation of Externalizable interface can [improve performance of application](http://javarevisited.blogspot.sg/2012/01/improve-performance-java-database.html) drastically.

### Question 2) How many methods Serializable has? If no method then what is the purpose of Serializable interface?

Serializable interface exists in java.io package and forms core of java serialization mechanism. It doesn't have any method and also called [Marker Interface in Java](http://javarevisited.blogspot.sg/2012/01/what-is-marker-interfaces-in-java-and.html). When your class implements java.io.Serializable interface it becomes Serializable in Java and gives compiler an indication that use Java Serialization mechanism to serialize this object.

### Question 3) What is serialVersionUID? What would happen if you don't define this?

One of my favorite question interview question on Java serialization. SerialVersionUID is an ID which is stamped on object when it get serialized usually hashcode of object, you can use tool serialver to see serialVersionUID of a serialized object . SerialVersionUID is used for version control of object. you can specify serialVersionUID in your [class file](http://javarevisited.blogspot.sg/2012/05/10-points-about-class-file-in-java.html) also. Consequence of not specifying serialVersionUID is that when you add or modify any field in class then already serialized class will not be able to recover because serialVersionUID generated for new class and for old serialized object will be different. Java serialization process relies on correct serialVersionUID for recovering state of serialized object and throws java.io.InvalidClassException in case of serialVersionUID mismatch, to learn more about serialversionuid see this [article](http://javarevisited.blogspot.sg/2014/05/why-use-serialversionuid-inside-serializable-class-in-java.html).

### Question 4) While serializing you want some of the members not to serialize? How do you achieve it?

Another frequently asked Serialization interview question. This is sometime also asked as what is the use of [transient variable](http://javarevisited.blogspot.sg/2011/09/transient-keyword-variable-in-java.html), does transient and [static variable](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html) gets serialized or not etc. so if you don't want any field to be part of object's state then declare it either static or transient based on your need and it will not be included during Java serialization process.

### Question 5) What will happen if one of the members in the class doesn't implement Serializable interface?

One of the easy question about Serialization process in Java. If you try to serialize an object of a class which implements Serializable, but the object includes a reference to an non- Serializable class then a ‘NotSerializableException’ will be thrown at runtime and this is why I always put a *SerializableAlert* (comment section in my code) , one of the [code comment best practices](http://javarevisited.blogspot.sg/2011/08/code-comments-java-best-practices.html), to instruct developer to remember this fact while adding a new field in a Serializable class.

### Question 6) If a class is Serializable but its super class in not, what will be the state of the instance variables inherited from super class after deserialization?

Java serialization process only continues in object hierarchy till the class is Serializable i.e. implements Serializable [interface in Java](http://javarevisited.blogspot.sg/2012/04/10-points-on-interface-in-java-with.html) and values of the instance variables inherited from super class will be initialized by calling constructor of Non-Serializable Super class during deserialization process. Once the [constructor chaining](http://javarevisited.blogspot.sg/2012/01/what-is-constructor-overloading-in-java.html) will started it wouldn't be possible to stop that , hence even if classes higher in hierarchy implements Serializable interface , there constructor will be executed. As you see from the statement this Serialization interview question looks very tricky and tough but if you are familiar with key concepts its not that difficult.

### Question 7) Can you Customize Serialization process or can you override default Serialization process in Java?

The answer is yes you can. We all know that for serializing an object ObjectOutputStream.writeObject (saveThisobject) is invoked and for reading object ObjectInputStream.readObject() is invoked but there is one more thing which Java Virtual Machine provides you is to define these two method in your class. If you define these two methods in your class then JVM will invoke these two methods instead of applying default serialization mechanism. You can customize behavior of object serialization and deserialization here by doing any kind of pre or post processing task. Important point to note is making these methods private to avoid being inherited, [overridden or overloaded](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html). Since only Java Virtual Machine can call private method integrity of your class will remain and Java Serialization will work as normal. In my opinion this is one of the best question one can ask in any Java Serialization interview, a good follow-up question is why should you provide custom serialized form for your object?

### Question 8) Suppose super class of a new class implement Serializable interface, how can you avoid new class to being serialized?

One of the tricky interview question in Serialization in Java. If Super Class of a Class already implements Serializable interface in Java then its already Serializable in Java, since you can not unimplemented an interface its not really possible to make it Non Serializable class but yes there is a way to avoid serialization of new class. To avoid Java serialization you need to implement writeObject() and readObject() method in your Class and need to throw NotSerializableException from those method. This is another benefit of customizing java serialization process as described in above [Serialization interview question](http://javarevisited.blogspot.sg/2011/04/top-10-java-serialization-interview.html) and normally it asked as follow-up question as interview progresses.

### Question 9) Which methods are used during Serialization and DeSerialization process in Java?

This is very common interview question in Serialization basically interviewer is trying to know; Whether you are familiar with usage of readObject(), writeObject(), readExternal() and writeExternal() or not. Java Serialization is done by java.io.ObjectOutputStream class. That class is a filter stream which is wrapped around a lower-level byte stream to handle the serialization mechanism. To store any object via serialization mechanism we call ObjectOutputStream.writeObject(saveThisobject) and to deserialize that object we call ObjectInputStream.readObject() method. Call to writeObject() method trigger serialization process in java. one important thing to note about readObject() method is that it is used to read bytes from the persistence and to create object from those bytes and its return an [Object](http://javarevisited.blogspot.sg/2012/03/10-object-oriented-design-principles.html) which needs to be type cast to correct type.