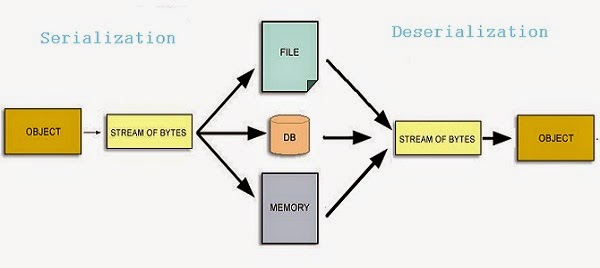
Most commercial project uses either database or memory mapped file or simply flat file for there persistence requirement and only few of them rely on serialization process in Java. Anyway this post is not a Java serialization tutorial or how to serialize object in java but about interview questions around serialization mechanism and Serialization API, Which is worth to have a look before going for any Java or [J2EE interview](http://javarevisited.blogspot.sg/2011/09/spring-interview-questions-answers-j2ee.html) and surprising yourself with some unknown contents. for those who are not familiar about java Serialization "Java serialization is the process which is used to serialize object in java by storing object’s state into a file with extension .ser and recreating object's state from that file, this reverse process is called deserialization.

[](http://4.bp.blogspot.com/-0v0DwGs6Ep0/VJ10DnPNs7I/AAAAAAAACQo/QVe5P1_y4DQ/s1600/Serialization%2Bin%2BJava.JPG)

The Java Serialization API provides a standard mechanism for developers to handle object serialization using Serializable and Externalizable interface. By the way this article is in continuation of my previous article [Top 20 design pattern interview questions](http://javarevisited.blogspot.sg/2012/06/20-design-pattern-and-software-design.html),  and [10 Interview questions on Singleton Pattern in Java](http://javarevisited.blogspot.com/2011/03/10-interview-questions-on-singleton.html) So here we go.

**What is Serialization in Java**

Object Serialization in Java is a process used to convert Object into a binary format which can be persisted into disk or sent over network to any other running [Java virtual machine](http://javarevisited.blogspot.sg/2011/11/hotspot-jvm-options-java-examples.html); the reverse process of creating object from binary stream is called deserialization in Java. Java provides Serialization API for serializing and deserializing object which includes java.io.Serializable, java.io.Externalizable, ObjectInputStream and ObjectOutputStream etc. Java programmers are free to use default Serialization mechanism which Java uses based upon structure of class but they are also free to use there own custom binary format, which is often advised as Serialization best practice, Because serialized binary format becomes part of Class's exported API and it can potentially break [Encapsulation in Java](http://javarevisited.blogspot.sg/2012/03/what-is-encapsulation-in-java-and-oops.html) provided by private and [package-private fields](http://javarevisited.blogspot.sg/2012/05/how-to-access-private-field-and-method.html). This pretty much answer the question What is Serialization in Java.

**How to make a Java class Serializable?**

Making a class Serializable in Java is very easy, Your Java class just needs to implements java.io.Serializable interface and JVM will take care of serializing object in default format. Decision to making a [Class](http://javarevisited.blogspot.sg/2011/10/class-in-java-programming-general.html) Serializable should be taken concisely because though near term cost of making a Class Serializable is low, long term cost is substantial and it can potentially limit your ability to further modify and change its implementation because like any public API, serialized form of an object becomes part of public API and when you change structure of your class by implementing addition interface, adding or removing any field can potentially break default serialization, this can be minimized by using a custom binary format but still requires lot of effort to ensure backward compatibility. One example of How Serialization can put constraints on your ability to change class is SerialVersionUID. If you don't explicitly declare SerialVersionUID then JVM generates its based upon structure of class which depends upon interfaces a class implements and several other factors which is subject to change. Suppose you implement another interface than [JVM](http://javarevisited.blogspot.sg/2011/12/jre-jvm-jdk-jit-in-java-programming.html) will generate a different SerialVersionUID for new version of class files and when you try to load old object object serialized by old version of your programyou will get InvalidClassException.

**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 Externalizableinterface 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.  
  
**Question 10) Suppose you have a class which you serialized it and stored in persistence and later modified that class to add a new field. What will happen if you deserialize the object already serialized?**

It depends on whether class has its own serialVersionUID or not. As we know from above question that if we don't provide serialVersionUID in our code java compiler will generate it and normally it’s [equal to hashCode of object](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html). by adding any new field there is chance that new serialVersionUID generated for that class version is not the same of already serialized object and in this case Java Serialization API will [throw](http://javarevisited.blogspot.sg/2012/02/difference-between-throw-and-throws-in.html) java.io.InvalidClassException and this is the reason its recommended to have your own serialVersionUID in code and make sure to keep it same always for a single class.

**11) What are the compatible changes and incompatible changes in Java Serialization Mechanism?**

The real challenge lies with change in class structure by adding any field, method or removing any field or method is that with already serialized object. As per Java Serialization specification adding any field or method comes under compatible change and changing class hierarchy or UN-implementing Serializable interfaces some under non compatible changes. For complete list of compatible and non compatible changes I would advise reading Java serialization specification.  
  
**12) Can we transfer a Serialized object vie network?**

Yes you can transfer a Serialized object via network because Java serialized object remains in form of bytes which can be transmitter via network. You can also store serialized object in Disk or database as Blob.

**13) Which kind of variables is not serialized during Java Serialization?**

This question asked sometime differently but the purpose is same whether Java developer knows specifics about [static and transient variable](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html) or not. Since static variables belong to the class and not to an object they are not the part of the state of object so they are not saved during Java Serialization process. As Java Serialization only persist state of object and not object itself. Transient variables are also not included in java serialization process and are not the part of the object’s serialized state. After this question sometime interviewer ask a follow-up if you don't store values of these variables then what would be value of these variable once you deserialize and recreate those object? This is for you guys to think about .

**Q1) What is Serialization?**

Ans) Serializable is a marker interface. When an object has to be transferred over a network ( typically through rmi or EJB) or to persist the state of an object to a file, the object Class needs to implement Serializable interface. Implementing this interface will allow the object converted into bytestream and transfer over a network.

**Q2) What is use of serialVersionUID?**

Ans) During object serialization, the default Java serialization mechanism writes the metadata about the object, which includes the class name, field names and types, and superclass. This class definition is stored as a part of the serialized object. This stored metadata enables the deserialization process to reconstitute the objects and map the stream data into the class attributes with the appropriate type  
Everytime an object is serialized the java serialization mechanism automatically computes a hash value. ObjectStreamClass's *computeSerialVersionUID*() method passes the class name, sorted member names, modifiers, and interfaces to the secure hash algorithm (SHA), which returns a hash value.The serialVersionUID is also called *suid*.  
So when the serilaize object is retrieved , the JVM first evaluates the *suid* of the serialized class and compares the *suid* value with the one of the object. If the suid values match then the object is said to be compatible with the class and hence it is de-serialized. If not *InvalidClassException*exception is thrown.  
  
Changes to a serializable class can be compatible or incompatible. Following is the list of changes which are compatible:

* Add fields
* Change a field from static to non-static
* Change a field from transient to non-transient
* Add classes to the object tree

List of incompatible changes:

* Delete fields
* Change class hierarchy
* Change non-static to static
* Change non-transient to transient
* Change type of a primitive field

So, if no suid is present, inspite of making compatible changes, jvm generates new *suid* thus resulting in an exception if prior release version object is used .

The only way to get rid of the exception is to recompile and deploy the application again.

If we explicitly mention the sUid using the statement:

private final static long serialVersionUID = <integer value>

then if any of the metioned compatible changes are made the class need not to be recompiled. But for incompatible changes there is no other way than to compile again.

**Q3) What is the need of Serialization?**

Ans) The serialization is used :-

* To send state of one or more object’s state over the network through a socket.
* To save the state of an object in a file.
* An object’s state needs to be manipulated as a stream of bytes.

**Q4) Other than Serialization what are the different approach to make object Serializable?**

Ans) Besides the Serializable interface, at least three alternate approaches can serialize Java objects:

* For object serialization, instead of implementing the Serializable interface, a developer can implement the Externalizable interface, which extends Serializable. By implementing Externalizable, a developer is responsible for implementing the writeExternal() and readExternal() methods. As a result, a developer has sole control over reading and writing the serialized objects.
* XML serialization is an often-used approach for data interchange. This approach lags runtime performance when compared with Java serialization, both in terms of the size of the object and the processing time. With a speedier XML parser, the performance gap with respect to the processing time narrows. Nonetheless, XML serialization provides a more malleable solution when faced with changes in the serializable object.
* Finally, consider a "roll-your-own" serialization approach. You can write an object's content directly via either the ObjectOutputStream or the DataOutputStream. While this approach is more involved in its initial implementation, it offers the greatest flexibility and extensibility. In addition, this approach provides a performance advantage over Java serialization.

**Q5) Do we need to implement any method of Serializable interface to make an object serializable?**

Ans) No. Serializable is a Marker Interface. It does not have any methods.

**Q6) What happens if the object to be serialized includes the references to other serializable objects?**

Ans) If the object to be serialized includes references to the other objects, then all those object’s state also will be saved as the part of the serialized state of the object in question. The whole object graph of the object to be serialized will be saved during serialization automatically provided all the objects included in the object’s graph are serializable.

**Q7) What happens if an object is serializable but it includes a reference to a non-serializable object?**

Ans- 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.

public class NonSerial {

//This is a non-serializable class

}

public class MyClass implements Serializable {

private static final long serialVersionUID = 1L;

private NonSerial nonSerial;

MyClass(NonSerial nonSerial){

this.nonSerial = nonSerial;

}

public static void main(String [] args) {

NonSerial nonSer = new NonSerial();

MyClass c = new MyClass(nonSer);

try {

FileOutputStream fs = new FileOutputStream("test1.ser");

ObjectOutputStream os = new ObjectOutputStream(fs);

os.writeObject(c);

os.close();

} catch (Exception e) { e.printStackTrace(); }

try {

FileInputStream fis = new FileInputStream("test1.ser");

ObjectInputStream ois = new ObjectInputStream(fis);

c = (MyClass) ois.readObject();

ois.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

On execution of above code following exception will be thrown;

java.io.NotSerializableException: NonSerial

at java.io.ObjectOutputStream.writeObject0(ObjectOutputStream.java)

**Q8) Are the static variables saved as the part of serialization?**

Ans) No. The static variables belong to the class are not the part of the state of the object so they are not saved as the part of serialized object.

**Q9)What is a transient variable?**

Ans) These variables are not included in the process of serialization and are not the part of the object’s serialized state.

**Q10) What will be the value of transient variable after de-serialization?**

Ans) It’s default value.  
e.g. if the transient variable in question is an int, it’s value after deserialization will be zero.

public class TestTransientVal implements Serializable {

private static final long serialVersionUID = -22L;

private String name;

transient private int age;

TestTransientVal(int age, String name) {

this.age = age;

this.name = name;

}

public static void main(String [] args) {

TestTransientVal c = new TestTransientVal(1,"ONE");

System.out.println("Before serialization:" + c.name + " "+ c.age);

try {

FileOutputStream fs =new FileOutputStream("testTransient.ser");

ObjectOutputStream os = new ObjectOutputStream(fs);

os.writeObject(c);

os.close();

} catch (Exception e) { e.printStackTrace(); }

try {

FileInputStream fis =new FileInputStream("testTransient.ser");

ObjectInputStream ois =new ObjectInputStream(fis);

c = (TestTransientVal) ois.readObject();

ois.close();

} catch (Exception e) { e.printStackTrace(); }

System.out.println("After de-serialization:" + c.name +" "+ c.age);

}

}

Result of executing above piece of code –  
Before serialization: - Value of non-transient variable ONE Value of transient variable 1   
After de-serialization:- Value of non-transient variable ONE Value of transient variable 0

Explanation –   
The transient variable is not saved as the part of the state of the serailized variable, it’s value after de-serialization is it’s default value.

**Q11) Does the order in which the value of the transient variables and the state of the object using the defaultWriteObject() method are saved during serialization matter?**

Ans) Yes, while restoring the object’s state the transient variables and the serializable variables that are stored must be restored in the same order in which they were saved.

**Q12) How can one customize the Serialization process? or What is the purpose of implementing the writeObject() and readObject() method?**

Ans) When you want to store the transient variables state as a part of the serialized object at the time of serialization the class must implement the following methods –

private void wrtiteObject(ObjectOutputStream outStream) {

//code to save the transient variables state

//as a part of serialized object

}

private void readObject(ObjectInputStream inStream) {

//code to read the transient variables state

//and assign it to the de-serialized object

}

public class TestCustomizedSerialization implements Serializable {

private static final long serialVersionUID =-22L;

private String noOfSerVar;

transient private int noOfTranVar;

TestCustomizedSerialization(int noOfTranVar, String noOfSerVar) {

this.noOfTranVar = noOfTranVar;

this.noOfSerVar = noOfSerVar;

}

private void writeObject(ObjectOutputStream os) {

try {

os.defaultWriteObject();

os.writeInt(noOfTranVar);

} catch (Exception e) { e.printStackTrace(); }

}

private void readObject(ObjectInputStream is) {

try {

is.defaultReadObject();

int noOfTransients = (is.readInt());

} catch (Exception e) {

e.printStackTrace(); }

}

public int getNoOfTranVar() {

return noOfTranVar;

}

The value of transient variable ‘noOfTranVar’ is saved as part of the serialized object manually by implementing writeObject() and restored by implementing readObject().  
The normal serializable variables are saved and restored by calling defaultWriteObject() and defaultReadObject()respectively. These methods perform the normal serialization and de-sirialization process for the object to be saved or restored respectively.

**Q13) If a class is serializable but its superclass in not, what will be the state of the instance variables inherited from super class after deserialization?**

Ans) The values of the instance variables inherited from superclass will be reset to the values they were given during the original construction of the object as the non-serializable super-class constructor will run.

E.g.

public class ChildSerializable extends ParentNonSerializable implements Serializable {

private static final long serialVersionUID = 1L;

String color;

ChildSerializable() {

this.noOfWheels = 8;

this.color = "blue";

}

}

public class SubSerialSuperNotSerial {

public static void main(String [] args) {

ChildSerializable c = new ChildSerializable();

System.out.println("Before : - " + c.noOfWheels + " "+ c.color);

try {

FileOutputStream fs = new FileOutputStream("superNotSerail.ser");

ObjectOutputStream os = new ObjectOutputStream(fs);

os.writeObject(c);

os.close();

} catch (Exception e) { e.printStackTrace(); }

try {

FileInputStream fis = new FileInputStream("superNotSerail.ser");

ObjectInputStream ois = new ObjectInputStream(fis);

c = (ChildSerializable) ois.readObject();

ois.close();

} catch (Exception e) { e.printStackTrace(); }

System.out.println("After :- " + c.noOfWheels + " "+ c.color);

}

}

Result on executing above code –

Before : - 8 blue

After :- 4 blue

The instance variable ‘noOfWheels’ is inherited from superclass which is not serializable. Therefore while restoring it the non-serializable superclass constructor runs and its value is set to 8 and is not same as the value saved during serialization which is 4.