**Q1) What is Serialization?**

**Serialization in java** is a mechanism of writing the state of an object into a byte stream.

It is mainly used in Hibernate, RMI, JPA, EJB and JMS technologies.

The reverse operation of serialization is called deserialization.

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.

**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.

**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.

**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

**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. Question 9) Which methods are used during Serialization and DeSerialization process in Java? Ans= 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 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.