Serialization and Deserialization in Java

Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.

[](http://www.geeksforgeeks.org/wp-content/uploads/gq/2016/01/serialize-deserialize-java.png)

The byte stream created is platform independent. So, the object serialized on one platform can be deserialized on a different platform.

To make a Java object serializable we implement the **java.io.Serializable** interface.  
The ObjectOutputStream class contains **writeObject()** method for serializing an Object.

public final void writeObject(Object obj)

throws IOException

The ObjectInputStream class contains **readObject()** method for deserializing an object.

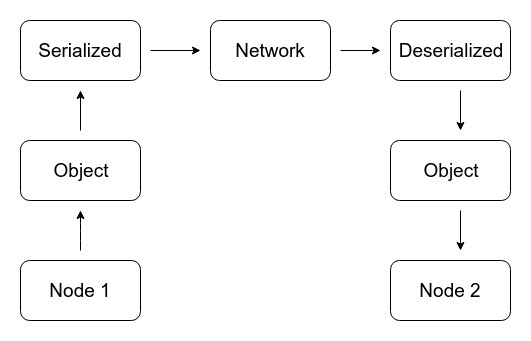
public final Object readObject()

throws IOException,

ClassNotFoundException

**Advantages of Serialization**

1. To save/persist state of an object.
2. To travel an object across a network.



Only the objects of those classes can be serialized which are implementing **java.io.Serializable** interface.

Serializable is a **marker interface** (has no data member and method). It is used to “mark” java classes so that objects of these classes may get certain capability. Other examples of marker interfaces are:- Cloneable and Remote.

**Points to remember**

1. If a parent class has implemented Serializable interface then child class doesn’t need to implement it but vice-versa is not true.
2. Only non-static data members are saved via Serialization process.
3. Static data members and transient data members are not saved via Serialization process. So, if you don’t want to save value of a non-static data member then make it transient.
4. Constructor of object is never called when an object is deserialized.
5. Associated objects must be implementing Serializable interface.

Example :

class A implements Serializable

{

// B also implements Serializable

// interface.

B ob=new B();

}

**SerialVersionUID**  
The Serialization runtime associates a version number with each Serializable class called a SerialVersionUID, which is used during Deserialization to verify that sender and reciever of a serialized object have loaded classes for that object which are compatible with respect to serialization.

If the reciever has loaded a class for the object that has different UID than that of corresponding sender’s class, the Deserialization will result in an **InvalidClassException**. A Serializable class can declare its own UID explicitly by declaring a field name.

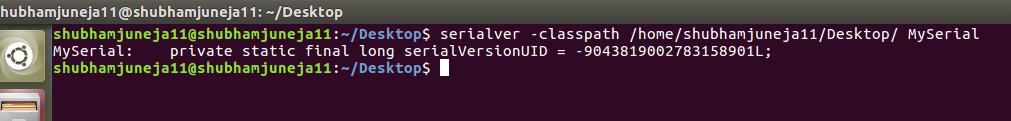
It must be static, final and of type long. i.e- ANY-ACCESS-MODIFIER static final long serialVersionUID=42L;

If a serializable class doesn’t explicitly declare a serialVersionUID, then the serialization runtime will calculate a default one for that class based on various aspects of class, as described in Java Object Serialization Specification. However it is strongly recommended that all serializable classes explicitly declare serialVersionUID value, since its computation is highly sensitive to class details that may vary depending on compiler implementations, any change in class or using different id may affect the serialized data.

It is also recommended to use private modifier for UID since it is not useful as inherited member.

**serialver**  
The serialver is a tool that comes with JDK. It is used to get serialVersionUID number for Java classes.  
You can run the following command to get serialVersionUID

serialver [-classpath classpath] [-show] [classname…]



**Example 1:**

|  |
| --- |
| // Java code for serialization and deserialization  // of a Java object  import java.io.\*;  class Demo implements java.io.Serializable{  public int a;  public String b;  // Default constructor  public Demo(int a, String b) {  this.a = a;  this.b = b;  }  }  class Test{  public static void main(String[] args){  Demo object = new Demo(1, "geeksforgeeks");  String filename = "file.ser";  // Serialization  try{  //Saving of object in a file  FileOutputStream file = new FileOutputStream(filename);  ObjectOutputStream out = new ObjectOutputStream(file);  // Method for serialization of object  out.writeObject(object);  out.close();  file.close();  System.out.println("Object has been serialized");  }  catch (IOException ex){  System.out.println("IOException is caught");  }  Demo object1 = null;  // Deserialization  try{  // Reading the object from a file  FileInputStream file = new FileInputStream(filename);  ObjectInputStream in = new ObjectInputStream(file);  // Method for deserialization of object  object1 = (Demo)in.readObject();  in.close();  file.close();  System.out.println("Object has been deserialized ");  System.out.println("a = " + object1.a);  System.out.println("b = " + object1.b);  }  catch (IOException ex){  System.out.println("IOException is caught");  }  catch (ClassNotFoundException ex){  System.out.println("ClassNotFoundException is caught");  }  }  } |