**Serialization and Deserialization in Java with Example**

1. 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.

make the object long lasting(by either storing it in a db file or a texttual file, or transpoerting it over a network.

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

3. 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

interface Clonable {} clone()

class Students implements Clonable

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

(while deserializing the class SerialID should not have modified)

5.To make a Java object serializable we implement the **java.io.Serializable** interface.

io package

The ObjectOutputStream class contains **writeObject()** method for serializing an Object.

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

6. 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. IS-A  
2. Only non-static data members (**instance variables**) 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.

7. 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 receiver of a serialized object have loaded classes for that object which are compatible with respect to serialization.

if we make changes to the class even after serilization then at deserilization it will not throw an excpetion

8. If the receiver 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**.

9. 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=123456789L

10. 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.

11. java.lang.ClassCastException

class with same name

12.java.io.InvalidClassException

class modified

13. In case of **transient variables:-** A variable defined with transient keyword is not serialized during serialization process.This variable will be initialized with default value during deserialization. (e.g: for objects it is null, for int it is 0).

int x,y;

transient int a=10;

### Why to use the transient keyword?

The transient keyword can be used with the data members of a class in order to avoid their serialization. For example, if a program accepts a user's login details and password. But we don't want to store the original password in the file. Here, we can use transient keyword and when JVM reads the transient keyword it ignores the original value of the object and instead stores the default value of the object.

In case of **static Variables:-** A variable defined with static keyword is not serialized during serialization process.This variable will be loaded with current value defined in the class during deserialization.

example 1

**public** **class** Demo **implements** java.io.Serializable

{

**private** **static** **final** **long** ***serialversionUID*** =123456789L;

**public** **int** a;

**public** String b;

**public** Demo(){

System.***out***.println("hi");

}

**public** Demo(**int** a, String b)

{

**this**.a = a;

**this**.b = b;

}

}

------

**public** **class** SerializeDemo {

**public** **static** **void** main(String[] args)

{

Demo d1 = **new** Demo(1, "Alice");

String filename = "aaa.txt";

// 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(d1);

out.close();

file.close();

System.***out***.println("Object has been serialized");

}

**catch**(IOException ex)

{

System.***out***.println("IOException is caught");

}

}

}

----

**public** **class** UnSerializeDemo {

**public** **static** **void** main(String[] args) {

Demo d2 = **null**;

String filename = "aaa.txt";

// Deserialization

**try**

{

// Reading the object from a file

FileInputStream file = **new** FileInputStream(filename);

ObjectInputStream in = **new** ObjectInputStream(file);

// Method for deserialization of object

d2 = (Demo)in.readObject();

in.close();

file.close();

System.***out***.println("Object has been deserialized ");

System.***out***.println("a = " + d2.a);

System.***out***.println("b = " + d2.b);

}

**catch**(IOException ex)

{

System.***out***.println("IOException is caught");

}

**catch**(ClassNotFoundException ex)

{

System.***out***.println("ClassNotFoundException is caught");

}

}

}

example 2 : using inheritance (IS-A)

**public** **class** Person **implements** Serializable{

**int** adharNo;

String name;

Person(**int** id, String name) {

**this**.adharNo = id;

**this**.name = name;

}

}

---

**public** **class** Student **extends** Person{

String course;

**int** fee;

**public** Student(**int** adharNo, String name, String course, **int** fee) {

**super**(adharNo,name);

**this**.course=course;

**this**.fee=fee;

}

}

----

**public** **class** SerializeDemoWithInheritance {

**public** **static** **void** main(String[] args) {

**try**{

//Creating the object

Student s1 =**new** Student(211,"ravi","Engineering",50000);

//Creating stream and writing the object

FileOutputStream fout=**new** FileOutputStream("f1.txt");

ObjectOutputStream out=**new** ObjectOutputStream(fout);

out.writeObject(s1);

out.close();

System.***out***.println("success");

}

**catch**(Exception e)

{

System.***out***.println(e);

}

}

}

---

**public** **class** UnSerializeDemoWithInheritance {

**public** **static** **void** main(String[] args) {

**try**{

//Creating stream to read the object

ObjectInputStream in=**new** ObjectInputStream(**new** FileInputStream("f.txt"));

Student s=(Student)in.readObject();

//printing the data of the serialized object

System.***out***.println(s.adharNo+" "+s.name+" "+s.course+" "+s.fee);

//closing the stream

in.close();

}**catch**(Exception e){System.***out***.println(e);}

}

}

## example 3: Java Serialization with Aggregation (HAS-A Relationship)

If a class has a reference to another class, all the references must be Serializable otherwise serialization process will not be performed. In such case, NotSerializableException is thrown at runtime.

**Address.java**

1. **class** Address{
2. String addressLine,city,state;
3. **public** Address(String addressLine, String city, String state) {
4. **this**.addressLine=addressLine;
5. **this**.city=city;
6. **this**.state=state;
7. }
8. }

**Student.java**

1. **import** java.io.Serializable;
2. **public** **class** Student **implements** Serializable{
3. **int** id;
4. String name;
5. Address address;//HAS-A
6. **public** Student(**int** id, String name) {
7. **this**.id = id;
8. **this**.name = name;
9. }
10. }

Since Address is not Serializable, you cannot serialize the instance of the Student class.