Serialization and Deserialization in Java

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.

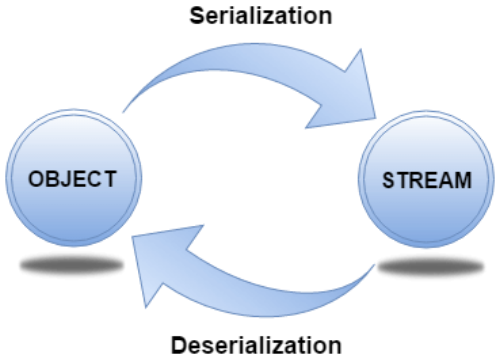
The reverse operation of serialization is called *deserialization* where byte-stream is converted into an object. The serialization and deserialization process is platform-independent, it means you can serialize an object on one platform and deserialize it on a different platform.

For serializing the object, we call the **writeObject()** method of *ObjectOutputStream* class, and for deserialization we call the **readObject()** method of *ObjectInputStream* class.

We must have to implement the *Serializable* interface for serializing the object.

Advantages of Java Serialization

It is mainly used to travel object's state on the network (that is known as marshalling).



java.io.Serializable interface

Serializable is a marker interface (has no data member and method). It is used to "mark" Java classes so that the objects of these classes may get a certain capability. The **Cloneable** and **Remote** are also marker interfaces.

The **Serializable** interface must be implemented by the class whose object needs to be persisted.

The String class and all the wrapper classes implement the *java.io.Serializable* interface by default.

Let's see the example given below:

Student.java

```
1. import java.io.Serializable;
2. public class Student implements Serializable{
3.  int id;
4.  String name;
5.  public Student(int id, String name) {
6.   this.id = id;
7.   this.name = name;
8.  }
9. }
```

In the above example, ***Student*** class implements Serializable interface. Now its objects can be converted into stream. The main class implementation of is showed in the next code.

ObjectOutputStream class

The ObjectOutputStream class is used to write primitive data types, and Java objects to an OutputStream. Only objects that support the java.io.Serializable interface can be written to streams.

Constructor

1) public ObjectOutputStream(OutputStream out) throws IOException {}	It creates an ObjectOutputStream that writes to the specified OutputStream.

Important Methods

Method	Description
1) public final void writeObject(Object obj) throws IOException {}	It writes the specified object to the ObjectOutputStream.
2) public void flush() throws IOException {}	It flushes the current output stream.
3) public void close() throws IOException {}	It closes the current output stream.

ObjectInputStream class

An ObjectInputStream deserializes objects and primitive data written using an ObjectOutputStream.

Constructor

1) public ObjectInputStream(InputStream in) throws IOException {}	It creates an ObjectInputStream that reads from the specified InputStream.

Important Methods

Method	Description
1) public final Object readObject() throws IOException, ClassNotFoundException{}	It reads an object from the input stream.
2) public void close() throws IOException {}	It closes ObjectInputStream.

Example of Java Serialization

In this example, we are going to serialize the object of ***Student*** class from above code. The writeObject() method of ObjectOutputStream class provides the functionality to serialize the object. We are saving the state of the object in the file named f.txt.

Persist.java

```
1. import java.io.*;
2. class Persist{
3.  public static void main(String args[]){
4.   try{
5.   //Creating the object
6.   Student s1 =new Student(211,"ravi");
7.   //Creating stream and writing the object
8.   FileOutputStream fout=new FileOutputStream("f.txt");
9.   ObjectOutputStream out=new ObjectOutputStream(fout);
10.  out.writeObject(s1);
11.  out.flush();
12.  //closing the stream
13.  out.close();
14.  System.out.println("success");
15.  }catch(Exception e){System.out.println(e);}
16. }
17.}
```

Output:

```
success
```

Example of Java Deserialization

Deserialization is the process of reconstructing the object from the serialized state. It is the reverse operation of serialization. Let's see an example where we are reading the data from a deserialized object.

Deserialization is the process of reconstructing the object from the serialized state. It is the reverse operation of serialization. Let's see an example where we are reading the data from a deserialized object.

Depersist.java

```
1. import java.io.*;
2. class Depersist{
3.  public static void main(String args[]){
4.   try{
5.   //Creating stream to read the object
6.   ObjectInputStream in=new ObjectInputStream(new FileInputStream("f.txt"));
7.   Student s=(Student)in.readObject();
8.   //printing the data of the serialized object
9.   System.out.println(s.id+" "+s.name);
10.  //closing the stream
11.  in.close();
12.  }catch(Exception e){System.out.println(e);}
13. }
14.}
```

Output:

```
211 ravi
```

Java Serialization with Inheritance (IS-A Relationship)

If a class implements **Serializable interface** then all its sub classes will also be serializable. Let's see the example given below:

SerializeISA.java

```
1. import java.io.Serializable;
2. class Person implements Serializable{
3.  int id;
4.  String name;
```

```
5.   Person(int id, String name) {
6.    this.id = id;
7.    this.name = name;
8.   }
9.  }
10.class Student extends Person{
11. String course;
12. int fee;
13. public Student(int id, String name, String course, int fee) {
14.  super(id,name);
15.  this.course=course;
16.  this.fee=fee;
17. }
18.}
19.public class SerializeISA
20.{
21. public static void main(String args[])
22. {
23.    try{
24.  //Creating the object
25.  Student s1 =new Student(211,"ravi","Engineering",50000);
26.  //Creating stream and writing the object
27.  FileOutputStream fout=new FileOutputStream("f.txt");
28.  ObjectOutputStream out=new ObjectOutputStream(fout);
29.  out.writeObject(s1);
30.  out.flush();
31.  //closing the stream
32.  out.close();
33.  System.out.println("success");
34.  }catch(Exception e){System.out.println(e);}
35.  try{
36.  //Creating stream to read the object
37.  ObjectInputStream in=new ObjectInputStream(new FileInputStream("f.txt"));
38.  Student s=(Student)in.readObject();
39.  //printing the data of the serialized object
40.  System.out.println(s.id+" "+s.name+" "+s.course+" "+s.fee);
41.  //closing the stream
```

```
42.  in.close();
43.  }catch(Exception e){System.out.println(e);}
44. }
45.}
```

Output:

```
success
211 ravi Engineering 50000
```

The SerializeISA class has serialized the Student class object that extends the Person class which is Serializable. Parent class properties are inherited to subclasses so if parent class is Serializable, subclass would also be.

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.

> Note: All the objects within an object must be Serializable.

Java Serialization with the static data member

If there is any static data member in a class, it will not be serialized because static is the part of class not object.

Employee.java

```
1.  class Employee implements Serializable{
2.   int id;
3.   String name;
4.   static String company="SSS IT Pvt Ltd";//it won't be serialized
5.   public Student(int id, String name) {
6.    this.id = id;
7.    this.name = name;
8.   }
9.  }
```

Java Serialization with array or collection

Rule: In case of array or collection, all the objects of array or collection must be serializable. If any object is not serialiizable, serialization will be failed.

Externalizable in java

The Externalizable interface provides the facility of writing the state of an object into a byte stream in compress format. It is not a marker interface.

The Externalizable interface provides two methods:

- **public void writeExternal(ObjectOutput out) throws IOException**
- **public void readExternal(ObjectInput in) throws IOException**

Java Transient Keyword

If you don't want to serialize any data member of a class, you can mark it as transient.

Employee.java

```
1. class Employee implements Serializable{
2.  transient int id;
3.  String name;
4.  public Student(int id, String name) {
5.   this.id = id;
6.   this.name = name;
7.  }
8. }
```

Now, id will not be serialized, so when you deserialize the object after serialization, you will not get the value of id. It will return default value always. In such case, it will return 0 because the data type of id is an integer.

Visit next page for more details.

SerialVersionUID

The serialization process at runtime associates an id with each Serializable class which is known as SerialVersionUID. It is used to verify the sender and receiver of the serialized object. The sender and receiver must be the same. To verify it, SerialVersionUID is used. The sender and receiver must have the same SerialVersionUID, otherwise, **InvalidClassException** will be thrown when you deserialize the object. We can also declare our own SerialVersionUID in the Serializable class. To do so, you need to create a field SerialVersionUID and assign a value to it. It must be of the long type with static and final. It is suggested to explicitly declare the serialVersionUID field in the class and have it private also. For example:

```
1. private static final long serialVersionUID=1L;
```

Now, the Serializable class will look like this:

Employee.java

```
1. import java.io.Serializable;
2. class Employee implements Serializable{
3.  private static final long serialVersionUID=1L;
4.  int id;
```

```
5.    String name;
6.    public Student(int id, String name) {
7.     this.id = id;
8.     this.name = name;
9.    }
10.}
```