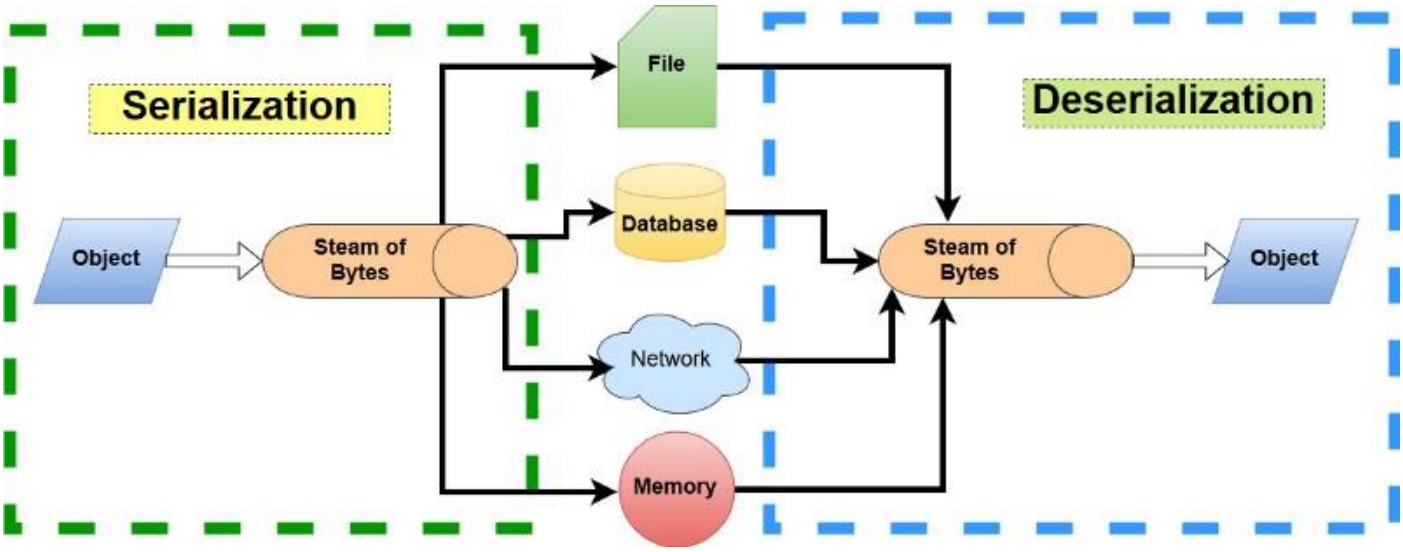
Serialization and Deserialization in Java

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.



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.

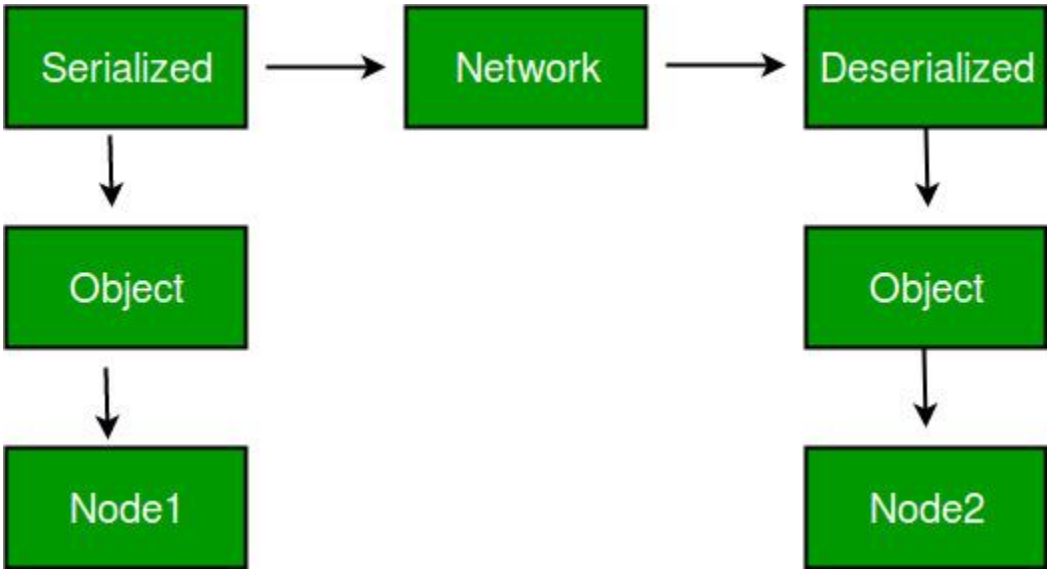
```
public final void writeObject(Object obj)
                        throws IOException
```

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.

A class must implement **Serializable** interface present in `java.io` package in order to serialize its object successfully. **Serializable** is a **marker interface** that adds serializable behaviour to the class implementing it.

Java provides **Serializable** API encapsulated under `java.io` package for serializing and deserializing objects which include,

- `java.io.serializable`
- `java.io.Externalizable`
- `ObjectInputStream`
- and `ObjectOutputStream` etc.

Marker interface

Marker Interface is a special interface in Java without any field and method. Marker interface is used to inform compiler that the class implementing it has some special behaviour or meaning. Some example of Marker interface are,

- `java.io.serializable`
- `java.lang.Cloneable`
- `java.rmi.Remote`
- `java.util.RandomAccess`

All these interfaces do not have any method and field. They only add special behaviour to the classes implementing them. However, marker interfaces have been deprecated since

Java 5, they were replaced by **Annotations**. Annotations are used in place of Marker Interface that play the exact same role as marker interfaces did before.

Signature of `writeObject()` and `readObject()`

`writeObject()` method of `ObjectOutputStream` class serializes an object and send it to the output stream.

```
public final void writeObject(object x) throws IOException
```

`readObject()` method of `ObjectInputStream` class references object out of stream and deserialize it.

```
public final Object readObject() throws IOException,ClassNotFoundException
```

while serializing if you do not want any field to be part of object state then declare it either static or transient based on your need and it will not be included during java serialization process.

INTERFACES

Every Object Stream class implements either of the two interfaces-

1. OBJECTINPUT – SUBINTERFACE OF DATAINPUT
2. OBJECTOUTPUT – SUBINTERFACE OF DATAOUTPUT

***Note*:-** *Since both the interfaces above are sub interface of interfaces Data streams interfaces, That means that all the primitive data I/O methods covered in Data Streams are also implemented in object streams.*

CLASSES FOR OBJECT STREAMS

Two classes that are used for Object Streams are –

1. OBJECTINPUTSTREAM
    - This Java class is responsible for deserializing the previously serialized objects and the primitive data.
    - It reads the object from the graph of objects stored while using FileInputStream.
    - Method **readObject()** is the main method used to deserialize the object.The class of the object, the signature of the class, and the values of the non-transient and non-static fields of the class and all of its super types are read.
        - public final Object **readObject()** throws IOException,ClassNotFoundException

2. OBJECTOUTPUTSTREAM
    - This Java class is used to store primitives and a graph of Java object which are available to ObjectInputStream to read data from.

- This does not care about saving the super class and sub class data but it could be achieved by using the writeObject method or by using the methods for primitive data types supported by DataOutput.
- Method **writeObject()** is a main method of this Java class which serializes the object directly.

    - public final void **writeObject(Object obj)** throws IOException

---

TRANSIENT KEYWORD

There are scenarios in which we want to store only a part of the object i.e. **we want to omit some specific fields from the stored object** just like password field from any user class or an Employee or you can think of any other sensitive information.

In these cases we mark these fields as transient and this keyword protects the field from being saved during the process of serialization.
Example 1– transient private String password;

Example 2

```
class studentinfo implements Serializable
{
    String name;
    transient int rid;
    static String contact;
}
```

- Making a data member `transient` will prevent its serialization.
- In this example `rid` will not be serialized because it is **transient**, and `contact` will also remain unserialized because it is **static**.

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 JVM associates a version (*long*) number with each serializable class. It is used to verify that the saved and loaded objects have the same attributes and thus are compatible on serialization.
This number can be generated automatically by most IDEs and is based on the class name, its attributes and associated access modifiers. Any changes result in a different number and can cause an *InvalidClassException*.
If a serializable class doesn't declare a *serialVersionUID*, the JVM will generate one automatically at run-time. However, it is highly recommended that each class declares its *serialVersionUID* as the generated one is compiler dependent and thus may result in unexpected *InvalidClassExceptions*.

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…]

Serializing an Object

```
import java.io.*;
class studentinfo implements Serializable
{
    String name;
    int rid;
    static String contact;
    studentinfo(string n, int r, string c)
    {
    this.name = n;
    this.rid = r;
```

```
    this.contact = c;
    }
}

class Test
{
    public static void main(String[] args)
    {
        try
        {
            Studentinfo si = new studentinfo("Abhi", 104, "110044");
            FileOutputStream fos = new FileOutputStream("student.ser");
            Objectoutputstream oos = new ObjectOutputStream(fos);
            oos.writeObject(si);
            oos.close();
            fos.close();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

Object of Studentinfo class is serialized using `writeObject()` method and written to `student.ser` file.

Deserialization of Object

```
import java.io * ;
class DeserializationTest
{
    public static void main(String[] args)
    {
        studentinfo si=null ;
        try
        {
            FileInputStream fis = new FileInputStream("student.ser");
            ObjectOutputStream ois = new ObjectOutputStream(fis);
            si = (studentinfo)ois.readObject();
        }
        catch (Exception e)
```

```
        {
            e.printStackTrace(); }
            System.out.println(si.name);
            System.out. println(si.rid);
            System.out.println(si.contact);
        }
}
```

```
Abhi
104
null
```

Contact field is null because,it was marked as static and as we have discussed earlier static fields does not get serialized.

NOTE: Static members are never serialized because they are connected to class not object of class.