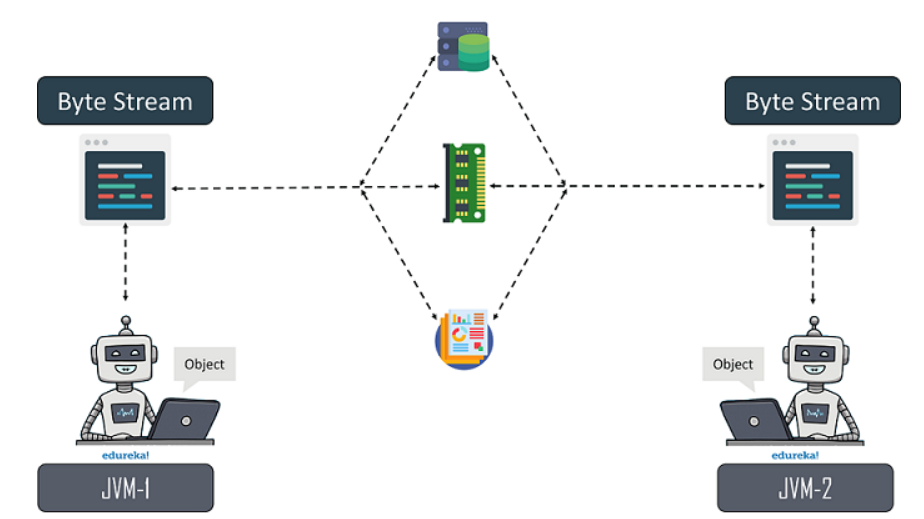
**Serialization in Java**

**What is serialization in Java?**

Serialization in java is the process of converting the java code Object into a Byte stream, to transfer object code from one java Virtual machine to another and recreate it using the process of Deserialization. It is mainly used in Hibernate, RMI, JPA, EJB and JMS technologies.



**Why do we need Serialization in Java?**

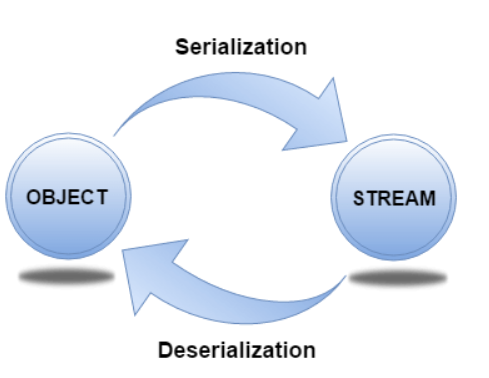
We need Serialization for the following reasons:

* **Communication**: Serialization involves the procedure of object serialization and transmission. This enables multiple computer systems to design, share and execute objects simultaneously.
* **Caching**: The time consumed in building an object is more compared to the time required for de-serializing it. Serialization minimizes time consumption by caching the giant objects.
* **Deep Copy**: Cloning process is made simple by using Serialization. An exact replica of an object is obtained by serializing the object to a byte array, and then de-serializing it.
* **Cross** **JVM Synchronization:** The major advantage of Serialization is that it works across different JVMs that might be running on different architectures or Operating Systems
* **Persistence:** The State of any object can be directly stored by applying Serialization on to it and stored in a database so that it can be retrieved later.

Advantages of Java Serialization

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

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



**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.

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

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.

// 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");

        }

    }

}

Output :

Object has been serialized

Object has been deserialized

a = 1

b = geeksforgeeks

