Serialization and Deserialization and Transient 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).

A picture containing diagram

Description automatically generated

**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:

**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.

**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. |

**Java transient Keyword**

In Java, Serialization is used to convert an object into a stream of the byte. The byte stream consists of the data of the instance as well as the type of data stored in that instance. Deserialization performs exactly opposite operation. It converts the byte sequence into original object data. During the serialization, when we do not want an object to be serialized we can use a **transient** keyword.

**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.

**When to use the transient keyword?**

1. The transient modifier can be used where there are data members derived from the other data members within the same instance of the class.
2. This transient keyword can be used with the data members which do not depict the state of the object.
3. The data members of a non-serialized object or class can use a transient modifier.

**Example of Java Transient Keyword**

Let's take an example, we have declared a class as ***Student***, it has three data members ***id, name*** and ***age***. If you serialize the object, all the values will be serialized but we don't want to serialize one value, e.g. ***age*** then we can declare the age data member as transient.

In this example, we have created two classes ***Student*** and ***PersistExample***. The ***age*** data member of the ***Student*** class is declared as transient, its value will not be serialized.

If you deserialize the object, you will get the default value for transient variable.