**OBJECT SERIALIZATION:**

**Done By:Surbhi Garg**

**Ritu Prasad**

**Venkat**

**OBJECT SERIALIZATION:**

**1.SERIALIZATION:**

Serialization is the conversion of an object to a series of bytes, so that the object can be easily saved to persistent storage or streamed across a communication link. The byte stream can then be deserialized - converted into a replica of the original 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

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

When you create a class, you may create an object for that particular class and once we execute/terminate the program, the object is destroyed by itself via the garbage collector thread.

What happens if you want to call that class without re-creating the object? In those cases, what you do is use the serialization concept by converting data into a byte stream.

Object Serialization is a process used to convert the state of an object into a byte stream, which can be persisted into disk/file or sent over the network to any other running Java virtual machine. The reverse process of creating an object from the byte stream is called deserialization. The byte stream created is platform independent. So, the object serialized on one platform can be deserialized on a different platform.

**Q.How to Make a Java Class Serializable?**

Serializability can be enabled in your Java class by implementing the java.io.Serializable interface. It is a marker interface that means it contains no methods or fields. 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 the certain capability. The Cloneable and Remote are also marker interfaces.

It must be implemented by the class whose object you want to persist.The String class and all the wrapper classes implement the java.io.Serializable interface by default

**Q.What if We Are Trying to Serialize a Non-Serializable Object?**

We will get a RuntimeException saying: Exception in thread "main" java.io.NotSerializableException: java.io.ObjectOutputStream

**Q.Why is Serialization done?**

When we want to Persist the Object. When we want the object to exist beyond the lifetime of the JVM.

2**.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 {} | writes the specified object to the ObjectOutputStream. |
| 2) public void flush() throws IOException {} | flushes the current output stream. |
| 3) public void close() throws IOException {} | closes the current output stream. |

### ObjectInputStream class

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

## NOTE: 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.

**EXAMPLE:**

1. **import** java.io.\*;
2. **class** Persist{
3. **public** **static** **void** main(String args[])**throws** Exception{
4. Student s1 =**new** Student(211,"ravi");
6. FileOutputStream fout=**new** FileOutputStream("f.txt");
7. ObjectOutputStream out=**new** ObjectOutputStream(fout);
9. out.writeObject(s1);
10. out.flush();
11. System.out.println("success");
12. }
13. }

In this example, we are going to serialize the object of Student class. 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.

**Q.What Is the serialVersionUID?**

SerialVersionUID is an ID, which is stamped on an object when it gets serialized usually with the hashcode of the object. We can find serialVersionUID for the object by the serialver tool in Java.

**3.TRANSIENT KEYWORD**

**Java transient** keyword is used in serialization. If you define any data member as transient, it will not be serialized.

The transient modifier/keyword is applicable only for variables but not for methods and classes.At the time of serialization, if we don't want to serialize the value of a particular variable to meet security constraints, then we should declare that variable as transient.While performing serialization, the JVM ignores the original value of the transient variable and save default value to the file. Hence, transient means not to serialize.

# 4.Externalizable interface in Java

Externalization serves the purpose of custom Serialization, where we can decide what to store in stream.

Externalizable interface present in java.io, is used for Externalization which extends Serializable interface. It consist of two methods which we have to override to write/read object into/from stream which are-

// to read object from stream

void readExternal(ObjectInput in)

// to write object into stream

void writeExternal(ObjectOutput out)

**Key differences between Serializable and Externalizable**

* **Implementation :** Unlike [Serializable interface](https://www.geeksforgeeks.org/serialization-in-java/)which will serialize the variables in object with just by implementing interface, here we have to explicitly mention what fields or variables you want to serialize.
* **Methods :** Serializable is marker interface without any methods. Externalizable interface contains two methods: writeExternal() and readExternal().
* **Process:** Default Serialization process will take place for classes implementing Serializable interface. Programmer defined Serialization process for classes implementing Externalizable interface.
* **Backward Compatibility and Control:** If you have to support multiple versions, you can have full control with Externalizable interface. You can support different versions of your object. If you implement Externalizable, it’s your responsibility to serialize super class.
* **public No-arg constructor:**Serializable uses reflection to construct object and does not require no arg constructor. But Externalizable requires public no-arg constructor.

// Java code for serialization and deserialization

// of a Java object

import java.io.\*;

class Emp implements Serializable {

private static final long serialversionUID =

                                 129348938L;

    transient int a;

    static int b;

    String name;

    int age;

    // Default constructor

public Emp(String name, int age, int a, int b)

    {

        this.name = name;

        this.age = age;

        this.a = a;

        this.b = b;

    }

}

public class SerialExample {

public static void printdata(Emp object1)

    {

        System.out.println("name = " + object1.name);

        System.out.println("age = " + object1.age);

        System.out.println("a = " + object1.a);

        System.out.println("b = " + object1.b);

    }

public static void main(String[] args)

    {

        Emp object = new Emp("ab", 20, 2, 1000);

        String filename = "shubham.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(object);

            out.close();

            file.close();

            System.out.println("Object has been serialized\n"

                              + "Data before Deserialization.");

            printdata(object);

            // value of static variable changed

            object.b = 2000;

        }

        catch (IOException ex) {

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

        }

        object = null;

        // Deserialization

        try {

            // Reading the object from a file

            FileInputStream file = new FileInputStream

                                         (filename);

            ObjectInputStream in = new ObjectInputStream

                                         (file);

            // Method for deserialization of object

            object = (Emp)in.readObject();

            in.close();

            file.close();

            System.out.println("Object has been deserialized\n"

                                + "Data after Deserialization.");

            printdata(object);

            // System.out.println("z = " + object1.z);

        }

        catch (IOException ex) {

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

        }

        catch (ClassNotFoundException ex) {

            System.out.println("ClassNotFoundException" +

                                " is caught");

        }

    }

}

s