Serialization and Deserialization

Uchithma Senevirathne

17.07.2024
GDSE 68

EXECUTIVE SUMMERY

In the realm of software development, the need to store and transmit data efficiently is paramount. Serialization and deserialization are critical processes that

enable these capabilities in Java. Serialization is the technique of converting an object's state into a byte stream, which allows the object to be easily saved to a file, sent over a network, or stored in a database. This byte stream preserves the object's data and structure, making it possible to recreate the original object later.

Deserialization, conversely, is the process of reconstructing an object from its serialized byte stream. This allows developers to retrieve the object's state and behavior seamlessly, facilitating data transfer and persistence.

These processes are integral to various applications, including distributed systems, remote method invocation, and data caching, thereby enhancing the efficiency and flexibility of Java applications. Understanding serialization and deserialization is essential for any developer working with data persistence and object management in Java.

TABLE OF CONTENTS

INTRODUCTION

Serialization is the process of converting an object's state into a byte stream. This byte stream can then be easily saved to a file, transmitted over a network, or stored in a database. The primary purpose of serialization is to persist the state of an object beyond the lifetime of the Java Virtual Machine (JVM) that created it. In Java, this is typically achieved by implementing the Serializable interface, which marks a class as capable of being serialized. The ObjectOutputStream class is used to write the object to an output stream, capturing its state and data.

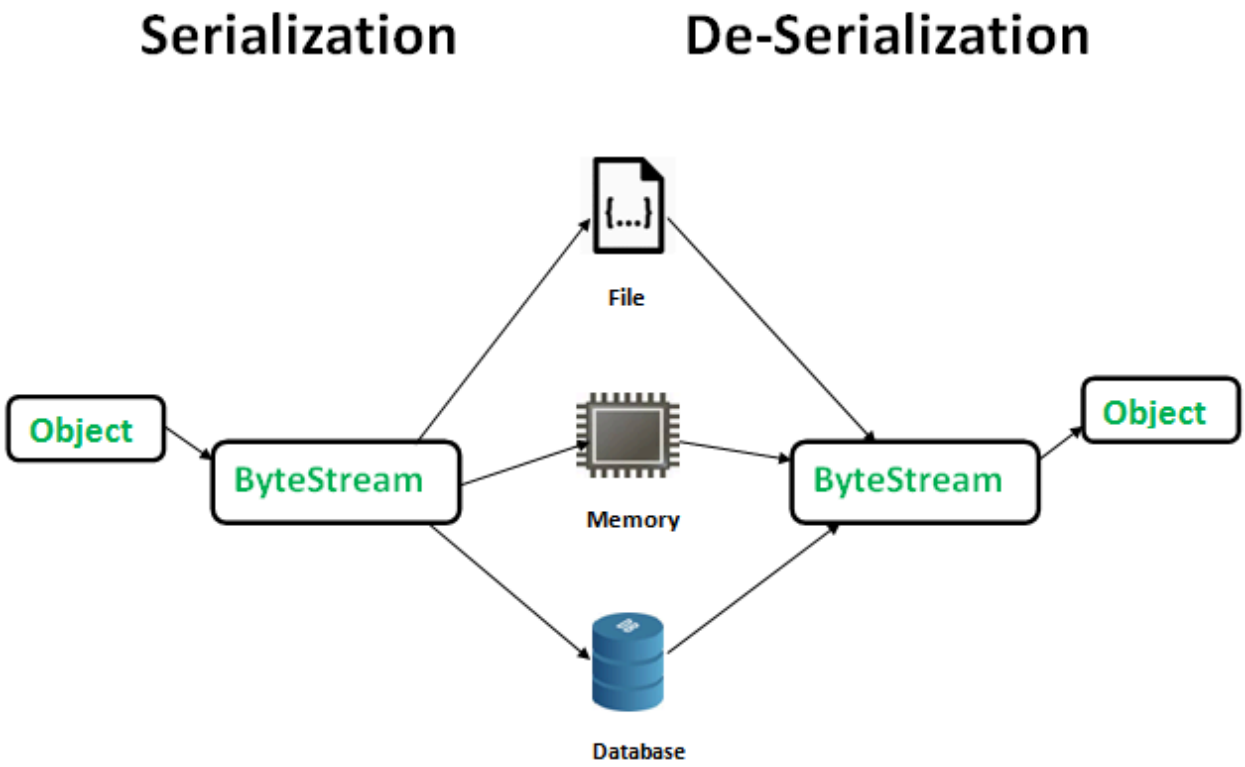
Deserialization is the reverse process of converting the serialized byte stream back into a Java object. This allows the application to reconstruct the original object from its persisted state. The ObjectInputStream class is used to read the byte stream and create the object in memory. For deserialization to work correctly, the class of the object being deserialized must be available in the current execution context.

BENEFITS

- This is a built-in feature of Java. Hence you need not use third-party services to implement Serialization.
- This concept is easy to understand.
- It is customizable as per the programmer's needs.
- This process is universal and all kinds of developers are familiar with the concept.
- This allows Java to perform Encryption, Authentication, and Compression and secure Java computing.
- Most of the technologies we use daily, rely on serialization.

(Team, D. 2024c)

MECHANISM

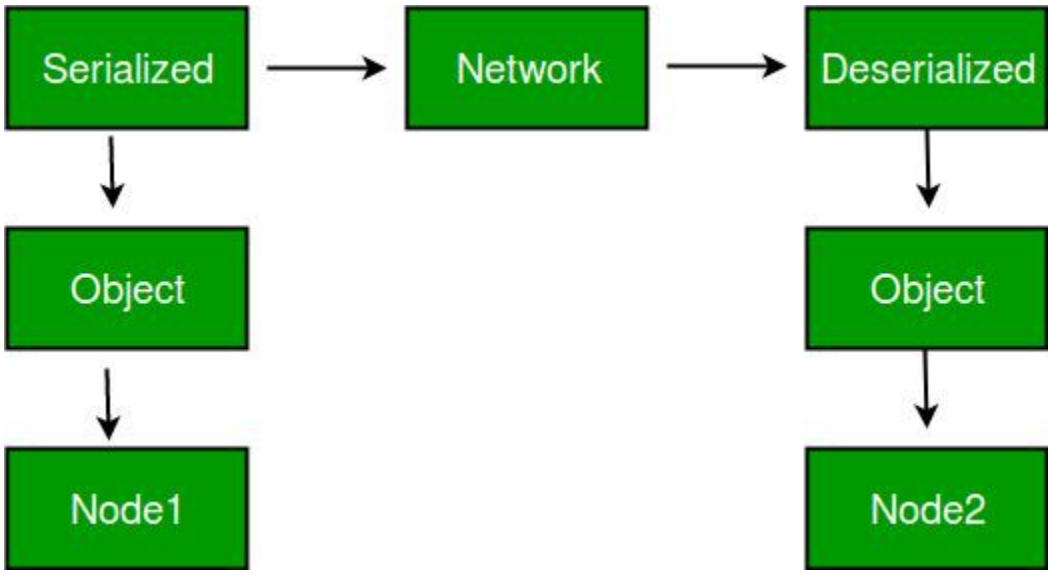


Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.The byte stream created is platform independent. So, the 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
```



(GeeksforGeeks . 2023)

Serialization

```
package org.example;

import java.io.*;

 UchithmaSenevirathne
public class SerializeDemo {
     UchithmaSenevirathne
    public static void main(String[] args) {
        Employee e = new Employee();
        e.name = "Uchithma Senevirathne";
        e.address = "Galle";
        e.NIC = 112233;
        e.number = 119;

        try {
            //create object from FileOutputStream and give the serializable type file name
            FileOutputStream fileOut = new FileOutputStream( name: "employee.ser");
            ObjectOutputStream out = new ObjectOutputStream(fileOut);
            //pass the object that we want to write
            out.writeObject(e);
            //close the FileOutputStream and ObjectOutputStream
            out.close();
            fileOut.close();

            System.out.println("serialized date is saved in employee.ser");

        }catch (IOException i){
            i.printStackTrace();
        }
    }
}
```

Employee.ser file

```
¬í □sr □org.example.EmployeeÕW□%□º v□ □I □numberL  addresst
□Ljava/lang/String;L □nameq ~ □xp   wt □Gallet □Uchithma
Senevirathne
```

Deserialization

```java
package org.example;

import java.io.*;

 UchithmaSenevirathne
public class DeserializeDemo {
     UchithmaSenevirathne
    public static void main(String[] args) {
        //create variable for assign the deserialized object
        Employee e = null;

        try {
            //create object to input ser file we want read
            FileInputStream fileIn = new FileInputStream( name: "employee.ser");
            ObjectInputStream in = new ObjectInputStream(fileIn);

            //cast the object to Employee type and assign to e
            e = (Employee) in.readObject();

            in.close();
            fileIn.close();

        } catch (IOException i){
            i.printStackTrace();
            return;
        }catch (ClassNotFoundException c){
            System.out.println("employee class not found");
            c.printStackTrace();
            return;
        }
        System.out.println("Deserialized Employee...");
        System.out.println("name : " + e.name);
        System.out.println("address : " + e.address);
        System.out.println("ssn : " + e.NIC);
        System.out.println("number : " + e.number);
    }
}
```

Deserialized Output:

```
Deserialized Employee...
name : Uchithma Senevirathne
address : Galle
ssn : 0
number : 119

Process finished with exit code 0
```

CONCLUSION

Serialization and deserialization are vital for efficient data management in Java applications. Understanding these processes allows developers to implement robust data storage and communication mechanisms, enhancing the overall functionality and performance of their software.

REFERENCES

Team, D. (2024c) *Serialization in Java - Deserialization in Java*, *DataFlair*. Available at: https://data-flair.training/blogs/serialization-and-deserialization-in-java/ (Accessed: 17 July 2024).

GeeksforGeeks (2023) *Serialization and deserialization in Java with example*, *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/serialization-in-java/ (Accessed: 17 July 2024).