BSc (Hons) in Computer Science via GDSE

Module Code ITS1114

Advanced API Development

Technical description based on Research & Development

(Serialization & Deserialization)

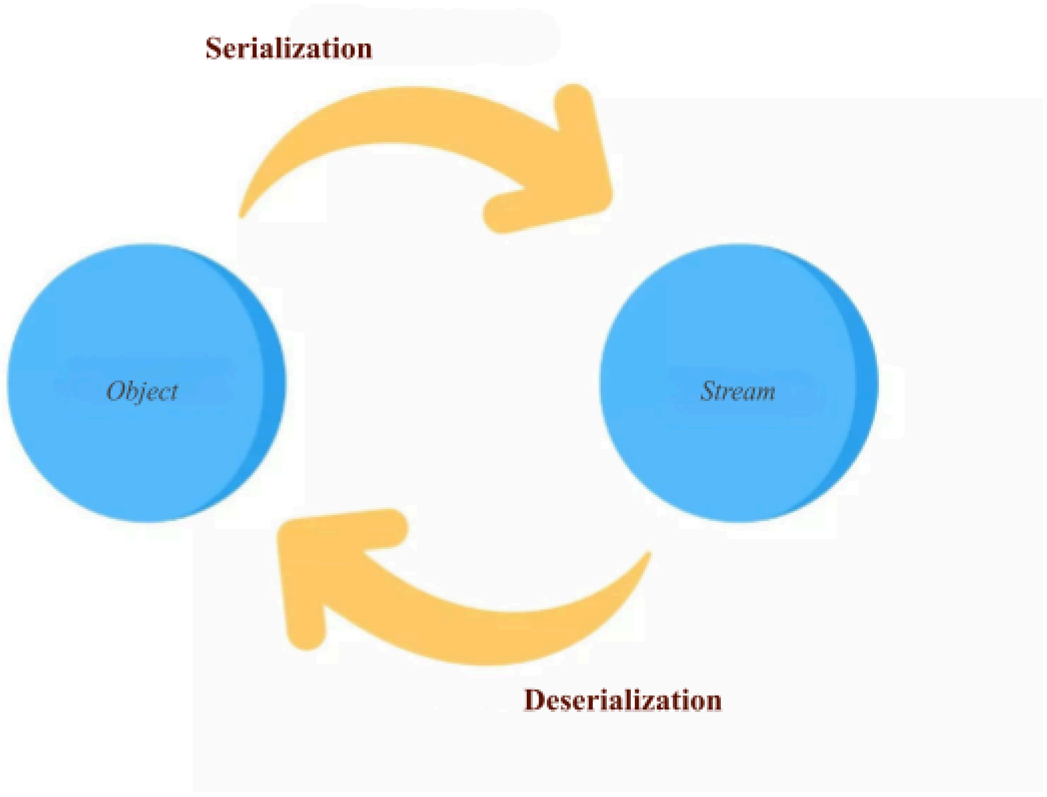
S. W. CHARMIE VIHARA

2301682063

GDSE 68

19 / 07 / 2024

EXECUTIVE SUMMARY



Serialization and deserialization are crucial processes in software engineering, facilitating the conversion of complex data structures into a format that can be easily stored, transmitted, and reconstructed. Serialization transforms objects into a byte stream, enabling data storage in files, databases, or transmission over networks.

Deserialization is the reverse process, converting the byte stream back into the original data structures, preserving the object's state and structure. These processes are essential for data persistence, communication between systems, and various applications like distributed computing, data exchange in web services, and caching mechanisms. Efficient serialization and deserialization ensure data integrity, interoperability, and optimized performance across diverse computing environments.

TABLE OF CONTENT

INTRODUCTION

Serialization

Serialization is the process of converting an object's state into a byte stream. This is a format that can be stored or transmitted and then reconstructed later. This process typically involves transforming the data within the object into a byte stream or other data format (such as JSON or XML) that can be easily saved to a file, sent over a network, or stored in a database. Serialization is essential for tasks like data persistence, communication between different parts of a system, and enabling interoperability between different systems and programming languages. It ensures that complex data structures can be effectively and accurately preserved and recreated as needed.

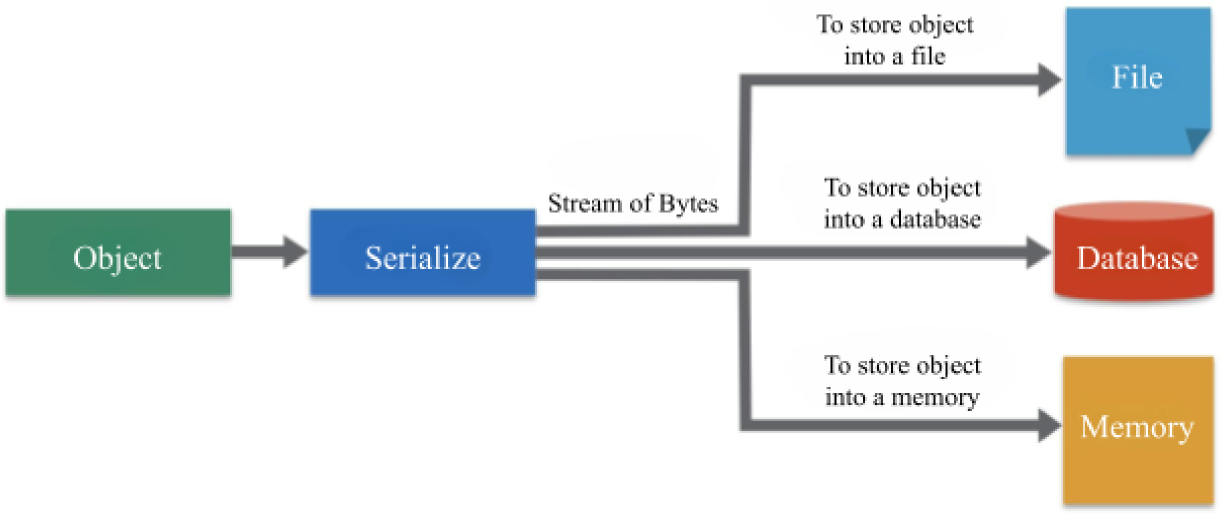

Object
Serialize
Stream of Bytes
To store object into a file
File
To store object into a database
Database
To store object into a memory
Memory

Figure 1 : Serialization Process

Deserialization

Deserialization is the process of converting a byte stream or a data format again into a java object. This byte stream can be JSON, XML, or binary and also java object can be a complex data structure or object that can be used by a program. This process essentially reconstructs the original state and structure of the data that was previously serialized. Deserialization is fundamental for applications that require data to be stored or transmitted and then later reconstructed, such as reading data from files, databases, or receiving data over a network. It ensures that the data integrity and structure are preserved, allowing the program to correctly interpret and manipulate the information as needed.

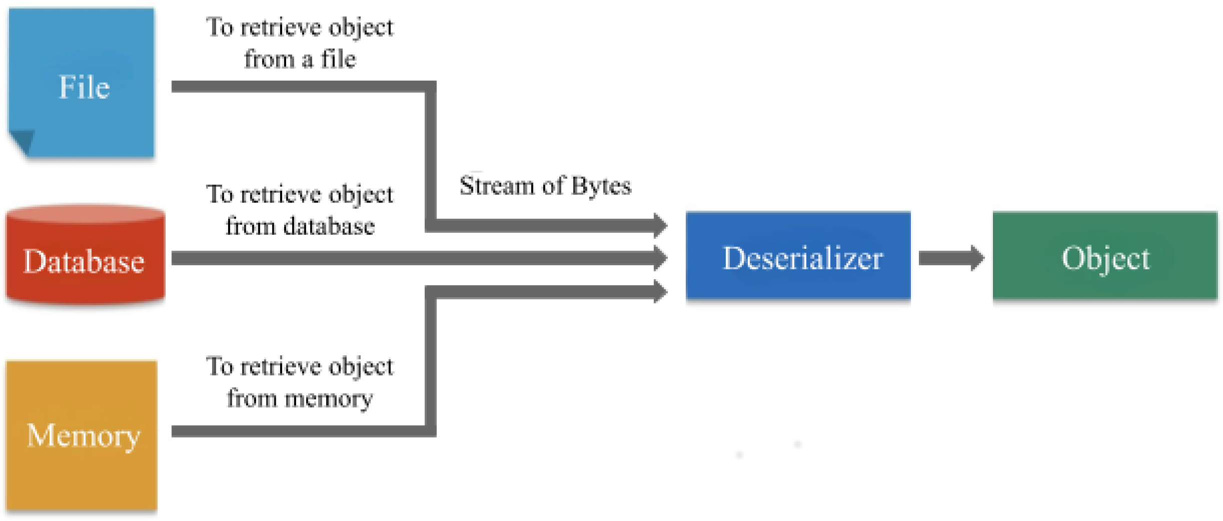

File
To retrieve object from a file
Database
To retrieve object from database
Stream of Bytes
Deserializer
Object
Memory
To retrieve object from memory

Figure 2 : Deserialization Process

ADVANTAGES & DISADVANTAGES

Advantages of Serialization & Deserialization

- Serialization is a built-in feature of Java, so you don't need third-party services to implement it.
- The concept is easy to understand.
- It is customizable according to the programmer's needs.
- This process is universal, and all kinds of developers are familiar with it.
- It enables Java to perform encryption, authentication, and compression, securing Java computing.
- Most of the technologies we use daily rely on serialization.

Disadvantages of Serialization & Deserialization

- Occasionally, byte streams fail to fully convert into objects, resulting in errors.
- Declaring a variable as transient causes the compiler to allocate memory for it, but the class constructor is not invoked, leading to deviations from the standard Java flow.
- This process can be inefficient in terms of memory utilization.
- Serialization is not suitable for applications requiring concurrent access without third-party APIs, as it lacks a transition control mechanism.
- It does not provide fine-grained control when accessing objects.

(DataFlair Web Services Pvt Ltd, n.d.)

MECHANISM

Mechanism of Serialization & Deserialization

Serialization is the process of converting the state of an object into a byte stream. Deserialization, on the other hand, reverses this process by reconstructing the actual Java object in memory from the byte stream. This mechanism is essential for persisting objects, enabling them to be stored in files, transmitted over networks, or stored in databases while maintaining their state and structure.

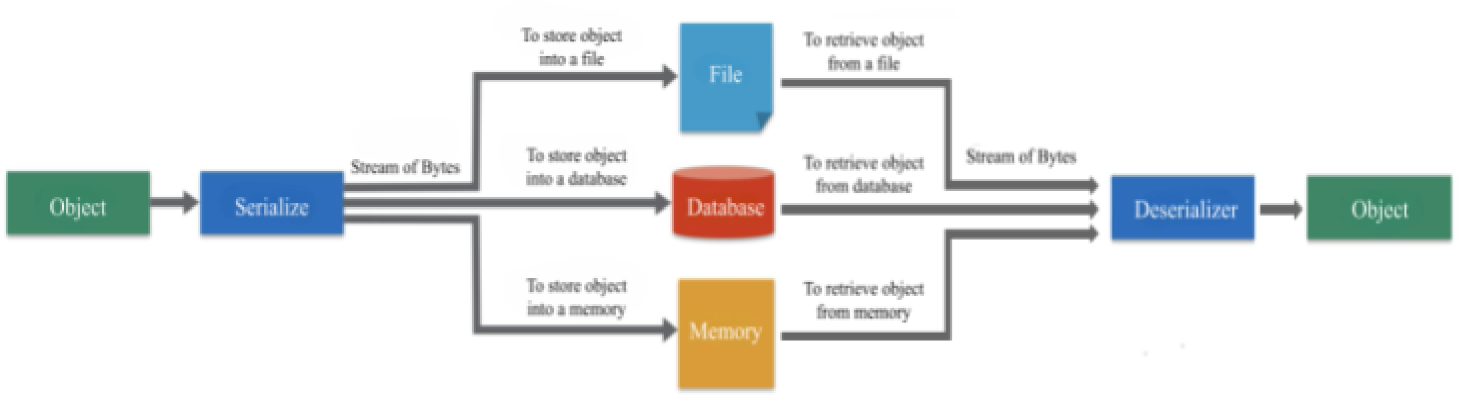

Object
Serialize
Stream of Bytes
To store object into a file
To store object into a database
To store object into a memory
File
Database
Memory
To retrieve object from a file
To retrieve object from database
To retrieve object from memory
Stream of Bytes
Deserializer
Object

Figure 3 : Full process in Serialization & Deserialization

The byte stream created during serialization is indeed platform-independent, allowing an object serialized on one platform to be deserialized on a different platform seamlessly. In Java, to make an object serializable, we implement the `java.io.Serializable` interface. The `ObjectOutputStream` class provides the `writeObject()` method, which is used for serializing an object, converting its state into a byte stream that can be stored or transmitted. This ensures that the object's state and structure are preserved and can be reconstructed accurately during deserialization on any compatible platform.

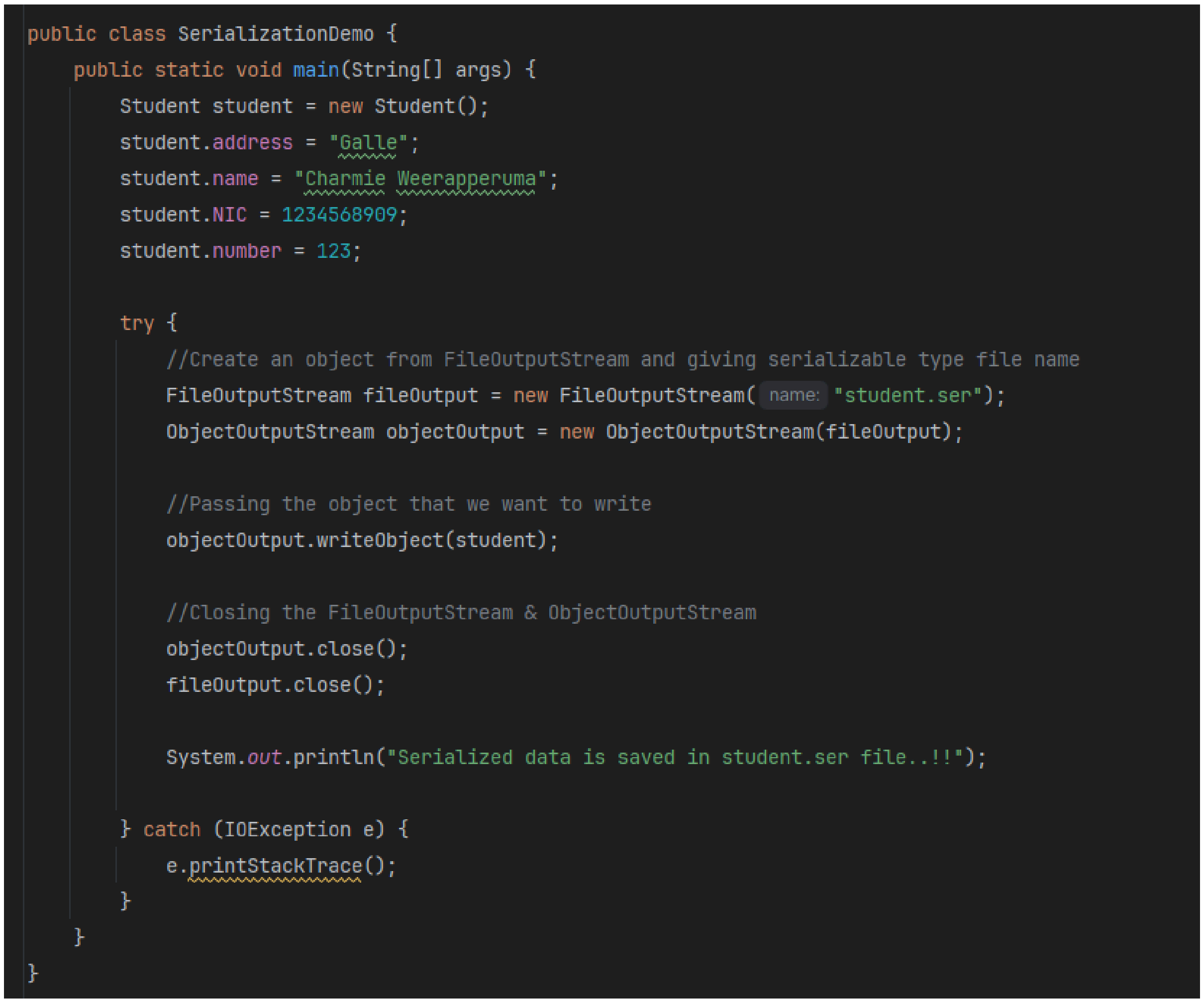
```
public final void writeObject (Object object)
                                Throws IOException
```

In Java, the `ObjectInputStream` class contains the `readObject()` method, which is used for deserializing an object. This method reads a byte stream from a source (like a file or network socket) and reconstructs the serialized object back into its original form in memory. Together with `ObjectOutputStream`'s `writeObject()` method for serialization, `ObjectInputStream`'s `readObject()` completes the cycle of converting objects to byte streams for storage or transmission and then restoring them back to their original Java objects.

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

(Jain and Sanchhaya Education Private Limited 2023)

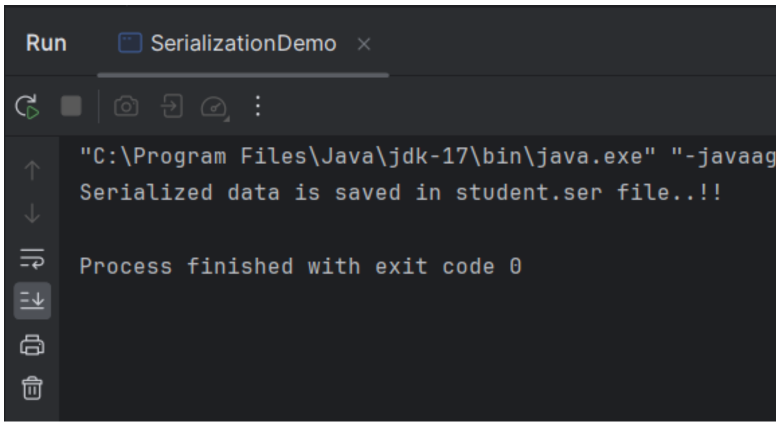
Serialization Example Code



```java
public class SerializationDemo {
    public static void main(String[] args) {
        Student student = new Student();
        student.address = "Galle";
        student.name = "Charmie Weerapperuma";
        student.NIC = 1234568909;
        student.number = 123;

        try {
            //Create an object from FileOutputStream and giving serializable type file name
            FileOutputStream fileOutput = new FileOutputStream( name: "student.ser");
            ObjectOutputStream objectOutput = new ObjectOutputStream(fileOutput);

            //Passing the object that we want to write
            objectOutput.writeObject(student);

            //Closing the FileOutputStream & ObjectOutputStream
            objectOutput.close();
            fileOutput.close();

            System.out.println("Serialized data is saved in student.ser file..!!");

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

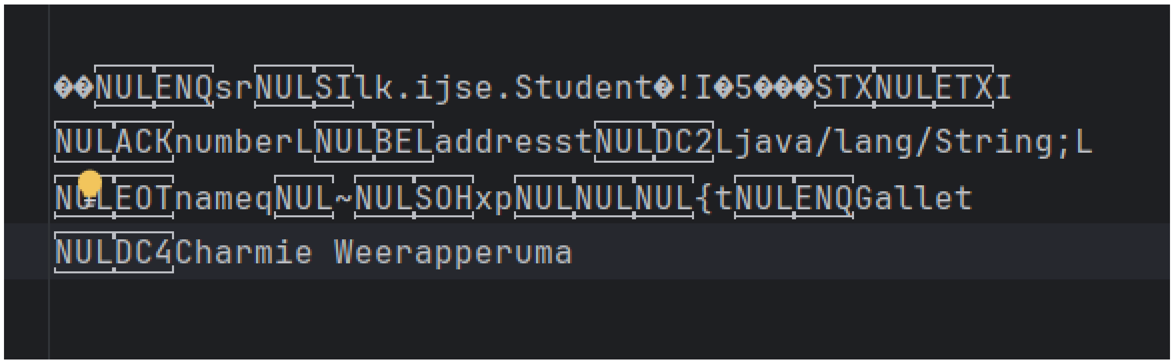
Figure 4 : Serialization code

Serialization Output



```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaag
Serialized data is saved in student.ser file..!!

Process finished with exit code 0
```

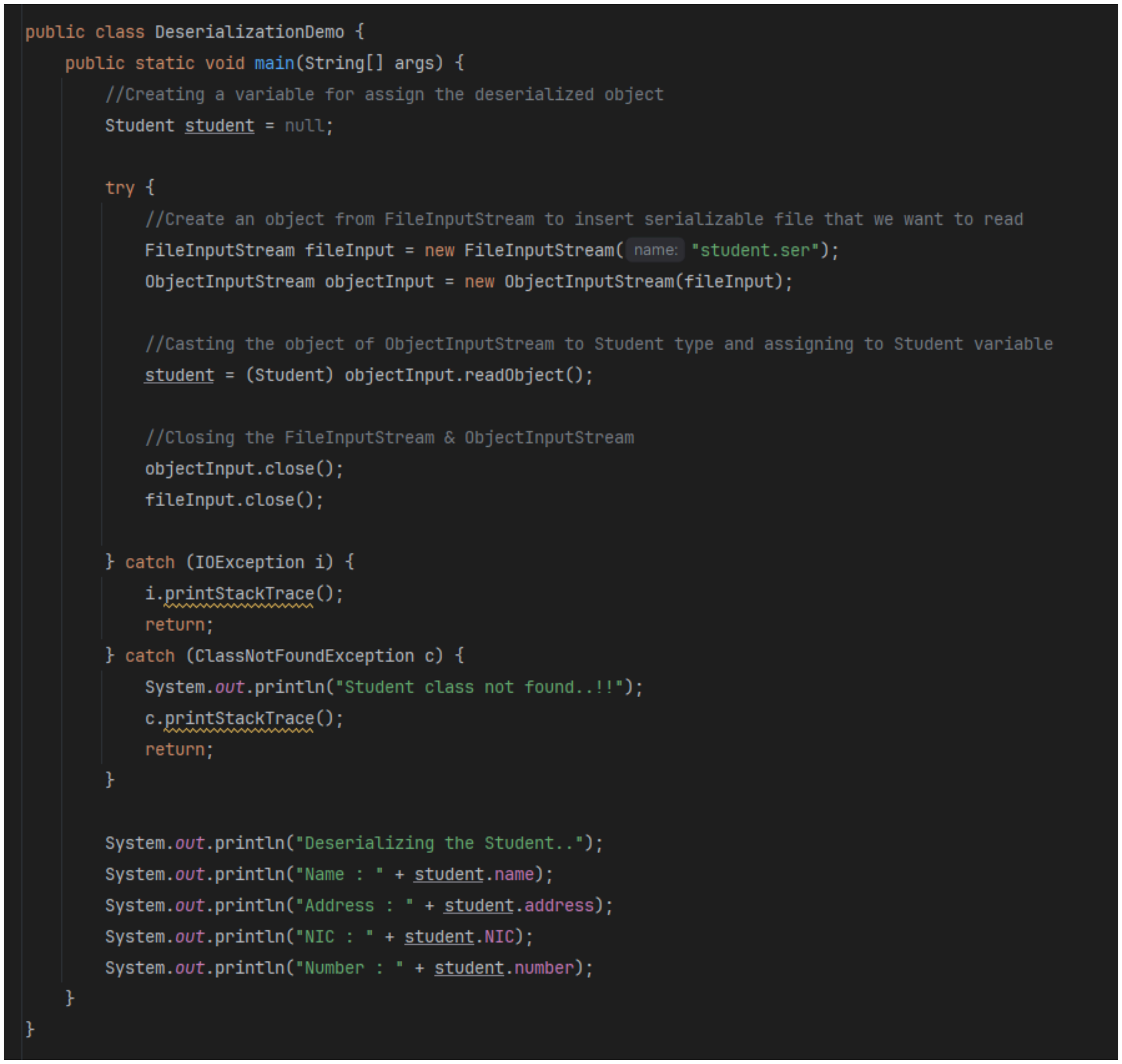
Figure 5 : Serialization Output



```
��NULENQsrNULSIlk.ijse.Student�!I�5���STXNULETXI
NULACKnumberLNULBELaddresstNULDC2Ljava/lang/String;L
NULEOTnameqNUL~NULSOHxpNULNULNUL{tNULENQGallet
NULDC4Charmie Weerapperuma
```

Figure 6 : Serializable file (student.ser)

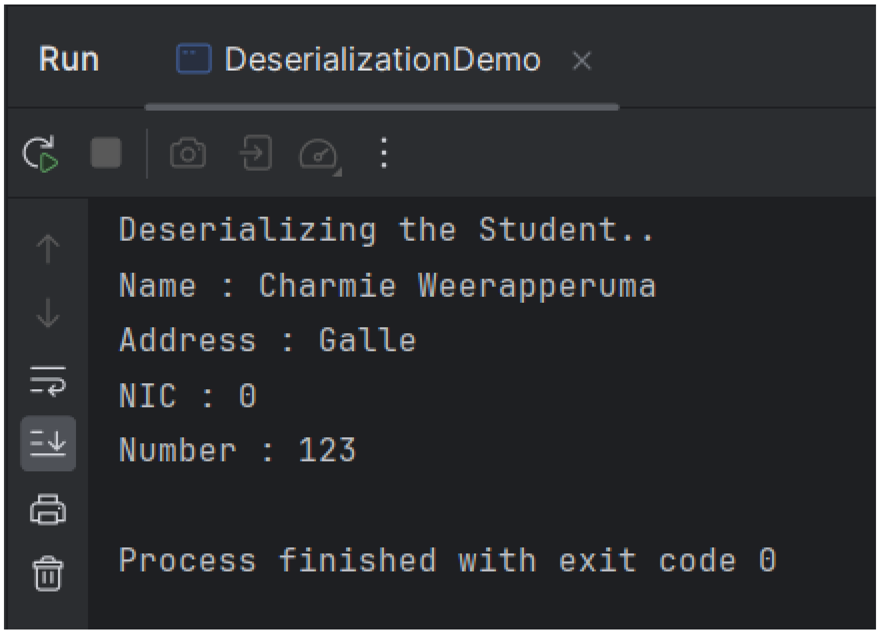
Deserialization Example Code



```java
public class DeserializationDemo {
    public static void main(String[] args) {
        //Creating a variable for assign the deserialized object
        Student student = null;

        try {
            //Create an object from FileInputStream to insert serializable file that we want to read
            FileInputStream fileInput = new FileInputStream( name: "student.ser");
            ObjectInputStream objectInput = new ObjectInputStream(fileInput);

            //Casting the object of ObjectInputStream to Student type and assigning to Student variable
            student = (Student) objectInput.readObject();

            //Closing the FileInputStream & ObjectInputStream
            objectInput.close();
            fileInput.close();

        } catch (IOException i) {
            i.printStackTrace();
            return;
        } catch (ClassNotFoundException c) {
            System.out.println("Student class not found..!!");
            c.printStackTrace();
            return;
        }

        System.out.println("Deserializing the Student..");
        System.out.println("Name : " + student.name);
        System.out.println("Address : " + student.address);
        System.out.println("NIC : " + student.NIC);
        System.out.println("Number : " + student.number);
    }
}
```

Figure 7 : Deserialization code

Deserialization Output



```
Deserializing the Student..
Name : Charmie Weerapperuma
Address : Galle
NIC : 0
Number : 123

Process finished with exit code 0
```

Figure 8 : Deserialization output

CONCLUSION

In conclusion, serialization and deserialization are fundamental processes in modern software development. They enable the efficient storage, transmission, and reconstruction of complex data structures and objects across different platforms and environments.

Serialization converts the state of an object into a byte stream, making it portable and persistent, while deserialization reconstructs the object from the byte stream, ensuring data integrity and usability.

These mechanisms not only facilitate data exchange and persistence but also support essential features like caching, distributed computing, and interoperability between diverse systems. As technologies evolve, serialization and deserialization remain indispensable tools for maintaining data consistency, efficiency, and flexibility in software applications.

REFERENCES

DataFlair Web Services Pvt Ltd. n.d. “Serialization in Java - Deserialization in Java.” DataFlair. Accessed July 18, 2024. https://data-flair.training/blogs/serialization-and-deserialization-in-java/.

Jain, Sandeep, and Sanchhaya Education Private Limited. 2023. “Serialization and Deserialization in Java with Example.” GeeksforGeeks. https://www.geeksforgeeks.org/serialization-in-java/.