ITS1114 - Advanced API Development

Serialization and Deserialization

Name : pavithrani perera

Student Id : 2301671041

Badge: GDSE -67

Table Of content:

1. Introduction

Serialization and Deserialization are processes used to convert an object into a byte stream and vice versa. This is essential for persisting objects, sending objects over a network, or deep cloning.

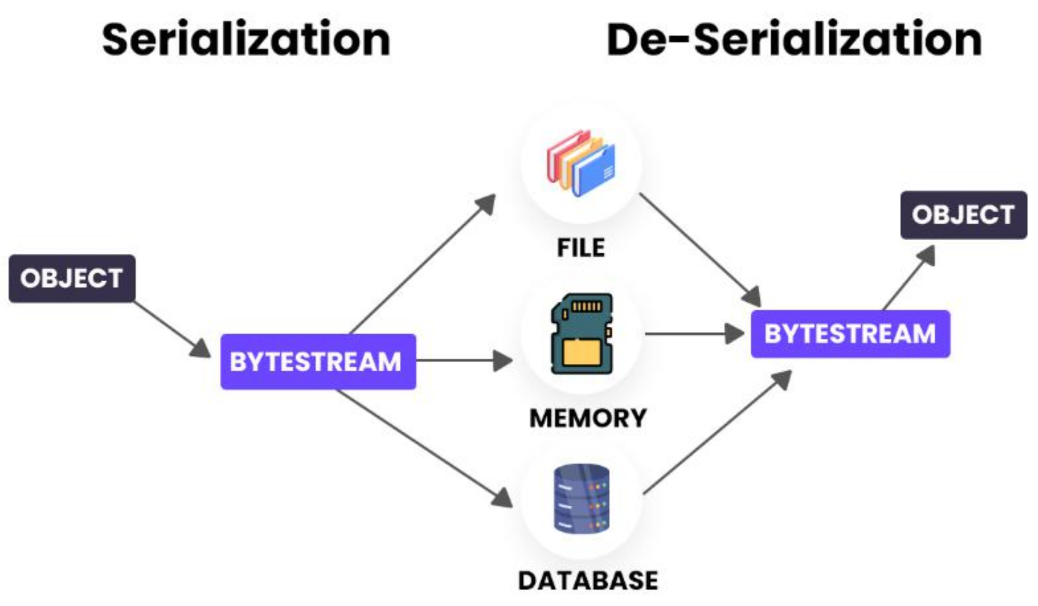

Figure 1 : Introduction

2. Benifits

- Persistence: Serialize objects to store them on disk, allowing them to be retrieved and used later.

- Communication: Serialize objects to send them over a network between different Java applications.

- Cloning: Deep clone objects by serializing and then deserializing them.

3. Mechanism

Serialization: Converts an object into a byte stream.
Implement Serializable interface.
Use ObjectOutputStream to write the object to a byte stream.

Deserialization: Converts a byte stream back into an object.
Use ObjectInputStream to read the object from the byte stream.

3.1 Example Scenario

Consider a scenario where we need to save the state of a User object to a file and retrieve it later.

GitHub Repository

You can find the complete code and additional examples in the following GitHub repository:
https://github.com/pavithraniperera/Serialization-and-Deserialization

```
import java.io.*;

// Define the User class
class User implements Serializable {
      private String name;
      private int age;

   // Constructor
   public User(String name, int age) {
      this.name = name;
      this.age = age;
   }

   // Getters and toString method
   public String getName() {
      return name;
   }

   public int getAge() {
      return age;
   }

   @Override
   public String toString() {
      return "User{name='" + name + "', age=" + age + "}";
   }
}

public class SerializationExample {
   public static void main(String[] args) {
      User user = new User("Alice", 30);

      // Serialization
      try (ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream("user.ser"))) {
         out.writeObject(user);
         System.out.println("User has been serialized: " + user);
      } catch (IOException e) {
         e.printStackTrace();
      }

      // Deserialization
      try (ObjectInputStream in = new ObjectInputStream(new
FileInputStream("user.ser"))) {
         User deserializedUser = (User) in.readObject();
         System.out.println("User has been deserialized: " + deserializedUser);
      } catch (IOException | ClassNotFoundException e) {
         e.printStackTrace();
      }
   }
}
```

Explanation

User Class: Implements Serializable interface, indicating it can be serialized.

Serialization: The user object is written to a file named user.ser using ObjectOutputStream.

Deserialization: The user object is read back from the file using ObjectInputStream.

4. Difference Between Serialization and Deserialization in Java.

Serialization	**Deserialization**
Serialization is the mechanism of conversion of an object to a stream of bytes.	Deserialization helps to convert the stream of objects to the original state of the object.
It helps to **write** byte stream to file,db, etc.	It helps to **read** byte stream from file, db, etc.
It is performed with the help of the ObjectOutputStream Class.	It is performed with the help of the ObjectInputStream class.

References

Baeldung, (n.d.). Java Serialization Basics.
Baeldung. Available at: https://www.baeldung.com/java-serialization
[Accessed 17 July 2024].

GeeksforGeeks, (n.d.). Serialization and Deserialization in Java with Example.
GeeksforGeeks. Available at: https://www.geeksforgeeks.org/serialization-in-java/
[Accessed 17 July 2024].

Tutorialspoint, (n.d.). Java Serialization.
https://www.tutorialspoint.com/java/java_serialization.htm
[Accessed 17 July 2024].