JULY 18, 2024

SERIALIZATION & DESERIALIZATION PROCESS

CHAMITH KAVINDA
GDSE 68
2301682028

Table of Content

Table of Figures

Introduction

In the Computer Science, Data Management and Application Development , the Concept of Serialization and Deserialization role in the efficient handling, storage and transmission of data. **Serialization** is a process of converting complex data structures or objects into a format that can be easily stored or transmitted as a byte stream or a human readable format like JSON or XML. **Deserialization** is the reverse process, where the stored or transmitted data is reconstructed back into its original complex structure or object form.

A Java object is serializable if its class or any of its subclasses implements java.io.Serializable or its sub interface java.io.Externalizable interface. The entire process is JVM independent meaning an object can be serialized on one platform and deserialized on an entirely different platform. Classes ObjectInputStream and ObjectOutputStream are high-level streams that contain the methods for serializing and deserializing an object.

Benefits

- Serialization allows objects to be converted into a format easily stored on disk, database.
- Inter-process Communication.
- Web services and APIs.
- Standardized formats.
- Optimized Data Handling.
- Streaming and lazy loading.
- Backward and forward compatibility.
- Simplified Data Structures.
- Readable Formats.

Serialization Mechanism

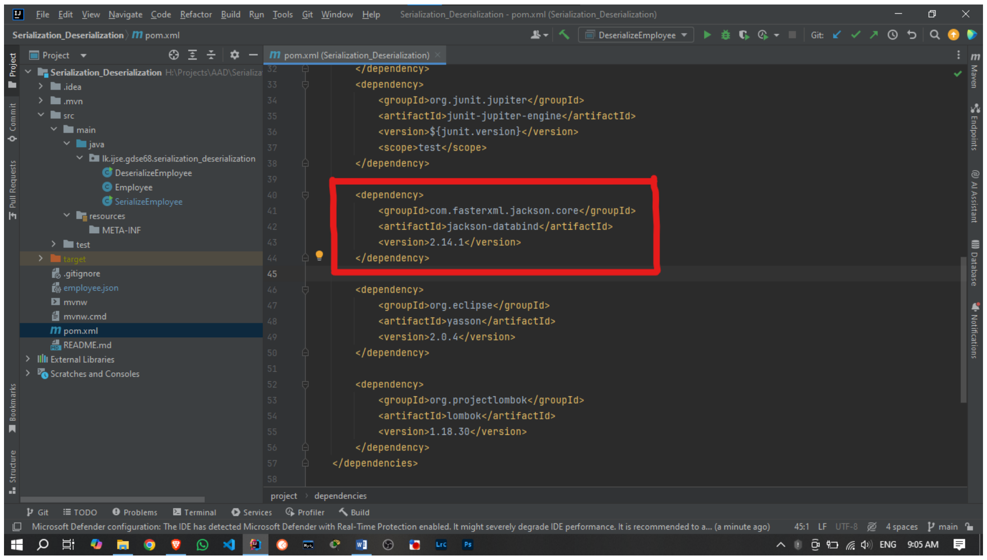
1. Process starts with the traversal of the object graph.
2. The collected data converted into a specific format. Commonly JSON, XML, binary, custom formats.
3. In JSON, XML serialization object's data is converted into a text-based format.
4. Handling the structures such as objects, arrays and collections.
5. Serialize follows a specific protocol to data is encoded correctly.
6. Final serialized data is generated as a string or a byte stream.

Deserialization Mechanism

1. This process begins with reading the serialized data form (Source, file, network stream, database).
2. Identify data types and structure.
3. Reconstructs the object graph by creating instances of the original objects and setting their fields to the parsed values.
4. Uses type information to instantiate the correct classes and set their fields.
5. Errors handling such as missing fields.
6. Final output fully reconstructed object.

Code Example

1. Add Jackson Dependency

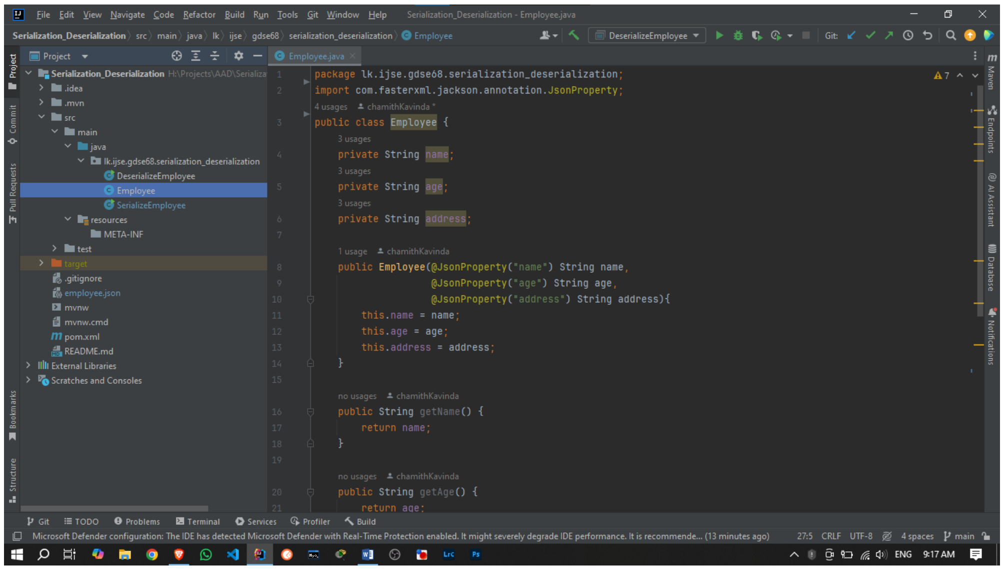


```
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.14.1</version>
        </dependency>

        <dependency>
            <groupId>org.eclipse</groupId>
            <artifactId>yasson</artifactId>
            <version>2.0.4</version>
        </dependency>

        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.18.30</version>
        </dependency>
    </dependencies>
```

Figure 1 : Add Jackson Dependency

2. Define a Employee Class

Create an "Employee" class with fields, annotation for JSON Process.

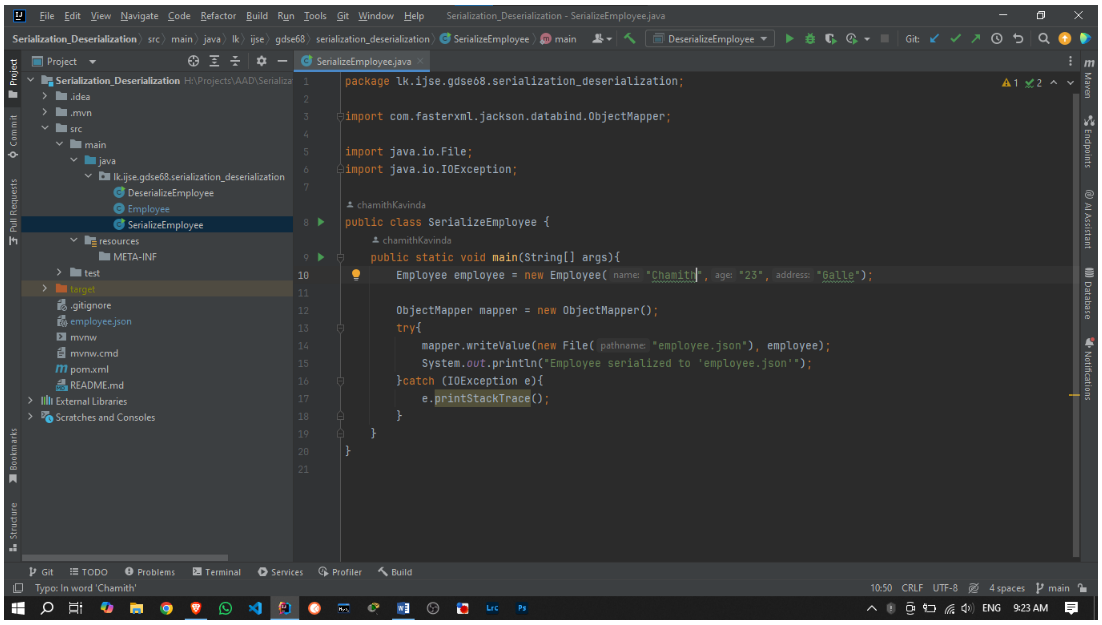


```
package lk.ijse.gdse68.serialization_deserialization;
import com.fasterxml.jackson.annotation.JsonProperty;
public class Employee {
    private String name;
    private String age;
    private String address;

    public Employee(@JsonProperty("name") String name,
                    @JsonProperty("age") String age,
                    @JsonProperty("address") String address){
        this.name = name;
        this.age = age;
        this.address = address;
    }

    public String getName() {
        return name;
    }

    public String getAge() {
        return age;
```

Figure 2 : Define a Employee Class

3. Serialize an Employee Object

   Create a "SerializeEmployee" class to serialize an "Employee" object to JSON.

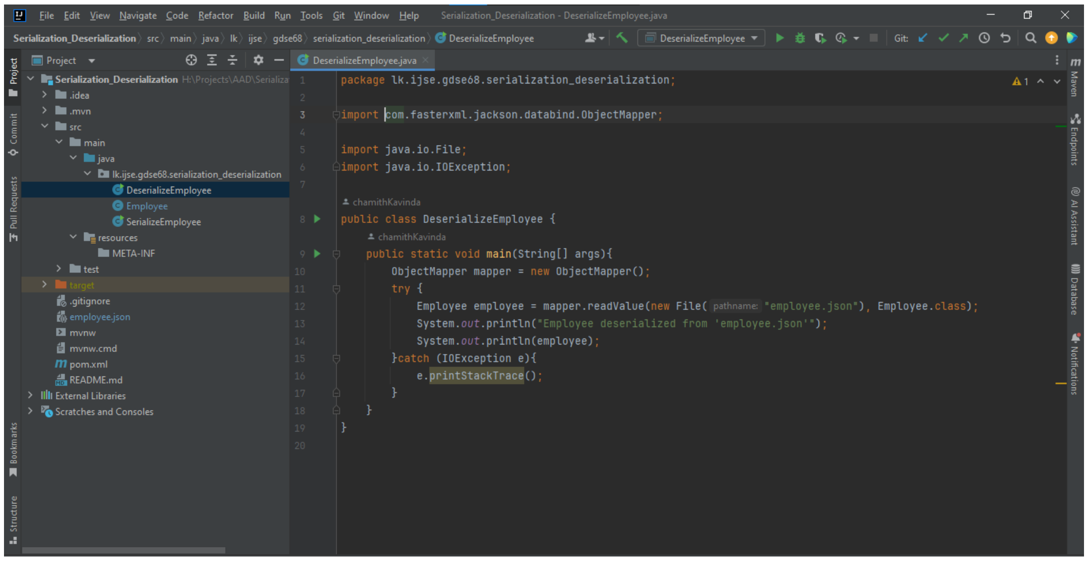


```java
package lk.ijse.gdse68.serialization_deserialization;

import com.fasterxml.jackson.databind.ObjectMapper;

import java.io.File;
import java.io.IOException;

public class SerializeEmployee {
    public static void main(String[] args){
        Employee employee = new Employee("Chamith", "23", "Galle");

        ObjectMapper mapper = new ObjectMapper();
        try{
            mapper.writeValue(new File("employee.json"), employee);
            System.out.println("Employee serialized to 'employee.json'");
        }catch (IOException e){
            e.printStackTrace();
        }
    }
}
```

Figure 3 : Serialize an Employee Object

4. Deserialize an Employee Object

   Create a "DeserializeEmployee" class to deserialize an "Employee" object from JSON.

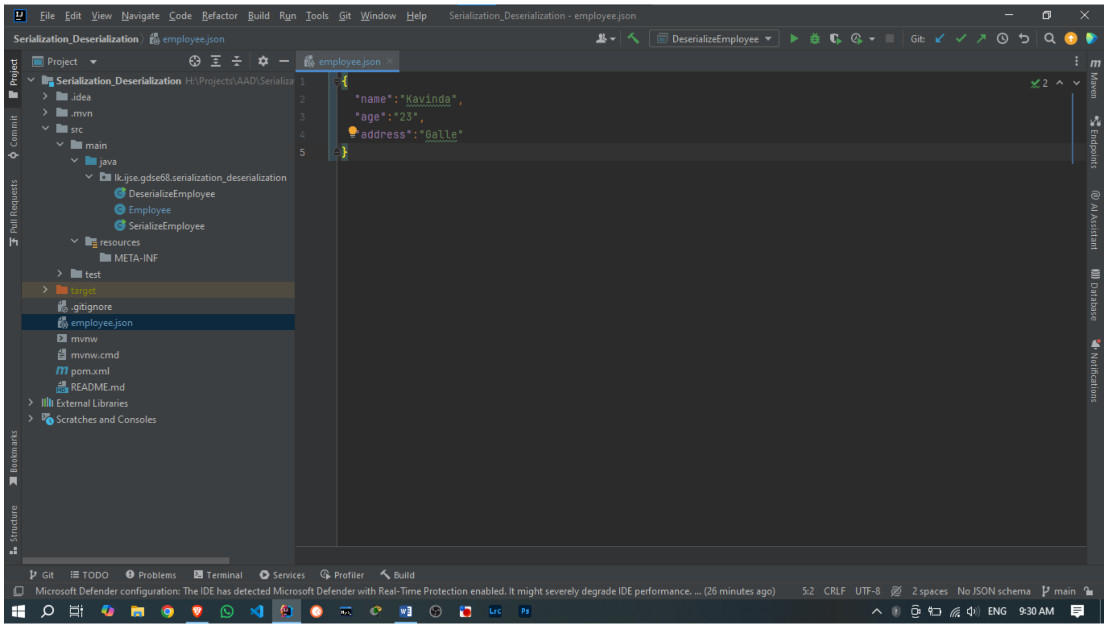


```java
package lk.ijse.gdse68.serialization_deserialization;

import com.fasterxml.jackson.databind.ObjectMapper;

import java.io.File;
import java.io.IOException;

public class DeserializeEmployee {
    public static void main(String[] args){
        ObjectMapper mapper = new ObjectMapper();
        try {
            Employee employee = mapper.readValue(new File("employee.json"), Employee.class);
            System.out.println("Employee deserialized from 'employee.json'");
            System.out.println(employee);
        }catch (IOException e){
            e.printStackTrace();
        }
    }
}
```

Figure 4 : Deserialize an Employee Object

5. JSON Output

After run the "SerializeEmployee" Class, it creates an "employee.json" file.



```
{
  "name":"Kavinda",
  "age":"23",
  "address":"Galle"
}
```

```
Microsoft Defender configuration: The IDE has detected Microsoft Defender with Real-Time Protection enabled. It might severely degrade IDE performance. ... (26 minutes ago)
```

Figure 5 : JSON Output

Check GitHub Repository for More Details:-

https://github.com/chamithKavinda/Serialization_Deserialization_Process

Conclusion

We have to use Jackson Library for fulfil serialization & deserialization process. This process is essential for data persistence and communication among systems. That data structures can be easily stored and retrieved.

References

https://www.geeksforgeeks.org/serialization-in-java/