Serialization and Deserialization in Java

Student's Name

Department

Institutional Affiliation

Course Name and Number

Due Date

Serialization and Deserialization in Java

Serialization is the conversion of the state of an object into a byte stream and deserialization is the other way around [1]. It's necessary when we want to save our objects to a database and then retrieve them from it.

Let's see an example:

```
public class SomeObject implements Serializable {
    private int someInteger;
    private String someString;
    private boolean someBoolean;

    //constructors, getters&setters, toString
}
```

We have some object with several non-static fields that we want to serialize. In this case, the object in question must implement a Serializable marker interface. If we want to ignore some unnecessary fields during the serialization, we need to use a transient keyword:

```
private transient boolean someBoolean;
```

Note: Static fields are not serialized because they belong to a class, not an object.

Let's create a class responsible for serialization and deserialization:

```
public class Serializator {

    public boolean serialize(SomeObject someObject){

        File file = new
File("C:\\Users\\invis\\Desktop\\save.data");

        try (FileOutputStream fos = new FileOutputStream(file);
             ObjectOutputStream oos = new
ObjectOutputStream(fos)){

            oos.writeObject(someObject);
            return true;
        }
        catch(IOException e){
            e.printStackTrace();
        }
        return false;
    }

    public SomeObject deserialize() throws
```

```
InvalidObjectException {
        File file = new
File("C:\\Users\\invis\\Desktop\\save.data");

        try (FileInputStream fis = new FileInputStream(file);
             ObjectInputStream ois = new
ObjectInputStream(fis)){

            return (SomeObject) ois.readObject();
        }
        catch (IOException | ClassNotFoundException e){
            e.printStackTrace();
        }

        throw new InvalidObjectException("Object is missing!");
    }
}
```

The serialize(SomeObject someObject) method

First, we need to create a File object with the path where our object will be saved. Then, we declare and initialize the ObjectOutputStream object in try-resources for saving the state of the object to a file using FileOutputStream. Try-with-resources is used to auto-close the streams [2]. After that, we can use the writeObject(object) method inside the try block to write our object to the file. This method throws an IOException, so it's necessary to catch this one. If serialization is successful, we return "true, otherwise false". It's all done, so let's test it:

```
    Serializator serializator = new Serializator();
SomeObject someObjectToSerialize = new SomeObject(1, "someStr",
true);
serializator.serialize(someObjectToSerialize);
```

After executing this block of code, the new file will be in the specified location. We can use this file to deserialize the object in the future.

The deserialize() method

The deserializing process is very similar to serializing. In the serialize method, we used output streams, but now, we are doing the same but with input streams. Then, inside the try block, we return the deserialized object using the readObject() method with a type cast, otherwise the InvalidObjectException will be thrown. According to this, the method signature

should have “throws InvalidObjectException”. Imagine we have a serialized object and we need to retrieve this one:

```
     SomeObject someObjectToDeserialize = null;

try {
    someObjectToDeserialize = serializator.deserialize();
} catch (InvalidObjectException e) {
    e.printStackTrace();
}
```

We have some object initialized to null, then inside the try block, we call the deserialize method to retrieve the object. Try block is needed because there is an exception in the method signature that can be thrown, and if there is an error, we must catch it. The result of the execution has been assigned to the object variable. So, in other words, we have deserialized the state of the object, and now we can continue working with this object:

```
SomeObject{someInteger=1, someString='someStr', someBoolean=true}
Successful serialization!

Successful deserialization!
SomeObject{someInteger=1, someString='someStr', someBoolean=true}
```

References

Baeldung (2021). *Introduction to Java Serialization.* Baeldung. https://www.baeldung.com/java-serialization

Baeldung (2022). *Java – Try with Resources.* Baeldung. https://www.baeldung.com/java-try-with-resources