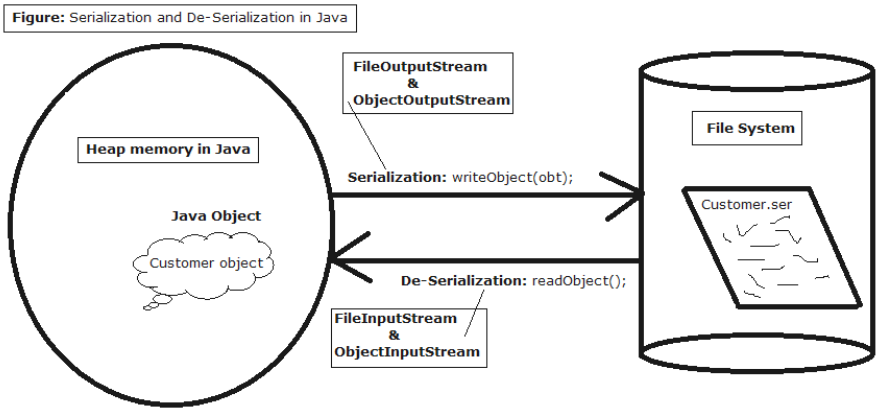
Serializable interface

November 2, 2016 SJ Serialization 0

In this article, we will discuss the important things we should know about *java.io.Serializable* interface in detail

Serializable interface:

- Present in *java.io* package
- Fully qualified class name is *java.io.Serializable*
- It is a *Marker interface* which means a Java class implementing marker interface has got certain capability
- It has no body i.e.; it doesn't contain any methods
- We can serialize, only serializable objects
- An Object said to be Serializable, if its corresponding class implements *java.io.Serializable* interface
- Serializing a non-serializable object (i.e.; class not implementing *java.io.Serializable* interface), then run time exception will be thrown stating *NotSerializableException*



Spring JDBC: An example on JdbcTemplate using Annotation
Java JDBC: An example to connect MS Access database
Oracle OSB 12c: Service Callout and Routing Table example
Oracle OSB 12c: Hello World service with both Business and Proxy Service

Demo example on Java Serialization & De-Serialization

For objects to participate in *serialization & de-serialization process*, corresponding class should *implement java.io.Serializable* interface

Exception: otherwise, run time exception will be thrown stating *NotSerializableException*

Step 1: Create POJO which implements *java.io.Serializable* interface

Customer.java

- Customer class is the one to be ***serialized***
- Therefore, it is must to implement ***java.io.Serializable*** interface
- Consists of ***3 member variables*** namely
- Two integer member (customer id and customer age) and String member (customer name)

```
1  package in.bench.resources.serialization;
2
3  import java.io.Serializable;
4
5  public class Customer implements Serializable {
6
7      // member variables
8      int customerId;
9      String customerName;
10     int customerAge;
11
12     // 3-arg parametrized constructor
13     public Customer(int customerId,
```

```
            String customerName, int customerAg
        super();
        this.customerId = customerId;
        this.customerName = customerName;
        this.customerAge = customerAge;
    }

    // overriding toString() method
    @Override
    public String toString() {
        return "Customer [customerId=" + custom
                + " customerName=" + customerNa
                + " customerAge=" + customerAge
                + "]";
    }
}
```

Step 2: Serialization and De-Serialization together in one class

To Serialize: any Object, we can use *ObjectOutputStream* & *FileOutputStream* to *write/save* to the *file* (in binary format)

To De-Serialize: any Object, we can use *ObjectInputStream* & *FileInputStream* to *read/restore* from *file* (which is in binary format) into Java *heap memory*

CustomerSerializeDeSerializeDemo.java

```
package in.bench.resources.serialization;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

public class CustomerSerializeDeSerializeDemo {

    public static void main(String[] args) {

        // create an customer object using 3-ar
        Customer serializeCustomer = new Custom

        // creating output stream variables
        FileOutputStream fos = null;
        ObjectOutputStream oos = null;

        // creating input stream variables
        FileInputStream fis = null;
        ObjectInputStream ois = null;

        // creating customer object reference
        // to hold values after de-serializatio
        Customer deSerializeCustomer = null;

```

```
        try {
            // for writing or saving binary dat
            fos = new FileOutputStream("Custome

            // converting java-object to binary
            oos = new ObjectOutputStream(fos);

            // writing or saving customer objec
            oos.writeObject(serializeCustomer);
            oos.flush();
            oos.close();

            System.out.println("Serialization:
                    + "Customer object saved to

            // reading binary data
            fis = new FileInputStream("Customer

            // converting binary-data to java-o
            ois = new ObjectInputStream(fis);

            // reading object's value and casti
            deSerializeCustomer = (Customer) oi
            ois.close();

            System.out.println("De-Serializatio
                    + "de-serialized from Custo
        }
        catch (FileNotFoundException fnfex) {
            fnfex.printStackTrace();
        }
        catch (IOException ioex) {
            ioex.printStackTrace();
        }
        catch (ClassNotFoundException ccex) {
            ccex.printStackTrace();
        }

        // printing customer object to console
        System.out.println("Printing customer v
                + "de-serialized object... \n"
    }
}
```

Output:

```
Serialization: Customer object saved to Custom

De-Serialization: Customer object de-serialized

Printing customer values from de-serialized obje
Customer [customerId=102, customerName=SR, custo
```

ObjectOutputStream and ObjectInputStream

ObjectOutputStream: An ObjectOutputStream writes primitive data types and graphs of Java objects to an OutputStream

ObjectInputStream: An ObjectInputStream deserializes primitive data and objects previously written using an ObjectOutputStream

ObjectOutputStream and *ObjectInputStream* can provide an application with persistent storage for graphs of objects when used with a *FileOutputStream* and *FileInputStream* respectively

Important points about Serialization and De-Serialization:

- A java object can be ***persisted*** into File storage only if its corresponding class implements ***java.io.Serializable*** interface
- Though ***java.io.Serializable*** is a ***Marker interface*** which contains no body (i.e.; no methods)
- But at run time JVM provides ***special capability*** to serialize an Object
- Using ***writeObject(Object)*** method of ObjectOutputStream, we can persist Object's state to file storage
- And similarly using ***readObject()*** method of ObjectInputStream, we can ***read or restore*** Object's state into Java heap memory from persistent storage (like file)
- If the class ***doesn't*** implement ***java.io.Serializable*** interface and ***still we try to*** serialize an Object, then program ***compiles*** successfully
- But, at ***run time*** JVM ***throws*** unchecked exception stating ***NotSerializableException***

References:

https://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html
https://docs.oracle.com/javase/7/docs/platform/serialization/spec/serial-arch.html
https://docs.oracle.com/javase/7/docs/api/java/io/ObjectOutputStream.html

https://docs.oracle.com/javase/7/docs/api/java/io/ObjectInputStream.html
https://docs.oracle.com/javase/7/docs/api/java/io/FileOutputStream.html
https://docs.oracle.com/javase/7/docs/api/java/io/FileInputStream.html

Read Also:

- Java Serialization and De-Serialization Tutorial Index
- Serialization and De-Serialization in Java
- Transient keyword with Serialization in Java
- Transient keyword with static variable in Serialization
- Transient keyword with final variable in Serialization
- Serializing a variable with transient modifier or keyword
- Order of Serialization and De-Serialization
- Serialization with Aggregation
- Serialization with Inheritance
- Externalizable interface with example
- Serializable v/s Externalizable
- Importance of SerialVersionUID in Serialization
- Singleton Design pattern with Serialization
- How to stop Serialization in Java
- How to construct a singleton class in a multi-threaded environment in Java
- How to serialize and de-serialize ArrayList in Java

Happy Coding !!
Happy Learning !!

« Transient keyword with Serialization in Java

Serialization and De-Serialization in Java »

Related Posts:

1. **Serialization and De-Serialization in Java**
2. **Transient keyword with Serialization in Java**
3. **Transient keyword with static variable in Serialization**
4. **Serializing a variable with transient modifier or keyword**

DESERIALIZATION JAVA SERIALIZABLE SERIALIZATION