Serialization with Aggregation

November 6, 2016 SJ Serialization 0

In this article, we will discuss *Serialization with Aggregation* i.e.; serializing class contains reference to other classes. It forms a *HAS-A relationship*

There are *2 scenarios* with respect to *HAS-A relationship*

1. *All reference classes*/objects inside a serializing class\object is *serializable*
2. *One or some of the reference classes*/objects inside a serializing class\object is *NOT serializable*

Here, serializing class must implement *java.io.Serializable*

Serialization process

During serialization process i.e.; saving the state of an Object to File, only instance variables will be participated and persisted to file storage or some other storage via network capability

De-Serialization process

During de-serialization process, Object's state will be restored back from file storage to java heap memory

Let's us discuss ***serialization with aggregation*** with 2 demo program

Serialization with Aggregation

Step 1: Create 2 POJO classes for **Address** and **Phone** –> implementing ***java.io.Serializable*** interface

- For any class said to be serializable, if it implement ***java.io.Serializable*** interface
- Otherwise, ***NotSerializableException*** will be thrown at run time, although ***program compiles successfully***
- Both Address & Phone POJO has 2-arg parameterized constructor
- Overrides ***toString()*** method to print values

Address.java

```
package in.bench.resources.serialization.aggre

import java.io.Serializable;

class Address implements Serializable {

    // instance variables
    int flatNo;
    String streetName;

    // 2-arg parameterized constructor
    public Address(int flatNo, String streetNam
        super();
        this.flatNo = flatNo;
        this.streetName = streetName;
    }

    // overriding toString() method
```

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

BenchR
G+ Fo
88 follower

BenchResou
Like Page 27
Be the first of your friends to l

```
    @Override
    public String toString() {
        return "Address [flatNo=" + flatNo
                + ", streetName=" + streetName
    }
}
```

Phone.java

```
package in.bench.resources.serialization.aggre?j

import java.io.Serializable;

class Phone implements Serializable {

    // instance variables
    int countryCode;
    int telephoneNumber;

    // 2-arg parameterized constructor
    public Phone(int countryCode, int telephone
        super();
        this.countryCode = countryCode;
        this.telephoneNumber = telephoneNumber;
    }

    // overriding toString() method
    @Override
    public String toString() {
        return "Phone [countryCode=" + countryC
                + ", telephoneNumber=" + teleph
    }
}
```

Step 2: Create another POJO class called **Customer** which will have reference to both *Address* and *Phone* classes

That is, Customer class aggregates both Address and Phone classes (*HAS-A relationship*)

- For any class said to be serializable, if it implement ***java.io.Serializable*** interface
- Otherwise, ***NotSerializableException*** will be thrown at run time, although ***program compiles successfully***
- Customer POJO has 4-arg parameterized constructor which includes both Address and Phone classes
- Overrides ***toString()*** method to print values

Customer.java

```java
package in.bench.resources.serialization.aggreg

import java.io.Serializable;

class Customer implements Serializable {

    // instance variables
    int customerId;
    String customerName;
    Address address;
    Phone phone;

    // 4-arg parameterized constructor
    public Customer(int customerId, String cust
            Address address, Phone phone) {
        super();
        this.customerId = customerId;
        this.customerName = customerName;
        this.address = address;
        this.phone = phone;
    }

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

As we are ready with POJOs, we will begin with our ***serialization and de-serialization process*** from main class

Step 3: Serialization and De-Serialization (with Aggregation)

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***

Case 1: When *all reference classes/objects* inside Customer class

is *serializable*

Here, both aggregating classes ***Address and Phone is serializable*** and ***main class Customer*** which has reference to Address and Phone is also ***serializable***

SerializationWithAggregation.java

```java
package in.bench.resources.serialization.aggre

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 SerializationWithAggregation {

    public static void main(String[] args) {

        // creating address object --&gt; imple
        Address address = new Address(402, "2nd

        // creating phone object --&gt; impleme
        Phone phone = new Phone(022, 27759868);

        // creating customer object --&gt; impl
        Customer serializeCustomer =
                new Customer(101, "SJ", address

        // time to play with Serialization and

        // 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:
                    + "saved to CustomerAggrega

            // 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
CustomerAggregation.ser file

Printing customer values from de-serialized obje
Customer [customerId=101, customerName=SJ,
address=Address [flatNo=402, streetName=2nd stre
phone=Phone [countryCode=18, telephoneNumber=277
```

Case 2: When *one or some of the reference classes*/objects inside Customer class is NOT *serializable*

For demo purpose we will remove "*implements Serializable*" from *Address class*

Exception: All classes inside Customer class should be serializable, otherwise at run time *NotSerializableException* will be thrown, although **program compiles successfully**

Here, Address class ***doesn't*** implement *java.io.Serializable* interface

Address.java

```java
package in.bench.resources.serialization.aggre?j

class Address {

    // instance variables
    int flatNo;
    String streetName;

    // 2-arg parameterized constructor
    public Address(int flatNo, String streetNam
        super();
        this.flatNo = flatNo;
        this.streetName = streetName;
    }

    // overriding toString() method
    @Override
    public String toString() {
        return "Address [flatNo=" + flatNo
                + ", streetName=" + streetName
    }
}
```

Note: This program is very same, as that of ***program 1*** or ***case 1***

SerializationWithAggregation.java

```java
package in.bench.resources.serialization.aggre?j

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 SerializationWithAggregation {

    public static void main(String[] args) {

        // creating address object --&gt; imple
        Address address = new Address(402, "2nd

        // creating phone object --&gt; impleme
        Phone phone = new Phone(022, 27759868);

        // creating customer object --&gt; impl
        Customer serializeCustomer =
                new Customer(101, "SJ", address

        // time to play with Serialization and

        // 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:
                    + "saved to CustomerAggrega

            // 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:

```
java.io.NotSerializableException: in.bench.re
.aggregation.Address
    at java.io.ObjectOutputStream.writeObject0(
    at java.io.ObjectOutputStream.defaultWriteF
    at java.io.ObjectOutputStream.writeSerialDa
    at java.io.ObjectOutputStream.writeOrdinary
    at java.io.ObjectOutputStream.writeObject0(
    at java.io.ObjectOutputStream.writeObject(U
    at in.bench.resources.serialization.aggrega
.SerializationWithAggregation.main(Serializatio
Printing customer values from de-serialized obj
null
```

Explanation:

- JVM throws ***NotSerializableException*** for Address class, while serializing Customer class
- So, it's very must ***for every class inside*** Serializing class to implement ***java.io.Serializable***

Case-study: try for Phone class by removing implements Serializable but before that rectify above exception by implementing serializable for Address class

Important points to remember while Serialization with Aggregation classes:

- **Rule 1:** all classes that need to be serialized must implement ***java.io.Serializable*** interface
- **Rule 2:** All reference classes inside a serializable class must be ***java.io.Serializable***
- **Rule 3:** If any of the class is not implementing ***java.io.Serializable*** in the serialization process, then JVM will throw ***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/FileOutputS

tream.html
https://docs.oracle.com/javase/7/docs/api/java/io/FileInputStr
eam.html
http://docs.oracle.com/javase/specs/jls/se7/html/jls-
8.html#jls-8.3.1.3

Read Also:

- Java Serialization and De-Serialization Tutorial Index
- Serialization and De-Serialization in Java
- Serializable interface
- 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 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 !!

« Serialization with Inheritance

Order of Serialization and De-Serialization »

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. **Order of Serialization and De-Serialization**

AGGREGATION DESERIALIZATION JAVA

SERIALIZABLE SERIALIZATION