

BenchResources.Net

Java, Collection, JDBC, Spring, Web Services, Maven, Android, Oracle SOA-OSB & Open Source

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

SEARCH TUTORIALS

SUBSCRIBE VIA EMAIL

Join 194 other subscribers

POPULAR ARTICLES

JDBC: An example to connect MS Access database in Java 8

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

```
1 package in.bench.resources.serialization.aggregation;
2
3 import java.io.Serializable;
4
5 class Address implements Serializable {
6     // instance variables
7     int flatNo;
8     String streetName;
9
10    // 2-arg parameterized constructor
11    public Address(int flatNo, String streetName) {
12        super();
13        this.flatNo = flatNo;
14        this.streetName = streetName;
15    }
16
17    // 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+ Fc

88 follower



Be the first of your friends to l

```

19     @Override
20     public String toString() {
21         return "Address [flatNo=" + flatNo
22             + ", streetName=" + streetName
23     }
24 }

```

Phone.java

```

1  package in.bench.resources.serialization.aggr;
2
3  import java.io.Serializable;
4
5  class Phone implements Serializable {
6
7      // instance variables
8      int countryCode;
9      int telephoneNumber;
10
11     // 2-arg parameterized constructor
12     public Phone(int countryCode, int telephone
13         super();
14         this.countryCode = countryCode;
15         this.telephoneNumber = telephoneNumber;
16     }
17
18     // overriding toString() method
19     @Override
20     public String toString() {
21         return "Phone [countryCode=" + countryCode
22             + ", telephoneNumber=" + telephoneNumber
23     }
24 }

```

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

■

```

1  package in.bench.resources.serialization.aggreg
2
3  import java.io.Serializable;
4
5  class Customer implements Serializable {
6
7      // instance variables
8      int customerId;
9      String customerName;
10     Address address;
11     Phone phone;
12
13     // 4-arg parameterized constructor
14     public Customer(int customerId, String cust
15                     Address address, Phone phone) {
16         super();
17         this.customerId = customerId;
18         this.customerName = customerName;
19         this.address = address;
20         this.phone = phone;
21     }
22
23     // overriding toString() method
24     @Override
25     public String toString() {
26         return "Customer [customerId=" + custom
27               + ", customerName=" + customerN
28               + ", address=" + address
29               + ", phone=" + phone + "]\n";
30     }
31 }

```

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

```
1  package in.bench.resources.serialization.aggrega
2
3  import java.io.FileInputStream;
4  import java.io.FileNotFoundException;
5  import java.io.FileOutputStream;
6  import java.io.IOException;
7  import java.io.ObjectInputStream;
8  import java.io.ObjectOutputStream;
9
10 public class SerializationWithAggregation {
11
12     public static void main(String[] args) {
13
14         // creating address object --> imple
15         Address address = new Address(402, "2nd
16
17         // creating phone object --> impleme
18         Phone phone = new Phone(022, 27759868);
19
20         // creating customer object --> impl
21         Customer serializeCustomer =
22             new Customer(101, "SJ", address
23
24         // time to play with Serialization and
25
26         // creating output stream variables
27         FileOutputStream fos = null;
28         ObjectOutputStream oos = null;
29
30         // creating input stream variables
31         FileInputStream fis = null;
32         ObjectInputStream ois = null;
33
34         // creating customer object reference
35         // to hold values after de-serializatio
36         Customer deSerializeCustomer = null;
37
38         try {
39             // for writing or saving binary dat
40             fos = new FileOutputStream("Custome
41
42             // converting java-object to binary
43             oos = new ObjectOutputStream(fos);
44
45             // writing or saving customer objec
46             oos.writeObject(serializeCustomer);
47             oos.flush();
48             oos.close();
49
50             System.out.println("Serialization:
51                 + "saved to CustomerAggrega
52
```

```

53         // reading binary data
54         fis = new FileInputStream("Customer
55
56         // converting binary-data to java-o
57         ois = new ObjectInputStream(fis);
58
59         // reading object's value and casti
60         deSerializeCustomer = (Customer) oi
61         ois.close();
62
63         System.out.println("De-Serializatio
64             + "de-serialized from Custo
65     }
66     catch (FileNotFoundException fnfex) {
67         fnfex.printStackTrace();
68     }
69     catch (IOException ioex) {
70         ioex.printStackTrace();
71     }
72     catch (ClassNotFoundException ccex) {
73         ccex.printStackTrace();
74     }
75
76     // printing customer object to console
77     System.out.println("Printing customer v
78         + "de-serialized object... \n"
79 }
80 }

```

Output:

```

1  Serialization: Customer object saved to Customo
2
3  De-Serialization: Customer object de-serialized
4  CustomerAggregation.ser file
5
6  Printing customer values from de-serialized obje
7  Customer [customerId=101, customerName=SJ,
8  address=Address [flatNo=402, streetName=2nd stre
9  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

```
1 package in.bench.resources.serialization.aggregation;
2
3 class Address {
4     // instance variables
5     int flatNo;
6     String streetName;
7
8     // 2-arg parameterized constructor
9     public Address(int flatNo, String streetName) {
10         super();
11         this.flatNo = flatNo;
12         this.streetName = streetName;
13     }
14
15     // overriding toString() method
16     @Override
17     public String toString() {
18         return "Address [flatNo=" + flatNo
19             + ", streetName=" + streetName
20         }
21     }
22 }
```

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

SerializationWithAggregation.java

```
1 package in.bench.resources.serialization.aggregation;
2
3 import java.io.FileInputStream;
4 import java.io.FileNotFoundException;
5 import java.io.FileOutputStream;
6 import java.io.IOException;
7 import java.io.ObjectInputStream;
8 import java.io.ObjectOutputStream;
9
10 public class SerializationWithAggregation {
11     public static void main(String[] args) {
12         // creating address object --> implement
13         Address address = new Address(402, "2nd Street");
14
15         // creating phone object --> implement
16         Phone phone = new Phone(022, 27759868);
17
18         // creating customer object --> implement
19         Customer serializeCustomer =
20             new Customer(101, "SJ", address);
21
22         // time to play with Serialization and
23
24         // creating output stream variables
25         FileOutputStream fos = null;
26         ObjectOutputStream oos = null;
27
28
29 }
```

```

30 // creating input stream variables
31 FileInputStream fis = null;
32 ObjectInputStream ois = null;
33
34 // creating customer object reference
35 // to hold values after de-serializatio
36 Customer deSerializeCustomer = null;
37
38 try {
39 // for writing or saving binary dat
40 fos = new FileOutputStream("Custome
41
42 // converting java-object to binary
43 oos = new ObjectOutputStream(fos);
44
45 // writing or saving customer objec
46 oos.writeObject(serializeCustomer);
47 oos.flush();
48 oos.close();
49
50 System.out.println("Serialization:
51 + "saved to CustomerAggrega
52
53 // reading binary data
54 fis = new FileInputStream("Customer
55
56 // converting binary-data to java-o
57 ois = new ObjectInputStream(fis);
58
59 // reading object's value and casti
60 deSerializeCustomer = (Customer) oi
61 ois.close();
62
63 System.out.println("De-Serializatio
64 + "de-serialized from Custo
65 }
66 catch (FileNotFoundException fnfex) {
67 fnfex.printStackTrace();
68 }
69 catch (IOException ioex) {
70 ioex.printStackTrace();
71 }
72 catch (ClassNotFoundException ccex) {
73 ccex.printStackTrace();
74 }
75
76 // printing customer object to console
77 System.out.println("Printing customer v
78 + "de-serialized object... \n"
79 }
80 }

```

Output:

```

1 java.io.NotSerializableException: in.bench.resour
2 .aggregation.Address
3 at java.io.ObjectOutputStream.writeObject0(
4 at java.io.ObjectOutputStream.defaultWriteF
5 at java.io.ObjectOutputStream.writeSerialDa
6 at java.io.ObjectOutputStream.writeOrdinary
7 at java.io.ObjectOutputStream.writeObject0(
8 at java.io.ObjectOutputStream.writeObject(U
9 at in.bench.resources.serialization.aggrega
10 .SerializationWithAggregation.main(Serializatio

```



```
11 | Printing customer values from de-serialized obj
12 | 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/FileOutputStream.html>

tream.html

<https://docs.oracle.com/javase/7/docs/api/java/io/FileInputStream.html>

<http://docs.oracle.com/javase/7/docs/api/java/io/FileInputStream.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](#)

0 Comments

BenchResources.Net



Recommend

Share

Sort by Best ▾

Start the discussion...

Be the first to comment.

ALSO ON BENCHRESOURCES.NET

Spring JDBC: Introduction and JDBC example without spring

2 comments • 2 years ago

BenchResources.Net — Mike, Thanks for your suggestions. This article is very

Importance of serialVersionUID in

2 comments • 2 years ago

BenchResources.Net — Yes, it is possible to serialize and de-serialize static variables but

Interview question and answer on Exception Handling in Java

1 comment • 2 years ago

shailendra bhadoriya — Hi I have doubt on below points. Q) Does a method can return an

Singleton design pattern – restricting all 4 ways of Object

3 comments • 2 years ago

Anil Nivargi — Nice explanation Thanks.....Singleton Design Pattern in Java with examples

Subscribe Add Disqus to your siteAdd DisqusAdd