Serialization with Inheritance

November 7, 2016 SJ Serialization 0

In this article, we will discuss ***Serialization with Inheritance*** i.e.; ***IS-A relationship*** with inheriting class in detail

It's an easy choice, when both super class & sub class are ***Serializable,*** because

- When ***super class*** is serialized, only ***properties of super class*** will be serialized
- When ***sub class*** is serialized, ***properties of sub class*** as well as ***inherited properties of super class*** will also be serialized

But we need to understand ***2 scenarios*** with respect to ***IS-A relationship,*** while ***serializing*** & ***de-serializing*** sub class, when

1. ***Super class*** implements ***java.io.Serializable*** but ***sub class*** doesn't implement ***java.io.Serializable***
2. ***Sub class*** implements ***java.io.Serializable*** but ***super class*** doesn't implement ***java.io.Serializable***

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

Serialization with Inheritance

Case 1: *Super class* implements *java.io.Serializable* but *sub class* doesn't implement *java.io.Serializable*

- When super class is serializable, then ***any class extending super class*** will also be serializable ***by default*** (inheritance principle)
- So, here sub class ***not required*** to implement ***java.io.Serializable*** explicitly
- When ***sub class is serialized***, then sub class properties as well as ***inherited*** super class properties will also be ***serialized*** (during serialization process)
- **Note:** To ***prevent sub class from serializing by default***, then we need to override ***writeObject()*** and ***readObject()*** methods

Step 1.1: Create super class *Customer* –> implementing *java.io.Serializable* interface

- For any class to be serializable, it must implement *java.io.Serializable* interface
- Otherwise, ***NotSerializableException*** will be thrown at run time, although ***program compiles successfully***
- Overrides ***toString()*** method to print values

Customer.java

```
1  package in.bench.resources.serialization.inhe
2
3  import java.io.Serializable;
4
5  class Customer implements Serializable {
6
7      // instance variables
8      int customerId;
9      String customerName;
10
11     // overriding toString() method
12     @Override
13     public String toString() {
14         return "Customer [customerId=" + custom
15                 + ", customerName=" + customerN
16     }
17 }
```

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

Step 1.2: Create sub class *PrivilegedCustomer* –> extending super class *Customer*

- For any class to be serializable, it must implement *java.io.Serializable* interface
- But here, sub class *PrivilegedCustomer* is also serializable ***by default***, although sub class ***doesn't*** implement *java.io.Serializable* interface explicitly
- Because ***super class*** implements ***serializable*** interface (inheritance principle)
- If ***any class doesn't*** implements serializable interface, then *NotSerializableException* will be thrown at run time, although *program compiles successfully*
- Overrides ***toString()*** method to print values

PrivilegedCustomer.java

```
1  package in.bench.resources.serialization.inher
2
3  class PrivilegedCustomer extends Customer {
4
5      // instance variables
6      float discountRate;
7      int bonusPoints;
8
9      @Override
10     public String toString() {
11         return "PrivilegedCustomer [customerId=
12                 + ", customerName=" + customerN
13                 + ", discountRate=" + discountR
14                 + ", bonusPoints=" + bonusPoint
15     }
16 }
```

As we are ready with POJOs implementing *java.io.Serializable*, we will begin with our ***serialization*** and ***de-serialization process*** from main class

Step 1.3: Serialization and De-Serialization (with Inheritance)

For any class to be serializable, it must implement *java.io.Serializable* interface directly or indirectly though inheritance

Otherwise, *NotSerializableException* will be thrown at run time, although program compiles successfully

To Serialize: any Object, we can use *ObjectOutputStream* & *FileOutputStream* to *write/save* to *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*

Serializing & De-Serializing *sub class PrivilegedCustomer*

SerializationWithInheritance.java

```
1  package in.bench.resources.serialization.inher
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 SerializationWithInheritance {
11
12     public static void main(String[] args) {
13
14         // creating Privileged Customer object
15         PrivilegedCustomer serializePrivilegedC
16                 new PrivilegedCustomer();
17
18         // initialize values for privileged cus
19         serializePrivilegedCustomer.customerId
20         serializePrivilegedCustomer.customerNam
21         serializePrivilegedCustomer.discountRat
22         serializePrivilegedCustomer.bonusPoints
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 deSerializePrivilegedCustomer
37
```

```
        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(serializePrivileged
            oos.flush();
            oos.close();

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

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

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

            // reading object's value and casti
            deSerializePrivilegedCustomer =
                    (PrivilegedCustomer) ois.re
            ois.close();

            System.out.println("De-Serializatio
                    + "Privileged Customer obje
                    + "from CustomerInheritance
        }
        catch (FileNotFoundException fnfex) {
            fnfex.printStackTrace();
        }
        catch (IOException ioex) {
            ioex.printStackTrace();
        }
        catch (ClassNotFoundException ccex) {
            ccex.printStackTrace();
        }

        // printing customer object to console
        System.out.println("Printing privilege
                + "from de-serialized object...
                + deSerializePrivilegedCustomer
    }
}
```

Output:

```
Serialization: Privileged Customer object save
CustomerInheritance.ser file

De-Serialization: Privileged Customer object de-
from CustomerInheritance.ser file

Printing privilege customer values from de-seria
PrivilegedCustomer [customerId=101, customerName
discountRate=12.5, bonusPoints=1000]
```

Case 2: *Sub class* implements *java.io.Serializable* but *super class* doesn't implement *java.io.Serializable*

- Before moving ahead, we should understand ***is it possible to serializable sub class***, if its ***super class isn't serializable?***
- The answer is ***yes***, because if the condition to serialize any class on the basis of its super classes implementing *java.io.Serializable* interface, then ***no class in Java can be serialized***
- **Reason:** *java.lang.Object* is the base class for any class defined in Java, and it ***doesn't*** implements *java.io.Serializable* interface
- In that way, it is very well possible to serialize a sub class even if its super class ***doesn't*** implement *java.io.Serializable* interface

Step 2.1: Create super class *Customer* –> which doesn't implement *java.io.Serializable* interafce

- For any class to be serializable, it must implement *java.io.Serializable* interface
- Otherwise, ***NotSerializableException*** will be thrown at run time, although ***program compiles successfully***
- Overrides ***toString()*** method to print values

Customer.java

```
1  package in.bench.resources.serialization.inhe
2
3  class Customer {
4
5      // instance variables
6      int customerId;
7      String customerName;
8
9      // overriding toString() method
10     @Override
11     public String toString() {
12         return "Customer [customerId=" + custom
13                 + ", customerName=" + customerN
14     }
15 }
```

Step 2.2: Create sub class *PrivilegedCustomer* –> extending super class *Customer* and also implementing *java.io.Serializable* interface

- For any class to be ***serializable***, it must implement ***java.io.Serializable*** interface
- Here, sub class ***PrivilegedCustomer*** implement ***java.io.Serializable*** interface explicitly and also extends super class ***Customer***
- If ***any class doesn't implements serializable*** interface, then ***NotSerializableException*** will be thrown at run time, although ***program compiles successfully***
- Overrides ***toString()*** method to print values

PrivilegedCustomer.java

```
1  package in.bench.resources.serialization.inher
2
3  import java.io.Serializable;
4
5  class PrivilegedCustomer extends Customer imple
6
7      // instance variables
8      float discountRate;
9      int bonusPoints;
10
11     @Override
12     public String toString() {
13         return "PrivilegedCustomer [customerId=
14                 + ", customerName=" + customerN
15                 + ", discountRate=" + discountR
16                 + ", bonusPoints=" + bonusPoint
17     }
18 }
```

As we are ready with POJOs implementing ***java.io.Serializable***, we will begin with our ***serialization*** and ***de-serialization process*** from main class

Step 2.3: Serialization and De-Serialization (with Inheritance)

The previous case is very simple like any independent class to serialize in Java. But this case is bit different with respect to ***serialization*** and ***de-serialization process***

Serialization process:

- While serializing sub class, ***JVM will check if there are any super class*** which is not implementing ***java.io.Serializable*** interface

- Then, inheriting instance variables of ***non-serializable*** super class will be stored to ***default value*** ignoring their original values
- Like 0 for Integer, null for String, etc

De-Serialization process:

- While de-serializing sub class, ***JVM will check if there are any non-serializable*** super class
- Then, it will ***execute instance initialization*** flow (i.e.; similar to object instantiation flow)
- **1st check:** if there are direct initialization at ***instance variable declaration***
- **2nd check:** if there are any ***initialization block*** for instance variable assignment
- **3rd check:** invokes ***no-argument constructor*** and looks for instance variable assignment
- To execute the ***3rd check***, non-serializable super class requires ***no-argument constructor***
- **Exception:** otherwise *InvalidClassException* will be thrown
- **Note:** For any other case, ***constructor is not invoked*** with only exception being for ***non-serializable super class***

Serializing & De-Serializing *sub class PrivilegedCustomer*

SerializationWithInheritance.java

```
1  package in.bench.resources.serialization.inhe
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 SerializationWithInheritance {
11
12     public static void main(String[] args) {
13
14         // creating Privileged Customer object
15         PrivilegedCustomer serializePrivilegedC
16                 new PrivilegedCustomer();
17
18         // initialize values for privileged cus
19         serializePrivilegedCustomer.customerId
20         serializePrivilegedCustomer.customerNam
21         serializePrivilegedCustomer.discountRat
22         serializePrivilegedCustomer.bonusPoints
```

```
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 deSerializePrivilegedCustomer
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(serializePrivileged
47              oos.flush();
48              oos.close();
49
50              System.out.println("Serialization:
51                      + "object saved to Customer
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              deSerializePrivilegedCustomer =
61                      (PrivilegedCustomer) ois.re
62              ois.close();
63
64              System.out.println("De-Serializatio
65                      + "Privileged Customer obje
66                      + "from CustomerInheritance
67          }
68          catch (FileNotFoundException fnfex) {
69              fnfex.printStackTrace();
70          }
71          catch (IOException ioex) {
72              ioex.printStackTrace();
73          }
74          catch (ClassNotFoundException ccex) {
75              ccex.printStackTrace();
76          }
77
78          // printing customer object to console
79          System.out.println("Printing privilege
80                  + "from de-serialized object...
81                  + deSerializePrivilegedCustomer
82      }
83  }
```

Output:

```
Serialization: Privileged Customer object saved
CustomerInheritance.ser file

De-Serialization: Privileged Customer object de-
from CustomerInheritance.ser file

Printing privilege customer values from de-seria
PrivilegedCustomer [customerId=0, customerName=n
discountRate=12.5, bonusPoints=1000]
```

Important points to remember while Serialization with Inheritance:

- If *super class* implements *java.io.Serializable* interface, then all *sub class* is also *serializable* by default
- It is possible to serialize sub class, even if its *corresponding super class doesn't* implements *java.io.Serializable* interface
- While serializing sub class whose super class doesn't implements *io.Serializable* interface, then during *serialization process inheriting instance variables* of *non-serializable* super class will be stored to *default value* ignoring their *original values* (like 0 for Integer, null for String, etc)
- During de-serialization process, JVM will execute instance initialization flow in 3 steps i.e.;
  1st checks direct variable assignment,
  2nd check inside initialization block and
  3rd check inside no-argument constructor
- For the *3rd check*, it is very must to code a *no-argument constructor* inside *non-serializable super class*
- Otherwise, *InvalidClassException* will be thrown at run time

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
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 Aggregation
- 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 !!

« Externalizable interface with example

Serialization with Aggregation »

Related Posts:

1. **Transient keyword with final variable in Serialization**
2. **Order of Serialization and De-Serialization**
3. **Singleton Design pattern with Serialization**
4. **How to stop Serialization in Java**

DESERIALIZATION INHERITANCE JAVA

SERIALIZABLE SERIALIZATION

ALSO ON BENCHRESOURCES.NET

Interview Question and Answers on final keyword in

2 comments • 2 years ago

BenchResources.Net — Anurag,You are most welcome !!There are various other Java

Sorting list of objects on multiple fields using

1 comment • 2 years ago

Juan Daniel Ornella — Thank You!!

Java JDBC: An example to connect MS Access database

2 comments • 2 years ago

BenchResources.Net — Buket,You need to include hsqldb-2.4.0.jar in your project

Importance of SerialVersionUID in

2 comments • 2 years ago

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