

BenchResources.Net

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

Externalizable interface with example

🕒 November 8, 2016 👤 SJ 📁 Serialization 💬 0

In this article, we will discuss *externalizable interface* with an example *to save & restore an object* in a customized way

Also, we will discuss *advantage* of using *Externalizable over Serializable* in detail

In next article we will cover important points while discussing **difference between Externalizable and Serializable interfaces**

Serializable interface:

Although, we have discussed **serializable interface in detail** in one of the **previous article**, here we will list out what are the various things that affects performance

- While serializable implemented class does the *necessary job of Serialization and de-serialization* in saving & restoring object *but in saves altogether all member variables* of an object
- This way, even if programmer *requires only couple of member variables of an Object to be saved*, Serializable doesn't allow

SEARCH TUTORIALS

SUBSCRIBE VIA EMAIL

Join 194 other subscribers

POPULAR ARTICLES

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

those kinds of flexibility

- That is ***no flexibility saving & restoring partial object***
- It is ***time consuming*** in ***saving and restoring object*** during both serialization and de-serialization process
- As ***JVM controls the complete serialization and de-serialization process*** and programmer has nothing to do with serializable interface
- With ***transient modifier*** also, we can ***stop serializing original value*** but still that particular member variable get ***saved to file storage*** although with default value
- Due to saving and restoring ***all member variables of an Object***, even if programmer requires only couple of variables to saved and restored back, there is ***big performance hit***

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

To overcome above ***listed performance issue with serializable***, we have to ***serialize with externalizable interface*** which is sub-interface of Serializable interface

Advantage of Externalizable over Serializable:

- Allows ***saving and restoring partial object*** i.e.; 2 or 3 member variables of an object out of total object
- As programmer has to ***code/write the custom logic for serialization and de-serialization*** process, so write logic to save/restore ***those variables which is required***
- This way, there is ***performance boost relatively when comparing*** with serializable interface
- **Transient:** variable is ***not required as programmer has the control*** over saving/restoring object and can easily ignore those variables whose value is secure or need to kept very secret
- By saving and restoring partial object instead of total object, ***time consumption decreases*** (i.e.; ***time is saved*** in externalizable interface)

BenchR

G+ Fc

88 follower



Be the first of your friends to l

Externalizable interface

- Externalizable interface is *sub-interface* of Serializable interface
- Present in *java.io* package
- Fully qualified class name is *java.io.Externalizable*
- It has got 2 methods namely, *writeExternal()* and *readExternal()*
- **Method 1:** with *writeExternal(ObjectOutput out)* method, programmer has to explicitly code/write *logic for saving only those required variables* to file storage
- **Method 2:** with *readExternal(ObjectInput in)* method, programmer has to explicitly code/write *logic for restoring object back* from *file storage*
- **Note:** class implementing *externalizable* interface should *definitely consist of a public no-arg constructor*, otherwise *InvalidClassException* is thrown
- **Design choice:** This is the best suit; when *partial object* or *few member variables of an object need to be serialized* to file storage, otherwise still *serializable interface is a good option for saving total object*

Write complete method signature of 2 methods?

```
1 public void writeExternal(ObjectOutput out) throws IOException {
2
3 public void readExternal(ObjectInput in)
4     throws IOException, ClassNotFoundException {
```

Class to be serialized:

- Below Customer class consists of 4 member variables, out of which 2 variables need to be serialized (and other variables are discarded)
- In *Externalization*, programmer need to implement/override 2 methods for saving and restoring object
- To partially serialize, we have to override 2 methods
writeExternal(); → for saving/writing in serialization process
readExternal(); → for restoring during de-serialization process

Serialization:

During serialization inside `writeExternal()` method, programmer *has to code/write custom logic to save/persist 2 member variables*

De-Serialization:

During de-serialization inside `readExternal()` method, programmer *has to code/write custom logic to read 2 variable* and then finally assigning to actual member variables

Customer.java

```
1  package in.bench.resources.externalization;
2
3  import java.io.Externalizable;
4  import java.io.IOException;
5  import java.io.ObjectInput;
6  import java.io.ObjectOutput;
7
8  // class implementing Externalizable interface
9  class Customer implements Externalizable {
10
11     // member variables for Customer
12     int customerId;
13     String customerName;
14     int customerAge;
15     String customerSSN;
16
17     // default public no-arg constructor
18     public Customer() {
19         System.out.println("public no-arg const
20             + "Externalizable, "
21             + "while restoring object back
22     }
23
24     // 4-arg parameterized constructor for Cust
25     public Customer(int customerId, String cust
26         int customerAge, String customerSSN
27         super();
28         this.customerId = customerId;
29         this.customerName = customerName;
30         this.customerAge = customerAge;
31         this.customerSSN = customerSSN;
32     }
33
34     @Override
35     public void writeExternal(ObjectOutput out)
36
37         // saving to file storage
38         out.writeInt(customerId);
39         out.writeObject(customerName);
```

```

40     }
41
42     @Override
43     public void readExternal(ObjectInput in)
44         throws IOException, ClassNotFoundException
45     {
46         // restoring variables, as per order of
47         int tempCustId = in.readInt();
48         String tempCustName = (String) in.readObject();
49
50         // assigning restored values to member
51         customerId = tempCustId;
52         customerName = tempCustName;
53     }
54
55     // to print nicely - customer object
56     @Override
57     public String toString() {
58         return "Customer [customerId=" + customerId + ", customerName=" + customerName + ", customerSSN=" + customerSSN + ", customerAge=" + customerAge + "]";
59     }
60
61 }

```

Serialization and De-serialization using Externalizable interface:

This program is the test class to *write/save customer object to file storage and then restoring* for reading customer object

- 1st part explains, complete serialization process
- 2nd explains, complete de-serialization process

Note: class that needs to be serialized is *implementing Externalizable interface* unlike Serializable interface in earlier examples

CustomerSerialization.java

```

1  package in.bench.resources.externalization;
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 CustomerSerialization {

```

```

11
12 public static void main(String[] args) {
13
14     // create an customer object using 4-ar
15     Customer serializeCustomer =
16         new Customer(102, "NK", 19, "SS
17
18     // creating output stream variables
19     FileOutputStream fos = null;
20     ObjectOutputStream oos = null;
21
22     // creating input stream variables
23     FileInputStream fis = null;
24     ObjectInputStream ois = null;
25
26     // creating customer object reference
27     // to hold values after de-serializatio
28     Customer deSerializeCustomer = null;
29
30     try {
31         // for writing or saving binary dat
32         fos = new FileOutputStream("Custome
33
34         // converting java-object to binary
35         oos = new ObjectOutputStream(fos);
36
37         // writing or saving customer objec
38         oos.writeObject(serializeCustomer);
39         oos.flush();
40         oos.close();
41
42         System.out.println("Externalization
43             + "Customer object saved to
44
45         // reading binary data
46         fis = new FileInputStream("Customer
47
48         // converting binary-data to java-o
49         ois = new ObjectInputStream(fis);
50
51         // reading object's value and casti
52         deSerializeCustomer = (Customer) oi
53         ois.close();
54
55         System.out.println("Externalization
56             + "de-serialized from Custo
57     }
58     catch (FileNotFoundException fnfex) {
59         fnfex.printStackTrace();
60     }
61     catch (IOException ioex) {
62         ioex.printStackTrace();
63     }
64     catch (ClassNotFoundException ccex) {
65         ccex.printStackTrace();
66     }
67
68     // printing customer object to console
69     System.out.println("Printing customer v
70         + "de-serialized object... \n"
71 }
72 }

```

Output:

```

1 Externalization: Customer object saved to Cust
2
3 public no-arg constructor is must for Externaliz
4 while restoring object back from file storage
5 Externalization: Customer object de-serialized f
6
7 Printing customer values from de-serialized obje
8 Customer [customerId=102, customerName=NK, custo

```

Explanation:

- Only *two variables is persisted and restored* back and other variables are discarded as it isn't required
- So, when we *print customer object using overridden toString() method*, only *customer Id* and *customer Name* is restored and *other variables assigned to default values*
- Like, *null* for customer SSN number and *0* for customer age
- **Note:** public no-arg constructor is very must *while restoring object back* from file storage
- Otherwise, *InvalidClassException* is thrown

Exception scenario:

Let us tweak above example by *removing public no-arg constructor* and *try to serialize & de-serialize* customer object,

Output:

```

1 Externalization: Customer object saved to Cust
2
3 java.io.InvalidClassException: in.bench.resourc
4 .Customer; no valid constructor
5     at java.io.ObjectStreamClass$ExceptionInfo
6     .newInvalidClassException(ObjectStreamClass.jav
7     at java.io.ObjectStreamClass
8     .checkDeserialize(ObjectStreamClass.java:790)
9     at java.io.ObjectInputStream
10    .readOrdinaryObject(ObjectInputStream.java:1775
11    at java.io.ObjectInputStream
12    .readObject0(ObjectInputStream.java:1351)
13    at java.io.ObjectInputStream
14    .readObject(ObjectInputStream.java:371)
15    at in.bench.resources.externalization
16    .CustomerSerialization.main(CustomerSerializati
17    Printing customer values from de-serialized obj
18    null

```

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](#)
- [Serialization with Inheritance](#)
- [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 !!



Serializable v/s Externalizable

Serialization with Inheritance



Related Posts:

1. [Transient keyword with Serialization in Java](#)
2. [Transient keyword with static variable in Serialization](#)
3. [Transient keyword with final variable in Serialization](#)
4. [Serializing a variable with transient modifier or keyword](#)

DESERIALIZATION

JAVA

SERIALIZABLE

SERIALIZATION

TRANSIENT

0 Comments

BenchResources.Net



Recommend

Share

Sort by Best ▾

Start the discussion...

Be the first to comment.

ALSO ON BENCHRESOURCES.NET

Java Collection Interview question and answers

2 comments • 2 years ago

Adil Khan — A small request,would be happy to see if you write article on how

RestEasy: JAX-RS web service + Integrating with Spring MVC

2 comments • 2 years ago

BenchResources.Net — Rohit,There is no explicit file named "springmvc-

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

Generics interview question and answer in Java

1 comment • a year ago

Adil Khan — Thanks for sharing.... its always deep learning with you :-)

Subscribe Add Disqus to your siteAdd DisqusAdd

