Importance of SerialVersionUID in Serialization

November 10, 2016 SJ Serialization 2

In this article, we will discuss ***importance of SerialVersionUID in Serialization*** and ***De-Serialization process***. And finally ***compare*** the ***compiler generated*** SerialVersionUID ***v/s programmer defined*** SerialVersionUID and decide ***which one to use?***

In all the previous articles, ***we haven't discussed*** anything about serialVersionUID

- Introduction to Serialization
- Serializable interface in detail with example
- Transient modifier in serialization process
- Order of serialization and de-serialization
- Serialization with Aggregation
- Serialization with Inheritance
- Externalizable interface in detail with example

But there is always serialVersionUID ***associated*** with every serializable class.

SEARCH TUTORIALS

SEARCH ...

SUBSCRIBE VIA EMAIL

Join 194 other subscribers

Email Address

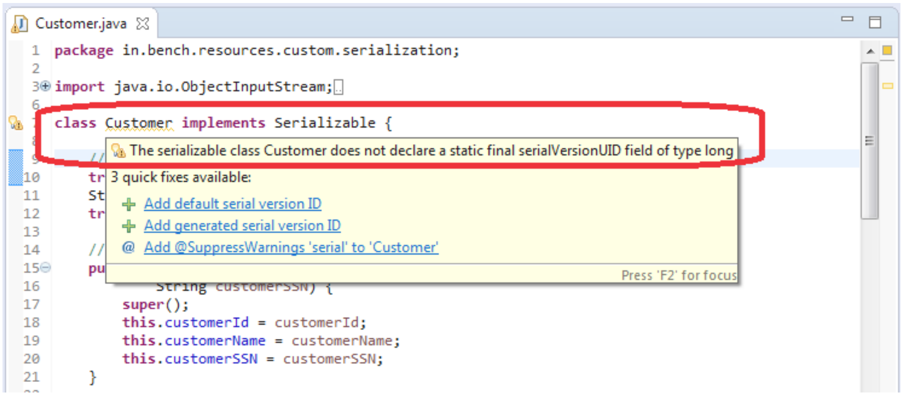
SUBSCRIBE

POPULAR ARTICLES

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

If you are using *IDE like Eclipse*, then it *warns* with following message

"*The serializable class <class-name> does not declare static final serialVersionUID field of type long*"



```
Customer.java

package in.bench.resources.custom.serialization;

import java.io.ObjectInputStream;

class Customer implements Serializable {

    //
    tr
    St
    tr

    //
    pu
            String customerSSN) {
        super();
        this.customerId = customerId;
        this.customerName = customerName;
        this.customerSSN = customerSSN;
    }
```

The serializable class Customer does not declare a static final serialVersionUID field of type long

3 quick fixes available:

- Add default serial version ID
- Add generated serial version ID
- Add @SuppressWarnings 'serial' to 'Customer'

Press 'F2' for focus

As we stated earlier that there is always a serialVersionUID associated with every serializable class, then **where we have declared in earlier example?**

Serialization: Actually, we haven't declared this field and if it isn't declared then compiler does the job for us by declaring this static field and it get saved to serialized file along with Object values

De-Serialization: while restoring object back from file storage then first thing it does is, compare stored serialVersionUID inside serialized file with serializable class

Exception: if there is a mismatch between serialVersionUID present in the serialized file and serializable class, then *InvalidClassException* will be thrown

Now, the *next question with serialVersionUID* is whether to use compiler generated serialVersionUID or programmer explicitly declaring serialVersionUID i.e.;

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

serialVersionUID: *Compiler generated v/s programmer defined*

- Compiler generated serialVersionUID is ***highly complex*** as it uses combination of class name and properties to generate this unique Id
- Due to the ***complexity in creation*** of this unique Id, ***performance*** of serialization and de-serialization process ***becomes slow***
- Therefore, it is ***highly recommended to define serialVersionUID*** inside serializable class and use for both serialization and de-serialization process
- Firstly it ***reduces the complexity*** in creating compiler generated serialVersionUID and comparing this unique Id during ***de-serialization process***
- Programmer has the ***flexibility*** of declaring any ***Long value***

Lets us see a simple demo program for both cases:

Exercise 1: serialVersionUID is same

Below customer class is a serializable class i.e.; it implements *java.io.Serializable* interface and programmer provides *serialVersionUID* with value *19L*

Customer.java

```
package in.bench.resources.serial.version.uid;

import java.io.Serializable;

class Customer implements Serializable {

    // default serialVersionUID
    private static final long serialVersionUID

    // member variables for Customer
    int customerId;
    String customerName;
    String customerSSN;

```

```java
    // 3-arg parameterized constructor for Cust
    public Customer(int customerId, String cust
            String customerSSN) {
        super();
        this.customerId = customerId;
        this.customerName = customerName;
        this.customerSSN = customerSSN;
    }

    // to print nicely - customer object
    @Override
    public String toString() {
        return "Customer [customerId=" + custom
                + ", customerName=" + customerN
                + ", customerSSN=" + customerSS
    }
}
```

This class is the main class which *serializes the Customer class* with *serialVersionUID 19L*

SerializeCustomer.java

```java
package in.bench.resources.serial.version.uid;

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

public class SerializeCustomer {

    public static void main(String[] args) {

        // create a customer object using 3-arg
        Customer customer = new Customer(101, "

        // creating output stream variables
        FileOutputStream fos = null;
        ObjectOutputStream oos = 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(customer);
            oos.flush();
            oos.close();
        }
        catch (FileNotFoundException fnfex) {
            fnfex.printStackTrace();
        }
        catch (IOException ioex) {
            ioex.printStackTrace();
        }
```

```
 
            System.out.println("Customer object sav
        }
    }
```

Output:

```
Customer object saved to Customer.ser file
```

This class de-serializes the *Customer class with the same serialVersionUID* used for serialization (i.e.; *19L*)

DeSerializeCustomer.java

```
package in.bench.resources.serial.version.uid;

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

public class DeSerializeCustomer {

    public static void main(String[] args) {

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

        // creating customer object reference
        // to hold values after de-serializatio
        Customer customer = null;
        try {
            // reading binary data
            fis = new FileInputStream("Customer

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

            // reading object's value and casti
            customer = (Customer) ois.readObjec
        }
        catch (FileNotFoundException fnfex) {
            fnfex.printStackTrace();
        }
        catch (IOException ioex) {
            ioex.printStackTrace();
        }
        catch (ClassNotFoundException ccex) {
            ccex.printStackTrace();
        }

        System.out.println("Customer object de-
                + "Customer.ser file\nLet's pri

        // printing customer object to console
        System.out.println(customer);
    }
}
```

Output:

```
Customer object de-serialized from Customer.se
Let's print to console...

Customer [customerId=101, customerName=SJ, custo
```

Exercise 2: serialVersionUID is different

Let us tweak above example by *changing serialVersionUID* after serialization process

- We will keep same ***serialVersionUID*** i.e.; ***19L*** while ***serialization***
- ***Change*** serialVersionUID ***after serialization***
- That's, ***change*** to ***21L***
- ***Serialization*** program will be executed and same ***output*** will be seen ***as per earlier case***
- But during ***de-serialization*** process, due to the ***difference of serialVersionUID***, runtime exception will be thrown i.e.; *InvalidClassException*

Steps:

- Keep same serialVersionUID (i.e.; ***19L***) in Customer class and ***execute serialize customer*** class
- Above step help in storing or ***saving customer object to serialized file***
- Now, ***change serialVersionUID*** to ***21L*** in Customer class and compile again
- Next step, ***execute de-serialize*** customer class

Output:

```
java.io.InvalidClassException: in.bench.resou
Customer; local class incompatible:
stream classdesc serialVersionUID = 19, local c
    at java.io.ObjectStreamClass.initNonProxy(
ObjectStreamClass.java:616)
    at java.io.ObjectInputStream.readNonProxyDe
ObjectInputStream.java:1623)
    at java.io.ObjectInputStream.readClassDesc(
ObjectInputStream.java:1518)
    at java.io.ObjectInputStream.readOrdinaryOb
ObjectInputStream.java:1774)
    at java.io.ObjectInputStream.readObject0(
ObjectInputStream.java:1351)
```

```
    at java.io.ObjectInputStream.readObject(
ObjectInputStream.java:371)
    at in.bench.resources.serial.version.uid.De
main(DeSerializeCustomer.java:27)
Customer object de-serialized from Customer.ser
Let's print to console...

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
- Externalizable interface with example
- Serializable v/s Externalizable
- 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 !!

« Singleton Design pattern with Serialization

Serializable v/s Externalizable »

Related Posts:

1. **Singleton Design pattern with Serialization**
2. **How to stop Serialization in Java**
3. **Serialization interview question and answer in Java**
4. **Serialization with Inheritance**

DESERIALIZATION JAVA SERIALIZABLE SERIALIZATION SERIALVERSIONUID

2 Comments BenchResources.Net Ava

Recommend Share Sort by Best

Join the discussion…

infoj • a year ago
Both the name of the class and its serialVersionUID are written to the object stream. One thing to notice here is that even if serialVersionUID field is static it is still serialized and stored.
• Reply • Share ›

BenchResources.Net Mod infoj • a year ago
Yes, it is possible to serialize and de-serialize static variables but there mechanism is bit different from instance variables.

Take a look at the below article for more explanation with example http://www.benchresources.n...
• Reply • Share ›

ALSO ON **BENCHRESOURCES.NET**

Sorting list of objects on multiple fields using
1 comment • 2 years ago
Juan Daniel Ornella — Thank You!!

JDBC: An example to connect MS Access database in Java 8
18 comments • 2 years ago
Daroga Jee — then tell us.. Where to go and connect to MS-Excel..

Java features version-wise
2 comments • a year ago

Java: StringBuffer length() method