Order of Serialization and De-Serialization

November 5, 2016 SJ Serialization 0

In this article, we will discuss *Order of Serialization* and *De-Serialization* and also,

- Why it is ***important to know order of serialization***
- What happens, if ***de-serialize in different order than serialization order***
- What is the ***readymade solution available*** from Java to overcome this situation, if we don't know the order of serialization

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

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

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

Order of Serialization and De-Serialization

Create 3 POJO classes for Customer, Employee and Student and all implementing *java.io.Serializable* interface

- Any class said to be serializable, if it implements *java.io.Serializable* interface
- Otherwise, ***NotSerializableException*** will be thrown at run time, although ***program compiles successfully***
- All POJO's has 2 instance variables & 2-arg parameterized constructor and override ***toString()*** method to print values

Customer.java

```
package in.bench.resources.serialization;

import java.io.Serializable;

class Customer implements Serializable {

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

    // 2-arg parameterized constructor for Cust
    public Customer(int customerId, String cust
        this.customerId = customerId;
        this.customerName = customerName;
    }

    @Override
    public String toString() {
        return "Customer [customerId=" + custom
                + ", customerName=" + customerN
    }
}
```

Employee.java

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

```
package in.bench.resources.serialization;

import java.io.Serializable;

class Employee implements Serializable {

    // member variables for Employee
    int employeeId;
    String employeeName;

    // 2-arg parameterized constructor for Empl
    public Employee(int employeeId, String empl
        this.employeeId = employeeId;
        this.employeeName = employeeName;
    }

    @Override
    public String toString() {
        return "Employee [employeeId=" + employ
                + ", employeeName=" + employeeN
    }
}
```

Student.java

```
package in.bench.resources.serialization;

import java.io.Serializable;

class Student implements Serializable {

    // member variables for Student
    int studentId;
    String studentName;

    // 2-arg parameterized constructor for Stud
    public Student(int studentId, String studen
        this.studentId = studentId;
        this.studentName = studentName;
    }

    @Override
    public String toString() {
        return "Student [studentId=" + studentI
                + ", studentName=" + studentNam
    }
}
```

As we are ready with POJOs, we will begin with our ***serialization*** and ***de-serialization*** in below 3 cases

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*

Program 1: When *order of serialization is known*, we can de-serialize in *same order*

Here, we know serialization order so it becomes easy for us to do de-serialization

OrderOfSerializationDeSerialization.java

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

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 OrderOfSerializationDeSerializatio

    public static void main(String[] args) {

        Customer customer = new Customer(101, "
        Employee employee = new Employee(111, "
        Student student = new Student(121, "Aze

        // 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;
        Employee deSerializeEmployee = null;
        Student deSerializeStudent = null;

        try {
            // for writing or saving binary dat
            fos = new FileOutputStream("OrderOf

            // converting java-object to binary
            oos = new ObjectOutputStream(fos);

            // writing or saving customer objec
            oos.writeObject(customer);
```

```java
                oos.writeObject(employee);
                oos.writeObject(student);
                oos.flush();
                oos.close();

                System.out.println("Serialization:
                        + "saved to OrderOfObjects.

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

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

                // reading object's value and casti
                deSerializeCustomer = (Customer) oi
                deSerializeEmployee = (Employee) oi
                deSerializeStudent = (Student) ois.
                ois.close();

                System.out.println("De-Serializatio
                        + "de-serialized from Order
            }
            catch (FileNotFoundException fnfex) {
                fnfex.printStackTrace();
            }
            catch (IOException ioex) {
                ioex.printStackTrace();
            }
            catch (ClassNotFoundException ccex) {
                ccex.printStackTrace();
            }

            // printing customer object to console
            System.out.println("Printing values "
                    + "from de-serialized object...
            System.out.println(deSerializeCustomer)
            System.out.println(deSerializeEmployee)
            System.out.println(deSerializeStudent);
        }
    }
```

Output:

```
Serialization: All objects saved to OrderOfObj

De-Serialization: All objects de-serialized from

Printing values from de-serialized object...

Customer [customerId=101, customerName=Jeremy Kr
Employee [employeeId=111, employeeName=Mike Gent
Student [studentId=121, studentName=Azeem Sayed]
```

Program 2: De-serialization is done in *different than serialization order*

In this program, irrespective of whether we know serialization order or NOT, we will perform de-serialization in some ***random order***

Let's see what happens, if we change the de-serialization order (other than from serialization order)

Serializing order

1. Customer
2. Employee
3. Student

De-Serializing order

1. Student
2. Customer
3. Employee

OrderOfSerializationDeSerialization.java

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

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 OrderOfSerializationDeSerializatio

    public static void main(String[] args) {

        Customer customer = new Customer(101, "
        Employee employee = new Employee(111, "
        Student student = new Student(121, "Aze

        // 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;
        Employee deSerializeEmployee = null;
        Student deSerializeStudent = null;

        try {
            // for writing or saving binary dat
            fos = new FileOutputStream("OrderOf

            // converting java-object to binary
            oos = new ObjectOutputStream(fos);

            // writing or saving customer objec
            oos.writeObject(customer);
            oos.writeObject(employee);
            oos.writeObject(student);
            oos.flush();
            oos.close();

            System.out.println("Serialization:
                    + "saved to OrderOfObjects.

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

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

            // reading object's value and casti
            deSerializeStudent = (Student) ois.
            deSerializeCustomer = (Customer) oi
            deSerializeEmployee = (Employee) oi
            ois.close();

            System.out.println("De-Serializatio
                    + "de-serialized from Order
        }
        catch (FileNotFoundException fnfex) {
            fnfex.printStackTrace();
        }
        catch (IOException ioex) {
            ioex.printStackTrace();
        }
        catch (ClassNotFoundException ccex) {
            ccex.printStackTrace();
        }

        // printing customer object to console
        System.out.println("Printing values"
                + " from de-serialized object..
        System.out.println(deSerializeCustomer)
        System.out.println(deSerializeEmployee)
        System.out.println(deSerializeStudent);
    }
}
```

Output:

```
Serialization: All objects saved to OrderOfObj

Exception in thread "main" java.lang.ClassCastEx
.serialization.Customer cannot be cast to
in.bench.resources.serialization.Student
    at in.bench.resources.serialization.OrderOfS
.main(OrderOfSerializationDeSerialization.java:1
```

Explanation:

- Here serialization order is ***Customer –> Employee –> Student***
- But we are de-serializing in different order i.e.; ***Student –> Customer –> Employee***
- So, while ***de-serializing 1st time*** when we read object from serialized file, it returns Customer object, as we ***serialized Customer object first***
- But instead of ***type-casting to Customer object***, we type-casted to ***Student*** object –> which results in throwing ***java.lang.ClassCastException***
- To overcome this exception, we can use ***instanceOf*** operator
- Move to ***program 3*** –> for much improved version using ***instanceOf*** operator

Program 3: When order of serialization is unknown, how can we overcome this situation?

We can use ***instanceOf*** operator to check the respective object first iterating through while loop

Later, we can assign it to correct class by type-casting

Note: here, program will throw ***java.io.EOFException*** for condition checked inside parenthesis of while loop

But we can catch this exception and take corrective action (like here, we can print "*End of file message*" to console)

OrderOfSerializationDeSerialization.java

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

import java.io.EOFException;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
```

```java

public class OrderOfSerializationDeSerializatio

    public static void main(String[] args) thro

        Customer customer = new Customer(101, "
        Employee employee = new Employee(111, "
        Student student = new Student(121, "Aze

        // 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;
        Employee deSerializeEmployee = null;
        Student deSerializeStudent = null;

        try {
            // for writing or saving binary dat
            fos = new FileOutputStream("OrderOf

            // converting java-object to binary
            oos = new ObjectOutputStream(fos);

            // writing or saving customer objec
            oos.writeObject(customer);
            oos.writeObject(employee);
            oos.writeObject(student);
            oos.flush();
            oos.close();

            System.out.println("Serialization:
                    + "saved to OrderOfObjects.

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

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

            // temp Object variable
            Object object = null;

            // iterating, reading & casting to
            while((object = ois.readObject()) !
                if(object instanceof Customer)
                    deSerializeCustomer = (Cust
                else if(object instanceof Emplo
                    deSerializeEmployee = (Empl
                else if(object instanceof Stude
                    deSerializeStudent = (Stude
            } // END of while loop
        }
        catch (EOFException eofex) {
            // eofex.printStackTrace();
            System.out.println("De-Serializatio
                    + "de-serialized from Order
            System.out.println("End of file rea
        }
        catch (FileNotFoundException fnfex) {
            fnfex.printStackTrace();
        }
        catch (IOException ioex) {
            ioex.printStackTrace();
        }
        catch (ClassNotFoundException ccex) {
            ccex.printStackTrace();
        }
        finally {
            ois.close(); // closing stream
        }

        // printing customer object to console
        System.out.println("Printing values "
                + "from de-serialized object...
        System.out.println(deSerializeCustomer)
        System.out.println(deSerializeEmployee)
        System.out.println(deSerializeStudent);
    }
}
```

Output:

```
Serialization: All objects saved to OrderOfOb

De-Serialization: All objects de-serialized fro

End of file reached...

Printing values from de-serialized object...

Customer [customerId=101, customerName=Jeremy K
Employee [employeeId=111, employeeName=Mike Gen
Student [studentId=121, studentName=Azeem Sayed
```

Important points about Serialization Order:

- **Rule 1:** all ***classes*** that need to be ***serialized*** must implement ***java.io.Serializable*** interface
- ***Order of Serialization*** is very ***important*** to know, because we need to follow the ***same order while de-serializing*** the objects
- If the Order of Serialization is ***unknown***, then it may throw ***java.lang.ClassCastException***
- To overcome ***ClassCastException***, we can 1st check type of object using ***instanceOf*** operator and then ***assign*** it to proper class after doing necessary ***type-casting***
- **Exception:** iterating through while loop may throw ***EOFException***, we need ***catch*** this exception and ***handle*** it properly

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
- Serialization with Aggregation
- 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 Aggregation

Serializing a variable with transient modifier or keyword »

Related Posts:

1. Transient keyword with final variable in Serialization
2. Serialization with Aggregation
3. Singleton Design pattern with Serialization
4. How to stop Serialization in Java

DESERIALIZATION INSTANCEOF OPERATOR JAVA

SERIALIZABLE SERIALIZATION

0 Comments **BenchResources.Net** Ava

Recommend Share Sort by Best

Start the discussion…

Be the first to comment.

ALSO ON **BENCHRESOURCES.NET**

Sorting list of objects on multiple fields using

1 comment • 2 years ago

Juan Daniel Ornella — Thank You!!

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

2 comments • 2 years ago

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

Exception Hierarchy in Java

1 comment • 2 years ago

Md.Ruhul Amin (Ruhul) — Thank you sir. Very good resource and explanation about

How to remove an entry from HashMap in Java 8

8 comments • a year ago

BenchResources.Net — Glad that it helped you !!Following article covers how to remove