

[Home](#) » [Java](#) » [Core Java](#) » [Java Best Practices – High performance Serialization](#)

## ABOUT BYRON KIOURTZOGLOU



Byron is a master software engineer working in the IT and Telecom domains. He is an applications developer in a wide variety of applications/services. He is currently acting as the team leader and technical architect for a proprietary service creation and integration platform for both the IT and Telecom industries in addition to a in-house big data real-time analytics solution. He is always fascinated by SOA, middleware services and mobile development. Byron is co-founder and Executive Editor at Java Code Geeks.



## Java Best Practices – High performance Serialization

Posted by: [Byron Kiourtoglou](#) in [Core Java](#) July 23rd, 2010 8 Comments 938 Views

Continuing our series of articles concerning proposed practices while working with the Java programming language, we are going to discuss and demonstrate how to utilize Object Serialization for high performance applications.

All discussed topics are based on use cases derived from the development of mission critical, ultra high performance production systems for the telecommunication industry.

Prior reading each section of this article it is highly recommended that you consult the relevant Java API documentation for detailed information and code samples.

All tests are performed against a Sony Vaio with the following characteristics :

- System : openSUSE 11.1 (x86\_64)
- Processor (CPU) : Intel(R) Core(TM)2 Duo CPU T6670 @ 2.20GHz
- Processor Speed : 1,200.00 MHz
- Total memory (RAM) : 2.8 GB
- Java : OpenJDK 1.6.0\_0 64-Bit

The following test configuration is applied :

- Concurrent worker Threads : 200
- Test repeats per worker Thread : 1000
- Overall test runs : 100

## High performance Serialization

Serialization is the process of converting an object into a stream of bytes. That stream can then be sent through a socket, stored to a file and/or database or simply manipulated as is. With this article we do not intend to present an in depth description of the serialization mechanism, there are numerous articles out there that provide this kind of information. What will be discussed here is our proposition for utilizing serialization in order to achieve high performance results.

The three main performance problems with serialization are :

- Serialization is a recursive algorithm. Starting from a single object, all the objects that can be reached from that object by following instance variables, are also serialized. The default behavior can easily lead to unnecessary Serialization overheads
- Both serializing and deserializing require the serialization mechanism to discover information about the instance it is serializing. Using the default serialization mechanism, will use reflection to discover all the field values. Furthermore if you don't explicitly set a „serialVersionUID“ class attribute, the serialization mechanism has to compute it. This involves going through all the fields and methods to generate a hash. The aforementioned procedure can be quite slow
- Using the default serialization mechanism, all the serializing class description information is included in the stream, such as :
  - The description of all the serializable superclasses
  - The description of the class itself
  - The instance data associated with the specific instance of the class

## NEWSLETTER

Insiders are already enjoying weekly up to date complimentary whitepapers!

**Join them now** to gain **exclusive access** to the latest news in the Java world as well as insights about Android, Scala, and other related technologies.

☐ I agree to the Terms and Privacy Policy

## JOIN US



With **1,240,600** unique visitors and **500** authors placed among related sites and constantly being looked out for by our partners, you can encourage your writing skills! So If you have unique and interesting content then you can check out our **JCG** partners program. You can be a **guest writer** for Java Code Geek and showcase your writing skills!

To solve the aforementioned performance problems you can use Externalization instead. The major difference between these two methods is that Serialization writes out class descriptions of all the serializable superclasses along with the information associated with the instance when viewed as an instance of each individual superclass. Externalization, on the other hand, writes out the identity of the class (the name of the class and the appropriate „serialVersionUID“ class attribute) along with the superclass structure and all the information about the class hierarchy. In other words, it stores all the metadata, but writes out only the local instance information. In short, Externalization eliminates almost all the reflective calls used by the serialization mechanism and gives you complete control over the marshalling and demarshalling algorithms, resulting in dramatic performance improvements.

Of course, Externalization efficiency comes at a price. The default serialization mechanism adapts to application changes due to the fact that metadata is automatically extracted from the class definitions. Externalization on the other hand isn't very flexible and requires you to rewrite your marshalling and demarshalling code whenever you change your class definitions.

What follows is a short demonstration on how to utilize Externalization for high performance applications. We will start by providing the "Employee" object to perform serialization and deserialization operations. Two flavors of the "Employee" object will be used. One suitable for standard serialization operations and another that is modified so as to be able to be externalized.

Below is the first flavor of the "Employee" object :

```
001 package com.javacodegeeks.test;
002
003 import java.io.Serializable;
004 import java.util.Date;
005 import java.util.List;
006
007 public class Employee implements Serializable {
008
009     private static final long serialVersionUID = 3657773293974543890L;
010
011     private String firstName;
012     private String lastName;
013     private String socialSecurityNumber;
014     private String department;
015     private String position;
016     private Date hireDate;
017     private Double salary;
018     private Employee supervisor;
019     private List<String> phoneNumbers;
020
021     public Employee() {
022     }
023
024     public Employee(String firstName, String lastName,
025         String socialSecurityNumber, String department, String position,
026         Date hireDate, Double salary) {
027         this.firstName = firstName;
028         this.lastName = lastName;
029         this.socialSecurityNumber = socialSecurityNumber;
030         this.department = department;
031         this.position = position;
032         this.hireDate = hireDate;
033         this.salary = salary;
034     }
035
036     public String getFirstName() {
037         return firstName;
038     }
039
040     public void setFirstName(String firstName) {
041         this.firstName = firstName;
042     }
043
044     public String getLastName() {
045         return lastName;
046     }
047
048     public void setLastName(String lastName) {
049         this.lastName = lastName;
050     }
051
052     public String getSocialSecurityNumber() {
053         return socialSecurityNumber;
054     }
055
056     public void setSocialSecurityNumber(String socialSecurityNumber) {
057         this.socialSecurityNumber = socialSecurityNumber;
058     }
059
060     public String getDepartment() {
061         return department;
062     }
063
064     public void setDepartment(String department) {
065         this.department = department;
066     }
067
068     public String getPosition() {
069         return position;
070     }
071
072     public void setPosition(String position) {
073         this.position = position;
074     }
075
076     public Date getHireDate() {
077         return hireDate;
078     }
079
080     public void setHireDate(Date hireDate) {
081         this.hireDate = hireDate;
```

```

082 }
083
084 public Double getSalary() {
085     return salary;
086 }
087
088 public void setSalary(Double salary) {
089     this.salary = salary;
090 }
091
092 public Employee getSupervisor() {
093     return supervisor;
094 }
095
096 public void setSupervisor(Employee supervisor) {
097     this.supervisor = supervisor;
098 }
099
100 public List<string> getPhoneNumbers() {
101     return phoneNumbers;
102 }
103
104 public void setPhoneNumbers(List<string> phoneNumbers) {
105     this.phoneNumbers = phoneNumbers;
106 }
107
108 }

```

Things to notice here :

- We assume that the following fields are mandatory :
  - "firstName"
  - "lastName"
  - "socialSecurityNumber"
  - "department"
  - "position"
  - "hireDate"
  - "salary"

Following is the second flavor of the "Employee" object :

```

001 package com.javacodegeeks.test;
002
003 import java.io.Externalizable;
004 import java.io.IOException;
005 import java.io.ObjectInput;
006 import java.io.ObjectOutput;
007 import java.util.Arrays;
008 import java.util.Date;
009 import java.util.List;
010
011 public class Employee implements Externalizable {
012
013     private String firstName;
014     private String lastName;
015     private String socialSecurityNumber;
016     private String department;
017     private String position;
018     private Date hireDate;
019     private Double salary;
020     private Employee supervisor;
021     private List<string> phoneNumbers;
022
023     public Employee() {
024     }
025
026     public Employee(String firstName, String lastName,
027         String socialSecurityNumber, String department, String position,
028         Date hireDate, Double salary) {
029         this.firstName = firstName;
030         this.lastName = lastName;
031         this.socialSecurityNumber = socialSecurityNumber;
032         this.department = department;
033         this.position = position;
034         this.hireDate = hireDate;
035         this.salary = salary;
036     }
037
038     public String getFirstName() {
039         return firstName;
040     }
041
042     public void setFirstName(String firstName) {
043         this.firstName = firstName;
044     }
045
046     public String getLastName() {
047         return lastName;
048     }
049
050     public void setLastName(String lastName) {
051         this.lastName = lastName;
052     }
053
054     public String getSocialSecurityNumber() {
055         return socialSecurityNumber;
056     }

```

```

057
058 public void setSocialSecurityNumber(String socialSecurityNumber) {
059     this.socialSecurityNumber = socialSecurityNumber;
060 }
061
062 public String getDepartment() {
063     return department;
064 }
065
066 public void setDepartment(String department) {
067     this.department = department;
068 }
069
070 public String getPosition() {
071     return position;
072 }
073
074 public void setPosition(String position) {
075     this.position = position;
076 }
077
078 public Date getHireDate() {
079     return hireDate;
080 }
081
082 public void setHireDate(Date hireDate) {
083     this.hireDate = hireDate;
084 }
085
086 public Double getSalary() {
087     return salary;
088 }
089
090 public void setSalary(Double salary) {
091     this.salary = salary;
092 }
093
094 public Employee getSupervisor() {
095     return supervisor;
096 }
097
098 public void setSupervisor(Employee supervisor) {
099     this.supervisor = supervisor;
100 }
101
102 public List<String> getPhoneNumbers() {
103     return phoneNumbers;
104 }
105
106 public void setPhoneNumbers(List<String> phoneNumbers) {
107     this.phoneNumbers = phoneNumbers;
108 }
109
110 public void readExternal(ObjectInput objectInput) throws IOException,
111     ClassNotFoundException {
112
113     this.firstName = objectInput.readUTF();
114     this.lastName = objectInput.readUTF();
115     this.socialSecurityNumber = objectInput.readUTF();
116     this.department = objectInput.readUTF();
117     this.position = objectInput.readUTF();
118     this.hireDate = new Date(objectInput.readLong());
119     this.salary = objectInput.readDouble();
120
121     int attributeCount = objectInput.read();
122
123     byte[] attributes = new byte[attributeCount];
124
125     objectInput.readFully(attributes);
126
127     for (int i = 0; i < attributeCount; i++) {
128         byte attribute = attributes[i];
129
130         switch (attribute) {
131             case (byte) 0:
132                 this.supervisor = (Employee) objectInput.readObject();
133                 break;
134             case (byte) 1:
135                 this.phoneNumbers = Arrays.asList(objectInput.readUTF().split(";"));
136                 break;
137         }
138     }
139
140 }
141
142 public void writeExternal(ObjectOutput objectOutput) throws IOException {
143
144     objectOutput.writeUTF(firstName);
145     objectOutput.writeUTF(lastName);
146     objectOutput.writeUTF(socialSecurityNumber);
147     objectOutput.writeUTF(department);
148     objectOutput.writeUTF(position);
149     objectOutput.writeLong(hireDate.getTime());
150     objectOutput.writeDouble(salary);
151
152     byte[] attributeFlags = new byte[2];
153
154     int attributeCount = 0;
155
156     if (supervisor != null) {
157         attributeFlags[0] = (byte) 1;
158         attributeCount++;
159     }
160     if (phoneNumbers != null && !phoneNumbers.isEmpty()) {
161         attributeFlags[1] = (byte) 1;

```

✖

✖

```

162     attributeCount++;
163 }
164
165 objectOutput.write(attributeCount);
166
167 byte[] attributes = new byte[attributeCount];
168
169 int j = attributeCount;
170
171 for (int i = 0; i < 2; i++)
172     if (attributeFlags[i] == (byte) 1) {
173         j--;
174         attributes[j] = (byte) i;
175     }
176
177 objectOutput.write(attributes);
178
179 for (int i = 0; i < attributeCount; i++) {
180     byte attribute = attributes[i];
181
182     switch (attribute) {
183     case (byte) 0:
184         objectOutput.writeObject(supervisor);
185         break;
186     case (byte) 1:
187         StringBuilder rowPhoneNumbers = new StringBuilder();
188         for(int k = 0; k < phoneNumbers.size(); k++)
189             rowPhoneNumbers.append(phoneNumbers.get(k) + ";");
190         rowPhoneNumbers.deleteCharAt(rowPhoneNumbers.lastIndexOf(";"));
191         objectOutput.writeUTF(rowPhoneNumbers.toString());
192         break;
193     }
194 }
195 }
196 }
197 }

```

Things to notice here :

- We implement the "writeExternal" method for marshalling the "Employee" object. All mandatory fields are written to the stream
- For the "hireDate" field we write only the number of milliseconds represented by this Date object. Assuming that the demarshaller will be using the same timezone as the marshaller the milliseconds value is all the information we need to properly deserialize the "hireDate" field. Keep in mind that we could serialize the entire "hireDate" object by using the "objectOutput.writeObject(hireDate)" operation. In that case the default serialization mechanism would kick in resulting in speed degradation and size increment for the resulting stream
- All the non mandatory fields ("supervisor" and "phoneNumbers") are written to the stream only when they have actual (not null) values. To implement this functionality we use the "attributeFlags" and "attributes" byte arrays. Each position of the "attributeFlags" array represents a non mandatory field and holds a "marker" indicating whether the specific field has a value. We check each non mandatory field and populate the "attributeFlags" byte array with the corresponding markers. The "attributes" byte array indicates the actual non mandatory fields that must be written to the stream by means of "position". For example if both "supervisor" and "phoneNumbers" non mandatory fields have actual values then "attributeFlags" byte array should be [1,1] and "attributes" byte array should be [0,1]. In case only "phoneNumbers" non mandatory field has a non null value "attributeFlags" byte array should be [0,1] and "attributes" byte array should be [1]. By using the aforementioned algorithm we can achieve minimal size footprint for the resulting stream. To properly deserialize the "Employee" object non mandatory parameters we must write to the steam only the following information :
  - The overall number of non mandatory parameters that will be written (aka the "attributes" byte array size – for the demarshaller to parse)
  - The "attributes" byte array (for the demarshaller to properly assign field values)
  - The actual non mandatory parameter values
- For the "phoneNumbers" field we construct and write to the stream a String representation of its contents. Alternatively we could serialize the entire "phoneNumbers" object by using the "objectOutput.writeObject(phoneNumbers)" operation. In that case the default serialization mechanism would kick in resulting in speed degradation and size increment for the resulting stream
- We implement the "readExternal" method for demarshalling the "Employee" object. All mandatory fields are written to the stream. For the non mandatory fields the demarshaller assigns the appropriate field values according to the protocol described above

For the serialization and deserialization processes we used the following four functions. These functions come in two flavors. The first pair is suitable for serializing and deserializing Externalizable object instances, whereas the second pair is suitable for serializing and deserializing Serializable object instances.

```

01 public static byte[][] serializeObject(Externalizable object) throws Exception {
02     ByteArrayOutputStream baos = null;
03     ObjectOutputStream oos = null;
04     byte[][] res = new byte[2][];
05
06     try {
07         baos = new ByteArrayOutputStream();
08         oos = new ObjectOutputStream(baos);
09
10         object.writeExternal(oos);
11         oos.flush();
12
13         res[0] = object.getClass().getName().getBytes();
14         res[1] = baos.toByteArray();
15
16     } catch (Exception ex) {
17         throw ex;
18     } finally {
19         try {
20             if(oos != null)
21                 oos.close();
22         } catch (Exception e) {

```

```

23     e.printStackTrace();
24 }
25 }
26
27 return res;
28 }

```

```

01 public static Externalizable deserializeObject(byte[][] rowObject) throws Exception {
02     ObjectInputStream ois = null;
03     String objectClassName = null;
04     Externalizable res = null;
05
06     try {
07
08         objectClassName = new String(rowObject[0]);
09         byte[] objectBytes = rowObject[1];
10
11         ois = new ObjectInputStream(new ByteArrayInputStream(objectBytes));
12
13         Class objectClass = Class.forName(objectClassName);
14         res = (Externalizable) objectClass.newInstance();
15         res.readExternal(ois);
16
17     } catch (Exception ex) {
18         throw ex;
19     } finally {
20         try {
21             if(ois != null)
22                 ois.close();
23         } catch (Exception e) {
24             e.printStackTrace();
25         }
26     }
27
28     return res;
29
30
31 }

```

```

01 public static byte[] serializeObject(Serializable object) throws Exception {
02     ByteArrayOutputStream baos = null;
03     ObjectOutputStream oos = null;
04     byte[] res = null;
05
06     try {
07         baos = new ByteArrayOutputStream();
08         oos = new ObjectOutputStream(baos);
09
10         oos.writeObject(object);
11         oos.flush();
12
13         res = baos.toByteArray();
14
15     } catch (Exception ex) {
16         throw ex;
17     } finally {
18         try {
19             if(oos != null)
20                 oos.close();
21         } catch (Exception e) {
22             e.printStackTrace();
23         }
24     }
25
26     return res;
27 }

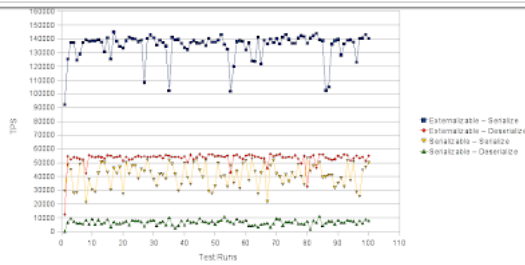
```

```

01 public static Serializable deserializeObject(byte[] rowObject) throws Exception {
02     ObjectInputStream ois = null;
03     Serializable res = null;
04
05     try {
06
07         ois = new ObjectInputStream(new ByteArrayInputStream(rowObject));
08         res = (Serializable) ois.readObject();
09
10     } catch (Exception ex) {
11         throw ex;
12     } finally {
13         try {
14             if(ois != null)
15                 ois.close();
16         } catch (Exception e) {
17             e.printStackTrace();
18         }
19     }
20
21
22     return res;
23
24 }

```

Below we present a performance comparison chart between the two aforementioned approaches



The horizontal axis represents the number of test runs and the vertical axis the average transactions per second (TPS) for each test run. Thus higher values are better. As you can see by using the Externalizable approach you can achieve superior performance gains when serializing and deserializing compared to the plain Serializable approach.

Lastly we must pinpoint that we performed our tests providing values for all non mandatory fields of the "Employee" object. You should expect even higher performance gains if you do not use all the non mandatory parameters for your tests, either when comparing between the same approach and most importantly when cross comparing between the Externalizable and Serializable approaches.

Happy coding!

Justin

#### Related Articles :

- Java Best Practices – DateFormat in a Multithreading Environment
- Java Best Practices – Vector vs ArrayList vs HashSet
- Java Best Practices – String performance and Exact String Matching
- Java Best Practices – Queue battle and the Linked ConcurrentHashMap
- Java Best Practices – Char to Byte and Byte to Char conversions

Tagged with: [JAVA BEST PRACTICES](#) [SERIALIZATION](#)

(0 rating, 0 votes)

You need to be a registered member to rate this. 8 Comments 938 Views Tweet it!

## Do you want to know how to develop your skillset to become a **Java Rockstar**?



Subscribe to our newsletter to start Rocking right now!  
To get you started we give you our best selling eBooks for **FREE!**

1. JPA Mini Book
2. JVM Troubleshooting Guide
3. JUnit Tutorial for Unit Testing
4. Java Annotations Tutorial
5. Java Interview Questions
6. Spring Interview Questions
7. Android UI Design

and many more ....

☐ I agree to the Terms and Privacy Policy

LIKE THIS ARTICLE? READ MORE FROM JAVA CODE GEEKS

8 Leave a Reply



Join the discussion...

Subscribe ▾

▲ newest ▲ oldest ▲ most voted



Proagent



thanks for this nice article,  
can you tell us what the tool you used for performance comparison ?

+ 0 - [Reply](#)

🕒 5 years ago ▲



Byron Kiourtzoglou



I have created a in-house performance measurement app

Guest

+ 0 - [Reply](#)

🕒 5 years ago ▲



Proagent



cool , its open source ?

Guest

+ 0 - [Reply](#)

🕒 5 years ago ▲



Byron Kiourtzoglou



Unfortunately no, its a proprietary code, but you can make your own very easily!

Regards

Guest

+ 0 - [Reply](#)

🕒 5 years ago



Andre



Tweaked your code a bit, this is 4 times faster roughly ... `package com.javacodegeeks.test;; import java.io.ByteArrayInputStream; import java.io.ByteArrayOutputStream; import java.io.DataInputStream; import java.io.DataOutputStream; import java.io.IOException; public class DataMessageTransmission_test { private DataMessageTransmission_test employee = this; private String firstName; private String lastName; private String socialSecurityNumber; private String department; private String position; private long hireDate; private Double salary; private String supervisor; private String[] phoneNumbers; private static byte[][] serial; public DataMessageTransmission_test() {} public DataMessageTransmission_test(String firstName, String lastName,String socialSecurityNumber, String department, String position, long hireDate, Double salary) { employee.firstName = firstName; employee.lastName = lastName; employee.socialSecurityNumber = socialSecurityNumber; employee.department = department; employee.position = position; employee.hireDate = hireDate; employee.salary...` [Read more »](#)

+ 0 - [Reply](#)

🕒 5 years ago ▲



Charles



Hello Andre,

Could you please email your code to play with? The one posted seems having some problem....  
Thanks in advance.

Charles\_L\_chan (at) me (dot) com

Guest

+ 0 - [Reply](#)

🕒 5 years ago



Rüdiger Möller



You should checkout <https://code.google.com/p/fast-serialization/> . This library outperforms manual serialization in many cases.

+ 0 - [Reply](#)

🕒 5 years ago



Johan Parent





Guest

Hi Justin, Great article! You show some useful techniques to use the ObjectOutputStream and ObjectInputStream APIs to get the best out of the JDK serialization algorithm. Having tried fast-serialization myself I can confirm it is indeed very fast. Great piece of code. Application servers and other JavaEE technologies do not always allow the use of an alternative serialization mechanism. If you are bound to the default JDK serialization you may want to take a look at Externalizer4J. It optimizes the serialization using techniques similar to the ones described in this post. But it does so automatically by analyzing the... [Read more »](#)

+ 0 -

[Reply](#)

🕒 3 years ago



## KNOWLEDGE BASE

[Courses](#)

[Examples](#)

[Minibooks](#)

[Resources](#)

[Tutorials](#)

## PARTNERS

[Mkyong](#)

## THE CODE GEEKS NETWORK

[.NET Code Geeks](#)

[Java Code Geeks](#)

[System Code Geeks](#)

[Web Code Geeks](#)

## HALL OF FAME

["Android Full Application Tutorial" series](#)

[11 Online Learning websites that you should check out](#)

[Advantages and Disadvantages of Cloud Computing – Cloud computing pros and cons](#)

[Android Google Maps Tutorial](#)

[Android JSON Parsing with Gson Tutorial](#)

[Android Location Based Services Application – GPS location](#)

[Android Quick Preferences Tutorial](#)

[Difference between Comparator and Comparable in Java](#)

[GWT 2 Spring 3 JPA 2 Hibernate 3.5 Tutorial](#)

[Java Best Practices – Vector vs ArrayList vs HashSet](#)

## ABOUT JAVA CODE GEEKS

JCGs (Java Code Geeks) is an independent online community focused on creating the ultimate Java to Java developers resource center; targeted at the technical and non-technical team lead (senior developer), project manager and junior developer. JCGs serve the Java, SOA, Agile and Telecom communities with daily news written by domain experts, articles, tutorials, reviews, announcements, code snippets and source projects.

## DISCLAIMER

All trademarks and registered trademarks appearing on Java Code Geeks are the property of their respective owners. Java is a trademark or registered trademark of Oracle Corporation in the United States and other countries. Examples Java Code Geeks is not connected to Oracle Corporation and is not sponsored by Oracle Corporation.

