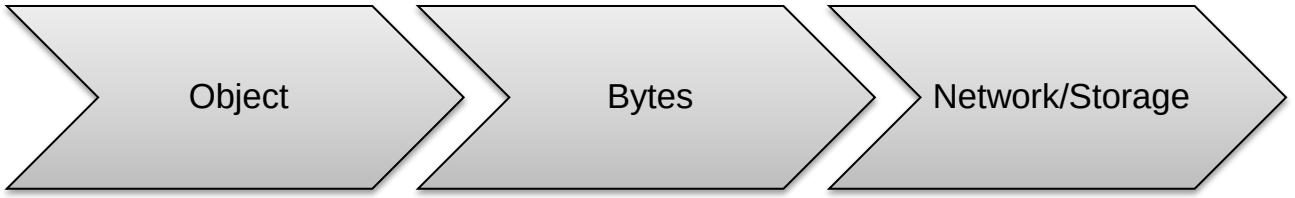
Java Serialization

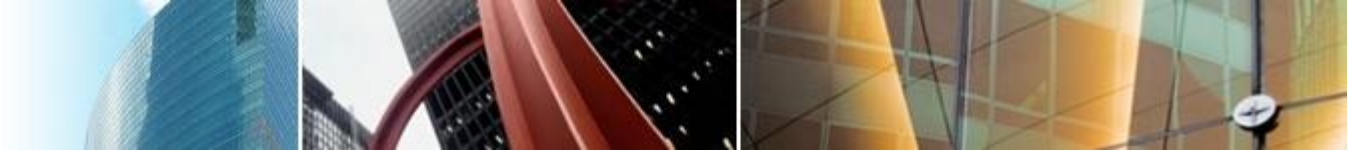
Facts and Fallacies

Serialization

… is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer, or transmitted across a network connection link) and resurrected later in the same or another computer environment.

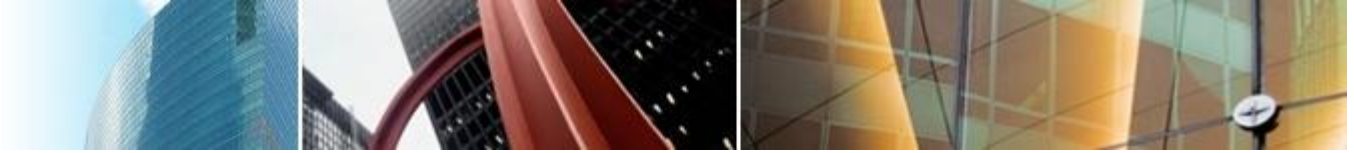
-- WIkipedia





Distributed system use-case

- Transfer data between cluster nodes or between servers and clients
- **Serialization** is a key technology behind any RPC/RMI
- Serialization if often used for **storage**, but then data transfer is still part of the picture



Java serialization facility (key facts)

- Had appeared in **1997** as part of JDK 1.1 release
- Uses **java.io.Serializable** as a marker interface
- Consists of **java.io.ObjectInputStream/ObjectOutputStream** with related classes and interfaces
- Has “**Java Object Serialization Specification”** that documents both API and object serialization stream protocol
- Is part of any Java platform (even Android has it)
- Somehow has a lot of myths around it

~~JEP 154: Remove Serialization~~

Author	Alan Bateman
Organization	Oracle
Created	2012/4/1
Updated	2012/12/4
Type	Feature
State	Withdrawn
Component	core/libs
Scope	SE
Internal-refs	Oracle:jplan:417072696c466f6f6c
Discussion	core dash libs dash dev at openjdk dot java dot net
Effort	M
Duration	L
Endorsed-by	Brian Goetz

It is a joke

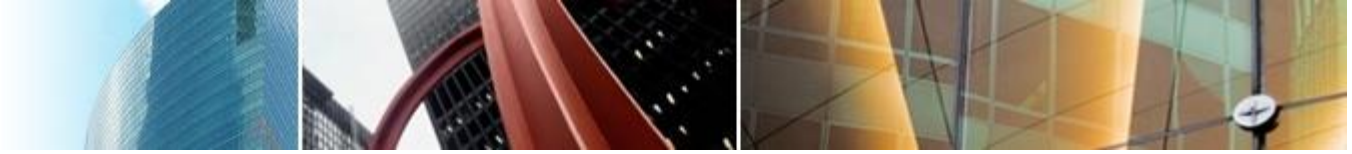
… and the only JEP joke (!)

… and it has a big grain of truth:

There are a lot of folk myths about serialization

DEVEXPERTS

SERIALIZATION



Hazelcast Portable

Google protobuf

Coherence POF

Hessian

Hadoop Writable

Protostuff

JBoss serialization

Kryo

quickser

fast-serialization

Apache Avro

GridGain serialization

jserial

Apache Thrift

msgpack

wobly

SERIALIZATION

Gson

EclipseLink MOXy

Burlap

Json-lib

JAXB

XStream

FLEXJSON

Javolution XML

Jackson

svenson

java.beans.XMLEncoder/XMLDecoder

jsonij

Obvious fact of live

Programming your own
serialization [framework] is fun!
… and gets you the highest-performance and
tailored solution to your particular problem.

Real hackers write everything from scratch ☺

Every mature appserver/framework/cache/eventbus/soa/rpc/db/etc must have its own serialization that is better than everybody else's ☺

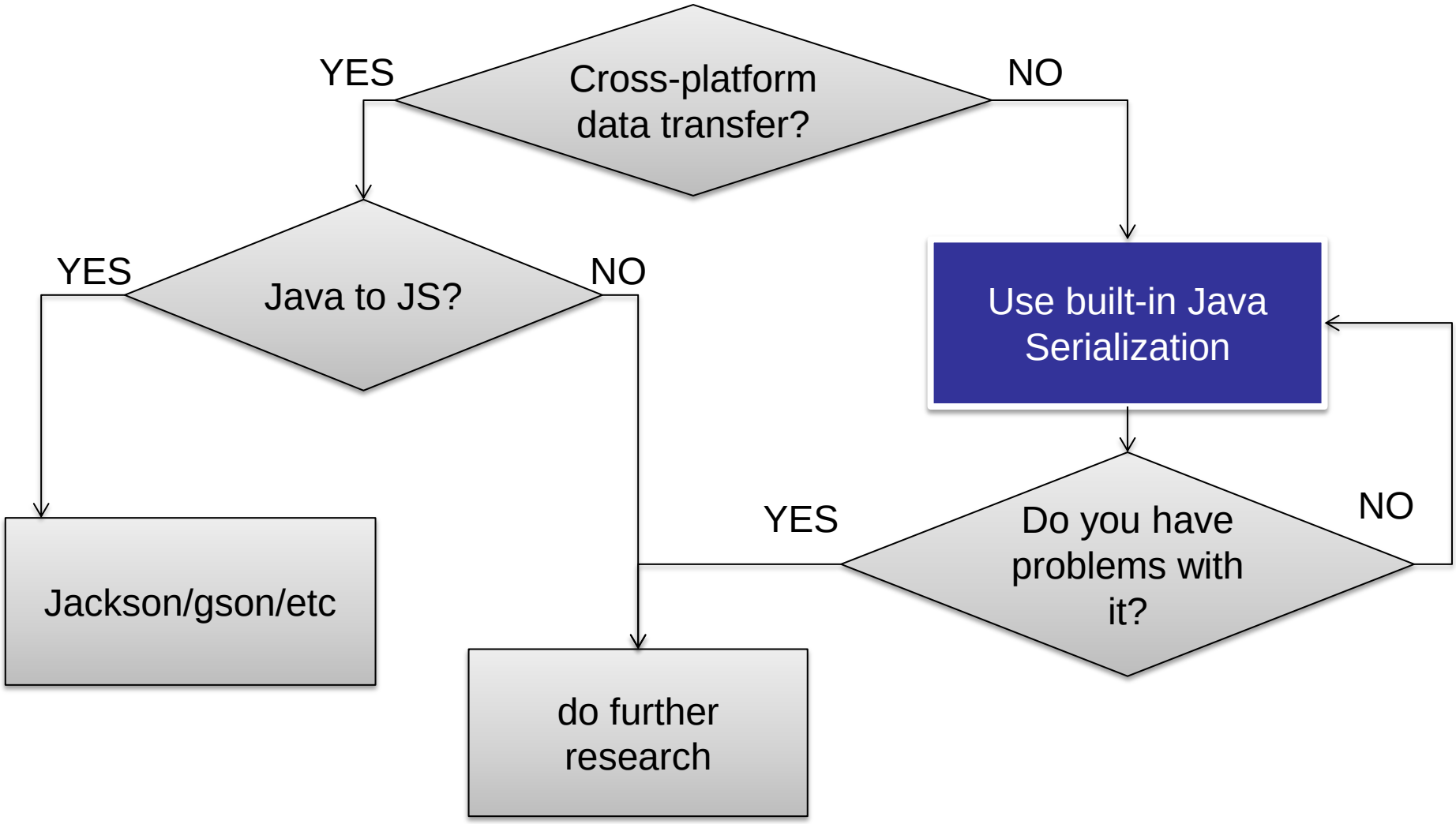
How to choose a serialization lib?

- Cross-language or Java-only
- Binary or text (XML/JSON/other) format
- Custom or automatic object-to-serialized format mapping
- Needs format description/mapping/annotation or not
- Writes object scheme (e.g. field names) with each object, once per request, once per session, never at all, or some mixed approach
- Writes object fields or bean properties
- Performance
  - CPU consumption to serialize/deserialize
  - Network consumption (bytes per object)

Java built-in serialization lib?

- Cross-language or Java-only
- Binary or text (XML/JSON/other) format
- Custom or automatic object-to-serialized format mapping
- Needs format description/mapping/annotation or not
- Writes object scheme (e.g. field names) with each object, once per request, once per session, never at all, or some mixed approach
- Writes object fields or bean properties
- Performance
  - CPU consumption to serialize/deserialize
  - Network consumption (bytes per object)

Serious serialization lib choice diagram



Serialization myth #1

I have this framework XYZ…

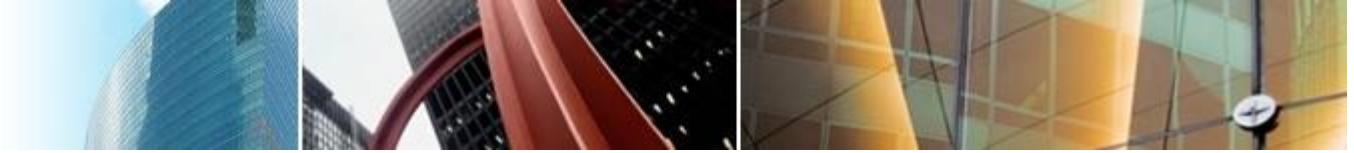
… it has really **simple** and **fast** serialization

All you have to do is to implement this interface! Easy!

```
public interface XYZSerializable {
    public void write(DataOutput out) throws IOException;
    public void read(DataInput in) throws IOException;
}
```

I've tested it! It is really fast!
It outperforms everything else I've tried!

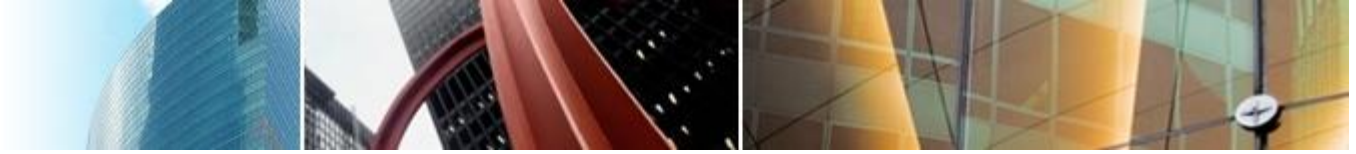
/* This is almost actual real-life conversation.
 * Names changed */



In fact this is corollary to “Obvious fact of life”

- This does not solve the most complex serialization problems
  - You can implement **java.io.Externalizable** if you are up to writing a custom serialization code for each object.
- Maintenance cost/efforts are often underestimated
  - How you are planning to evolve you system? Add new fields? Classes?
    - What will keep you from forgetting to read/write them?
    - Ah… yes, you write a lot of extra tests in addition to writing your write/read methods to make sure your serialization works
  - How about on-the-wire compatibility of old code and new one?
    - But I deploy all of my system with a new version at once!
    - But how do you work with it during development?

DEVEXPERTS



So what about this code evolution?

```
public class Order implements Serializable {
    private String symbol;
    private double quantity;
    private double price;

    // ... methods follow ...
```

evolve

write

read

got

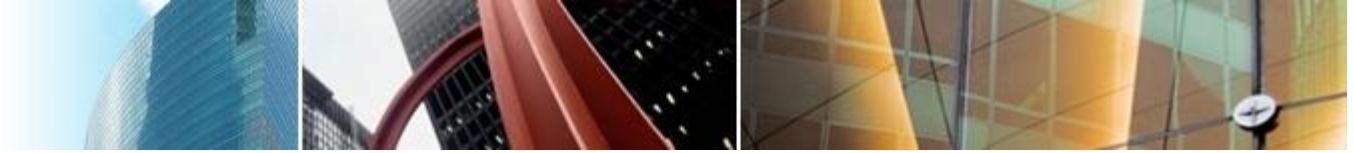
```
public class Order implements Serializable {
    private String symbol;
    private double quantity;
    private double price;
    private OrderType orderType;

    // ... methods follow ...
```

```
java.io.InvalidClassException: ser.Order; local class incompatible: stream classdesc serialVers
    at java.io.ObjectStreamClass.initNonProxy(ObjectStreamClass.java:617)
    at java.io.ObjectInputStream.readNonProxyDesc(ObjectInputStream.java:1620)
    at java.io.ObjectInputStream.readClassDesc(ObjectInputStream.java:1515)
    at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1769)
    at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1348)
    at java.io.ObjectInputStream.readObject(ObjectInputStream.java:370)
```

serialVersionUID

- **serialVersionUID** = crazy hash of your class
  - If you change your class -> serialVersionUID change -> incompatible
- But you can set **serialVersionUID** as a "private static final long" field in your class to fix it.
- With a fixed **serialVersionUID** java serialization supports:
  - Adding/removing fields without any additional hurdles and annotations
  - Moving fields around in class definition
  - Extracting/inlining super-classes (but without moving serializable fields up/down class hierarchy).
- It basically means you can evolve your class at will, just a you would do with key-value map in json/other-untyped-system



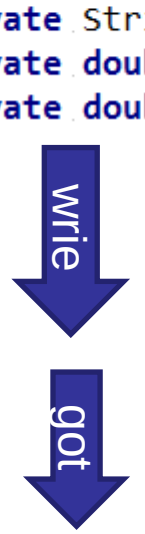
serialver

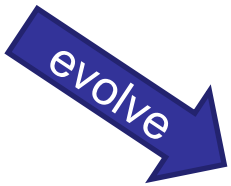
- That's the tool to compute **serialVersionUID** as the crazy hash of your class
  - You only need it when your class is "in the wild" w/o explicitly assigned **serialVersionUID** and you want to be backwards compatible with that version of it
- **DO NOT USE serialver** for freshly created classes
  - It is a pseudo random number that serves no purpose, but to make your gzipped class files and gzipped serial files larger
  - Just set **serialVersionUID = 0**
    - It is no worse, but better than other values

So what about this code evolution (2)?

```
public class Order implements Serializable {
    private static final long serialVersionUID = 0;

    private String symbol;
    private double quantity;
    private double price;
```






```
public class Order implements Serializable {
    private static final long serialVersionUID = 0;

    private String symbol;
    private double quantity;
    private double price;
    private OrderType orderType;
```

symbol – as written
quantity – as written
price – as written
orderType – **null**

It works the other way around, too!

What if I want to make my class incompatible?

- You have two choices
  - serialVersionUID++
  - Rename class
- So, having serialVersionUID in a modern serialization framework is an optional feature
  - They just did not have refactorings back then in 1996, so renaming a class was not an option for them
  - They chose a default of “your changes break serialization” and force you to explicitly declare serialVersionUID if you want “your changes are backwards compatible”
    - I would have preferred the other default, but having to manually add serialVersionUID is a small price to pay for a serialization facility that you have out-of-the box in Java.

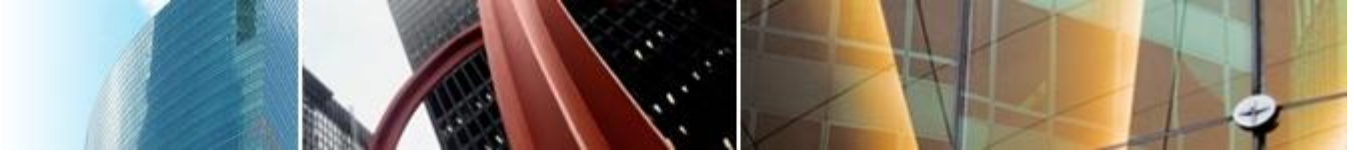
Complex changes made compatible

- Java serialization has a bunch of features to support radical evolution of serializable classes in backwards-compatible way:
  - **writeReplace/readResolve**
    - Lets you migrate your code to a completely new class, but still use the old class for serialization.
  - **putFields/writeFields/readFields**
    Lets you complete redo the fields of the object, but still be compatible with the old set of fields for serialization.

It really helps if you are working on, evolving, improving, and refactoring really big projects

Serialization myth #2

Serialization performance matters



In practice

- It really matters when very verbose text formats like XML are used in a combination with not-so-fast serializer implementations
- Most binary formats are fast and compact enough, so that they do not constitute a bottleneck in the system
  - Data is usually expensive to acquire (compute, fetch from db, etc)
  - Otherwise (if data is cache) and the only action of the system is to retrieve it from cache, then cache serialized data

Popular myth #3

Java's standard serialization
is known to be slow,
and to use a huge
amount of bytes on disk

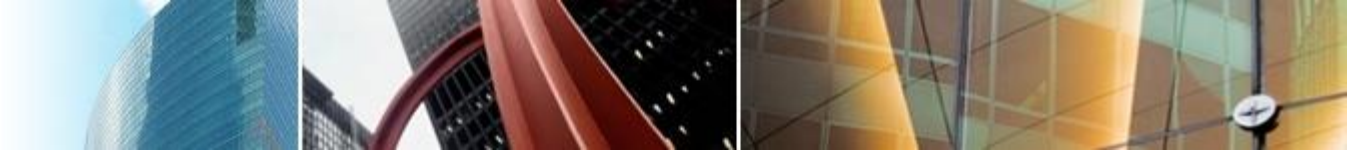
© Stack Overflow authors, emphasis is mine

The source of this myth is a popular way to benchmark serialization performance

```
public T deserialize(byte[] array) throws Exception
{
    ObjectInputStream ois = new ObjectInputStream(new ByteArrayInputStream(array));
    /unchecked/
    T v = (T) ois.readObject();
    return v;
}

public byte[] serialize(T data) throws IOException
{
    ByteArrayOutputStream baos = outputStream(data);
    ObjectOutputStream oos = new ObjectOutputStream(baos);
    oos.writeObject(data);
    return baos.toByteArray();
}
```

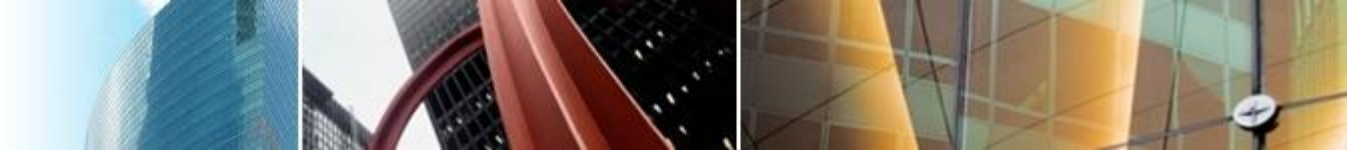
https://github.com/eishay/jvm-serializers/blob/master/tpc/src/serializers/JavaBuiltIn.java



The real fact about serialization performance

- ObjectOutputStream/ObjectInputStream are expensive to create
  - e.g. slow
- Reuse oos/ois instances
  - It is easy when you write a single stream of values
    - Just keep doing writeObject(!)
    - That is what it was designed for
  - It is slightly tricky if you want to optimize it for one-of tasks

TBD: Reuse code



TBD: More on performance

46000
Users on-line in one day

305400
Sent orders in one day

We create professional financial software for Brokerage and Exchanges since 2002

DEVEXPERTS

99.99%
Our software index
of trouble-free operation

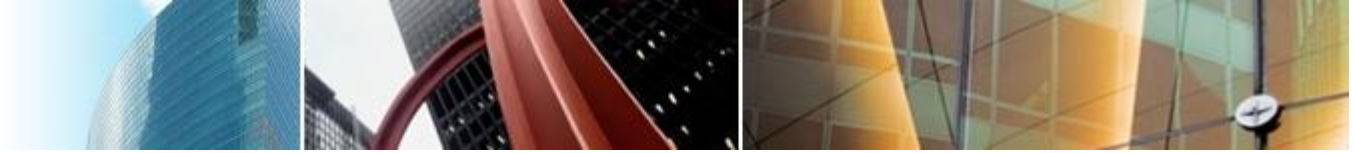
365 7 24
We support our products
around-the-clock

350
Peoples in our team

DEVEXPERTS

Headquarters
197110, 10/1 Barochnaya st.
Saint Petersburg, Russia
+7 812 438 16 26
mail@devexperts.com

www.devexperts.com



Thank you! Questions?

Contact me by email: elizarov at devexperts.com
Read my blog: http://elizarov.livejournal.com/