Java serialization, cloning, and casting interview Q&A

Posted on August 16, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

Q1. Which Java interface must be implemented by a class whose instances are transported via a Web service?

a. Accessible
b. BeanInfo
c. Remote
d. Serializable

A1. Answer is “d”.

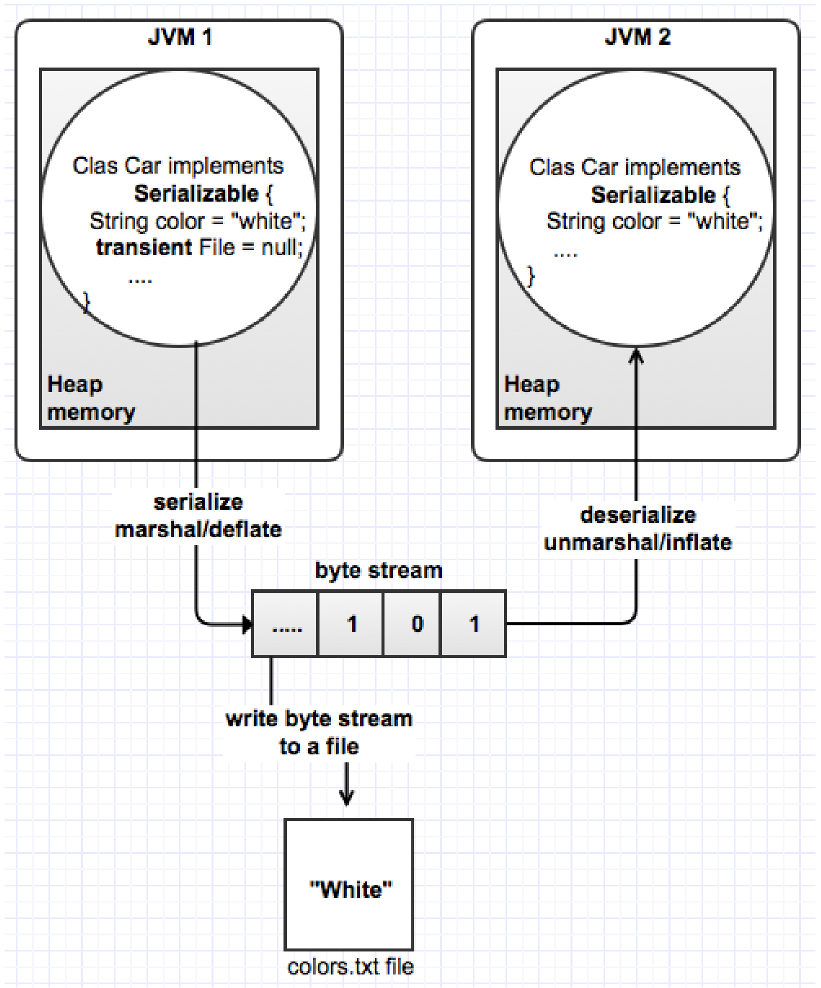
Q2. What is serialization?
A2. Object serialization is a process of reading or writing an object. It is a process of saving an **object’s state to a sequence of bytes**, as well as a process of **rebuilding**

9 tips to earn more | *What can u do to go places?* | ***945+*** *members.* *LinkedIn Group*. ***Reviews***

600+ Full Stack Java/JEE Interview Q&As ♥Free ♦FAQs

open all | *close all*
⊞ *Ice Breaker Interview*
⊟ *Core Java Interview (*
  ⊞ *Java Overview (4)*
  ⊞ *Data types (6)*
  ⊞ *constructors-methc*
  ⊞ *Reserved Key Wor*
  ⊞ *Classes (3)*
  ⊟ *Objects (8)*
    ► *Beginner Java*
    ♥♦ *HashMap & H*
    ♦ *5 Java Object*

those bytes back into a live object at some future time. This happens in between two different processes (i.e. JVM or heap memory). So, you can't serialize non memory resources like file handles, sockets, threads, etc.



Java Serialization

An object is marked serializable by implementing the java.io.Serializable interface. This simply allows the serialization mechanism to verify that a class can be persisted, typically to a file. The common process of serialization is also called marshaling or deflating when an object is flattened into byte streams. The flattened byte streams can be unmarshaled or inflated back to an object.

To persist objects, you need to keep 5 rules in mind:

♦ Java enum inte
♦ Java immutabl
♦♥ Object equals
Java serializatio
Mocks, stubs, do
OOP (10)
GC (2)
Generics (5)
FP (8)
IO (7)
Multithreading (12)
Algorithms (5)
Annotations (2)
Collection and Data
Differences Betwee
Event Driven Progr
Exceptions (2)
Java 7 (2)
Java 8 (24)
JVM (6)
Reactive Programm
Swing & AWT (2)
JEE Interview Q&A (3
Pressed for time? Jav
SQL, XML, UML, JSO
Hadoop & BigData Int
Java Architecture Inte
Scala Interview Q&As
Spring, Hibernate, & I
Testing & Profiling/Sa
Other Interview Q&A
Free Java Intervie

As a Java Architect

Java architecture & design concepts interview Q&As with diagrams | What should

Rule #1: The object to be persisted must implement the Serializable interface or inherit that interface from its object hierarchy. Alternatively, you can use an Externalizable interface to have full control over your serialization process. For example, to construct an object from a pdf file.

Rule #2: The object to be persisted must mark all non-serializable fields as transient. For example, file handles, sockets, threads, etc.

Rule #3: You should make sure that all the included objects are also serializable. If any of the objects is not serializable, then it throws a NotSerializableException. In the example shown below, the Pet class implements the Serializable interface, and also the containing field types String and Integer are also serializable.

Rule #4: Base or parent class fields are only handled if the base class itself is serializable.

Rule #5: Serialization ignores static fields, because they are not part of any particular state.

Q3.How would you exclude a field of a class from serialization?
A3.By marking it as **transient**. The fields marked as transient in a serializable object will not be transmitted in the byte stream. An example would be a file handle, a database connection, a system thread, etc. Such objects are only meaningful locally. So they should be marked as transient in a Serializable class.

Q4. What happens to static fields during serialization?
A4. erialization persists only the state of a single object. Static fields are not part of the state of an object – they're effectively the state of the class shared by many other instances.

Q5. What are the common uses of serialization? Can you give me an instance where you used serialization?

be a typical Java EE architecture?

Senior Java developers must have a good handle on

open all | close all
⊞ Best Practice (6)
⊞ Coding (26)
⊞ Concurrency (6)
⊞ Design Concepts (7)
⊞ Design Patterns (11)
⊞ Exception Handling (
⊞ Java Debugging (21)
⊞ Judging Experience I
⊞ Low Latency (7)
⊞ Memory Managemen
⊞ Performance (13)
⊞ QoS (8)
⊞ Scalability (4)
⊞ SDLC (6)
⊞ Security (13)
⊞ Transaction Managen

80+ step by step Java Tutorials

open all | close all
⊞ Setting up Tutorial (6)
⊞ Tutorial - Diagnosis (
⊞ Akka Tutorial (9)
⊞ Core Java Tutorials (
⊞ Hadoop & Spark Tuto
⊞ JEE Tutorials (19)
⊞ Scala Tutorials (1)

A5.

1. allows you to **persist objects with state to a text file** on a disk, and re-assemble them by reading this back in. Application servers can do this to conserve memory. For example, stateful EJBs can be activated and passivated using serialization. The objects stored in an HTTP session should be Serializable to support in-memory replication of sessions to achieve scalability.

2. Allows you to **send objects from one Java process to another** using sockets, RMI, RPC, Web service, etc.

3. allows you to **deeply clone** any arbitrary object graph.

Q6. What is a serial version id?
A6. Say you create a "Pet" class, and instantiate it to "myPet", and write it out to an object stream. This flattened "myPet" object sits in the file system for some time. Meanwhile, if the "Pet" class is modified by adding a new field, and later on, when you try to read (i.e. deserialize or inflate) the flattened "Pet" object, you get the java.io.InvalidClassException – because all serializable classes are automatically given a unique identifier, and serial version id has now changed. This exception is thrown when the serial version id of the class is not equal to the serial version id of the flattened object. If you really think about it, the exception is thrown because of the addition of the new field. You can avoid this exception being thrown by controlling the versioning yourself by declaring an explicit serialVersionUID. There is also a small performance benefit in explicitly declaring your serialVersionUID because it does not have to be calculated.

Best practice: So it is a best practice to add your own serialVersionUID to your Serializable classes as soon as you define them. If no serialVersionUID is declared, JVM will use its own algorithm to generate a default SerialVersionUID. The default serialVersionUID computation is highly sensitive to class details and may vary from different JVM

⊞ Spring & HIbernate Tu
⊞ Tools Tutorials (19)
⊞ Other Tutorials (45)

Preparing for Java written & coding tests

open all | close all
⊞ ♦ Complete the given
⊞ Can you write code?
⊞ Converting from A to
⊞ Designing your classe
⊞ Java Data Structures
⊞ Passing the unit tests
⊞ What is wrong with th
⊞ Writing Code Home A
⊞ Written Test Core Jav
⊞ Written Test JEE (1)

How good are your…to go places?

open all | close all
⊞ Career Making Know-
⊞ Job Hunting & Resum

implementation, and result in an unexpected InvalidClassExceptions during deserialization process.

Q7. Are there any disadvantages in using serialization?
A7. Yes. Serialization can adversely affect performance since it:

— Depends on reflection.
— Has an incredibly verbose data format.
— is very easy to send surplus data.

So don’t use serialization if you do not have to.

Q8. What is the difference between Serializable and Externalizable interfaces? How can you customize the serialization process?
A8. An object must implement either Serializable or Externalizable interface before it can be written to a byte stream. When you use Serializable interface, your class is serialized automatically by default. But you can override writeObject(..) and readObject(…) methods to control or customize your object serialization process. For example, you can add the following methods to your Pet class.

```
private void writeObject(ObjectOutputStream out)
         //any write customization goes in this 
        System.out.println("Started writing obje
        out.writeObject(this);
    }

private void readObject(ObjectInputStream in) th
      //any read customization goes in this meth
      System.out.println("Started reading object
        in.readObject( );
   }

```

Note: Both the above methods must be declared private.

No changes are required for FlattenPet and InflatePet classes.

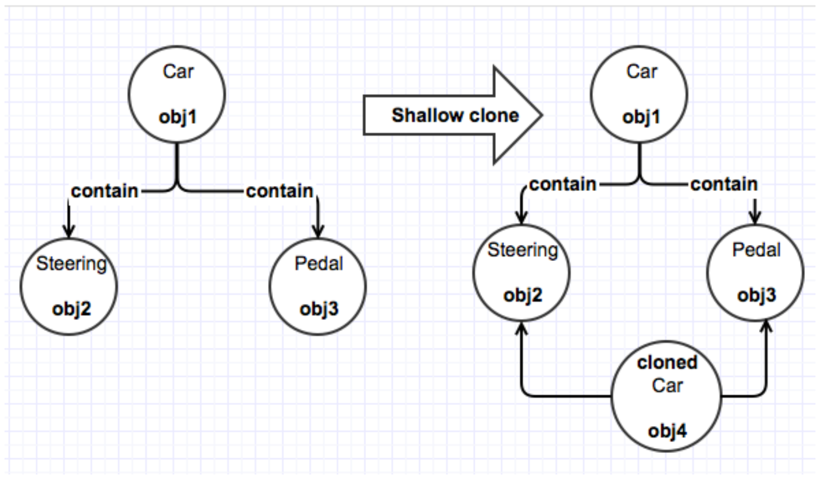
When you use **Externalizable** *interface instead of the Serializable interface, you have a complete control over your* *class's serialization process. This interface contains two methods namely readExternal(…) and writeExternal(…) to achieve this total customization. You can change the Pet class to implement the Externalizable interface and then provide implementation for following 2 methods.*

```

public void writeExternal(ObjectOutput out) throw
public void readExternal(ObjectInput in) throws I

```

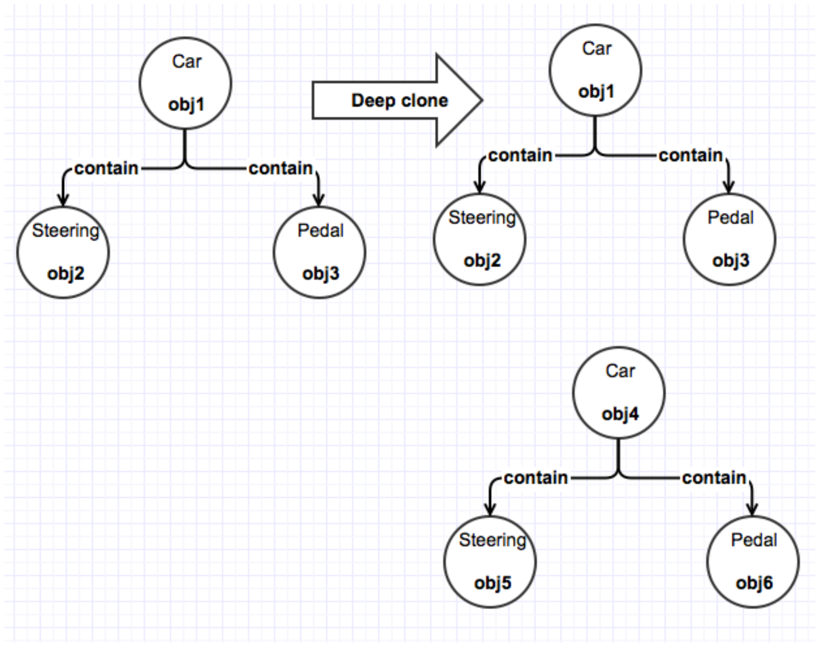
An example situation for this full control will be to read and write PDF files with a Java application. If you know how to write and read PDF (the sequence of bytes required), you could provide the PDF specific protocol in the writeExternal(…) and readExternal(…) methods.

Just as before, there is no difference in how a class that implements Externalizable is used. Just call writeObject( ) or readObject and, those externalizable methods will be called automatically.

Q9. What is the main difference between shallow cloning and deep cloning of objects?
A9. ***Shallow copy:*** *If a shallow copy is performed, the contained objects are not cloned. Java supports shallow cloning of objects by default when a class implements the java.lang.Cloneable interface. For example, invoking clone( ) method on a collection like HashMap, List, etc returns a shallow copy of the HashMap, List, instances. This means if you clone a HashMap, the map instance is cloned but the keys and values themselves are not cloned, but are shared by pointing to the original objects.*



Java shallow cloning



Java deep cloning or copying

***Deep copy**: If a deep copy is performed, then not only the original object has been copied, but the objects contained within it have been copied as well. Serialization can be used to achieve deep cloning. For example, you can serialize a HashMap to a ByteArrayOutputStream and then deserialize it. This creates a deep copy, but does require that all keys and values in the HashMap to be Serializable. The main advantage of this approach is that it will deep copy any arbitrary object graph. Deep cloning through serialization is faster to develop and easier to maintain, but carries a* *performance overhead. Alternatively, you can provide a static factory method to deep copy as shown below:*

```java

public static List deepCopy(List listCars) {
     List copiedList = new ArrayList(10);
     for (Object object : listCars) {
        Car original = (Car)object;
        Car carCopied = new Car( );  //instantia
        carCopied.setColor((original.getColor( )
        copiedList.add(carCopied);
    }
    return copiedList;
}

```

Q10. What is type casting? Explain up casting vs. down casting? When do you get ClassCastException?
A10. Type casting means treating a variable of one type as though it is another type.

***byte** (1 byte) → **short** (2 bytes) → **char** (2 bytes) → **int** (4 bytes) → **long** (8 bytes) → **float** (4 bytes) → **double** (8 bytes)*

***Note**: Want anything larger than long or double? Then look at BigInteger and BigDecimal.*

*When up casting **primitives** from left to right, automatic conversion occurs. But if you go from right to left, down casting or explicit casting is required.*

*When it comes to object references, you can **always cast from a subclass to a super class** because a subclass object is also a super class object. You can cast an object implicitly to a super class type (i.e. up casting). If this were not the case, polymorphism wouldn't be possible.*

You can cast down the hierarchy as well, but you must explicitly write the cast and the object must be a legitimate instance of the class you are casting to. The ClassCastException is thrown to indicate that code has attempted to cast an object to a subclass of which it is not an instance. If you are using JSE 5.0 or later version, then *"**generics**" will minimize the need for casting, and otherwise you can deal with the problem of incorrect down casting in two ways:*

1. Using the exception handling mechanism to catch ClassCastException:

```

Object o = null;
try{
    o = new Integer(1);
    System.out.println((String) o);
}
catch(ClassCastException cce) {
        logger.log("Invalid casting, String is e
        System.out.println(((Integer) o).toStrin
}

```

*2. Using the **instanceof** statement to guard against incorrect casting:*

```

if(o instanceof String) {
   String s2 = (String) o;
}
else if (o instanceof Integer) {
     Integer i2 = (Integer) o;
}


```

The "instanceof" and "typecast" constructs can make your code unmaintainable due to large "if" and "else if" statements, and also can adversely affect performance if used in frequently accessed methods or loops. Look at using generics or visitor design pattern to avoid or minimize these casting constructs where applicable.

***Note**: You can also get a ClassCastException when two different class loaders load the same class because they are treated as two different classes.*

Popular Posts

- ♦ *11 Spring boot interview questions & answers*

857 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

825 views

18 Java scenarios based interview Questions and Answers

447 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

401 views

♦ 7 Java debugging interview questions & answers

311 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

302 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

292 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

286 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

263 views

8 Git Source control system interview questions & answers

215 views

Bio **Latest Posts**

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.

‹ *Java EE interview questions and answers*

♦ *Java immutable objects interview questions & answers* ›

Posted in *member-paid, Objects*

Tags: *Core Java FAQs, Java/JEE FAQs, Novice FAQs*

Leave a Reply

Logged in as geethika. Log out?

Comment

Post Comment

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from "Blah blah" to "Wow"? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category ▼

© Disclaimer