♥♦ Q1-Q10: Top 50+ Core Java Interview Questions & Answers

Posted on March 18, 2015 by Arulkumaran Kumaraswamipillai — 11 Comments ↓

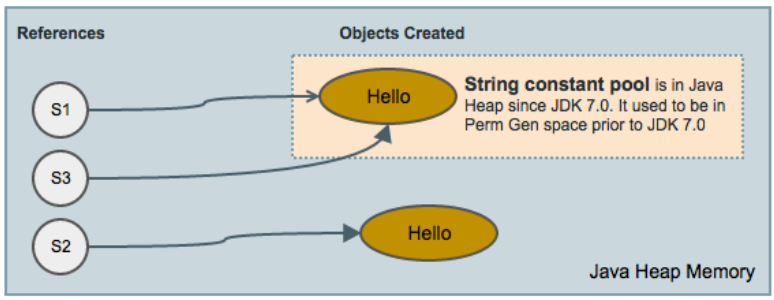
Must know core Java interview questions answered with lots of diagrams & code.

Q1. What is the difference between "==" and "equals(…)" in comparing Java String objects?

A1. When you use "==" (i.e. shallow comparison), you are actually comparing the two object references to see if they point to the same object. When you use "equals(…)", which is a "deep comparison" that compares the actual string values. For example:

```
1  public class StringEquals {
2   public static void main(String[ ] args) {
3     String s1 = "Hello";
4     String s2 = new String(s1);
5     String s3 = "Hello";
6
7     System.out.println(s1 + " equals " + s2 + "--
8
9     System.out.println(s1 + " == " + s2 + " --> "
10    System.out.println(s1 + " == " + s3+ " --> "
11  }
12 }
13
```

The variable **s1** refers to the String instance created by "Hello". The object referred to by **s2** is created with s1 as an initializer, thus the contents of the two String objects are identical, but they are 2 distinct objects having 2 distinct references s1 and s2. This means that s1 and s2 do not refer to the same object and are, therefore, not ==, but *equals( )* as they have the same value "Hello". The s1 == s3 is true, as they both point to the same object due to internal caching. The references s1 and s3 are interned and points to the same object in the string pool.



Create a String object as a literal without the "new" keyword for caching

How about comparing the other objects like Integer, Boolean, and custom objects like "Pet"?
Object **equals** Vs "**==**", and pass by **reference** Vs **value**.

Q2. Can you explain how Strings are interned in Java?
A2. String class is designed with the **Flyweight design pattern** in mind. Flyweight is all about re-usability without having to create too many objects in memory. A pool of

SQL, XML, UML, JSC
Hadoop & BigData Int
Java Architecture Inte
Scala Interview Q&As
Spring, Hibernate, & M
Testing & Profiling/Sa
Other Interview Q&A f
Free Java Interviev

16 Technical Key Areas

open all | close all
Best Practice (6)
Coding (26)
Concurrency (6)
Design Concepts (7)
Design Patterns (11)
Exception Handling (3
Java Debugging (21)
Judging Experience In
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 (2
Akka Tutorial (9)
Core Java Tutorials (2

Strings is maintained by the String class. When the *intern( )* method is invoked, *equals(..)* method is invoked to determine if the String already exist in the pool. If it does then the String from the pool is returned instead of creating a new object. If not already in the string pool, a new String object is added to the pool and a reference to this object is returned. For any two given strings s1 & s2, *s1.intern( ) == s2.intern( )* only if *s1.equals(s2)* is true.

Two String objects are created by the code shown below. Hence s1 == s2 returns false.

```
1 //Two new objects are created. Not interned and n
2 String s1 = new String("A");
3 String s2 = new String("A");
4
```

s1.intern() == s2.intern() returns **true**, but you have to remember to make sure that you actually do intern() all of the strings that you're going to compare. It's easy to forget to intern() all strings and then you can get confusingly incorrect results. Also, why unnecessarily create more objects?

Instead use string literals as shown below to intern automatically:

```
1 String s1 = "A";
2 String s2 = "A";
3
```

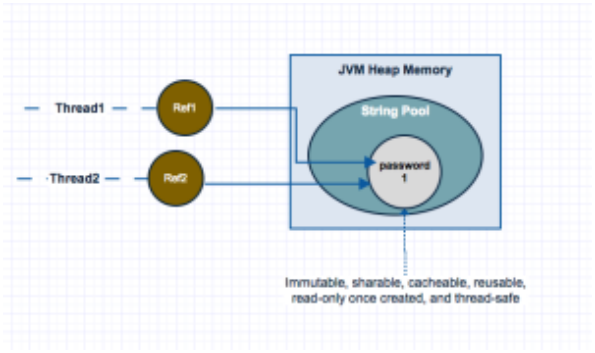
s1 and s2 point to the same String object in the pool. Hence s1 == s2 returns true.

Since interning is automatic for String literals String s1 = "A", the *intern( )* method is to be used on Strings constructed with new String("A").

Q3. Why String class has been made immutable in Java?
A3. For security, performance, and thread-safety.

Hadoop & Spark Tuto
JEE Tutorials (19)
Scala Tutorials (1)
Spring & HIbernate Tu
Tools Tutorials (19)
Other Tutorials (45)

1. Performance: Immutable classes are ideal for representing values of abstract data (i.e. value objects) types like numbers, enumerated types, etc. If you need a different value, create a different object. In Java, *Integer*, *Long*, *Float*, *Character*, *BigInteger* and *BigDecimal* are all immutable objects. Optimization strategies like caching of hashcode, caching of objects, object pooling, etc can be easily applied to improve performance. If String were made mutable, string pooling would not be possible as changing the string with one reference will lead to the wrong value for the other references.



String Pool

In Java 6 — all interned strings were stored in the **PermGen** – the fixed size part of heap mainly used for storing loaded classes and string pool.

In Java 7 – the string pool was relocated to the **heap**. So, you are not restricted by the limited size.

2. Thread safety as immutable classes are inherently thread safe as they cannot be modified once created. They can only be used as a read only objects. They can easily be shared among multiple threads for better scalability.

3. Errors & Security Vulnerabilities: In Java you pass sensitive information like file names, host names, login names, passwords, customer account numbers, etc as a string object. If String were not immutable, a password or account number can be accidentally & easily changed, which can cause errors and security vulnerabilities.

Shouldn't there be more FAQs on Java strings? : 10 FAQ Java String class interview questions and answers as it is one of the very frequently used data types.

Let's move on to the next set of FAQ Core Java interview questions and answers focusing on the java modifiers final, finally and finalize.

Q4. In Java, what purpose does the key words **final**, **finally**, and **finalize** fulfill?
A4. '**final**' makes a variable reference not changeable, makes a method not overridable, and makes a class not inheritable. Learn more about the drill down question on 6 Java modifiers that interviewers like to quiz you on…

'**finally**' is used in a try/catch statement to almost always execute the code. Even when an exception is thrown, the finally block is executed. This is used to close non-memory resources like file handles, sockets, database connections, etc till Java 7. This is is no longer true in Java 7.

Java 7 has introduced the *AutoCloseable* interface to avoid the unsightly try/catch/finally(within finally try/catch) blocks to close a resource. It also prevents potential resource leaks due to not properly closing a resource.

// pre Java 7

```
1   BufferedReader br = null;
2   try {
3     File f = new File("c://temp/simple.txt");
4     InputStream is = new FileInputStream(f);
5     InputStreamReader isr = new InputStreamReader
6     br = new BufferedReader(isr);
7
8     String read;
9
10    while ((read = br.readLine()) != null) {
11       System.out.println(read);
12    }
13  } catch (IOException ioe) {
14    ioe.printStackTrace();
15  } finally {
16    //Hmmm another try catch. unsightly
17    try {
18      if (br != null) {
19         br.close();
```

```
20        }
21      } catch (IOException ex) {
22        ex.printStackTrace();
23    }
24 }
25
```

Java 7 – try can have ***AutoCloseble*** types. *InputStream* and *OutputStream* classes now implements the Autocloseable interface.

```
1  try (InputStream is = new FileInputStream(new Fi
2   InputStreamReader isr = new InputStreamReader(i
3   BufferedReader br2 = new BufferedReader(isr);)
4     String read;
5
6     while ((read = br2.readLine()) != null) {
7        System.out.println(read);
8     }
9   }
10
11  catch (IOException ioe) {
12    ioe.printStackTrace();
13  }
14
```

try can now have multiple statements in the parenthesis and each statement should create an object which implements the new *java.lang.AutoClosable* interface. The ***AutoClosable*** interface consists of just one method. void close() throws *Exception* {}. Each *AutoClosable* resource created in the try statement will be automatically closed without requiring a finally block. If an exception is thrown in the try block and another Exception is thrown while closing the resource, the first Exception is the one eventually thrown to the caller. Think of the *close( )* method as implicitly being called as the last line in the try block.

'**finalize**' is called when an object is garbage collected. You rarely need to override it. It should not be used to release non-memory resources like file handles, sockets, database connections, etc because Java has only a finite number of these resources and you do not know when the garbage collection is going to kick in to release these non-memory resources through the *finalize( )* method.

So, final and finally are used very frequently in your Java code, but the key word finalize is hardly or never used.

Q5. What value will the following method return?

```
1 public static int getSomeNumber( ){
2  try{
3    return 2;
4  } finally {
5    return 1;
6  }
7 }
8
```

A5. 1 is returned because 'finally' has the right to override any exception/returned value by the try..catch block. It is a bad practice to return from a finally block as it can suppress any exceptions thrown from a try..catch block. For example, the following code will not throw an exception.

```
1 public static int getSomeNumber( ){
2   try{
3      throw new RuntimeException( );
4   } finally {
5      return 12;
6   }
7  }
8
```

Q6. What can prevent execution of a code in a finally block?

A6. **a)** An end-less loop.

```
1  public static void main(String[ ] args) {
2   try {
3     System.out.println("This line is printed ....
4     //endless loop
5     while(true){
6       //...
7     }
8   }
9   finally{
10    System.out.println("Finally block is reached.
11  }
12 }
13
```

b) *System.exit(1)* statement.

```
1  public class Temp {
2
3   public static void main(String[ ] args) {
4     try {
5       System.out.println("This line is printed ..
6       System.exit(1);
7     }
8     finally{
9       System.out.println("Finally block is reache
10    }
11  }
12 }
13
```

c) Thread death or turning off the power to CPU.
d) An exception arising in a finally block itself.
e) *Process p = Runtime.getRuntime( ).exec(“”)*;

If using Java 7 or later editions, use *AutoCloseable* statements within the try block.

It is important to understand **compile-time** vs **runtime** from core and enterprise Java interview questions & answers perspective. I have covered a more detailed discussion Compile-time Vs Run-time Interview Q&As

Q7. Can you describe “method overloading” versus “method overriding”? Does it happen at **compile time** or **runtime**?
A7. **Method overloading**: Overloading deals with multiple methods in the same class with the same name but different method signatures. Both the below methods have the same method names but different method signatures, which mean the methods are overloaded.

```
1 public class {
2   public static void evaluate(String param1); //
3   public static void evaluate(int param1); // met
4 }
5
```

This happens at compile-time. This is also called **compile-time polymorphism** because the compiler must decide how to select which method to run based on the data types of the arguments. If the compiler were to compile the statement:

```
1 evaluate(“My Test Argument passed to param1”);
2
```

it could see that the argument was a string literal, and generate byte code that called method #1.

Overloading lets you define the **same operation in different ways for different data**.

Method overriding: Overriding deals with two methods, one in the parent class and the other one in the child class and has the same name and signatures. Both the below methods have the same method names and the signatures but the method in the subclass MyClass overrides the method in the superclass BaseClass.

```
1 public class A {
2  public int compute(int input) { //method #3
3    return 3 * input;
4  }
5 }
6
```

```
1 public class B extends A {
2  @Override
3  public int compute(int input) { //method #4
4    return 4 * input;
5  }
6 }
7
```

This happens at runtime. This is also called **runtime polymorphism** because the compiler does not and cannot know which method to call. Instead, the JVM must make the determination while the code is running.

The method compute(..) in subclass “B” overrides the method compute(..) in super class “A”. If the compiler has to compile the following method,

```
1 public int evaluate(A reference, int arg2) {
2   int result = reference.compute(arg2);
3 }
4
```

The compiler would not know whether the input argument 'reference' is of type "A" or type "B". This must be determined during runtime whether to call method #3 or method #4 depending on what type of object (i.e. instance of Class A or instance of Class B) is assigned to input variable "reference".

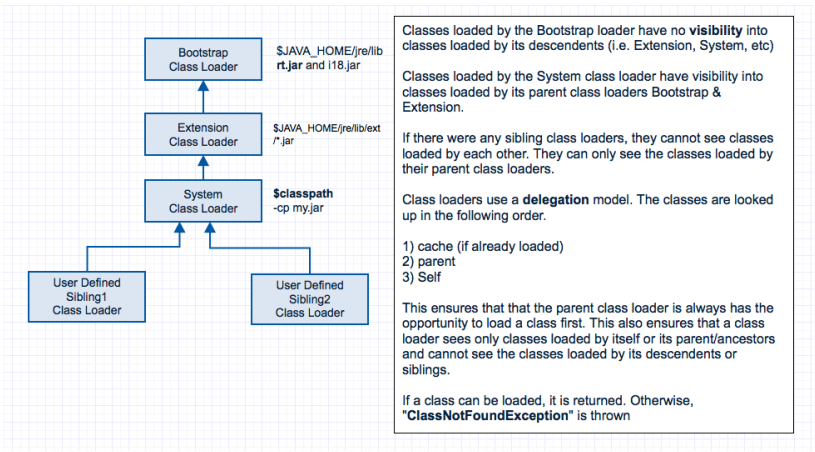
```
1 A obj1 = new B( );
2 A obj2 = new A( );
3 evaluate(obj1, 5); // 4 * 5 = 20. method #4 is in
4 evaluate(obj2, 5); // 3 * 5 = 15. method #3 is in
5
```

Overriding lets you define **the same operation in different ways for different object types**.

Core Java Interview Questions and answers will not be complete without class loaders and class loading issues.

Q8. What do you know about class loading? Explain Java class loaders? If you have a class in a package, what do you need to do to run it? Explain dynamic class loading?

A8. Class loaders are hierarchical. Classes are introduced into the JVM as they are referenced by name in a class that is already running in the JVM. So, how is the very first class loaded? The very first class is specially loaded with the help of static *main( )* method declared in your class. All the subsequently loaded classes are loaded by the classes, which are already loaded and running. A class loader creates a namespace. All JVMs include at least one class loader that is embedded within the JVM called the primordial (or bootstrap) class loader. The JVM has hooks in it to allow user defined class loaders to be used in place of primordial class loader. Let us look at the class loaders created by the JVM.



Java Class Loader Basics

Class loaders are hierarchical and use a delegation model when loading a class. Class loaders request their parent to load the class first before attempting to load it themselves. When a class loader loads a class, the child class loaders in the hierarchy will never reload the class again. Hence uniqueness is maintained. Classes loaded by a child class loader have visibility into classes loaded by its parents up the hierarchy but the reverse is not true as explained in the above diagram.

Q9. Explain static vs. dynamic class loading?

A9. Classes are **statically loaded** with Java's "new" operator.

```
1 class MyClass {
2  public static void main(String args[]) {
3    Car c = new Car( );
4  }
5 }
6
```

Dynamic loading is a technique for programmatically invoking the functions of a class loader at run time. Let us look at how to load classes dynamically.

```
1 //static method which returns a Class
2 Class.forName (String className);
```

```
3
```

The above static method returns the class object associated with the class name. The string className can be supplied dynamically at run time. Unlike the static loading, the dynamic loading will decide whether to load the class Car or the class Jeep at runtime based on a properties file and/or other runtime conditions. Once the class is dynamically loaded the following method returns an instance of the loaded class. It's just like creating a class object with no arguments.

```
1 // A non-static method, which creates an instance
2 // class (i.e. creates an object).
3 class.newInstance ( );
4
```

Static class loading throws "***NoClassDefFoundError***" if the class is not found and the dynamic class loading throws "***ClassNotFoundException***" if the class is not found.

Q10. What tips would you give to someone who is experiencing a class loading or "Class Not Found" exception?
A10. "***ClassNotFoundException***" could be quite tricky to troubleshoot. When you get a *ClassNotFoundException*, it means the JVM has traversed the entire classpath and not found the class you've attempted to reference.

1) Stand alone Java applications use -cp or -classpath to define all the folders and jar files to look for. In windows separated by ";" and in Unix separated by ":".

```
1 java -classpath "C:/myproject/classes;C:/myprojec
2
```

2) Determine the jar file that should contain the class file within the classpath — war/ear archives and application server lib directories. Search recursively for the class.

```
1 $ find . -name "*.jar" -print -exec jar -tf '{}'
2
```

3) Check the version of the jar in the manifest file MANIFEST.MF, access rights (e.g. read-only) of the jar file, presence of multiple versions of the same jar file and any jar corruption by trying to unjar it with "jar -xvf …". If the class is dynamically loaded with `Class.forName("com.myapp.Util")`, check if you have spelled the class name correctly.

4) Check if the application is running under the right JDK? Check the JAVA_HOME environment property

```
1 $ echo $JAVA_HOME
2
```

5) -verbose:class option in your JVM. With the -verbose option all the classes that are loaded are listed, along with the JAR file or directory from which they were loaded. The "class" output shows additional information, such as when superclasses are being loaded, and when static initializers are being run.

6) Creating a **Java dump and analyzing the Java dump** for class loading issues. The Java dumps are created under following circumstances.

— When a fatal native JVM error is thrown.
— When the JVM runs out of heap memory space.
— When a signal is sent to the JVM (e.g. Control-Break is pressed on Windows, Control-\ on Linux, or kill -3 on Unix)

There are tools like **jstack**, **jmap**, **hprof**, and Eclipse Memory Analyzer (MAT) to analyze the Java dumps.

7) Some of the libraries provide **API to list the version number**. For example, The Eclipse link MOXy library provides a method as shown below.

```
1
2    org.eclipse.persistence.Version.getVersion();
3
```

8) The org.jboss.test.util.Debug class has a method *displayClassInfo(Class clazz, StringBuffer results)* to display the loaded class details. This is done programmatically. What this class essentially does is

```
1
2       URL loc = MyClass.class.getProtectionDomain
3
```

9) The http://www.findjar.com is an online search engine that can list possible jar files in which a particular class file like java.sql.Connection can be found.

You can learn more at Java class loading interview Q&As to ascertain your depth of Java knowledge

Top 50+ FAQ Core Java Interview Questions & Answers:

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

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

3) Q37-Q42: Top 50+ Core on Java Garbage Collection Interview Questions & Answers

4) Q43-Q54: Top 50+ Core on Java Objects Interview Questions & Answers

5) Q01-Q28: Top 50+ EE Java interview questions & answers

6) Q29-Q53: Top 50+ JEE Interview Questions & Answers

Popular Posts

♦ 11 Spring boot interview questions & answers

825 views

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

766 views

18 Java scenarios based interview Questions and Answers

400 views

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

388 views

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

295 views

♦ 7 Java debugging interview questions & answers

293 views

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

285 views

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

279 views

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

239 views

001B: ♦ Java architecture & design concepts interview questions & answers

201 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.**945+** paid members. join my LinkedIn Group. **Reviews**

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.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ Q08-Q15 written test questions and answers on core Java

05: ♦ Finding the 2nd highest number in an array ›

Posted in FAQ Core Java Job Interview Q&A Essentials, Top 50+ FAQ Core Java Interview Q&A

Tags: Core Java FAQs, Free Content, Free FAQs, Novice FAQs, TopX

11 comments on "♥♦ Q1-Q10: Top 50+ Core Java Interview Questions & Answers"

Savita says:
September 1, 2016 at 3:30 am

Thank you sir it really good

Reply

Mala chanal says:
August 17, 2016 at 9:39 pm

Thank you so much sir

Reply

Ashwini Baban Ghanghav says:
May 23, 2016 at 5:33 pm

dear sir,
Thanks for posting very good information about the core java questions and answers

Reply

marco says:
February 6, 2016 at 8:06 am

Is this a typo or have I misunderstood?

Answer to Q3 in paragraph under performance.

“If String were made immutable, string pooling would not be possible as changing the string with one reference will lead to the wrong value for the other references.”

Should that be mutable?

Reply

Arulkumaran Kumaraswamipillai says:
February 6, 2016 at 1:06 pm

You were spot on. It was a mistake. I have rectified it.

Reply

Mark says:
February 6, 2016 at 2:13 am

For question 3 is this sentence correct?

“If String were made immutable, string pooling would not be possible as changing the string with one reference will lead to the wrong value for the other references.”

Should that not be mutable, as String is immutable in Java?

Reply

johan says:
January 12, 2016 at 5:16 pm

Nice write keep it up, IT Intern

Reply

akansha mandloi says:
December 17, 2015 at 7:21 pm

Hello,sir
Thank you so much
it is very helpful.

Reply

amrutha says:
October 20, 2015 at 3:50 am

Tnq sir ……its rlly useful 2 us.

.

Reply

Badrinath says:
September 24, 2015 at 8:25 pm

Hi Arulkumaran,
Thanks for posting very good information about the core java questions and answers.
Where can I find remaining 40+ questions? as I am not able to find the link.
Thanks,

Reply

Arulkumaran Kumaraswamipillai says:

October 11, 2015 at 9:07 pm

It is now added at the bottom. You can also find the links on the RHS navigation panel.

Reply

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

 ↑