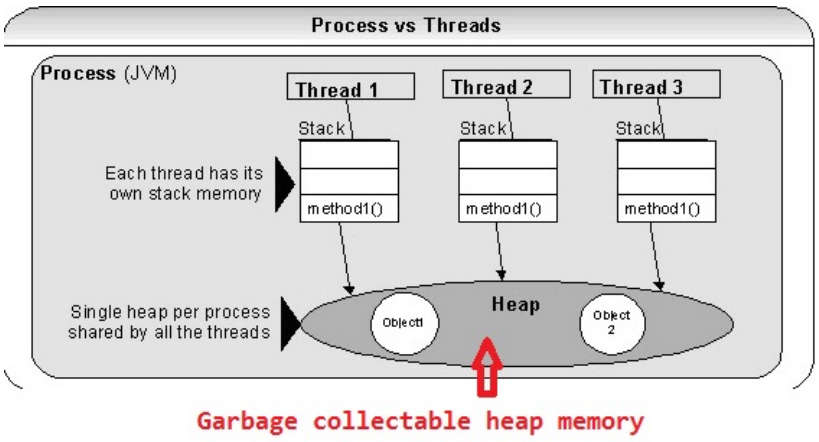
♦ Java Garbage Collection interview Q&A to ascertain your depth of Java knowledge

Posted on August 14, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

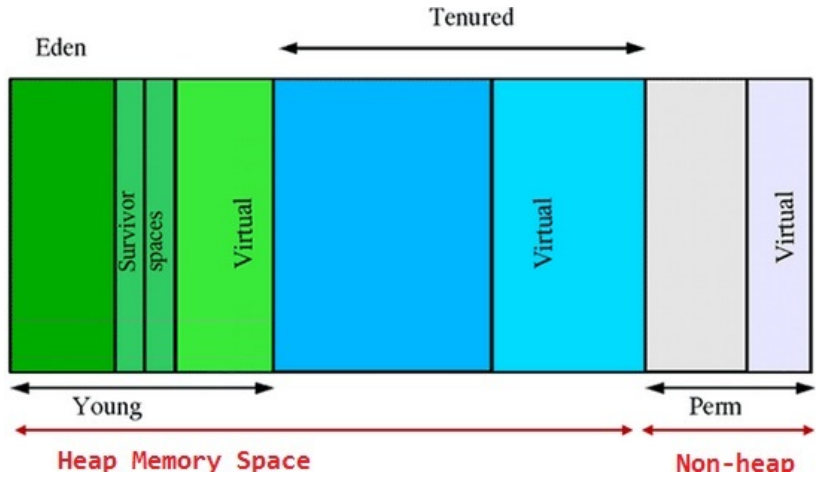
Q1. In which part of memory does Java collection occur? When does the garbage collection occur? In which thread does the GC run? Why is it knowing about GC is very important?

A1. Each time an object is created in Java, it goes into the area of memory known as heap. The Java heap is called the garbage collectable heap.



Garbage Collection in Heap

The garbage collection cannot be forced. The garbage collector runs in low memory situations. When it runs, it releases the memory allocated by an unreachable object. The garbage collector runs on a separate JVM created low priority daemon (i.e. background) thread. You can nicely ask the garbage collector to collect garbage by calling System.gc( ) but you can't force it. It runs when it decides it is best to run. This is what a heap memory space is divided into:



Java heap sections

Initially all new Java objects are stored in the Eden section. Most of these objects will then be destroyed at the next GC run because they are not used anymore. But some of them need to be kept because they have a longer lifetime and they will be used in the future. Therefore they are moved into the Survivor bucket. In the Survivor bucket GC calls are less

frequent than in the Eden bucket. These two buckets represent the Young generation and contains all the “newly created” objects. If objects stored in Survivor buckets, survive to other GC visits they are then moved into the Tenured generation (or Old generation) bucket until they are destroyed by the Garbage Collector.

It is vital to know about GC as one of the main reasons for performance issues in Java is due to JVM spending more time performing garbage collection. This can happen due to

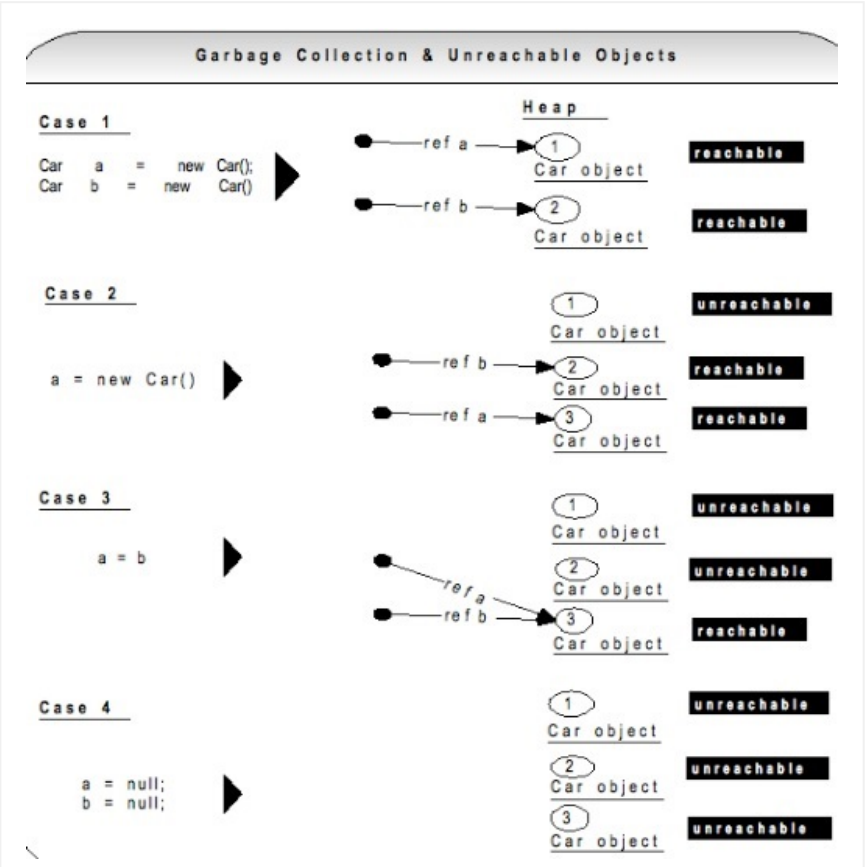
1) Improper Garbage Collection (GC) configuration. E.g. Young generation being too small.
2) Heap size is too small (use -Xmx). The application footprint is larger than the allocated heap size.
3) Wrong use of libraries taking up lots of the heap space. For example, XML based report generation using DOM parser as opposed to StAX for large reports generated concurrently by multiple users. DOM is very memory hungry.
4) Incorrectly creating and discarding objects without astutely reusing them with a flyweight design pattern or proper caching strategy.
5) Other OS activities like swap space or networking activities during GC can make GC pauses last longer.
6) Any explicit System.gc( ) from your application or third party modules.

You can log GC activities by running your JVM with GC options such as

-verbose:gc (print the GC logs)
-Xloggc: (comprehensive GC logging)
-XX:+PrintGCDetails (for more detailed output)
-XX:+PrintTenuringDistribution (tenuring thresholds)

Q2. What is an unreachable object?
A2. An object’s life has no meaning unless something has reference to it. If you can’t reach it then you can’t ask it to do anything. Then the object becomes unreachable and the garbage collector will figure it out. Java automatically collects all the unreachable objects periodically and releases the

memory consumed by those unreachable objects to be used by the future reachable objects.



Java unreachable vs reachable

Q3. What is the difference between a weak reference and a soft reference? Which one would you use for caching?

A3. Weak reference: A weak reference, simply put, is a reference that isn't strong enough to force an object to remain in memory. Weak references allow you to leverage the garbage collector's ability to determine reachability for you, so you don't have to do it yourself. You create a weak reference like this:

```
1
2 Car c1 = new Car( ); //referent is c1 is a strong
3 WeakReference<Car> wr = new WeakReference<Car>(c1
4
```

A weak reference is a holder for a reference to an object, called the referent. Weak references and weak collections are powerful tools for heap management, allowing the application

to use a more sophisticated notion of reachability, rather than the “all or nothing” reachability offered by ordinary (i.e. strong) references.

A WeakHashMap stores the keys using WeakReference objects, which means that as soon as the key is not referenced from somewhere else in your program, the entry may be removed and is available for garbage collection. One common use of WeakReferences and WeakHashMaps in particular is for adding properties to objects. If the objects you are adding properties to tend to get destroyed and created a lot, you can end up with a lot of old objects in your map taking up a lot of memory. If you use a WeakHashMap instead the objects will leave your map as soon as they are no longer used by the rest of your program, which is the desired behavior.

Soft reference is similar to a weak reference, except that it is less eager to throw away the object to which it refers. An object which is only weakly reachable will be discarded at the next garbage collection cycle, but an object which is softly reachable will generally stick around for a while as long as there is enough memory. Hence the soft references are good candidates for a cache.

```
1
2 byte[ ] cache = new byte[1024];
3 //... populate the cache. The referent is  cache
4 SoftReference<byte> sr = new    SoftReference<byt
5
```

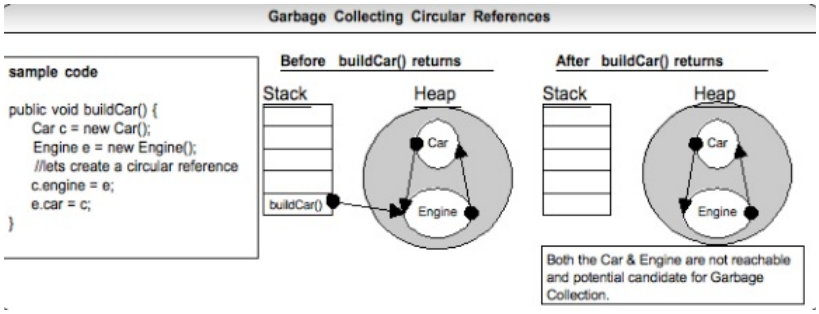
The garbage collector may or may not reclaim a softly reachable object depending on how recently the object was created or accessed, but is required to clear all soft references before throwing an OutOfMemoryError.

Note: The weak references are eagerly garbage collected, and the soft references are lazily garbage collected under low memory situations.

Q4. If you have a circular reference of objects, but you no longer reference it from an execution thread, will this object

be a potential candidate for garbage collection?

A4. Yes. Refer diagram below.



Java GC cyclic refrence

Q5. When you have automatic memory management in Java via GC, why do you still get memory leaks in Java?

A5. In Java, memory leak can occur due to

1) Long living objects having reference to short living objects, causing the memory to slowly grow. For example, singleton classes referring to short lived objects. This prevents short-lived objects being garbage collected.

2) Improper use of thread-local variables. The thread-local variables will not be removed by the garbage collector as long as the thread itself is alive. So, when threads are pooled and kept alive forever, the object might never be removed by the garbage collector.

3) Using mutable static fields to hold data caches, and not explicitly clearing them. The mutable static fields and collections need to be explicitly cleared.

4) Objects with circular or bidirectional references referenced from a thread.

5) JNI (Java Native Interface) memory leaks.

Q6. How will you go about creating a memory leak in Java?

A6. In Java, memory leaks are possible under a number of scenarios. Here is a typical example where hashCode( ) and equals( ) methods are not implemented for a custom Key class that is used to store key/value pairs in a HashMap. This will end up creating a large number of duplicate objects if you run in a large or endless while(true) loop for demonstration purpose. All memory leaks in Java end up with

java.lang.OutOfMemoryError, and it is a matter of time before you get this error.

Q7. How will you fix the above memory leak?
A7. By providing proper implentation for the key class as shown below with the equals() and hashCode() methods.

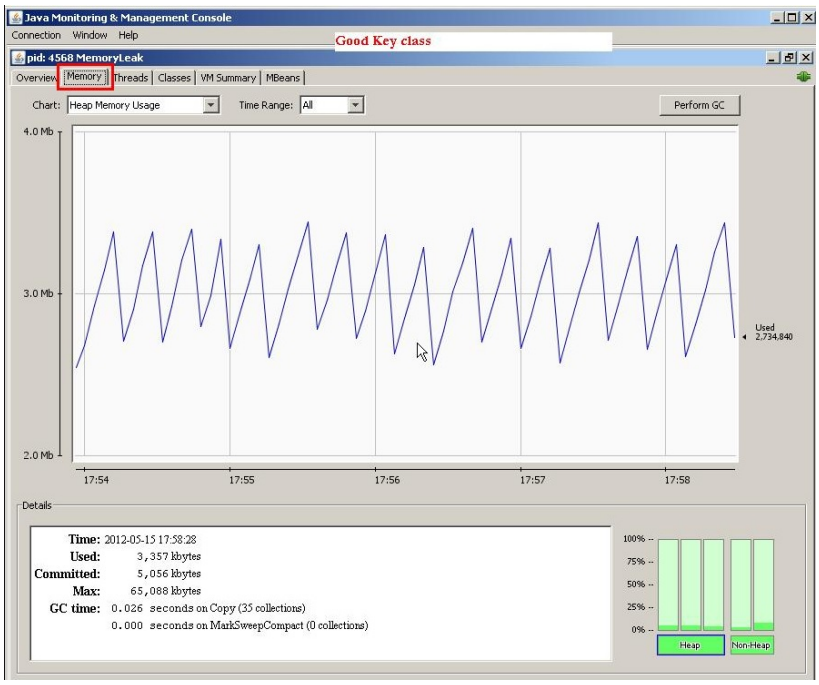
Q8. In real applications, how do you know that you have a memory leak?
A8. If you profile your application, you can notice a graph like a saw tooth. Here is how you can determine this with the help of jconsole for the above bad key class example. All you have to to do is while your memory leaking application is running, get the Java process id by typing

```
1
2 C:\>jps
3 5808 Jps
4 4568 MyApp
5 3860 Main
6
```

Now, open up the jconsole as shown below on a command line

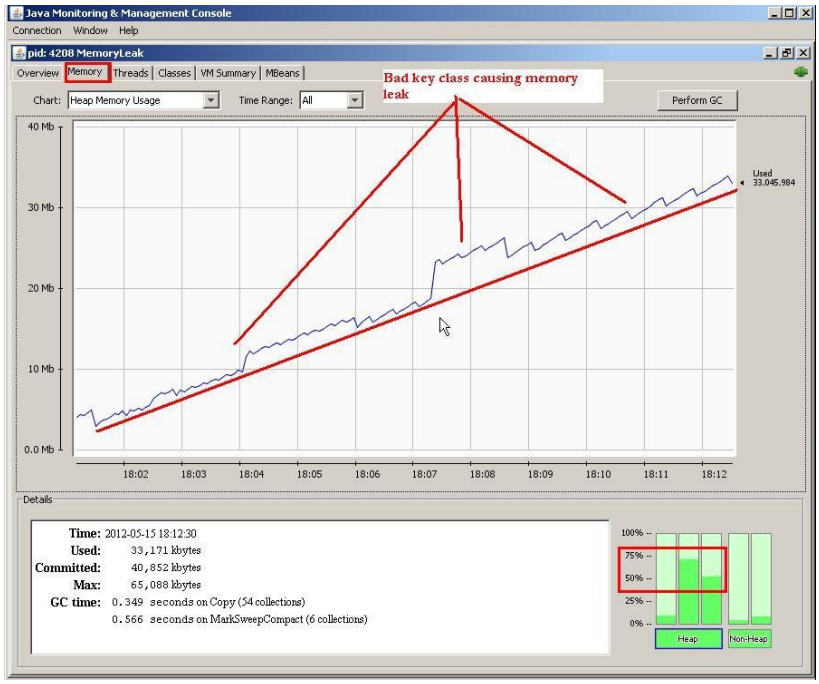
```
1
2 C:\>jconsole 4568
3
```

No memory Leak:



jconsole memory graph – no leak

With Memory Leak (saw tooth graph):



jconsole memory graph – with memory leak

Visual VM for moinitoring Java heap memory

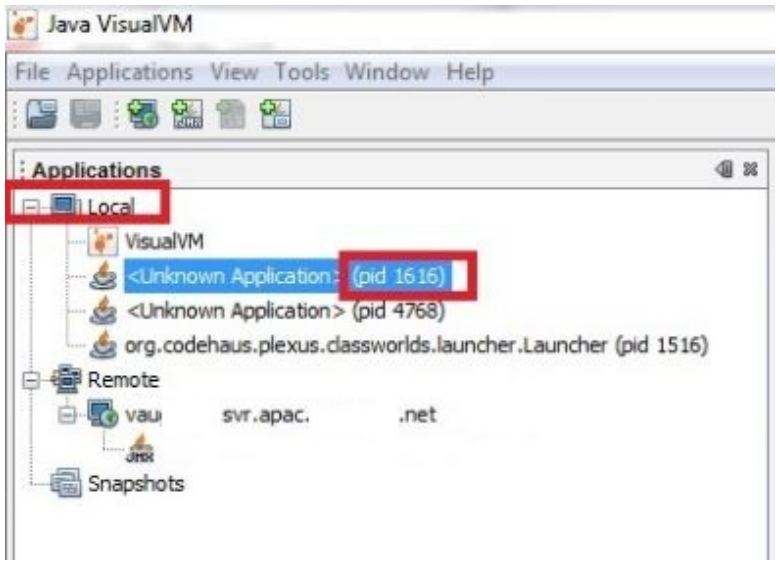
VisualVM is a visual tool integrating several commandline JDK tools and lightweight profiling capabilities. Designed for both production and development time use, it further enhances the capability of monitoring and performance analysis for the Java SE platform. It is packaged as an exe file.

Step 1: You can start the visual vm by double clicking on %JAVA_HOME%/bin/jvisualvm.exe from Java 1.6 version onwards.

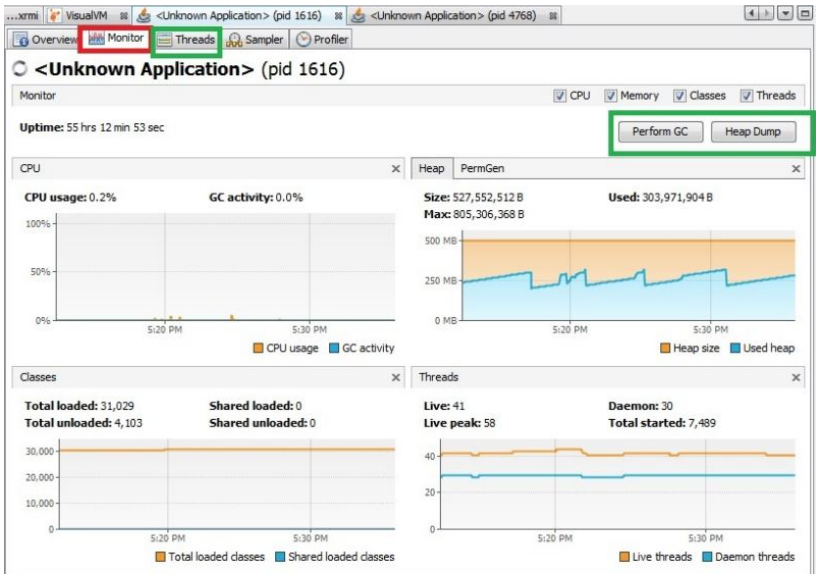
Step 2: Your local processes will be monitored under the local tab. The Visual vm can also used to open the heap dump files i.e *.hprof files to analyze the menory usages. You can find out the process ids of your local Java applications via

1. netstat -anp | grep 8088 or
2. In windows via the “Windows Task Manager” –> Processes tab. You need to click on View –> Select Columns and then “tick” PID (Process Identifier) check box.

Double click on the relevant PID in Visual VM console.



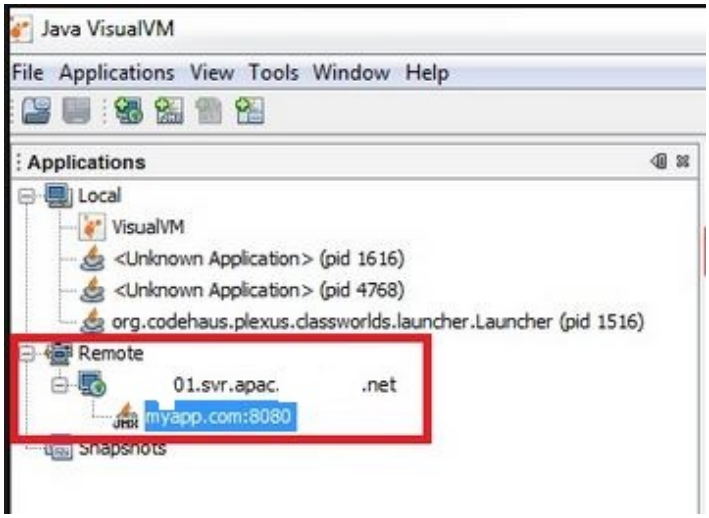
Visual VM



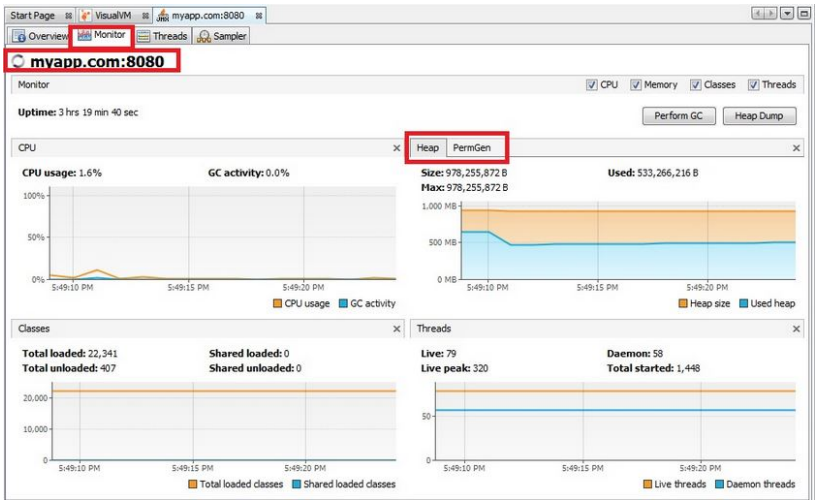
Visual VM

You can add remote processes by following the steps shown below.

1. Right click on “Remote” and then select “Add Remote Host…”.
2. Provide the host name like “myapp.com”.
3. It searches and adds the host.
4. Right click on the added host name and then select “Add JMX Connection” and in the “Connection” field type the hostname:JMX port number like myapp.com:8083.
5. Double click on this JMX connection to monitor CPU, memory, thread, etc.



Visual VM



Visual VM

Popular Posts

♦ 11 Spring boot interview questions & answers

861 views

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

829 views

18 Java scenarios based interview Questions and Answers

448 views

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

407 views

♦ 7 Java debugging interview questions & answers

311 views

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

303 views

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

294 views

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

288 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.

‹ ♦ 12 Java Generics interview Q&A

♦ 8 Java EE (aka JEE) Overview interview questions & answers ›

Posted in GC, JVM, member-paid, Performance

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

The contents in this Java-Success are copy righted. The author has the right to correct or enhance the current content without any prior notice.

These are general advice only, and one needs to take his/her own circumstances into consideration. The author will not be held liable for any damages caused or alleged to be caused either directly or indirectly by these materials and resources. Any trademarked names or labels used in this blog remain the property of their respective trademark owners. No guarantees are made regarding the accuracy or usefulness of content, though I do make an effort to be accurate. Links to external sites do not imply endorsement of the linked-to sites.

↑