

Industrial strength Java/JEE Career Companion to open more doors

[Home](#)
[Java FAQs](#)
[600+ Java Q&As](#)
[Career](#)
[Tutorials](#)
[Member](#)
[Why?](#)
[Can u Debug?](#)
[Java 8 ready?](#)
[Top X](#)
[Productivity Tools](#)
[Judging Experience?](#)

[Home](#) › [Interview](#) › [Pressed for time? Java/JEE Interview FAQs](#) › [Java Key Area Essentials](#) › [4 FAQ Performance tuning in Java interview Q&As](#)

4 FAQ Performance tuning in Java interview Q&As

Posted on [September 8, 2014](#) by [Arulkumaran Kumaraswamipillai](#) — [No Comments](#) ↓

0
Like
Share

Tweet

0
G+1
Share

Q1. In your experience, what are some of the major causes of performance bottlenecks in Java applications?

A1.

Cause #1: The JVM spends **more time performing garbage collection** due to

- improper Garbage Collection (GC) configuration. E.g. Young generation being too small.
- Heap size is too small (use -Xmx). The application footprint is larger than the allocated heap size.
- Wrong use of libraries. For example, XML based report generation using DOM parser as opposed to

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

[open all](#) | [close all](#)

✚ [Ice Breaker Interview](#)

✚ [Core Java Interview C](#)

✚ [JEE Interview Q&A \(3](#)

✚ [Pressed for time? Jav](#)

✚ [Job Interview Ice B](#)

✚ [FAQ Core Java Jot](#)

✚ [FAQ JEE Job Inter](#)

✚ [FAQ Java Web Ser](#)

✚ [Java Application Ar](#)

✚ [Hibernate Job Inter](#)

✚ [Spring Job Intervie](#)

✚ [Java Key Area Ess](#)

♦ [Design pattern](#)

♥ [Top 10 causes](#)

♥♦ [01: 30+ Writir](#)

♦ [12 Java designr](#)

♦ [18 Agile Develo](#)

♦ [5 Ways to debi](#)

♦ [9 Java Transac](#)

♦ [Monitoring/Pro](#)

02: ♥♦ [13 Tips to](#)

StAX for large reports generated concurrently by multiple users.

- Incorrectly creating and discarding objects without astutely reusing them with a flyweight design pattern or proper caching strategy.
- Other OS activities like swap space or networking activity during GC can make GC pauses last longer.
- Any explicit `System.gc()` from your application or third party modules.

Run 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)

to understand the GC patterns.

Cause #2: Bad use of application algorithms, strategies, and queries. For example

- SQL queries with Cartesian joins.
- SQL queries invoking materialized views
- Regular expressions with back tracking algorithms.
- Inefficient Java coding and algorithms in frequently executed methods leading to death by thousand cuts.
- Excessive data caching or inappropriate cache refresh strategy.
- Overuse of pessimistic locking as opposed to favoring optimistic locking.

Cause #3: Memory leaks due to

- 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.
- 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

15	Security key :
4	FAQ Performa
4	JEE Design Pa
5	Java Concurr
6	Scaling your J
8	Java memory i
	OOP & FP Essenti
	Code Quality Job I
	SQL, XML, UML, JSC
	Hadoop & BigData Int
	Java Architecture Inte
	Scala Interview Q&As
	Spring, Hibernate, & I
	Spring (18)
	Spring boot (4)
	Spring IO (1)
	Spring JavaConl
01:	♥♦ 13 Spring
01b:	♦ 13 Spring
02:	► Spring DI
03:	♥♦ Spring DI
04	♦ 17 Spring b
05:	♦ 9 Spring B
06:	♥ Debugging
07:	Debugging S
	Spring loading p
	Hibernate (13)
01:	♥♦ 15+ Hiber
01b:	♦ 15+ Hiber
02:	Understandir
03:	Identifying ar
04:	Identifying ar
05:	Debugging H
06:	Hibernate Fil
07:	Hibernate mi
08:	Hibernate au
09:	Hibernate en
10:	Spring, Java
11:	Hibernate de
12:	Hibernate cu
	AngularJS (2)

threads are pooled and kept alive forever, the object might never be removed by the garbage collector.

- Using mutable static fields to hold data caches, and not explicitly clearing them. The mutable static fields and collections need to be explicitly cleared.
- Objects with circular or bidirectional references.
- JNI memory leaks.

Cause #4: Poor integration with external systems without proper design & testing.

- Not properly deciding between synchronous vs asynchronous calls to internal and external systems. Long running tasks need to be performed asynchronously.
- Not properly setting service timeouts and retries or setting service time out values to be too high.
- Not performing non happy path testing and not tuning external systems to perform efficiently.
- Unnecessarily making too many network round trips.

Cause #5: Improper use of Java frameworks and libraries.

- Using Hibernate without properly understanding lazy loading versus eager fetching and other tuning capabilities.
- Not inspecting the SQLs internally generated by your ORM tools like Hibernate.
- Using the deprecated libraries like *Vector* or *Hashtable* as opposed to the new concurrent libraries that allow concurrent reads.
- Using blocking I/O where the the Java NIO (New I/O) with non blocking capability is favored.
- Database deadlocks due bad schema design or application logic.
- Spinning out your own in efficient libraries as opposed to favoring proven frameworks.

Cause #6: Multi-threading issues due to to deadlocks, thread starvation, and thread contention.

- Using coarse grained locks over fine grained locks.
- Not favoring concurrent utility classes like *ConcurrentHashMap*, *CopyOnWriteArrayList*, etc.

 [Git & SVN \(6\)](#)

 [JMeter \(2\)](#)

 [JSF \(2\)](#)

 [Maven \(3\)](#)

 [Testing & Profiling/Sa](#)

 [Other Interview Q&A 1](#)

 [Free Java Interview](#)

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 I](#)

 [Low Latency \(7\)](#)

 [Memory Management](#)

 [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 \(2\)](#)

 [Hadoop & Spark Tuto](#)

- Not favoring lock free algorithms.

Cause #7: Not managing and recycling your non memory resources properly.

- Not pooling your valuable non memory resources like sockets, connections, file handles, threads, etc.
- Not properly releasing your resources back to its pool after use can lead to resource leaks and performance issues.
- Use of too many threads leading to more CPU time being used for context switching.
- Hand rolling your own pooling without favoring proven libraries.
- Using a third-party library with resource leaks.
- Load balancers leaking sockets.

Cause #8: Bad infrastructure designs and bugs.

- Databases tables not properly partitioned.
- Not enough physical memory on the box.
- Not enough hard disk space.
- Bad network latency.
- Too many nodes on the server.
- Load balancers not working as intended and not performing outage testing.
- Not tuning application servers properly.
- Not performing proper capacity planning.
- router, switch, and DNS server failures.

Cause #9: Excessive logging and not using proper logging libraries with capabilities to control log levels like debug, info, warning, etc. *System.out.println(...)* are NOT ALLOWED. Favor asynchronous logging in mission critical and high volume applications like trading systems.

Cause #10: Not conducting performance tests, not monitoring the systems, and lack of documentation and performance focus from the start of the project.

- Not performing performance test plans with tools like JMeter with production like data prior to each deployment.

- [JEE Tutorials \(19\)](#)
- [Scala Tutorials \(1\)](#)
- [Spring & Hibernate T](#)
- [Tools Tutorials \(19\)](#)
- [Other Tutorials \(45\)](#)

100+ Java pre-interview coding tests

[open all](#) | [close all](#)

- [Can you write code? \(](#)
- [♦ Complete the given](#)
- [Converting from A to I](#)
- [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?

[open all](#) | [close all](#)

- [Career Making Know-](#)
- [Job Hunting & Resurr](#)

- Not monitoring the systems for CPU usage, memory usage, thread usage, garbage collection patterns, I/O, etc on an on going basis.
- Not defining SLA's (Service Level Agreements) from the start in the non-functional specification.
- Not maintaining proper performance benchmarks to be compared against with the successive releases and performance tests.
- Not looking for performance issues in peer code reviews or not having code reviews at all.

Q2. How would you go about performance testing your Java application?

A2.

1. Using a profiler on your running code. It should help you identify the bottlenecks. For example, jprofiler or Netbeans profiler. A profiler is a program that examines your application as it runs. It provides you with useful run time information such as time spent in particular code blocks, memory / heap, etc. In Java 6, you could use the **JConsole with Visual VM**.

2. You also need to set up performance test scripts with JMeter to simulate the load. Most issues relating to performance bottlenecks, memory leaks, thread-safety, deadlocks, etc only surface under certain load. The performance testing scripts can be recorded while the application is being used and then manually refined. The tools like JMeter HTTP Proxy or Badboy software can be used to trace the script.

3. You could provide a custom solution with the help of AOP (aspect oriented programming) or dynamic proxies to intercept your method calls. Dynamic proxies allow you to intercept method calls so you can interpose additional behavior between a class caller and its "callee".

4. Yammer metrics can be included in your code to gather various statistics like request response times, request counts,

etc and produce reports in different formats like JSON, log4j out put, etc.

5. If you have a highly distributed system with lots of asynchronous processing, you use log4j with AOP to write metrics to a JMS queue, and have a separate process listen to the JMS queue and write aggregated and consolidated stats written to database tables for further analysis. Splunk is another alternative.

Q3. What tips do you give someone regarding Java performance?

A3. #1. Don't compromise on design: You should not compromise on architectural principles for just performance. You should make effort to write architecturally sound programs as opposed to writing only fast programs. If your architecture is sound enough then it would allow your program not only to scale better but also allows it to be optimized for performance if it is not fast enough. If you write applications with poor architecture but performs well for the current requirements, what will happen if the requirements grow and your architecture is not flexible enough to extend and creates a maintenance nightmare where fixing a code in one area would break your code in another area. This will cause your application to be re-written. So you should think about extendibility (i.e. ability to evolve with additional requirements), maintainability, ease of use, performance and scalability (i.e. ability to run in multiple servers or machines) during the design phase. List all possible design alternatives and pick the one which is conducive to sound design architecturally (i.e. scalable, easy to use, maintain and extend) and will allow it to be optimized later if not fast enough. Once you get the correct design, then measure with a profiler and optimize it.

#2. Be aware of the death by thousand-cuts: Having said not to compromise on the design, one needs to be mindful of performance inefficiencies that can creep in throughout the software development. For example, an inefficient method being called 50-100 times can adversely impact performance. A real-life example would be a JSF application that invokes its

life-cycle methods many times. So, having a long-running subroutine within the life-cycle method might end up calling it more than once. So, know your best practices and potential pitfalls. Here are a few things to keep in mind.

— Use immutable objects where applicable. Immutable objects are inherently thread-safe and can also be reused. A good candidate for implementing the flyweight design pattern. The following Java method is an example of flyweight. E.g. `Integer.valueOf(String s)`.

— Check your regexes and SQL queries for backtracking and Cartesian joins respectively.

— Use proven libraries, frameworks, built-in algorithms, and data structures as opposed to creating your own. For example, when handling concurrency use `java.util.concurrent` package.

#3. Always have a performance focus by developing proper load testing scripts and benchmarks. The non-functional requirements should cover the relevant SLAs (i.e. Service Level Agreements). Tune your application server, database server, application, etc where required with proper bench marking and load testing scripts. The mission critical applications must have run time metrics gathering in place commercial tools. There are tools that can be used in production environment like YourKit for Java, JProfiler for Java, etc and for larger distributed and clustered systems with large number of nodes there are profilers like CA Wiley Introscope for Java, ClearStone for Java, and HP Performance managers. These tools are handy for proactive detection and isolation of server/application bottlenecks, historical performance trend tracking for capacity planning, and real-time monitoring of system performance.

Q4. When designing your new code, what level of importance would you give to the following attributes? Rank the above attributes in order of importance?

- Performance
- Maintainability
- Extendibility
- Ease of use
- Scalability

Rank the above attributes in order of importance?

A4. This is an open-ended question. There is no one correct order for this question. The order can vary from application to application, but typically if you write **1 – extendable, 2 – maintainable and 3 – ease of use** code with some high level performance considerations, then it should allow you to optimize/tune for **4 – performance and 5 – scalability**. But if you write some code, which only perform fast but not flexible enough to grow with the additional requirements, then you may end up re-writing or carrying out a major revamp to your code.

Popular Posts

♦ [11 Spring boot interview questions & answers](#)

827 views

♦ [Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers](#)

768 views

[18 Java scenarios based interview Questions and Answers](#)

400 views

001A: ♦ [7+ Java integration styles & patterns interview questions & answers](#)

389 views

01b: ♦ [13 Spring basics Q8 – Q13 interview questions & answers](#)

296 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](#)

286 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

240 views

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

202 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](https://www.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](https://www.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](#)

15 Security key area interview Q&A for Java developers >

Posted in Java Key Area Essentials, member-paid, Performance

Tags: Architect 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.