04: Can you think of a time where you ...? open-ended Java interview Q&A

Posted on September 19, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

0 Like Share Tweet 0 G+1 Share

It really pays to jog your memory prior to job interviews.

Q1. Can you think of a time where you accomplished **QuickWins** for your company?
A1. The focus is to improve the overall effectiveness and usefulness of a system through small changes in a collaborative effort with the business.

Example: Spearheaded the “Quick Wins” project by working very closely with the business and end users to improve the

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
  - 01: ♦ 15 Ice breake
  - 02: ♥♦ 8 real life ex
  - 03: ♦10+ Know you
  - 04: Can you think c
  - 05: ♥ What job inte
  - 06: ► Tell me abou
  - 07: ♥ 20+ Pre inter
- Core Java Interview C
  - Java Overview (4)
    - 01: ♦ ♥ 17 Java c

current website's ranking from being 23rd to 6th in just 3 months.

Q2. Can you think of times where you were **proud of your accomplishments**?
A2.

- Reduced the over-night batch job runtime from 3 hours to 40 minutes by optimizing the SQL that was performing very badly in the query plan, and replacing the pessimistic locking with optimistic locking.
- On taking up the position as a Java technical lead, I set up a team of 8 developers to successfully complete the project on time and within budget.
- Instrumental in designing and developing a Java/JEE based online insurance system, which serves 400 concurrent users using Spring, Hibernate, JSF, ajax, and jQuery.
- Designed and developed the payment and claims module, which is capable of handling 120 requests per second and runs as a true 24 x 7 system module.
- Championed the iterative and test driven development (TDD) with regular code reviews, bamboo based continuous integration, and sonar based code coverage, which resulted in not only more maintainable and extensible code, but also roughly 30% drop in bugs.
- Increased the number of JUnit tests from 30+ to 200+ in my watchful eye to improve the overall quality of the application.
- Conceptualized and implemented a large scale multi-threaded socket server to communicate with 450 retail stores and seamlessly integrated it with 3 other back end systems via Web services and asynchronous messaging.

Q3. Can you think of a time where you solved hard to debug **“intermittent issues”**?

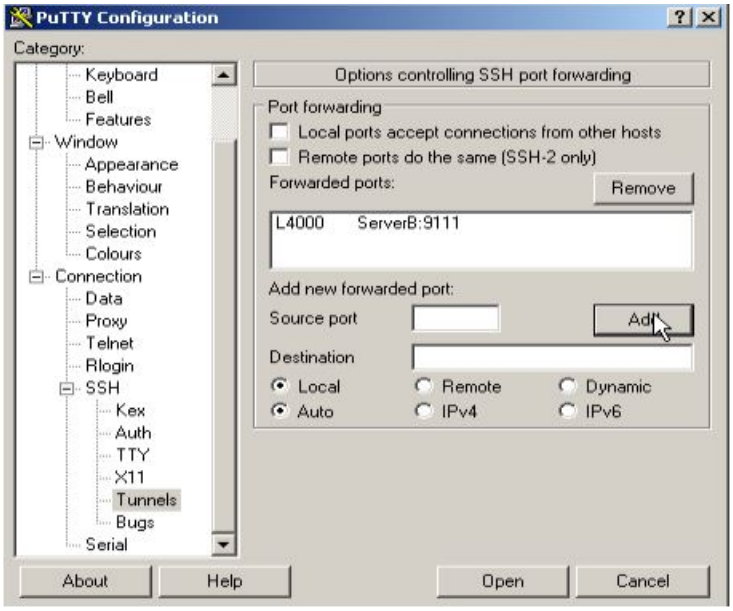
A3. It is always a challenge to isolate, reproduce, and fix intermittent issues. You may have come across intermittent issues that are hard to reproduce and debug. A novice developer will promptly mark those defects in the bug tracking system as “cannot be reproduced” without having the skill to isolate, reproduce and then fix the issue. An experienced developer with good grasp on the key areas will not only have the skill to identify these potential issues by analyzing the code, but also will have the competency to isolate, reproduce, and fix these issues:

- Not adhering to language basics and contracts can lead to non-deterministic behaviors. For example, in Java, incorrect implementation of hashCode(), equals(..), and compareTo(..) method contracts can lead to unpredictable
- Servlets, Struts action classes, SimpleDateFormat, etc are not inherently thread-safe and can be accessed by multiple threads. Hence not using them in a thread-safe manner can cause intermittent issues.
- Some operations not only need to be thread-safe, but also must be atomic. Issues arising from incorrect transaction management can cause intermittent issues by corrupting some records in the database tables. The database operations need to be atomic. If the transactions are carried across two distributed systems, then the 2-phase commit transaction management needs to be used. If proper transaction isolation levels are not used, the flight seats can be double booked due to dirty reads, phantom updates, or phantom inserts. Be aware of the ACID (Atomic, Consistent, Isolation, and Durability) properties to ensure that the database transactions are processed reliably. The web services based transactions need to be using compensation based transaction management as

opposed to the ACID based transaction management.

- You could also have intermittent issues due to proxy server or load balancer timeouts or connection leaks. Your application might not be coded with appropriate connection retries or appropriate time out values. You could simulate connectivity issues by creating SSH tunnels to the actual destination server.

  For example, your application may have connectivity to a Vignette ServerB running on port 9111. To simulate connectivity issues, you may create a local SSH tunnel to a UNIX ServerA, which connects to Vignette application running on ServerB:9111. Once you log on to ServerA via PutTTY (i.e. a Unix client on Windows) as shown below, a tunnel will be opened to ServerB:9111. Any calls to localhost:4000 will be forwarded to ServerB:9111 via ServerA. In your application, you could use the localhost:4000 instead of using ServerB:9111 to connect to the Vignette application. This will enable you to simulate the scenario of the Vignette server on ServerB being down by just destroying the tunnel, and not the actual server, which might be used by others. By destroying the tunnel, you could improve your application code to graciously handle connectivity issues, timeouts, and retries. The exception handling logic also needs to be verified so that that it does not expose any internal server details.



- There could be general runtime production issues that either slow down or make a system to hang. In these situations, the general approach for troubleshooting would be to analyze the thread dumps to isolate the threads which are causing the system to slow-down or hang. For example, a Java thread dump gives you a snapshot of all threads running inside a Java Virtual Machine. There are graphical tools like Samurai to help you analyze the thread dumps more effectively.

  Application seems to consume 100% CPU and throughput has drastically reduced – Get a series of thread dumps, say 7 to 10 at a particular interval, say 5 to 8 seconds and analyze these thread dumps by inspecting closely the “runnable” threads to ensure that if a particular thread is progressing well. If a particular thread is executing the same method through all the thread dumps, then that particular method could be the root cause. You can now continue your investigation by inspecting the code.

  Application consumes very less CPU and response times are very poor due to heavy I/O operations like file or database read/write operations – Get a series of thread dumps and inspect for threads that are in “blocked” status. This analysis

can also be used for situations where the application server hangs due to running out of all runnable threads due to a deadlock or a thread is holding a lock on an object and never returns it while other threads are waiting for the same lock. The solution to the above problems could vastly vary from fixing the thread safety issue(s) to reducing the size of synchronization granularity, and from implementing appropriate caching strategies to setting the appropriate connection timeouts.

- Intermittent issues could also arise due to environmental complexities. Your application might be running under an uncontrolled environment. Lack of communication among multiple projects using the same environment could cause intermittent issues. For example, security certificates or passwords could have been modified by the system administrators. Database table or LDAP server schemas could have been modified by the other developers. Messages published to a queue may have been consumed by another process listening on the same queue.
- Speaking of intermittent issues, database deadlock issues are very common. Whenever you have competing DML (Data Manipulation Language) running against the same data, you run the risk of a deadlock. When this happens, the database server identifies the problem and ends the deadlock by automatically choosing one process and aborting the other process, allowing the unaborted process to continue. The aborted transaction is rolled back and an error message is sent to the user of the aborted process. For example, in Sybase database

```
1 Your server command (family id #%d, proce
```

Generally, the transaction that requires the least amount of overhead to rollback is the transaction that is aborted. You can deal with database

deadlock situations in two ways. The first option is to redesign the application so that the deadlock does not take place in the first place. This is the preferred option, but at times not practical to redesign an existing application without significant effort. The second option is to retry the aborted task after receiving a deadlock message, and it will most likely succeed during 2-5 additional retries. If the retry happens very frequently, the performance of the application can be adversely impacted. Here are some general tips on how to avoid deadlocking

- Aim for properly normalized database designs.
- Avoid or minimize the use of cursors.
- Keep transactions as short as possible, use lower isolation levels and minimize the number of round trips between your application and the database server.
- Reduce lock time. Try to develop your application so that it grabs locks at the latest possible time, and then releases them at the very earliest time.
- Make relevant design or coding changes like single-threading related updates, and re-scheduling batch update jobs to low-update time period can often remove deadlocks.

Q4. Can you think of a time where you had to identify **"performance issues and bench mark them"** with the view to improve?

A4. I have never been in a project or organization that is yet to have any performance or scalability issues. It is a safe bet to talk up your achievements in fixing performance or resource leak issues in job interviews. Unlike web security testing, performance testing is very prevalent in many applications. Premature optimization of your code is bad as it can compromise on good design or writing maintainable and testable code. But one needs to be aware of potential causes for performance issues that can occur due to major

bottlenecks in a handful of places or minor inefficiencies in thousands of places (i.e.death by thousand cuts). Let's look at some of the common causes of performance issues.

- Too many database calls and inefficient SQL queries performing full table scan can cause performance issues. SQL statements need to be carefully constructed and prepared statements need to be favored over ordinary statements, as it improves performance by pre-compiling the execution plan for repeated calls. In some scenarios, more data is requested than actually is required by the current page. Eagerly fetching data from an ORM tool can bring in more data than actually required. In other scenarios, too many fine grained calls are made to the database instead of eagerly fetching the required data in 1 or fewer calls. At times, not using proper pagination strategies to divide and conquer how data is accessed or not carefully thinking through ways to improve re-usability of the same data requested multiple times through caching strategies can lead to bad performance. Excessive or wrong caching strategies to minimize remote calls could adversely impact performance due to increased garbage collection activity to make the memory available.
- A frequently back tracking regular expression can adversely impact performance. Care should be taken to not run a web service or a web application that allows users to supply their own regular expressions. People with little regular expression experience can come up with an exponentially complex regular expressions.
- Frequent garbage collections can adversely impact performance. Tune GC and allocate adequate heap space.
- Memory leaks and connection leaks (e.g. database connections, LDAP connections, sockets) can cause performance and scalability issues. Wasteful handling of these finite resources can adversely

impact performance. These resources must be judiciously used for a request and promptly returned to the pool to be reused by other users or requests. It should not be kept open for the whole session for a particular user because a user session can span multiple clicks interleaved with long pauses and user think times.

- Third party rich web frameworks can make your front end HTML source code to bloat. For example, overuse of some libraries or rich widgets can not only make the HTML code bloated, but also its aggressive use of ajax/JavaScript can result in performance problems on the browser. This might not be a problem for an internal application, but can be a problem for all external customer facing applications as some users may not have super fast internet access. Some older versions of some browsers may also take longer time to render the bloated HTML code.
- Not setting approprite service time outs.
- Not performance testing your application, or performance testing with non-production like scenario — cut down data sets, no or fewer concurrent access, etc

Once you identify a performance issue, proper test scripts need to be written and run to benchmark a particular application. Open source tools like BadBoy or JMeter proxy server can be used to record the web actions, and the recorded scripts can be converted to JMeter scripts to perform stress/load tests to benchmark the application. The initial test conditions and results need to be properly documented so that the results can be compared after improving or fixing the performance issues. The load/stress tests can also be used to reproduce and fix intermittent issues like thread-safety, memory leak, connection leak, and database contention issues that mostly occur above certain load.

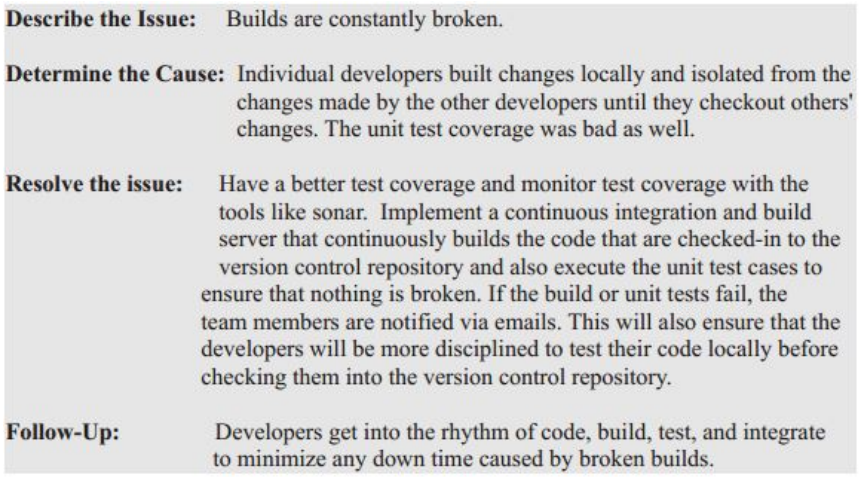
Q5. Can you think of a time where you **championed a continuos improvement** program?

A5. Organizations must develop a culture of continuous improvement in order to flourish. I am yet to work for an organization that did not have any environmental or process related issues and challenges. Many of these tasks can be simplified, automated, and integrated to avoid monotonous and error prone human intervention. So, experience with the agile practices, continuous integration and build, environmental improvements, etc will be of great asset to any organization. A software must be architected for deployability. Deployability is a non-functional requirement that addresses how reliably and easily a
software can be deployed from development into the production environment. Firstly, for a software to be deployable, there must be minimal differences between environments. The more similar the environments, the simpler and more reliable it is to deploy the software. It is not possible to completely eliminate any differences between various environments, but certainly can be minimized. The continuous improvement programs can be championed as described below:

```
1 Describe the Issue --> Determine the Cause -->  R
```

Example 1:

Describe the Issue :	Frequent intermittent database errors in DEV environment.
Determine the Cause:	Multiple project teams are using the same database.
Resolve the issue:	Refresh the database weekly from the test environment, and get the DEV teams sharing the same database to submit the scripts to a central utility program that will run the delta scripts to make the relevant project specific schema changes and the data load. Create a dash-board that will enable project teams to book and share relevant databases. It will also increase communication across project teams sharing the same database. This will create a more controlled environment.
Follow-Up:	The intermittent issues were minimized due to better coordinated and disciplined database changes, improved visibility of who is using what, and better communication across project teams.



Describe the Issue:	Builds are constantly broken.
Determine the Cause:	Individual developers built changes locally and isolated from the changes made by the other developers until they checkout others' changes. The unit test coverage was bad as well.
Resolve the issue:	Have a better test coverage and monitor test coverage with the tools like sonar. Implement a continuous integration and build server that continuously builds the code that are checked-in to the version control repository and also execute the unit test cases to ensure that nothing is broken. If the build or unit tests fail, the team members are notified via emails. This will also ensure that the developers will be more disciplined to test their code locally before checking them into the version control repository.
Follow-Up:	Developers get into the rhythm of code, build, test, and integrate to minimize any down time caused by broken builds.

Some organizations are very passionate about hiring professionals with experience in agile practices. If you had experience in facilitating agile practices, it will be looked upon very favorably by those organizations.

It is a best practice to have project retrospective meetings after each major mile stones of a project to see what worked well, what did not work well, lessons learned, and how could you improve on things that did not work well for the next milestone release.

Making technical changes are easier than changing human behavior. So, to champion continuous improvement processes, one needs to have good technical skills complemented with great soft skills.

Q6. Can you think of a time where you proactively prevented the likelyhood of embarassing mistakes?
A6. Here are some examples of the embarrassing mistakes that software professionals need to
be aware of.

- Inadvertently spamming your clients with unintended emails during development or testing phases. Avoid these types of issues with proper configuration files using internal email exchange servers and mocked up data using your email address to receive and send emails. A -DisProd=false JVM argument check will make it more fool proof.

- Hard coding production URLs directly or indirectly in your code base or performance test scripts. Avoid these issues by always externalizing server, URL, and other environment specific details to relevant configuration files.
- Security holes that allow URL parameters to be manipulated to view others' personal details and sensitive information. Test your application by modifying the URL parameters to identify any obvious security holes. For example, if you are using a URL like http://myapp.com/accountId=123 , try replacing the accountId to some other accountId.
- Input search boxes break when special characters like '%', &, etc are entered. Test your input boxes for special characters and implement proper client side and server side input validation.
- Other environmental issues like testing the application against a cut-down database to later find that the application does not perform or scale well with the production size data. Avoid these issues with proper performance testing in a production like environment.
- It is a known fact that the software artifacts need to be properly versioned. This is a mostly adhered standard practice. But when designing an application with dynamic application configuration values or validation rules via database, some designers overlook the importance of versioning. In larger applications, at any point in time there will be multiple streams of parallel development work going on. Many development streams will be sharing the same database. So, if you don't have proper versioning, any changes to the configuration values or validation rules can break other development streams. With proper versioning, multiple streams can use the relevant version numbers.
- Some developers fail to understand or ignore the difference between a local call and a remote call. The remote calls have more performance overheads

due to network round trips and serialization/deserialization of data. So, avoid making remote calls (via RPC or web services) from a loop on the client side. This will end up in many remote calls. It is better to have the loop on the service side and get the client to make a single remote call by passing a collection of data. This collection of data can be looped through on the server side by making local calls within the same process.

- Never hard code system error or warning messages, config parameters like host name, web service URLs, service timeouts, etc and application specific business data like max account count, bulk data load threshold, feature on/off choices, etc. The error/warning messages and UI labels need to be stored in internationalizable files. Additional files can be added relevant to the locale. The system configuration parameters need to be configured via environment specific properties files and stored separately from the deployable artifacts. The business specific configuration data can be stored either in a config table in a database or in a configuration file stored outside the deployable artifact.

Q7. Can you think of times where you properly though through **security** implications?
A7. For example, if your application is sending reports containing sensitive information to clients, it is imperative that the reports are adequately protected. This can be achieved by sending the reports as password protected zip files. This involves additional considerations listed below to minimize security issues.

- The generated report and the password to unzip the report(s) need to be sent out in separate emails.
- Individual (i.e. per client) passwords need to be used and regularly (say every 3 months) recycled.

As a Java Architect

Java architecture & design concepts interview Q&As with diagrams | What should be a typical Java EE architecture?

Senior Java developers must have a good handle on

- These individual passwords need to be stored encrypted in a database or a file system.
- A master password needs to be used to encrypt and decrypt the individual passwords. Never store them in clear text.
- The master password itself needs to be securely stored.

Q8. Can you think of times where using the **right tool** and your **"know how"** icreased your productivity?
A8. For example, if your application is sending reports containing sensitive information to clients, it is imperative that the reports are adequately protected. This can be achieved by sending the reports as password protected zip files. This involves additional considerations listed below to minimize security issues.

- JMS based log4j logging to measure metrics, especially all the asynchronous processes in a large trading application.
  - Use log4j JMS appender or a custom JMS appender to send log messages to a queue.
  - Use this appender in your application via Aspect Oriented Programming (AOP – e.g Spring AOP, AspectJ, etc) or dynamic proxy classes to non-intrusively log relevant metrics to a queue. It is worth looking at Perf4j and context based logging with MDC (Mapped Diagnostic Contexts) or NDC (Nested Diagnostic Contexts) to log on a per thread basis to correlate or link relevant operations. Perf4J is a great framework for performance logging. It's non-intrusive and really fills the need for accurate performance logging
  - A stand-alone listener application needs to be developed to dequeue the performance metrics messages from the queue and write to a database or a file system for further analysis and reporting purpose. This listener could be written in Java as a JMX service using JMS or

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

Preparing for Java written & coding tests

via broker service like webMethods, TIBCO, etc. This service needs to correlate related messages using a correlation id to determine the elapsed time at the various points in the system.
    - Finally, relevant SQL (for database) or Splunk/regular expression (for flat files) based queries can be written to aggregate and report relevant metrics in a customized way.
- Tools like regexpal.com is very handy to quickly verify your regular expressions while working on a project. The regex keywords and the results are highlighted in color for better clarity.
- Notepad++ is a powerful text editor with many handy features like converting an unquoted CSV string to quoted csv string, converting comma separated entries to “new line” separated entries, extracting column valuee, converting these column values to single quoted csv string to be used in SQL “in” clasuse, etc.
- Spreadsheets like MS Excel is handy to construct SQL queries with static and dynamic data. The Excel concatenate character “&” can be used to construct insert statements from tabular data. The formula is constructed for the initial row 2 data, and then copied down for the remaining rows. Saving you lots of typing. The “$” symbols are used to fix row, column or both.

```
"insert into person ("
&
$A$1
&
", "
&
$B$1
&
", "
$C$1
&
") values ('"
&
$A2
&
"','"
```

open all | close all
♦ Complete the given
Can you write code? (
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...to go places?

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

```
17 &
18 $B2
19 &
20 "','"
21 &
22 $C2
23 &
24 ")"
25
```

- SoapUI and Firefox plugins (e.g. poster plugin) to debug issues rtelating to web services.

Q8. Can you think of a time where you acted as a **"change agent"**?
A8. A change agent not only changes a system's or application's behavior but also people's behavior, attitude, and culture towards writing quality software. People's behavior can be changed by enforcing good coding standards. One way to enforce these standards is through regular peer code reviews. Code review sessions can catch more bugs earlier on in the software development life cycle. It gives junior developers to learn from more experienced professionals. It motivates developers to avoid common mistakes, sloppy code, and inadequate unit test coverage. Automated code review tools like sonar can also be used to enforce consistency in code, identify potential bugs and redundant code snippets. The peer code reviews can be better managed through tools like Crucible.

Substandard software development processes can also demotivate developers and adversely impact their behavior and attitude. The development processes need to be continuously improved by adopting good tools and practices. A build and deploy process that you can repeat consistently across team members, across environments, and across different versions of the software is critical to successfully shipping your software or
deploying it into production. This can make developers' life a lot easier by allowing them to concentrate more on what they do best rather than wasting time on fixing bad processes or performing administrative tasks.

Finally, a framework needs to be put in place for adequate documentation. Most software professionals don't like to write documentation. Proper guidelines and templates need to be created to encourage developers to write simple enough, but not too simple documentation. No body is going to read complex documentation. Documentation needs to be written with the target audience in mind. All the documents should be easily accessible. Good document management systems/repositories allow developers to find documents quickly by project, contents, date, author, etc and saves lots of frustrations.

Popular Posts

♦ 11 Spring boot interview questions & answers

852 views

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

824 views

18 Java scenarios based interview Questions and Answers

447 views

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

400 views

♦ 7 Java debugging interview questions & answers

311 views

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

301 views

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

291 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.

‹ When to use which Java data structure? and why?

Are you reinventing yourself as a programmer? ›

Posted in Ice Breaker Interview Q&A, member-paid

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.

© 2016 Java-Success.com ↑ Responsive Theme powered by WordPress