♦ Design pattern intents interview Q&A

Posted on September 9, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

Q1. Why do Proxy, Decorator, Adapter, Bridge, and Facade design patterns look very similar?

A1. Some design patterns do have subtle differences, and it is important to understand the intent of a pattern. Decorator and chain of responsibility may look similar but the intent is different. Decorator has a subtle difference with a proxy design pattern. Proxy, Decorator, Adapter, and Bridge are all variations on “wrapping” a class. Facade design pattern is a container for the classes in another sub system.

One of the reasons to use a design pattern is to talk the common vocabulary with the designers and the developers. The intent and naming conventionsof these design patterns

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

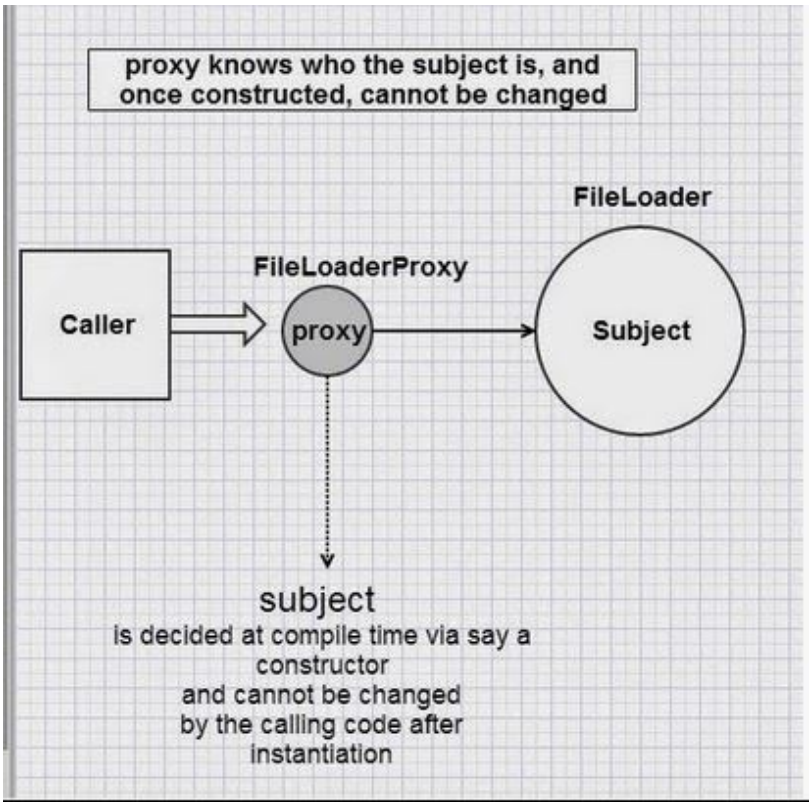
open all | close all

- Ice Breaker Interview
- Core Java Interview C
- JEE Interview Q&A (3
- Pressed for time? Jav
  - Job Interview Ice B
  - FAQ Core Java Jol
  - 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 desigr
    - ♦ 18 Agile Devel
    - ♦ 5 Ways to debu
    - ♦ 9 Java Transac
    - ♦ Monitoring/Pro
    - 02: ♥♦ 13 Tips to

form a common vocabulary. For example, FileLoaderProxy, FileLoaderBridege, etc.

Proxy

is good to act as a surrogate to provide performance optimization, synchronization, remote access, house keeping, etc. The binding of the actual subject to the proxy happens at **compile-time**.

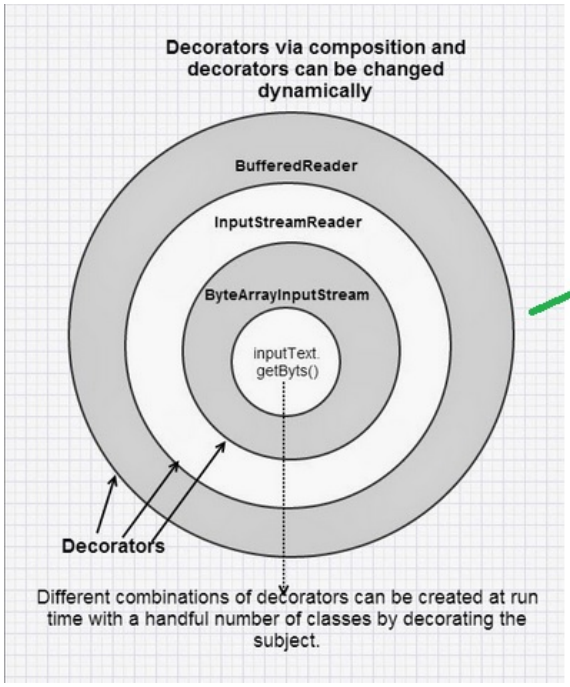


proxy design pattern

Decorator

is good to avoid out of control type hierarchies. A decorator is also known as a “smart proxy” because it enhances the behavior of a subject at run time. In other words binding is dynamic. The Java I/O classes are good example of a decorator design pattern. At run time different permutations can can be carried out **via composition**.

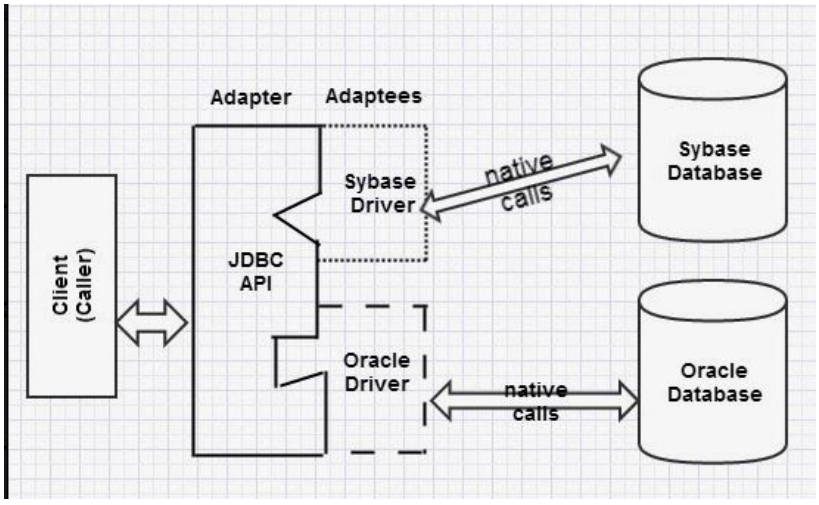
15 Security key a
4 FAQ Performa
4 JEE Design Pa
5 Java Concurre
6 Scaling your Ja
8 Java memory r
OOP & FP Essentia
Code Quality Job In
SQL, XML, UML, JSC
Hadoop & BigData Int
Java Architecture Inte
Scala Interview Q&As
Spring, Hibernate, & M
Spring (18)
Spring boot (4)
Spring IO (1)
Spring JavaConf
01: ♥♦ 13 Spring
01b: ♦ 13 Spring
02: ► Spring DII
03: ♥♦ Spring DI
04 ♦ 17 Spring b
05: ♦ 9 Spring Be
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 Fir
07: Hibernate mi
08: Hibernate au
09: Hibernate en
10: Spring, Java
11: Hibernate de
12: Hibernate cu
AngularJS (2)



decorator design pattern

Adapter

Adapts at run time like the decorator design pattern. Adapter design pattern is one of the structural design patterns and its intent is to get two unrelated interfaces work together. Think of using a laptop in UK that was bought in Japan as the sockets are different, and you need an adapter. So, the adapter's intent is to adapt between the Japanese laptop plug with UKs wall socket. The key point is that parties are different. Japanese laptop used in third-party or external (i.e. UK) wall socket.



Git & SVN (6)
JMeter (2)
JSF (2)
Maven (3)
Testing & Profiling/Sa
Other Interview Q&A
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 I
Low Latency (7)
Memory Managemen
Performance (13)
QoS (8)
Scalability (4)
SDLC (6)
Security (13)
Transaction Managen

80+ step by step Java Tutorials

open all | close all
Setting up Tutorial (6)
Tutorial - Diagnosis (2
Akka Tutorial (9)
Core Java Tutorials (2
Hadoop & Spark Tuto

Adapter design pattern

Adapter is used when you have an abstract interface, for example a JDBC API and you want to map that interface to another object which has similar functional role, but a different interface, for example different JDBC drivers for different databases like Oracle, Sybase, DB2, SQL server, MySQL, etc. The JEE have multiple adaptors for JMS, JNDI, JDBC, JCA, etc. The drivers and implementations are generally provided by the third party vendors. For example, JMS implementations provided by third-party vendors and open source providers web Methods, IBM MQ Series, ActiveMQ, etc. You can accomplish this using either inheritance or composition.

— **Class Adapter** – This form uses Java inheritance and extends the source interface.
— **Object Adapter** – This form uses Java Composition and adapter contains the source object.

Bridge

is very similar to Adapter, but you call it a Bridge when you define both the abstract interface and the underlying implementation (no external or third-party vendors). Adapter makes things work after they're designed, but bridge makes them work before they are. Abstraction and implementation should be bound at compile time. You need to swap for different implementations. For example, you are designing a graphics application that needs to swap between graphics driver and printer driver for the same draw( ) .

As shown below, you can have different permutations of a user like local user, external user, provisional user, local admin user, local support user, and so on. re-factoring this into users and roles will prevent the explosion of class hierarchy. A decorator does this at run-time, whereas a bridge does this at compile-time.

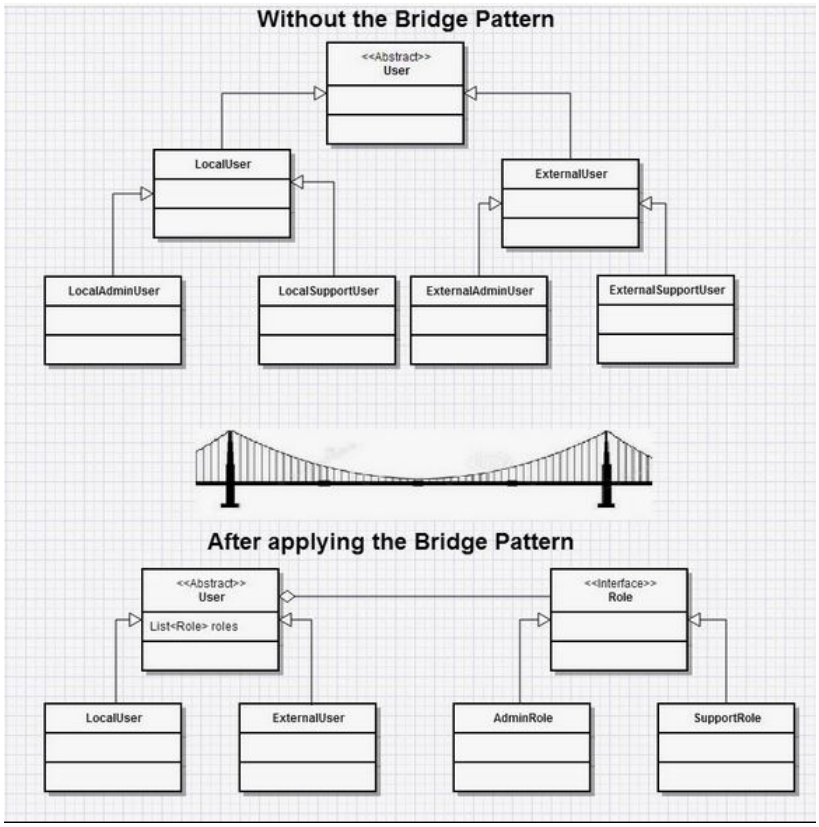
JEE Tutorials (19)
Scala Tutorials (1)
Spring & HIbernate Tu
Tools Tutorials (19)
Other Tutorials (45)

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 & Resum

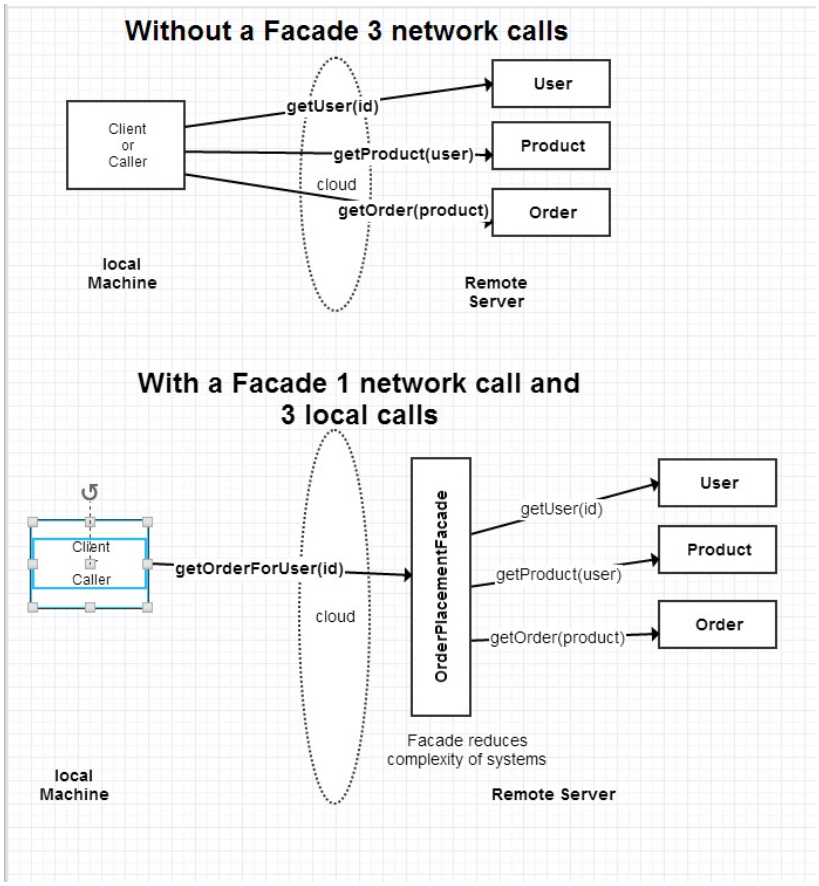


Bridge design pattern

If you are wring unit tests, the bridge pattern is indispensable when it comes down to implementing mocks and mock services. You can use a bridge (or mock) web services in integration testing until the real services become available.

Facade

is a higher-level interface to a subsystem of one or more classes. Think of Facade as a sort of container for other objects, as opposed to a wrapper.



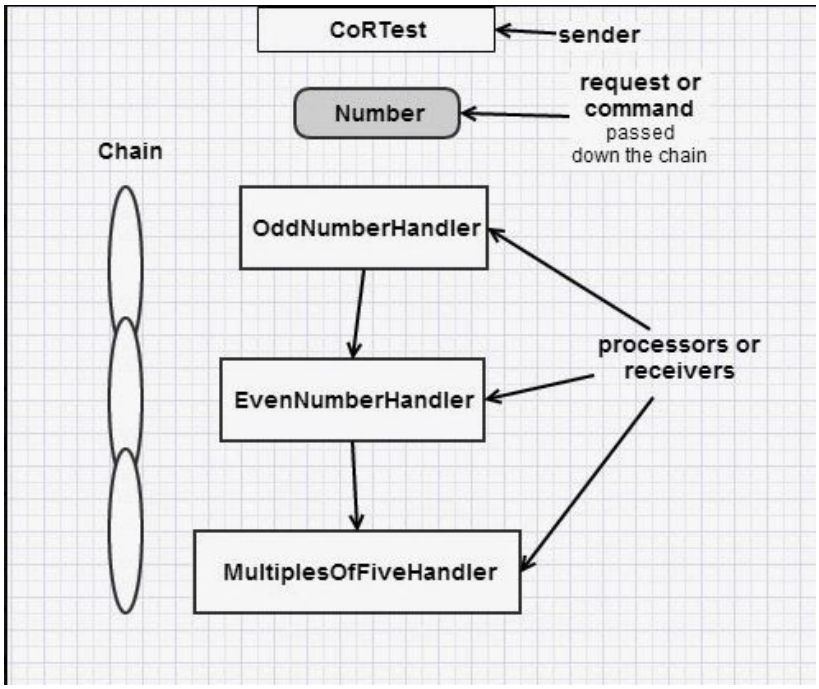
Facade design pattern

For example, when you have remote calls via EJBs, Web Services, RMI, etc, making fine grained remote calls each time can be expensive and adversely affects performance. So, in between the caller and the callee, you can define a facade that makes many fine grained local calls for a single remote call from the caller.It also simplifies the client code with just a single method call by hiding the complexity of the subsystem classes via a facade.

Chain of Responsibility

Use the Chain of Responsibility pattern when you can conceptualize your program as a chain made up of links, where each link (or processor) can either handle a request/command or pass it down the chain. The fact that you can break the chain at any point and each chain can either handle or pass it differentiates the Chain of Responsibility pattern from the Decorator pattern. In other words, if you

chain, each link will be called along the chain, and with a decorator, you're not guaranteed of this order, but you can be confident that additional responsibilities can be attached.



Chain of Responsibility design pattern

Example 1: Java's own Exception Handling mechanism because

1. Sender of an exception will not know which object in the chain will serve its request.
2. Each processor in the chain may decide to serve the exception by catching and logging or
3. wrapping it with an application specific exception and then rethrowing it to the caller or
4. don't handle it and leave it to the caller

Example 2: Servlet filters are implementation of the chain of responsibility pattern.

```
1 void doFilter(..) {
2     // do things like security check, parsing pay
3
4     // invoke the servlet, or any other filters m
5     chain.doFilter(..);
6
```

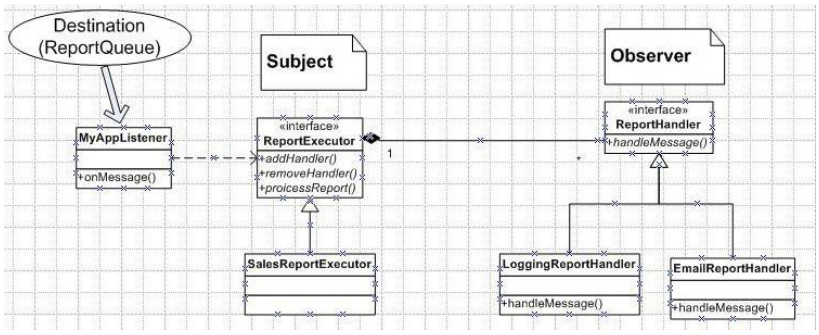
```
7       // do stuff after the servlet finishes
8   }
9
```

Example 3: An interceptor is generally a link in a chain of responsibility. Interceptors in Java frameworks like Spring MVC, Struts 2, etc intercept, handle or pass a message down the chain for other interceptors to handle.

Observer

The Observer pattern is a behavioral design pattern that allows an object (an Observer) to watch another object (a Subject). The subject and observer to have a publish/subscribe relationship. Observers can register to receive events from the Subject. Some of the practical uses of observer pattern are:

— When a change to one object requires changing of others, and you don't know how many objects need to be changed.
— When an object should be able to notify other objects without making assumptions about who these objects are and not tightly coupling them.
— When a report is received or an event occurs, a message needs to be sent to the subscribed handlers.



Observer design pattern

Examples:

— The Java Message Service (JMS) models the observer pattern, with its guaranteed delivery, non-local distribution, and persistence, to name a few of its benefits. The JMS

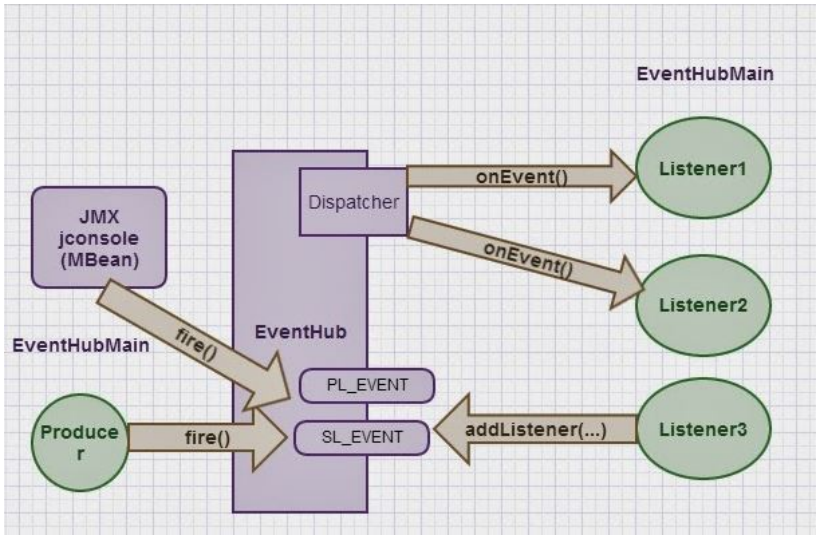
publish-subscribe messaging model allows any number of subscribers to listen to topics of interest. When a message for the published topic is produced, all the associated subscribers are notified.

— The Java Foundation Classes (JFC) like JList, JTree and the JTable components manipulate data through their respective data models. The components act as observers of their data models.

— In the java.util package, we have the Observer interface and the Observable class.

— In an MVC (Model-View-Controller) architecture, the view gets its own data from the model or in some cases the controller may issue a general instruction to the view to render itself. In others, the view acts as an observer and is automatically notified by the model of changes in state that require a screen update.

— Event Driven Architecture aka EDA loosely couples event producers and event consumers.



EDA

An event can be defined as “a change in state”. For example, when an event producer fires an event to notify all its registered listeners that either “securities” or “security prices”

have been loaded, the listeners are notified to update their data via a synchronous or asynchronous dispatcher. Both the “Event” producers and listeners are loosely coupled via an “EventHub” and “Event”. An “EventHub” is used to register and unregister listeners.

The “EventHub” can be registed as a JMX bean to control behaviors at runtime via a jconsole like firing an event, count number of events, etc.

Popular Posts

♦ 11 Spring boot interview questions & answers

827 views

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

767 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

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

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 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**

‹ Java BDD (Behavior Driven Development) interview Q&A

How will you go about improving on the following Java code? ›

Posted in GoF Patterns, Java Key Area Essentials, 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

 ↑