

# Java-Success.com

Industrial strength Java/JEE Career Companion for those who want to go places


[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](#) › [Core Java Interview Q&A](#) › [Java Overview](#) › 02: ♥♦ Java

Compile-time Vs Run-time Interview Q&As

## 02: ♥♦ Java Compile-time Vs Run-time Interview Q&As

Posted on [September 17, 2014](#) by [Arulkumaran Kumaraswamipillai](#) — No

[Comments](#) ↓

16

Like

Share

Tweet

1

G+1

6

Share

During development and design, one needs to think in terms of **compile-time**, **run-time**, and **build-time**. It will also help you understand the fundamentals better. These are beginner to intermediate level questions.

**Q1.** What is the difference between line A & line B in the following code snippet?

```
1 public class ConstantFolding {
2     static final int number1 = 5;
3     static final int number2 = 6;
```

[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 breakers
- 02: ♥♦ 8 real life examples
- 03: ♦10+ Know your Java
- 04: Can you think of 10 Java
- 05: ♥ What job interviews
- 06: ► Tell me about Java
- 07: ♥ 20+ Pre interview

[Core Java Interview Questions](#)

[Java Overview \(4\)](#)

01: ♦ ♥ 17 Java Overview

```

4      static int number3 = 5;
5      static int number4 = 6;
6
7      public static void main(String[] args) {
8          int product1 = number1 * number2;
9          int product2 = number3 * number4;
10     }
11 }
12

```

**A1.** Line A, evaluates the product at **compile-time**, and Line B evaluates the product at **runtime**. If you use a Java Decompiler (e.g. jd-gui), and decompile the compiled *ConstantFolding.class* file, you will see why as shown below.

```

1 public class ConstantFolding
2 {
3     static final int number1 = 5;
4     static final int number2 = 6;
5     static int number3 = 5;
6     static int number4 = 6;
7
8     public static void main(String[] args)
9     {
10         int product1 = 30;
11         int product2 = number3 * number4;
12     }
13 }
14

```

Constant folding is an optimization technique used by the Java compiler. Since final variables cannot change, they can be optimized. Java Decompiler and javap command are handy tool for inspecting the compiled (i.e. byte code ) code.

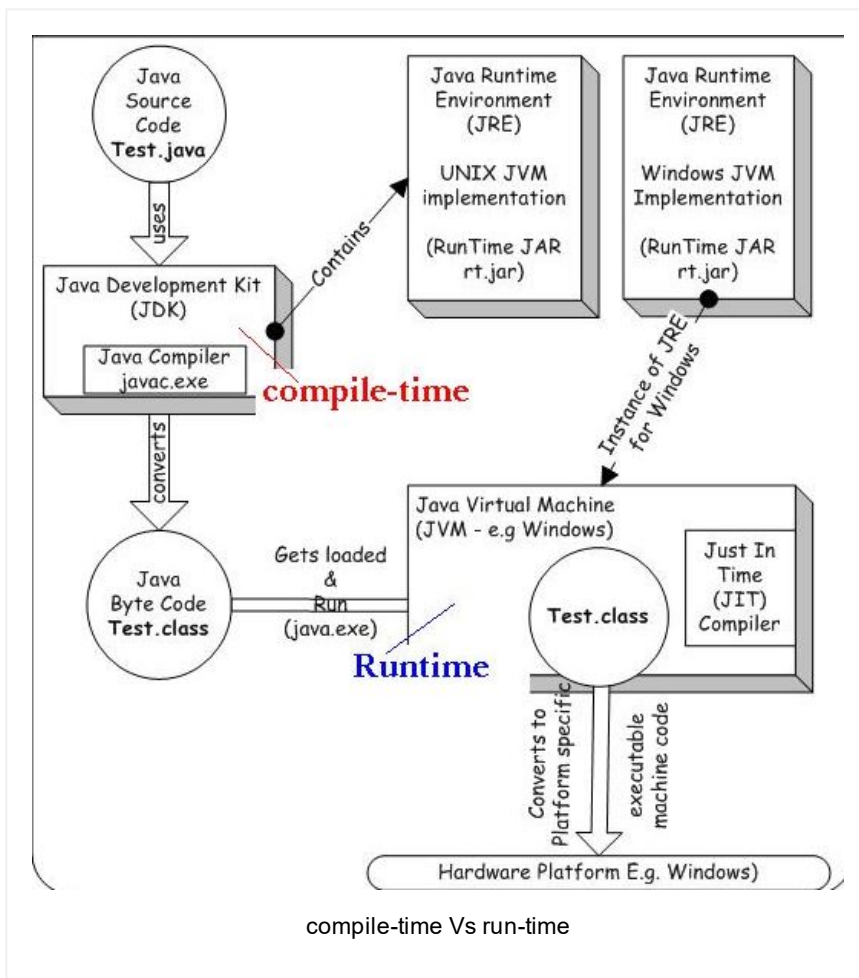
**Q2** Can you think of other scenarios other than code optimization, where inspecting a compiled code is useful?

**A2** Generics in Java are compile-time constructs, and it is very handy to inspect a compiled class file to understand and troubleshoot generics.

**Q3.** Does this happen during compile-time, runtime, or both?

**A3.**

02: ♥♦ Java Con
03: ♦ 9 Core Jav
04: ♦ Top 10 mos
☐ Data types (6)
01: Java data ty
02: ♥♦ 10 Java S
03: ♦ ♥ Java aut
04: Understandir
05: Java primitiv
Working with Da
☐ constructors-methc
Java initializers,
☐ Reserved Key Wor
♥♦ 6 Java Modifi
Java identifiers
☐ Classes (3)
♦ Java abstract c
♦ Java class load
♦ Java classes a
☐ Objects (8)
▶ Beginner Jav
♥♦ HashMap & H
♦ 5 Java Object i
♦ Java enum inte
♦ Java immutabl
♦♥ Object equals
Java serialization
Mocks, stubs, dc
☐ OOP (10)
♥ Design princip
♦ 30+ FAQ Java
♦ Why favor cor
08: ♦ Write code
Explain abstracti
How to create a
Top 5 OOPs tips
Top 6 tips to go
Understanding C
What are good r
☐ GC (2)
♦ Java Garbage



**Method overloading:** 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.

```

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

```

If the compiler were to compile the statement:

```

1 evaluate("My Test Argument passed to param1");

```

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

**Method overriding:** This happens at runtime. This is also called runtime polymorphism because the compiler does not

03: Java GC tun

Generics (5)

♥ Java Generics

♥ Overloaded m

♦ 12 Java Gener

♦ 7 rules to reme

3 scenarios to ge

FP (8)

01: ♦ 19 Java 8 I

02: ♦ Java 8 Stre

03: ♦ Functional

04: ♥♦ Top 6 tips

05: ♥ 7 Java FP

Fibonacci numb

Java 8 String str

Java 8: What is c

IO (7)

♥ Reading a text

♦ 15 Java old I/C

06: ♥ Java 8 way

Processing large

Processing large

Read a text file f

Reloading config

Multithreading (12)

01: ♥♦ 15 Beginn

02: ♥♦ 10+ Java

03: ♦ More Java

04: ♦ 6 popular J

05: ♦ How a thre

06: ♦ 10+ Atomic

07: 5 Basic multi

08: ♦ ThreadLoc

09: Java FutureT

10: ♦ ExecutorSe

Java ExecutorSe

Producer and Co

Algorithms (5)

♦ Splitting input t

♦ Tree traversal :

♥ ♦ Java coding

and cannot know which method to call. Instead, the JVM must make the determination while the code is running.

```

1 public class A {
2     public int compute(int input) {           //me
3         return 3 * input;
4     }
5 }
6
7 public class B extends A {
8     @Override
9     public int compute(int input) {           //me
10        return 4 * input;
11    }
12 }
13

```

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 }

```

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”.

**Generics (aka type checking):** This happens at compile-time. The compiler checks for the type correctness of the program and translates or rewrites the code that uses generics into non-generic code that can be executed in the current JVM. This technique is known as “type erasure”. In other words, the compiler erases all generic type information contained within the angle brackets to achieve backward compatibility with JRE 1.4.0 or earlier editions.

```

1 List<String> myList = new ArrayList<Stri

```

after compilation becomes:

Searching algori

Swapping, partiti

Annotations (2)

8 Java Annotati

More Java annot

Collection and Data

Find the first ne

Java Collection

Java Iterable \

HashMap & H

Sorting objects

02: Java 8 Stre

04: Understandi

4 Java Collection

If Java did not ha

Java 8: Different

Part-3: Java Tre

Sorting a Map by

When to use whi

Differences Between

Java Iterable \

Multithreading

Why do Proxy,

Core Java Modif

Differences betw

Java Collection i

Event Driven Progr

Event Driven Pr

Event Driven Pr

Exceptions (2)

Java exception

Top 5 Core Java

Java 7 (2)

Java 7 fork and j

Java 7: Top 8 ne

Java 8 (24)

01: 19 Java 8 I

02: Java 8 Stre

03: Functional

04: Top 6 tips

04: Convert Lists

```
1 List myList = new ArrayList(10);
```

**Annotations:** You can have either run-time or compile-time annotations.

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

@Override is a simple compile-time annotation to catch little mistakes like typing toString( ) instead of toString( ) in a subclass. User defined annotations can be processed at compile-time using the Annotation Processing Tool (APT) that comes with Java 5. In Java 6, this is included as part of the compiler itself.

```
1 public class MyTest{
2     @Test
3     public void testEmptyness( ){
4         org.junit.Assert.assertTrue(getList( ).
5     }
6
7     private List getList( ){
8         //implemenation goes here
9     }
10 }
```

@Test is an annotation that JUnit framework uses at runtime with the help of reflection to determine which method(s) to execute within a test class.

```
1 @Test (timeout=100)
2 public void testTimeout( ) {
3     while(true);    //infinite loop
4 }
```

The above test fails if it takes more than 100ms to execute at runtime.

```
1 @Test (expected=IndexOutOfBoundsException.class)
2 public void testOutOfBounds( ) {
3     new ArrayList<Object>( ).get(1);
4 }
```

04: Understanding

05: ♥ 7 Java FP

05: ♦ Finding the

06: ♥ Java 8 way

07: ♦ Java 8 API

08: ♦ Write code

10: ♦ ExecutorSe

Fibonacci numbe

Java 8 String str

Java 8 using the

Java 8: 7 useful

Java 8: Different

Java 8: Does "O

Java 8: What is c

Learning to write

Non-trivial Java 8

Top 6 Java 8 fea

Top 8 Java 8 fea

Understanding J

☐ JVM (6)

♦ Java Garbage

01: jvisualvm to

02: jvisualvm to

05: Java primitiv

06: ♦ 10+ Atomic

5 JMX and MBea

☐ Reactive Program

07: Reactive Pro

10: ♦ ExecutorSe

3. Multi-Threadir

☐ Swing & AWT (2)

5 Swing & AWT i

Q6 – Q11 Swing

☐ JEE Interview Q&A (3

☐ JEE Overview (2)

♦ 8 Java EE (aka

Java EE interview

☐ Web basics (8)

01: ♦ 12 Web ba

02: HTTP basics

03: Servlet inter

The above code fails if it does not throw `IndexOutOfBoundsException` or if it throws a different exception at runtime. User defined annotations can be processed at runtime using the new `AnnotatedElement` and “Annotation” element interfaces added to the Java reflection API.

**Exceptions:** You can have either runtime or compile-time exceptions.

`RuntimeException` is also known as the unchecked exception indicating not required to be checked by the compiler.

`RuntimeException` is the superclass of those exceptions that can be thrown during the execution of a program within the JVM. A method is not required to declare in its throws clause any subclasses of `RuntimeException` that might be thrown during the execution of a method but not caught.

Example: `NullPointerException`,  
`ArrayIndexOutOfBoundsException`, etc

Checked exceptions are verified by the compiler at compile-time that a program contains handlers like throws clause or `try{} catch{} blocks` for handling the checked exceptions, by analyzing which checked exceptions can result from execution of a method or constructor.

**Aspect Oriented Programming (AOP):** Aspects can be weaved at compile-time, post-compile time, load-time or runtime.

- **Compile-time:** weaving is the simplest approach. When you have the source code for an application, the AOP compiler (e.g. `ajc` – AspectJ Compiler) will compile from source and produce woven class files as output. The invocation of the weaver is integral to the AOP compilation process. The aspects themselves may be in source or binary form. If the aspects are required for the affected classes to compile, then you must weave at compile-time. ?

04: JSP overview

05: Web patterns:

06: ♦ MVC0, MV

07: When to use

08: Web.xml inte

WebService (11)

01: ♥♦ 40+ Java

02: ♦ 6 Java RE

03: ♥ JAX-RS hc

04: 5 JAXB inter

05: RESTful We

06: RESTful Wel

07: HATEOAS R

08: REST constr

09: 11 SOAP We

10: SOAP Web

11: ♥ JAX-WS hc

JPA (2)

10: Spring, Java

8 JPA interview c

JTA (1)

JTA interview Q&

JDBC (4)

♦ 12 FAQ JDBC

JDBC Overview

NamedParamete

Spring, JavaCon

JMS (5)

♦ 16 FAQ JMS ir

Configuring JMS

JMS versus AMC

Spring JMS with

Spring JMS with

JMX (3)

5 JMX and MBe

Event Driven Pr

Yammer metrics

JNDI and LDAP (1)

JNDI and LDAP

Pressed for time? Jav

Job Interview Ice B



- **Post-compile:** weaving is also sometimes called binary weaving, and is used to weave existing class files and JAR files. As with compile-time weaving, the aspects used for weaving may be in source or binary form, and may themselves be woven by aspects.?
- **Load-time:** weaving is simply binary weaving deferred until the point that a class loader loads a class file and defines the class to the JVM. To support this, one or more “weaving class loaders”, either provided explicitly by the run-time environment or enabled through a “weaving agent” are required.
- **Runtime:** weaving of classes that have already been loaded to the JVM.

**Inheritance** – happens at compile-time, hence is static.

**Delegation or composition** – happens at run-time, hence is dynamic and more flexible.

**Q.** Have you heard the term “**composition should be favored over inheritance**”? If yes, what do you understand by this phrase?

**A.** Inheritance is a polymorphic tool and is not a code reuse tool. Some developers tend to use inheritance for code reuse when there is no polymorphic relationship. The guide is that inheritance should be only used when a subclass ‘is a’ super class.

- Don’t use inheritance just to get code reuse. If there is no ‘is a’ relationship then use composition for code reuse. Overuse of implementation inheritance (uses the “extends” key word) can break all the subclasses, if the super class is modified. This is due to **tight coupling** occurring between the parent and the child classes happening at **compile time**.
- Do not use inheritance just to get polymorphism. If there is no ‘is a’ relationship and all you want is polymorphism then use interface inheritance with composition, which gives you code reuse and **runtime** flexibility.

01: ♦ 15 Ice bre  
02: ♥♦ 8 real life  
03: ♦10+ Know y  
FAQ Core Java J  
♥♦ Q1-Q10: Top  
♦ Q11-Q23: Top  
♦ Q24-Q36: Top  
♦ Q37-Q42: Top  
♦ Q43-Q54: Top  
01: ♥♦ 15 Beginr  
02: ♥♦ 10+ Java  
FAQ JEE Job Inter  
♦ 12 FAQ JDBC  
♦ 16 FAQ JMS ir  
♦ 8 Java EE (aka  
♦ Q01-Q28: Top  
♦ Q29-Q53: Top  
01: ♦ 12 Web ba  
06: ♦ MVC0, MV  
JavaScript mista  
JavaScript Vs Ja  
JNDI and LDAP  
JSF interview Q  
JSON interview  
FAQ Java Web Ser  
01: ♥♦ 40+ Java  
02: ♦ 6 Java RE  
05: RESTFul We  
06: RESTful Wel  
09: 11 SOAP We  
Java Application Ar  
001A: ♦ 7+ Java  
001B: ♦ Java arc  
04: ♦ How to go  
Hibernate Job Inter  
01: ♥♦ 15+ Hiber  
01b: ♦ 15+ Hiber  
06: Hibernate Fil  
8 JPA interview c  
Spring Job Intervie  
♦ 11 Spring boot

This is the reason why the GoF (Gang of Four) design patterns favor composition over inheritance. The interviewer will be looking for the key terms — “**coupling**“, “**static versus dynamic**” and “**happens at compile-time vs runtime**” in your answers. The runtime flexibility is achieved in composition as the classes can be composed **dynamically** at runtime either conditionally based on an outcome or unconditionally. Whereas an inheritance is **static**.

**Q4.** Can you differentiate compile-time inheritance and runtime inheritance with examples and specify which Java supports?

**A4.**

The term “inheritance” refers to a situation where behaviors and attributes are passed on from one object to another. The Java programming language natively only supports compile-time inheritance through subclassing as shown below with the keyword “extends”.

```

1 public class Parent {
2     public String saySomething( ) {
3         return "Parent is called";
4     }
5 }
6

```

```

1 public class Child extends Parent {
2     @Override
3     public String saySomething( ) {
4         return super.saySomething( ) + ", Child";
5     }
6 }
7

```

A call to saySomething( ) method on the class “Child” will return “Parent is called, Child is called” because the Child class inherits “Parent is called” from the class Parent. The keyword “super” is used to call the method on the “Parent” class. Runtime inheritance refers to the ability to construct the parent/child hierarchy at runtime. Java does not natively support runtime inheritance, but there is an alternative concept known as “delegation” or “composition”, which refers to constructing a hierarchy of object instances at runtime.

[01: ♥♦ 13 Spring](#)  
[01b: ♦ 13 Spring](#)  
[04 ♦ 17 Spring b](#)  
[05: ♦ 9 Spring B](#)  
[Java Key Area Ess](#)  
[♦ Design pattern](#)  
[♥ Top 10 causes](#)  
[♥♦ 01: 30+ Writir](#)  
[♦ 12 Java desigr](#)  
[♦ 18 Agile Develo](#)  
[♦ 5 Ways to debi](#)  
[♦ 9 Java Transac](#)  
[♦ Monitoring/Pro](#)  
[02: ♥♦ 13 Tips to](#)  
[15 Security key :](#)  
[4 FAQ Performa](#)  
[4 JEE Design Pa](#)  
[5 Java Concurr](#)  
[6 Scaling your J](#)  
[8 Java memory i](#)  
[OOP & FP Essenti](#)  
[♦ 30+ FAQ Java](#)  
[01: ♦ 19 Java 8 I](#)  
[04: ♥♦ Top 6 tips](#)  
[Code Quality Job I](#)  
[♦ Ensuring code](#)  
[♦ 5 Java unit tes](#)  
[SQL, XML, UML, JSC](#)  
[ERD \(1\)](#)  
[♦ 10 ERD \(Entity](#)  
[NoSQL \(2\)](#)  
[♦ 9 Java Transac](#)  
[3. Understanding](#)  
[Regex \(2\)](#)  
[♥♦ Regular Expr](#)  
[Regular Express](#)  
[SQL \(7\)](#)  
[♦ 15 Database d](#)  
[♦ 14+ SQL interv](#)  
[♦ 9 SQL scenari](#)  
[Auditing databas](#)



This allows you to simulate runtime inheritance. In Java, delegation is typically achieved as shown below:

```

1 public class Parent {
2     public String saySomething( ) {
3         return "Parent is called";
4     }
5 }
6

```

```

1 public class Child {
2     public String saySomething( ) {
3         return new Parent( ).saySomething( ) +
4     }
5 }
6

```

The Child class delegates the call to the Parent class.  
Composition can be achieved as follows:

```

1 public class Child {
2     private Parent parent = null;
3
4     public Child( ){
5         this.parent = new Parent( );
6     }
7
8     public String saySomething( ) {
9         return this.parent.saySomething( ) +
10    }
11 }
12

```

## Popular Posts

♦ 11 Spring boot interview questions & answers

852 views

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

825 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

Deleting records

SQL Subquery ir

Transaction man

UML (1)

♦ 12 UML intervi

JSON (2)

JSON interview (

JSON, Jackson,

XML (2)

XML basics inter

XML Processing

XSD (2)

11 FAQ XSD inte

XSD reuse inter

YAML (2)

YAML with Java

YAML with Sprin

Hadoop & BigData Int

♥ 01: Q1 – Q6 Had

02: Q7 – Q15 Hada

03: Q16 – Q25 Hac

04: Q27 – Q36 Apa

05: Q37 – Q50 Apa

05: Q37-Q41 – Dat

06: Q51 – Q61 HBa

07: Q62 – Q70 HDI

Java Architecture Inte

♥♦ 01: 30+ Writing

001A: ♦ 7+ Java int

001B: ♦ Java archil

01: ♥♦ 40+ Java W

02: ♥♦ 13 Tips to w

03: ♦ What should l

04: ♦ How to go ab

05: ETL architectur

1. Asynchronous pi

2. Asynchronous pi

Scala Interview Q&As

01: ♥ Q1 – Q6 Scal

02: Q6 – Q12 Scal

03: Q13 – Q18 Sca

311 views

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

301 views

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

292 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](http://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](http://Amazon.com) in 2005, and sold

- 04: Q19 – Q26 Sca
- 05: Q27 – Q32 Sca
- 06: Q33 – Q40 Sca
- 07: Q41 – Q48 Sca
- 08: Q49 – Q58 Sca
- 09: Q59 – Q65 Hig
- 10: Q66 – Q70 Pat
- 11: Q71 – Q77 – S
- 12: Q78 – Q80 Rec
- Spring, Hibernate, & I
- Spring (18)
- Spring boot (4)
- ♦ 11 Spring bc
- 01: Simple Sp
- 02: Simple Sp
- 03: Spring box
- Spring IO (1)
- Spring IO tuto
- Spring JavaConl
- 10: Spring, Ja
- Spring, JavaC
- Spring, JavaC
- Spring, JavaC
- 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

35,000+ copies. Books are outdated and replaced with this subscription based site.

[QoS interview Q&A](#)[4 Java Collection Quizzes](#)

Posted in [Java Overview](#)

## Leave a Reply

Logged in as geethika. [Log out?](#)

### Comment

[08: Hibernate au](#)[09: Hibernate en](#)[10: Spring, Java](#)[11: Hibernate de](#)[12: Hibernate cu](#)[AngularJS \(2\)](#)[♥ 8 AngularJS in](#)[More Angular JS](#)[Git & SVN \(6\)](#)[♥ Git & Maven fc](#)[♥ Merging Vs rel](#)[♥ Understanding](#)[6 more Git interv](#)[8 Git Source cor](#)[Setting up Cygw](#)[JMeter \(2\)](#)[♥ JMeter for test](#)[♦ JMeter perform](#)[JSF \(2\)](#)[JSF interview Q&](#)[More JSF intervi](#)[Maven \(3\)](#)[♥ Git & Maven fc](#)[12 Maven intervi](#)[7 More Maven ir](#)[Testing & Profiling/Sa](#)[Automation Testing](#)[♥ Selenium and](#)[Code Coverage \(2\)](#)[Jacoco for unit te](#)[Maven and Cobr](#)[Code Quality \(2\)](#)[♥ 30+ Java Code](#)[♦ Ensuring code](#)[jvisualvm profiling \(](#)[01: jvisualvm to :](#)[02: jvisualvm to :](#)[03: jvisualvm to :](#)[Performance Testir](#)[♥ JMeter for test](#)[♦ JMeter perform](#)

## Unit Testing Q&A (2)

### BDD Testing (4)

Java BDD (Be

jBehave and E

jBehave and j

jBehave with t

### Data Access Uni

♥ Unit Testing

Part #3: JPA H

Unit Test Hibe

Unit Test Hibe

### JUnit Mockito Sp

#### JUnit Mockito

Spring Con

Unit Testing

Part 1: Unit te

Part 2: Mockit

Part 3: Mockit

Part 4: Mockit

Part 5: Mockit

### Testing Spring T

Integration Un

Unit testing Sp

♦ 5 Java unit tes

JUnit with Hamc

Spring Boot in ui

## Other Interview Q&A 1

### Finance Domain In

12+ FX or Forex

15 Banking & fin

### FIX Interview Q&A

20+ FIX basics in

Finding your way

### Groovy Interview Q

#### Groovy Coding C

Cash balance

Sum grades C

♥ Q1 – Q5 Groov

♦ 20 Groovy clos

♦ 9 Groovy meta

Groovy method c

- Q6 – Q10 Groov
- JavaScript Interview
- JavaScript Top I
- ♥ Q1 – Q10 J
- ♦ Q11 – Q20
- ♦ Q21 – Q30
- ♦ Q31 – Q37
- JavaScript mis
- JavaScript Vs Ja
- JavaScript Vs
- Unix Interview Q&A
- ♥ 14 Unix intervi
- ♥ Hidden Unix, C
- sed and awk to v
- Shell script inter
- Unix history com
- Unix remoting in
- Unix Sed comm
- Free Java Interview
- ▶ Java Integration
- ▶ Java Beginner I
- 02: ▶ Spring DIP, I
- 06: ▶ Tell me abou

## 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**

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

## Preparing for Java written & coding tests



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

## 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.