

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](#) › [Classes](#) › ♦ Java classes and interfaces are the building blocks

♦ Java classes and interfaces are the building blocks

Posted on [August 13, 2014](#) by [Arulkumaran Kumaraswamipillai](#) — No

[Comments](#) ↓

0

Like

Share

Tweet

0

G+1

Share

Q1. Which class declaration is correct if A and B are classes and C and D are interfaces?

- a) class Z extends A implements C, D{ }
- b) class Z extends A,B implements D { }
- c) class Z extends C implements A,B { }
- d) class Z extends C,D implements B { }

A1. a). class Z extends A implements C, D{ }

[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 Interview

A class is a template. A class can extend only a single class (i.e. single inheritance. Java does not support multiple implementation inheritance), but can implement multiple interfaces to achieve multiple interface inheritance. An interface can also extend more than one other interfaces.

```

1
2 interface E extends C,D { //.... }
3

```

Q2. What happens when a parent and a child class has the same variable name?

A2. When both a parent class and its subclass have a field with the same name, this technique is called **variable shadowing or variable hiding**. The variable shadowing depends on the static type of the variable in which the object's reference is stored at compile time and NOT based on the dynamic type of the actual object stored at runtime as demonstrated in polymorphism via method overriding.

Q3. What happens when the parent and the child class has the same non-static method with the same signature?

A3. Unlike variables, when a parent class and a child class each has a non-static method (aka an instance method) with the same signature, the method of the child class overrides the method of the parent class. The method overriding depends on the dynamic type of the actual object being stored and NOT the static type of the variable in which the object reference is stored. The dynamic type can only be evaluated at runtime. As you can see, the rules for variable shadowing and method overriding are directly opposed. The **method overriding enables polymorphic behavior**.

Q4. What happens when the parent and the child class has the same static method with the same signature?

A4. The behavior of static methods will be similar to the **variable shadowing or variable hiding**, and not recommended. It will be invoking the static method of the referencing static object type determined at compile time, and NOT the dynamic object type being stored at runtime.

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 a

Understanding C

What are good r

☐ GC (2)

♦ Java Garbage

Q5. What is the difference between an abstract class and an interface and when should you use them?

A5. In design, you want the base class to present only an interface for its derived classes. This means, you don't want anyone to actually instantiate an object of the base class. You only want to upcast to it (implicit **upcasting**, which gives you polymorphic behavior), so that its interface can be used. This is accomplished by making that class abstract using the **abstract** keyword. If anyone tries to make an object of an abstract class, the compiler prevents it.

The **interface** keyword takes this concept of an abstract class a step further by preventing any method or function implementation at all. You can only declare a method or function but not provide the implementation till Java 7. The class, which is implementing the interface, should provide the actual implementation. The interface is a very useful and commonly used aspect in OO design, as it provides the separation of interface and implementation and enables you to

— Capture similarities among unrelated classes without artificially forcing a class relationship.

Declare methods that one or more classes are expected to implement.

— Reveal an object's programming interface without revealing its actual implementation.

— Model multiple interface inheritance in Java, which provides some of the benefits of full on multiple inheritances, a feature that some object-oriented languages support that allow a class to have more than one super class.

Q6 When will you use an abstract class?

A6 In case where you want to use implementation inheritance then it is usually provided by an abstract base class. Abstract classes are excellent candidates inside of application frameworks. Abstract classes let you define some default behavior and force subclasses to provide any specific behavior. Care should be taken not to overuse implementation inheritance as discussed before. The **template method design pattern** is a good example to use

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 Beginr

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

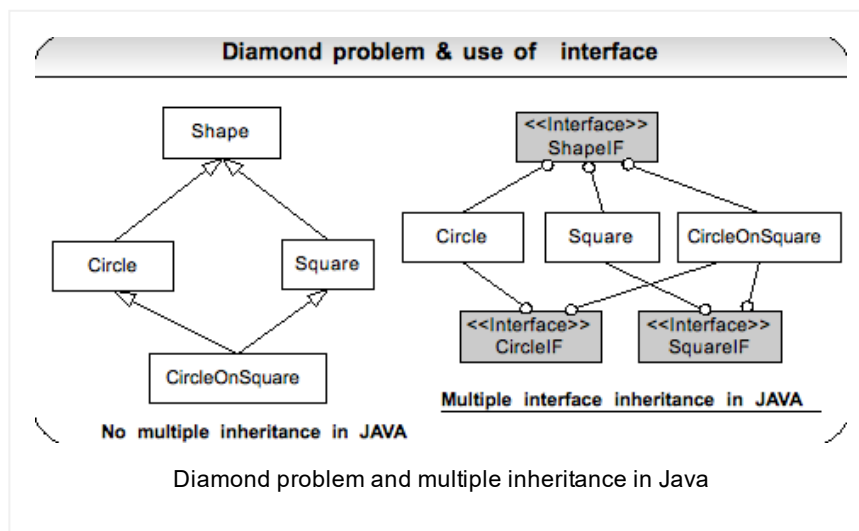
♦ Tree traversal :

♥ ♦ Java coding

an abstract class where the abstract class provides a default implementation.

Q7. When will you use an interface?

A7. For polymorphic interface inheritance, where the client wants to only deal with a type and does not care about the actual implementation, then use interfaces. If you need to change your design frequently, you should prefer using interface to abstract class. Coding to an interface reduces coupling. Another justification for using interfaces is that they solve the '**diamond problem**' of traditional multiple inheritance as shown in the diagram. Java does not support multiple inheritance. Java only supports multiple interface inheritance. Interface will solve all the ambiguities caused by this 'diamond problem'. Java 8 has introduced functional interfaces, and partially solves the diamond problem by allowing default and static method implementations in interfaces to inherit multiple behaviors. **Strategy design pattern** lets you swap new algorithms and processes into your program without altering the objects that use them.



Q8. What is a marker or a tag interface? Why there are some interfaces with no defined methods (i.e. marker interfaces) in Java?

A8. The interfaces with no defined methods act like markers. They just tell the compiler that the objects of the classes implementing the interfaces with no defined methods need to be treated differently. For example, java.io.Serializable, java.lang.Cloneable, java.util.EventListener, etc. Marker

Searching algori

Swapping, partiti

Annotations (2)

8 Java Annotati

More Java annot

Collection and Data

Find the first n

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

interfaces are also known as “tag” interfaces since they tag all the derived classes into a category based on their purpose.

Now with the introduction of **annotations** in Java 5, the marker interfaces make less sense from a design standpoint. Everything that can be done with marker or tag interfaces in earlier versions of Java can now be done with **annotations at runtime using reflection**. One of the common problems with the marker or tag interfaces like Serializable, Cloneable, etc is that when a class implements them, all of its subclasses inherit them as well whether you want them to or not. You cannot force your subclasses to un-implement an interface. Annotations can have parameters of various kinds, and they’re much more flexible than the marker interfaces. This makes tag or marker interfaces obsolete, except for situations in which empty or tag interfaces can be checked at compile-time using the type-system in the compiler

Q9. What is a functional interface?

A9. Functional interfaces are introduced in Java 8 to allow default and static method implementations to enable functional programming (aka closures) with lambda expressions.

```

1
2  @FunctionalInterface
3  public interface Summable {
4      abstract int sum(int input1, int input2);
5  }
6

```

Q10. What does the @FunctionalInterface do, and how is it different from the @Override annotation?

A10. The @Override annotation takes advantage of the compiler checking to make sure you actually are overriding a method when you think you are. This way, if you make a common mistake of misspelling a method name or not correctly matching the parameters, you will be warned that your method does not actually override as you think it does. Secondly, it makes your code easier to understand because it is more obvious when methods are overwritten.

[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 MBe](#)

[Reactive Programn](#)

[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 annotation `@FunctionalInterface` acts similarly to `@Override` by signaling to the compiler that the interface is intended to be a functional interface. The compiler will then throw an error if the interface has multiple abstract methods.

Q10. Does Java 8 solve the “diamond problem” discussed earlier? If yes how?

A10. Partially yes.

We know that Java does not support multiple implementation inheritance to solve the diamond problem (till Java 8). Java did only support multiple interface inheritance. That is, a class can implement multiple interfaces. By having default method implementations in interfaces, you can now have multiple behavioral inheritance in Java 8. Partially solving the diamond problem.

Q11. Does Java 8 solve the need to have a separate helper classes? For example, you have

- 1) *java.util. Collection* interface and a separate *java.util.Collections* helper class with static methods.
- 2) *java.nio.file.Path* interface and a separate *java.util.Paths* helper or utility class with static methods.

A11 Yes. In Java 8, interfaces can have static helper methods. Here is an example of a Java 8 interface with both default and static methods.

```

1
2 package com.java8.examples;
3
4 import java.util.function.BinaryOperator;
5 import java.util.function.Function;
6 import java.util.Objects;
7
8 @FunctionalInterface
9 public interface Operation<Integer> {
10
11     //SAM -- Single Abstract Method.
12     //identifier abstract is optional
13     Integer operate(Integer operand);
14
15     default Operation<Integer> add(Integer o){
16         r return (o1) -> operate(o1) + o;
17     }

```

[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 AM](#)

[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](#)

```

18
19     default Operation<Integer> multiply(Integer
20         return (o1) -> operate(o1) * o;
21     }
22
23     //define other default methods for divide, s
24     default Integer getResult() {
25         return operate(0);
26     }
27
28     default void print(){
29         System.out.println("result is = " + getRes
30     }
31
32
33     //helper -- adds 5 to a given number
34     static Integer plus5(Integer input) {
35         return input + 5 ;
36     }
37 }
38

```

Now, how to invoke the default and static methods via
Lambda expressions:

```

1
2 package com.java8.examples;
3
4 import static java.lang.System.out;
5
6 public class OperationTest {
7
8     public static void main(String[] args) {
9
10        //plus5 is an expressive static helper method
11        Operation<Integer> calc = (x) -> Operation.plu
12
13        Operation complexOp = calc.add(3)
14            .multiply(4)
15            .multiply(2)
16            .multiply(2)
17            .add(4);
18
19        complexOp.print();
20
21        int result = complexOp.getResult();
22
23    }
24 }
25

```

So, if you are coding in Java 8, a separate helper class with static methods may no longer be required.

Q12. Does this mean that abstract classes may not be required as you can have default and static methods in Java 8 interfaces?

01: ♦ 15 Ice breas
02: ♥♦ 8 real life
03: ♦10+ Know y
FAQ Core Java Jot
♥♦ 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

A12 No. You still can't have states in Java 8 interfaces. You can only have behavior. So, Java 8 interfaces only gives your **multiple behaviour implementation inheritance**. You still need abstract classes to maintain default states via instance or member variables.

Popular Posts

♦ 11 Spring boot interview questions & answers

857 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

401 views

♦ 7 Java debugging interview questions & answers

311 views

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

302 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

**Arulkumar
Kumaraswamipillai**

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 debu

♦ 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



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.

◀ ♦ Java class loading interview Q&A to ascertain your depth of Java knowledge

♦ 12 Java Generics interview Q&A ▶

Posted in Classes, member-paid

Tags: Core Java FAQs, Java/JEE FAQs, Novice FAQs

Leave a Reply

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

Comment

- 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 Hadc
 - 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

Post Comment

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

	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 u

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 \(1\)](#)
- [Low Latency \(7\)](#)
- [Memory Management \(1\)](#)
- [Performance \(13\)](#)
- [QoS \(8\)](#)
- [Scalability \(4\)](#)
- [SDLC \(6\)](#)
- [Security \(13\)](#)
- [Transaction Management \(1\)](#)

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 Tutorial \(1\)](#)
- [JEE Tutorials \(19\)](#)
- [Scala Tutorials \(1\)](#)
- [Spring & Hibernate Tutorial \(1\)](#)
- [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.