

# 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 abstract classes

Vs interfaces

## ♦ Java abstract classes Vs interfaces

Posted on [February 2, 2015](#) by [Arulkumaran Kumaraswamipillai](#) — No

[Comments](#) ↓

11  
Like  
Share

Tweet

2  
G+1

1

Share

**Q1.** When to use an abstract class over an interface?

**Q2.** What is a diamond problem?

**Q3.** Does Java support multiple inheritance?

In design, you want the base class to present only an interface (or a contract) for its derived classes. This means, you don't want anyone to actually instantiate an object of the base class. You only want to **up cast** to it (implicit up casting, 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

[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

make an object of an abstract class, the compiler prevents it with a compile-time error.

**Q.** What is the purpose of an abstract class?

**A.** The purpose of abstract classes is to function as base classes which can be extended by sub classes to create a full implementation. The base class is used for code reuse and gives **polymorphism** by up casting to it.

**Q.** Should it have at least one abstract member?

**A.** It is subjective, but it is preferred. If your intention is to prevent a class from being instantiated, then the best way to handle this is with a private or protected constructor, and not by marking it abstract.

**Q.** Can you give an example of an abstract class where it is used prevalently?

**A. Template Method design pattern** is a good example of using an abstract class and this pattern is used very prevalently in application frameworks. The **Template Method** design pattern is about providing partial implementations in the abstract base classes, and the subclasses can complete when extending the Template Method base class(es). Here is an example

```

1  //cannot be instantiated
2  public abstract class BaseTemplate {
3
4      public void process() {
5          fillHead();
6          //some default logic
7          fillBody();
8          //some default logic
9          fillFooter();
10     }
11
12     //to be overridden by sub class
13     public abstract void fillBody();
14
15     //template method. Sub classes can override or
16     public void fillHead() {
17         System.out.println("Simple header");
18     }
19
20     //template method. Sub classes can override o
21     public void fillFooter() {
22         System.out.println("Simple footer");
23     }
24

```

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 loa

♦ 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

```

25 //more template methods can be defined here
26 }
27

```

```

1 public class InvoiceLetterProcessor extends Base
2
3 @Override
4 public void fillBody() {
5     System.out.println("Invoice body" );
6 }
7
8 // template method
9 public void fillHead() {
10     System.out.println("Invoice header");
11 }
12 }
13

```

```

1 public class InvoiceTestMain {
2
3     public static void main(String[] args) {
4         //subclass is up cast to base class -- p
5         BaseTemplate template = new Invoicelette
6         template.process();
7     }
8
9 }
10
11

```

Another design pattern that makes use of abstract classes is the **composite design pattern**. A **node** or a **component** is the parent or base class and derivatives can either be leaves (singular), or collections of other nodes, which in turn can contain leaves or collection-nodes. When an operation is performed on the parent, that operation is recursively passed down the hierarchy. An interface can be used instead of an abstract class, but an abstract class can provide some default behavior for the *add()*, *remove()* and *getChild()* methods.

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)

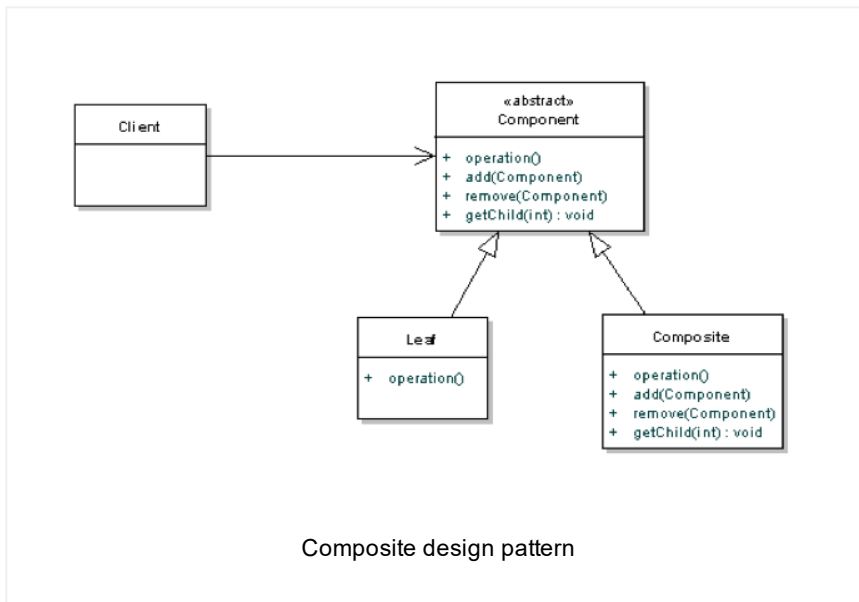
♥ Reading a text  
♦ 15 Java old I/C  
06: ♥ Java 8 way  
Processing large  
Processing large  
Read a text file f  
Reloading config

#### 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



**Java 8:** If you are using Java 8 or later versions of Java, you can have **default methods** and static helper methods in Java 8 interface definition.

**Q.** What is the purpose of an **interface**?

**A.** The interface keyword takes this concept of an abstract class a step further where till Java 7, you can't have implementations in an interface, but from **Java 8** onwards, the concept of **functional interfaces** was introduced where you can implement **default methods** and **static helper methods**.

**Q.** What are the differences between abstract classes and interfaces?

**A.**

Before Java 8:

Abstract class	Interface
Can maintain state in instance and static variables.	No state.

Searching algorithm

Swapping, partitioning

Annotations (2)

8 Java Annotations

More Java annotations

Collection and Data Structures

Find the first non-repeating character in a string

Java Collections Framework

Java Iterable & Iterator

HashMap & HashSet

Sorting objects

Java 8 Streams

Understanding Java 8 Streams

Java Collections Framework

If Java did not have Collections Framework

Java 8: Different ways to iterate over a collection

Part-3: Java Tree Traversal

Sorting a Map by value

When to use which Collection

Differences Between Abstract Class and Interface

Java Iterable & Iterator

Multithreading

Why do Proxy, Decorator, Adapter, Strategy, Singleton, Factory, Builder, etc. exist?

Core Java Modifications in Java 8

Differences between Abstract Class and Interface

Java Collections Framework

Event Driven Programming

Event Driven Programming

Event Driven Programming

Exceptions (2)

Java exception handling

Top 5 Core Java Interview Questions

Java 7 (2)

Java 7 fork and join

Java 7: Top 8 Interview Questions

Java 8 (24)

19 Java 8 Interview Questions

Java 8 Streams

Functional Programming

Top 6 tips for Java 8

Convert Lists to Arrays

Have executable methods and abstract methods.	Have no implementation code. All methods are abstract.
Can only extend one abstract class.	A class can implement any number of interfaces.
public ClassB extends ClassA {}	public ClassB implements InterfaceA, InterfaceB {}

### Java 8 onwards:

Abstract class	Interface
Can maintain state in instance and static variables.	No state.
Have executable methods and abstract methods.	Have executable default methods and static helper methods.
Can only subclass one abstract class.	A class can implement any number of interfaces.

So, from Java 8 onwards, the key difference is only one class can extend an abstract class, but a class can implement more than one interfaces.

**Q.** What is the diamond problem?

**A.** The diamond problem is that in the **multiple-inheritance diagram** on the left hand side below where *CircleOnSquare* inherits **state** and **behavior** from both *Circle* and *Square*. So, when we instantiate an object of class *CircleOnSquare*, any calls to method definitions in class *Shape* will be ambiguous –

[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 s](#)

[02: jvisualvm to c](#)

[05: Java primitiv](#)

[06: ♦ 10+ Atomic](#)

[5 JMX and MBe](#)

[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](#)

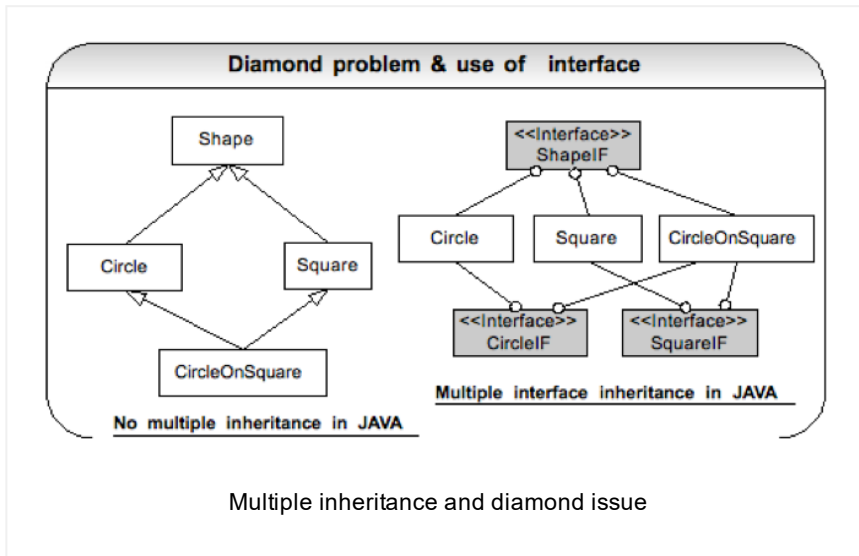
[Web basics \(8\)](#)

[01: ♦ 12 Web ba](#)

[02: HTTP basics](#)

[03: Servlet inter](#)

because it's not sure whether to call the version of the method derived from class *Circle* or class *Square*.



Java does not have **multiple inheritance**, as you can only extend one class. This means that Java is not at risk of suffering the consequences of the diamond problem. But, Java does support **multiple interface inheritance** as shown above on the right hand side of the diagram as a Java class can implement multiple interfaces. Hang on!!!! From **Java 8** onwards, you can have functional interfaces where you can implement default and static methods. This means, Java supports **multiple behavior inheritance**, but not full multiple inheritance as state cannot be inherited because you can't define instance variables in an interface.

Before Java 8 interface example:

```
1 public interface Summable {
2     abstract int sum(int input1, int input2);
3 }
4
```

After Java 8 interface example with default and static helper methods:

```
1 @FunctionalInterface
2 public interface Operation<Integer> {
3
4     Integer operate(Integer operand);
5 }
```

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



```

5
6     default Operation<Integer> add(Integer o){
7         return (o1) -> operate(o1) + o;
8     }
9
10    default Operation<Integer> multiply(Integer
11        return (o1) -> operate(o1) * o;
12    }
13
14    //ads 5 to a given number
15    static Integer plus5(Integer input) {
16        return input + 5 ;
17    }
18
19 }
20

```

Now, if we have a class **Calculator**, which implements two functional interfaces *Operation* and *Sum*, and both has a default method *add(Integer o)*. How does it know which default method to use? The one from *Operation* or the one from *Sum*. The *Calculator* class must solve the **multiple behavior inheritance ambiguity** by throwing a compile-time error

java: class Impl inherits unrelated defaults for *add(Integer o)* from types *Operation* and *Sum*.

In order to fix this class, the **Calculator** class needs to implement the *add(Integer o)* method to resolve the ambiguity.

**Q.** When to use an abstract class over an interface?

**A.**

### Abstract classes

With the advent of Default Methods in Java 8, it seems that Interfaces and abstract Classes are same as you can implement behavior in both. However, they are still different concepts as

- An Abstract Class can **define constructor(s)**.
- Abstract classes are **more structured and can have a state associated with them**. While in contrast,

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

default method can be implemented only in the terms of invoking other Interface methods, with no reference to a particular implementation's state.

Hence, both are used for different purposes and choosing between two really depends on the scenario. Abstract methods are good for implementing template method and composite design patterns with state and behavior.

## Interfaces

If you need to change your design frequently, you should prefer using interfaces to abstract classes. Coding to an interface reduces coupling and interface inheritance can achieve code reuse with the help of object composition. For example: The Spring frameworks' dependency injection promotes code to an interface principle. Another justification for using interfaces is that they solve the 'diamond problem' of traditional multiple inheritance. Java does not support multiple inheritance, but supports multiple behavior inheritance.

**Strategy design pattern** lets you swap new algorithms and processes into your program without altering the objects that use them by making use of interfaces.

**Q.** What are the different ways to get **code reuse**?

**A.** There are 3 approaches

1. **Implementation inheritance** with **abstract** classes.
2. **Composition**.
3. **Delegation** to a helper class.

Implementation inheritance gives you **polymorphism** in addition to code reuse. You can up cast your child class to your abstract base class. If you are using composition for code reuse, then you need to use **behavior inheritance** with interfaces to get **polymorphism** (i.e. code to interface).

The **GoF** design patterns favor **composition** for code reuse with **polymorphism** with **interfaces** over **implementation**

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



**inheritance** with **abstract classes** for **code reuse** and **polymorphism**.

## Why?

1. Code reuse via composition happens at run time whereas code reuse via inheritance happens at compile time. Hence, composition is more flexible and less fragile.

2. It is easy to misuse inheritance when there is really no “is a” relationship, hence breaking the Lithkov’s Substitution Principle (**LSP**). For example, a square is not a rectangle. Overuse of implementation inheritance (uses the “extends” key word) can break all the sub classes, if the super class is modified. 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.

3. Composition offers better testability than Inheritance. Composition is easier to test because inheritance tends to create very coupled classes that are more fragile (i.e. fragile parent class) and harder to test in isolation. The IoC containers like Spring, make testing even easier through injecting the composed objects via constructor or setter injection.

[Why favor composition over inheritance?](#) detailed discussion.

**Q.** What is the difference between the default methods introduced in Java 8 and regular methods?

**A.** In summary, **default methods** are like regular methods, but

- the default methods come with the default modifier.
- the default methods can only access its arguments as Interfaces do not have any state.

Regular methods in Classes can use and modify method arguments as well as the variables (i.e state) of their *Class*.

	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 HB	
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	

The default (aka defender) methods allow you to add new methods to interfaces without breaking the existing implementations of your interfaces. This provides an added flexibility by allowing interfaces to provide default implementations in situations where concrete classes fail to provide implementations for a methods.

## 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

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

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

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 JavaConf

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



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

◀ Part 4: Java Tree structure interview and coding questions

Dealing with concurrent modifications in Java ▶

Posted in [Classes](#)

## 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 in

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

Post Comment

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