

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](#) › 03: ♦ 9 Core Java

Concepts you can't afford to not know

03: ♦ 9 Core Java Concepts you can't afford to not know

Posted on [September 9, 2014](#) by [Arulkumaran Kumaraswamipillai](#) — No

[Comments](#) ↓

0

Like

Share

Tweet

0

G+1

Share

Q1. How will you go about explaining the following Java concepts to a beginner who is starting to learn Java?

1. Process Vs Threads
2. Heap versus Stack
3. Local variables versus instance variables
4. How do threads communicate with each other?
5. Are Java methods reentrant?
6. Does Java support recursive method calls?
7. Object creation and Garbage collection
8. Can you garbage collect objects that have a circular

[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

reference?

9. Producer-consumer design pattern.

A1. The best way to do this is to write some basic code and then have a pictorial representation as to how the objects are created, how do threads work and communicate with each other, and what is stored in a heap and what is stored in a stack, etc.

Here is an example of a producer thread (thread-0) producing by incrementing the counter from 0, and the consumer thread (i.e. thread-1) consumes by decrementing the counter. These two user created threads are spawned by the main thread, which is created by the JVM and is always there by default. The **ConsumerProducer** is the shared object with synchronized methods that communicate with each other via `wait()` and `notifyAll()` methods. Only one of the two synchronized methods can be executed at a time. The `wait()` call in `consume()` relinquishes the lock to the `produce()` method, and once the `produce` method has incremented the count, it notifies all threads and one of the waiting threads will resume. In this example, there is only one waiting consumer (i.e. Thread-1) thread. So, both threads will be communicating with each other via the `wait()` and `notifyAll()` calls in the shared object **ConsumerProducer**. This is an example of the producer-consumer design pattern.

Firstly, look at the code and then the diagram. The diagram is simplified to get an understanding and should not be construed as exactly what happens in the JVM.

```

1 public class ConsumerProducer {
2
3     private int count;
4
5     public synchronized void consume() {
6         while (count == 0) { // keep waiting if no
7             try {
8                 wait(); // give up lock and wait
9             } catch (InterruptedException e) {
10                // keep trying
11            }
12        }
13
14        count--; // consume
15        System.out.println(Thread.currentThread().g

```

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

```

16     }
17
18     public synchronized void produce() {
19         count++;           //produce
20         System.out.println(Thread.currentThread().getName() + " produced " + count);
21         notifyAll();       // notify waiting threads
22     }
23
24 }

```

The main thread spawns a consumer and a producer thread. The ConsumerProducer is shared between two threads. The boolean flag is used to signal if it is a consumer thread or a producer thread to invoke the relevant methods.

```

1  public class ConsumerProducerTest implements Runnable {
2
3      boolean isConsumer;
4      ConsumerProducer cp;
5
6      public ConsumerProducerTest(boolean isConsumer, ConsumerProducer cp) {
7          this.isConsumer = isConsumer;
8          this.cp = cp;
9      }
10
11     public static void main(String[] args) {
12         ConsumerProducer cp = new ConsumerProducer();
13
14         Thread producer = new Thread(new ConsumerProducerTest(false, cp));
15         Thread consumer = new Thread(new ConsumerProducerTest(true, cp));
16
17         producer.start();
18         consumer.start();
19     }
20
21     @Override
22     public void run() {
23         for (int i = 1; i <= 10; i++) {
24             if (!isConsumer) {
25                 cp.produce();
26             } else {
27                 cp.consume();
28             }
29         }
30
31         //try with introducing a sleep for 100ms.
32     }
33 }
34

```

The output will vary, but the last thing consumed will be 0.

```

1 Thread-0 after producing 1
2 Thread-0 after producing 2
3 Thread-0 after producing 3
4 Thread-0 after producing 4
5 Thread-0 after producing 5

```

03: Java GC tuning

Generics (5)

♥ Java Generics
♥ Overloaded methods
♦ 12 Java Generics
♦ 7 rules to remember
3 scenarios to get

FP (8)

01: ♦ 19 Java 8 FP
02: ♦ Java 8 Stream
03: ♦ Functional
04: ♥♦ Top 6 tips
05: ♥ 7 Java FP
Fibonacci number
Java 8 String stream
Java 8: What is a

IO (7)

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

Multithreading (12)

01: ♥♦ 15 Beginner
02: ♥♦ 10+ Java
03: ♦ More Java
04: ♦ 6 popular J
05: ♦ How a thread
06: ♦ 10+ Atomic
07: 5 Basic multi
08: ♦ ThreadLocal
09: Java FutureTask
10: ♦ ExecutorService
Java ExecutorService
Producer and Consumer

Algorithms (5)

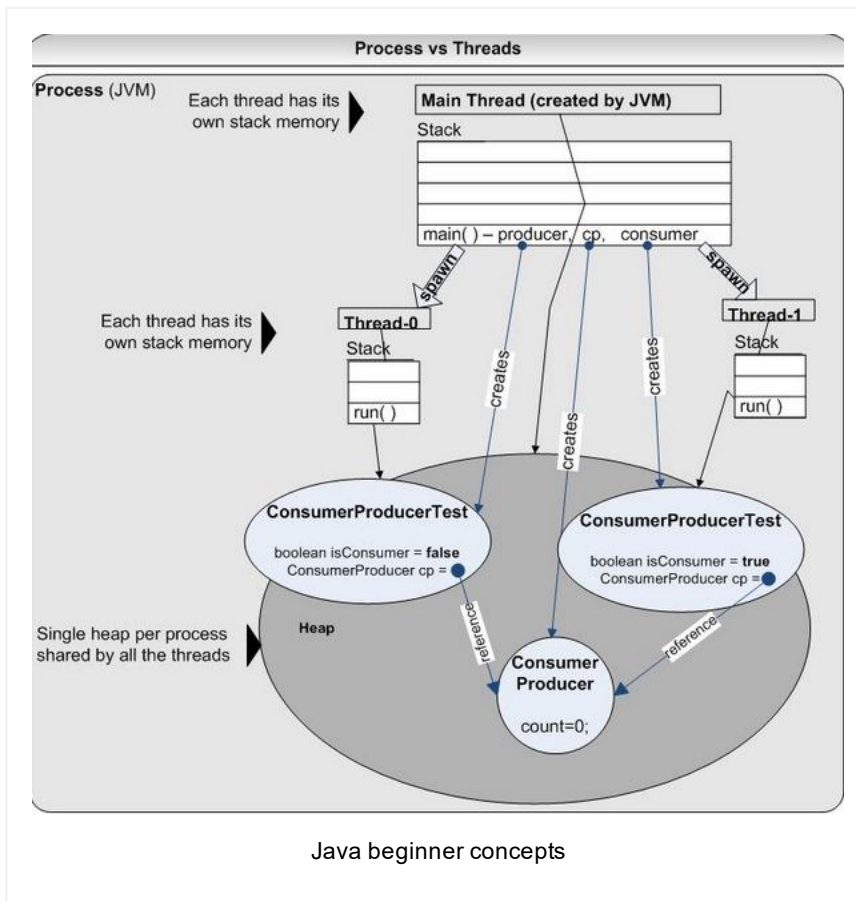
♦ Splitting input
♦ Tree traversal
♥ ♦ Java coding

```

6 Thread-0 after producing 6
7 Thread-0 after producing 7
8 Thread-0 after producing 8
9 Thread-0 after producing 9
10 Thread-0 after producing 10
11 Thread-1 after consuming 9
12 Thread-1 after consuming 8
13 Thread-1 after consuming 7
14 Thread-1 after consuming 6
15 Thread-1 after consuming 5
16 Thread-1 after consuming 4
17 Thread-1 after consuming 3
18 Thread-1 after consuming 2
19 Thread-1 after consuming 1
20 Thread-1 after consuming 0
21

```

The above code can be diagrammatically represented as shown below.



- The JVM is a process. As you could see in the diagram above that **each thread has its own stack**, but **share the same heap space**. One heap space per process.
- The **local variables** like producer, cp, and consumer are **stored in the stack** along with the method calls

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

Java Iterable

HashMap & HashSet

Sorting objects

02: Java 8 Streams

04: Understanding Java Collections

If Java did not have Collections

Java 8: Different ways to sort a Map by value

When to use which Collection

Differences Between Java Collections

Java Iterable

Multithreading

Why do Proxy, Core Java Modifications

Differences between Java Collections

Event Driven Programming

Event Driven Programming

Event Driven Programming

Exceptions (2)

Java exception handling

Top 5 Core Java Concepts

Java 7 (2)

Java 7 fork and join

Java 7: Top 8 new features

Java 8 (24)

01: 19 Java 8 Interview Questions

02: Java 8 Streams

03: Functional Programming

04: Top 6 tips for Java 8

04: Convert Lists to Arrays

like `main()`, `run()`, etc.

- The **objects are created in the heap**. For example, 2 instances of *ConsumerProducerTest* and a single instance of *ConsumerProducer*. Unless the methods of these instances are properly managed, multiple threads can concurrently access these instances to cause thread-safety issues.
- There are scenarios where the threads want to communicate with each other. For example, a worker thread produces something and a consumer thread consumes what the producer thread consumed. This can be achieved through thread coordination with the `wait` and `notify()` / `notifyAll()` methods. In the above example, since both the `produce()` and `consume()` methods of the object *ConsumerProducer* are synchronized, only one thread can execute either one of the methods. If the `count == 0`, the consumer thread relinquishes the lock by invoking the `wait()` method and waits for it to be notified again to resume. The producer thread will increment the counter and then will notify all the waiting threads via the `notifyAll()` method. One of the waiting threads will then be able to resume from its waiting state.
- The local variable **isConsumer** is used to differentiate between a consumer thread and a producer thread.
- The **count** is an instance variable shared by both the consumer and producer thread in a thread-safe manner.

Here are the points that answers the above questions.

- Each time an object is created in Java it goes into the area of memory known as heap.
- The primitive variables like `int`, `double`, etc are allocated in the stack if they are local variables and in

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 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 heap if they are instance variables (i.e. fields of a class).

- Java is a stack based language and local variables are pushed into stack when a method is invoked and stack pointer is decremented when a method call is completed.
- In a multi-threaded application, each thread will have its own stack but will share the same heap. This is why care should be taken in your code to avoid any concurrent access issues in the heap space.
- The stack is thread-safe because each thread will have its own stack with say 1MB RAM allocated for each thread but the heap is not thread-safe unless guarded with synchronization through your code. The stack space can be increased with the `-Xss` option.
- All Java methods are automatically **re-entrant**. It means that several threads can be executing the same method at once, each with its own copy of the local variables.
- A Java method may call itself without needing any special declarations. This is known as a **recursive** method call. Given enough stack space, recursive method calls are perfectly valid in Java though it is tough to debug. Recursive methods are useful in removing iterations from many sorts of algorithms. All recursive functions are re-entrant but not all re-entrant functions are recursive.
- **Idempotent** methods are methods, which are written in such a way that repeated calls to the same method with the same arguments yield same results.
- Each time an object is created in Java, it goes into the area of memory known as heap. The Java heap is called the **garbage collectable heap**. The **garbage collection cannot be forced**. The garbage collector runs in low memory situations. When it runs, it releases the memory allocated by an unreachable object. The garbage collector runs on a low priority daemon (i.e. background thread). You can nicely ask the garbage collector to collect garbage by calling `System.gc()` but you can't force it.

[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](#)

- An object's life has no meaning unless something has reference to it. If you can't reach it then you can't ask it to do anything. Then the object becomes unreachable and the garbage collector will figure it out. Java automatically collects all the unreachable objects periodically and releases the memory consumed by those unreachable objects to be used by the future reachable objects.
- If two objects have **circular reference** with each other in the heap, but none of them are reachable from any thread, then those circular referenced objects can be garbage collected.

Q3. What will be the output of the following code snippet?

```

1 public class MethodOverrideVsOverload {
2
3     public boolean equals( MethodOverrideVsOverload o2) {
4         System.out.println("MethodOverrideVsOverload equals method reached");
5         return true;
6     }
7
8     public static void main(String[] args) {
9         Object o1 = new MethodOverrideVsOverload();
10        Object o2 = new MethodOverrideVsOverload();
11
12        MethodOverrideVsOverload o3 = new MethodOverrideVsOverload();
13        MethodOverrideVsOverload o4 = new MethodOverrideVsOverload();
14
15        if(o1.equals(o2)){
16            System.out.println("objects o1 and o2 are equal");
17        }
18
19        if(o3.equals(o4)){
20            System.out.println("objects o3 and o4 are equal");
21        }
22    }
23 }
24

```

A3. The output will be

```

1 MethodOverrideVsOverload equals method reached
2 objects o3 and o4 are equal
3

```

What concepts does this question try to test?

01: ♦ 15 Ice breakers
02: ♥♦ 8 real life
03: ♦10+ Know your Java
FAQ Core Java Jobs
♥♦ Q1-Q10: Top
♦ Q11-Q23: Top
♦ Q24-Q36: Top
♦ Q37-Q42: Top
♦ Q43-Q54: Top
01: ♥♦ 15 Beginner
02: ♥♦ 10+ Java
FAQ JEE Job Interviews
♦ 12 FAQ JDBC
♦ 16 FAQ JMS interview
♦ 8 Java EE (aka J2EE)
♦ Q01-Q28: Top
♦ Q29-Q53: Top
01: ♦ 12 Web basics
06: ♦ MVC0, MVC1, MVC2
JavaScript mistakes
JavaScript Vs Java
JNDI and LDAP
JSF interview Questions
JSON interview Questions
FAQ Java Web Services
01: ♥♦ 40+ Java
02: ♦ 6 Java REST
05: RESTful Web
06: RESTful Web
09: 11 SOAP Web
Java Application Architecture
001A: ♦ 7+ Java
001B: ♦ Java architecture
04: ♦ How to go
Hibernate Job Interviews
01: ♥♦ 15+ Hiber
01b: ♦ 15+ Hiber
06: Hibernate Fil
8 JPA interview c
Spring Job Interviews
♦ 11 Spring boot

- In Java, a class can only extend a single class (i.e. single inheritance), and when it does not explicitly extend a class, it **implicitly** extends the class **Object**. So, *MethodOverrideVsOverload* implicitly extends the class Object.
- The majority of the non final Object class methods are meant to be overridden by the sub classes.

```
1 public boolean equals(Object obj); // make
2 public int hashCode();
3 public String toString();
```

- The method overloading takes place at **compile time** (i.e. static binding) and method overriding takes place at **runtime** (i.e. dynamic binding). Static binding means the JVM decides, which class or method to call during compile time. Dynamic binding means, the JVM decides, which class or method to call during runtime. The polymorphism is possible because of dynamic binding.
- The method overriding must adhere to the following rules

Arguments	Must not change.
Return type	Can't change except for covariant (subtype) returns.
Exceptions	The extending class can eliminate or call fewer exceptions than its parent, but must not throw new or broader checked exceptions.
Access	Must not be more restrictive than the class it extends. Can be less restrictive.
Invocation	Which method to call is based on object type, at runtime time (i.e. dynamic binding).

[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](#)

Now, if you look at the above code

The “*equals(MethodOverrideVsOverload other)*” method in class *MethodOverrideVsOverload* **does not** actually override the *Object* class’s “*public boolean equals(Object obj)*” method. This is because it fails to adhere to the “Arguments” rule as both methods have different arguments as one is of type “*MethodOverrideVsOverload*” and the other of type “*Object*”. So, the two methods are **overloaded** (happens at compile time) and not **overridden**.

So, when **o1.equals(o2)** is invoked, the *public boolean equals(Object obj)* method of the object class is invoked because during compile time, the o1 and o2 are of type *Object*. The *Object* class’s *equals(...)* method returns false as it compares the memory address (e.g. *Object@235f56* and *Object@653af32*) of both objects.

When o3.equals(o4) is invoked, the “*equals(MethodOverrideVsOverload other)*” of the class *MethodOverrideVsOverload* is invoked as during compile time o3 and o4 are of type *MethodOverrideVsOverload*, hence you get the above output.

What follow on questions can you expect?

Q. How will you fix the above issue?

A. In Java 5, annotations were introduced and one of the handy **compile time annotations** is the `@override`, which will ensure that the methods are overridden correctly. If you had this annotation, when you override it incorrectly as in the above example, a compile time error will be thrown.

So, to fix it, add the `@override` annotation to the “*boolean equals(MethodOverrideVsOverload other)*” of the *MethodOverrideVsOverload* class. This will give you a compile time error indicating that the method is not properly overridden. Now, fix the method signature by changing the argument type in the method signature from

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 archi
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

“MethodOverrideVsOverload” to “Object” as shown below.

```

1 public class MethodOverrideVsOverload {
2
3     @Override
4     public boolean equals( Object other ) {
5         System.out.println("MethodOverrideVsOverl
6         return true;
7     }
8
9     public static void main(String[] args) {
10        Object o1 = new MethodOverrideVsOverload()
11
12        Object o2 = new MethodOverrideVsOverload()
13
14
15        MethodOverrideVsOverload o3 = new MethodOv
16
17        MethodOverrideVsOverload o4 = new MethodOv
18
19
20        if(o1.equals(o2)){
21            System.out.println("objects o1 and o2 ar
22        }
23
24        if(o3.equals(o4)){
25            System.out.println("objects o3 and o4 ar
26        }
27    }
28 }
29

```

The output will be

```

1 MethodOverrideVsOverload equals method reached
2 objects o1 and o2 are equal
3 MethodOverrideVsOverload equals method reached
4 objects o3 and o4 are equal
5

```

This is because now the methods are overridden and this happens at runtime. This is a bit tricky question, and think out loud at the interview to show that you understand the fundamentals.

Popular Posts

♦ 11 Spring boot interview questions & answers

852 views

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

♦ 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



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](https://www.amazon.com) in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.



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



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.

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

Recursion Vs Tail Recursion ▶

Posted in Java Overview, member-paid

Tags: Java/JEE FAQs, Novice FAQs

Leave a Reply

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

Comment

[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 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.