Java-Success.com

Industrial strength Java/JEE Career Companion to open more doors

01: ♥♦ 15 Beginner level Java multi-threading interview Q&A

Posted on August 22, 2014 by Arulkumaran Kumaraswamipillai

Why multithreading questions are very popular?

It is because NOT many developers have a good grasp on multithreading. Comprehensive list of Java multi-threading Q&A and basics

Q1. What is a thread?
A1. A thread is a thread of execution in a program. The JVM allows an application to have multiple threads of execution running concurrently. In the Hotspot JVM there is a direct mapping between a Java Thread and a native operating

system (i.e. OS) Thread. The native OS thread is created after preparing the state for a Java thread involving thread-local storage, allocation of buffers, creating the synchronization objects, stacks and the program counter. The OS is responsible for scheduling all threads and dispatching them to any available CPU. In a multi-core CPU, you will get real parallelism. When the thread terminates all resources for both the native and Java thread are released.

Q2. What are the JVM or system created threads?
A2. The **main thread** and a number of **background threads**.

1) **main thread**, which is created as part of invoking public static void main(String[])

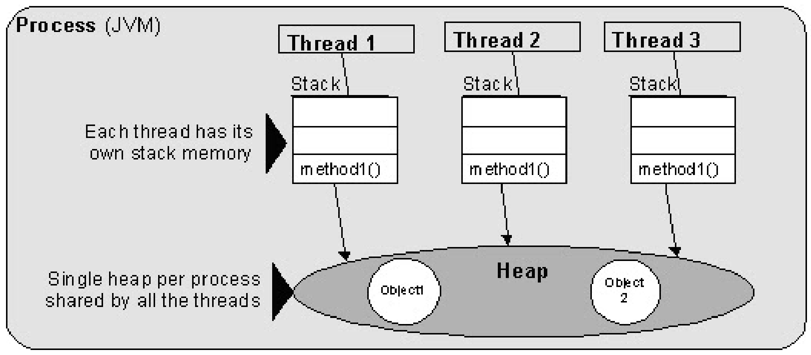
2) **VM background thread** to perform major GC, thread dumps, thread suspension, etc.

3) **Garbage Collection** low priority background thread for GC activities.

4) **Compiler** background thread to compile byte code to native code at run-time.

5) Other background threads such as signal dispatcher thread and periodic task thread.

Q3. What is the difference between processes and threads?
A3. A process is an execution of a program but a thread is a single execution sequence within the process. A process can contain multiple threads. A thread is sometimes called a lightweight process.



As a Java Architect

Java architecture & design concepts

Process Vs Threads

A JVM runs in a single process and threads in a JVM share the heap belonging to that process. That is why several threads may access the same object. Threads share the heap and have their own stack space. This is how one thread's invocation of a method and its local variables are kept thread safe from other threads. But the heap is not thread-safe and must be synchronized for thread safety.

Q4. Explain different ways of creating a thread?
A4. Threads can be created in a number of different ways

1) Extending the **java.lang.Thread** class.

2) Implementing the **java.lang.Runnable** interface.

3) Implementing the **java.util.concurrent.Callable** interface with the

java.util.concurrent.Executor framework to pool the threads.

4) Using the Fork/Join Pool. Java 7 fork and join tutorial with a diagram and an example.

5) The **actor model** using frameworks like **Akka**.

The java.util.concurrent package was added in Java 5.

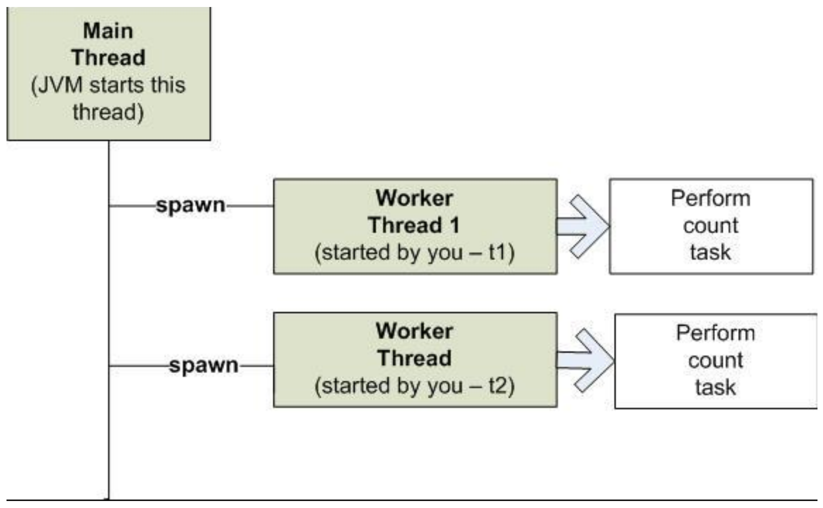
interview Q&As with diagrams | What should be a typical Java EE architecture?

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 I
⊞ Low Latency (7)
⊞ Memory Managemen
⊞ Performance (13)
⊞ QoS (8)
⊞ Scalability (4)
⊞ SDLC (6)
⊞ Security (13)
⊞ Transaction Managen

80+ step by step Java Tutorials

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


Main Thread (JVM starts this thread)
spawn
Worker Thread 1 (started by you – t1)
Perform count task
spawn
Worker Thread (started by you – t2)
Perform count task

Creating a Thread in Java

1. Extending the **java.lang.Thread** class

```
class Counter extends Thread {

    //method where the thread execution will sto
    public void run(){
        //logic to execute in a thread
        //e.g. performing a count task
    }

    //let's see how to start the threads
    public static void main(String[] args){
       Thread t1 = new Counter();
       Thread t2 = new Counter();
       t1.start();  //start the first thread. Th
       t2.start(); //this starts the 2nd thread.
    }
}

```

2. Implementing the **java.lang.Runnable** interface. The Thread class takes a runnable object as a constructor argument.

```
class Counter extends Base implements Runnable{

    //method where the thread execution will sto
    public void run(){
        //logic to execute in a thread
        //e.g. performing a count task
    }

    //let us see how to start the threads
    public static void main(String[] args){
         Thread t1 = new Thread(new Counter());
```

100+ Preparing for pre-interview Java written home assignments & coding tests

How good are your…to go places?

```
        Thread t2 = new Thread(new Counter());
        t1.start();  //start the first thread.
        t2.start();  //this starts the 2nd thre
    }
}

```

3. Implementing the **java.util.concurrent.Callable** interface.

```
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

class Counter implements Callable<String> {

    private static final int THREAD_POOL_SIZE =

    // method where the thread execution takes 
    public String call() {
        return Thread.currentThread().getName()
    }

    public static void main(String[] args) throw
            ExecutionException {
        // create a pool of 2 threads
        ExecutorService executor = Executors
                .newFixedThreadPool(THREAD_POOL_

        Future<String> future1 = executor.submi
        Future<String> future2 = executor.submi

        System.out.println(Thread.currentThread(

        //asynchronously get from the worker thr
        System.out.println(future1.get());
        System.out.println(future2.get());

    }
}

```

Q. Which approach would you favor and why?
A. Favor Callable interface with the Executor framework for thread pooling.

1) The thread pool is more efficient. Even though the threads are light-weighted than creating a process, creating them utilizes a lot of resources. Also, creating a new thread for each task will consume more stack memory as each thread will have its own stack and also the CPU will spend more time in context switching. Creating a lot many threads with no bounds to the maximum threshold can cause application to run out of heap memory. So, creating a Thread Pool is a better solution as a finite number of threads can be pooled and reused. The runnable or callable tasks will be placed in a queue, and the finite number of threads in the pool will take turns to process the tasks in the queue.

2) The Runnable or Callable interface is preferred over extending the Thread class, as it does not require your object to inherit a thread because when you need multiple inheritance, only interfaces can help you. Java class can extend only one class, but can implement many interfaces.

3. The Runnable interface's void run( ) method has no way of returning any result back to the main thread. The executor framework introduced the Callable interface that returns a value from its call( ) method. This means the asynchronous task will be able to return a value once it is done executing.

Q.What design pattern does the executor framework use?
A. The java.util.concurrent.**Executor** is based on the **producer-consumer design pattern**, where threads that submit tasks are producers and the threads that execute tasks are consumers. In the above examples, the main thread is the producer as it loops through and submits tasks to the worker threads. The "Counter" is the consumer that executes the tasks submitted by the main thread.

Q5. What is the difference between yield and sleep? What is the difference between the methods sleep( ) and wait( )?
A5. When a task invokes yield( ), it changes from running state to runnable state. When a task invokes sleep ( ), it changes from running state to waiting/sleeping state.

The method wait(1000) causes the current thread to wait up to one second a signal from other threads. A thread could wait less than 1 second if it receives the notify( ) or notifyAll( ) method call. The call to sleep(1000) causes the current thread to sleep for t least 1 second.

Q6. Why is locking of a method or block of code for thread safety is called "**synchronized**" and not "lock" or "locked"?

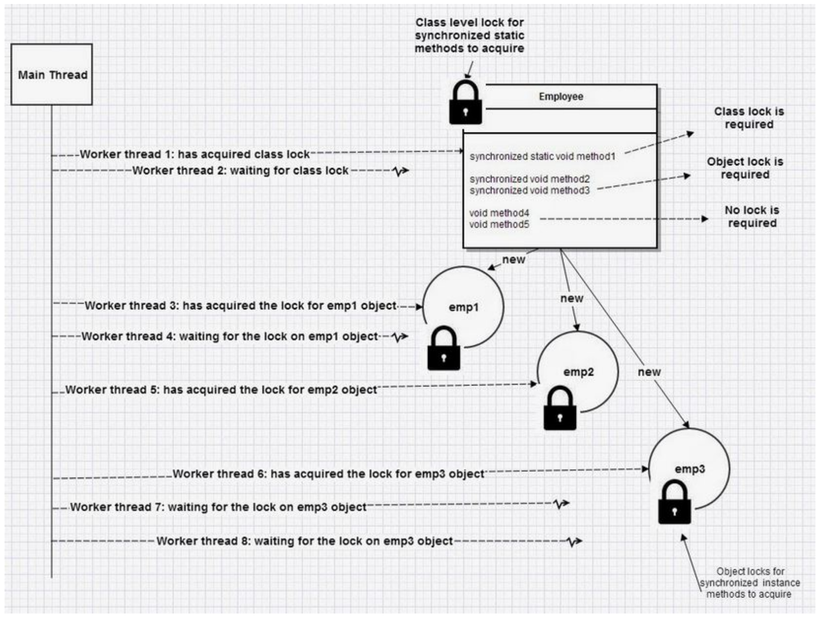
A6. When a method or block of code is locked with the reserved "synchronized" key word in Java, the memory (i.e. heap) where the shared data is kept is synchronized. This means,

When a synchronized block or method is entered after the lock has been acquired by a thread, it first reads any changes to the locked object from the main heap memory to ensure that the thread that has the lock has the current info before start executing.
After the synchronized block has completed and the thread is ready to relinquish the lock, all the changes that were made to the object that was locked is written or flushed back to the main heap memory so that the other threads that acquire the lock next has the current info.

This is why it is called "synchronized" and not "locked". This is also the reason why the immutable objects are inherently thread-safe and does not require any synchronization. Once created, the immutable objects cannot be modified.

Learn more about the memory model & the synchronization at: 10+ Atomicity, Visibility, and Ordering interview Q&A in Java multi-threading

Q7. Can you explain what an intrinsic lock or monitor is?
A7. Here are 7 things you must know about Java locks and synchronized key word. The diagram below depicts the interactions beyween Java threads and 3 instances (i.e. emp1, emp2, emp3 objects) of an "**Employee**" class. Access to synchronized blocks or methods require acquisition of a lock. Every Java class and instance of a class has an intrinsic lock.


Class level lock for synchronized static methods to acquire
Main Thread
Employee
synchronized static void method1
synchronized void method2
synchronized void method3
void method4
void method5
Class lock is required
Object lock is required
No lock is required
Worker thread 1: has acquired class lock
Worker thread 2: waiting for class lock
new
new
new
emp1
emp2
emp3
Worker thread 3: has acquired the lock for emp1 object
Worker thread 4: waiting for the lock on emp1 object
Worker thread 5: has acquired the lock for emp2 object
Worker thread 6: has acquired the lock for emp3 object
Worker thread 7: waiting for the lock on emp3 object
Worker thread 8: waiting for the lock on emp3 object
Object locks for synchronized instance methods to acquire

Java locks – object level and class level

1) Each Java class and object (i.e. instance of a class) has an **intrinsic lock** or **monitor**. Don't confuse this with **explicit lock** utility classes that were added in Java 1.5, and I will discuss this later.

2) If a method is declared as synchronized, then it will acquire either the **instance intrinsic lock** or the **static intrinsic lock** when it is invoked. The two types of lock have similar behavior, but are completely independent of each other.

3) Acquiring the instance lock only blocks other threads from invoking a synchronized instance method. It does not block other threads from invoking an un-synchronized method, nor does it block them from invoking a static synchronized method.

4) Any thread entering a synchronized method or a block of code needs to acquire that object's or class's lock before entering to execute that method or block of code.

5) Acquired lock is released when leaving a synchronized method or a block of code for other waiting or blocked threads to acquire.

6) When an object has 1 or more synchronized methods or blocks of code, only one thread can acquire the lock for that object, all other threads will be blocked, and will be waiting to acquire the lock once released.

7) When an object has 1 or more methods that are not synchronized, one or more threads can execute those methods or blocks of code simultaneously or concurrently. Threads are Not blocked, and waiting is not required.

Q8. What does reentrancy mean regarding intrinsic or explcit locks?
A8. Reentrancy means that locks are acquired on a per-thread rather than per-invocation basis. In Java, both intrinsic and explicit locks are re-entrant.

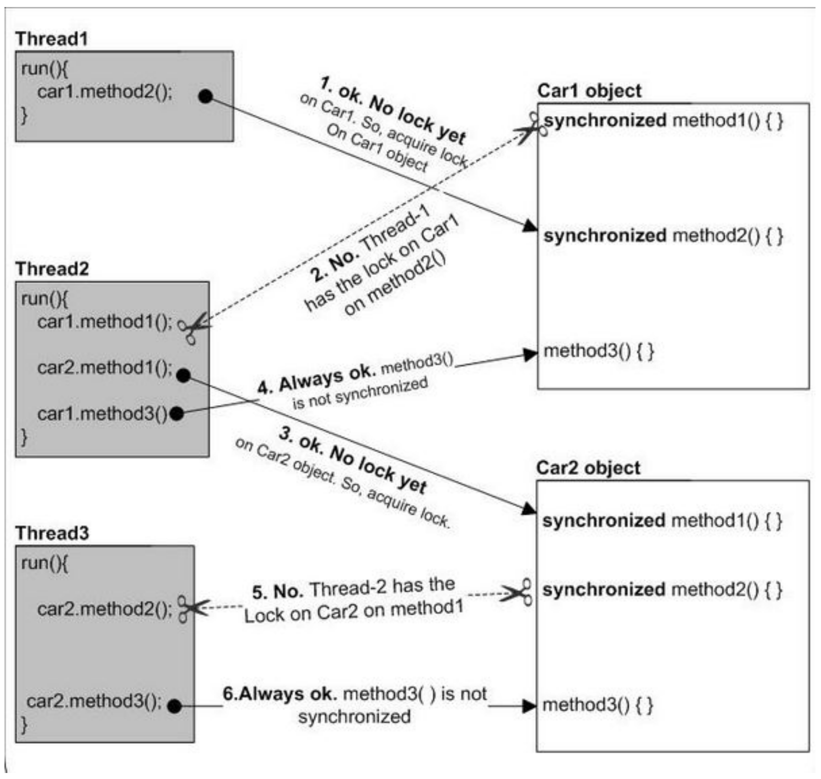
```
public synchronized void method1(){
  //intrinsic lock is acquired
  operation1(); //ok to enter this synchronized
                //as locks are on per thread bas
  operation2(); //ok to enter this synchronized
                //as locks are on per thread bas
  //intrinsic lock is released
}

public synchronized void operation1(){
    //process 1
}

public synchronized void operation2(){
    //process 1
}

```

Q9. If 2 different threads hit 2 different synchronized methods in an object at the same time will they both continue?
A9. No. Only one thread can acquire the lock in a synchronized method of an object. Each object has a synchronization lock. No 2 synchronized methods within an object can run at the same time. One synchronized method should wait for the other synchronized method to release the lock. This is demonstrated here with method level lock. Same concept is applicable for block level locks as well.


Thread1
run(){
car1.method2();
}
1. ok. No lock yet on Car1. So, acquire lock On Car1 object
Car1 object
synchronized method1() { }
synchronized method2() { }
method3() { }
Thread2
run(){
car1.method1();
car2.method1();
car1.method3()
}
2. No. Thread-1 has the lock on Car1 on method2()
4. Always ok. method3() is not synchronized
3. ok. No lock yet on Car2 object. So, acquire lock.
Car2 object
synchronized method1() { }
synchronized method2() { }
method3() { }
Thread3
run(){
car2.method2();
car2.method3();
}
5. No. Thread-2 has the Lock on Car2 on method1
6.Always ok. method3( ) is not synchronized

Java synchronization

Q10. Why synchronization is important?
A10. Without synchronization, it is possible for one thread to modify a shared object while another thread is in the process of using or updating that object’s value. This often causes dirty data and leads to significant errors.

Q11. What is the disadvantage of synchronization?
A11. The disadvantage of synchronization is that it can cause deadlocks when two threads are waiting on each other to do something. Also, synchronized code has the overhead of acquiring lock, and preventing concurrent access, which can adversely affect performance.

Q12. When every object has an intrinsic lock in Java, why were explicit lock utility classes introduced in Java 5?
A12. An intrinsic locking mechanism is a clean approach in terms of writing code, and is pretty good for most of the use-cases. But, intrinsic locking mechanism do have some limitations in certain scenarios:

— It is not possible to have more control, for example, read concurrently when not writing.
— Intrinsic locks must be released in the same block in which they are acquired.
— It is not possible to interrupt a thread waiting to acquire a lock.
— It is not possible to attempt to acquire a lock without waiting for it forever.

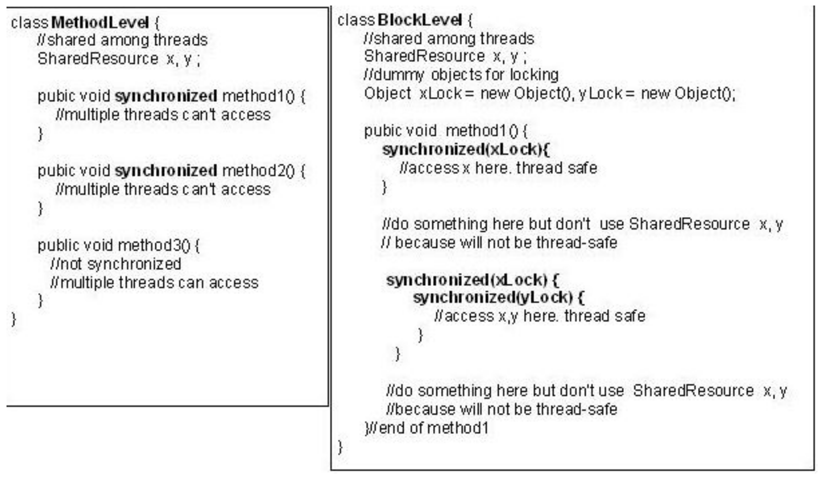
Q13. How are explicit locks laid out in Java?
A13. Laid out with 2 interfaces Lock and ReadWriteLock.

Q14. What are the disadvantages of explicit locks?
A14. It is more complicated to use it properly, and incorrect usage can lead to unexpected issues leading to deadlocks, thread starvation, etc. So, you need to remember the following best practices when using explicit locks.

— Release the explicit locks in a finally block.
— Favor intrinsic locks where possible to avoid bugs and to keep your code cleaner and easier to maintain.
— Use tryLock( ) if you don’t want a thread waiting indefinitely to acquire a lock. This is similar to how databases prevent dead locks with wait lock timeouts.
— When using ReentrantLocks for frequent concurrent reads and occasional writes, be mindful of the possibility that a writer could wait a very long time sometimes forever) if there are constantly read locks held by other threads.

Q15. How does thread synchronization occurs inside a monitor? What levels of synchronization can you apply? What is the difference between synchronized method and synchronized block?
A15. In Java programming, each object has a lock. A thread can acquire the lock for an object by using the synchronized keyword. The synchronized keyword can be applied in method level (coarse grained lock – can affect performance adversely) or block level of code (fine grained lock). Often using a lock on a method level is too coarse. Why lock up a piece of code that does not access any shared resources by locking up an entire method. Since each object has a lock,

dummy objects can be created to implement block level synchronization. The block level is more efficient because it does not lock the whole method.



```
class MethodLevel {
    //shared among threads
    SharedResource  x, y ;

    pubic void synchronized method1() {
       //multiple threads can't access
    }

    pubic void synchronized method2() {
       //multiple threads can't access
    }

    public void method3() {
      //not synchronized
      //multiple threads can access
    }
}
```

```
class BlockLevel {
    //shared among threads
    SharedResource  x, y ;
    //dummy objects for locking
    Object  xLock = new Object(), yLock = new Object();

    pubic void  method1() {
       synchronized(xLock){
          //access x here. thread safe
       }

       //do something here but don't  use SharedResource  x, y
       // because will not be thread-safe

        synchronized(xLock) {
            synchronized(yLock) {
                //access x,y here. thread safe
            }
        }

        //do something here but don't use  SharedResource  x, y
        //because will not be thread-safe
    }//end of method1
}
```

coarse grained Vs fine grained locks

The JVM uses locks in conjunction with monitors. A monitor is basically a guardian who watches over a sequence of synchronized code and making sure only one thread at a time executes a synchronized piece of code. Each monitor is associated with an object reference. When a thread arrives at the first instruction in a block of code it must obtain a lock on the referenced object. The thread is not allowed to execute the code until it obtains the lock. Once it has obtained the lock, the thread enters the block of protected code. When the thread leaves the block, no matter how it leaves the block, it releases the lock on the associated object. For static methods, you acquire a class level lock.

1. Comprehensive list of Java multi-threading Q&A and basics.

2. 45+ Java multi threading interview Q&A with scenarios for intermediate to experienced.

Popular Posts

♦ 11 Spring boot interview questions & answers

889 views

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

847 views

18 Java scenarios based interview Questions and Answers

454 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

411 views

♦ 7 Java debugging interview questions & answers

315 views

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

313 views

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

302 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

287 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

266 views

8 Git Source control system interview questions & answers

223 views

Bio | **Latest Posts**

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.

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.

‹ ♦ 12 UML interview Questions & Answers

04: ♦ 6 popular Java multi-threading interview questions and answers with diagrams and code ›

Posted in FAQ Core Java Job Interview Q&A Essentials, Multithreading

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

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