Javarevisited

Blog about Java, Programming, Spring, Hibernate, Interview Questions, Books and Online Course Recommendations from Udemy, Pluralsight, Coursera, etc

Home core java spring hibernate collections multithreading design patterns interview questions coding data str

Java Certifications JDBC jsp-servlet JSON SQL Linux Courses online resources jvm-internals REST Eclipse

THURSDAY, MARCH 9, 2017

Top 50 Java Thread Interview Questions Answers for Experienced

You go to any Java interview, senior or junior, experience or freshers, you are bound to see a couple of questions from the thread, concurrency, and multi-threading. In fact, this built-in concurrency support is one of the strongest points of Java programming language and helped it to gain popularity among the enterprise world and programmers equally. Most of the lucrative Java developer position demands *excellent core Java multi-threading skills* and experience in developing, debugging, and tuning high-performance low latency concurrent Java applications. This is the reason, it is one of the most sought after skills in Java interviews. The multithreading and concurrency are also hard to master the concept and only good developers with solid experience can effectively deal with concurrency issues.

In a typical Java interview, the Interviewer slowly starts from basic concepts of Thread by asking questions like, why you need threads, how to create threads, which one is a better way to create threads e.g. by extending thread class or implementing Runnable and then slowly goes into Concurrency issues, challenges faced during the development of concurrent Java applications, Java memory model, higher-order concurrency utilities introduced in JDK 1.5, principles and design patterns of concurrent Java applications, classical multi-threading problems e.g. producer-consumer, dining philosopher, reader-writer or simply bounded buffer problems.

Since its also not enough just to know the basics of threading, you must know how to deal with concurrency problems like deadlock, race conditions, memory inconsistency, and various thread safety-related issues. These skills are thoroughly get tested by presenting various multi-threading and concurrency problems.

Many Java developers are used to only look and read interview questions before going for the interview, which is not bad but you should not be too far away. Also collecting questions and going through the same exercise is too much time consuming, that's why I have created this list of the *top 50 Java multi-threading and concurrency related questions*, collected from various interviews. I am only going to add new and recent interview questions as and when I am going to discover them.

Though you need good knowledge and solid experience to do well on Java interviews focused on advanced multithreading and concurrency skill, I strongly recommend Java programmers to read Effective Java and Java Concurrency in Practice twice before going to an interview. They do not only help you to answer questions better but also help you to present your idea clearly.

And, if you are serious about mastering Java multi-threading and concurrency then I also suggest you take a look at the **Java Multithreading, Concurrency, and Performance Optimization** course by Michael Pogrebinsy on Udemy. It's an advanced course to become an expert in Multithreading, concurrency, and Parallel programming in Java with a strong emphasis on high performance

By the way, I have not provided answers to some questions here, Why? because I expect most of Java developers to know the answers to this question and if not, also answers are widely available by using Google. If you don't find the answer to any particular question, you can always ask me in the comments section. You can even find answers to a few questions on the link provided or my earlier post Top 12 Java Thread Questions with Answers.

50 Interview questions from Java Multithreading and Concurrency

Here is our list of top questions from Java thread, concurrency, and multi-threading. You can use this list to prepare well for your Java interview.

Interview Questions
core java interview question (171)
Coding Interview Question (73)
data structure and algorithm (73)
interview questions (56)
SQL Interview Questions (33)
object oriented programming (32)
design patterns (30)
thread interview questions (30)
collections interview questions (25)
spring interview questions (20)
database interview questions (16)
servlet interview questions (15)
Programming interview question (6)
hibernate interview questions (6)

Best of Javarevisited
How Spring MVC works internally?
How to design a vending machine in Java?
How HashMap works in Java?
Why String is Immutable in Java?
10 Articles Every Programmer Must Read
How to convert lambda expression to method reference in Java 8?
10 Tips to improve Programming Skill
10 OOP design principles programmer should know
How Synchronization works in Java?
10 tips to work fast in Linux
5 Books to improve Coding Skills

Java Tutorials
date and time tutorial (22)
FIX protocol tutorial (15)
Java Certification OCPJP SCJP (30)
java collection tutorial (77)
java IO tutorial (28)
Java JSON tutorial (12)
Java multithreading Tutorials (59)
Java Programming Tutorials (18)
Java xml tutorial (16)
JDBC (30)
jsp-servlet (37)
online resources (214)

Categories
courses (206)
SQL (61)
database (43)
linux (40)
Eclipse (30)
Java Certification OCPJP SCJP (30)
JVM Internals (24)
JQuery (21)
REST (17)
Testing (13)
Maven (12)
general (12)

Blog Archive

1) What is Thread in Java? (answer)
The thread is an independent path of execution. It's a way to take advantage of multiple CPU available in a machine. By employing multiple threads you can speed up CPU bound tasks. For example, if one thread takes 100 milliseconds to do a job, you can use 10 threads to reduce that task into 10 milliseconds. Java provides excellent support for multithreading at the language level, and it's also one of the strong selling points.

2) What is the difference between Thread and Process in Java? (answer)
The thread is a subset of Process, in other words, one process can contain multiple threads. Two processes runs on different memory spaces, but all threads share the same memory space. Don't confuse this with stack memory, which is different for the different threads and used to store local data to that thread. For more detail see the answer.

3) How do you implement Thread in Java? (answer)
At the language level, there are two ways to implement Thread in Java. An instance of `java.lang.Thread` represents a thread but it needs a task to execute, which is an instance of `interface java.lang.Runnable`. Since `Thread` class itself implement `Runnable`, you can override `run()` method either by extending Thread class or just implementing Runnable interface. For a detailed answer and discussion see this article.

4) When to use Runnable vs Thread in Java? (answer)
This is a follow-up of previous multi-threading interview question. As we know we can implement thread either by extending `Thread` class or implementing `Runnable` interface, the question arise, which one is better and when to use one? This question will be easy to answer if you know that Java programming language doesn't support multiple inheritances of class, but it allows you to implement multiple interfaces. Which means, it's better to implement `Runnable` then extends `Thread` if you also want to extend another class e.g. `Canvas` or `CommandListener`. For more points and discussion you can also refer this post.

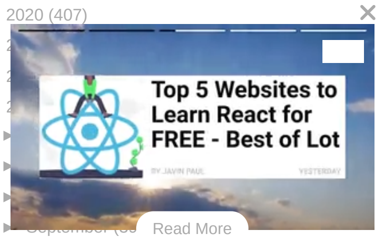
6) What is the difference between start() and run() method of Thread class? (answer)
One of trick Java question from early days, but still good enough to differentiate between shallow understanding of Java threading model `start()` method is used to start newly created thread, while `start()` internally calls `run()` method, there is difference calling `run()` method directly. When you invoke `run()` as normal method, its called in the same thread, no new thread is started, which is the case when you call `start()` method. Read this answer for much more detailed discussion.

7) What is the difference between Runnable and Callable in Java? (answer)
Both Runnable and Callable represent task which is intended to be executed in a separate thread. Runnable is there from JDK 1.0 while Callable was added on JDK 1.5. Main difference between these two is that Callable's `call()` method can return value and throw Exception, which was not possible with Runnable's `run()` method. Callable return Future object, which can hold the result of computation. See my blog post on the same topic for a more in-depth answer to this question.

8) What is the difference between CyclicBarrier and CountDownLatch in Java? (answer)
Though both CyclicBarrier and CountDownLatch wait for number of threads on one or more events, the main difference between them is that you can not re-use `CountDownLatch` once count reaches to zero, but you can reuse same `CyclicBarrier` even after barrier is broken. See this answer for few more points and sample code example.

9) What is Java Memory model? (answer)
Java Memory model is set of rules and guidelines which allows Java programs to behave deterministically across multiple memory architecture, CPU, and operating system. It's particularly important in case of multi-threading. Java Memory Model provides some guarantee on which changes made by one thread should be visible to others, one of them is happens-

► 2020 (407)

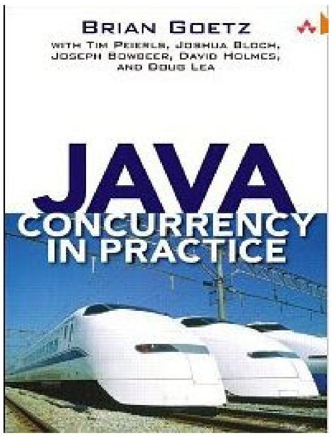


► August (50)
► July (51)
► June (50)
► May (45)
► April (74)
▼ March (97)
- How to use EnumSet in Java with Example
- Failed to connect to queue manager - WebSphere MQ ...
- Private in Java: Why should you always keep fields...
- What is Effectively Final variable of Java 8
- How to Compare two String in Java - String Compari...
- How to format Date in SQL Server and Sybase Example
- Hibernate NullPointerException due to Space in HQL...
- How to loop ArrayList in Java - List Itrator Trave...
- What is Assertion in Java - Java Assertion Tutoria
- 2 Scenario Based Data Structure and Algorithm Desi...
- How to Convert and Print Byte array to Hex String ...
- Parsing Large JSON Files using Jackson Streaming A...
- 3 ways to Find First Non Repeated Character in a S...
- Difference between Stack and Queue Data Structure ...
- How to Reverse Array in Place in Java? Solution Wi...
- Difference between WeakReference vs SoftReference ...
- Java 1.5 Generics Tutorial: How Generics in Java w...
- 10 Exception handling Best Practices in Java Progr...
- Difference between start and run method in Thread ...
- Difference between Wait and Sleep, Yield in Java
- Top 10 Servlet Interview Question Answers - J2EE
- Difference between Stack and Heap memory in Java
- Why use PreparedStatement in Java JDBC – Example T...
- SimpleDateFormat in Java is not Thread-Safe Use Ca...
- How to find file and directory size in Unix with E...
- How Garbage Collection works in Java
- 10 difference between Java and JavaScript for Prog...
- Difference between transient and volatile keyword ...
- How to fix java.io.FileNotFoundException: (Access ...
- Top 10 basic networking commands in linux/unix
- How to Reverse an Array in Java - Integer and Stri...
- Top 30 Scala and Functional Programming Interview ...
- Binary Search vs Contains Performance in Java List
- How to check if two String are Anagram in Java - P...
- How to Generate MD5 Hash in Java - String Byte Arr...

before relationship. This relationship defines several rules which allows programmers to anticipate and reason behaviour of concurrent Java programs. For example, happens-before relationship guarantees :

- Each action in a thread happens-before every action in that thread that comes later in the program order, this is known as program order rule.
- An unlock on a monitor lock happens-before every subsequent lock on that same monitor lock, also known as Monitor lock rule.
- A write to a volatile field happens-before every subsequent read of that same field, known as Volatile variable rule.
- A call to Thread.start on a thread happens-before any other thread detects that thread has terminated, either by successfully return from `Thread.join()` or by `Thread.isAlive()` returning false, also known as Thread start rule.
- A thread calling `interrupt()` on another thread happens-before the interrupted thread detects the interrupt(either by having InterruptedException thrown, or invoking isInterrupted or interrupted), popularly known as Thread Interruption rule.
- The end of a constructor for an object happens-before the start of the finalizer for that object, known as Finalizer rule.
- If A happens-before B, and B happens-before C, then A happens-before C, which means happens-before guarantees Transitivity.

I strongly suggest reading Chapter 16 of Java Concurrency in Practice to understand Java Memory model in more detail.


BRIAN GOETZ
WITH TIM PEIERLS, JOSHUA BLOCH, JOSEPH BOWBEER, DAVID HOLMES, AND DOUG LEA
JAVA
CONCURRENCY IN PRACTICE

10) What is volatile variable in Java? (answer)
volatile is a special modifier, which can only be used with instance variables. In concurrent Java programs, changes made by multiple threads on instance variables is not visible to other in absence of any synchronizers like synchronized keyword or locks. Volatile variable guarantees that a write will happen before any subsequent read: as stated: *"volatile variable rule"* in previous question. Read this answer to learn more about volatile variable and when to use them.

11) What is thread-safety? is Vector a thread-safe class? (Yes, see details)
Thread-safety is a property of an object or code which guarantees that if executed or used by multiple threads in any manner e.g. read vs write it will behave as expected. For example, a thread-safe counter object will not miss any count if same instance of that counter is shared among multiple threads. Apparently, you can also divide collection classes in two category, thread-safe and non-thread-safe. Vector is indeed a thread-safe class and it achieves thread-safety by synchronizing methods which modify state of Vector, on the other hand, its counterpart `ArrayList` is not thread-safe.

12) What is race condition in Java? Given one example? (answer)
Race condition are cause of some subtle programming bugs when Java programs are exposed to concurrent execution environment. As the name suggests, a race condition occurs due to race between multiple threads, if a thread which is supposed to execute first lost the race and executed second, behaviour of code changes, which surface as non-deterministic bugs. This is one of the hardest bugs to find and re-produce because of random nature of racing between

What is difference between SQL, T-SQL and

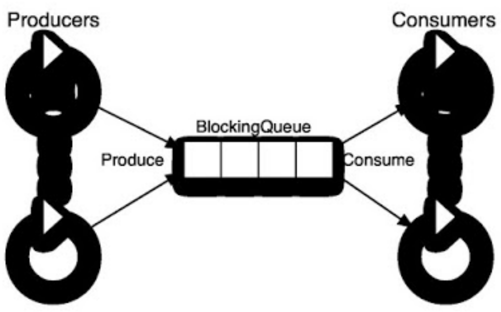
Can You Overlo... Static methods in Java
Java Synchronization Tutorial : What, How and Why?
How to Check if a Number is Binary in Java - Progr...
What is Daemon thread in Java and Difference to No...
How to get ServletContext in Servlet, JSP, Action ...
How to Convert an Array to Comma Separated String ...
Why character array is better than String for Stor...
JDBC - java.lang.ClassNotFoundException: com.mysql...
How to find Largest Prime Factor of a Number in Ja...
Mixing static and non static synchronized method -...
Why use Underscore in Numbers from Java SE 7 - Und...
What is Method References in Java 8? An Example
Difference between Struts 1 and Struts 2 Web Devel...
10 Hibernate Interview Questions and Answers fo J...
How to delete a directory with files in Java - Exa...
Bitwise and BitShift Operators in Java - AND, OR, ...
How to Clone Collection in Java - Deep copy of Arr...
How to Remove First and Last Character of String i...
Difference between Singleton Pattern vs Static Cla...
What is GET and POST method in HTTP and HTTPS Prot...
What is Static and Dynamic binding in Java with Ex...
10 points on finalize method in Java – Tutorial Ex...
What is Encapsulation in Java and OOPS with Example
10 example of chmod command in UNIX Linux
Java 8 Date Time - 20 Examples of LocalDate, Local...
Top 50 Java Thread Interview Questions Answers for...
How to Increase Console Buffer Size in Eclipse IDE...
How to Implement Stack in Java Using Array and Gen...
Top 10 EJB Interview Question and Answer asked in ...
java.lang.UnsatisfiedLinkError: no dll in java.lib...
ReentrantLock Example in Java, Difference between ...
Why we use Threads in Java?
How to read Properties File in Java – XML and Text...
Top 20 FIX Protocol Interview Questions
Top 10 MQ Series Interview Questions Answers - Web...
How to add or list certificates from keystore or t...
How to create Immutable Class and Object in Java -...
Tibco tutorial : Tibco RV tips and commands

threads. One example of race condition is out-of-order processing, see this answer for some more example of race conditions in Java programs.

13) How to stop a thread in Java? (answer(answer)
I always said that Java provides rich APIs for everything but ironically Java doesn't provide a sure shot way of stopping thread. There was some control methods in JDK 1.0 e.g. `stop()`, `suspend()` and `resume()` which was deprecated in later releases due to potential deadlock threats, from then Java API designers has not made any effort to provide a consistent, thread-safe and elegant way to stop threads. Programmers mainly rely on the fact that thread stops automatically as soon as they finish execution of `run()` or `call()` method. To manually stop, programmers either take advantage of volatile boolean variable and check in every iteration if run method has loops or interrupt threads to abruptly cancel tasks. See this tutorial for sample code of stopping thread in Java.

14) What happens when an Exception occurs in a thread? (answer)
This is one of the good tricky Java question I have seen in interviews. In simple words, If not caught thread will die, if an uncaught exception handler is registered then it will get a call back. `Thread.UncaughtExceptionHandler` is an interface, defined as nested interface for handlers invoked when a Thread abruptly terminates due to an uncaught exception. When a thread is about to terminate due to an uncaught exception the Java Virtual Machine will query the thread for its `UncaughtExceptionHandler` using `Thread.getUncaughtExceptionHandler()` and will invoke the handler's `uncaughtException()` method, passing the thread and the exception as arguments.

15) How do you share data between two thread in Java? (answer)
You can share data between threads by using shared object, or concurrent data structure like BlockingQueue. See this tutorial to learn inter-thread communication in Java. It implements Producer consumer pattern using wait and notify methods, which involves sharing objects between two threads.



16) What is the difference between notify and notifyAll in Java? (answer)
This is another tricky questions from core Java interviews, since multiple threads can wait on single monitor lock, Java API designer provides method to inform only one of them or all of them, once waiting condition changes, but they provide half implementation. There `notify()` method doesn't provide any way to choose a particular thread, that's why its only useful when you know that there is only one thread is waiting. On the other hand, `notifyAll()` sends notification to all threads and allows them to compete for locks, which ensures that at-least one thread will proceed further. See my blog post on similar topic for a more detailed answer and code example.

17) Why wait, notify and notifyAll are not inside thread class? (answer)
This is a design related question, which checks what candidate thinks about existing system or does he ever thought of something which is so common but looks in-appropriate at first. In order to answer this question, you have to give some reasons why it make sense for these three method to be in Object class, and why not on Thread class. One reason which is obvious is that Java provides lock at object level not at thread level. Every object has lock, which is acquired by thread. Now if thread needs to wait for certain lock it make sense to call wait() on that object

Tibco Tutorial: Fundamentals of Tibco RV

Read More

References

1. Oracle's Java Tech Network
2. jQuery Documentation
3. Microsoft SQL Server Documentation
4. Java SE 8 API Documentation
5. Spring Documentation
6. Oracle's JAva Certification
7. Spring Security 5 Documentation

Pages

Privacy Policy
Terms and Conditions

rather than on that thread. Had `wait()` method declared on Thread class, it was not clear that for which lock thread was waiting. In short, since wait, notify and notifyAll operate at lock level, it make sense to defined it on object class because lock belongs to object. You can also see this article for more elaborate answer of this question.

18) What is ThreadLocal variable in Java? (answer)
ThreadLocal variables are special kind of variable available to Java programmer. Just like instance variable is per instance, `ThreadLocal` variable is per thread. It's a nice way to achieve thread-safety of expensive-to-create objects, for example you can make SimpleDateFormat thread-safe using `ThreadLocal`. Since that class is expensive, its not good to use it in local scope, which requires separate instance on each invocation. By providing each thread their own copy, you shoot two birds with one arrow. First, you reduce number of instance of expensive object by reusing fixed number of instances, and Second, you achieve thread-safety without paying cost of synchronization or immutability. Another good example of thread local variable is `ThreadLocalRandom` class, which reduces number of instances of `expensive-to-create` Random object in multi-threading environment. See this answer to learn more about thread local variables in Java.

19) What is FutureTask in Java? (answer)
`FutureTask` represents a cancellable asynchronous computation in concurrent Java application. This class provides a base implementation of Future, with methods to start and cancel a computation, query to see if the computation is complete, and retrieve the result of the computation. The result can only be retrieved when the computation has completed; the get methods will block if the computation has not yet completed. A `FutureTask` object can be used to wrap a Callable or Runnable object. Since `FutureTask` also implements `Runnable`, it can be submitted to an Executor for execution.

20) What is the difference between the interrupted() and isInterrupted() method in Java? (answer)
Main difference between `interrupted()` and `isInterrupted()` is that former clears the interrupt status while later does not. The interrupt mechanism in Java multi-threading is implemented using an internal flag known as the interrupt status. Interrupting a thread by calling `Thread.interrupt()` sets this flag. When interrupted thread checks for an interrupt by invoking the static method `Thread.interrupted()`, interrupt status is cleared. The non-static `isInterrupted()` method, which is used by one thread to query the interrupt status of another, does not change the interrupt status flag. By convention, any method that exits by throwing an `InterruptedException` clears interrupt status when it does so. However, it's always possible that interrupt status will immediately be set again, by another thread invoking interrupt

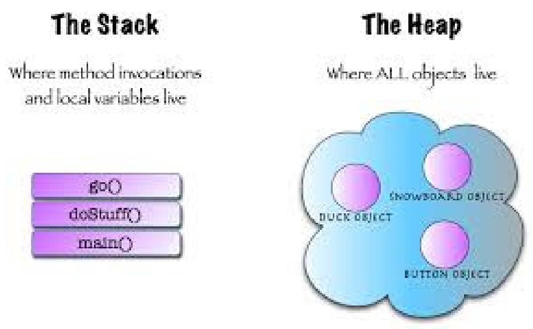
21) Why wait and notify method are called from synchronized block? (answer)
Main reason for calling wait and notify method from either synchronized block or method is that it made mandatory by Java API. If you don't call them from synchronized context, your code will throw `IllegalMonitorStateException`. A more subtle reason is to avoid the race condition between wait and notify calls. To learn more about this, check my similarly titled post here.

22) Why should you check condition for waiting in a loop? (answer)
Its possible for a waiting thread to receive false alerts and spurious wake up calls, if it doesn't check the waiting condition in loop, it will simply exit even if condition is not met. As such, when a waiting thread wakes up, it cannot assume that the state it was waiting for is still valid. It may have been valid in the past, but the state may have been changed after the `notify()` method was called and before the waiting thread woke up. That's why it always better to call wait() method from loop, you can even create template for calling wait and notify in Eclipse. To learn more about this question, I would recommend you to read Effective Java items on thread and synchronization.

23) What is the difference between synchronized and concurrent collection in

Java? (answer)
Though both synchronized and concurrent collection provides thread-safe collection suitable for multi-threaded and concurrent access, later is more scalable than former. Before Java 1.5, Java programmers only had synchronized collection which becomes source of contention if multiple thread access them concurrently, which hampers scalability of system. Java 5 introduced concurrent collections like `ConcurrentHashMap`, which not only provides thread-safety but also improves scalability by using modern techniques like lock stripping and partitioning internal table. See this answer for more differences between synchronized and concurrent collection in Java.

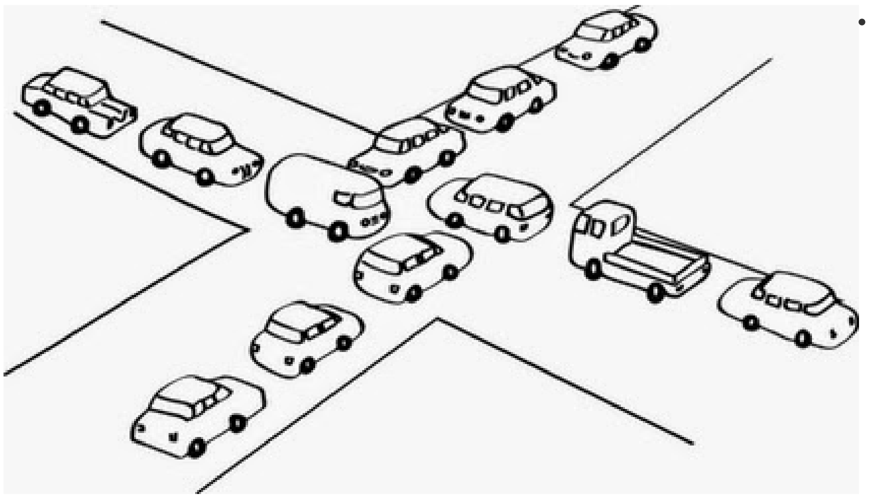
24) What is the difference between Stack and Heap in Java? (answer)
Why does someone this question as part of multi-threading and concurrency? because Stack is a memory area which is closely associated with threads. To answer this question, both stack and heap are specific memories in Java application. Each thread has their own stack, which is used to store local variables, method parameters and call stack. Variable stored in one Thread's stack is not visible to other. On another hand, the heap is a common memory area which is shared by all threads. Objects whether local or at any level is created inside heap. To improve performance thread tends to cache values from heap into their stack, which can create problems if that variable is modified by more than one thread, this is where volatile variables come into the picture. volatile suggest threads read the value of variable always from main memory. See this article for learning more about stack and heap in Java to answer this question in greater detail.



25) What is thread pool? Why should you thread pool in Java? (answer)
Creating thread is expensive in terms of time and resource. If you create thread at time of request processing it will slow down your response time, also there is only a limited number of threads a process can create. To avoid both of these issues, a pool of thread is created when application starts-up and threads are reused for request processing. This pool of thread is known as "thread pool" and threads are known as worker thread. From JDK 1.5 release, Java API provides Executor framework, which allows you to create different types of thread pools e.g. single thread pool, which process one task at a time, fixed thread pool (a pool of fixed number of threads) or cached thread pool (an expandable thread pool suitable for applications with many short lived tasks). See this article to learn more about thread pools in Java to prepare detailed answer of this question.

26) Write code to solve Producer Consumer problem in Java? (answer)
Most of the threading problem you solved in the real world are of the category of Producer consumer pattern, where one thread is producing task and another thread is consuming that. You must know how to do inter thread communication to solve this problem. At the lowest level, you can use wait and notify to solve this problem, and at a high level, you can leverage Semaphore or BlockingQueue to implement Producer consumer pattern, as shown in this tutorial.

27) How do you avoid deadlock in Java? Write Code?
Deadlock is a condition in which two threads wait for each other to take action which allows them to move further. It's a serious issue because when it happen your program hangs and doesn't do the task it is intended for. In order for deadlock to happen, following four conditions must be true:



- **Mutual Exclusion :** At least one resource must be held in a non-shareable mode. Only one process can use the resource at any given instant of time.
- **Hold and Wait:** A process is currently holding, at least, one resource and requesting additional resources which are being held by other processes.
- **No Pre-emption:** The operating system must not de-allocate resources once they have been allocated; they must be released by the holding process voluntarily.
- **Circular Wait:** A process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource.

The easiest way to avoid deadlock is to prevent *Circular wai*t, and this can be done by acquiring locks in a particular order and releasing them in reverse order so that a thread can only proceed to acquire a lock if it held the other one. Check this tutorial for the actual code example and detailed discussion on techniques for avoiding deadlock in Java.

28) What is the difference between livelock and deadlock in Java? (answer)
This question is extension of previous interview question. A livelock is similar to a deadlock, except that the states of the threads or processes involved in the livelock constantly change with regard to one another, without any one progressing further. Livelock is a special case of resource starvation. A real-world example of livelock occurs when two people meet in a narrow corridor, and each tries to be polite by moving aside to let the other pass, but they end up swaying from side to side without making any progress because they both repeatedly move the same way at the same time. In short, the main difference between livelock and deadlock is that in former state of process change but no progress is made.

29) How do you check if a Thread holds a lock or not? (answer)
I didn't even know that you can check if a Thread already holds lock before this question hits me in a telephonic round of Java interview. There is a method called `holdsLock()` on `java.lang.Thread`, it returns true if and only if the current thread holds the monitor lock on the specified object. You can also check this article for the more detailed answer.

30) How do you take thread dump in Java? (answer)
There are multiple ways to take thread dump of Java process depending upon operating system. When you take thread dump, JVM dumps state of all threads in log files or standard error console. In windows you can use `Ctrl + Break` key combination to take thread dump, on Linux you can use `kill -3` command for same. You can also use a tool called `jstack` for taking thread dump, it operate on process id, which can be found using another tool called `jps`.

31) Which JVM parameter is used to control stack size of a thread? (answer)
This is the simple one, -Xss parameter is used to control stack size of Thread in Java. You can see this list of JVM options to learn more about this parameter.

32) What is the difference between synchronized and ReentrantLock in Java? (answer)
There were days when the only way to provide mutual exclusion in Java was via synchronized keyword, but it has several shortcomings e.g. you can not extend lock beyond a method or block boundary, you can not give up trying for a lock etc. Java 5 solves this problem by providing more sophisticated control via Lock interface. ReentrantLock is a common implementation of Lock interface and provides re-entrant mutual exclusion Lock with the same basic behavior and semantics as the implicit monitor lock accessed using synchronized methods and statements, but with extended capabilities. See this article learn about those capabilities and some more differences between synchronized vs ReentrantLock in Java.

33) There are three threads T1, T2, and T3? How do you ensure sequence T1, T2, T3 in Java? (answer)
Sequencing in multi-threading can be achieved by different means but you can simply use the join() method of thread class to start a thread when another one has finished its execution. To ensure three threads execute you need to start the last one first e.g. T3 and then call join methods in reverse order e.g. T3 calls T2. join and T2 calls T1.join, these ways T1 will finish first and T3 will finish last. To learn more about join method, see this tutorial.

34) What does yield method of Thread class do? (answer)
Yield method is one way to request current thread to relinquish CPU so that other thread can get a chance to execute. Yield is a static method and only guarantees that current thread will relinquish the CPU but doesn't say anything about which other thread will get CPU. Its possible for the same thread to get CPU back and start its execution again. See this article to learn more about yield method and to answer this question better.

35) What is the concurrency level of ConcurrentHashMap in Java? (answer)
ConcurrentHashMap achieves it's scalability and thread-safety by partitioning actual map into a number of sections. This partitioning is achieved using concurrency level. Its optional parameter of ConcurrentHashMap constructor and it's default value is 16. The table is internally partitioned to try to permit the indicated number of concurrent updates without contention. To learn more about concurrency level and internal resizing, see my post How ConcurrentHashMap works in Java.

36) What is Semaphore in Java? (answer)
Semaphore in Java is a new kind of synchronizer. It's a counting semaphore. Conceptually, a semaphore maintains a set of permits. Each acquire() blocks if necessary until a permit is available, and then takes it. Each release() adds a permit, potentially releasing a blocking acquirer. However, no actual permit objects are used; the Semaphore just keeps a count of the number available and acts accordingly. Semaphore is used to protect an expensive resource which is available in fixed number e.g. database connection in the pool. See this article to learn more about counting Semaphore in Java.

37) What happens if you submit a task when the queue of the thread pool is already filled? (answer)
This is another tricky question on my list. Many programmers will think that it will block until a task is cleared but its true. `ThreadPoolExecutor's submit()` method throws `RejectedExecutionException` if the task cannot be scheduled for execution.

38) What is the difference between the submit() and execute() method thread pool in Java? (answer)
Both methods are ways to submit a task to thread pools but there is a slight difference between them. execute(Runnable command) is defined in Executor interface and executes given task in future, but more importantly, it does not return anything. Its return type is void. On other hand `submit()` is an overloaded method, it can take either `Runnable` or `Callable` task and can return Future object which can hold the pending result of computation. This method is defined on `ExecutorService` interface, which extends `Executor` interface, and every other thread pool class e.g. `ThreadPoolExecutor` or `ScheduledThreadPoolExecutor` gets these methods. To learn more about thread pools you can check this article.

39) What is blocking method in Java? (answer)
A blocking method is a method which blocks until the task is done, for example, accept() method of ServerSocket blocks until a client is connected. here blocking means control will not return to the caller until the task is finished. On the other hand, there is an asynchronous or non-blocking method which returns even before the task is finished. To learn more about blocking method see this answer.

40) Is Swing thread-safe? What do you mean by Swing thread-safe? (answer)
You can simply this question as No, Swing is not thread-safe, but you have to explain what you mean by that even if the interviewer doesn't ask about it. When we say swing is not thread-safe we usually refer its component, which can not be modified in multiple threads. All update to GUI components has to be done on AWT thread, and Swing provides synchronous and asynchronous callback methods to schedule such updates. You can also read my article to learn more about swing and thread-safety to better answer this question. Even next two questions are also related to this concept.

41) What is the difference between invokeAndWait and invokeLater in Java? (answer)
These are two methods Swing API provides Java developers for updating GUI components from threads other than Event dispatcher thread. `InvokeAndWait()` synchronously update GUI component, for example, a progress bar, once progress is made, the bar should also be updated to reflect that change. If progress is tracked in a different thread, it has to call `invokeAndWait()` to schedule an update of that component by Event dispatcher thread. On another hand, `invokeLater()` is an asynchronous call to update components. You can also refer this answer for more points.

42) Which method of Swing API are thread-safe in Java? (answer)
This question is again related to swing and thread-safety though components are not thread-safe there is a certain method which can be safely called from multiple threads. I know about `repaint()`, and `revalidate()` being thread-safe but there are other methods on different swing components e.g. `setText()` method of `JTextComponent`, `insert()` and `append()` method of `JTextArea` class.

43) How to create an Immutable object in Java? (answer)
This question might not look related to multi-threading and concurrency, but it is. Immutability helps to simplify already complex concurrent code in Java. Since immutable object can be shared without any synchronization its very dear to Java developers. Core value object, which is meant to be shared among thread should be immutable for performance and simplicity. Unfortunately there is no @Immutable annotation in Java, which can make your object immutable, hard work must be done by Java developers. You need to keep basics like initializing state in constructor, no setter methods, no leaking of reference, keeping separate copy of mutable object to create Immutable object. For step by step guide see my post, how to make an object Immutable in Java. This will give you enough material to answer this question with confidence.

44) What is ReadWriteLock in Java? (answer)
In general, read write lock is the result of lock stripping technique to improve the performance of concurrent applications. In Java, `ReadWriteLock` is an interface which was added in Java 5 release. A `ReadWriteLock` maintains a pair of associated locks, one for read-only operations and one for writing. The read lock may be held simultaneously by multiple reader threads, so long as there are no writers. The write lock is exclusive. If you want you can implement this interface with your own set of rules, otherwise you can use `ReentrantReadWriteLock`, which comes along with JDK and supports a maximum of 65535 recursive write locks and 65535 read locks.

45) What is busy spin in multi-threading? (answer)
Busy spin is a technique which concurrent programmers employ to make a thread wait on

certain condition. Unlike traditional methods e.g. wait(), sleep() or yield() which all involves relinquishing CPU control, this method does not relinquish CPU, instead it the just runs empty loop. Why would someone do that? to preserve CPU caches. In a multi-core system, it's possible for a paused thread to resume on a different core, which means rebuilding cache again. To avoid cost of rebuilding cache, programmer prefer to wait for much smaller time doing busy spin. You can also see this answer to learn more about this question.

46) What is the difference between the volatile and atomic variable in Java? (answer)
This is an interesting question for Java programmer, at first, volatile and atomic variable look very similar, but they are different. Volatile variable provides you happens-before guarantee that a write will happen before any subsequent write, it doesn't guarantee atomicity. For example count++ operation will not become atomic just by declaring count variable as volatile. On the other hand `AtomicInteger` class provides atomic method to perform such compound operation atomically e.g. `getAndIncrement()` is atomic replacement of increment operator. It can be used to atomically increment current value by one. Similarly you have atomic version for other data type and reference variable as well.

47) What happens if a thread throws an Exception inside synchronized block? (answer)
This is one more tricky question for average Java programmer, if he can bring the fact about whether lock is released or not is a key indicator of his understanding. To answer this question, no matter how you exist synchronized block, either normally by finishing execution or abruptly by throwing exception, thread releases the lock it acquired while entering that synchronized block. This is actually one of the reasons I like synchronized block over lock interface, which requires explicit attention to release lock, generally this is achieved by releasing the lock in a finally block.

48) What is double checked locking of Singleton? (answer)
This is one of the very popular question on Java interviews, and despite its popularity, chances of candidate answering this question satisfactory is only 50%. Half of the time, they failed to write code for double checked locking and half of the time they failed how it was broken and fixed on Java 1.5. This is actually an old way of creating thread-safe singleton, which tries to optimize performance by only locking when Singleton instance is created first time, but because of complexity and the fact it was broken for JDK 1.4, I personally don't like it. Anyway, even if you not prefer this approach its good to know from interview point of view. Since this question deserve a detailed answer, I have answered in a separate post, you can read my post how double checked locking on Singleton works to learn more about it.

49) How to create thread-safe Singleton in Java? (answer)
This question is actually follow-up of the previous question. If you say you don't like double checked locking then Interviewer is bound to ask about alternative ways of creating thread-safe Singleton class. There are actually man, you can take advantage of class loading and static variable initialization feature of JVM to create instance of Singleton, or you can leverage powerful enumeration type in Java to create Singleton. I actually preferred that way, you can also read this article to learn more about it and see some sample code.

50) List down 3 multi-threading best practice you follow? (answer)
This is my favorite question because I believe that you must follow certain best practices while writing concurrent code which helps in performance, debugging and maintenance. Following are three best practices, I think an average Java programmer should follow:

- **Always give meaningful name to your thread**This goes a long way to find a bug or trace an execution in concurrent code. `OrderProcessor`, `QuoteProcessor` or `TradeProcessor` is much better than Thread-1. Thread-2 and Thread-3. The name should say about task done by that thread. All major framework and even JDK follow this best practice.
- **Avoid locking or Reduce scope of Synchronization**
Locking is costly and context switching is even costlier. Try to avoid synchronization and locking as much as possible and at a bare minimum, you should reduce critical section.

That's why I prefer synchronized block over synchronized method because it gives you absolute control on the scope of locking.

- **Prefer Synchronizers over wait and notify**
  Synchronizers like `CountDownLatch`, `Semaphore`, `CyclicBarrier` or `Exchanger` simplifies coding. It's very difficult to implement complex control flow right using wait and notify. Secondly, these classes are written and maintained by best in business and there is good chance that they are optimized or replaced by better performance code in subsequent JDK releases. By using higher level synchronization utilities, you automatically get all these benefits.
- **Prefer Concurrent Collection over Synchronized Collection**
  This is another simple best practice which is easy to follow but reap good benefits. Concurrent collection are more scalable than their synchronized counterpart, that's why its better to use them while writing concurrent code. So next time if you need map, think about `ConcurrentHashMap` before thinking `Hashtable`. See my article Concurrent Collections in Java, to learn more about modern collection classes and how to make best use of them.

Top 5 Websites to Learn React for FREE - Best of Lot

BY JAVIN PAUL YESTERDAY

Read More

51) How do you force to start a Thread in Java? (answer)
This question is like how do you force garbage collection in Java, there is no way though you can make a request using `System.gc()` but it's not guaranteed. On Java multi-threading there is absolutely no way to force start a thread, this is controlled by thread scheduler and Java exposes no API to control thread schedule. This is still a random bit in Java.

52) What is the fork-join framework in Java? (answer)
The fork-join framework, introduced in JDK 7 is a powerful tool available to Java developer to take advantage of multiple processors of modern day servers. It is designed for work that can be broken into smaller pieces recursively. The goal is to use all the available processing power to enhance the performance of your application. One significant advantage of The `fork/join` framework is that it uses a work-stealing algorithm. Worker threads that run out of things to do can steal tasks from other threads that are still busy. See this article for the much more detailed answer to this question.

53) What is the difference between calling wait() and sleep() method in Java multi-threading? (answer)
Though both wait and sleep introduce some form of pause in Java application, they are the tool for different needs. Wait method is used for inter-thread communication, it relinquishes lock if waiting for a condition is true and wait for notification when due to an action of another thread waiting condition becomes false. On the other hand `sleep()` method is just to relinquish CPU or stop execution of current thread for specified time duration. Calling sleep method doesn't release the lock held by the current thread. You can also take look at this article to answer this question with more details.

That's all on this list of **top 50 Java multi-threading and concurrency interview questions**. I have not shared answers of all the questions but provided enough hints and links to explore further and find answers by yourselves. As I said, let me know if you don't find answer of any particular question and I will add answer here.

You can use this list to not only to prepare for your core Java and programming interviews but also to check your knowledge about basics of threads, multi-threading, concurrency, design patterns and threading issues like race conditions, deadlock and thread safety problems.

My intention is to make this list of question as the mother of all list of Java Multi-threading questions, but this can not be done without your help. You can also share any question with us, which has been asked to you or any question for which you yet to find an answer.

This master list is equally useful to Java developers of all levels of experience. You can read through this list even if you have 2 to 3 years of working experience as a junior developer or 5 to 6 years as a senior developer. It's even useful for freshers and beginners to expand their knowledge. I will add new and latest multi-threading question as and when I come across, and I request you all to ask, share and answer questions via comments to keep this list relevant to all Java programmers.

Further Learning
Multithreading and Parallel Computing in Java
Applying Concurrency and Multi-threading to Common Java Patterns
Java Concurrency in Practice Course by Heinz Kabutz

By javin paul at March 09, 2017
Labels: concurrency , core java interview question , Java multithreading Tutorials , thread interview questions

33 comments :

Anonymous said...
Hi Javin,

Excellent piece! I have a question tho, is there a decent step by step guide out there for working through threading in Java? Reason I ask is, I find it is helpful to do assignments with the concept and while I do have knowledge about threading in java I am just not sure I have a complete picture.

Again excellent work, I hope to see more articles! :)

July 9, 2014 at 7:31 AM

Anonymous said...
I would argue that for practical knowledge, familiarity with the java.util.concurrent packages is much more important than knowing implementation details of the Thread class.

July 9, 2014 at 1:01 PM

Anonymous said...
Hi,

Can you please tell me about finalize block purpose except below thing which i know.
1. Before object going to garbage collection, the finalized method is call.
2. In this method we can do connection close but this we can already handle in finally block.

Thanks

October 2, 2014 at 12:10 AM

Anonymous said...
One questions I found interesting from threads was , what is difference between interrupt(), isInterrupted() and interrupted()? first one interrupt the thread on which object it was called, second is called from the thread which was interrupted and return true if it was indeed interrupted, otherwise false. The last one clear the interrupted status of thread.

November 20, 2014 at 7:36 PM

Anonymous said...
Hi Javin,

"10) What is volatile variable in Java?
volatile is a special modifier, which can only be used with instance variables."

This statement of your is not correct, because you can associate volatile with static and instance variables. The only restriction I can see is that we cannot associate final with volatile, which will eventually throw compile time error.

April 16, 2015 at 3:12 AM

javin paul said...

@Anonymous, Yes, the correct wording should be "volatile" modifier can only be used with member variable, both instance and class variable but cannot be used with local variables.

June 23, 2015 at 6:36 AM

javin paul said...

@Anonymous, Thanks, glad to hear that you find these interview questions helpful

June 23, 2015 at 6:36 AM

Anonymous said...

RE: #33 (Executing T1, T2, T3 sequentially)

Isn't it simpler to join all three threads to main?

```
t1.start();
t1.join();
t2.start();
t2.join();
t3.start();
t3.join();
```

January 12, 2016 at 7:13 PM

javin paul said...

@Anonymous, sure, that will work, good solution

January 13, 2016 at 4:45 AM

Unknown said...

Very good Article on Threads. I have been asked one open question about threads. How will you detect if Deadlock is happened in production environment. How will you detect which piece of code caused that deadlock? In this case its production not the local machine to run Jconsole or any custom options or programs. I will be glad if its answered.
I have answered generically like , I will check the logs at which point of time The thread is waiting for resource and no continuation afterwards Etc. I will check the logic, I will try to reproduce the same in Dev env Etc. but i did not satisfy myself.

February 2, 2016 at 10:05 AM

javin paul said...

@Unknown, if you hear that your program is stuck in production, the first thing you should do is to take the thread dump by executing kill -3 and heap dump. You can later analyze those two things to find any problem. For finding deadlok there are tools which can read your thread dump and say which two threads are engaged in deadlock and can also point to the Java class file and line numbers.

February 6, 2016 at 1:25 AM

Anonymous said...

Thanks! Very useful.

February 19, 2016 at 11:11 AM

sree said...

I tried a online tool http://gceasy.io , it can read my sun jdk 1.8 gc log.

March 12, 2016 at 11:51 PM

Akbar G said...

Small correction...
In Question No. 10(What is volatile variable in Java?) you said:
"volatile is a special modifier, which can only be used with instance variables."

I think it should be member variables (not instance variables).

April 12, 2016 at 1:03 AM

Anonymous said...

Scenario question - In a high volume fast trading application incoming requests are being picked up and processed by thread. Assume user places and order but then instantly realizes made a mistake and hit cancel. Both requests picked up by threads, however the Cancel Order thread starts running first. What needs to be done. The Q was not very clear if what needs to be done to avoid such situation or what has to be done if this happens.

my responses were -
Not have such a UI where user can cancel before the order response is received(like grey out the cancel button till some response for order request is recvd.).

One more was that the Cancel order will operate on certain generated order id(or order object) and since the order thread hasnt run the cancel thread wouldnt have anything to cancel so the cancel thread can terminate(or better return back to the thread pool). The guy interviewing said how to determine if order id(object exists). I initially said fetch from db and check to which he was not happy at all. I then said lookup in cache since it would be there. But couldnt satisfy.

Read More

Also said maybe use a queue on which the threads place the requests which are picked pick up for processing later . Now being a queue the order request has to be processed first and the the cancel will be processed later and will have an order object to operate on.

Folks, how to approach such a scenario - any suggestions please?

April 18, 2016 at 1:39 PM

Anonymous said...

Excellent post.Very useful

May 11, 2016 at 12:04 AM

Kumar Raju said...

Pretty useful for my interview preparation. Thank you, Javin.

June 9, 2016 at 7:52 AM

GOPI said...

Great post JAvin!!! Few answer links are broken so please correct it

July 20, 2016 at 11:31 AM

Unknown said...

Nice post.
@Anonymous : Regarding the scenario on High volume trading application , Your answer on Queue seems to be convincing but not sure if it is the best way.However java executor framework provides newSingleThreadExecutor() which is commonly used to handle the UI events one after the other.This probably can assure you that the order is placed first and then the cancel request is also processed without any exception.
correct me if I'm wrong.

thanks

September 5, 2016 at 10:25 AM

Anonymous said...

Your answer on Queue seems to be convincing but not sure if it is the best way.However java executor framework provides newSingleThreadExecutor() which is commonly used to handle the UI events one after the other.This probably can assure you that the order is placed first and then the cancel request is also processed without any exception.
correct me if I'm wrong

October 9, 2016 at 9:42 AM

Unknown said...

Great work

December 11, 2016 at 12:32 PM

Anonymous said...

Very good collection of multi threading questions, keep it up.

January 31, 2017 at 8:59 AM

Unknown said...

Greate work!!! keeo it up...

February 21, 2017 at 9:14 AM

Anonymous said...

Excellent work

March 10, 2017 at 1:53 AM

Anonymous said...

Answer for question 37 is not clear. It says:

"37) What happens if you submit a task when the queue of the thread pool is already filled? (answer)
This is another tricky question on my list. Many programmers will think that it will block until a task is cleared but its true. ThreadPoolExecutor's submit() method throws RejectedExecutionException if the task cannot be scheduled for execution."

Will it block or it will throw exception?

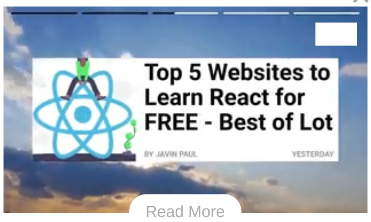
May 2, 2017 at 3:07 AM

javin paul said...

@Anonymous, it will throw exception, the RejectedExecutionException. No blocking.

May 2, 2017 at 4:58 AM

Anonymous said...



One person asked me that, If he wants to download hundreds or thousands of images from server, using java Treads that how will he do that?
How don't know, please suggest.

April 23, 2018 at 10:49 AM

Anonymous said...

If there are total 5 threads and only 3 threads should execute to complete tasks, how can we do? Could you please guide me.

December 12, 2018 at 10:12 AM

naveen said...

For question 37, there are 2 possibilities depending on which type of queue you use.
1. unbounded queue(like ArrayBlockingQueue) - if this queue is used then RejectedExecutionException is thrown
2. bounded queue(like LinkedBlockingDeque) - if this queue is used the no exception is thrown , but eventually system might face resource issues

January 3, 2019 at 12:47 AM

javin paul said...

@Naveen, well said, thanks for your input.

January 3, 2019 at 3:46 AM

Piyush said...

For question 37, I'm trying it out as such, but it still doesn't throw any exception. Am I missing something?

```
BlockingQueue workQueue = new ArrayBlockingQueue(3);
ExecutorService threadExecutor = new ThreadPoolExecutor(3, 5, 1, TimeUnit.MINUTES, workQueue,
Executors.defaultThreadFactory());
for(int i=0; i<5; i++) {
threadExecutor.submit(new ThreadExecutorTest());
}
threadExecutor.shutdown();
```

January 12, 2019 at 6:20 AM

Thirumalai said...

excellent and very important questions for experienced.good job.

August 15, 2019 at 5:32 PM

卡农 - Carolina Farmer, CFA, FRM said...

For 37):
https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executors.html#newFixedThreadPool-int-
" If additional tasks are submitted when all threads are active, they will wait in the queue until a thread is available."

https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html
Rejected tasks
New tasks submitted in method execute(Runnable) will be rejected when the Executor has been shut down, and also when the Executor uses finite bounds for both maximum threads and work queue capacity, and is saturated. In either case, the execute method invokes the RejectedExecutionHandler.rejectedExecution(Runnable, ThreadPoolExecutor) method of its RejectedExecutionHandler. Four predefined handler policies are provided:
In the default ThreadPoolExecutor.AbortPolicy, the handler throws a runtime RejectedExecutionException upon rejection.
In ThreadPoolExecutor.CallerRunsPolicy, the thread that invokes execute itself runs the task. This provides a simple feedback control mechanism that will slow down the rate that new tasks are submitted.
In ThreadPoolExecutor.DiscardPolicy, a task that cannot be executed is simply dropped.
In ThreadPoolExecutor.DiscardOldestPolicy, if the executor is not shut down, the task at the head of the work queue is dropped, and then execution is retried (which can fail again, causing this to be repeated.)
It is possible to define and use other kinds of RejectedExecutionHandler classes. Doing so requires some care especially when policies are designed to work only under particular capacity or queuing policies.

October 13, 2019 at 5:22 PM

Post a Comment

Enter your comment...

Comment as: kum.aniket@gr Sign out

Publish Preview Notify me

Newer Post Home Older Post

Read More

Subscribe to: Post Comments (Atom)

Search This Blog

h