

76% completed

Search Course

Multithreading in Java

Atomic Assignments

Thread Safety & Synchronized

Wait & Notify

Interrupting Threads

Volatile

Reentrant Locks & Condition Variables

Missed Signals

Semaphore in Java

Spurious Wakeups

Atomic Classes

More on Atomics

Non-blocking Synchronization

Miscellaneous Topics

Java's Memory Model

Interview Practice Problems

Practice Mock Interview

Java Multithreading for Senior Engineering Interviews / ... / Miscellaneous Topics

Miscellaneous Topics

Lock Fairness

We'll briefly touch on the topic of fairness in locks since its out of scope for this course. When locks get acquired by threads, there's no guarantee of the order in which threads are granted access to a lock. A thread requesting lock access more frequently may be able to acquire the lock unfairly greater number of times than other locks. Java locks can be turned into fair locks by passing in the fair constructor parameter. However, fair locks exhibit lower throughput and are slower compared to their unfair counterparts.

Thread Pools

Imagine an application that creates threads to undertake short-lived tasks. The application would incur a performance penalty for first creating hundreds of threads and then tearing down the allocated resources for each thread at the ends of its life. The general way programming frameworks solve this problem is by creating a pool of threads, which are handed out to execute each concurrent task and once completed, the thread is returned to the pool

Java offers thread pools via its **Executor Framework**. The framework includes classes such as the `ThreadPoolExecutor` for creating thread pools.

← Back lesson

Completed

Next →

Non-blocking Synchronization

Memory Model