

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

- More on Atomics
- Non-blocking Synchronization
- Miscellaneous Topics

Java's Memory Model

Interview Practice Problems

Practice Mock Interview

Java Multithreading for Senior Engineering Interviews / ... / Missed Signals

Missed Signals

A missed signal happens when a signal is sent by a thread before the other thread starts waiting on a condition. This is exemplified by the following code snippet. Missed signals are caused by using the wrong concurrency constructs. In the example below, a condition variable is used to coordinate between the **signaller** and the **waiter** thread. The condition is signaled at a time when no thread is waiting on it causing a missed signal.

In later sections, you'll learn that the way we are using the condition variable's `await` method is incorrect. The idiomatic way of using `await` is in a while loop with an associated boolean condition. For now, observe the possibility of losing signals between threads.

Java

main.java

```
1- import java.util.concurrent.locks.Condition;
2- import java.util.concurrent.locks.ReentrantLock;
3-
4- class Demonstration {
5-
6-     public static void main(String args[]) throws InterruptedException {
7-         MissedSignalExample.example();
8-     }
9- }
10-
11- class MissedSignalExample {
```

Java

main.java

```
1- import java.util.concurrent.locks.Condition;
2- import java.util.concurrent.locks.ReentrantLock;
3-
4- class Demonstration {
5-
6-     public static void main(String args[]) throws InterruptedException {
7-         MissedSignalExample.example();
8-     }
9- }
10-
11- class MissedSignalExample {
12-
```

Missed Signal Example

The above code when ran, will never print the statement `Program Exiting` and execution would time out. Apart from refactoring the code to match the idiomatic usage of condition variables in a while loop, the other possible fix is to use a **semaphore** for signalling between the two threads as shown below

Java

main.java

```
1- import java.util.concurrent.Semaphore;
2-
3- class Demonstration {
4-
5-     public static void main(String args[]) throws InterruptedException {
6-         FixedMissedSignalExample.example();
7-     }
8- }
9-
10- class FixedMissedSignalExample {
11-
12-     public static void example() throws InterruptedException {
```

Fixed Missed Signal

Back lesson

CompletedNext

Reentrant Locks & Condition Variables

Semaphore in Java