

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.

```
1 import java.util.concurrent.locks.ReentrantLock;
2 import java.util.concurrent.locks.Condition;
3
4 class Demonstration {
5
6     public static void main(String[] args) {
7         MissedSignalExample.example();
8     }
9 }
10
11 class MissedSignalExample {
12
13     public static void example() {
14
15         final ReentrantLock lock = new ReentrantLock();
16         final Condition condition = lock.newCondition();
17
18         Thread signaller = new Thread() {
19
20             public void run() {
21                 lock.lock();
22                 condition.signal();
23             }
24         };
25
26         Thread waiter = new Thread() {
27
28             public void run() {
29                 lock.lock();
30                 condition.await();
31             }
32         };
33
34         signaller.start();
35         waiter.start();
36     }
37 }
```



```

23         System.out.println("Program Exiting");
24         lock.unlock();
25     }
26 });
27
28 Thread waiter = new Thread() {
29     @Override
30     public void run() {
31         while (true) {

```



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

```

1  import java.util.concurrent.Semaphore;
2
3  class Demonstration {
4
5      public static void main(String[] args) {
6          FixedMissedSignalExample example = new FixedMissedSignalExample();
7      }
8  }
9
10 class FixedMissedSignalExample {
11
12     public static void example() {
13
14         final Semaphore semaphore = new Semaphore(1);
15
16         Thread signaller = new Thread() {
17
18             public void run() {
19                 semaphore.release();
20                 System.out.println("Signaller");
21             }
22         });
23
24         Thread waiter = new Thread() {
25
26             public void run() {
27                 try {
28                     semaphore.acquire();
29                     System.out.println("Waiter");
30                 } catch (InterruptedException e) {
31                     // handle

```





Fixed Missed Signal