

This site uses cookies to improve the user experience.

TRADE ONLINE

FX

Forex

Indices

Gold

Oil

USAIN BOLT

OFFICIAL SPONSOR

XM

MEMBER OF TRADING POINT GROUP

*T&Cs apply.

Your capital is at risk.

GET £20 TRADING BONUS*

Java Concurrency

1. [Java Concurrency / Multithreading Tutorial](#)
2. [Multithreading Benefits](#)
3. [Multithreading Costs](#)
4. [Concurrency Models](#)
5. [Same-threading](#)
6. [Concurrency vs. Parallelism](#)
7. [Creating and Starting Java Threads](#)
8. [Race Conditions and Critical Sections](#)
9. [Thread Safety and Shared Resources](#)
10. [Thread Safety and Immutability](#)
11. [Java Memory Model](#)
12. [Java Synchronized Blocks](#)
13. [Java Volatile Keyword](#)
14. [Java ThreadLocal](#)
15. [Thread Signaling](#)
16. [Deadlock](#)
17. [Deadlock Prevention](#)
18. [Starvation and Fairness](#)
19. [Nested Monitor Lockout](#)
20. [Slipped Conditions](#)
21. [Locks in Java](#)
22. [Read / Write Locks in Java](#)
23. [Reentrance Lockout](#)
24. [Semaphores](#)
25. **[Blocking Queues](#)**
26. [Thread Pools](#)
27. [Compare and Swap](#)
28. [Anatomy of a Synchronizer](#)
29. [Non-blocking Algorithms](#)
30. [Amdahl's Law](#)
31. [Java Concurrency References](#)

Blocking Queues

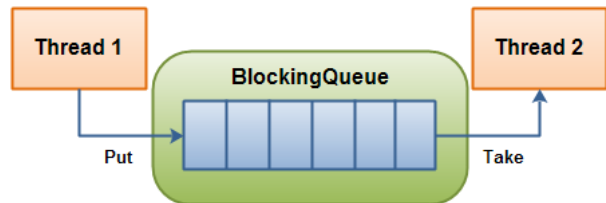


Jakob Jenkov
Last update: 2014-06-23



A blocking queue is a queue that blocks when you try to dequeue from it and the queue is empty or you try to enqueue items to it and the queue is already full. A thread trying to dequeue from an empty queue is blocked until some other thread inserts an item into the queue. A thread trying to enqueue an item in a full queue is blocked until some other thread makes space in the queue, either by dequeuing one or more items or clearing the queue completely.

Here is a diagram showing two threads cooperating via a blocking queue:



A BlockingQueue with one thread putting into it, and another thread taking from it

Java 5 comes with blocking queue implementations in the `java.util.concurrent` package. You read about that class in my [java.util.concurrent.BlockingQueue](#) tutorial. Even if Java 5 comes with a blocking queue implementation, it can be useful to know the theory behind their implementation.

Blocking Queue Implementation

The implementation of a blocking queue looks similar to a [Bounded Semaphore](#). Here is a simple implementation of a blocking queue:

```
public class BlockingQueue {  
  
    private List queue = new LinkedList();  
    private int limit = 10;  
  
    public BlockingQueue(int limit){  
        this.limit = limit;  
    }  
  
    public synchronized void enqueue(Object item)  
        throws InterruptedException {  
        while(this.queue.size() == this.limit) {  
            wait();  
        }  
        if(this.queue.size() == 0) {  
            notifyAll();  
        }  
        this.queue.add(item);  
    }  
  
    public synchronized Object dequeue()  
        throws InterruptedException{  
        while(this.queue.size() == 0){  
            wait();  
        }  
        if(this.queue.size() == this.limit){  
            notifyAll();  
        }  
  
        return this.queue.remove(0);  
    }  
}
```



Notice how `notifyAll()` is only called from `enqueue()` and `dequeue()` if the queue size is equal size bounds (0 or limit). If the queue size is not equal to either bound when `enqueue()` or `dequeue()` called, there can be no threads waiting to either enqueue or dequeue items.

Next: [Thread Pools](#)



Tweet



Jakob Jenkov



All Trails

Trail TOC

Page TOC

Previous

Next

BLACK FRIDAY

SALE: RELOADED
26 & 27 November

New deals every hour

SEE THE DEALS

amazon

Terms and conditions apply.

9 Comments [tutorials.jenkov.com](#)

Recommend 2 Share



Join the discussion...



Amandeep · a year ago

I guess `notifyAll` statement should be written in the last as shown below:-

(Why: say suppose we assume above implementation is correct, and if we talk about `enqueue` method when queue is empty where we are firstly notify other threads (for consumer).

But say suppose, consumer come into execution all of a sudden because of this `notifyAll()` and consumer went into spin lock waiting because size is 0. And now our producer thread again comes into execution and we added first element in queue and it terminates now. Consumer will keep waiting till producer calls `notify`, so this should be corrected as follows)

```
public synchronized void enqueue(Object item)
throws InterruptedException {
while(this.queue.size() == this.limit) {
wait();
}

```

```
this.queue.add(item);

```

```
if(this.queue.size() == 1) {
notifyAll();
}

```

[see more](#)

^ v · Reply · Share



Jakob Jenkov Mod → Amandeep · a year ago

There will never be two threads executing inside `enqueue()`, `dequeue()` or one in each. If a thread wakes a waiting thread, the awakened thread gets to execute until the first thread has exited the `enqueue()` / `dequeue()` method. An awakened thread needs to get the lock of the `BlockingQueue` instance after being awakened, and that can only happen if the thread owning the lock leaves the synchronized method that took the lock.

1 ^ v · Reply · Share



pymd · a year ago

Really awesome tutorial! Thanks a lot for this.

I've doubt. Suppose that I use the above blocking queue and its methods as it is, won't it result in a deadlock once the queue is empty and another method acquires the lock? I mean, once a thread is inside "dequeue" of an object, no other thread can be inside "enqueue" right? In such a case, v

method acquires the lock: I mean, once a thread is inside `dequeue()` of an object, no other thread can be inside `enqueue()`, right? In such a case, it is in a deadlock where the program is waiting for a task to be written to queue, but no task can be written as the lock is held by the "waiting" process?

^ | v • Reply • Share ›



Jakob Jenkov Mod → pymd • a year ago

The call to `object.wait()` releases the lock on the object (the queue) so other threads can call the `enqueue()` method.

^ | v • Reply • Share ›



planeptdisqus • a year ago

```
if(this.queue.size() == this.limit){
    notifyAll();
}
```

This wakes up a waiting enqueue thread. If the element has been dequeued, all is good, but if the thread wakes up before the element is removed, the queue is still at limit.

All Trails

Trail TOC

Page TOC

Previous

Next

Then `wait()` again.

^ | v • Reply • Share ›



Jakob Jenkov Mod → planeptdisqus • a year ago

The enqueueing thread cannot exit the `wait()` call until the dequeuing thread leaves the `dequeue()` method. Thus, the enqueueing (waiting) thread is able to execute the while loop until the element is actually dequeued. Awakening a thread does not give it the lock of the object. It just wakes it up, making it ready to enter the synchronized block when possible.

^ | v • Reply • Share ›



Adrian • a year ago

Oops. Forgot that it must be while not if. I stand corrected. Thank you for pointing that out.

^ | v • Reply • Share ›



Adrian • a year ago

This example is great and I have been playing with it. However, I think you should add a check on queue first if its empty before removing element because another thread might have emptied the queue already. I think the return statement of `dequeue` method should be `return (queue.isEmpty()) ? queue.remove(0)`.

^ | v • Reply • Share ›



Jakob Jenkov Mod → Adrian • a year ago

The `dequeue()` is synchronized, so once a thread leaves the

```
while(this.queue.size() == 0) { ... }
```

There is at least 1 element in the queue, and no other thread can take it (because the thread owns the lock on the `dequeue()` method).

Therefore it is not necessary with another check to see if the queue is empty.

1 ^ | v • Reply • Share ›

ALSO ON TUTORIALS.JENKOV.COM

Boon - ObjectMapper

2 comments • a year ago •



Jakob Jenkov — You should do your own measurements to make sure.

Android Toast

1 comment • 2 years ago •



BASAM SRERAM — very nice tutorial..

Vert.x Verticles

2 comments • a year ago •



Jakob Jenkov — The `EventBusReceiverVerticle` extends `AbstractVerticle`. When an `AbstractVerticle` is deployed to the Vert.x instance, the `vert.x` instance automatically sets it on it.

Java Memory Model

10 comments • a year ago •



Jakob Jenkov — 1) You need to know from the manufacturer of your hardware. Most likely it contains a single CPU with multiple cores (mini-CPU's). Each core can only run a single thread at a time, but can switch very fast between threads.

✉ Subscribe • ➕ Add Disqus to your site Add Disqus Add • 🔒 Privacy