

This site uses cookies to improve the user experience.



## java.util.concurrent - Java Concurrency Utilities

1. [java.util.concurrent - Java Concurrency Utilities](#)
2. **BlockingQueue**
3. [ArrayBlockingQueue](#)
4. [DelayQueue](#)
5. [LinkedBlockingQueue](#)
6. [PriorityBlockingQueue](#)
7. [SynchronousQueue](#)
8. [BlockingDeque](#)
9. [LinkedBlockingDeque](#)
10. [ConcurrentMap](#)
11. [ConcurrentNavigableMap](#)
12. [CountDownLatch](#)
13. [CyclicBarrier](#)
14. [Exchanger](#)
15. [Semaphore](#)
16. [ExecutorService](#)
17. [ThreadPoolExecutor](#)
18. [ScheduledExecutorService](#)
19. [Java Fork and Join using ForkJoinPool](#)
20. [Lock](#)
21. [ReadWriteLock](#)
22. [AtomicBoolean](#)
23. [AtomicInteger](#)
24. [AtomicLong](#)
25. [AtomicReference](#)
26. [AtomicStampedReference](#)
27. [AtomicIntegerArray](#)
28. [AtomicLongArray](#)
29. [AtomicReferenceArray](#)

## BlockingQueue

- [BlockingQueue Usage](#)
- [BlockingQueue Implementations](#)
- [Java BlockingQueue Example](#)



Jakob Jenkov  
Last update: 2014-06-23

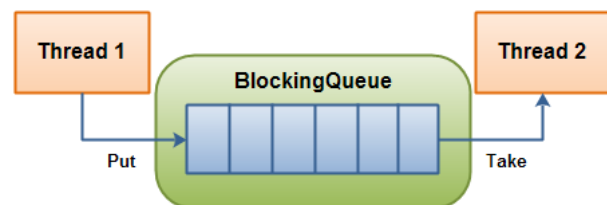


The Java `BlockingQueue` interface in the `java.util.concurrent` package represents a queue with thread safe to put into, and take instances from. In this text I will show you how to use this `BlockingQueue`.

This text will not discuss how to implement a `BlockingQueue` in Java yourself. If you are interested that, I have a text on [Blocking Queues](#) in my more theoretical [Java Concurrency Tutorial](#).

### BlockingQueue Usage

A `BlockingQueue` is typically used to have one thread produce objects, which another thread consumes. Here is a diagram that illustrates this principle:



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

The producing thread will keep producing new objects and insert them into the queue, until the queue reaches some upper bound on what it can contain. It's limit, in other words. If the blocking queue reaches its upper limit, the producing thread is blocked while trying to insert the new object. It remains blocked until a consuming thread takes an object out of the queue.

The consuming thread keeps taking objects out of the blocking queue, and processes them. If the consuming thread tries to take an object out of an empty queue, the consuming thread is blocked until a producing thread puts an object into the queue.

### BlockingQueue Methods

A `BlockingQueue` has 4 different sets of methods for inserting, removing and examining the elements of the queue. Each set of methods behaves differently in case the requested operation cannot be carried out immediately. Here is a table of the methods:

	Throws Exception	Special Value	Blocks	Times Out
<b>Insert</b>	<code>add(o)</code>	<code>offer(o)</code>	<code>put(o)</code>	<code>offer(o, timeout, timeunit)</code>
<b>Remove</b>	<code>remove(o)</code>	<code>poll()</code>	<code>take()</code>	<code>poll(timeout, timeunit)</code>
<b>Examine</b>	<code>element()</code>	<code>peek()</code>		

The 4 different sets of behaviour means this:

1. **Throws Exception:**  
If the attempted operation is not possible immediately, an exception is thrown.
2. **Special Value:**  
If the attempted operation is not possible immediately, a special value is returned (often `true` or `false`).

**3. Blocks:**

If the attempted operation is not possible immediately, the method call blocks until it is.

**4. Times Out:**

If the attempted operation is not possible immediately, the method call blocks until it is, but no longer than the given timeout. Returns a special value telling whether the operation succeeded or not (typically true / false).

It is not possible to insert null into a `BlockingQueue`. If you try to insert null, the `BlockingQueue` throws a `NullPointerException`.

It is also possible to access all the elements inside a `BlockingQueue`, and not just the elements at the start and end. For instance, say you have queued an object for processing, but your application wants to cancel it. You can then call e.g. `remove(o)` to remove a specific object in the queue. However, this is not done very efficiently, so you should not use these `Collection` methods unless you really have to.

## BlockingQueue Implementations

Since `BlockingQueue` is an interface, you need to use one of its implementations to use it. The

[All Trails](#)
[Trail TOC](#)
[Page TOC](#)
[Previous](#)
[Next](#)

- [ArrayBlockingQueue](#)
- [DelayQueue](#)
- [LinkedBlockingQueue](#)
- [PriorityBlockingQueue](#)
- [SynchronousQueue](#)

Click the links in the list to read more about each implementation. If a link cannot be clicked, that implementation has not yet been described. Check back again in the future, or check out the Java documentation for more detail.

## Java BlockingQueue Example

Here is a Java `BlockingQueue` example. The example uses the `ArrayBlockingQueue` implementation of the `BlockingQueue` interface.

First, the `BlockingQueueExample` class which starts a `Producer` and a `Consumer` in separate threads. The `Producer` inserts strings into a shared `BlockingQueue`, and the `Consumer` takes them out.

```
public class BlockingQueueExample {

    public static void main(String[] args) throws Exception {

        BlockingQueue queue = new ArrayBlockingQueue(1024);

        Producer producer = new Producer(queue);
        Consumer consumer = new Consumer(queue);

        new Thread(producer).start();
        new Thread(consumer).start();

        Thread.sleep(4000);
    }
}
```

Here is the `Producer` class. Notice how it sleeps a second between each `put()` call. This will cause the consumer to block, while waiting for objects in the queue.

```
public class Producer implements Runnable{

    protected BlockingQueue queue = null;

    public Producer(BlockingQueue queue) {
        this.queue = queue;
    }

    public void run() {
        try {
            queue.put("1");
            Thread.sleep(1000);
            queue.put("2");
            Thread.sleep(1000);
            queue.put("3");
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Here is the `Consumer` class. It just takes out the objects from the queue, and prints them to the system.

```
public class Consumer implements Runnable{

    protected BlockingQueue queue = null;

    public Consumer(BlockingQueue queue) {
        this.queue = queue;
    }

    public void run() {
        try {
```

```
        System.out.println(queue.take());
        System.out.println(queue.take());
        System.out.println(queue.take());
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

Next: [ArrayBlockingQueue](#)

Share

Tweet

Jakob Jenkov

All Trails	Trail TOC	Page TOC	Previous	Next
------------	-----------	----------	----------	------

DELTA

NEED A LITTLE LIFT?

BOOK NOW >

2 Comments   tutorials.jenkov.com

Recommend 15   Share

Join the discussion...

- Mithil Shah** • a year ago  
Hi, I have one query, Are Multiple producers thread (for example 2 threads ) and single consumer thread scenario possible for LinkedBlockingDeque  
 • Reply • Share ›
- Jakob Jenkov** Mod → Mithil Shah • a year ago  
It should be, as far as I know. LinkedBlockingDeque is a concurrent data structure, so it should allow for multiple producers and consumers. in mind that multiple readers / writers makes a concurrent data structure slower. Actually, mostly multiple writers.  
 • Reply • Share ›

ALSO ON TUTORIALS.JENKOV.COM

**Gradle Dependency Management**

1 comment • a year ago•

Alps — Thanks for Gradle tutorial.It's really helpful

**Boon - ObjectMapper**

2 comments • a year ago•

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

**Java Exercises**

2 comments • 2 years ago•

Liguo Jia — Thanks a lot Jenkov. I like these articles very much, sho lot content. I have read most of your articles about Java, and learne idea about writing is great, I'll follow it.

**Android Toast**

1 comment • 2 years ago•

BASAM SRERAM — very nice tutorial..

Subscribe   Add Disqus to your site Add Disqus Add   Privacy