| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/BlockingQueue.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/util/concurrent/BlockingDeque.html)   [**NEXT CLASS**](http://docs.google.com/java/util/concurrent/BrokenBarrierException.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/concurrent/BlockingQueue.html)    [**NO FRAMES**](http://docs.google.com/BlockingQueue.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#3znysh7) | DETAIL: FIELD | CONSTR | [METHOD](#3dy6vkm) |

## **java.util.concurrent**

Interface BlockingQueue<E>

**Type Parameters:**E - the type of elements held in this collection **All Superinterfaces:** [Collection](http://docs.google.com/java/util/Collection.html)<E>, [Iterable](http://docs.google.com/java/lang/Iterable.html)<E>, [Queue](http://docs.google.com/java/util/Queue.html)<E> **All Known Subinterfaces:** [BlockingDeque](http://docs.google.com/java/util/concurrent/BlockingDeque.html)<E> **All Known Implementing Classes:** [ArrayBlockingQueue](http://docs.google.com/java/util/concurrent/ArrayBlockingQueue.html), [DelayQueue](http://docs.google.com/java/util/concurrent/DelayQueue.html), [LinkedBlockingDeque](http://docs.google.com/java/util/concurrent/LinkedBlockingDeque.html), [LinkedBlockingQueue](http://docs.google.com/java/util/concurrent/LinkedBlockingQueue.html), [PriorityBlockingQueue](http://docs.google.com/java/util/concurrent/PriorityBlockingQueue.html), [SynchronousQueue](http://docs.google.com/java/util/concurrent/SynchronousQueue.html)

public interface **BlockingQueue<E>**extends [Queue](http://docs.google.com/java/util/Queue.html)<E>

A [Queue](http://docs.google.com/java/util/Queue.html) that additionally supports operations that wait for the queue to become non-empty when retrieving an element, and wait for space to become available in the queue when storing an element.

BlockingQueue methods come in four forms, with different ways of handling operations that cannot be satisfied immediately, but may be satisfied at some point in the future: one throws an exception, the second returns a special value (either null or false, depending on the operation), the third blocks the current thread indefinitely until the operation can succeed, and the fourth blocks for only a given maximum time limit before giving up. These methods are summarized in the following table:

|  | *Throws exception* | *Special value* | *Blocks* | *Times out* |
| --- | --- | --- | --- | --- |
| **Insert** | [add(e)](http://docs.google.com/java/util/concurrent/BlockingQueue.html#add(E)) | [offer(e)](http://docs.google.com/java/util/concurrent/BlockingQueue.html#offer(E)) | [put(e)](http://docs.google.com/java/util/concurrent/BlockingQueue.html#put(E)) | [offer(e, time, unit)](http://docs.google.com/java/util/concurrent/BlockingQueue.html#offer(E,%20long,%20java.util.concurrent.TimeUnit)) |
| **Remove** | [remove()](http://docs.google.com/java/util/concurrent/BlockingQueue.html#remove(java.lang.Object)) | [poll()](http://docs.google.com/java/util/concurrent/BlockingQueue.html#poll(long,%20java.util.concurrent.TimeUnit)) | [take()](http://docs.google.com/java/util/concurrent/BlockingQueue.html#take()) | [poll(time, unit)](http://docs.google.com/java/util/concurrent/BlockingQueue.html#poll(long,%20java.util.concurrent.TimeUnit)) |
| **Examine** | [element()](http://docs.google.com/java/util/Queue.html#element()) | [peek()](http://docs.google.com/java/util/Queue.html#peek()) | *not applicable* | *not applicable* |

A BlockingQueue does not accept null elements. Implementations throw NullPointerException on attempts to add, put or offer a null. A null is used as a sentinel value to indicate failure of poll operations.

A BlockingQueue may be capacity bounded. At any given time it may have a remainingCapacity beyond which no additional elements can be put without blocking. A BlockingQueue without any intrinsic capacity constraints always reports a remaining capacity of Integer.MAX\_VALUE.

BlockingQueue implementations are designed to be used primarily for producer-consumer queues, but additionally support the [Collection](http://docs.google.com/java/util/Collection.html) interface. So, for example, it is possible to remove an arbitrary element from a queue using remove(x). However, such operations are in general *not* performed very efficiently, and are intended for only occasional use, such as when a queued message is cancelled.

BlockingQueue implementations are thread-safe. All queuing methods achieve their effects atomically using internal locks or other forms of concurrency control. However, the *bulk* Collection operations addAll, containsAll, retainAll and removeAll are *not* necessarily performed atomically unless specified otherwise in an implementation. So it is possible, for example, for addAll(c) to fail (throwing an exception) after adding only some of the elements in c.

A BlockingQueue does *not* intrinsically support any kind of "close" or "shutdown" operation to indicate that no more items will be added. The needs and usage of such features tend to be implementation-dependent. For example, a common tactic is for producers to insert special *end-of-stream* or *poison* objects, that are interpreted accordingly when taken by consumers.

Usage example, based on a typical producer-consumer scenario. Note that a BlockingQueue can safely be used with multiple producers and multiple consumers.

class Producer implements Runnable {  
 private final BlockingQueue queue;  
 Producer(BlockingQueue q) { queue = q; }  
 public void run() {  
 try {  
 while (true) { queue.put(produce()); }  
 } catch (InterruptedException ex) { ... handle ...}  
 }  
 Object produce() { ... }  
 }  
  
 class Consumer implements Runnable {  
 private final BlockingQueue queue;  
 Consumer(BlockingQueue q) { queue = q; }  
 public void run() {  
 try {  
 while (true) { consume(queue.take()); }  
 } catch (InterruptedException ex) { ... handle ...}  
 }  
 void consume(Object x) { ... }  
 }  
  
 class Setup {  
 void main() {  
 BlockingQueue q = new SomeQueueImplementation();  
 Producer p = new Producer(q);  
 Consumer c1 = new Consumer(q);  
 Consumer c2 = new Consumer(q);  
 new Thread(p).start();  
 new Thread(c1).start();  
 new Thread(c2).start();  
 }  
 }

Memory consistency effects: As with other concurrent collections, actions in a thread prior to placing an object into a BlockingQueue [*happen-before*](http://docs.google.com/package-summary.html#MemoryVisibility) actions subsequent to the access or removal of that element from the BlockingQueue in another thread.

This interface is a member of the  [Java Collections Framework](http://docs.google.com/technotes/guides/collections/index.html).

**Since:** 1.5

| **Method Summary** | |
| --- | --- |
| boolean | [**add**](http://docs.google.com/java/util/concurrent/BlockingQueue.html#add(E))([E](http://docs.google.com/java/util/concurrent/BlockingQueue.html) e)            Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning true upon success and throwing an IllegalStateException if no space is currently available. |
| boolean | [**contains**](http://docs.google.com/java/util/concurrent/BlockingQueue.html#contains(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) o)            Returns true if this queue contains the specified element. |
| int | [**drainTo**](http://docs.google.com/java/util/concurrent/BlockingQueue.html#drainTo(java.util.Collection))([Collection](http://docs.google.com/java/util/Collection.html)<? super [E](http://docs.google.com/java/util/concurrent/BlockingQueue.html)> c)            Removes all available elements from this queue and adds them to the given collection. |
| int | [**drainTo**](http://docs.google.com/java/util/concurrent/BlockingQueue.html#drainTo(java.util.Collection,%20int))([Collection](http://docs.google.com/java/util/Collection.html)<? super [E](http://docs.google.com/java/util/concurrent/BlockingQueue.html)> c, int maxElements)            Removes at most the given number of available elements from this queue and adds them to the given collection. |
| boolean | [**offer**](http://docs.google.com/java/util/concurrent/BlockingQueue.html#offer(E))([E](http://docs.google.com/java/util/concurrent/BlockingQueue.html) e)            Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning true upon success and false if no space is currently available. |
| boolean | [**offer**](http://docs.google.com/java/util/concurrent/BlockingQueue.html#offer(E,%20long,%20java.util.concurrent.TimeUnit))([E](http://docs.google.com/java/util/concurrent/BlockingQueue.html) e, long timeout, [TimeUnit](http://docs.google.com/java/util/concurrent/TimeUnit.html) unit)            Inserts the specified element into this queue, waiting up to the specified wait time if necessary for space to become available. |
| [E](http://docs.google.com/java/util/concurrent/BlockingQueue.html) | [**poll**](http://docs.google.com/java/util/concurrent/BlockingQueue.html#poll(long,%20java.util.concurrent.TimeUnit))(long timeout, [TimeUnit](http://docs.google.com/java/util/concurrent/TimeUnit.html) unit)            Retrieves and removes the head of this queue, waiting up to the specified wait time if necessary for an element to become available. |
| void | [**put**](http://docs.google.com/java/util/concurrent/BlockingQueue.html#put(E))([E](http://docs.google.com/java/util/concurrent/BlockingQueue.html) e)            Inserts the specified element into this queue, waiting if necessary for space to become available. |
| int | [**remainingCapacity**](http://docs.google.com/java/util/concurrent/BlockingQueue.html#remainingCapacity())()            Returns the number of additional elements that this queue can ideally (in the absence of memory or resource constraints) accept without blocking, or Integer.MAX\_VALUE if there is no intrinsic limit. |
| boolean | [**remove**](http://docs.google.com/java/util/concurrent/BlockingQueue.html#remove(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) o)            Removes a single instance of the specified element from this queue, if it is present. |
| [E](http://docs.google.com/java/util/concurrent/BlockingQueue.html) | [**take**](http://docs.google.com/java/util/concurrent/BlockingQueue.html#take())()            Retrieves and removes the head of this queue, waiting if necessary until an element becomes available. |

| **Methods inherited from interface java.util.**[**Queue**](http://docs.google.com/java/util/Queue.html) |
| --- |
| [element](http://docs.google.com/java/util/Queue.html#element()), [peek](http://docs.google.com/java/util/Queue.html#peek()), [poll](http://docs.google.com/java/util/Queue.html#poll()), [remove](http://docs.google.com/java/util/Queue.html#remove()) |

| **Methods inherited from interface java.util.**[**Collection**](http://docs.google.com/java/util/Collection.html) |
| --- |
| [addAll](http://docs.google.com/java/util/Collection.html#addAll(java.util.Collection)), [clear](http://docs.google.com/java/util/Collection.html#clear()), [containsAll](http://docs.google.com/java/util/Collection.html#containsAll(java.util.Collection)), [equals](http://docs.google.com/java/util/Collection.html#equals(java.lang.Object)), [hashCode](http://docs.google.com/java/util/Collection.html#hashCode()), [isEmpty](http://docs.google.com/java/util/Collection.html#isEmpty()), [iterator](http://docs.google.com/java/util/Collection.html#iterator()), [removeAll](http://docs.google.com/java/util/Collection.html#removeAll(java.util.Collection)), [retainAll](http://docs.google.com/java/util/Collection.html#retainAll(java.util.Collection)), [size](http://docs.google.com/java/util/Collection.html#size()), [toArray](http://docs.google.com/java/util/Collection.html#toArray()), [toArray](http://docs.google.com/java/util/Collection.html#toArray(T%5B%5D)) |

| **Method Detail** |
| --- |

### add

boolean **add**([E](http://docs.google.com/java/util/concurrent/BlockingQueue.html) e)

Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning true upon success and throwing an IllegalStateException if no space is currently available. When using a capacity-restricted queue, it is generally preferable to use [offer](http://docs.google.com/java/util/concurrent/BlockingQueue.html#offer(E)).

**Specified by:**[add](http://docs.google.com/java/util/Collection.html#add(E)) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/concurrent/BlockingQueue.html)>**Specified by:**[add](http://docs.google.com/java/util/Queue.html#add(E)) in interface [Queue](http://docs.google.com/java/util/Queue.html)<[E](http://docs.google.com/java/util/concurrent/BlockingQueue.html)> **Parameters:**e - the element to add **Returns:**true (as specified by [Collection.add(E)](http://docs.google.com/java/util/Collection.html#add(E))) **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if the element cannot be added at this time due to capacity restrictions [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the class of the specified element prevents it from being added to this queue [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified element is null [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if some property of the specified element prevents it from being added to this queue

### offer

boolean **offer**([E](http://docs.google.com/java/util/concurrent/BlockingQueue.html) e)

Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning true upon success and false if no space is currently available. When using a capacity-restricted queue, this method is generally preferable to [add(E)](http://docs.google.com/java/util/concurrent/BlockingQueue.html#add(E)), which can fail to insert an element only by throwing an exception.

**Specified by:**[offer](http://docs.google.com/java/util/Queue.html#offer(E)) in interface [Queue](http://docs.google.com/java/util/Queue.html)<[E](http://docs.google.com/java/util/concurrent/BlockingQueue.html)> **Parameters:**e - the element to add **Returns:**true if the element was added to this queue, else false **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the class of the specified element prevents it from being added to this queue [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified element is null [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if some property of the specified element prevents it from being added to this queue

### put

void **put**([E](http://docs.google.com/java/util/concurrent/BlockingQueue.html) e)  
 throws [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html)

Inserts the specified element into this queue, waiting if necessary for space to become available.

**Parameters:**e - the element to add **Throws:** [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html) - if interrupted while waiting [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the class of the specified element prevents it from being added to this queue [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified element is null [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if some property of the specified element prevents it from being added to this queue

### offer

boolean **offer**([E](http://docs.google.com/java/util/concurrent/BlockingQueue.html) e,  
 long timeout,  
 [TimeUnit](http://docs.google.com/java/util/concurrent/TimeUnit.html) unit)  
 throws [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html)

Inserts the specified element into this queue, waiting up to the specified wait time if necessary for space to become available.

**Parameters:**e - the element to addtimeout - how long to wait before giving up, in units of unitunit - a TimeUnit determining how to interpret the timeout parameter **Returns:**true if successful, or false if the specified waiting time elapses before space is available **Throws:** [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html) - if interrupted while waiting [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the class of the specified element prevents it from being added to this queue [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified element is null [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if some property of the specified element prevents it from being added to this queue

### take

[E](http://docs.google.com/java/util/concurrent/BlockingQueue.html) **take**()  
 throws [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html)

Retrieves and removes the head of this queue, waiting if necessary until an element becomes available.

**Returns:**the head of this queue **Throws:** [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html) - if interrupted while waiting

### poll

[E](http://docs.google.com/java/util/concurrent/BlockingQueue.html) **poll**(long timeout,  
 [TimeUnit](http://docs.google.com/java/util/concurrent/TimeUnit.html) unit)  
 throws [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html)

Retrieves and removes the head of this queue, waiting up to the specified wait time if necessary for an element to become available.

**Parameters:**timeout - how long to wait before giving up, in units of unitunit - a TimeUnit determining how to interpret the timeout parameter **Returns:**the head of this queue, or null if the specified waiting time elapses before an element is available **Throws:** [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html) - if interrupted while waiting

### remainingCapacity

int **remainingCapacity**()

Returns the number of additional elements that this queue can ideally (in the absence of memory or resource constraints) accept without blocking, or Integer.MAX\_VALUE if there is no intrinsic limit.

Note that you *cannot* always tell if an attempt to insert an element will succeed by inspecting remainingCapacity because it may be the case that another thread is about to insert or remove an element.

**Returns:**the remaining capacity

### remove

boolean **remove**([Object](http://docs.google.com/java/lang/Object.html) o)

Removes a single instance of the specified element from this queue, if it is present. More formally, removes an element e such that o.equals(e), if this queue contains one or more such elements. Returns true if this queue contained the specified element (or equivalently, if this queue changed as a result of the call).

**Specified by:**[remove](http://docs.google.com/java/util/Collection.html#remove(java.lang.Object)) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/concurrent/BlockingQueue.html)> **Parameters:**o - element to be removed from this queue, if present **Returns:**true if this queue changed as a result of the call **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the class of the specified element is incompatible with this queue (optional) [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified element is null (optional)

### contains

boolean **contains**([Object](http://docs.google.com/java/lang/Object.html) o)

Returns true if this queue contains the specified element. More formally, returns true if and only if this queue contains at least one element e such that o.equals(e).

**Specified by:**[contains](http://docs.google.com/java/util/Collection.html#contains(java.lang.Object)) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/concurrent/BlockingQueue.html)> **Parameters:**o - object to be checked for containment in this queue **Returns:**true if this queue contains the specified element **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the class of the specified element is incompatible with this queue (optional) [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified element is null (optional)

### drainTo

int **drainTo**([Collection](http://docs.google.com/java/util/Collection.html)<? super [E](http://docs.google.com/java/util/concurrent/BlockingQueue.html)> c)

Removes all available elements from this queue and adds them to the given collection. This operation may be more efficient than repeatedly polling this queue. A failure encountered while attempting to add elements to collection c may result in elements being in neither, either or both collections when the associated exception is thrown. Attempts to drain a queue to itself result in IllegalArgumentException. Further, the behavior of this operation is undefined if the specified collection is modified while the operation is in progress.

**Parameters:**c - the collection to transfer elements into **Returns:**the number of elements transferred **Throws:** [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - if addition of elements is not supported by the specified collection [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the class of an element of this queue prevents it from being added to the specified collection [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified collection is null [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if the specified collection is this queue, or some property of an element of this queue prevents it from being added to the specified collection

### drainTo

int **drainTo**([Collection](http://docs.google.com/java/util/Collection.html)<? super [E](http://docs.google.com/java/util/concurrent/BlockingQueue.html)> c,  
 int maxElements)

Removes at most the given number of available elements from this queue and adds them to the given collection. A failure encountered while attempting to add elements to collection c may result in elements being in neither, either or both collections when the associated exception is thrown. Attempts to drain a queue to itself result in IllegalArgumentException. Further, the behavior of this operation is undefined if the specified collection is modified while the operation is in progress.

**Parameters:**c - the collection to transfer elements intomaxElements - the maximum number of elements to transfer **Returns:**the number of elements transferred **Throws:** [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - if addition of elements is not supported by the specified collection [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the class of an element of this queue prevents it from being added to the specified collection [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified collection is null [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if the specified collection is this queue, or some property of an element of this queue prevents it from being added to the specified collection

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/BlockingQueue.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/util/concurrent/BlockingDeque.html)   [**NEXT CLASS**](http://docs.google.com/java/util/concurrent/BrokenBarrierException.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/concurrent/BlockingQueue.html)    [**NO FRAMES**](http://docs.google.com/BlockingQueue.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#3znysh7) | DETAIL: FIELD | CONSTR | [METHOD](#3dy6vkm) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)

For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).