| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/CyclicBarrier.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/CountDownLatch.html)   [**NEXT CLASS**](http://docs.google.com/java/util/concurrent/Delayed.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/concurrent/CyclicBarrier.html)    [**NO FRAMES**](http://docs.google.com/CyclicBarrier.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | [CONSTR](#3znysh7) | [METHOD](#2et92p0) | DETAIL: FIELD | [CONSTR](#3dy6vkm) | [METHOD](#2s8eyo1) |

## **java.util.concurrent**

Class CyclicBarrier

[java.lang.Object](http://docs.google.com/java/lang/Object.html)  
 **java.util.concurrent.CyclicBarrier**

public class **CyclicBarrier**extends [Object](http://docs.google.com/java/lang/Object.html)

A synchronization aid that allows a set of threads to all wait for each other to reach a common barrier point. CyclicBarriers are useful in programs involving a fixed sized party of threads that must occasionally wait for each other. The barrier is called *cyclic* because it can be re-used after the waiting threads are released.

A CyclicBarrier supports an optional [Runnable](http://docs.google.com/java/lang/Runnable.html) command that is run once per barrier point, after the last thread in the party arrives, but before any threads are released. This *barrier action* is useful for updating shared-state before any of the parties continue.

**Sample usage:** Here is an example of using a barrier in a parallel decomposition design:

class Solver {  
 final int N;  
 final float[][] data;  
 final CyclicBarrier barrier;  
  
 class Worker implements Runnable {  
 int myRow;  
 Worker(int row) { myRow = row; }  
 public void run() {  
 while (!done()) {  
 processRow(myRow);  
  
 try {  
 barrier.await();  
 } catch (InterruptedException ex) {  
 return;  
 } catch (BrokenBarrierException ex) {  
 return;  
 }  
 }  
 }  
 }  
  
 public Solver(float[][] matrix) {  
 data = matrix;  
 N = matrix.length;  
 barrier = new CyclicBarrier(N,  
 new Runnable() {  
 public void run() {  
 mergeRows(...);  
 }  
 });  
 for (int i = 0; i < N; ++i)  
 new Thread(new Worker(i)).start();  
  
 waitUntilDone();  
 }  
 }

Here, each worker thread processes a row of the matrix then waits at the barrier until all rows have been processed. When all rows are processed the supplied [Runnable](http://docs.google.com/java/lang/Runnable.html) barrier action is executed and merges the rows. If the merger determines that a solution has been found then done() will return true and each worker will terminate.

If the barrier action does not rely on the parties being suspended when it is executed, then any of the threads in the party could execute that action when it is released. To facilitate this, each invocation of [await()](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#await()) returns the arrival index of that thread at the barrier. You can then choose which thread should execute the barrier action, for example:

if (barrier.await() == 0) {  
 // log the completion of this iteration  
 }

The CyclicBarrier uses an all-or-none breakage model for failed synchronization attempts: If a thread leaves a barrier point prematurely because of interruption, failure, or timeout, all other threads waiting at that barrier point will also leave abnormally via [BrokenBarrierException](http://docs.google.com/java/util/concurrent/BrokenBarrierException.html) (or [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html) if they too were interrupted at about the same time).

Memory consistency effects: Actions in a thread prior to calling await() [*happen-before*](http://docs.google.com/package-summary.html#MemoryVisibility) actions that are part of the barrier action, which in turn *happen-before* actions following a successful return from the corresponding await() in other threads.

**Since:** 1.5 **See Also:**[CountDownLatch](http://docs.google.com/java/util/concurrent/CountDownLatch.html)

| **Constructor Summary** | |
| --- | --- |
| [**CyclicBarrier**](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#CyclicBarrier(int))(int parties)            Creates a new CyclicBarrier that will trip when the given number of parties (threads) are waiting upon it, and does not perform a predefined action when the barrier is tripped. |
| [**CyclicBarrier**](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#CyclicBarrier(int,%20java.lang.Runnable))(int parties, [Runnable](http://docs.google.com/java/lang/Runnable.html) barrierAction)            Creates a new CyclicBarrier that will trip when the given number of parties (threads) are waiting upon it, and which will execute the given barrier action when the barrier is tripped, performed by the last thread entering the barrier. |

| **Method Summary** | |
| --- | --- |
| int | [**await**](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#await())()            Waits until all [parties](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#getParties()) have invoked await on this barrier. |
| int | [**await**](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#await(long,%20java.util.concurrent.TimeUnit))(long timeout, [TimeUnit](http://docs.google.com/java/util/concurrent/TimeUnit.html) unit)            Waits until all [parties](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#getParties()) have invoked await on this barrier, or the specified waiting time elapses. |
| int | [**getNumberWaiting**](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#getNumberWaiting())()            Returns the number of parties currently waiting at the barrier. |
| int | [**getParties**](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#getParties())()            Returns the number of parties required to trip this barrier. |
| boolean | [**isBroken**](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#isBroken())()            Queries if this barrier is in a broken state. |
| void | [**reset**](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#reset())()            Resets the barrier to its initial state. |

| **Methods inherited from class java.lang.**[**Object**](http://docs.google.com/java/lang/Object.html) |
| --- |
| [clone](http://docs.google.com/java/lang/Object.html#clone()), [equals](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)), [finalize](http://docs.google.com/java/lang/Object.html#finalize()), [getClass](http://docs.google.com/java/lang/Object.html#getClass()), [hashCode](http://docs.google.com/java/lang/Object.html#hashCode()), [notify](http://docs.google.com/java/lang/Object.html#notify()), [notifyAll](http://docs.google.com/java/lang/Object.html#notifyAll()), [toString](http://docs.google.com/java/lang/Object.html#toString()), [wait](http://docs.google.com/java/lang/Object.html#wait()), [wait](http://docs.google.com/java/lang/Object.html#wait(long)), [wait](http://docs.google.com/java/lang/Object.html#wait(long,%20int)) |

| **Constructor Detail** |
| --- |

### CyclicBarrier

public **CyclicBarrier**(int parties,  
 [Runnable](http://docs.google.com/java/lang/Runnable.html) barrierAction)

Creates a new CyclicBarrier that will trip when the given number of parties (threads) are waiting upon it, and which will execute the given barrier action when the barrier is tripped, performed by the last thread entering the barrier.

**Parameters:**parties - the number of threads that must invoke [await()](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#await()) before the barrier is trippedbarrierAction - the command to execute when the barrier is tripped, or null if there is no action **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if parties is less than 1

### CyclicBarrier

public **CyclicBarrier**(int parties)

Creates a new CyclicBarrier that will trip when the given number of parties (threads) are waiting upon it, and does not perform a predefined action when the barrier is tripped.

**Parameters:**parties - the number of threads that must invoke [await()](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#await()) before the barrier is tripped **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if parties is less than 1

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

### getParties

public int **getParties**()

Returns the number of parties required to trip this barrier.

**Returns:**the number of parties required to trip this barrier

### await

public int **await**()  
 throws [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html),  
 [BrokenBarrierException](http://docs.google.com/java/util/concurrent/BrokenBarrierException.html)

Waits until all [parties](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#getParties()) have invoked await on this barrier.

If the current thread is not the last to arrive then it is disabled for thread scheduling purposes and lies dormant until one of the following things happens:

* The last thread arrives; or
* Some other thread [interrupts](http://docs.google.com/java/lang/Thread.html#interrupt()) the current thread; or
* Some other thread [interrupts](http://docs.google.com/java/lang/Thread.html#interrupt()) one of the other waiting threads; or
* Some other thread times out while waiting for barrier; or
* Some other thread invokes [reset()](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#reset()) on this barrier.

If the current thread:

* has its interrupted status set on entry to this method; or
* is [interrupted](http://docs.google.com/java/lang/Thread.html#interrupt()) while waiting

then [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html) is thrown and the current thread's interrupted status is cleared.

If the barrier is [reset()](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#reset()) while any thread is waiting, or if the barrier [is broken](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#isBroken()) when await is invoked, or while any thread is waiting, then [BrokenBarrierException](http://docs.google.com/java/util/concurrent/BrokenBarrierException.html) is thrown.

If any thread is [interrupted](http://docs.google.com/java/lang/Thread.html#interrupt()) while waiting, then all other waiting threads will throw [BrokenBarrierException](http://docs.google.com/java/util/concurrent/BrokenBarrierException.html) and the barrier is placed in the broken state.

If the current thread is the last thread to arrive, and a non-null barrier action was supplied in the constructor, then the current thread runs the action before allowing the other threads to continue. If an exception occurs during the barrier action then that exception will be propagated in the current thread and the barrier is placed in the broken state.

**Returns:**the arrival index of the current thread, where index [getParties()](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#getParties()) - 1 indicates the first to arrive and zero indicates the last to arrive **Throws:** [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html) - if the current thread was interrupted while waiting [BrokenBarrierException](http://docs.google.com/java/util/concurrent/BrokenBarrierException.html) - if *another* thread was interrupted or timed out while the current thread was waiting, or the barrier was reset, or the barrier was broken when await was called, or the barrier action (if present) failed due an exception.

### await

public int **await**(long timeout,  
 [TimeUnit](http://docs.google.com/java/util/concurrent/TimeUnit.html) unit)  
 throws [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html),  
 [BrokenBarrierException](http://docs.google.com/java/util/concurrent/BrokenBarrierException.html),  
 [TimeoutException](http://docs.google.com/java/util/concurrent/TimeoutException.html)

Waits until all [parties](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#getParties()) have invoked await on this barrier, or the specified waiting time elapses.

If the current thread is not the last to arrive then it is disabled for thread scheduling purposes and lies dormant until one of the following things happens:

* The last thread arrives; or
* The specified timeout elapses; or
* Some other thread [interrupts](http://docs.google.com/java/lang/Thread.html#interrupt()) the current thread; or
* Some other thread [interrupts](http://docs.google.com/java/lang/Thread.html#interrupt()) one of the other waiting threads; or
* Some other thread times out while waiting for barrier; or
* Some other thread invokes [reset()](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#reset()) on this barrier.

If the current thread:

* has its interrupted status set on entry to this method; or
* is [interrupted](http://docs.google.com/java/lang/Thread.html#interrupt()) while waiting

then [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html) is thrown and the current thread's interrupted status is cleared.

If the specified waiting time elapses then [TimeoutException](http://docs.google.com/java/util/concurrent/TimeoutException.html) is thrown. If the time is less than or equal to zero, the method will not wait at all.

If the barrier is [reset()](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#reset()) while any thread is waiting, or if the barrier [is broken](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#isBroken()) when await is invoked, or while any thread is waiting, then [BrokenBarrierException](http://docs.google.com/java/util/concurrent/BrokenBarrierException.html) is thrown.

If any thread is [interrupted](http://docs.google.com/java/lang/Thread.html#interrupt()) while waiting, then all other waiting threads will throw [BrokenBarrierException](http://docs.google.com/java/util/concurrent/BrokenBarrierException.html) and the barrier is placed in the broken state.

If the current thread is the last thread to arrive, and a non-null barrier action was supplied in the constructor, then the current thread runs the action before allowing the other threads to continue. If an exception occurs during the barrier action then that exception will be propagated in the current thread and the barrier is placed in the broken state.

**Parameters:**timeout - the time to wait for the barrierunit - the time unit of the timeout parameter **Returns:**the arrival index of the current thread, where index [getParties()](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#getParties()) - 1 indicates the first to arrive and zero indicates the last to arrive **Throws:** [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html) - if the current thread was interrupted while waiting [TimeoutException](http://docs.google.com/java/util/concurrent/TimeoutException.html) - if the specified timeout elapses [BrokenBarrierException](http://docs.google.com/java/util/concurrent/BrokenBarrierException.html) - if *another* thread was interrupted or timed out while the current thread was waiting, or the barrier was reset, or the barrier was broken when await was called, or the barrier action (if present) failed due an exception

### isBroken

public boolean **isBroken**()

Queries if this barrier is in a broken state.

**Returns:**true if one or more parties broke out of this barrier due to interruption or timeout since construction or the last reset, or a barrier action failed due to an exception; false otherwise.

### reset

public void **reset**()

Resets the barrier to its initial state. If any parties are currently waiting at the barrier, they will return with a [BrokenBarrierException](http://docs.google.com/java/util/concurrent/BrokenBarrierException.html). Note that resets *after* a breakage has occurred for other reasons can be complicated to carry out; threads need to re-synchronize in some other way, and choose one to perform the reset. It may be preferable to instead create a new barrier for subsequent use.

### getNumberWaiting

public int **getNumberWaiting**()

Returns the number of parties currently waiting at the barrier. This method is primarily useful for debugging and assertions.

**Returns:**the number of parties currently blocked in [await()](http://docs.google.com/java/util/concurrent/CyclicBarrier.html#await())

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/CyclicBarrier.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/CountDownLatch.html)   [**NEXT CLASS**](http://docs.google.com/java/util/concurrent/Delayed.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/concurrent/CyclicBarrier.html)    [**NO FRAMES**](http://docs.google.com/CyclicBarrier.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | [CONSTR](#3znysh7) | [METHOD](#2et92p0) | DETAIL: FIELD | [CONSTR](#3dy6vkm) | [METHOD](#2s8eyo1) |

[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).