| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/AbstractQueuedSynchronizer.ConditionObject.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/locks/AbstractQueuedSynchronizer.html)   [**NEXT CLASS**](http://docs.google.com/java/util/concurrent/locks/Condition.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/concurrent/locks/AbstractQueuedSynchronizer.ConditionObject.html)    [**NO FRAMES**](http://docs.google.com/AbstractQueuedSynchronizer.ConditionObject.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | [CONSTR](#3znysh7) | [METHOD](#2et92p0) | DETAIL: FIELD | [CONSTR](#3dy6vkm) | [METHOD](#4d34og8) |

## **java.util.concurrent.locks**

Class AbstractQueuedSynchronizer.ConditionObject

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

**All Implemented Interfaces:** [Serializable](http://docs.google.com/java/io/Serializable.html), [Condition](http://docs.google.com/java/util/concurrent/locks/Condition.html) **Enclosing class:**[AbstractQueuedSynchronizer](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html)

public class **AbstractQueuedSynchronizer.ConditionObject**extends [Object](http://docs.google.com/java/lang/Object.html)implements [Condition](http://docs.google.com/java/util/concurrent/locks/Condition.html), [Serializable](http://docs.google.com/java/io/Serializable.html)

Condition implementation for a [AbstractQueuedSynchronizer](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html) serving as the basis of a [Lock](http://docs.google.com/java/util/concurrent/locks/Lock.html) implementation.

Method documentation for this class describes mechanics, not behavioral specifications from the point of view of Lock and Condition users. Exported versions of this class will in general need to be accompanied by documentation describing condition semantics that rely on those of the associated AbstractQueuedSynchronizer.

This class is Serializable, but all fields are transient, so deserialized conditions have no waiters.

**See Also:**[Serialized Form](http://docs.google.com/serialized-form.html#java.util.concurrent.locks.AbstractQueuedSynchronizer.ConditionObject)

| **Constructor Summary** | |
| --- | --- |
| [**AbstractQueuedSynchronizer.ConditionObject**](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.ConditionObject.html#AbstractQueuedSynchronizer.ConditionObject())()            Creates a new ConditionObject instance. |

| **Method Summary** | |
| --- | --- |
| void | [**await**](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.ConditionObject.html#await())()            Implements interruptible condition wait. |
| boolean | [**await**](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.ConditionObject.html#await(long,%20java.util.concurrent.TimeUnit))(long time, [TimeUnit](http://docs.google.com/java/util/concurrent/TimeUnit.html) unit)            Implements timed condition wait. |
| long | [**awaitNanos**](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.ConditionObject.html#awaitNanos(long))(long nanosTimeout)            Implements timed condition wait. |
| void | [**awaitUninterruptibly**](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.ConditionObject.html#awaitUninterruptibly())()            Implements uninterruptible condition wait. |
| boolean | [**awaitUntil**](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.ConditionObject.html#awaitUntil(java.util.Date))([Date](http://docs.google.com/java/util/Date.html) deadline)            Implements absolute timed condition wait. |
| protected  [Collection](http://docs.google.com/java/util/Collection.html)<[Thread](http://docs.google.com/java/lang/Thread.html)> | [**getWaitingThreads**](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.ConditionObject.html#getWaitingThreads())()            Returns a collection containing those threads that may be waiting on this Condition. |
| protected  int | [**getWaitQueueLength**](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.ConditionObject.html#getWaitQueueLength())()            Returns an estimate of the number of threads waiting on this condition. |
| protected  boolean | [**hasWaiters**](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.ConditionObject.html#hasWaiters())()            Queries whether any threads are waiting on this condition. |
| void | [**signal**](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.ConditionObject.html#signal())()            Moves the longest-waiting thread, if one exists, from the wait queue for this condition to the wait queue for the owning lock. |
| void | [**signalAll**](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.ConditionObject.html#signalAll())()            Moves all threads from the wait queue for this condition to the wait queue for the owning lock. |

| **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** |
| --- |

### AbstractQueuedSynchronizer.ConditionObject

public **AbstractQueuedSynchronizer.ConditionObject**()

Creates a new ConditionObject instance.

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

### signal

public final void **signal**()

Moves the longest-waiting thread, if one exists, from the wait queue for this condition to the wait queue for the owning lock.

**Specified by:**[signal](http://docs.google.com/java/util/concurrent/locks/Condition.html#signal()) in interface [Condition](http://docs.google.com/java/util/concurrent/locks/Condition.html) **Throws:** [IllegalMonitorStateException](http://docs.google.com/java/lang/IllegalMonitorStateException.html) - if [AbstractQueuedSynchronizer.isHeldExclusively()](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html#isHeldExclusively()) returns false

### signalAll

public final void **signalAll**()

Moves all threads from the wait queue for this condition to the wait queue for the owning lock.

**Specified by:**[signalAll](http://docs.google.com/java/util/concurrent/locks/Condition.html#signalAll()) in interface [Condition](http://docs.google.com/java/util/concurrent/locks/Condition.html) **Throws:** [IllegalMonitorStateException](http://docs.google.com/java/lang/IllegalMonitorStateException.html) - if [AbstractQueuedSynchronizer.isHeldExclusively()](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html#isHeldExclusively()) returns false

### awaitUninterruptibly

public final void **awaitUninterruptibly**()

Implements uninterruptible condition wait.

1. Save lock state returned by [AbstractQueuedSynchronizer.getState()](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html#getState())
2. Invoke [AbstractQueuedSynchronizer.release(int)](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html#release(int)) with saved state as argument, throwing IllegalMonitorStateException if it fails.
3. Block until signalled
4. Reacquire by invoking specialized version of [AbstractQueuedSynchronizer.acquire(int)](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html#acquire(int)) with saved state as argument.

**Specified by:**[awaitUninterruptibly](http://docs.google.com/java/util/concurrent/locks/Condition.html#awaitUninterruptibly()) in interface [Condition](http://docs.google.com/java/util/concurrent/locks/Condition.html)

### await

public final void **await**()  
 throws [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html)

Implements interruptible condition wait.

1. If current thread is interrupted, throw InterruptedException
2. Save lock state returned by [AbstractQueuedSynchronizer.getState()](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html#getState())
3. Invoke [AbstractQueuedSynchronizer.release(int)](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html#release(int)) with saved state as argument, throwing IllegalMonitorStateException if it fails.
4. Block until signalled or interrupted
5. Reacquire by invoking specialized version of [AbstractQueuedSynchronizer.acquire(int)](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html#acquire(int)) with saved state as argument.
6. If interrupted while blocked in step 4, throw exception

**Specified by:**[await](http://docs.google.com/java/util/concurrent/locks/Condition.html#await()) in interface [Condition](http://docs.google.com/java/util/concurrent/locks/Condition.html) **Throws:** [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html) - if the current thread is interrupted (and interruption of thread suspension is supported)

### awaitNanos

public final long **awaitNanos**(long nanosTimeout)  
 throws [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html)

Implements timed condition wait.

1. If current thread is interrupted, throw InterruptedException
2. Save lock state returned by [AbstractQueuedSynchronizer.getState()](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html#getState())
3. Invoke [AbstractQueuedSynchronizer.release(int)](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html#release(int)) with saved state as argument, throwing IllegalMonitorStateException if it fails.
4. Block until signalled, interrupted, or timed out
5. Reacquire by invoking specialized version of [AbstractQueuedSynchronizer.acquire(int)](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html#acquire(int)) with saved state as argument.
6. If interrupted while blocked in step 4, throw InterruptedException

**Specified by:**[awaitNanos](http://docs.google.com/java/util/concurrent/locks/Condition.html#awaitNanos(long)) in interface [Condition](http://docs.google.com/java/util/concurrent/locks/Condition.html) **Parameters:**nanosTimeout - the maximum time to wait, in nanoseconds **Returns:**an estimate of the nanosTimeout value minus the time spent waiting upon return from this method. A positive value may be used as the argument to a subsequent call to this method to finish waiting out the desired time. A value less than or equal to zero indicates that no time remains. **Throws:** [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html) - if the current thread is interrupted (and interruption of thread suspension is supported)

### awaitUntil

public final boolean **awaitUntil**([Date](http://docs.google.com/java/util/Date.html) deadline)  
 throws [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html)

Implements absolute timed condition wait.

1. If current thread is interrupted, throw InterruptedException
2. Save lock state returned by [AbstractQueuedSynchronizer.getState()](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html#getState())
3. Invoke [AbstractQueuedSynchronizer.release(int)](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html#release(int)) with saved state as argument, throwing IllegalMonitorStateException if it fails.
4. Block until signalled, interrupted, or timed out
5. Reacquire by invoking specialized version of [AbstractQueuedSynchronizer.acquire(int)](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html#acquire(int)) with saved state as argument.
6. If interrupted while blocked in step 4, throw InterruptedException
7. If timed out while blocked in step 4, return false, else true

**Specified by:**[awaitUntil](http://docs.google.com/java/util/concurrent/locks/Condition.html#awaitUntil(java.util.Date)) in interface [Condition](http://docs.google.com/java/util/concurrent/locks/Condition.html) **Parameters:**deadline - the absolute time to wait until **Returns:**false if the deadline has elapsed upon return, else true **Throws:** [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html) - if the current thread is interrupted (and interruption of thread suspension is supported)

### await

public final boolean **await**(long time,  
 [TimeUnit](http://docs.google.com/java/util/concurrent/TimeUnit.html) unit)  
 throws [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html)

Implements timed condition wait.

1. If current thread is interrupted, throw InterruptedException
2. Save lock state returned by [AbstractQueuedSynchronizer.getState()](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html#getState())
3. Invoke [AbstractQueuedSynchronizer.release(int)](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html#release(int)) with saved state as argument, throwing IllegalMonitorStateException if it fails.
4. Block until signalled, interrupted, or timed out
5. Reacquire by invoking specialized version of [AbstractQueuedSynchronizer.acquire(int)](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html#acquire(int)) with saved state as argument.
6. If interrupted while blocked in step 4, throw InterruptedException
7. If timed out while blocked in step 4, return false, else true

**Specified by:**[await](http://docs.google.com/java/util/concurrent/locks/Condition.html#await(long,%20java.util.concurrent.TimeUnit)) in interface [Condition](http://docs.google.com/java/util/concurrent/locks/Condition.html) **Parameters:**time - the maximum time to waitunit - the time unit of the time argument **Returns:**false if the waiting time detectably elapsed before return from the method, else true **Throws:** [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html) - if the current thread is interrupted (and interruption of thread suspension is supported)

### hasWaiters

protected final boolean **hasWaiters**()

Queries whether any threads are waiting on this condition. Implements [AbstractQueuedSynchronizer.hasWaiters(java.util.concurrent.locks.AbstractQueuedSynchronizer.ConditionObject)](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html#hasWaiters(java.util.concurrent.locks.AbstractQueuedSynchronizer.ConditionObject)).

**Returns:**true if there are any waiting threads **Throws:** [IllegalMonitorStateException](http://docs.google.com/java/lang/IllegalMonitorStateException.html) - if [AbstractQueuedSynchronizer.isHeldExclusively()](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html#isHeldExclusively()) returns false

### getWaitQueueLength

protected final int **getWaitQueueLength**()

Returns an estimate of the number of threads waiting on this condition. Implements [AbstractQueuedSynchronizer.getWaitQueueLength(java.util.concurrent.locks.AbstractQueuedSynchronizer.ConditionObject)](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html#getWaitQueueLength(java.util.concurrent.locks.AbstractQueuedSynchronizer.ConditionObject)).

**Returns:**the estimated number of waiting threads **Throws:** [IllegalMonitorStateException](http://docs.google.com/java/lang/IllegalMonitorStateException.html) - if [AbstractQueuedSynchronizer.isHeldExclusively()](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html#isHeldExclusively()) returns false

### getWaitingThreads

protected final [Collection](http://docs.google.com/java/util/Collection.html)<[Thread](http://docs.google.com/java/lang/Thread.html)> **getWaitingThreads**()

Returns a collection containing those threads that may be waiting on this Condition. Implements [AbstractQueuedSynchronizer.getWaitingThreads(java.util.concurrent.locks.AbstractQueuedSynchronizer.ConditionObject)](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html#getWaitingThreads(java.util.concurrent.locks.AbstractQueuedSynchronizer.ConditionObject)).

**Returns:**the collection of threads **Throws:** [IllegalMonitorStateException](http://docs.google.com/java/lang/IllegalMonitorStateException.html) - if [AbstractQueuedSynchronizer.isHeldExclusively()](http://docs.google.com/java/util/concurrent/locks/AbstractQueuedSynchronizer.html#isHeldExclusively()) returns false

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/AbstractQueuedSynchronizer.ConditionObject.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/locks/AbstractQueuedSynchronizer.html)   [**NEXT CLASS**](http://docs.google.com/java/util/concurrent/locks/Condition.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/concurrent/locks/AbstractQueuedSynchronizer.ConditionObject.html)    [**NO FRAMES**](http://docs.google.com/AbstractQueuedSynchronizer.ConditionObject.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | [CONSTR](#3znysh7) | [METHOD](#2et92p0) | DETAIL: FIELD | [CONSTR](#3dy6vkm) | [METHOD](#4d34og8) |

[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).