| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/ReentrantLock.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/ReadWriteLock.html)   [**NEXT CLASS**](http://docs.google.com/java/util/concurrent/locks/ReentrantReadWriteLock.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/concurrent/locks/ReentrantLock.html)    [**NO FRAMES**](http://docs.google.com/ReentrantLock.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.locks**

Class ReentrantLock

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

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

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

A reentrant mutual exclusion [Lock](http://docs.google.com/java/util/concurrent/locks/Lock.html) with the same basic behavior and semantics as the implicit monitor lock accessed using synchronized methods and statements, but with extended capabilities.

A ReentrantLock is *owned* by the thread last successfully locking, but not yet unlocking it. A thread invoking lock will return, successfully acquiring the lock, when the lock is not owned by another thread. The method will return immediately if the current thread already owns the lock. This can be checked using methods [isHeldByCurrentThread()](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#isHeldByCurrentThread()), and [getHoldCount()](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#getHoldCount()).

The constructor for this class accepts an optional *fairness* parameter. When set true, under contention, locks favor granting access to the longest-waiting thread. Otherwise this lock does not guarantee any particular access order. Programs using fair locks accessed by many threads may display lower overall throughput (i.e., are slower; often much slower) than those using the default setting, but have smaller variances in times to obtain locks and guarantee lack of starvation. Note however, that fairness of locks does not guarantee fairness of thread scheduling. Thus, one of many threads using a fair lock may obtain it multiple times in succession while other active threads are not progressing and not currently holding the lock. Also note that the untimed [tryLock](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#tryLock()) method does not honor the fairness setting. It will succeed if the lock is available even if other threads are waiting.

It is recommended practice to *always* immediately follow a call to lock with a try block, most typically in a before/after construction such as:

class X {  
 private final ReentrantLock lock = new ReentrantLock();  
 // ...  
  
 public void m() {  
 lock.lock(); // block until condition holds  
 try {  
 // ... method body  
 } finally {  
 lock.unlock()  
 }  
 }  
 }

In addition to implementing the [Lock](http://docs.google.com/java/util/concurrent/locks/Lock.html) interface, this class defines methods isLocked and getLockQueueLength, as well as some associated protected access methods that may be useful for instrumentation and monitoring.

Serialization of this class behaves in the same way as built-in locks: a deserialized lock is in the unlocked state, regardless of its state when serialized.

This lock supports a maximum of 2147483647 recursive locks by the same thread. Attempts to exceed this limit result in [Error](http://docs.google.com/java/lang/Error.html) throws from locking methods.

**Since:** 1.5 **See Also:**[Serialized Form](http://docs.google.com/serialized-form.html#java.util.concurrent.locks.ReentrantLock)

| **Constructor Summary** | |
| --- | --- |
| [**ReentrantLock**](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#ReentrantLock())()            Creates an instance of ReentrantLock. |
| [**ReentrantLock**](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#ReentrantLock(boolean))(boolean fair)            Creates an instance of ReentrantLock with the given fairness policy. |

| **Method Summary** | |
| --- | --- |
| int | [**getHoldCount**](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#getHoldCount())()            Queries the number of holds on this lock by the current thread. |
| protected  [Thread](http://docs.google.com/java/lang/Thread.html) | [**getOwner**](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#getOwner())()            Returns the thread that currently owns this lock, or null if not owned. |
| protected  [Collection](http://docs.google.com/java/util/Collection.html)<[Thread](http://docs.google.com/java/lang/Thread.html)> | [**getQueuedThreads**](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#getQueuedThreads())()            Returns a collection containing threads that may be waiting to acquire this lock. |
| int | [**getQueueLength**](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#getQueueLength())()            Returns an estimate of the number of threads waiting to acquire this lock. |
| 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/ReentrantLock.html#getWaitingThreads(java.util.concurrent.locks.Condition))([Condition](http://docs.google.com/java/util/concurrent/locks/Condition.html) condition)            Returns a collection containing those threads that may be waiting on the given condition associated with this lock. |
| int | [**getWaitQueueLength**](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#getWaitQueueLength(java.util.concurrent.locks.Condition))([Condition](http://docs.google.com/java/util/concurrent/locks/Condition.html) condition)            Returns an estimate of the number of threads waiting on the given condition associated with this lock. |
| boolean | [**hasQueuedThread**](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#hasQueuedThread(java.lang.Thread))([Thread](http://docs.google.com/java/lang/Thread.html) thread)            Queries whether the given thread is waiting to acquire this lock. |
| boolean | [**hasQueuedThreads**](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#hasQueuedThreads())()            Queries whether any threads are waiting to acquire this lock. |
| boolean | [**hasWaiters**](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#hasWaiters(java.util.concurrent.locks.Condition))([Condition](http://docs.google.com/java/util/concurrent/locks/Condition.html) condition)            Queries whether any threads are waiting on the given condition associated with this lock. |
| boolean | [**isFair**](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#isFair())()            Returns true if this lock has fairness set true. |
| boolean | [**isHeldByCurrentThread**](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#isHeldByCurrentThread())()            Queries if this lock is held by the current thread. |
| boolean | [**isLocked**](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#isLocked())()            Queries if this lock is held by any thread. |
| void | [**lock**](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#lock())()            Acquires the lock. |
| void | [**lockInterruptibly**](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#lockInterruptibly())()            Acquires the lock unless the current thread is [interrupted](http://docs.google.com/java/lang/Thread.html#interrupt()). |
| [Condition](http://docs.google.com/java/util/concurrent/locks/Condition.html) | [**newCondition**](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#newCondition())()            Returns a [Condition](http://docs.google.com/java/util/concurrent/locks/Condition.html) instance for use with this [Lock](http://docs.google.com/java/util/concurrent/locks/Lock.html) instance. |
| [String](http://docs.google.com/java/lang/String.html) | [**toString**](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#toString())()            Returns a string identifying this lock, as well as its lock state. |
| boolean | [**tryLock**](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#tryLock())()            Acquires the lock only if it is not held by another thread at the time of invocation. |
| boolean | [**tryLock**](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#tryLock(long,%20java.util.concurrent.TimeUnit))(long timeout, [TimeUnit](http://docs.google.com/java/util/concurrent/TimeUnit.html) unit)            Acquires the lock if it is not held by another thread within the given waiting time and the current thread has not been [interrupted](http://docs.google.com/java/lang/Thread.html#interrupt()). |
| void | [**unlock**](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#unlock())()            Attempts to release this 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()), [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** |
| --- |

### ReentrantLock

public **ReentrantLock**()

Creates an instance of ReentrantLock. This is equivalent to using ReentrantLock(false).

### ReentrantLock

public **ReentrantLock**(boolean fair)

Creates an instance of ReentrantLock with the given fairness policy.

**Parameters:**fair - true if this lock should use a fair ordering policy

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

### lock

public void **lock**()

Acquires the lock.

Acquires the lock if it is not held by another thread and returns immediately, setting the lock hold count to one.

If the current thread already holds the lock then the hold count is incremented by one and the method returns immediately.

If the lock is held by another thread then the current thread becomes disabled for thread scheduling purposes and lies dormant until the lock has been acquired, at which time the lock hold count is set to one.

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

### lockInterruptibly

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

Acquires the lock unless the current thread is [interrupted](http://docs.google.com/java/lang/Thread.html#interrupt()).

Acquires the lock if it is not held by another thread and returns immediately, setting the lock hold count to one.

If the current thread already holds this lock then the hold count is incremented by one and the method returns immediately.

If the lock is held by another thread then the current thread becomes disabled for thread scheduling purposes and lies dormant until one of two things happens:

* The lock is acquired by the current thread; or
* Some other thread [interrupts](http://docs.google.com/java/lang/Thread.html#interrupt()) the current thread.

If the lock is acquired by the current thread then the lock hold count is set to one.

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 acquiring the lock,

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

In this implementation, as this method is an explicit interruption point, preference is given to responding to the interrupt over normal or reentrant acquisition of the lock.

**Specified by:**[lockInterruptibly](http://docs.google.com/java/util/concurrent/locks/Lock.html#lockInterruptibly()) in interface [Lock](http://docs.google.com/java/util/concurrent/locks/Lock.html) **Throws:** [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html) - if the current thread is interrupted

### tryLock

public boolean **tryLock**()

Acquires the lock only if it is not held by another thread at the time of invocation.

Acquires the lock if it is not held by another thread and returns immediately with the value true, setting the lock hold count to one. Even when this lock has been set to use a fair ordering policy, a call to tryLock() *will* immediately acquire the lock if it is available, whether or not other threads are currently waiting for the lock. This "barging" behavior can be useful in certain circumstances, even though it breaks fairness. If you want to honor the fairness setting for this lock, then use [tryLock(0, TimeUnit.SECONDS)](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#tryLock(long,%20java.util.concurrent.TimeUnit))  which is almost equivalent (it also detects interruption).

If the current thread already holds this lock then the hold count is incremented by one and the method returns true.

If the lock is held by another thread then this method will return immediately with the value false.

**Specified by:**[tryLock](http://docs.google.com/java/util/concurrent/locks/Lock.html#tryLock()) in interface [Lock](http://docs.google.com/java/util/concurrent/locks/Lock.html) **Returns:**true if the lock was free and was acquired by the current thread, or the lock was already held by the current thread; and false otherwise

### tryLock

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

Acquires the lock if it is not held by another thread within the given waiting time and the current thread has not been [interrupted](http://docs.google.com/java/lang/Thread.html#interrupt()).

Acquires the lock if it is not held by another thread and returns immediately with the value true, setting the lock hold count to one. If this lock has been set to use a fair ordering policy then an available lock *will not* be acquired if any other threads are waiting for the lock. This is in contrast to the [tryLock()](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html#tryLock()) method. If you want a timed tryLock that does permit barging on a fair lock then combine the timed and un-timed forms together:

if (lock.tryLock() || lock.tryLock(timeout, unit) ) { ... }

If the current thread already holds this lock then the hold count is incremented by one and the method returns true.

If the lock is held by another thread then the current thread becomes disabled for thread scheduling purposes and lies dormant until one of three things happens:

* The lock is acquired by the current thread; or
* Some other thread [interrupts](http://docs.google.com/java/lang/Thread.html#interrupt()) the current thread; or
* The specified waiting time elapses

If the lock is acquired then the value true is returned and the lock hold count is set to one.

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 acquiring the lock,

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 the value false is returned. If the time is less than or equal to zero, the method will not wait at all.

In this implementation, as this method is an explicit interruption point, preference is given to responding to the interrupt over normal or reentrant acquisition of the lock, and over reporting the elapse of the waiting time.

**Specified by:**[tryLock](http://docs.google.com/java/util/concurrent/locks/Lock.html#tryLock(long,%20java.util.concurrent.TimeUnit)) in interface [Lock](http://docs.google.com/java/util/concurrent/locks/Lock.html) **Parameters:**timeout - the time to wait for the lockunit - the time unit of the timeout argument **Returns:**true if the lock was free and was acquired by the current thread, or the lock was already held by the current thread; and false if the waiting time elapsed before the lock could be acquired **Throws:** [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html) - if the current thread is interrupted [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the time unit is null

### unlock

public void **unlock**()

Attempts to release this lock.

If the current thread is the holder of this lock then the hold count is decremented. If the hold count is now zero then the lock is released. If the current thread is not the holder of this lock then [IllegalMonitorStateException](http://docs.google.com/java/lang/IllegalMonitorStateException.html) is thrown.

**Specified by:**[unlock](http://docs.google.com/java/util/concurrent/locks/Lock.html#unlock()) in interface [Lock](http://docs.google.com/java/util/concurrent/locks/Lock.html) **Throws:** [IllegalMonitorStateException](http://docs.google.com/java/lang/IllegalMonitorStateException.html) - if the current thread does not hold this lock

### newCondition

public [Condition](http://docs.google.com/java/util/concurrent/locks/Condition.html) **newCondition**()

Returns a [Condition](http://docs.google.com/java/util/concurrent/locks/Condition.html) instance for use with this [Lock](http://docs.google.com/java/util/concurrent/locks/Lock.html) instance.

The returned [Condition](http://docs.google.com/java/util/concurrent/locks/Condition.html) instance supports the same usages as do the [Object](http://docs.google.com/java/lang/Object.html) monitor methods ([wait](http://docs.google.com/java/lang/Object.html#wait()), [notify](http://docs.google.com/java/lang/Object.html#notify()), and [notifyAll](http://docs.google.com/java/lang/Object.html#notifyAll())) when used with the built-in monitor lock.

* If this lock is not held when any of the [Condition](http://docs.google.com/java/util/concurrent/locks/Condition.html) [waiting](http://docs.google.com/java/util/concurrent/locks/Condition.html#await()) or [signalling](http://docs.google.com/java/util/concurrent/locks/Condition.html#signal()) methods are called, then an [IllegalMonitorStateException](http://docs.google.com/java/lang/IllegalMonitorStateException.html) is thrown.
* When the condition [waiting](http://docs.google.com/java/util/concurrent/locks/Condition.html#await()) methods are called the lock is released and, before they return, the lock is reacquired and the lock hold count restored to what it was when the method was called.
* If a thread is [interrupted](http://docs.google.com/java/lang/Thread.html#interrupt()) while waiting then the wait will terminate, an [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html) will be thrown, and the thread's interrupted status will be cleared.
* Waiting threads are signalled in FIFO order.
* The ordering of lock reacquisition for threads returning from waiting methods is the same as for threads initially acquiring the lock, which is in the default case not specified, but for *fair* locks favors those threads that have been waiting the longest.

**Specified by:**[newCondition](http://docs.google.com/java/util/concurrent/locks/Lock.html#newCondition()) in interface [Lock](http://docs.google.com/java/util/concurrent/locks/Lock.html) **Returns:**the Condition object

### getHoldCount

public int **getHoldCount**()

Queries the number of holds on this lock by the current thread.

A thread has a hold on a lock for each lock action that is not matched by an unlock action.

The hold count information is typically only used for testing and debugging purposes. For example, if a certain section of code should not be entered with the lock already held then we can assert that fact:

class X {  
 ReentrantLock lock = new ReentrantLock();  
 // ...  
 public void m() {  
 assert lock.getHoldCount() == 0;  
 lock.lock();  
 try {  
 // ... method body  
 } finally {  
 lock.unlock();  
 }  
 }  
 }

**Returns:**the number of holds on this lock by the current thread, or zero if this lock is not held by the current thread

### isHeldByCurrentThread

public boolean **isHeldByCurrentThread**()

Queries if this lock is held by the current thread.

Analogous to the [Thread.holdsLock(java.lang.Object)](http://docs.google.com/java/lang/Thread.html#holdsLock(java.lang.Object)) method for built-in monitor locks, this method is typically used for debugging and testing. For example, a method that should only be called while a lock is held can assert that this is the case:

class X {  
 ReentrantLock lock = new ReentrantLock();  
 // ...  
  
 public void m() {  
 assert lock.isHeldByCurrentThread();  
 // ... method body  
 }  
 }

It can also be used to ensure that a reentrant lock is used in a non-reentrant manner, for example:

class X {  
 ReentrantLock lock = new ReentrantLock();  
 // ...  
  
 public void m() {  
 assert !lock.isHeldByCurrentThread();  
 lock.lock();  
 try {  
 // ... method body  
 } finally {  
 lock.unlock();  
 }  
 }  
 }

**Returns:**true if current thread holds this lock and false otherwise

### isLocked

public boolean **isLocked**()

Queries if this lock is held by any thread. This method is designed for use in monitoring of the system state, not for synchronization control.

**Returns:**true if any thread holds this lock and false otherwise

### isFair

public final boolean **isFair**()

Returns true if this lock has fairness set true.

**Returns:**true if this lock has fairness set true

### getOwner

protected [Thread](http://docs.google.com/java/lang/Thread.html) **getOwner**()

Returns the thread that currently owns this lock, or null if not owned. When this method is called by a thread that is not the owner, the return value reflects a best-effort approximation of current lock status. For example, the owner may be momentarily null even if there are threads trying to acquire the lock but have not yet done so. This method is designed to facilitate construction of subclasses that provide more extensive lock monitoring facilities.

**Returns:**the owner, or null if not owned

### hasQueuedThreads

public final boolean **hasQueuedThreads**()

Queries whether any threads are waiting to acquire this lock. Note that because cancellations may occur at any time, a true return does not guarantee that any other thread will ever acquire this lock. This method is designed primarily for use in monitoring of the system state.

**Returns:**true if there may be other threads waiting to acquire the lock

### hasQueuedThread

public final boolean **hasQueuedThread**([Thread](http://docs.google.com/java/lang/Thread.html) thread)

Queries whether the given thread is waiting to acquire this lock. Note that because cancellations may occur at any time, a true return does not guarantee that this thread will ever acquire this lock. This method is designed primarily for use in monitoring of the system state.

**Parameters:**thread - the thread **Returns:**true if the given thread is queued waiting for this lock **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the thread is null

### getQueueLength

public final int **getQueueLength**()

Returns an estimate of the number of threads waiting to acquire this lock. The value is only an estimate because the number of threads may change dynamically while this method traverses internal data structures. This method is designed for use in monitoring of the system state, not for synchronization control.

**Returns:**the estimated number of threads waiting for this lock

### getQueuedThreads

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

Returns a collection containing threads that may be waiting to acquire this lock. Because the actual set of threads may change dynamically while constructing this result, the returned collection is only a best-effort estimate. The elements of the returned collection are in no particular order. This method is designed to facilitate construction of subclasses that provide more extensive monitoring facilities.

**Returns:**the collection of threads

### hasWaiters

public boolean **hasWaiters**([Condition](http://docs.google.com/java/util/concurrent/locks/Condition.html) condition)

Queries whether any threads are waiting on the given condition associated with this lock. Note that because timeouts and interrupts may occur at any time, a true return does not guarantee that a future signal will awaken any threads. This method is designed primarily for use in monitoring of the system state.

**Parameters:**condition - the condition **Returns:**true if there are any waiting threads **Throws:** [IllegalMonitorStateException](http://docs.google.com/java/lang/IllegalMonitorStateException.html) - if this lock is not held [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if the given condition is not associated with this lock [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the condition is null

### getWaitQueueLength

public int **getWaitQueueLength**([Condition](http://docs.google.com/java/util/concurrent/locks/Condition.html) condition)

Returns an estimate of the number of threads waiting on the given condition associated with this lock. Note that because timeouts and interrupts may occur at any time, the estimate serves only as an upper bound on the actual number of waiters. This method is designed for use in monitoring of the system state, not for synchronization control.

**Parameters:**condition - the condition **Returns:**the estimated number of waiting threads **Throws:** [IllegalMonitorStateException](http://docs.google.com/java/lang/IllegalMonitorStateException.html) - if this lock is not held [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if the given condition is not associated with this lock [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the condition is null

### getWaitingThreads

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

Returns a collection containing those threads that may be waiting on the given condition associated with this lock. Because the actual set of threads may change dynamically while constructing this result, the returned collection is only a best-effort estimate. The elements of the returned collection are in no particular order. This method is designed to facilitate construction of subclasses that provide more extensive condition monitoring facilities.

**Parameters:**condition - the condition **Returns:**the collection of threads **Throws:** [IllegalMonitorStateException](http://docs.google.com/java/lang/IllegalMonitorStateException.html) - if this lock is not held [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if the given condition is not associated with this lock [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the condition is null

### toString

public [String](http://docs.google.com/java/lang/String.html) **toString**()

Returns a string identifying this lock, as well as its lock state. The state, in brackets, includes either the String "Unlocked" or the String "Locked by" followed by the [name](http://docs.google.com/java/lang/Thread.html#getName()) of the owning thread.

**Overrides:**[toString](http://docs.google.com/java/lang/Object.html#toString()) in class [Object](http://docs.google.com/java/lang/Object.html) **Returns:**a string identifying this lock, as well as its lock state

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/ReentrantLock.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/ReadWriteLock.html)   [**NEXT CLASS**](http://docs.google.com/java/util/concurrent/locks/ReentrantReadWriteLock.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/concurrent/locks/ReentrantLock.html)    [**NO FRAMES**](http://docs.google.com/ReentrantLock.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).