| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/LockSupport.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/Lock.html)   [**NEXT CLASS**](http://docs.google.com/java/util/concurrent/locks/ReadWriteLock.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/concurrent/locks/LockSupport.html)    [**NO FRAMES**](http://docs.google.com/LockSupport.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#3znysh7) | DETAIL: FIELD | CONSTR | [METHOD](#tyjcwt) |

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

Class LockSupport

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

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

Basic thread blocking primitives for creating locks and other synchronization classes.

This class associates, with each thread that uses it, a permit (in the sense of the [Semaphore](http://docs.google.com/java/util/concurrent/Semaphore.html) class). A call to park will return immediately if the permit is available, consuming it in the process; otherwise it *may* block. A call to unpark makes the permit available, if it was not already available. (Unlike with Semaphores though, permits do not accumulate. There is at most one.)

Methods park and unpark provide efficient means of blocking and unblocking threads that do not encounter the problems that cause the deprecated methods Thread.suspend and Thread.resume to be unusable for such purposes: Races between one thread invoking park and another thread trying to unpark it will preserve liveness, due to the permit. Additionally, park will return if the caller's thread was interrupted, and timeout versions are supported. The park method may also return at any other time, for "no reason", so in general must be invoked within a loop that rechecks conditions upon return. In this sense park serves as an optimization of a "busy wait" that does not waste as much time spinning, but must be paired with an unpark to be effective.

The three forms of park each also support a blocker object parameter. This object is recorded while the thread is blocked to permit monitoring and diagnostic tools to identify the reasons that threads are blocked. (Such tools may access blockers using method [getBlocker(java.lang.Thread)](http://docs.google.com/java/util/concurrent/locks/LockSupport.html#getBlocker(java.lang.Thread)).) The use of these forms rather than the original forms without this parameter is strongly encouraged. The normal argument to supply as a blocker within a lock implementation is this.

These methods are designed to be used as tools for creating higher-level synchronization utilities, and are not in themselves useful for most concurrency control applications. The park method is designed for use only in constructions of the form:

while (!canProceed()) { ... LockSupport.park(this); }

where neither canProceed nor any other actions prior to the call to park entail locking or blocking. Because only one permit is associated with each thread, any intermediary uses of park could interfere with its intended effects.

**Sample Usage.** Here is a sketch of a first-in-first-out non-reentrant lock class:

class FIFOMutex {  
 private final AtomicBoolean locked = new AtomicBoolean(false);  
 private final Queue<Thread> waiters  
 = new ConcurrentLinkedQueue<Thread>();  
  
 public void lock() {  
 boolean wasInterrupted = false;  
 Thread current = Thread.currentThread();  
 waiters.add(current);  
  
 // Block while not first in queue or cannot acquire lock  
 while (waiters.peek() != current ||  
 !locked.compareAndSet(false, true)) {  
 LockSupport.park(this);  
 if (Thread.interrupted()) // ignore interrupts while waiting  
 wasInterrupted = true;  
 }  
  
 waiters.remove();  
 if (wasInterrupted) // reassert interrupt status on exit  
 current.interrupt();  
 }  
  
 public void unlock() {  
 locked.set(false);  
 LockSupport.unpark(waiters.peek());  
 }  
 }

| **Method Summary** | |
| --- | --- |
| static [Object](http://docs.google.com/java/lang/Object.html) | [**getBlocker**](http://docs.google.com/java/util/concurrent/locks/LockSupport.html#getBlocker(java.lang.Thread))([Thread](http://docs.google.com/java/lang/Thread.html) t)            Returns the blocker object supplied to the most recent invocation of a park method that has not yet unblocked, or null if not blocked. |
| static void | [**park**](http://docs.google.com/java/util/concurrent/locks/LockSupport.html#park())()            Disables the current thread for thread scheduling purposes unless the permit is available. |
| static void | [**park**](http://docs.google.com/java/util/concurrent/locks/LockSupport.html#park(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) blocker)            Disables the current thread for thread scheduling purposes unless the permit is available. |
| static void | [**parkNanos**](http://docs.google.com/java/util/concurrent/locks/LockSupport.html#parkNanos(long))(long nanos)            Disables the current thread for thread scheduling purposes, for up to the specified waiting time, unless the permit is available. |
| static void | [**parkNanos**](http://docs.google.com/java/util/concurrent/locks/LockSupport.html#parkNanos(java.lang.Object,%20long))([Object](http://docs.google.com/java/lang/Object.html) blocker, long nanos)            Disables the current thread for thread scheduling purposes, for up to the specified waiting time, unless the permit is available. |
| static void | [**parkUntil**](http://docs.google.com/java/util/concurrent/locks/LockSupport.html#parkUntil(long))(long deadline)            Disables the current thread for thread scheduling purposes, until the specified deadline, unless the permit is available. |
| static void | [**parkUntil**](http://docs.google.com/java/util/concurrent/locks/LockSupport.html#parkUntil(java.lang.Object,%20long))([Object](http://docs.google.com/java/lang/Object.html) blocker, long deadline)            Disables the current thread for thread scheduling purposes, until the specified deadline, unless the permit is available. |
| static void | [**unpark**](http://docs.google.com/java/util/concurrent/locks/LockSupport.html#unpark(java.lang.Thread))([Thread](http://docs.google.com/java/lang/Thread.html) thread)            Makes available the permit for the given thread, if it was not already available. |

| **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)) |

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

### unpark

public static void **unpark**([Thread](http://docs.google.com/java/lang/Thread.html) thread)

Makes available the permit for the given thread, if it was not already available. If the thread was blocked on park then it will unblock. Otherwise, its next call to park is guaranteed not to block. This operation is not guaranteed to have any effect at all if the given thread has not been started.

**Parameters:**thread - the thread to unpark, or null, in which case this operation has no effect

### park

public static void **park**([Object](http://docs.google.com/java/lang/Object.html) blocker)

Disables the current thread for thread scheduling purposes unless the permit is available.

If the permit is available then it is consumed and the call returns immediately; otherwise the current thread becomes disabled for thread scheduling purposes and lies dormant until one of three things happens:

* Some other thread invokes [unpark](http://docs.google.com/java/util/concurrent/locks/LockSupport.html#unpark(java.lang.Thread)) with the current thread as the target; or
* Some other thread [interrupts](http://docs.google.com/java/lang/Thread.html#interrupt()) the current thread; or
* The call spuriously (that is, for no reason) returns.

This method does *not* report which of these caused the method to return. Callers should re-check the conditions which caused the thread to park in the first place. Callers may also determine, for example, the interrupt status of the thread upon return.

**Parameters:**blocker - the synchronization object responsible for this thread parking**Since:** 1.6

### parkNanos

public static void **parkNanos**([Object](http://docs.google.com/java/lang/Object.html) blocker,  
 long nanos)

Disables the current thread for thread scheduling purposes, for up to the specified waiting time, unless the permit is available.

If the permit is available then it is consumed and the call returns immediately; otherwise the current thread becomes disabled for thread scheduling purposes and lies dormant until one of four things happens:

* Some other thread invokes [unpark](http://docs.google.com/java/util/concurrent/locks/LockSupport.html#unpark(java.lang.Thread)) with the current thread as the target; or
* Some other thread [interrupts](http://docs.google.com/java/lang/Thread.html#interrupt()) the current thread; or
* The specified waiting time elapses; or
* The call spuriously (that is, for no reason) returns.

This method does *not* report which of these caused the method to return. Callers should re-check the conditions which caused the thread to park in the first place. Callers may also determine, for example, the interrupt status of the thread, or the elapsed time upon return.

**Parameters:**blocker - the synchronization object responsible for this thread parkingnanos - the maximum number of nanoseconds to wait**Since:** 1.6

### parkUntil

public static void **parkUntil**([Object](http://docs.google.com/java/lang/Object.html) blocker,  
 long deadline)

Disables the current thread for thread scheduling purposes, until the specified deadline, unless the permit is available.

If the permit is available then it is consumed and the call returns immediately; otherwise the current thread becomes disabled for thread scheduling purposes and lies dormant until one of four things happens:

* Some other thread invokes [unpark](http://docs.google.com/java/util/concurrent/locks/LockSupport.html#unpark(java.lang.Thread)) with the current thread as the target; or
* Some other thread [interrupts](http://docs.google.com/java/lang/Thread.html#interrupt()) the current thread; or
* The specified deadline passes; or
* The call spuriously (that is, for no reason) returns.

This method does *not* report which of these caused the method to return. Callers should re-check the conditions which caused the thread to park in the first place. Callers may also determine, for example, the interrupt status of the thread, or the current time upon return.

**Parameters:**blocker - the synchronization object responsible for this thread parkingdeadline - the absolute time, in milliseconds from the Epoch, to wait until**Since:** 1.6

### getBlocker

public static [Object](http://docs.google.com/java/lang/Object.html) **getBlocker**([Thread](http://docs.google.com/java/lang/Thread.html) t)

Returns the blocker object supplied to the most recent invocation of a park method that has not yet unblocked, or null if not blocked. The value returned is just a momentary snapshot -- the thread may have since unblocked or blocked on a different blocker object.

**Returns:**the blocker**Since:** 1.6

### park

public static void **park**()

Disables the current thread for thread scheduling purposes unless the permit is available.

If the permit is available then it is consumed and the call returns immediately; otherwise the current thread becomes disabled for thread scheduling purposes and lies dormant until one of three things happens:

* Some other thread invokes [unpark](http://docs.google.com/java/util/concurrent/locks/LockSupport.html#unpark(java.lang.Thread)) with the current thread as the target; or
* Some other thread [interrupts](http://docs.google.com/java/lang/Thread.html#interrupt()) the current thread; or
* The call spuriously (that is, for no reason) returns.

This method does *not* report which of these caused the method to return. Callers should re-check the conditions which caused the thread to park in the first place. Callers may also determine, for example, the interrupt status of the thread upon return.

### parkNanos

public static void **parkNanos**(long nanos)

Disables the current thread for thread scheduling purposes, for up to the specified waiting time, unless the permit is available.

If the permit is available then it is consumed and the call returns immediately; otherwise the current thread becomes disabled for thread scheduling purposes and lies dormant until one of four things happens:

* Some other thread invokes [unpark](http://docs.google.com/java/util/concurrent/locks/LockSupport.html#unpark(java.lang.Thread)) with the current thread as the target; or
* Some other thread [interrupts](http://docs.google.com/java/lang/Thread.html#interrupt()) the current thread; or
* The specified waiting time elapses; or
* The call spuriously (that is, for no reason) returns.

This method does *not* report which of these caused the method to return. Callers should re-check the conditions which caused the thread to park in the first place. Callers may also determine, for example, the interrupt status of the thread, or the elapsed time upon return.

**Parameters:**nanos - the maximum number of nanoseconds to wait

### parkUntil

public static void **parkUntil**(long deadline)

Disables the current thread for thread scheduling purposes, until the specified deadline, unless the permit is available.

If the permit is available then it is consumed and the call returns immediately; otherwise the current thread becomes disabled for thread scheduling purposes and lies dormant until one of four things happens:

* Some other thread invokes [unpark](http://docs.google.com/java/util/concurrent/locks/LockSupport.html#unpark(java.lang.Thread)) with the current thread as the target; or
* Some other thread [interrupts](http://docs.google.com/java/lang/Thread.html#interrupt()) the current thread; or
* The specified deadline passes; or
* The call spuriously (that is, for no reason) returns.

This method does *not* report which of these caused the method to return. Callers should re-check the conditions which caused the thread to park in the first place. Callers may also determine, for example, the interrupt status of the thread, or the current time upon return.

**Parameters:**deadline - the absolute time, in milliseconds from the Epoch, to wait until

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/LockSupport.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/Lock.html)   [**NEXT CLASS**](http://docs.google.com/java/util/concurrent/locks/ReadWriteLock.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/concurrent/locks/LockSupport.html)    [**NO FRAMES**](http://docs.google.com/LockSupport.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#3znysh7) | DETAIL: FIELD | CONSTR | [METHOD](#tyjcwt) |

[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).