| | [**Overview**](http://docs.google.com/overview-summary.html) | **Package** | Class | [**Use**](http://docs.google.com/package-use.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 PACKAGE**](http://docs.google.com/java/util/package-summary.html)   [**NEXT PACKAGE**](http://docs.google.com/java/util/concurrent/atomic/package-summary.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/concurrent/package-summary.html)    [**NO FRAMES**](http://docs.google.com/package-summary.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |

## Package java.util.concurrent

Utility classes commonly useful in concurrent programming.

**See:**

[**Description**](#3znysh7)

| **Interface Summary** | |
| --- | --- |
| [**BlockingDeque<E>**](http://docs.google.com/java/util/concurrent/BlockingDeque.html) | A [Deque](http://docs.google.com/java/util/Deque.html) that additionally supports blocking operations that wait for the deque to become non-empty when retrieving an element, and wait for space to become available in the deque when storing an element. |
| [**BlockingQueue<E>**](http://docs.google.com/java/util/concurrent/BlockingQueue.html) | 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. |
| [**Callable<V>**](http://docs.google.com/java/util/concurrent/Callable.html) | A task that returns a result and may throw an exception. |
| [**CompletionService<V>**](http://docs.google.com/java/util/concurrent/CompletionService.html) | A service that decouples the production of new asynchronous tasks from the consumption of the results of completed tasks. |
| [**ConcurrentMap<K,V>**](http://docs.google.com/java/util/concurrent/ConcurrentMap.html) | A [Map](http://docs.google.com/java/util/Map.html) providing additional atomic putIfAbsent, remove, and replace methods. |
| [**ConcurrentNavigableMap<K,V>**](http://docs.google.com/java/util/concurrent/ConcurrentNavigableMap.html) | A [ConcurrentMap](http://docs.google.com/java/util/concurrent/ConcurrentMap.html) supporting [NavigableMap](http://docs.google.com/java/util/NavigableMap.html) operations, and recursively so for its navigable sub-maps. |
| [**Delayed**](http://docs.google.com/java/util/concurrent/Delayed.html) | A mix-in style interface for marking objects that should be acted upon after a given delay. |
| [**Executor**](http://docs.google.com/java/util/concurrent/Executor.html) | An object that executes submitted [Runnable](http://docs.google.com/java/lang/Runnable.html) tasks. |
| [**ExecutorService**](http://docs.google.com/java/util/concurrent/ExecutorService.html) | An [Executor](http://docs.google.com/java/util/concurrent/Executor.html) that provides methods to manage termination and methods that can produce a [Future](http://docs.google.com/java/util/concurrent/Future.html) for tracking progress of one or more asynchronous tasks. |
| [**Future<V>**](http://docs.google.com/java/util/concurrent/Future.html) | A Future represents the result of an asynchronous computation. |
| [**RejectedExecutionHandler**](http://docs.google.com/java/util/concurrent/RejectedExecutionHandler.html) | A handler for tasks that cannot be executed by a [ThreadPoolExecutor](http://docs.google.com/java/util/concurrent/ThreadPoolExecutor.html). |
| [**RunnableFuture<V>**](http://docs.google.com/java/util/concurrent/RunnableFuture.html) | A [Future](http://docs.google.com/java/util/concurrent/Future.html) that is [Runnable](http://docs.google.com/java/lang/Runnable.html). |
| [**RunnableScheduledFuture<V>**](http://docs.google.com/java/util/concurrent/RunnableScheduledFuture.html) | A [ScheduledFuture](http://docs.google.com/java/util/concurrent/ScheduledFuture.html) that is [Runnable](http://docs.google.com/java/lang/Runnable.html). |
| [**ScheduledExecutorService**](http://docs.google.com/java/util/concurrent/ScheduledExecutorService.html) | An [ExecutorService](http://docs.google.com/java/util/concurrent/ExecutorService.html) that can schedule commands to run after a given delay, or to execute periodically. |
| [**ScheduledFuture<V>**](http://docs.google.com/java/util/concurrent/ScheduledFuture.html) | A delayed result-bearing action that can be cancelled. |
| [**ThreadFactory**](http://docs.google.com/java/util/concurrent/ThreadFactory.html) | An object that creates new threads on demand. |

| **Class Summary** | |
| --- | --- |
| [**AbstractExecutorService**](http://docs.google.com/java/util/concurrent/AbstractExecutorService.html) | Provides default implementations of [ExecutorService](http://docs.google.com/java/util/concurrent/ExecutorService.html) execution methods. |
| [**ArrayBlockingQueue<E>**](http://docs.google.com/java/util/concurrent/ArrayBlockingQueue.html) | A bounded [blocking queue](http://docs.google.com/java/util/concurrent/BlockingQueue.html) backed by an array. |
| [**ConcurrentHashMap<K,V>**](http://docs.google.com/java/util/concurrent/ConcurrentHashMap.html) | A hash table supporting full concurrency of retrievals and adjustable expected concurrency for updates. |
| [**ConcurrentLinkedQueue<E>**](http://docs.google.com/java/util/concurrent/ConcurrentLinkedQueue.html) | An unbounded thread-safe [queue](http://docs.google.com/java/util/Queue.html) based on linked nodes. |
| [**ConcurrentSkipListMap<K,V>**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListMap.html) | A scalable concurrent [ConcurrentNavigableMap](http://docs.google.com/java/util/concurrent/ConcurrentNavigableMap.html) implementation. |
| [**ConcurrentSkipListSet<E>**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) | A scalable concurrent [NavigableSet](http://docs.google.com/java/util/NavigableSet.html) implementation based on a [ConcurrentSkipListMap](http://docs.google.com/java/util/concurrent/ConcurrentSkipListMap.html). |
| [**CopyOnWriteArrayList<E>**](http://docs.google.com/java/util/concurrent/CopyOnWriteArrayList.html) | A thread-safe variant of [ArrayList](http://docs.google.com/java/util/ArrayList.html) in which all mutative operations (add, set, and so on) are implemented by making a fresh copy of the underlying array. |
| [**CopyOnWriteArraySet<E>**](http://docs.google.com/java/util/concurrent/CopyOnWriteArraySet.html) | A [Set](http://docs.google.com/java/util/Set.html) that uses an internal [CopyOnWriteArrayList](http://docs.google.com/java/util/concurrent/CopyOnWriteArrayList.html) for all of its operations. |
| [**CountDownLatch**](http://docs.google.com/java/util/concurrent/CountDownLatch.html) | A synchronization aid that allows one or more threads to wait until a set of operations being performed in other threads completes. |
| [**CyclicBarrier**](http://docs.google.com/java/util/concurrent/CyclicBarrier.html) | A synchronization aid that allows a set of threads to all wait for each other to reach a common barrier point. |
| [**DelayQueue<E extends Delayed>**](http://docs.google.com/java/util/concurrent/DelayQueue.html) | An unbounded [blocking queue](http://docs.google.com/java/util/concurrent/BlockingQueue.html) of Delayed elements, in which an element can only be taken when its delay has expired. |
| [**Exchanger<V>**](http://docs.google.com/java/util/concurrent/Exchanger.html) | A synchronization point at which threads can pair and swap elements within pairs. |
| [**ExecutorCompletionService<V>**](http://docs.google.com/java/util/concurrent/ExecutorCompletionService.html) | A [CompletionService](http://docs.google.com/java/util/concurrent/CompletionService.html) that uses a supplied [Executor](http://docs.google.com/java/util/concurrent/Executor.html) to execute tasks. |
| [**Executors**](http://docs.google.com/java/util/concurrent/Executors.html) | Factory and utility methods for [Executor](http://docs.google.com/java/util/concurrent/Executor.html), [ExecutorService](http://docs.google.com/java/util/concurrent/ExecutorService.html), [ScheduledExecutorService](http://docs.google.com/java/util/concurrent/ScheduledExecutorService.html), [ThreadFactory](http://docs.google.com/java/util/concurrent/ThreadFactory.html), and [Callable](http://docs.google.com/java/util/concurrent/Callable.html) classes defined in this package. |
| [**FutureTask<V>**](http://docs.google.com/java/util/concurrent/FutureTask.html) | A cancellable asynchronous computation. |
| [**LinkedBlockingDeque<E>**](http://docs.google.com/java/util/concurrent/LinkedBlockingDeque.html) | An optionally-bounded [blocking deque](http://docs.google.com/java/util/concurrent/BlockingDeque.html) based on linked nodes. |
| [**LinkedBlockingQueue<E>**](http://docs.google.com/java/util/concurrent/LinkedBlockingQueue.html) | An optionally-bounded [blocking queue](http://docs.google.com/java/util/concurrent/BlockingQueue.html) based on linked nodes. |
| [**PriorityBlockingQueue<E>**](http://docs.google.com/java/util/concurrent/PriorityBlockingQueue.html) | An unbounded [blocking queue](http://docs.google.com/java/util/concurrent/BlockingQueue.html) that uses the same ordering rules as class [PriorityQueue](http://docs.google.com/java/util/PriorityQueue.html) and supplies blocking retrieval operations. |
| [**ScheduledThreadPoolExecutor**](http://docs.google.com/java/util/concurrent/ScheduledThreadPoolExecutor.html) | A [ThreadPoolExecutor](http://docs.google.com/java/util/concurrent/ThreadPoolExecutor.html) that can additionally schedule commands to run after a given delay, or to execute periodically. |
| [**Semaphore**](http://docs.google.com/java/util/concurrent/Semaphore.html) | A counting semaphore. |
| [**SynchronousQueue<E>**](http://docs.google.com/java/util/concurrent/SynchronousQueue.html) | A [blocking queue](http://docs.google.com/java/util/concurrent/BlockingQueue.html) in which each insert operation must wait for a corresponding remove operation by another thread, and vice versa. |
| [**ThreadPoolExecutor**](http://docs.google.com/java/util/concurrent/ThreadPoolExecutor.html) | An [ExecutorService](http://docs.google.com/java/util/concurrent/ExecutorService.html) that executes each submitted task using one of possibly several pooled threads, normally configured using [Executors](http://docs.google.com/java/util/concurrent/Executors.html) factory methods. |
| [**ThreadPoolExecutor.AbortPolicy**](http://docs.google.com/java/util/concurrent/ThreadPoolExecutor.AbortPolicy.html) | A handler for rejected tasks that throws a RejectedExecutionException. |
| [**ThreadPoolExecutor.CallerRunsPolicy**](http://docs.google.com/java/util/concurrent/ThreadPoolExecutor.CallerRunsPolicy.html) | A handler for rejected tasks that runs the rejected task directly in the calling thread of the execute method, unless the executor has been shut down, in which case the task is discarded. |
| [**ThreadPoolExecutor.DiscardOldestPolicy**](http://docs.google.com/java/util/concurrent/ThreadPoolExecutor.DiscardOldestPolicy.html) | A handler for rejected tasks that discards the oldest unhandled request and then retries execute, unless the executor is shut down, in which case the task is discarded. |
| [**ThreadPoolExecutor.DiscardPolicy**](http://docs.google.com/java/util/concurrent/ThreadPoolExecutor.DiscardPolicy.html) | A handler for rejected tasks that silently discards the rejected task. |

| **Enum Summary** | |
| --- | --- |
| [**TimeUnit**](http://docs.google.com/java/util/concurrent/TimeUnit.html) | A TimeUnit represents time durations at a given unit of granularity and provides utility methods to convert across units, and to perform timing and delay operations in these units. |

| **Exception Summary** | |
| --- | --- |
| [**BrokenBarrierException**](http://docs.google.com/java/util/concurrent/BrokenBarrierException.html) | Exception thrown when a thread tries to wait upon a barrier that is in a broken state, or which enters the broken state while the thread is waiting. |
| [**CancellationException**](http://docs.google.com/java/util/concurrent/CancellationException.html) | Exception indicating that the result of a value-producing task, such as a [FutureTask](http://docs.google.com/java/util/concurrent/FutureTask.html), cannot be retrieved because the task was cancelled. |
| [**ExecutionException**](http://docs.google.com/java/util/concurrent/ExecutionException.html) | Exception thrown when attempting to retrieve the result of a task that aborted by throwing an exception. |
| [**RejectedExecutionException**](http://docs.google.com/java/util/concurrent/RejectedExecutionException.html) | Exception thrown by an [Executor](http://docs.google.com/java/util/concurrent/Executor.html) when a task cannot be accepted for execution. |
| [**TimeoutException**](http://docs.google.com/java/util/concurrent/TimeoutException.html) | Exception thrown when a blocking operation times out. |

## Package java.util.concurrent Description

Utility classes commonly useful in concurrent programming. This package includes a few small standardized extensible frameworks, as well as some classes that provide useful functionality and are otherwise tedious or difficult to implement. Here are brief descriptions of the main components. See also the locks and atomic packages.

## Executors

**Interfaces.** [Executor](http://docs.google.com/java/util/concurrent/Executor.html) is a simple standardized interface for defining custom thread-like subsystems, including thread pools, asynchronous IO, and lightweight task frameworks. Depending on which concrete Executor class is being used, tasks may execute in a newly created thread, an existing task-execution thread, or the thread calling execute(), and may execute sequentially or concurrently. [ExecutorService](http://docs.google.com/java/util/concurrent/ExecutorService.html) provides a more complete asynchronous task execution framework. An ExecutorService manages queuing and scheduling of tasks, and allows controlled shutdown. The [ScheduledExecutorService](http://docs.google.com/java/util/concurrent/ScheduledExecutorService.html) subinterface and associated interfaces add support for delayed and periodic task execution. ExecutorServices provide methods arranging asynchronous execution of any function expressed as [Callable](http://docs.google.com/java/util/concurrent/Callable.html), the result-bearing analog of [Runnable](http://docs.google.com/java/lang/Runnable.html). A [Future](http://docs.google.com/java/util/concurrent/Future.html) returns the results of a function, allows determination of whether execution has completed, and provides a means to cancel execution. A [RunnableFuture](http://docs.google.com/java/util/concurrent/RunnableFuture.html) is a Future that possesses a run method that upon execution, sets its results.

**Implementations.** Classes [ThreadPoolExecutor](http://docs.google.com/java/util/concurrent/ThreadPoolExecutor.html) and [ScheduledThreadPoolExecutor](http://docs.google.com/java/util/concurrent/ScheduledThreadPoolExecutor.html) provide tunable, flexible thread pools. The [Executors](http://docs.google.com/java/util/concurrent/Executors.html) class provides factory methods for the most common kinds and configurations of Executors, as well as a few utility methods for using them. Other utilities based on Executors include the concrete class [FutureTask](http://docs.google.com/java/util/concurrent/FutureTask.html) providing a common extensible implementation of Futures, and [ExecutorCompletionService](http://docs.google.com/java/util/concurrent/ExecutorCompletionService.html), that assists in coordinating the processing of groups of asynchronous tasks.

## Queues

The java.util.concurrent [ConcurrentLinkedQueue](http://docs.google.com/java/util/concurrent/ConcurrentLinkedQueue.html) class supplies an efficient scalable thread-safe non-blocking FIFO queue. Five implementations in java.util.concurrent support the extended [BlockingQueue](http://docs.google.com/java/util/concurrent/BlockingQueue.html) interface, that defines blocking versions of put and take: [LinkedBlockingQueue](http://docs.google.com/java/util/concurrent/LinkedBlockingQueue.html), [ArrayBlockingQueue](http://docs.google.com/java/util/concurrent/ArrayBlockingQueue.html), [SynchronousQueue](http://docs.google.com/java/util/concurrent/SynchronousQueue.html), [PriorityBlockingQueue](http://docs.google.com/java/util/concurrent/PriorityBlockingQueue.html), and [DelayQueue](http://docs.google.com/java/util/concurrent/DelayQueue.html). The different classes cover the most common usage contexts for producer-consumer, messaging, parallel tasking, and related concurrent designs. The [BlockingDeque](http://docs.google.com/java/util/concurrent/BlockingDeque.html) interface extends BlockingQueue to support both FIFO and LIFO (stack-based) operations. Class [LinkedBlockingDeque](http://docs.google.com/java/util/concurrent/LinkedBlockingDeque.html) provides an implementation.

## Timing

The [TimeUnit](http://docs.google.com/java/util/concurrent/TimeUnit.html) class provides multiple granularities (including nanoseconds) for specifying and controlling time-out based operations. Most classes in the package contain operations based on time-outs in addition to indefinite waits. In all cases that time-outs are used, the time-out specifies the minimum time that the method should wait before indicating that it timed-out. Implementations make a "best effort" to detect time-outs as soon as possible after they occur. However, an indefinite amount of time may elapse between a time-out being detected and a thread actually executing again after that time-out. All methods that accept timeout parameters treat values less than or equal to zero to mean not to wait at all. To wait "forever", you can use a value of Long.MAX\_VALUE.

## Synchronizers

Four classes aid common special-purpose synchronization idioms. [Semaphore](http://docs.google.com/java/util/concurrent/Semaphore.html) is a classic concurrency tool. [CountDownLatch](http://docs.google.com/java/util/concurrent/CountDownLatch.html) is a very simple yet very common utility for blocking until a given number of signals, events, or conditions hold. A [CyclicBarrier](http://docs.google.com/java/util/concurrent/CyclicBarrier.html) is a resettable multiway synchronization point useful in some styles of parallel programming. An [Exchanger](http://docs.google.com/java/util/concurrent/Exchanger.html) allows two threads to exchange objects at a rendezvous point, and is useful in several pipeline designs.

## Concurrent Collections

Besides Queues, this package supplies Collection implementations designed for use in multithreaded contexts: [ConcurrentHashMap](http://docs.google.com/java/util/concurrent/ConcurrentHashMap.html), [ConcurrentSkipListMap](http://docs.google.com/java/util/concurrent/ConcurrentSkipListMap.html), [ConcurrentSkipListSet](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html), [CopyOnWriteArrayList](http://docs.google.com/java/util/concurrent/CopyOnWriteArrayList.html), and [CopyOnWriteArraySet](http://docs.google.com/java/util/concurrent/CopyOnWriteArraySet.html). When many threads are expected to access a given collection, a ConcurrentHashMap is normally preferable to a synchronized HashMap, and a ConcurrentSkipListMap is normally preferable to a synchronized TreeMap. A CopyOnWriteArrayList is preferable to a synchronized ArrayList when the expected number of reads and traversals greatly outnumber the number of updates to a list.

The "Concurrent" prefix used with some classes in this package is a shorthand indicating several differences from similar "synchronized" classes. For example java.util.Hashtable and Collections.synchronizedMap(new HashMap()) are synchronized. But [ConcurrentHashMap](http://docs.google.com/java/util/concurrent/ConcurrentHashMap.html) is "concurrent". A concurrent collection is thread-safe, but not governed by a single exclusion lock. In the particular case of ConcurrentHashMap, it safely permits any number of concurrent reads as well as a tunable number of concurrent writes. "Synchronized" classes can be useful when you need to prevent all access to a collection via a single lock, at the expense of poorer scalability. In other cases in which multiple threads are expected to access a common collection, "concurrent" versions are normally preferable. And unsynchronized collections are preferable when either collections are unshared, or are accessible only when holding other locks.

Most concurrent Collection implementations (including most Queues) also differ from the usual java.util conventions in that their Iterators provide *weakly consistent* rather than fast-fail traversal. A weakly consistent iterator is thread-safe, but does not necessarily freeze the collection while iterating, so it may (or may not) reflect any updates since the iterator was created.

## Memory Consistency Properties

[Chapter 17 of the Java Language Specification](http://java.sun.com/docs/books/jls/third_edition/html/memory.html) defines the *happens-before* relation on memory operations such as reads and writes of shared variables. The results of a write by one thread are guaranteed to be visible to a read by another thread only if the write operation *happens-before* the read operation. The synchronized and volatile constructs, as well as the Thread.start() and Thread.join() methods, can form *happens-before* relationships. In particular:

* Each action in a thread *happens-before* every action in that thread that comes later in the program's order.
* An unlock (synchronized block or method exit) of a monitor *happens-before* every subsequent lock (synchronized block or method entry) of that same monitor. And because the *happens-before* relation is transitive, all actions of a thread prior to unlocking *happen-before* all actions subsequent to any thread locking that monitor.
* A write to a volatile field *happens-before* every subsequent read of that same field. Writes and reads of volatile fields have similar memory consistency effects as entering and exiting monitors, but do *not* entail mutual exclusion locking.
* A call to start on a thread *happens-before* any action in the started thread.
* All actions in a thread *happen-before* any other thread successfully returns from a join on that thread.

The methods of all classes in java.util.concurrent and its subpackages extend these guarantees to higher-level synchronization. In particular:

* Actions in a thread prior to placing an object into any concurrent collection *happen-before* actions subsequent to the access or removal of that element from the collection in another thread.
* Actions in a thread prior to the submission of a Runnable to an Executor *happen-before* its execution begins. Similarly for Callables submitted to an ExecutorService.
* Actions taken by the asynchronous computation represented by a Future *happen-before* actions subsequent to the retrieval of the result via Future.get() in another thread.
* Actions prior to "releasing" synchronizer methods such as Lock.unlock, Semaphore.release, and CountDownLatch.countDown *happen-before* actions subsequent to a successful "acquiring" method such as Lock.lock, Semaphore.acquire, Condition.await, and CountDownLatch.await on the same synchronizer object in another thread.
* For each pair of threads that successfully exchange objects via an Exchanger, actions prior to the exchange() in each thread *happen-before* those subsequent to the corresponding exchange() in another thread.
* Actions prior to calling CyclicBarrier.await *happen-before* actions performed by the barrier action, and actions performed by the barrier action *happen-before* actions subsequent to a successful return from the corresponding await in other threads.

**Since:** 1.5

| | [**Overview**](http://docs.google.com/overview-summary.html) | **Package** | Class | [**Use**](http://docs.google.com/package-use.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 PACKAGE**](http://docs.google.com/java/util/package-summary.html)   [**NEXT PACKAGE**](http://docs.google.com/java/util/concurrent/atomic/package-summary.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/concurrent/package-summary.html)    [**NO FRAMES**](http://docs.google.com/package-summary.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |

[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).