java.util.concurrent

**Interface Future<V>**

* **Type Parameters:**

V - The result type returned by this Future's get method

**All Known Subinterfaces:**

[Response](https://docs.oracle.com/javase/7/docs/api/javax/xml/ws/Response.html)<T>, [RunnableFuture](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/RunnableFuture.html)<V>, [RunnableScheduledFuture](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/RunnableScheduledFuture.html)<V>, [ScheduledFuture](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ScheduledFuture.html)<V>

**All Known Implementing Classes:**

[ForkJoinTask](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ForkJoinTask.html), [FutureTask](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/FutureTask.html), [RecursiveAction](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/RecursiveAction.html), [RecursiveTask](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/RecursiveTask.html), [SwingWorker](https://docs.oracle.com/javase/7/docs/api/javax/swing/SwingWorker.html)

public interface **Future<V>**

A Future represents the result of an asynchronous computation. Methods are provided to check if the computation is complete, to wait for its completion, and to retrieve the result of the computation. The result can only be retrieved using method get when the computation has completed, blocking if necessary until it is ready. Cancellation is performed by the cancel method. Additional methods are provided to determine if the task completed normally or was cancelled. Once a computation has completed, the computation cannot be cancelled. If you would like to use a Future for the sake of cancellability but not provide a usable result, you can declare types of the form Future<?> and return null as a result of the underlying task.

**Sample Usage** (Note that the following classes are all made-up.)

interface ArchiveSearcher { String search(String target); }

class App {

ExecutorService executor = ...

ArchiveSearcher searcher = ...

void showSearch(final String target)

throws InterruptedException {

Future<String> future

= executor.submit(new Callable<String>() {

public String call() {

return searcher.search(target);

}});

displayOtherThings(); // do other things while searching

try {

displayText(future.get()); // use future

} catch (ExecutionException ex) { cleanup(); return; }

}

}

The [FutureTask](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/FutureTask.html) class is an implementation of Future that implements Runnable, and so may be executed by an Executor. For example, the above construction with submit could be replaced by:

FutureTask<String> future =

new FutureTask<String>(new Callable<String>() {

public String call() {

return searcher.search(target);

}});

executor.execute(future);

Memory consistency effects: Actions taken by the asynchronous computation [*happen-before*](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/package-summary.html#MemoryVisibility) actions following the corresponding Future.get() in another thread.

**Since:**

1.5

**See Also:**

[FutureTask](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/FutureTask.html), [Executor](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Executor.html)

* + **Method Summary**

|  |  |
| --- | --- |
| **Methods** | |
| **Modifier and Type** | **Method and Description** |
| boolean | [**cancel**](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Future.html#cancel(boolean))(boolean mayInterruptIfRunning)  Attempts to cancel execution of this task. |
| [**V**](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Future.html) | [**get**](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Future.html#get())()  Waits if necessary for the computation to complete, and then retrieves its result. |
| [**V**](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Future.html) | [**get**](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Future.html#get(long,%20java.util.concurrent.TimeUnit))(long timeout, [**TimeUnit**](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/TimeUnit.html) unit)  Waits if necessary for at most the given time for the computation to complete, and then retrieves its result, if available. |
| boolean | [**isCancelled**](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Future.html#isCancelled())()  Returns true if this task was cancelled before it completed normally. |
| boolean | [**isDone**](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Future.html#isDone())()  Returns true if this task completed. |

* + **Method Detail**
    - **cancel**

boolean cancel(boolean mayInterruptIfRunning)

Attempts to cancel execution of this task. This attempt will fail if the task has already completed, has already been cancelled, or could not be cancelled for some other reason. If successful, and this task has not started when cancel is called, this task should never run. If the task has already started, then the mayInterruptIfRunning parameter determines whether the thread executing this task should be interrupted in an attempt to stop the task.

After this method returns, subsequent calls to [isDone()](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Future.html#isDone()) will always return true. Subsequent calls to [isCancelled()](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Future.html#isCancelled()) will always return true if this method returned true.

**Parameters:**

mayInterruptIfRunning - true if the thread executing this task should be interrupted; otherwise, in-progress tasks are allowed to complete

**Returns:**

false if the task could not be cancelled, typically because it has already completed normally; true otherwise

* + - **isCancelled**

boolean isCancelled()

Returns true if this task was cancelled before it completed normally.

**Returns:**

true if this task was cancelled before it completed

* + - **isDone**

boolean isDone()

Returns true if this task completed. Completion may be due to normal termination, an exception, or cancellation -- in all of these cases, this method will return true.

**Returns:**

true if this task completed

* + - **get**
    - [V](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Future.html) get()
    - throws [InterruptedException](https://docs.oracle.com/javase/7/docs/api/java/lang/InterruptedException.html),

[ExecutionException](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ExecutionException.html)

Waits if necessary for the computation to complete, and then retrieves its result.

**Returns:**

the computed result

**Throws:**

[CancellationException](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/CancellationException.html) - if the computation was cancelled

[ExecutionException](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ExecutionException.html) - if the computation threw an exception

[InterruptedException](https://docs.oracle.com/javase/7/docs/api/java/lang/InterruptedException.html) - if the current thread was interrupted while waiting

* + - **get**
    - [V](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Future.html) get(long timeout,
    - [TimeUnit](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/TimeUnit.html) unit)
    - throws [InterruptedException](https://docs.oracle.com/javase/7/docs/api/java/lang/InterruptedException.html),
    - [ExecutionException](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ExecutionException.html),

[TimeoutException](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/TimeoutException.html)

Waits if necessary for at most the given time for the computation to complete, and then retrieves its result, if available.

**Parameters:**

timeout - the maximum time to wait

unit - the time unit of the timeout argument

**Returns:**

the computed result

**Throws:**

[CancellationException](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/CancellationException.html) - if the computation was cancelled

[ExecutionException](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ExecutionException.html) - if the computation threw an exception

[InterruptedException](https://docs.oracle.com/javase/7/docs/api/java/lang/InterruptedException.html) - if the current thread was interrupted while waiting

[TimeoutException](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/TimeoutException.html) - if the wait timed out