A Java *Future*, java.util.concurrent.Future, represents the result of an asynchronous computation. When the asynchronous task is created, a Java Future object is returned. This Future object functions as a handle to the result of the asynchronous task. Once the asynchronous task completes, the result can be accessed via the Future object returned when the task was started.

Some of Java's built-in concurrency utilities, like e.g. the [**Java ExecutorService**](http://tutorials.jenkov.com/java-util-concurrent/executorservice.html), return a Java Future object from some of their methods. In the case of the ExecutorService, it returns a Future when you submit a Callable for it to execute concurrently (asynchronously).

**Java Future Interface Definition**

To understand how the Java Future interface works, here is an approximation of the interface definition:

public interface Future<V> {

boolean cancel(boolean mayInterruptIfRunning)

V get();

V get(long timeout, TimeUnit unit);

boolean isCancelled();

boolean isDone();

}

Each of these methods will be covered in later sections - but as you can see, the Java Future interface is not that advanced.

**Get Result From Future**

As mentioned earlier, a Java Future represents the result of an asynchronous task. To obtain the result, you call one of the two get() methods on the Future. The get() methods both return an Object, but the return type can also be a generic return type (meaning an object of a specific class, and not just an Object). Here is an example of obtaining the result from a Java Future via its get() method:

Future future = ... // get Future by starting async task

// do something else, until ready to check result via Future

// get result from Future

try {

Object result = future.get();

} catch (InterruptedException e) {

e.printStackTrace();

} catch (ExecutionException e) {

e.printStackTrace();

}

If you call the get() method before the asynchronous task has completed, the get() method will block until the result is ready.

There is a version of the get() method which can time out after an amount of time has passed which you can specify via method parameters. Here is an example of calling that get() version:

try {

Object result =

future.get(1000, TimeUnit.MILLISECONDS);

} catch (InterruptedException e) {

} catch (ExecutionException e) {

} catch (TimeoutException e) {

// thrown if timeout time interval passes

// before a result is available.

}

The example above waits for a maximum of 1000 milliseconds for the result to be available in the Future. If no result is available within 1000 milliseconds, a TimeoutException is thrown.

**Cancel Task via Future cancel()**

You can cancel the asynchronous task represented by a Java Future instance by calling the Future cancel() method. The asynchronous task execution must be implemented in to support cancellation. Without such support, calling cancel() will have no effect. Here is an example of canceling a task via the Java Future cancel() method:

future.cancel();

**Check if Task is Done**

You can check if the asynchronous task is done (and a result available) by calling the Future isDone() method. Here is an example of calling the Java Future isDone() method:

Future future = ... // Get Future from somewhere

if(future.isDone()) {

Object result = future.get();

} else {

// do something else

}

**Check if Task is Cancelled**

It is also possible to check if the asynchronous task represented by a Java Future is cancelled. You check that by calling the Future isCancelled() method. Here is an example of checking if a task was cancelled:

Future future = ... // get Future from somewhere

if(future.isCancelled()) {

} else {

}