

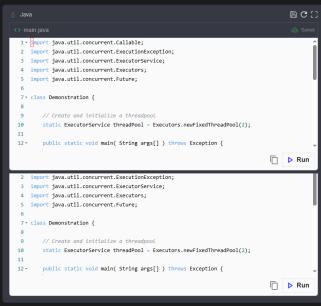
Java Multithreading for Senior Engineering Interviews / ... / Future Interface

## **Future Interface**

This lesson discusses the Future interface.

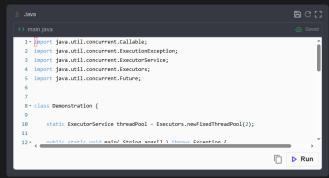
## **Future Interface**

The Future interface is used to represent the result of an asynchronous computation. The interface also provides methods to check the status of a submitted task and also allows the task to be cancelled if possible. Without further ado, let's dive into an example and see how callable and future objects work in tandem. We'll continue with our sumTask example from the previous lesson.



Using Future and Callable Together

Thread pools implementing the ExecutorService return a future for their task submission methods. In the above code on **line 29** we get back a future when submitting our task. We retrieve the result of the task by invoking the get method on the future. The get method will return the result or throw an instance of ExecutionException. Let's see an example of that now.

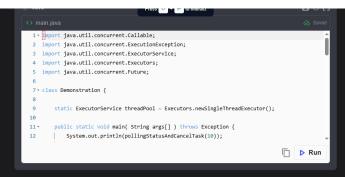


Callable throwing Exception

On line 31 of the above code, we make a get method call. The method throws an execution exception, which we catch. The reason for the exception can be determined by using the getcause method of the execution exception. If you run the above snippet, you'll see it prints the runtime exception that we throw on line 24.

The get method is a blocking call. It'll block till the task completes. We can also write a polling version, where we poll periodically to check if the task is complete or not. Future also allows us to cancel tasks. If a task has been submitted but not yet executed, then it'll be cancelled. However, if a task is currently running, then it may or may not be cancellable. We'll discuss cancelling tasks in detail in future lessons.

Below is an example where we create two tasks. We poll to check if the task has completed. Also, we cancel the second submitted task.



Note the following about the above code

- On lines 44 and 45 we submit two tasks for execution.
- line 45 the second task submitted doesn't return any value so the future is parametrized with void.
- On line 52, we cancel the second task. Since our thread pool consists of a single thread
  and the first task sleeps for a bit before it starts executing, we can assume that the
  second task will not have started executing and can be cancelled. This is verified by
  checking for and printing the value of the iscancelled method later in the program.
- On lines 56 58, we repeatedly poll for the status of the first task.

The final output of the program shows messages from polling and the status of the second task cancellation request.

## **FutureTask**

Java also provides an implementation of the future interface called the FutureTask. It can wrap a callable or runnable object and in turn be submitted to an executor. Though, the class may not be very useful if you don't intend to create customized tasks but we mention it for the sake of completeness.

