- CompletionService Interface
- ThreadLocal
- ThreadLocalRandom
- CountDownLatch
- CyclicBarrier
- Concurrent Collections
- ConcurrentHashMap
- ConcurrentModificationException

Practice Mock Interview →

CancellationException

Learn the reasons that cause CancellationException to be thrown.

*If you are interviewing, consider buying our **number#1** course for **Java Multithreading Interviews**.*

Overview

`CancellationException` is thrown to indicate that the output or result of a value producing task can't be retrieved because it was cancelled. `CancellationException` is an unchecked exception and extends `IllegalStateException`, which in turn extends the `RuntimeException`. Some of the classes that throw `CancellationException` exception are:

- FutureTask
- CompletableFuture
- ForkJoinTask

Example

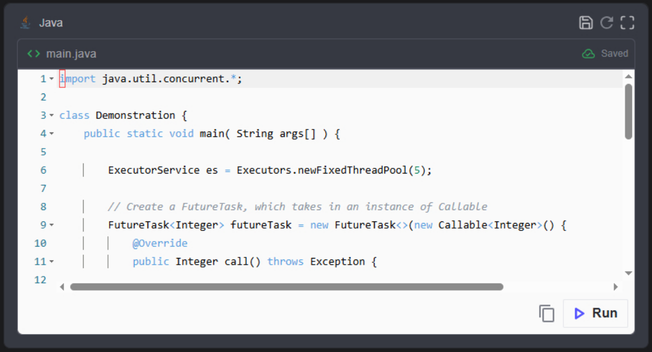
The following program demonstrates a scenario where an instance of `CancellationException` is thrown by the program. We create a `FutureTask` that has the thread sleep for one second intervals for a total of one hour. The main thread submits the task to the executor service, waits for three seconds and then attempts to cancel the task.Next when the main thread attempts to retrieve the result of the task by invoking the

- ForkJoinTask

Example

The following program demonstrates a scenario where an instance of `CancellationException` is thrown by the program. We create a `FutureTask` that has the thread sleep for one second intervals for a total of one hour. The main thread submits the task to the executor service, waits for three seconds and then attempts to cancel the task.Next when the main thread attempts to retrieve the result of the task by invoking the

`get()` method on the task object, `CancellationException` exception is thrown.



```java
1  import java.util.concurrent.*;
2
3  class Demonstration {
4      public static void main( String args[] ) {
5
6          ExecutorService es = Executors.newFixedThreadPool(5);
7
8          // Create a FutureTask, which takes in an instance of Callable
9          FutureTask<Integer> futureTask = new FutureTask<>(new Callable<Integer>() {
10             @Override
11             public Integer call() throws Exception {
12
```

Some other nuances about the above program include:

- If the main thread doesn't invoke `task.get()` then `CancellationException` isn't thrown even if the main thread invokes `cancel()` on the task object.
- The `cancel()` method on the task object takes in a boolean `mayInterruptIfRunning`, indicating if the task should be interrupted in case it is running. In the above program if we pass `false` instead of `true` on line#29, the main thread will still see the `CancellationException` being thrown by the `get()` method, however, the task keeps running and the print statement on line#18 doesn't appear in the program output.

← Back lesson
TimeoutException

☑ Mark As Completed

Next →
ExecutionException