Testing Coroutines and Asynchronous Calls

We'll cover the following

- Making asynchronous calls
- Finding the class name
- Mocking the class

The `getAirportStatus()` function is making synchronous calls, one at a time, to the `getAirportData()` function. If we receive a large number of airport codes, then making blocking calls, one by one, won't be efficient. If we make the calls to `getAirportData()` asynchronous, using coroutines, then we can get a better throughput. Of course, we'll have to test first and then write the code for asynchronous execution.

Making asynchronous calls

For `getAirportStatus()` to make asynchronous calls, we have to do three things. First, mark the function with the `suspend` keyword. Then execute the body of the function in the context of a `Dispatchers.IO` thread pool. Finally, embed the calls to `getAirportData()` within `async` calls and `await` for the results by applying the techniques we saw in Chapter 17, Asynchronous Programming. We'll write a test for each of these three steps and then implement the code.

The instant we mark the `getAirportStatus()` function with `suspend`, the previous test will fail compilation with the error "Suspension functions can be called only within the coroutine body." To address this, let's go back to the previous test and embed the call to `getAirportStatus()` within a call to `runBlocking()`, like so:

```
// AirportStatusTest.kt
runBlocking { getAirportStatus(input) shouldBe result }
```

Remember to import the necessary coroutines library for this code to compile:

```
// AirportStatusTest.kt
```

```
import kotlinx.coroutines.*
```

Now we can mark the function `getAirportStatus()` with the `suspend` keyword.

```
// AirportStatus.kt
suspend fun getAirportStatus(airportCodes: List<String>): List<Airport> =
  Airport.sort(
    airportCodes.map { code -> Airport.getAirportData(code) })
```

Before we can make an asynchronous call, we have to decide which thread pool to run the request in. Since `getAirportData()` will be making a call to a remote web service, it's appropriate to run the request in an IO thread pool. For this reason, we'll embed the body of `getAirportStatus()` within a call to `withContext()` so the code will run in the coroutine context we provide to that function. In short, our test needs to verify that `getAirportStatus()` makes a call to `withContext()`. How in the world can we do that?

We can write an interaction test by mocking the `withContext()` function. But that requires a bit of digging in. The `withContext()` function is a top-level function defined in the `kotlinx.coroutines` package. Sadly, we can't simply tell the Mockk library to mock the `withContext` function, as it won't know which function we're referring to. Even though Kotlin has top-level functions, at the bytecode level they don't reside directly in the package, but within a class. We have to track down which class, in the bytecode, the top-level function `withContext()` has been compiled into and ask Mockk to mock that class—yikes, that's going to take some effort.

Finding the class name

Visit the documentation for the `withContext()` function, and click the source link that is next to the return type `T` of the function. You'll notice that the file that has this function is `Builders.common.kt`, and in that file `withContext()` is defined as a top-level function. Now we have to find what class this code is compiled into. For this, you'll need the super-investigative skills of Sherlock Holmes and the `jar` tool that is part of the JDK. Download the jar file `kotlinx-coroutines-core-1.2.2.jar` by clicking the *jar* link in the Maven repository page, and use the `jar -tf` command, on the command line, to find the class name. If you're on a Unix-like system, use this command:

```
jar -tf kotlinx-coroutines-core-1.2.2.jar | grep Builders | grep common
```

If you're on Windows, then use the following command:

```
jar -tf kotlinx-coroutines-core-1.2.2.jar | Find "Builders" | Find "common"
```

The fruit of the that effort is the output:

```
kotlinx/coroutines/BuildersKt__Builders_commonKt.class
```

This tells us the code in the file `Builders.common.kt` is compiled into the class named `BuildersKt__Builders_commonKt`. That's the class we should mock to replace the `withContext()` method with the fake implementation in test.

Let's write the test for verifying that `getAirportStatus()` is calling `withContext()` with the appropriate arguments.

```
"getAirportStatus runs in the Dispatcher.IO context" {

    mockkStatic("kotlinx.coroutines.BuildersKt__Builders_commonKt")

    coEvery {
        withContext<List<Airport>>(
        context = Dispatchers.IO, block = captureCoroutine())
    } answers {
        listOf(iah)
    }

    getAirportStatus(listOf("IAH")) shouldBe listOf(iah)

    coVerify {
        withContext<List<Airport>>(Dispatchers.IO, block = any())
    }
}
```

AirportStatusTest.kt

Mocking the class

Since the top-level function `withContext()` is compiled as a static method in the bytecode within the class `BuildersKt__Builders_commonKt`, we ask Mockk to mock that class. Then, using Mockk's `coEvery` function, which is like the `every()` function but for mocking functions that use coroutines, we mock the `withContext()` function. The `withContext()` function takes two parameters: `CoroutineContext` and a lambda to be executed as a coroutine. We pass `Dispatchers.IO` for the first argument context and Mockk's special

`captureCoroutine()` as the second argument block. As the name alludes to, `captureCoroutine()` serves as a place holder for the coroutine that is passed to `withContext()` by the caller—that is, the code being tested. In response to the call to `withContext()`, our fake implementation will return a canned response of a list with `iah` Airport.

After the `coEvery()` call, we call the code under test and verify that the result of the call is the expected list. Finally, we verify, using `coVerify()`, that the `withContext()` function was in fact called by the code being tested.

Running the tests now will fail because our `getAirportStatus()` function isn't calling the `withContext()` function. Let's change the function to make the test pass.

```
package com.agiledeveloper.airportstatus

import kotlinx.coroutines.*

suspend fun getAirportStatus(airportCodes: List<String>): List<Airport> =
  withContext(Dispatchers.IO) {
    Airport.sort(
      airportCodes.map { code -> Airport.getAirportData(code) })
  }
```

AirportStatus.kt

With this change, the test will pass. We're now ready to make asynchronous the call to `getAirportData()` within `getAirportStatus()`.

```
"getAirportStatus calls getAirportData asynchronously" {

    mockkStatic("kotlinx.coroutines.BuildersKt__Builders_commonKt")

    coEvery {
        any<CoroutineScope>().async<Airport>(
        context = any(), block = captureCoroutine())
    } answers {
        CompletableDeferred(iad)
    }

    getAirportStatus(listOf("IAD")) shouldBe listOf(iad)

    coVerify {
        any<CoroutineScope>().async<Airport>(context = any(), block = any())
    }
}
```

AirportStatusTest.kt

In this test we again mock the `BuildersKt__Builders_commonKt` class, but this time

we mock the `async()` function to receive any object for context and a coroutine for the second argument named `block`. In response to the call, we return a `Deferred<Airport>` object that holds the `iad` mock instance. After the mock is arranged, we call the function under test and verify the result is what we expected. Finally, we verify that the `async()` function was called by the code under test.

To make the above test pass, let's modify the `getAirportStatus()` function.

```
package com.agiledeveloper.airportstatus

import kotlinx.coroutines.*

suspend fun getAirportStatus(airportCodes: List<String>): List<Airport> =
  withContext(Dispatchers.IO) {
    Airport.sort(
      airportCodes.map { code -> async { Airport.getAirportData(code) } }
        .map { response -> response.await() })
  }
```

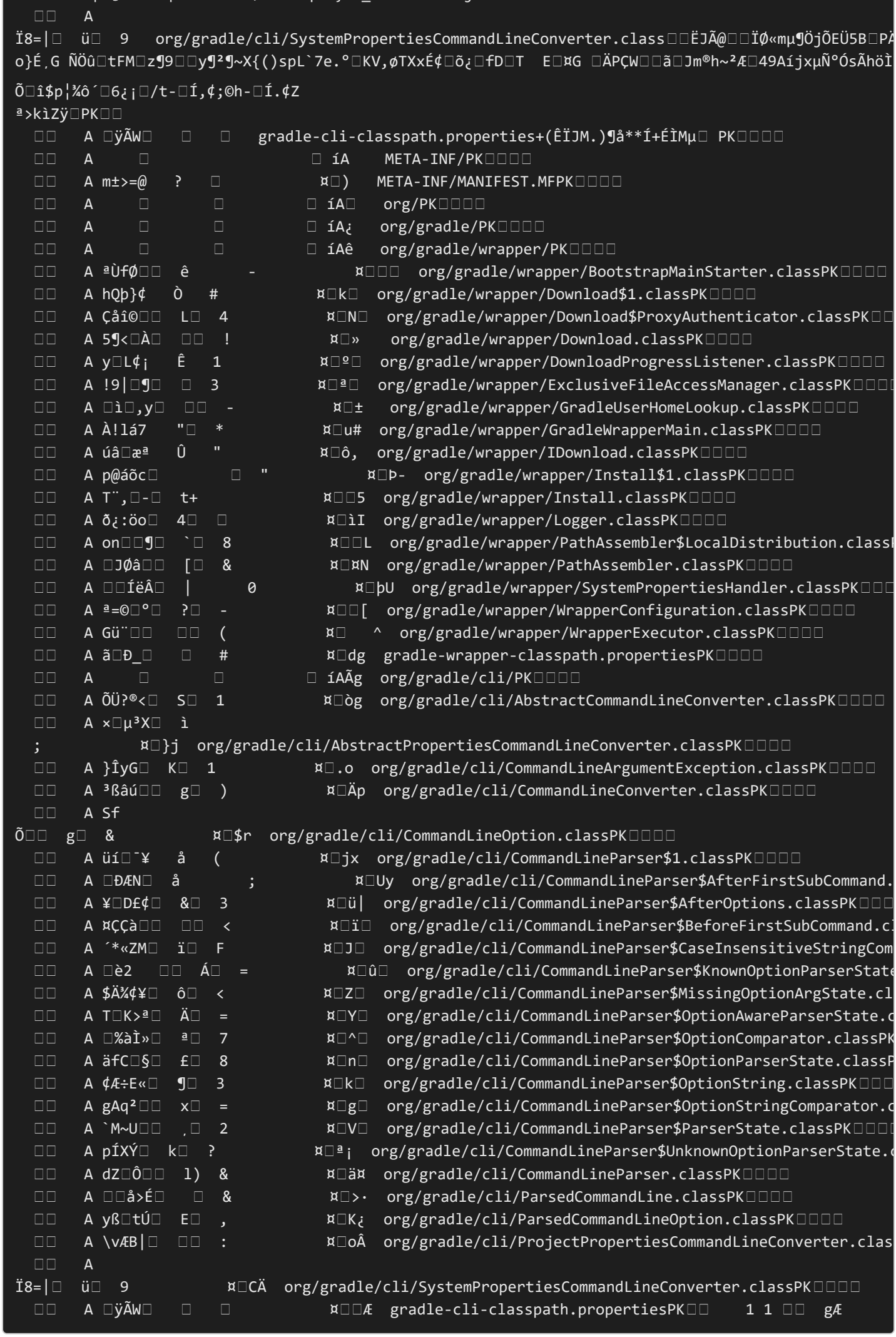
AirportStatus.kt

Instead of making a synchronous call to `getAirportData()`, the `getAirportStatus()` now wraps the call within `async()`. The call will now run in the `Dispatchers.IO` thread pool. After dispatching the calls to `getAirportData()` for each airport code in the given list `airportCodes`, the `getAirportStatus()` awaits the response using the `await()` function call on the response returned by `async()`, the `Deferred<Airport>` object.

Run the test and make sure it passes.

```
PK□□
  □□   A     □            META-INF/□ PK□□
  □□   A m±>=@   ?   □   META-INF/MANIFEST.MFóMÌËLK-.ÑK-*ÎÌÏ³R0Ô3àåòÌ-ÈIÍMÍ+I,□
ê□d□ä¤Z),□%¦ä¤*□□%□□¤□ñrñr□ PK□□
  □□   A     □       □   org/□ PK□□
  □□   A     □       □   org/gradle/□ PK□□
  □□   A     □       □   org/gradle/wrapper/□ PK□□
  □□   A ªÙfØ□□  ê      -   org/gradle/wrapper/BootstrapMainStarter.class□V[W□W□þ□□d`□□c@□□□7
?(□â□□□LÊxKÆE□¼,¤à2®(□ÂU±L+□!£`□³b¹Ö□ëBòv□ædde□(èÂ¼□□□g±(ãÆNÄð□□\□n
wKb¹%□Û□xWÁ{x_□Þ□e±Í+8□□Î□9gX□;Á□J$□□¤I»@Ò®□añÙji□;óú²I□hÆÎëæ¢î□â¼.□ÜU£Â□ÌØN1]tô□ÉÓ÷□½\æNú□
□ÚàñÊ`JÅ·øN¤ö=Ãày¼]û·ëCºA□GÉv-µ6JØµ□æ¦□ÚD²ñzL□7□{ª®□Ù¢□!Í□□r□í□+r7û°âòRóE<´Zx·□»×u□[43□ÁÌ□"m
vå«□p□¼Ç□ÂúR□|□»□mÆE_â□á:MÚ×çÅñÄÖaµ□`□ZP□7ÛØ,m±I¾hÐDë□}□ºúl£/ÿ^4ùª□□[ETÔ¼,1G[□J□D□jj\q.¦d"¹¥
□D□□Ô□"-2f□¤'□□□□¬aÇì□□rÒ□□r¡Ñ□Â5D²ë□kèXCç□´□%.Õº3.EÕ□v□□ãáènÒîÊ□¢ÝÙ□ö□Eâ□F□h.4Lï{oÄ¥¿□{,
&Ë□□Ñ-£×û□Éè□é÷□0æeÍ0N□è□SgýgÊ*BÏé'èÍG÷a□ñ□□§□¡3R□tèG□□é□N □jè□□¨áàcDâá_□Iì§çÿ<Á¡\´?àj□□× ý
  □□   A hQþ}¢   Ò   #   org/gradle/wrapper/Download$1.class}□M
Â0□□ßh5µ*v/□□×□¾□?àB\x□Ø□Ø□□□T{7□□ÀC□)âÖ□æ□¾÷^ïÇ□À□C□Æ0 $g{s□Ü□Z□Æ[Û□mE¾,Å]□&;□ië□£□²¾Ú□
  □□   A Çåî©□□  L□  4   org/gradle/wrapper/Download$ProxyAuthenticator.class□S]O□A□=³-l]¶XWmEä
ó□nb!□Å□□Ìö□□eÜv□□ocÅÆ□□ñga□êç      ?¿Å□,Ë□`,X     cñº¾»#Ô□¾□QÄ«È*□¶,
Í¹□LêFØ¢Ü□□{Ýµ¶n□X□U®¥bp_Æ±På□·Z□0Å□Tõ ®x-□AGñfS¨à□ìÄ□äµ¥³□%□©ºÐ□DïHU□È□2fXôó□÷ü□□b¡□á □□»
  □□   A 5¶<□À□  □□  !   org/gradle/wrapper/Download.class□Y  |\ÇYÿ℡Òî¾§ÕÓáõ¹¾²NìXÖi;□□Ë□□K¾dË
```

B£□□*^□ÙvÒ#8G«Ni$èy6□¢□□+□ùÿÑ1E□83$r£]□ØoPK□□
□□ A □%àÌ»□ ª□ 7 org/gradle/cli/CommandLineParser$OptionComparator.class□TmOÓP□~îÖÑQ:□¯
¾□Ý
□2@□□□.Ñ,YÀd□Ä□e4³¤kIÛ□□□?□/~□#□á³ÿÆ¨_□ç¶N6).]ÏË}Îsî=çô~ýõù□□E□%d0#¡□¿ú0+bN□ UÂ□J"□E,IHs;□'
ùä□#fÂdYþä"□5□□¨>¬{}FCÀåNS□Æ#ZOâq□Ï£□H□·INQÍ2(RÔ8éôË¦ÉSÄ<Ùi□Ë□ÿPK□□
□□ A äfC□§□ £□ 8 org/gradle/cli/CommandLineParser$OptionParserState.class□□ßJ□A□Æ¿³Y³º¦
}□¯□/ú }¨Ò3□□□□4»□ß□³ßù□Î□ýýçá□□#ÔB□°□ÂÇF□Í UBñXie?□
õ%□Áo¦}IXl)-O□□®4ç¢□pf¹□öDÒ□F¹ó8éÛK□□□Î®Jõ□a2iÚVXþTú®µ4ÍDd□dËÇVjâ(6¢□È¨□¨¨□□□B÷]□Q]í□¤AXH5□A
□□ A ¢Æ÷E«□ ¶□ 3 org/gradle/cli/CommandLineParser$OptionString.class□TÿN□A□þözåÊq□Z□DQ□
□□ A gAq²□□ x□ = org/gradle/cli/CommandLineParser$OptionStringComparator.class□TÏO□A□þf
U□□,ç°¥²□□¿^`Èý□□¡¿n7]□¶¤□ë□bÓ
mÇü)Þz÷ì□TJ¯{-¿amÛr´LW»yyÈ□□□□%,kxÌ°Ù□cE□VÕ,7°CûÄê%□□=□²ã¤□Qd□Ò¿ófXýÏ□t¥XG,□□ëÔb□êýOÐ%q,·□□D
.□¥h□"□,#□□% □G□Ç□I□ä
Òä□□ n`□,□òÄ□?C¡<°U<□¢%□¦%□ªlÌý□¨ÙSè£ê□□%□=□î)□jû□Ô□ì□□"¡□m*¢D□Çé³□□À O"Ë5Ló>ä¹□□□À□OE□□
□□ A `M~U□□ ,□ 2 org/gradle/cli/CommandLineParser$ParserState.class□SßoÒP□þN)□□s□ÛÔ9□□
□□ A pÍXÝ□ k□ ? org/gradle/cli/CommandLineParser$UnknownOptionParserState.class½UËR□Q□=w
BY°»$Sqt2CM&Ê/ø/.¤□G□□?À□□_□7nÜè/XvO¦B□P!.ÜtßÛ·Ïé%§o&ï□¿}□`
·5Ä□ÕÐ□±6Z□³É±□à□I□R¦£,□Å5□µìIÏ□â□@û#é□w□□□ú□Û6Ü%K□ËFY`|Ùq□¹¢+□□□Ë[fnÉ)□¤]X6m£□□®□Ï □¤
□□ A dZ□Ô□□ 1) & org/gradle/cli/CommandLineParser.class□Yi`\Å□þJ□□7zz¶eÙ²=`lá□ë□1¶°åS>
zE*ÕJ□_ªÕ.3Usº.³ä5□É□|¬□□¿□©É\5□§¶¬õË|¥÷□¿,T¿□i²XG3□Ë□□åHêx□Ê¬f□¯ÉR¿,Sv-×±UV¨ÑY~9[yç□]êe¥.çÊ*
Æ□□hL□u□ÉË□q³>□7#ñP"t□□M$(°¹ÓùÎ□L8?²=□Ý□±□×□cq3Ö□□&LMÖsÇsO°¶A0Úeú□AQC(□çÖöj]¬ÝZ¢-um 3f
Åâ□¦îmÔkG0ÒJ□KÍ¶hÌÌ□×dcÊ□u;□13cßM□ü□□ÍÜ0Ý[□1«×¬«_Ý,µ±®aÅÖ5uëÖXÛ(□¬ê^□¬îN□ÂÕ1³ÝÜU½&□ V□ù□QQ[Ù¥
µG□□î□-\□¹ºÀþ□□FÚ«m□óWEcíÕí±`kØ¬n □ª□ÛV□"¦ìüE□9:□□Gw6□v□□»□Êú1Ö|ÆyqeÌ4□□1ë×òP<,-lÒ□¾□¡H
kÙÎ□°X5¹P°jØìÃ:ÄÉ□)B□p□(^□□H□i%□□¹□¯n ZÍük□n»□□Ý□□É_□>N¶§¢□
□□□□Ï□□&3A□32g□□R4ìÑd«□Z□8?□lçy□ÒO¤®«□□m'□áÂ□f@çØh¬Õ□%sÆ□UÊuÅÌ6ë□□3□iÊ
Å□¾¹ÛÍ□¢Ä%Áp77õ□□DLý^a÷E□□9Í□V\Ñî□□EW□p% vX8□)â¬ä4iFf(Må¾aú□}î□à.ÁÔl?ºFò´,□N@6ßÍ7üwå□3ô½áÂ§ÉE
□□Áz¢Xç>"£Nd1ìV□f□Y□72³pLbÊ h1s}Á,¼6=£Jq□®ÒÁÅ´Ï*£É9¡¤ôDuXà□U ªëÇ»?□¹²}□eÊÐ□³¡□□§[»□□Ö□eN(åóq□
ç4□6º□ð□<[]¿k□t6SN¶¾oÌ□Æ;:ë□□□´»Ô□ÚC□êÕqÖ»□¦[àc¥¡`ÒÛÑ|Ý½ó□Û(~®zvú□V□□7\e□ □£□□"□zü□,ª>²Ý
9□Q□å}(à□Â□ðôa´`·¾®F□□¿Â□ZoeÀÛ□±»å±□·°¨□ãv£ □ã□□ë□Æ}r□§□¾8i7N"eÀKâ□{1±Ö□ðõâ□□\N□I□AUÀ[°°□□
ÕO¬õ□ü½¨Ü□Â□?i.UÁMjºªÑ[□W□§\Vý æÒ*þRv□×ê□"uG¤□□YÕxm□ì;þâ>,¦.□®®³¾□Ó□ÒK□éÖ□Íl Üwlfe/N¯ØoE□□§'Q
ÌórËög¡ÐTþ¶4□Ò□3□+®"d□□;(âN□□W16□±E□□¶k□Û¬ðæ f5□7V□ªñäÖx□¾E□□□ÊEÞY µ¾□dN□□□>ÌÎÁ□k½□ûßUôa□ :
Ýñ□¦□Q³û□c"□□á□Û%¹Ïp;Â°üª□s□Ë□`□q·□x<¿□ð¹à □6V¦□¹□□è□~,¦ï□Ôz*-wÔ□ü□□¾®F□G|¶□:%f)ÅX□J¨X□□²
Öd^pôK¼ÆPÎ□□ñ7x□Y□□§ñ+òçb□Ü□7X:y¨ñóx□Yçµ²Få□É¯S□ü-1ÍÞáï8gËèOÉø{zSe×□y□µNv½¥a®□`¿aZ□x¾□úà□5Öø
□eÍc%,3w□□Çä]}□s□Úêê□²S□}ÆÞÇ~e?□□2□□<□Ê_□¯på²>lJ¯□0Ý©□j½§ìA^□Ãió□
u□oQ£ÜEû□¿Ãø¹p Ãª□8Ð~?Þâ□oSÑw□/¿å-ýOÄ³w□g¿Ã□Þ¶Kñ/´ÿý□□U°□üGÒy□^¤}×B±□□«□,ÌZJ¾÷,;□□ó□ô;É¢□□½
cÑaÝ*^®□°òHÝ*□R·Ê!çV¹□g©ò-y□hÇà·.□#¨f□N>□©_À{□EV43$§□øñïø□çf®f¯□N¹8÷Ñ¬·Á□´;ÙËlçÎdþOü□□sNöÃÂ□ù
;{±□□□g□<¹Oajaø v<□sð[©□Gò,%.ÇX"□üÐ"ù□Q□kG□Oa©XÇg\□□p□³□±□Ï>Î>Á¾□â.a¿□□»Ø÷xtòvâ2Î□g□9û+Ø□□
□□ A □□å>É□ □ & org/gradle/cli/ParsedCommandLine.class□Wiw□g□~FÛÈòÄ□ê%v□TMc[□å□□ ñR·
□Í¾□1s□#+3¶□É¥©□¾ /ë□E;□□□ÒóÃ
êf2é□n□-CÁÉê§#Î2«çÒ□□Àð¤i¥□Ò□□Ê□□□ÙÌÀ□ºU0RãæÒ□□KMfrÆ)yâð(□□ò□Q0r¶£*T□>cØ4y¤Zãq □²□%sÙH□□□□
´¾àÌ FA|»□ÒWÓÝ³wó□/ë□½□)xØÖàÓV5KÔ¶x± :Ô□|)ÜÃ□ö#÷wOP¿cÆÖ□□c□I(□ßVñ□□Ã^yØ□¢ÈUØ6K¥Þ□ïõ¤ÿbÑ´□
´RÎ□|{□çY□#Û¶õô·nQ/□ª¬5^kÑKo□□u·□O;=F□,m,·RÚÙ□¯h¤q3□5□$ßbwcÖLO□ËF¶Ü□mÞÆ
□□□¶□¾ßûLïÐÏn"ª□Ýöûª9mØG7M□h©z*gJw□r□ vè©□D□Õ³Ec□úY®Áeçy□íK□?ëa{ÿòÿ?ÂÕ□qZ:¯«f□y Ó×È□Y´□□c□Ñ
Ý é,^Xd□kèÇþ□□□ù□Þ□□□□á□□□6□□Y□¾"□Öð□|PÃ[x[Ãwñ=ßÇ□□Äîç1sµ=>4ü□?Rñc¶ÂfÖUüDÃOñ3□□Ýñ8
|É□□_à□□□=□¦0aç4ü
WU\Ó0□³□~□ßp´WL□öHZÏ□¾:zyÁpyØéø□ëé*ôÄr¦□K□ç□}j¿È□cü□ë,ªá□□Ñð□®Ö□[Ã□»£v.□□2}Ri□□Æ□□c9SV´UõP
Óù´ØQ¥¨Z³□y□□Û□ãµàµ□O□BçMkI'Æ GÃ>ýÞ □ï□=□7r©□□W=Â¶~©□□Ì□gDô□1□□[¾f(□O□:º½6y□ÙP=ð,□³J8□J□e³
Zç3¾□]□^á□□□¢kéô>□□l□Jþ;ï k>Ú□□eÞ'*ù6zþ□□Eä□□Ge¤)ö¤Aù<□],;[í¢ÃS□;\ô¬Oº□□r□=l8>Y □Ãß□ä□□□^U
□dÈ¹ÿ□PK□□
□□ A yß□tÚ□ E□ , org/gradle/cli/ParsedCommandLineOption.class□S]OÔ@□=Ý¯îG□eùP□a□□eQª¨ø
□,□¦å□□É$ô4□,□Á-L%q[¾ï¨,+ßÓr¹§â¾□□
□[ÌlpWA¶´É¶□Þð□©□□ëÍ(H□□a1¯áp□#-□g□³É,C/{□°□□9*IÌ
Kxs
¢□ñU□±E»JÅ□%añ□□Ú:w^³u□"¹□]aæ*s□<□Á□·!□I¡d;□n8¬jr½b
ý□s\^]´k5fU%ÔJÝ□¶Eí□□÷V¥ □Ý□ñ³□□´□=Vy÷□ÕÃ□©£
j□;*i□=z&x□Ô«V□□£Qã□§ §p6K:□dÇ)¹□ñKn07T@□¾U□.Û§Â□ Évà□é□□IC□z□t□°]p□¶-)kèDNÃ(Æ□□t]6Mn0□,ñ
÷±(# ÍWmîæ-ÛËo°-□gÖvÞ¿□□²Ñ°□□□f0« ?L'æùZÃôDÝäA¦K©□@ÓÏ¶*Õð□ó□□$¡Ñÿ□ï)□□õM^!□ÛO□□î□+v□ï□i¥áÊQ
×$ÈÄ□Ú"xÓjÊ□2D¬Ï§ìç□xr7□k>º¾!âó□V.p£]Á.ÔØG²àÓ±ô□ãD□Ç\□Ô8"Ùy Rð□□ÿ□PK□□
□□ A \vÆB|□ □□ : org/gradle/cli/ProjectPropertiesCommandLineConverter.class□□KOÂ@□□Ï□□
}FHWkw□LSº□°Ü□!¿□□]®nY×7ÎZK:Ì¿cJDóØÌè□□□□ZRy¶§□ësÛñ□ÝVò□;ÚH□+-ø)□□□n´kS□#cruLXõøgh|Ó×B□□□j
ßµ¦XUÞð□V½□□£Üi0□1□ÈkÂÁ□□□p8□□w□Z±ß8T0gî?□□Pôa¦Î□□m□□Îí=□□ÌC
□S³□s□□¦□§□£□□| Ë1\□áôZêq□-}CÓ èJ□□□EË□èj□□□E+ ¨ùw'©õ□PK□□



Let’s revisit the two tests we just wrote, to reflect on the efforts and the benefits.

The nice part of the tests is that by mocking the functions `withContext()` and `async()`, we were able to perform interaction tests and didn't have to worry about actually running asynchronous operations and waiting for results in the tests. This makes the tests fast and deterministic. The biggest downside to this approach is figuring out which class to mock in order to reach in and replace the functions like `withContext()`. To say the least, this isn't easy, and it can be frustrating to find the correct class to mock. But the fast feedback we get from the tests is worth the effort.

In the next lesson, we'll try to use a service with our application.