KOTLIN UNIT TESTING

TODAY'S AGENDA

- Mockk Unit testing Library - Kotlin
- Reading and writing from files

mockK – testing library

Mocking is not a rocket science

mockK.io

- In kotlin, all classes and methods are final by default. But it causes some problems while writing unit tests.
- Most JVM mock libraries have problems with mocking or stubbing final classes. We can add the "open" keyword to classes and methods that we want to mock. But changing classes only for mocking some code doesn't feel like the best approach. By doing this, we are reducing the performance and which is not suitable for High-quality codebase.

mockK.io

- In Android, there are many frameworks for mocking in unit testing, such as PowerMock, Mockito, etc. Mockito is a popular framework used by Java developers. With StackOverflow, we can find many solutions for mockito. But it does have some fundamental issues when used with Kotlin. MockK, being specially designed for Kotlin, is a more reliable and pleasant experience.
- MockK is a better option for other mocking frameworks for Kotlin, the official development language for Android. Its support for Kotlin features comprehensively.

Why is it better than a well known Mockito library for Kotlin?

- Mockk supports some important language features within Kotlin.
  - Final by default(Classes and methods in kotlin) :
    Concerning Java, Kotlin classes (and methods) are final by default. That means Mockito requires some extra things to make it to work, whereas Mockk can do this efficiently without any extra things.
  - Object mocking :
    Kotlin objects mean Java statics. Mockito alone doesn’t support mocking of statics. There are the same other frameworks required with Mockito to write tests, but again Mockk provides this without any extra things.
    - mockObject(MyObject)
    - every { MyObject.someMethod() } returns "Something"

- 3. Extension functions :
  - Since extension functions map to Java statics, again, Mockito doesn’t support mocking them. With Mockk, you can mock them without any extra configuration.
- 4. Chained mocking :
  - With Mockk you can chain your mocking, with this we can mock statements quite easily like you can see in the example below. We are using every{} block to mock methods.
    - val mockedClass = mockk()
    - every { mockedClass.someMethod().someOtherMethod() } returns "Something"

- 5. Mocking static methods is not easy in mockito but using mockK java static methods can easily be mocked using mockStatic.
  - mockkStatic(TextUtils::class)
  - @Test fun validateString() { every { TextUtils.isEmpty(param } returns true }This is how you can mock static method isEmpty(param) of TextUtils class easily.
- So, with those impressive features in mockk makes mocking in Kotlin great.
-

features

- **Mocking**
  - **Mock a class**
    - Firstly, let’s see how to specify what exactly should be returned by the function of the mocked class:
    - With this code, the “**Mocked value**” String will be returned for each invocation of the **publicFunction**.
  - Mock a property
    - This time, our result will be 7, even though the original function should return 50.

- **Spy The Object**
  - Sometimes, we would like to have the possibility to check the real behavior of the class, as well as a mocked one. For that case, we can use a spy: with the fn `Spy a class`
    - val exampleClass = spyk()
    - assertEquals("Returned value", exampleClass.publicFunction())
- As we can see here, the function returns the value that has been implemented in our code.

Mock Private Function Behavior

- In our class implementation, **publicFunction** returns the value from **privateFunction**. Let’s check, how we can specify the behavior of the private function:
  - fun `Mock a private function` () {
  - val exampleClass = spyk(recordPrivateCalls = true)
  - every { exampleClass["privateFunction"]() }
  - returns "Mocked value"
  - assertEquals("Mocked value", exampleClass.publicFunction())
  - }
-

Mock an Object

- the Kotlin object is a special singleton class. To mock it, we will use **mockkObject**:
  - fun `Mock an object` () {
  - mockkObject(ExampleObject)
  - every { ExampleObject.concat(any(), any()) } returns "Mocked value"
  - val result = ExampleObject.concat("", "")
  - assertEquals("Mocked value", result)
  - }

Verification

- If we would like to check how many times the function has been invoked, we can use verify:
- @Test
  - fun `Verify calls`() {
  - val exampleClassMock = mockk()
  - every { exampleClassMock.multiplyByTen(any()) } returns 5

  - exampleClassMock.multiplyByTen(10)
  - exampleClassMock.multiplyByTen(20)

  - verify(exactly = 2) { exampleClassMock.multiplyByTen(any()) }
  - confirmVerified(exampleClassMock)
  - }
- In the above example, we've used the ***exactly*** parameter to verify that the behavior happened exactly 2 times. If we would specify another number, the test would fail and inform us, that the verification failed.

Let’s code something!

Thank you