COMO ESCREVER BONS TESTES UNITÁRIOS

THE DEVELOPER'S CONFERENCE

@orogersilva

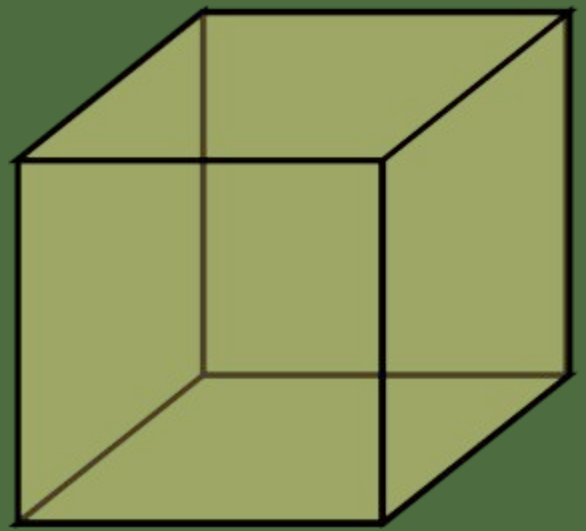
DESENVOLVEDOR DE SOFTWARE - MOBILE

@orogersilva

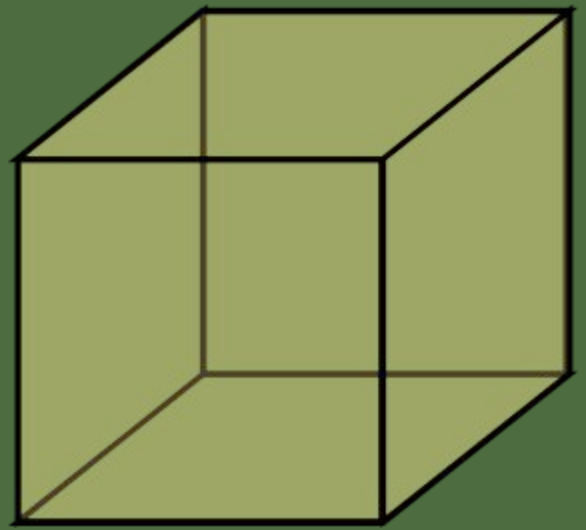
@orogersilva

@orogersilva

O QUE SÃO TESTES UNITÁRIOS?

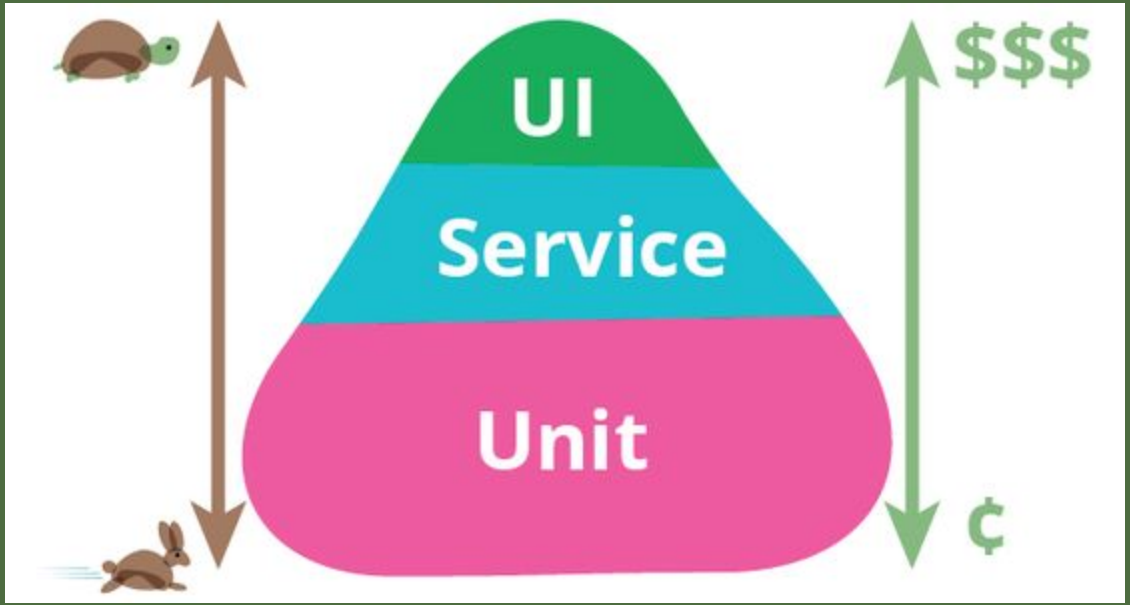


CLASSE



MÉTODO

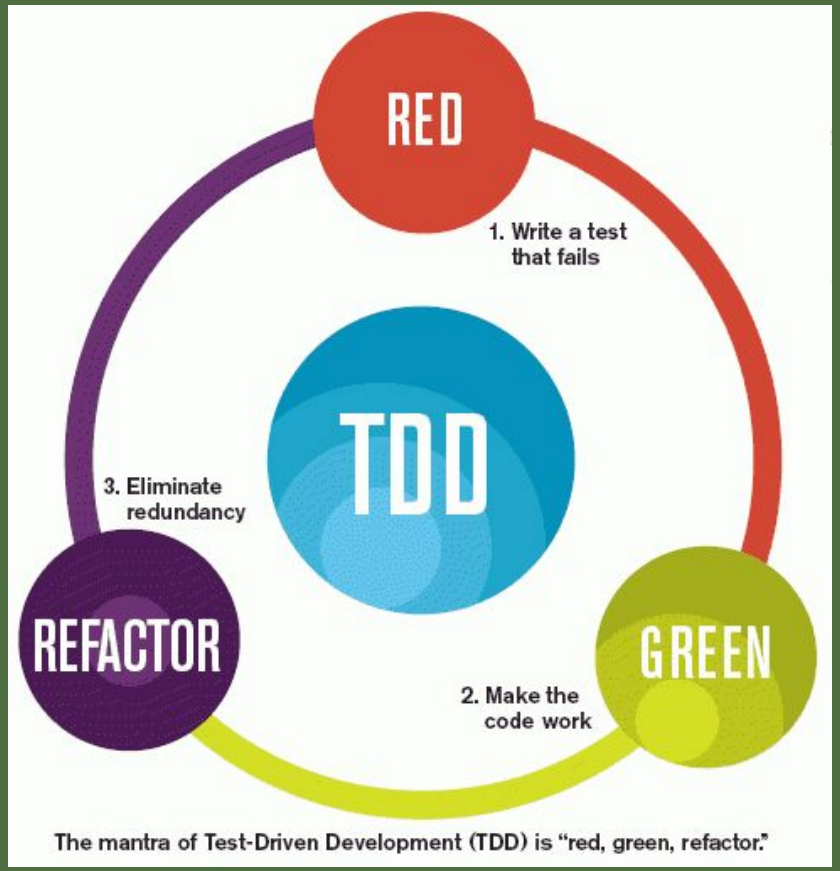
ou



Fonte: https://martinfowler.com/bliki/images/testPyramid/test-pyramid.png

Fonte: https://twitter.com/cermak_cz/status/743719704406036480/photo/1

@orogersilva



Fonte: https://twitter.com/cermak_cz/status/743719704406036480/photo/1

@orogersilva

“COMO DEVO TESTAR?”

- SETUP
- EXERCISE
- VERIFY
- TEARDOWN

CLASSE DE TESTE

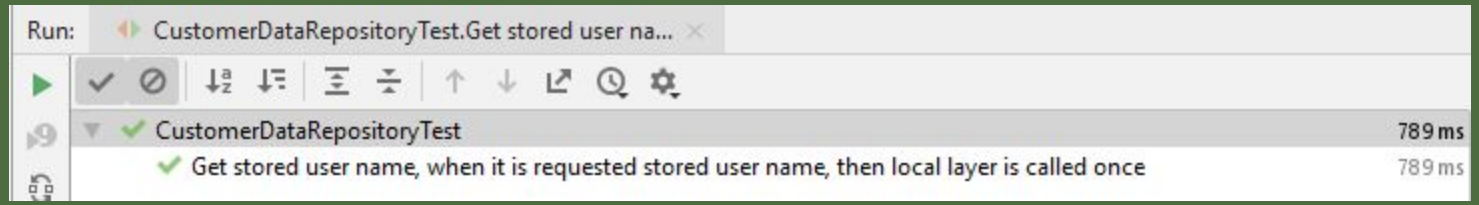
MÉTODO DE TESTE

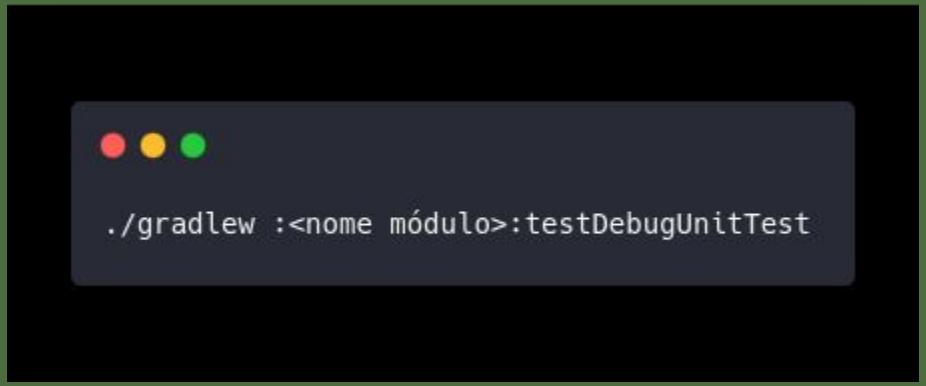
```
class CustomerDataRepositoryTest {

    @Test fun 'Get stored user name, when it is requested stored user name, then local layer is called
once'() {

        // ARRANGE

        val EXPECTED_USER_NAME = "Mr. John"

        whenever(customerLocalDataSourceMock.getStoredUserName()).thenReturn(EXPECTED_USER_NAME)

        // ACT

        val userName = customerRepository.getStoredUserName()

        // ASSERT

        assertEquals(EXPECTED_USER_NAME, userName)
}
```

```
class ClasseDeTeste {

    @Test fun '<Nome do método testado>, when <Condição>, then <Resultado esperado>'() {

        (...)
    }
}
```

```
class ClasseDeTeste {

    @Test fun '<Nome do método testado>, when <Condição>, then <Resultado esperado>'() {

        // ARRANGE

        // ACT

        // ASSERT
    }
}
```

```
@Test
fun `Get stored user name, when it is requested stored user name, then local layer is called once`() {

    // ARRANGE

    val EXPECTED_USER_NAME = "Mr. John"

    whenever(customerLocalDataSourceMock.getStoredUserName()).thenReturn(EXPECTED_USER_NAME)

    // ACT

    val userName = customerRepository.getStoredUserName()

    // ASSERT

    assertEquals(EXPECTED_USER_NAME, userName)
}
```


Run:
CustomerDataRepositoryTest.Get stored user na...
CustomerDataRepositoryTest
789 ms
Get stored user name, when it is requested stored user name, then local layer is called once
789 ms



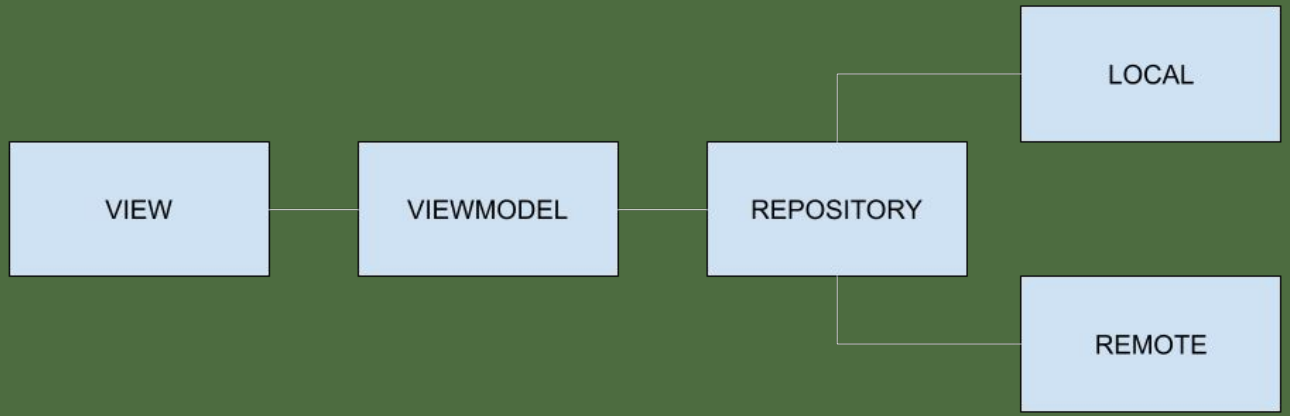
```
CustomerDataRepositoryTest > Get stored user name, when it is requested stored user name, then local layer is called once PASSED
```

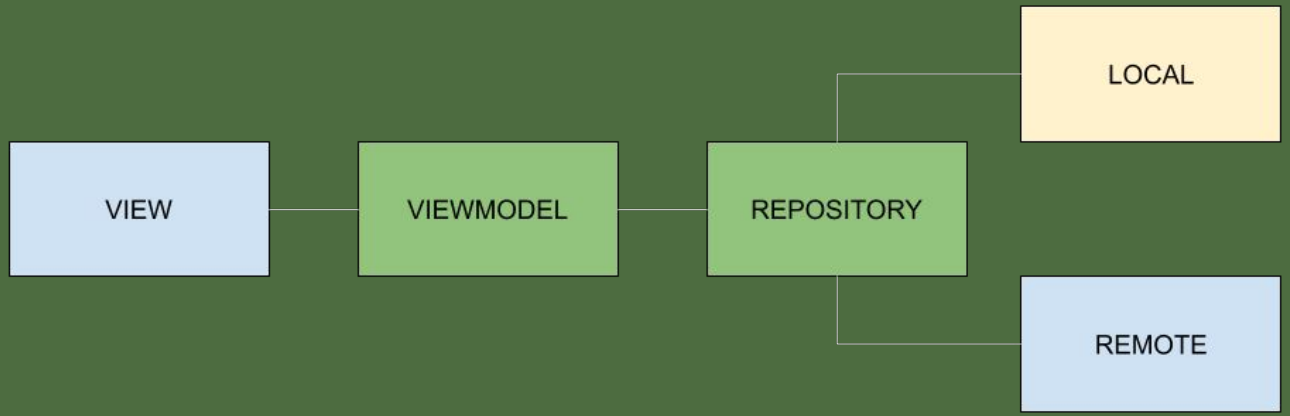

Dependency

INJEÇÃO DE DEPENDÊNCIA

```
class CustomerDataRepository(private val customerLocalDataSource: CustomerDataContact.Local,
                             private val customerRemoteDataSource: CustomerDataContract.Remote) :
CustomerRepository {

    override fun getStoredUserName(): String = customerLocalDataSource.getStoredUserName()
}
```

UMA BOA ARQUITETURA





MOCKS

Mocks Aren't Stubs

The term 'Mock Objects' has become a popular one to describe special case objects that mimic real objects for testing. Most language environments now have frameworks that make it easy to create mock objects. What's often not realized, however, is that mock objects are but one form of special case test object, one that enables a different style of testing. In this article I'll explain how mock objects work, how they encourage testing based on behavior verification, and how the community around them uses them to develop a different style of testing.

02 January 2007

Martin Fowler

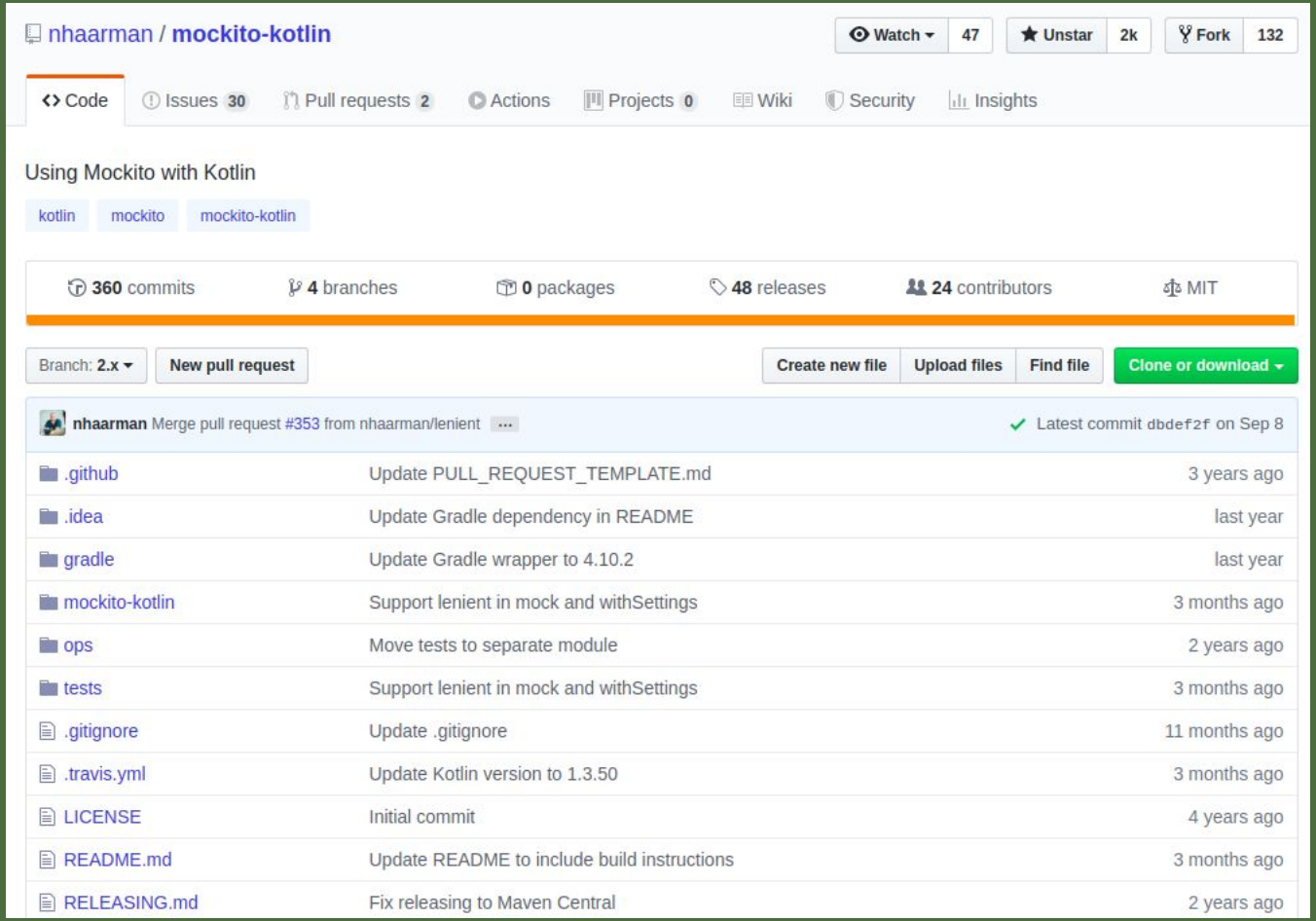
POPULAR

TESTING

TRANSLATIONS: French · Italian · Spanish · Portuguese · Korean

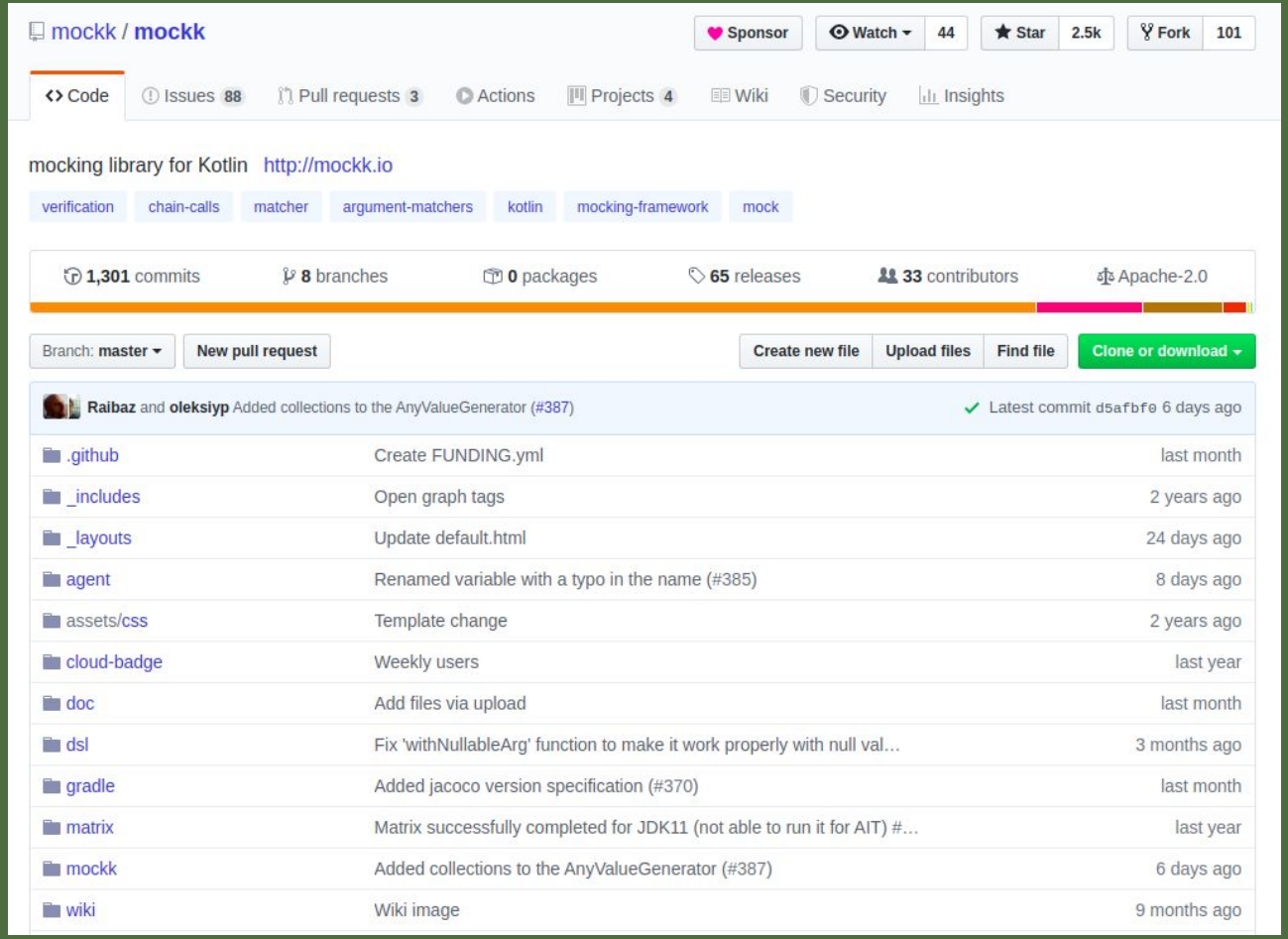
CONTENTS

Fonte: https://martinfowler.com/articles/mocksArentStubs.html



Fonte: https://github.com/nhaarman/mockito-kotlin

@orogersilva



Fonte: https://github.com/mockk/mockk

TESTANDO REPOSITORY

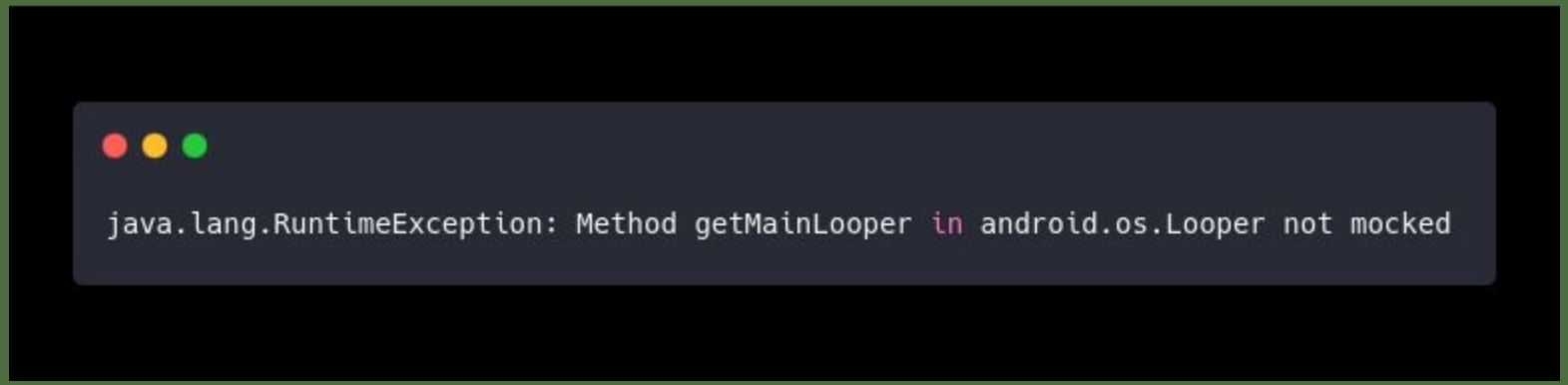
```
class CustomerDataRepository(private val customerLocalDataSource: CustomerDataContact.Local,
                             private val customerRemoteDataSource: CustomerDataContract.Remote) :
CustomerRepository {

    override fun getStoredUserName(): String = customerLocalDataSource.getStoredUserName()
}
```

```
class CustomerDataRepositoryTest {

    private lateinit var customerLocalDataSourceMock: CustomerDataContract.Local
    private lateinit var customerRemoteDataSourceMock: CustomerDataContract.Remote

    private lateinit var customerRepository: CustomerDataRepository

    @Before fun setUp() {

        customerLocalDataSource = mock()
        customerRemoteDataSource = mock()

        customerRepository = CustomerDataRepository(
            customerLocalDataSourceMock,
            customerRemoteDataSourceMock
        )
    }

    @Test fun 'Get stored user name, when it is requested stored user name, then local layer is called
once'() {

      // ARRANGE

      val EXPECTED_USER_NAME = "Mr. John"

      whenever(customerLocalDataSourceMock.getStoredUserName()).thenReturn(EXPECTED_USER_NAME)

      // ACT

      val userName = customerRepository.getStoredUserName()

      // ASSERT

      assertEquals(EXPECTED_USER_NAME, userName)
}
```

TESTANDO UTILS

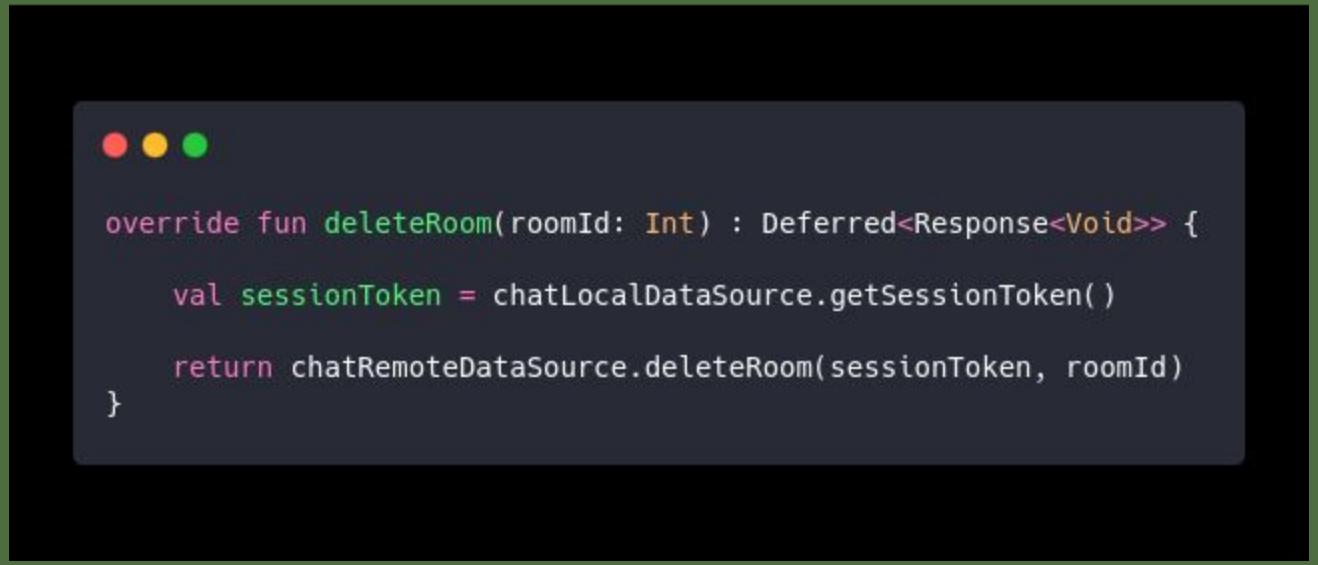
```kotlin
class DateUtilsTest {

    @Test fun 'Get month in full, when it is passed a valid month index, then returns month name'() {

        // ARRANGE

        val EXPECTED_MONTH_NAME = "april"
        val MONTH_INDEX = 3

        // ACT

        val MONTH_NAME = DateUtil.getMonthInFull(MONTH_INDEX)

        // ASSERT

        assertEquals(EXPECTED_MONTH_NAME, monthName)
    }
}
```

TESTANDO VIEWMODEL



TESTANDO COROUTINE

@orogersilva



```
override fun deleteRoom(roomId: Int) : Deferred<Response<Void>> {

    val sessionToken = chatLocalDataSource.getSessionToken()

    return chatRemoteDataSource.deleteRoom(sessionToken, roomId)
}
```

```
@Test fun 'Close room, when it is requested, then remote layer is called once'() {

    // ARRANGE

    val SESSION_TOKEN = "7JZp0YplwEHO3smSHhx+Epa1NTmNobzuYXPrnZEjsR8U="

    val ROOM_ID = 231

    val expectedResponse = Response.success<Void>(null)

    val expectedDeferredResponse = CompletableDeferred<Response<Void>>(expectedResponse)

    whenever(chatLocalDataSourceMock.getSessionToken()).thenReturn(SESSION_TOKEN)

    whenever(chatRemoteDataSourceMock.deleteRoom(SESSION_TOKEN, ROOM_ID))
        .thenReturn(expectedDeferredResponse)

    // ACT

    runBlocking { chatRepository.deleteRoom(SESSION_TOKEN, ROOM_ID).await() }

    // ASSERT

    verify(chatRemoteDataSourceMock, times(1)).deleteRoom(SESSION_TOKEN, ROOM_ID)
}
```

TESTES UNITÁRIOS DEVEM VALIDAR COMPORTAMENTO E NÃO FLUXO DE EXECUÇÃO

COBERTURA DE CÓDIGO


JACOCO
Java Code Coverage

APLICAÇÃO DE PLUGIN + ARQUIVO DE CONFIGURAÇÃO


apply plugin: 'jacoco'

https://github.com/orogersilva/spotmusic-alarm-android/blob/master/gradle/jacoco.gradle

```
def fileFilter = [
        '**/R.class',
        '**/R$*.class',
        '**/BuildConfig.*',
        '**/Manifest*.*',
        '**/*Test*.*',
        'android/**/*.*',

        // Dagger 2
        '**/*Dagger*.*',
        '**/*Dagger*Component*.*',
        '**/*Module.*',
        '**/*Module$*.*',
        '**/*MembersInjector*.*',
        '**/*_Factory*.*',
        '**/*Provide*Factory*.*',
        '**/*$ViewInjector*.*',
        '**/*$*$*.*',

        // Classes I intentionally don't want to test...
        '**/*Activity.*',
        '**/*Binding.*',
        '**/BR.*',
        '**/adapter/**/*.*',
        '**/adapters/**/*.*',
        '**/dao/**/*.*',
        '**/databinding/**/*.*',
        '**/dto/**/*.*',
        '**/enums/**/*.*',
        '**/entity/**/*.*',
        '**/factory/**/*.*',
        '**/pagination/**/*.*',
        '**/relation/**/*.*',
        '**/shared/**/*.*',
        '**/typeconverter/**/*.*',
        '**/view/**/*.*',
        '**/*Activity.*',
        '**/*Database*.*',
        '**/*DataBinderMapperImpl.*',
        '**/*_Factory*.*',
        '**/*Fragment.*'
]
```

```
task jacocoFullTestReport(type: JacocoReport) {

    group = 'Reporting'
    description = "Generate Jacoco coverage reports for the debug build."

    //Make sure that tests from all modules are run before coverage report
    dependsOn ":dashboard-data:testDebugUnitTest"
    dependsOn ":feature-dashboard:testDebugUnitTest"
    dependsOn ":dashboard-data:createDebugCoverageReport"
    dependsOn ":feature-dashboard:createDebugCoverageReport"

    (...)
}
```

```
task jacocoFullTestReport(type: JacocoReport) {

    (...)

    classDirectories = files([
            fileTree(
                    dir: "$project.rootDir/dashboard-data/build/intermediates/classes/debug",
                    excludes: fileFilter
            ) + fileTree(
                    dir: "$project.rootDir/dashboard-data/build/tmp/kotlin-classes/debug",
                    excludes: fileFilter
            ),
            fileTree(
                    dir: "$project.rootDir/feature-dashboard/build/intermediates/classes/debug",
                    excludes: fileFilter
            ) + fileTree(
                    dir: "$project.rootDir/feature-dashboard/build/tmp/kotlin-classes/debug",
                    excludes: fileFilter
            )
    ])

    (...)
}
```

```
task jacocoFullTestReport(type: JacocoReport) {

    (...)

    executionData = fileTree(dir: project.rootDir, includes: [
            'dashboard-data/build/jacoco/testDebugUnitTest.exec',
            'dashboard-data/build/outputs/code-coverage/connected/*coverage.ec',
            'feature-dashboard/build/jacoco/testDebugUnitTest.exec',
            'feature-dashboard/build/outputs/code-coverage/connected/*coverage.ec'
    ])

    (...)
}
```


./gradlew jacocoFullTestReport

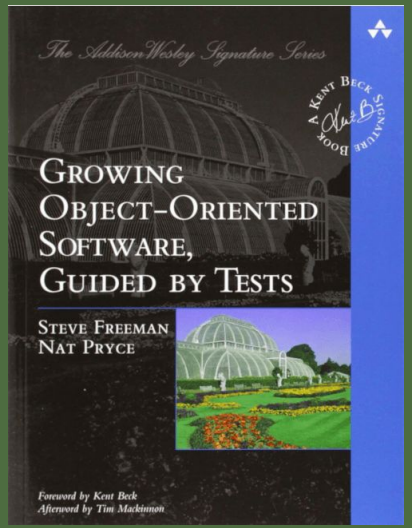
É NECESSÁRIO 100% DE COBERTURA DE CÓDIGO?

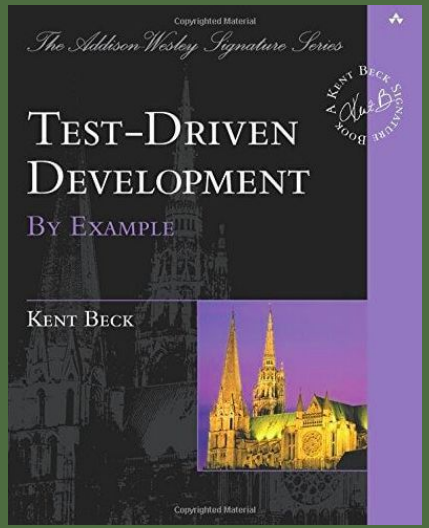
TESTES UNITÁRIOS SÃO SUFICIENTES?

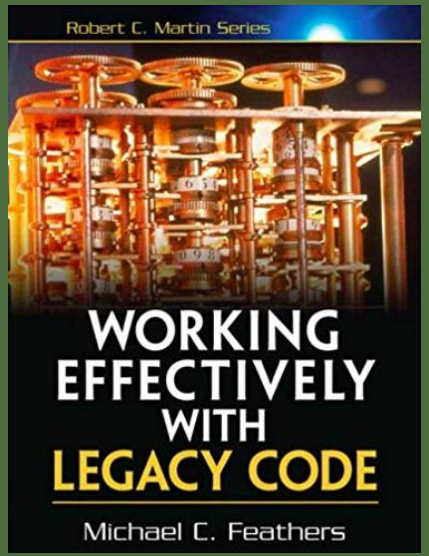
TESTES UNITÁRIOS DEVEM SER MUTÁVEIS

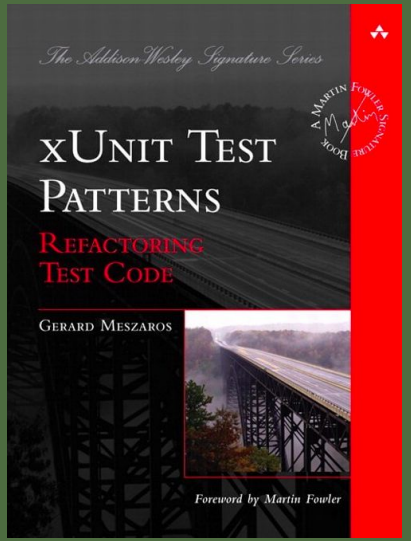
TESTES UNITÁRIOS DEVEM SER CONSTANTEMENTE EXECUTADOS

REFERÊNCIA SOBRE TESTES UNITÁRIOS









@orogersilva

@orogersilva

OBRIGADO!

@orogersilva

COMO ESCREVER BONS TESTES UNITÁRIOS

THE DEVELOPER'S CONFERENCE

@orogersilva