Best Practices for Unit Testing in Kotlin

@philipp_hauer
Spreadshirt

KotlinConf, Amsterdam

Oct 05, 2018

Question

My First Test in Kotlin...

```
class UserController
    companion object {
        @JvmStatic lateinit var controller: UserController
        @JvmStatic lateinit var repo: UserRepository
        @BeforeClass @JvmStatic  initialize() {
            repo = mock()
            controller = UserController(repo)
        }
    }
    @Test
    fun findUser_UserFoundAndHasCorrectValues() {
        `when`(repo.findUser(1)).thenReturn(User(1, "Peter"))
        val user = controller.getUser(1)
        assertEquals(user?.name, "Peter")
    }
}
```

Boilerplate!

```
open class UserRepository
```

open required

mutable! reassignable!

buh, static!

Hard to Read!

Better Mock API?

Poor Error Message

We can do better!

Readable

Idiomatic

Clean

Concise

Reasonable Failure Messages

How?

Test Lifecycle

Naming, Grouping

Test Libraries

Mock Handling

Spring Integration

The Power of Data Classes

Recap:
Idiomatic Kotlin Code

Idiomatic Kotlin Code

Immutability

`val`

~~`var`~~

Non-Nullability

`String`

~~`String?`~~

No Static Access

No direct language feature

Test Class Lifecycle

JUnit4: Always New Test Class Instances

```
class RepositoryTest {

    val mongo = startMongoContainer()

    @Test
    fun test1() { ... }

    @Test
    fun test2() { ... }

}
```

Executed for each test

instance1: RepositoryTest

instance2: RepositoryTest

Where to put the initial setup code?

JUnit4: Static for the Initial Setup Code

Boilerplate!

null

workaround

mutable

static

```
class RepositoryTest {
    companion object {
        @JvmStatic private lateinit var mongo: GenericContainer
        @JvmStatic private lateinit var repo: Repository
        @BeforeClass @JvmStatic
        fun initialize() {
            mongo = startMongoContainer()
            repo = Repository(mongo.host, mongo.port)
        }
    }
}
```

JUnit5 to the Rescue!

JUnit5: Reuse the Test Class Instance

```
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
class RepositoryTest {
    private val mongo = startMongoContainer().apply {
        configure()
    }
    private val repo = Repository(mongo.host, mongo.port)

    @Test
    fun test1() { }

}
```

Concise

Idiomatic

JUnit5: Reuse the Test Class Instance

```
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
class RepositoryTest {
    private val mongo: GenericContainer
    private val repo: Repository
    init {
        mongo = startMongoContainer().apply {
            configure()
        }
        repo = Repository(mongo.host, mongo.port)
    }
}
```

JUnit5: Change the Lifecycle Default

src/test/resources/junit-platform.properties:

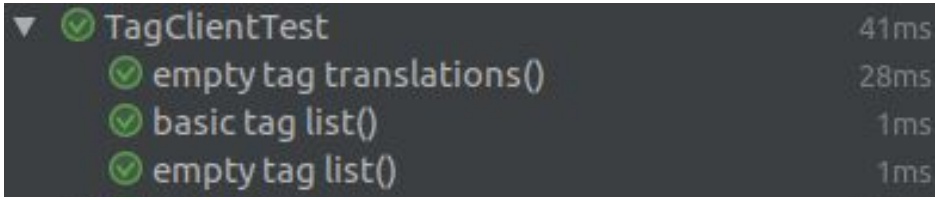
```
junit.jupiter.testinstance.lifecycle.default = per_class
```

~~@TestInstance(TestInstance.Lifecycle.PER_CLASS)~~

Test Names and Grouping

Backticks

```
class TagClientTest {
    @Test
    fun `basic tag list`() {}
    @Test
    fun `empty tag list`() {}
}
```



Test Results	3 s 353 ms
IdeaDAOTest	3 s 353 ms
check if setting Google Vision state works()	1 s 630 ms
check that idea will be updated properly()	83 ms
check if setting recalculated prediction data throws IdeaDAOException()	91 ms
check if setting Google Vision data throws IdeaDAOException()	52 ms
check if getting ideas which are ready for prediction throws IdeaDAOException()	125 ms
check if only ideas without failed flag are fetched()	71 ms
check if setting Google Vision data works()	32 ms
check that prediction will be updated properly()	35 ms
check that predictions will be removed properly()	29 ms
check that predictions without changes will be ignored()	12 ms
check if getting ideas which are ready for being sent throws IdeaDAOException 2()	62 ms
check if getting ideas which are ready for being sent throws IdeaDAOException 3()	55 ms
check if setting Google Vision state throws IdeaDAOException()	57 ms
check that getIdeasWithoutGoogleData throws IdeaDAOException()	79 ms
check, if bulkUpdateHumanDecisions returns 0 when using an empty list()	3 ms
check that ideas will be added properly()	20 ms
check if only ideas with successful google vision state and without prediction data are fetched 2()	35 ms
check if getting ideas for prediction recalculation throws IdeaDAOException()	58 ms
check that removing prediction data throws IdeaDAOException()	41 ms
check if setting sent legal rule set data throws IdeaDAOException()	83 ms
check if only ideas with successful google vision state and without prediction data are fetched()	37 ms
check if human decision are set correctly in bulk()	41 ms
check that addOrUpdateIdeaWithMasterData throws IdeaDAOException()	47 ms
check if setting prediction data throws IdeaDAOException()	55 ms
check if setting sent prediction data throws IdeaDAOException()	48 ms
check if IdeaDAOException gets thrown 2()	48 ms
check if IdeaDAOException gets thrown()	54 ms
check if setting prediction data works properly()	48 ms
check if setting sent prediction data works properly()	22 ms
check if fetching ideas without GoogleVision data works()	34 ms
check if only ideas without human decision are fetched and without sent rule set()	48 ms
check if setting sent legal rule set data works properly()	45 ms
only sent idea without human decision so we don't override poolboy rule set()	28 ms
check if only ideas with successful google vision state and with prediction data are fetched()	32 ms
check if only ideas with successful google vision state and without rule set data are fetched()	28 ms
check that idea with dirty sent prediction will be fetched()	47 ms
check that last id will be taken into account()	38 ms

Which test belongs to which method?

@Nested Inner Classes

```
class DesignControllerTest {
    @Nested
    inner class GetDesigns {
        @Test
        fun `all fields are included`() {}
        @Test
        fun `limit parameter`() {}
    }
    @Nested
    inner class DeleteDesign {
        @Test
        fun `design is removed in db`() {}
    }
}
```

getDesign()

deleteDesign()

Test Results	3 s 764 ms
IdeaDAOTest	3 s 764 ms
AddOrUpdateIdeasWithMasterData	1 s 993 ms
check that idea will be updated properly()	1 s 894 ms
check that ideas will be added properly()	38 ms
check that addOrUpdateIdeaWithMasterData throws IdeaDAOException()	61 ms
GetIdeasWithoutGoogleData	142 ms
check that getIdeasWithoutGoogleData throws IdeaDAOException()	74 ms
check if fetching ideas without GoogleVision data works()	68 ms
SetGoogleVisionState	62 ms
check if setting Google Vision state works()	24 ms
check if setting Google Vision state throws IdeaDAOException()	38 ms
SetGoogleData	89 ms
check if setting Google Vision data throws IdeaDAOException()	58 ms
check if setting Google Vision data works()	31 ms
GetIdeasReadyForPrediction	202 ms
check if getting ideas which are ready for prediction throws IdeaDAOException()	132 ms
check if only ideas with successful google vision state and without prediction data are fetched()	70 ms
GetIdeasForPredictionRecalculation	137 ms
check if getting ideas for prediction recalculation throws IdeaDAOException()	78 ms
check if only ideas with successful google vision state and with prediction data are fetched()	29 ms
check that last id will be taken into account()	30 ms
GetIdeasWithoutSentPrediction	180 ms
check if only ideas with successful google vision state and without prediction data are fetched()	54 ms
check if getting ideas which are ready for being sent throws IdeaDAOException()	63 ms
check that idea with dirty sent prediction will be fetched()	63 ms
GetIdeasReadyForSending	189 ms
check if only ideas without failed flag are fetched()	57 ms
check if getting ideas which are ready for being sent throws IdeaDAOException()	46 ms
only sent idea without human decision so we don't override poolboy rule set()	57 ms
check if only ideas with successful google vision state and without rule set data are fetched()	29 ms

Kotlin Test Libraries

Being Spoilt for Choice

	Test Frameworks	Mocking	Assertions	
Kotlin	Spek KotlinTest	Mockito-Kotlin MockK	Strikt HamKrest Kluent	Atrium Expekt AssertK
Java	JUnit5		AssertJ	

My personal choice (for now)

Incomplete list.
Some libraries fit into multiple categories.

Test-Specific Extension Functions

```
assertThat(taxRate1).isCloseTo(0.3f, Offset.offset(0.001f))
assertThat(taxRate2).isCloseTo(0.2f, Offset.offset(0.001f))
assertThat(taxRate3).isCloseTo(0.5f, Offset.offset(0.001f))
```

Duplication

```
fun AbstractFloatAssert<*>.isCloseTo(expected: Float)
    = this.isCloseTo(expected, Offset.offset(0.001f))
```

```
// Usage:
assertThat(taxRate1).isCloseTo(0.3f)
assertThat(taxRate2).isCloseTo(0.2f)
assertThat(taxRate3).isCloseTo(0.5f)
```

Clean

Idiomatic

Mock Handling

Classes Are Final by Default

Solutions

- Interfaces
- `open` explicitly
- Mockito: Enable incubating feature to mock final classes
- **MockK**

MockK


```
mockk(relaxed=true)
```

```
val clientMock: UserClient = mockk()
every { clientMock.getUser(any()) }
   returns User(id = 1, name = "Ben")
```

```
val updater = UserUpdater(clientMock)
updater.updateUser(1)
```

```
verify { clientMock.getUser(1) }
```

MockK


```
verifySequence {
    clientMock.getUser(2)
    repoMock.saveUser(user)
}
```

```
java.lang.AssertionError: Verification failed: calls are
not exactly matching verification sequence

Matchers:
UserClient(#5).getUser(eq(2)))
UserRepo(#4).saveUser(eq(User(id=1, name=Ben, age=29))))

Calls:
1) UserClient(#5).getUser(1)
2) UserRepo(#4).saveUser(User(id=1, name=Ben, age=29))
```

Does Test Speed Matter?

▼ <default package>	2 s 154 ms
▶ AdminViewTest	1 s 60 ms
▶ StatisticsViewTest	225 ms
▶ ExecutionRunnerTest	815 ms
▶ SchedulerTest	54 ms

2 s for 31 Unit Tests?

Don't Recreate Mocks

```
class DesignControllerTest {
    private lateinit var repo: DesignRepository
    private lateinit var client: DesignClient
    private lateinit var controller: DesignController
    @BeforeEach
    fun init() {
        repo = mockk()
        client = mockk()
        controller = DesignController(repo, client)
    }
}
```

Expensive!

Create Mocks Once, Reset Them

```
class DesignControllerTest {
    private val repo: DesignRepository = mockk()
    private val client: DesignClient = mockk()
    private val controller = DesignController(repo, client)
    @BeforeEach
    fun init() {
        clearMocks(repo, client)
    }
}
```

Fast

Create Mocks Once, Reset Them

<default package>	2 s 154 ms
AdminViewTest	1 s 60 ms
StatisticsViewTest	225 ms
ExecutionRunnerTest	815 ms
SchedulerTest	54 ms

2.1 s

<default package>	440 ms
AdminViewTest	206 ms
StatisticsViewTest	165 ms
ExecutionRunnerTest	48 ms
SchedulerTest	21 ms

0.4 s

Handle Classes with State

```
class DesignViewTest {
    private val repo: DesignRepository = mockk()
    private lateinit var view: DesignView
    @BeforeEach
    fun init() {
        clearMocks(repo)
        view = DesignView(repo)
    }
    @Test
    fun changeButton() {
        assertThat(view.button.caption).isEqualTo("Hi")
        view.changeButton()
        assertThat(view.button.caption).isEqualTo("Guten Tag")
    }
}
```

stateful

re-creation required

Spring Integration

All-Open Compiler Plugin

```
@Configuration
class SpringConfiguration{
  @Bean fun objectMapper()
      = ObjectMapper().registerKotlinModule()
}
```

```
BeanDefinitionParsingException: Configuration problem:
@Configuration class 'SpringConfiguration' may not be final.
```

```
<dependency>
 <groupId>org.jetbrains.kotlin</groupId>
 <artifactId>kotlin-maven-allopen</artifactId>
 <version>${kotlin.version}</version>
</dependency>
```

```
<compilerPlugins>
  <plugin>spring</plugin>
</compilerPlugins>
```

Constructor Injection for Spring-free Testing

```
@Component
class DesignController(
    private val designRepo: DesignRepository,
    private val designClient: DesignClient,
) {}
```

Easy to test Logic without Spring:

```
val repo: DesignRepository = mockk()
val client: DesignClient = mockk()
val controller = DesignController(repo, client)
```

Utilize Data Classes

Data Classes for Assertions

```
org.junit.ComparisonFailure: expected:<[2]> but was:<[1]>
Expected :2
Actual   :1
```

???

```
assertThat(actualDesign.id).isEqualTo(2)
assertThat(actualDesign.userId).isEqualTo(9)
assertThat(actualDesign.name).isEqualTo("Cat")
```

Data Classes for Assertions

```
val expectedDesign = Design(id = 2, userId = 9, name = "Cat")
assertThat(actualDesign).isEqualTo(expectedDesign)
```

```
org.junit.ComparisonFailure: expected:<Design(id=[2], userId=9,
name=Cat...> but was:<Design(id=[1], userId=9, name=Cat...>
Expected :Design(id=2, userId=9, name=Cat)
Actual   :Design(id=1, userId=9, name=Cat)
```

self-explanatory

Data Classes for Assertions

```
assertThat(actualDesigns).containsExactly(
    Design(id = 1, userId = 9, name = "Cat"),
    Design(id = 2, userId = 4, name = "Dog")
)
```

```
Expecting:
  <[Design(id=1, userId=9, name=Cat),
    Design(id=2, userId=4, name=Dogggg)]>
to contain exactly (and in same order):
  <[Design(id=1, userId=9, name=Cat),
    Design(id=2, userId=4, name=Dog)]>
but some elements were not found:
  <[Design(id=2, userId=4, name=Dog)]>
and others were not expected:
  <[Design(id=2, userId=4, name=Dogggg)]>
```

Great!

Data Classes for Assertions

Single Element

```
assertThat(actualDesign)
    .isEqualToIgnoringGivenFields(expectedDesign,"id")
assertThat(actualDesign)
    .isEqualToComparingOnlyGivenFields(expectedDesign,"name")
```

Lists

```
assertThat(actualDesigns)
    .usingElementComparatorIgnoringFields("id")
    .containsExactly(expectedDesign1, expectedDesign2)
assertThat(actualDesigns)
    .usingElementComparatorOnFields("name")
    .containsExactly(expectedDesign1, expectedDesign2)
```

Helper Function for Object Creation

```
val testDesign = Design(
      id = 1,
      userId = 9
      name = "Fox",
      dateCreated = Instant.now(),
      tags = mapOf()
)
val testDesign2 = Design(
      id = 2,
      userId = 9
      name = "Cat",
      dateCreated = Instant.now(),
      tags = mapOf()
)
```

- Bloats code
- Are all props relevant for the test?

Helper Function for Object Creation

```
fun createDesign(
    id: Int = 1,
    name: String = "Cat",
    date: Instant = Instant.ofEpochSecond(1518278198),
    tags: Map<Locale, List<Tag>> = mapOf(
        Locale.US to listOf(Tag(value = "$name in English")),
    )
) = Design(
    id = id,
    userId = 9,
    name = name,
    dateCreated = date,
    tags = tags
)
```

```
// Usage:
val testDesign = createDesign()
val testDesign2 = createDesign(
        id = 1,
        name = "Fox"
)
```

Concise

Helper Function for Object Creation

CurrentTest.kt:

```
repo.saveAll(
  createDesign(isEnabled = true, language = Locale.US),
  createDesign(isEnabled = true, language = Locale.GERMANY),
  createDesign(isEnabled = false, language = Locale("nl","NL"))
)
```

Tailored Creation Function for CurrentTest

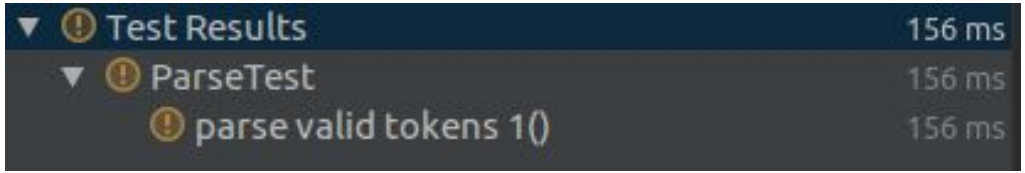
Helper Function for Object Creation

CurrentTest.kt:

```
fun createDesign(
    isEnabled: Boolean,
    language: Locale
) = createDesign(
    description = createDescription(
      translations = createTranslationsFor(language)
    ),
    state = if (isEnabled) createDisabledState() else
createEnabledState()
)
```

CreationUtils.kt

Data Classes for Parameterized Tests




```
@Test
fun `parse valid tokens`() {
  assertThat(parse("1511443755_2")).isEqualTo(Token(1511443755, "2"))
  assertThat(parse("151175_13521")).isEqualTo(Token(151175, "13521"))
  assertThat(parse("151144375_id")).isEqualTo(Token(151144375, "id"))
  assertThat(parse("1511443759_1")).isEqualTo(Token(1511443759, "1"))
  assertThat(parse(null)).isEqualTo(null)
}
```

Which one failed?

Data Classes for Parameterized Tests

```
data class TestData(
    val input: String?,
    val expected: Token?
)
```

Data Classes for Parameterized Tests

```
@ParameterizedTest
@MethodSource("validTokenProvider")
fun `parse valid tokens`(testData: TestData) {
  assertThat(parse(testData.value)).isEqualTo(testData.expectedToken)
}
private fun validTokenProvider() = Stream.of(
  TestData(input = "1511443755_2", expected = Token(1511443755, "2")),
  TestData(input = "151175_13521", expected = Token(151175, "13521")),
  TestData(input = "151144375_id", expected = Token(151144375, "id")),
                                                                  "")),
```

- Test Results
  - ParseTest
    - parse valid tokens(TestData)
      - [1] TestData(input=1511443755_2, expected=Token(timestamp=1511443755, id=2))
      - [2] TestData(input=151175_13521, expected=Token(timestamp=151175, id=13521))
      - [3] TestData(input=151144375_id, expected=Token(timestamp=151144375, id=id))
      - [4] TestData(input=15114437599_1, expected=Token(timestamp=15114437599, id=2))
      - [5] TestData(input=null, expected=null)

Conclusion

```
open class UserRepository
```

```
class UserControllerTest {
    companion object {
        @JvmStatic lateinit var controller: UserController
        @JvmStatic lateinit var repo: UserRepository
        @BeforeClass @JvmStatic  initialize() {
            repo = mock()
            controller = UserController(repo)
        }
    }
    @Test
    fun findUser_UserFoundAndHasCorrectValues() {
        `when`(repo.findUser(1)).thenReturn(User(1, "Peter"))
        val user = controller.getUser(1)
        assertEquals(user?.name, "Peter")
    }
}
```

```
class UserControllerTest {
    private val repo: UserRepository = mockk()
    private val controller = UserController(repo)
    @Test
    fun `find user with correct values`() {
        every { repo.findUser(1) } returns User(1, "Peter")
        val user = controller.getUser(1)
        assertEquals(user).isEqualTo(User(1, "Peter"))
    }
}
```

Best Practices for Testing in Kotlin

JUnit5 ♥ Kotlin

@TestInstance(PER_CLASS)

Naming, Grouping

Backticks

@Nested

Libraries

Choose your own gear

Mock Handling

Don't recreate; reset!

MockK

Data Classes FTW

Equals Assertions

Creation Helper

@ParameterizedTest

Philipp Hauer's Blog

Web Architecture, Java Ecosystem, Software Craftsmanship

Best Practices for Unit Testing in Kotlin

POSTED ON FEB 12, 2018

Unit Testing in Kotlin is fun and tricky at the same time. We can benefit a lot from Kotlin's powerful language features to write readable and concise unit tests. But in order to write idiomatic Kotlin test code in the first place, there is a certain test setup required. This post contains best practices and guidelines to write unit test code in Kotlin that is idiomatic, readable, concise and produces reasonable failure messages.

https://blog.philipphauer.de/best-practices-unit-testing-kotlin/

Thank you!

@philipp_hauer

Spreadshirt

KotlinConf, Amsterdam

Oct 05, 2018

Backup Slides

Test-Specific Extension Functions

```
mvc.perform(get("designs/123?platform=$invalidPlatform"))
    .andExpect(status().isBadRequest)
    .andExpect(jsonPath("errorCode").value(code))
    .andExpect(jsonPath("details", startsWith(msg)))
```

```
fun ResultActions.andExpectErrorPage(code: Int, msg: String) =
    this.andExpect(status().isBadRequest)
    .andExpect(jsonPath("errorCode").value(code))
    .andExpect(jsonPath("details", startsWith(msg)))

// Usage:
mvc.perform(get("designs/123?platform=$invalidPlatform"))
    .andExpectErrorPage(130, "Invalid platform.")
```

Spring Integration

Mock-based Spring Test Context

```
@ExtendWith(SpringExtension::class)
@WebMvcTest(DesignController::class)
@Import(TestConfig::class)
class DesignControllerTest {
    @Autowired private lateinit var mvc: MockMvc
    @Autowired private lateinit var repoMock: DesignRepository
    @BeforeEach
    fun init() {
        clearMocks(repoMock)
    }
    @Test
    fun test() {}
}
```

```
@Configuration
private class TestConfig {
    @Bean
    fun repoMock(): DesignRepository
      = mockk()
}
```

Spring Test Context for an Integration Test

```
@Configuration
private class TestConfig {
    @Bean fun repo() = repo
    private val repo: DesignRepository
    init {
        val mongo = startMongoContainer()
        val mongoTemplate = createMongoTemplate(mongo.host, mongo.port
        repo = DesignRepository(mongoTemplate)
    }
}
```

Initial setup available for Spring

About Spreadshirt

Spreadshirt

For two years