Interactor: тестування бізнес-логіки

Попереднього разу ми навчилися визначати, які саме методи потрібно перевіряти, а сьогодні ви дізнаєтеся, як написати тести для класу CreateOrderInteractor, який містить усю бізнес-логіку додатку CleanStore.

Interactor: створення XCTestCase

У Xcode виберіть File -> New -> File, а потім Test Case Class. Для класу введіть назву CreateOrderInteractorTests. Переконайтеся, що обрано схему CleanStoreTests. Натисніть кнопку Create. Щойно ви створили файл CreateOrderInteractorTests.swift:

```
import UIKit
import XCTest

class CreateOrderInteractorTests: XCTestCase {
    override func setUp() {
        super.setUp()
        // Put setup code here. This method is called before
           the invocation of each test method in the class.
    }

    override func tearDown()  {
        // Put teardown code here. This method is called
           after the invocation of each test method in the
           class.
        super.tearDown() } }
```

Із моєї останньої статті ви повинні були засвоїти, що перевірці підлягають, в основному, **граничні методи**. Тому давайте звернемо нашу увагу на вхідні і вихідні протоколи у CreateOrderInteractor:

```
protocol CreateOrderInteractorInput {
     var shippingMethods: [String] { get }

     func formatExpirationDate(request:
                    CreateOrder_FormatExpirationDate_Request)
}

protocol CreateOrderInteractorOutput {
     func presentExpirationDate(response:
                    CreateOrder_FormatExpirationDate_Response)
}
```

Одна з переваг архітектури **Clean Swift** полягає у тому, що необхідні для тестування методи знаходяться у верхній частині файлу. Для CreateOrderInteractor нам потрібно написати тести для змінної shippingMethods і методу formatExpirationDate(). Ви можете запитати: "Як перевірити змінну"? Справа в тому, що змінна shippingMethods — це також геттер-метод, який нічим не відрізняється від звичайного методу.

Ваш перший тест

Давайте напишемо наш перший тест, щоб переконатися, що **Interactor**-метод `formatExpirationDate()` викликає у **Presenter** метод `presentExpirationDate()`. Врешті-решт, форматування даних — це робота для **Presenter**. Таким чином, нам важливо переконатися, що **Presenter** виконує свої обов'язки.

Для того, щоб отримати посилання на `CreateOrderInteractor`, необхідно спочатку створити екземпляр досліджуваного об'єкта `CreateOrderViewController`. Після побудови і запуску додатка за допомогою **iOS** необхідний нам `інстанс` завантажиться із `Storyboard`. Але у випадку з тестами це не працює. Необхідно створити свій власний `інстанс` із `Storyboard`.

Давайте відкриємо файл `Storyboard`. Переконаємось, що праворуч в `Utilities Area` у розділі `File Inspector` обрано `CleanStoreTests`. Це засвідчить про те, що наш тестовий пакет бачить `CreateOrderInteractor`.

Додамо наступний метод у файл `CreateOrderInteractorTests.swift`:

```
func
testFormatExpirationDateShouldAskPresenterToFormatExpirationD
ate() {
    let bundle = NSBundle(forClass: self.dynamicType)
    let storyboard = UIStoryboard(name: "Main", bundle:
bundle)
    let createOrderViewController =
storyboard.instantiateViewControllerWithIdentifier("CreateOrd
erViewController") as! CreateOrderViewController
    let createOrderInteractor =
createOrderViewController.output as! CreateOrderInteractor
}
```

Далі все просто. Спочатку обираємо `bundle`, **потім** `Storyboard` **і нарешті створюємо** `CreateOrderViewController`. **Таким чином,** `createOrderViewController.output` **— це і є наш Interactor. В архітектурі Clean Swift компонент configurator містить усі необхідні дані для налаштування VIP-циклу.**

ОНОВЛЕННЯ
Насправді, є досить простий спосіб створення досліджуваного об’єкта. Ми можемо створити екземпляр CreateOrderInteractor без використання Storyboard.

Такий підіхід більш раціональний! Раніше я копіював і вставляв код з існуючого проекту в свої тести.

Тепер, коли ми перевіряємо **Interactor**, там не повинно бути ніякого UI або **View Controller.**

Так ми зменшуємо розмір методу testFormatExpirationDateShouldAskPresenterToFormatExpirationDate():

```
func testFormatExpirationDateShouldAskPresenterToFormatExpirationDate() {
    createOrderInteractor = CreateOrderInteractor()
}
```

Давайте скоригуємо наш тестовий код:

```
func testFormatExpirationDateShouldAskPresenterToFormatExpirationDate() {
    createOrderInteractor = CreateOrderInteractor()

    // Given
    let request = CreateOrder_FormatExpirationDate_Request(date: NSDate())

    // When
    createOrderInteractor.formatExpirationDate(request)

    // Then …
}
```

Для виклику методу `formatExpirationDate()` нам потрібно передати йому аргумент `CreateOrder_FormatExpirationDate_Request` у вигляді звичайної структури, що містить поточну дату.

Нехай spy працює на вас

Що можна сказати про розділ `Then...`? Як переконатися, що у компонента **Presenter** викликається метод `presentExpirationDate()`? Щоб відповісти на ці питання ми спочатку поговоримо про mocking.

Для `CreateOrderInteractor` виходом виступає `CreateOrderPresenter`. Нам не потрібно створювати екземпляр **Presenter** для перевірки **Interactor**. Це тому, що ми зацікавлені лише в перевірці поведінки `CreateOrderInteractor` у `CreateOrderInteractorTests`. Ми не будемо переглядати, що робить `CreateOrderPresenter` після того, як **Interactor** закінчує свою роботу і передає `потік управління` до **Presenter**.

Що робити далі? Створимо **spy** для **Presenter**. Він повинен потай і мовчки записувати все, що з ним відбувається під час виконання тесту (зараз саме час розглянути різні типи `test doubles`).

Процес створення **spy**, **stub** і **mock** настільки простий, що нам не доведеться розбиратися з новими залежностями.

Ось приклад написання **spy:**

```
class CreateOrderInteractorOutputSpy:
CreateOrderInteractorOutput {
    var presentExpirationDateCalled = false

    func presentExpirationDate(response:
CreateOrder_FormatExpirationDate_Response) {
        presentExpirationDateCalled = true
    }
}
```

Просто, чи не так?

Вихід Interactor відповідає протоколу `CreateOrderInteractorOutput`**, так що наш spy повинен зробити те ж саме — реалізувати метод** `presentExpirationDate()`**. Під час виклику методу потрібно просто встановити** `presentExpirationDateCalled = true`**. Пізніше для перевірки факту виклику можна по черзі змінювати значення** `presentExpirationDateCalled` **з істинного на хибне і навпаки.**

Ось твердження:

```
XCTAssert(createOrderInteractorOutputSpy.presentExpirationDat
eCalled, "Formatting an expiration date should ask presenter
to do it")
```

Також перед запуском теста необхідно встановити `вихід CreateOrderInteractor` **на наш новий spy замість реального Presenter:**

```
let createOrderInteractorOutputSpy =
CreateOrderInteractorOutputSpy()
createOrderInteractor.output = createOrderInteractorOutputSpy
```

Тепер наш тест виглядає так:

```
class CreateOrderInteractorTests: XCTestCase {
    override func setUp() {
        super.setUp()
        // Put setup code here. This method is called before
the invocation of each test method in the class.
    }

    override func tearDown() {
        // Put teardown code here. This method is called
after the invocation of each test method in the class.
        super.tearDown()
    }

    class CreateOrderInteractorOutputSpy:
CreateOrderInteractorOutput  {
        var presentExpirationDateCalled = false

        func presentExpirationDate(response:
CreateOrder_FormatExpirationDate_Response)  {
            presentExpirationDateCalled = true
        }
    }
```

```
    func
testFormatExpirationDateShouldAskPresenterToFormatExpirationD
ate() {
        createOrderInteractor = CreateOrderInteractor()

        // Given
        let createOrderInteractorOutputSpy =
CreateOrderInteractorOutputSpy()
        createOrderInteractor.output =
createOrderInteractorOutputSpy
        let request =
CreateOrder_FormatExpirationDate_Request(date: NSDate())

        // When
        createOrderInteractor.formatExpirationDate(request)

        // Then

XCTAssert(createOrderInteractorOutputSpy.presentExpirationDat
eCalled, "Formatting an expiration date should ask presenter
to do it")
    }
}
```

Зробіть замовлення зараз і його відправлять вже сьогодні

У протоколі `CreateOrderInteractorInput` **є ще один метод. Ми можемо розглядати змінну** `shippingMethods` **як геттер-метод. Таким чином, її також потрібно перевірити.**

Коли View Controller звертається до вихідного параметру shippingMethods, потік управління негайно повертається назад у View Controller. Змінна shippingMethods служить для швидкого доступу до VIP-циклу.

Тест дуже простий:

```
func
testShippingMethodsShouldReturnAllAvailableShippingMethods()
{
    createOrderInteractor = CreateOrderInteractor()

    // Given
    let allAvailableShippingMethods = [
        "Standard Shipping",
        "Two-Day Shipping ",
        "One-Day Shipping "
    ]

    // When
    let returnedShippingMethods =
createOrderInteractor.shippingMethods

    // Then
    XCTAssertEqual(returnedShippingMethods,
allAvailableShippingMethods, "Shipping Methods should list
all available shipping methods»)
}
```

Установка наступна: ми просто стверджуємо, що методи повернення ідентичні доступним методам доставки.

Давайте зробимо невеличкий рефакторинг, щоб усунути дублювання коду і дамо цьому методу назву `setUp()`. Для початку тестування `CreateOrderInteractorTests` нажміть комбінацію ⌘–U:

```
import UIKit
import XCTest

class CreateOrderInteractorTests: XCTestCase {
    // Subject under test
    var createOrderInteractor: CreateOrderInteractor!

    // MARK: Test lifecycle
    override func setUp() {
        super.setUp()
        setupCreateOrderInteractor()
    }

    override func tearDown() {
        super.tearDown()
    }

    // MARK: Test setup
    func setupCreateOrderInteractor() {
        createOrderInteractor = CreateOrderInteractor()
    }

    // MARK: Test doubles
    class CreateOrderInteractorOutputSpy:
CreateOrderInteractorOutput {
        var presentExpirationDateCalled = false
```

```
        func presentExpirationDate(response:
CreateOrder_FormatExpirationDate_Response)  {
            presentExpirationDateCalled = true
        }
    }

    // MARK: Test expiration date
    func
testFormatExpirationDateShouldAskPresenterToFormatExpirationD
ate() {
        // Given
        let createOrderInteractorOutputSpy =
CreateOrderInteractorOutputSpy()
        createOrderInteractor.output =
createOrderInteractorOutputSpy
        let request =
CreateOrder_FormatExpirationDate_Request(date: NSDate())

        // When
        createOrderInteractor.formatExpirationDate(request)

        // Then

XCTAssert(createOrderInteractorOutputSpy.presentExpirationDat
eCalled, "Formatting an expiration date should ask presenter
to do it")
    }
```

```
    // MARK: Test shipping methods
    func
testShippingMethodsShouldReturnAllAvailableShippingMethods()
{
        // Given
        let allAvailableShippingMethods = [
            "Standard Shipping",
            "Two-Day Shipping ",
            "One-Day Shipping "
        ]

        // When
        let returnedShippingMethods =
createOrderInteractor.shippingMethods

        // Then
        XCTAssertEqual(returnedShippingMethods,
allAvailableShippingMethods, "Shipping Methods should list
all available shipping methods")
    }
}
```

Вітаю, ви пройшли довгий шлях і тепер знаєте:

- як розділити свій код на різні компоненти архітектури **Clean Swift**;
- як використовувати **VIP-цикл** для виділення двох логік: бізнесу і представлення;
- як пишуться тести.

Наступним буде `Presenter`. Я покажу, як в нашому додатку CleanStore перевірити `CreateOrderPresenter`. Зразки коду і тесту доступні у репозиторії GitHub.

До наступного посту!