Clean Swift TDD Part 4 — Presenter

У цьому пості ми завершимо розгляд процесу тестування компонента **Presenter** для вибірки переліку Замовлень. Із попередніх статей ви можете дізнатися про те, як тестувати інші компоненти **VIP-циклу**: View Controller, Interactor і Worker.

Робота Presenter полягає у форматуванні даних, які показуються користувачеві.

У нашого Замовлення атрибут date має тип NSDate, а total — тип NSDecimal. Але текстові мітки в табличному рядку для відображення даних мають тип String.

Наш контролер представлення (**View Controller**) не повинен знати, як поля date і total представлені в додатку. Йому просто потрібно отримати кілька стрічок, щоб показати їх користувачеві. Ми розглянемо, як відформатувати дату і суму Замовлення у **Presenter** так, щоб **наш контролер представлення був незалежним від формату самих даних**.

Якщо пізніше ми вирішимо зберігати дату і суму Замовлення по-іншому, нам буде достатньо змінити тільки **Presenter**, в той час як **View Controller** залишиться без змін. Такий підхід спрощує застосування модульного тестування, тому що чітко розділяє вхід і вихід.

Ми навіть можемо вивести `форматтер` на більш високий рівень, щоб його можна було повторно використовувати в різних частинах програми, в яких необхідно відобразити `дату` і суму `Замовлення` в тому ж форматі. Таким чином, зміни у `форматтер` треба буде вносити лише один раз.

Ізоляція залежностей

Ми перевіряємо `ListOrdersPresenter`, який має одну зовнішню залежність — `ListOrdersPresenterOutput`. Тепер давайте “скопіюємо” її шляхом створення `ListOrdersPresenterOutputSpy`.

```
class ListOrdersPresenterOutputSpy: ListOrdersPresenterOutput
{
     // MARK: Method call expectations
     var displayFetchedOrdersCalled = false

     // MARK: Argument expectations
     var listOrders_fetchOrders_viewModel:
ListOrders_FetchOrders_ViewModel!

     // MARK: Spied methods
     func displayFetchedOrders(viewModel:
ListOrders_FetchOrders_ViewModel) {
          displayFetchedOrdersCalled = true
          listOrders_fetchOrders_viewModel = viewModel
     }
}
```

Ми хочемо переконатися, що в результаті виклику методу presentFetchedOrders() в **Presenter** викликається вихідний метод displayFetchedOrders(). Якщо це так, то змінна displayFetchedOrdersCalled отримає значення true (істина).

Ми також хочемо переконатися, що **Presenter** правильно форматує Замовлення для відображення на екрані. Тому необхідно зберегти модель представлення, яка передається у метод displayFetchedOrders() як значення змінної listOrders_fetchOrders_viewModel. Ми можемо написати твердження, щоб перевірити його в секції Then нашого тесту.

Але як повинна виглядати змінна ListOrders_FetchOrders_ViewModel?

Пам'ятаймо, що ми хочемо, щоб наш **View Controller** проводив діагностику моделі представлення даних Замовлення в самому додатку. Таким чином, ми не хочемо створення будь-яких посилань на Замовлення, NSDate або NSDecimal. Текстові мітки приймають тільки строкові значення. Отже, давайте створимо нову структуру DisplayedOrder, щоб представити Замовлення на екрані.

```
struct ListOrders_FetchOrders_ViewModel {
    struct DisplayedOrder {
        var id: String
        var date: String
        var email: String
        var name: String
        var total: String
    }

    var displayedOrders: [DisplayedOrder]
```

```
}
```

Структура DisplayedOrder потрібна тільки для моделі представлення, тому ми можемо просто вставити її всередину структури ListOrders_FetchOrders_ViewModel.

Яку інформацію про Замовлення потрібно знати користувачу? Вона міститься у DisplayedOrder. Якщо наша модель Замовлення містить дані, такі як дата виконання, місце розташування складу, номер партії, тощо, необхідно утриматися від їх розміщення у DisplayedOrder. Користувачеві не потрібно знати про внутрішню структуру компанії.

Я також використовую поле name замість окремих first та last names в DisplayedOrder, оскільки форматування імен є роботою компонента **Presenter**. Скоріш за все, ви виконаєте: name = firstName + a space + lastName в **Presenter** під час форматування Order у DisplayedOrder. Але давайте не будемо забігати вперед. Спочатку напишемо **TDD**-тест.

Після цього ми отримуємо вибірку списку Замовлень, так що displayedOrders = array of displayedOrders в ListOrders_FetchOrders_ViewModel.

Наш перший тест

У компоненті **Present** за допомогою методу `presentFetchedOrders()` ми хочемо перевірити дві речі:

- форматування отриманих `Замовлень` для відображення на екрані — перетворення `[Order] -> [DisplayedOrder]`.
- вказати компоненту **View Controller** на необхідність відобразити на екрані відформатований `список Замовлень` — виконати метод `displayFetchedOrders()` на `вході`.

Давайте напишемо перший тест:

```
func
testPresentFetchedOrdersShouldFormatFetchedOrdersForDisplay()
{
     // Given
     let listOrdersPresenterOutputSpy =
ListOrdersPresenterOutputSpy()
     sut.output = listOrdersPresenterOutputSpy

     let dateComponents = NSDateComponents()
     dateComponents.year = 2007
     dateComponents.month = 6
     dateComponents.day = 29
     let date =
NSCalendar.currentCalendar().dateFromComponents(dateComponent
s)!
```

```
    let orders = [Order(id: "abc123", date: date, email:
"amy.apple@clean-swift.com", firstName: "Amy", lastName:
"Apple", total: NSDecimalNumber(string: "1.23"))]
    let response = ListOrders_FetchOrders_Response(orders:
orders)

    // When
    sut.presentFetchedOrders(response)

    // Then
    let displayedOrders =
listOrdersPresenterOutputSpy.listOrders_fetchOrders_viewModel
.displayedOrders

    for displayedOrder in displayedOrders{
        XCTAssertEqual(displayedOrder.id, "abc123",
"Presenting fetched orders should properly format order ID")
        XCTAssertEqual(displayedOrder.date, "6/29/07",
"Presenting fetched orders should properly format order
date")
        XCTAssertEqual(displayedOrder.email,
"amy.apple@clean-swift.com", "Presenting fetched orders
should properly format email")
        XCTAssertEqual(displayedOrder.name, "Amy Apple",
"Presenting fetched orders should properly format name")
        XCTAssertEqual(displayedOrder.total, "$1.23",
"Presenting fetched orders should properly format total")
    }
}
```

По-перше, ми хочемо використати ListOrdersPresenterOutputSpy, який створили раніше у місці фактичного виходу.

Спочатку створимо відому дату, потім — масив з одним тестовим Замовленням і об'єкт ListOrders_FetchOrders_Response з масивом тестового Замовлення.

Викликавши метод presentFetchedOrders() з цією відповіддю, ми можемо перебирати значення масива displayedOrders з моделі представлення, щоб переконатися, що ідентифікатор, дата, адреса електронної пошти, ім'я та сума відформатовані у строкові значення з метою коректного відображення для користувача.

Давайте перейдемо до другого тесту:

```
func
testPresentFetchedOrdersShouldAskViewControllerToDisplayFetch
edOrders() {
    // Given
    let listOrdersPresenterOutputSpy =
ListOrdersPresenterOutputSpy()
    sut.output = listOrdersPresenterOutputSpy

    let orders = [Order(id: "abc123", date: NSDate(), email:
"amy.apple@clean-swift.com", firstName: "Amy", lastName:
"Apple", total: NSDecimalNumber(string: "1.23"))]
    let response = ListOrders_FetchOrders_Response(orders:
orders)

    // When
    sut.presentFetchedOrders(response)
```

```
    // Then

XCTAssert(listOrdersPresenterOutputSpy.displayFetchedOrdersCa
lled, "Presenting fetched orders should ask view controller
to display them")
}
```

Як і в першому тесті, ми створимо `ListOrdersPresenterOutputSpy` і `ListOrders_FetchOrders_Response` з масивом одного тестового Замовлення.

Коли ми звертаємось до методу `presentFetchedOrders()`, то просто хочемо переконатися, що метод `displayFetchedOrders()` викликається у файлі `ListOrdersPresenterOutputSpy`. Це означає, що компонент **View Controller** отримає команду на відображення списку Замовлень, форматування якого ми перевірили у першому тесті.

Визначення меж

Межі — це досить просто. Нам просто потрібно додати метод `displayFetchedOrders()` в обидва протоколи `ListOrdersPresenterOutput` і `ListOrdersViewControllerInput`.

```
protocol ListOrdersPresenterOutput: class {
    func displayFetchedOrders(viewModel:
ListOrders_FetchOrders_ViewModel)
}
```

```
protocol ListOrdersViewControllerInput {
     func displayFetchedOrders(viewModel:
ListOrders_FetchOrders_ViewModel)
}
```

Зараз просто додайте порожню реалізацію `displayFetchedOrders()` у `ListOrdersViewController`, щоб компілятор не видавав помилку. Вам слід зосередитись просто на тестуванні **`Presenter`** і не хвилюватися щодо того, як бачитиме `Замовлення` користувач.

```
func displayFetchedOrders(viewModel:
ListOrders_FetchOrders_ViewModel) {
}
```

Реалізація логіки

Ми писали тести, щоб переконатися у тому, що **`Presenter`**:

- форматує `список Замовлень` для відображення на екрані
- наказує **`View Controller`** відобразити `список Замовлень`

Давайте зараз реалізуємо ці задачі:

```
class ListOrdersPresenter: ListOrdersPresenterInput {
     weak var output: ListOrdersPresenterOutput!
     let dateFormatter: NSDateFormatter = {
          let dateFormatter = NSDateFormatter()
          dateFormatter.dateStyle = .ShortStyle
```

```
        dateFormatter.timeStyle =
NSDateFormatterStyle.NoStyle

        return dateFormatter
    }()

    let currencyFormatter: NSNumberFormatter = {
        let currencyFormatter = NSNumberFormatter()
        currencyFormatter.numberStyle = .CurrencyStyle

        return currencyFormatter
    }()

    // MARK: Presentation logic

    func presentFetchedOrders(response:
ListOrders_FetchOrders_Response) {
        var displayedOrders:
[ListOrders_FetchOrders_ViewModel.DisplayedOrder] = []

        for order in response.orders {
            let date =
dateFormatter.stringFromDate(order.date!)
            let total =
currencyFormatter.stringFromNumber(order.total!)
            let displayedOrder =
ListOrders_FetchOrders_ViewModel.DisplayedOrder(id:
order.id!, date: date, email: order.email!, name: "\
(order.firstName!) \(order.lastName!)", total: total!)
        displayedOrders.append(displayedOrder)
        }
```

```
        let viewModel =
ListOrders_FetchOrders_ViewModel(displayedOrders:
displayedOrders)
    output.displayFetchedOrders(viewModel)
     }
}
```

Створити NSDateFormatter і NSNumberFormatter досить складно, тому давайте створимо відповідні константи dateFormatter і CurrencyFormatter.

За допомогою метода presentFetchedOrders() ми можемо перебрати Замовлення, отримані як відповідь від компонента **Interactor**. Для кожного Замовлення, ми будете конвертувати дату з NSDate –> String, а суму з NSDecimalNumber –> String. Також ми повинні задати формат поля name як результат конкатенації first name, space, last name.

Далі ми створимо DisplayedOrder і додамо його у масив displayedOrders.

Наступна річ, яку ми повинні зробити за допомогою методу presentFetchedOrders(), полягає в тому, щоб створити ListOrders_FetchOrders_ViewModel з масивом displayedOrders. І, нарешті, тепер ми можемо просто звернутися до вихідного методу displayFetchedOrders(), який містить модель представлення.

Висновки

Нагадаю, що ми зробили сьогодні:

- По-перше, ми виділили залежність шляхом створення `ListOrdersPresenterOutputSpy`.
- По-друге, ми визначили `ListOrders_FetchOrders_ViewModel` як масив з об'єктів `DisplayedOrder` для передачі отриманих типо-незалежних Замовлень компоненту **View Controller** для відображення. Структура `DisplayedOrder` містить тільки дані з типом `Strings`, які готові до використання елементами інтерфейсу `UILabels`.
- Ми написали тест `testPresentFetchedOrdersShouldFormatFetchedOrdersForDisplay()` щоб переконатися, що отримані Замовлення правильно відформатовані.
- Ми також написали тест `testPresentFetchedOrdersShouldAskViewControllerToDisplayFetchedOrders()` щоб переконатися, що метод `displayFetchedOrders()` викликається на виході.
- Під час реалізації методу `presentFetchedOrders()` ми використали `NSDateFormatter` і `NSNumberFormatter` для форматування дати і суми Замовлення відповідно. Результатом став масив з об'єктів `DisplayedOrder`, які складаються тільки зі строкових полів, що підходять для відображення за допомогою елементів `UILabels`.

В процесі роботи у файлі `ListOrdersViewController` ми розмістили метод `displayFetchedOrders()`. Але в даний момент він порожній. Це означає, що отримані `Замовлення` відформатовані для відображення, але насправді не будуть показані. Ми вирішимо цю проблему у наступному пості.

Ви можете знайти повний вихідний код з тестами на `GitHub`.