Тестування через розробку в архітектурі Clean Swift. Interactor

У першій публікації про TDD ви написали тест, щоб впевнитися, що ListOrdersViewController викликає метод fetchOrders() з *ListOrdersInteractor*. Прийшов час перевірити, що робить із запитом **Interactor**. Ви також дізнаєтеся, як делегувати бізнес-логіку отримання переліку Замовлень від **Interactor до Worker**.

Підготовка OrdersWorker для TDD

З огляду на те, що ми робитимемо в майбутньому, нам слід створити **OrdersWorker** для наступного CRUD:

- у fetchOrders() **буде перелічено** отримані Замовлення.
- fetchOrder() **показуватиме** вибране Замовлення.
- createOrder() **створюватиме** нове Замовлення.
- updateOrder() **оновлюватиме** існуюче Замовлення.
- deleteOrder() **видалятиме** Замовлення.

Також Замовлення можна зберігати у **Core Data** або у мережі. Тому результати можуть бути доступні не відразу. Давайте зробимо ці операції асинхронними і повернемо результат в обробник завершального блоку (completion block handler).

Пізніше ви зможете використати цей **OrdersWorker** для створення нового Замовлення. На першому екрані повинна відобразитися сцена ListOrder. Давайте спершу впровадимо метод fetchOrders(). У розробці через тестування **(TDD)** вам не потрібно забігати далеко наперед.

Ізолювання залежностей

Як і у [попередній публікації](), результатом виконання ListOrdersInteractor є тип, який відповідає протоколу ListOrdersInteractorOutput. Він пов'язаний залежністю з **Interactor-ом**, тому давайте зімітуємо його поведінку у ListOrdersInteractorTests.swift.

```
class ListOrdersInteractorOutputSpy:
ListOrdersInteractorOutput {
     // MARK: Method call expectations
     var presentFetchedOrdersCalled = false

     // MARK: Spied methods
     func presentFetchedOrders(response:
ListOrders_FetchOrders_Response) {
       presentFetchedOrdersCalled = true
     }
}
```

Після того, як **Interactor** підхопить перелік Замовлень, він передасть їх у **Presenter**, викликавши метод *presentFetchedOrders()* з вхідним параметром-об'єктом ListOrders_FetchOrders_Response, який містить список Замовлень. Тому вам потрібно перевірити поведінку метода presentFetchedOrders().

Давайте створимо модель ListOrders_FetchOrders_Response, в якій буде міститися також результат виконання - список Замовлень:

```
struct ListOrders_FetchOrders_Response {
    var orders: [Order]
}
```

Це все що вам потрібно зімітувати у **VIP cycle**.

Але у ListOrdersInteractor є ще одна залежність.

З огляду на те, що ми хочемо делегувати підхоплення з ListOrdersInteractor в **OrdersWorker**, потрібно посилатися на **Worker**, який є зовнішнім по відношенню до I**nteractor**-а.

Цей **OrdersWorker** не є результатом виконання I**nteractor**-а і не бере участі у **VIP cycle** явним чином. Але він є невід'ємною частиною підхоплення Замовлень. Якщо **OrdersWorker** натрапляє на помилку або не повертає жодних результатів, ListOrdersInteractor не знає, як рухатися далі. Тому ListOrdersInteractor залежить від **OrdersWorker**.

Давайте зімітуємо і його:

```
class OrdersWorkerSpy: OrdersWorker {
     // MARK: Method call expectations
     var fetchOrdersCalled = false

     // MARK: Spied methods
     override func fetchOrders(completionHandler: (orders:
[Order]) -> Void) {
          fetchOrdersCalled = true
          completionHandler(orders: [])
     }
}
```

Зімітуйте метод `fetchOrders()` **і зробіть так, щоб він повертав масив об'єктів** `Order` **у** `completion Handler`**. Зараз ви можете просто зробити так, щоб він повертав пустий масив. Вам також потрібно пам'ятати, що метод насправді був викликаний** `fetchOrdersCalled`**.**

`OrdersWorker` покищо не існує. Тому створіть його за допомогою методу `fetchOrders()`**. Залишаючись вірними принципам TDD, виберемо найпростіший варіант імплементації, а саме повернення порожнього масиву замовлень.**

```
class OrdersWorker {
     func fetchOrders(completionHandler: (orders: [Order]) ->
Void) {
          completionHandler(orders: [])
     }
}
```

Також потрібно створити модель `Order`. Поки що залишимо її порожньою. Заповнити її атрибутами можна буде пізніше, коли це буде необхідно.

```
struct Order {
}
```

Гаразд, тепер проект можна компілювати. Повернемося до основної роботи.

Спершу написання тесту

У `ListOrdersInteractor` вам потрібно делегувати підхоплення Замовлень **`OrdersWorker`**. Давайте напишемо тест `testFetchOrdersShouldAskOrdersWorkerToFetchOrders()`, щоб перевірити, як це працює.

```
func
testFetchOrdersShouldAskOrdersWorkerToFetchOrdersAndPresenter
ToFormatResult() {
    // Given
    let listOrdersInteractorOutputSpy =
ListOrdersInteractorOutputSpy()
    sut.output = listOrdersInteractorOutputSpy
    let ordersWorkerSpy = OrdersWorkerSpy()
    sut.ordersWorker = ordersWorkerSpy
```

```
    // When
    let request = ListOrders_FetchOrders_Request()
    sut.fetchOrders(request)

    // Then
    XCTAssert(ordersWorkerSpy.fetchOrdersCalled,
"FetchOrders() should ask OrdersWorker to fetch orders”)


XCTAssert(listOrdersInteractorOutputSpy.presentFetchedOrdersC
alled, "FetchOrders() should ask presenter to format orders
result")
}
```

Спершу використайте `ListOrdersInteractorOutputSpy` і `OrdersWorkerSpy` **замість справжнього** `ListOrdersPresenter` і **`OrdersWorker`, тому що ці залежності виходять за межі об’єкта тестування.**

Потім викличте `fetchOrders()` **за допомогою об’єкта** `ListOrders_FetchOrders_Request`.

Нарешті вкажіть, що метод `fetchOrders()` **викликається** `OrdersWorkerSpy`, **а метод** `presentFetchedOrders()` **—** `ListOrdersInteractorOutputSpy`.

Визначення меж

Як і у випадку з ListOrdersViewController **розмістіть метод** presentFetchedOrders() **поміж** ListOrdersInteractor **і** ListOrdersPresenter. **Давайте напишемо тест** testFetchOrdersShouldAskOrdersWorkerToFetchOrders(), **щоб це перевірити.**

```
protocol ListOrdersInteractorOutput {
    func presentFetchedOrders(response:
ListOrders_FetchOrders_Response)
}

protocol ListOrdersPresenterInput {
    func presentFetchedOrders(response:
ListOrders_FetchOrders_Response)
}
```

Покищо додайте порожню імплементацію presentFetchedOrders() **у** ListOrdersPresenter. **Вам просто необхідно впевнитися, що** **Interactor** **надсилає** P**resenter-у запит на форматування отриманих** Замовлень. **Вас не цікавить, як саме він це робить. Ви перевірите цей метод згідно** **TDD, коли писатимете тести для** ListOrdersPresenter.

```
func presentFetchedOrders(response:
ListOrders_FetchOrders_Response) {
}
```

Впровадження логіки

У ListOrdersInteractor вам потрібно подбати про наступне:

- створіть екземпляр **OrdersWorker** і викличте його метод fetchOrders().
- після повернення отриманих Замовлень у блок викличте метод presentFetchedOrders() для результатів, щоб відправити їх у **VIP cycle**.

Змініть ListOrdersInteractor.swift наступним чином:

```
class ListOrdersInteractor: ListOrdersInteractorInput {
    var output: ListOrdersInteractorOutput!
    var ordersWorker = OrdersWorker()

     // MARK: Business logic
    func fetchOrders(request:
ListOrders_FetchOrders_Request) {
        ordersWorker.fetchOrders { (orders) -> Void in
            let response =
ListOrders_FetchOrders_Response(orders: orders)
            self.output.presentFetchedOrders(response)
    }
    }
}
```

Не переплутайте синхронний метод fetchOrders() у ListOrdersInteractor **з асинхронним методом** fetchOrders() у OrdersWorker. **Просто у них однакові імена.**

Тепер тест повинен завершитися успішно. Код і тест для цього прикладу **TDD** знаходяться на GitHub.

Підведемо підсумки

Давайте згадаємо, що ви зробили, використовуючи **принципи тестування через розробку** разом з **Clean Swift**.

- ви ізолювали залежність компонента `ListOrdersPresenter`, створивши `ListOrdersInteractorOutputSpy`. Ви також ізолювали залежність компонента `OrdersWorker`, створивши `OrdersWorkerSpy`.

- ви написали тест `testFetchOrdersShouldAskOrdersWorkerToFetchOrdersAndPresenterToFormatResult()`, який перевіряє, що задача по підхопленню `Замовлень` делегується компоненту **OrdersWorker**. Ви також переконалися, що кінцевий результат у вигляді переліку `Замовлень` передається у **Presenter** згідно **VIP cycle** для форматування.

- ви впровадили у компонент **OrdersWorker** мінімальний метод `fetchOrders()` для повернення порожнього масиву `Замовлень`. Після того, як ви протестуєте **OrdersWorker** у наступному пості, ми змінимо цей метод так, щоб він повертав більш осмислені результати.

- ви додали у ListOrdersPresenter порожній метод presentFetchedOrders(). Ви знатимете що робити далі після того, як протестуєте ListOrdersPresenter.

- ви завершили впровадження методу fetchOrders() для ListOrdersInteractor, викликавши fetchOrders() для **OrdersWorker**. Після асинхронного повернення результату списку Замовлень у блок completion handler ви викликаєте метод presentFetchedOrders() для **Presenter** з метою форматування.

Наступного разу ви дізнаєтеся, як тестувати **OrdersWorker**. З огляду на те, що Замовлення можна зберігати в **Core Data** або у мережі, ви створите спільне API для інтерфейсу за допомогою **OrdersWorker**. Ви створите сховище даних для швидшого запуску тестів, сховище **Core Data** для локального збереження даних і сховище API для зберігання замовлень у бекенді.