[YSK] Clean Swift 스터디

학습 내용 공유 1차

박혜정 (2023년 8월 18일)

오늘 공유의 순서

이런 걸 얘기해보면 좋을 것 같았습니다

- MVC의 dark side
- VIP cycle의 컴포넌트
- 클린 스위프트의 목표

MVC의 dark side

왜 그렇게 짜면 안 되는 걸까

Massive View Controller
악의 근원

- view controller가 너무 많은 역할을 가지고 있다.
  - 좋은 테스트를 작성하기 어렵고 누수 지점을 찾기 어렵다.
  - 전체 코드의 맥락을 이해하기 어렵다.
  - 문제 해결이 어려워진다.
  - 컨텍스트 스위칭이 어려워진다.
  - 유지 보수가 많은 비용이 든다.

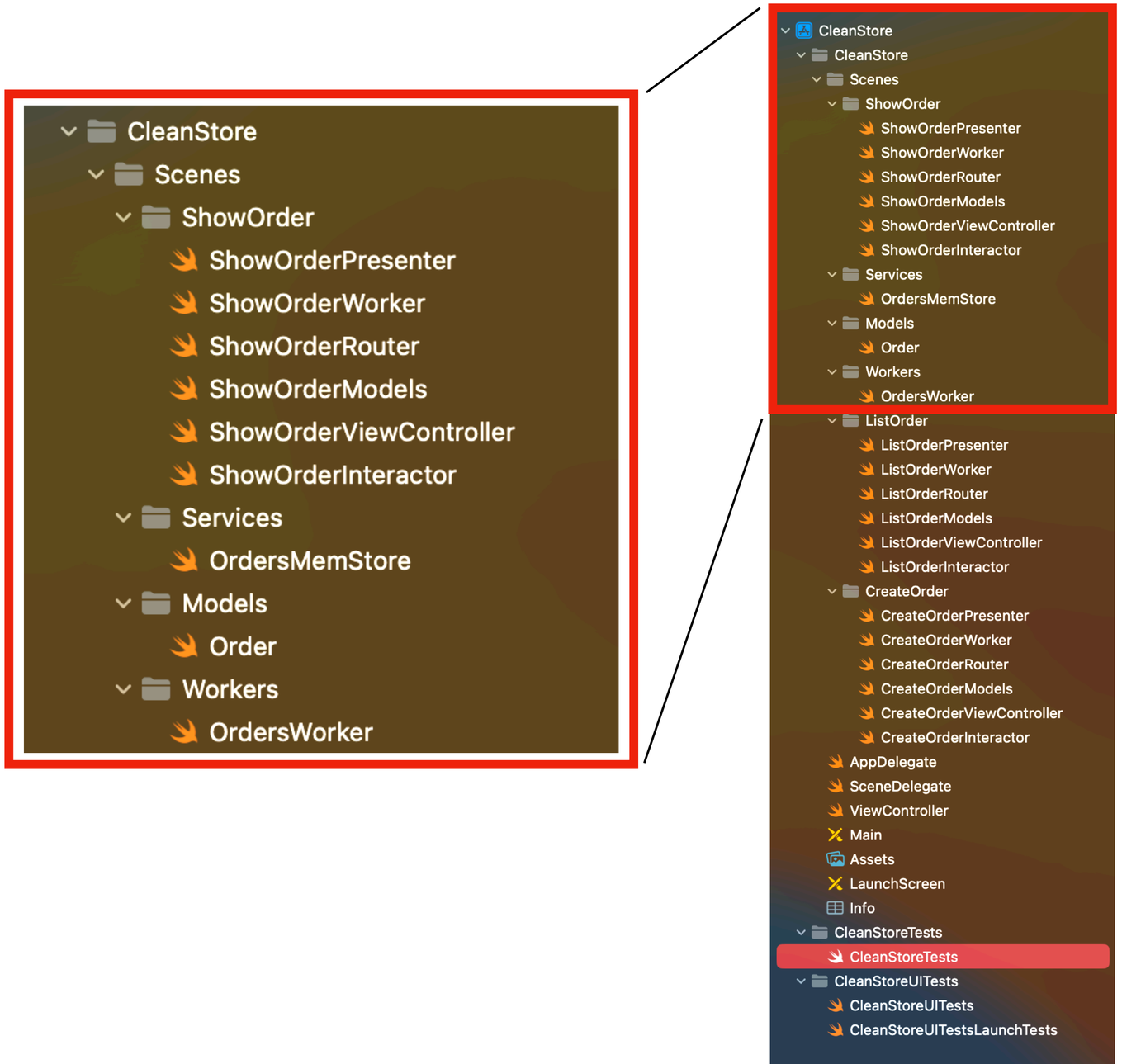
리팩토링 하면 되잖아요
아 할 수 있다구요

- massive viewDidLoad() 를 쪼개도 여전히 massive ViewController
- NetworkManager 역할을 분리한다고 해도 여전히 의존성이 너무 높다.
- 효율적인 개발과 유지 보수를 위해서는 Refactoring 이 아니라 애초에 **Factored 코드**가 필요

VIP cycle

주요 컴포넌트와 주변 요소의 역할과 관계

각 scene 에 대해 하나의 컴포넌트 그룹 생성


CleanStore
Scenes
ShowOrder
ShowOrderPresenter
ShowOrderWorker
ShowOrderRouter
ShowOrderModels
ShowOrderViewController
ShowOrderInteractor
Services
OrdersMemStore
Models
Order
Workers
OrdersWorker
CleanStore
CleanStore
Scenes
ShowOrder
ShowOrderPresenter
ShowOrderWorker
ShowOrderRouter
ShowOrderModels
ShowOrderViewController
ShowOrderInteractor
Services
OrdersMemStore
Models
Order
Workers
OrdersWorker
ListOrder
ListOrderPresenter
ListOrderWorker
ListOrderRouter
ListOrderModels
ListOrderViewController
ListOrderInteractor
CreateOrder
CreateOrderPresenter
CreateOrderWorker
CreateOrderRouter
CreateOrderModels
CreateOrderViewController
CreateOrderInteractor
AppDelegate
SceneDelegate
ViewController
Main
Assets
LaunchScreen
Info
CleanStoreTests
CleanStoreTests
CleanStoreUITests
CleanStoreUITests
CleanStoreUITestsLaunchTests

각 use case의 각의 phase 마다 하나의 인터페이스 호출

ViewController -> BusinessLogic
Interactor -> PresentationLogic
Presenter -> DisplyLogic

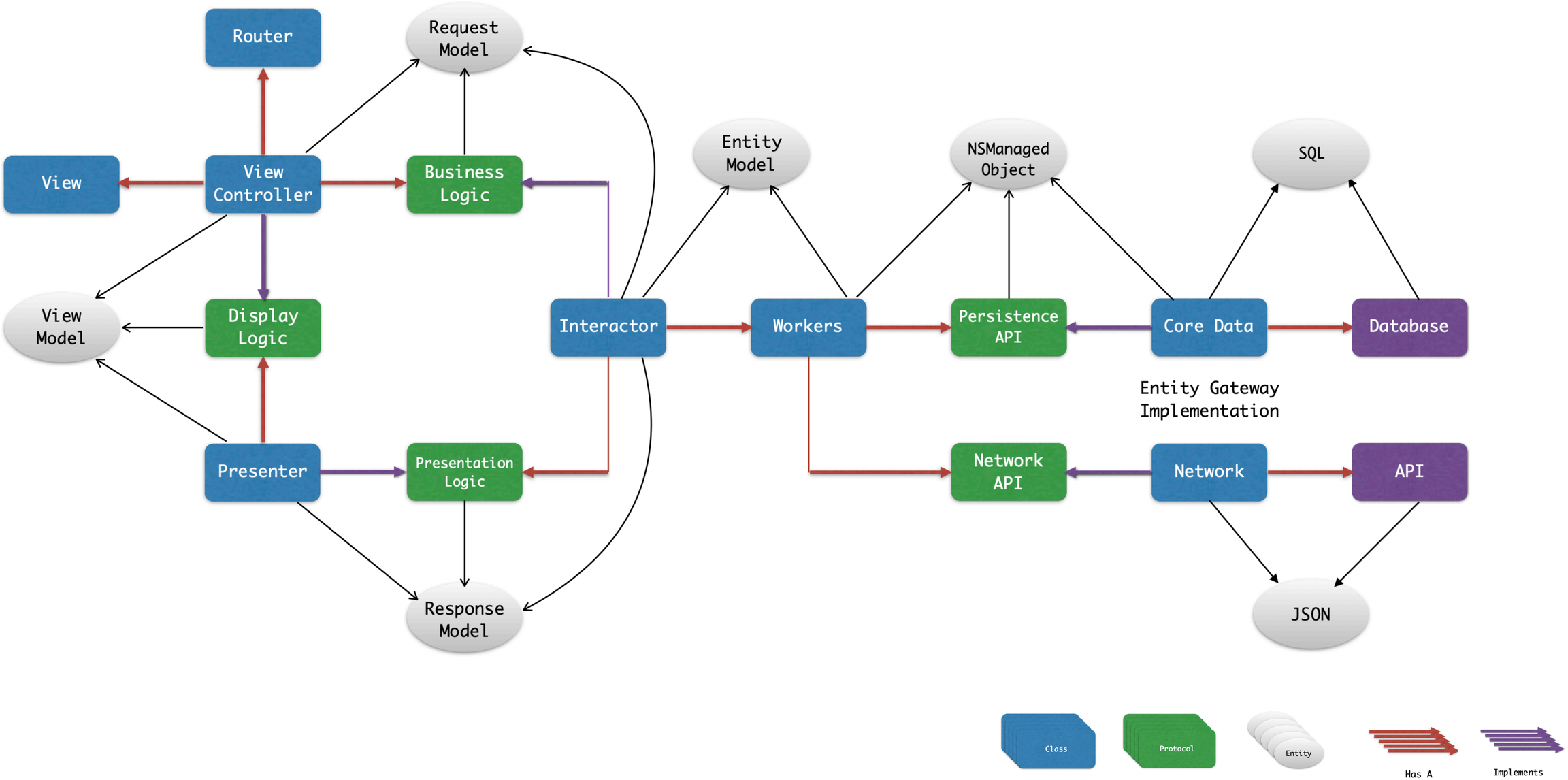
```
protocol CreateOrderBusinessLogic // Interactor
{
  var shippingMethods: [ShipmentMethod] { get }

(1) func formatExpirationDate(request: CreateOrder.FormatExpirationDate.Request)

(2) func showOrderToEdit(request: CreateOrder.EditOrder.Request)

(3) func createOrder(request: CreateOrder.CreateOrder.Request)

(4) func updateOrder(request: CreateOrder.UpdateOrder.Request)
}
```

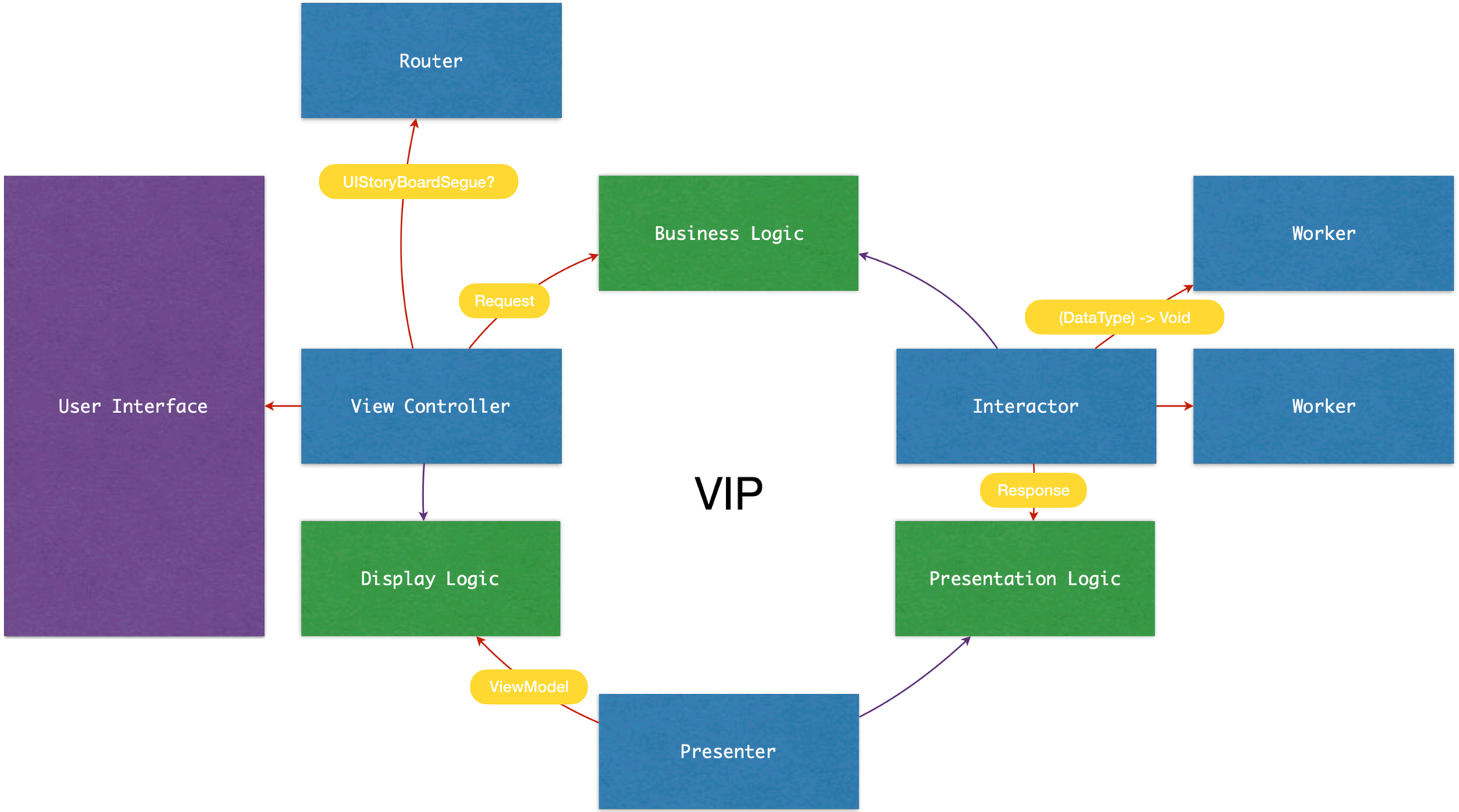
(1) Date Picker의 value가 바뀔 때 마다 text field에 표시되는 Date가 바뀐다.

(2) CreateOrder 화면이 로드되면 편집할 주문의 기존 정보를 각 필드에 표시한다.

(3) save 버튼을 누르면 입력된 정보로 새 주문을 생성한다.

(4) save 버튼을 누르면 입력된 정보로 기존 주문의 정보를 업데이트 한다.


Router
Request Model
View
View Controller
Business Logic
Entity Model
NSManaged Object
SQL
View Model
Display Logic
Interactor
Workers
Persistence API
Core Data
Database
Entity Gateway Implementation
Presenter
Presentation Logic
Network API
Network
API
Response Model
JSON
Class
Protocol
Entity
Has A
Implements


Router
UIStoryBoardSegue?
Business Logic
Worker
Request
(DataType) -> Void
User Interface
View Controller
Interactor
Worker
VIP
Response
Display Logic
Presentation Logic
ViewModel
Presenter

View Controller

이벤트 컨트롤 타워

1. **controller**
   - 사용자 입력(UI event) 또는 view lifecycle 처리
   - 알맞은 request를 생성해서 알맞은 비즈니스 로직을 트리거 *BussinessLogic 소유*
2. **view**
   - display 로직 호출시 전달된 view model을 활용해 view 설정 DisplayLogic 구현
3. **navigation 처리**
   - 현재 화면에서 다른 화면으로의 전환 로직 및 전환 시 필요한 데이터 전달 로직을 호출 *RoutingLogic 소유*

Interactor

비즈니스 로직의 구현체

- 비즈니스 로직 처리 필요한 데이터 처리 *BussinessLogic 구현*
- 로직 처리 후 Response 객체 생성해 presenting 로직 트리거 *PresentationLogic 소유*
- 역할 분담
  - 로직을 내부적으로 직접 처리하기도 하지만
  - 특정 로직을 처리하는 전문 일꾼(**Worker**) 고용
    - interator 클래스가 worker를 소유하는 형태
    - worker가 처리한 결과 데이터를 worker 메소드 호출시 전달한 completion handler에서 활용

Presenter

- 전달된 비즈니스 로직 수행 결과를 사용자에게 보여줄 형태로 만들어 view model 생성 *PresentationLogic 구현*
- 생성된 view model과 함께 display 로직 트리거 *DisplayLogic 소유 (weak)*
  - 이때 DisplayLogic 타입을 약하게 참조 (참조 사이클 방지)

Payload Model
감싸서 넘겨주자

- 사이클의 각 컴포넌트 경계에서 데이터를 주고 받을 때 사용하는 타입
  - V > I : Request
  - I > P : Response
  - P > V : ViewModel
- 특정 데이터 타입에 독립적으로 사용하기 위해 래핑해서 사용
- 네스팅을 통해 네임스페이싱
  - enum *SceneName* > enum *UseCaseName* > struct Request, Response, ViewModel
- 원시 타입 원칙: 원시 타입만 포함하는 것을 권장하지만 유연하게 사용할 수 있음

Worker
전문 일꾼

- 비즈니스 로직(Use case) 처리의 일부를 담당하며, interactor 역할을 분리하기 위해 사용됨
- worker를 사용하면 외부 모듈에 대한 의존성을 분리할 수 있음
- worker는 클래스로 구현되고, interactor는 구현체로 직접 소유
- Scene-specialized / Global
  - 특정한 scene에서만 쓰이는 로직도 있고,
  - 같은 다양한 곳에 쓰일 수 있으므로 공유할 수도 있음

Router

화면 전환 전문

- view controller 에서 navigation 로직을 분리한 것
- view controller 가 호출할 메소드 구현 RoutingLogic 구현
- 화면(scene) 간 전환 로직(navigation logic)과 화면 간 데이터 전달 로직(data passing)이 필요
- 인터페이스 호출시 segue를 전달 받음
  - optional -> 있을 때 없을 때 모두 대응해야 함
- 이 라우터를 사용할 source scene의 view controller 를 약하게 참조
- 필요에 따라 전달할, 전달 받은 데이터를 저장하기 위한 dataStore를 참조

클린 스위프트의 목표

어떻게 달성하려는 걸까

VIP가 어떻게 문제를 해결한다는 걸까

- 응집도 높이기 - 역할 쪼개기 (책임의 분리)
  - Scene 단위로 V, I, P, Router로 분리
  - use case 단위로 호출할 메소드 분리
  - 비즈니스 로직의 일부 역할을 Worker로 분리
- 의존성 낮추기
  - 구체 타입 대신 인터페이스(protocol)을 사용 (ex. VC이 Interactor 대신 BusinessLogic 메소드 호출)
  - 계층 간 데이터를 고정된 payload 모델을 사용해 전달