[YSK] Clean Swift 스터디

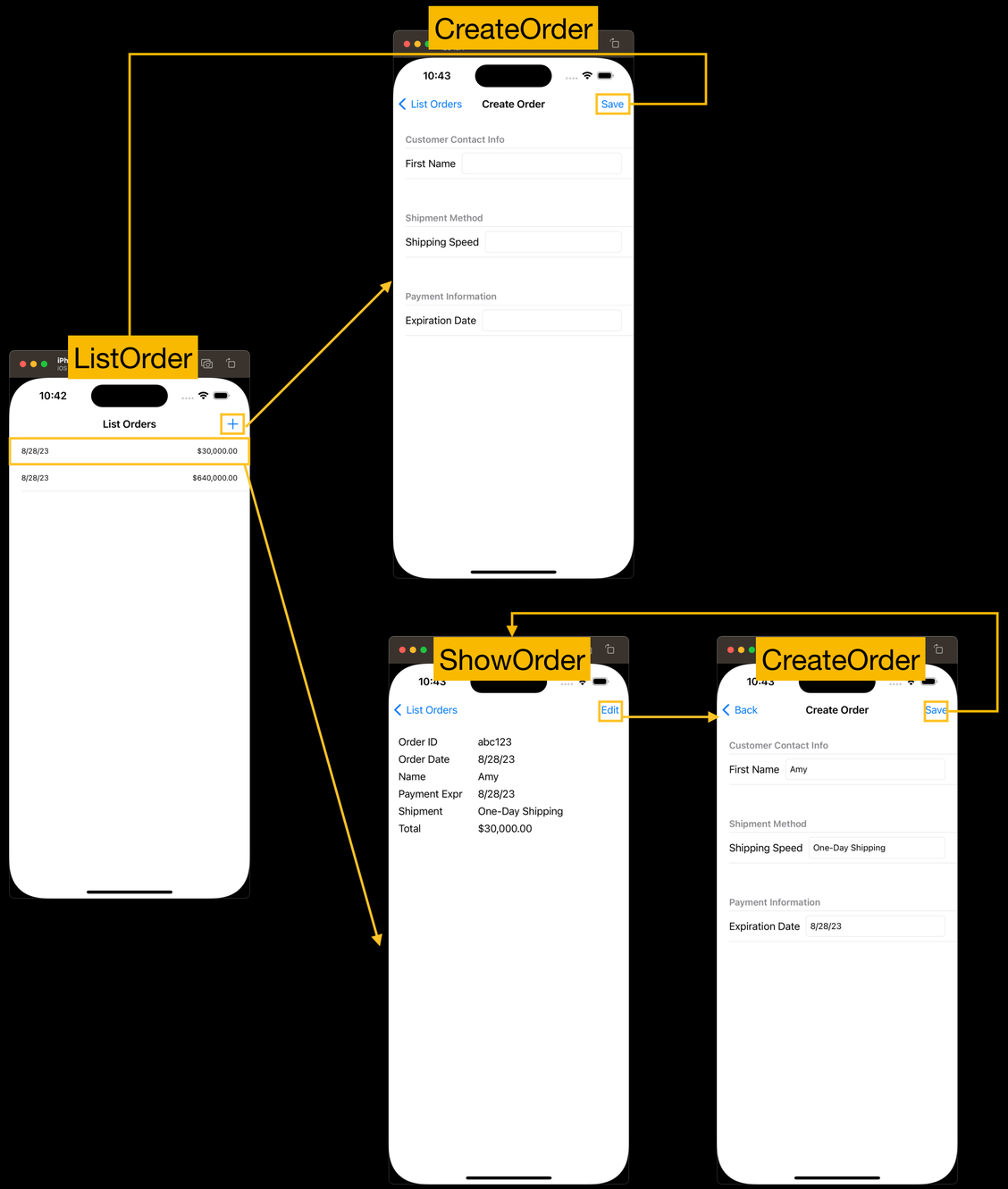
학습 내용 공유 2차

박혜정 (2023년 8월 28일)

오늘 공유의 순서

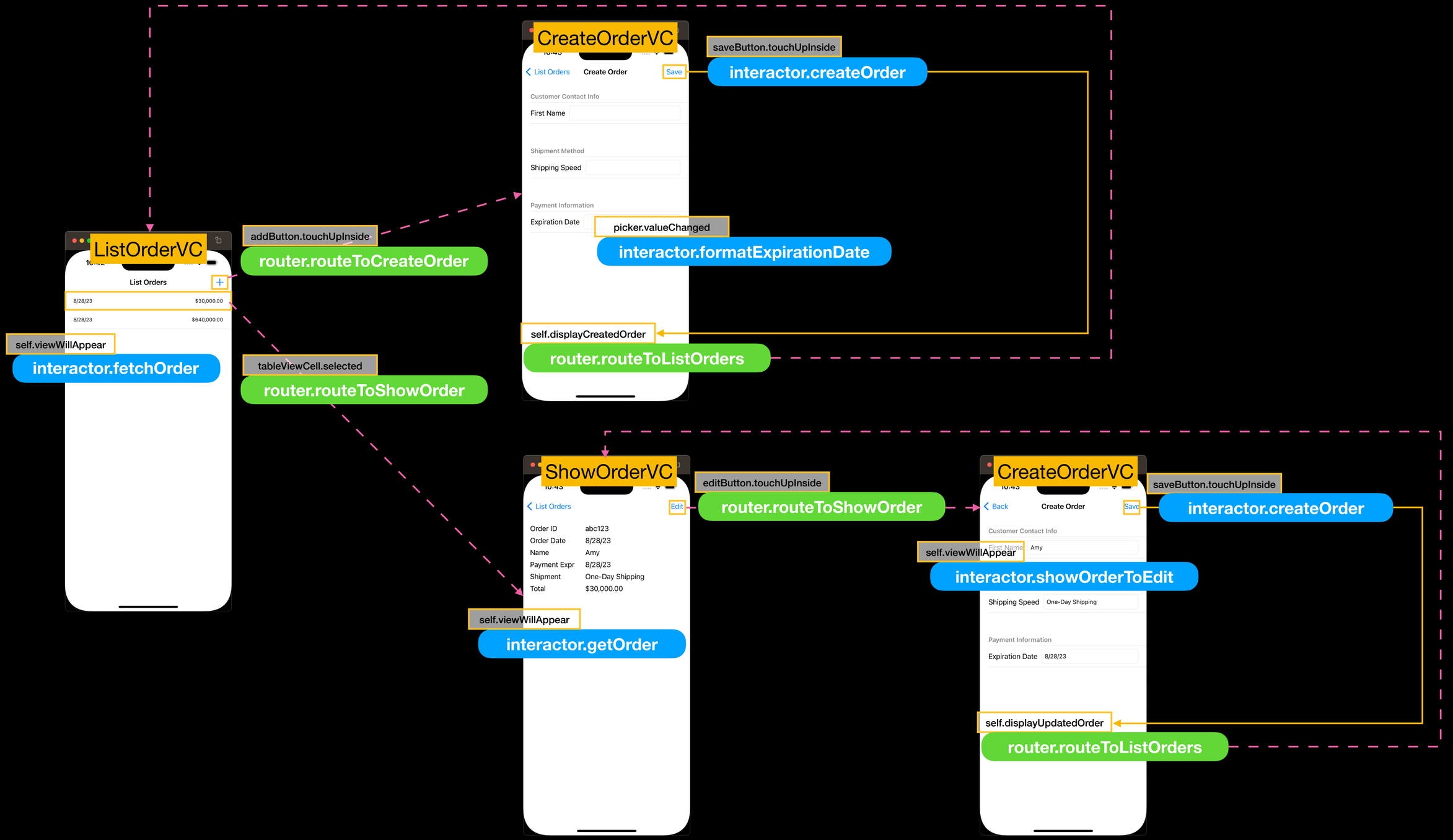
0. *CleanStore* Use Case 뒤져보기
1. VIP Cycle
2. Worker
3. Router

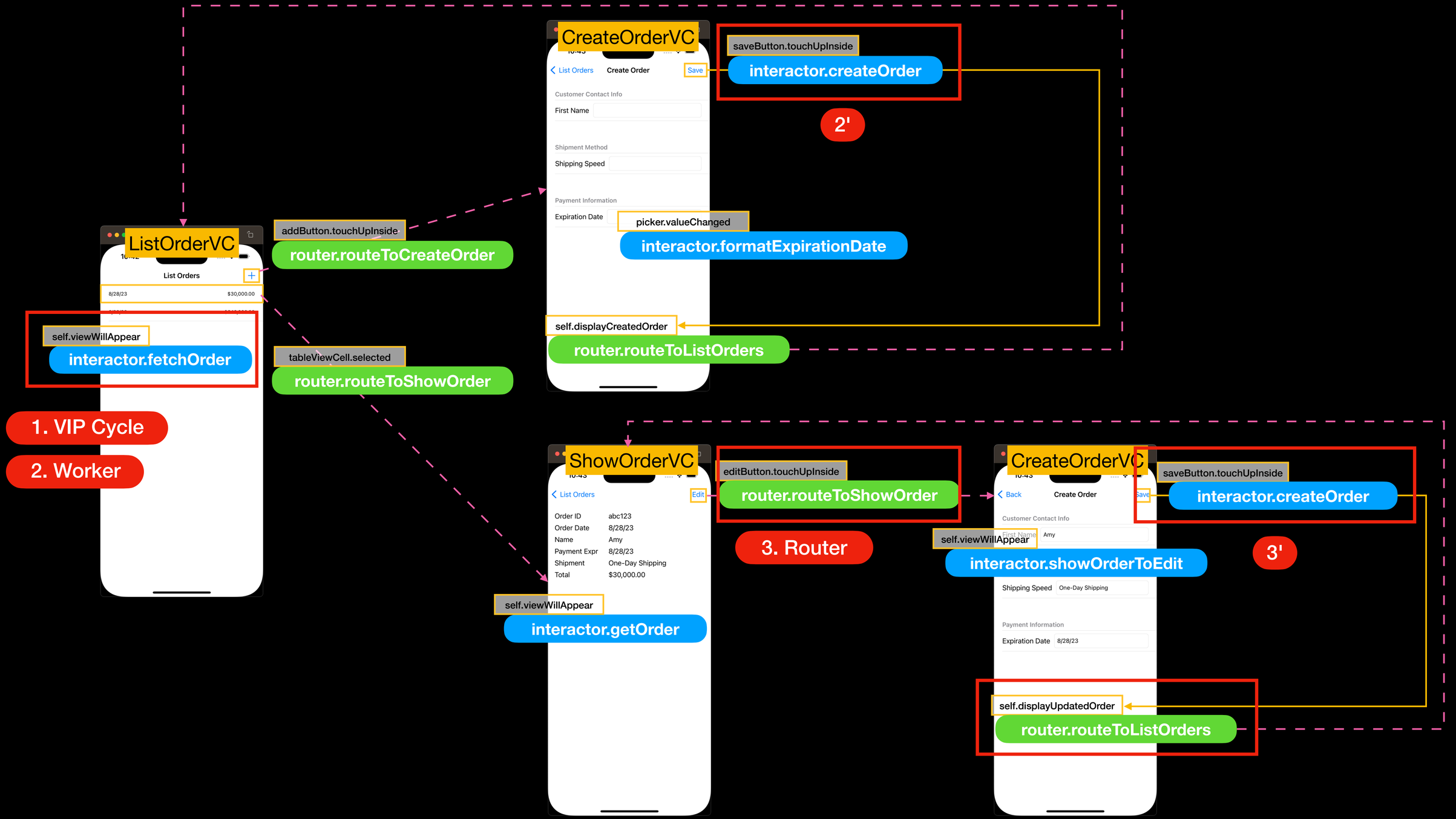
0. *CleanStore* Use Cases 뒤져보기


CreateOrder
10:43
List Orders
Create Order
Save
Customer Contact Info
First Name
Shipment Method
Shipping Speed
Payment Information
Expiration Date
ListOrder
10:42
List Orders
8/28/23
$30,000.00
8/28/23
$640,000.00
ShowOrder
List Orders
Edit
Order ID
abc123
Order Date
8/28/23
Name
Amy
Payment Expr
8/28/23
Shipment
One-Day Shipping
Total
$30,000.00
CreateOrder
Back
Create Order
Save
Customer Contact Info
First Name
Amy
Shipment Method
Shipping Speed
One-Day Shipping
Payment Information
Expiration Date
8/28/23

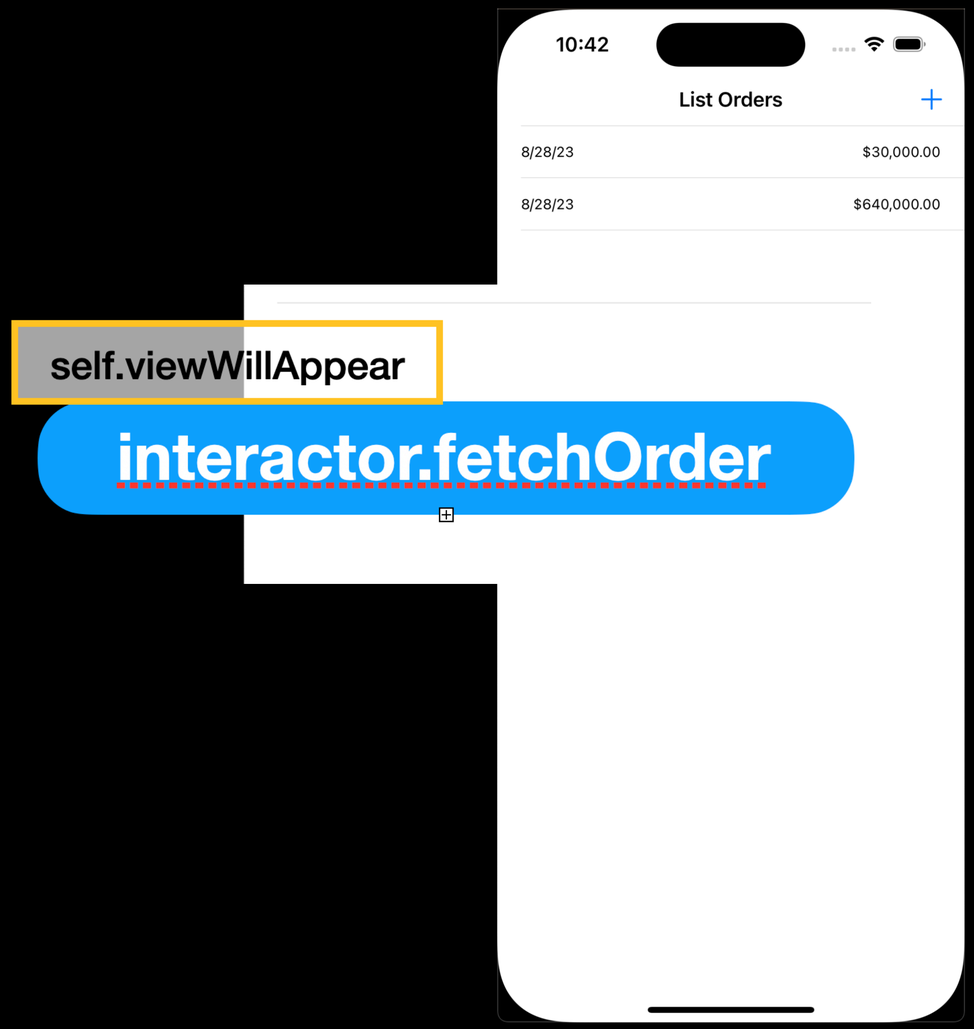
어떤 트리거는 BussinessLogic을,
어떤 트리거는 RoutingLogic을 호출한다

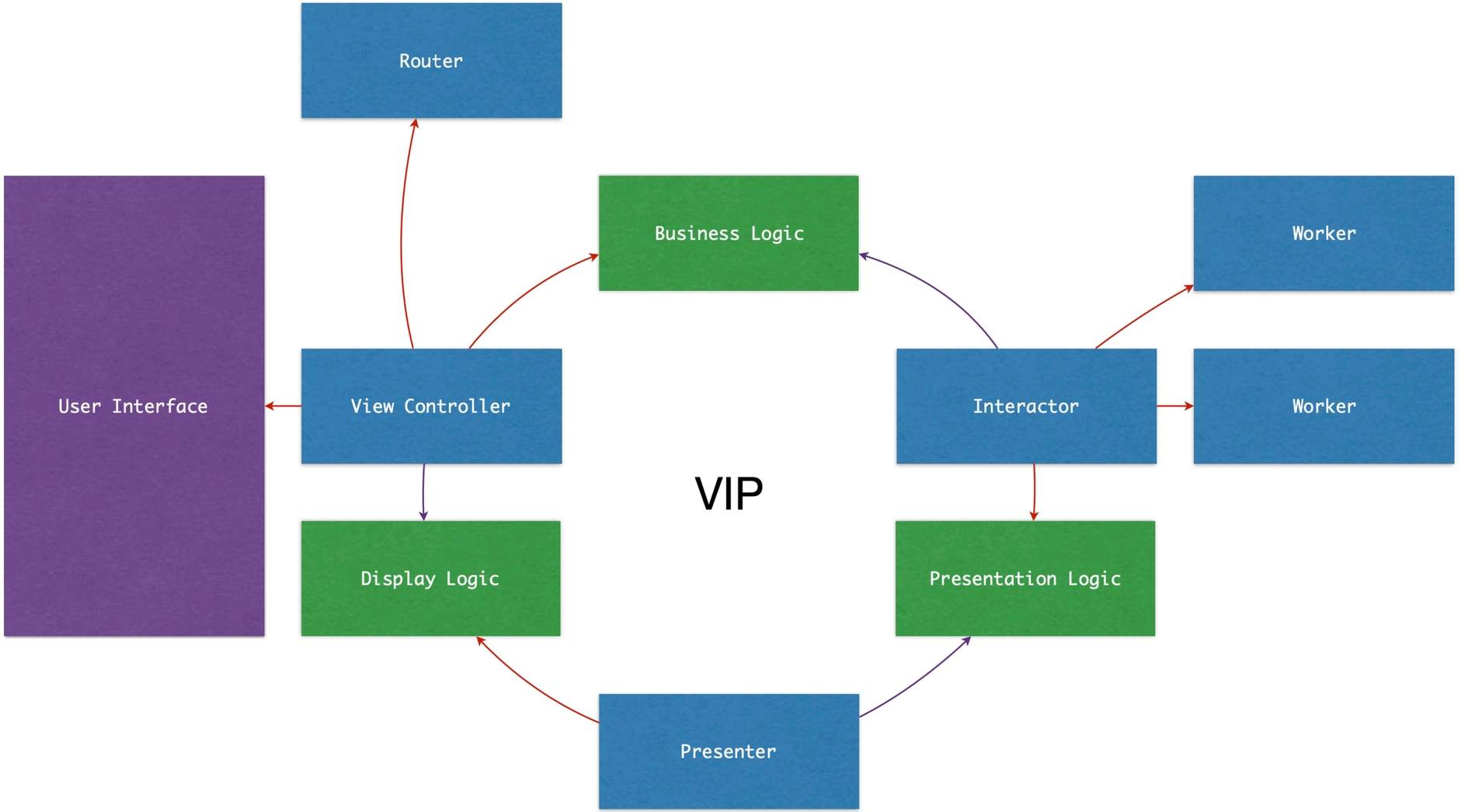
일반 use case와 마찬가지로 라우팅도 일종의 비지니스 로직.
다른 use case와 다르게 request, response, view model을 사용하지 않으며, router의 navigation으로 로직은 마무리 된다.

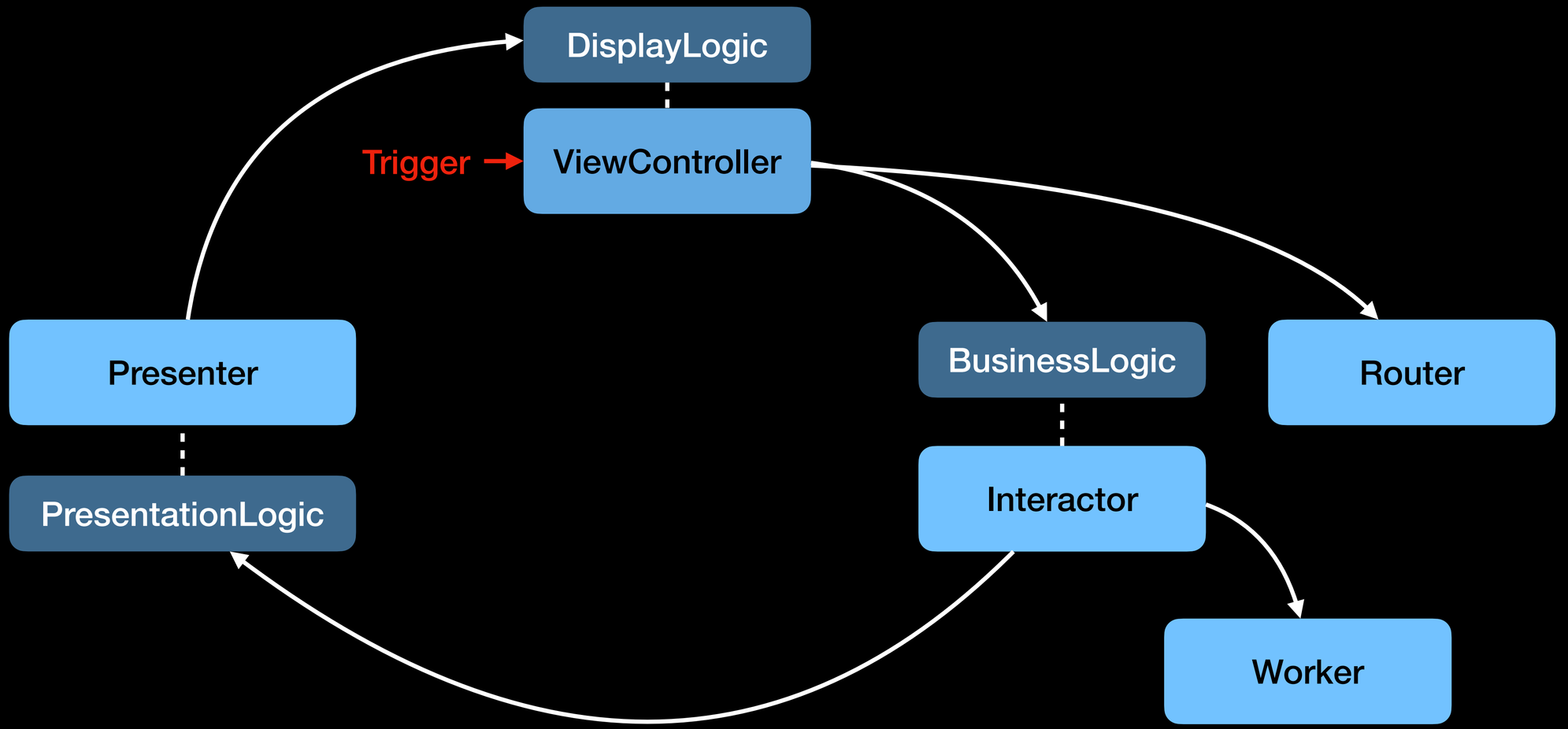

CreateOrderVC
saveButton.touchUpInside
interactor.createOrder
List Orders
Create Order
Save
Customer Contact Info
First Name
Shipment Method
Shipping Speed
Payment Information
Expiration Date
picker.valueChanged
interactor.formatExpirationDate
ListOrderVC
addButton.touchUpInside
router.routeToCreateOrder
List Orders
8/28/23
$30,000.00
8/28/23
$640,000.00
self.viewWillAppear
interactor.fetchOrder
tableViewCell.selected
router.routeToShowOrder
self.displayCreatedOrder
router.routeToListOrders
ShowOrderVC
editButton.touchUpInside
router.routeToShowOrder
CreateOrderVC
saveButton.touchUpInside
interactor.createOrder
List Orders
Edit
Order ID
abc123
Order Date
8/28/23
Name
Amy
Payment Expr
8/28/23
Shipment
One-Day Shipping
Total
$30,000.00
self.viewWillAppear
interactor.getOrder
Back
Create Order
Save
Customer Contact Info
First Name
Amy
self.viewWillAppear
interactor.showOrderToEdit
Shipping Speed
One-Day Shipping
Payment Information
Expiration Date
8/28/23
self.displayUpdatedOrder
router.routeToListOrders


ListOrderVC
List Orders
8/28/23
$30,000.00
addButton.touchUpInside
router.routeToCreateOrder
tableViewCell.selected
router.routeToShowOrder
self.viewWillAppear
interactor.fetchOrder
1. VIP Cycle
2. Worker
CreateOrderVC
List Orders
Create Order
Save
Customer Contact Info
First Name
Shipment Method
Shipping Speed
Payment Information
Expiration Date
saveButton.touchUpInside
interactor.createOrder
2'
picker.valueChanged
interactor.formatExpirationDate
self.displayCreatedOrder
router.routeToListOrders
ShowOrderVC
List Orders
Edit
Order ID abc123
Order Date 8/28/23
Name Amy
Payment Expr 8/28/23
Shipment One-Day Shipping
Total $30,000.00
self.viewWillAppear
interactor.getOrder
editButton.touchUpInside
router.routeToShowOrder
3. Router
CreateOrderVC
Back
Create Order
Save
Customer Contact Info
First Name
Amy
self.viewWillAppear
interactor.showOrderToEdit
Shipping Speed
One-Day Shipping
Payment Information
Expiration Date
8/28/23
saveButton.touchUpInside
interactor.createOrder
3'
self.displayUpdatedOrder
router.routeToListOrders

1. VIP Cycle


10:42
List Orders
8/28/23
$30,000.00
8/28/23
$640,000.00
self.viewWillAppear
interactor.fetchOrder


Router
Business Logic
Worker
User Interface
View Controller
Interactor
Worker
VIP
Display Logic
Presentation Logic
Presenter
Class
Protocol
Has A
Implements


DisplayLogic
Trigger
ViewController
Presenter
BusinessLogic
Router
PresentationLogic
Interactor
Worker

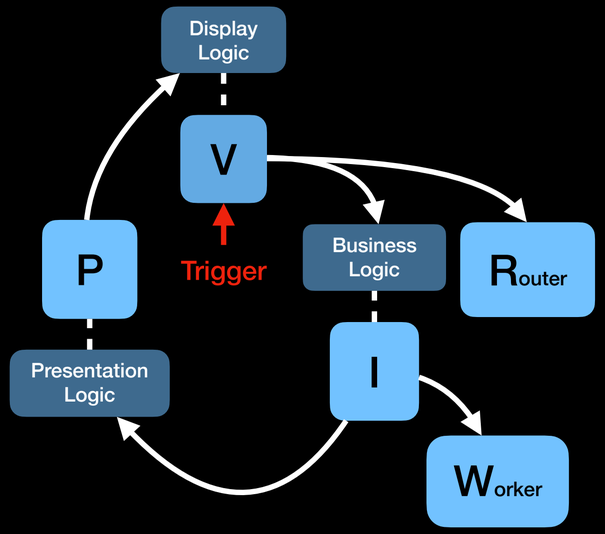
View Controller

이벤트 컨트롤 타워

1. **controller**
   - 사용자 입력(UI event) 또는 view lifecycle 등 트리거 처리
   - 알맞은 request를 생성해서 알맞은 비즈니스 로직을 트리거 *BussinessLogic 소유**
2. **view**
   - display 로직 호출시 전달된 view model을 활용해 view 설정
3. **navigation 처리**
   - 화면 전환 트리거 발생 시 필요한 Routing 로직을 호출
4. **component 설정**
   - Interactor, Presenter, Router 설정

*참조를 소유

View Controller
컴포넌트 설정

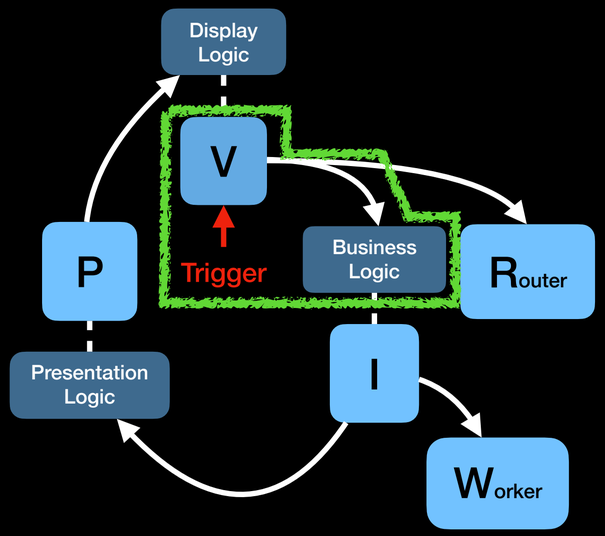


- VIP cycle의 각 컴포넌트 설정
  - 생성 시점에 설정 (init)
  - ViewController > Interactor, Router > Presenter > ViewController 차례대로 설정

```
class ListOrderViewController {
    ...
  var interactor: ListOrderBusinessLogic?
  var router: (NSObjectProtocol & ListOrderRoutingLogic & ListOrderDataPassing)?

  override init(nibName nibNameOrNil: String?, bundle nibBundleOrNil: Bundle?)
  {
    super.init(nibName: nibNameOrNil, bundle: nibBundleOrNil)
    setup()
  }

  private func setup()
  {
    let viewController = self
    let interactor = ListOrderInteractor()
    let presenter = ListOrderPresenter()
    let router = ListOrderRouter()
    viewController.interactor = interactor
    viewController.router = router
    interactor.presenter = presenter
    presenter.viewController = viewController
    router.viewController = viewController
    router.dataStore = interactor
  }
    ...
}
```

View Controller

trigger 대응



- controller 로서 view lifecycle 대응
  - viewWillAppear 구현
- BusinessLogic 호출
  - interactor.fetchOrders 호출

```
class ListOrderViewController {
  ...

  var interactor: ListOrderBusinessLogic?

  override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    fetchOrdersOnLoad()
  }

  func fetchOrdersOnLoad() {
    let request = ListOrder.FetchOrders.Request()
    interactor?.fetchOrders(request: request)
  }
}
```

trigger는 viewWillAppear

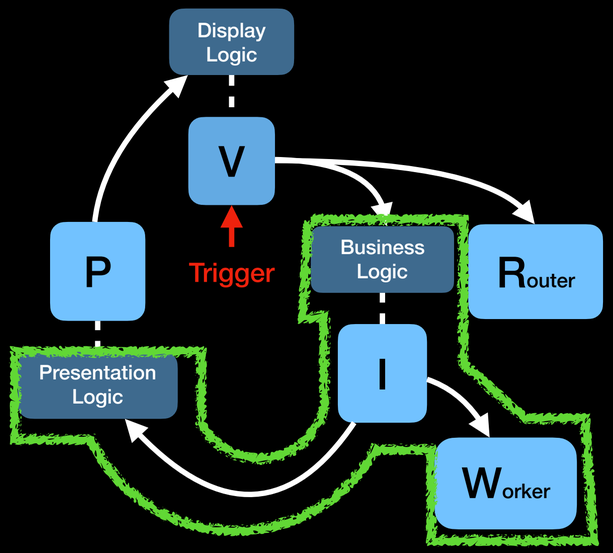
interactor를 사용하는 경우에는 이렇게 UseCase 별로 모델을 구조화해서 사용

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

Interactor
비즈니스 로직 처리



- 비즈니스 로직을 일꾼에게 맡김
  - worker의 fetchOrders 호출
- PresentationLogic 호출
  - worker 호출 시 비동기로 처리할 코드 전달 하며 presenter의 presentFetchedOrders 호출

```
protocol ListOrderBusinessLogic
{
  func fetchOrders(request: ListOrder.FetchOrders.Request)
}

class ListOrderInteractor: ListOrderBusinessLogic
{
  var ordersWorker = OrdersWorker(ordersStore: OrdersMemStore())
  var presenter: ListOrderPresentationLogic?

  func fetchOrders(request: ListOrder.FetchOrders.Request)
  {
    ordersWorker.fetchOrders { [weak self] (orders) -> Void in
      guard let self else { return }
      self.orders = orders
      let response = ListOrder.FetchOrders.Response(orders: orders)
      self.presenter?.presentFetchedOrders(response: response)
    }
  }
}
```

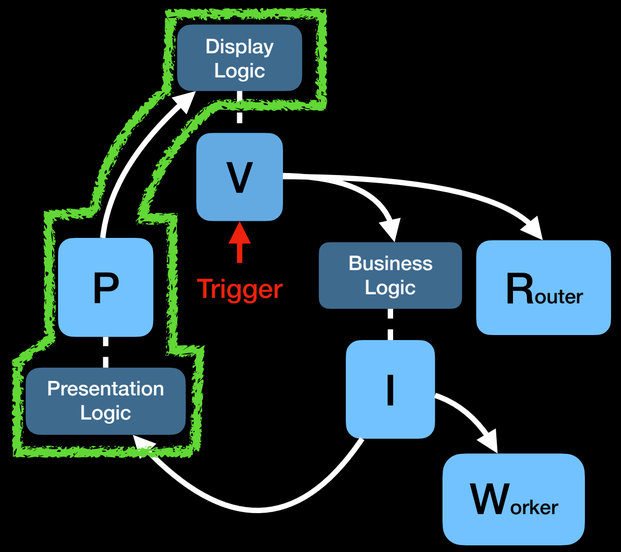
interactor □=

Presenter

데이터 형식 변환

- 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)

Presenter
데이터 형식 변환



- 비즈니스 로직 수행 결과를 전달받아 가공 후 ViewModel로 래핑
- DisplayLogic 호출
  - DisplayLogic의 displayFetchedOrders 호출

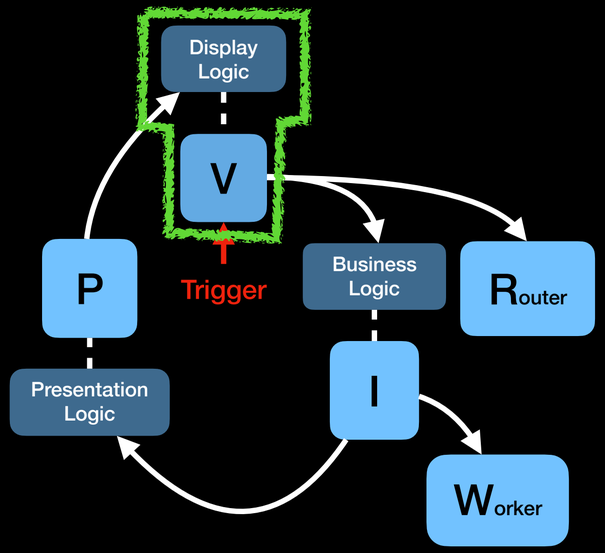
```
protocol ListOrderPresentationLogic
{
  func presentFetchedOrders(response: ListOrder.FetchOrders.Response)
}

class ListOrderPresenter: ListOrderPresentationLogic
{
  weak var viewController: ListOrderDisplayLogic?

  let dateFormatter: DateFormatter = { ...  }()

  let currencyFormatter: NumberFormatter = { ...  }()

  func presentFetchedOrders(response: ListOrder.FetchOrders.Response) {
    var displayedOrders: [ListOrder.FetchOrders.ViewModel.DisplayedOrder] = []
    for order in response.orders {
      let date = dateFormatter.string(from: order.date)
      ...
      let displayedOrder = ListOrder.FetchOrders.ViewModel.DisplayedOrder( ... )
      displayedOrders.append(displayedOrder)
    }
    let viewModel = ListOrder.FetchOrders.ViewModel(displayedOrders: displayedOrders)
    viewController?.displayFetchedOrders(viewModel: viewModel)
  }
}
```

View Controller

이벤트 컨트롤 타워

1. **controller**
   - 사용자 입력(UI event) 또는 view lifecycle 등 트리거 처리
   - 알맞은 request를 생성해서 알맞은 비즈니스 로직을 트리거
2. **view**
   - display 로직 호출시 전달된 view model을 활용해 view 설정 DisplayLogic 구현
3. **navigation 처리**
   - 화면 전환 트리거 발생 시 필요한 Routing 로직을 호출
4. **component 설정**
   - Interactor, Presenter, Router 설정

View Controller
화면 표시 로직 처리



- 전달받은 ViewModel를 활용해 화면에 데이터 표시
  - DisplayLogic 구현

```
protocol ListOrderDisplayLogic: AnyObject
{
  func displayFetchedOrders(viewModel: ListOrder.FetchOrders.ViewModel)
}

class ListOrderViewController: UITableViewController, ListOrderDisplayLogic
{
   ...
  var displayedOrders: [ListOrder.FetchOrders.ViewModel.DisplayedOrder] = []

  func displayFetchedOrders(viewModel: ListOrder.FetchOrders.ViewModel) {
    displayedOrders = viewModel.displayedOrders
    tableView.reloadData()
  }
   ...
}
```

display 로직 구현에서 전달된 ViewModel 을 바로 사용할 수도 있겠지만,
이 결우에는 ViewModel의 하위 타입인 DisplayedOrder 의 배열을 ViewController가 저장하고,
view controller가 데이터를 표시하는 데에 사용

ListOrder.FetchOrder

VIP Cycle

ListOrderPresenter

```
func presentFetchedOrders(response: ListOrder.FetchOrders.Response) {
  var displayedOrders: [ListOrder.FetchOrders.ViewModel.DisplayedOrder] = []
  for order in response.orders {
      let paymentExprDate = dateFormatter.string(from: order.paymentMethod.expirationDate)
      let date = dateFormatter.string(from: order.date)
      let total = currencyFormatter.string(from: order.total)
    let displayedOrder = ListOrder.FetchOrders.ViewModel.DisplayedOrder(
        id: order.id,
        date: date,
        name: "\(order.firstName)",
        total: total!,
        payment: paymentExprDate,
        shipment: order.shipmentMethod.toString()
    )
    displayedOrders.append(displayedOrder)
  }
  let viewModel = ListOrder.FetchOrders.ViewModel(displayedOrders: displayedOrders)
  viewController?.displayFetchedOrders(viewModel: viewModel)
}
```

ListOrderDisplayLogic

```
func displayFetchedOrders(viewModel: ListOrder.FetchOrders.ViewModel)
```

ListOrderViewController

```
override func viewWillAppear(_ animated: Bool) {
  super.viewWillAppear(animated)
  fetchOrdersOnLoad()
}

func fetchOrdersOnLoad() {
  let request = ListOrder.FetchOrders.Request()
  interactor?.fetchOrders(request: request)
}
```

```
func displayFetchedOrders(viewModel: ListOrder.FetchOrders.ViewModel) {
  displayedOrders = viewModel.displayedOrders
  tableView.reloadData()
}
```

ListOrderPresentationLogic

```
func presentFetchedOrders(response: ListOrder.FetchOrders.Response)
```

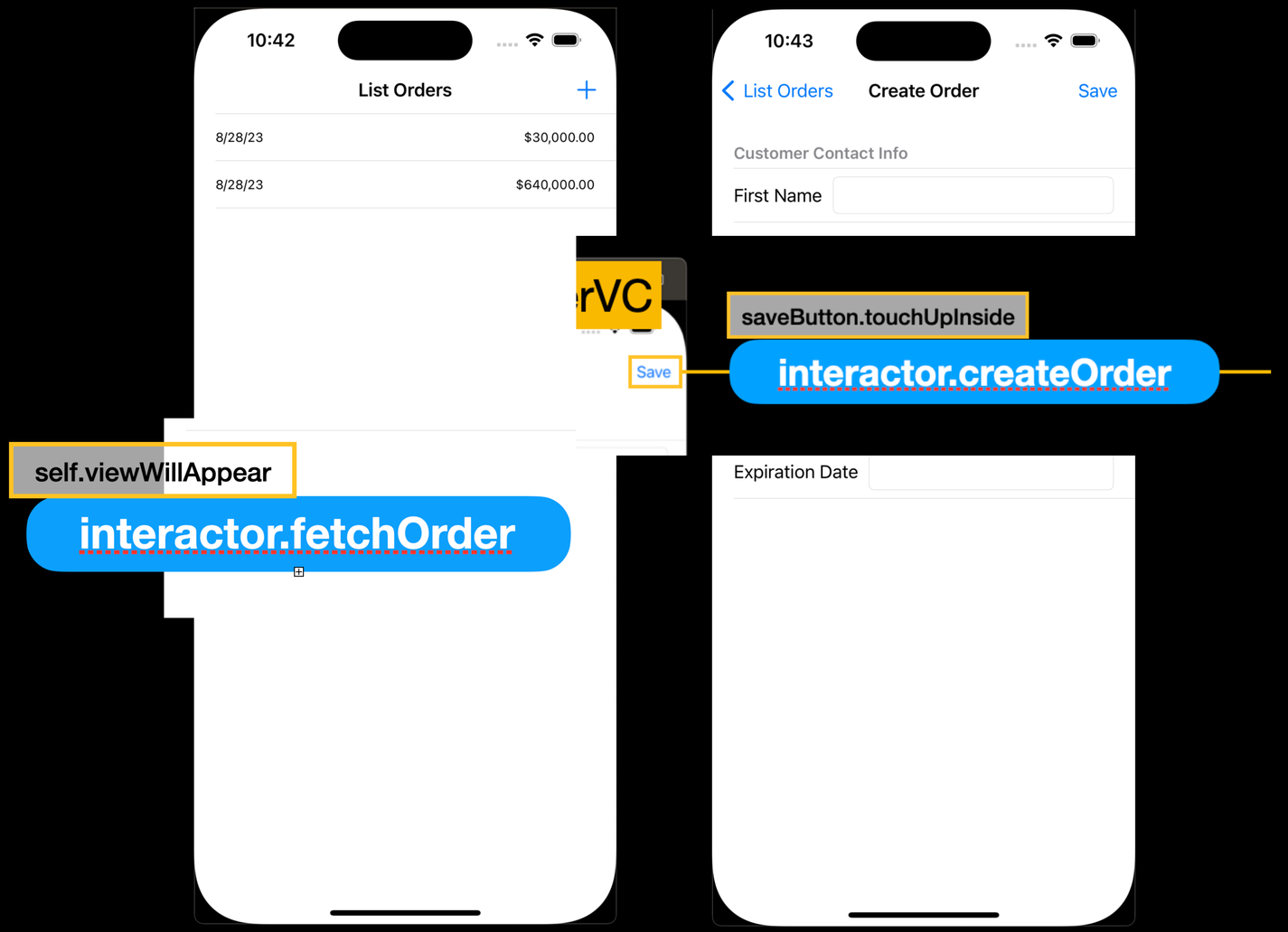
ListOrderInteractor

```
func fetchOrders(request: ListOrder.FetchOrders.Request)
  {
    ordersWorker.fetchOrders { (orders) -> Void in
      self.orders = orders
      let response = ListOrder.FetchOrders.Response(orders: orders)
      self.presenter?.presentFetchedOrders(response: response)
    }
  }
```

ListOrderBusinessLogic

```
func fetchOrders(request: ListOrder.FetchOrders.Request)
```

2. Worker

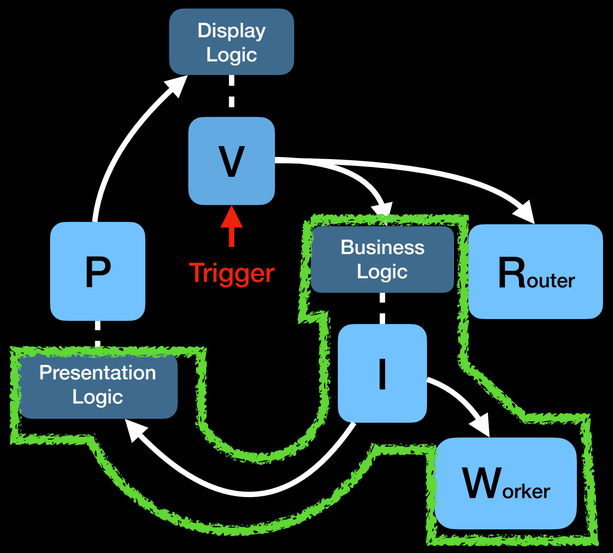

10:42
List Orders
8/28/23
$30,000.00
8/28/23
$640,000.00
self.viewWillAppear
interactor.fetchOrder
rVC
Save
10:43
List Orders
Create Order
Save
Customer Contact Info
First Name
saveButton.touchUpInside
interactor.createOrder
Expiration Date

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

Interactor
비즈니스 로직 처리 및



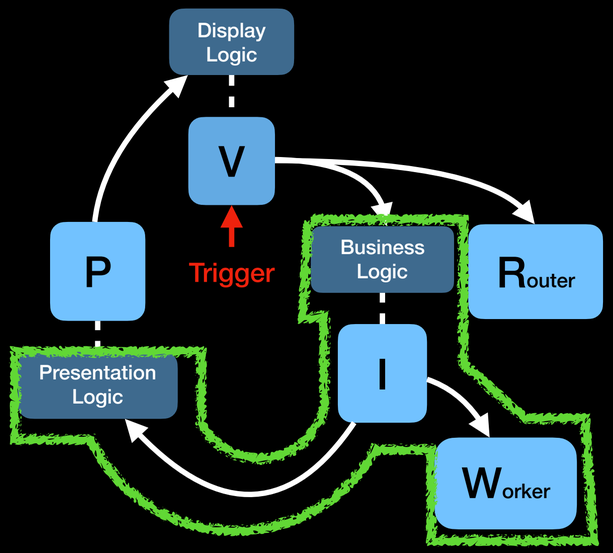
- 비즈니스 로직의 일부를 대리할 worker를 직접 참조 (구체 타입으로 참조)
- BusinessLogic 인터페이스가 호출되면 worker의 fetchOrders 호출
- worker 호출 시 비동기로 처리할 코드(PresentationLogic)를 클로저로 전달

```
protocol ListOrderBusinessLogic
{
  func fetchOrders(request: ListOrder.FetchOrders.Request)
}

class ListOrderInteractor: ListOrderBusinessLogic
{
  var ordersWorker = OrdersWorker(ordersStore: OrdersMemStore())
  var presenter: ListOrderPresentationLogic?

  func fetchOrders(request: ListOrder.FetchOrders.Request)
  {
    ordersWorker.fetchOrders { [weak self] (orders) -> Void in
      guard let self else { return }
      self.orders = orders
      let response = ListOrder.FetchOrders.Response(orders: orders)
      self.presenter?.presentFetchedOrders(response: response)
    }
  }
}
```

Worker
전문 일꾼

- 비즈니스 로직(Use case) 처리의 일부를 담당하며, interactor의 역할 일부를 분리하기 위해 사용됨
  - worker를 사용하면 외부 모듈에 대한 의존성을 분리가 용이 해짐
- worker는 클래스로 구현되고, interactor는 worker를 직접 참조
- Scene-specialized / Global
  - 특정한 scene에서만 쓰이는 로직도 있고,
  - 다양한 scene에 쓰일 수 있으므로 공유할 수도 있음

Worker
비즈니스 로직의 분리



- 비즈니스 로직을 일부를 interactor 대신 처리
- worker가 처리한 로직의 결과를 사용할 코드를 파라미터로 전달받음 (비동기로 처리)
- 여러 scene 에서 공통적으로 사용하는 로직은 전용 worker를 사용 (ListOrderWorker -> OrdersWorker)

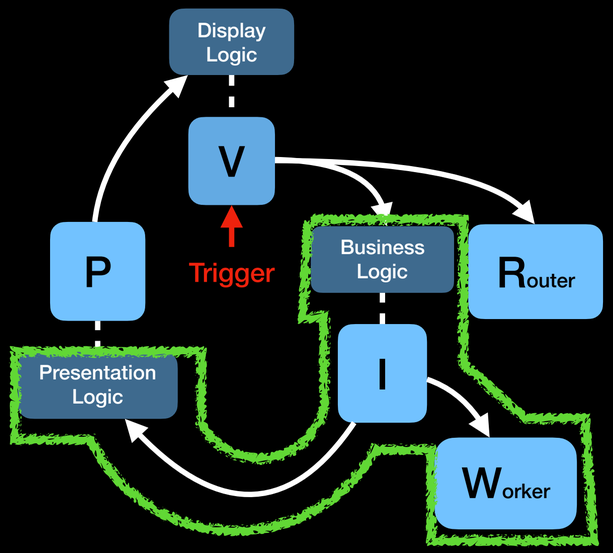
```
class OrdersWorker
{
  var ordersStore: OrdersStoreProtocol

  init(ordersStore: OrdersStoreProtocol) {
    self.ordersStore = ordersStore
  }

  func fetchOrders(completionHandler: @escaping ([Order]) -> Void) {
    ordersStore.fetchOrders { (ordersGiver: () throws -> [Order]) -> Void in
      let orders: [Order] = (try? ordersGiver()) ?? []
      DispatchQueue.main.async {
        completionHandler(orders)
      }
    }
  }
}
```

worker를 호출할 때 Order 배열을 사용할 코드를 전달.
이때 전달한 코드가 비동기로 실행되도록 구현.

호출 코드 보기

StoreProtocol

entity 관리



- worker 가 사용할 entity를 직접 가지지 않음
- entity를 직접 저장하고 관리하는 객체에 인터페이스를 통해 접근하는 방식으로 사용
- 여기서는 worker와 마찬가지로 결과를 비동기로 처리하는 코드를 전달받는 패턴

```
protocol OrdersStoreProtocol
{
  func fetchOrders(completionHandler: @escaping (() throws -> [Order]) -> Void)

  func createOrder(orderToCreate: Order, completionHandler: @escaping (() throws -> Order?) -> Void)

  func updateOrder(orderToUpdate: Order, completionHandler: @escaping (() throws -> Order?) -> Void)
}
```

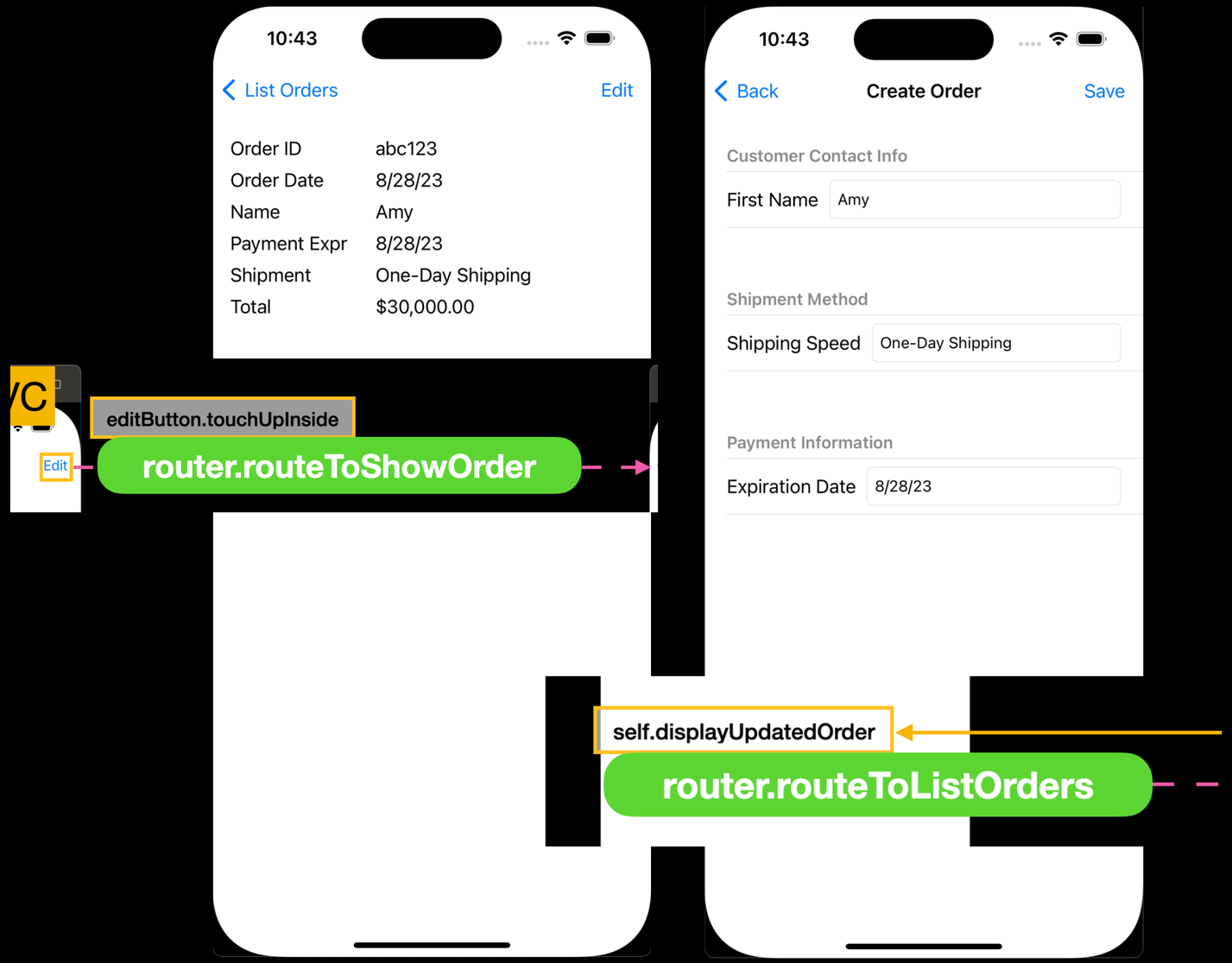
CreateOrder.CreateOrder

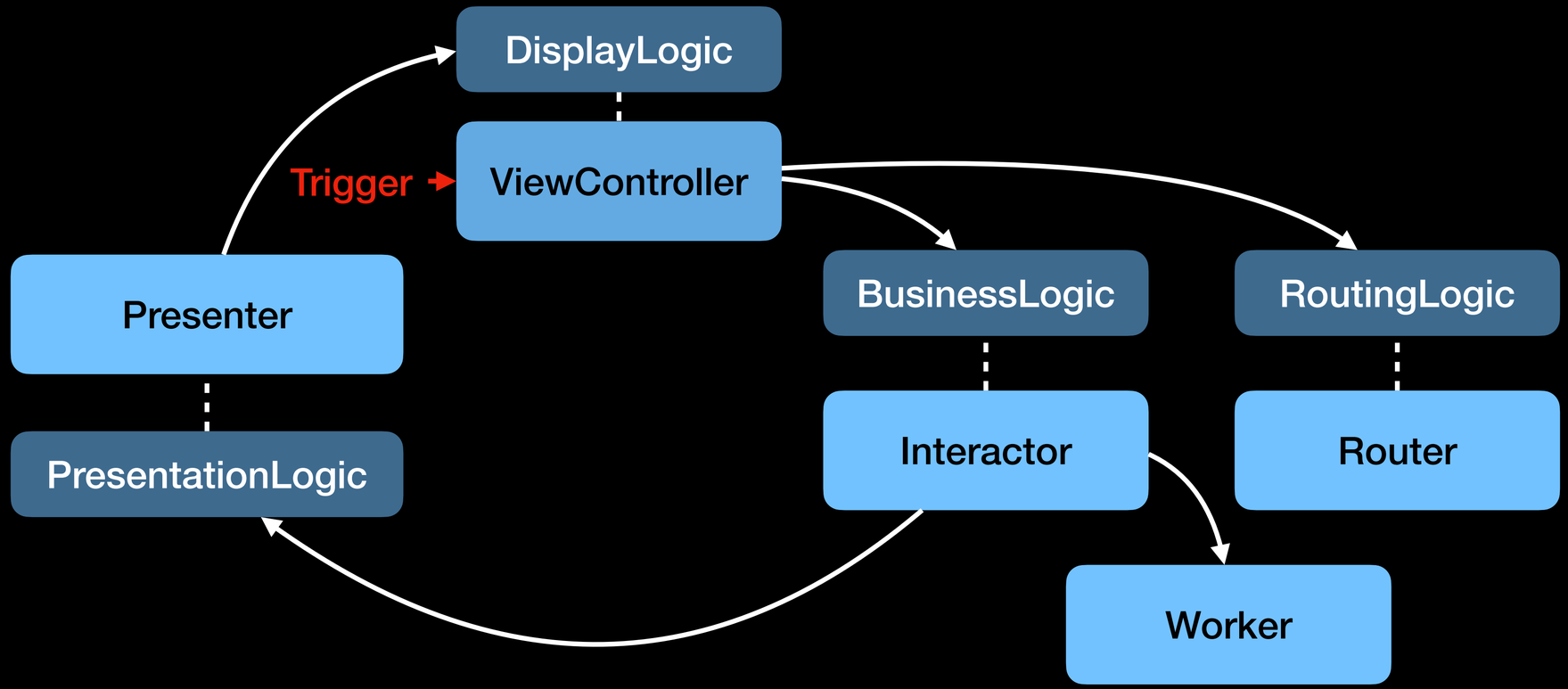
```swift
class CreateOrderInteractor: CreateOrderBusinessLogic, CreateOrderDataStore
{
  var order: Order?
  var ordersWorker = OrdersWorker(ordersStore: OrdersMemStore())
  var presenter: CreateOrderPresentationLogic?

  private func buildOrderFromOrderFields(_ orderFormFields: CreateOrder.OrderFormFields) -> Order {
    ...
    return Order( ... )
  }

  func createOrder(request: CreateOrder.CreateOrder.Request) {
    let orderToCreate = buildOrderFromOrderFields(request.orderFormFields)
    ordersWorker.createOrder(orderToCreate: orderToCreate) { [weak self] (order: Order?) in
      self?.order = order
      let response = CreateOrder.CreateOrder.Response(order: order)
      self?.presenter?.presentCreatedOrder(response: response)
    }
  }

  ...
}
```

interactor 는 전달된 request로부터 Entity를 생성하고,
결과(새 주문)를 활용할 비동기 코드와 함께 entity를 worker에게 전달.
이때 비동기 코드 PresentationLogic 호출
결과(새 주문)를 생성한 worker 는 함께 전달된 클로저 실행

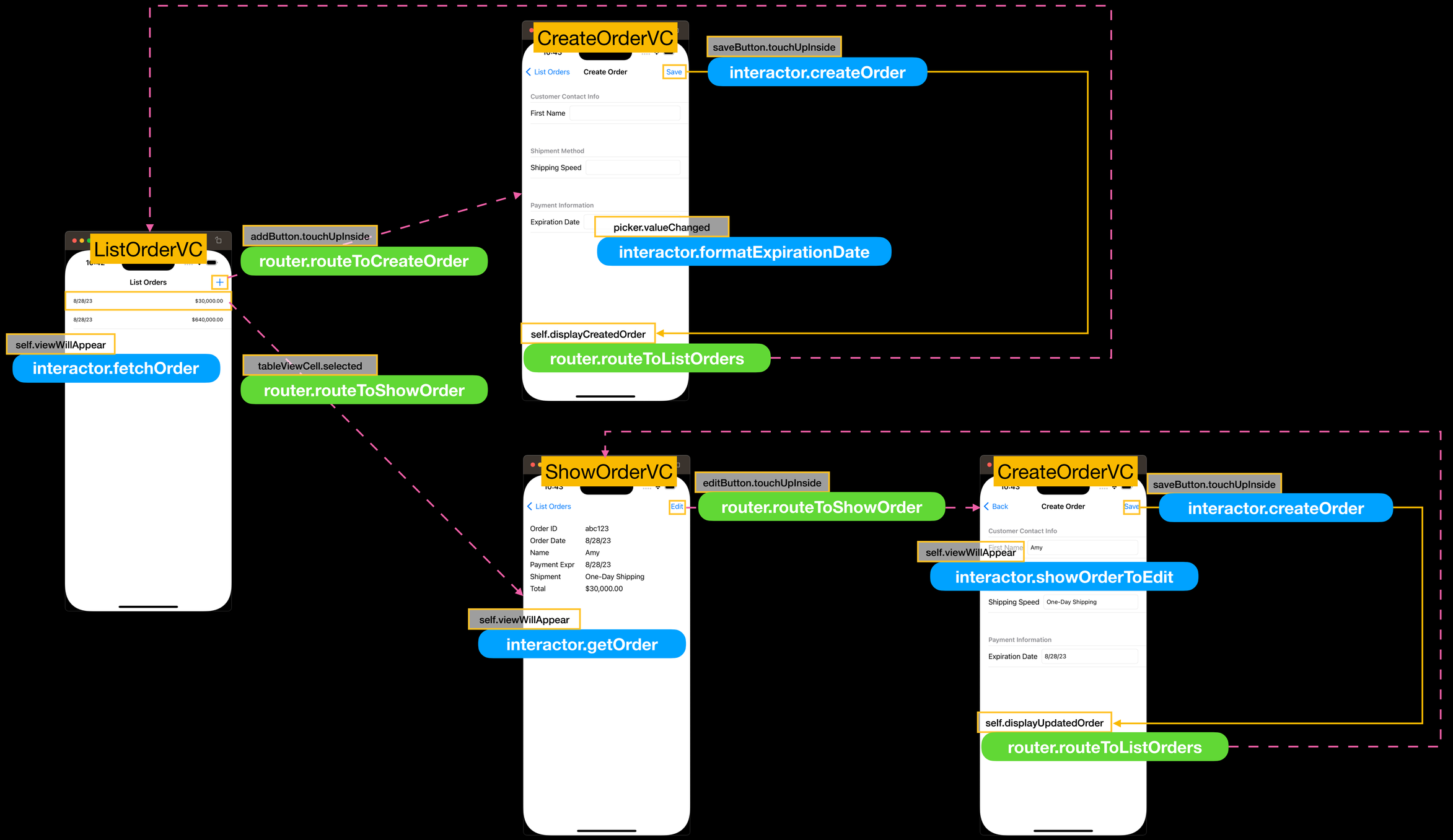
```swift
class OrdersWorker
{
  var ordersStore: OrdersStoreProtocol

  func createOrder(orderToCreate: Order, completionHandler: @escaping (Order?) -> Void) {
    ordersStore.createOrder(orderToCreate: orderToCreate) { (orderGiver: OrderGiver) -> Void in
      let order = try? orderGiver()
      DispatchQueue.main.async { completionHandler(order) }
    }
  }
  ...
}
```

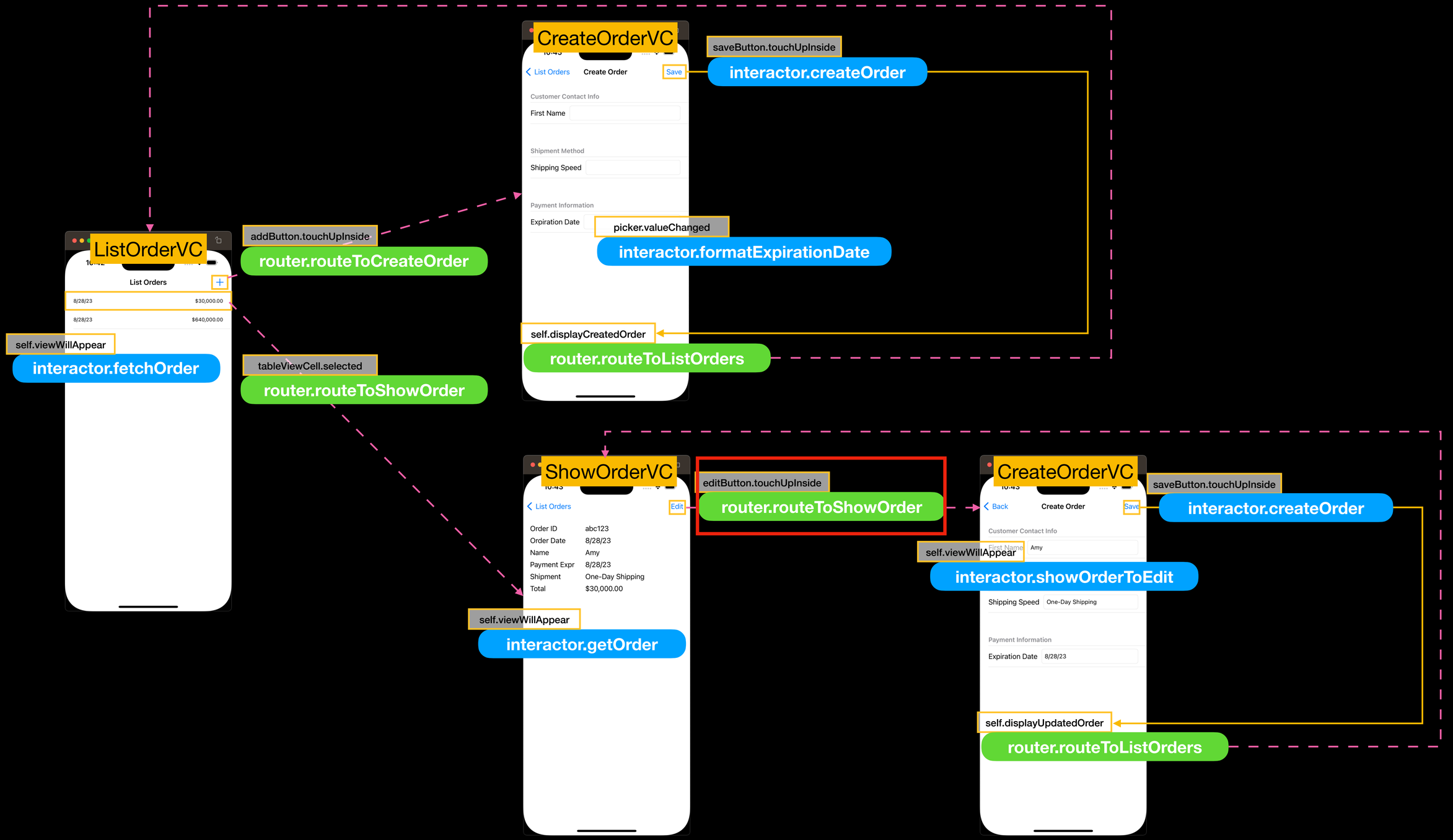
3. Router


10:43
List Orders
Edit
Order ID abc123
Order Date 8/28/23
Name Amy
Payment Expr 8/28/23
Shipment One-Day Shipping
Total $30,000.00
Edit
editButton.touchUpInside
router.routeToShowOrder
10:43
Back
Create Order
Save
Customer Contact Info
First Name Amy
Shipment Method
Shipping Speed One-Day Shipping
Payment Information
Expiration Date 8/28/23
self.displayUpdatedOrder
router.routeToListOrders



일반 use case와 마찬가지로 라우팅도 일종의 비지니스 로직.
다른 use case와 다르게 request, response, view model을 사용하지 않으며, router의 navigation으로 로직은 마무리 된다.


CreateOrderVC
saveButton.touchUpInside
interactor.createOrder
List Orders
Create Order
Save
Customer Contact Info
First Name
Shipment Method
Shipping Speed
Payment Information
Expiration Date
picker.valueChanged
interactor.formatExpirationDate
ListOrderVC
addButton.touchUpInside
router.routeToCreateOrder
List Orders
8/28/23
$30,000.00
8/28/23
$640,000.00
self.viewWillAppear
interactor.fetchOrder
tableViewCell.selected
router.routeToShowOrder
self.displayCreatedOrder
router.routeToListOrders
ShowOrderVC
editButton.touchUpInside
router.routeToShowOrder
CreateOrderVC
saveButton.touchUpInside
interactor.createOrder
List Orders
Edit
Order ID
abc123
Order Date
8/28/23
Name
Amy
Payment Expr
8/28/23
Shipment
One-Day Shipping
Total
$30,000.00
self.viewWillAppear
interactor.getOrder
Back
Create Order
Save
Customer Contact Info
First Name
Amy
self.viewWillAppear
interactor.showOrderToEdit
Shipping Speed
One-Day Shipping
Payment Information
Expiration Date
8/28/23
self.displayUpdatedOrder
router.routeToListOrders


CreateOrderVC
saveButton.touchUpInside
interactor.createOrder
List Orders
Create Order
Save
Customer Contact Info
First Name
Shipment Method
Shipping Speed
Payment Information
Expiration Date
picker.valueChanged
interactor.formatExpirationDate
self.displayCreatedOrder
router.routeToListOrders
ListOrderVC
addButton.touchUpInside
router.routeToCreateOrder
List Orders
8/28/23
$30,000.00
8/28/23
$640,000.00
self.viewWillAppear
interactor.fetchOrder
tableViewCell.selected
router.routeToShowOrder
ShowOrderVC
editButton.touchUpInside
router.routeToShowOrder
List Orders
Edit
Order ID
abc123
Order Date
8/28/23
Name
Amy
Payment Expr
8/28/23
Shipment
One-Day Shipping
Total
$30,000.00
self.viewWillAppear
interactor.getOrder
CreateOrderVC
saveButton.touchUpInside
interactor.createOrder
Back
Create Order
Save
Customer Contact Info
First Name
Amy
self.viewWillAppear
interactor.showOrderToEdit
Shipping Speed
One-Day Shipping
Payment Information
Expiration Date
8/28/23
self.displayUpdatedOrder
router.routeToListOrders

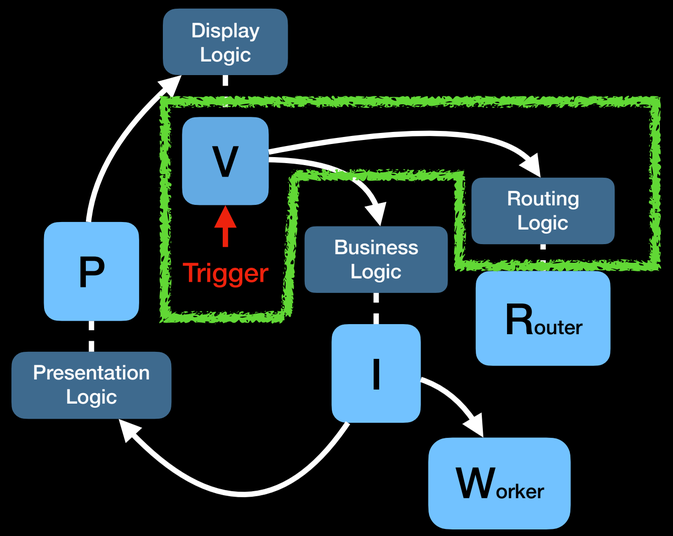
View Controller

이벤트 컨트롤 타워

1. **controller**
   - 사용자 입력(UI event) 또는 view lifecycle 등 트리거 처리
   - 알맞은 request를 생성해서 알맞은 비즈니스 로직을 트리거
2. **view**
   - display 로직 호출시 전달된 view model을 활용해 view 설정
3. **navigation 처리**
   - 화면 전환 트리거 발생 시 필요한 Routing 로직을 호출 *RoutingLogic 소유*
4. **component 설정**
   - Interactor, Presenter, Router 설정

ViewController

화면 전환 트리거 처리




```swift
class ShowOrderViewController: UIViewController
{
  var interactor: ShowOrderBusinessLogic?
  var router: (ShowOrderRoutingLogic & ShowOrderDataPassing)?

  override init(nibName nibNameOrNil: String?, bundle nibBundleOrNil: Bundle?)
  {
    super.init(nibName: nibNameOrNil, bundle: nibBundleOrNil)
    setup()
  }

  private func setup()
  {
    let viewController = self
    let interactor = ShowOrderInteractor()
    let presenter = ShowOrderPresenter()
    let router = ShowOrderRouter()
    viewController.interactor = interactor
    viewController.router = router
    interactor.presenter = presenter
    presenter.viewController = viewController
    router.viewController = viewController
    router.dataStore = interactor
  }

  override func prepare(for segue: UIStoryboardSegue, sender: Any?)
  {
    if (sender as? UIBarButtonItem) == editButtonItem {
      router?.routeToEditOrder(segue: segue)
    }
  }
}
```

- setup 에서 router 초기화
- prepare(for:sender:) 또는 action 메소드에서 RoutingLogic 호출

Router

화면 전환 처리

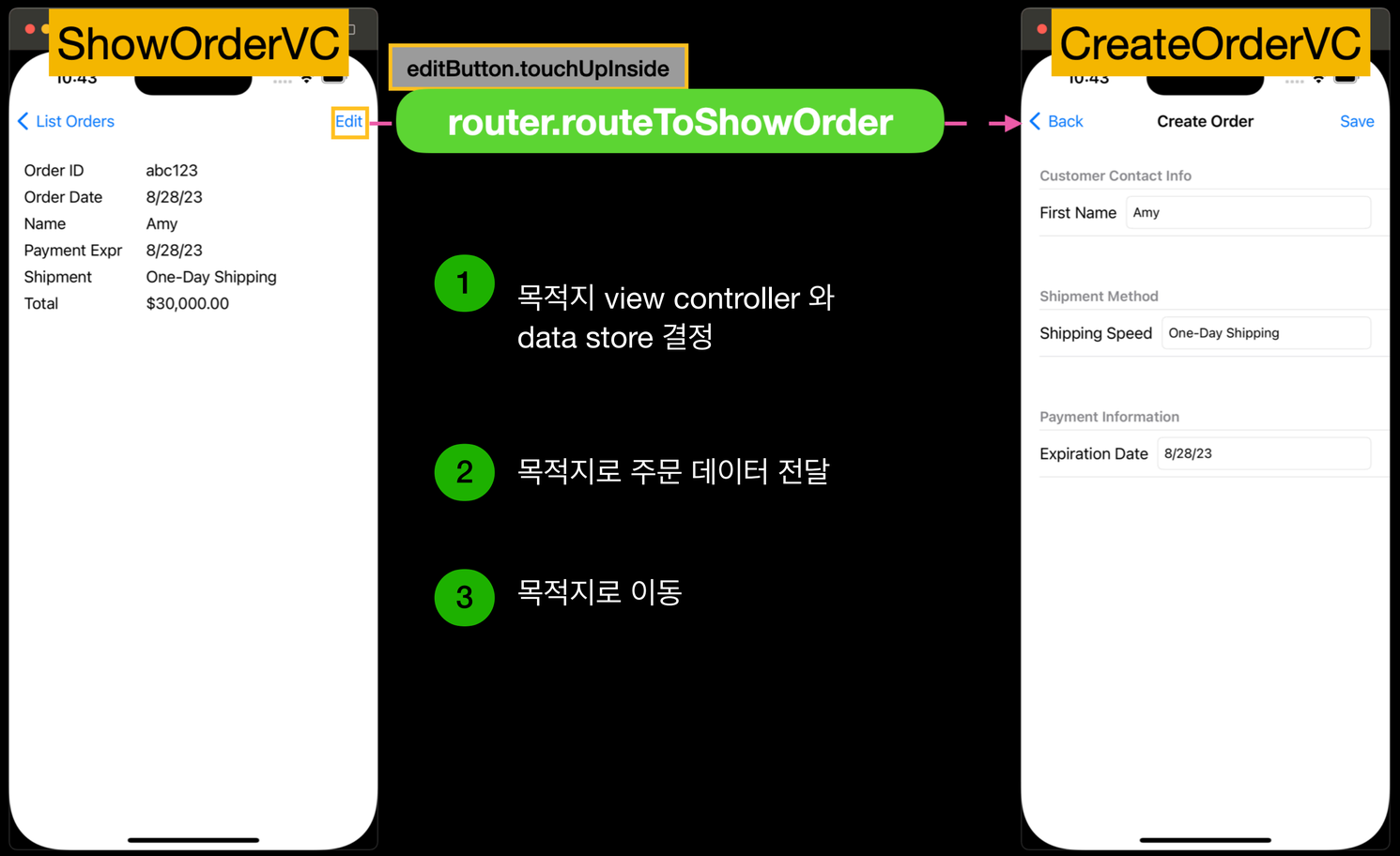
- view controller 에서 navigation 로직을 분리한 것
- 화면 전환 트리거가 발생했을 때 view controller 가 호출할 메소드 구현 RoutingLogic 구현
  - view controller는 segue 실행 전에 호출되는 prepare(for:sender:)
  - 또는 action 메소드에서 RoutingLogic 호출하게 됨
- 화면(scene) 간 전환 로직(navigation logic)과 화면 간 데이터 전달 로직(data passing)이 필요
- 인터페이스 호출시 segue를 전달 받음
  - optional -> 있을 때 없을 때 모두 대응해야 함
- router는 이 라우터를 사용할 source scene의 view controller 를 약하게 참조
- 필요에 따라 전달할, 전달 받을 데이터를 저장하기 위한 dataStore를 참조

3 steps for routing
한 번의 라우팅에 필요한 세 가지 단계

1. **Getting a hold of the destination** 1
   - 목적지 붙잡기 (목적지 결정하기)
   - routeToNextScene(segue:)
   - segue 활용해서 목적지 view controller 와 data store 결정
   - segue 없으면 직접 만들기

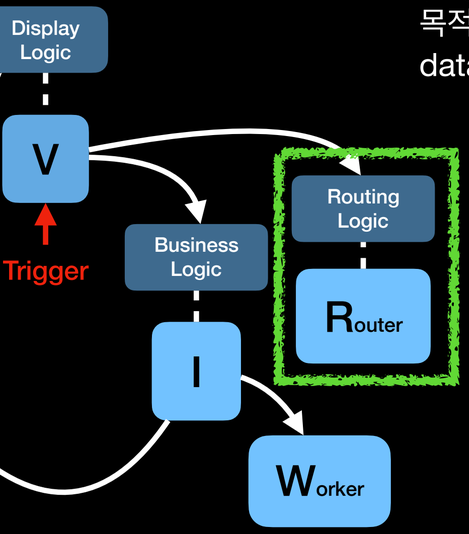
2. **Passing data to the destination** 2
   - 목적지로 데이터 전달하기
   - passDataToNextScene(source:destination:)

3. **Navigating to the destination** 3
   - 목적지로 이동하기
   - navigateToNextScene(source:destination:)


ShowOrderVC
editButton.touchUpInside
router.routeToShowOrder
CreateOrderVC
List Orders
Edit
Order ID abc123
Order Date 8/28/23
Name Amy
Payment Expr 8/28/23
Shipment One-Day Shipping
Total $30,000.00
Back
Create Order
Save
Customer Contact Info
First Name Amy
Shipment Method
Shipping Speed One-Day Shipping
Payment Information
Expiration Date 8/28/23
1
목적지 view controller 와 data store 결정
2
목적지로 주문 데이터 전달
3
목적지로 이동

Router

화면 전환 처리



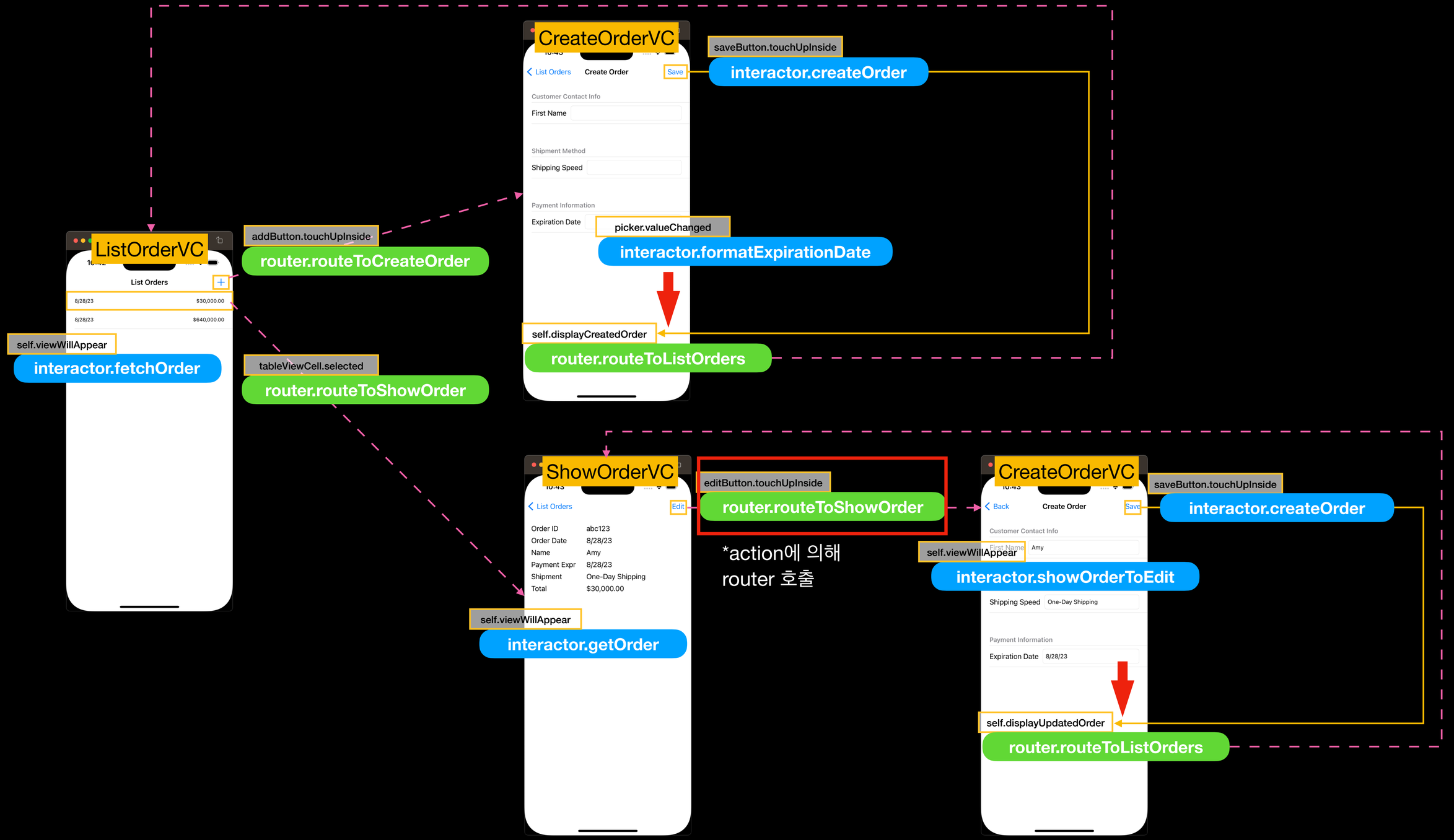
(segue 활용해서)
목적지 view controller 와
data store 결정

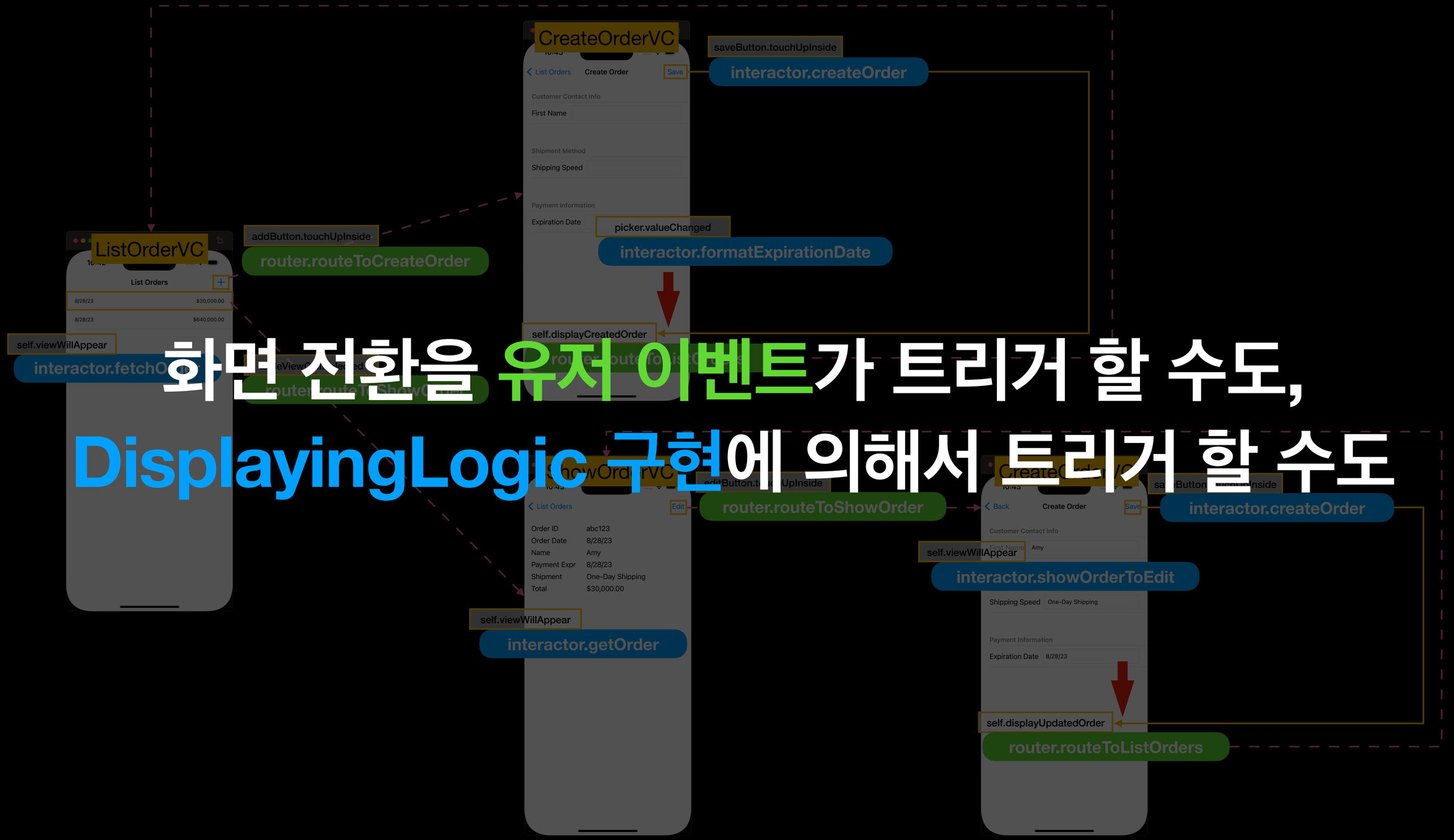
목적지로 데이터 전달

목적지로 이동

```swift
protocol ShowOrderRoutingLogic
{
  func routeToEditOrder(segue: UIStoryboardSegue?)
}

protocol ShowOrderDataPassing
{
  var dataStore: ShowOrderDataStore? { get }
}

class ShowOrderRouter: NSObject, ShowOrderRoutingLogic, ShowOrderDataPassing
{
  weak var viewController: ShowOrderViewController?
  var dataStore: ShowOrderDataStore?

  // 1
  func routeToEditOrder(segue: UIStoryboardSegue?)
  {
    if let segue {
      guard let dataStore,
            let destinationVC = segue.destination as? CreateOrderViewController,
            var destinationDS = destinationVC.router?.dataStore else { return }
      passDataToEditOrder(source: dataStore, destination: &destinationDS)
    } else {
      let nib = UIStoryboard(name: "Main", bundle: nil)
      guard let viewController,
            let dataStore,
            let destinationVC = nib.instantiateViewController(withIdentifier: "CreateOrderViewController") as?
CreateOrderViewController,
            var destinationDS = destinationVC.router?.dataStore else { return }

      passDataToEditOrder(source: dataStore, destination: &destinationDS)
      navigateToEditOrder(source: viewController, destination: destinationVC)
    }
  }

  // 2
  func passDataToEditOrder(source: ShowOrderDataStore, destination: inout CreateOrderDataStore) {
    guard let order = source.order else { return }
    destination.order = order
  }

  // 3
  func navigateToEditOrder(source: ShowOrderViewController, destination: CreateOrderViewController) {
    if let navigationController = source.navigationController {
      navigationController.pushViewController(destination, animated: true)
    } else {
      source.show(destination, sender: self)
    }
  }
}
```

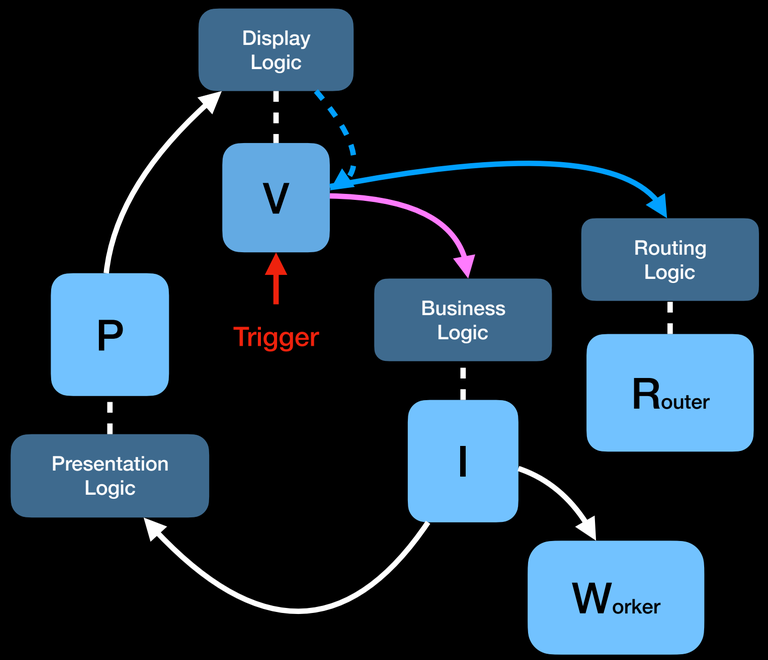
결정된 vc와 store을 전달하며 data passing, navigation 코드 호출


ListOrderVC
List Orders
8/28/23
$30,000.00
8/28/23
$640,000.00
addButton.touchUpInside
router.routeToCreateOrder
tableViewCell.selected
router.routeToShowOrder
self.viewWillAppear
interactor.fetchOrder
CreateOrderVC
List Orders
Create Order
Save
Customer Contact Info
First Name
Shipment Method
Shipping Speed
Payment Information
Expiration Date
saveButton.touchUpInside
interactor.createOrder
picker.valueChanged
interactor.formatExpirationDate
self.displayCreatedOrder
router.routeToListOrders
ShowOrderVC
List Orders
Edit
Order ID abc123
Order Date 8/28/23
Name Amy
Payment Expr 8/28/23
Shipment One-Day Shipping
Total $30,000.00
self.viewWillAppear
interactor.getOrder
editButton.touchUpInside
router.routeToShowOrder
*action에 의해
router 호출
CreateOrderVC
Back
Create Order
Save
Customer Contact Info
First Name
Amy
Shipping Speed
One-Day Shipping
Payment Information
Expiration Date
8/28/23
saveButton.touchUpInside
interactor.createOrder
self.viewWillAppear
interactor.showOrderToEdit
self.displayUpdatedOrder
router.routeToListOrders


CreateOrderVC
saveButton.touchUpInside
interactor.createOrder
List Orders
Create Order
Save
Customer Contact Info
First Name
Shipment Method
Shipping Speed
Payment Information
Expiration Date
picker.valueChanged
interactor.formatExpirationDate
ListOrderVC
addButton.touchUpInside
router.routeToCreateOrder
List Orders
8/28/23
$30,000.00
8/28/23
$640,000.00
self.viewWillAppear
interactor.fetchOrders
self.displayCreatedOrder
화면 전환을 유저 이벤트가 트리거 할 수도,
DisplayingLogic 구현에 의해서 트리거 할 수도
editButton.touchUpInside
router.routeToShowOrder
List Orders
Edit
Order ID abc123
Order Date 8/28/23
Name Amy
Payment Expr 8/28/23
Shipment One-Day Shipping
Total $30,000.00
self.viewWillAppear
interactor.getOrder
Back
Create Order
Save
Customer Contact Info
First Name Amy
self.viewWillAppear
interactor.showOrderToEdit
Shipping Speed One-Day Shipping
Payment Information
Expiration Date 8/28/23
interactor.createOrder
self.displayUpdatedOrder
router.routeToListOrders

CreateOrderViewController

유저 이벤.. 아니.. Displaying Logic이 쏘아 올린 Routing




```swift
extension CreateOrderViewController {
  @IBAction func saveButtonTapped(_ sender: UIBarButtonItem) {
    ...

    let orderFormFields = CreateOrder.OrderFormFields(...)

    if interactor?.order == nil {
      let request = CreateOrder.CreateOrder.Request(orderFormFields: orderFormFields)
      interactor?.createOrder(request: request)
    } else {
      let request = CreateOrder.UpdateOrder.Request(orderFormFields: orderFormFields)
      interactor?.updateOrder(request: request)
    }
  }

  func displayCreatedOrder(viewModel: CreateOrder.CreateOrder.ViewModel) {
    if viewModel.order != nil {
      router?.routeToListOrders(segue: nil)
    } else {
      showOrderFailueAlert(
        title: "",
        message: ""
      )
    }
  }

  func displayUpdatedOrder(viewModel: CreateOrder.UpdateOrder.ViewModel) {
    if viewModel.order != nil {
      router?.routeToShowOrder(segue: nil)
    } else {
      showOrderFailueAlert(
        title: "",
        message: ""
      )
    }
  }
}
```

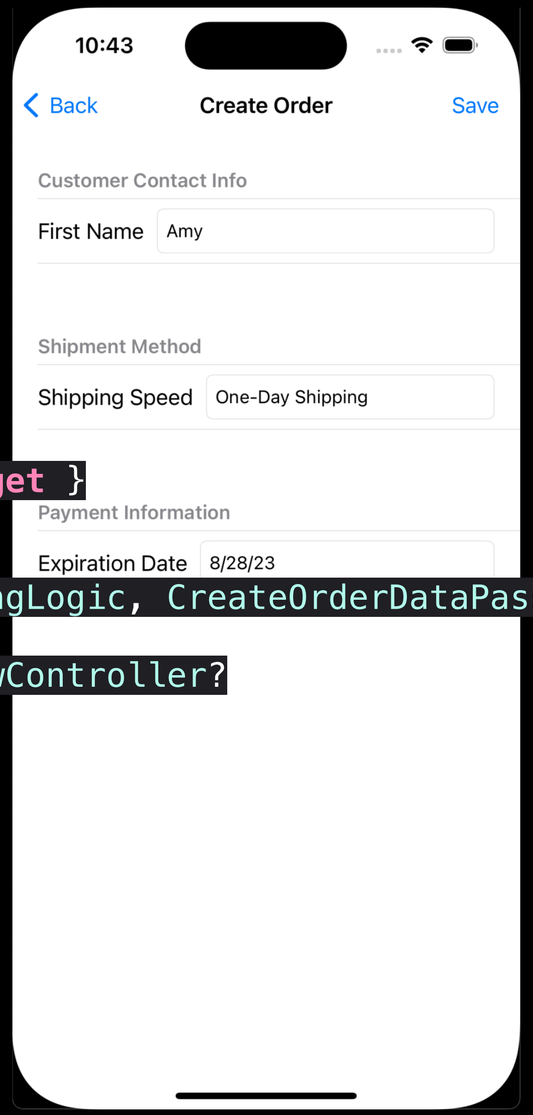
CreateOrderRouter

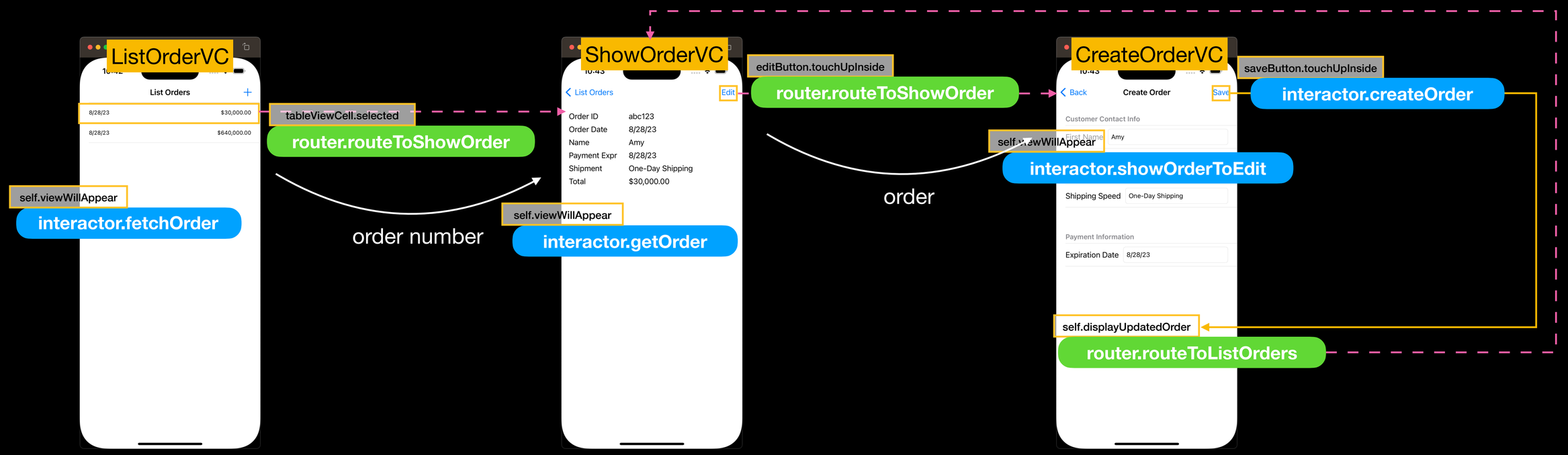
```swift
class CreateOrderRouter: NSObject, CreateOrderRoutingLogic, CreateOrderDataPassing
{
  weak var viewController: CreateOrderViewController?
  var dataStore: CreateOrderDataStore?

  func routeToListOrders(segue: UIStoryboardSegue?) {
    if let segue {
      guard let dataStore,
            let destinationViewController = segue.destination as? ListOrderViewController,
            var destinationDataStore = destinationViewController.router?.dataStore else { return }
      passDataToListOrders(source: dataStore, destination: &destinationDataStore)
    } else {
      guard let viewController, let dataStore,
            let destinationViewController = viewController.navigationController?.viewControllers.first(where: { vc in vc is ListOrderViewController }) as? ListOrderViewController,
            var destinationDataStore = destinationViewController.router?.dataStore else { return }

      passDataToListOrders(source: dataStore, destination: &destinationDataStore)
      navigateToListOrders(source: viewController, destination: destinationViewController)
    }
  }

  func routeToShowOrder(segue: UIStoryboardSegue?) {
    if let segue {
      guard let dataStore,
            let destinationViewController = segue.destination as? ShowOrderViewController,
            var destinationDataStore = destinationViewController.router?.dataStore else { return }
      passDataToShowOrder(source: dataStore, destination: &destinationDataStore)
    } else {
      guard let viewController, let dataStore,
            let destinationViewController = viewController.navigationController?.viewControllers.first(where: { vc in vc is ShowOrderViewController }) as? ShowOrderViewController,
            var destinationDataStore = destinationViewController.router?.dataStore else { return }

      passDataToShowOrder(source: dataStore, destination: &destinationDataStore)
      navigateToShowOrders(source: viewController, destination: destinationViewController)
    }
  }

  func navigateToListOrders(source: CreateOrderViewController, destination: ListOrderViewController)
  {
    source.navigationController?.popViewController(animated: true)
  }

  func navigateToShowOrders(source: CreateOrderViewController, destination: ShowOrderViewController)
  {
    source.navigationController?.popViewController(animated: true)
  }

  func passDataToListOrders(source: CreateOrderDataStore, destination: inout ListOrderDataStore) { }

  func passDataToShowOrder(source: CreateOrderDataStore, destination: inout ShowOrderDataStore)
  {
    destination.order = source.order
  }
}
```

```swift
@objc protocol CreateOrderRoutingLogic
{
  func routeToListOrders(segue: UIStoryboardSegue?)

  func routeToShowOrder(segue: UIStoryboardSegue?)
}

protocol CreateOrderDataPassing
{
  var dataStore: CreateOrderDataStore? { get }
}
```

DataStore?

```
protocol CreateOrderDataPassing
{
  var dataStore: CreateOrderDataStore? { get }
}

class CreateOrderRouter: CreateOrderRoutingLogic, CreateOrderDataPassing
{
  weak var viewController: CreateOrderViewController?
  var dataStore: CreateOrderDataStore?
```




ListOrderVC
List Orders
8/28/23
$30,000.00
8/28/23
$640,000.00
self.viewWillAppear
interactor.fetchOrder
tableViewCell.selected
router.routeToShowOrder
order number
ShowOrderVC
List Orders
Edit
Order ID abc123
Order Date 8/28/23
Name Amy
Payment Expr 8/28/23
Shipment One-Day Shipping
Total $30,000.00
self.viewWillAppear
interactor.getOrder
editButton.touchUpInside
router.routeToShowOrder
order
CreateOrderVC
Back
Create Order
Save
Customer Contact Info
First Name Amy
self.viewWillAppear
interactor.showOrderToEdit
Shipping Speed One-Day Shipping
Payment Information
Expiration Date 8/28/23
saveButton.touchUpInside
interactor.createOrder
self.displayUpdatedOrder
router.routeToListOrders

ShowOrderRouter

```
protocol ShowOrderDataPassing
{
  var dataStore: ShowOrderDataStore? { get }
}

class ShowOrderRouter: NSObject, ShowOrderRoutingLogic, ShowOrderDataPassing
{
  weak var viewController: ShowOrderViewController?
  var dataStore: ShowOrderDataStore?

  func routeToEditOrder(segue: UIStoryboardSegue?)
  {
    if let segue {
      guard let dataStore,
            let destinationVC = segue.destination as? CreateOrderViewController,
            var destinationDS = destinationVC.router?.dataStore else { return }
      passDataToEditOrder(source: dataStore, destination: &destinationDS)
    } else {
      let nib = UIStoryboard(name: "Main", bundle: nil)
      guard let viewController,
            let dataStore,
            let destinationVC = nib.instantiateViewController(withIdentifier: "CreateOrderViewController") as? CreateOrderViewController,
            var destinationDS = destinationVC.router?.dataStore else { return }
      passDataToEditOrder(source: dataStore, destination: &destinationDS)
      navigateToEditOrder(source: viewController, destination: destinationVC)
    }
  }

  func passDataToEditOrder(source: ShowOrderDataStore, destination: inout CreateOrderDataStore) {
    guard let order = source.order else { return }
    destination.order = order
  }

  func navigateToEditOrder(source: ShowOrderViewController, destination: CreateOrderViewController) {
    if let navigationController = source.navigationController {
      navigationController.pushViewController(destination, animated: true)
    } else {
      source.show(destination, sender: self)
    }
  }
}
```

CreateOrderViewController

dataStore setup

```
class CreateOrderViewController: UITableViewController, CreateOrderDisplayLogic
{
  var interactor: (CreateOrderBusinessLogic & CreateOrderDataStore)?
  var router: (NSObjectProtocol & CreateOrderRoutingLogic & CreateOrderDataPassing)?

  override init(nibName nibNameOrNil: String?, bundle nibBundleOrNil: Bundle?)
  {
    super.init(nibName: nibNameOrNil, bundle: nibBundleOrNil)
    setup()
  }

  required init?(coder aDecoder: NSCoder)
  {
    super.init(coder: aDecoder)
    setup()
  }

  private func setup()
  {
    let viewController = self
    let interactor = CreateOrderInteractor()
    let presenter = CreateOrderPresenter()
    let router = CreateOrderRouter()
    viewController.interactor = interactor
    viewController.router = router
    interactor.presenter = presenter
    presenter.viewController = viewController
    router.viewController = viewController
    router.dataStore = interactor
  }
}
```

init 에서 dataStore 초기화

ShowOrderRouter

데이터 전달(destination dataStore 설정)

```swift
protocol ShowOrderDataPassing
{
  var dataStore: ShowOrderDataStore? { get }
}

class ShowOrderRouter: NSObject, ShowOrderRoutingLogic, ShowOrderDataPassing
{
  weak var viewController: ShowOrderViewController?
  var dataStore: ShowOrderDataStore?

  func routeToEditOrder(segue: UIStoryboardSegue?)
  {
    if let segue {
      guard let dataStore,
            let destinationVC = segue.destination as? CreateOrderViewController,
            var destinationDS = destinationVC.router?.dataStore else { return }
      passDataToEditOrder(source: dataStore, destination: &destinationDS)
    } else {
      let nib = UIStoryboard(name: "Main", bundle: nil)
      guard let viewController,
            let dataStore,
            let destinationVC = nib.instantiateViewController(withIdentifier: "CreateOrderViewController") as? CreateOrderViewController,
            var destinationDS = destinationVC.router?.dataStore else { return }
      passDataToEditOrder(source: dataStore, destination: &destinationDS)
      navigateToEditOrder(source: viewController, destination: destinationVC)
    }
  }

  func passDataToEditOrder(source: ShowOrderDataStore, destination: inout CreateOrderDataStore) {
    guard let order = source.order else { return }
    destination.order = order
  }

  func navigateToEditOrder(source: ShowOrderViewController, destination: CreateOrderViewController) {
    if let navigationController = source.navigationController {
      navigationController.pushViewController(destination, animated: true)
    } else {
      source.show(destination, sender: self)
    }
  }
}
```

view controller 초기화(init) 후에 dataStore 접근

CreateOrderInteractor

dataStore setup

```
protocol CreateOrderDataStore
{
  var order: Order? { get set }
}

class CreateOrderInteractor: CreateOrderBusinessLogic, CreateOrderDataStore
{
  var order: Order?

  var presenter: CreateOrderPresentationLogic?

  // MARK: Edit

  func showOrderToEdit(request: CreateOrder.EditOrder.Request) {
    guard let order else { return }
    let response = CreateOrder.EditOrder.Response(order: order)
    presenter?.presentOrderToEdit(response: response)
  }
   ...
}
```

화면 구성에 router에서 설정한 order 활용

질문