

코드리뷰

- DesignableStrategy
- AttributedFormattable
- MVVM/POP

복잡도(Complexity)

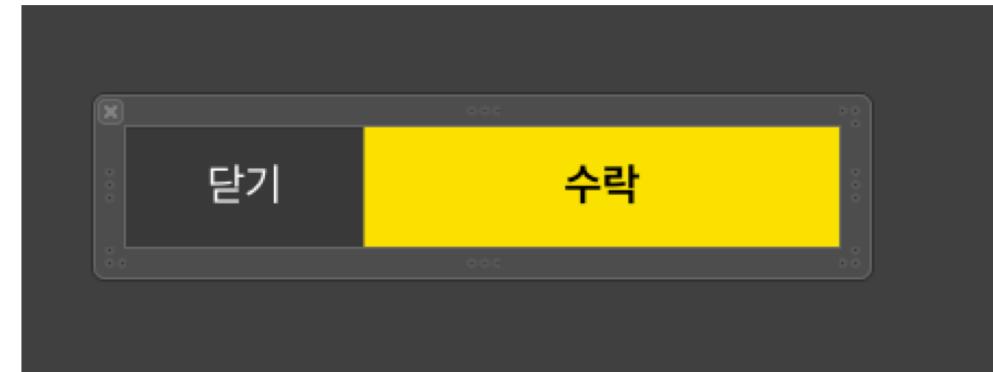
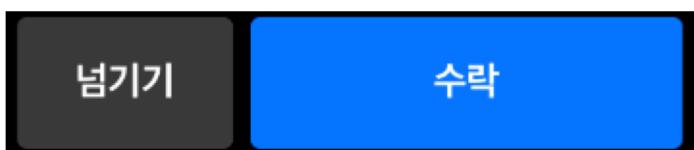
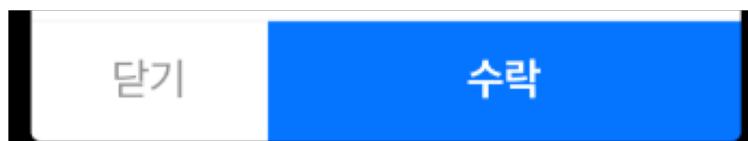
- 프로그램의 로직의 복잡도는 절대값.
- 프로그램 복잡도 = 툴의 복잡도 + 표현의 복잡도
- 프로그램의 복잡도 크기에 따라 툴/표현 복잡도의 비율이 달라져야 한다. (이걸 바꾸는 과정이 리팩토링: 공통 표현을 툴로 이동)
- 간단한 프로젝트: 간단한 툴 + 중간 복잡도의 표현
- 복잡한 프로젝트: 복잡한 툴 + 낮은 복잡도의 표현
- Enterprise급 iOS 개발환경의 문제: Storyboard, ViewController (화면 중심 로직)에 너무 의존한다.

Storyboard(Xib)의 문제점

- =Objective C의 문제점 (Linear함, 표현력이 부족함. 마치 메모리 배열에 직접 접근 하는 느낌)
- vs Swift의 표현력 (enum, type safety)
- 예) 동일한 화면이지만 동적으로 표시되는 UIButton의 경우
 - @IBOutlet에 접근하는게 의미 있는지???
 - @IBAction이 호출될 가능성이 있는지???

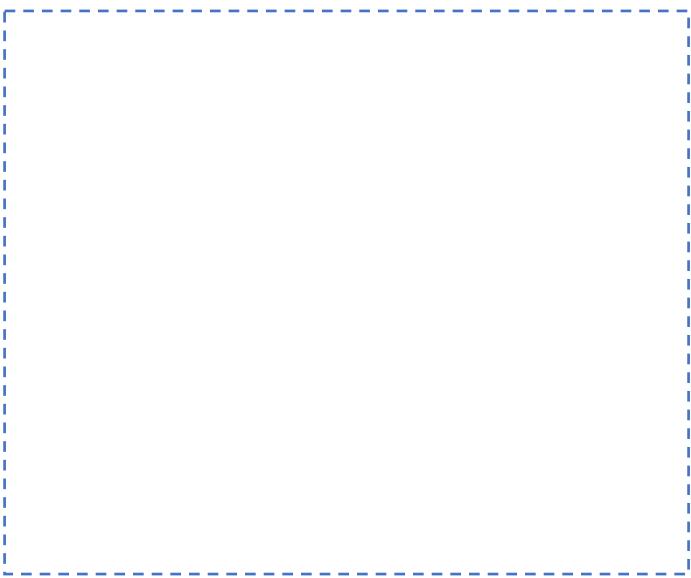
View Injection의 한계

- LookAndFeelDesignView: Xib단위 모듈화
- Property들이 flat하다.
- Class 지정이 어려움.



ConfirmControl.swift

DesignableStrategy



```
protocol DesignableStrategy {  
    func asView() -> UIView  
}
```

표현에 필요한 데이터는?

경유지 상세 주소

역삼동

역삼세무서

닫기

경유지 상세 주소

신대방동 64-12

성공회포천나눔의집 포천 나눔의
집이주민지원센터 / 포천푸드뱅크 /
포천나눔돌봄사회서비스센터

이태원동 124-5

성공회포천나눔의집 포천 나눔의
집이주민지원센터 / 포천푸드뱅크 /
포천나눔돌봄사회서비스센터

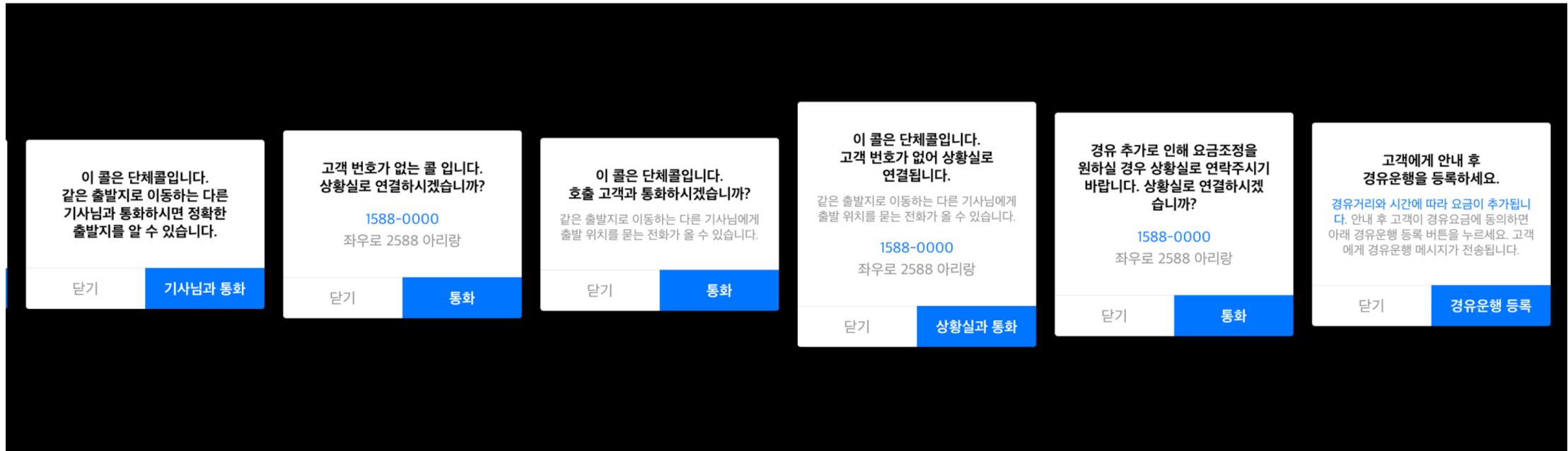
북아현동 141-52

성공회포천나눔의집 포천 나눔의
집이주민지원센터 / 포천푸드뱅크 /
포천나눔돌봄사회서비스센터

닫기

```
let view = DesignStretegy.LinearWithSeparator(  
    contents: datas.map { data in  
        return DesignStretegy.Linear(  
            contents: [  
                DesignStretegy.Label(  
                    font: UIFont.appleSDGothicNeoRegular(ofSize: 20),  
                    textColor: UIColor.wheelBlue,  
                    numberOfLines: 2,  
                    text: data.dongName,  
                    alignment: .center  
                ),  
                DesignStretegy.FixedHeight(  
                    height: 5  
                ),  
                DesignStretegy.Label(  
                    font: UIFont.appleSDGothicNeoRegular(ofSize: 20),  
                    textColor: UIColor.black,  
                    numberOfLines: 0,  
                    text: data.displayName,  
                    alignment: .center  
                )  
            ],  
            alignment: .center,  
            axis: .vertical  
        )  
    },  
    separatorImage: UIImage(named: "iconArrowWhiteS", in: Bundle(for: MediatesListDetailView.self), compatibleWith: nil)!,  
    separatingSpace: 20  
)  
.asView()
```

팝업...



KDUIConfirmConfigurables.swift

Review

- 구현체에 의존하지 않는다. 정의에 집중한다.
 - 리스트의 경우 Tableview를 쓰던, StackView를 쓰던 상관 없음.
 - (= 이식성이 있다. UIView -> NSView, 심지어 HTML으로 만들수도. UIView가 프로토콜에 노출되잖아요. addSubview때문에.)
- like XML that not XML: 표현력이 좋고, 안전하다.
(Complie_Check)
- Button Action은 어떻게??
 - Action이 ViewController에 의존하는 순간부터 지옥이 펼쳐짐, 단위 View 자체가 해결해야함. (IBAction, Delegate, notification 전부 UIViewController로 감... 차이는?)

AttributedFormattable

- AttributedString 만들기가 너무 어려워요.
- setColor, setFont ... setUnderline
- “맥도날드 (맥도날드 직영점)”.setColor(“맥도날드”, .blue)
- 어떻게 만드느냐에 집중하면(순서) 복잡하다.
- 어떻게 표시되어야 하나에 집중하면 편하다.
- 토큰단위로

```
protocol AttributedFormatable {  
    var formatted: NSAttributedString { get }  
}
```

```
enum AttributedFormat {  
  
    struct ImageCentered: AttributedFormatable { ... }  
  
    struct FixedWidth: AttributedFormatable { ... }  
  
    struct Text: AttributedFormatable { ... }  
  
    struct SingleSpacing: AttributedFormatable { ... }  
  
    struct Line: AttributedFormatable { ... }  
  
    struct Context: AttributedFormatable { ... }  
}
```



```
return AttributedFormat.Context(  
    data: [  
        AttributedFormat.Text(  
            text: "도보",  
            font: FontConfigurable.regular(fontSize),  
            color: UIColor.wheel8E  
        ),  
        AttributedFormat.SingleSpacing(font: FontConfigurable.regular(fontSize)),  
        AttributedFormat.Text(  
            text: distance.callCardDistanceDescription,  
            font: FontConfigurable.regular(fontSize),  
            color: UIColor.white  
        ),  
        AttributedFormat.SingleSpacing(font: FontConfigurable.regular(fontSize)),  
        AttributedFormat.Text(  
            text: "/",  
            font: FontConfigurable.regular(fontSize),  
            color: UIColor.wheel53  
        ),  
        AttributedFormat.SingleSpacing(font: FontConfigurable.regular(fontSize)),  
        AttributedFormat.Text(  
            text: "약",  
            font: FontConfigurable.regular(fontSize),  
            color: UIColor.wheel8E  
        ),  
        AttributedFormat.SingleSpacing(font: FontConfigurable.regular(fontSize)),  
        AttributedFormat.Text(  
            text: time.minuteDescription,  
            font: FontConfigurable.regular(fontSize),  
            color: UIColor.white  
        )  
    ],  
    alignment: .center  
).formatted
```

Typeface: "도보"

AppleSDGothicNeo-Regular

Size: 27pt

Align: Center

#g5
#8e8e8e

Typeface

AppleSDGothicNeo-Regular

Size: 27pt

Align: Center

#w1
#ffffff

Typeface: "/"

AppleSDGothicNeo-Regular

Size: 27pt

Align: Center

#g4
#535353

Content

도보 130m / 약 5분

Attributed Strings

```
let attributedString = NSMutableAttributedString(string: "도보 130m / 약 5분",  
    attributes: [  
        .font: UIFont(name: "AppleSDGothicNeo-Regular", size: 27.0)!,  
        .foregroundColor: UIColor.w1  
    ])  
attributedString.addAttribute(.foregroundColor, value: UIColor.g5, range:  
    NSRange(location: 0, length: 2))  
attributedString.addAttribute(.foregroundColor, value: UIColor.g4, range:  
    NSRange(location: 8, length: 1))  
attributedString.addAttribute(.foregroundColor, value: UIColor.g5, range:  
    NSRange(location: 10, length: 1))
```

Review

- DesignableStrategy랑 비슷한 구조.
- 어떻게 만드느냐가 아니라, 어떻게 표현 되느냐에 집중함.
- 토큰 단위로 분리해서 모듈화 하고, 조합을 통해 복잡도를 제어.

MVVM/POP

- Pain Points
 - 'API 모델이 아직 안정해졌어요.'
 - 'API 로직에 문제가 있어요.'
 - '기사 도착 화면 디자인이 보고 싶어요.'
 - '기사 도착 화면 디자인이 보고 싶어요. 3분 지났을 때요'
 - '기사 도착 화면 디자인이 보고 싶어요. 슈퍼 배차일 때요'
 - '기사 도착 화면 디자인이 보고 싶어요. 에러 났을 때요'
- 디자인/기획/서버가 다 명확해야 개발 가능.
- 테스트가 어려움.

Why? (Model편)

- ViewController가 공용 모델을 쓴다.
 - 전지전능하고 모든걸 다 할수 있는 Call 모델이시여.
- 공용 모델을 쓰면 API <-> Logic <-> View 까지 의존성이 생김.
 - 의존성을 확인하는 쉬운 방법: 별도 타겟생성하여 단위 빌드가 되는가?
- Testing/Mocking이 어렵다.
- Layer가 필요하다.
- MVVM이란? (서로 다른 계층 간에 Layer이자 mapping에 관한 것)

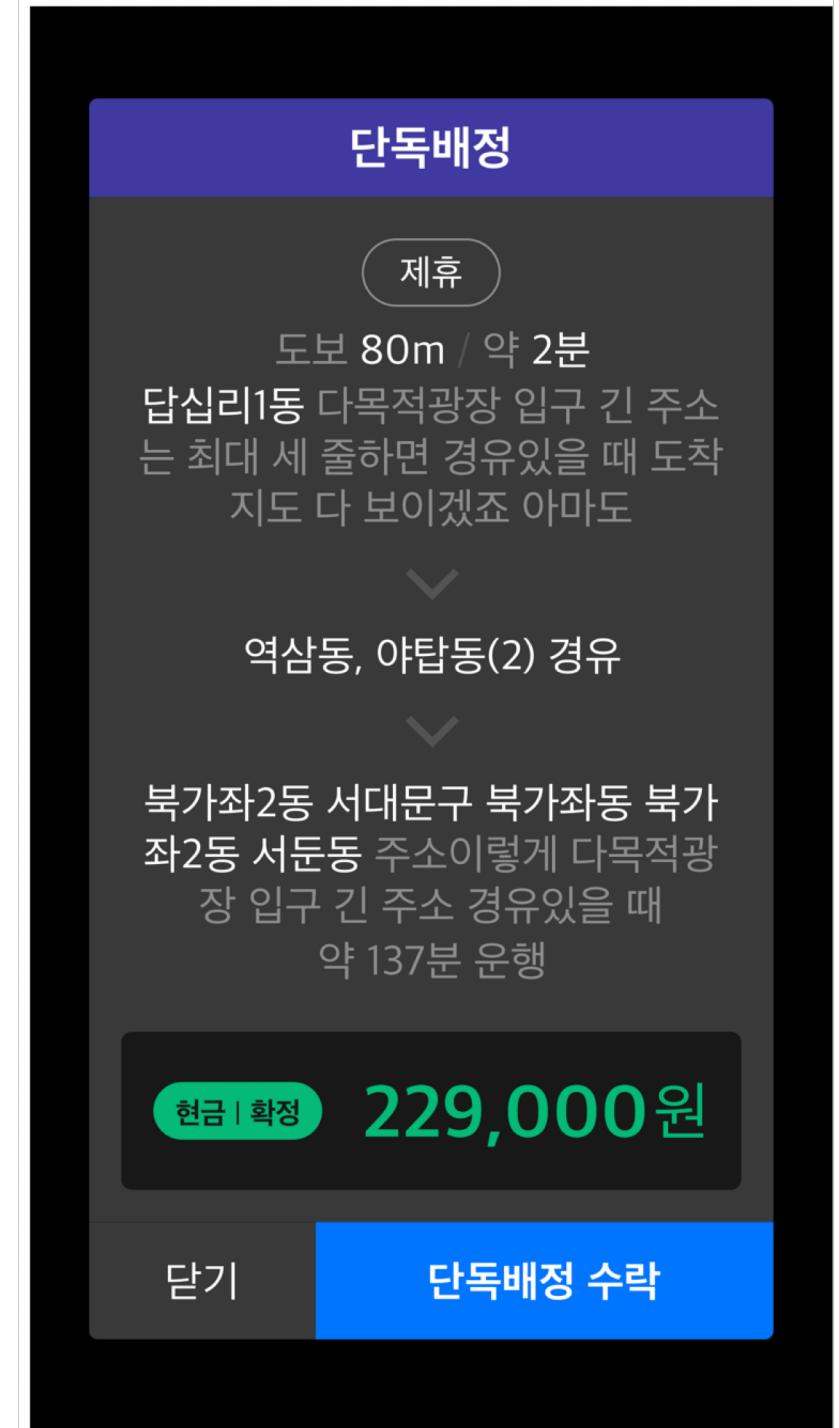
콜리스트 확인팝업

콜카드 선택시 표시됨 (Model: CallCard)
콜리스트 선택시 표시됨 (Model: CallList)

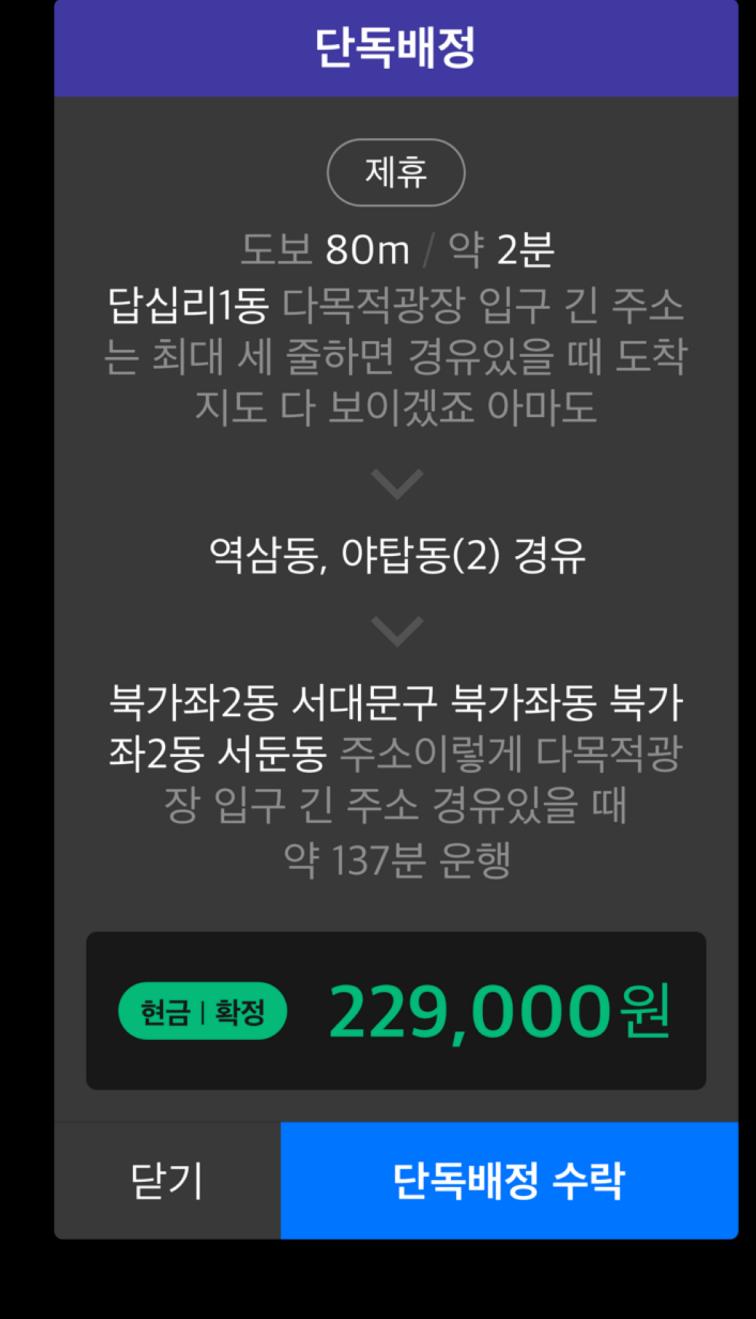
ViewController가 모델을 알게 한다면.

- 콜카드인 case 분기
- 콜리스트인 case 분기
- 테스트...

필요한 데이터에만 집중하자.



```
protocol ListConfirmVCCContentsConfigurable {
    var routingType: RoutingDescriptionView.RoutingType { get }
    var originDisplay: LocationDisplay { get }
    var destDisplay: LocationDisplay { get }
    var farePaymentViewData: GeneralFarePaymentViewData { get }
    var stopover: String? { get }
    var isFavorite: Bool { get }
    var isOpenCall: Bool { get }
    var exclusiveDispatch: CallCard.ExclusiveDispatch? { get }
    var timeToDestDescrption: String? { get }
}
```



Mapping

- 모델에 protocol Extension할 경우: namespace pollution이 생김.
- Call 모델을 화면 A에서 쓰고, B에서도 쓴다면 프로퍼티 중복 발생 가능성
- 전혀 다른 namespace에 정의 하자. 대신 찾기 쉬운.
- RxSwift: Disposable / Disposables (프로토콜 / 구현체)
- 프로토콜에 -s를 붙이자.
- 모델 소스는 해당 구현체의 초기화 시점에 속성으로 할당.
(Wrapper)

```
enum ListConfirmVCContentsConfigurables {
    struct ByCallCard: ListConfirmVCContentsConfigurable { ... }
    struct ByCallList: ListConfirmVCContentsConfigurable { ... }
}
```

```
struct ByCallCard: ListConfirmVCContentsConfigurable {
    let callCard: CallCard

    var routingType: RoutingDescriptionView.RoutingType {
        switch callCard.origin.route.routeType {
        case .car:
            return RoutingDescriptionView.RoutingType.car(time: callCard.origin.route.time, distance: callCard.origin.route.distance)
        case .walk:
            return RoutingDescriptionView.RoutingType.walk(time: callCard.origin.route.time, distance: callCard.origin.route.distance)
        }
    }

    var originDisplay: LocationDisplay {
        return callCard.origin.info.display
    }

    var destDisplay: LocationDisplay {
        return callCard.destination.info.display
    }

    var farePaymentViewData: GeneralFarePaymentViewData {
        return GeneralFarePaymentViewDataByCallCard(callCard: callCard).farePaymentViewData
    }

    var stopover: String? {
        return callCard.stopover?.displayName
    }

    var isFavorite: Bool {
        return callCard.isFavorite
    }

    var isOpenCall: Bool {
        return callCard.isOpenCall
    }

    var exclusiveDispatch: CallCard.ExclusiveDispatch? {
        return callCard.exclusiveDispatch
    }

    var timeToDestDescrption: String? {
        return callCard.timeToDestDisplay
    }
}
```

```
ListConfirmVC.instance(callCard)
ListConfirmVC.instance(ListConfirmVCCContentsConfigurables.ByCallCard(callCard: callCard))
```

- Extension
 - 콜카드 모델을 사용할 거예요.
- Wrapping
 - 콜카드 모델로 ByCallCard 에 정의된 맵핑을 사용할 거예요. (Layer)

Mock



```
private struct Data: ListConfirmVCContentsConfigurable {
    let farePaymentViewData: GeneralFarePaymentViewData

    let routingType: RoutingDescriptionView.RoutingType

    let originDisplay: LocationDisplay

    let destDisplay: LocationDisplay

    let stopover: String?

    let isFavorite: Bool

    let isOpenCall: Bool

    let exclusiveDispatch: CallCard.ExclusiveDispatch?

    let timeToDestDescrpption: String?
}

static func mock1() -> ListConfirmVCContentsConfigurable {
    return Data(
        farePaymentViewData: GeneralFarePaymentViewData(
            fareType: GeneralFarePaymentViewData.FareType.meter(GeneralFarePaymentViewData.FareType.MeterPaymentType.card),
            fare: 2000
        ),
        routingType: RoutingDescriptionView.RoutingType.walk(time: TimeInSecond(degree: 60), distance: DistanceInMeter(degree: 58)),
        originDisplay: LocationDisplay(
            first: "백현동",
            second: "알파돔타워"
        ),
        destDisplay: LocationDisplay(
            first: "광진구 자양4동",
            second: "한국기술자격검정장"
        ),
        stopover: nil,
        isFavorite: false,
        isOpenCall: false,
        exclusiveDispatch: nil,
        timeToDestDescrpption: "약 20분 운행"
    )
}

static func mock2() -> ListConfirmVCContentsConfigurable {
    return Data(
        farePaymentViewData: GeneralFarePaymentViewData(
            fareType: GeneralFarePaymentViewData.FareType.meter(GeneralFarePaymentViewData.FareType.MeterPaymentType.cash),
            fare: 3000
        ),
        routingType: RoutingDescriptionView.RoutingType.walk(time: TimeInSecond(degree: 60), distance: DistanceInMeter(degree: 58)),
        originDisplay: LocationDisplay(
            first: "백현동",
            second: "알파돔타워"
        ),
        destDisplay: LocationDisplay(
            first: "광진구 자양4동",
            second: "한국기술자격검정장"
        ),
        stopover: nil,
        isFavorite: false,
        isOpenCall: false,
        exclusiveDispatch: nil,
        timeToDestDescrpption: "약 20분 운행"
    )
}
```

Why? (Singleton편)

- 모델을 분리해도 Mock 테스트하기 어렵다.
- 모델 뿐 아니라, API/Local_Storage/Manager/Date 등에 의존성이 있음.
- 동일하게 분리하자.

Ex) 로컬 저장소 분리

- 운행 중에 내비 길안내를 실행하면 운행 종료 시점에 길안내를 종료 해주어야 함.
- 앱을 강제종료해도 해당 동작이 되어야함.
- 내비 길안내 시작 시점에 콜아이디를 userdefault에 넣고 종료 시점에 이를 확인해서 종료 스킴을 실행할지 여부를 결정함.

Before

```
}

WheelCashManager.naviStartedLastCallID = self.injected.call.id
return Observable<Bool>.empty()
```

```
WheelCashManager.naviStartedLastCallID == self.injected.call.id }
```

```
_ = ApplicationSchemer.NaviExternalEnd().run()
WheelCashManager.naviStartedLastCallID = nil
observer.onNext(true)
```

After

```
protocol InAppNaviDrivingCongifurable {
    var inAppNaviConfig: InAppNavigateConfigurable { get }
    var meterStream: KakaoNaviWrapperVC.MeterStream { get }
    func setAppToAppNaviCycle(start: Bool)
    func getAppToAppNaviCycle() -> Bool
}

func setAppToAppNaviCycle(start: Bool) {
    if start {
        WheelCashManager.naviStartedLastCallID = call.id
    } else {
        WheelCashManager.naviStartedLastCallID = nil
    }
}

func getAppToAppNaviCycle() -> Bool {
    return (WheelCashManager.naviStartedLastCallID == call.id)
}
```

Mock

```
class Mock: InAppNaviDrivingCongifurable {
    func setAppToAppNaviCycle(start: Bool) {
        data = start
    }

    func getAppToAppNaviCycle() -> Bool {
        return data
    }

    var data: Bool = false
}
```

API, Singleton도 똑같음

- 인터페이스에 return Type이 필요한가? (Setter/Getter)
- 데이터 방향은 어느 쪽인가?
 - 동기 vs 비동기
 - OutStream vs InStream
- 호출할 때마다 값이 변할 수 있는가?

확인해서 인터페이스로 분리.

모듈화 순서

기존 코드

1. 모델 의존성 확인
2. 인터페이스 정의
3. 모델 의존성 추출/매핑
4. Mock layer 테스트

신규 코드

1. 인터페이스 정의
2. Mock layer 테스트
3. 모델 연동(API 등)

결론.

- 세 주제다 똑같은 결론.
- 구현과 인터페이스와 분리.
- 구현이 아니라 인터페이스에 프로그래밍 하자.

- 토비의 스프링 1권.