

# iOS Seminar 4

Wafflestudio 2024



# SwiftUI+Data Managing

```
struct SimpleSeminar {  
    var title = "iOS Seminar"  
    var isDone = false  
}
```

```
final class SimpleSeminar {  
    var title = "iOS Seminar"  
    var isDone = false  
}
```

struct 대신 class를 사용한다면?

- struct와 class의 차이

```
import SwiftUI  
  
struct UsingStateView: View {  
  
    @State private var seminar: SimpleSeminar = .init()  
  
    var body: some View {  
        VStack(spacing: 16) {  
            Spacer()  
            Image(systemName: "swift")  
                .resizable()  
                .frame(width: 24, height: 24)  
            Text(seminar.title)  
                .font(.system(size: 20))  
                .strikethrough(seminar.isDone)  
            Button {  
                seminar.isDone.toggle()  
                print(seminar.isDone)  
            } label: {  
                Text("Done")  
                    .font(.system(size: 14))  
            }  
                .tint(.blue)  
                .buttonStyle(.bordered)  
                .clipShape(Capsule())  
                .disabled(seminar.isDone)  
            Spacer()  
        }  
    }  
}
```



## SwiftUI가 View를 업데이트하는 방식

- 관련 문서
- SwiftUI는 Data Driven
  - a. View state 공유 (지난 세미나에서 다뤘던 내용)
  - b. Model data의 변경사항 관찰
- View에 의존성이 있는 데이터가 업데이트되면, 그 데이터에 연관된 인터페이스만 자동으로 업데이트
  - 데이터와 View의 분리

## iOS 13.0+에서의 Model data Observation

- Combine(Framework) 이용
  - Publisher과 Subscriber
  - Chain of publishers 끝에 subscriber
    - `send()`
    - `sink(receiveCompletion:receiveValue:)` 혹은 `assign(to:on:)`
- Custom Publisher를 만드는 방법
  - `PassthroughSubject`
  - `CurrentValueSubject`
  - 변화를 알리고 싶은 변수에 `@Published` 이용

## iOS 13.0+에서의 Model data Observation

- `ObservableObject` 프로토콜 채택 (공식문서)
  - @Published가 붙은 property가 변화하기 직전, 변화할 값을 방출
- 변화를 알리고 싶은 변수에 `@Published` 이용
  - class 내의 변수에만 사용 가능 (struct ❌)
  - upstream publisher에서 받은 값을 republish하는 용도로도 활용 가능



```
import Foundation
import Combine

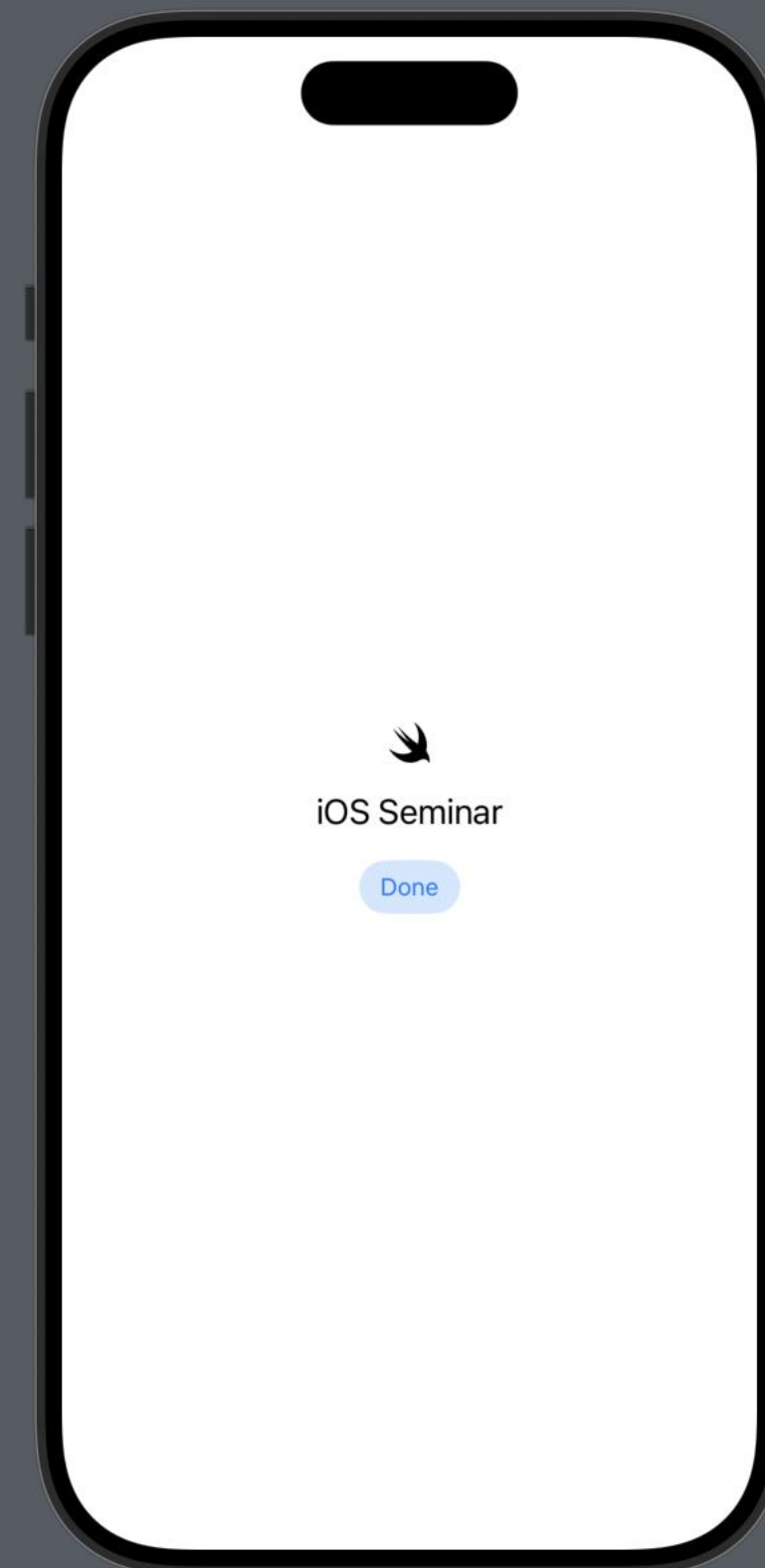
final class PublishingSeminar: ObservableObject {
    let title = "iOS Seminar"
    @Published var isDone = false

    func completeSeminar() {
        isDone = true
    }
}
```

```
import SwiftUI

struct PublishedExample: View {
    @StateObject var seminar = PublishingSeminar()

    var body: some View {
        VStack(spacing: 16) {
            Spacer()
            Image(systemName: "swift")
                .resizable()
                .frame(width: 24, height: 24)
            Text(seminar.title)
                .font(.system(size: 20))
                .strikethrough(seminar.isDone)
            Button {
                seminar.completeSeminar()
            } label: {
                Text("Done")
                    .font(.system(size: 14))
            }
                .tint(.blue)
                .buttonStyle(.bordered)
                .clipShape(Capsule())
                .disabled(seminar.isDone)
            Spacer()
        }
    }
}
```



```
import SwiftUI

@main
struct Seminar4DemoApp: App {
    @StateObject var seminar = PublishingSeminar()

    var body: some Scene {
        WindowGroup {
            PublishedExample(seminar: seminar)
        }
    }
}
```

```
import Foundation
import Combine

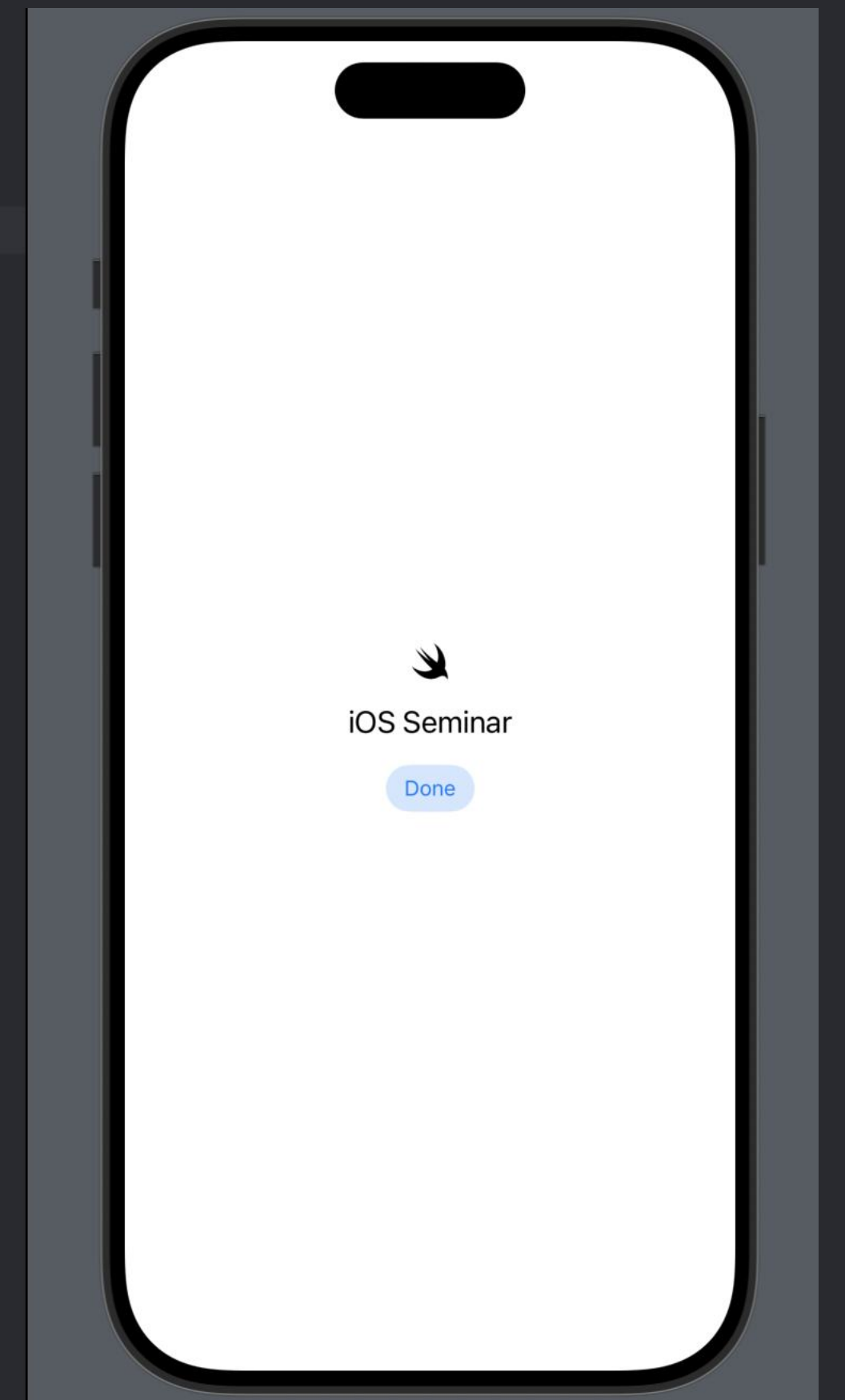
final class PublishingSeminar: ObservableObject {
    let title = "iOS Seminar"
    @Published var isDone = false

    func completeSeminar() {
        isDone = true
    }
}
```

```
import SwiftUI

struct PublishedExample: View {
    @ObservedObject var seminar: PublishingSeminar

    var body: some View {
        VStack(spacing: 16) {
            Spacer()
            Image(systemName: "swift")
                .resizable()
                .frame(width: 24, height: 24)
            Text(seminar.title)
                .font(.system(size: 20))
                .strikethrough(seminar.isDone)
            Button {
                seminar.completeSeminar()
            } label: {
                Text("Done")
                    .font(.system(size: 14))
            }
                .tint(.blue)
                .buttonStyle(.bordered)
                .clipShape(Capsule())
                .disabled(seminar.isDone)
            Spacer()
        }
    }
}
```





## @StateObject

- `ObservableObject` 프로토콜을 채택하는 object 생성
- 선언하는 쪽의 lifetime 동안 단 한번 생성

## @ObservedObject

- `ObservableObject`를 준수하는 타입을 subscribe
- Observable Object에서 변화가 감지되면, view를 다시 그림
- 단, @State처럼 initial value나 default 값을 넘기지 말 것

## iOS 17.0+에서의 Model data Observation

- Observation(Framework) 이용
  - Observable Object와 Observer
- `@Observable` 매크로 이용
- 기본적으로, 접근 가능한 모든 property는 변화가 감지됨
  - 접근 가능한 property의 변화를 추적하고 싶지 않다면 `@ObservationIgnored` 매크로 이용

```
import Foundation
import Observation

@Observable final class ObservableSeminar {
    @ObservationIgnored var title = "iOS Seminar"
    var isDone = false

    func completeSeminar() {
        isDone = true
    }

    func changeSeminarTitle() {
        title = "Spring Seminar"
    }
}
```

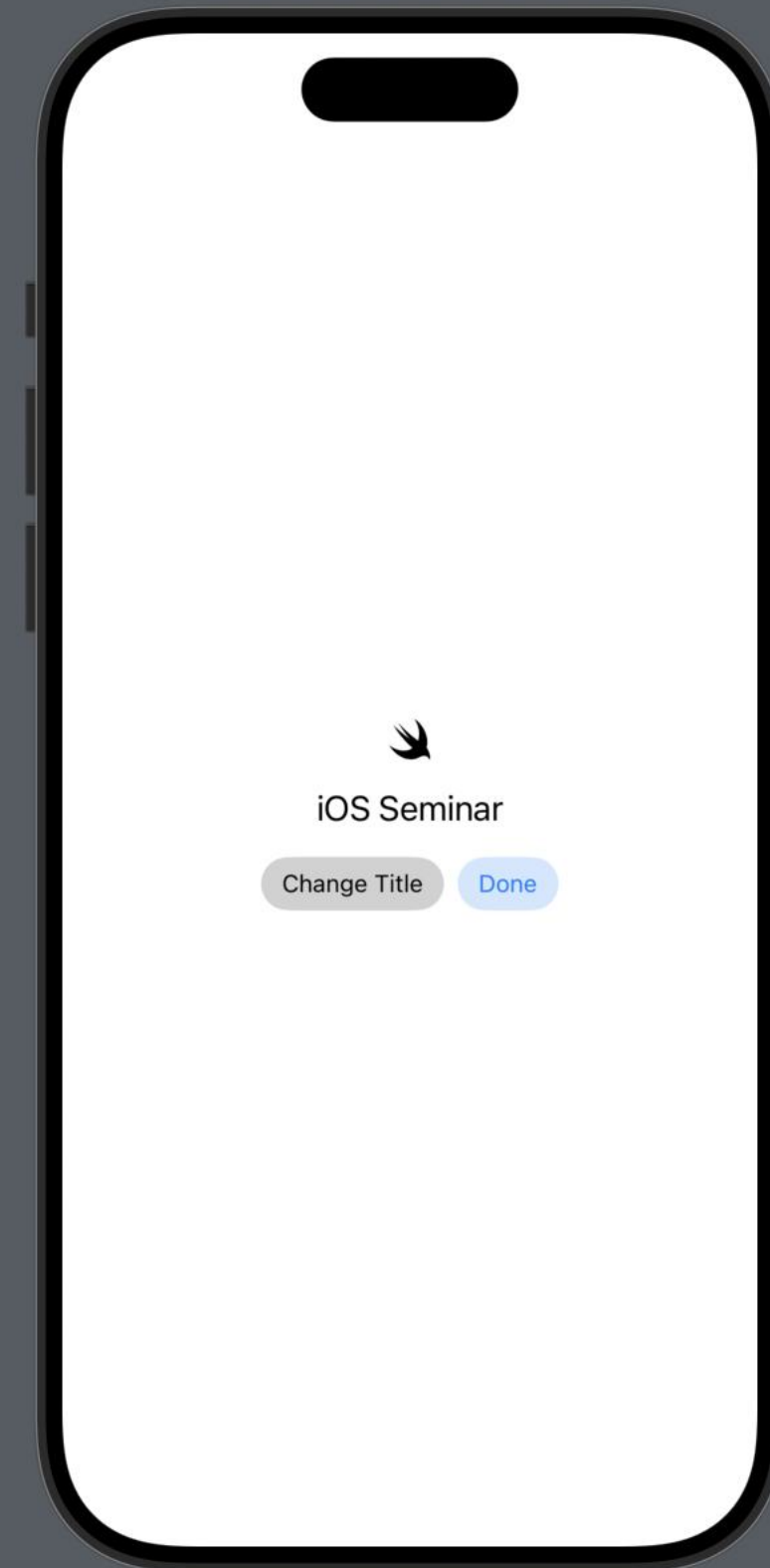
title의 @ObservationIgnored를  
지운다면?

```
import SwiftUI

struct ObservableExample: View {

    var seminar = ObservableSeminar()

    var body: some View {
        VStack(spacing: 16) {
            Spacer()
            Image(systemName: "swift")
                .resizable()
                .frame(width: 24, height: 24)
            Text(seminar.title)
                .font(.system(size: 20))
                .strikethrough(seminar.isDone)
            HStack {
                Button {
                    seminar.changeSeminarTitle()
                } label: {
                    Text("Change Title")
                }
                .tint(.black)
                .clipShape(Capsule())
                Button {
                    seminar.completeSeminar()
                } label: {
                    Text("Done")
                }
                .tint(.blue)
                .clipShape(Capsule())
                .disabled(seminar.isDone)
            }
            .buttonStyle(.bordered)
            .font(.system(size: 14))
            Spacer()
        }
    }
}
```



## Observation은 기존의 방식과 완전히 동일한가?

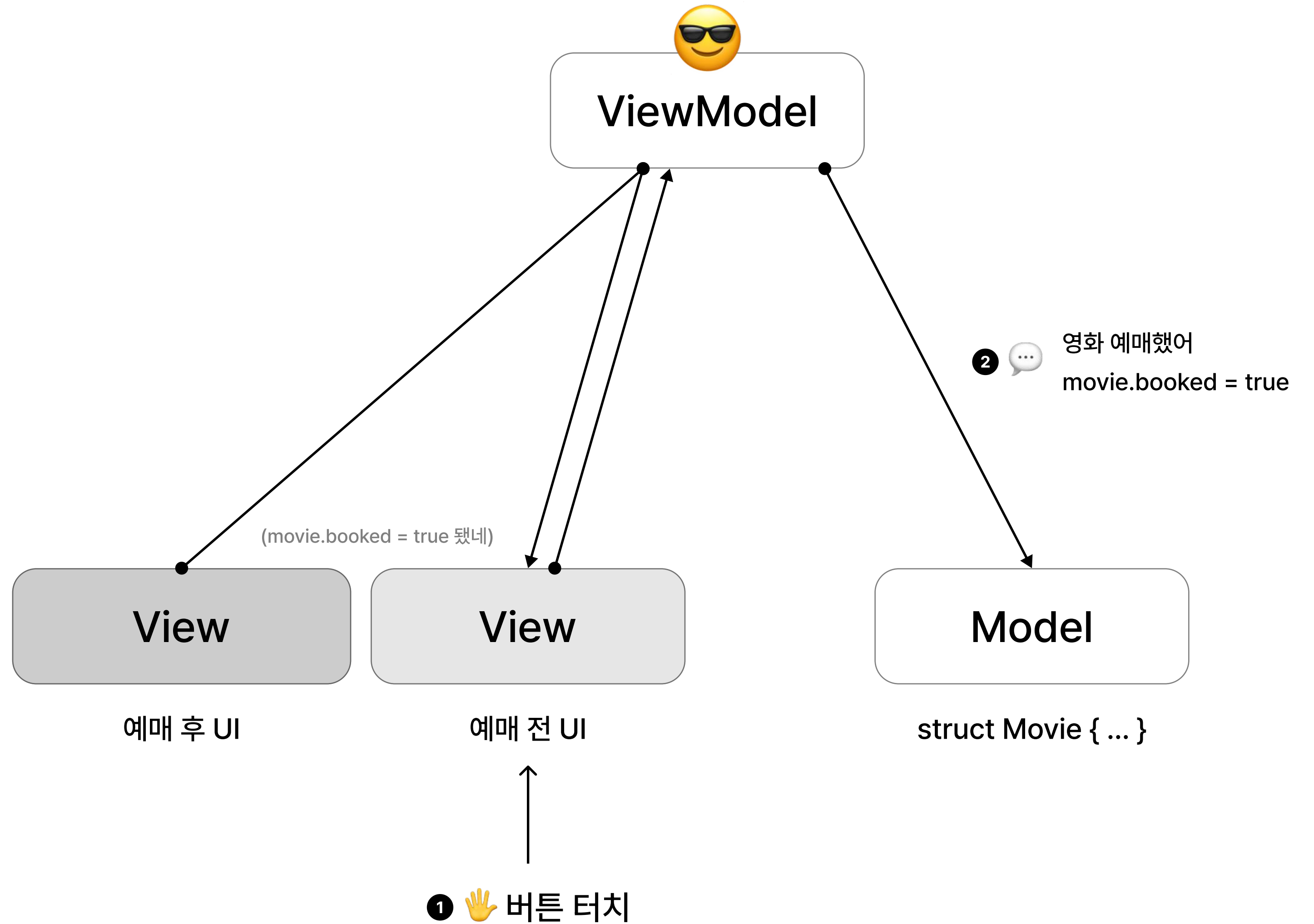
- @Observable을 이용하는 경우
  - View 업데이트: (1) Observable한 속성이 바뀌고 (2) View의 `body`가 그 속성을 직접적으로 읽는 경우  
→ View가 사용하지 않는(읽지 않는) 값이 업데이트되면, View는 변하지 않는다
- ObservableObject + @Published를 이용하는 경우
  - View 업데이트: ObservableObject의 @Published가 붙은 값에서 어느 것이든 바뀌는 경우  
→ View가 사용하지 않는(읽지 않는) 값이라도 업데이트가 이뤄지는 경우 View도 함께 변한다
- 관련 내용



# MVVM 적용

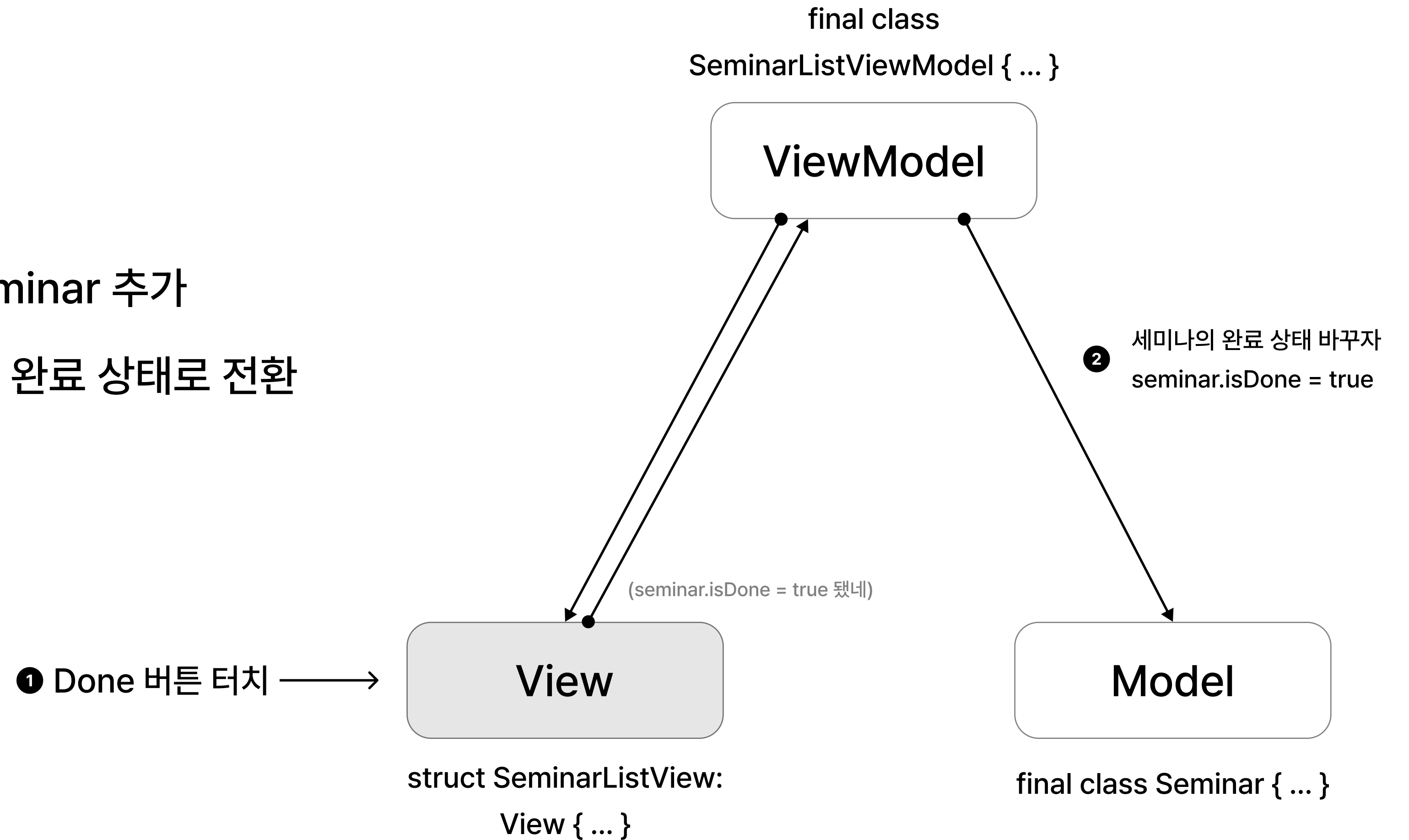
## MVVM (Recap)

- Model - View - ViewModel
- ViewModel이 바뀌면 View도 바뀐다



## 간단한 ViewModel 실습

- Seminar 목록을 보여주는 View
  - Add 버튼을 누르면 새로운 Seminar 추가
  - Done 버튼을 누르면 Seminar 완료 상태로 전환



# SwiftUI+Data Managing

```
@Observable final class Seminar: Identifiable {
    let title = "iOS Seminar"
    let session: Int
    var isDone = false

    init(session: Int, isDone: Bool = false) {
        self.session = session
        self.isDone = isDone
    }

    func completeSeminar() {
        isDone = true
    }
}
```

```
import SwiftUI

@main
struct Seminar4DemoApp: App {

    @State private var viewModel = SeminarListViewModel()

    var body: some Scene {
        WindowGroup {
            SeminarListView(viewModel: viewModel)
        }
    }
}
```

```
import Foundation
import Observation

@Observable final class SeminarListViewModel {
    var seminarList: [Seminar] = [
        .init(session: 0, isDone: true)
    ]

    @ObservationIgnored private var lastSession = 0

    func addNewSeminar() {
        lastSession += 1
        seminarList.append(.init(session: lastSession))
    }

    func complete(session: Int) {
        if let idx = seminarList.firstIndex(where: { $0.session == session }) {
            seminarList[idx].completeSeminar()
        }
    }

    func resetSeminarList() {
        seminarList = [.init(session: 0, isDone: true)]
        lastSession = 0
    }
}
```



## ObservableObject

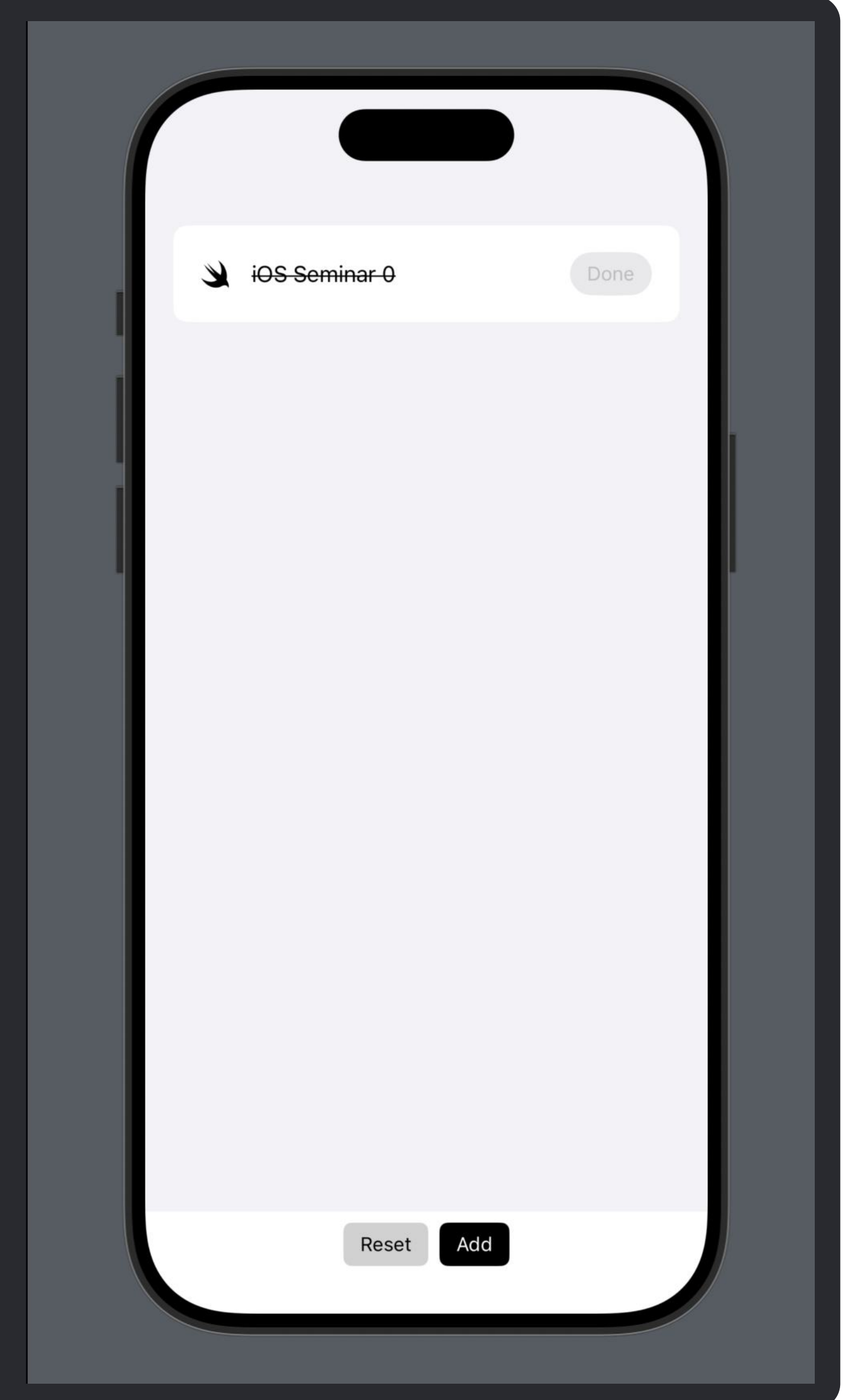
이용하는 버전도 비슷합니다

```
import SwiftUI

struct SeminarListView: View {

    var viewModel: SeminarListViewModel

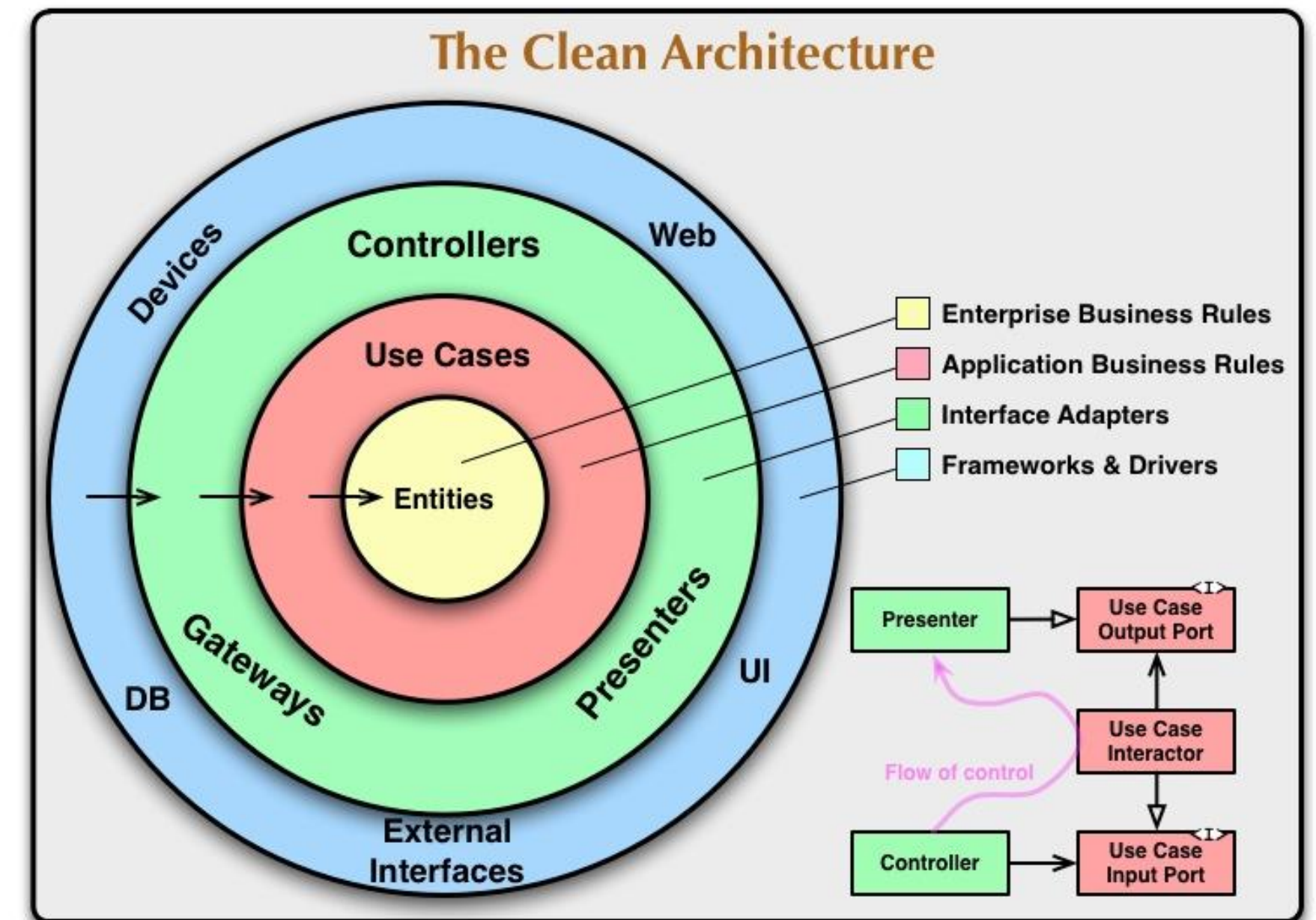
    var body: some View {
        VStack {
            List(viewModel.seminarList) { seminar in
                HStack(spacing: 16) {
                    swiftImage()
                    Text(seminar.title + " \ \(seminar.session)")
                        .font(.system(size: 16))
                        .strikethrough(seminar.isDone)
                    Spacer()
                    Button {
                        viewModel.complete(session: seminar.session)
                    } label: {
                        Text("Done")
                            .font(.system(size: 14))
                    }
                        .modifier(CapsuleButtonModifier())
                        .disabled(seminar.isDone)
                }
                .listRowSeparator(.hidden)
                .padding(.vertical, 8)
            }
            HStack {
                smallButton(label: "Reset", style: .bordered) {
                    viewModel.resetSeminarList()
                }
                smallButton(label: "Add", style: .borderedProminent) {
                    viewModel.addNewSeminar()
                }
            }
        }
    }
}
```



# Clean Architecture

## 앞에서 배운 ViewModel 만으로는 부족하다

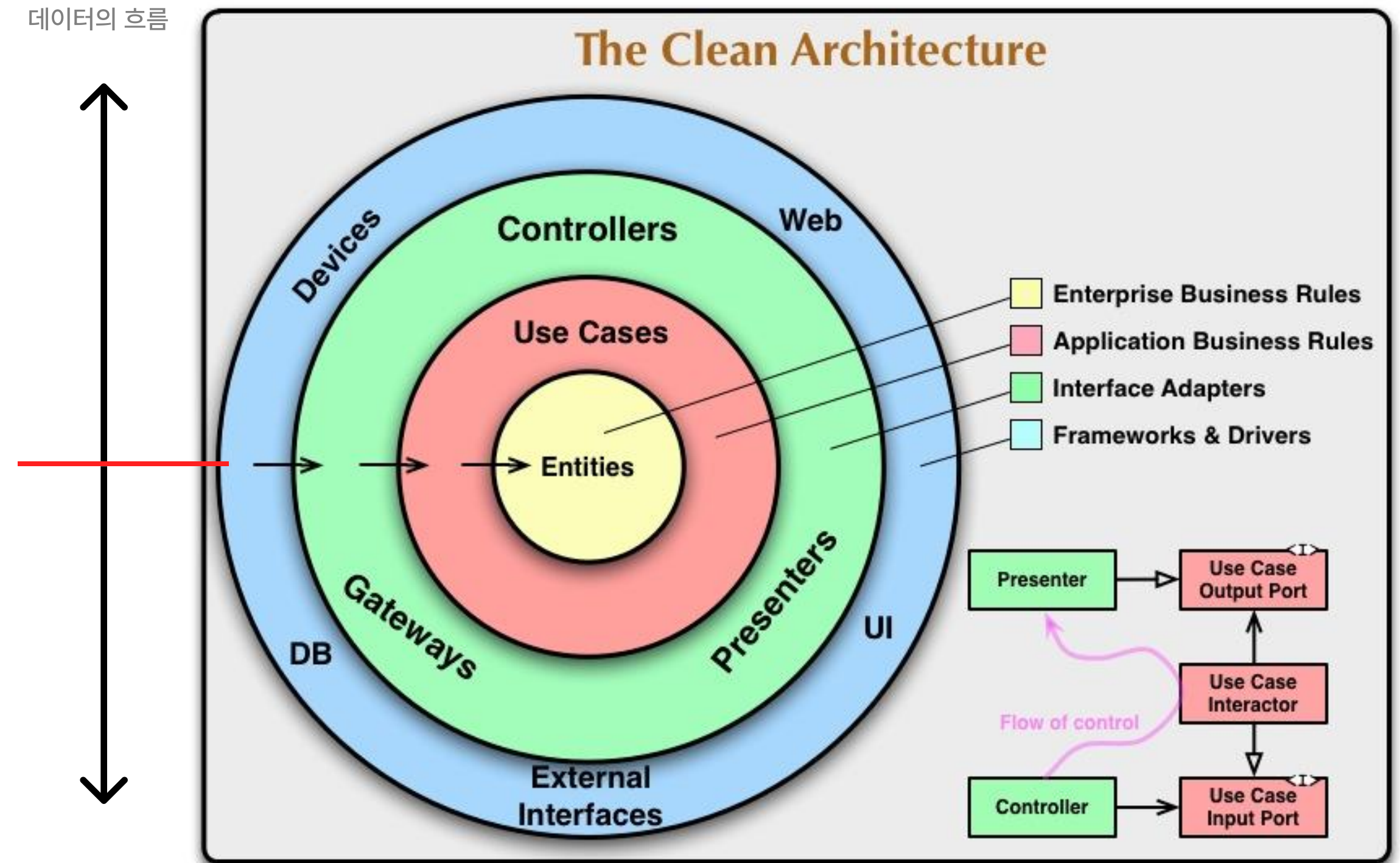
- 네트워크 통신은 누가 담당해야 할까?
- UserDefaults 등 로컬 저장소에 접근하는 역할은 누가 담당해야 할까?
- Clean Architecture
  - 이미 들어봤을 수도 있고, 들어보지 않았다면  
앞으로 많이 들을 키워드
  - 책 "Clean Code"의 저자 로버트 마틴  
(Robert C. Martin)이 제안한 시스템 architecture





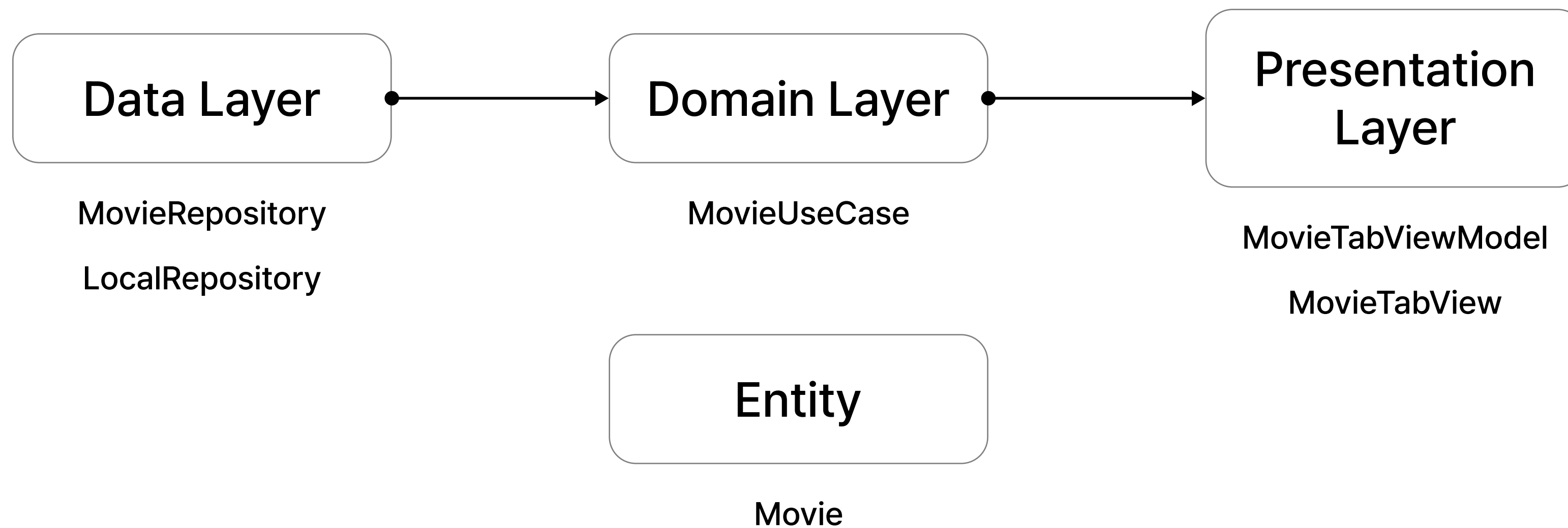
## Presentation - Domain - Data

- Presentation Layer
  - View, ViewModel 등
- Domain Layer
  - Presentation Layer과 Data Layer 사이에 위치
  - 원하는 데이터를 얻기 위한 가공 로직
  - Use Case 등
- Data Layer
  - 네트워크 통신, 로컬 통신 등을 담당
  - Repository, Data Source 등





## Presentation - Domain - Data



- `MovieTabView`는 `MovieRepository`의 존재를 모른다
  - 과제2처럼 `ViewController`가 `movieRepository.fetchPopularMovie()` 호출하는 행위 ❌



## Assignment 4

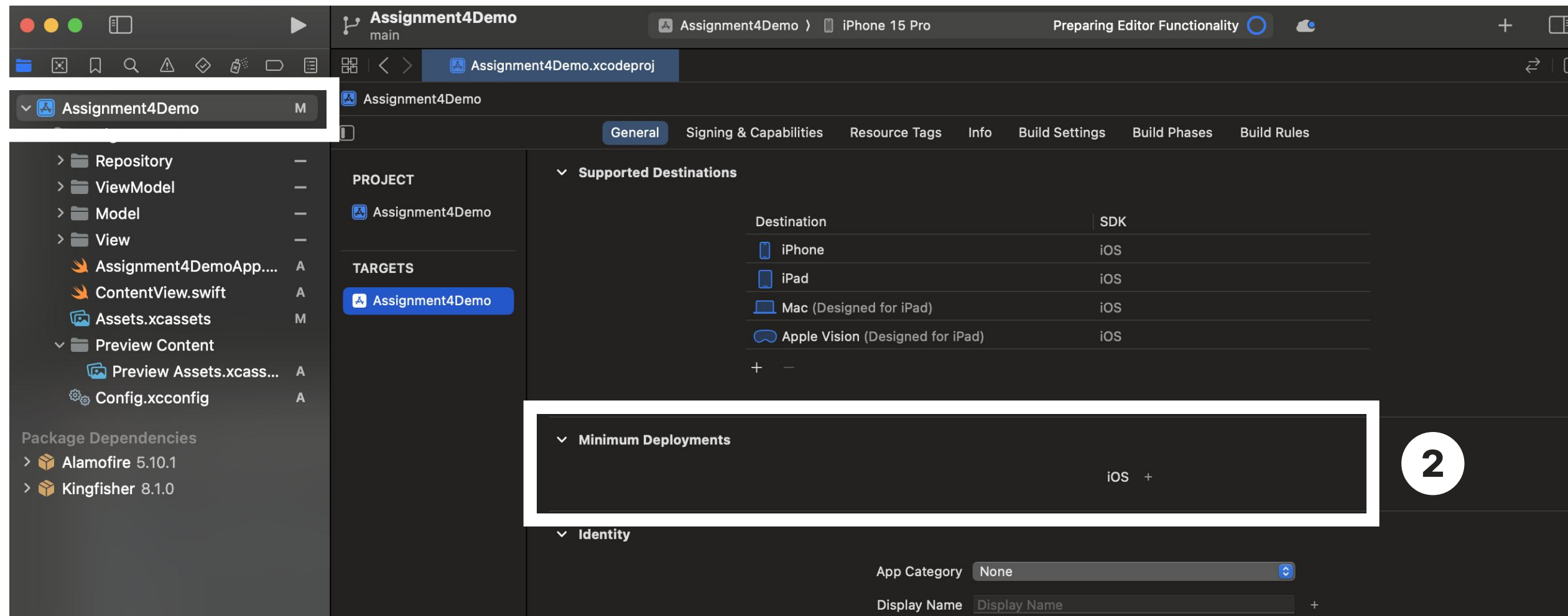
# 과제 4 안내

## 영화 예매 앱 만들기 (업그레이드)

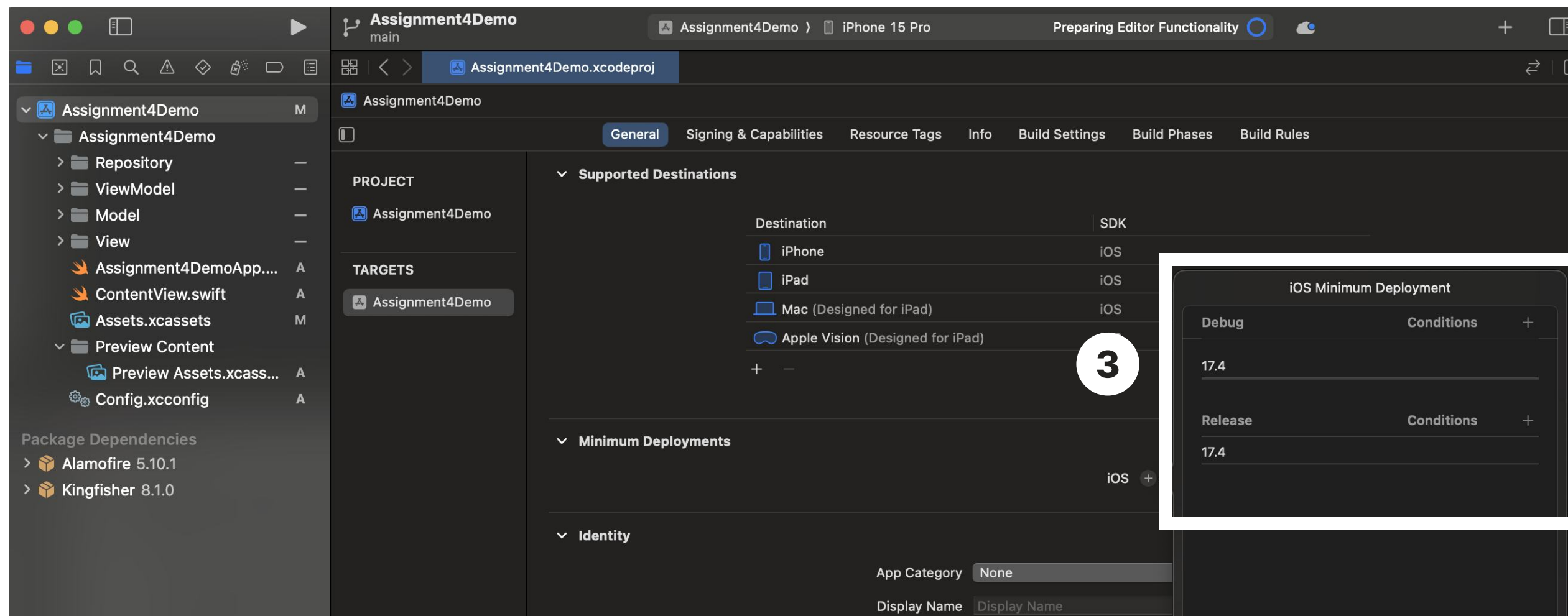
- 과제 4 디자인 링크(Figma)
- 과제 4 스켈레톤 코드 링크(Github)
- 과제 목표
  - SwiftUI를 이용하여 MVVM 패턴 적용
- 마감: 12월 1일 일 오전 8시

## 과제 4 세팅 (1)

1

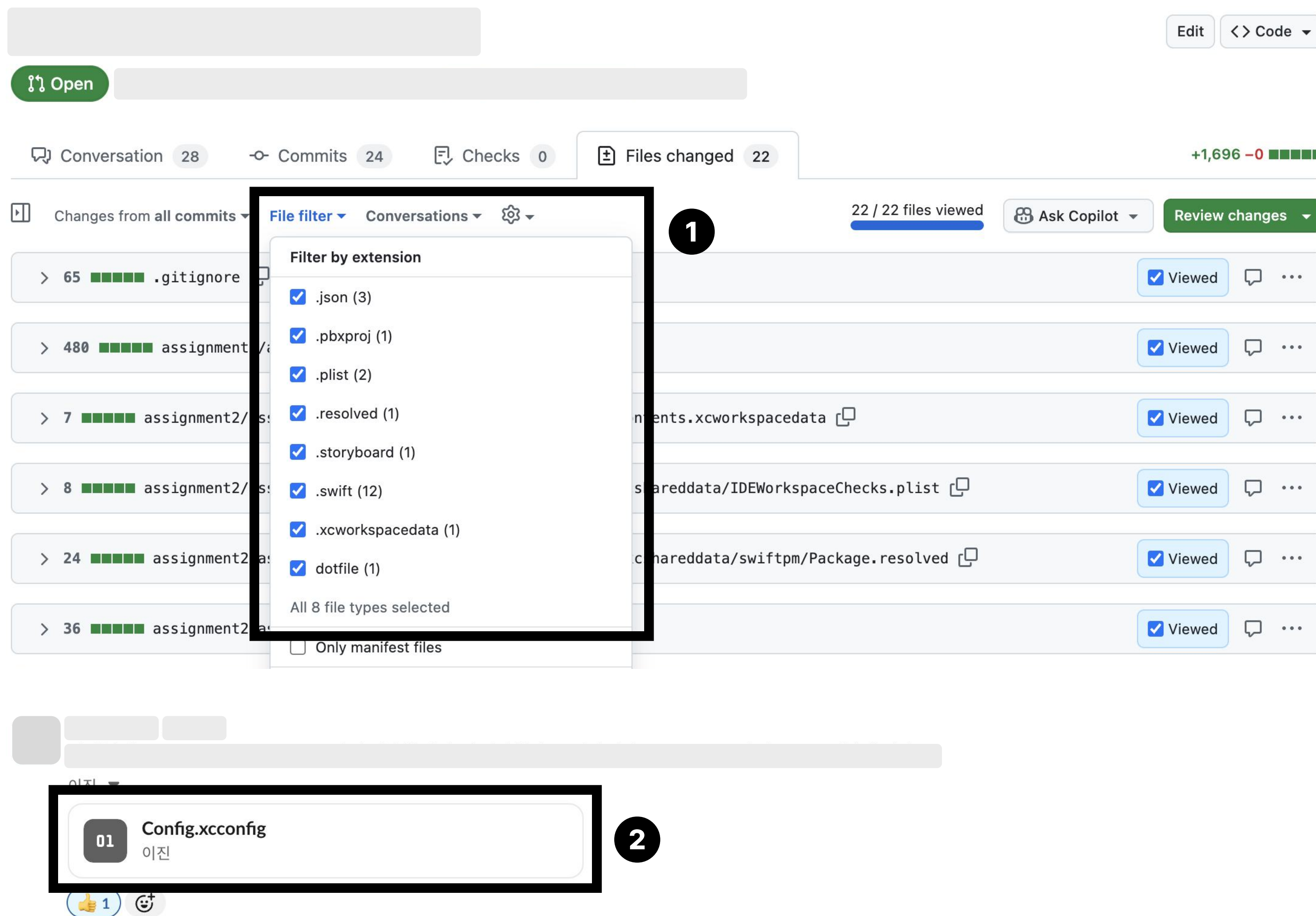


- Minimum Deployments 설정
  - 17.0 이상 18.0 미만으로 설정해 주세요
  - 설정 누락 시 스펙 구현 여부와 무관하게 탈락





## 과제 4 세팅 (2)



1

2

- Configuration 파일 생성 및 전달
  - Github에서 Files changed를 통해 Api Key 및 Access Token이 업로드되지 않았는지 확인
- `.xcconfig` 확장자를 가진 파일을 DM으로 전송
- 과제2에서 사용한 값과 동일하거나 이미 과제 2에서 올바르게 파일을 보냈어도 다시 전송합니다
- 파일을 전송하지 않거나, 일반 텍스트만 전송 시 스펙 구현 여부와 무관하게 탈락

## 과제 상세 스펙

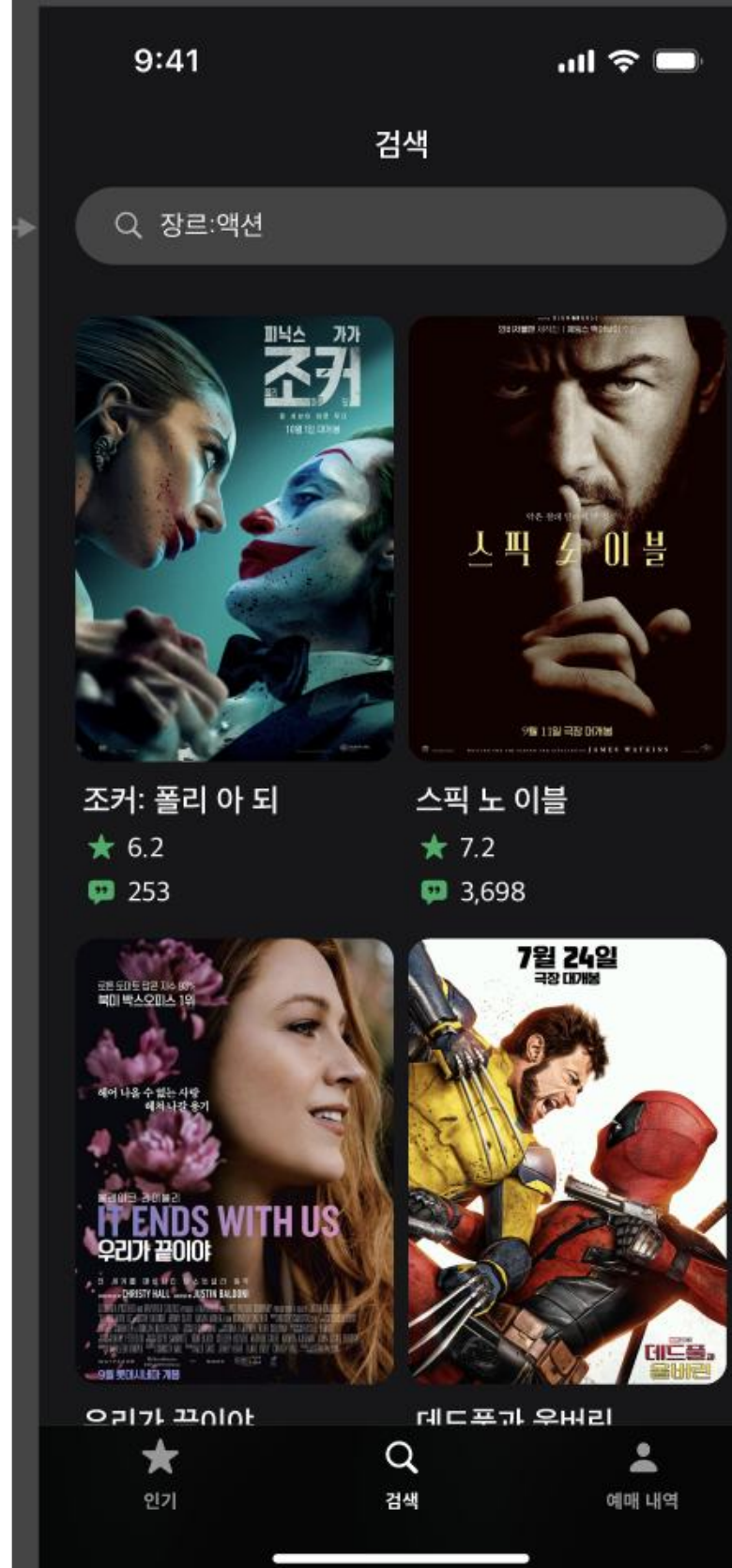
- 전체
  - Text, Color, Constraint, Corner Radius 등은 피그마에 주어진 값을 사용하며, 누락된 부분이 있다면 슬랙에서 질문 부탁드립니다
  - 상세 디자인 스펙은 피그마에 텍스트로 작성되어 있습니다
  - **MVVM 구조로 구현합니다**
  - Clean Architecture를 필수적으로 적용할 필요는 없으나, 제공되는 스켈레톤 코드는 Clean Architecture를 가정합니다
    - 스켈레톤 코드의 사용 여부는 과제 채점에 영향을 미치지 않습니다
    - 스켈레톤 코드는 과제 구현에 최소한의 도움만을 주고 있으므로 자유롭게 수정하여 사용해 주시기 바랍니다
- Font
  - SF Pro / Apple SD Gothic Neo는 Apple의 시스템 폰트이므로 과제 진행 시 size, weight, color만 고려합니다
- Alert
  - 별도의 디자인이 필요하지 않습니다. 문구와 동작만 알맞게 설정합니다

## 과제 상세 스펙 (2)

- 검색 탭
  - 제목 검색에서 사용하는 API ([링크](#))
  - 장르 검색에서 사용하는 API ([링크](#))

접두어 "장르:"가 붙는 경우에만 장르 검색  
이외의 경우는 제목 검색

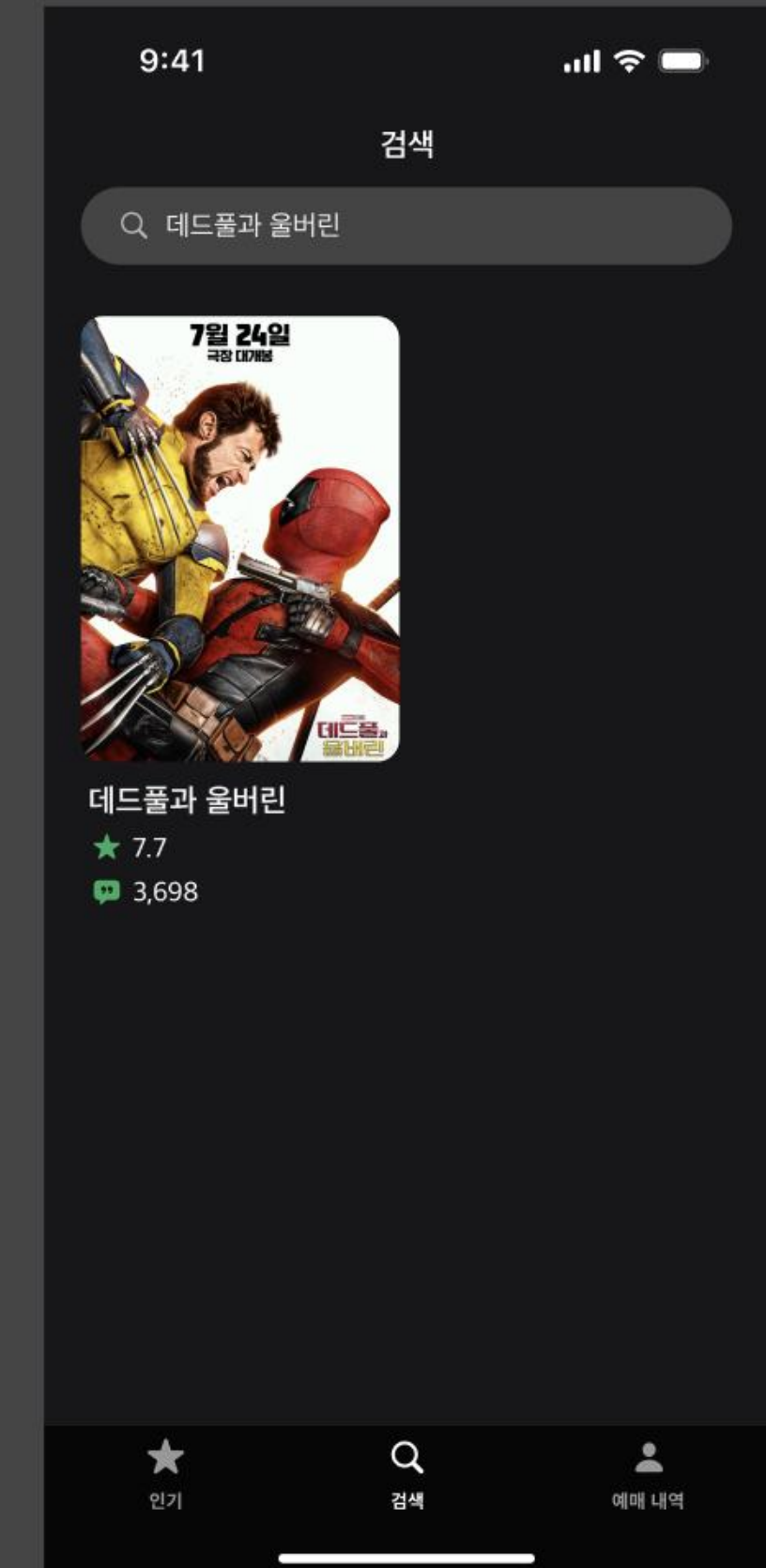
Search\_Success\_Tag



\*무한 scroll 가능해야 합니다

\*cell을 눌러도 상세 페이지로 이동 ❌

Search\_Success\_Title



\*무한 scroll 가능해야 합니다

\*cell을 눌러도 상세 페이지로 이동 ❌



## 과제 Bonus 스펙

- 과제 4 기본 스펙은 “예매 내역” 탭을 생성하고, 올바른 제목을 설정하는 것  
까지만 포함합니다
  - 기본 스펙에서 `MovieDetailView`의 예매 버튼 동작
    - 진입 시 항상 예매 전 상태 (.mainGreen)
    - 누르는 경우 alert를 띄우고, “확인”을 선택하면 예매 후 상태로 전환
- 과제 4 보너스 스펙 : 예매 내역 저장
  - 과제 2에서처럼 영화의 예매 여부를 저장하여 “예매 내역” 탭에서  
보여주고, 앱을 껐다가 다시 실행하는 경우에도 유지되도록 합니다
- 보너스 스펙에서 `MovieDetailView`의 예매 버튼 동작
  - 진입 시 예매 여부에 따라 다른 상태로 보여줍니다
  - 예매가 한번 완료되면 예매 내역에서 삭제되기 전까지는  
“예매 후 상태”를 유지합니다

