iOS 세미나 1

2023 와플스튜디오 루키 세미나

2023.09.17, 박신홍

오늘 배울 내용

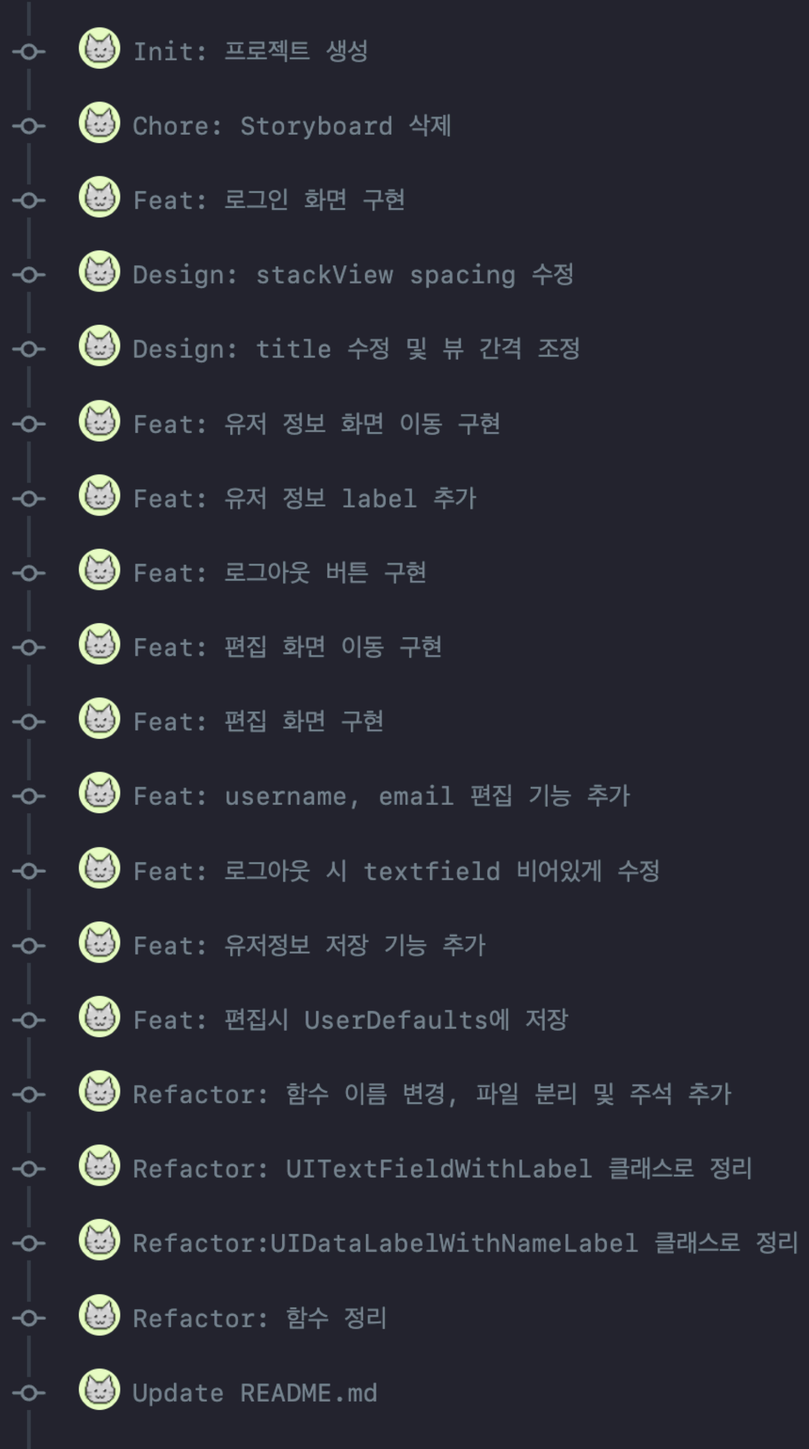
- 과제 0 공통 피드백 및 리뷰
  - SnapKit
  - UserDefaults, Codable
  - Lazy Var
- UITableView
  - 셀 재사용하기
  - Delegate 패턴
- 좋은, 구조화된 코드에 대해
  - MVC 패턴
  - MVVM 패턴
  - 클린 아키텍쳐 맛보기

과제는 어땠나요?

- 난이도
- 재미
- 질문
- 제출 방법

Git 사용 관련

- 커밋을 작업 단위로 잘 쪼개서 올려주세요.
- 커밋 메시지를 성의 있게 작성해주면 함께 협업하는 사람들이 좋아합니다.
- 좋은 예 →
  - 파일 or 화면 단위로만 쪼개도 충분합니다.



팁) SnapKit
AutoLayout을 더 편하게

```
NSLayoutConstraint.activate([
    redView.widthAnchor.constraint(equalToConstant: 100),
    redView.heightAnchor.constraint(equalToConstant: 100),
    redView.centerXAnchor.constraint(equalTo: view.centerXAnchor),
    redView.centerYAnchor.constraint(equalTo: view.centerYAnchor)
])
```

```
redView.snp.makeConstraints { make in
    make.width.height.equalTo(100)
    make.center.equalToSuperview()
}
```

팁) SnapKit
AutoLayout을 더 편하게

- 깃헙: https://github.com/SnapKit/SnapKit
- 사용법: https://snapkit.github.io/SnapKit/docs/
- 설치: Swift Package Manager
  - Xcode에서 클릭 몇 번이면 프로젝트에 추가 가능!

UserDefaults
데이터를 저장해보자

- UserDefaults: "User"들의 "Defaults" 설정값들을 저장하는 가벼운 로컬 스토리지
- Key-Value 형태로 저장
- 유저의 기본적인 설정 정보들을 저장하고 싶을 때

- 번외) iOS에서 데이터를 영속적으로 저장하는 방법 네 가지
  - UserDefaults: 가벼운 정보를 저장하고 싶을 때
  - CoreData: 복잡하고 많은 양의 데이터를 DB에 저장하고 싶을 때
  - FileManager: 이미지처럼 파일의 형태로 저장하고 싶을 때
  - KeyChain: 비밀번호, 암호화 키처럼 보안이 필요한 정보들을 저장하고 싶을 때

UserDefaults
예제 (feat. SNUTT)

```swift
struct TimetableDto: Codable {
    let year: Int
    let semester: Int
    let title: String
    let lecture_list: [LectureDto]
}
```

```swift
enum STDefaultsKey: String {
    case currentTimetable
    case timetableConfig
}
```

```swift
func set<T: Codable>(_: T.Type, key: STDefaultsKey, value: T?) {
    let encodedData = try? JSONEncoder().encode(value)
    storage.set(encodedData, forKey: key.rawValue)
}
```

```swift
func get<T: Codable>(_: T.Type, key: STDefaultsKey) -> T? {
    guard let existing = storage.object(forKey: key.rawValue) as? Data else {
        return nil
    }
    let decodedData = try? JSONDecoder().decode(T.self, from: existing)
    return decodedData
}
```

```swift
set(TimetableDto.self, key: .currentTimetable, value: codableTimetableStruct)
```

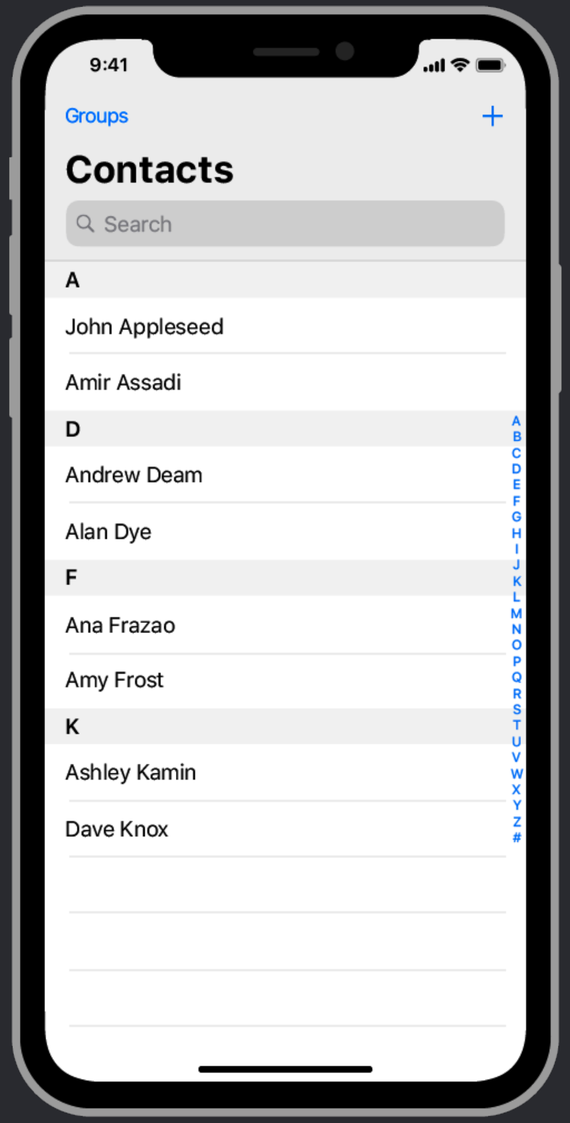
lazy var
사용을 권장합니다

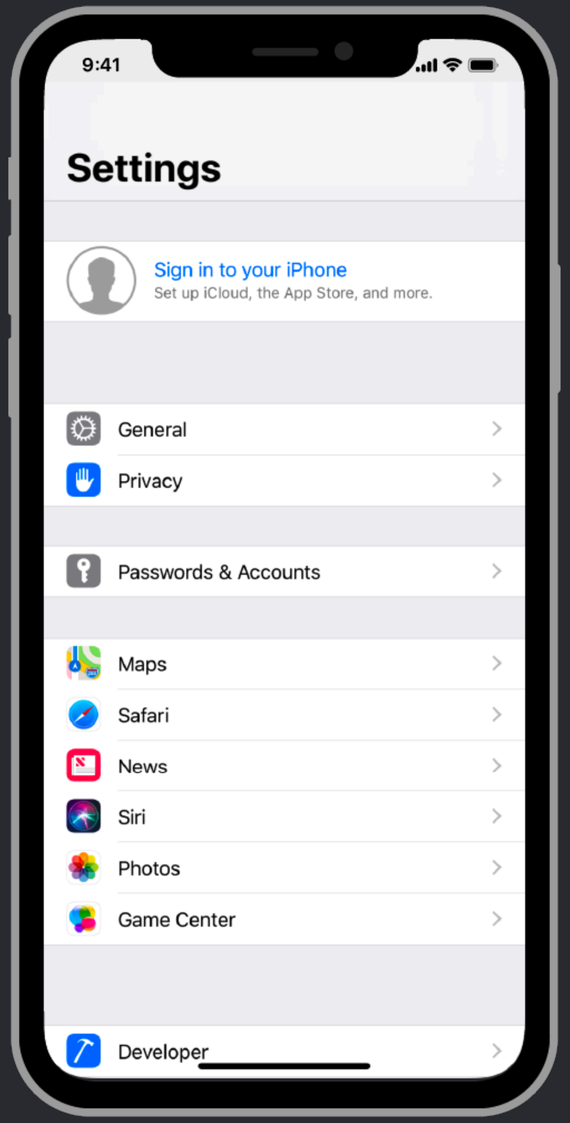
```
private lazy var loginButton = {
    var config = UIButton.Configuration.filled()
    config.baseBackgroundColor = .systemPink
    let btn = UIButton(configuration: config)
    btn.setTitle("로그인", for: .normal)
    btn.addTarget(self, action: #selector(loginButtonTapped), for: .touchUpInside)
    return btn
}()
```

- loginButton이 최초로 접근될 때 클로저가 실행되면서 객체 생성
- 이미 클래스가 초기화된 상태이므로, self 변수도 자유롭게 사용할 수 있음
- UI 컴포넌트 단위로 코드를 분리하면 코드를 깔끔하게 유지할 수 있음

UITableView
리스트를 만들어보자

- 오른쪽 화면을 만들 때 사용
- Row와 Section으로 구성
- UITableViewCell들의 집합



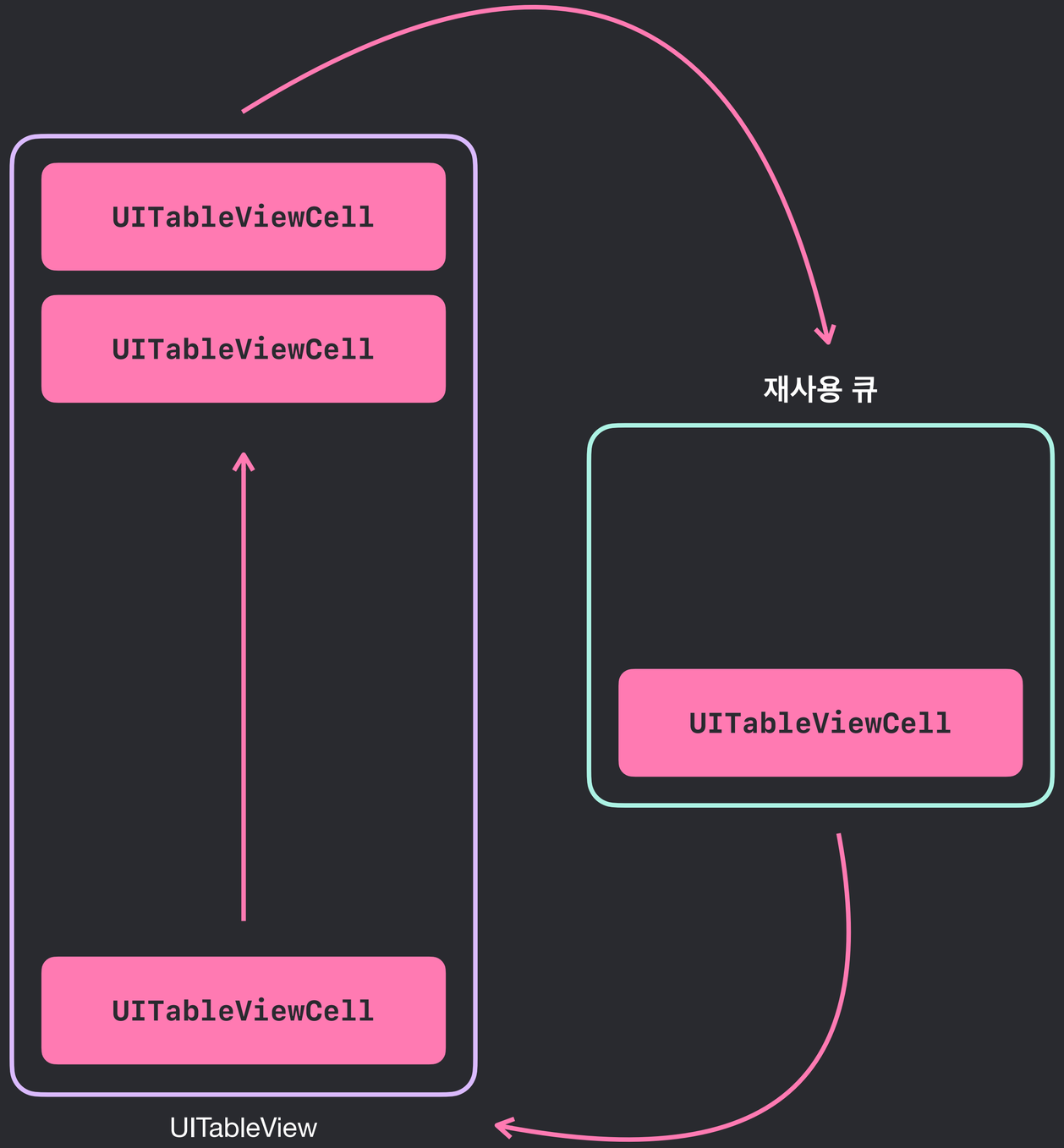


UITableView
셀 재사용하기

- 수만개의 데이터를 테이블 뷰에 표현하려면?
  - 데이터 개수만큼 UITableViewCell을 만들어야?
  - 그런데 UIView는 꽤 무거운 객체임
  - 수만개를 만들 시간(8ms...)도, 메모리도 없다.
- 사실 화면에 보이는 Cell은 몇 개 안됨
  - 그냥 보이는 것만큼만 만들어서 재사용하자!

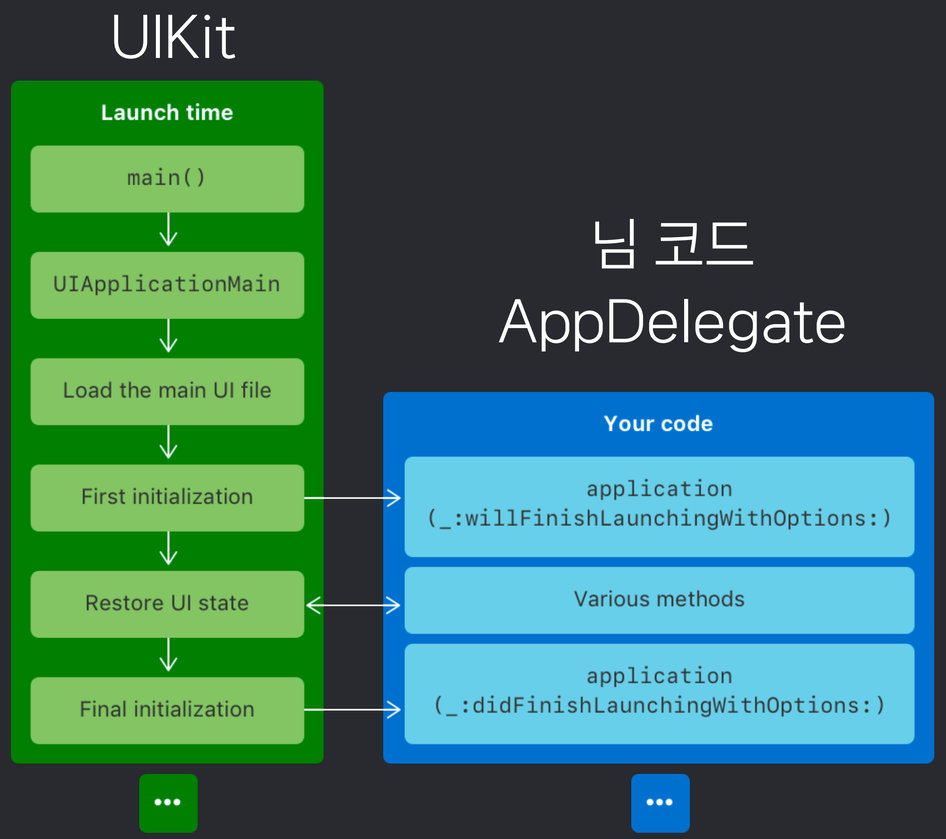
UITableView
셀 재사용하기

- 상수개의 Cell만 생성 → 시간, 메모리 절약
- 주의: 상태를 제대로 초기화해주지 않으면 재사용 전 정보가 그대로 나타날 수 있다.
  - `cf) prepareForReuse()`



Delegate 패턴
Recap

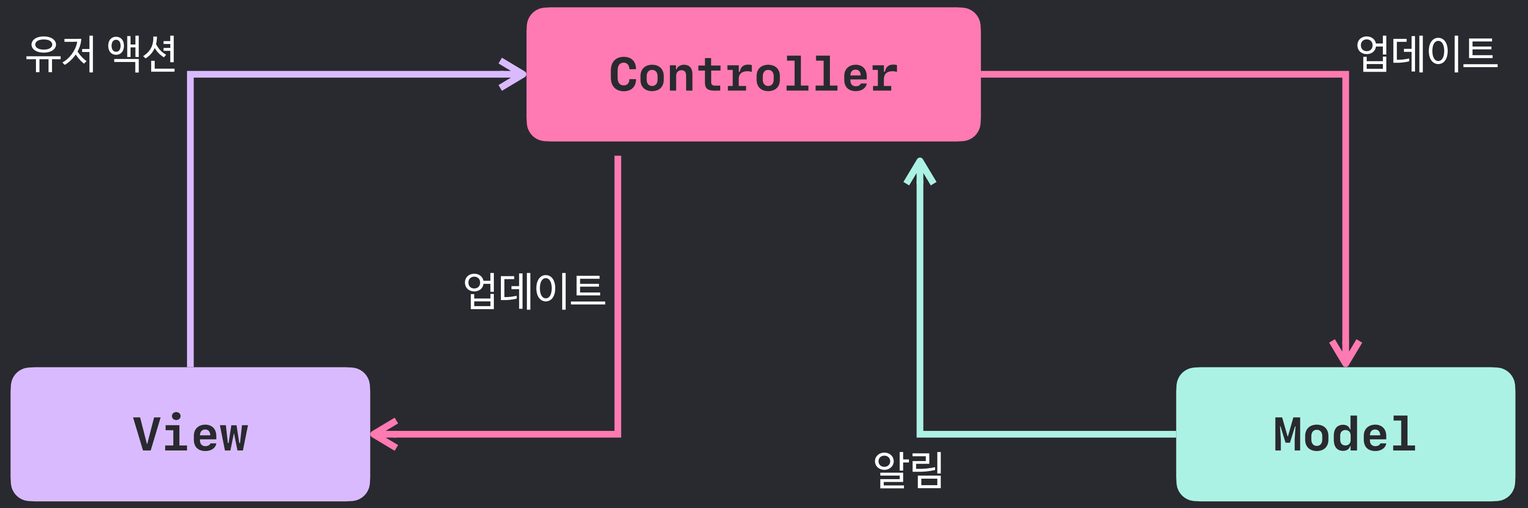
- UIKit은 프레임워크이다. 즉, 당신의 코드를 UIKit이 실행한다.
- 다르게 표현하면, UIKit은 중간중간 대리인(delegate)에게 어떤 코드를 실행해야 할지 "물어본다".
- 예시)
  - "앱이 수행이 완료되었을 때 뭘 해야 하나요?"
  - "테이블 뷰에 표시할 데이터는 몇개인가요?"
  - "테이블 뷰 5번째 행에는 어떤 정보를 보여주어야 하나요?"



- SnapKit 설치
- UITableView & Delegate 실습

MVC 패턴

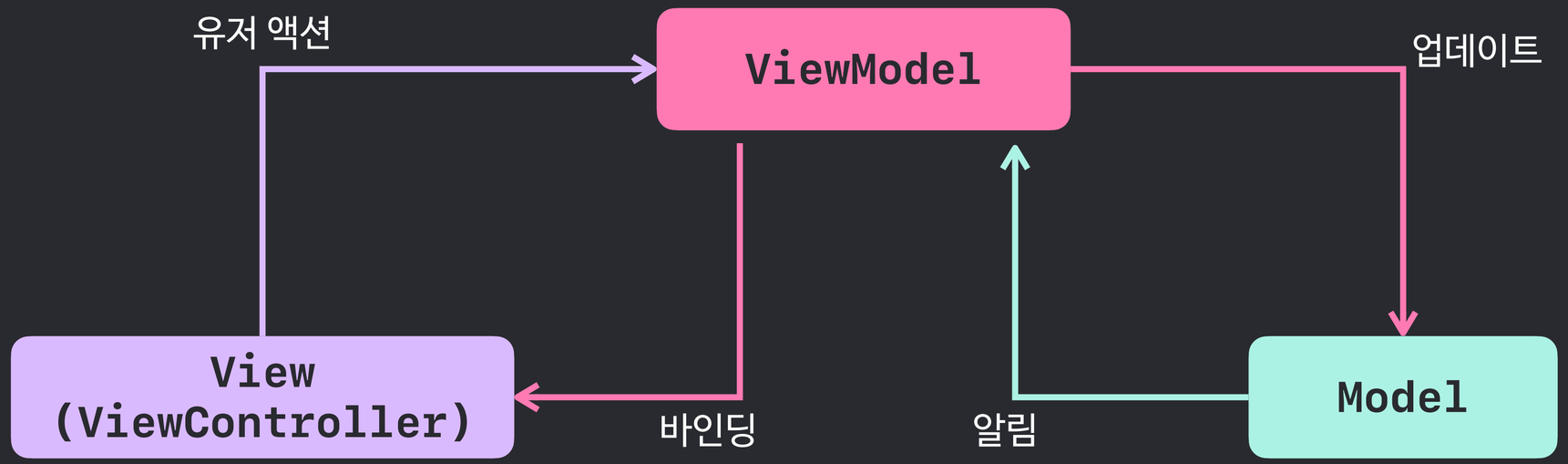
지금까지 했던 것을 MVC 패턴이라고 합니다



MVC 패턴

- UIKit으로 개발할 때 애플에서 사용을 권장하고 있는 패턴
  - https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html#//apple_ref/doc/uid/TP40008195-CH32-SW1
- UIViewController는 Controller의 대표적인 예시
- 그런데 MVC 패턴은 잘못 작성하기가 너무 쉽다.
  - UIViewController에서 데이터에 직접 접근하거나, 중요한 비즈니스 로직을 작성하는 경우
  - UIViewController 객체가 매우 난잡해지고(Massive View Controller), UI를 조작하는 코드와 섞여서 테스트가 불가능한 코드가 되어버린다.

MVVM 패턴
Model - View - ViewModel



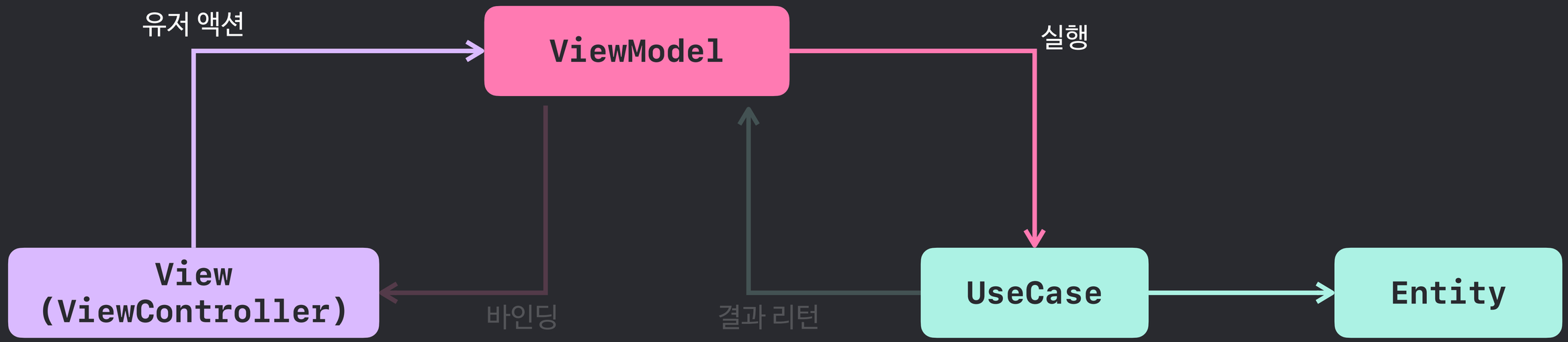
MVVM 패턴
Model - View - ViewModel

- 기존의 MVC 구조에서 ViewModel이 추가되면서 각 객체의 역할이 재편됨
  - View (+ ViewController): 오로지 View와 라이프사이클을 관리하는 객체. 유저의 눈에 보이는 것들(레이아웃 등)만 관리하고, 비즈니스 로직에는 전혀 관여하지 않는다.
  - ViewModel: 데이터를 UI에 뿌려주기 전, 전처리 로직을 담당한다.
  - Model: 데이터 구조의 정의, 데이터를 불러오는 로직 또는 복잡한 비즈니스 로직을 담당한다.

- 다 좋은데 Model의 비중이 너무 커보인다... 🤔 조금 더 쪼개보자.

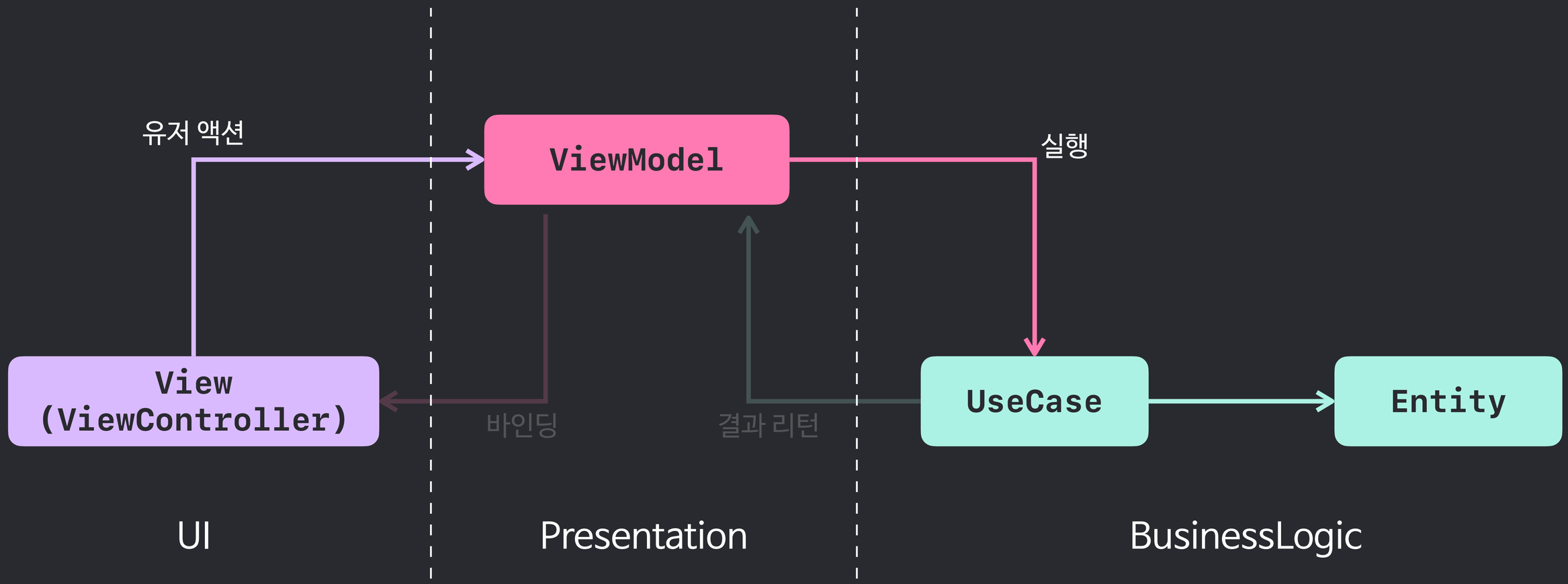
맛보기: 클린 아키텍쳐

MVVM 패턴의 최종(?) 진화 버전



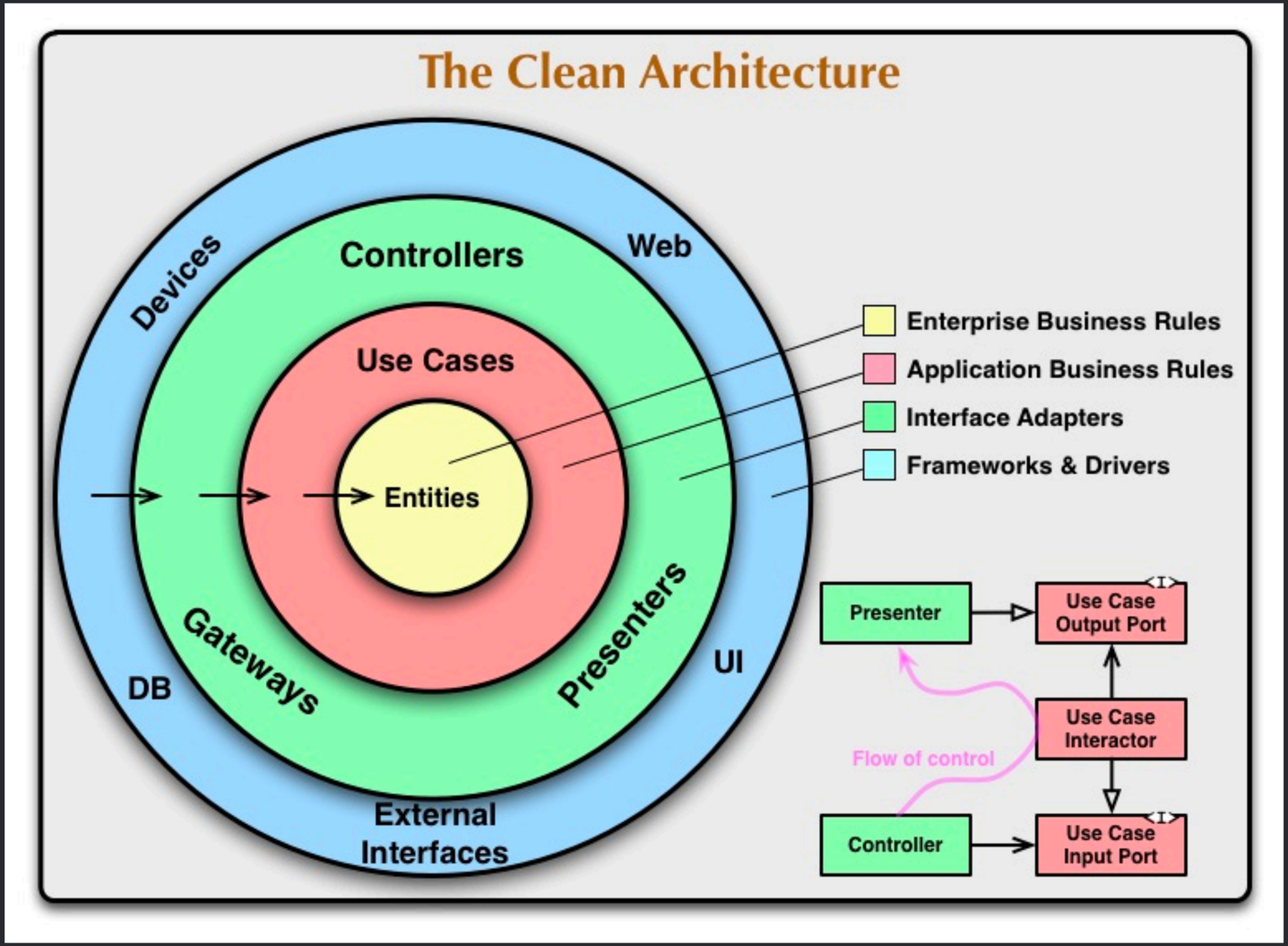
맛보기: 클린 아키텍쳐

MVVM 패턴의 최종(?) 진화 버전



클린 아키텍쳐

- 로버트 마틴 아저씨가 <클린 코드> 라는 책에서 처음으로 소개한 개념
- 프론트, 백엔드 도메인을 막론하고 널리 사용되는 개념
- 핵심은 딱 두 가지!
  - 소프트웨어는 여러 레이어로 나눌 수 있다.
  - 안쪽 레이어는 바깥 레이어에 대해 아무것도 알지 못한다.



- MVVM 패턴으로 리팩토링하기

세미나1 과제

기한: 10월 6일 (토) 밤 12시

1. iOS 미리 알림 클론코딩하기
    - 구체적인 스펙과 제출 방법은 세미나 레포에서 확인 가능합니다.

Q & A