# WEEK 1

윤사라

1. 자신이 터득한 오토레이아웃 구성 노하우

2. IB Outlet과 IB Action을 사용할때 주의할 점들

3. 자신이 자주 사용하는 앱의 여러 view controller에서 라이프 사이클에 따라 어떤 동작들이 이루어지고 있을지 상상해보기

4. 자신이 자주 사용하는 앱의 여러 뷰 컨트롤러에서 MVC디자인 패턴을 따라 어떻게 클래스가 구성되어 있을지 상상해보기

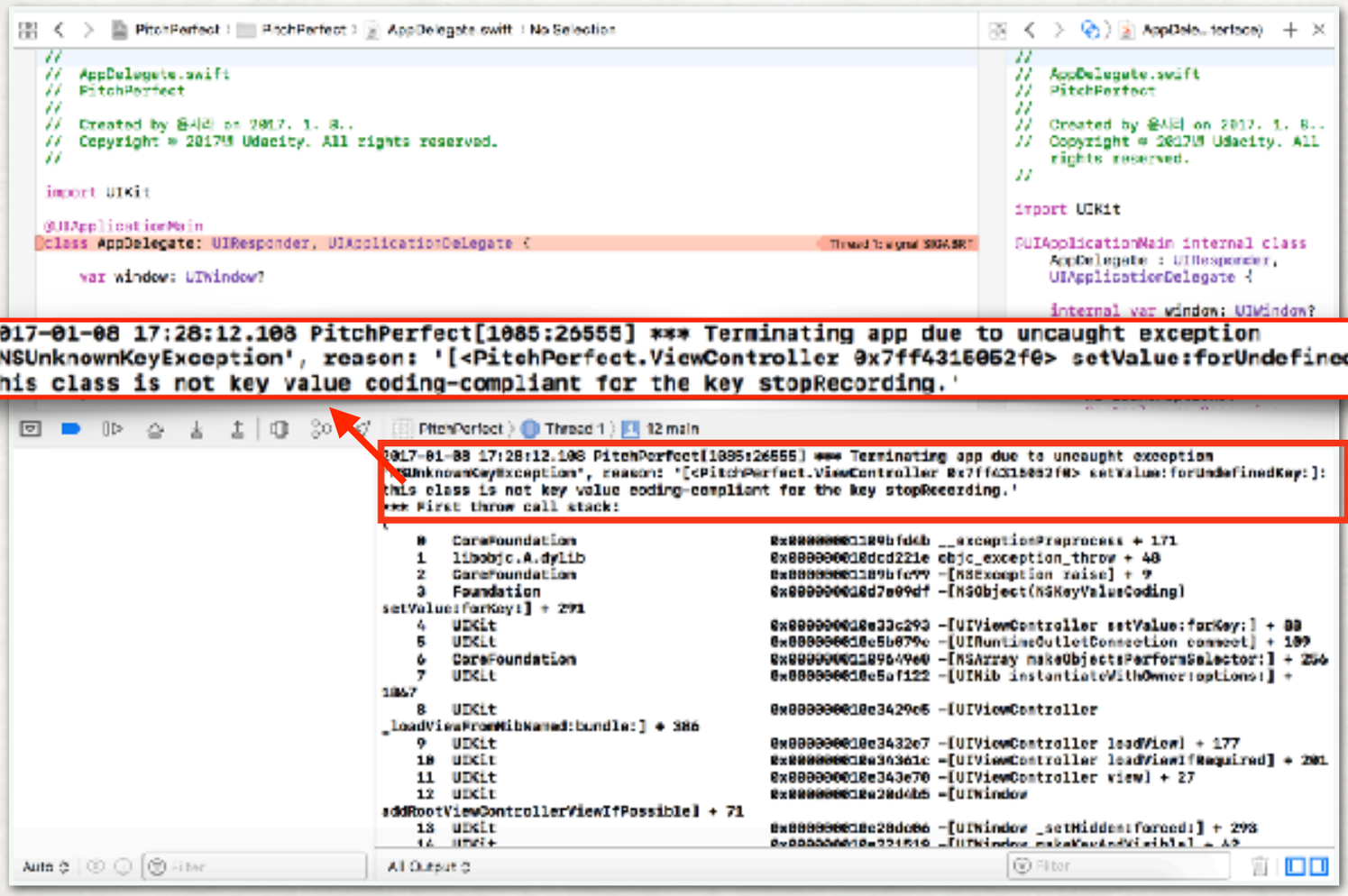
5. 프로그래밍에서 디자인 패턴이란 무엇인가, 디자인 패턴이 가지는 의미는 무엇인가

6. 네비게이션 컨트롤러의 동작은 자료구조의 형태에서 어떤 자료구조 방법론과 유사할까? iOS에서 네비게이션 컨트롤러를 사용하지 않고 화면을 전환하는 다른 방법에는 어떤 것들이 있을까?

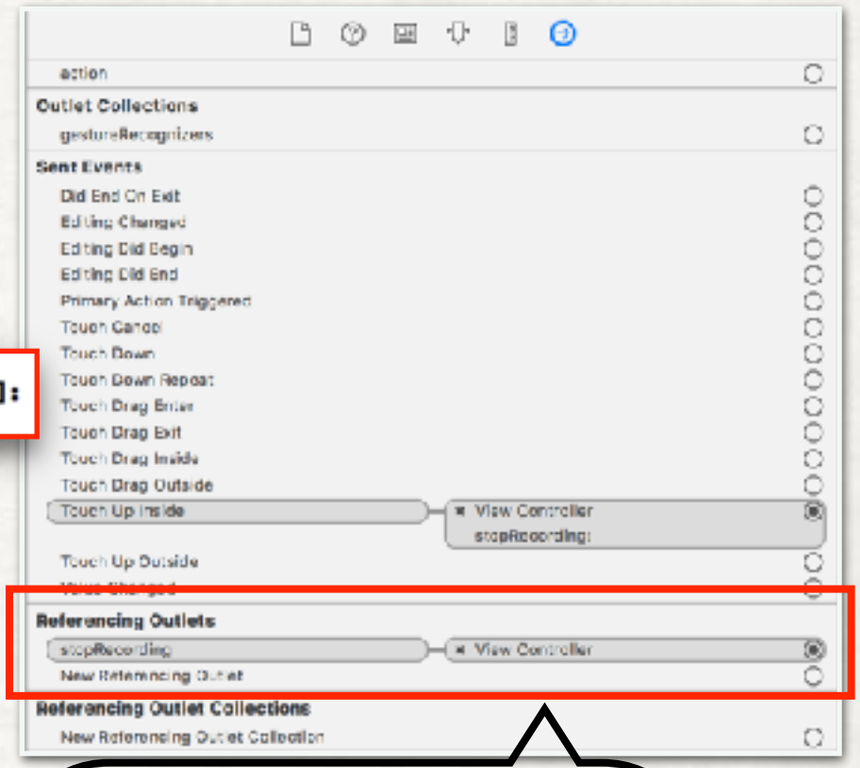
# @IB ACTION 사용시 주의할 점

1. @IBAction을 생성해야 하는 대신, 실수로 @IBOutlet을 생성한 후 이를 삭제했을 때, 연결정보가 남아있는 경우가 있어 커넥션 인스펙터에서 확인할 필요가 있음

*실수로 @IBOutlet을 생성한 후 이를 삭제한 후, @IBAction을 생성한 경우 발생하는 에러 메시지




*커넥션 인스펙터 확인

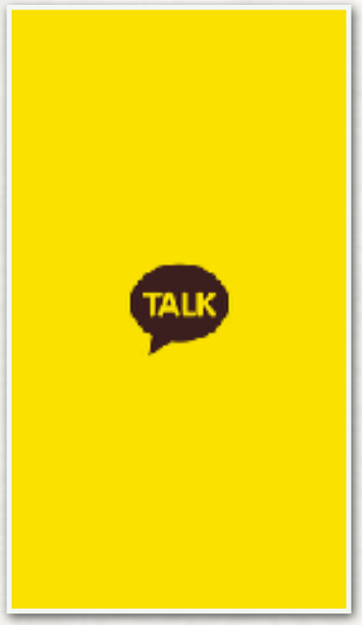


# @IB OUTLET 사용시 주의할 점

1. Storage 항목(Strong/Weak 타입) 주의
   - Strong 타입의 변수나 상수는 프로그램의 어느 곳에서도 더 이상 참조하지 않을 때 메모리에서 제거되지만, Weak 타입으로 선언된 변수나 상수는 그에 상관없이 시스템에서 임의로 메모리에서 제거 가능
   - Strong 타입 객체들끼리 상호 참조되는 일이 발생하는 경우, 어떤 경우에도 참조 카운트가 0이 되지 않으므로 애플리케이션이 실행되는 한 영원히 메모리에서 제거되지 않아 메모리 누수로 이어진다.
   - 메모리 관리 이슈로 인하여 Weak 타입으로 설정하면 시스템에 의해 임의로 제거가 가능하므로 순환되는 상호 참조로부터 벗어날 수 있음
2. 변수선언 시, 옵셔널 연산자인 !를 추가
   - @IBOutlet변수를 선언할 땐 그 변수의 타입이 옵셔널이 아니어야 한다는 제약조건이 있으므로 옵셔널 강제 해제 연산자를 붙여야 한다.

# VIEW CONTROLLER 생명주기_카카오 채널

Launch Screen

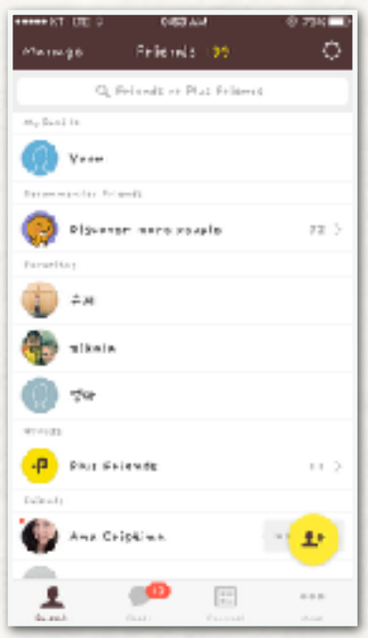



ViewDidLoad

Appearing

ViewWillAppear

Appeared



ViewDidAppear

Disappearing

ViewWillDisappear

Disappeared

ViewDidDisappear

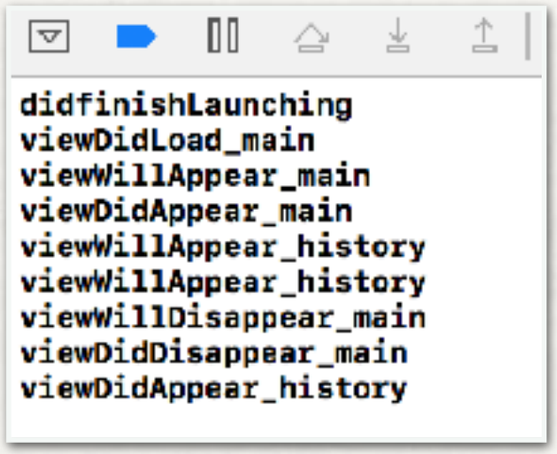
Appearing

ViewWillAppear

Appeared


ViewDidAppear

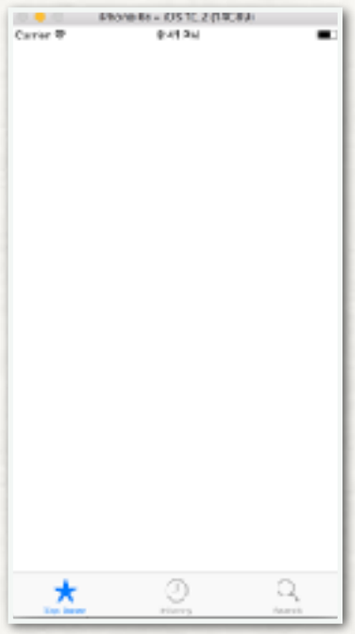
# VIEW CONTROLLER_로그찍어 확인



```
didfinishLaunching
viewDidLoad_main
viewWillAppear_main
viewDidAppear_main
viewWillAppear_history
viewWillAppear_history
viewWillDisappear_main
viewDidDisappear_main
viewDidAppear_history
```

didfinishLaunching

Appearing



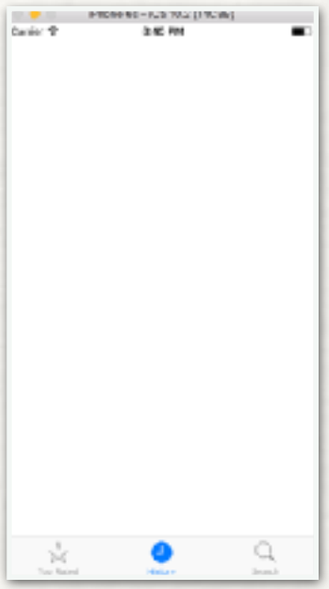
ViewDidAppear

ViewDidLoad → ViewWillAppear

History

ViewWillAppear

Appearing



ViewDidAppear

Main_TopRated

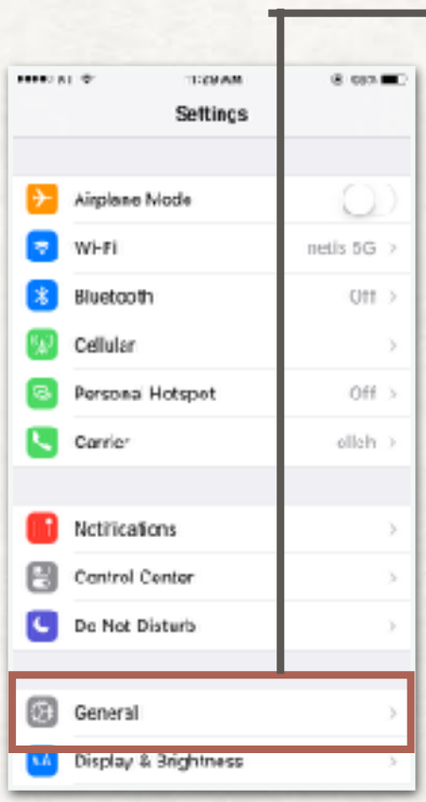
ViewWillDisappear

Disappearing

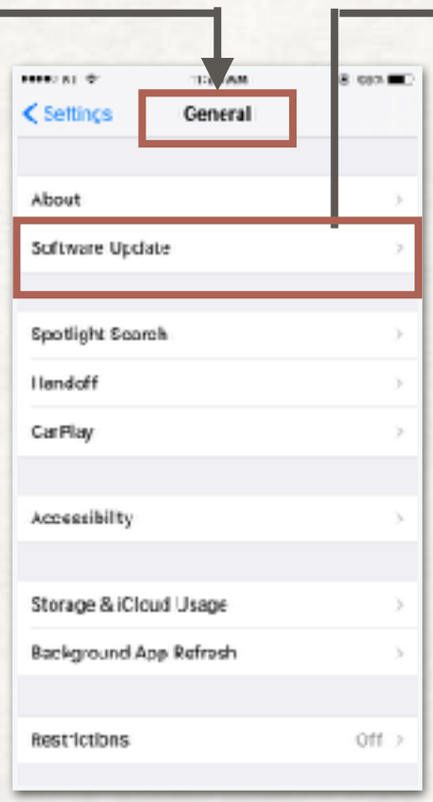
ViewDidDisappear

# 네비게이션 컨트롤러의 동작

- 네비케이션 컨트롤러의 동작은 스택 방식과 유사
  - 가장 아래에 있는 첫 번째 뷰 컨트롤러는 루트 뷰 컨트롤러이고, 최상위에 있는 마지막 뷰 컨트롤러는 현재 화면에 표시되고 있는 뷰 컨트롤러
  - 기본적으로 스택의 최상위 뷰 컨트롤러를 추가할 때는 pushViewController(_:animated:)메소드를 사용하며, 스택의 최상위 뷰 컨트롤러를 제거할 때에는 popViewControllerAnimated(_:)메소드 사용

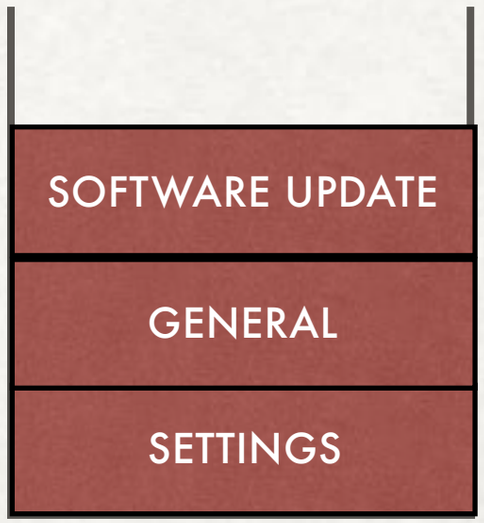



루트 뷰 컨트롤러



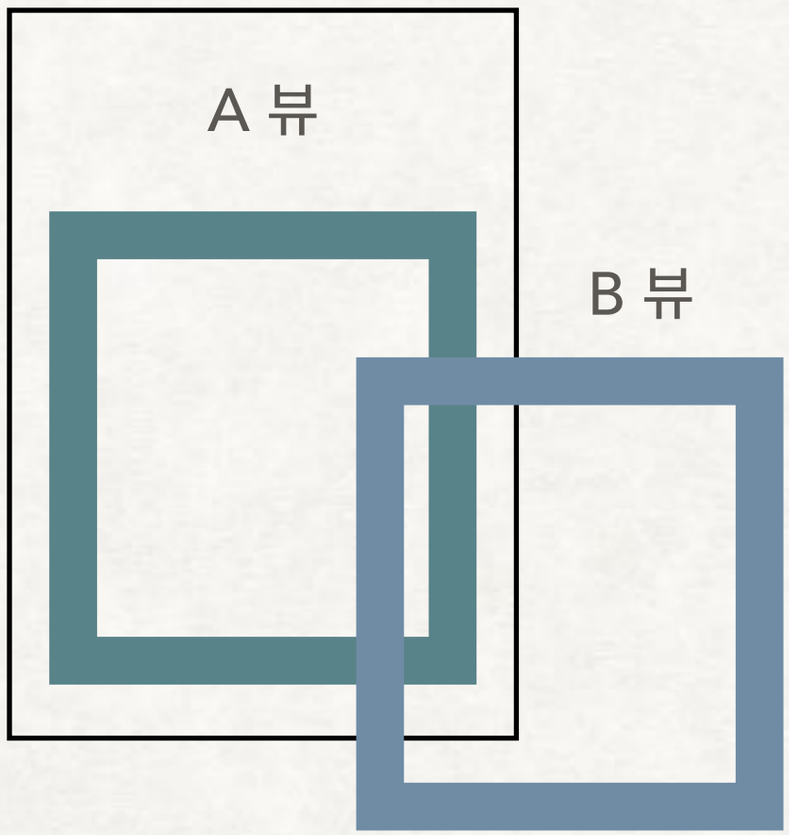

뷰 컨트롤러 1


뷰 컨트롤러 2



# 화면전환 방법_1

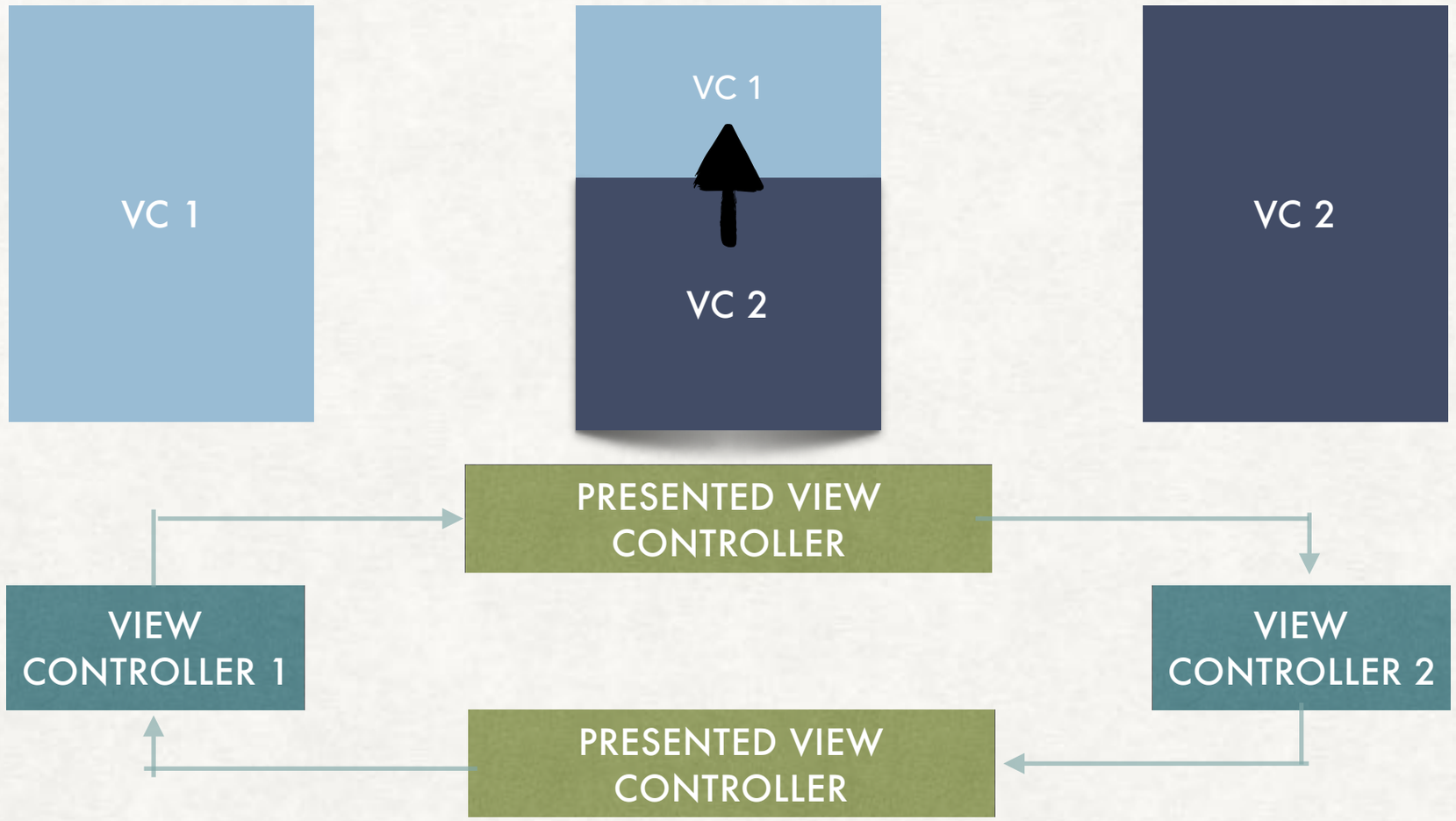
- 뷰 컨트롤러의 뷰 위에 다른 컨트롤러의 뷰를 가져와 바꿔치기 하기




- 뷰를 두 개 준비하고 상태에 따라 뷰를 교체, 뷰를 완전히 바꿔치기 할 수도 있지만, 때에 따라서는 기존 뷰 위에 다른 뷰를 덮는 방식을 사용
- 하나의 뷰 컨트롤러가 두 개 이상의 싱글 뷰를 관리하는 결과를 가져와 좋은 방법은 아님

# 화면전환 방법_2

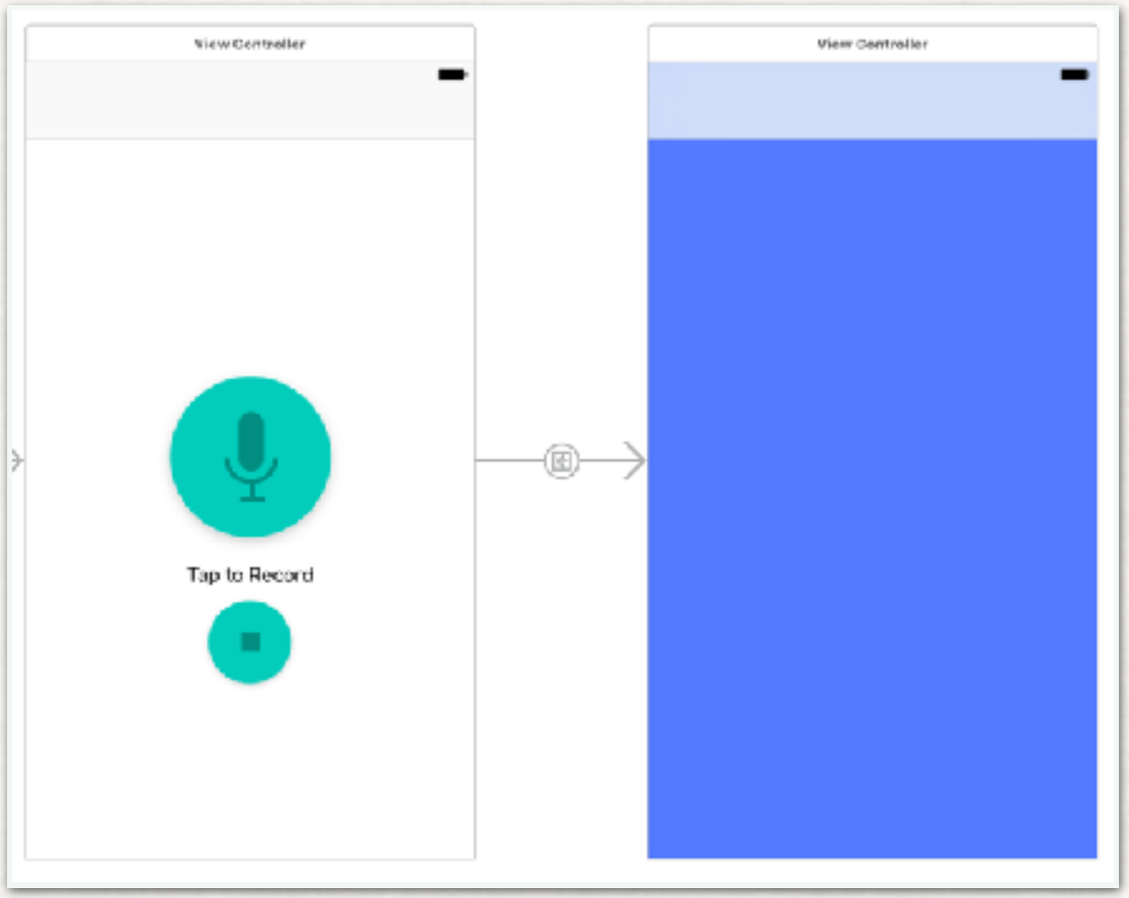
- 뷰 컨트롤러에서 다른 뷰 컨트롤러를 호출하여 화면 전환하기




- 현재의 뷰 컨트롤러에서 이동할 대상 뷰 컨트롤러를 직접 호출해서 화면을 표시하는 방식
- 모든 뷰 컨트롤러는 UIViewController 클래스를 상속받는데, 이 클래스에서 정의된 presentViewController(_:animated:completion:)메소드를 사용하면 화면 전환이 가능함
- 화면전환은 기존의 뷰 컨트롤러가 자기 자신을 유지한 채 새로운 뷰 컨트롤러를 호출하여 화면에 표시하는 방식으로 처리

# 화면전환 방법_3

- 화면 전환용 객체 세그웨이를 사용하여 화면 전환하기




- 화면과 화면의 연결을 위한 소스 코드 없이도 스토리보드 상에서 화면 전환 기능을 직접 구현할 수 있는 장점이 있다.
- 스토리보드상에서 세그웨이는 뷰 컨트롤러 사이의 화살표로 표시되며 뷰 컨트롤러와 뷰 컨트롤러 또는 화면 전환의 매개체가 되는 버튼과 뷰 컨트롤러 사이를 직접 연결하여 화면전환을 처리