Week 3.
iOS Seminar

ScrollView, Delegate Pattern, TableView, CollectionView

이번 세미나에서 다룰 주제는요!

01 ScrollView

02 Delegate Pattern

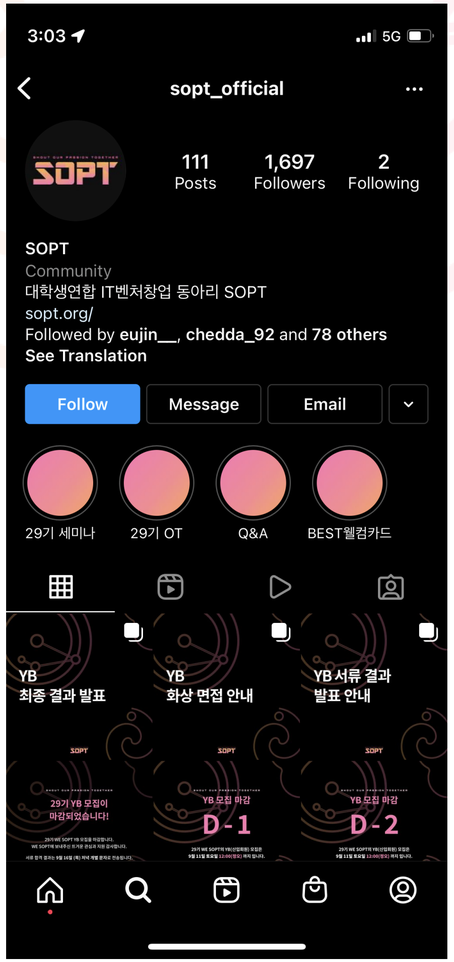
03 TableView

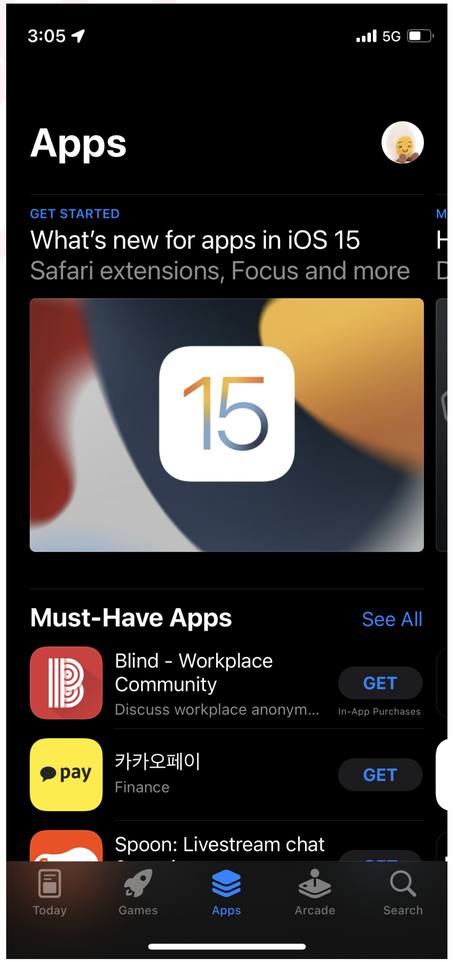
04 CollectionView

01

ScrollView

ScrollView





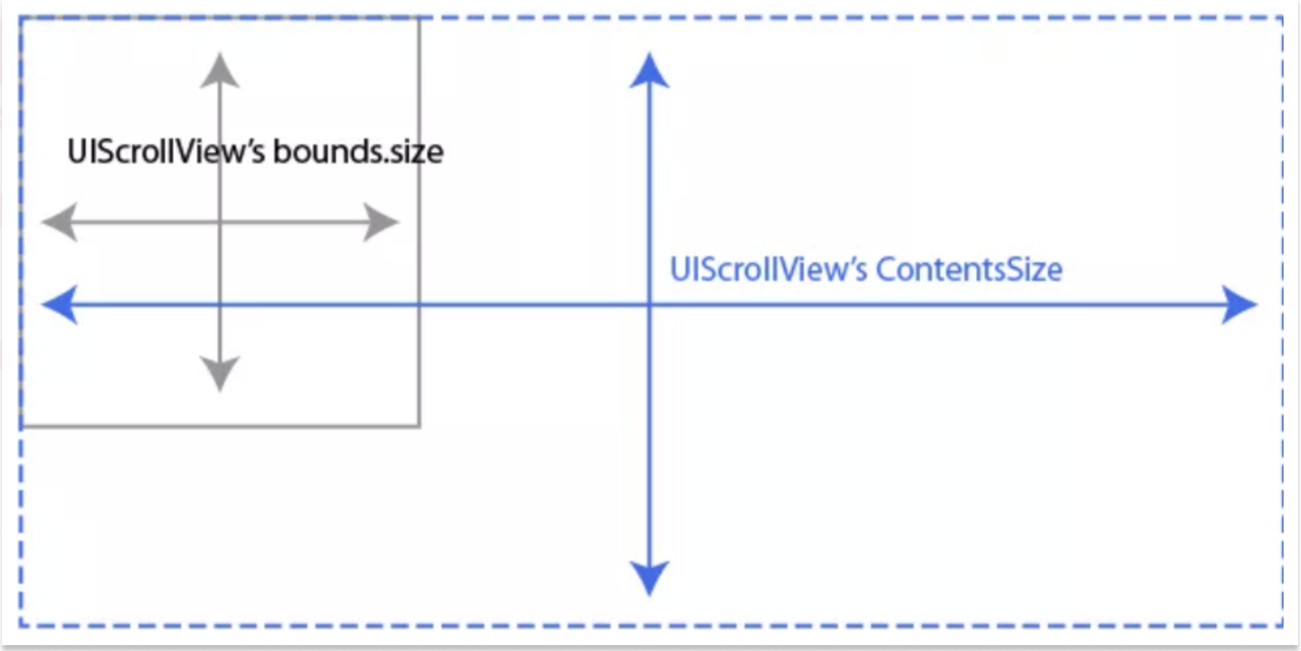
📌 ScrollView란?

말 그대로 사용자의 손가락에 따라
스크롤 기능을 구현하는 뷰입니다.

스크롤 기능을 통해 상하좌우로 움직이며
추가 컨텐츠를 보여주는 뷰이기 때문에
다양한 어플에서 당연하게 만나보셨을 겁니다!

UIScrollView는 오늘 배울 UITableView, UICollectionView의
Super class 이기 때문에 중요한 클래스입니다

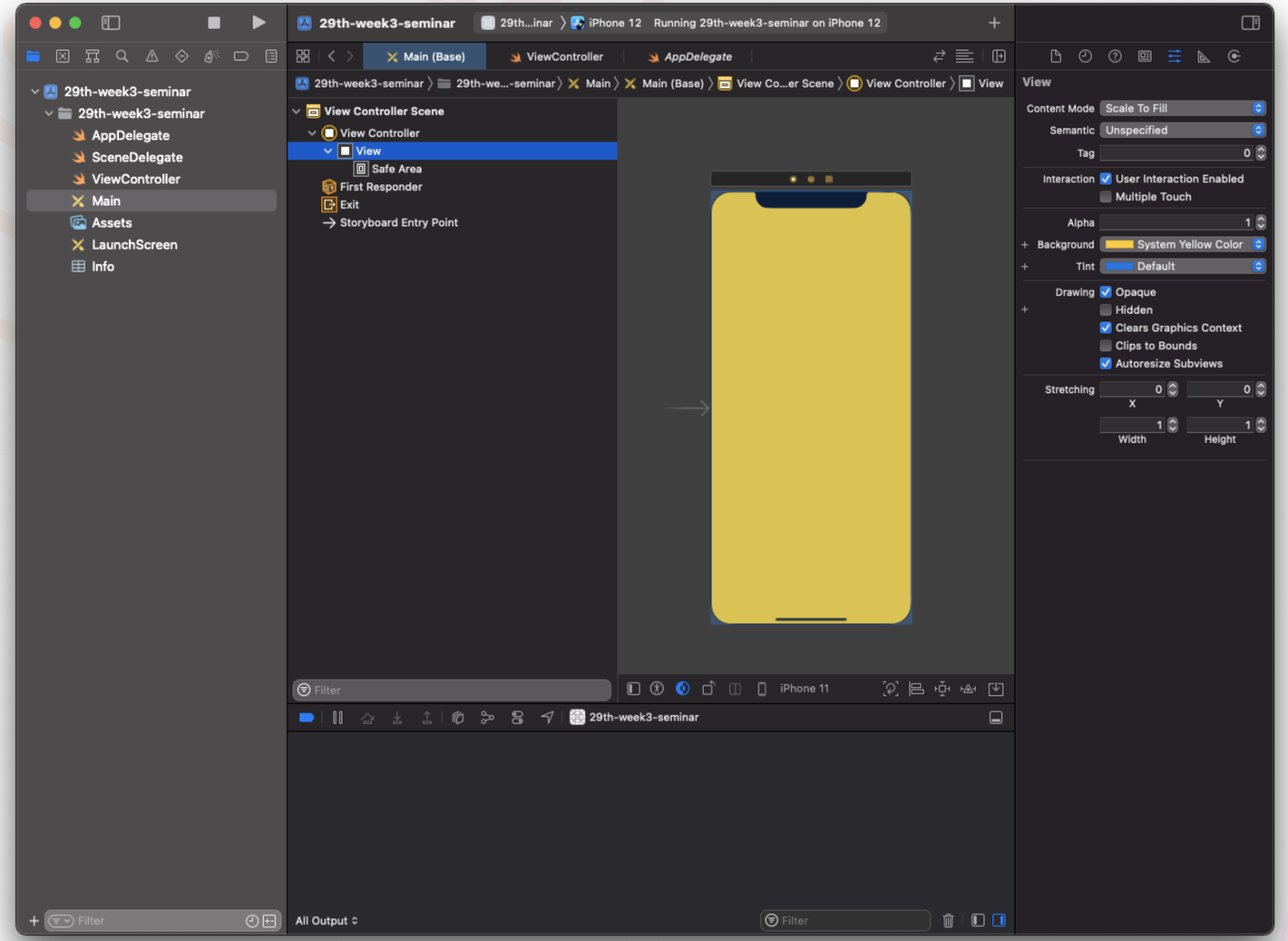
ScrollView



실제로는 ContentsSize만큼 보여줄 내용이 있지만,
scrollView가 해당 내용을 전부 담지 못할 때 사용하게 됩니다!

반대로, ScrollView가 해당 ContentSize를 모두 담을 수 있게 된다면
스크롤은 더 이상 되지 않습니다.

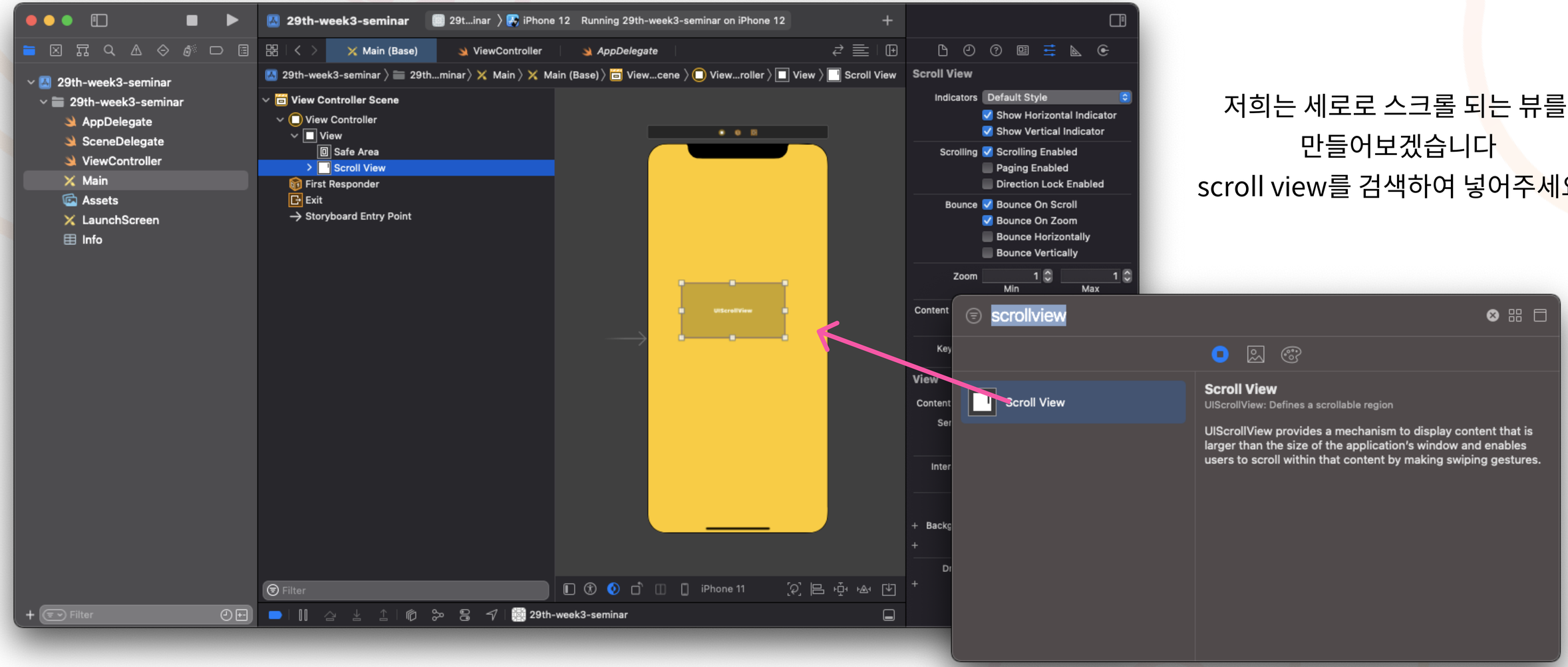
ScrollView



새 프로젝트를 만들어봅시다!

새 프로젝트를 만들고
스크롤뷰를 잘 보이게 하기 위해
viewcontroller의 view의
background color를 변경해주세요:)

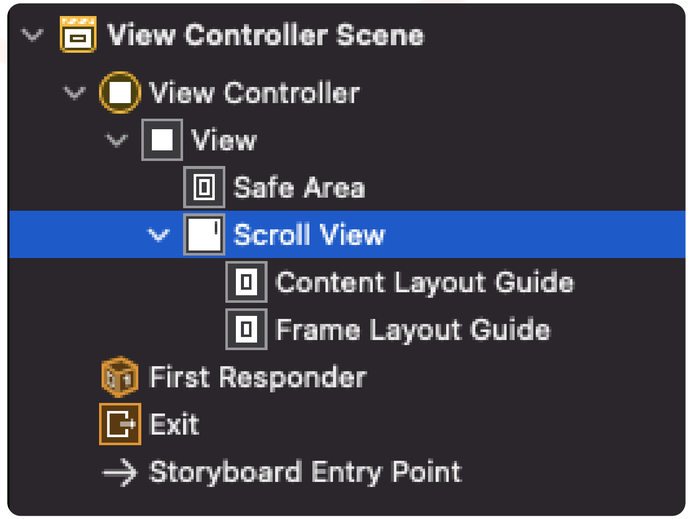
ScrollView



저희는 세로로 스크롤 되는 뷰를
만들어보겠습니다
scroll view를 검색하여 넣어주세요!

ScrollView

scroll view를 누르면
두개의 layout guide가
기본적으로 들어있습니다.



이 두개의 layout guide들은
scroll view의 instance property들 입니다

레이아웃 가이드 얻기

```
var frameLayoutGuide: UILayoutGuide
```

스크롤 뷰의 변환되지 않은 프레임 사각형을 기반으로 하는 레이아웃 가이드입니다.

```
var contentLayoutGuide: UILayoutGuide
```

스크롤 뷰의 번역되지 않은 콘텐츠 사각형을 기반으로 하는 레이아웃 가이드입니다.

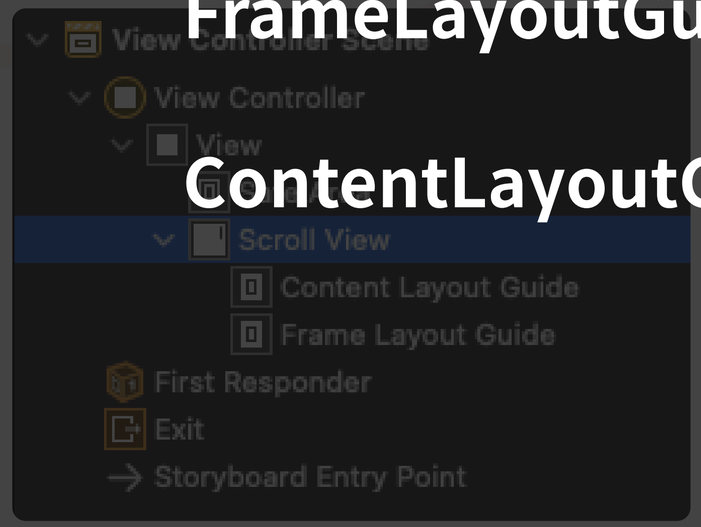
ScrollView

scroll view를 누르면
두개의 layout guide가
기본적으로 들어있습니다.

이 두개의 layout guide들은
scroll view의 instance property들 입니다

FrameLayoutGuide → 실제 화면에 보여지는 부분에 대한 layout

ContentLayoutGuide → 스크롤 될 content 전체에 대한 layout



레이아웃 가이드 얻기

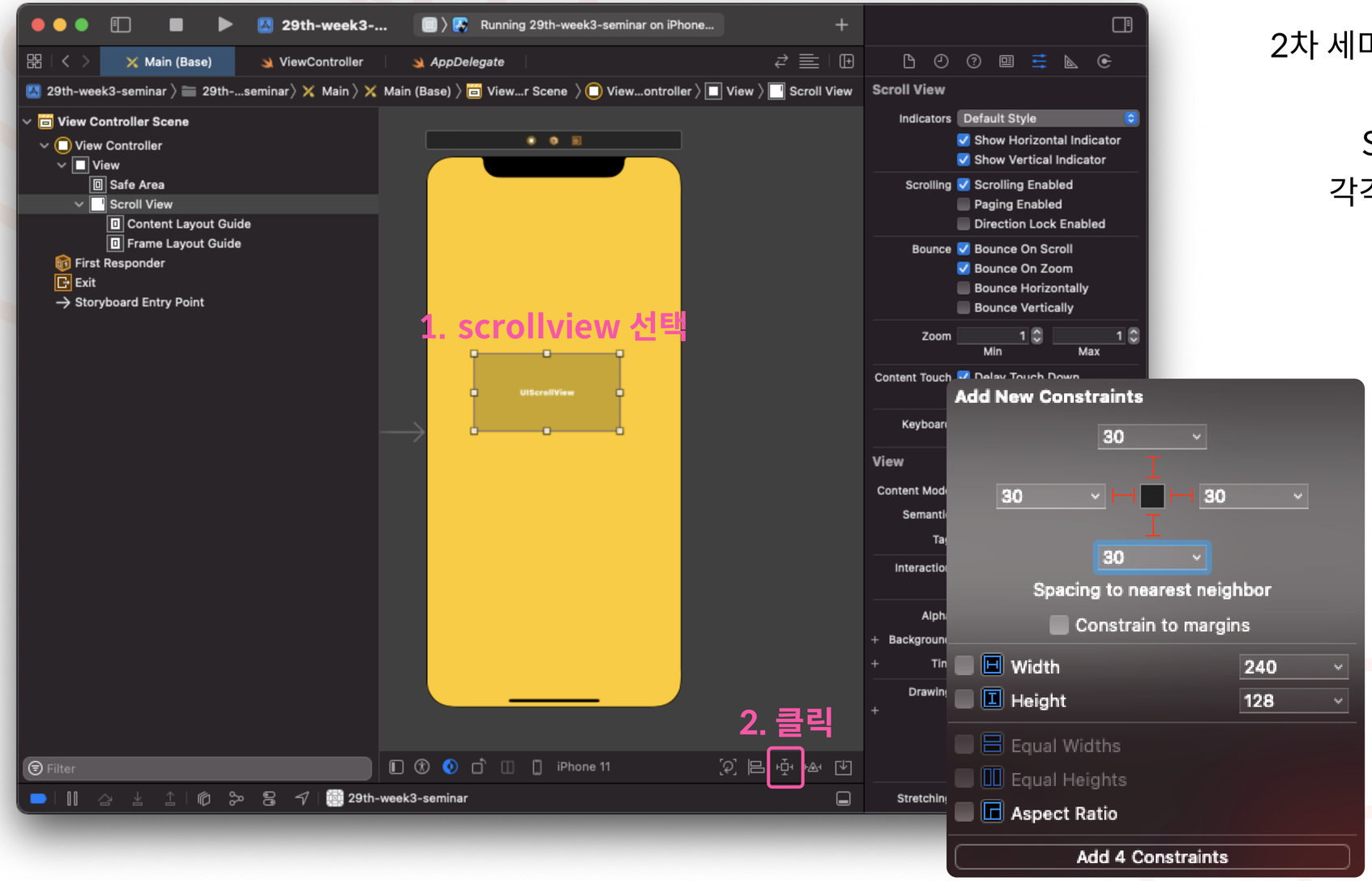
var frameLayoutGuide: UILayoutGuide

스크롤 뷰의 변환되지 않은 프레임 사각형을 기반으로 하는 레이아웃 가이드입니다.

var contentLayoutGuide: UILayoutGuide

스크롤 뷰의 번역되지 않은 콘텐츠 사각형을 기반으로 하는 레이아웃 가이드입니다.

ScrollView



2차 세미나 때 배웠던 AutoLayout 기억하시죠?

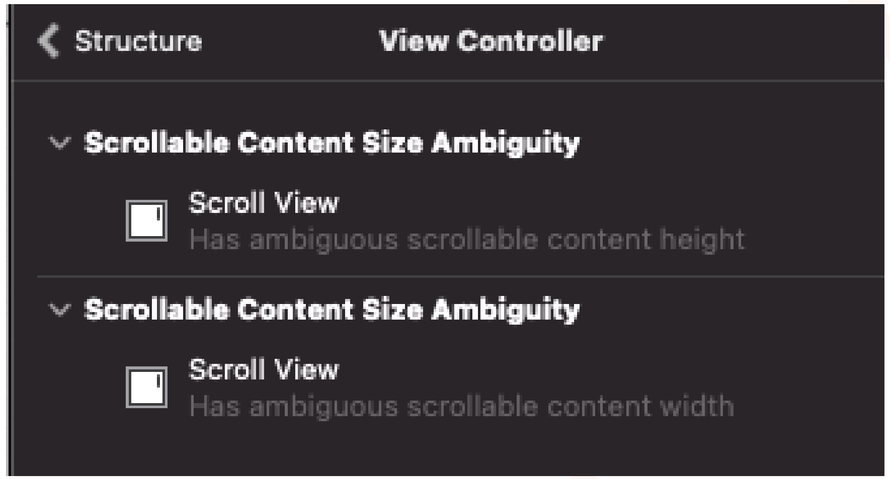
Scrollview가 좀더 잘 보이기 위해
각각 top, leading, trailing, bottom을
safe area에서 30씩 잡아서
constraints를 추가해주세요!

이렇게 잡으면
frame layout guide와
content layout guide 중
frame layout guide에
제약조건을 설정한 것입니다!

ScrollView

클릭해서 에러확인

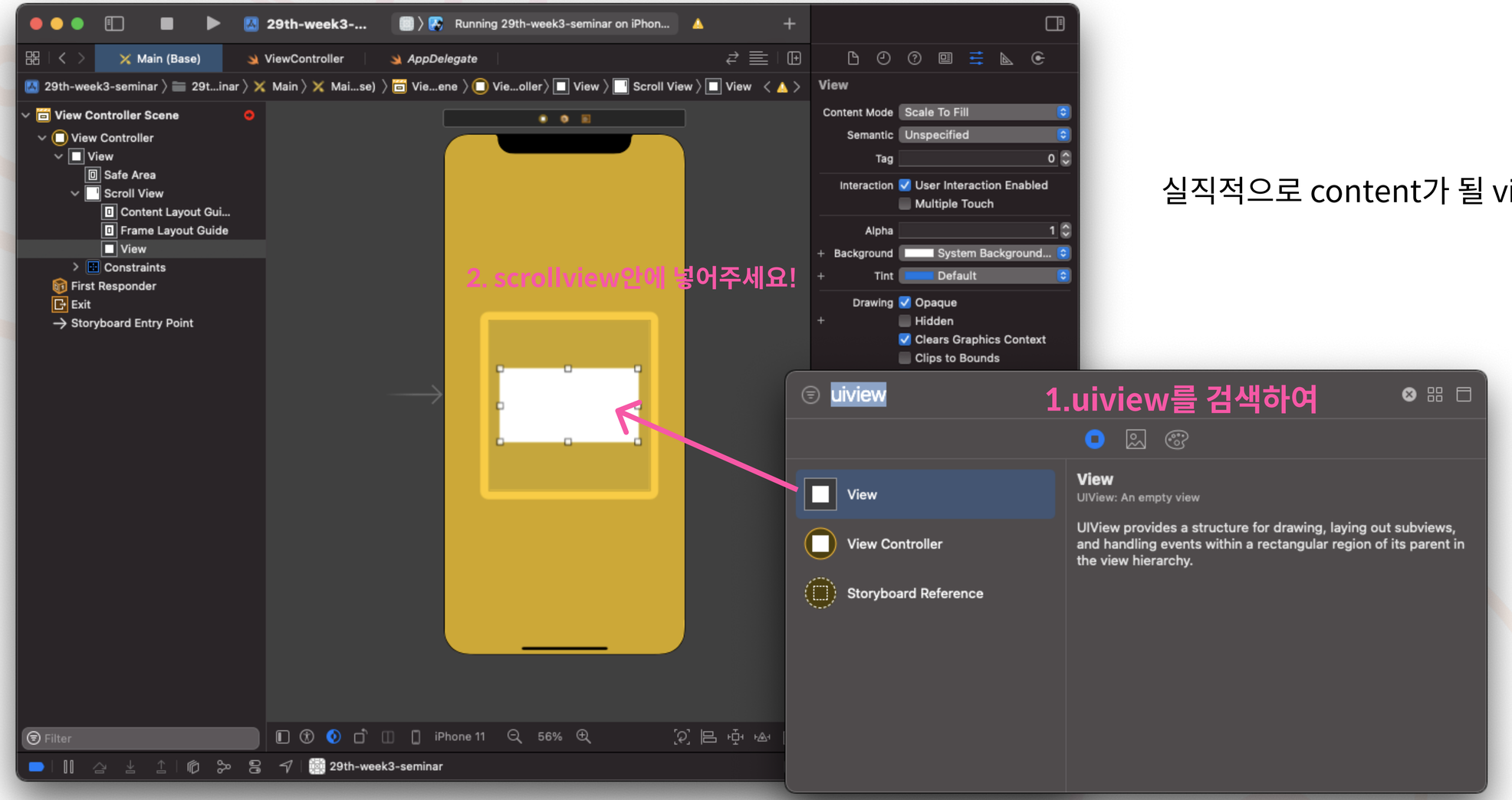
설정하고나면 에러가 뜨는데요!
당황하지 않고~ 에러를 확인해보면



Has ambiguous scrollable content width/height

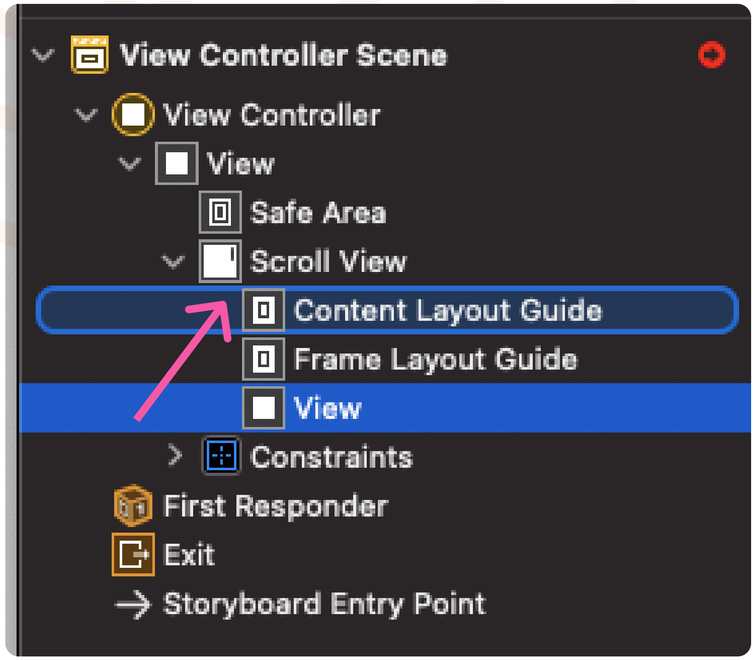
content의 size가 모호하다는 경고를 나타나 있습니다!

ScrollView

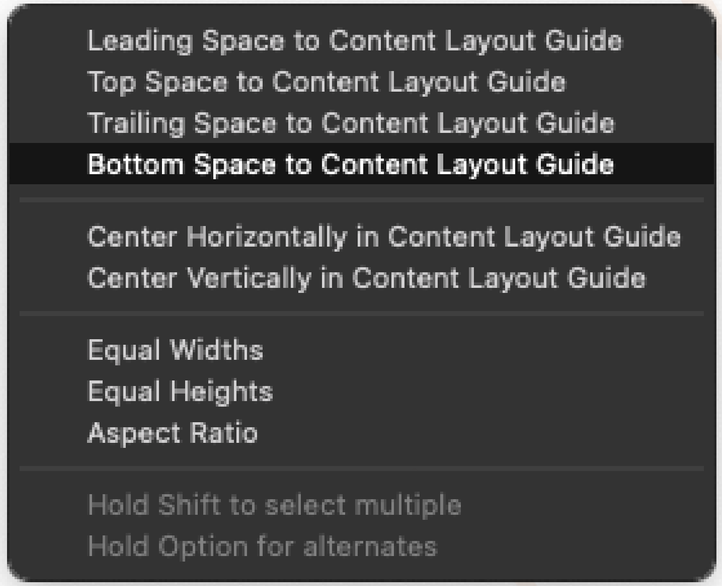


실직적으로 content가 될 view를 추가해봅시다!

ScrollView

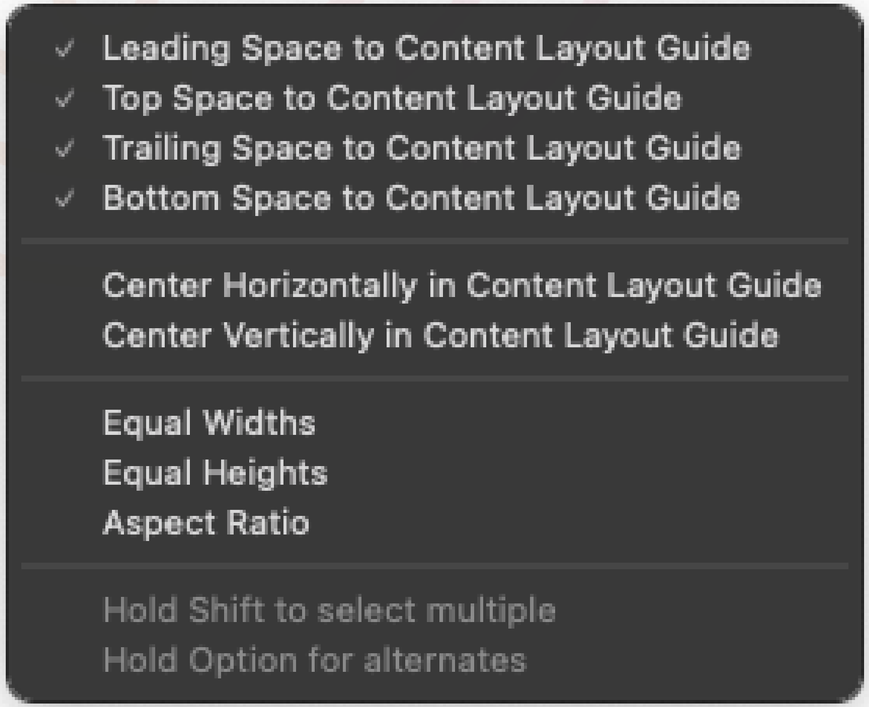


view를 클릭한 후
control를 누른채로
Content Layout guide로
drag & drop을 해주세요



이 창이 잘 뜨는지 확인해주세요:)

ScrollView

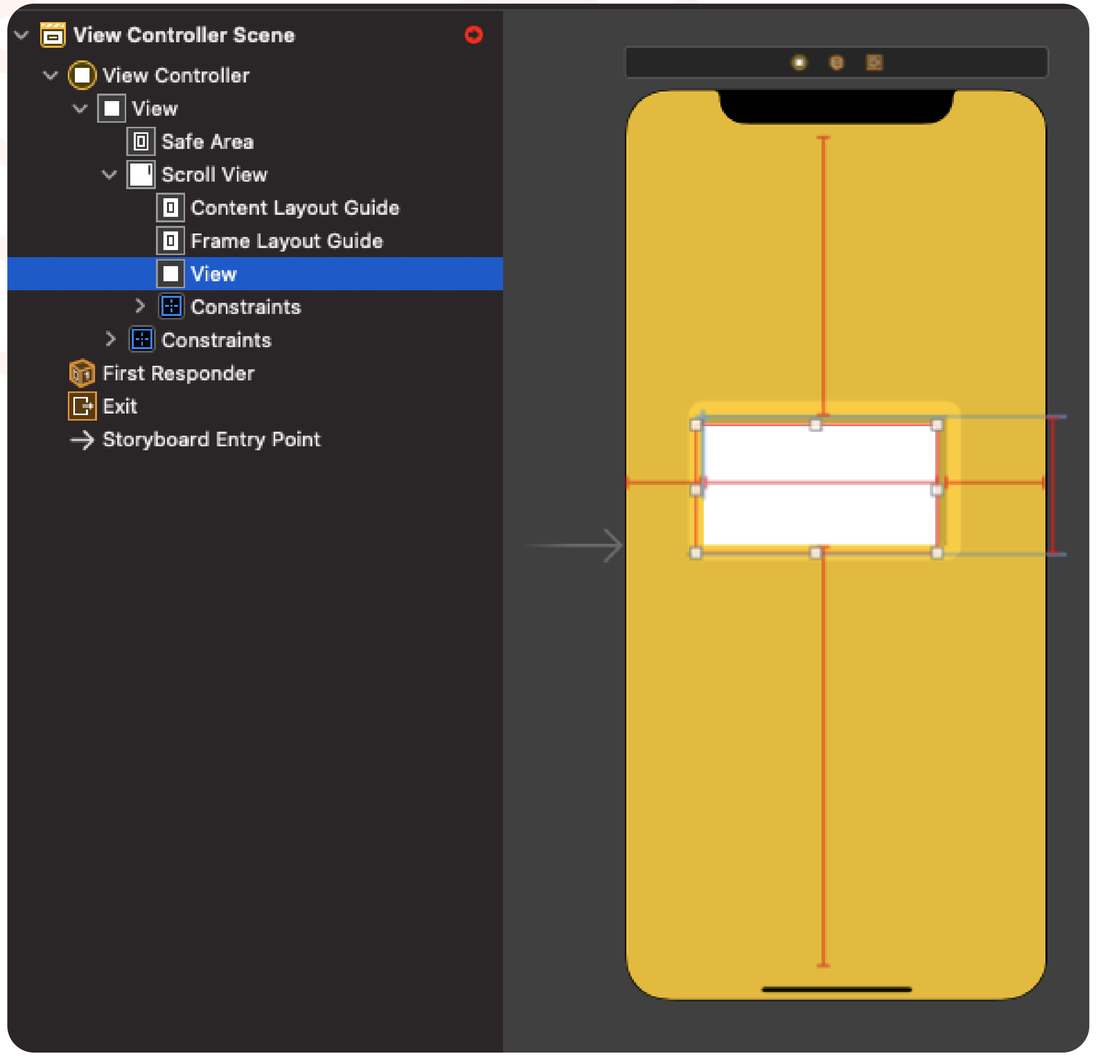


Leading Sapce to Content Layout Guide는
content Layout guide의 leading과 지금 제약을 걸 view의 leading간에
거리(spacing)를 두겠다라는 제약입니다

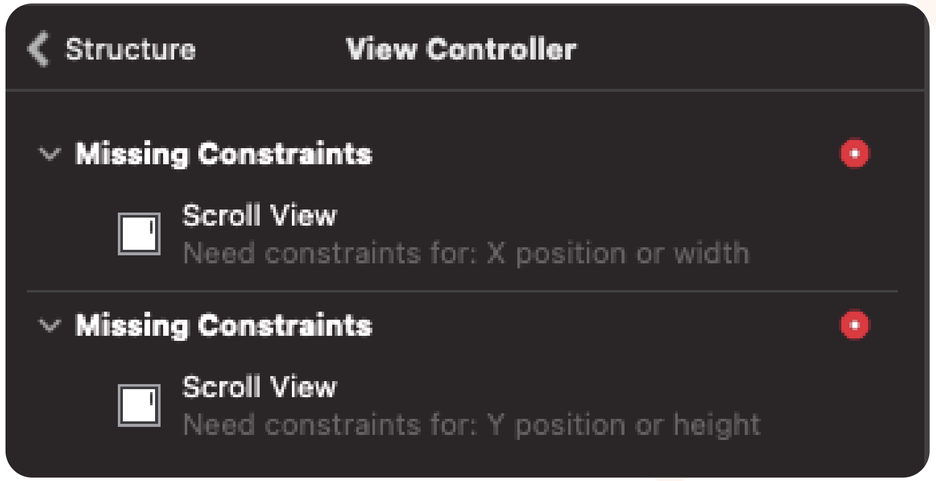
저희는 content뷰를
content layout guide에 딱 맞출 것이기 때문에
leading, top, trailling, bottom 4가지 모두 선택해주세요!

***command(⌘)를 누르고 선택하면 한번에 중복선택 할 수 있습니다**

ScrollView

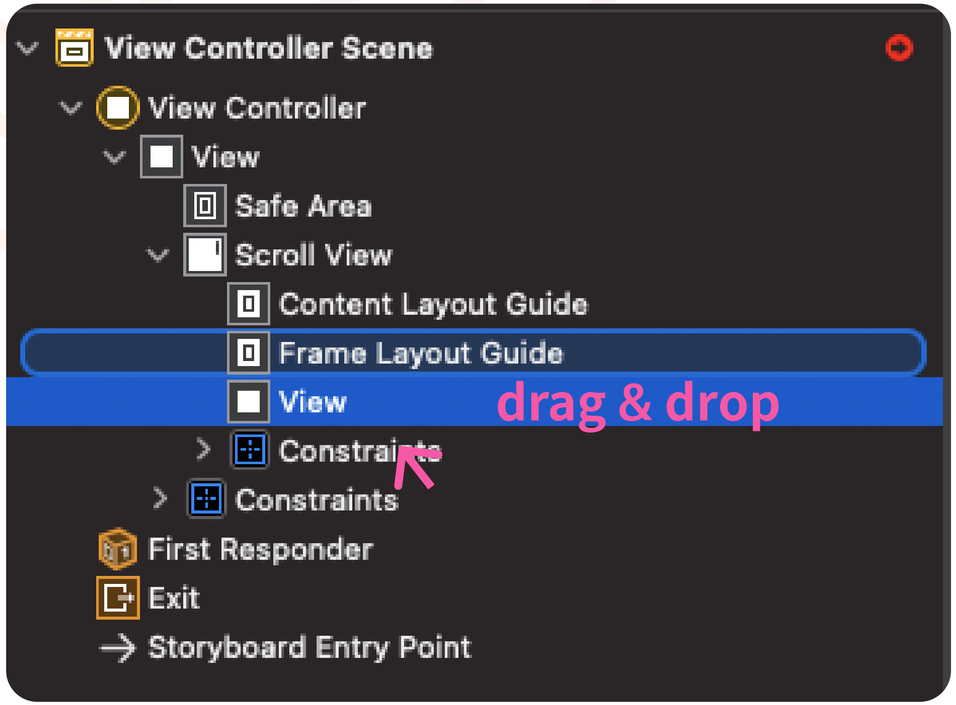


그래도 아직 뜨는 에러..😰



아직 contentView의 실질적인 width와 height가
명확하게 없기 때문에 경고가 뜨게 됩니다.

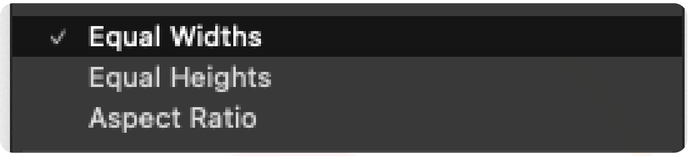
ScrollView



그럼 이제 Content View의 사이즈를 잡아봅시다!
먼저 저희는 세로 스크롤로 구현할 것이기 때문에
가로는 고정, 세로는 동적으로 변할 수 있어야합니다!

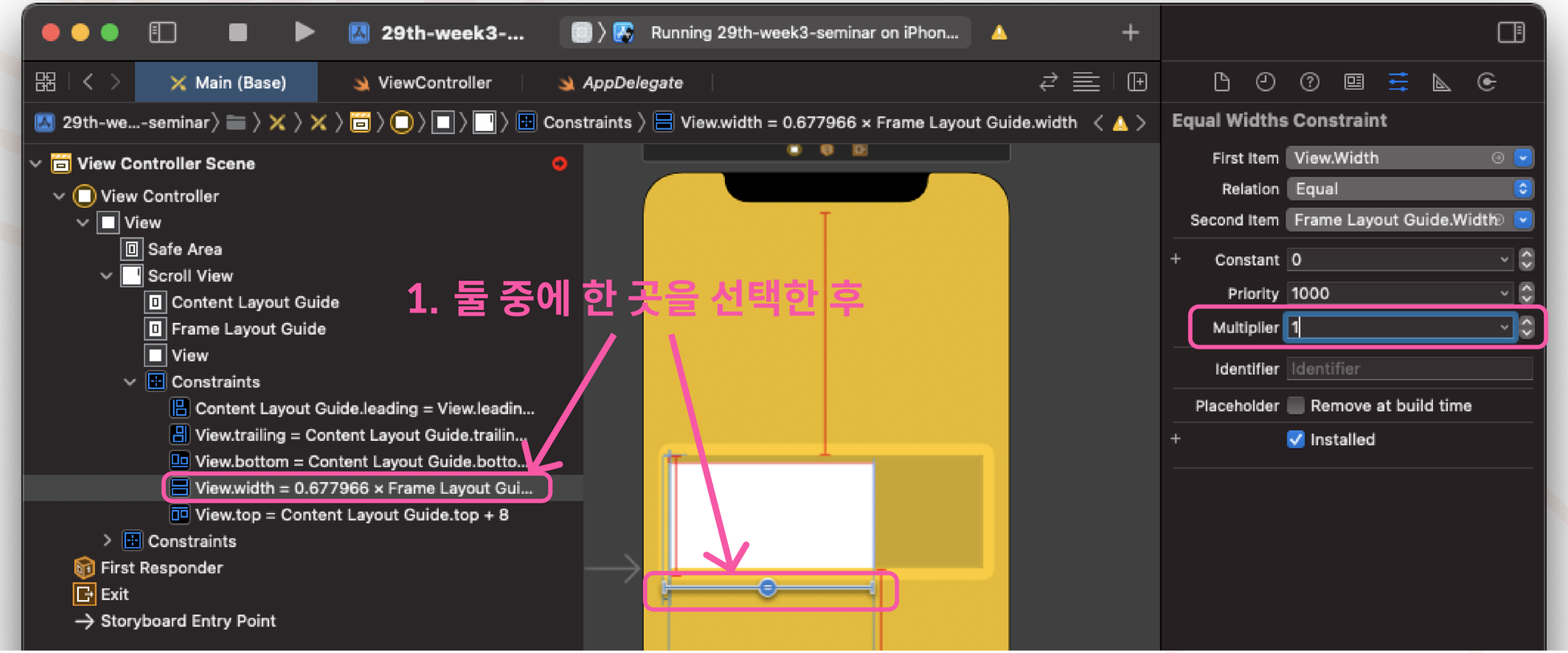
먼저, 가로를 고정하기 위해
화면에 보일 부분인 Frame layout guide와 가로를 똑같이 지정해봅시다

view를 클릭한 후 control를 누른채로
Frame Layout guide로
drag & drop을 해주세요



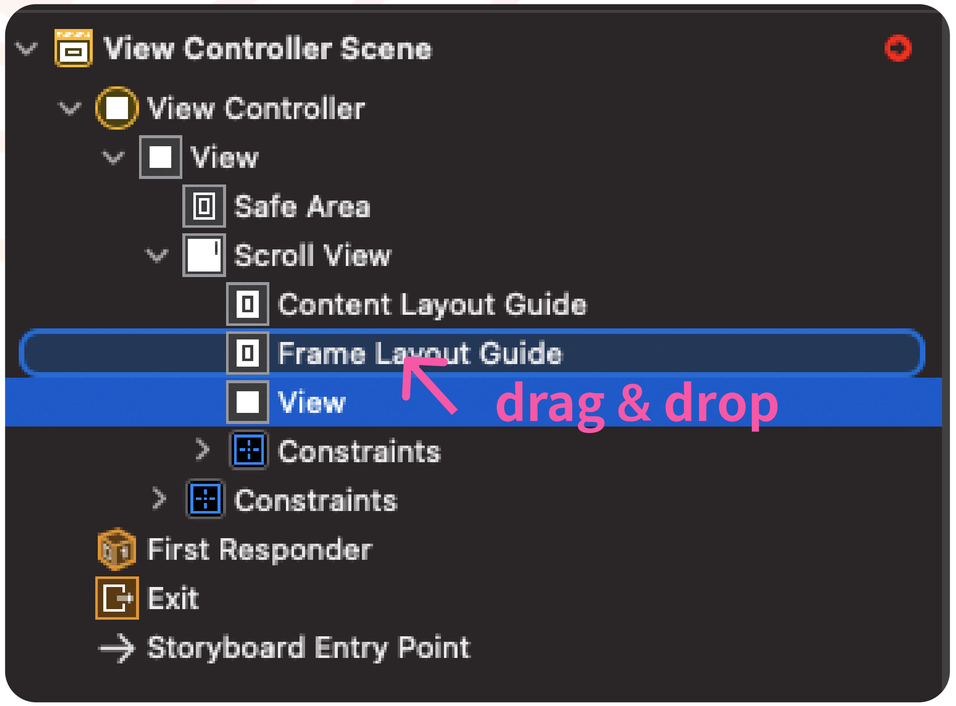
제약조건을 선택하는 창이 뜨면
Equal widths를 클릭해주세요!

ScrollView

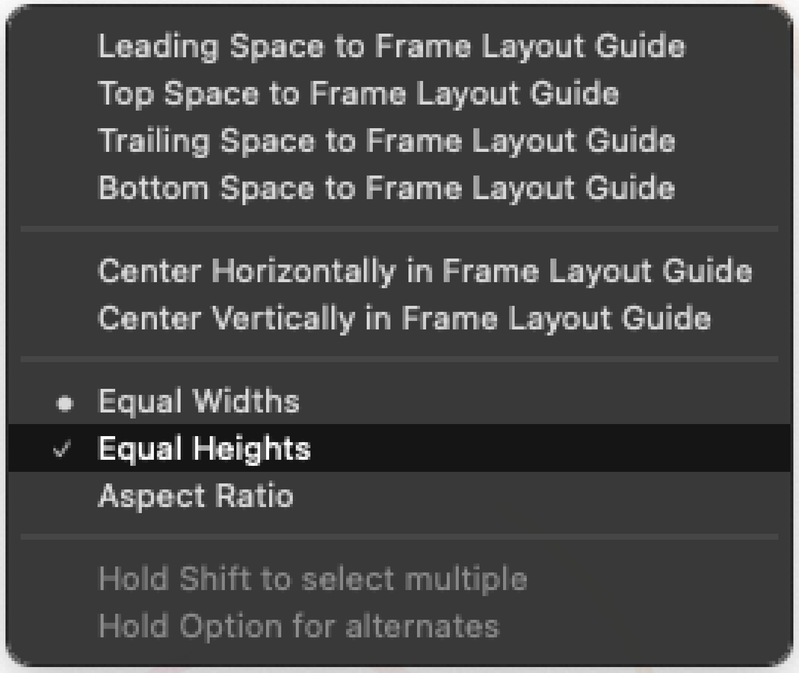


2. multiplier를
1로
변경해주세요

ScrollView



width 설정과 동일하게
height도 제약조건을 선택해주세요!



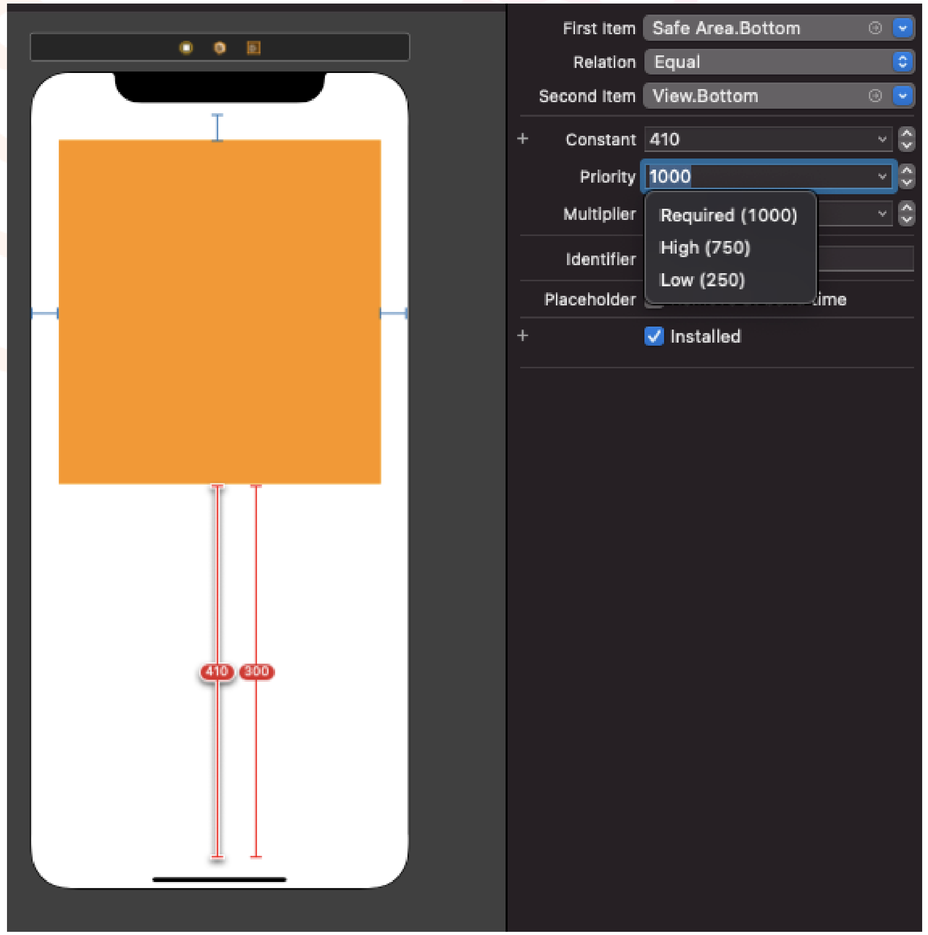
- Safe Area (안전 영역)
- AutoLayout 속성
- Constraint (제약)
- Layout Margin (레이아웃 여백)
- Intrinsic Content Size (고유 컨텐츠 크기)
- Priority (우선순위)

📌 Priority

- 제약조건의 우선순위를 나타내는 NSLayoutConstraint의 instance property 입니다.
- 더 높은 우선 순위 제약 조건은 낮은 우선 순위 제약 조건보다 먼저 충족됩니다
- Priority는 1~1000 사이의 값을 가지고 있는데 값이 크면 클수록 우선순위가 높다는 의미입니다.
- 필수(1000), 높음(750), 중간(500), 낮음(250)을 사용하길 권장합니다.

ScrollView

📌 Priority



현재 orange view bottom에 대한 제약조건을

- safe area와 420 거리가 있는 제약조건
- safe area와 300 거리가 있는 제약조건

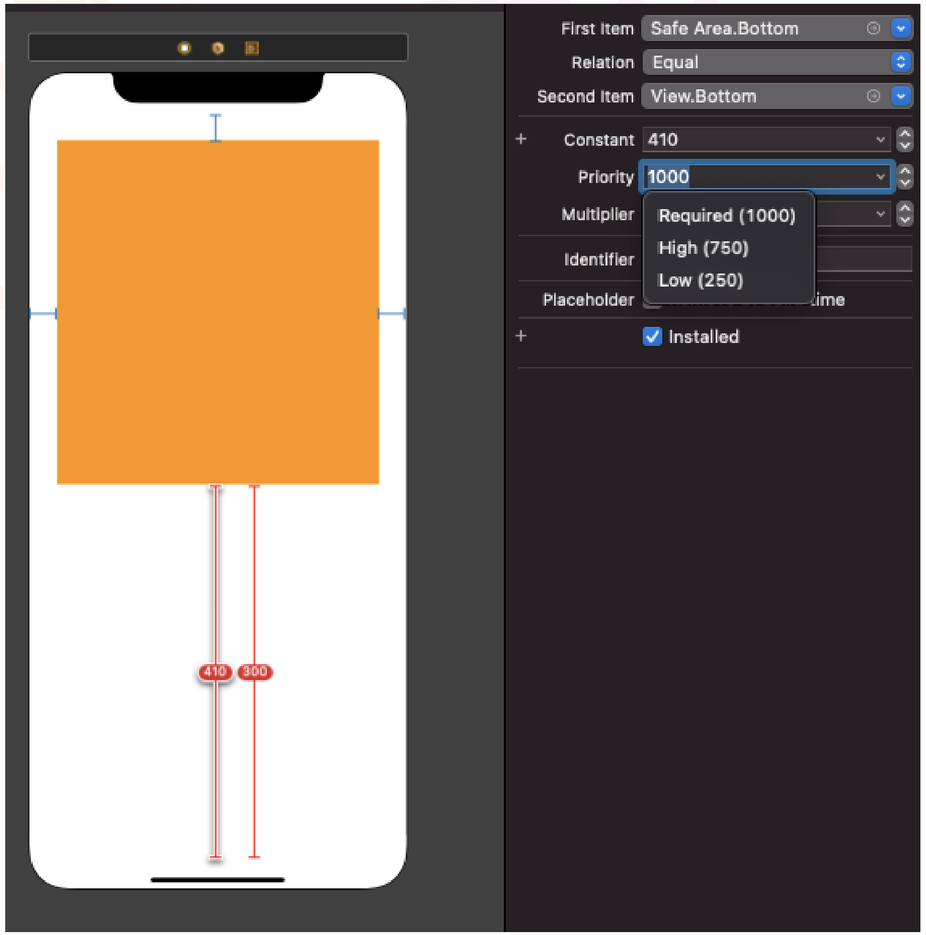
2가지 모두 추가했습니다.

이렇듯 단순히 같은 부분에 제약조건을 2가지 이상 주었을 때는 에러가 발생합니다!
→ 뷰가 그려질 때 어떤 제약조건에 맞춰서 그릴지 결정할 수 없기 때문

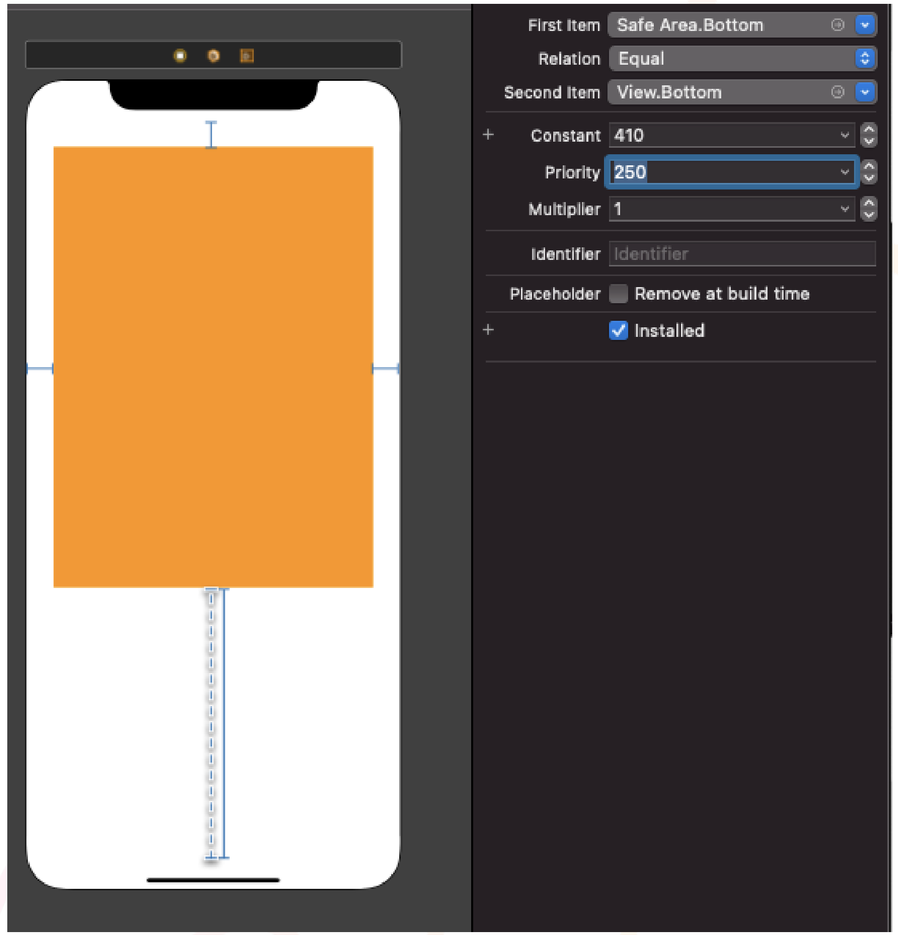
그래서 하나의 제약의 우선순위를 낮추어
우선순위가 더 높은 제약조건이 적용되도록 수정해야합니다

ScrollView

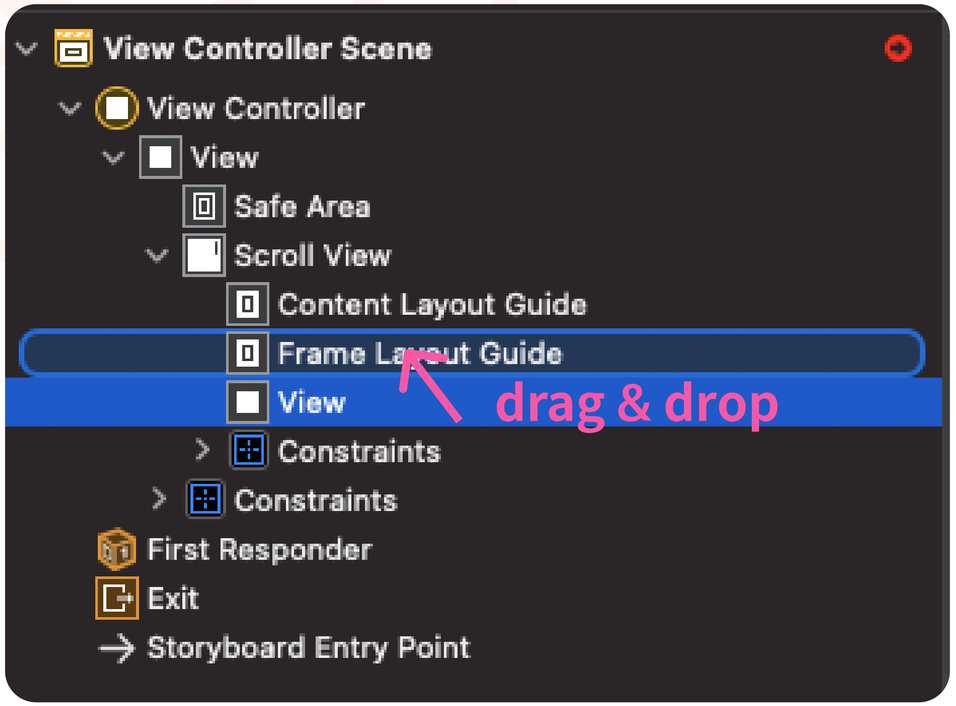
📌 Priority



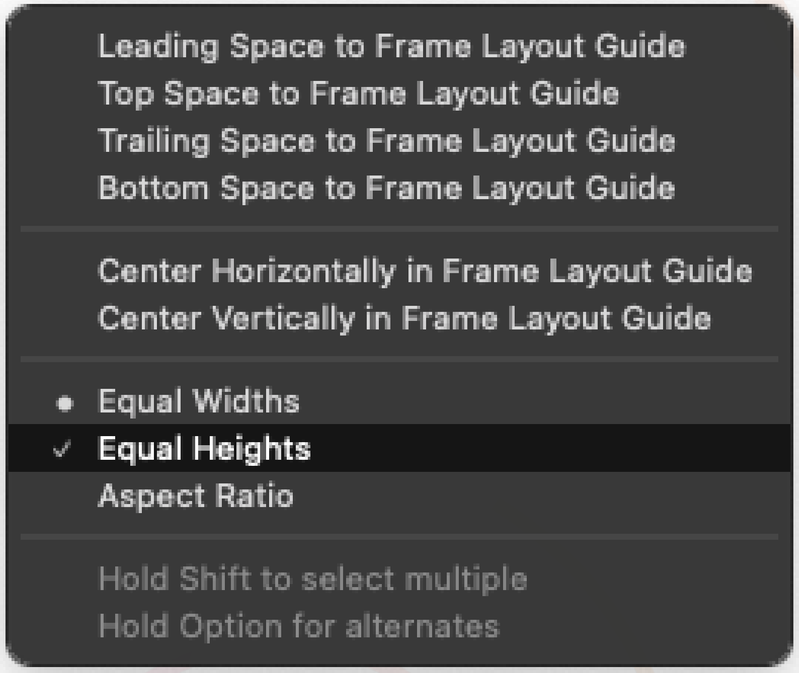
기존에 420으로 설정했던
제약조건의 우선순위를
250으로 내리면서
orange view bottom은
우선순위가 높은
300으로 뷰가 그려집니다.



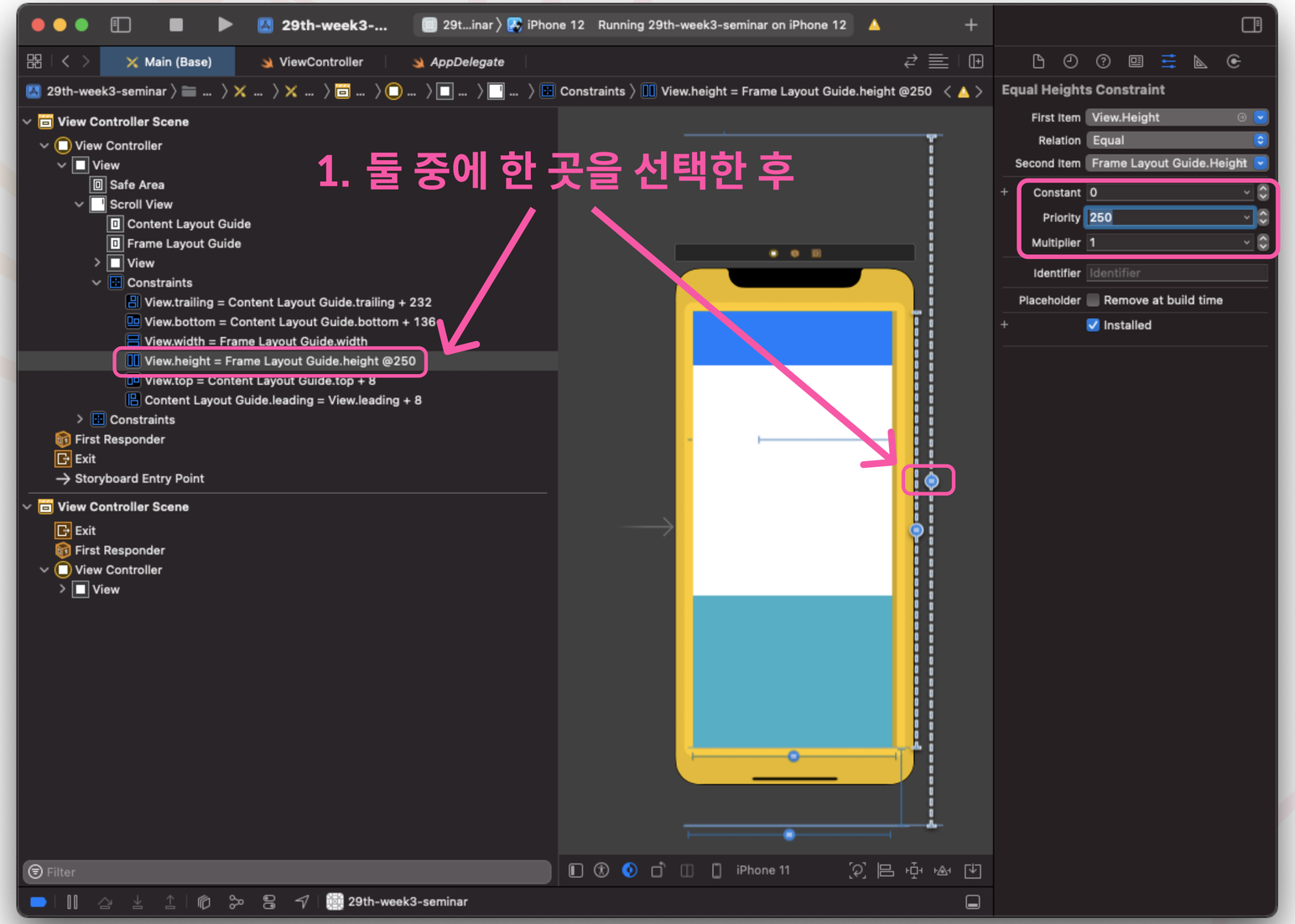
ScrollView



width 설정과 동일하게
height도 제약조건을 선택해주세요!

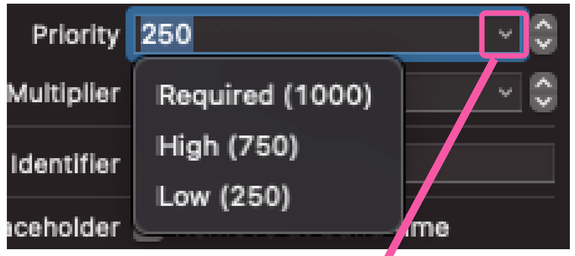


ScrollView

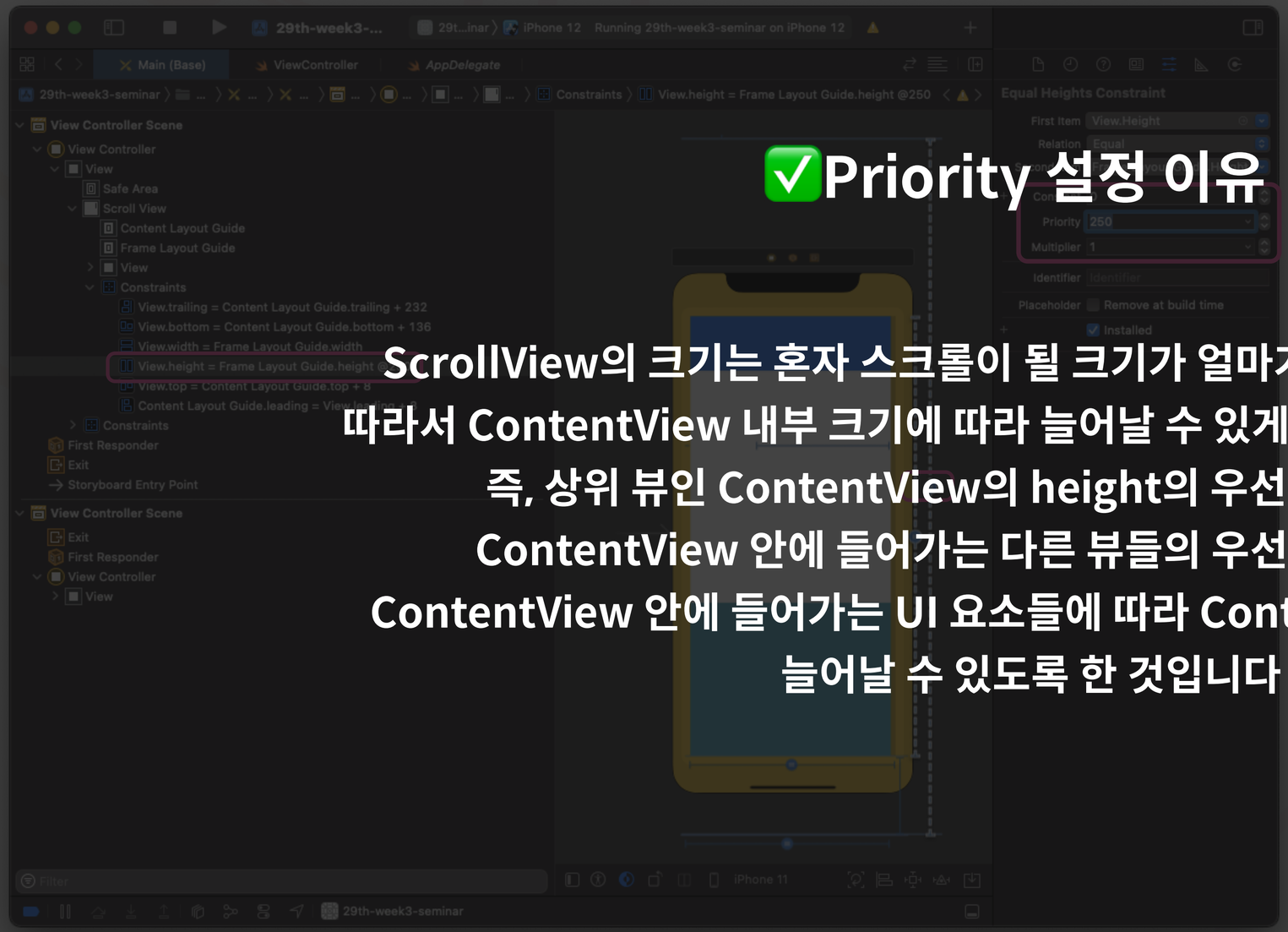


2. multiplier를 1로 변경해주세요

3. Prioity는 250으로 변경해주세요



이곳을 누르면 직접 입력이 아닌
선택을 할 수도 있습니다!

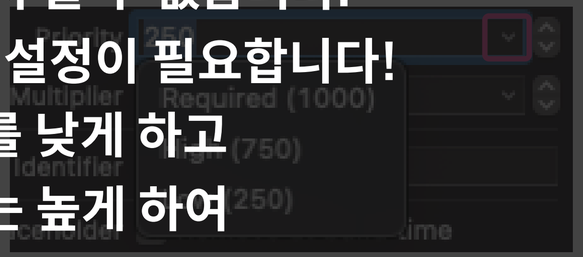


✅Priority 설정 이유

ScrollView의 크기는 혼자 스크롤이 될 크기가 얼마가 될 지 알 수 없습니다.
따라서 ContentView 내부 크기에 따라 늘어날 수 있게 하는 설정이 필요합니다!
즉, 상위 뷰인 ContentView의 height의 우선순위를 낮게 하고
ContentView 안에 들어가는 다른 뷰들의 우선순위는 높게 하여
ContentView 안에 들어가는 UI 요소들에 따라 ContentView의 height이
늘어날 수 있도록 한 것입니다

2. multiplier를 1로 변경해주세요

3. Prioity는 250으로 변경해주세요

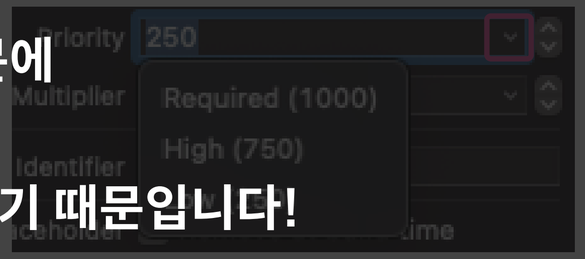


이곳을 누르면 직접 입력이 아닌
선택을 할 수도 있습니다!

✅contentView의 witdh의 Priority는 1000으로 유지한 이유

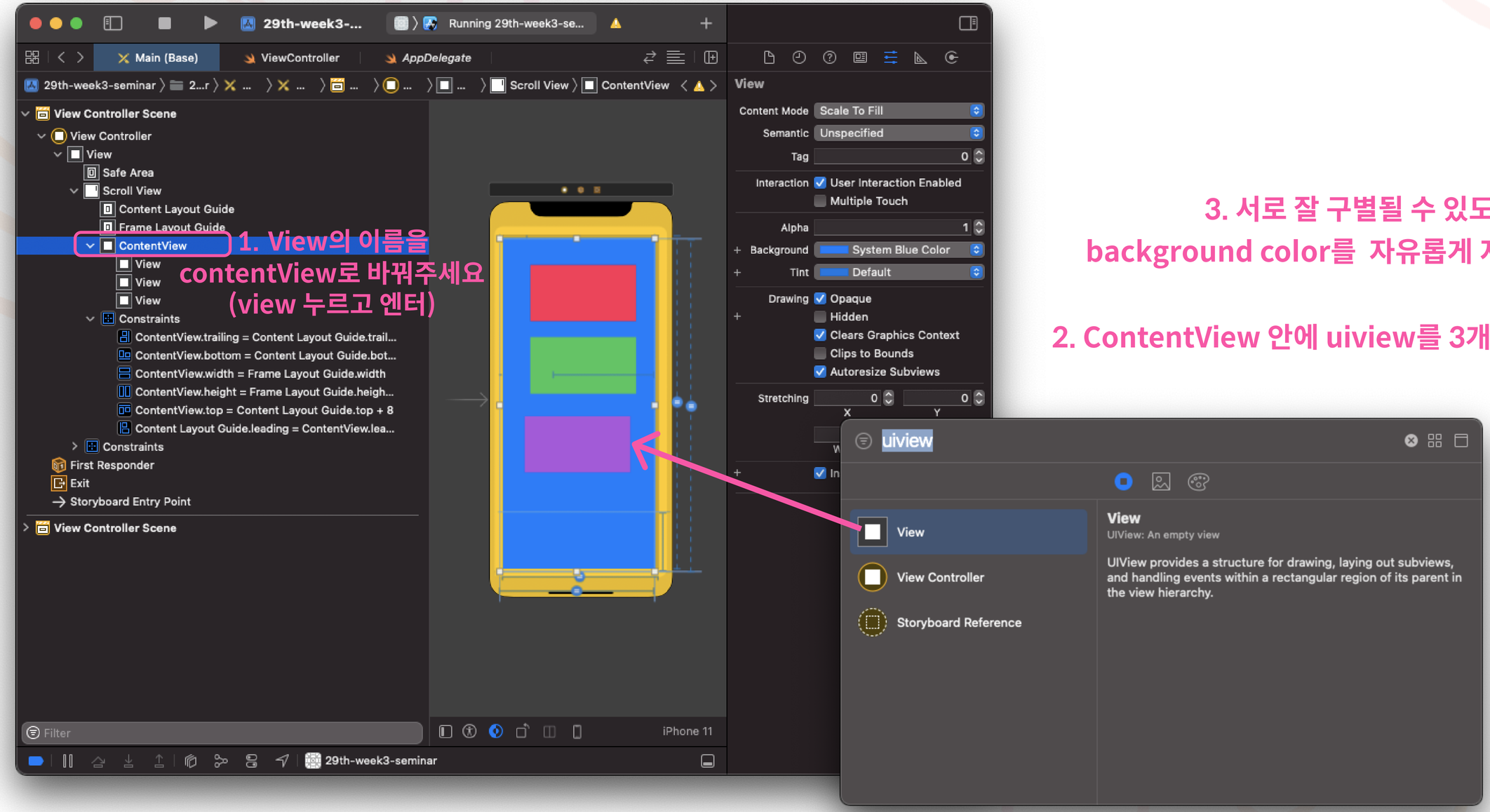
현재 구현하려 하는 것은 세로 스크롤이기 때문에
Content View 내에 존재하는 UI요소의
Constraint의 영향을 받지 않고 화면 크기를 유지해야 하기 때문입니다!

3. Prioity는 250으로 변경해주세요



이곳을 누르면 직접 입력이 아닌
선택을 할 수도 있습니다!

ScrollView



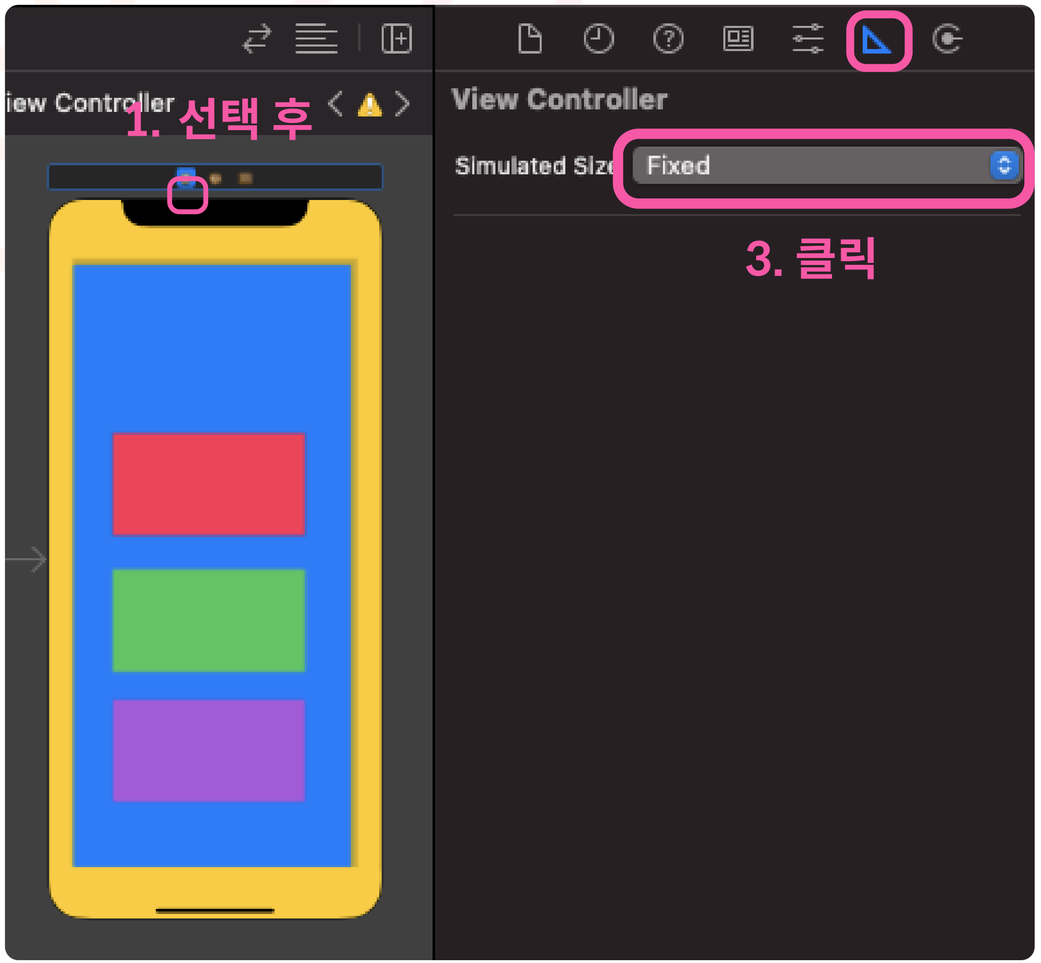
3. 서로 잘 구별될 수 있도록
background color를 자유롭게 지정해주세요!

2. ContentView 안에 uiview를 3개를 추가해주세요

ScrollView

layout을 길게 잡을 때 화면이 작아 불편하니까 뷰를 늘려보겠습니다~!

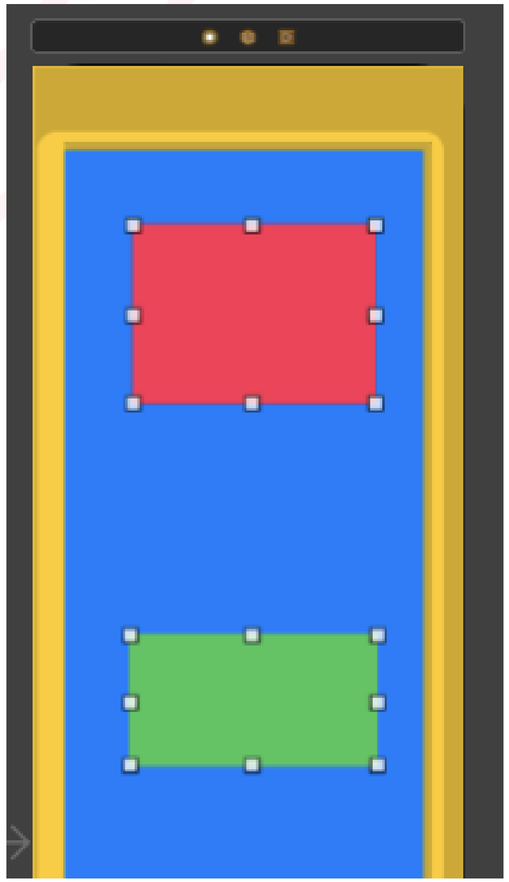
2. size indecator 선택



height을 크게 설정해주세요!

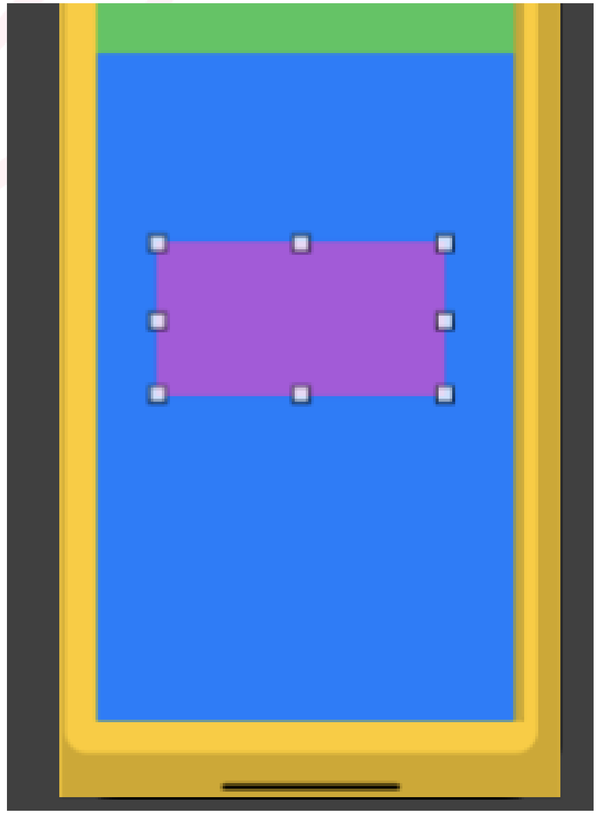
ScrollView

content view 내부의 view들의 auto layout을 잡아주세요!



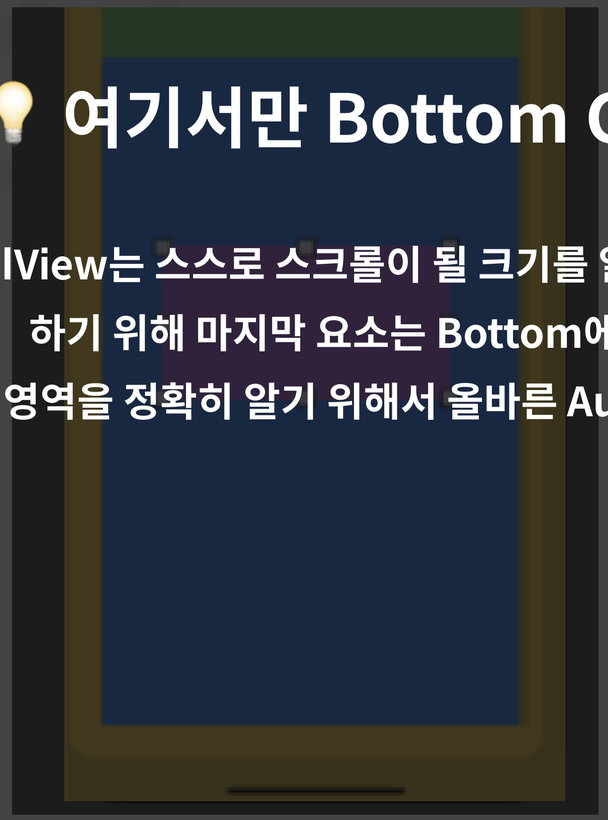
ScrollView

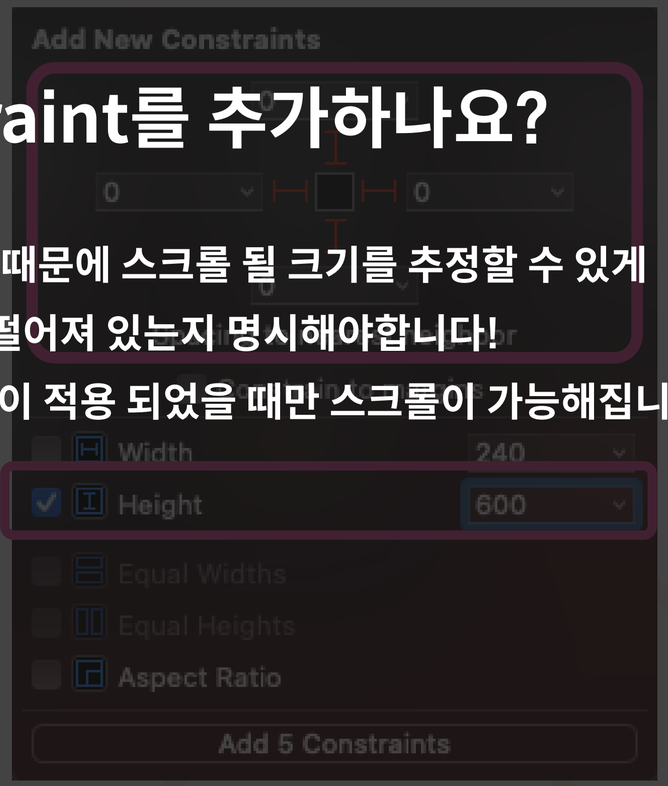
content view 내부의 view들의 auto layout을 잡아주세요!



ScrollView

content view 내부의 view들의 auto layout을 잡아주세요!

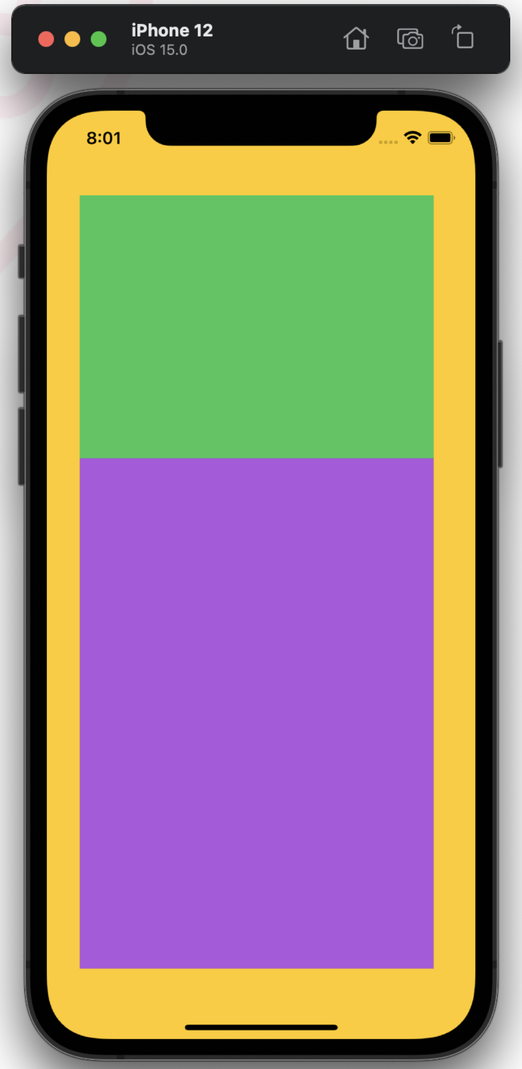




💡 여기서만 Bottom Constraint를 추가하나요?

- ScrollView는 스스로 스크롤이 될 크기를 알지 못하기 때문에 스크롤 될 크기를 추정할 수 있게 하기 위해 마지막 요소는 Bottom에서 얼마나 떨어져 있는지 명시해야합니다!
- 스크롤 될 영역을 정확히 알기 위해서 올바른 AutoLayout이 적용 되었을 때만 스크롤이 가능해집니다.

ScrollView



빌드시켜서 확인해보세요:)

02

Delegate Pattern

Delegate Pattern

UITextViewDelegate
UIKit > UITextViewDelegate
The methods for receiving editing-related messages for text view objects.
DOCUMENTATION Objective-C | Swift

UITableViewDelegate
UIKit > UITableViewDelegate
Methods for managing selections, configuring section headers and footers, deleting and reordering cells, and performing other actions in a table view.
DOCUMENTATION Objective-C | Swift

UIPickerViewDelegate
UIKit > UIPickerViewDelegate
The delegate of a UIPickerView object must adopt this protocol and implement at least some of its methods to provide the picker view with the data it needs to construct itself.
DOCUMENTA

iOS 개발을 하다보면 많고 다양한 Delegate를 많이 만나보실 거에요!

그래서 먼저 Delegate가 무엇인지부터 이해하고 가보자구요~!

UITextInputDelegate
An intermediary between a document and the text input system.
DOCUMENTATION Objective-C | Swift

UICollectionViewDelegate
UIKit > UICollectionViewDelegate
The methods adopted by the object you use to manage user interactions with items in a collection view.
DOCUMENTATION Objective-C | Swift

UITextFieldDelegate
UIKit > UITextFieldDelegate
A set of optional methods to manage the editing and validation of text in a text field object.
DOCUMENTATION Objective-C | Swift

UIPickerViewAccessibilityDelegate
UIKit > UIPickerViewAccessibilityDelegate
A set of methods you can implement to provide accessibility information for individual components of a picker view.
DOCUMENTATION Objective-C | Swift

Delegate Pattern

delegate 미국식 영국식 동사 [ˈdeɪɪgeɪt]

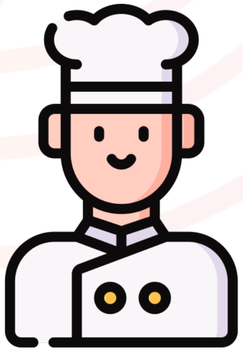
1. [명사] (집단의 의사를 대표하는) 대표(자)
2. [동사] (권한업무 등을) 위임하다
3. [동사] (대표를) 뽑다[선정하다]

delegate의 뜻 중 “위임하다”의 뜻에 주목해주세요!
*위임하다 : 어떤 일을 책임 지워 맡기다

아팟장이 iOS 세미나 조 편성하는 일을 회장에게 위임했다
-> 앞으로 iOS 세미나 조 편성일을 아팟장이 아닌 회장이 하겠죠?

할 일을 분배해서 다른 사람에게 해당 일에 “위임”시킨 후
해당 일이 들어오면 그 일을 맡은 사람이 한다는 것이 뽀인트입니다

Delegate Pattern



솝트 레스토랑 총괄 셰프

한식, 중식, 일식 메뉴가 솝트 레스토랑에 추가되었습니다!

이 총괄 셰프 혼자서 한식, 중식, 일식을 다 할 수도 있겠지만,
그것보다는 각각의 요리를 할 수 있는 다른 셰프를 구해서
해당 메뉴 주문이 들어오면 그 셰프들에게 맡기는게 더 효율적이겠지요??

그래서, 솝트 레스토랑의 셰프로 일할 사람을 구인해야하는데
단순히 아무나 오면 저희 레스토랑 망하겠죠..?

“현재 솝트 레스토랑에 있는 메뉴들을 만들 수 있는 사람”
이라는 조건을 걸어야합니다!

Delegate Pattern

솝트 레스토랑 총괄 셰프

한식, 중식, 일식 메뉴가 솝트 레스토랑에 추가되었습니다!

이 총괄 셰프 혼자서 한식, 중식, 일식을 다 할 수도 있겠지만,
그것보다는 각각의 요리를 할 수 있는 다른 셰프를 구해서
해당 메뉴 주문이 들어오면 그 셰프들에게 맡기는게 효율적이겠지요??

그래서, 솝트 레스토랑의 셰프로 일할 사람을 구인해야하는데
단순히 아무나 오면 저희 레스토랑 망하겠죠..?

“현재 솝트 레스토랑에 있는 메뉴들을 만들 수 있는 사람”
이라는 조건을 걸어야합니다!

“현재 솝트 레스토랑에 있는 메뉴들을 만들 수 있는 사람” 이라는 조건

✨Protocol✨

📌 Protocol

- Protocol은 특정한 작업이나 기능 부분적인 부분에 적합한 메소드, 프로퍼티 그리고 다른 요구사항의 청사진을 정의합니다.
  그런 다음 프로토콜을 클래스, 구조체, 열거형에서 채택하여 해당 요구사항의 실제 구현을 제공할 수 있습니다.
- 즉, 프로토콜에서는 **해야할 일을 정의**만 하고 **구현은 프로토콜을 채택한 객체**에서 이루어집니다.
- 자바의 interface와 비슷한 개념이라고 생각하셔도 괜찮습니다.(다른 특징들은 많지만요)

```
protocol MakeKoreaFoodDelegate{
  func makeBebimbab()
  func makeGukbab()
}
```

```
class JunradoPerdon: MakeKoreaFoodDelegate{
  func makeBebimbab(){
    print("고사리 들어간 비빔밥")
  }
  func makeGukbab(){
    print("콩나물 국밥")
  }
}
```

```
class GangwondoPerdon: MakeKoreaFoodDelegate{
  func makeBebimbab(){
    print("곤드레 나물 들어간 비빔밥")
  }
  func makeGukbab(){
    print("순대국밥")
  }
}
```

📌 Protocol

```
@MainActor protocol UITextViewDelegate
```

```
@MainActor protocol UITextFieldDelegate
```

```
@MainActor protocol UIPickerViewDelegate
```

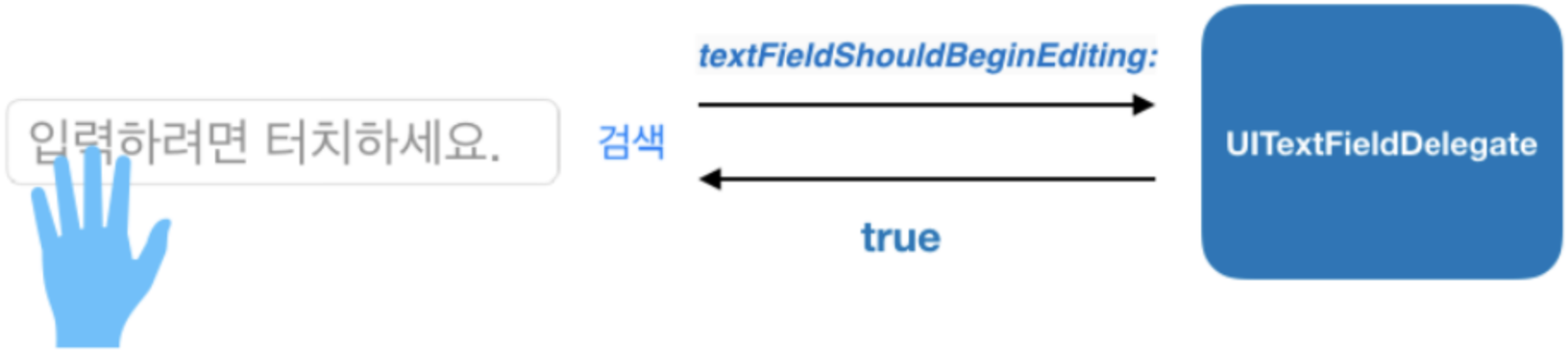
```
@MainActor protocol UITableViewDelegate
```

```
@MainActor protocol UICollectionViewDelegate
```

많이 사용하는 UIKit의 delegate들도
모두 Protocol로 되어있습니다
저 Protocol안에는 UIKit에서
위임받아 할 일들이 정의가 되어있겠죠?

📌 Delegate Pattern

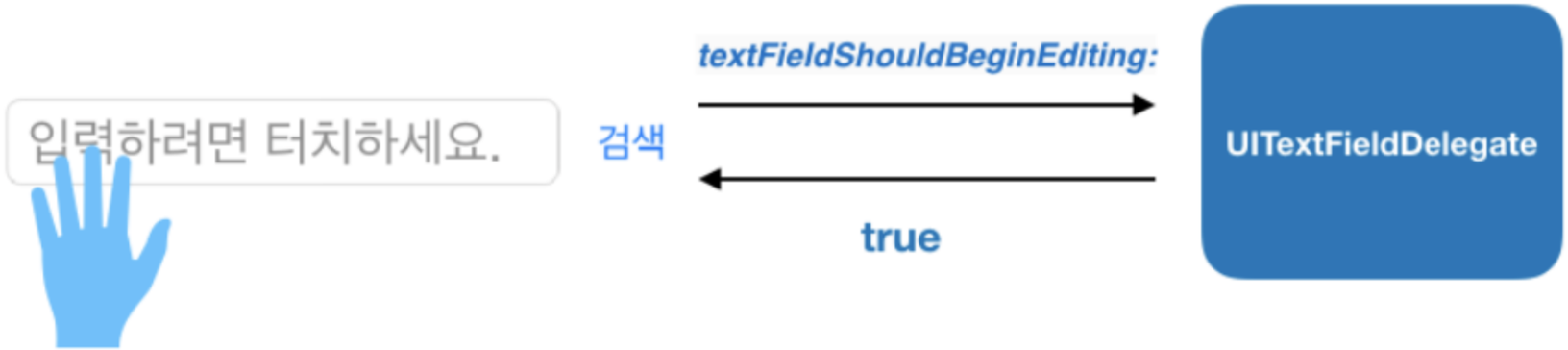
- Delegate Pattern은 쉽게 말해 객체지향 프로그래밍에서 하나의 객체가 모든 일을 처리하는 것이 아니라 **처리 해야 하는 일 중 일부를 다른 객체에 넘기는 것**을 말합니다.
- 주로 **프레임워크 객체가 위임을 요청**하며, **커스텀 컨트롤러 객체가 위임을 받아** 특정 이벤트에 대한 기능을 구현합니다.
- 커스텀 컨트롤러에서 세부 동작을 구현함으로써 동일한 동작에 대해 다양한 대응을 할 수 있게 해줍니다.
- Foundation, UIKit 그리고 Cocoa Touch 등 애플의 프레임워크에서 광범위하게 활용됩니다.



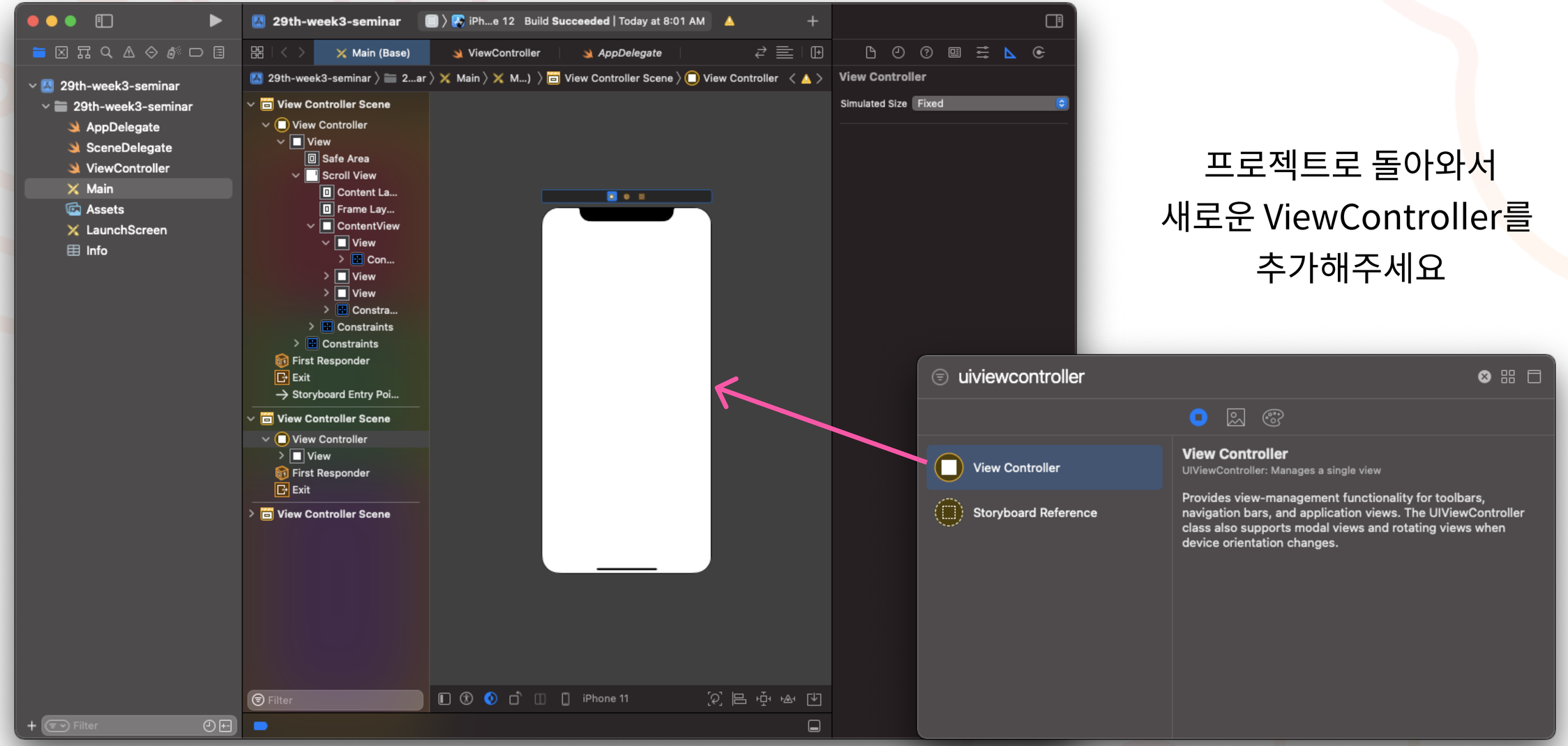
📌 Delegate Pattern

- Delegate Pattern은 쉽게 말해 객체지향 프로그래밍에서 하나의 객체가 모든 일을 처리하는 것이 아니라 **처리 해야 하는 일 중 일부를 다른 객체에 넘기는 것**을 말합니다.
- 주로 **프레임워크 객체가 위임을 요청**하며, **커스텀 컨트롤러 객체가 위임을 받아** 특정 이벤트에 대한 기능을 구현합니다.
- 커스텀 컨트롤러에서 세부 동작을 구현함으로써 동일한 동작에 대해 다양한 대응을 할 수 있게 해줍니다.
- Foundation, UIKit 그리고 Cocoa Touch 등 애플의 프레임워크에서 광범위하게 활용됩니다.

Delegate Pattern 연습해봅시다!

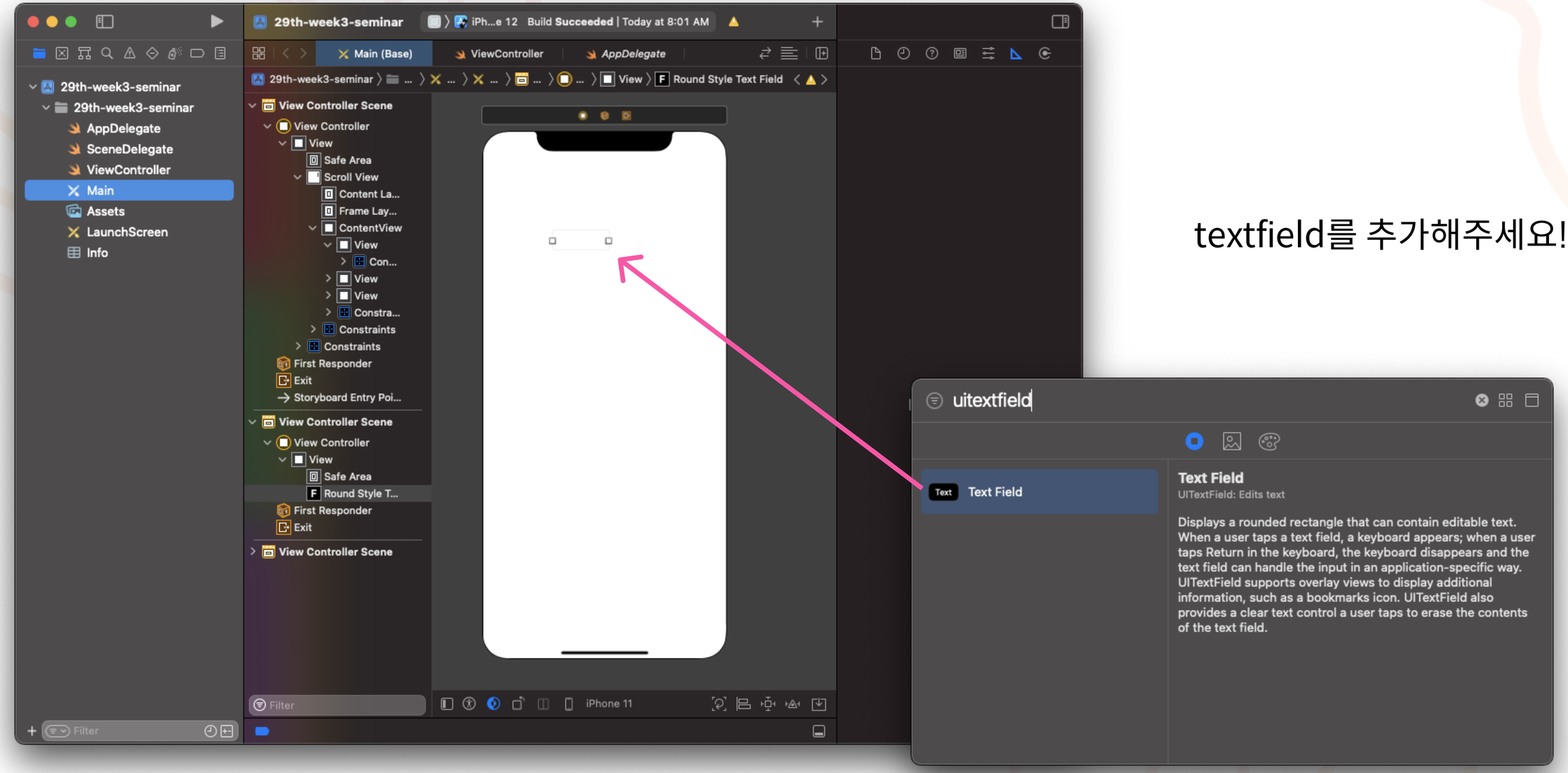


Delegate Pattern | UITextFieldDelegate



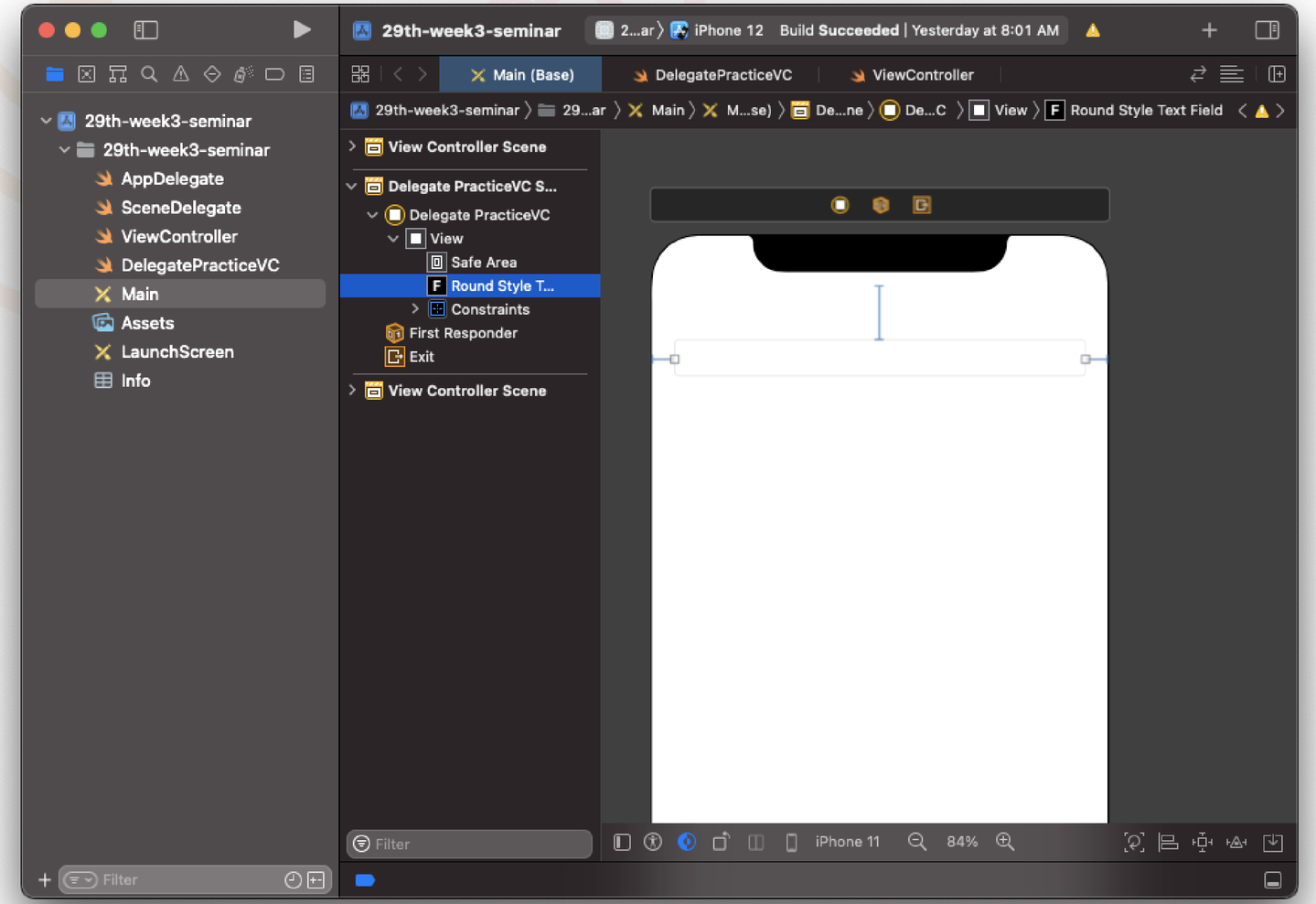
프로젝트로 돌아와서
새로운 ViewController를
추가해주세요

Delegate Pattern | UITextFieldDelegate



textfield를 추가해주세요!

Delegate Pattern | UITextFieldDelegate



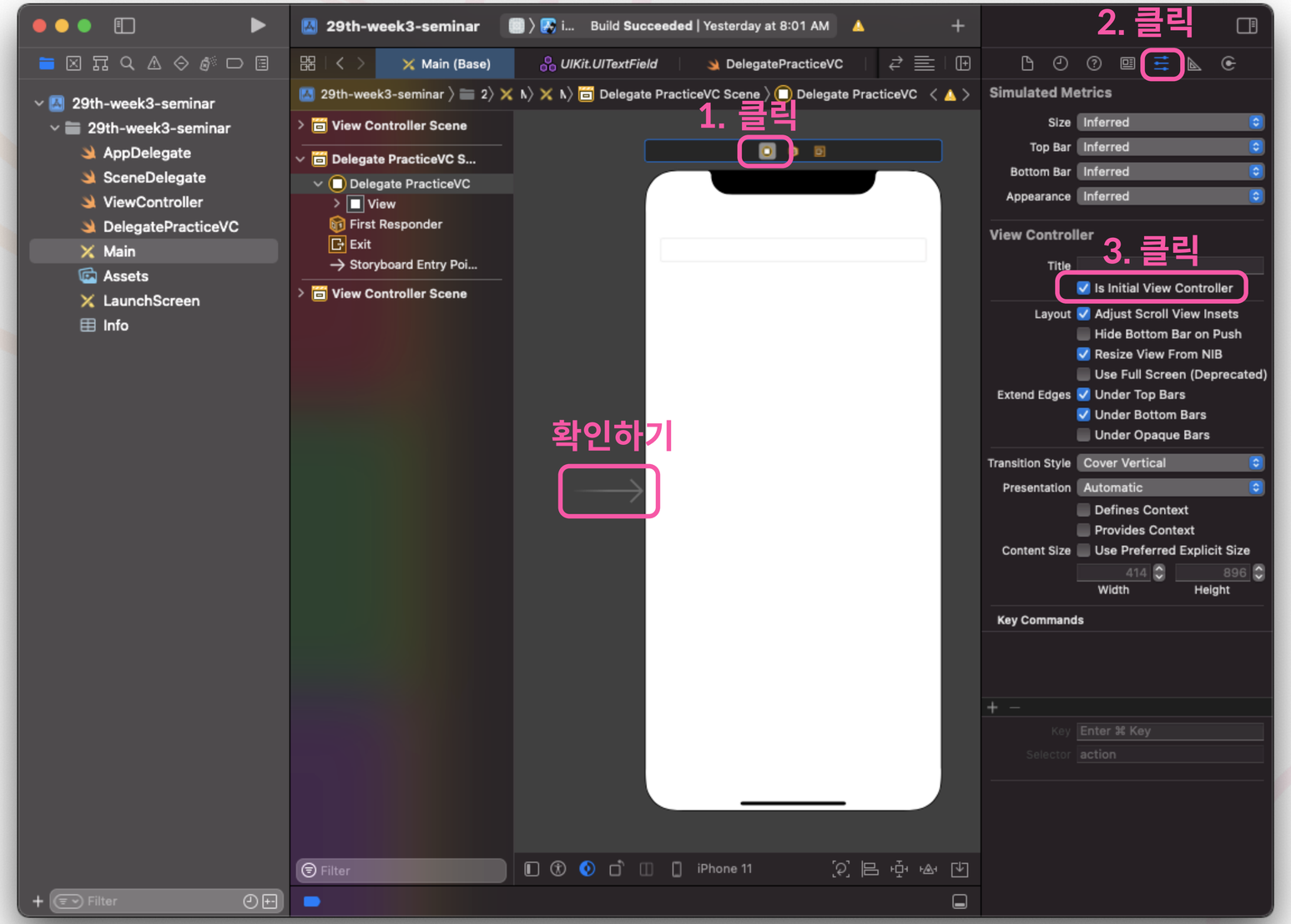
textfield에 constraint를 잡아주세요

top : 50

leading: 20

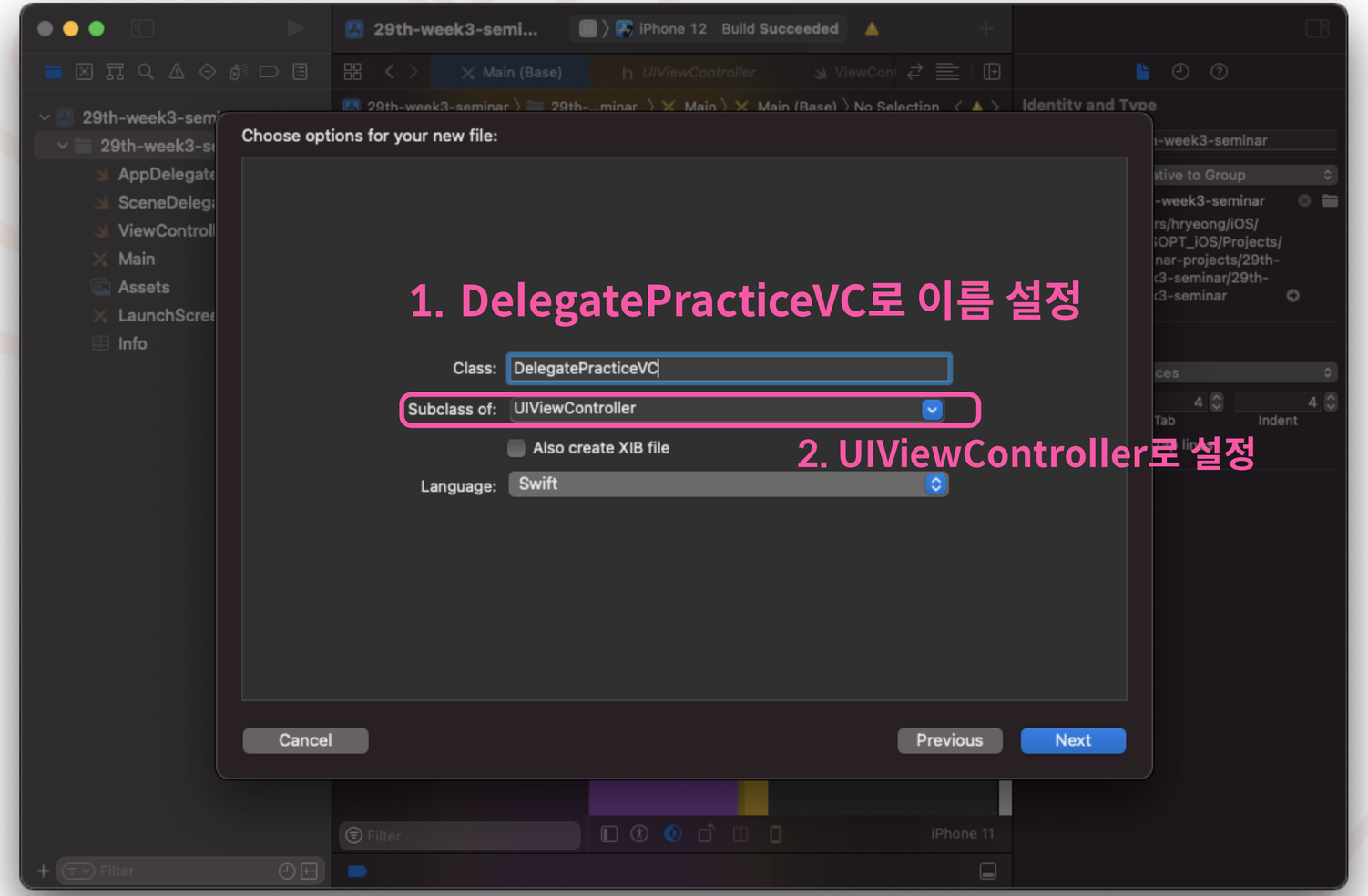
trailing : 20

Delegate Pattern | UITextFieldDelegate



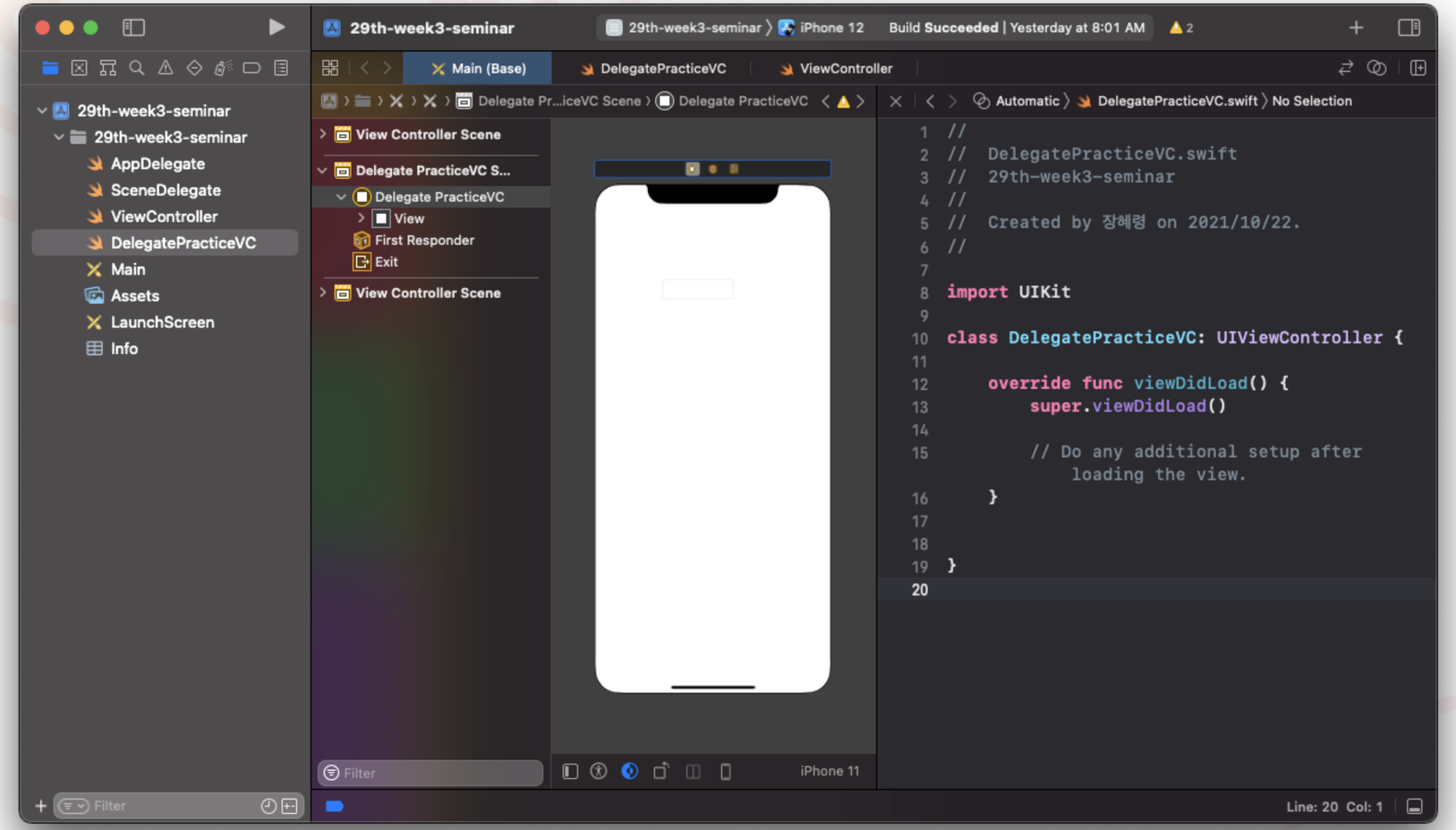
해당 ViewController를
Inital View Controller로
설정해주세요!

Delegate Pattern | UITextFieldDelegate



ViewController 파일을
새로 만들어주세요

Delegate Pattern | UITextFieldDelegate



방금 만든
DelegatePracticeVC와 연결하여
어시스턴트 창을 띄어주세요
control + option + command + enter

*코드창이 뜨지 않는다면
Xcode 한번 째려보고
껐다 키세요..

📌 UITextFieldDelegate

- 텍스트 필드 개체의 텍스트 편집 및 유효성 검사를 관리하는 선택적 메서드 집합입니다.
- 다양한 메소드 들이 존재 (https://developer.apple.com/documentation/uikit/uitextfielddelegate)

```
func textFieldShouldBeginEditing(UITextField) -> Bool
```
지정된 텍스트 필드에서 편집을 시작할지 여부를 대리자에게 묻습니다.

```
func textFieldDidBeginEditing(UITextField)
```
지정된 텍스트 필드에서 편집이 시작될 때 대리자에게 알려줍니다.

```
func textFieldShouldEndEditing(UITextField) -> Bool
```
지정된 텍스트 필드에서 편집을 중지할지 여부를 대리인에게 묻습니다.

```
func textFieldDidEndEditing(UITextField, reason: UITextField.DidEndEditingReason)
```
지정된 텍스트 필드에 대한 편집이 중지되는 시점과 중지된 이유를 대리자에게 알려줍니다.

```
func textFieldDidEndEditing(UITextField)
```
지정된 텍스트 필드에 대한 편집이 중지될 때 대리자에게 알립니다.

```
enum UITextField.DidEndEditingReason
```
텍스트 필드에서 편집을 종료하는 이유를 나타내는 상수입니다.

```
func textField(UITextField, shouldChangeCharactersIn: NSRange, replacementString: String) -> Bool
```
지정된 텍스트를 변경할지 여부를 대리자에게 묻습니다.

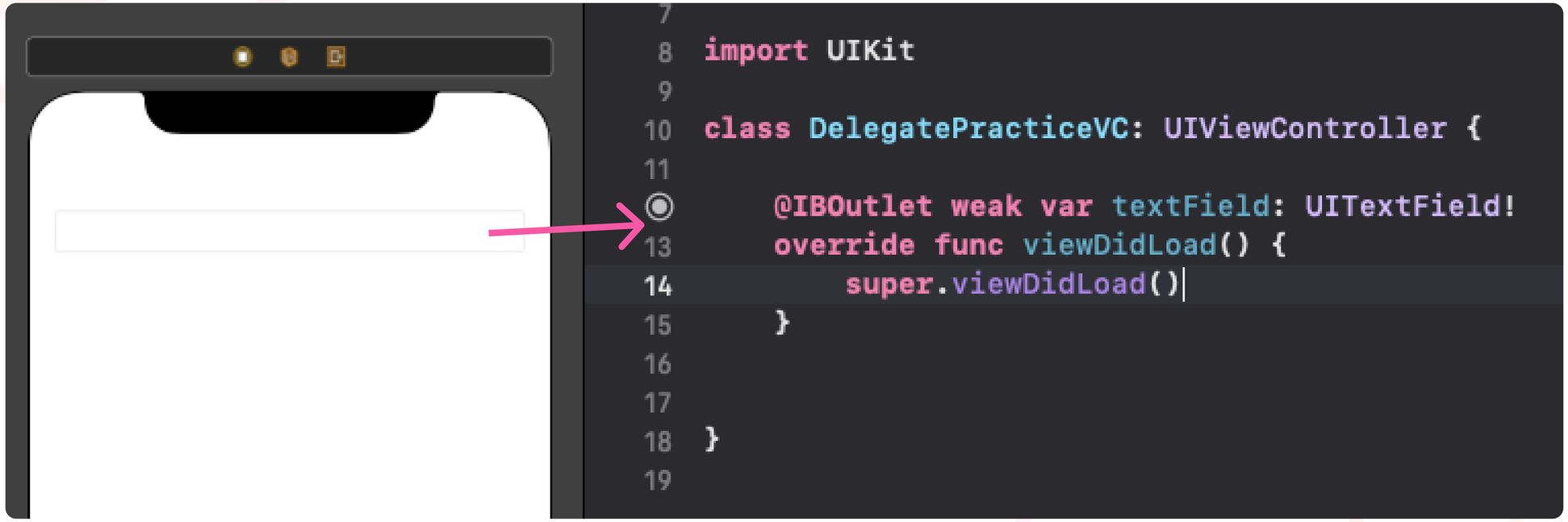
```
func textFieldShouldClear(UITextField) -> Bool
```
텍스트 필드의 현재 내용을 제거할지 여부를 대리자에게 묻습니다.

```
func textFieldShouldReturn(UITextField) -> Bool
```
대리자에게 텍스트 필드에 대한 Return 버튼 누름을 처리할지 여부를 묻습니다.

```
func textFieldDidChangeSelection(UITextField)
```
지정된 텍스트 필드에서 텍스트 선택이 변경되면 대리자에게 알립니다.

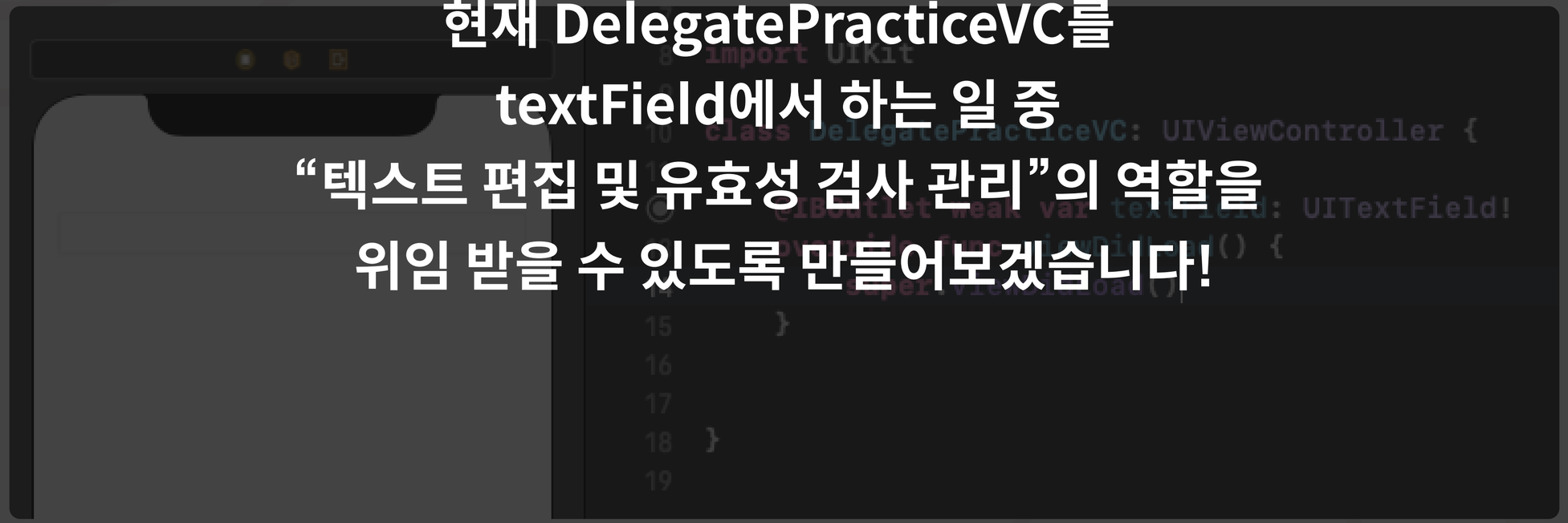
Delegate Pattern | UITextFieldDelegate

textfield outlet을 연결해주세요!



textfield outlet을 연결해주세요!

현재 DelegatePracticeVC를
textField에서 하는 일 중
“텍스트 편집 및 유효성 검사 관리”의 역할을
위임 받을 수 있도록 만들어보겠습니다!



Delegate Pattern | UITextFieldDelegate

class 밑에 extension을 사용하여 아래와 같이 코드를 작성해주세요 :)

```
class DelegatePracticeVC: UIViewController {

    @IBOutlet weak var textField: UITextField!
    override func viewDidLoad() {
        super.viewDidLoad()
    }

}

extension DelegatePracticeVC: UITextFieldDelegate{

}
```

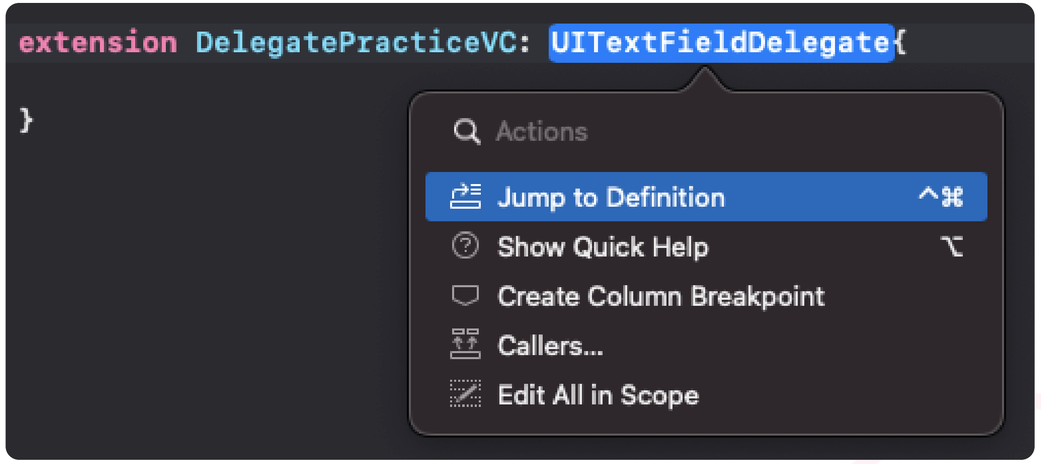
📌 extension

- 스위프트의 강력한 기능 중 하나로 구조체, 클래스, 열거형, 프로토콜 타입에 새로운 기능을 추가할 수 있습니다.
- 하지만, extension은 기존에 존재하는 기능을 재정의 할 수는 없습니다

```
extension 확장할 타입 이름 {
  타입에 추가될 새로운 기능 구현
  긴 VC안에 코드를 분리할 때도 잘 사용됨
}

extension 확장할 타입 이름: 프로토콜1, 프로토콜2, 프로토콜3{
  프로토콜 요구 사항 구현
}
```

Delegate Pattern | UITextFieldDelegate

이제 이 안에 위임받을(대신할) 기능을 넣어야하는데요!
어떤 기능이 있는지 보고싶을 때 공식문서를 보는 방법도 좋지만
궁금한 타입을 드래그한 후 command를 누르면 아래처럼 보입니다.
Jump to Definition (^⌘) 를 눌러봅시다!



Delegate Pattern | UITextFieldDelegate

이곳을 눌러 가보면, 실제 UITextFieldDelegate 안에
어떤 함수들이 있는지 확인할 수 있습니다.
이곳에서 저희가 대신할 기능을 선택하여 구현해봅시다!

```
extension DelegatePracticeVC: UITextFieldDelegate{

}
```

- Actions
- Jump to Definition ^⌘
- Show Quick Help ⌥
- Create Column Breakpoint
- Callers...
- Edit All in Scope

```
public protocol UITextFieldDelegate : NSObjectProtocol {


    @available(iOS 2.0, *)
    optional func textFieldShouldBeginEditing(_ textField: UITextField) -> Bool // return
        NO to disallow editing.

    @available(iOS 2.0, *)
    optional func textFieldDidBeginEditing(_ textField: UITextField) // became first
        responder

    @available(iOS 2.0, *)
    optional func textFieldShouldEndEditing(_ textField: UITextField) -> Bool // return
        YES to allow editing to stop and to resign first responder status. NO to disallow
        the editing session to end

    @available(iOS 2.0, *)
    optional func textFieldDidEndEditing(_ textField: UITextField) // may be called if
        forced even if shouldEndEditing returns NO (e.g. view removed from window) or
        endEditing:YES called

    @available(iOS 10.0, *)
    optional func textFieldDidEndEditing(_ textField: UITextField, reason:
        UITextField.DidEndEditingReason) // if implemented, called in place of
        textFieldDidEndEditing:


    @available(iOS 2.0, *)
    optional func textField(_ textField: UITextField, shouldChangeCharactersIn range:
        NSRange, replacementString string: String) -> Bool // return NO to not change text


    @available(iOS 13.0, *)
    optional func textFieldDidChangeSelection(_ textField: UITextField)


    @available(iOS 2.0, *)
    optional func textFieldShouldClear(_ textField: UITextField) -> Bool // called when
        clear button pressed. return NO to ignore (no notifications)

    @available(iOS 2.0, *)
    optional func textFieldShouldReturn(_ textField: UITextField) -> Bool // called when
        'return' key pressed. return NO to ignore.
}
```

Delegate Pattern | UITextFieldDelegate

func textfield~ 를 치면 자동완성으로 넣을 수 있는 메소드들이 나오는데
그 중에 아래 3가지 메소드를 추가해주세요!
그리고 그 안에서 각 메소드 실행을 구별할 수 있도록 적당한 Print 문을 넣어주세요:)

```
extension DelegatePracticeVC: UITextFieldDelegate{
    func textFieldDidBeginEditing(_ textField: UITextField) {
        print("textFieldDidBeginEditing")
    }

    func textFieldDidEndEditing(_ textField: UITextField) {
        print("textFieldDidEndEditing")
    }

    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        print("textFieldShouldReturn")
        textField.endEditing(true)
        return true
    }
}
```

- textFieldDidBeginEditing : 텍스트 필드를 터치하고 편집이 시작될 때 실행되는 메소드
- textFieldDidEndEditing : 텍스트 필드의 편집을 끝낼 때 실행 되는 메소드
- textFieldShouldReturn : 텍스트 필드 편집 중 키보드에서 'return'키를 눌렀을 때 실행되는 메소드

* textField.endEditing(true)
: 텍스트 필드의 편집을 끝내주는 메소드

Delegate Pattern | UITextFieldDelegate

```swift
extension DelegatePracticeVC: UITextFieldDelegate{
    func textFieldDidBeginEditing(_ textField: UITextField) {
        print("textFieldDidBeginEditing")
    }

    func textFieldDidEndEditing(_ textField: UITextField) {
        print("textFieldDidEndEditing")
    }

    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        print("textFieldShouldReturn")
        textField.endEditing(true)
        return true
    }
}
```

```swift
protocol MakeKoreaFoodDelegate{
  func makeBebimbab()
  func makeGukbab()
}
```

```swift
class JunradoPerdon: MakeKoreaFoodDelegate{
  func makeBebimbab(){
    print("고사리 들어간 비빔밥")
  }
  func makeGukbab(){
    print("콩나물 국밥")
  }
}
```

아직 끝이 아닙니다!
현재 여기까지 했다면, textfield가 하는 일을 대신하기 위해서 가져야할 조건인
UITextFieldDelegate protocol을 채택해서 해야할 함수들을 구현한 상태입니다.

전라도사람이란 클래스에서
MakeKoreaFoodDelegate를 채택해
비빔밥과 국밥을 만들 수 있는 함수를 구현함으로써
솝트 레스토랑에 한식 세프가 되기 위한 조건을 맞춘 상황!
그럼 실제로 솝트 레스토랑의 한식 세프로 공식 위임을 받아야겠죠?

Delegate Pattern | UITextFieldDelegate

```
class DelegatePracticeVC: UIViewController {

    @IBOutlet weak var textField: UITextField!
    override func viewDidLoad() {
        super.viewDidLoad()
        textField.delegate = self
    }
}

extension DelegatePracticeVC: UITextFieldDelegate{
    func textFieldDidBeginEditing(_ textField: UITextField) {
        print("textFieldDidBeginEditing")
    }

    func textFieldDidEndEditing(_ textField: UITextField) {
        print("textFieldDidEndEditing")
    }

    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        print("textFieldShouldReturn")
        textField.endEditing(true)
        return true
    }
}
```

여기서 말하는 self는 DelegatePracticeVC 자신을 말합니다.

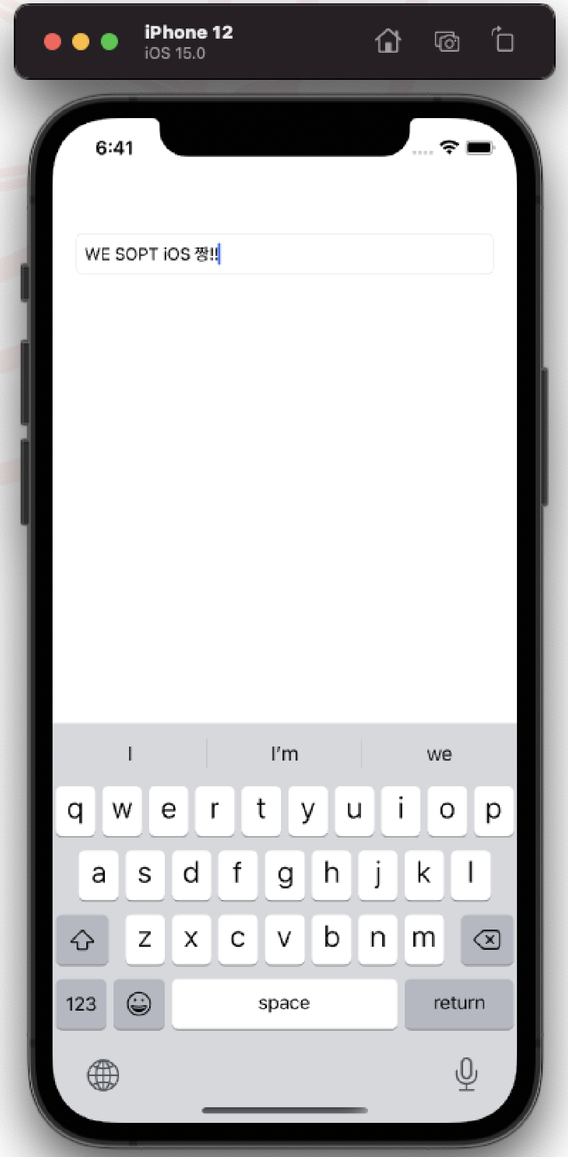
```
UITextField 〉 UITextField
    weak open var delegate: UITextFieldDelegate?
```

현재 DelegatePracticeVC는 extension으로
UITextFieldDelegate Protocol을 채택해
textfield의 기능을 위임받아 대신할 함수들을 넣어줬습니다.

textField의 대리자가 될 수 있는 조건이 갖춰졌기 때문에
대리자로 DelegatePracticeVC 를 넣을 수 있게 된 것입니다!

textfield.delegate = self 와
솝트레스토랑.한식셰프 = JunradoPerson()
같은 맥락입니다!

Delegate Pattern | UITextFieldDelegate

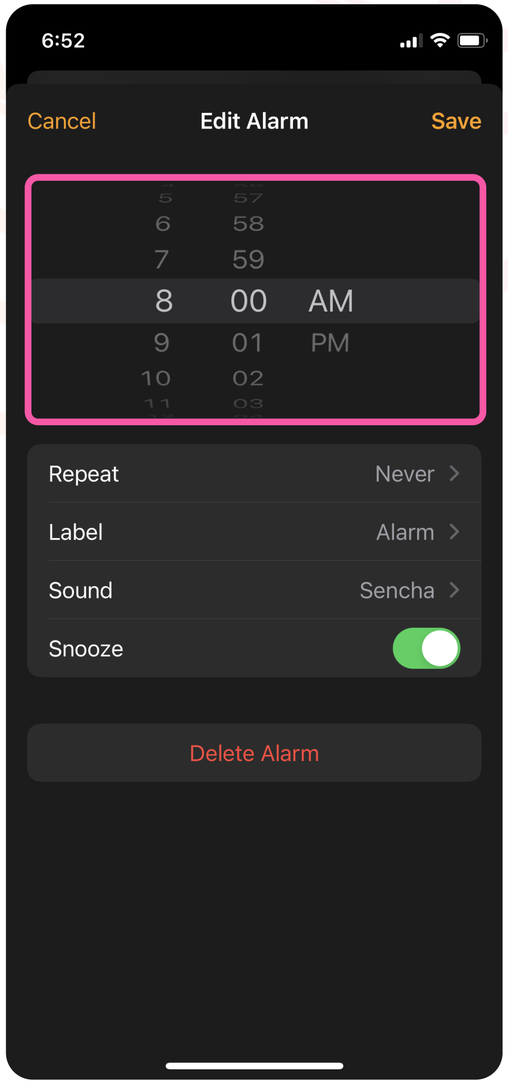



```
29th-week3-seminar
2021-10-22 06:40:23.220951+0900 29th-week3-seminar[21533:501640]
    Writing analzed variants.
2021-10-22 06:40:29.056249+0900 29th-week3-seminar[21533:501640]
    Writing analzed variants.
textFieldDidBeginEditing
textFieldShouldReturn
textFieldDidEndEditing
```

빌드시켜서 저희가 delegate 함수들 안에서
프린트문들이 원하는 때에 잘 찍히는지 확인해보세요:)

*commond(⌘) + k 누르면 키보드가 나옵니다

Delegate Pattern | UIPickerView



📌 UIPickerView

- Wheel 모양으로 된 항목들이 회전하면서 값을 선택할 수 있는 UI입니다.
- pickerview를 사용하기 위해서는 2가지리 Protocol을 채택해야합니다
- UIPickerViewDataSoucre : 데이터를 제어하는 기능 protocol
- UIPickerViewDelegate : 사용자 인터페이스(동작 처리) 제어하는 기능 protocol

피커 데이터 제공

`var dataSource: UIPickerViewDataSource?`

선택기 보기의 데이터 원본입니다.

`protocol UIPickerViewDataSource`

프로토콜은 객체로 채택해야하는 사이를 중개 객체와 그 피커 뷰에 대한 응용 프로그램의 데이터 모델. 데이터 소스는 선택기 보기 데이터를 표시하기 위해 구성요소 수와 각 구성요소의 행 수를 선택기 보기에 제공합니다. 이 프로토콜의 두 가지 방법이 모두 필요합니다.UIPickerViewDataSourceUIPickerView

선택기 동작 사용자 지정

`var delegate: UIPickerViewDelegate?`

선택기 보기의 대리자입니다.

`protocol UIPickerViewDelegate`

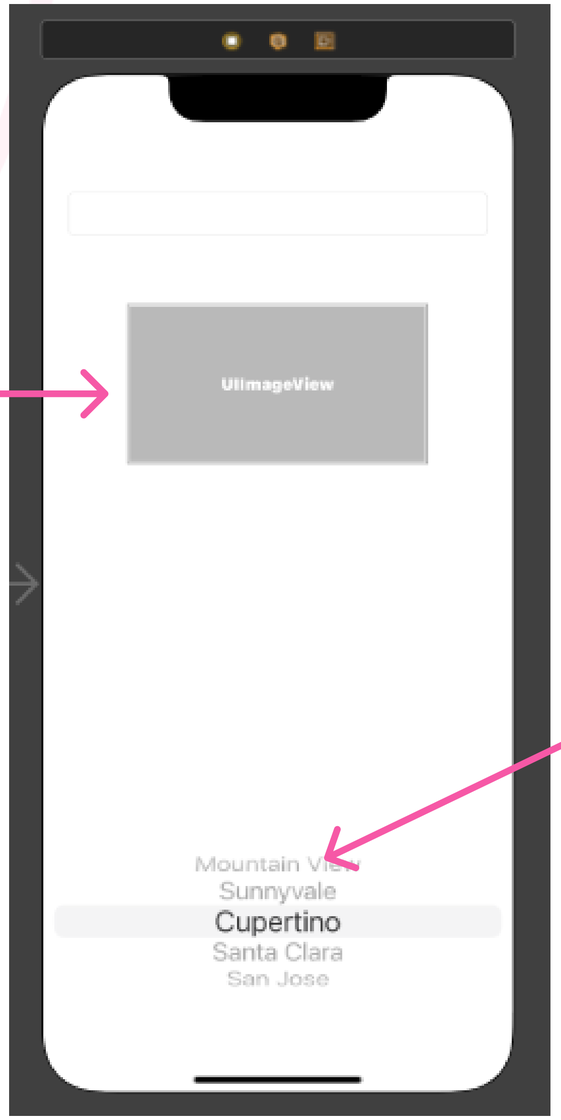
개체 의 대리자는 이 프로토콜을 채택하고 자체 구성에 필요한 데이터를 선택기 보기에 제공하기 위해 최소한 일부 메서드를 구현해야 합니다.UIPickerView

Delegate Pattern | UIPickerView

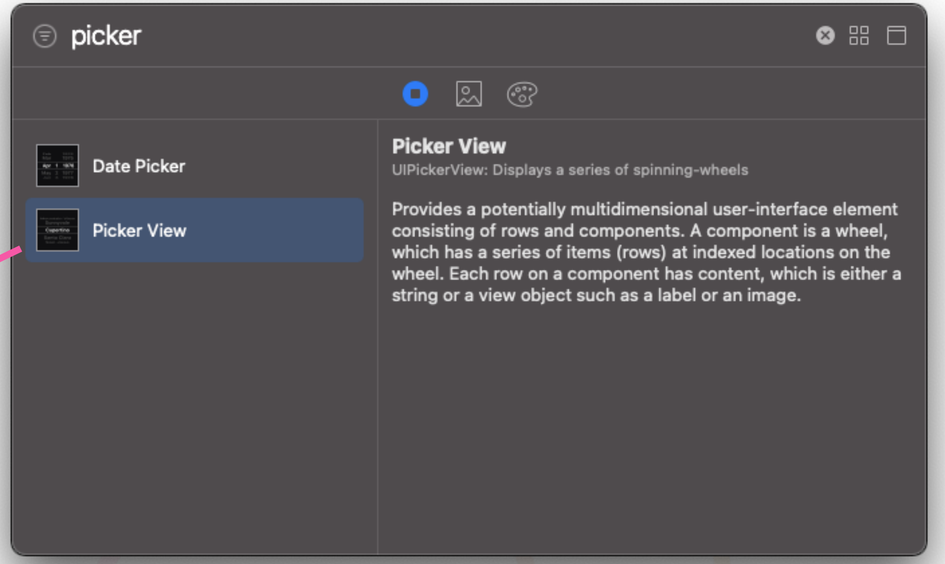
UIPickerView를 활용해서
만들어봅시다!

Delegate Pattern | UIPickerView

UIImageView 추가



pickerview 추가

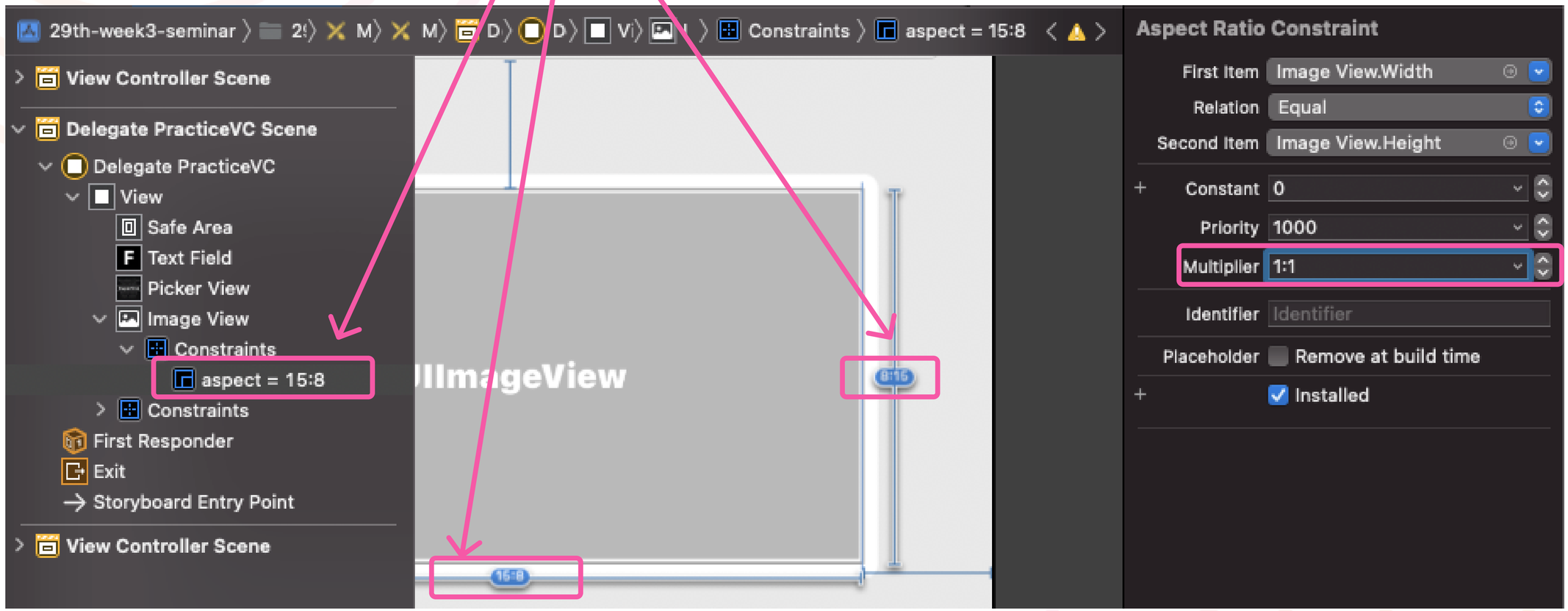


Delegate Pattern | UIPickerView

top, leading, trailing 50씩 설정하고
정사각형 크기로 만들기 위해
Aspect Ratio를 선택하여
총 4개의 Constraints를 추가해주세요!

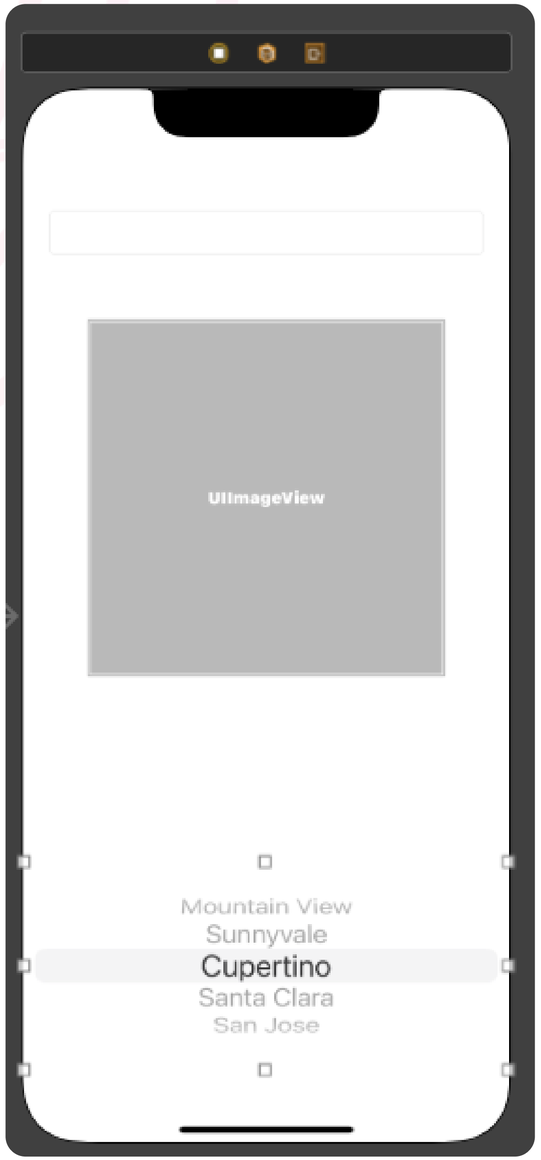
Delegate Pattern | UIPickerView

1. 셋 중에 하나를 선택



2. 1 : 1 로
바꿔주세요!

Delegate Pattern | UIPickerView

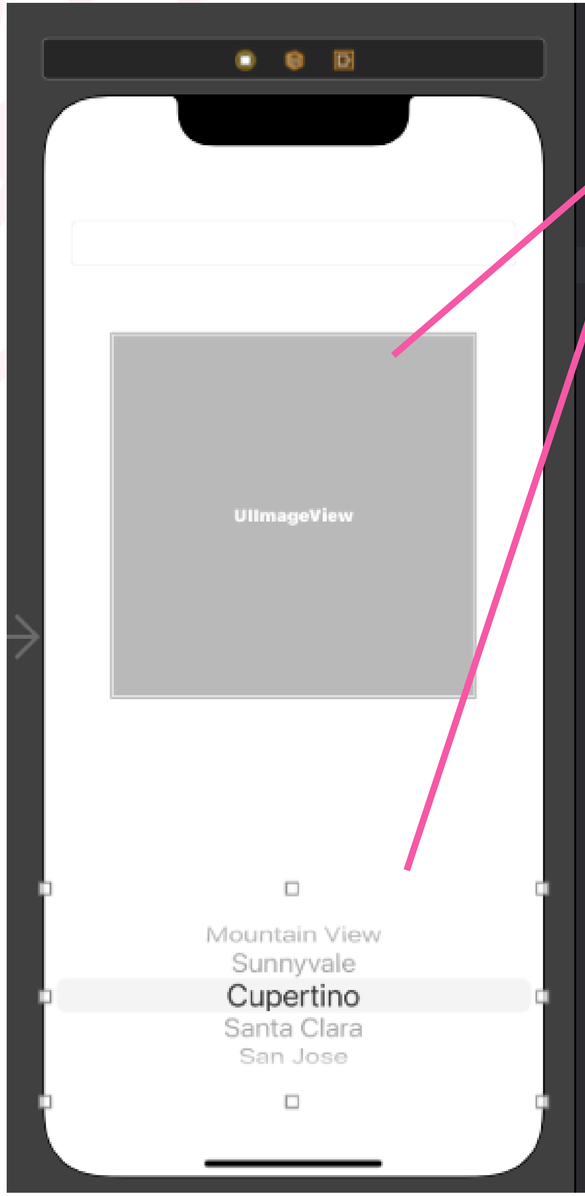


1. PickerView 클릭

2. leading, trailing, bottom 을 safe area에 0으로 맞춰주세요!
그 후 3가지 Constraints를 추가해주세요:)

Delegate Pattern | UIPickerView

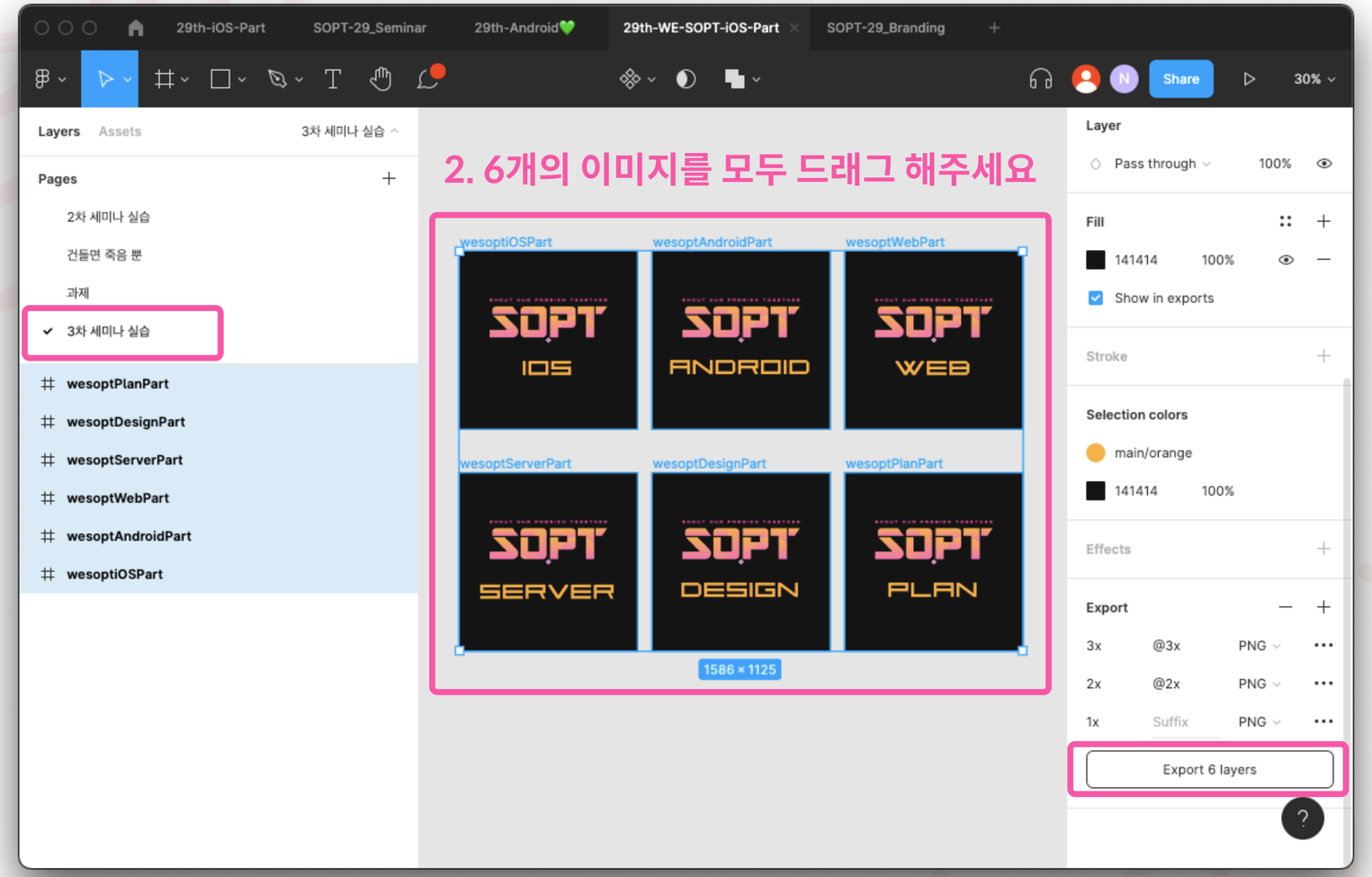
partImageView
partPickerView로
각각 outlet을
연결해주세요!




```
class DelegatePracticeVC: UIViewController {

    @IBOutlet weak var textField: UITextField!
    @IBOutlet weak var partImageView: UIImageView!
    @IBOutlet weak var partPickerView: UIPickerView!


    override func viewDidLoad() {
        super.viewDidLoad()
        textField.delegate = self
    }
}

extension DelegatePracticeVC: UITextFieldDelegate{
    func textFieldDidBeginEditing(_ textField:
        UITextField) {
        print("textFieldDidBeginEditing")
    }

    func textFieldDidEndEditing(_ textField: UITextField)
        {
        print("textFieldDidEndEditing")
    }

    func textFieldShouldReturn(_ textField: UITextField)
        -> Bool {
        print("textFieldShouldReturn")
        textField.endEditing(true)
        return true
    }
}
```

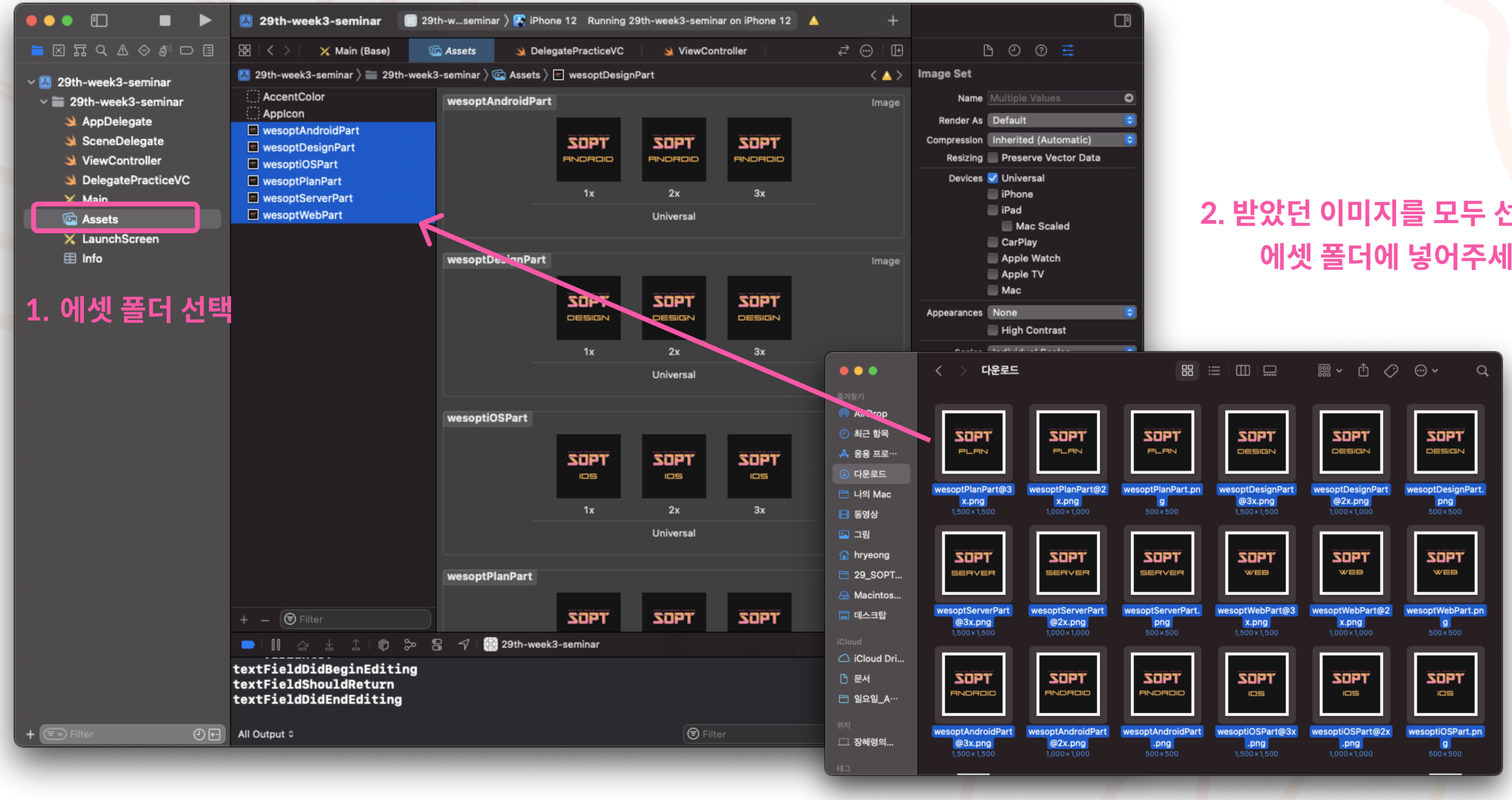
Delegate Pattern | UIPickerView

1. 피그마에서
3차 세미나 실습 페이지로
들어가주세요



3. 클릭하여 이미지를
받아주세요

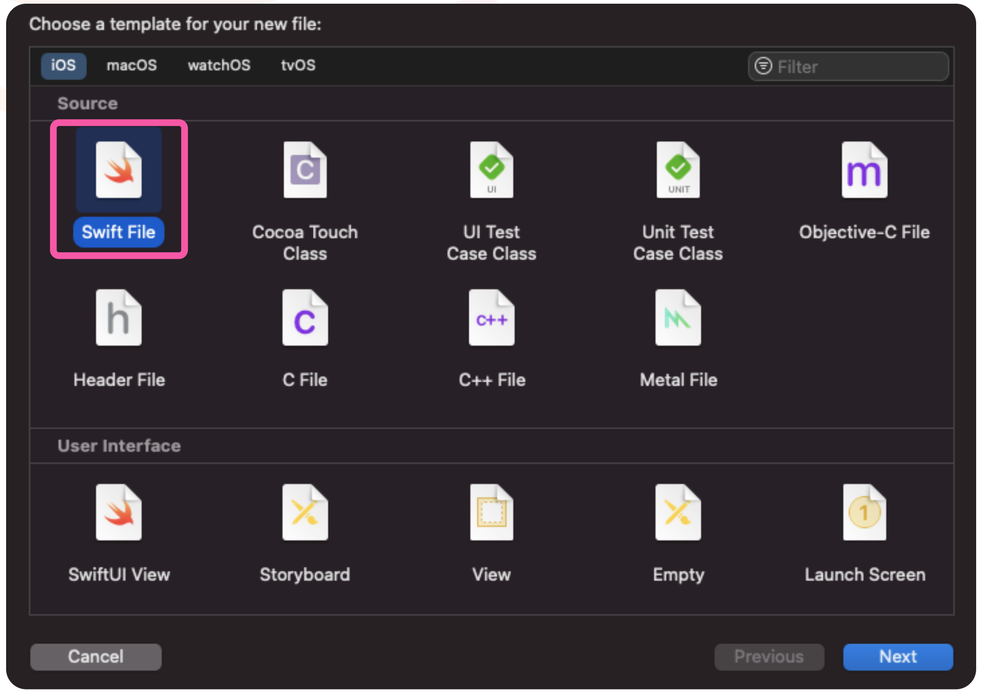
Delegate Pattern | UIPickerView



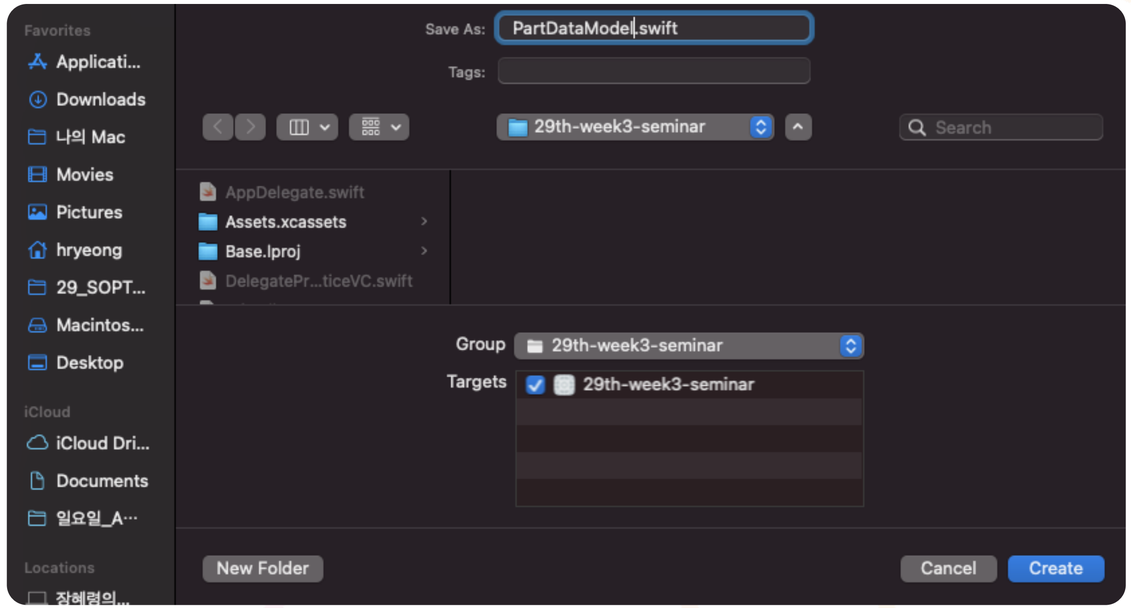
Delegate Pattern | UIPickerView

다음으로, Part별 이미지와 이름을 관리하기 쉽도록
Struct 구조체를 만들어 보겠습니다.

1. 파일추가(cmd+n)를 눌러 Swift 파일을 선택한 후 Next



2. 파일명 “PartDataModel.swift” Create



Delegate Pattern | UIPickerView

```
import UIKit

struct PartData{
    var imageName: String
    var partName: String
}
```

‘PartData’라는 모델을 ‘구조체(struct)’로 만들어 봅시다!

Delegate Pattern | UIPickerView

UIImage는
Foundation 프레임 워크에는 없고
UIKit에 있기때문에 Foundation을 지우고
import UIKit으로 변경해주세요

```
import UIKit

struct PartData{
    var imageName: String
    var partName: String

    func makeImage() -> UIImage? {
        return UIImage(named: imageName)
    }
}
```

image 이름으로 UIImage를 생성하는 메소드 추가해주세요

Delegate Pattern | UIPickerView

DelegatePracticeVC.swift 파일로 돌아옵시다

```
class DelegatePracticeVC: UIViewController {

    @IBOutlet weak var textField: UITextField!
    @IBOutlet weak var partImageView: UIImageView!
    @IBOutlet weak var partPickerView: UIPickerView!

    var partList : [PartData] = []

    override func viewDidLoad() {
        super.viewDidLoad()
        textField.delegate = self
        initPartData()
    }

    func initPartData(){

    }
}
```

Part 정보를 저장할 배열을 멤버 변수로 선언

view가 로드될 시에 호출될 수 있도록 viewDidLoad에 메소드 호출

partList에 초기 데이터를 넣을 메소드 추가

Delegate Pattern | UIPickerView

```
func initPartData(){
    partList.append(contentsOf: [
        PartData(imageName: "wesoptiOSPart", partName: "iOS"),
        PartData(imageName: "wesoptAndroidPart", partName: "Android"),
        PartData(imageName: "wesoptWebPart", partName: "Web"),
        PartData(imageName: "wesoptServerPart", partName: "Server"),
        PartData(imageName: "wesoptDesignPart", partName: "Design"),
        PartData(imageName: "wesoptPlanPart", partName: "Plan")
    ])
}
```

PartData 생성하여 partList에 추가

노션에서 코드를 복사해서 올 때 indent(들여쓰기)가 잘 안 맞는다면
그 부분을 드래그해서 control(^) + i 를 눌러주면 Xcode가 맞춰줍니다:)

Delegate Pattern | UIPickerView

UIPickerView에서 채택하여 구현해야할 Protocol은 2가지가 있습니다!

```
extension DelegatePracticeVC: UIPickerViewDataSource{
}

extension DelegatePracticeVC: UIPickerViewDelegate{
}
```

데이터를 제어하는 기능 Protocol

사용자 동작 처리하는 기능 Protocol

Delegate Pattern | UIPickerView

저희가 앞서 사용했던 UITextFieldDelegate나,
UIPickerViewDelegate와는 다르게 오류가 발생하고 있습니다.

클릭

```
extension DelegatePracticeVC: UIPickerViewDataSource{
}
```

Type 'DelegatePracticeVC' does n...

Type 'DelegatePracticeVC' does not conform to protocol 'UIPickerViewDataSource'

Do you want to add protocol stubs? Fix

Delegate Pattern | UIPickerView

```
public protocol UIPickerViewDataSource : NSObjectProtocol {

    // returns the number of 'columns' to display.
    @available(iOS 2.0, *)
    func numberOfComponents(in pickerView: UIPickerView) -> Int

    // returns the # of rows in each component..
    @available(iOS 2.0, *)
    func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int
}
```

UIPickerViewDataSource에서 구현해야할 함수들은 optional이 붙지 않은, 즉, 필수적으로 구현해야하는 함수들입니다.

```
public protocol UIPickerViewDelegate : NSObjectProtocol {

    // returns width of column and height of row for each
    @available(iOS 2.0, *)
    optional func pickerView(_ pickerView: UIPickerView, 

    @available(iOS 2.0, *)
    optional func pickerView(_ pickerView: UIPickerView, 
        CGFloat
```

그와 다르게 UIPickerViewDelegate에서 구현할 함수들 앞에는 optional이 붙어 **선택적으로** 구현하면 되는 함수들입니다

Delegate Pattern | UIPickerView

클릭

```
extension DelegatePracticeVC: UIPickerViewDataSource{ ⊙ Type 'DelegatePracticeVC' does n...

}
```

Type 'DelegatePracticeVC' does not conform to protocol 'UIPickerViewDataSource'

Do you want to add protocol stubs? Fix

Fix를 눌러주세요

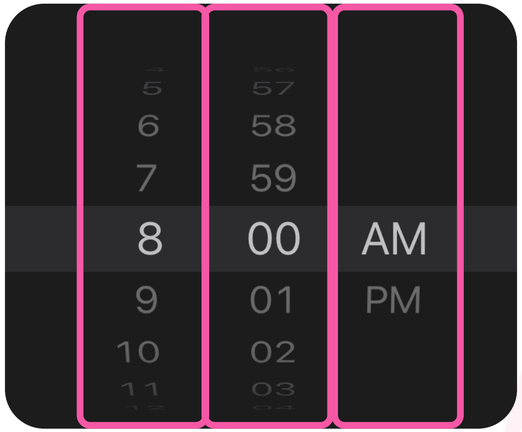
Xcode에서 필수적으로 구현해야하는
함수들을 자동으로 넣어줍니다!

```
extension DelegatePracticeVC: UIPickerViewDataSource{
    func numberOfComponents(in pickerView: UIPickerView) -> Int {
        code
    }

    func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int {
        code
    }
}
```

Delegate Pattern | UIPickerView

```
extension DelegatePracticeVC: UIPickerViewDataSource{
    func numberOfComponents(in pickerView: UIPickerView) -> Int {
        // PickerView의 컴포넌트 숫자를 지정해 주는 메소드
    }

    func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int {
        // PickerView의 각 컴포넌트에서 행의 개수를 지정해 주는 메소드
    }
}
```

Component 개수는 3개

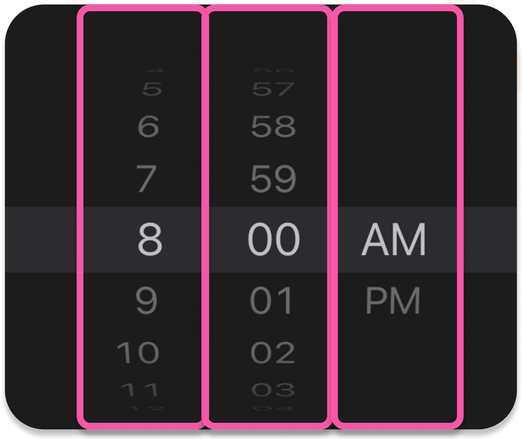


컴포넌트 0 -> row 12
컴포넌트 1 -> row 60
컴포넌트 2 -> row 2

Delegate Pattern | UIPickerView

```
extension DelegatePracticeVC: UIPickerViewDataSource{
    func numberOfComponents(in pickerView: UIPickerView) -> Int {
        return 3
    }

    func pickerView(_ pickerView: UIPickerView,
                    numberOfRowsInComponent component: Int) -> Int {
        switch component {
        case 0:
            return 12
        case 1:
            return 60
        case 2:
            return 2
        default:
            return 0
        }
    }
}
```

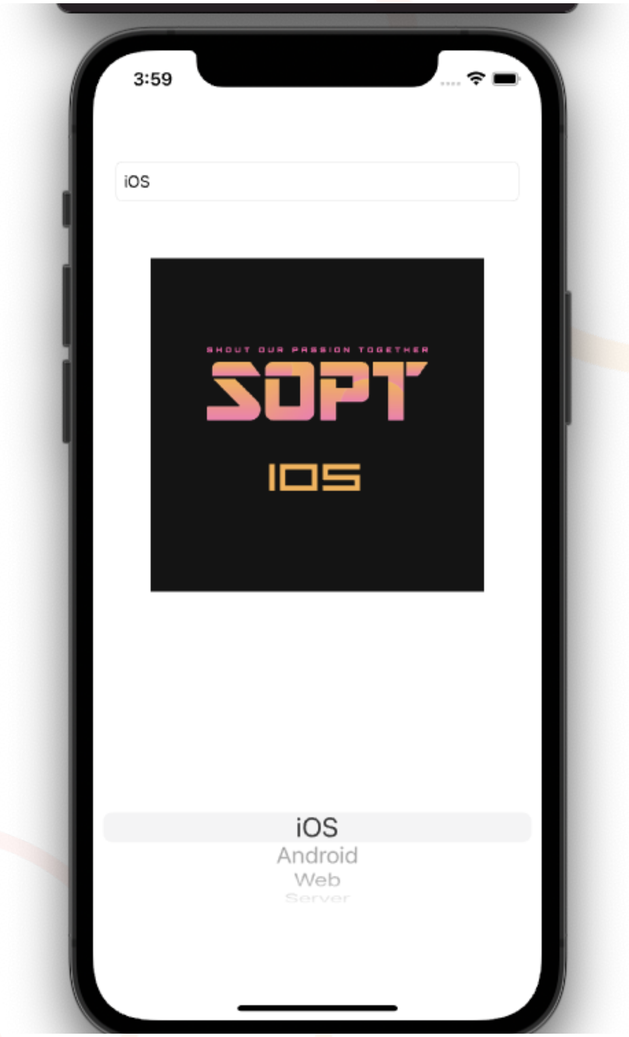
Component 개수는 3개



컴포넌트 0 -> row 12
컴포넌트 1 -> row 60
컴포넌트 2 -> row 2

Delegate Pattern | UIPickerView

```
extension DelegatePracticeVC: UIPickerViewDataSource{
    func numberOfComponents(in pickerView: UIPickerView) -> Int {
        // PickerView의 컴포넌트 숫자를 지정해 주는 메소드
        return 1
    }

    func pickerView(_ pickerView: UIPickerView,
                    numberOfRowsInComponent component: Int) -> Int {
        // PickerView의 각 컴포넌트에서 행의 개수를 지정해 주는 메소드
        // partList 데이터의 개수 만큼의 행이 필요
        return partList.count
    }
}
```



Delegate Pattern | UIPickerView

```
class DelegatePracticeVC: UIViewController {

    @IBOutlet weak var textField: UITextField!
    @IBOutlet weak var partImageView: UIImageView!
    @IBOutlet weak var partPickerView: UIPickerView!

    var partList : [PartData] = []

    override func viewDidLoad() {
        super.viewDidLoad()
        initPartData()
        textField.delegate = self
        partPickerView.dataSource = self
    }
```

PickerView의 데이터를
제어하는 기능(DataSource)을
뷰 컨트롤러에 위임

📌 지금까지 실행 결과

- DataSource 프로토콜을 구현 후 위임하였습니다.
- 컴포넌트의 개수와 행의 개수만을 지정한 상태입니다.
- 각행을 채우는 동작과 선택되었을 때 동작하는 부분을 아직 구현하지 않았습니다!
- UIPickerViewDelegate 프로토콜을 통해 이 부분을 구현해봅시다

*UIPickerViewDelegate(https://developer.apple.com/documentation/uikit/uipickerviewdelegate)

Delegate Pattern | UIPickerView

```
public protocol UIPickerViewDelegate : NSObjectProtocol {


    // returns width of column and height of row for each component.
    @available(iOS 2.0, *)
    optional func pickerView(_ pickerView: UIPickerView, widthForComponent component: Int) -> CGFloat

    @available(iOS 2.0, *)
    optional func pickerView(_ pickerView: UIPickerView, rowHeightForComponent component: Int) -> CGFloat


    // these methods return either a plain NSString, a NSAttributedString, or a view (e.g UILabel) to
        display the row for the component.
    // for the view versions, we cache any hidden and thus unused views and pass them back for reuse.
    // If you return back a different object, the old one will be released. the view will be centered in
        the row rect
    @available(iOS 2.0, *)
    optional func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int)
        -> String?

    @available(iOS 6.0, *)
    optional func pickerView(_ pickerView: UIPickerView, attributedTitleForRow row: Int, forComponent
        component: Int) -> NSAttributedString? // attributed title is favored if both methods are
        implemented

    @available(iOS 2.0, *)
    optional func pickerView(_ pickerView: UIPickerView, viewForRow row: Int, forComponent component: Int,
        reusing view: UIView?) -> UIView


    @available(iOS 2.0, *)
    optional func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int, inComponent component: Int)
}
```

행 내용을 그리는 데 사용할 행 높이가 필요할 때

행 내용을 그리는 데 사용할 행 너비가 필요할 때

주어진 구성 요소의 주어진 행에 사용할 제목이 필요할 때

주어진 구성 요소의 주어진 행에
사용할스타일이 지정된 제목이 필요할 때

지정된 구성 요소의 지정된 행에 사용할 뷰가 필요할 때

사용자가 구성 요소에서 행을 선택할 때

Delegate Pattern | UIPickerView

```
extension DelegatePracticeVC: UIPickerViewDelegate{
    func title
}
    M pickerView(_ pickerView: UIPickerView, titleForRow ro
```

title을 치면 자동완성에
titleForRow가 나옵니다
클릭해서 해당 메소드를 추가해주세요!

```
func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int) ->
    String? {
    code
}
```

```
func didselec
  M pickerView(_ pickerView: UIPickerView, didSelectRow row: Int,
  M textFieldDidChangeSelection(_ textField: UITextField)
```

didSelect을 치면 자동완성에
didSelectRow가 나옵니다
클릭해서 해당 메소드를 추가해주세요!

```
func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int, inComponent component: Int) {
    code
}
```

Delegate Pattern | UIPickerView

```
// PickerView의 각 행(row)의 title을 지정해주는 메소드
func pickerView(_ pickerView: UIPickerView,
                titleForRow row: Int,
                forComponent component: Int) -> String? {
    //partList에서 row번째에 있는 PartData의 partName으로 title 설정
    return partList[row].partName
}
```

각 row에 들어갈 title

partList에서 row번째에 있는 PartData의 partName으로 title 설정합니다.

Delegate Pattern | UIPickerView

```
// PickerView에서 행(row)를 선택했을 때 수행할 동작을 지정해 주는 메소드
func pickerView(_ pickerView: UIPickerView,
                didSelectRow row: Int, inComponent component: Int) {
    //partList에서 row번째에 있는 PartData의 UIimage를 가져오는 함수 호출하여
    //partImageView의 image로 설정합니다.
    partImageView.image = partList[row].makeImage()

    //partList에서 row번째에 있는 PartData의 partName을 가져와
    //textfield의 text로 지정합니다.
    textField.text = partList[row].partName
}
```

특정 row가 선택되었을 때 수행할 동작 지정

1. partList에서 row번째에 있는 PartData의 UIimage를 가져오는 함수 호출하여 partImageView의 image로 설정합니다
2. partList에서 row번째에 있는 PartData의 partName을 가져와 textfield의 text로 지정합니다.

Delegate Pattern | UIPickerView

```ios/DelegatePracticeVC.swift
extension DelegatePracticeVC: UIPickerViewDelegate{

    // PickerView의 각 행(row)의 title을 지정해주는 메소드
    func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int) ->
        String? {
        //partList에서 row번째에 있는 PartData의 partName으로 title 설정
        return partList[row].partName
    }

    // PickerView에서 행(row)를 선택했을 때 수행할 동작을 지정해 주는 메소드
    func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int, inComponent component: Int) {
        //partList에서 row번째에 있는 PartData의 UIimage를 가져오는 함수 호출하여
        //partImageView의 image로 설정합니다.
        partImageView.image = partList[row].makeImage()

        //partList에서 row번째에 있는 PartData의 partName을 가져와
        //textfield의 text로 지정합니다.
        textField.text = partList[row].partName
    }

}
```

Delegate Pattern | UIPickerView

```
func initPartLayout(){
    partImageView.image = partList[0].makeImage()
    textField.text = partList[0].partName
}
```

현재는 imageview와 textfield에 초기값을 지정해주지 않았기 때문에
빌드시 처음에는 아무것도 뜨지 않습니다!
그래서 이 함수를 view did load에 추가하여
imageView와 textfield에 초기값을 지정해줄 수 있습니다

Delegate Pattern | UIPickerView

```
class DelegatePracticeVC: UIViewController {

    @IBOutlet weak var textField: UITextField!
    @IBOutlet weak var partImageView: UIImageView!
    @IBOutlet weak var partPickerView: UIPickerView!

    var partList : [PartData] = []

    override func viewDidLoad() {
        super.viewDidLoad()
        initPartData()
        textField.delegate = self
        partPickerView.dataSource = self
        partPickerView.delegate = self
    }
```

PickerView의 동작을 제어하는
기능(Delegate)을 뷰 컨트롤러에 위임

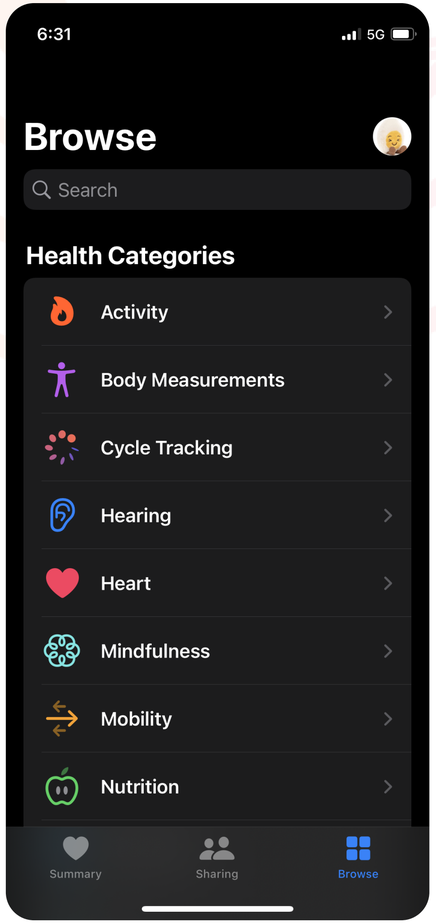
Delegate Pattern | UIPickerView

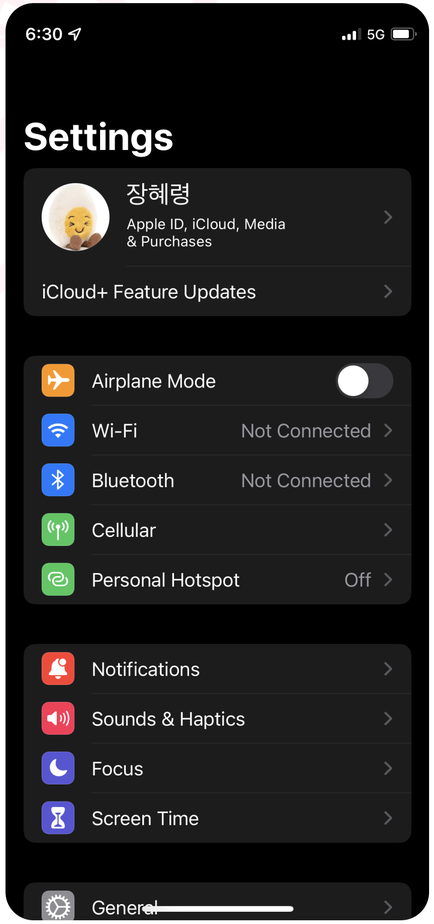
빌드시켜 잘 작동되는지 확인해보세요:)

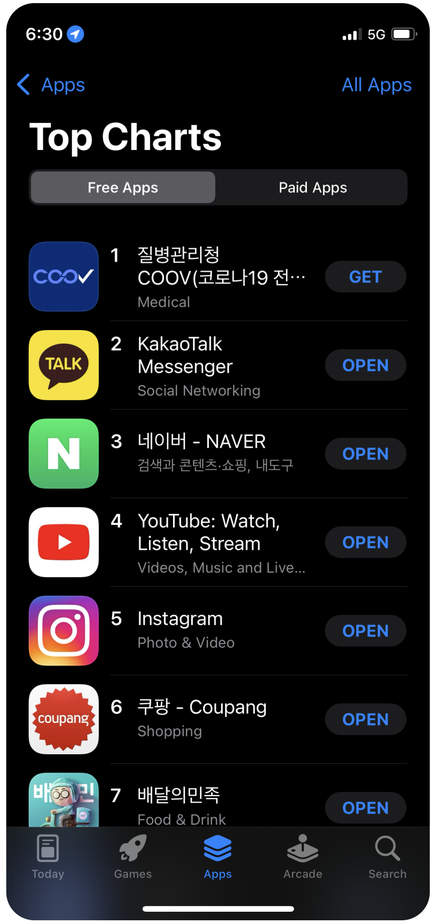
03

TableView

UITableView



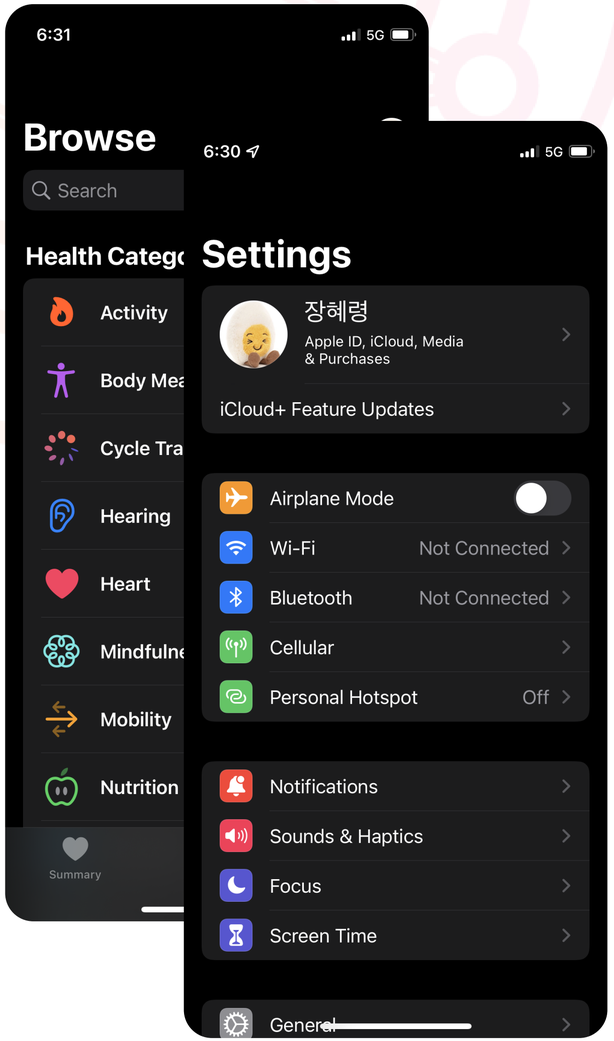




왼쪽과 같은 화면 구성을
iOS 앱을 사용하다 보면
정말 많이 보셨을거에요!

바로 이 뷰들을 구성할 수 있는
가장 대표적인 방식이
UITableView입니다

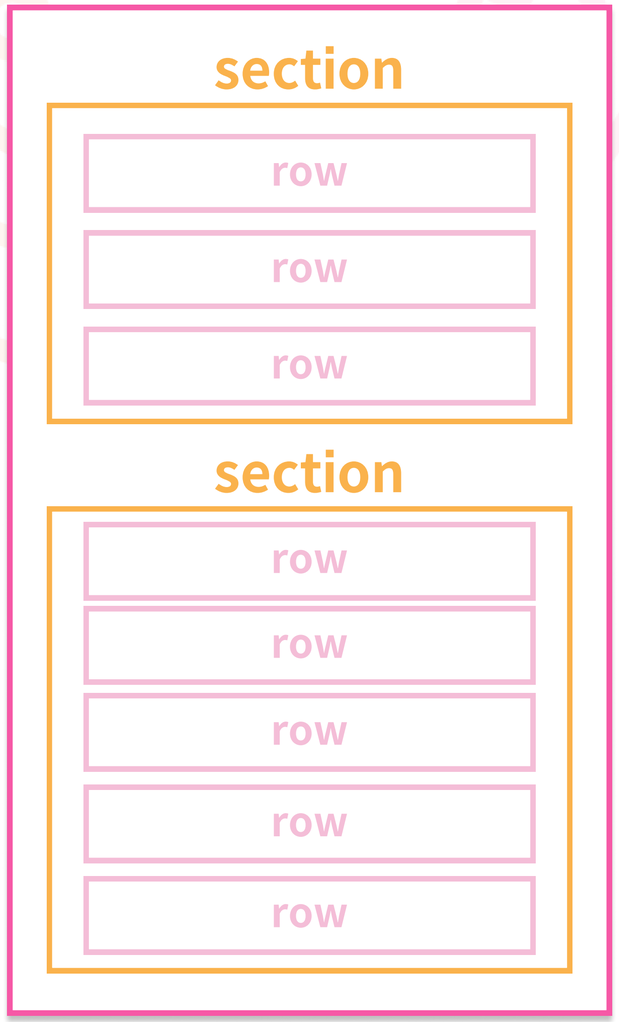
UITableView

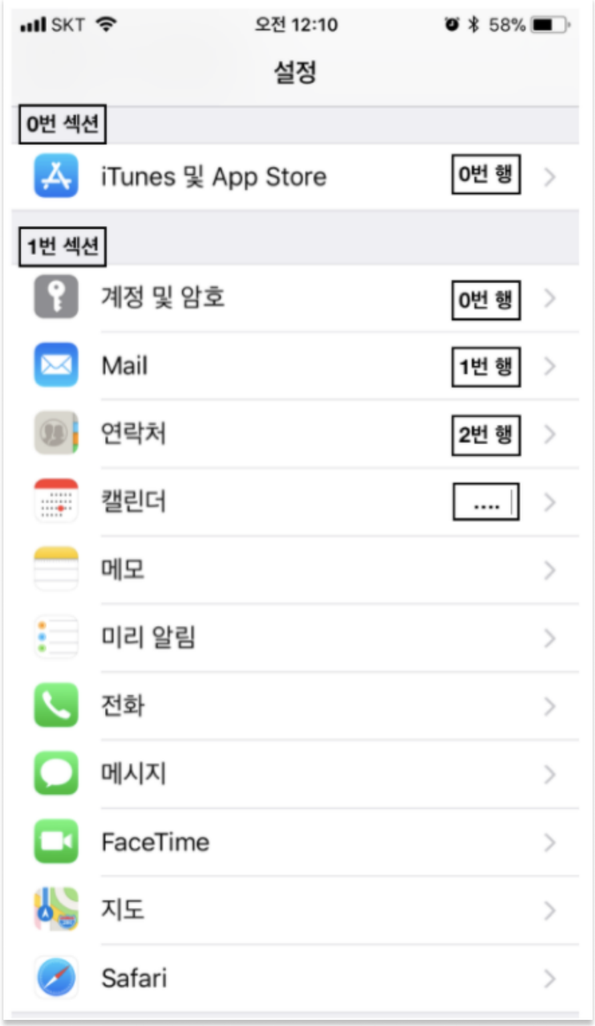


📌 UITableView

- 여러 아이템을 테이블(리스트) 형태로 보여주기 위한 뷰입니다.
- iOS 개발을 하다 보면 자주 사용하게 되는 인터페이스입니다.
- 리스트 형태로 이루어져 있으면 스크롤이 가능하고
  데이터를 동적으로 받아서 표현할 수 있습니다.
- UIScrollView의 상속을 받은 객체입니다.
- Delegate Pattern을 사용해 구현하는 인터페이스입니다.

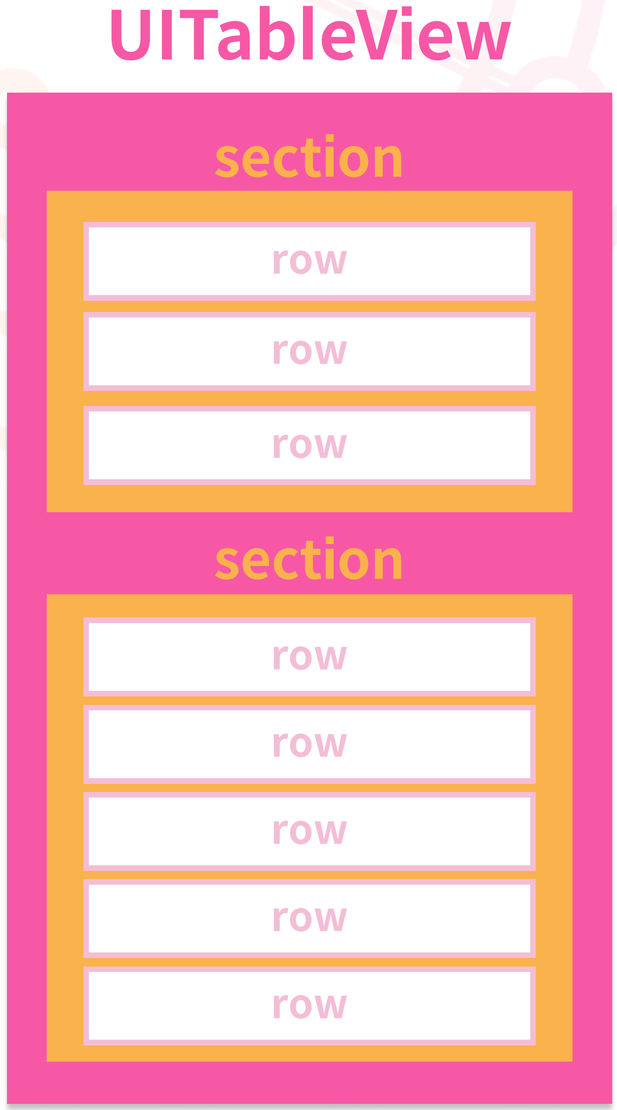
📌 UITableView 구성요소





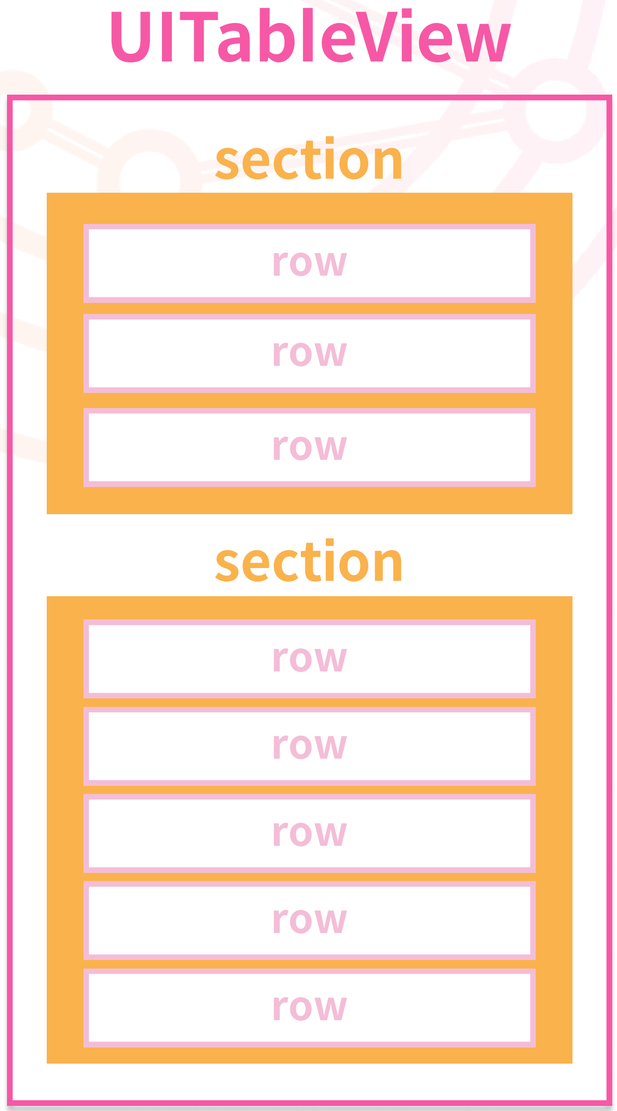

```
하나의 테이블 뷰 {
  여러 개의 테이블 뷰 섹션 {
    여러 개의 테이블 뷰 셀(row){
      테이블 뷰 셀마다 하나 씩의 content view{
        컨텐츠
      }
    }
  }
}
```

📌 UITableView 구성요소



하나의 테이블 뷰 {
여러 개의 테이블 뷰 섹션 {
여러 개의 테이블 뷰 셀(row){
테이블 뷰 셀마다 하나 씩의 content view{
컨텐츠
}
}
}
}
}

📌 UITableView 구성요소



하나의 테이블 뷰 {

여러 개의 테이블 뷰 섹션 {

여러 개의 테이블 뷰 셀(row){

테이블 뷰 셀마다 하나 씩의 content view{

컨텐츠

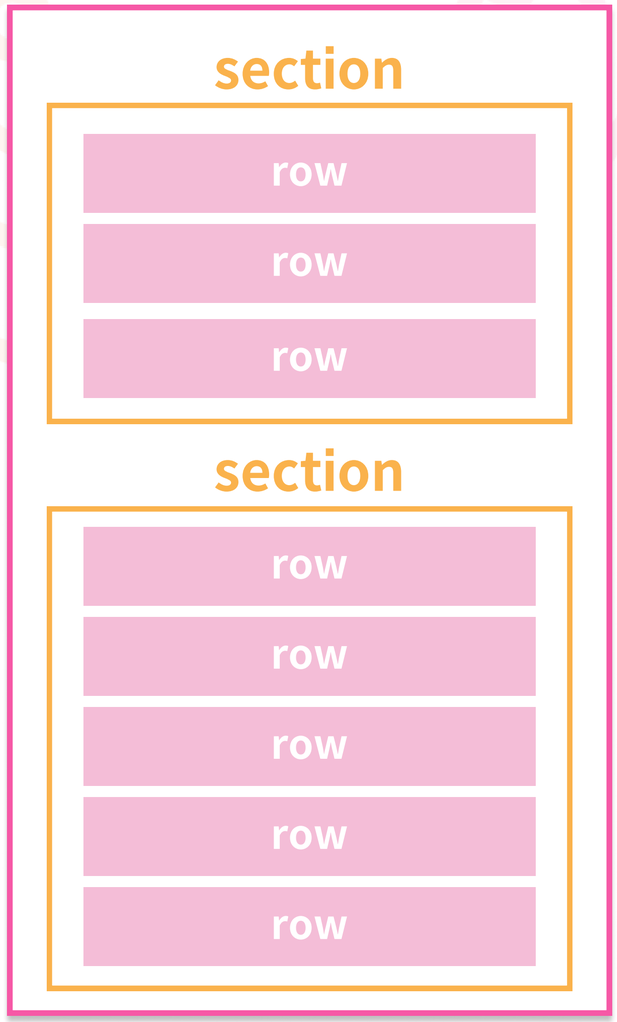
}

}

}

}

📌 UITableView 구성요소



하나의 테이블 뷰 {

여러 개의 테이블 뷰 섹션 {

여러 개의 테이블 뷰 셀(row){

테이블 뷰 셀마다 하나 씩의 content view{

컨텐츠

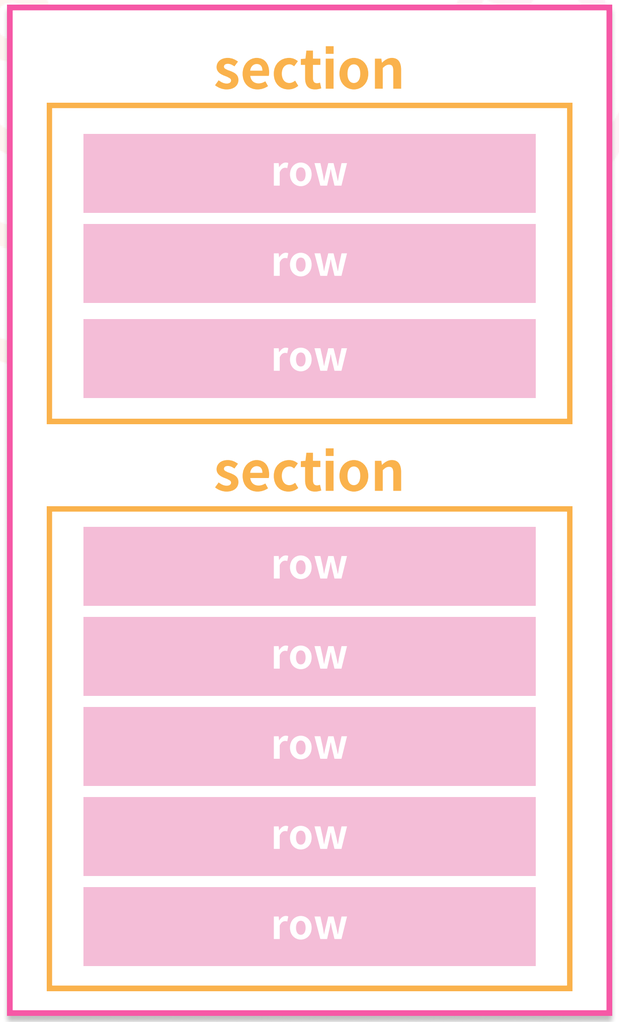
}

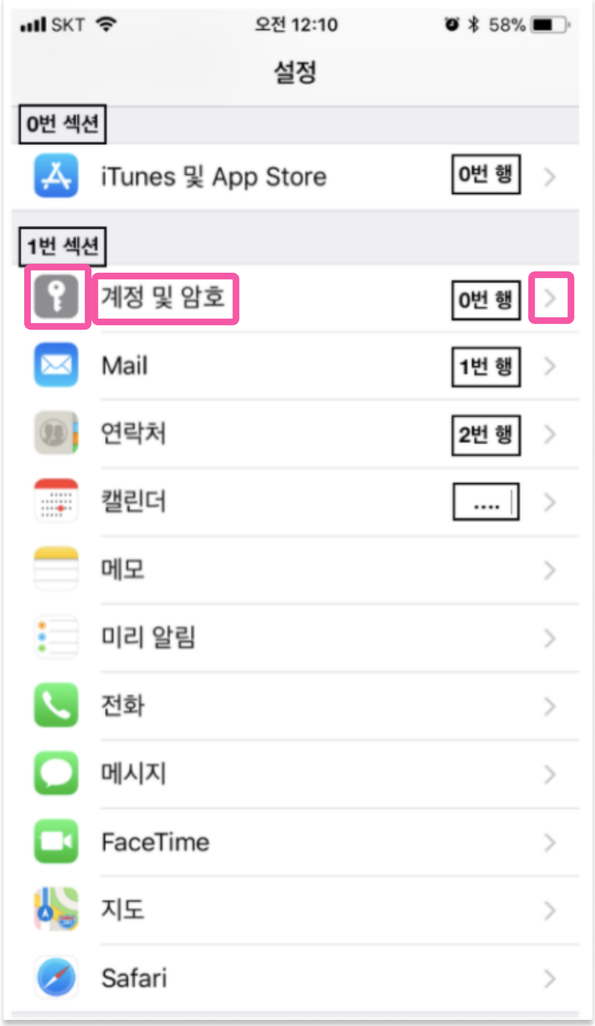
}

}

}

📌 UITableView 구성요소





하나의 테이블 뷰 {

여러 개의 테이블 뷰 섹션 {

여러 개의 테이블 뷰 셀(row){

테이블 뷰 셀마다 하나 씩의 content view{

컨텐츠

}

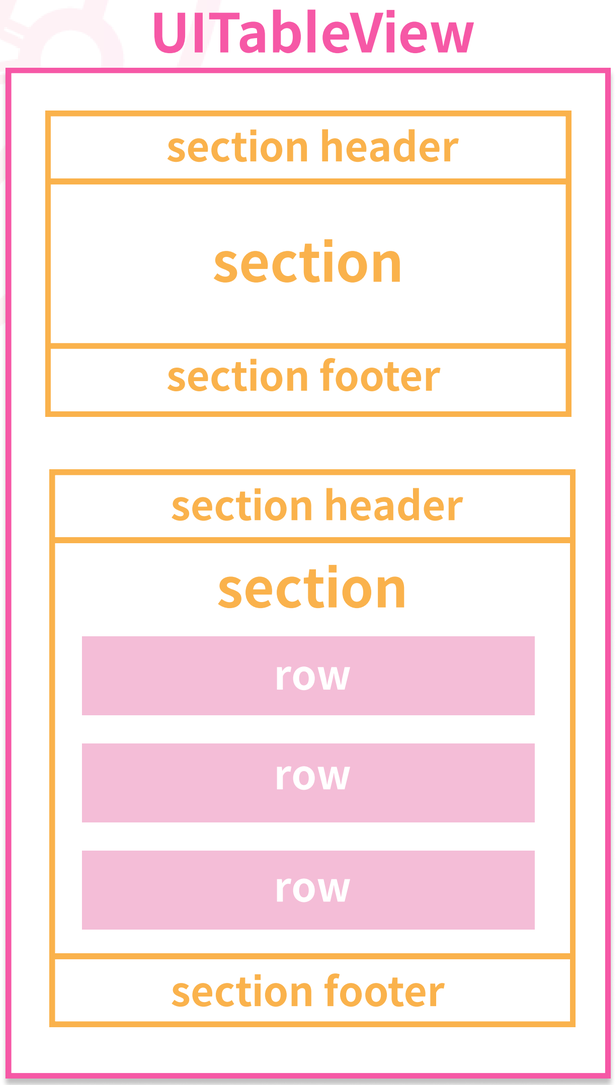
}

}

}

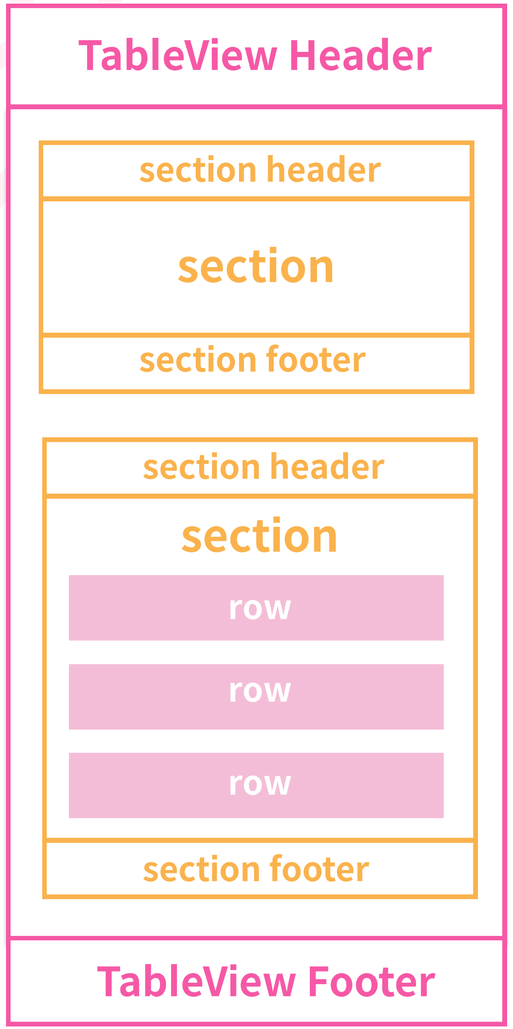
📌 UITableView 구성요소 - section

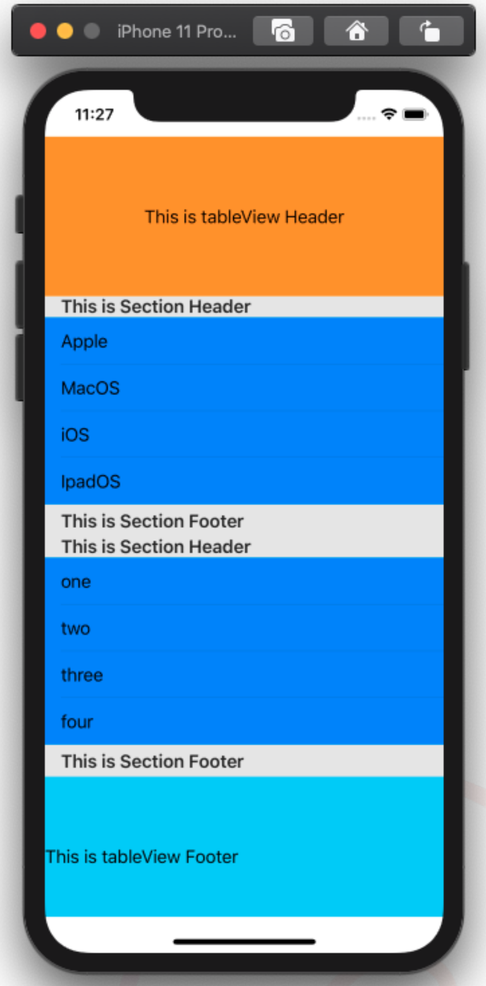
- 각각의 section에는 header와 footer가 붙을 수 있습니다.



📌 UITableView 구성요소

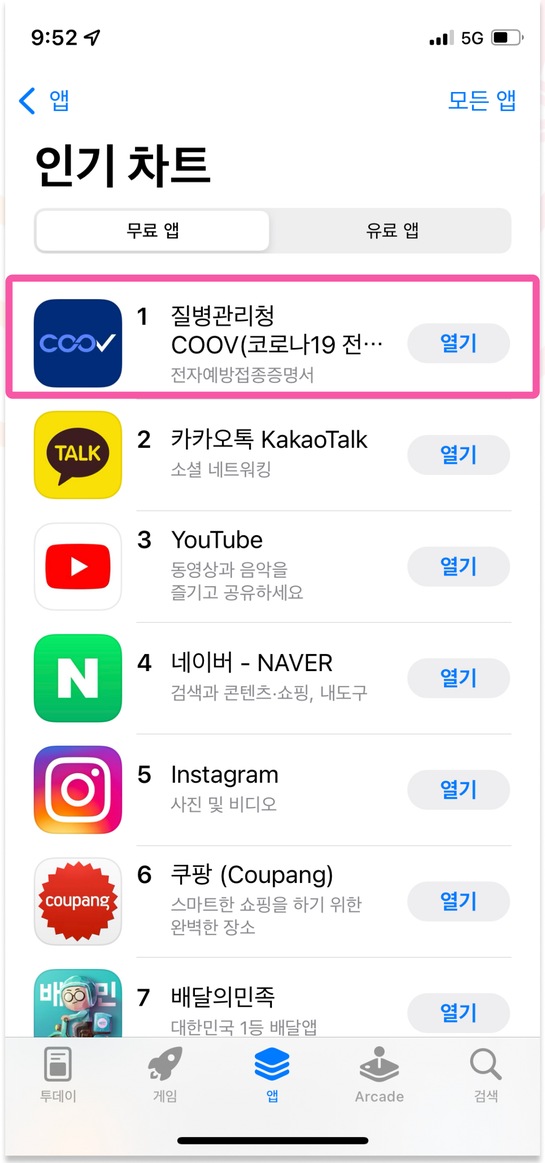
- TableView에도 Header와 Footer가 붙을 수 있으며, 스크롤 영역에 포함됩니다.

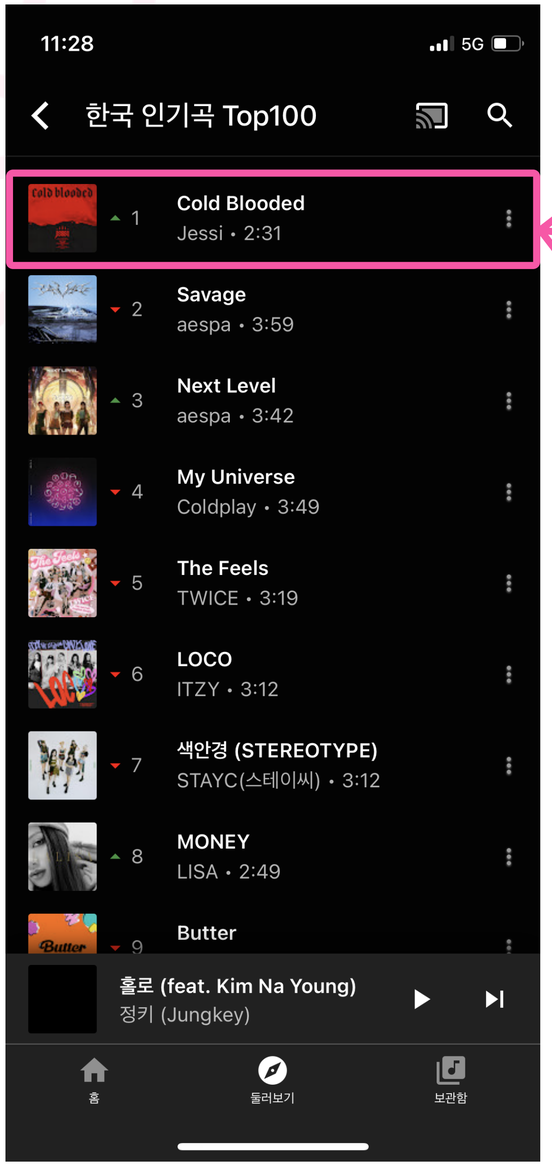


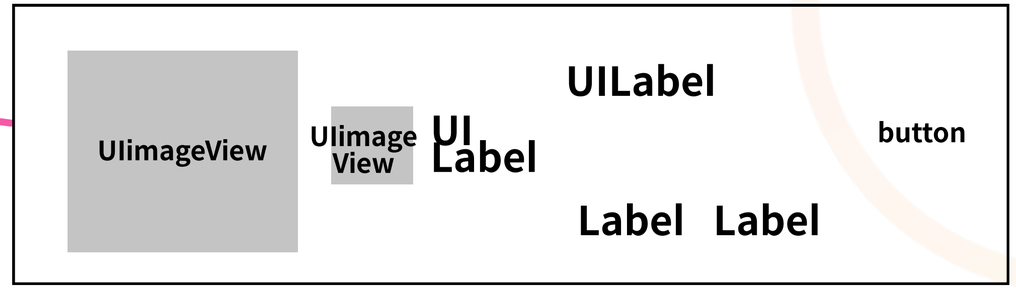


(출처 : https://duwjdtn11.tistory.com/560)

UITableView







데이터만 바뀌고 동일한 layout인게 보이시나요?

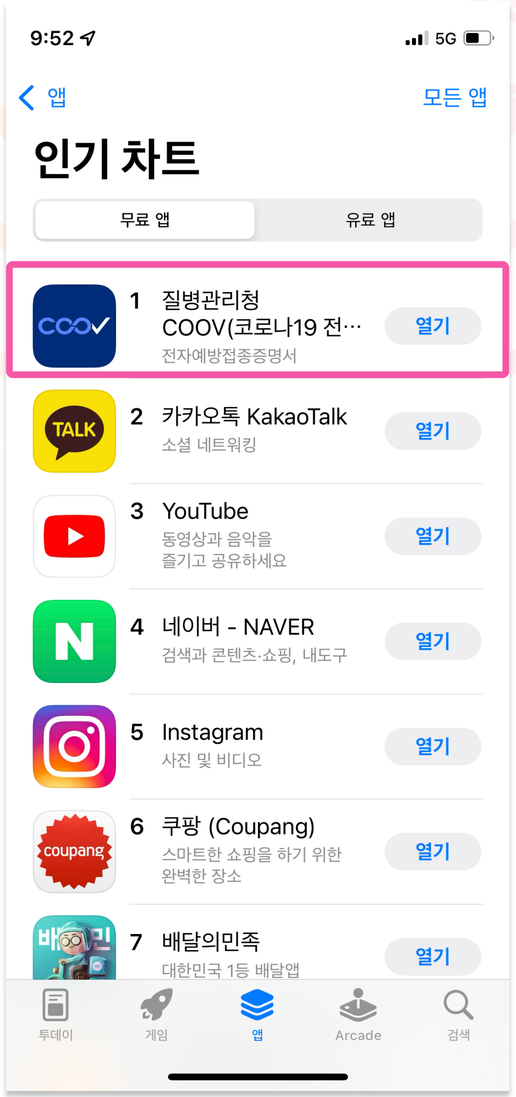
동일한 layout으로 반복되는 뷰인데,

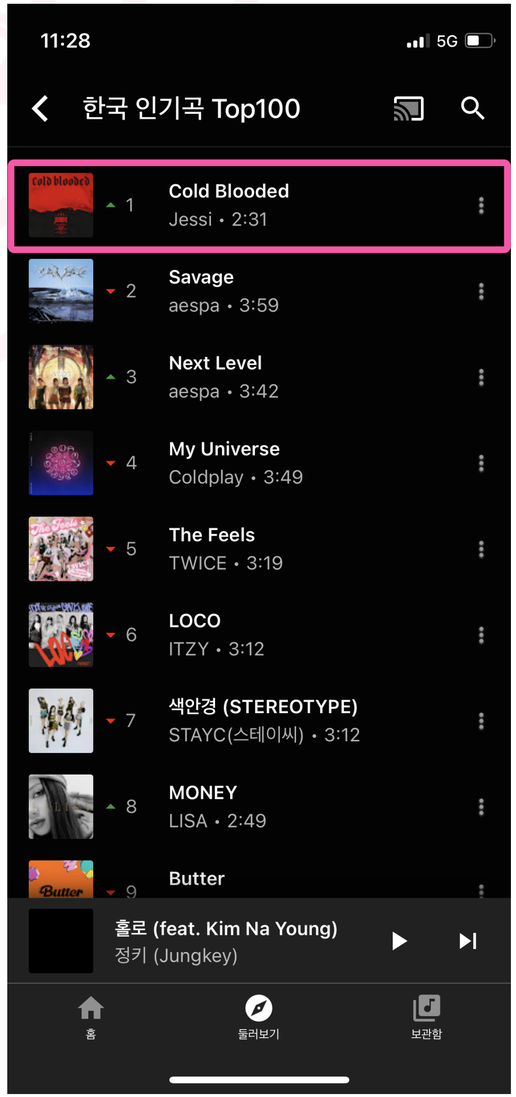
이걸 하나하나 다 짜려니...아찔 (이마짚)

이럴때 더욱 유용한 UI가 바로

tableview와 tableview cell 입니다!

UITableView

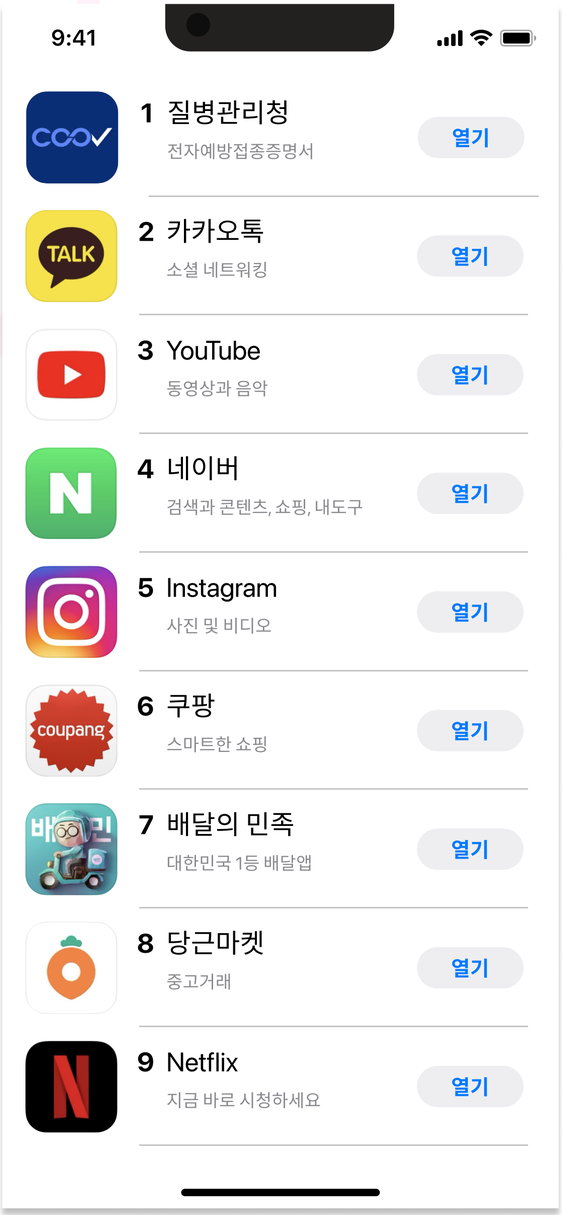




📌 UITableViewCell

- 테이블 행의 내용을 관리하기 위해 사용되는 클래스
- 테이블 관련 동작을 지원하기 위해 몇 가지 특정 사용자 지정을 제공합니다.

UITableView

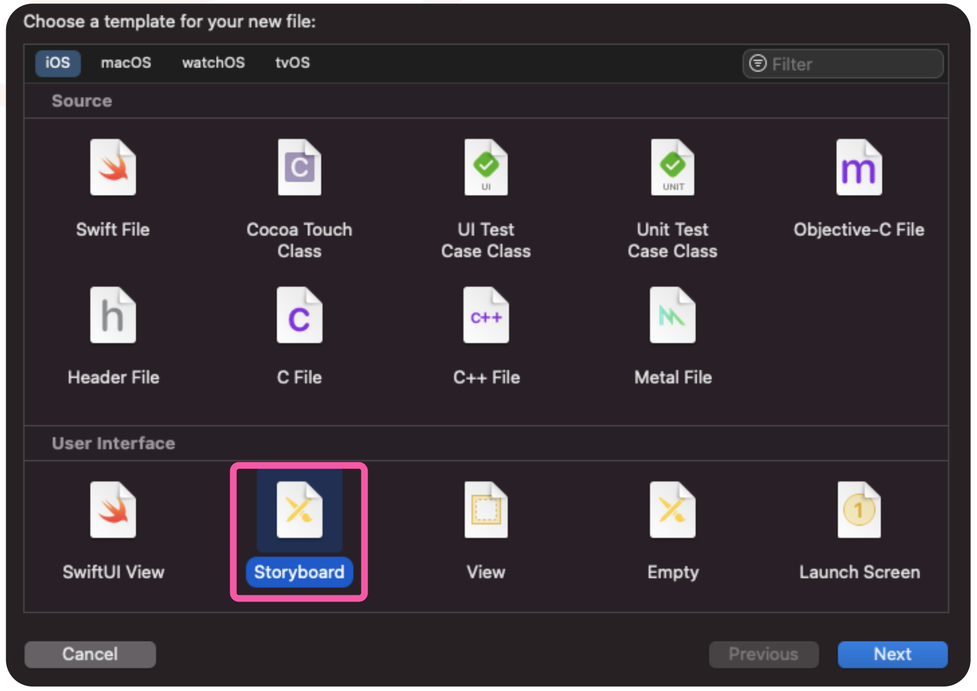


실습해봅시다!

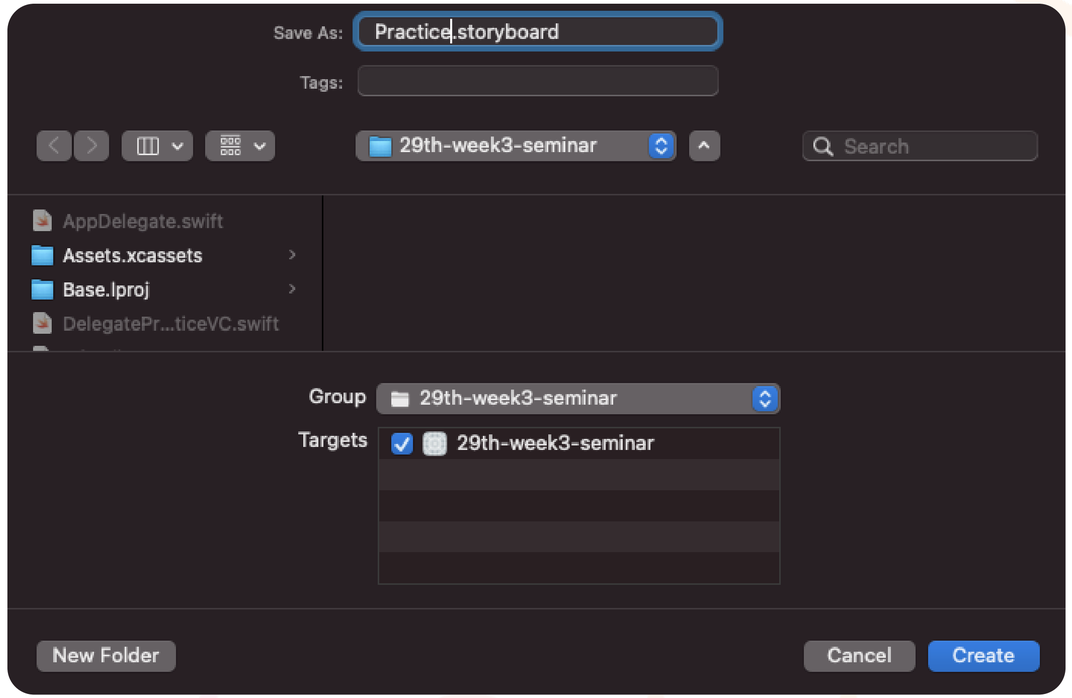
UITableView

먼저, view controller들을 분리하기 위해
새로운 스토리보드를 추가해주세요!

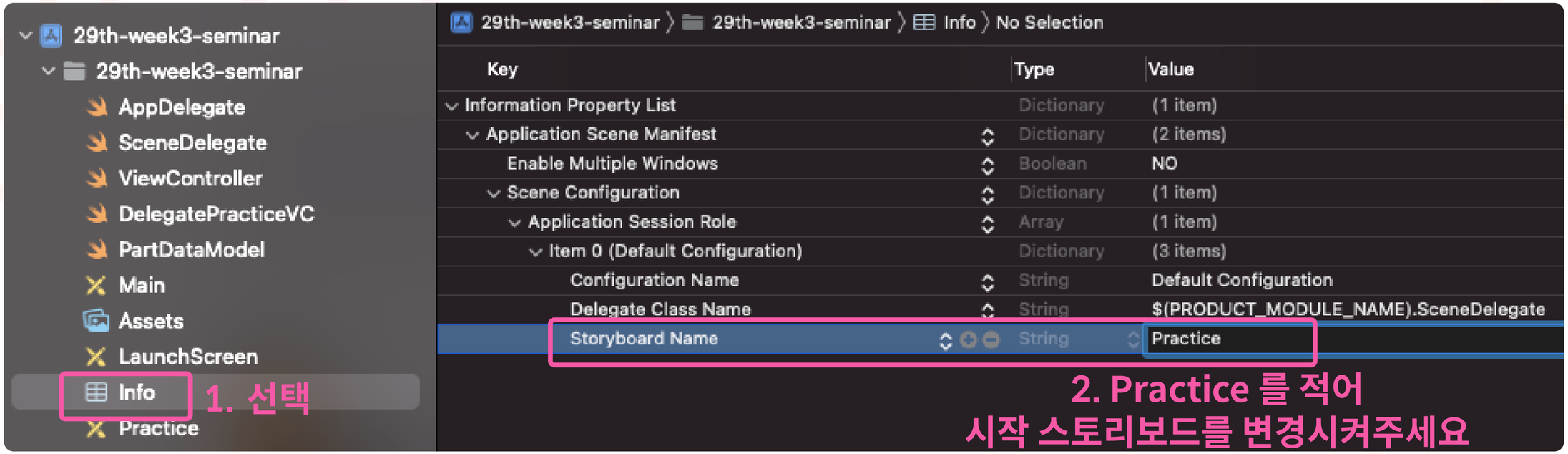
1. 파일추가(cmd+n)를 눌러 storyboard 파일을 선택한 후 Next



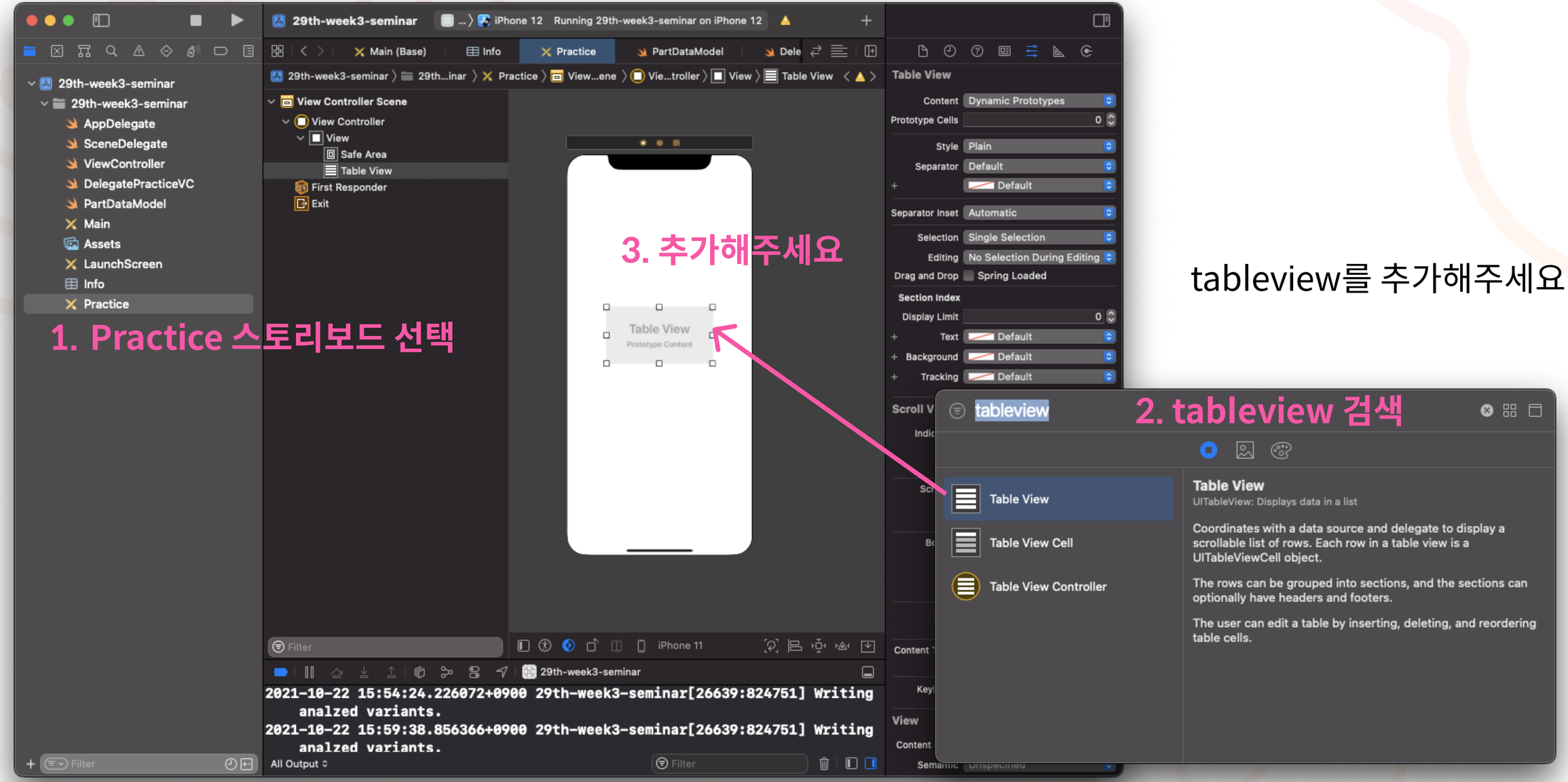
2. 파일명 “Practice.storyboard” Create



시작 스토리보드를 변경해주세요!

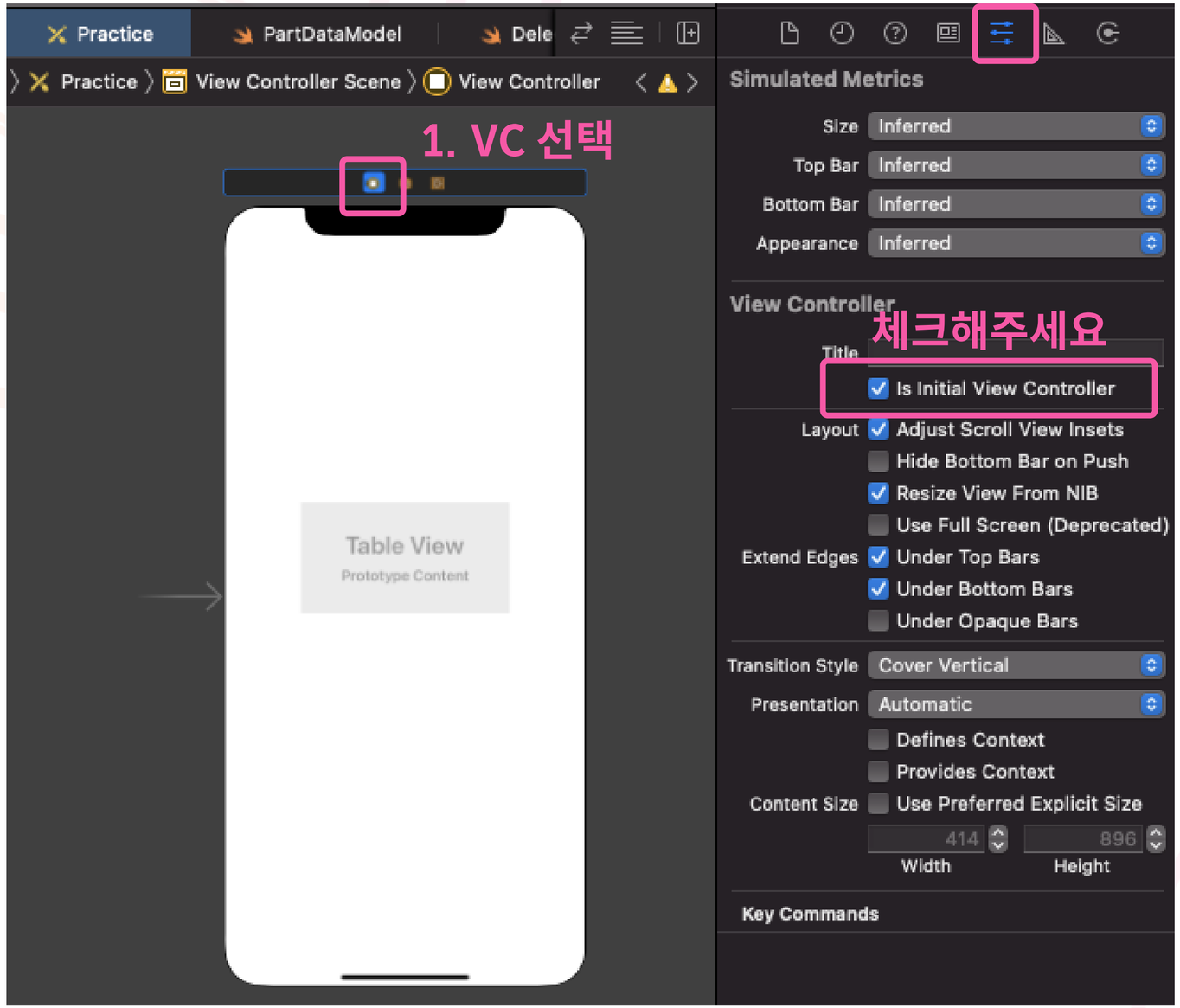


UITableView



tableview를 추가해주세요!

UITableView

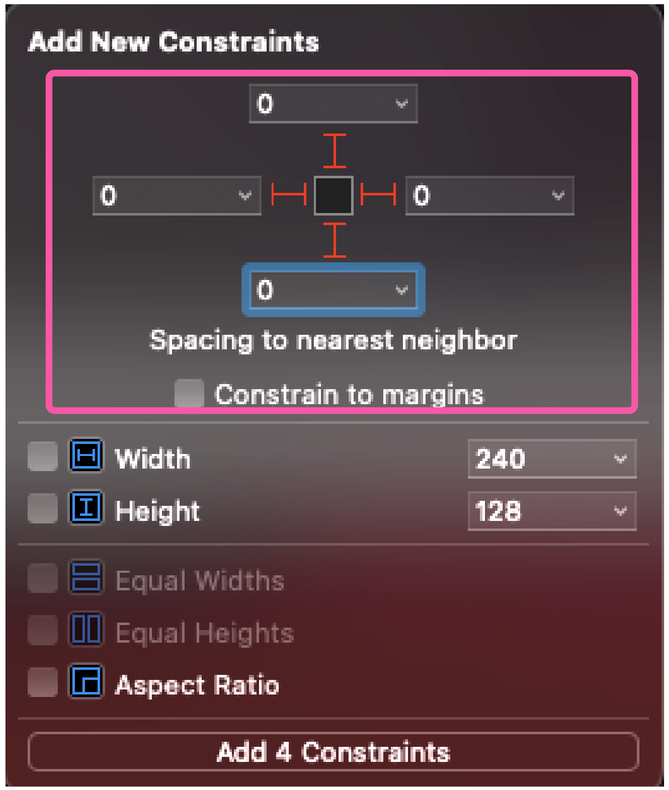


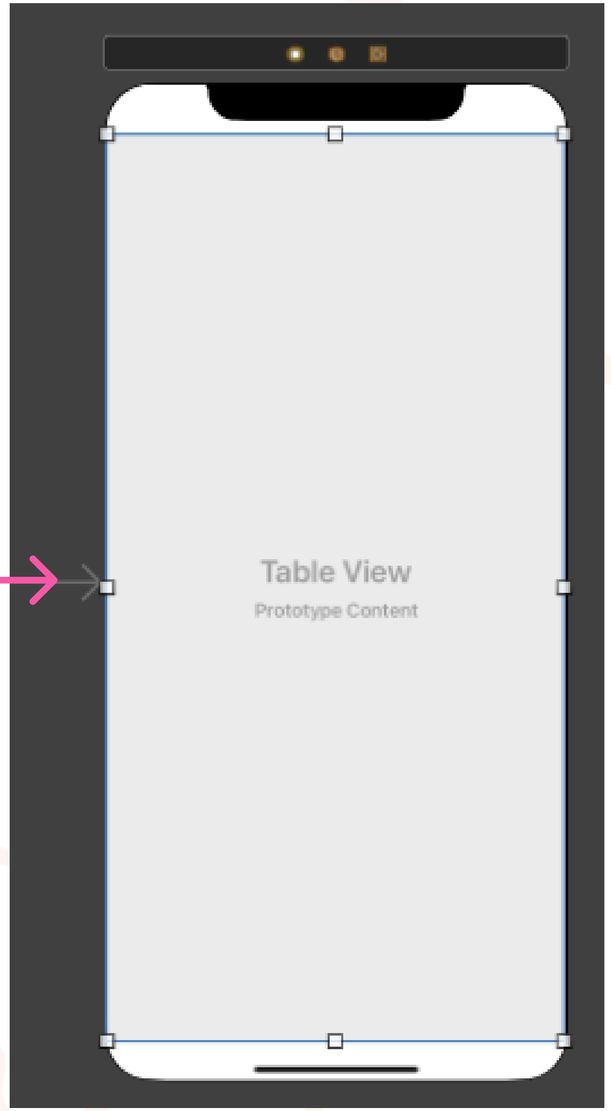
잊지말고 스토리보드의
시작 view controller로 지정해주세요:)

UITableView

1. tableview 선택

2. top, leading, trailing, bottom
모두 0으로 잡은 뒤
constraints를 추가해주세요

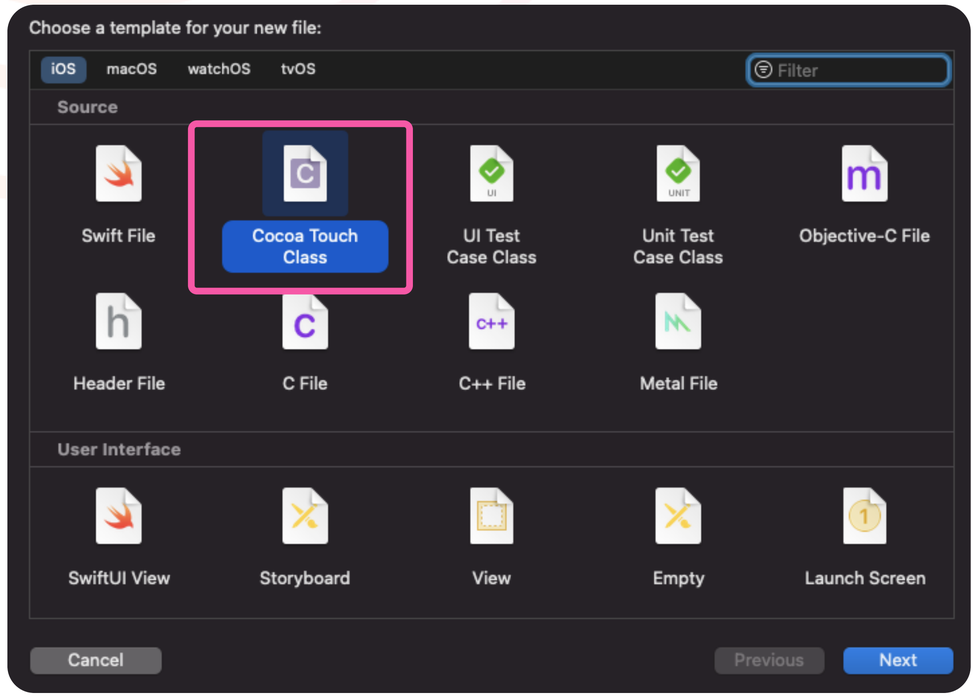




UITableView

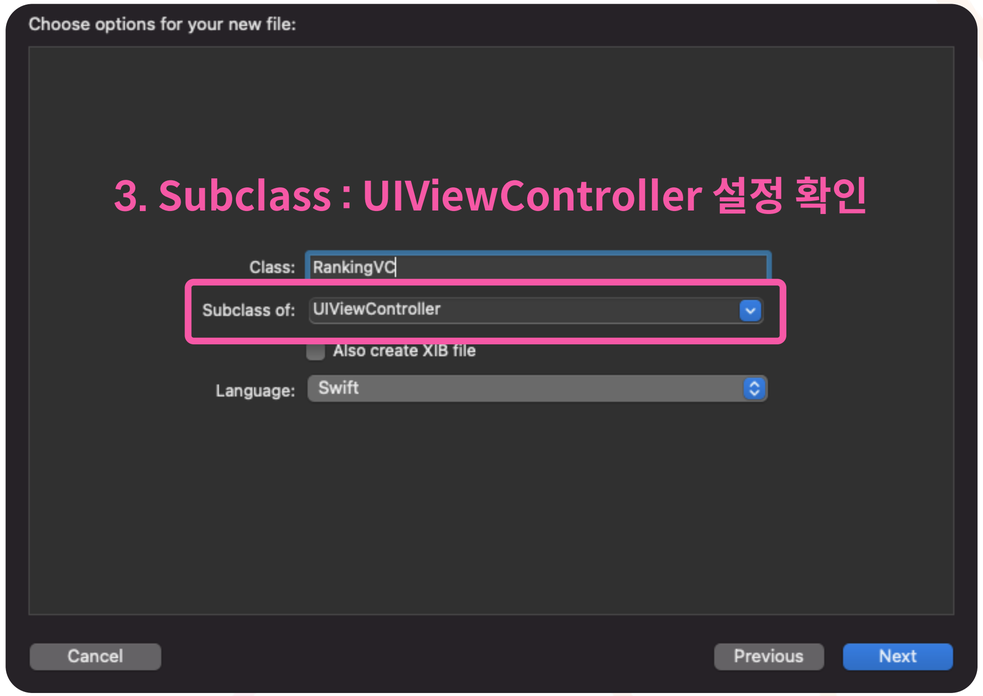
tableview 를 추가한 view controller랑 연결할 파일을 추가합시다!

1. Cocoa Touch Class 선택 후 Next

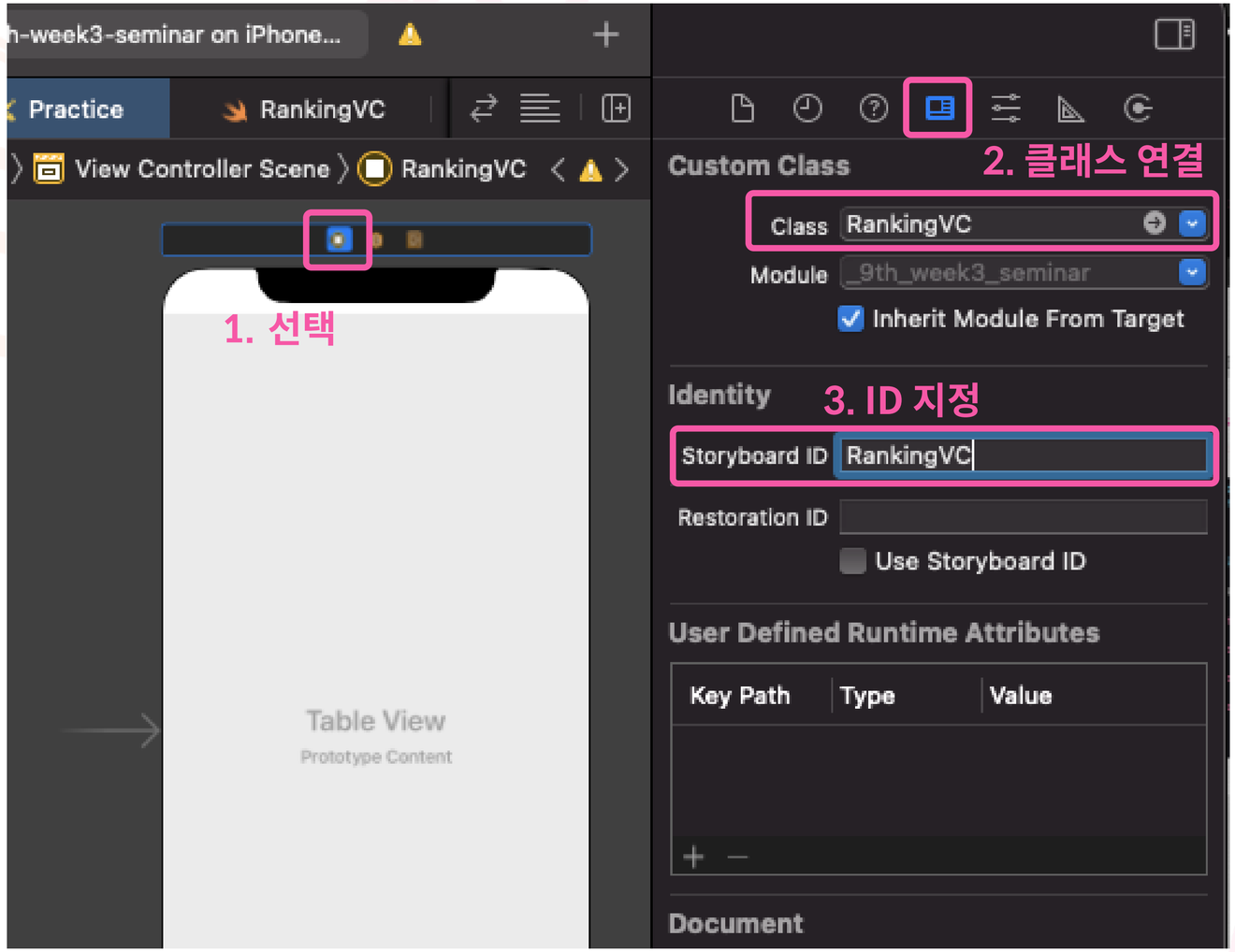


2. “RankingVC”로 클래스 이름 설정

(VC : ViewController 약어)



UITableView

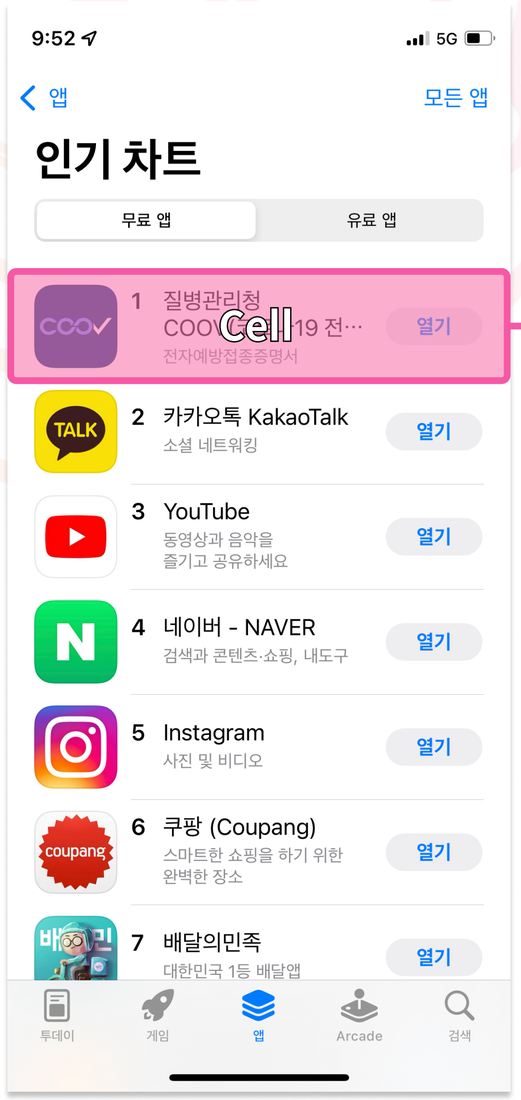


tableview 를 추가한 view controller랑
방금 만든 클래스를 연결합니다.

UITableView

tableview outlet 까지 연결해주세요!

UITableView

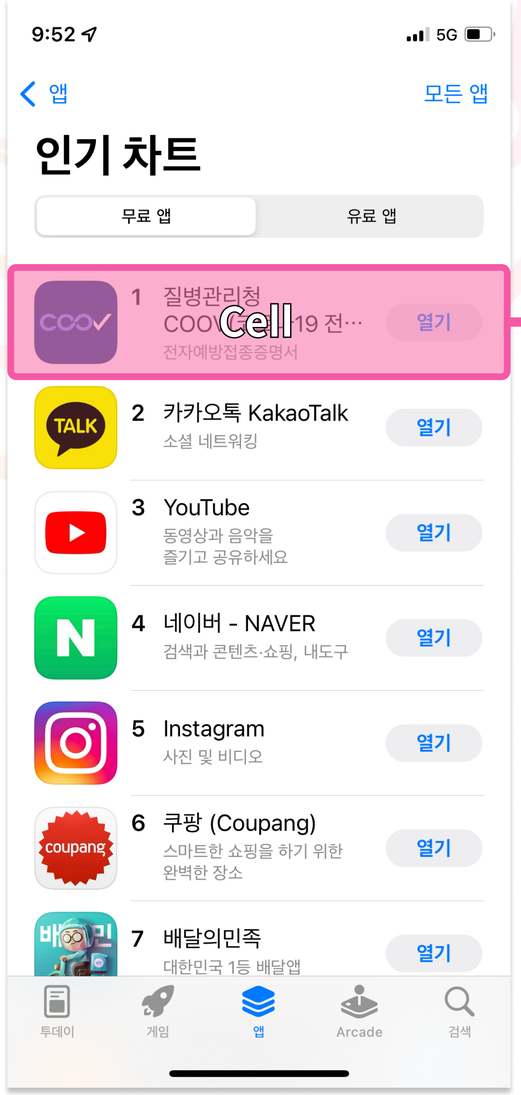


저희는 이제 왼쪽과 같은 cell을 구현해야합니다!

tableview cell을 구현하는 방법은 크게 3가지가 있습니다

1. 스토리보드 내에서 TableviewCell 활용해서 구성 (Prototype Cell)
2. Xib 파일로 따로 분리해서 Cell UI 구성
3. Code base로 Cell 구성

UITableView



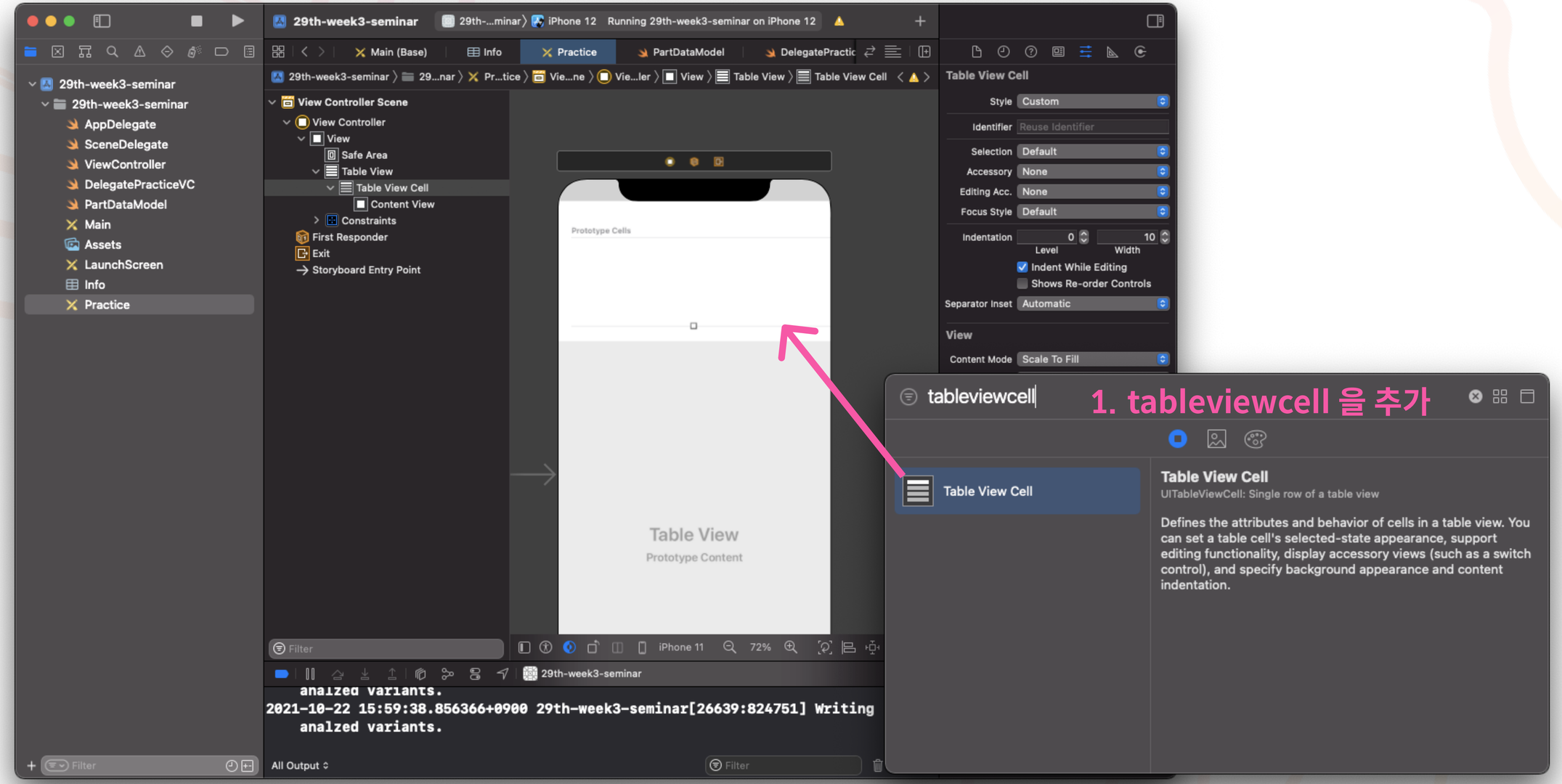
저희는 이제 왼쪽과 같은 cell을 구현해야합니다!

tableview cell을 구현하는 방법은 크게 3가지가 있습니다

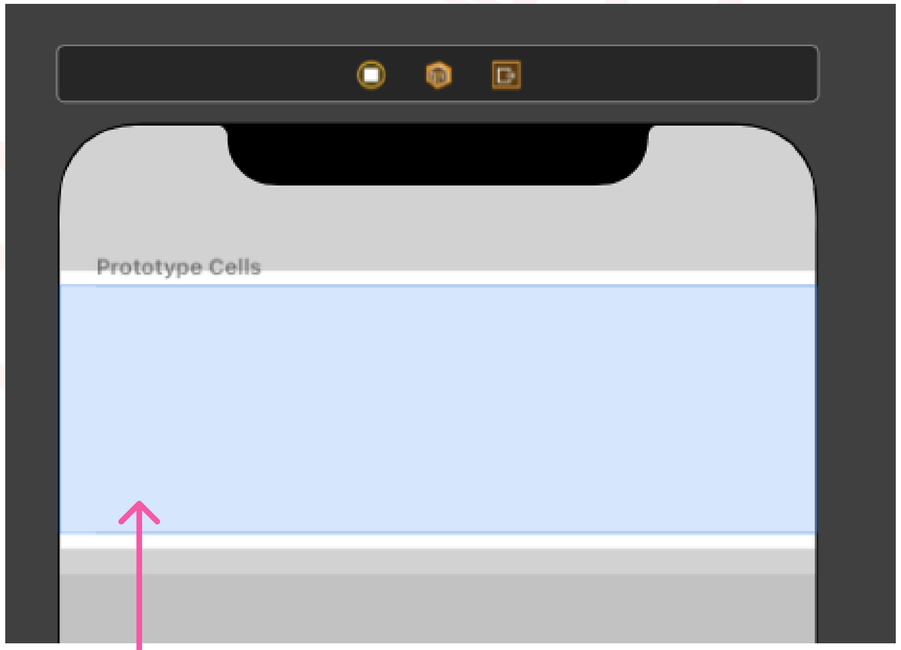
1. 스토리보드 내에서 TableviewCell 활용해서 구성 (Prototype Cell)
2. Xib 파일로 따로 분리해서 Cell UI 구성
3. Code base로 Cell 구성

협업에서 많이 사용하는 방법으로
앞으로 다가올 디자인합동 세미나를 대비하여
이 방법으로 세미나 실습을 진행하겠습니다!

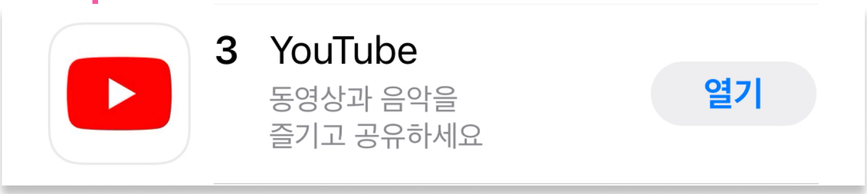
📌 PrototypeCell 간단 사용보기



UITableView



이곳에 cell ui를 구성하면됨



📌 PrototypeCell 간단 사용보기

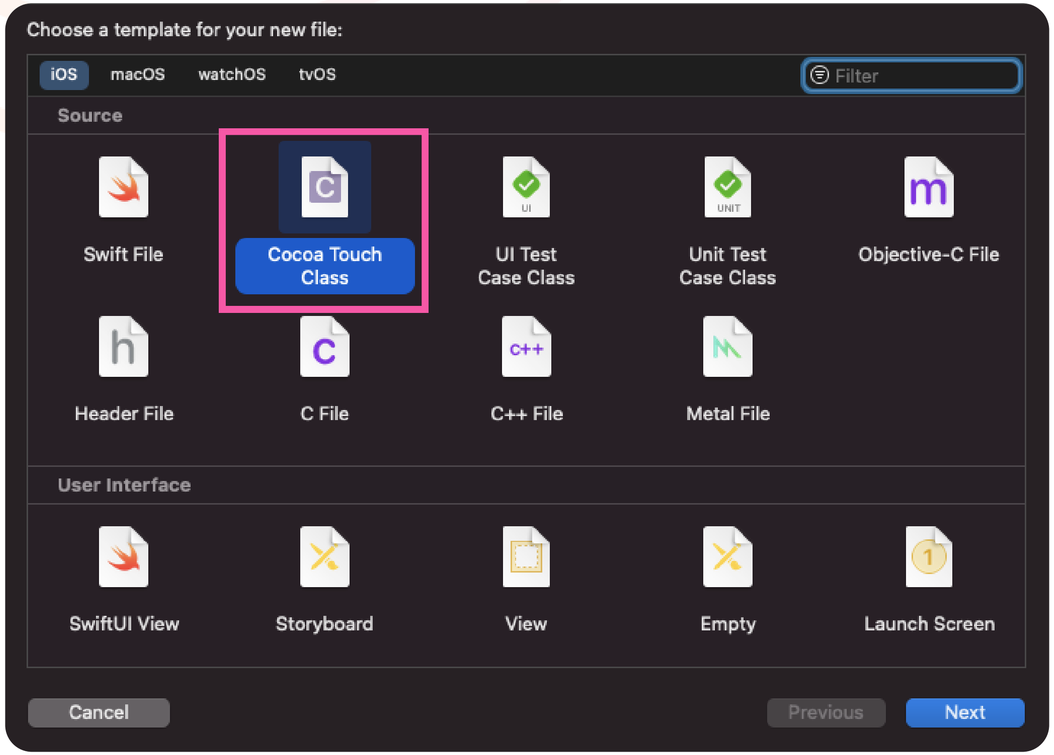
- 테이블 뷰의 셀을 원하는 대로 쉽게 디자인 할 수 있도록 해주는 객체
- 버튼이나 텍스트 레이블과 달리 직접 데이터를 표시하거나 공간을 차지하지 않습니다.
- 그저 테이블 뷰가 화면에 표현될 때

  셀의 구성을 미리 보여주기 위한 가상의 틀에 불과합니다.

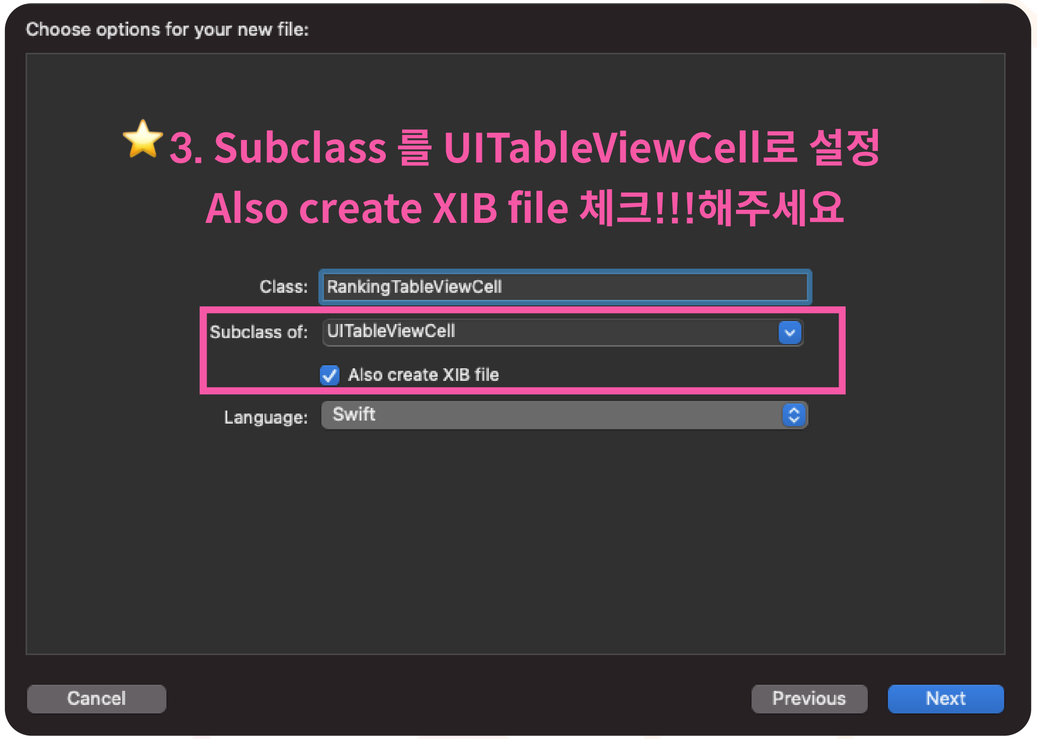
UITableView

xib를 활용하여 tableview cell을 만들어봅시다!

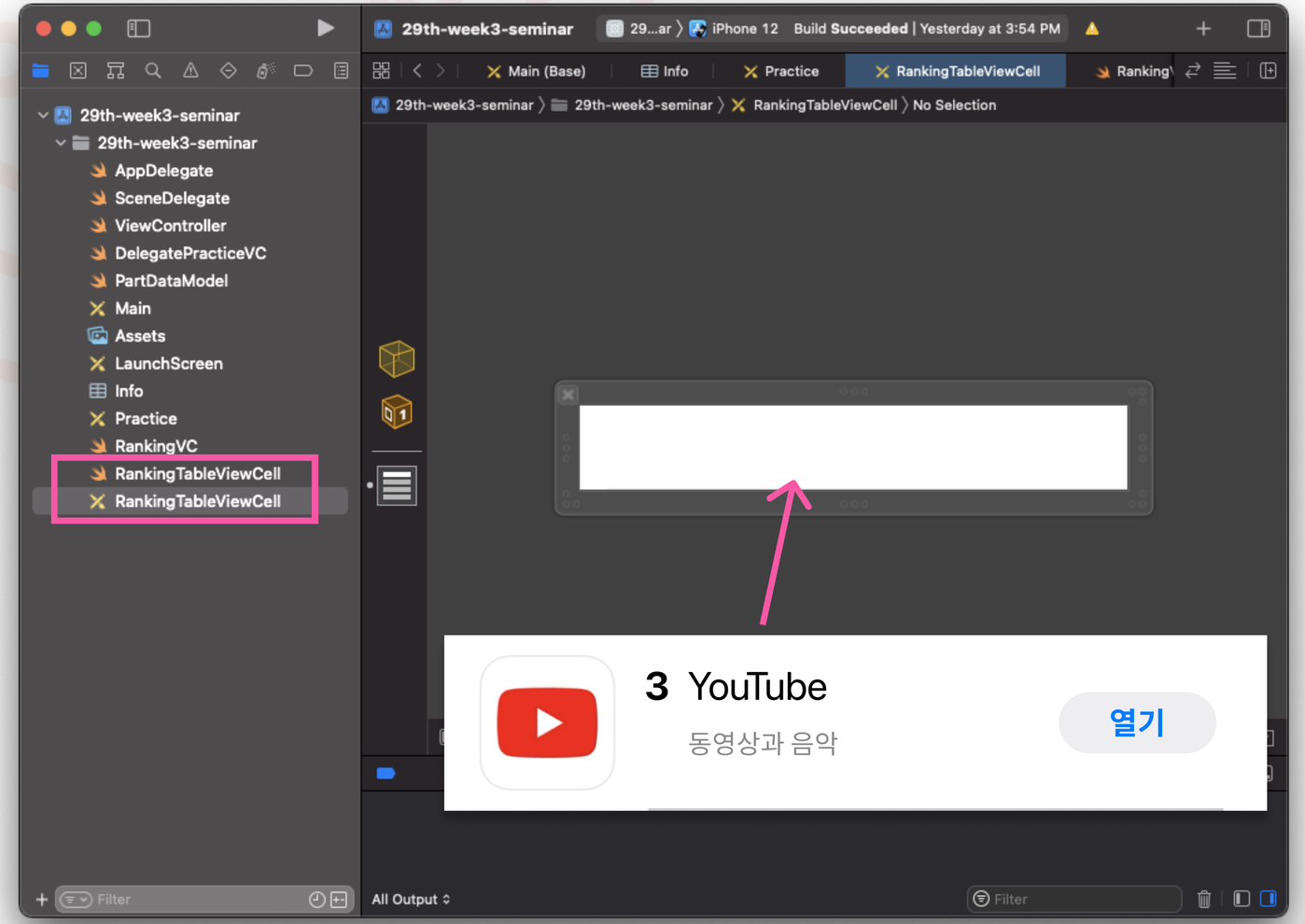
1. 파일추가(cmd+n)를 눌러
storyboard 파일을 선택한 후 Next



2. 파일명 “RankingTableViewCell” Create



UITableView



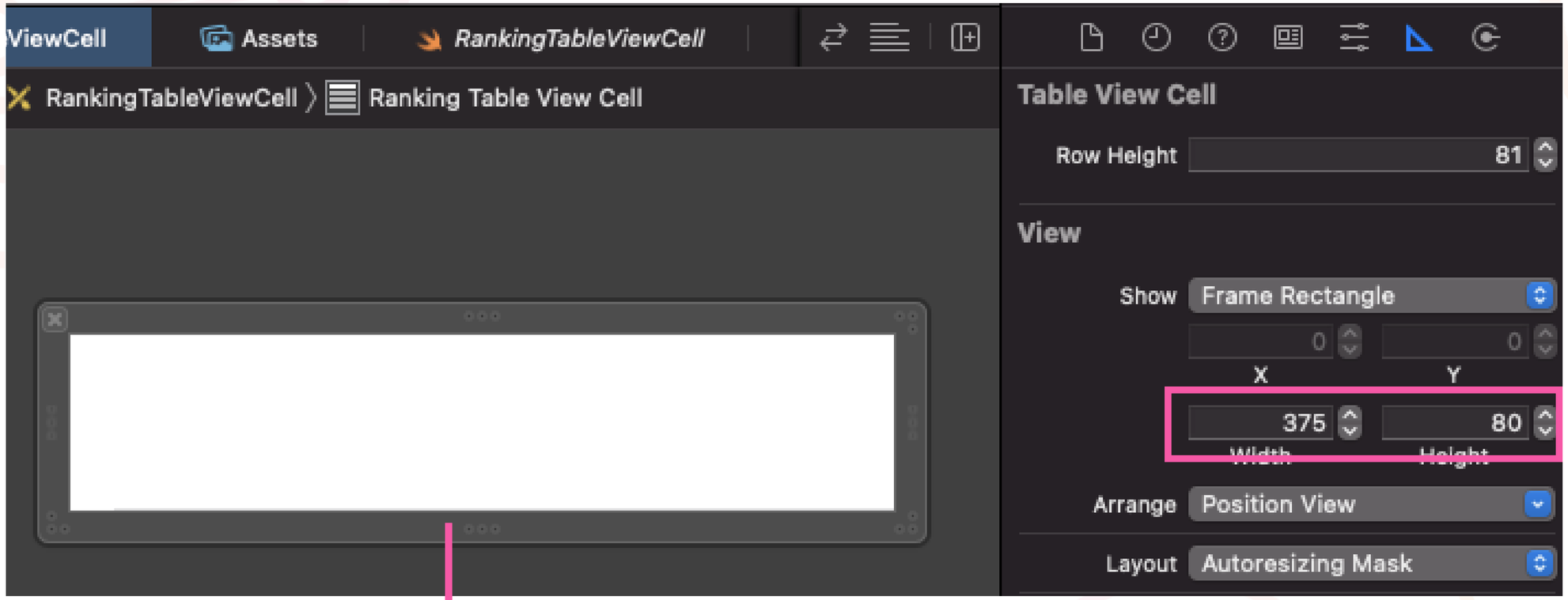
처음보는 파일이 열렸을 거에요!

이곳은 스토리보드와 동일하게
코드가 아닌 저희가 지금까지 뷰를 구성해왔던
그래픽 사용자 인터페이스 방식으로
뷰를 구성할 수 있습니다!

저희는 이곳에
TableviewCell의 UI를 구성해보겠습니다!

UITableView

화면이 좀 작죠? 먼저 화면부터 키워볼까요?

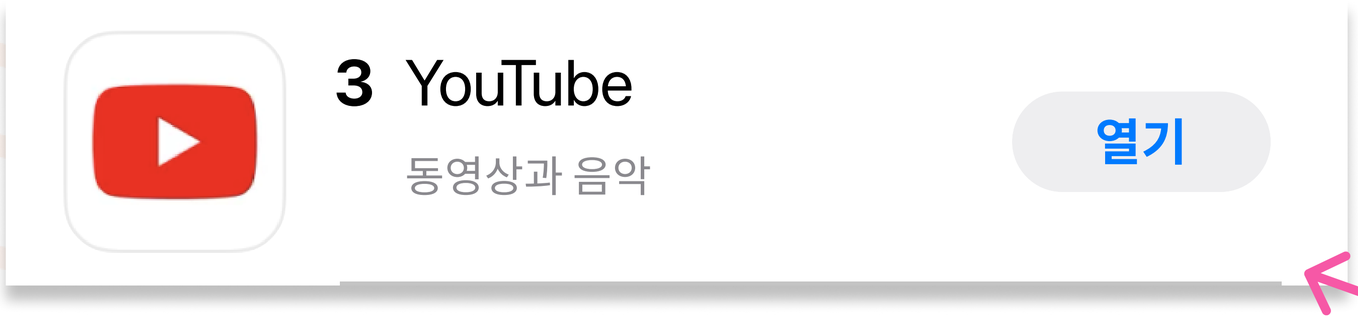


이 화면 크기는 개발할 때 보는 용도이지 사용
실제 cell 크기와 상관이 없습니다!

뷰의 크기 늘리듯이 늘려도 좋고

직접 가로,세로 크기를 지정해도 됩니다!
현재 디자인된 cell 크기가
375 * 80 에 맞춰 크기를 잡아봅시다!

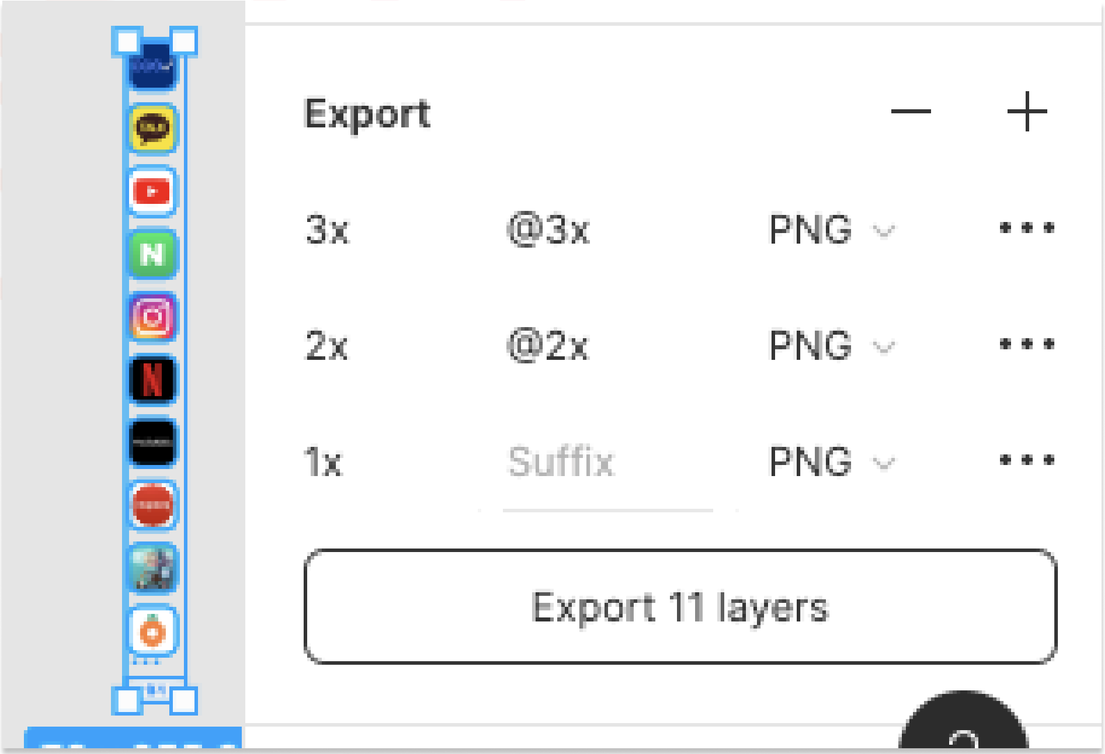
UITableView



구현할 cell에 들어갈
앱 로고가 들어갈 image view,
순위를 나타낼 Label,
앱 이름을 나태낼 Label,
열기 버튼 button,

구분선처럼 쓰일
uiview 추가하여
background color
변경해주세요

auto layout 잡기 전, 이미지 먼저 추가할게요!

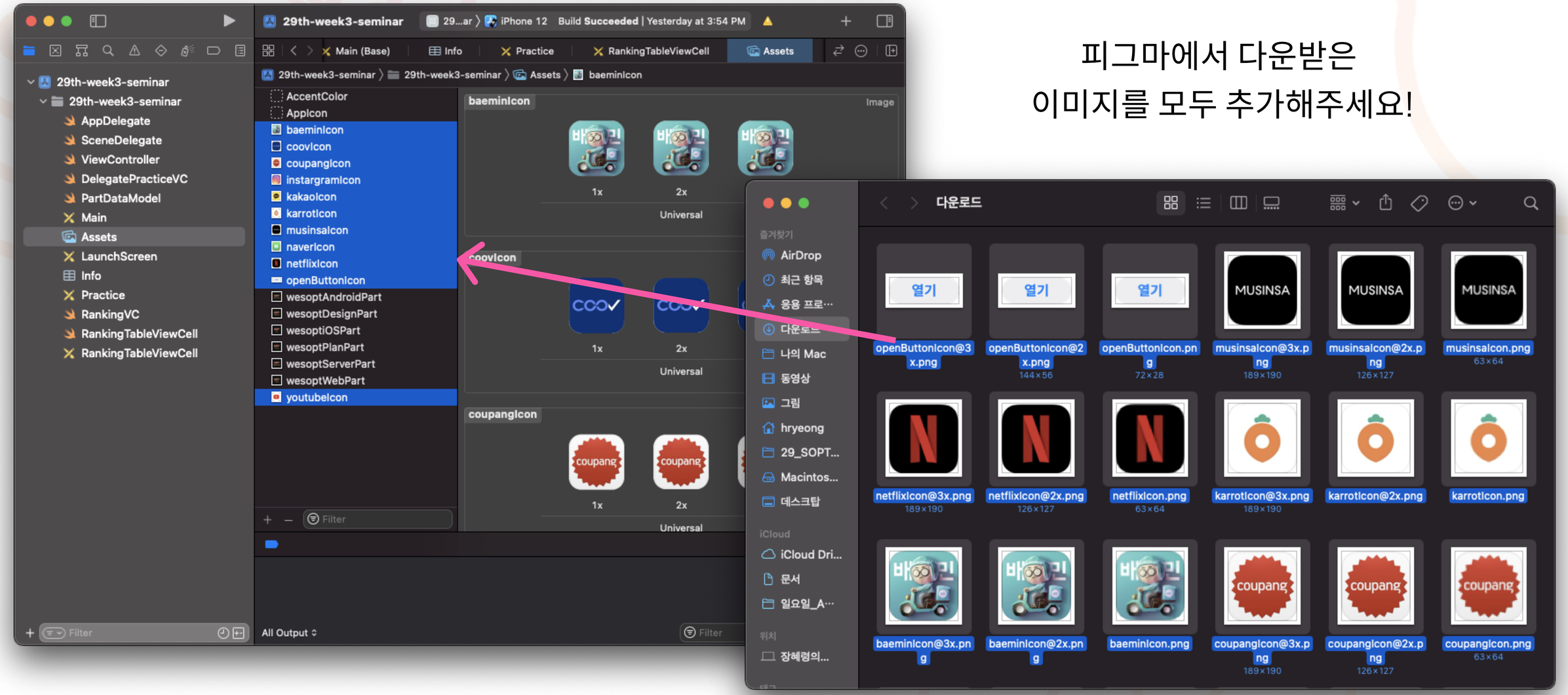


3차 세미나 실습에 넣어놓은
앱 아이콘과 열기버튼 총 11개의
이미지를 다운받아주세요!

UITableView

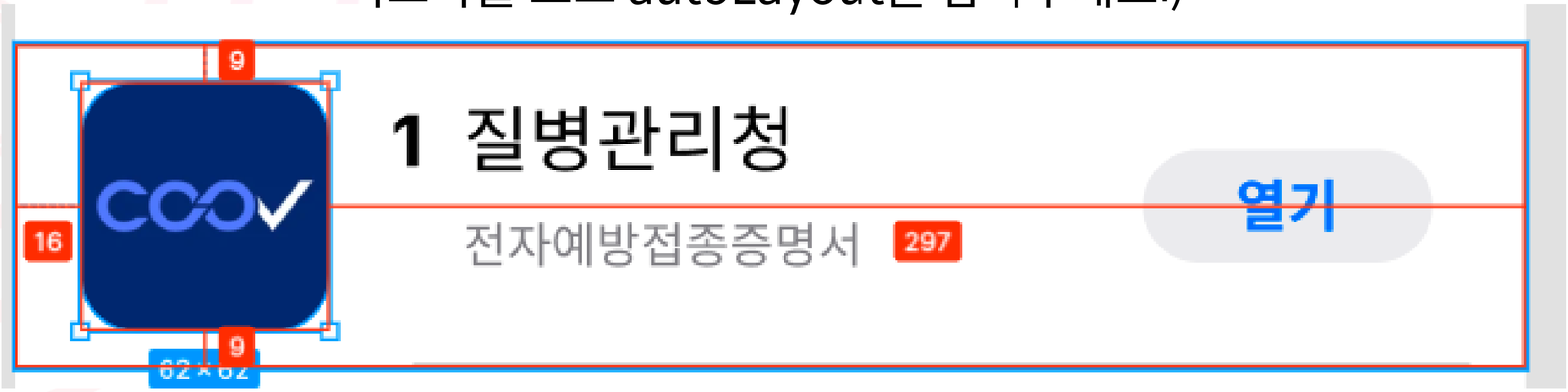
auto layout 잡기 전, 이미지 먼저 추가할게요!

피그마에서 다운받은
이미지를 모두 추가해주세요!

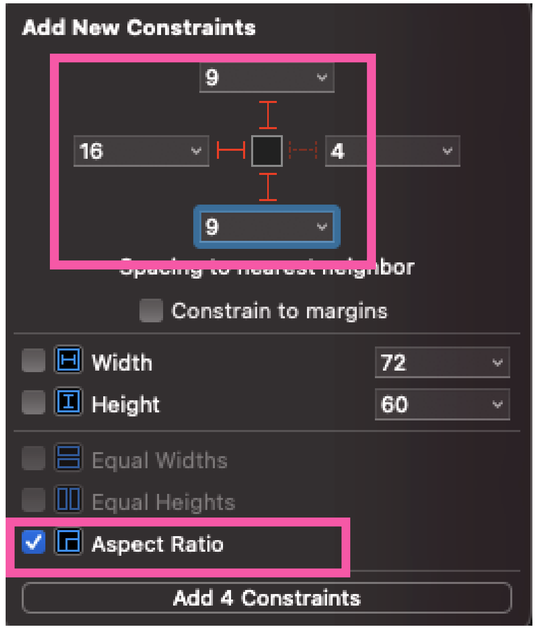


UITableView

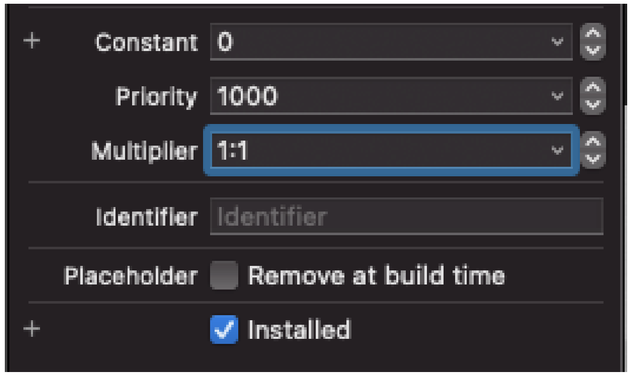
피그마를 보고 autoLayout을 잡아주세요:)



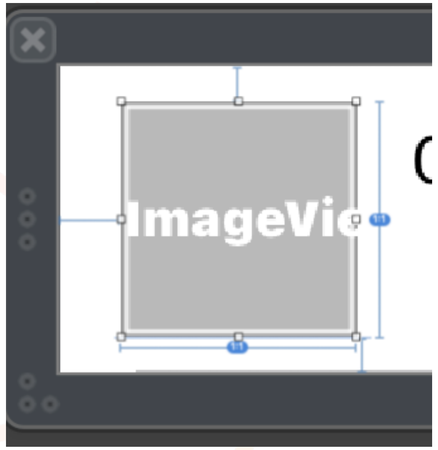
ImageView를 간격을 맞추면서 정사각형 크기로 설정하는 방법



constraint를 top, leading, bottom를 잡아주면
현재는 height을 계산할 수 있겠죠?
거기에 Aspect Ratio를 체크하고 비율을 1:1로 맞춰주면!



height값은 top, bottom에 따라 계산되고
witdh는 그 height에 1:1 비율로
계산되어 정사각형이 됩니다!



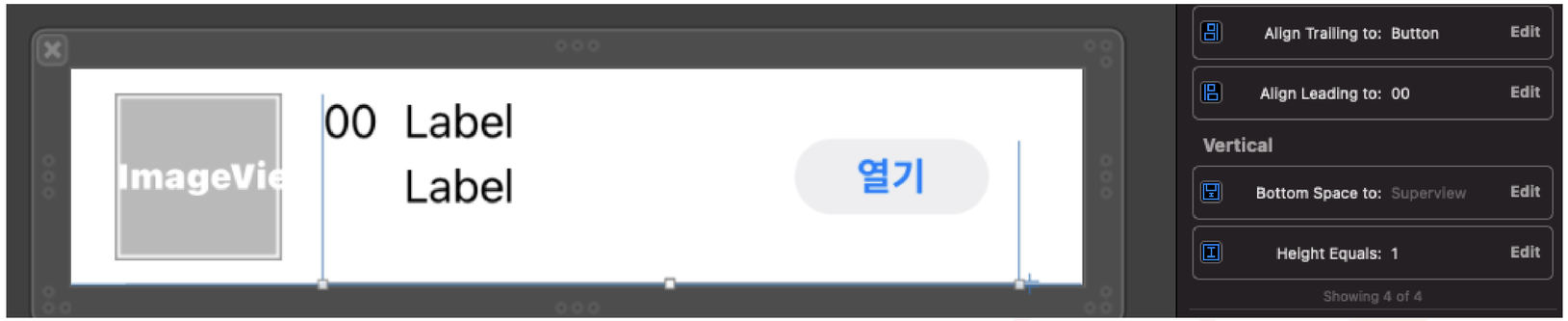
UITableView

피그마에서 바닥에 잇는 회색선의 layout이 잘 안보인다면
아래와 같이 잡아주세요!

순위 Text에 Leading

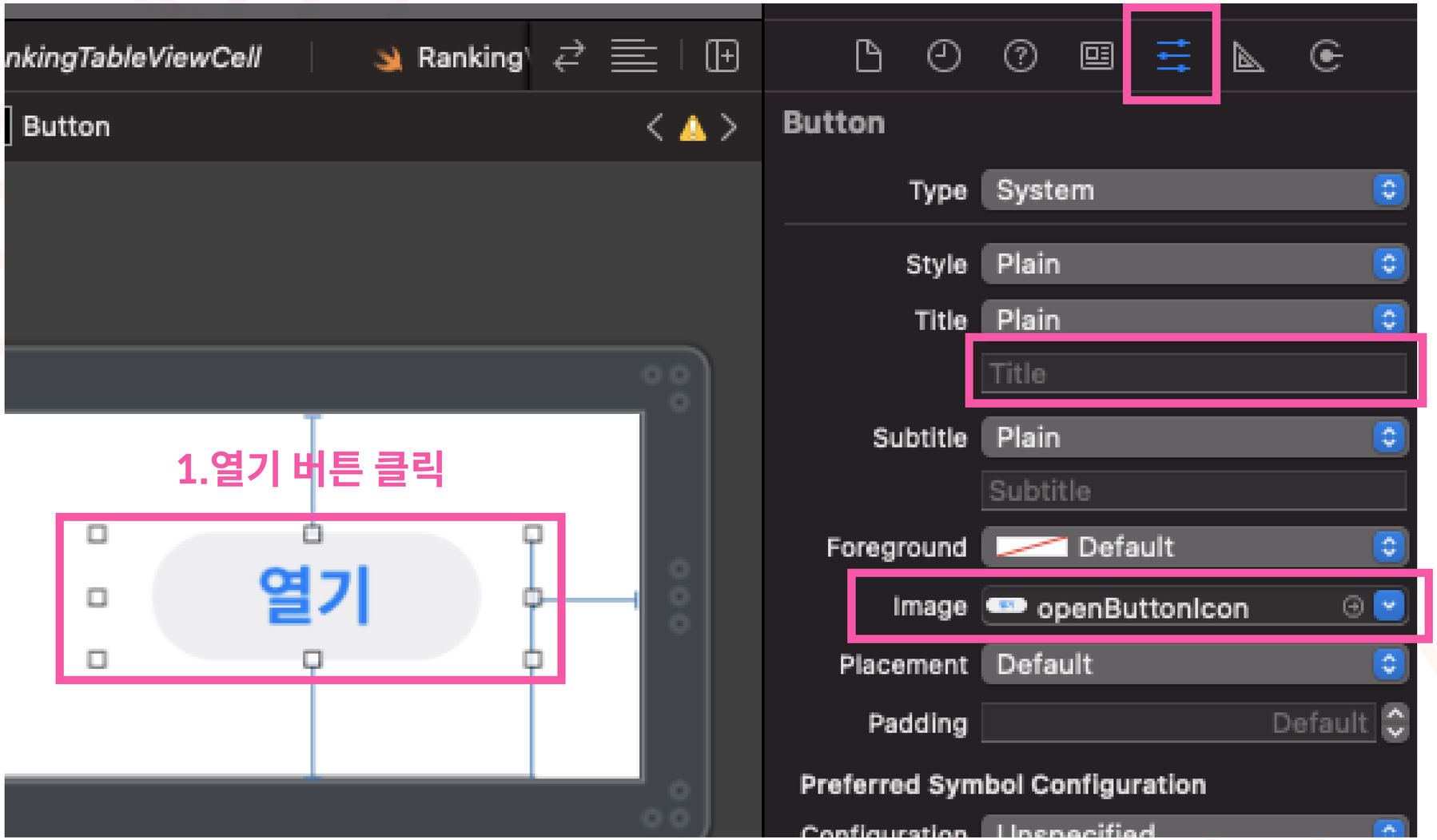
열기 버튼의 trailing

bottom은 0
height 1



UITableView

2. attribute inspactor 선택

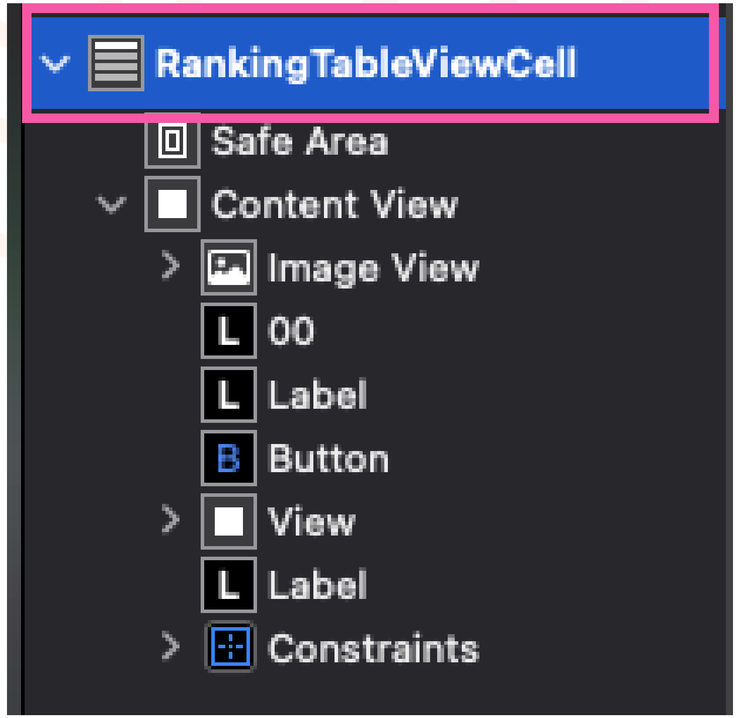


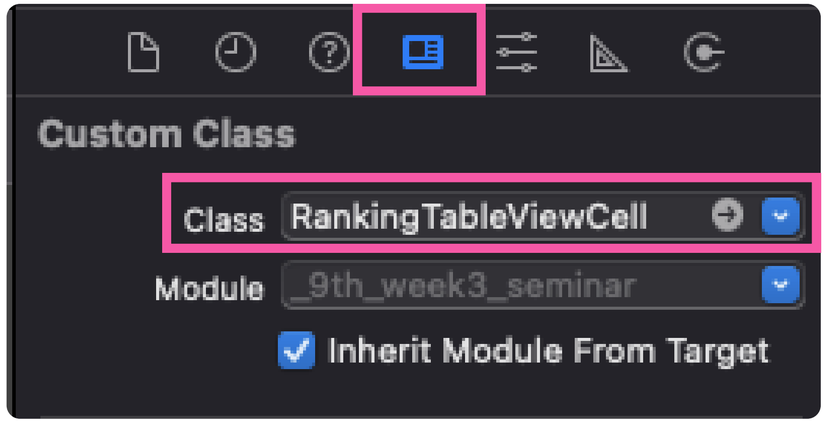
3. title 삭제

4. image를 눌러 확인 버튼 이미지 넣어주세요

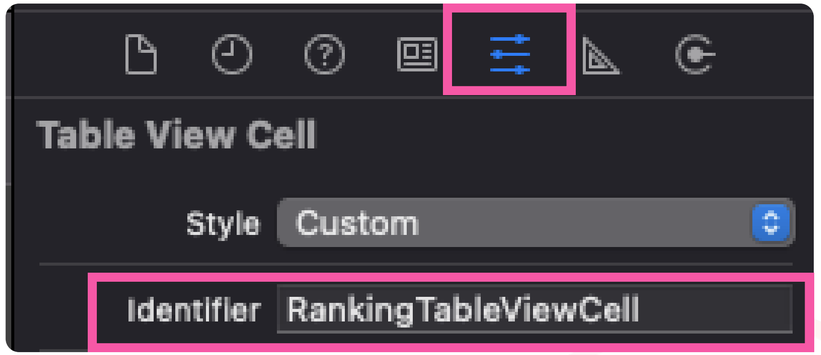
UITableView

1. 선택 후





2. Xib로 한 경우에는 알아서 연결이 되어있지만, 잘 되어있는지 확인



3. identifier를 class 이름과 동일한 이름으로 적어주세요!

모두 설정이 되었으면, control + option + command + enter를 눌러 어시스턴스를 열어주세요!

UITableView

```
10  class RankingTableViewCell: UITableViewCell {
11
    @IBOutlet weak var appIconImageView: UIImageView!
    @IBOutlet weak var rankLabel: UILabel!
    @IBOutlet weak var titleLabel: UILabel!
    @IBOutlet weak var subTitleLabel: UILabel!
    @IBOutlet weak var openButton: UIButton!
17
18      override func awakeFromNib() {
19          super.awakeFromNib()
20      }
21
22      override func setSelected(_ selected: Bool, animated: Bool) {
23          super.setSelected(selected, animated: animated)
24
25          // Configure the view for the selected state
26      }
27
28  }
```

outlet을 설정해주세요!

UITableView

```
class RankingTableViewCell: UITableViewCell {

    @IBOutlet weak var appIconImageView: UIImageView!
    @IBOutlet weak var rankLabel: UILabel!
    @IBOutlet weak var titleLabel: UILabel!
    @IBOutlet weak var subTitleLabel: UILabel!
    @IBOutlet weak var openButton: UIButton!

    override func awakeFromNib() {
        super.awakeFromNib()
    }

    override func setSelected(_ selected: Bool, animated: Bool) {
        super.setSelected(selected, animated: animated)

        // Configure the view for the selected state
    }

}
```

이 클래스 안에는 처음 보는 낯선 함수가 들어있습니다..!

📌 awakeFromNib()

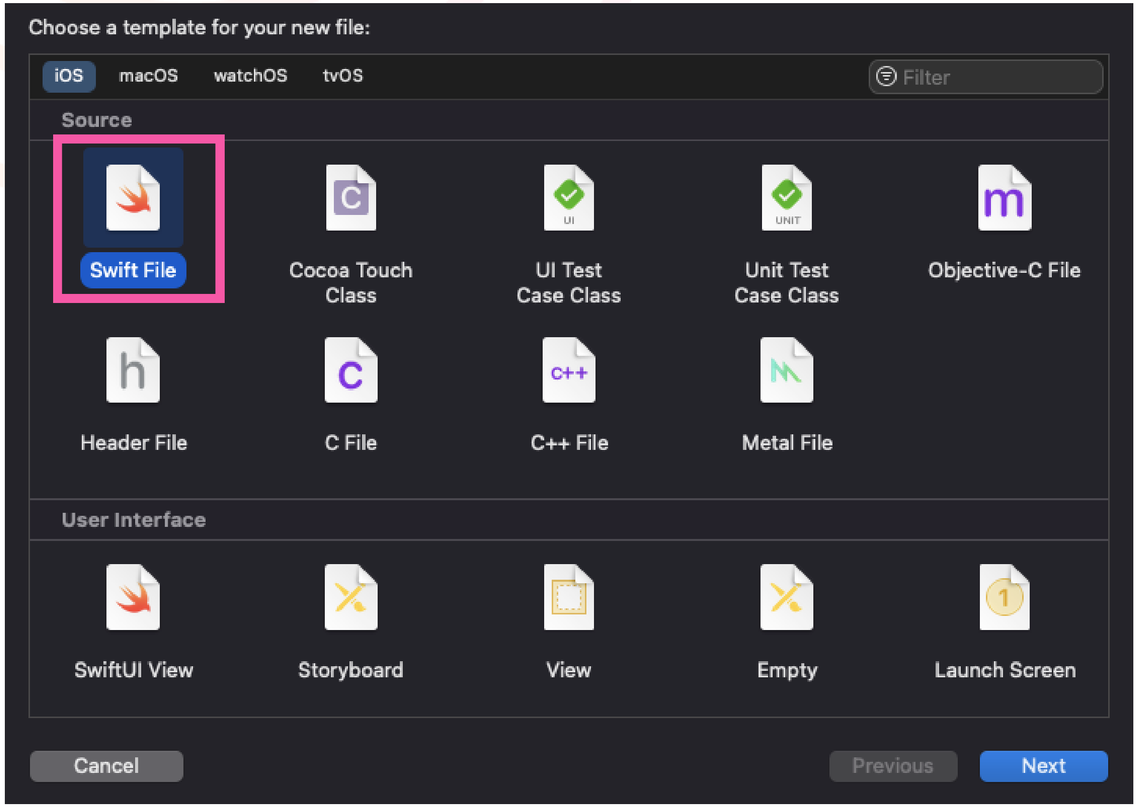
- 객체가 초기화(인스턴스화)된 후 호출되는 메서드입니다.
- ViewDidLoad( ) 와 비슷한 개념으로 생각하시면 됩니다!
- awakeFromNib( )가 일어나기 전에 Init(coder:) 가 호출됩니다.
  → 코드로 Cell을 짜는 경우에는 이 부분을 작성해야합니다.
- 스토리보드나 ,xib 형태로 구성하는 경우에는 init을 따로 해주지 않더라도 자동으로 awakeFromNib( )으로 호출이 됩니다

- init(coder:) 시점에는 frame 이나 layer 관련 없는 값들만 접근이 가능하고 awakeFromNib( ) 시점에는 Frame,Layer,@IBoutlet 관련 값 접근 가능합니다!

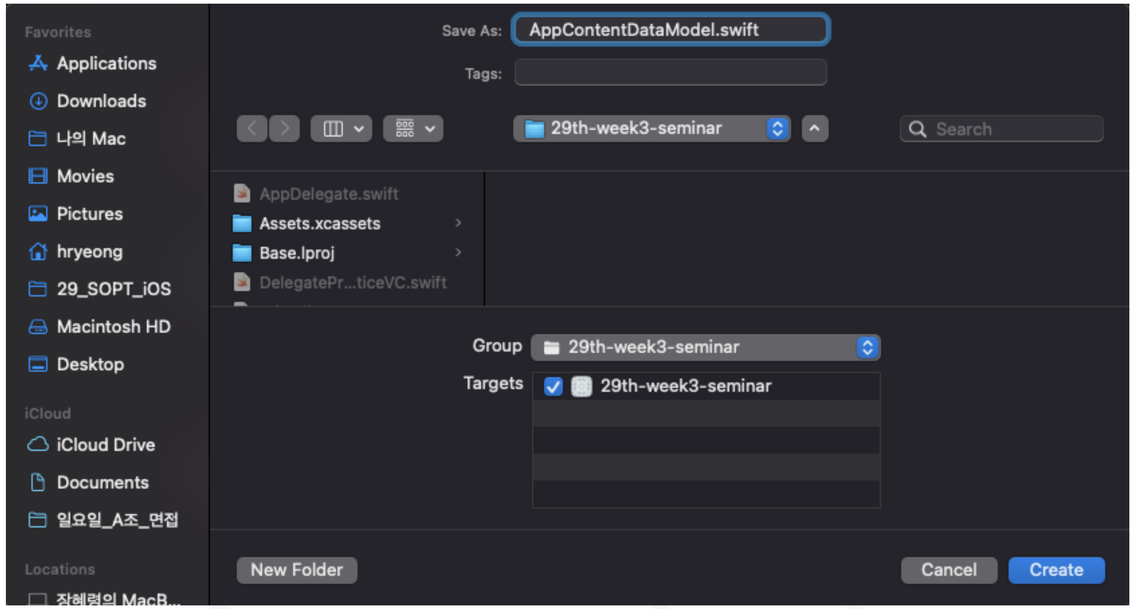
UITableView

먼저, 데이터 전달을 쉽게 하기 위해서
구조체로 - 새로운 데이터형을 만들어봅시다!

1. 파일추가(cmd+n)를 눌러 Swift 파일을 선택한 후 Next



2. 파일명 “AppContentDataModel.swift” Create



UITableView

```swift
import UIKit

struct AppContentData{
    let appName: String
    let description: String
    let appIconName: String

    func makeImage() -> UIImage? {
        return UIImage(named: appIconName)
    }
}
```

앱 이름, 설명, 앱 아이콘 이미지 이름을 넣는
AppContentData 모델 추가

UITableView

```
class RankingTableViewCell: UITableViewCell {
    static let identifier = "RankingTableViewCell"

    @IBOutlet weak var appIconImageView: UIImageView!
    @IBOutlet weak var rankLabel: UILabel!
    @IBOutlet weak var titleLabel: UILabel!
    @IBOutlet weak var subTitleLabel: UILabel!
    @IBOutlet weak var openButton: UIButton!

    override func awakeFromNib() {
        super.awakeFromNib()
    }

    override func setSelected(_ selected: Bool, animated: Bool) {
        super.setSelected(selected, animated: animated)
    }

    func setData(rank: Int, appData: AppContentData) {
        rankLabel.text = "\(rank)"
        appIconImageView.image = appData.makeImage()
        titleLabel.text = appData.appName
        subTitleLabel.text = appData.description
    }

}
```

Cell을 구분하기 위한 Identifier을
static을 활용해 타입 프로퍼티로 선언합니다!
(이렇게 하면 오타로 인한 오류를 줄이는데 좋습니다!)

cell 별로 다른 데이터가 들어와야 합니다!
새로 만든 데이터 타입인 AppContentData를
함수 파라미터로 받고 이 매개변수를 통해
그 안에는 outlet으로 선언했었던
변수들에 데이터를 넣어줍니다!

UITableView

```
class RankingTableViewCell: UITableViewCell {
    static let identifier = "RankingTableViewCell"

    @IBOutlet weak var appIconImageView: UIImageView!
    @IBOutlet weak var rankLabel: UILabel!
    @IBOutlet weak var titleLabel: UILabel!
    @IBOutlet weak var subTitleLabel: UILabel!
    @IBOutlet weak var openButton: UIButton!

    override func awakeFromNib() {
        super.awakeFromNib()
    }

    override func setSelected(_ selected: Bool, animated: Bool) {
        super.setSelected(selected, animated: animated)
    }

    func setData(rank: Int, appData: AppContentData) {
        rankLabel.text = "\(rank)"
        appIconImageView.image = appData.makeImage()
        titleLabel.text = appData.appName
        subTitleLabel.text = appData.description
    }

}
```

Cell을 구분하기 위한 Identifier을
static을 활용해 타입 프로퍼티로 선언합니다!
(이렇게 하면 오타로 인한 오류를 줄이는데 좋습니다!)

Tableview에 들어가는 Cell에 대한
UI 구성과 데이터 넣는 것까지 되었습니다!

이제 실제 Tableview에 이 cell 어떻게 보여주는지로 가보겠습니다!

cell 별로 다른 데이터가 들어와야 합니다!
새로 만든 데이터 타입인 AppContentData를
함수 파라미터로 받고 이 매개변수를 통해
그 안에는 outlet으로 선언했었던
변수들에 데이터를 넣어줍니다!

UITableView

tableview cell에 넣을 데이터 준비합니다

```
class RankingVC: UIViewController {

    @IBOutlet weak var rankTableView: UITableView!

    var appContentList: [AppContentData] = []
```

1. tableview cell에 넣을 앱 데이터 리스트 프로퍼티를 빈 배열로 선언

```
    override func viewDidLoad() {
        super.viewDidLoad()
        initAppContentList()
    }
```

3. viewDidLoad에서 호출시킵니다!

2. 실제 구조체 데이터를 넣는 함수를 만든 후

```
    func initAppContentList(){
        appContentList.append(contentsOf: [
            AppContentData(appName: "질병관리청",description: "전자예방접종증명서",appIconName: "coovIcon"),
            AppContentData(appName: "카카오톡", description: "소셜 네트워킹", appIconName: "kakaoIcon"),
            AppContentData(appName: "YouTube", description: "동영상과 음악", appIconName: "youtubeIcon"),
            AppContentData(appName: "네이버", description: "검색과 콘텐츠, 쇼핑, 내도구", appIconName: "naverIcon"),
            AppContentData(appName: "Instagram", description: "사진 및 비디오", appIconName: "instagramIcon"),
            AppContentData(appName: "쿠팡", description: "스마트한 쇼핑", appIconName: "coupangIcon"),
            AppContentData(appName: "배달의 민족", description: "대한민국 1등 배달앱", appIconName: "baeminIcon"),
            AppContentData(appName: "당근마켓", description: "중고거래", appIconName: "karrotIcon"),
            AppContentData(appName: "Netflix", description: "지금 바로 시청하세요", appIconName: "netfilxIcon"),
            AppContentData(appName: "무신사", description: "브랜드 패션 쇼핑", appIconName: "musinsaIcon")
        ])
    }

}
```

UITableView

```
class RankingVC: UIViewController {

    @IBOutlet weak var rankTableView: UITableView!

    override func viewDidLoad() {
        super.viewDidLoad()
    }

}

extension RankingVC: UITableViewDelegate {

}

extension RankingVC: UITableViewDataSource{

}
```

tableview를 구현하기 위해서는
중요한 2개의 프로토콜을 채택해야합니다!

✅UITableViewDelegate
✅UITableViewDataSource

📌 UITableView 동작원리

- TableView는 재사용큐를 활용해서 메모리를 효율적으로 사용 할 수 있도록 해주는 클래스입니다.
- 1000개의 Data가 있다고 생각하면 TableView는 1000개의 데이터를 먼저 생성하지 않습니다.
  우선 화면에 보이는 것 먼저 생성합니다. 즉, 예로 6개가 화면에 먼저 생성되고 스크롤을 하면 보이지 않는 영역은
  Reuse Queue의 풀에 들어가고 새롭게 보이는 Cell에 해당 풀에서 꺼내 와서 새롭게 보여주게 됩니다.
- 이렇게 재사용되지 때문에 각 cell의 identifier가 굉장히 중요합니다!

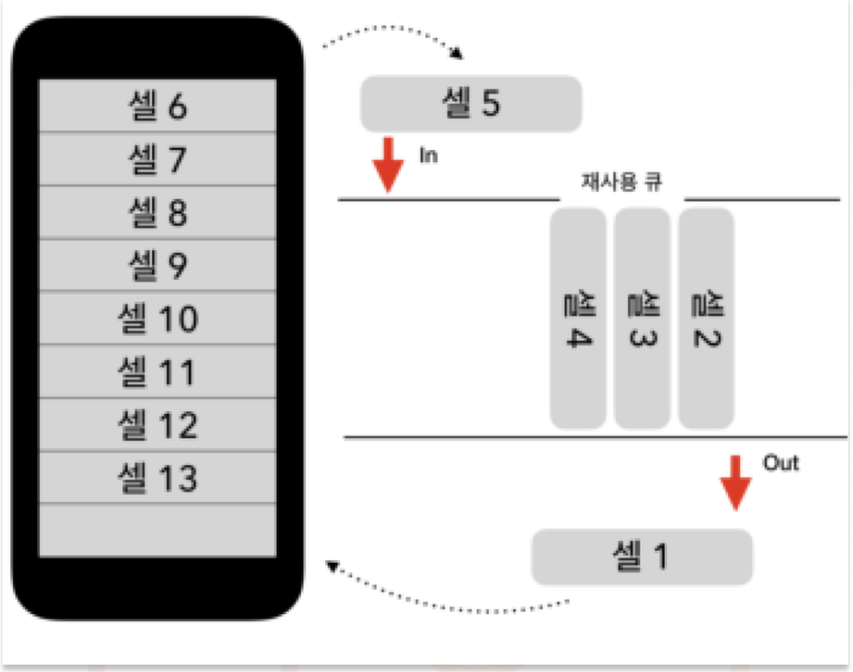
테이블 뷰 입장에서 필요할 때마다

재사용 큐한테 물어봅니다!

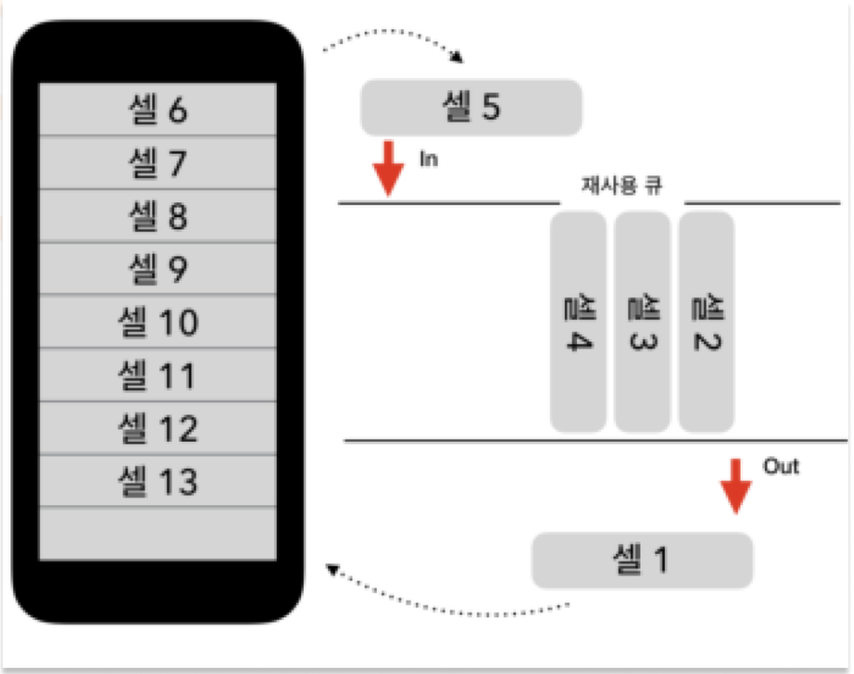
“ 나 이 Cell 이 필요한데 지금 그 Cell 큐에 있어? “

없으면 새로운 셀 생성

있으면 재사용 큐에서 그 Cell을 재사용합니다!



📌 UITableView 동작원리



테이블의 동작원리는 복잡합니다.. 😫

그렇기 때문에

이러한 세세한 동작 원리를 이해하지 못하더라도

사용할 수 있도록 2가지 프로토콜 형식으로 제공하고 있습니다

✅UITableViewDelegate

화면에 보이는 모습과 행동을 담당

(테이블 뷰의 동작과 모양을 관리)

- cell의 높이는 얼마인지?
- cell을 선택하면 무엇을 하면 되는지?

✅UITableViewDataSource

테이블을 만들 때 필요한 정보를 제공

(데이터를 받아서 뷰를 그려주는 역할)

- 총 섹션과 그 섹션안에 들어가는 cell 개수는 몇개인지?
- 각 row에 어떤 cell을 보여줄 것인지?

✅UITableViewDataSource

테이블을 만들 때 필요한 정보를 제공

(데이터를 받아서 뷰를 그려주는 역할)

- 총 섹션과 그 섹션안에 들어가는 cell 개수는 몇개인지?
- 각 row에 어떤 cell을 보여줄 것인지?

=> tableview를 구성하는데 필수적인 내용들

```
extension RankingVC: UITableViewDataSource{

}
```

Type 'RankingVC' does not conform to protocol 'UITableViewDataSource'

Do you want to add protocol stubs? Fix

fix를 눌러 필수적으로 구현할 함수들을 추가해주세요!

UITableView

RankingVC.swift

```
extension RankingVC: UITableViewDataSource{
    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return appContentList.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        guard let cell = tableView.dequeueReusableCell(withIdentifier: RankingTableViewCell.identifier) as?
            RankingTableViewCell else {return UITableViewCell()}

        cell.setData(rank: indexPath.row, appData: appContentList[indexPath.row])
        return cell
    }
}
```

section마다 몇개의 row를 넣어야 하는지 지정하는 함수

현재 저희가 구현할 tableview는 section이 하나밖에 없고,

appContend 리스트에 넣은 10개의 데이터 만큼

row가 그려져야 한다는 의미입니다.

UITableView

RankingVC.swift

```
extension RankingVC: UITableViewDataSource{
    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return appContentList.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        guard let cell = tableView.dequeueReusableCell(withIdentifier: RankingTableViewCell.identifier) as?
            RankingTableViewCell else {return UITableViewCell()}

        cell.setData(rank: indexPath.row, appData: appContentList[indexPath.row])
        return cell
    }



}
```

indexPath의 프로퍼티를 살펴보면 대표적으로 Section과 Row가 있습니다.
이 indexPath를 활용해 TableviewCell 별로 데이터를 다르게 지정해줄 수 있습니다.

UITableView

RankingVC.swift

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    guard let cell = tableView.dequeueReusableCell(withIdentifier: RankingTableViewCell.identifier) as?
        RankingTableViewCell else {return UITableViewCell()}

    cell.setData(rank: indexPath.row, appData: appContentList[indexPath.row])
    return cell
}
```

이전에 tableview 동작원리 얘기하면서 cell을 재사용할 때 tableview에게 물어본다고 했었습니다.

지금 tableview에게 " 나 지금 이 Cell이 필요한데 해당 Identifier 가진 Cell이 재사용 큐에 있을까??"

있으면 UITableViewCell 객체가 넘어오고 아니면 nil이 넘어오게 됩니다

그리고 UITableviewCell 타입으로 넘어온 객체를 as?를 통해 원하는 하위클래스로 타입변환을 합니다.

이렇게 되면, let cell은 RankingTableViewCell 객체이기 때문에

저희가 이전에 RankingTableViewCell에 정의한 데이터를 전달받는 함수를 사용할 수 있습니다!

UITableView

UITableViewDelegate는 테이블뷰의 시각적인 부분 수정, 행의 선택 관리, 액세서리뷰 지원 그리고 테이블뷰의 개별 행 편집을 도와줍니다.
이 안에도 구현할 수 있는 기능이 다양한데 이번에는 heightForRowAt해서 indexPath별의 높이만 지정하는 함수 추가해주세요!

```
extension RankingVC: UITableViewDelegate {
    func tableView(_ tableView: UITableView,
                   heightForRowAt indexPath: IndexPath) -> CGFloat {
        return 80
    }
}
```

// 특정 위치 행의 높이를 묻는 메서드
func tableView(UITableView, heightForRowAt: IndexPath)
// 특정 위치 행의 들여쓰기 수준을 묻는 메서드
func tableView(UITableView, indentationLevelForRowAt: IndexPath)

// 지정된 행이 선택되었음을 알리는 메서드
func tableView(UITableView, didSelectRowAt: IndexPath)

// 지정된 행의 선택이 해제되었음을 알리는 메서드
func tableView(UITableView, didDeselectRowAt: IndexPath)

// 특정 섹션의 헤더뷰 또는 푸터뷰를 요청하는 메서드
func tableView(UITableView, viewForHeaderInSection: Int)
func tableView(UITableView, viewForFooterInSection: Int)

// 특정 섹션의 헤더뷰 또는 푸터뷰의 높이를 물어보는 메서드
func tableView(UITableView, heightForHeaderInSection: Int)
func tableView(UITableView, heightForFooterInSection: Int)

// 테이블뷰가 편집모드에 들어갔음을 알리는 메서드
func tableView(UITableView, willBeginEditingRowAt: IndexPath)

// 테이블뷰가 편집모드에서 빠져나왔음을 알리는 메서드
func tableView(UITableView, didEndEditingRowAt: IndexPath?)

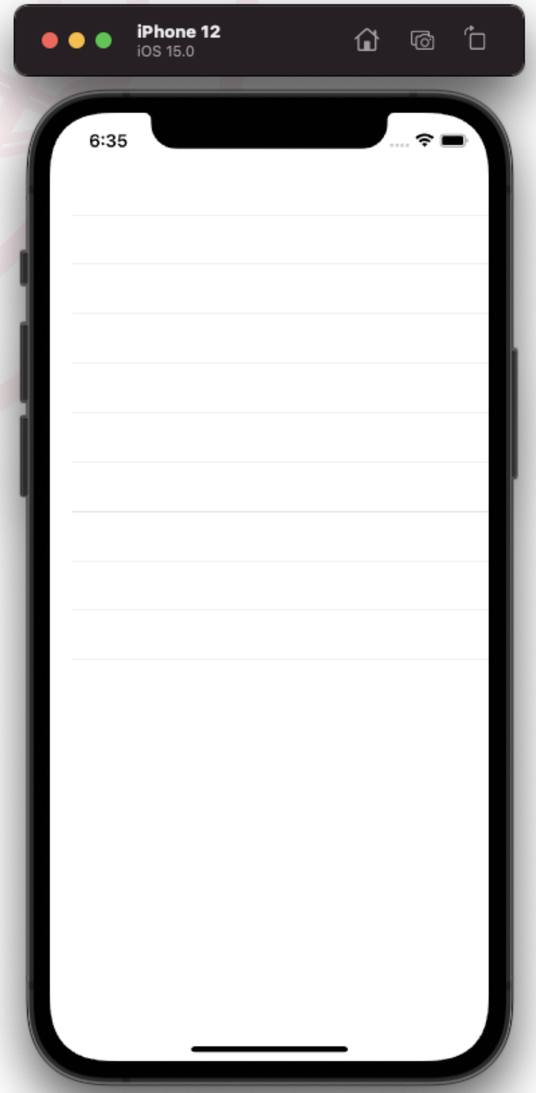
UITableView

```
override func viewDidLoad() {
    super.viewDidLoad()
    initAppContentList()
    rankTableView.dataSource = self
    rankTableView.delegate = self
}
```

rankTableView의
dataSource, delegate
역할을 수행할 대리인으로서
self (RankingVC)으로 위임하였습니다.

이제 빌드시켜 확인해볼까요??

UITableView



아무것도 안뜬다....😰

UITableView

tableview cell을 구현하는 방법이 크게 3가지라고 그랬죠?

1. 스토리보드 내에서 TableviewCell 활용해서 구성 (Prototype Cell)
2. Xib 파일로 따로 분리해서 Cell UI 구성
3. Code base로 Cell 구성

tableview 마다 cell로서 사용할 객체 or 클래스를 등록해야합니다!

```
func register(UINib?, forCellReuseIdentifier: String)
```

지정된 식별자 아래의 테이블 보기가 있는 셀을 포함하는 nib 객체를 등록합니다.

```
func register(AnyClass?, forCellReuseIdentifier: String)
```

새 테이블 셀을 만드는 데 사용할 클래스를 등록합니다.

2번 xib 등록방법

3번 코드베이스로 짰을 때 클래스 등록 방법

UITableView

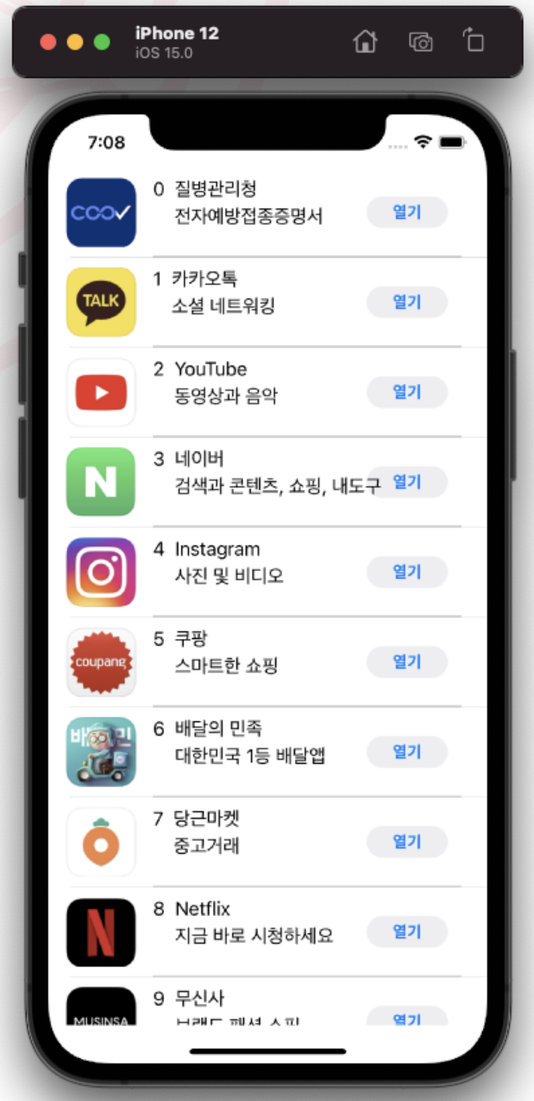
```
override func viewDidLoad() {
    super.viewDidLoad()
    initAppContentList()
    registerXib()
    rankTableView.dataSource = self
}

func registerXib(){
    let xibName = UINib(nibName: RankingTableViewCell.identifier, bundle: nil)
    rankTableView.register(xibName, forCellReuseIdentifier: RankingTableViewCell.identifier)
}
```

Xib? Nib? 뭐가 다른거에요? 🤔

- nib(NeXT Interface Builder): 뷰의 layout, display등의 요소들을 object graph로 만들어서 직렬화한 파일
- 인터페이스 빌더에서 구성한 모든 정보는 .xib파일(XML Interface Builder)라는 파일로 저장
- 프로젝트 컴파일 시 바이너리파일인 .nib파일이 되는 것

=> xib 파일이 컴파일된 파일이 nib 파일입니다 즉, 둘은 근복적으로 같은 이야기를 하고 있는 파일입니다!

UITableView

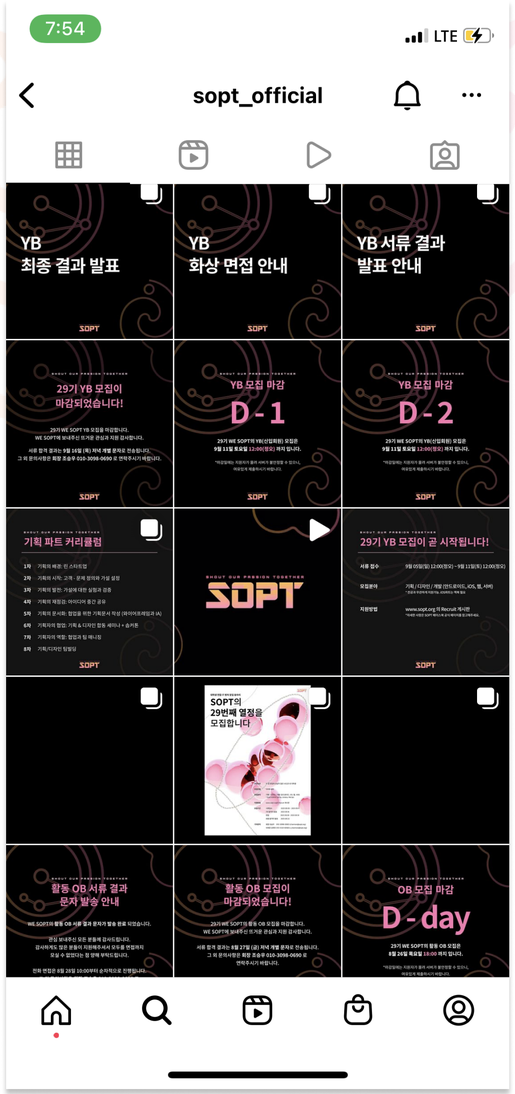


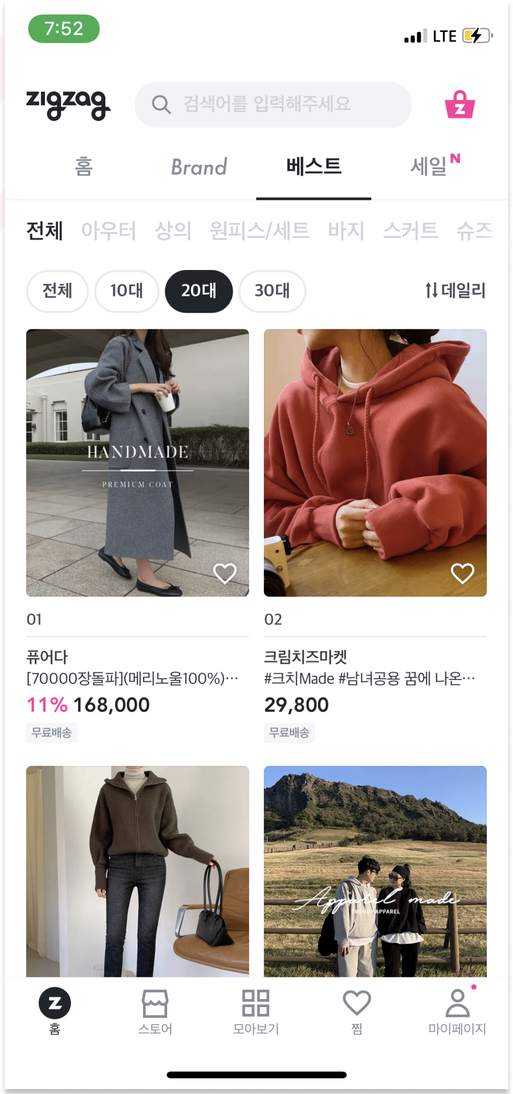
빌드시켜확인해보세요 :)

04

CollectionView

UICollectionView

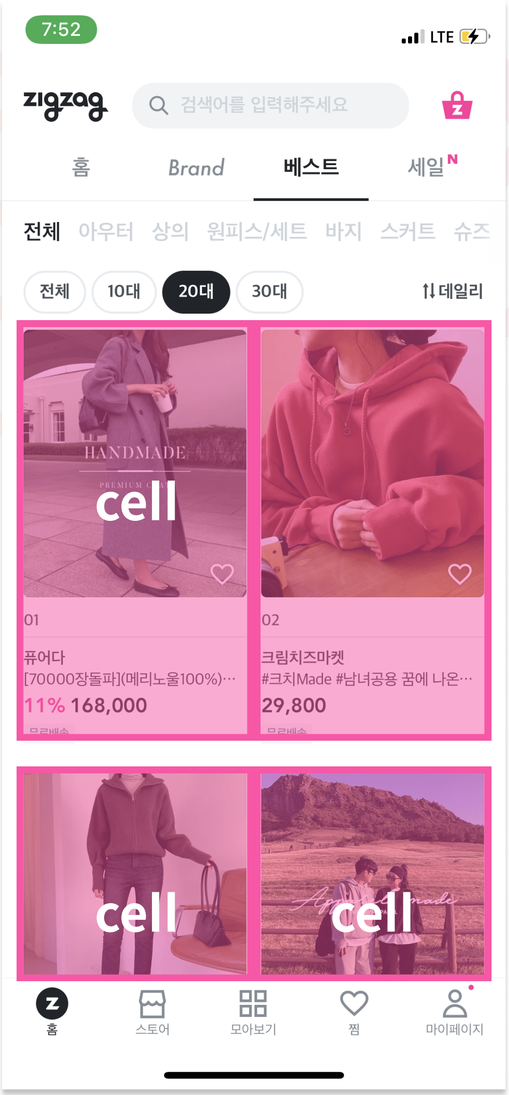


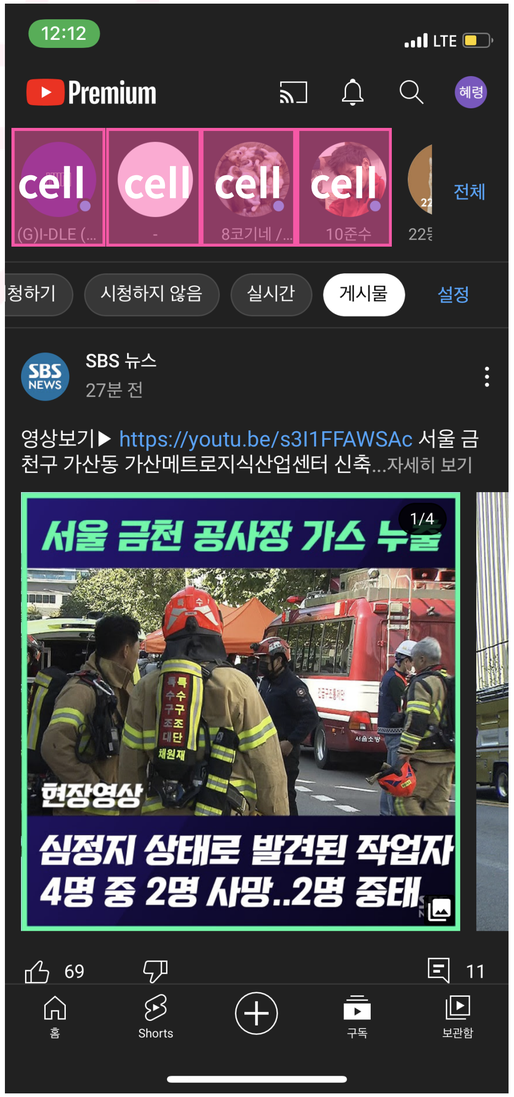


평소에 우리가 서비스를 사용하다보면,
일자 목록이 아닌
그리드 형태로 아이템이 나열 된 것을 볼 수 있는데
대부분 CollectionView로 구현된 형태입니다

CollectionView도
Tableview와 비슷하게 UIScrollView를 상속 받았고,
Delegate Pattern 을 활용해 구현하게 됩니다!
TableView 사용법과 매우매우 비슷합니다

UICollectionView





CollectionView도 Tableview와 마찬가지로

Cell 내부에 라벨, 이미지, 버튼 등을 넣은 컨텐츠가

같은 형태, 다른 데이터 구조로 반복되고 있는 것을 볼 수있습니다!

TableView하고의 차이점은

테이블 뷰는 리스트 목록 구현에 맞춘 형태라면,

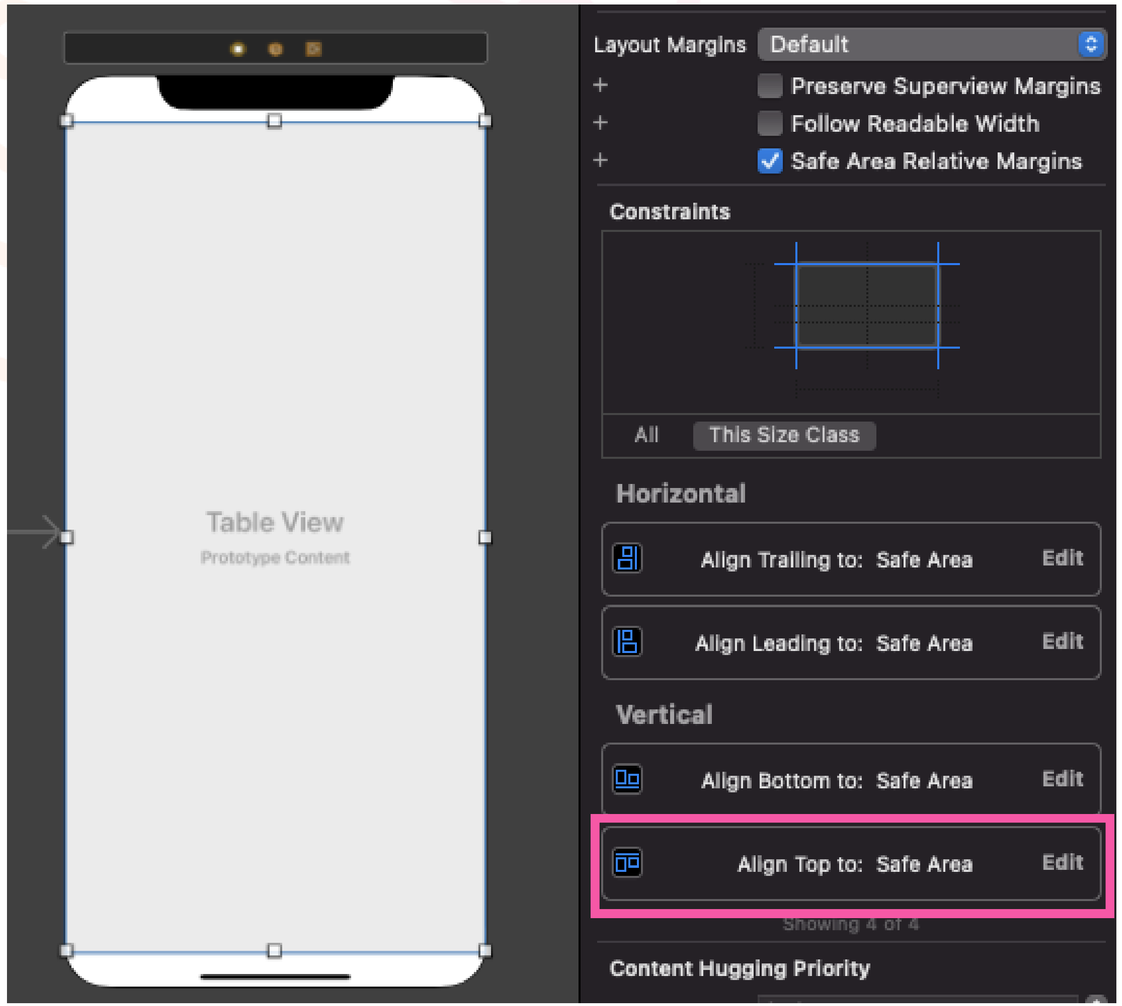
컬렉션뷰는 그리드 기반 레이아웃으로

더 복잡하고 다양한 형태로 커스텀이 가능합니다!

UICollectionView

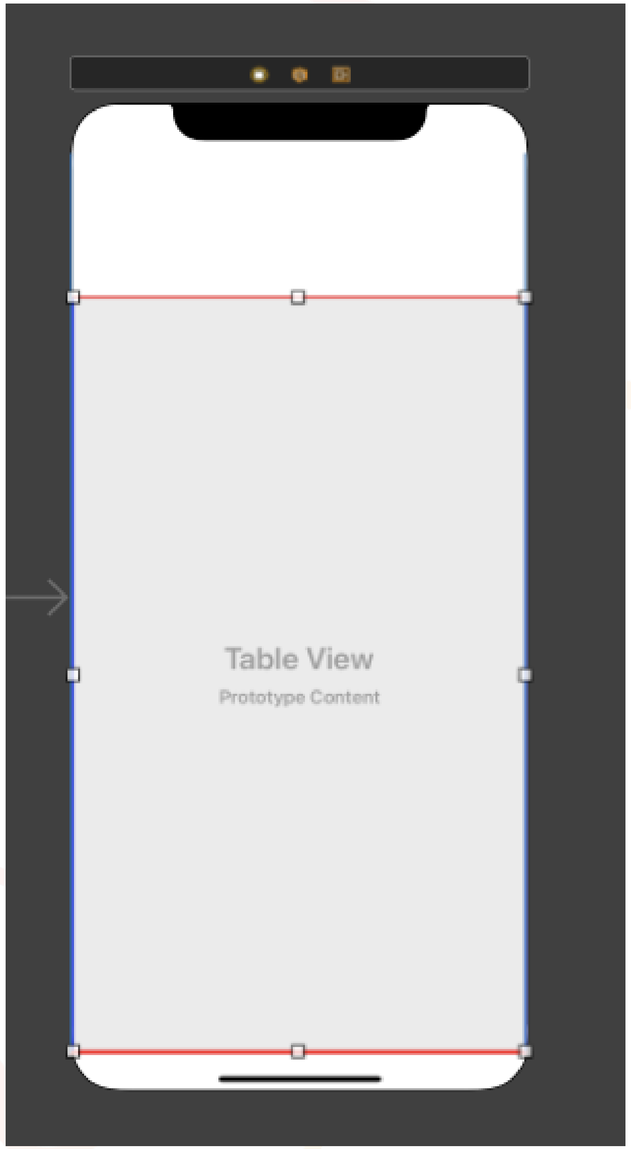
실습해봅시다!

UICollectionView

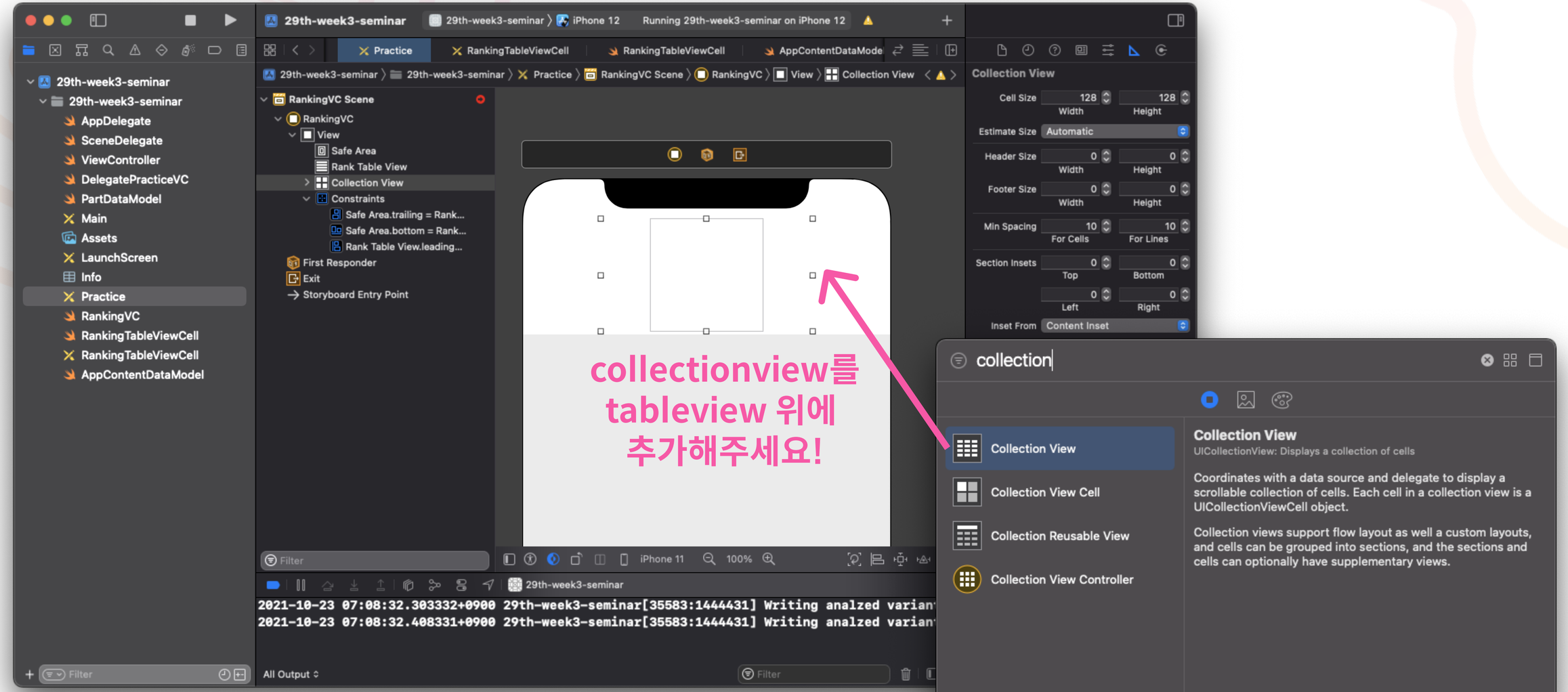


1. 클릭하고
delete키 눌러서
삭제해주세요

2. 위에 collectionview를
넣을 수 있도록
tableview top을
조금 내려주세요!

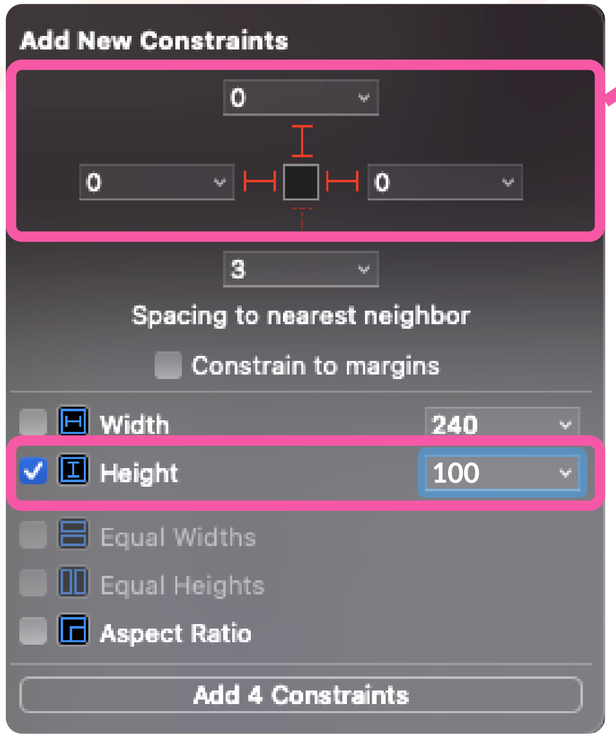


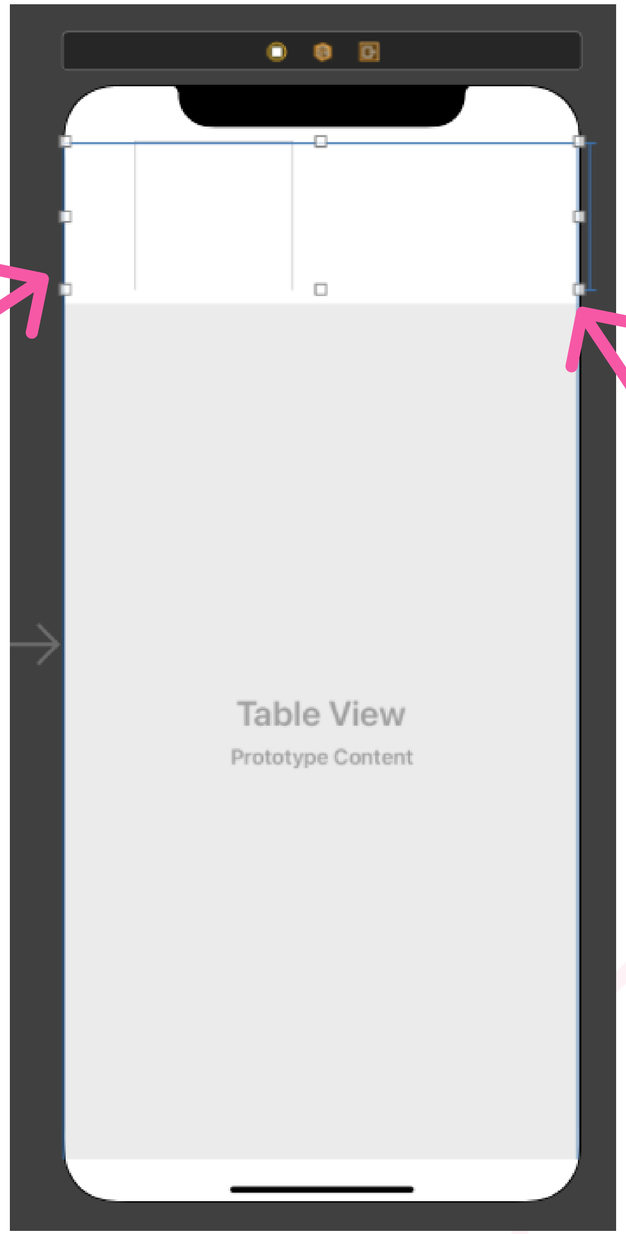
UICollectionView



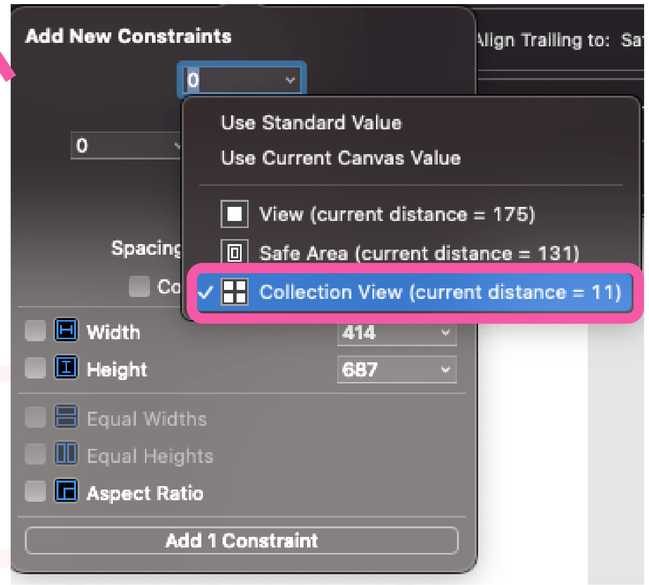
UICollectionView

collectionview
top, leading, trailing : 0
height: 100
총 4개의 constraints 추가



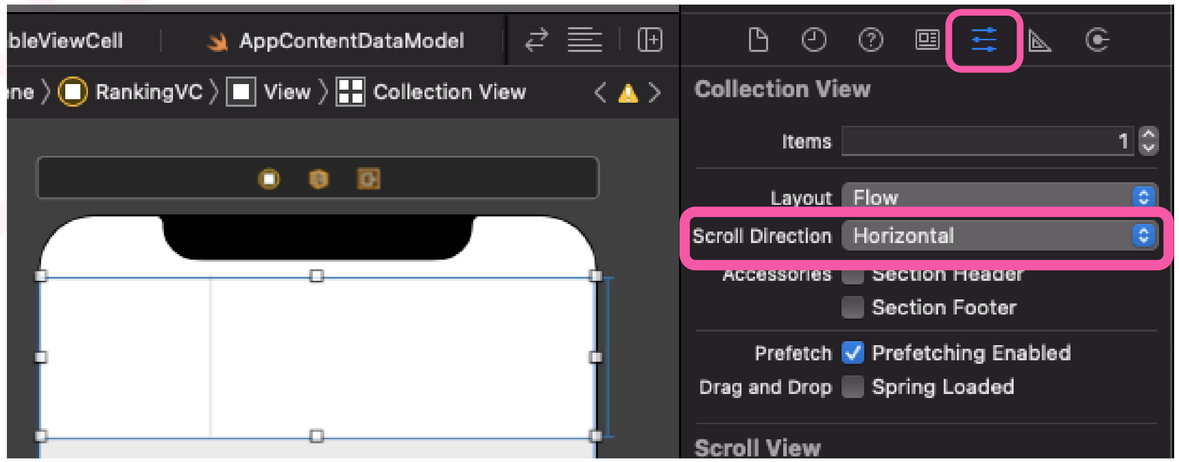


잃어버렸던 tableview의 top을
collection view의 bottom과
0이 되도록 추가해주세요!



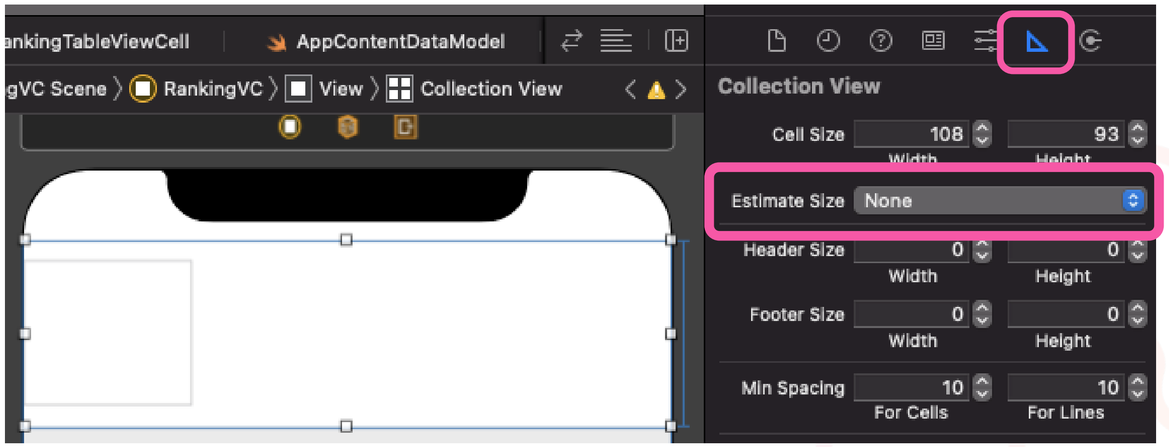
UICollectionView

가로 스크롤로 구현할 것이기 때문에 방향을 Horizontal로 바꿔주세요!

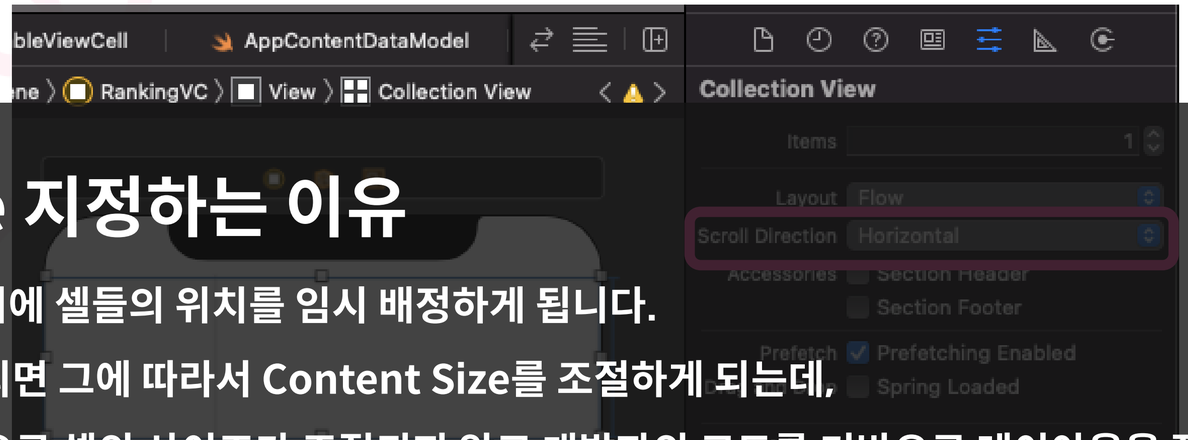


collection view를 선택한 후

Estimate Size는 none으로 설정해주세요!



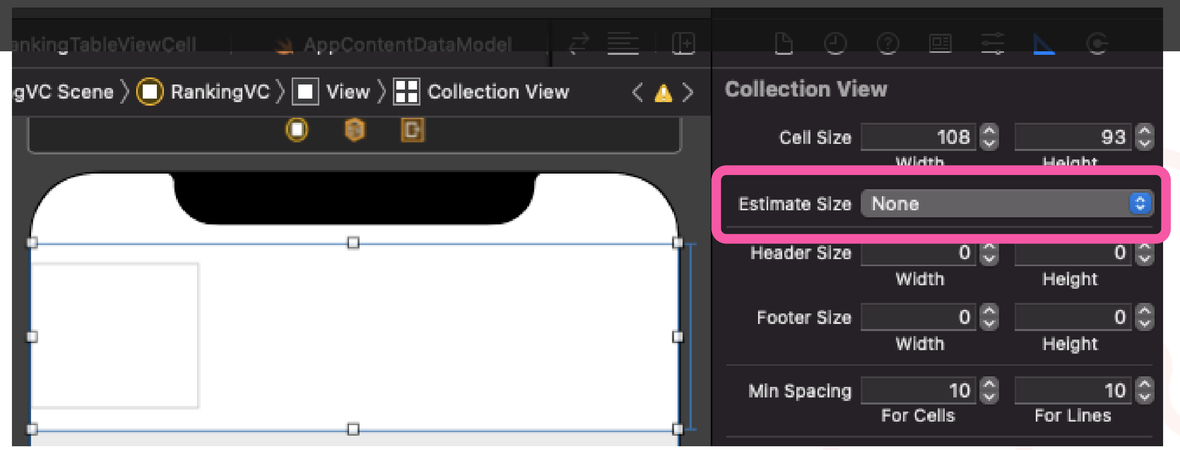
가로 스크롤로 구현할 것이기 때문에 방향을 Horizontal로 바꿔주세요!



💡 Estimate Size 지정하는 이유

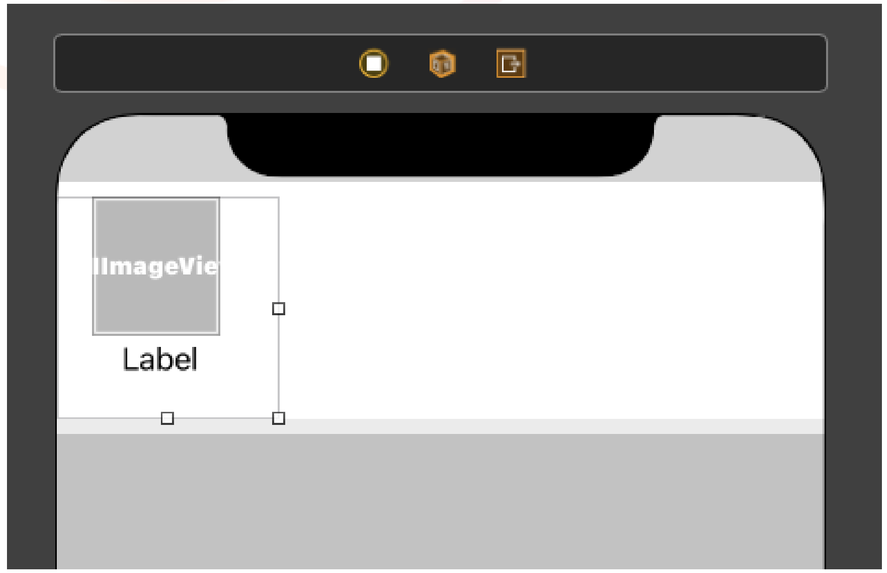
- Estimate Size를 통해서 초기에 셀들의 위치를 임시 배정하게 됩니다.
- Autolayout 연산을 마치게 되면 그에 따라서 Content Size를 조절하게 되는데,
  Cell의 AutoLayout을 기반으로 셀의 사이즈가 조절되지 않고 개발자의 코드를 기반으로 레이아웃을 잡기 위해 None으로 지정합니다.
- UICollectionViewDelegateFlowLayout을 채택하여 메소드를 구현하여 해당 Cell의 사이즈를 동적으로 조절해줄 수 있습니다.

Estimate Size는 none으로 설정해주세요!

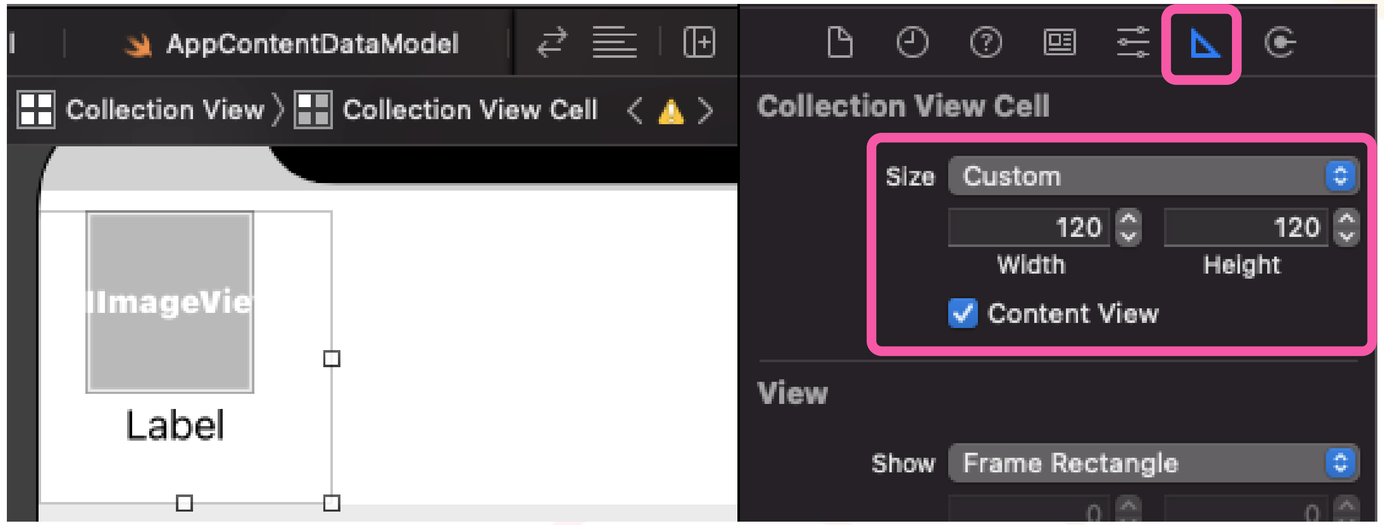


UICollectionView

cell안에
image view와
label을 하나씩 넣어주세요

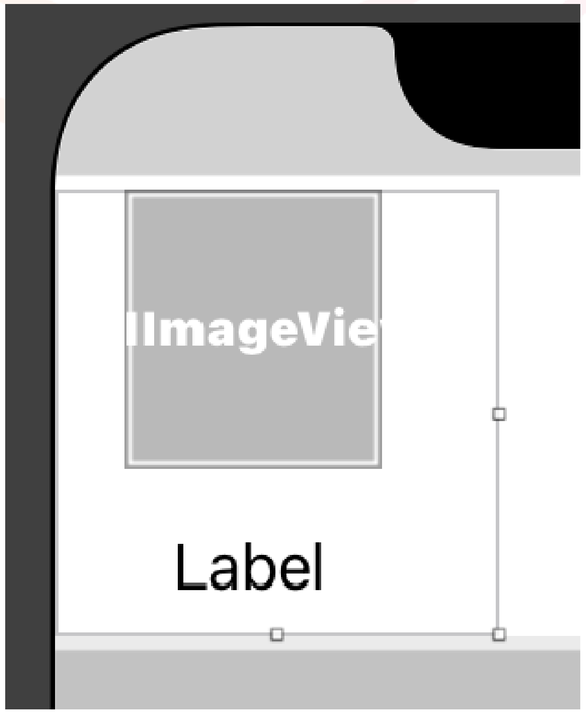


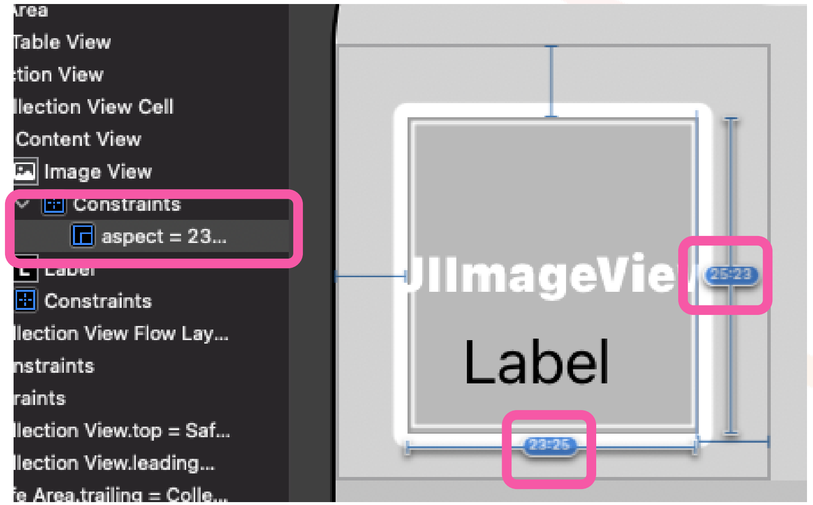
Autolayout 좀 더 편하게 잡기 위해
cell size를 width, height : 120으로 늘려주세요!



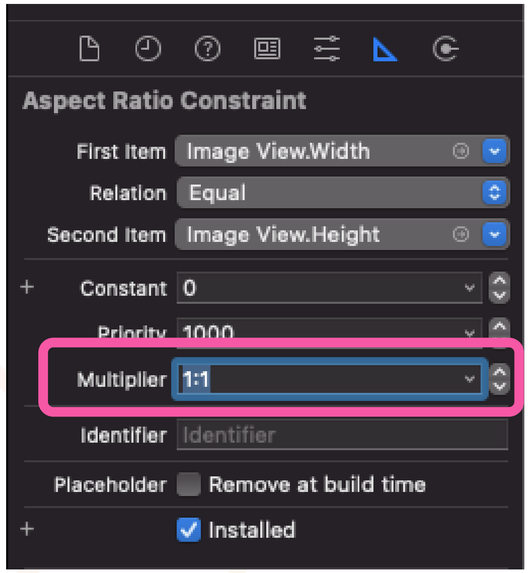
UICollectionView

imageView
top : 10
leading, trailing : 20
Aspect Ratio 체크





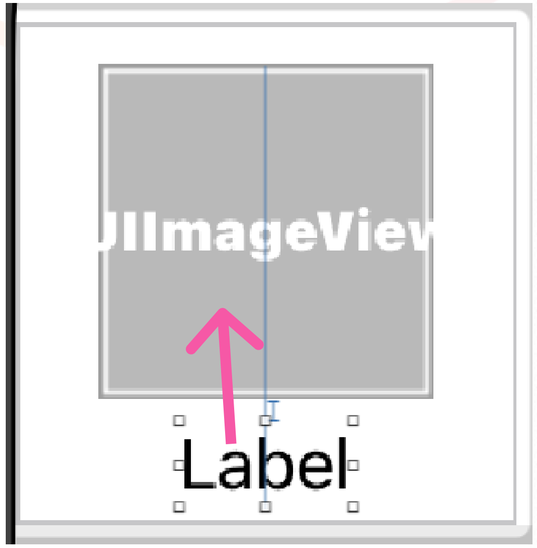
셋중에 하나
선택하여
비율을 1:1로
바꿔주세요:)

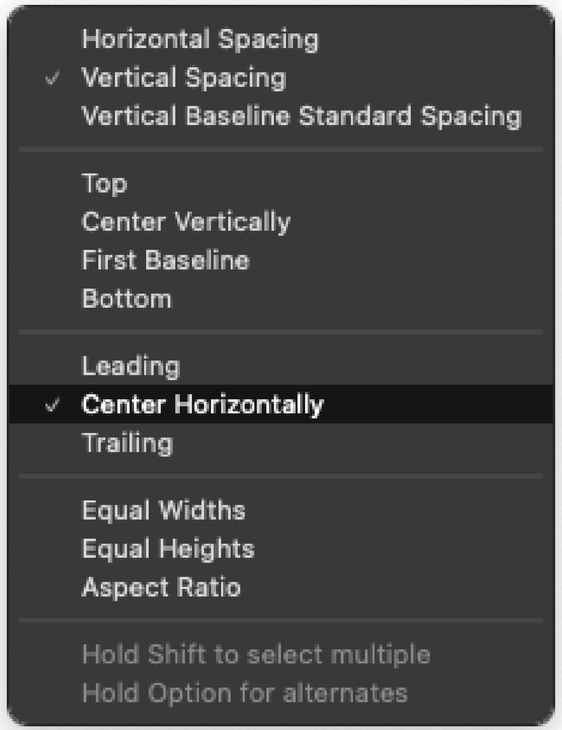


UICollectionView

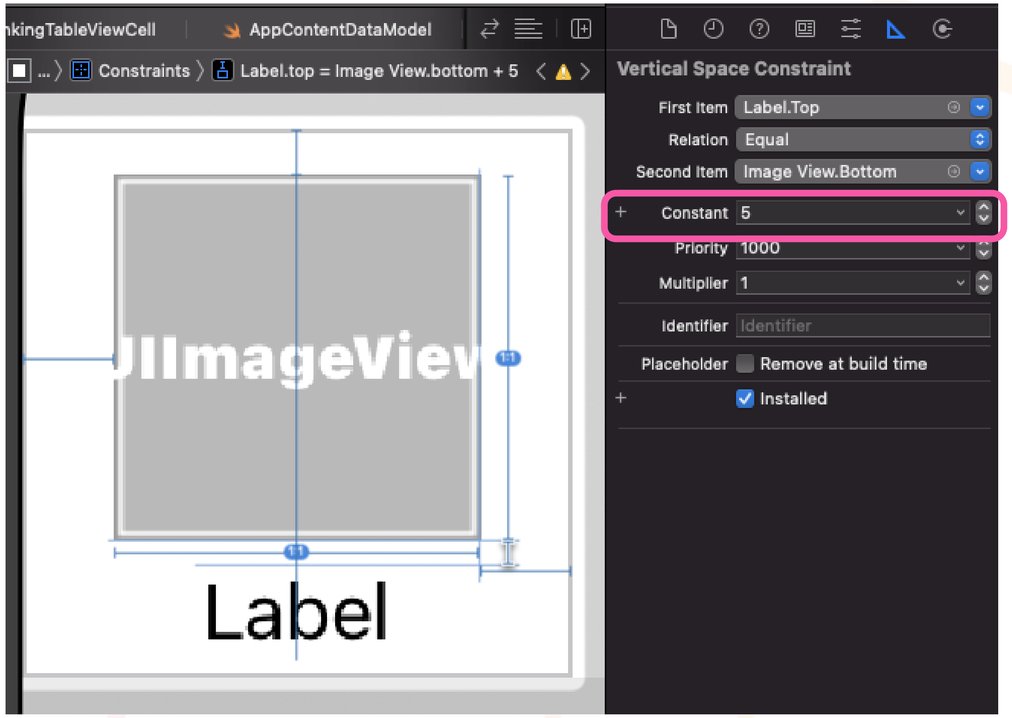
label을 imageview에
control를 누른 채 끌어다 놓은 후
Vertical Spacing (수직간격)
Center Horizontally (수평에서 센터)
두 개의 constraint를 추가해주세요!

*commond 누르면 중복선택 가능합니다



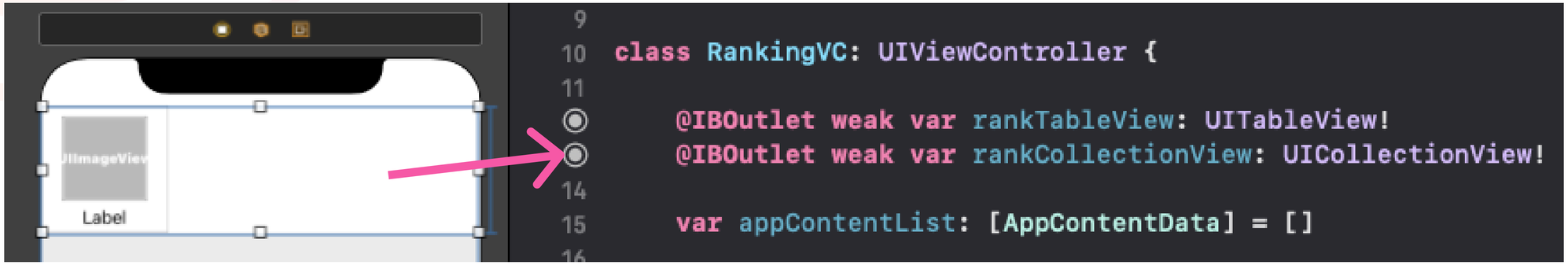


Label top과 imageview와의 간격을 5로 줄여주세요!



UICollectionView

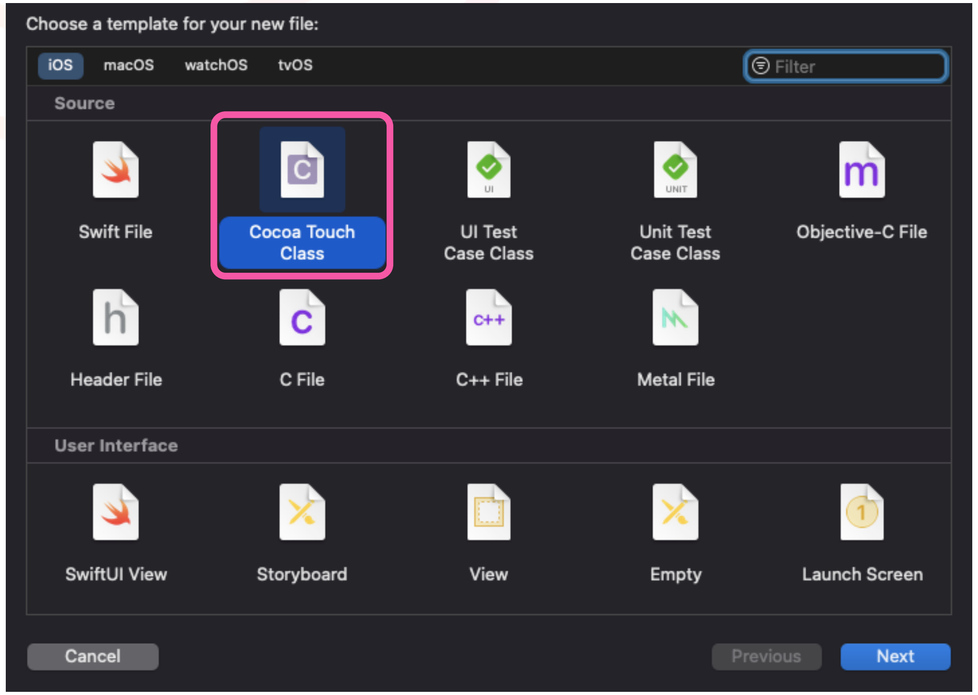
collection view outlet 연결해주세요



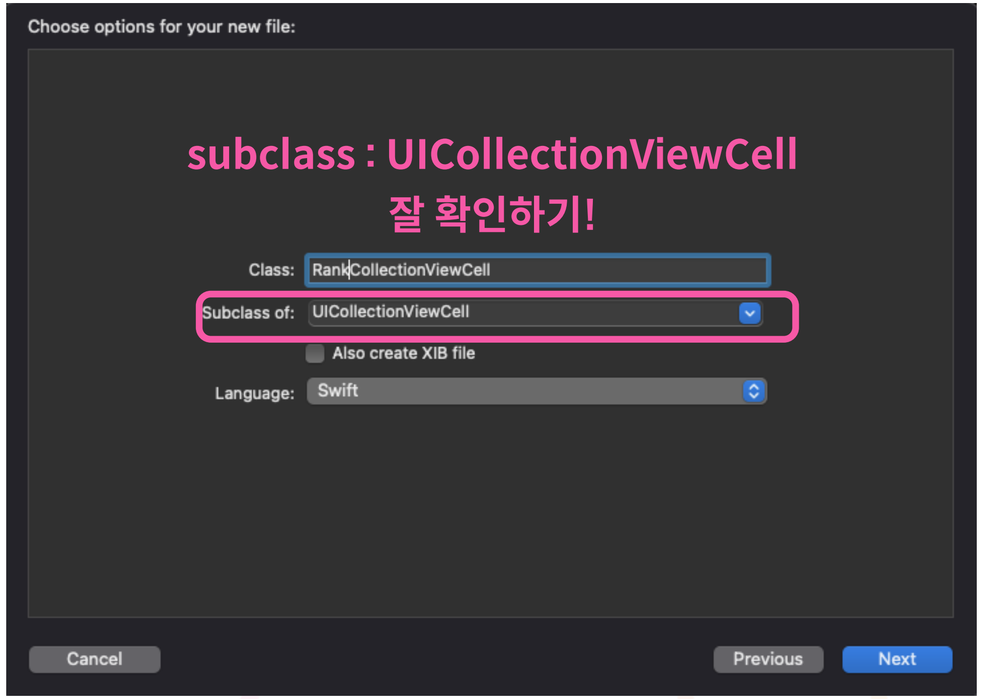
UICollectionView

collection view cell과 연결할 class 추가해주세요

1. 파일추가(cmd+n)를 눌러
코코아 터치 파일을 선택한 후 Next

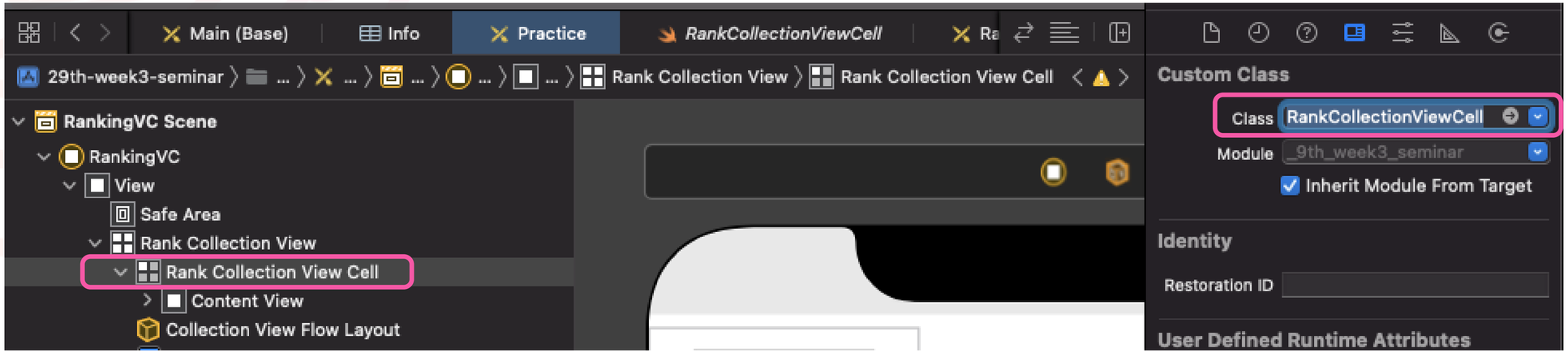


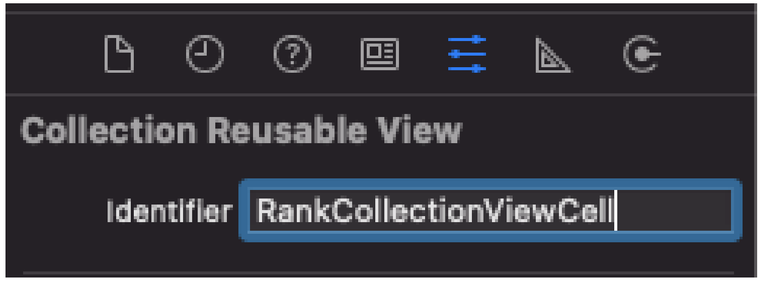
2. 파일명 “RankCollectionViewCell”



UICollectionView

collection view cell 을 눌러 방금만든
RankCollectionViewCell 클래스와 연결해주세요

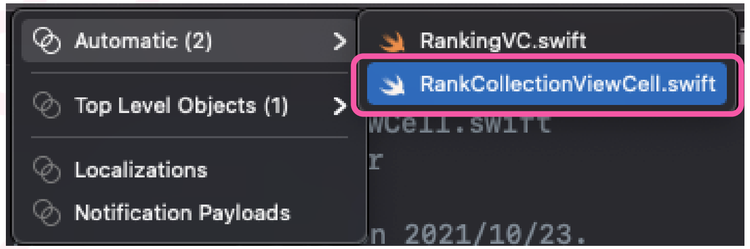




collection cell에도 클래스명을
identifier에 적어주세요!

UICollectionView

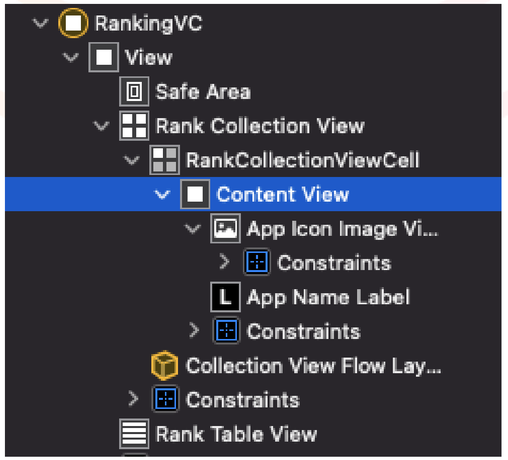
선택해주세요!




```swift
import UIKit

class RankCollectionViewCell: UICollectionViewCell {

}
```

collection cell 파일을

열어주세요:)



```swift
import UIKit

class RankingVC: UIViewController {

    @IBOutlet weak var rankTableView: UITableView!
    @IBOutlet weak var rankCollectionView: UICollectionView!

    var appContentList: [AppContentData] = []

    override func viewDidLoad() {
        super.viewDidLoad()
        initAppContentList()
        registerXib()
        rankTableView.dataSource = self
        rankTableView.delegate = self
    }
```

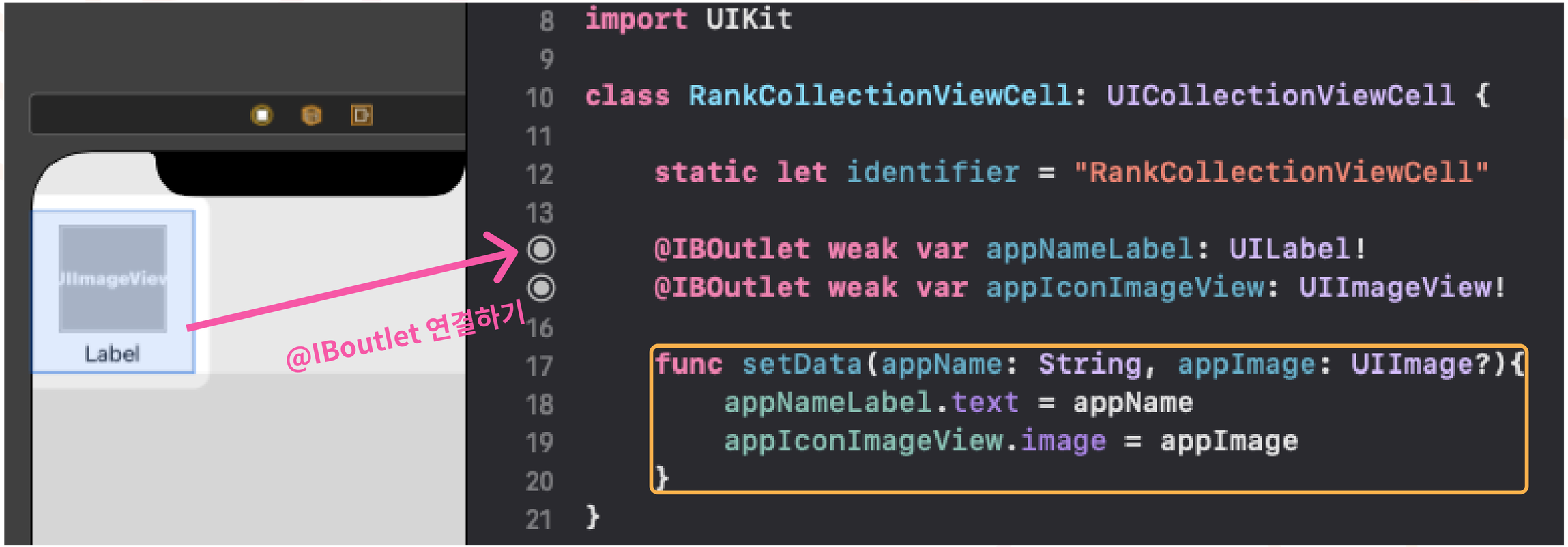
content view를 선택해서

(Cell이 아닌 Cell의 하위 뷰인 conten View를 눌러주세요!)

control + option + commond + enter를 눌러서

Assistant를 열어주세요!

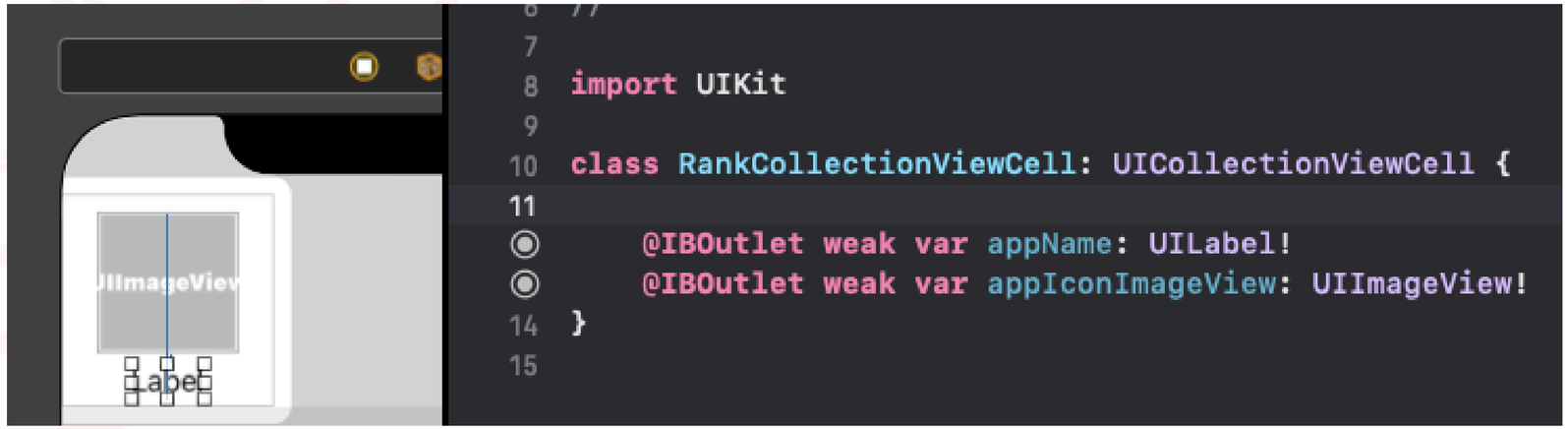
UICollectionView



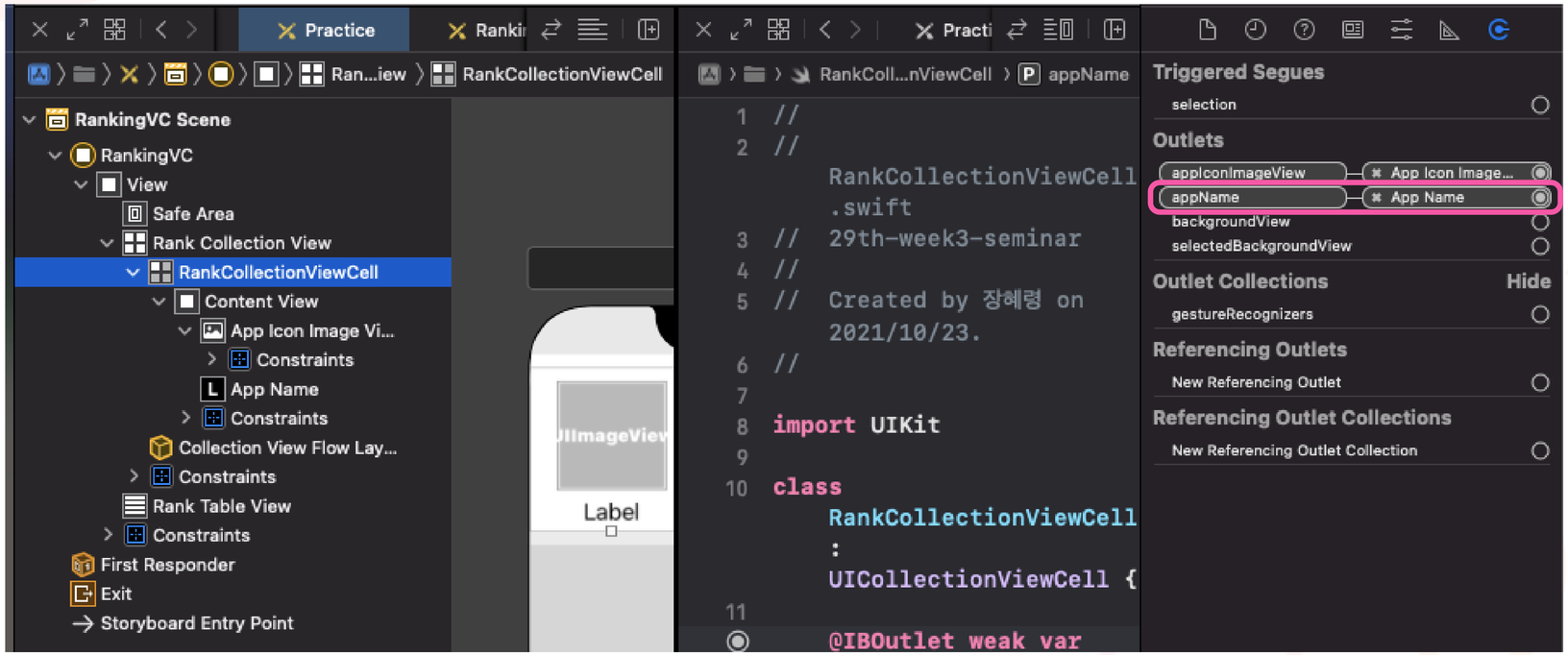

```
8   import UIKit
9
10  class RankCollectionViewCell: UICollectionViewCell {
11
12      static let identifier = "RankCollectionViewCell"
13
        @IBOutlet weak var appNameLabel: UILabel!
        @IBOutlet weak var appIconImageView: UIImageView!
16
17      func setData(appName: String, appImage: UIImage?){
18          appNameLabel.text = appName
19          appIconImageView.image = appImage
20      }
21  }
```

외부에서 데이터 넣어줄 함수 추가

UICollectionView

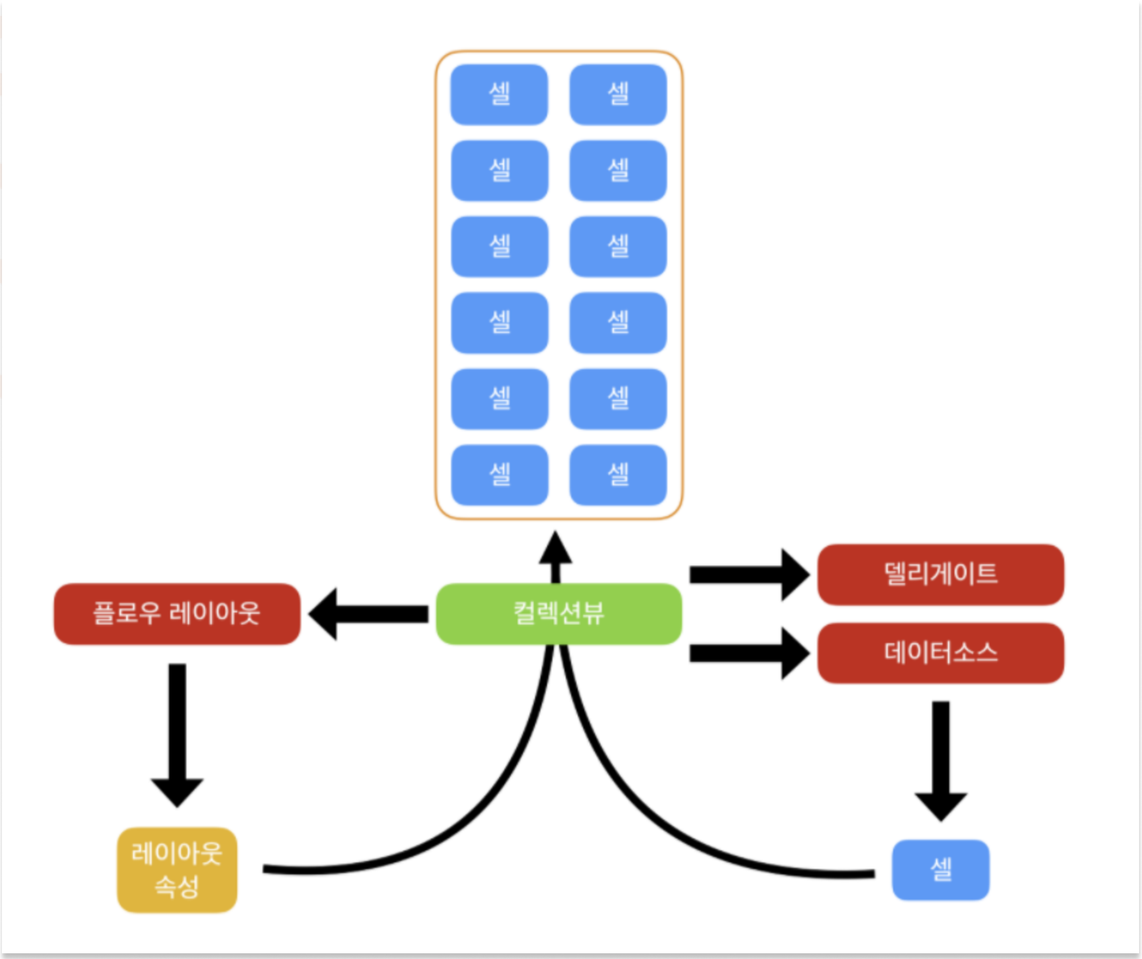


다시 연결하고 싶을 때!



여기서 X를 눌러
연결을 끊어주어야
깔끔하게 끊깁니다!

UICollectionView



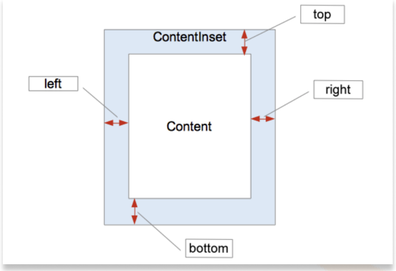
CollectionView의 경우에는
Delegate 와 **Datasource**를 통해서
Cell 자체를 구성하고

Cell 들의 간격 / 크기 등을
FlowLayout을 통해 결정하게 됩니다

그래서 collectionview에는 중요한 프로토콜이 3가지 있습니다
✅ UICollectionViewDataSource
✅ UICollectionViewDelegate
✅ UICollectionViewDelegateFlowLayout

📌 UICollectionViewDelegateFlowLayout

- 그리드 기반 레이아웃을 구현하기 위해 흐름 레이아웃 개체와 조정할 수 있는 메서드입니다.
- 4가지 메서드를 통해 Cell들의 Layout을 구성하게 됩니다.



📍 **sizeForItemAt :** cell 들을 size 지정, indexPath에 따라 달리 지정 가능

```
func collectionView(_ collectionView: UICollectionView,
                    layout collectionViewLayout: UICollectionViewLayout,
                    sizeForItemAt indexPath: IndexPath) -> CGSize
```

📍 **ContentInset :** Cell에서 Content 외부에 존재하는 Inset의 크기를 결정

```
func collectionView(_ collectionView: UICollectionView,
                    layout collectionViewLayout: UICollectionViewLayout,
                    insetForSectionAt section: Int) -> UIEdgeInsets
```

📍 **mininumLineSpacing :** Cell 들의 상,하 간격 조정

```
func collectionView(_ collectionView: UICollectionView,
                    layout collectionViewLayout: UICollectionViewLayout,
                    minimumLineSpacingForSectionAt section: Int) -> CGFloat
```

📍 **minimumInteritemSpacing :** Cell 들의 좌,우 간격 조정

```
func collectionView(_ collectionView: UICollectionView,
                    layout collectionViewLayout: UICollectionViewLayout,
                    minimumInteritemSpacingForSectionAt section: Int) -> CGFloat
```

UICollectionView

UICollectionViewDataSource

```
extension RankingVC: UICollectionViewDataSource{
    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
        return appContentList.count
    }

    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {
        guard let cell = collectionView.dequeueReusableCell(withReuseIdentifier: RankCollectionViewCell.identifier, for: indexPath)
            as? RankCollectionViewCell else {return UICollectionViewCell()}

        cell.setData(appName: appContentList[indexPath.row].appName,
                     appImage: appContentList[indexPath.row].makeImage())
        return cell
    }
}
```

section 마다 몇 개의 아이템이 들어가는지 결정하는 메소드

현재는 Section이 하나이기 때문에 appContentlist count로 item 개수 지정

UICollectionView

UICollectionViewDataSource

```
extension RankingVC: UICollectionViewDataSource{
    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
        return appContentList.count
    }
```

tableview와 비슷하게 해당 indexPath에 어떤 cell 데이터를 넣을지 결정하는 메소드

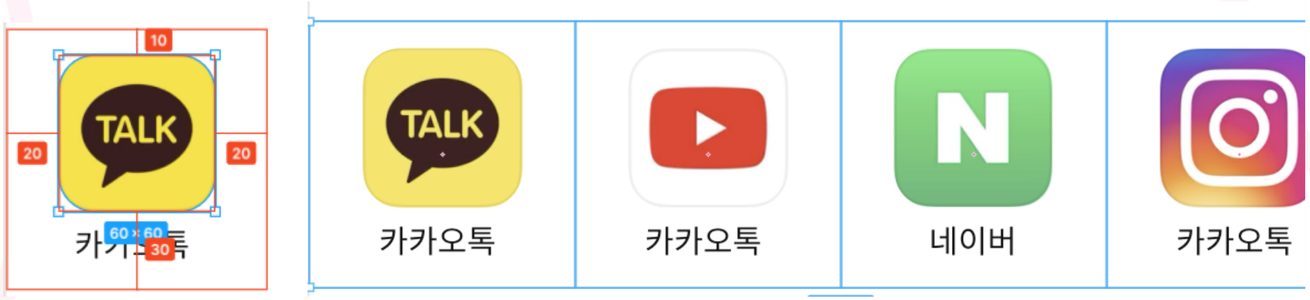
```
    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {
        guard let cell = collectionView.dequeueReusableCell(withReuseIdentifier: RankCollectionViewCell.identifier, for: indexPath)
            as? RankCollectionViewCell else {return UICollectionViewCell()}

        cell.setData(appName: appContentList[indexPath.row].appName,
                     appImage: appContentList[indexPath.row].makeImage())
        return cell
    }
}
```

거의 비슷한데 파라미터로

indexPath를 넘겨주는 것이 다릅니다.

UICollectionView

UICollectionViewDataSource

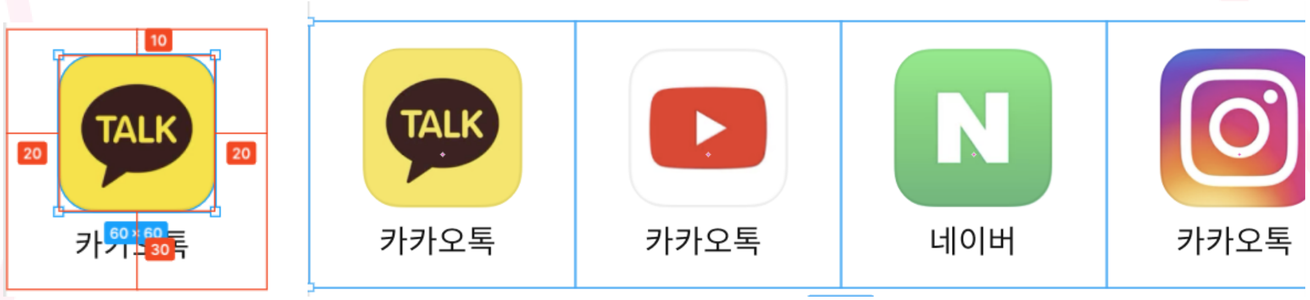



```
extension RankingVC: UICollectionViewDelegateFlowLayout{
    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, sizeForItemAt
        indexPath: IndexPath) -> CGSize {
        return CGSize(width: 100, height: 100)
    }

    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, insetForSectionAt
        section: Int) -> UIEdgeInsets {
        return UIEdgeInsets.zero
    }

    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout,
        minimumLineSpacingForSectionAt section: Int) -> CGFloat {
        return 0
    }

    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout,
        minimumInteritemSpacingForSectionAt section: Int) -> CGFloat {
        return 0
    }
}
```

item size (width: 100, height: 100)

UICollectionView

UICollectionViewDataSource

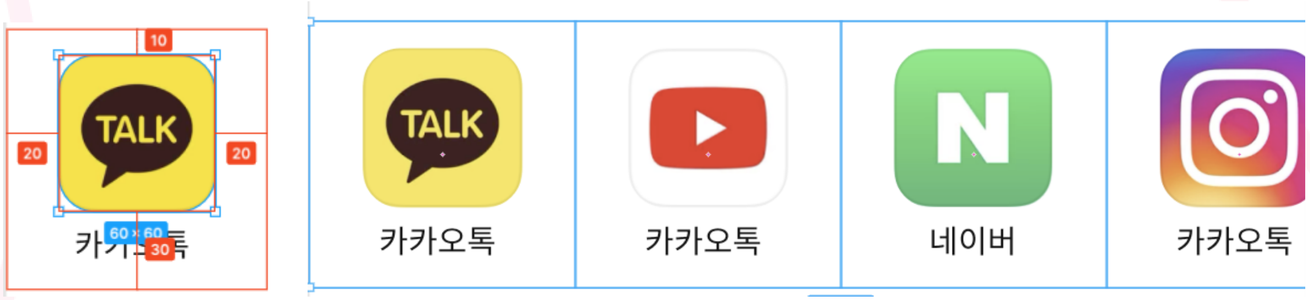



```
extension RankingVC: UICollectionViewDelegateFlowLayout{
    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, sizeForItemAt
        indexPath: IndexPath) -> CGSize {
        return CGSize(width: 100, height: 100)
    }

    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, insetForSectionAt
        section: Int) -> UIEdgeInsets {
        return UIEdgeInsets.zero
    }

    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout,
        minimumLineSpacingForSectionAt section: Int) -> CGFloat {
        return 0
    }

    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout,
        minimumInteritemSpacingForSectionAt section: Int) -> CGFloat {
        return 0
    }
}
```

content 외부에 감쌀 inset이 없기 때문에 (0,0,0,0)과 같은 의미인 UIEdgeInset.zero 넣어줍니다

UICollectionView

UICollectionViewDataSource

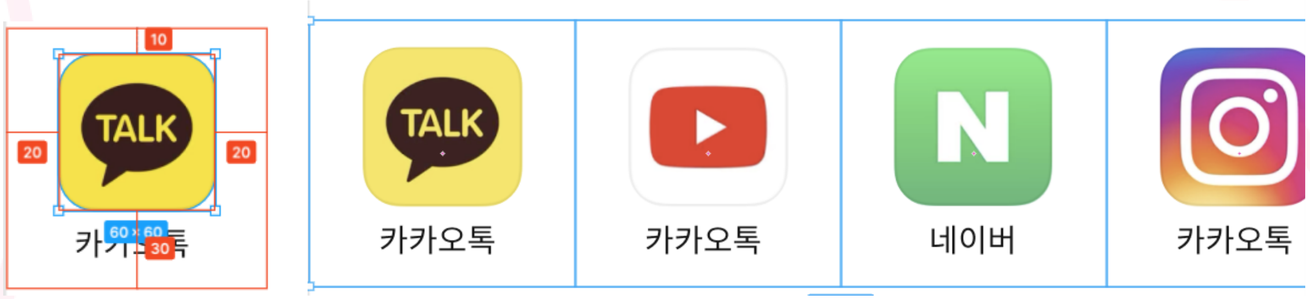



```
extension RankingVC: UICollectionViewDelegateFlowLayout{
    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, sizeForItemAt
        indexPath: IndexPath) -> CGSize {
        return CGSize(width: 100, height: 100)
    }

    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, insetForSectionAt
        section: Int) -> UIEdgeInsets {
        return UIEdgeInsets.zero
    }

    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout,
        minimumLineSpacingForSectionAt section: Int) -> CGFloat {
        return 0
    }

    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout,
        minimumInteritemSpacingForSectionAt section: Int) -> CGFloat {
        return 0
    }
}
```

Cell 간의 상,하 간격이 없기 때문에 0으로 지정

UICollectionView

UICollectionViewDataSource




```
extension RankingVC: UICollectionViewDelegateFlowLayout{
    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, sizeForItemAt
        indexPath: IndexPath) -> CGSize {
        return CGSize(width: 100, height: 100)
    }

    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, insetForSectionAt
        section: Int) -> UIEdgeInsets {
        return UIEdgeInsets.zero
    }

    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout,
        minimumLineSpacingForSectionAt section: Int) -> CGFloat {
        return 0
    }

    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout,
        minimumInteritemSpacingForSectionAt section: Int) -> CGFloat {
        return 0
    }
}
```

Cell 간의 좌, 우 간격이 없기 때문에 0으로 지정

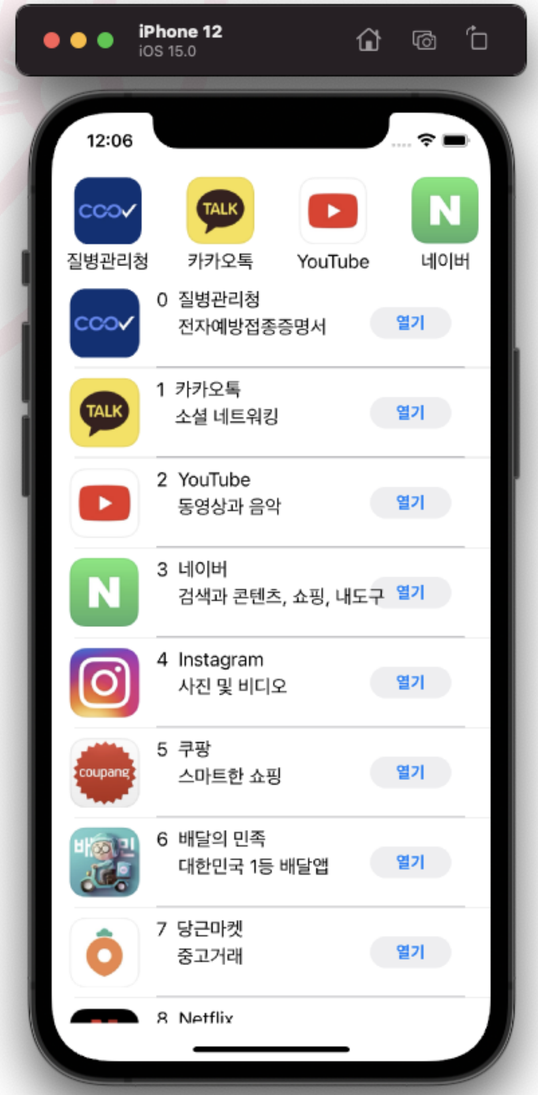
UICollectionView

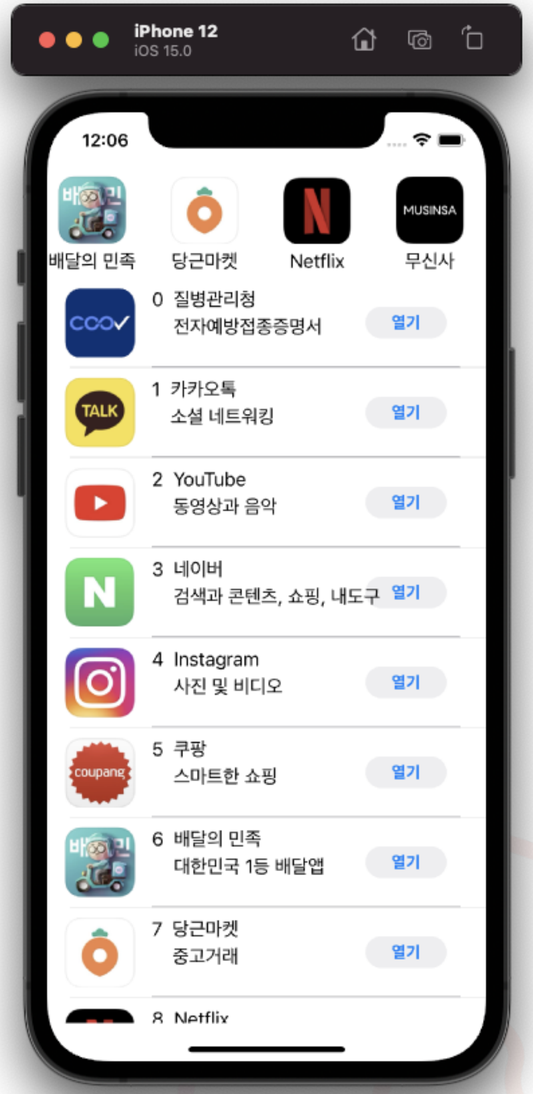
```
override func viewDidLoad() {
    super.viewDidLoad()
    initAppContentList()
    registerXib()
    rankTableView.dataSource = self
    rankTableView.delegate = self
    rankCollectionView.dataSource = self
    rankCollectionView.delegate = self
}
```

RankingVC를
collectionview의
dataSource와
delegate를 위임해주세요~!

UICollectionView

빌드해서 확인해보세요!





감사합니다

여러분 오늘 실습도 많았는데
정말 고생많으셨어요!!
아직 시험 남으신분들은 끝까지 파이팅하시고
뒷풀이팀은 즐겁게 놀구 위아요 한번 외쳐주세요~!

긴 시간동안 너무 고생많으셨습니다!
맛있는 저녁 드세요!

3차 세미나 피드백

더 좋은 iOS 파트를 위해 소중한 의견 꼭 부탁드리겠습니다 ❤️