

# iOS 멘토링 - 2주차

앱 디자인 해보기, UI Components 다루기

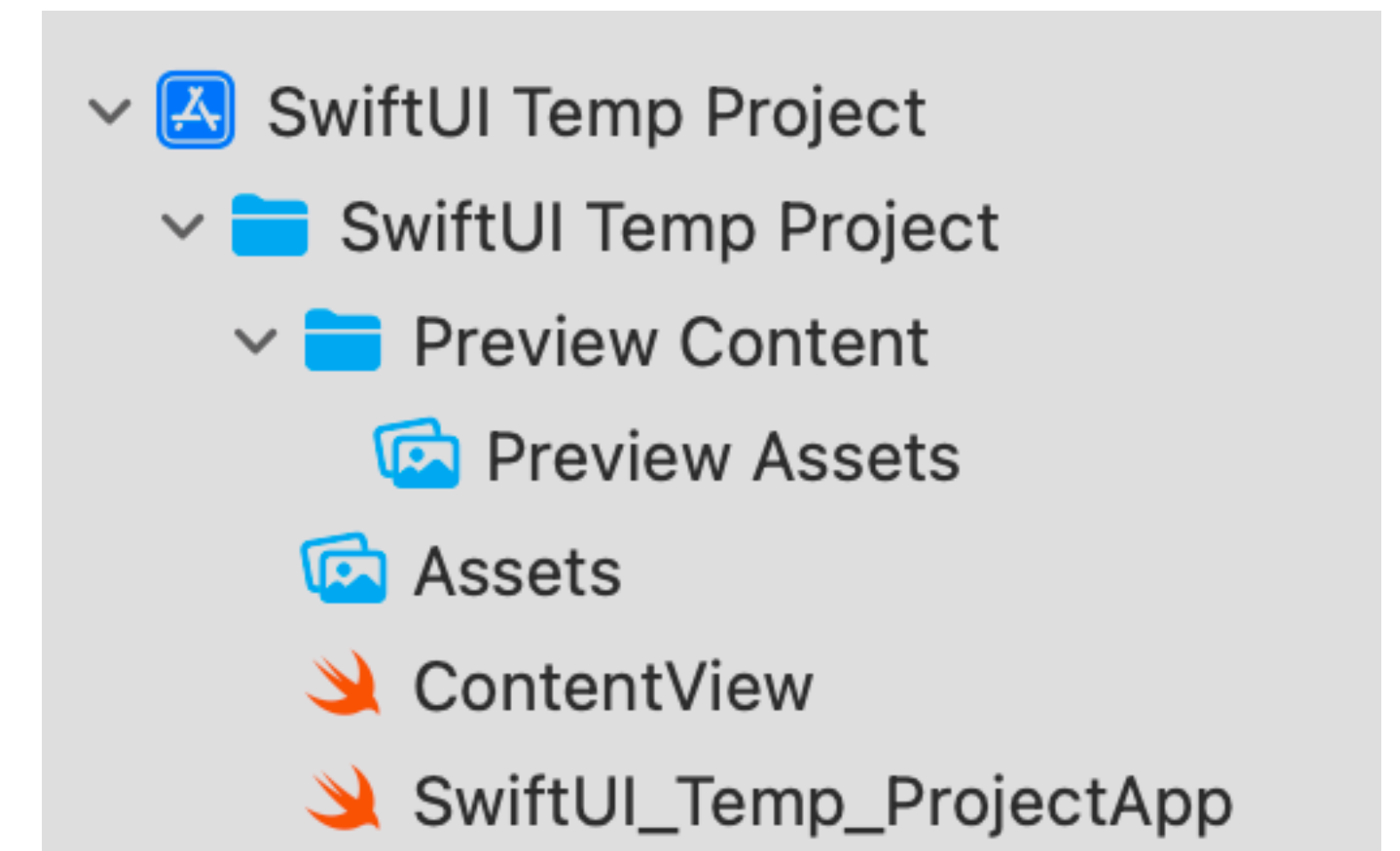
국립금오공대 컴퓨터공학전공 20학번 박효준  
2025-04-03 (목)

# 목차

- 프로젝트 구조
- SwiftUI의 UI Components 알아보기
- Color와 Image 리소스 설정 방법
- 프로토콜 개념
- View 프로토콜
- 실습

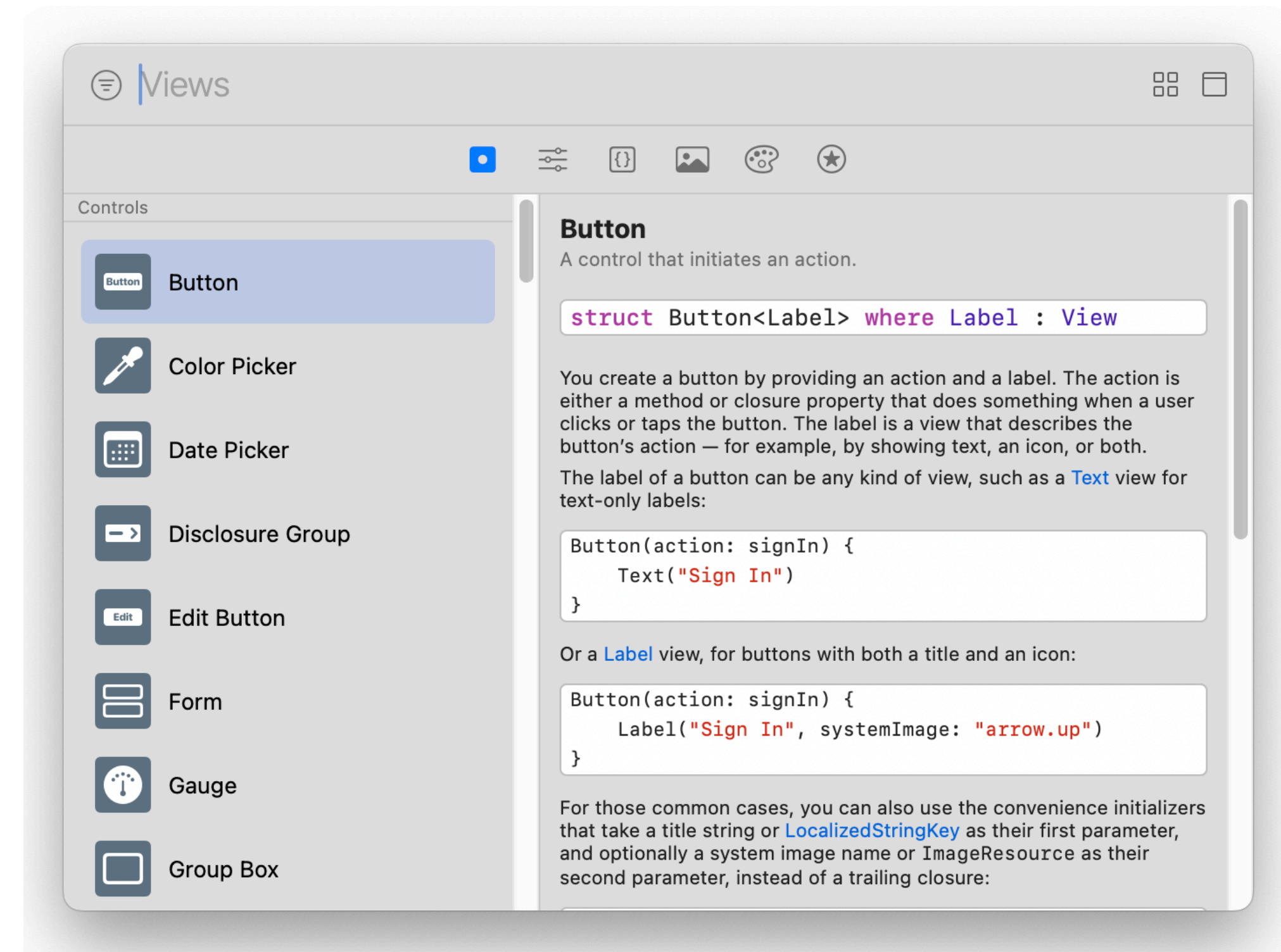
# 프로젝트 구조

- Xcode - "Create New Project" - iOS - App - Interface에서 SwiftUI 선택
- SwiftUI에는 주요 파일 두 가지가 있음
  - ContentView.swift
    - UI를 담아낼 하나의 화면, 혹은 그 자식 화면
  - 프로젝트명App.swift
    - C언어의 메인 함수라 생각하면 됨
    - View를 따르는 파일 중 어떤 화면을 처음 오게할 지 설정 가능
- 이외에도 Assets를 통해 이미지 설정과 폰트 설정 등이 가능



# 디자인을 위한 SwiftUI의 UI 요소들

- CMD + Shift + L을 누르면 "Show Library"가 뜸 (우측 상단 '+' 이미지)
  - Button, Text, ... 등등 모든 UI 요소가 있음
- option을 누르고 코드에 있는 UI 요소를 클릭해보자
  - 사용법부터 세세한 설정까지 팁을 줌
- command를 누르고 클릭하면 선언부를 볼 수 있음
  - 파라미터로 뭘 넣어야 할지,
  - 리턴은 뭘 해주는지 등등

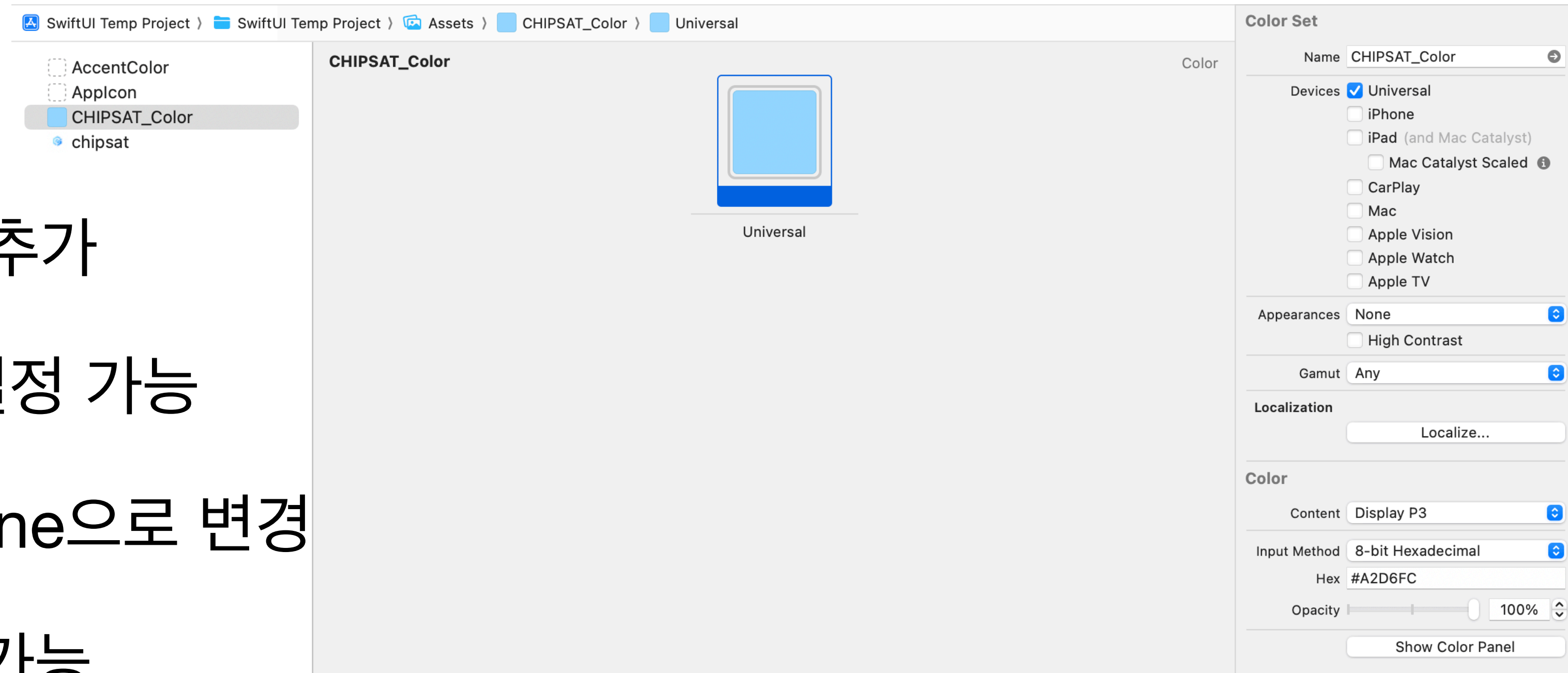


# Color와 Image 리소스 관리

- 다른 컴포넌트는 코드로만 사용해도 충분
- 색상
  - 기본으로 제공해주는 "빨강, 초록 .." 국룰 색은 제공해줌
  - 파스텔톤 느낌, 내가 커스텀한 색상을 사용하고 싶을 수 있음
- 이미지
  - Image(systemName:)을 사용하면 애플이 제공하는 심볼 이미지 사용 가능
  - 내가 보여주고 싶은 이미지가 있을 수 있음
  - 설명에 필요한 사진, 프로필 사진, 앱 아이콘 ...
- 내가 만든 색상, 이미지를 사용하고 싶다면 "Assets.xcasset" 파일 활용

# Assets - 색상

- 색상의 경우 +를 눌러 "Color Set" 추가
- 우측 인스펙터 창을 열어 디테일한 설정 가능
  - Appearances - Dark모드를 None으로 변경
  - 색상은 16진수, 스포이드로 설정 가능
- 등록된 색상은 우측 이미지와 같이 사용 가능



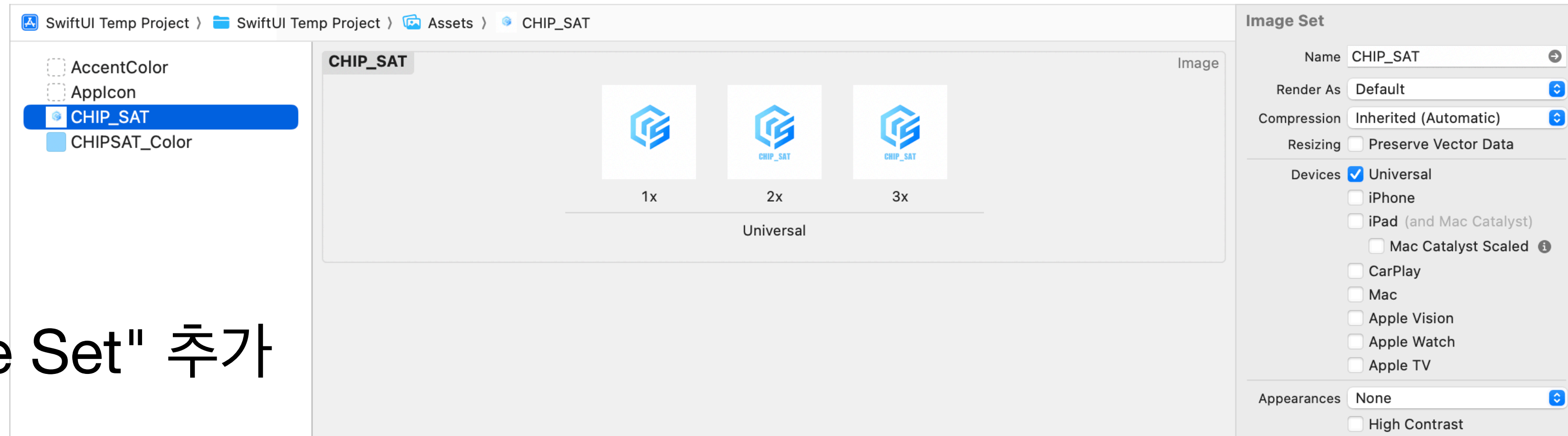
```
Text("CHIP_SAT")  
    .foregroundColor(.white, .black)
```

A list of color names with corresponding color swatches. The first item is 'CHIPSAT' with a blue swatch. The following three items are 'background', 'bar', and 'black', each with a blue swatch that has a white letter 'M' on it.

- CHIPSAT
- background
- bar
- black

# Assets - 이미지

- 이미지의 경우 +를 눌러 "Image Set" 추가
- 레스터 기반의 PNG, 혹은 벡터 기반 SVG 이미지를 넣는 두 가지가 있음
  - PNG의 경우, 1x, 2x, 3x 이미지를 각각 다 채워주는 게 좋음
  - 기기 해상도에 따라 최적화된 이미지가 자동으로 보여짐
  - SVG의 경우, 하나의 이미지만 넣어줘도 됨
- 이미지를 사용할 때는 문자열로 처리해야 함, e.g. Image("CHIP\_SAT")





```
1  import SwiftUI
2
3  struct ContentView: View {
4
5      var body: some View {
6          VStack {
7              Spacer()
8              Image("CHIP_SAT")
9              Spacer()
10         }
11     }
12 }
13
14 #Preview {
15     ContentView()
16 }
17
```



ContentView





# **끄적끄적 타임**

**Xcode -> iOS -> App -> Interface에서 SwiftUI  
UI 요소를 배치해서 프리뷰에 그려보자**

# View 프로토콜

# View 프로토콜

- struct SwiftUIView { }만 했더니 프리뷰에서 에러가 남

```
10 struct SwiftUIView {  
11  
12 }  
13  
14 #Preview {  
15     SwiftUIView()  
16 }  
17
```

✖ 'buildExpression' is unavailable: this expression does not conform to 'View'

! 'buildExpression' has been explicitly marked unavailable here  
(SwiftUICore.ViewBuilder)

Show

- SwiftUIView 라는 구조체에 View를 붙여줬음

```
10 struct SwiftUIView: View {  
11  
12 }  
13  
14 #Preview {  
15     SwiftUIView()  
16 }
```

● Type 'SwiftUIView' does not conform to protocol 'View'

🩹 Add stubs for conformance

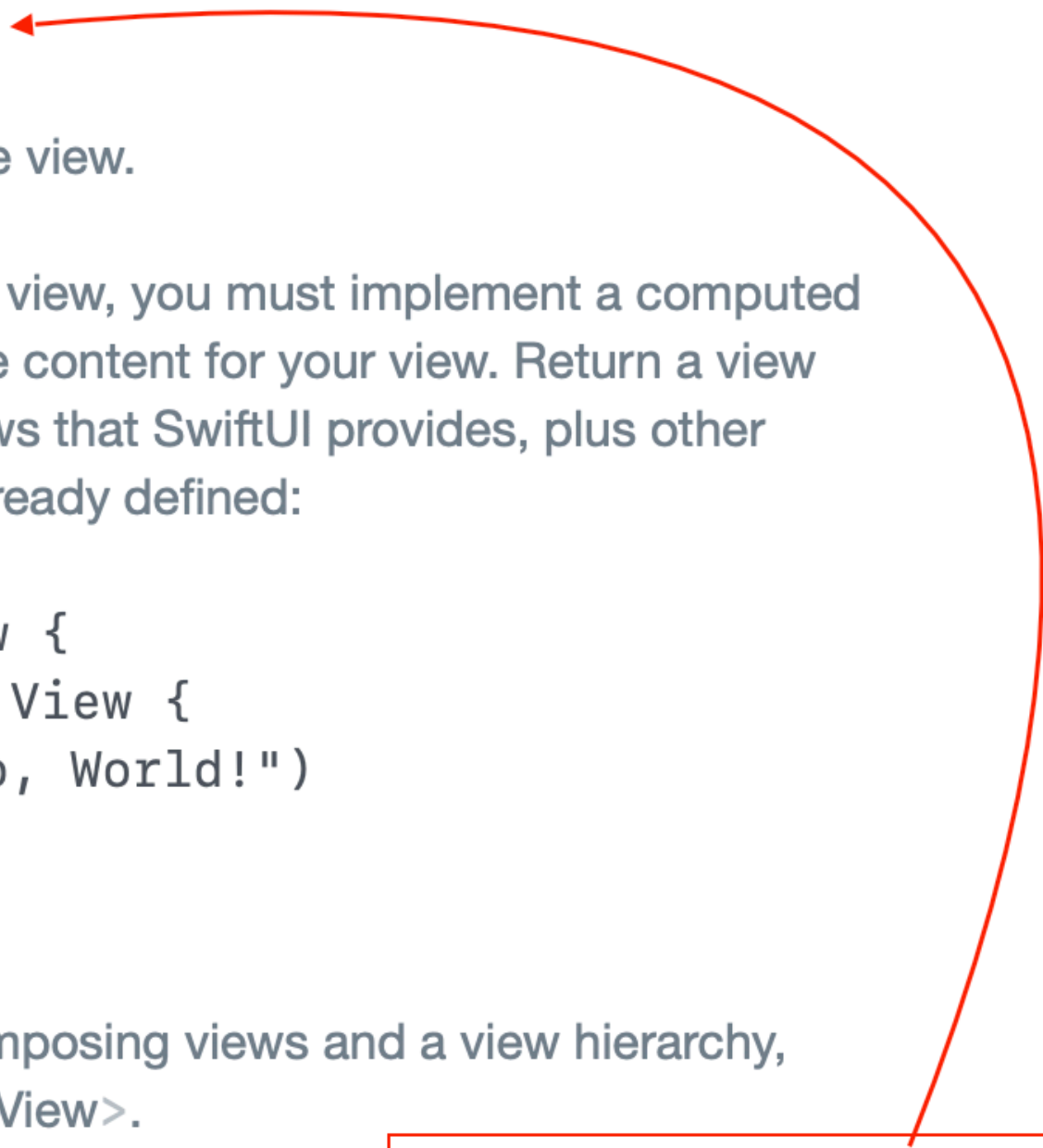
Fix

- 프로토콜을 따르지 않는다고 에러가 발생

# View 프로토콜

```
3 struct ContentView: View {
4
5     var body: some View {
6         VStack {
7             Text("가나다")
8                 .font(.title)
9                 .background(Color.red)
10                .foregroundColor(.white)
11             Text("라마바")
12             Text("사자")
13         }
14     }
15 }
16 }
```

```
30956 @MainActor @preconcurrency public protocol View {
30957
30958     /// The type of view representing the body of this view.
30959     ///
30960     /// When you create a custom view, Swift infers this type from your
30961     /// implementation of the required ``View/body-swift.property`` property.
30962     associatedtype Body : View
30963
30964     /// The content and behavior of the view.
30965     ///
30966     /// When you implement a custom view, you must implement a computed
30967     /// `body` property to provide the content for your view. Return a view
30968     /// that's composed of built-in views that SwiftUI provides, plus other
30969     /// composite views that you've already defined:
30970     ///
30971     ///     struct MyView: View {
30972     ///         var body: some View {
30973     ///             Text("Hello, World!")
30974     ///         }
30975     ///     }
30976     ///
30977     /// For more information about composing views and a view hierarchy,
30978     /// see <doc:Declaring-a-Custom-View>.
30979     @ViewBuilder @MainActor @preconcurrency var body: Self.Body { get }
30980 }
```



# View 프로토콜

- 앱의 화면은 하나가 아니며 수 없이 많을 수 있음
  - 그럼 ContentView 하나로 어떻게 작업을 할까 ?
  - -> 못함 새로운 ContentView와 같은 역할을 해줄 파일이 필요함
- 새로운 ContentView 뷰를 만들어야 함
  - 디렉토리 우클릭 -> "New Empty File" -> 파일명 설정
  - 혹은 "New File from Templates" -> SwiftUI File
- struct 새파일명: View { }를 입력하는데, 여기서 View가 뭘까?
- View 없이도 Text()와 같은 UI를 배치할 수 있을까 ?

# 프로토콜이 뭐죠 ?

- 동작보다 기능에 초점을 둠
  - 어떤 기능들이 있어야 하는지 정해놓은 규칙
  - 기능이 어떤 동작을 해야 하는지는 정해놓지 않음
  - 그렇기 때문에 Text(), Image(), Button() 우리 마음대로 넣을 수 있는 것
- 즉, UI를 담으려면 View를 따라야 하고, View를 따르면 body를 항상 구현해야 함
- 제네릭, Associated 등은 생략

# Swift와 POP

- Swift의 핵심 철학 중 하나는 POP(Protocol-Oriented Programming)
- $POP = OOP + FP + \text{Value Semantic}$
- Swift에는 값 타입과 참조 타입이 모두 존재
  - 값 타입은 C언어의 구조체라고 생각 == 빠름, 복사, 메모리 효율성 좋음
  - 참조 타입은 자바의 클래스라고 생각 == 느림, 참조(공유), OOP 특징 활용 가능
- 프로토콜 (= 인터페이스)
  - 자바, C# 등의 언어에서 인터페이스는 클래스만 따르는 것이 가능
  - Swift는 구조체나 열거형도 OOP 처럼 사용 가능함



# 정리

- 프로젝트 구조를 보았듯 앱 개발도 Main 함수가 있고, 거기서 시작을 함
- 색상, 이미지는 Assets.xcasset 파일을 이용해서 관리하면 됨
- UI 컴포넌트가 뭐 있는지 궁금하면 CMD + SHIFT + L
- 구조체가 UI를 그릴 환경을 갖추려면 View 프로토콜을 채택하고 body를 정의해야 함
- Text() 하나를 사용하더라도 설정할 수 있는 것들이 많음
  - 그림자, 곡선 굴곡, 폰트, 글자색, 배경색 .. 등등
  - 디자인을 보고 코드로 만들어내는 것이 중요
  - 우리가 할 일은 디자인 고안하고 그걸 보고 코드로 만들기

**Q & A**

**다음 주 멘토링 할말 ?**