**CS193p(Lecture3) 2023**

**MVVM**

**모델 분리하기**

1. 뷰 안에 @State로 분리하기(거의 안 씀)
2. “View Model”을 경유해서만 접근 가능(완전한 분리)
3. “View Model class”를 만들지만 모델은 여전히 직접적으로 접근 가능(부분 분리)

위 세 가지 방법 중 사실상 거의 매번 2번이 사용된다. 정말 간단한 로직이나 데이터만 포함하는 경우 1번, 3번은 1번과 2번 사이의 애매한 경우 사용할 수 있는데 확장성을 고려한다면 사실상 2번을 쓰는 게 맞다.

2번은 **Model-View-ViewModel이라고** 불리며 MVVM이 된다.

뷰 모델이 모델과 뷰를 연결하는 역할을 한다.

모델은 UI에 독립적으로 데이터와 로직의 조합이다. 모델은 본질과 같다. 모델은 뷰모델을 통해 read-only로 뷰에 전달된다. 뷰는 모델을 Stateless하게 보여준다. 뷰는 명령형이 아닌 선언형으로 작성된다. 또한 반응형이다.

**ObservableObject, @Published, objectWillChange.send()**

뷰모델은 뷰를 모델에 바인딩 한다. 또한 통역 역할도 수행해 sql을 로직에 반영하거나 반대를 수행한다. 또한 문지기 역할도 수행한다. 데이터 흐름은

모델 -> 뷰모델 -> 뷰로 향한다.

**@ObservedObject, @Binding, .onReceive, @EnvironmentObject, .environmentObject()**

모델이 변경 사항을 뷰모델에 알리면 뷰모델은 변경 사항을 배포한다. 뷰는 배포사항을 자동으로 감시하다가 변경이 발생하면 데이터를 pull해 리빌드한다.

위 어노테이션과 메소드 등은 추가적으로 학습하면 좋다. 모델에서 뷰 방향의 데이터 플로우를 알아봤다. 그렇다면 유저에서 모델로 흐르는 피드백 플로우는 어떻게 될까?

**우선 Intent에 대해 알아야 한다.**

인텐트는 뷰모델이 목적과 행위에 따라 뷰와 소통하는 방식을 이르는 말이다. iOS에서 뷰는 전형적으로 UIViews 나 UIViewControllers가 될 수 있다. 이때 뷰는 정보를 유저에게 진열하고 유저의 인풋을 포착한다. 뷰모델은 모델의 데이터를 뷰가 진열하기 쉬운 형식으로 처리하며 뷰가 포착한 유저의 인풋과 상호작용을 핸들링한다. 이런 맥락에서 인텐트는 뷰모델이 모델로 보내는 액션이나 명령을 의미한다. 정리하자면 인텐트는 뷰모델이 모델에게 요구하는 요청을 캡슐화해 모델이 특정 요청을 수행하게 한다.

인텐트를 알았으면 유저 반응의 흐름은 간단하다. 유저의 인풋이 인텐트로 변환되어 뷰모델이 모델에 변경을 가하면 모델이 수정되며 이전의 데이터 흐름이 발생하는 것이다.

**Struct vs Class**

이전에 미리 정리했지만 추가적으로 클래스로 할당된 객체는 힙 영역에 저장된다. 그리고 일반적으로는 구조체를 클래스보다 자주 사용한다.

Struct는 Copy on Write 전략을 취한다. 이 전략은 복사본을 만들 때 수정이 없다면 다른 객체가 같은 메모리를 공유하는 전략이다. 복사 시 바로 복사가 일어나지 않고 실제 값이 바뀔 때에 복사본을 만드는 방식이다. 불필요한 복사를 줄여 메모리 사용량을 줄일 수 있다는 장점이 있다. 중복이 많고 크기가 큰 자료구조에 적용 시 유용하다. 스탠다드 라이브러리의 Array, String, Dictionary에 이 전략이 사용된다.

Class는 참조 타입으로 ARC(Automatically reference counted) 전략을 취한다. 이는 가비지 콜렉터가 해당 객체가 참조되는 개수를 카운팅 해 카운팅이 0이 되면 메모리를 회수하는 전략이다. 컴파일 타임에 컴파일러가 이를 카운팅 해 카운팅이 0이 되는 라인에 적절히 release 코드를 삽입한다. 가비지 콜렉터는 런타임에 프로그램을 감시함으로 컴파일 타임에 메모리 해제 시점을 결정하는 ARC와 다르다. ARC는 이런 특징들로 런타임에 메모리 오버헤드를 방지하고 CPU를 보다 효율적으로 사용할 수 있다는 장점이 있다. 하지만 개발자의 실수에 의한 순환 참조가 발생할 수 있으니 사전에 점검/예방해야 한다.

또한 구조체는 함수형 프로그래밍 패러다임을 가지고 클래스는 보다 객체 지향 프로그래밍 패러다임에 가깝다. 함수형 프로그래밍에서 중요한 것은 Provability이다. 또 기본 생성자 사용 시 struct는 @AllArgsConstructor가 들어가지만 class에는 @NoArgsConstructor가 들어간다. 구조체는 가변성을 명시해야 하지만 클래스는 항상 가변적이라는 것도 차이이다.

UIKit는 클래스 베이스로 작성됐으며 뷰모델은 항상 클래스이다.

**제네릭스**

Element 타입의 제네릭스를 쓰면 “Don’t care” 타입이 된다.

**프로토콜**

프로토콜은 단순화한 구조체/클래스이다. 함수와 변수는 있지만 구현(메모리 점유)는 없다. 프로토콜은 행위를 묘사하기 위해 존재한다. 자바의 인터페이스와 유사하다. 프로토콜은 행위를 정의하고 프로토콜을 채택한 대상으로 하여금 구현을 강제한다.

프로토콜은 “constrains and gains”이다.

struct Array<Element> where Element: Equatable

과 같은 선언이 있을 때 이쿼터블을 채택한 무엇만이 어레이의 인자가 될 수 있다. 이것이 프로토콜 지향 프로그래밍의 핵심이다. 프로토콜은 extension을 통해 강력해진다. 이에 대해서는 나중에 배운다. 프로토콜 채택은 그들이 무엇을 할 수 있는지 보여주고 그러한 행위를 강제한다.

**함수**

함수는 타입이다.

**클로저**

인라인 함수가 클로저이다.