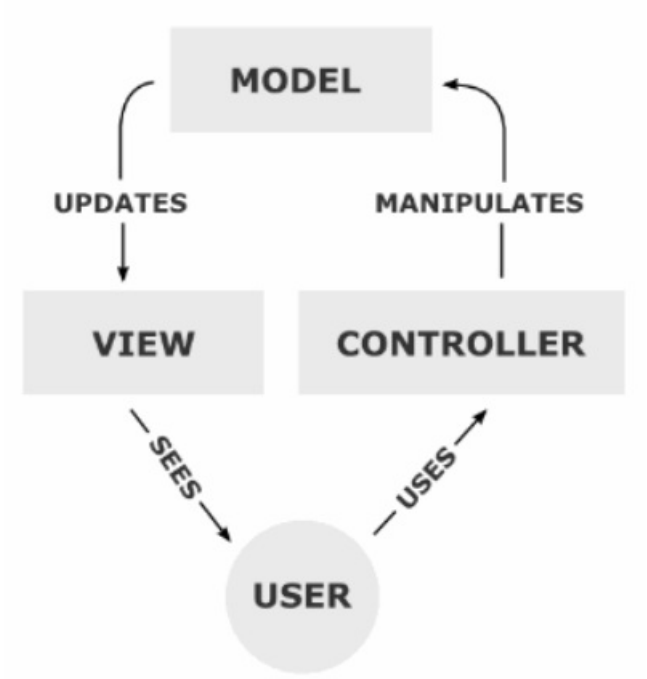
디자인 패턴이란 무엇인가?

소프트웨어의 개발 방법을 공식화 한 것입니다. 소수의 뛰어난 엔지니어가 해결한 문제를 다수의 엔지니어들이 처리 할 수 있도록 한 규칙이면서, 구현자들 간의 커뮤니케이션의 효율성을 높이는 기법입니다. 프로그래밍을 많이 접하다보면, 프로그램안에 클래스들이 갖는 구조에 일정한 '형태' 혹은 '패턴' 이 존재한다는 것을 알 수 있습니다. 이를 바탕으로 관계, 행동양식을 분류하고 각각에 대해 객체지향적인 설계를 구축하여 보다 효율적인 프로그래밍을 하도록 만든 것이 디자인 패턴입니다. 디자인 패턴은 프로그래머가 어플리케이션이나 시스템을 디자인할 때 공통된 문제들을 해결하는데에 쓰입니다.

1. MVC 패턴



모델

모델은 어떠한 동작을 수행하는 코드로, 모델의 상태에 변화가 있을 때 컨트롤러와 뷰에 이를 통보합니다
데이터 + 상태 + 비즈니스 로직으로, 앱의 두뇌 역할을 합니다. 뷰나 컨트롤러에 묶이지 않으므로 많은 곳에서 재사용할 수 있습니다.

뷰

사용자가 볼 결과물을 생성하기 위해 모델로부터 정보를 얻어 옵니다.
MVC에서 모델은 여러 개의 뷰(view)를 가질 수 있으며, 뷰는 모델에게 질의를 하여 모델로 부터 값을 가져와 사용자에게 보여줍니다. 그러므로 뷰는 모델의 표현입니다. UI를 그리고 사용자가 앱과 상호작용할 때 컨트롤러와 통신하는 책임을 맡습니다. MVC 구조에서 뷰는 하위 모델에 대한 지식이나 상태에 대한 이해가 없고, 사용자가 버튼을 클릭하거나 값을 입력하는 등의 행동을 할 때 무엇을 해야 하는지 모릅니다. 때문에 뷰가 모델에 종속되지 않아 보다 변화에 유연할 수 있게됩니다.

컨트롤러

컨트롤러는 모델에 명령을 보냄으로써 모델의 상태를 변경할 수 있습니다.
MVC의 뷰는 여러 개의 컨트롤러(Controller)를 가지고 있는데, 사용자는 컨트롤러를 모델의 mutator 함수를 호출하여 모델의 상태를 바꿉니다. 이때 모델의 상태가 바뀌면 모델은 등록된 뷰에 자신의 상태가 바뀌었다는 것을 알리고 뷰는 거기에 맞게 사용자에게 모델의 상태를 보여 줍니다. 그런 의미에서 컨트롤러는 애플리케이션에서 발생하는 일을 담당하는 마스터 컨트롤러 역할을 한다고 볼 수 있습니다. 뷰가 컨트롤러에게 사용자가 버튼을 눌렀다고 알리면, 컨트롤러는 그에 따라 어떻게 모델과 상호작용할지 결정합니다. 모델에서 데이터가 변화되는 것에 따라 컨트롤러는 뷰의 상태를 적절하게 업데이트하도록 결정할 수 있습니다. 안드로이드 앱에서는 컨트롤러가 주로 액티비티나 프래그먼트로 표현됩니다.

특징

뷰는 반드시 모델에게 질의하여 업데이트하는 부분을 구현해야 합니다. 모델은 addObserver라는 함수를 이용하여 뷰를 자신에게 등록시키며 모델은 자신에게 등록된 모든 뷰를 기억하고 있다가 자신의 상태가 바뀌게 되면 등록된 모든 뷰에 notify 함수를 호출하여 뷰를 update합니다. 모델은 뷰를 여러 개 가질 수 있으며,또한 뷰도 여러개의 모델에 등록될 수 있습니다.
MVC는 훌륭하게 모델과 뷰를 분리해줍니다. 모델이 어디에도 종속되지 않으며, 쉽게 테스트할 수 있습니다. 하지만 컨트롤러가 뷰에 단단히 결합되어 있어 뷰를 변경하면 컨트롤러로 돌아가서 변경해야 하는 단점이 있습니다. 그리고 시간이 지남에 따라 보다 많은 코드가 컨트롤러로 모이면서 비대해지고 문제가 발생하기 쉬워 유지보수가 어려워집니다.

2. MVP 패턴

MVP는 컨트롤러의 책임에 묶이지 않고도 뷰와 액티비티가 자연스럽게 결합하도록 합니다.

모델

데이터 + 상태 + 비즈니스 로직으로, 앱의 두뇌 역할을 합니다. 뷰나 컨트롤러에 묶이지 않으므로 많은 곳에서 재사용할 수 있습니다.

뷰

액티비티/프래그먼트가 이제 뷰의 일부로 간주됩니다. 따라서 이들이 서로에게 연관되는 자연스러운 현상을 극복할 필요가 없습니다. 액티비티가 뷰 인터페이스를 구현해서 프리젠터가 코드를 만들 인터페이스를 갖도록 하는 것이 좋습니다. 이렇게 하면 특정 뷰와 결합되지 않고 가상 뷰를 구현해서 간단한 유닛 테스트를 실행할 수 있습니다.

프레젠터

본질적으로는 MVC의 컨트롤러와 같지만, 뷰에 연결되는 것이 아니라 그냥 인터페이스라는 점이 다릅니다. 이에 따라 MVC가 가진 테스트 가능성 문제와 함께 모듈화/유연성 문제 역시 해결합니다.

특징

컨트롤러처럼 프리젠터에도 시간이 지남에 따라 추가 비즈니스 로직이 모이는 경향이 있습니다.

3. MVVM 패턴

안드로이드의 데이터 바인딩을 사용하는 MVVM은 테스트와 모듈화가 쉽고 뷰와 모델을 연결하기 위해 사용해야 하는 연결 코드를 줄일 수 있다는 장점이 있습니다.

모델

데이터 + 상태 + 비즈니스 로직으로, 앱의 두뇌 역할을 합니다. 뷰나 컨트롤러에 묶이지 않으므로 많은 곳에서 재사용할 수 있습니다.

뷰

뷰는 뷰모델에 의해 보여지는 옵저버블 변수와 액션에 유연하게 바인딩됩니다.

뷰모델

뷰모델은 모델을 래핑하고 뷰에 필요한 옵저버블 데이터를 준비합니다. 또한 뷰가 모델에 이벤트를 전달할 수 있도록 훅(hook)을 준비합니다. 그러면서도 뷰모델이 뷰에 종속되지는 않습니다.

특징

뷰가 변수와 표현식 모두에 바인딩될 수 있으므로 시간이 지남에 따라 관계없는 프리젠테이션 로직이 늘어나 XML에 코드를 추가하게 될 수 있습니다. 이를 방지하려면 뷰 바인딩 표현식에서 값을 계산하거나 파생하지 말고 항상 뷰모델에서 직접 값을 가져오는 것이 좋습니다.