1. MVP

\* Model, View, Presenter

- MVC와 다르게 UI(View)와 비즈니스 로직(Model)을 분리하고, 서로 간에 상호작용을 다른 객체(Prensenter)에 그 역할을 줌으로서 서로의 의존성을 최소화

- Model :

프로그램 내부적으로 쓰이는 데이터를 저장하고, 처리하는 역할을 함

View 또는 Presenter 등 다른 어떤 요소에도 의존적이지 않은 독립적인 영역

- View :

UI를 담당 - Android에서는 Activity, Fragment가 대표적인 예

Model에서 처리된 데이터를 Presenter를 통해 받아서 유저에게 보여줌

유저 액선 및 액티비티 라이플 사이클 상태 변경을 주시하며 Presenter에게 보내줌

Presenter 의존적

- Prensenter :

Model과 View 사이의 매개체

모델과 뷰를 매게체라는 점에서 Controller와 유사하지만 View에 직접 연결되는 대신 인터페이스를 통해 상호작용 함

인터페이스를 통해 상호작용 하므로 MVC가 가진 테스트 문제와 함께 모듈화/유연성 문제 해결 할 수 있음

뷰에게 표시할 내용만 전달하며 어떻게 보여줄지는 View가 담당

2. MVVM

\* Model, View, ViewModel

하나의 소프트웨어를 최대한 기능적으로 작은 단위로 나누어 테스트가 쉽고 큰 프로젝트도 상대적으로 관리하기 좋은 구조

모든 입력들은 View로 전달되며 ViewModel은 입력에 해당하는 Presentation Logic을 처리하며 View에 데이터를 전달한다. ViewModel은 View를 따로 참조하지 않기 때문에 독립적이며 ViewModel과 View는 1:n 의 관계이다.

3. MVC

\* Model, View, Control

입력은 모두 Control에서 발생하게 되며 관리하게 도는 구조

이벤트가 발생한 Control에 의해 각 모듈의 정의와 View의 사용 용도가 달라지게 됨

참고 블로그

[velog.io/@jojo\_devstory/%EC%95%88%EB%93%9C%EB%A1%9C%EC%9D%B4%EB%93%9C-%EC%95%84%ED%82%A4%ED%85%8D%EC%B2%98-%ED%8C%A8%ED%84%B4-MVP%EA%B0%80-%EB%AD%98%EA%B9%8C](https://velog.io/@jojo_devstory/%EC%95%88%EB%93%9C%EB%A1%9C%EC%9D%B4%EB%93%9C-%EC%95%84%ED%82%A4%ED%85%8D%EC%B2%98-%ED%8C%A8%ED%84%B4-MVP%EA%B0%80-%EB%AD%98%EA%B9%8C)

[velog.io/@jojo\_devstory/%EC%95%88%EB%93%9C%EB%A1%9C%EC%9D%B4%EB%93%9C-%EC%95%84%ED%82%A4%ED%85%8D%EC%B3%90-%ED%8C%A8%ED%84%B4-MVC%EA%B0%80-%EB%AD%98%EA%B9%8C](https://velog.io/@jojo_devstory/%EC%95%88%EB%93%9C%EB%A1%9C%EC%9D%B4%EB%93%9C-%EC%95%84%ED%82%A4%ED%85%8D%EC%B3%90-%ED%8C%A8%ED%84%B4-MVC%EA%B0%80-%EB%AD%98%EA%B9%8C)