Android MVVM Pattern

JEON YONGTAE

https://github.com/yongtaii/yongapps

1

Android MVC 패턴

model –view-control

MVC 패턴

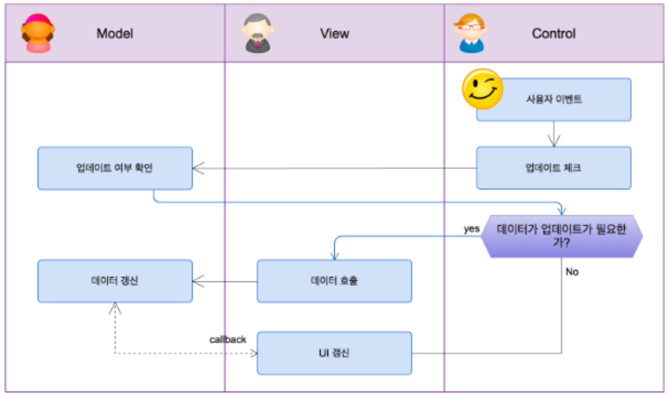
- 주로 웹에서 사용되고, 가장 널리 사용되는 구조 중 하나
- Model : 데이터를 가짐
- View : 사용자에게 보일 화면 표현
- Control : 사용자로부터 입력을 받고, 이를 모델에 의해 View를 정의

안드로이드에서 MVC 패턴

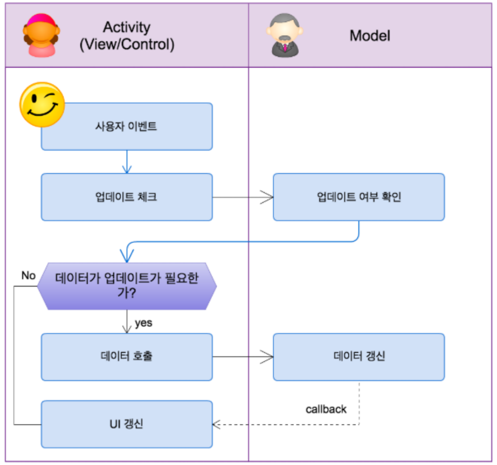
- Activity가 View와 연결되어 유저와 상호작용도 하고, Model과 연결되어 데이터도 처리
- Model : 데이터 저장 모델
- View : Activity
- Control : Activity

안드로이드에서 MVC 패턴

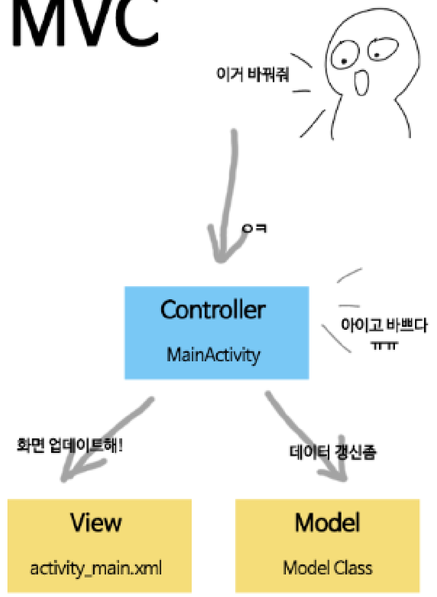
MVC in WEB



MVC in Android



MVC



한 화면에서 모든 데이터를 처리

한 눈에 코드 파악이 가능 하나 일정 범주 이상이 되면 어렵다

장점

- 개발 기간이 짧을 수 있다.
- 처음 보는 사람도 패턴구분 안하고 쉽게 파악이 가능하다

단점

- 코드 양이 증가한다.
- 스파게티 코드 가능성 : 코드 분리가 안되어 있어 빙빙 꼬여있는 스파게티 코드가 될 수 있다.
- 유지/보수의 어려움 : View와 Model 간의 결합도가 높아 테스트 코드 작성이 어렵다

2

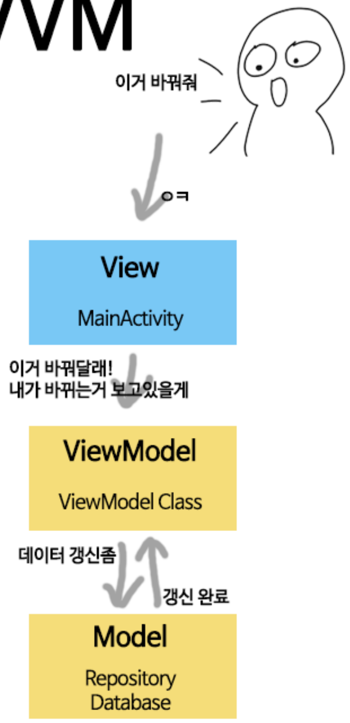
Android MVVM 패턴

View-ViewModel-Model

안드로이드에서 MVVM 패턴

- MVC, MVP and MVVM are among the most commonly used and widely accepted ones.
- MVVM pattern is comparatively new to Android world
- Newly launched Android architectural components from Google are compatible and ideal match for the MVVM pattern.

MVVM



동작순서

1. 사용자의 Action들은 View를 통해 들어오게 됩니다.

2. View에 Action이 들어오면, Command 패턴으로 View Model에 Action을 전달합니다.

3. View Model은 Model에게 데이터를 요청합니다.

4. Model은 View Model에게 요청받은 데이터를 응답합니다.

5. View Model은 응답 받은 데이터를 가공하여 저장합니다.

6. View는 View Model과 Data Binding하여 화면을 나타냅니다.

MVVM 패턴 – View

- The View is responsible for handling following UI constructs
- Menus, Permissions,Event listeners,Showing dialogs, Toasts, SnackbarsWorking with Android View and Widget, Starting activities, All functionality which is related to the Android Context.
- 사용자로부터 입력을 받는다
- Model의 존재를 모른다
- View가 ViewModel을 구독하는 중 ViewModel의 상태가 변경되면 UI를 갱신한다
- Activity/Fragment 등에서 View 역할이 수행된다

MVVM 패턴 – Model

- Contains a data provider and the code to fetch and update the data. The data can be retrieved from different sources, for example:
- REST API, any type of databases , Handles broadcast, Shared Preferences, Firebase.
- Network ,DB, SharedPreference 등 으로부터 필요한 데이터를 처리

MVVM 패턴 ViewModel

- ViewModel is a helper class that contains UI data and UI Logic for an Activity or Fragment.
- It uses observable data (LiveData or RxJava Observable) to notify the view about changes and get the data from Model (Database or network call) layer.
- The ViewModel has the following responsibilities:

  – Exposing data, Executing calls to the model, Executing methods in the view. The view model should only know about the application context. the application context can: Start a service, Bind to a service, Send a broadcast, Register a broadcast receiver, Load resource values
- ViewModel Class에서 역할을 수행한다

MVVM 패턴 – ViewModel

- View로 받은 데이터를 Model로 전달, 처리 결과를 화면에 표시하기 위해 데이터를 View에 맞게 변환
- Model에서 제공받은 데이터를 UI에서 필요한 정보로 가공한뒤 View가 가져갈 수 있도록 데이터 변경에 대한 이벤트를 보내주는 역할
- View 또는 액티비티 Context 에 대한 레퍼런스를 가져서는 안된다. View는 ViewModel 의 레퍼런스를 가지지만, ViewModel 은 View에 대한 정보가 전혀 없어야 한다.
- View의 존재를 알지 못한다

Benefits for using MVVM

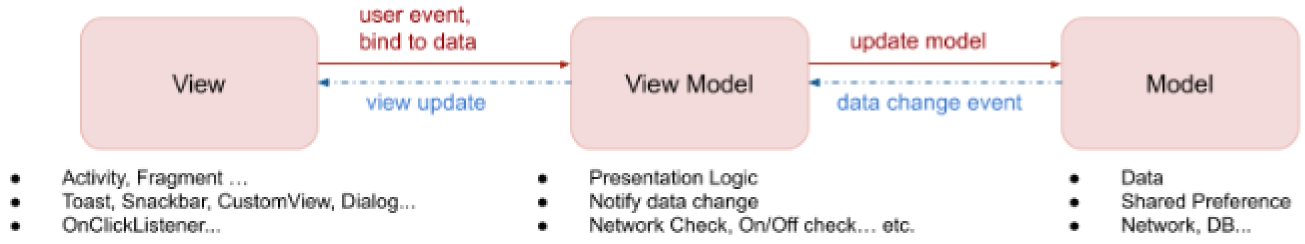
- ViewModels are persisted over **configuration and rotation changes,** so there’s no need to re-query an external source for data (such as a database or the network) when a rotation happens.
- When long-running operations finish, the observables in the ViewModel are updated. Hence event is emitted only for objects that are being currently observed. Hence if some one has presssed back button and the view is gone, **no null pointer exceptions** happen when trying to update the nonexistent View.
- ViewModels don’t reference views so there’s less risk of **memory leaks.**
- Writing test cases for **ViewModel** is easy. Because a **ViewModel** doesn’t have a reference to the Activity or Fragment.

Benefits for using MVVM

- You don’t need to worry about **UI data holder lifecycle.** ViewModel will be created automatically by a factory and you don’t need to handle creating it and destroying the ViewModel based on View lifecycle on your own.
- Be always updated — you’ll get **the same data** after phone rotation as it was before. You don’t need to pass manually data to the new activity or make a second call to the database. Data has been preserved by **LiveData** in **ViewModel**.
- A clear separation of concerns where View has no business logic or data providers thus providing easy code maintainability and debugging.
- Network calls are managed at a single point and hence network libraries can be seamlessly replaced without impacting other parts of the code.

In android **NullPointerException** is always a headache for any developer and one of the major reason for this exception is update in the UI after the UI is dismissed, which can be solved by using ViewModel.

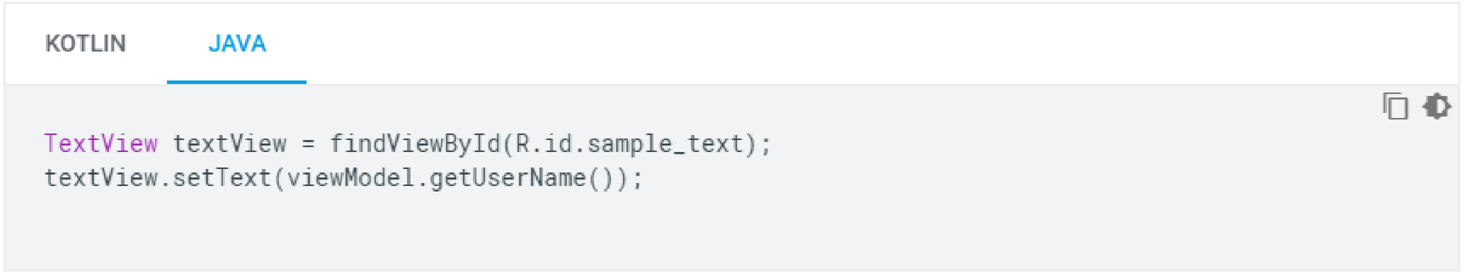
DataBinding



- The Data Binding Library is a support library that allows you to bind UI components in your layouts to data sources in your app using a declarative format rather than programmatically.

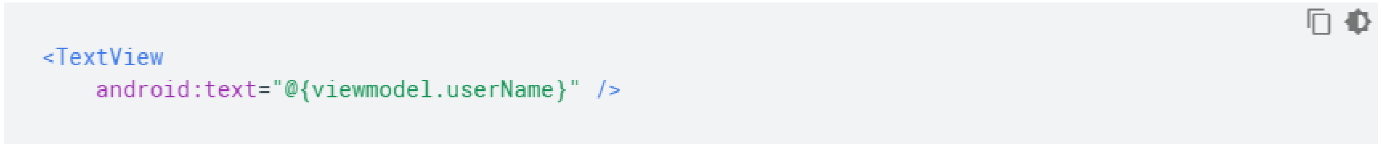
DataBinding

기존 방법



```
TextView textView = findViewById(R.id.sample_text);
textView.setText(viewModel.getUserName());
```

DataBinding



```
<TextView
    android:text="@{viewmodel.userName}" />
```

The following example shows how to use the Data Binding Library to assign text to the widget directly in the layout file. This removes the need to call any of the Java code shown above. Note the use of @{} syntax in the assignment expression:

Thanks!

Any *questions* ?

You can find me at

- jeonyt89@gmail.com