WaffleStudio Android Seminar - 1

이승민 (안드로이드 세미나장)

2021.09.04.(토) 11:30 ~

Review

Context?

```
val intent = Intent( packageContext: this, IntroduceActivity::class.java)
startActivity(intent)
```

Programmer가 Android OS와 소통하는 방식

(IntroduceActivity 열어줘!)

What we will learn in Seminar 1

- Gradle
- View Binding
- Logging (Timber)
- ViewModel (MVVM Architecture)
- LiveData (Reactive Programming)

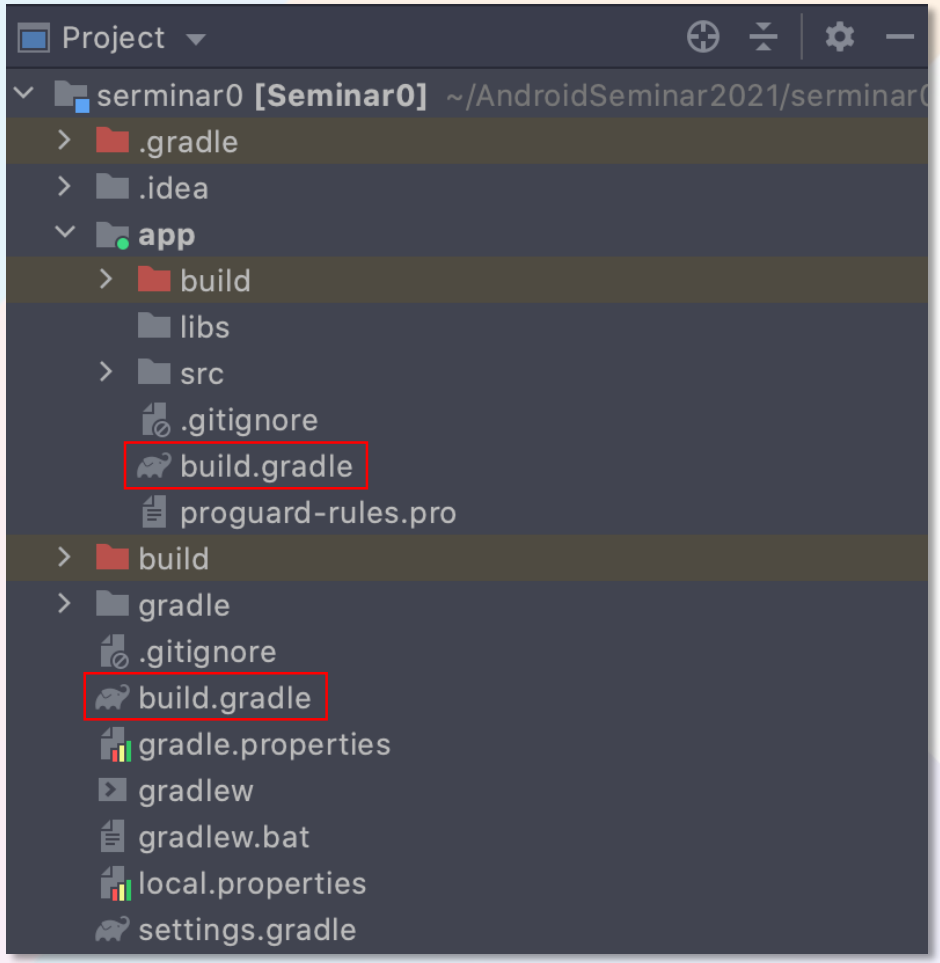
Gradle

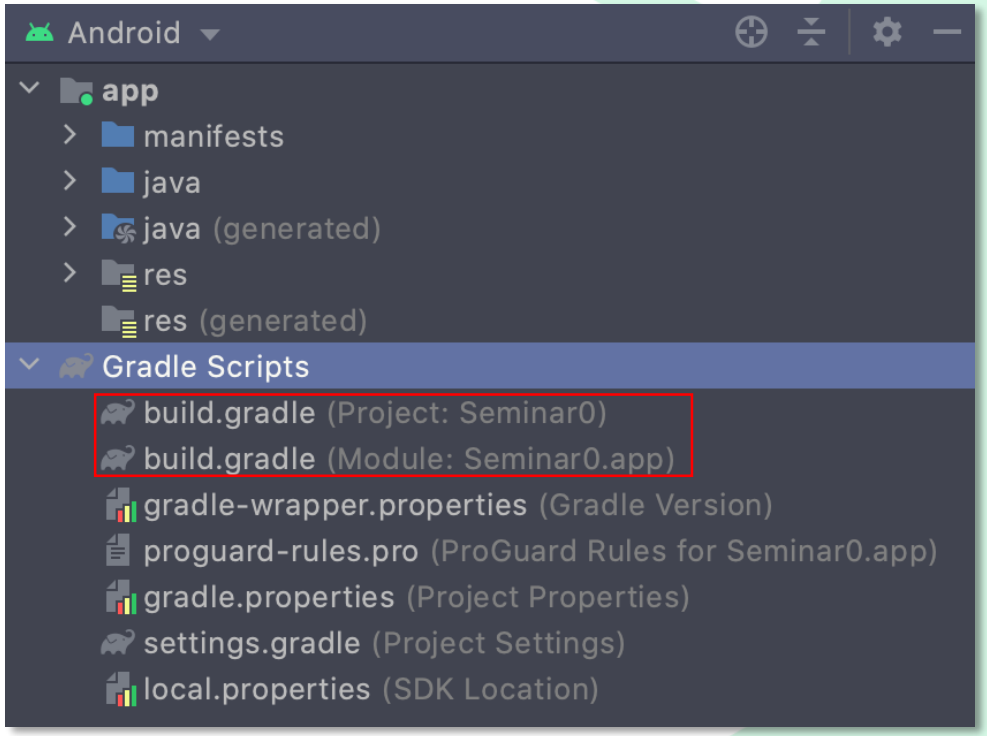
Android Studio uses Gradle, an advanced build toolkit, to automate and manage the build process, while allowing you to define flexible custom build configurations.

-> 대충 편하게 build할 수 있는 도구

- App build에 사용되는 기본적인 정보 (SDK version, App version 등)
- App 전반적으로 사용되는 const value들 (URL, access key 등)
- App 에서 사용되는 plugin 들의 dependency

Gradle

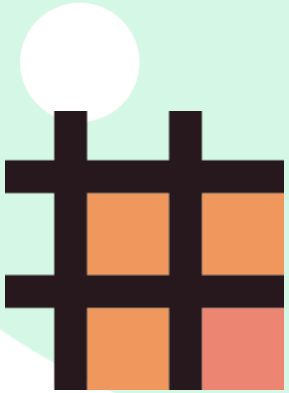

Project
serminar0 [Seminar0] ~/AndroidSeminar2021/serminar0
.gradle
.idea
app
build
libs
src
.gitignore
build.gradle
proguard-rules.pro
build
gradle
.gitignore
build.gradle
gradle.properties
gradlew
gradlew.bat
local.properties
settings.gradle


Android
app
manifests
java
java (generated)
res
res (generated)
Gradle Scripts
build.gradle (Project: Seminar0)
build.gradle (Module: Seminar0.app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for Seminar0.app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)

Gradle

```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
}

android {
    compileSdkVersion 30

    defaultConfig {
        applicationId "com.veldic.seminar0"
        minSdkVersion 26
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
}
```

```
dependencies {

    implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
    implementation 'androidx.core:core-ktx:1.6.0'
    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.4.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.0'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}
```



View Binding

Assignment 0을 했다면…

- 꼭 layout에 있는 필요한 요소들을 findViewById로 모두 선언해줘야 하나?
- 다른 layout과 id가 중복되는데 걱정 안해도 되나…?
- 뭔가 더 편한 방법이 없을까…?
- id로 찾는 방식이 느리지는 않을까?

-> View Binding!

View Binding

일단 따라해봅시다.

build.gradle (Module ~~)

```
android {
    buildFeatures {
        viewBinding = true
    }
}
```

MainActivity.kt

```
private lateinit var binding: ActivityMainBinding

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)
```

View Binding

binding.~~ 을 통해 layout의 View들에 접근 가능!

```
binding.textHello.text = "SOMETHING"

binding.buttonHello.setOnClickListener {  it: View!
    // SOMETHING
}
```

Logging

Logging은 디버깅에 매우 매우 중요하다.
원하는 부분에 원하는 log를 남겨야 효율적인 디버깅이 가능

Android에서 지원하는 로깅
Log.d(TAG, content) -> tagging 귀찮음, 어디서 부르는지 모름
-> Timber!

Logging

일단 따라해봅시다.

build.gradle (Module ~~)

dependencies {

```
// Timber
implementation 'com.jakewharton.timber:timber:5.0.1'
```

}

Logging

```
6   class Seminar1Application : Application() {
7
8       override fun onCreate() {
9           super.onCreate()
10
11          if (BuildConfig.DEBUG) {
12              Timber.plant(Timber.DebugTree())
13          }
14      }
15  }
```

AndroidMenfest.xml

```
<application
    android:name=".Seminar1Application"
```

Logging

이제 앱의 아무 곳에서나 Timber을 불러 Log를 남길 수 있습니다.

```
binding.buttonHello.setOnClickListener {  it: View!
    Timber.d( message: "Hello!")
}
```

In Logcat…

```
2021-09-03 22:27:05.566 3609-3609/com.veldic.seminar1 D/MainActivity: Hello!
```

ViewModel (MVVM Architecture)

문제 의식

-> Activity의 화면을 돌리면…?

-> Data가 날아간다.

-> Activity Lifecycle에 종속되지 않은 무언가가 필요!

-> 다양한 해결책

(onSavedInstance, ViewModel 등)

ViewModel (MVVM Architecture)

Seminar 0 remind…
KotlinPracticeActivity에서
editText의 정보 – Activity가 소유
editText를 가공하는 로직 – Acitivty가 소유
가공한 정보 – Activity가 소유

-> 화면 돌리면 다 날아감!

그래서 우리는 Activity Lifecycle에 종속되지 않은 ViewModel을 사용

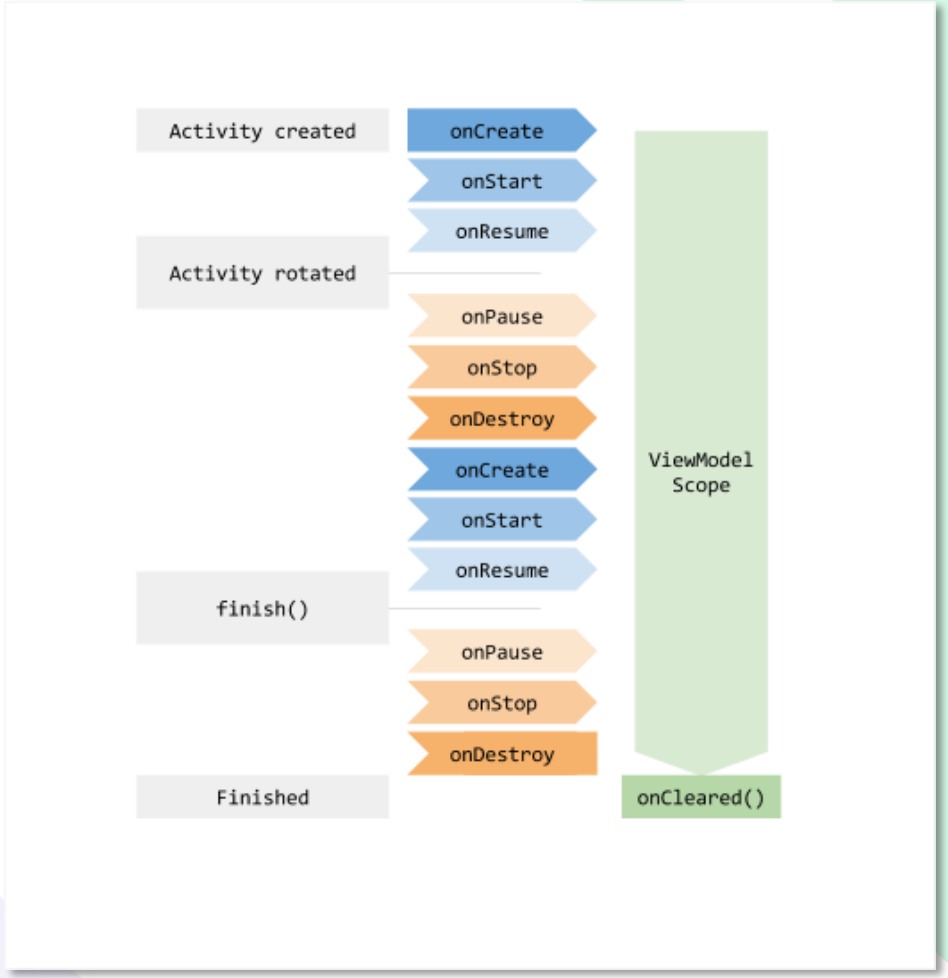
ViewModel (MVVM Architecture)

ViewModel의 Lifecycle을 살펴보면…

Activity가 회전할 때

Destroy되고 다시 Create되더라도

ViewModel은 사라지지 않는 것을 볼 수 있음!



ViewModel (MVVM Architecture)

이외에도…

ViewModel을 사용했을 때
– UI를 담당하는 View (우리에겐 Activity)
- 비즈니스 로직 및 data 관리를 담당하는 ViewModel
을 나눌 수 있음!

MVVM
Model – View – View Model 구조에서 View – View Model 에 해당
* 이런게 있다 정도만 짚고 넘어가기

ViewModel (MVVM Architecture)

구현해보기

build.gradle 의 dependencies

```
implementation 'androidx.activity:activity-ktx:1.3.1'
```

ViewModel 파일 만들기

```
7    class MainViewModel : ViewModel() {
8
9    }
```

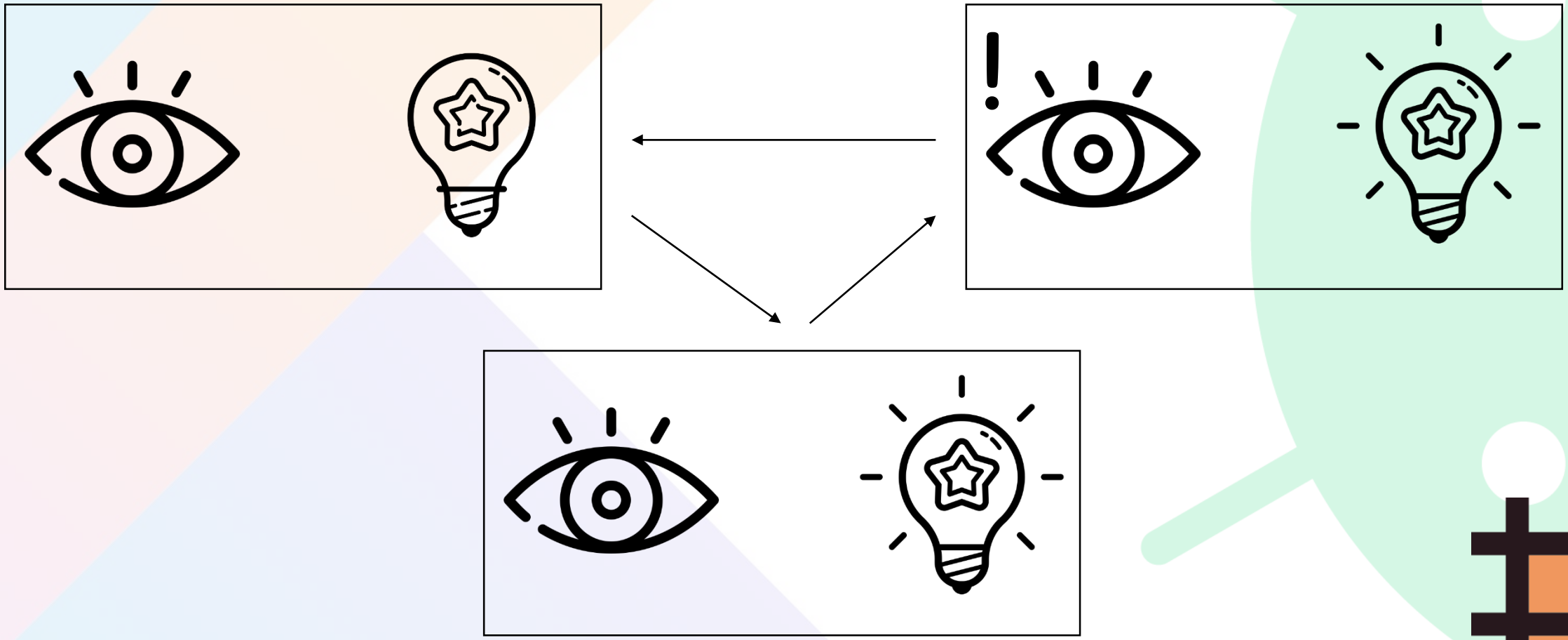
MainActivity에서 가져오기

```
private val viewModel: MainViewModel by viewModels()
```

LiveData

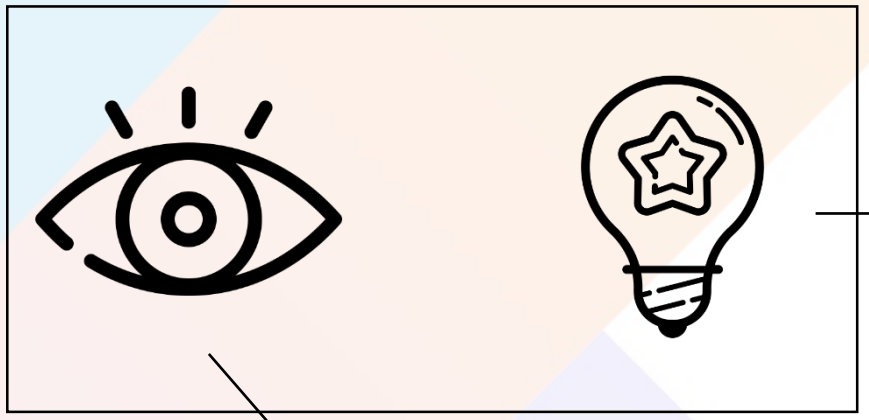
Reactive Programming

피관찰자의 변화를 감지



LiveData

Reactive Programming



LiveData

Observer

LiveData

```
private val _count = MutableLiveData( value: 0)
val count: LiveData<Int> = _count

fun addCount() {
    _count.value = _count.value?.plus( other: 1)
}
```

```
viewModel.count.observe( owner: this, {  it: Int!
    Timber.d( message: "count value $it")
})
```

Observer

Summary

Gradle

- Android Build 를 쉽게 하기 위한 도구
- 우리는 이를 App에 전반적으로 사용되는 정보들을 조작 하기 위해 사용함

View Binding

- findViewById 의 대안으로 사용하는 방식

Logging

- Timber을 사용
- 안드로이드에서 기본적으로 제공하는 Logging보다 좋음

Summary

ViewModel

- Activity의 Lifecycle을 따르지 않음
- Activity의 정보를 안정적으로 저장
- UI와 비즈니스 로직을 분리

LiveData

- Reactive Programming
- 관찰당하는 변수의 변화를 감지하여 로직 실행

QA

감사합니다

Thanks to. 기존 세미나 자료를 참고할 수 있도록 해준 김상민(@sangggggg)