Android App Architecture with Jetpack

Mobile Software

2019 Fall

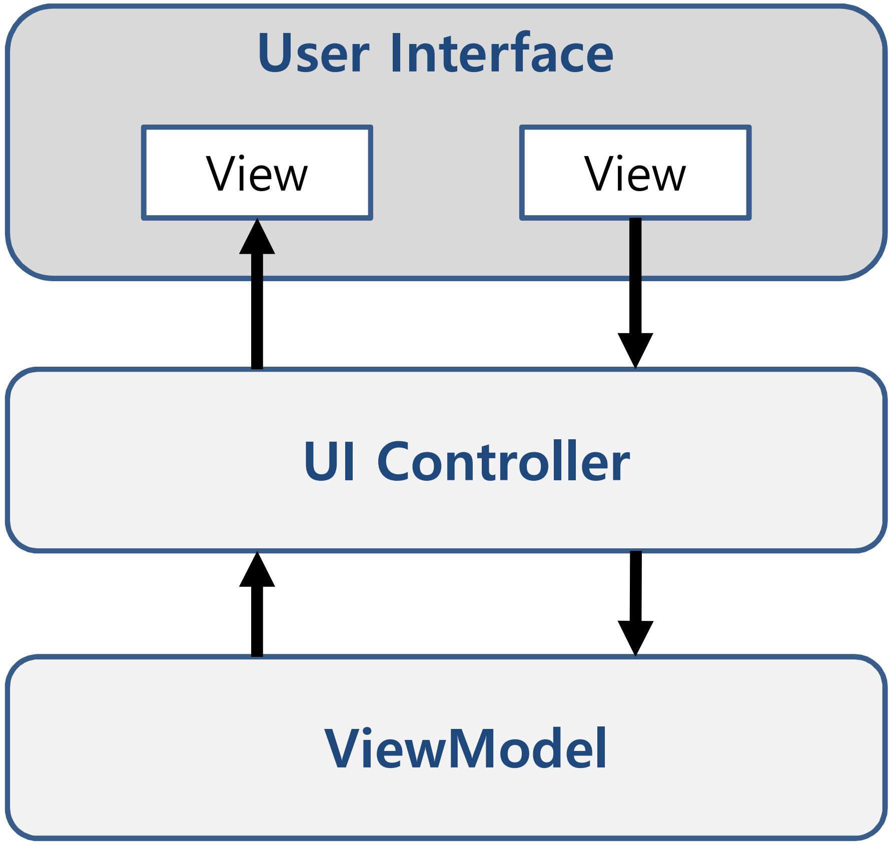
Android Architecture

- Android Jetpack
  - 2018년 정식 출시(개념은 2017년에 도입)
  - 구성
    - Android Studio + Android Architecture Components
    - Android support library
    - a set of **guidelines** how an Android App should be structured

  - **Android Architecture Components**
    - **ViewModel, LiveData, LifeCycle**
    - **Data Binding and use of Repositories**

- 어떻게 바뀌었을까?
  - Old architecture
    - **Multiple activities app.** ➔ one for each screen within the app
  - Modern architecture
    - **Single activity app**. where different screens are loaded as content within the same activity

The ViewModel Component



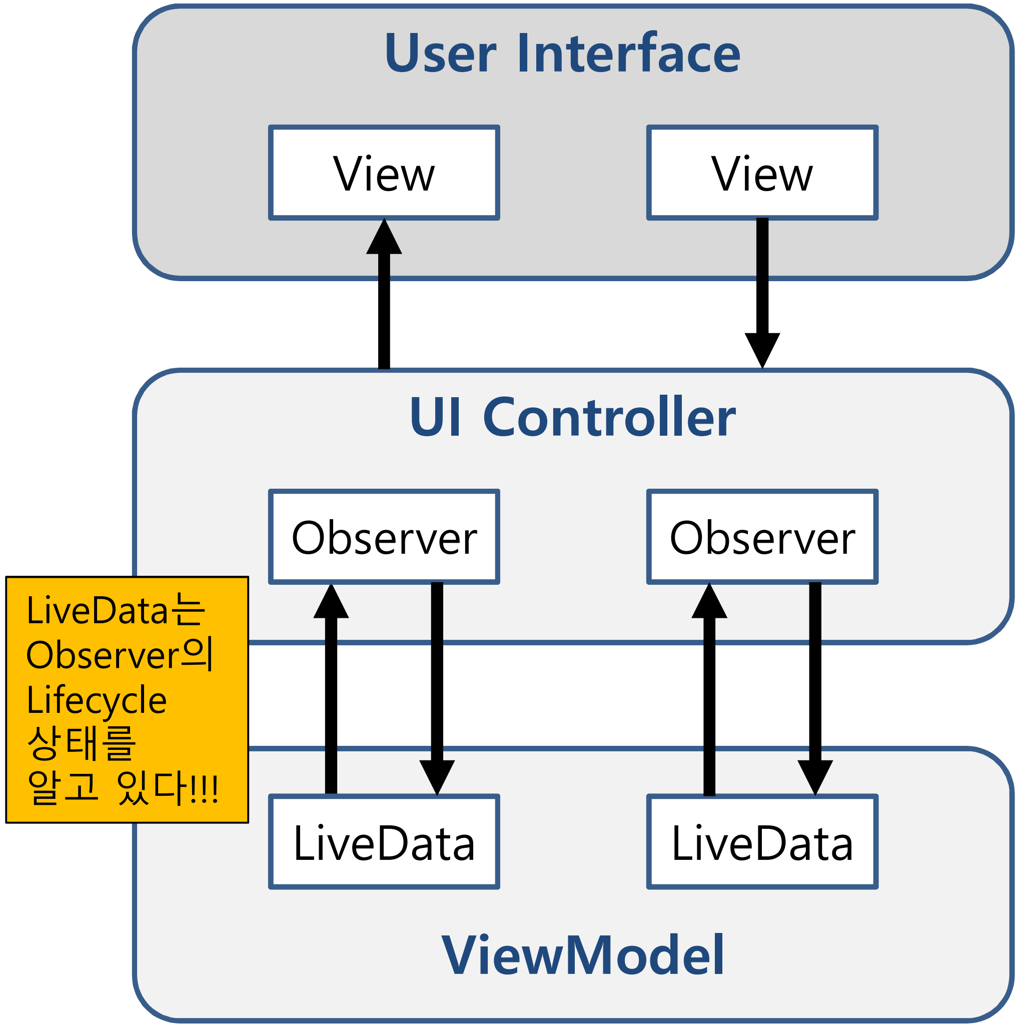
UI controller는 ViewModel이 관리하는 데이터가 변경된 것을 확인해 화면 출력 내용을 update해야 한다

responsible for displaying and managing user interface and interacting with OS.

Responsible for UI related data model and logic of an app

Activity의 상태가 몇 번 바뀌더라도 상관없이 ViewModel은 메모리에 그대로 남아있어 data consistency를 유지할 수 있다.

The LiveData Component



How can the UI controller ensure that the **latest data is displayed** in the user interface?

⬇

The UI controller **receives a notification** when a specific data item within a ViewModel changes.

⬇

LiveData component

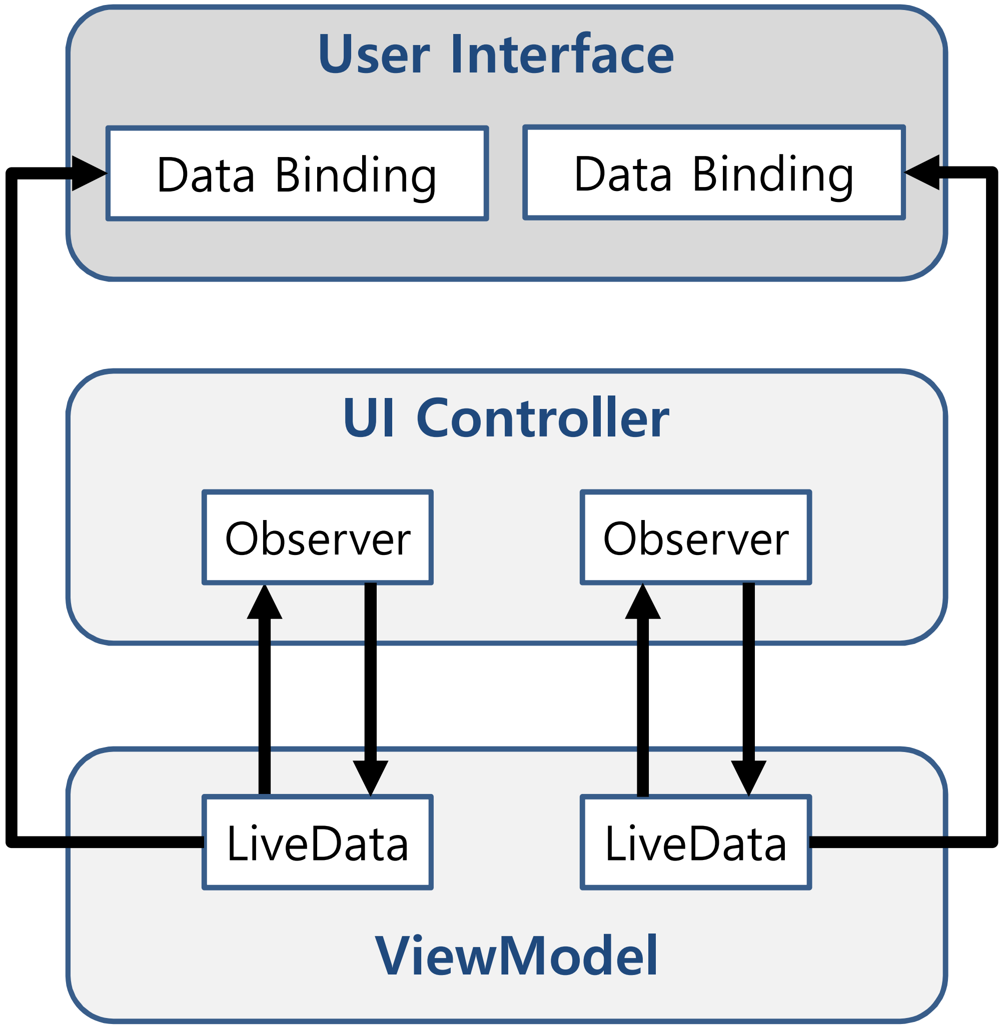
⬇

a data holder that allows a value to become *observable*.

⬇

An *observable object* has the ability to **notify other objects** when changes to its data occur.

LiveData and Data Binding

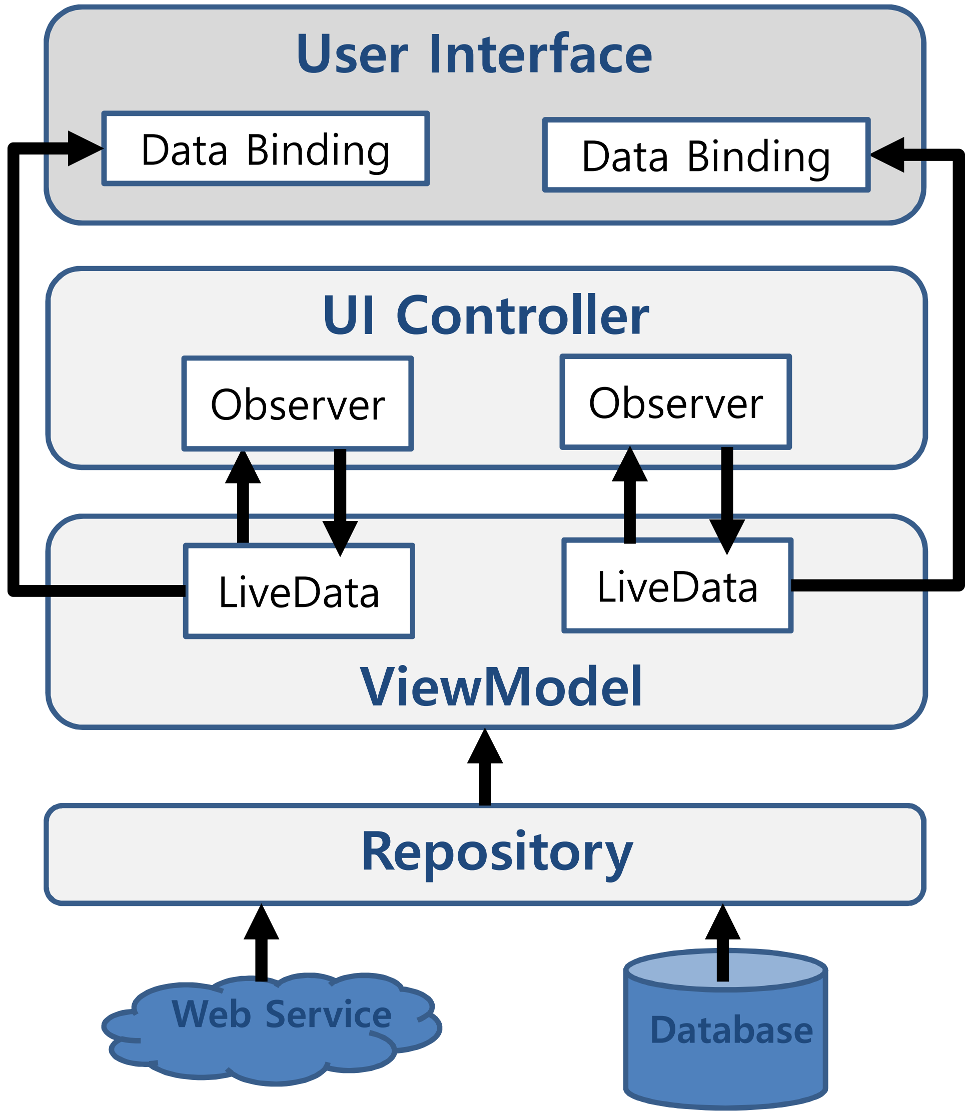


DataBinding Library allows data in a ViewModel **to be mapped directly** to specific views within the XML layout file.

Android Lifecycles

- **Lifecycle-aware**
  - Activity의 lifecycle state가 **pause** 상태
    - LiveData는 데이터를 Observer에게 전송하는 것을 멈춤.
  - Activity의 lifecycle state가 **start**(또는 **restart)** 상태
    - LiveData는 데이터를 Observer에게 전송.
  - Activity의 lifecycle state가 **destroy** 상태
    - LiveData는 Observer를 제거하고 할당된 resource를 해지.
  - Objects that are able to detect and react to lifecycle state changes in other objects are said to be *lifecycle-aware*.
- *Lifecycle observers* can be used so that an object receives notification **when the lifecycle state of another object changes**
  - This is the technique used by **the ViewModel** to identify when **an observer has restarted or destroyed**.

Repository Modules



ViewModel obtains data from one or more external sources.

A repository is **not an Android Architecture**, but rather a Java class that is responsible for *interfacing with the various data sources.*

실습 1,2,3: 환율 변환

- **실습 1**: 3-1. 첫 번째 애플리케이션에서
  - Old architecture로 만들었던 환율 변환 app을
  - Jetpack ViewModel을 사용하여
    - Modern architecture로 다시 만들어보자!

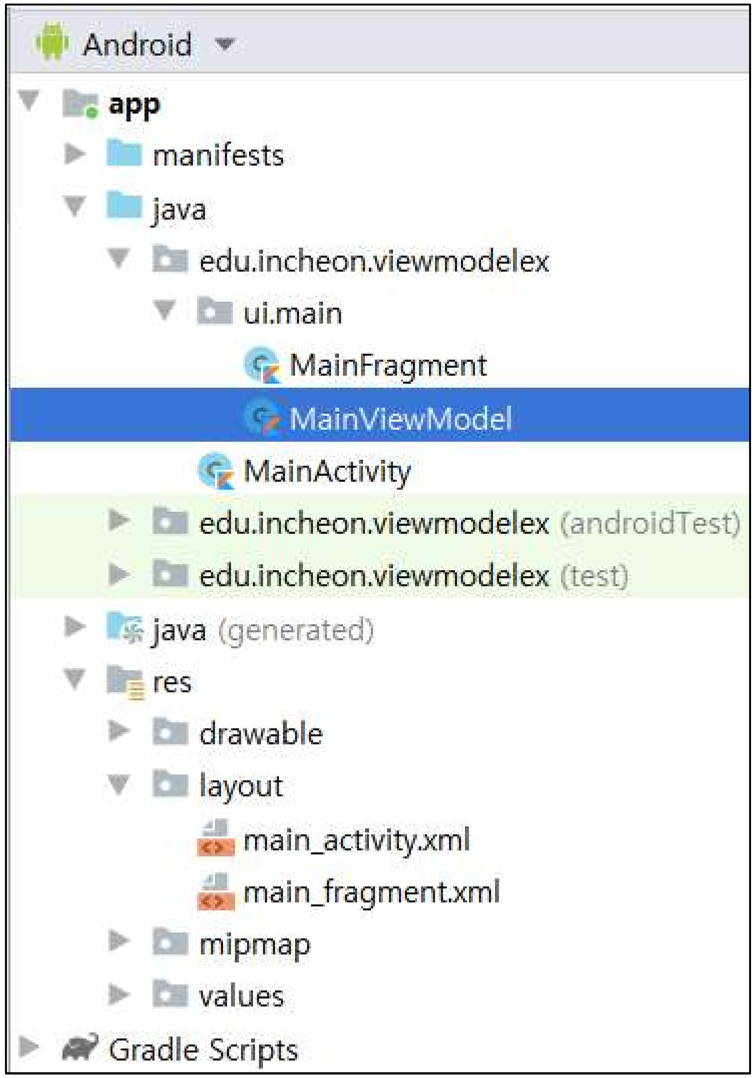
- **실습 2**: 이렇게 만든 app.을
  - LiveData를 사용하는 모델로 수정해보자

- **실습 3**: 이제 마지막으로
  - Data Binding을 적용한 모델로 수정하자!

실습 1: Jetpack ViewModel

- Choose your project
  - Add No Activity
- Configure your project
  - Name : **ViewModelEx**
  - Minimum API level
    - API 26: Android 8.0 (Oreo)

- app > java > 패키지
  - New > Activity >
  - Fragment + ViewModel
    - Enable the **Launcher Activity** option

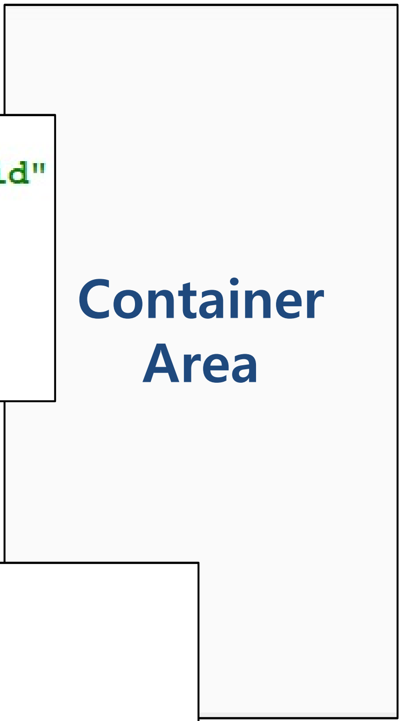


```
Android ▾
▼ app
  ▶ manifests
  ▼ java
    ▼ edu.incheon.viewmodelex
      ▼ ui.main
          MainFragment
          MainViewModel
        MainActivity
    ▶ edu.incheon.viewmodelex (androidTest)
    ▶ edu.incheon.viewmodelex (test)
  ▶ java (generated)
  ▼ res
    ▶ drawable
    ▼ layout
        main_activity.xml
        main_fragment.xml
    ▶ mipmap
    ▶ values
▶ Gradle Scripts
```

실습 1: The Main Activity

res/layout/activity_main.xml

```
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" />
```



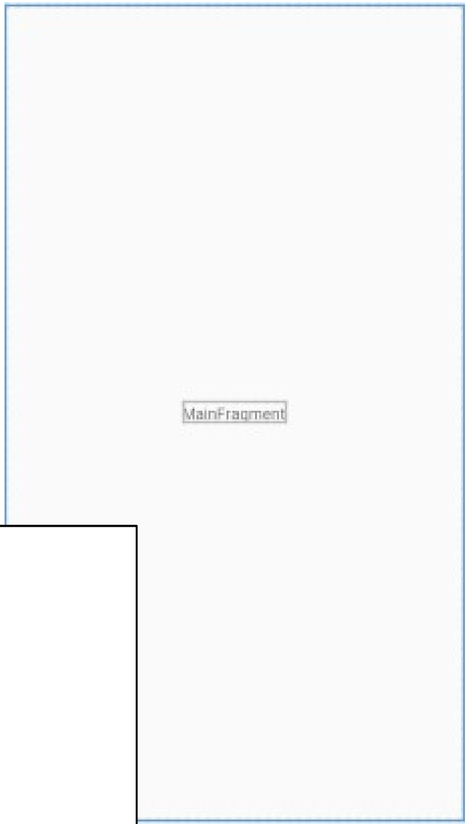
MainActivity.kt

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main_activity)
        if (savedInstanceState == null) {
            supportFragmentManager.beginTransaction()
                .replace(R.id.container, MainFragment.newInstance())
                .commitNow()
        }
    }

}
```

실습 1: The Content Fragment

res/layout/main_fragment.xml

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ui.main.MainFragment">

    <TextView...>

</androidx.constraintlayout.widget.ConstraintLayout>
```

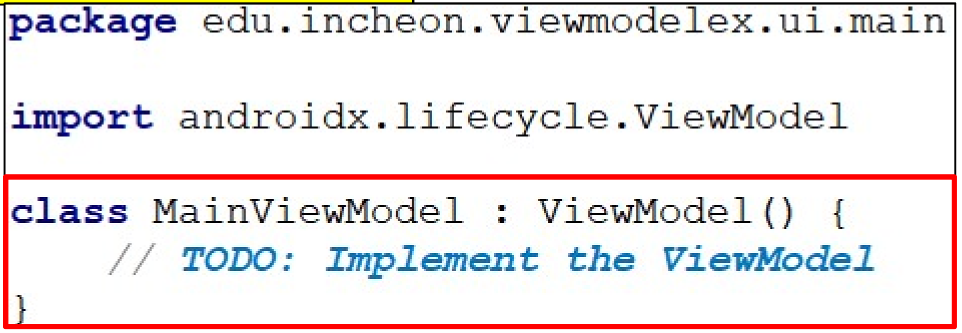


MainFragment.kt

```
class MainFragment : Fragment() {

    companion object {...}

    private lateinit var viewModel: MainViewModel

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {...}

    override fun onActivityCreated(savedInstanceState: Bundle?) {...}

}
```

실습 1: The ViewModel

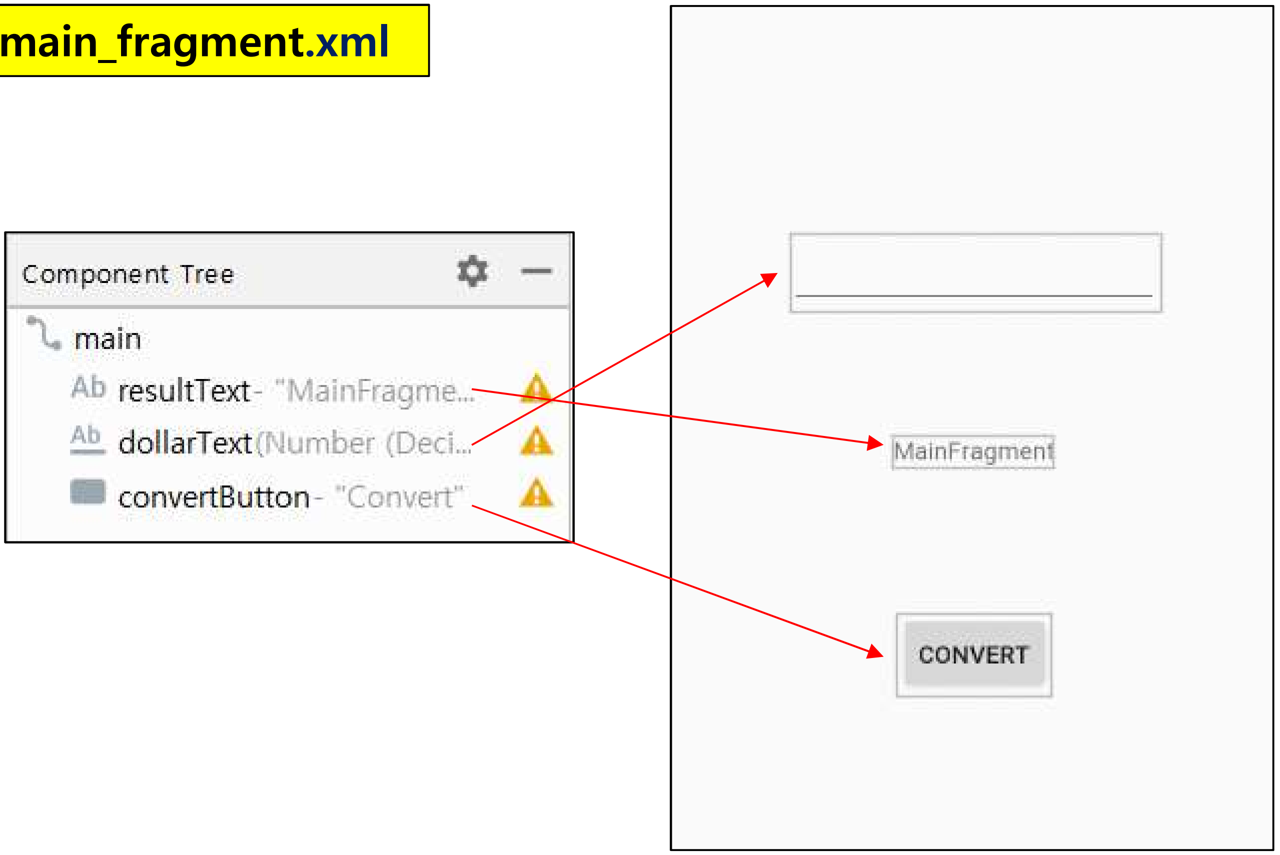
MainViewModel.kt



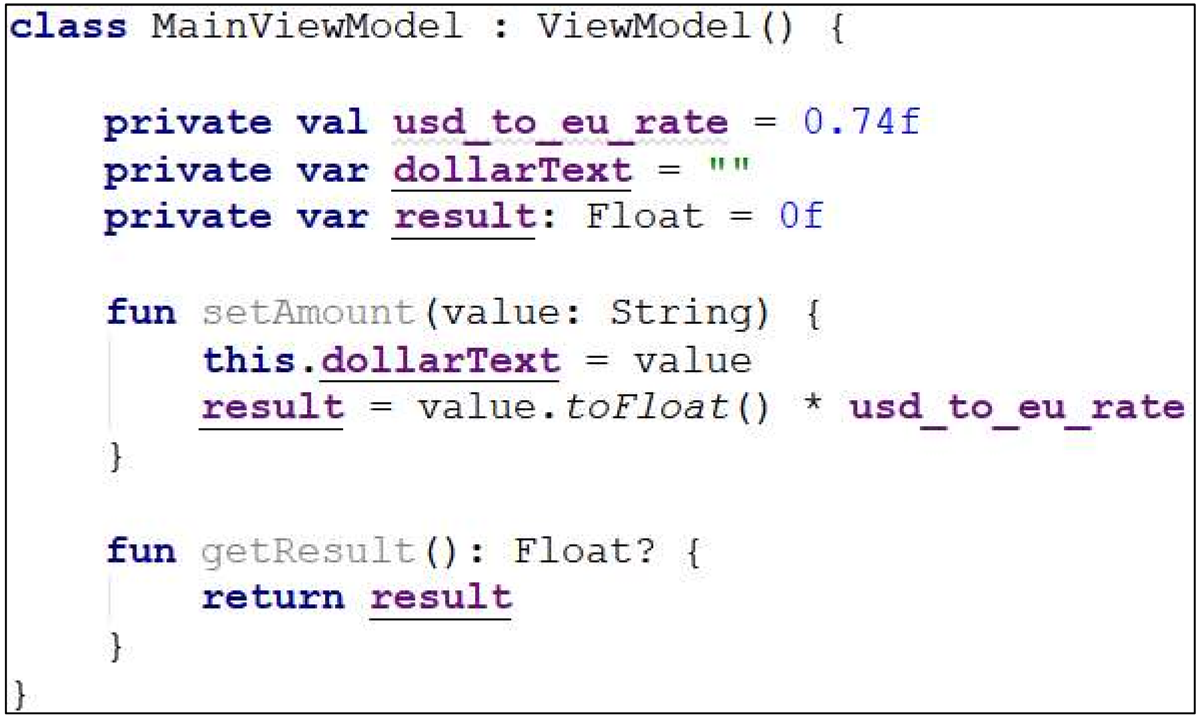
```kotlin
package edu.incheon.viewmodelex.ui.main

import androidx.lifecycle.ViewModel

class MainViewModel : ViewModel() {
    // TODO: Implement the ViewModel
}
```

실습 1: main_fragment.xml

res/layout/main_fragment.xml

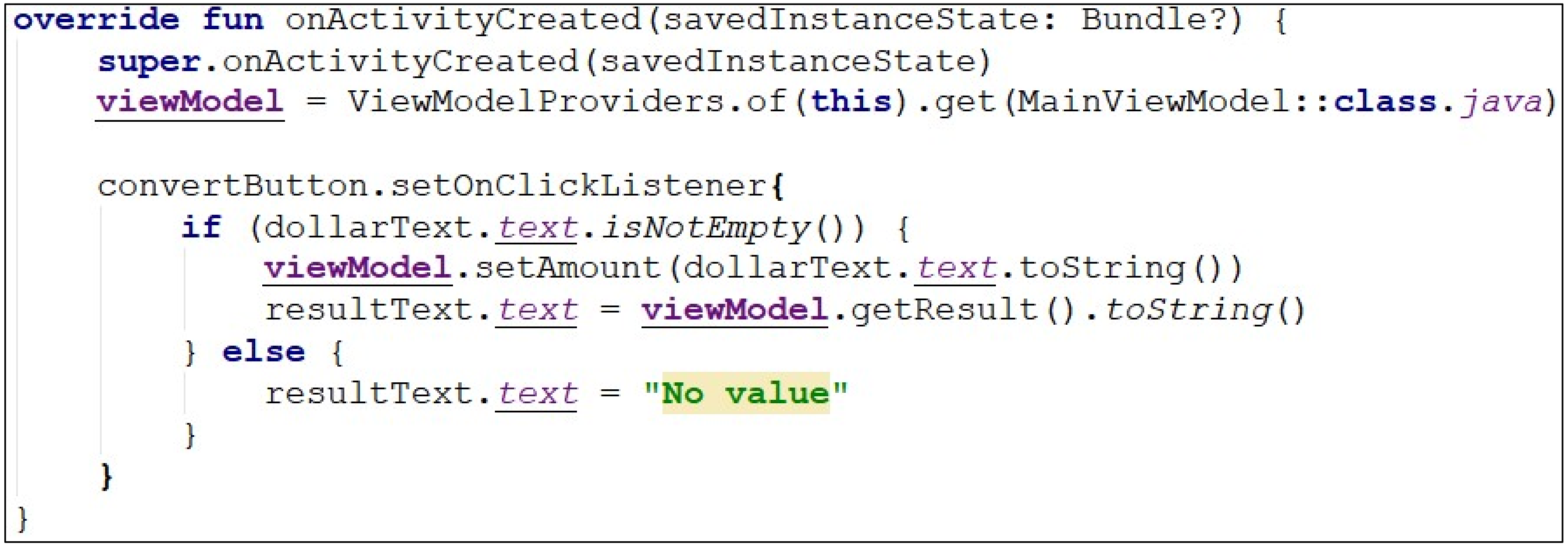

Component Tree
main
resultText - "MainFragme...
dollarText (Number (Deci...
convertButton - "Convert"
MainFragment
CONVERT

실습 1: MainViewModel.kt



```kotlin
class MainViewModel : ViewModel() {

    private val usd_to_eu_rate = 0.74f
    private var dollarText = ""
    private var result: Float = 0f

    fun setAmount(value: String) {
        this.dollarText = value
        result = value.toFloat() * usd_to_eu_rate
    }

    fun getResult(): Float? {
        return result
    }
}
```

실습 1: MainFragment.kt



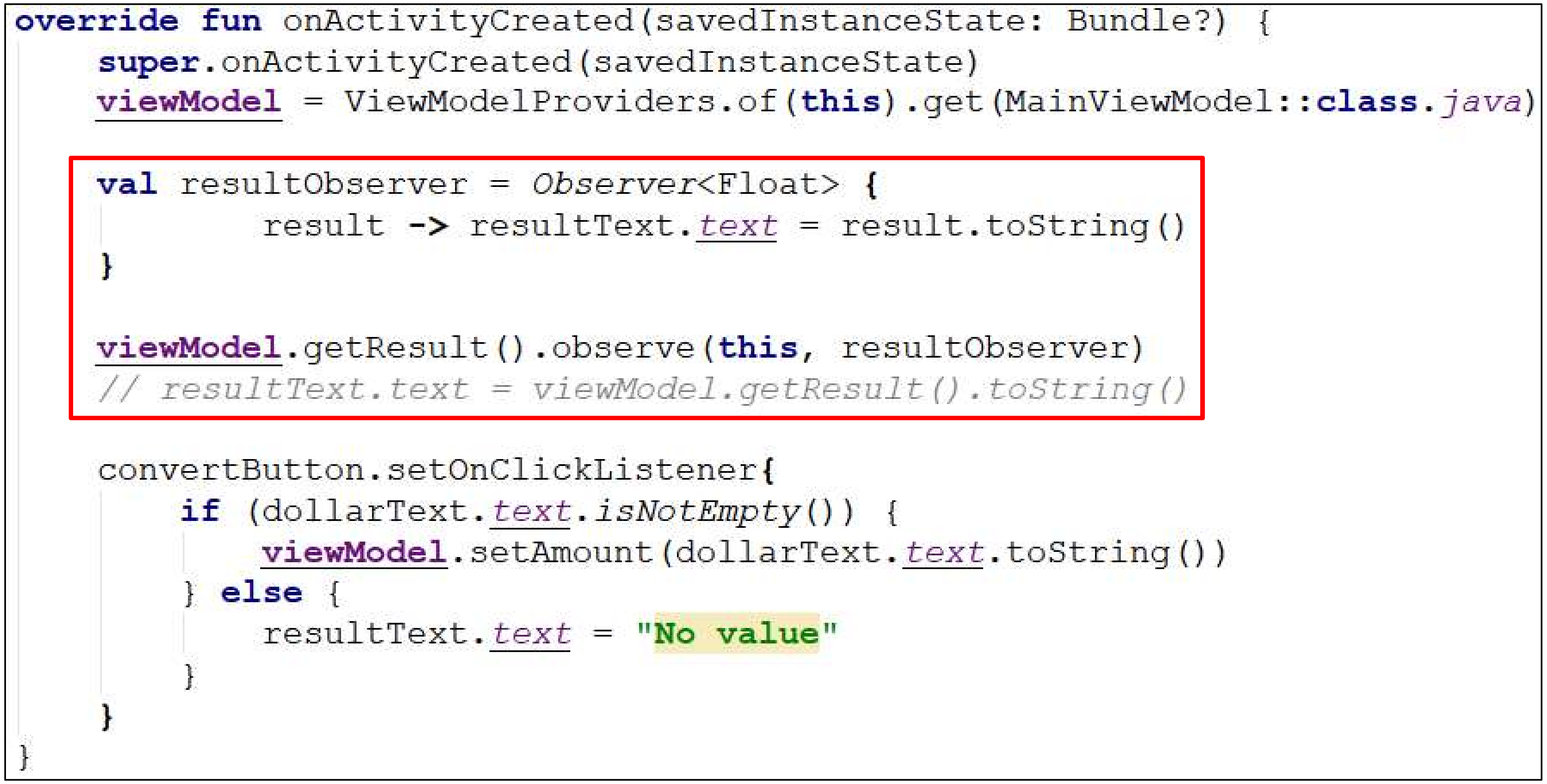
```kotlin
override fun onActivityCreated(savedInstanceState: Bundle?) {
    super.onActivityCreated(savedInstanceState)
    viewModel = ViewModelProviders.of(this).get(MainViewModel::class.java)

    convertButton.setOnClickListener{
        if (dollarText.text.isNotEmpty()) {
            viewModel.setAmount(dollarText.text.toString())
            resultText.text = viewModel.getResult().toString()
        } else {
            resultText.text = "No value"
        }
    }
}
```

AVD에서 실행할 때 환전을 실행시킨 후,
90도 회전시켜 보자.
조금 전에 입력했던 값이 남아있을까?

실습 2: MainViewModel.kt

```kotlin
package edu.incheon.viewmodelex.ui.main

import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel

class MainViewModel : ViewModel() {

    private val usd_to_eu_rate = 0.74f
    private var dollarText = ""
    // private var result: Float = 0f
    private var result: MutableLiveData<Float> = MutableLiveData()

    fun setAmount(value: String) {
        this.dollarText = value
        // result = value.toFloat() * usd_to_eu_rate
        result.value = value.toFloat() * usd_to_eu_rate
    }

    // fun getResult(): Float? {
    fun getResult(): MutableLiveData<Float> {
        return result
    }
}
```

실습 2: MainFragment.kt



```kotlin
override fun onActivityCreated(savedInstanceState: Bundle?) {
    super.onActivityCreated(savedInstanceState)
    viewModel = ViewModelProviders.of(this).get(MainViewModel::class.java)

    val resultObserver = Observer<Float> {
            result -> resultText.text = result.toString()
    }

    viewModel.getResult().observe(this, resultObserver)
    // resultText.text = viewModel.getResult().toString()

    convertButton.setOnClickListener{
        if (dollarText.text.isNotEmpty()) {
            viewModel.setAmount(dollarText.text.toString())
        } else {
            resultText.text = "No value"
        }
    }
}
```