

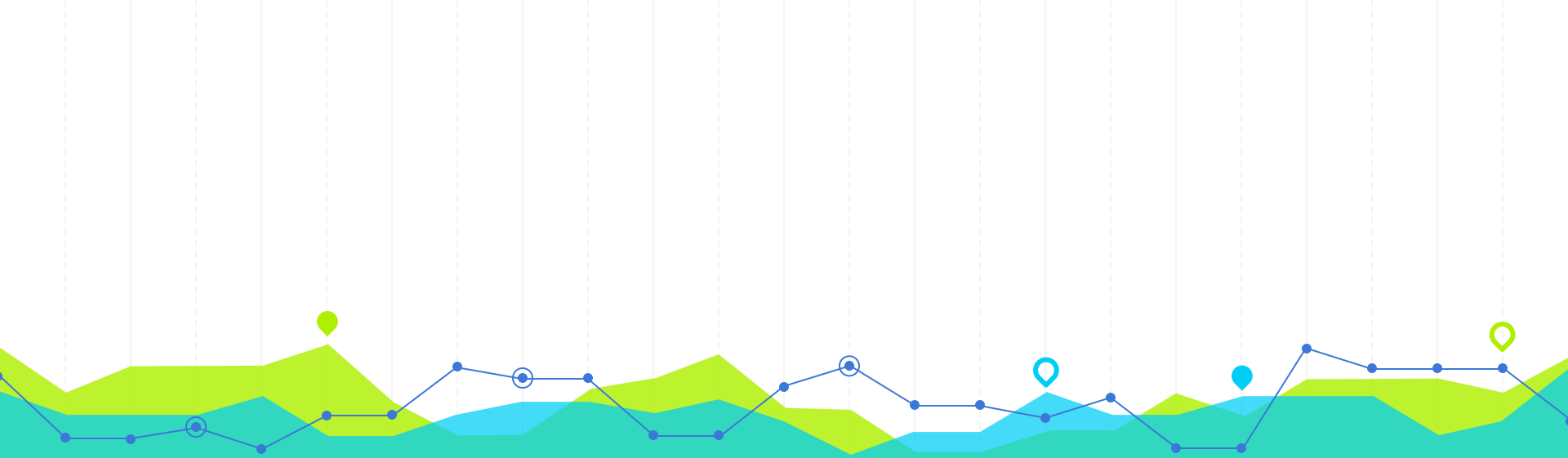
MVVM

Advanced Native Mobile Programming
Week 4

Teknik Informatika - Universitas Surabaya

Topics

- MVVM Concept
- Tutorial: View, Model, ViewModel

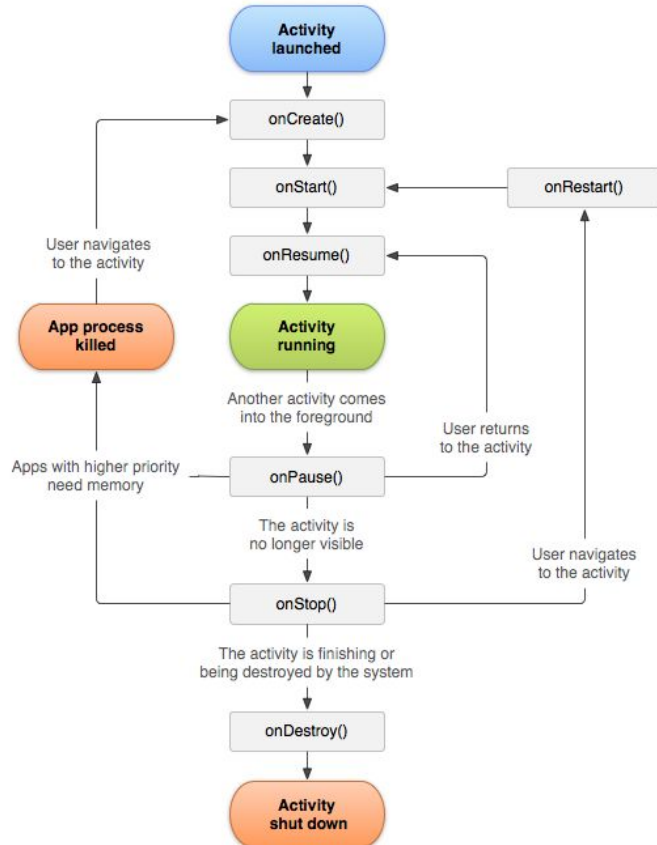


THE CONCEPT

Lifecycle, LiveData, MVVM

1

ACTIVITY LIFECYCLES



- Collection of callback methods to inform object state condition
- Activity, Fragment and ViewModel has its own lifecycle

FRAGMENT LIFECYCLES

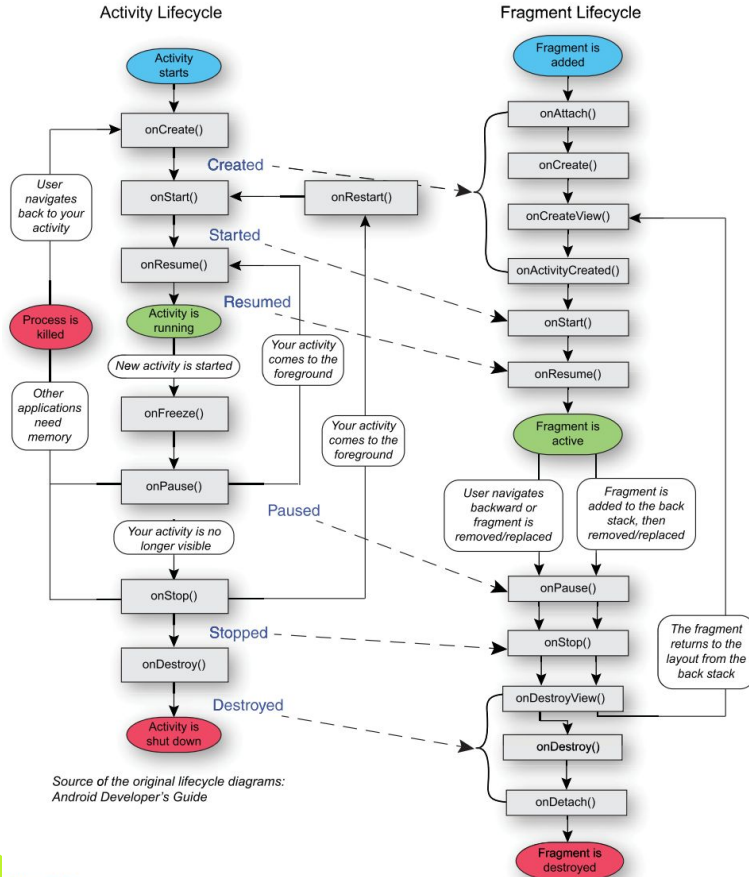


Figure 20.1 Activity and fragment lifecycles

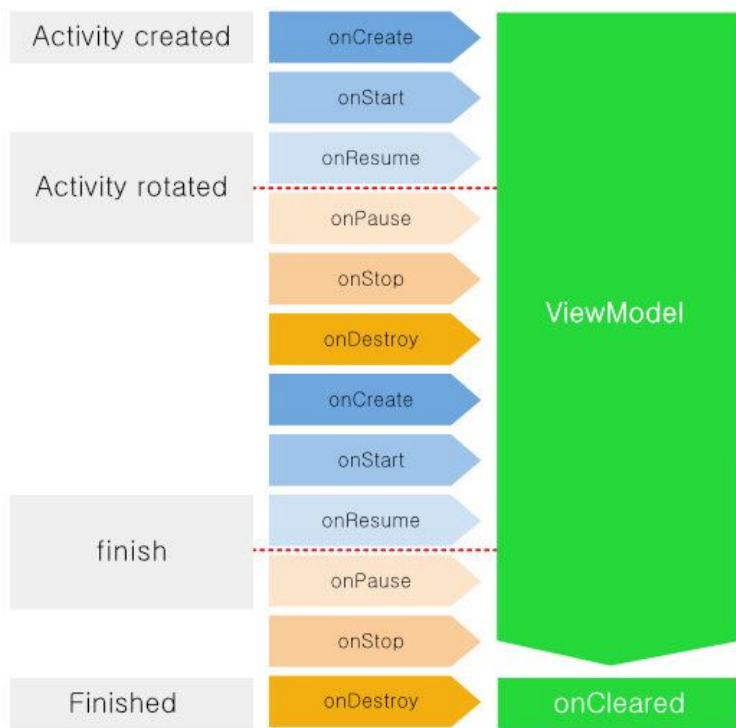
- Fragment lifecycles more complex
- Fragment must be attached first
- At least need one activity to hold/handle a fragment

COMPLEX MICRO MANAGEMENT

- Handle asynchronous calls
- Handle configuration changes
- Data update
- Memory leak problem

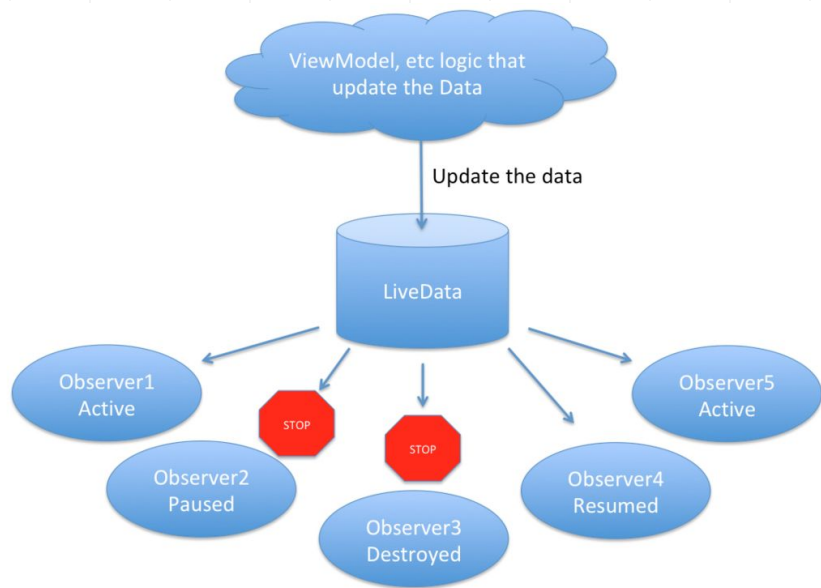


VIEWMODEL LIFECYCLES



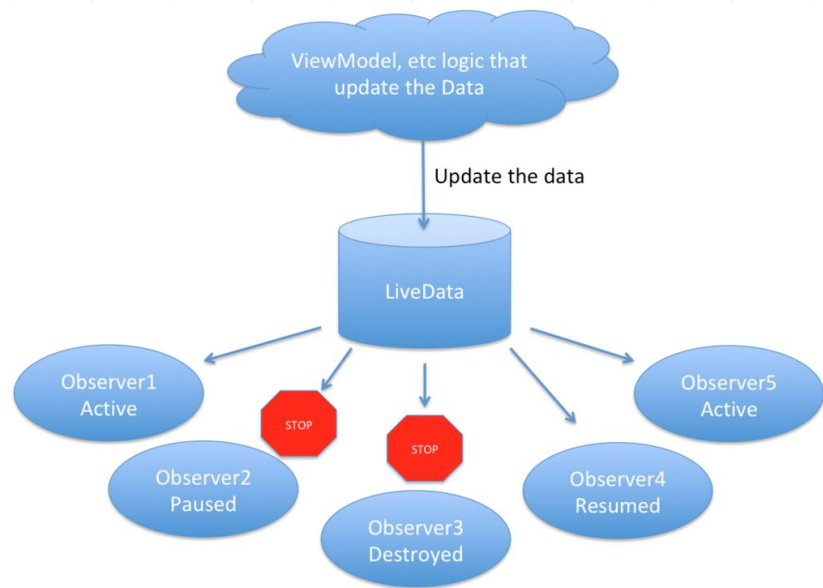
- ViewModel encapsulate the data for a UI controller to let the data survive configuration changes
- ViewModel offer much more simple lifecycles
- ViewModel is lifecycle aware
- ViewModel have one method to handle those state

LIVEDATA

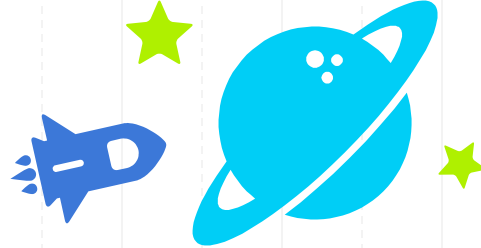


- **Observable** is an object that emit value/data
- **Observer** is attached to the observable and receive the data that observable emit
- **LiveData** is an **observable** data holder class and its lifecycle-aware
- LiveData only notifies **active** observers about updates (and livedata aware that)

LIVEDATA

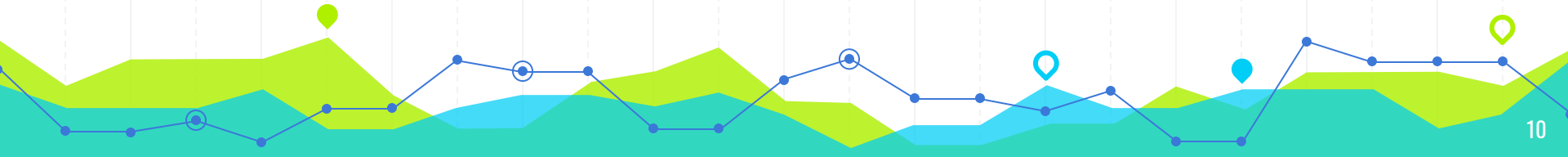


- No memory leaks
- Always up to date data
- Manages configuration changes



MVVM

Model View View Model Architecture

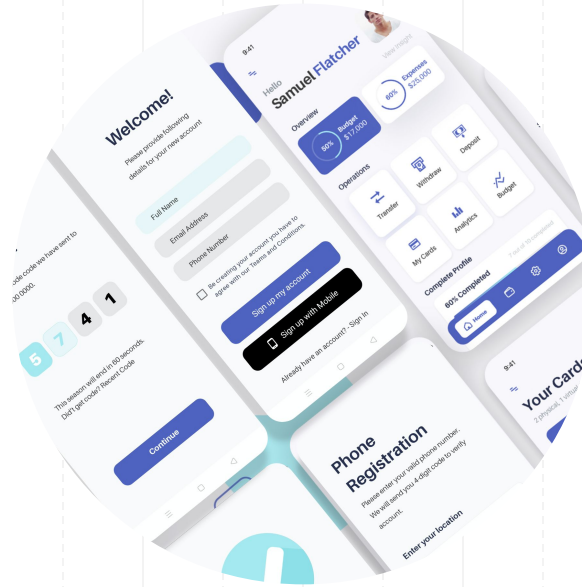


MVVM ARCHITECTURE

- Without architecture -> all UI logic and I/O logic handle by single activity -> God Object
- Provide a way to structure our code in a way that provides some advantages in this terms: separate the business and presentation logic from the UI (means that the model don't need to know what view/view model doing)
- It's associated with View, Model, and ViewModel

VIEW

View is responsible for the layout structure displayed on the screen.



You can also execute UI logic.

MODEL

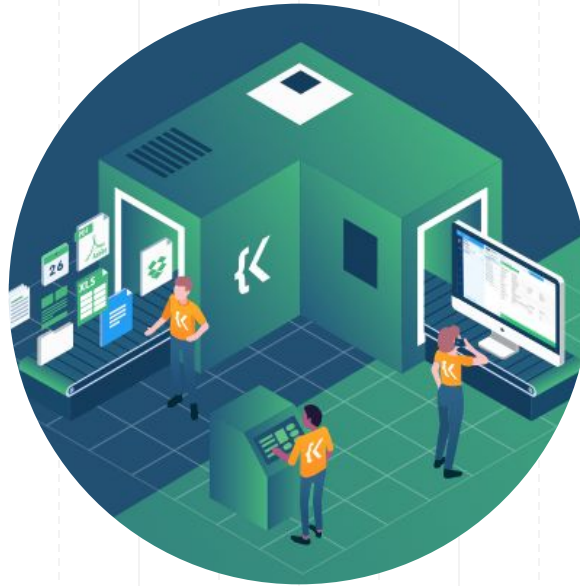
Model is a non-visual class that has the data to use



Usually we retrieve data from REST-API in form of JSON, or from internal sqlite database (DAO)

VIEWMODEL

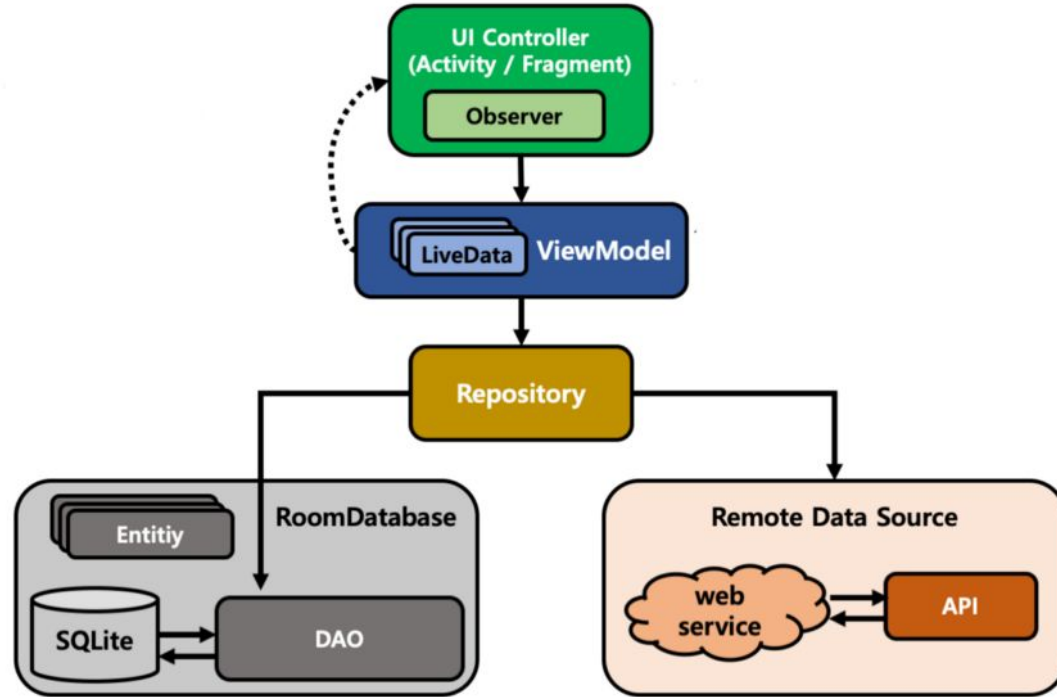
The ViewModel implements the data and commands connected to the View to notify the View of state changes via change notification events.

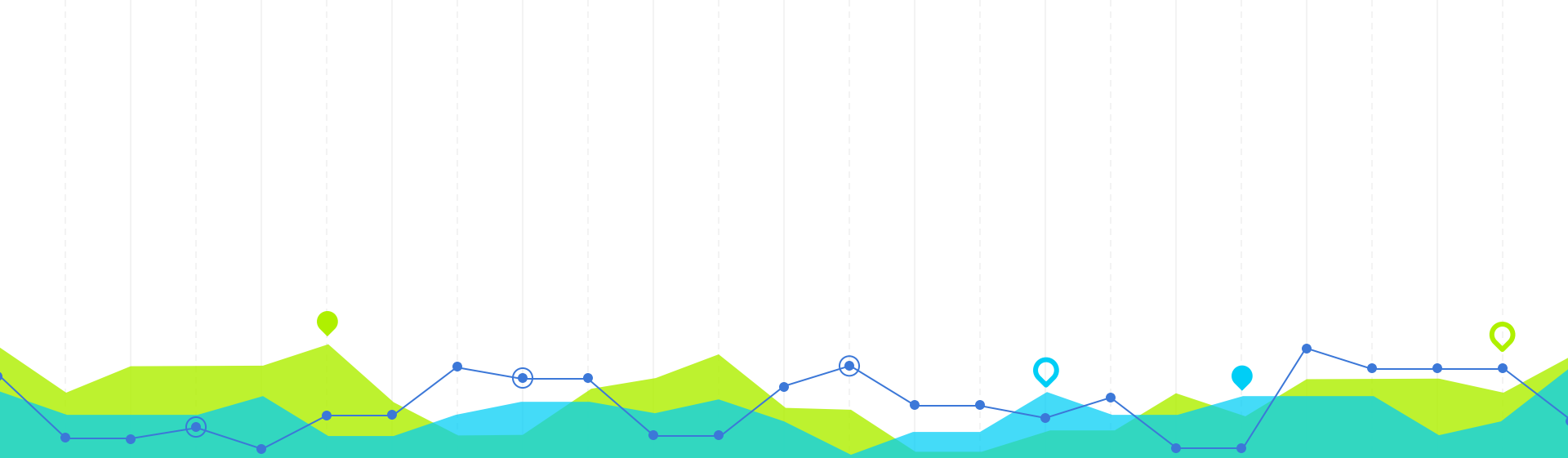


View that receives the state change notification determines whether to apply the change

ViewModel prepare the data for View

MVVM ARCHITECTURE





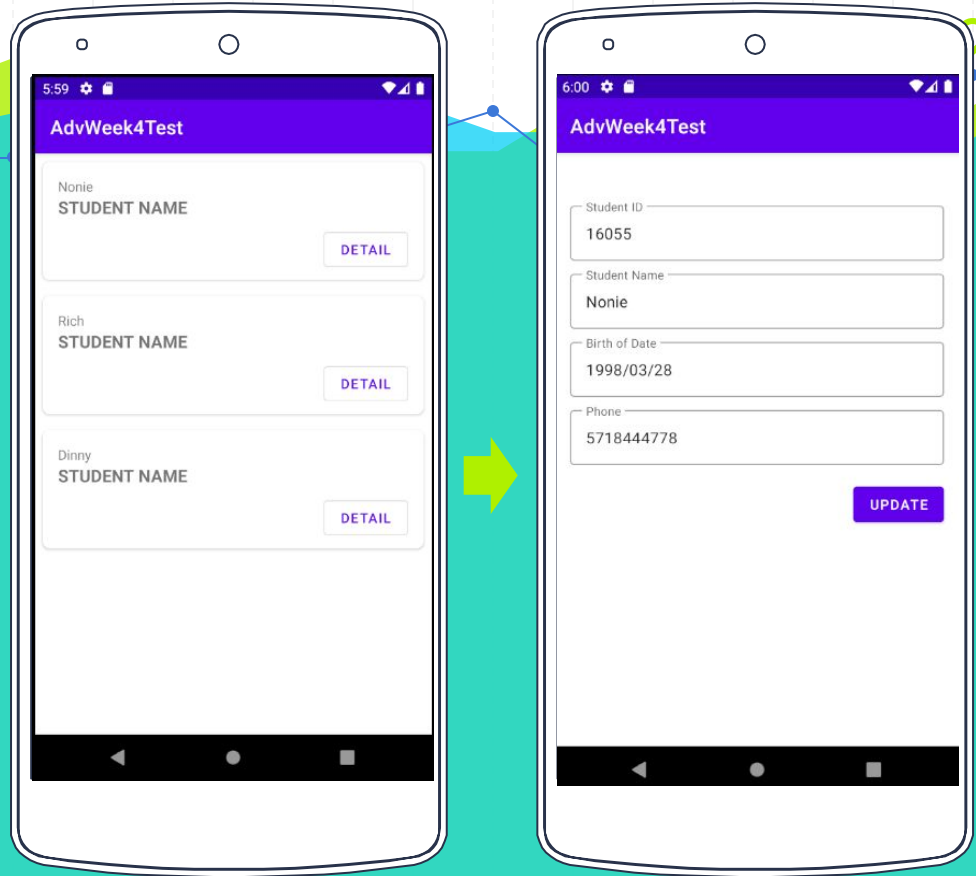
THE TUTORIAL

Develop project that uses MVVM architecture

2

STUDENT LIST

A classic example of data in list




CREATE A NEW PROJECT

- Start a new Android Studio Project, name it as “**AdvWeek4**”
- Use default project configuration and template
- Create a **new github repo** with any name according to your preferences
- Connect/establish the github repo to this project, and make your first **commit and push**

SETUP DEPENDENCIES

- As usual we need to include safeargs dependencies and activate Kotlin extensions
- Open build.gradle (module) and look at the plugins object on top. Add following lines of configuration:

```
plugins {  
    id 'com.android.application'  
    id 'kotlin-android'  
    id 'kotlin-android-extensions'  
    id 'androidx.navigation.safeargs.kotlin'
```

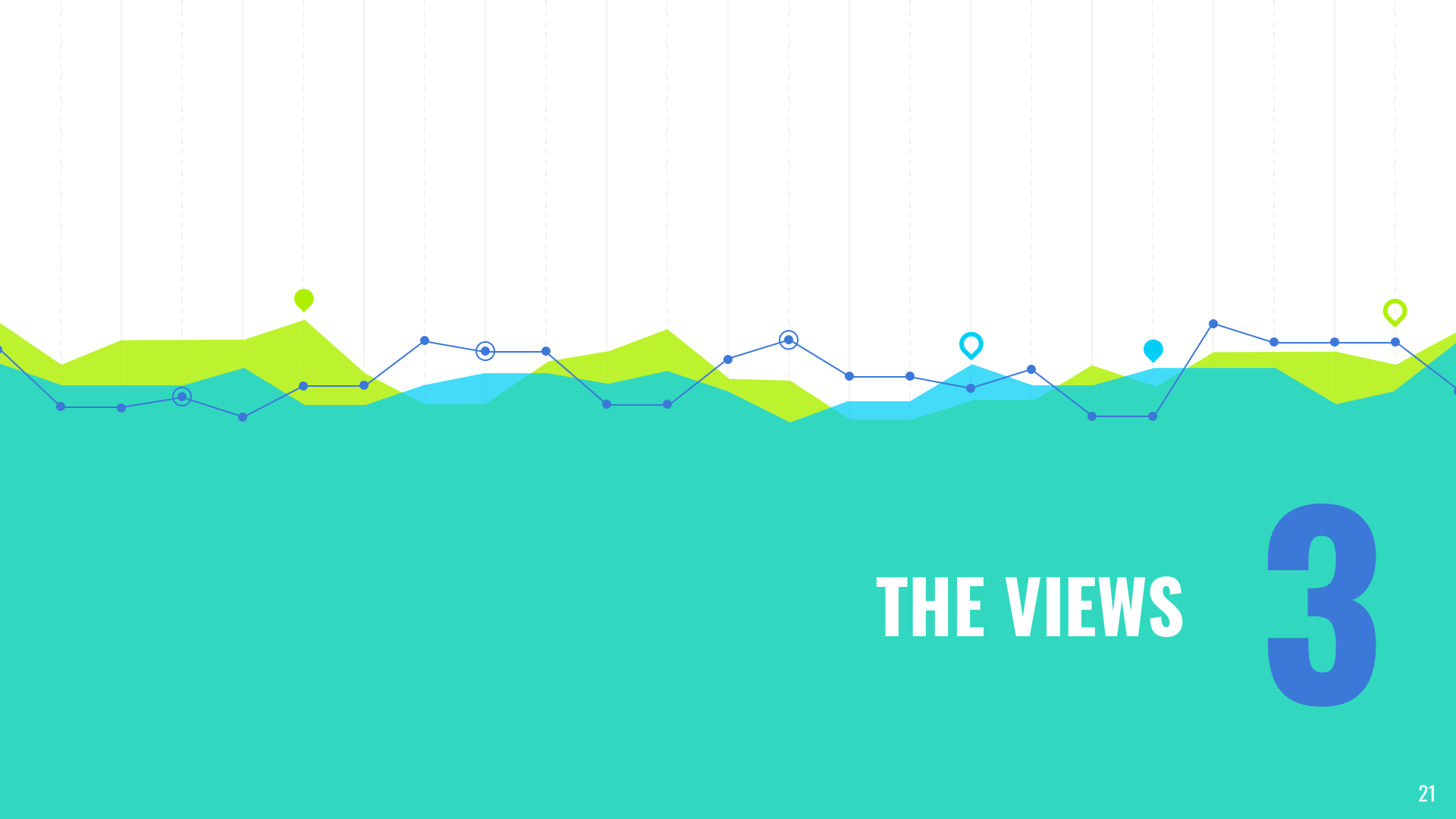


```
}
```

SETUP DEPENDENCIES

- Now open the build.gradle (project), and add following config at the top, then don't forget to sync now:

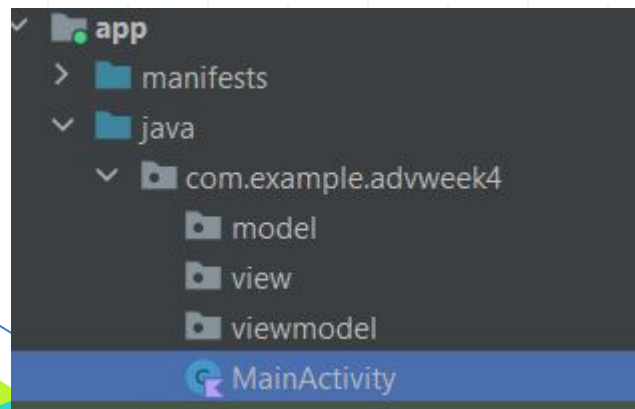
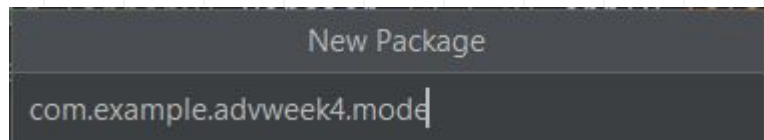
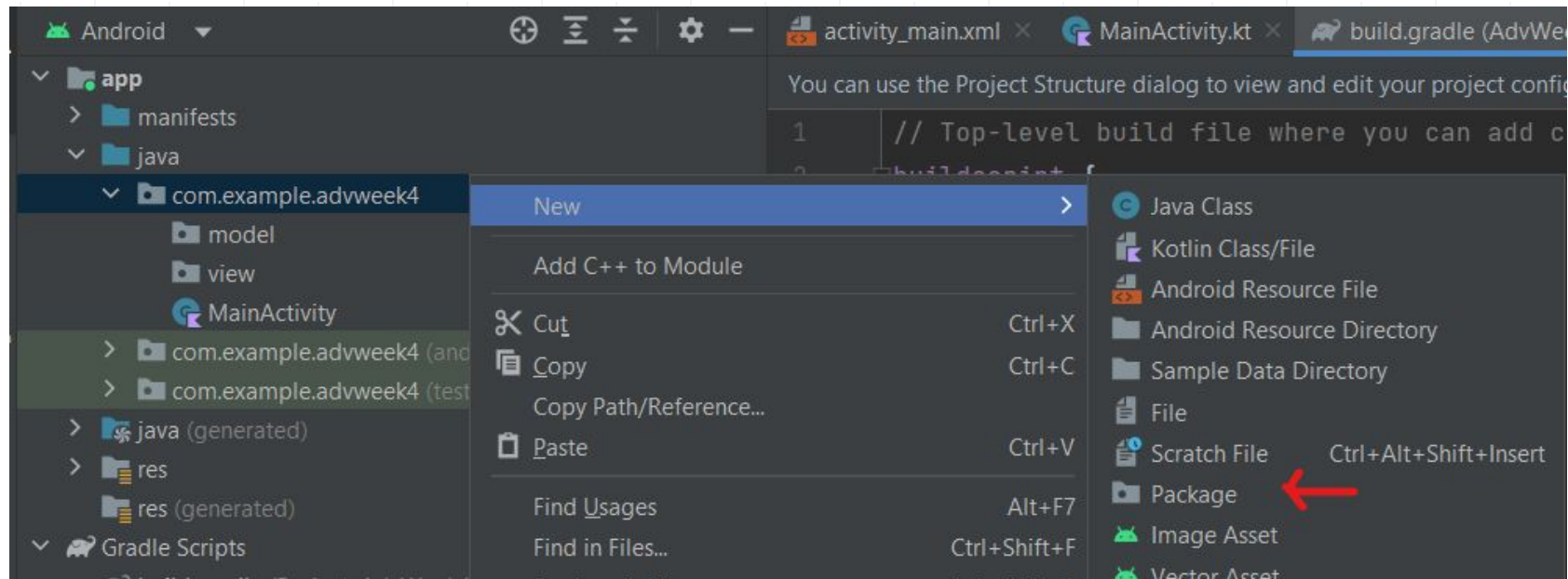
```
buildscript {  
    repositories {  
        google()  
    }  
    dependencies {  
        classpath  
'androidx.navigation:navigation-safe-args-gradle-plugin:2.4.1'  
    }  
}
```



THE VIEWS 3

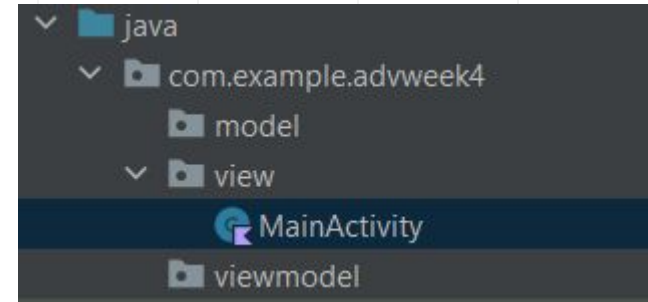
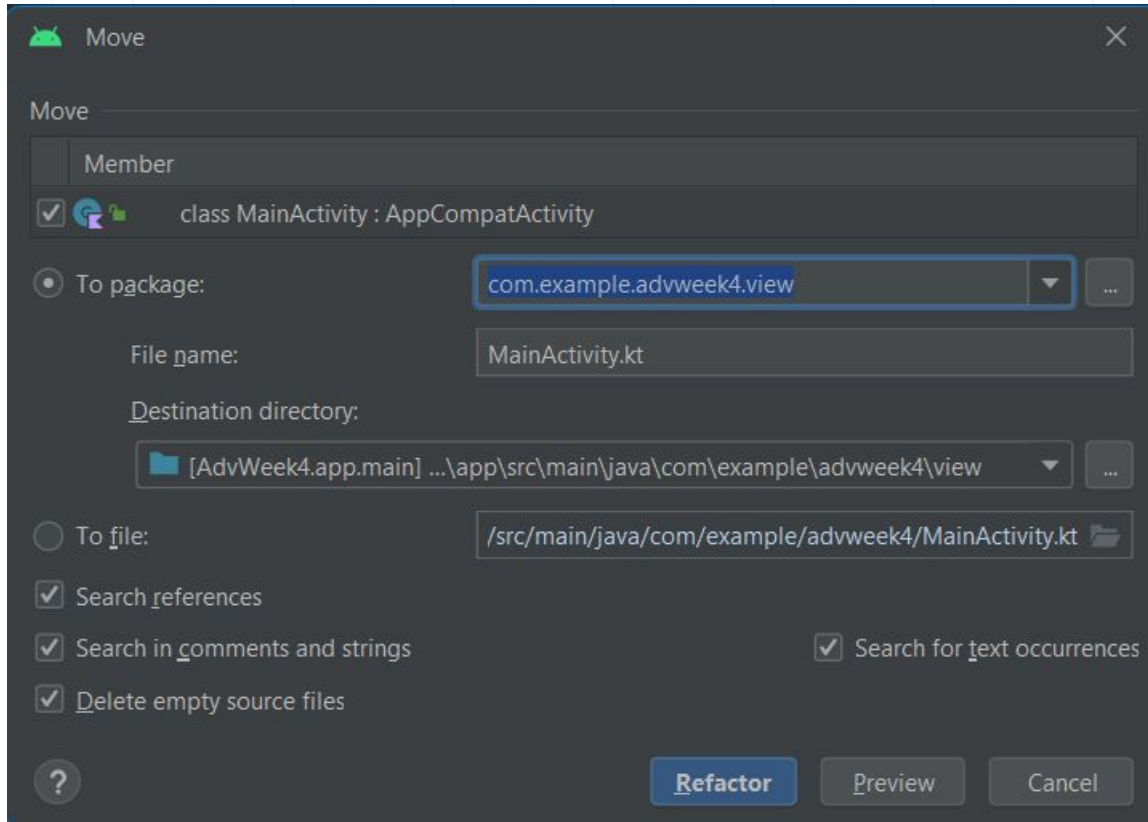
CREATE PACKAGES

- Project should be structured in MVVM way to make it clear and tidy
- Create three new packages “model”, “view”, and “viewmodel”
- Steps:
 - Right click on package name > new > package
 - Type package name



MOVE ACTIVITY

- According to MVVM architecture, every object that related with UI logic must be placed inside “view” package
- This include activities
- Drag and drop current MainActivity.kt inside the “view”
- Go with the default and click “refactor”
- Build > rebuild project to see the changes



CREATE FRAGMENTS

- In “view” package, create two fragments:
 - StudentListFragment
Layout -> fragment_student_list.xml
 - StudentDetailFragment
Layout -> fragment_student_detail.xml

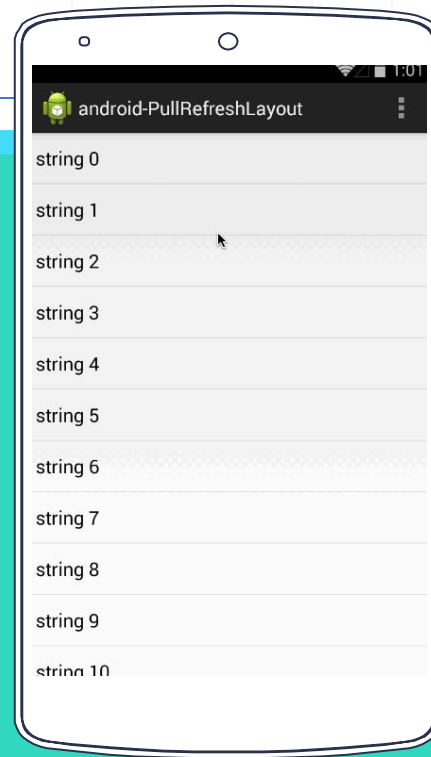
STUDENT LIST LAYOUT

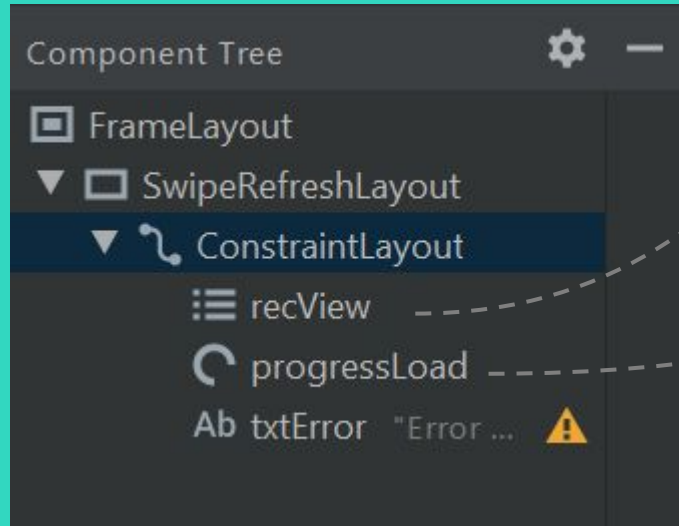
- Delete the default “textview” and add “constraint layout”
- Open the xml codes to add following xml manually:

```
<FrameLayout . . . >  
    <androidx.swiperefreshlayout.widget.SwipeRefreshLayout  
        android:layout_width="match_parent"  
        android:layout_height="match_parent"  
        tools:context=".view.StudentListFragment">  
  
        <constraint layout. . . />  
  
    </androidx.swiperefreshlayout.widget.SwipeRefreshLayout>  
</FrameLayout>
```

SWIPE REFRESH LAYOUT

It detects the vertical swipe, displays a distinctive progress bar, and triggers callback methods in your app (usually initiate fetching data action)





RecyclerView

Progress Bar

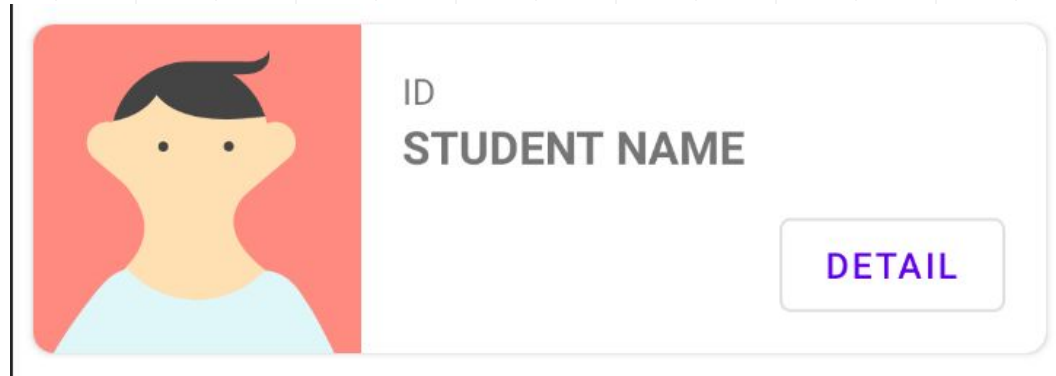
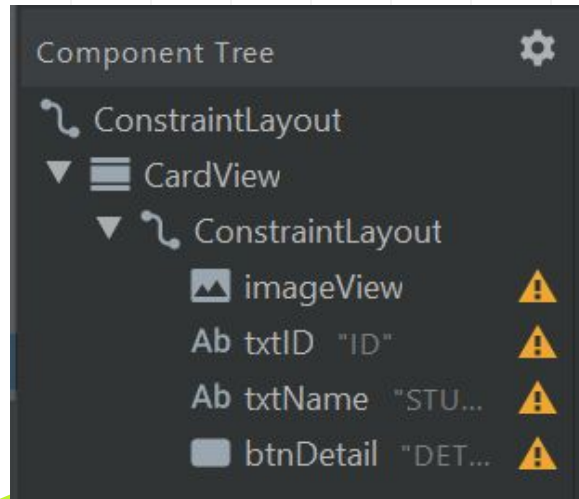
Text View

tem 0
tem 1
tem 2
tem 3
tem 4
tem 5
tem 6
tem 7
tem 8
tem 9

Error while loading data

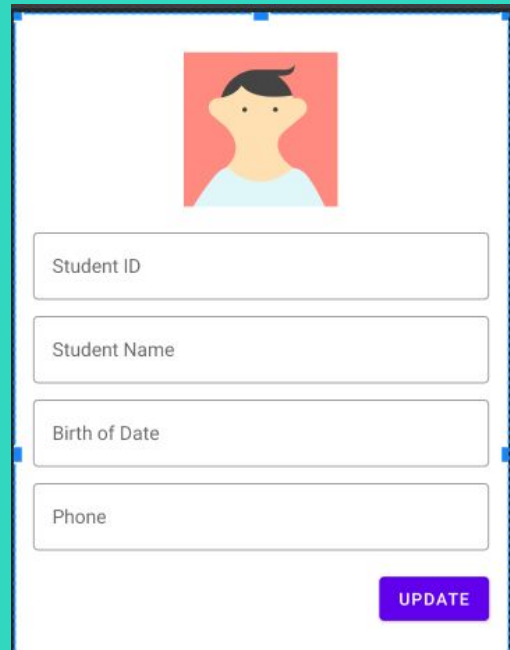
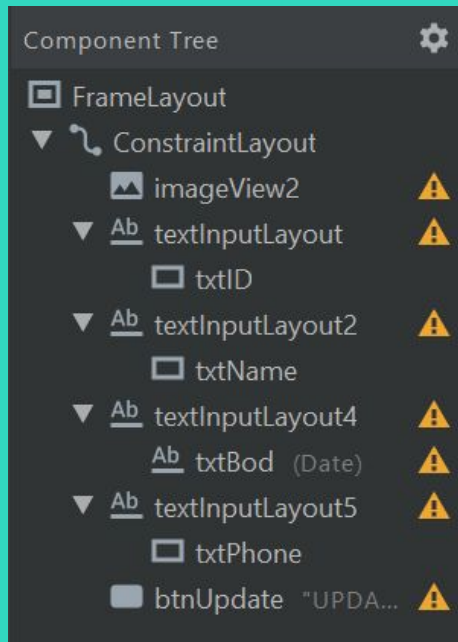
STUDENT LIST ITEM LAYOUT

- Recycler View needs item layout to render each data into the list
- Create a new layout, name it as “student_list_item”



FRAGMENT STUDENT DETAIL LAYOUT

Mostly its used TextInputLayout with Material Design Outline Style

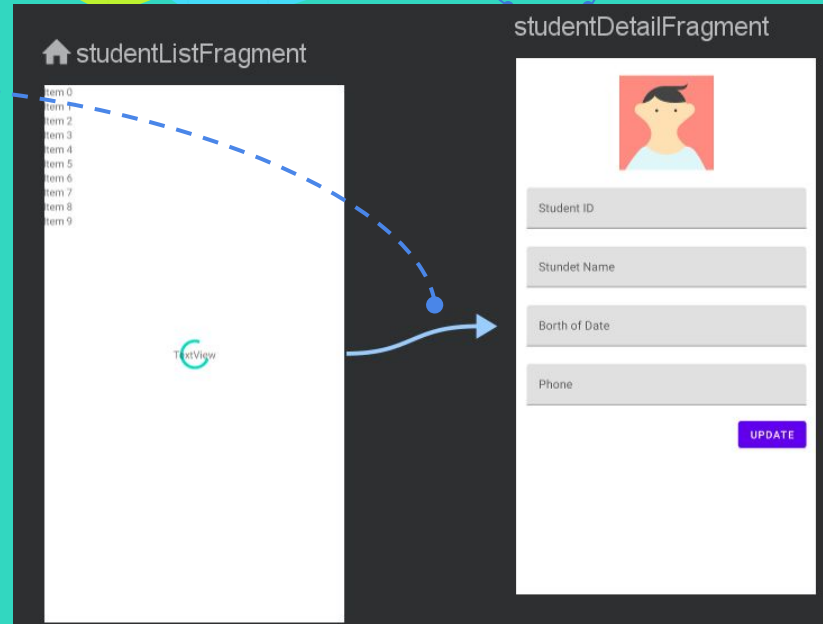


Id = actionStudentDetail

NAVIGATION GRAPH

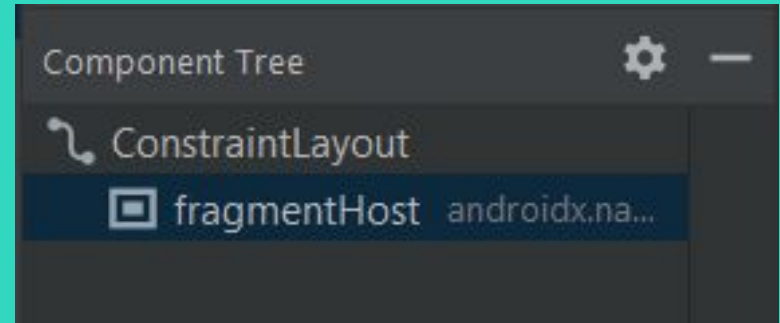
Next create new navigation graph, name it as “main_navigation.xml”

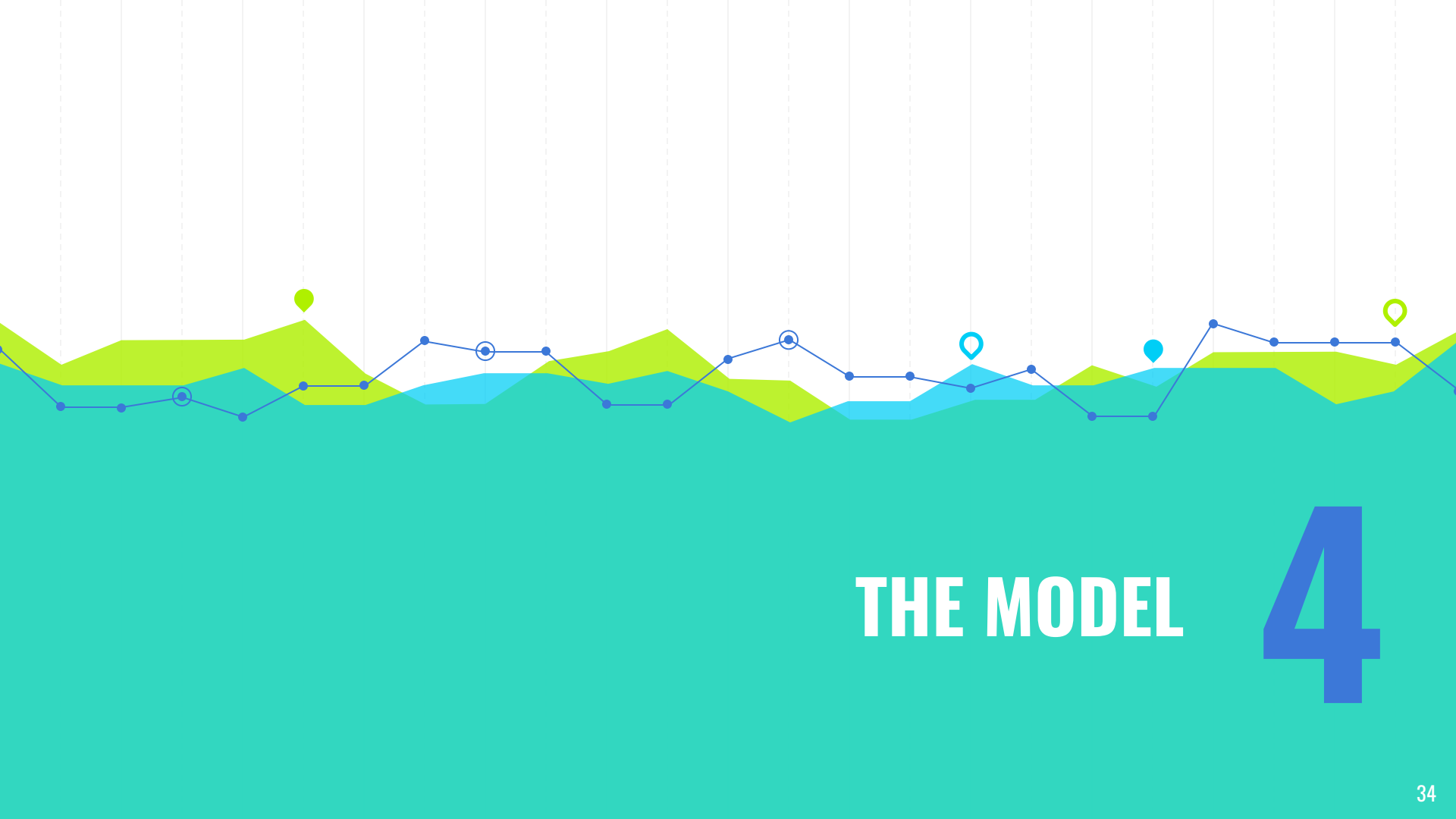
Dont forget to rebuild after finished



SETUP HOST FRAGMENT

- Open activity_main.xml, delete the default text view.
- Drag and drop “nav host fragment”
- Choose the “main_navigation”
- Set it full width and height
- Set id to “fragmentHost”





THE MODEL

4

CREATE MODEL

- Model is just simple file that hold “data” blueprint in form of data class.
- Model may have more than one class
- Right click on package “model” > new > Kotlin/Class file
- Named it as “Model”, choose “File” and then press enter key

DATACLASS STUDENT

```
data class Student(  
    val id:String?,  
    val name:String?,  
    val dob:String?,  
    val phone:String?,  
    val photoUrl:String?  
)
```

- To simplify things, our model only accept “String”
- dob is “date of birth”

CREATE ADAPTER

- To make RecyclerView work as intended, then we need to supply it with adapter
- Right click on “view” package > new > “Kotlin/File class”
- Named it as “StudentListAdapter”
- Choose “class”, and then press enter

Why this adapter put in the “View” package? Why don't we put it in the “Model” package?

This adapter requires ArrayList data of Student

CREATE ADAPTER

```
class StudentListAdapter(val studenList:ArrayList<Student>) {  
    class StudentViewHolder(var view: View) : RecyclerView.ViewHolder(view)  
}
```

This is an inner class that primarily used by the adapter to update the UI

CREATE ADAPTER

```
class StudentListAdapter(val studenList:ArrayList<Student>)  
:RecyclerView.Adapter<StudentListAdapter.StudentViewHolder>()  
{  
    class StudentViewHolder(var view: View) : RecyclerView.ViewHolder(view)  
}
```

Extend the class to
RecyclerView.Adapter that requires the
innerclass “student view holder”

IMPLEMENTS ALL MEMBER

```
override fun onCreateView(parent: ViewGroup, viewType:  
Int):StudentViewHolder {  
}
```

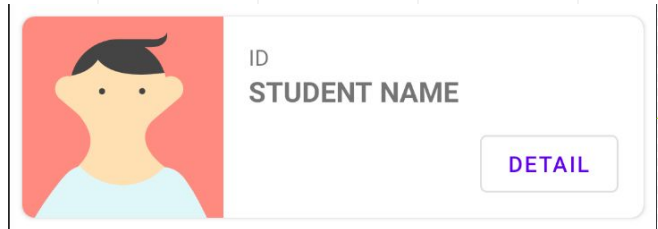
```
override fun onBindViewHolder(holder: StudentViewHolder, position: Int) { }
```

```
override fun getItemCount(): Int {  
    return studenList.size  
}
```

Use auto-complete helper from android studio. Do not type these all manually!

CREATE VIEW HOLDER

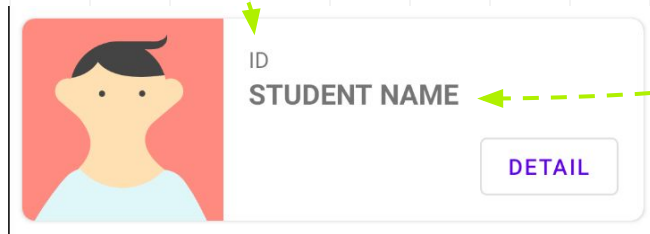
```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):  
    StudentViewHolder {  
        val inflater = LayoutInflater.from(parent.context)  
        val view = inflater.inflate(R.layout.student_list_item, parent, false)  
  
        return StudentViewHolder(view)  
    }
```



Load the views that will be rendered on each item in the list

BIND VIEW HOLDER

```
override fun onBindViewHolder(holder: StudentViewHolder, position: Int) {  
    holder.view.txtID.text = studenList[position].id  
    holder.view.txtName.text = studenList[position].name  
}
```



BUTTON DETAIL LISTENER

```
override fun onBindViewHolder(holder: StudentViewHolder, position: Int) {  
    holder.view.txtID.text = studenList[position].id  
    holder.view.txtName.text = studenList[position].name  
  
    holder.view.btnDetail.setOnClickListener {  
        val action = StudentListFragmentDirections.actionStudentDetail()  
        Navigation.findNavController(it).navigate(action)  
    }  
}
```

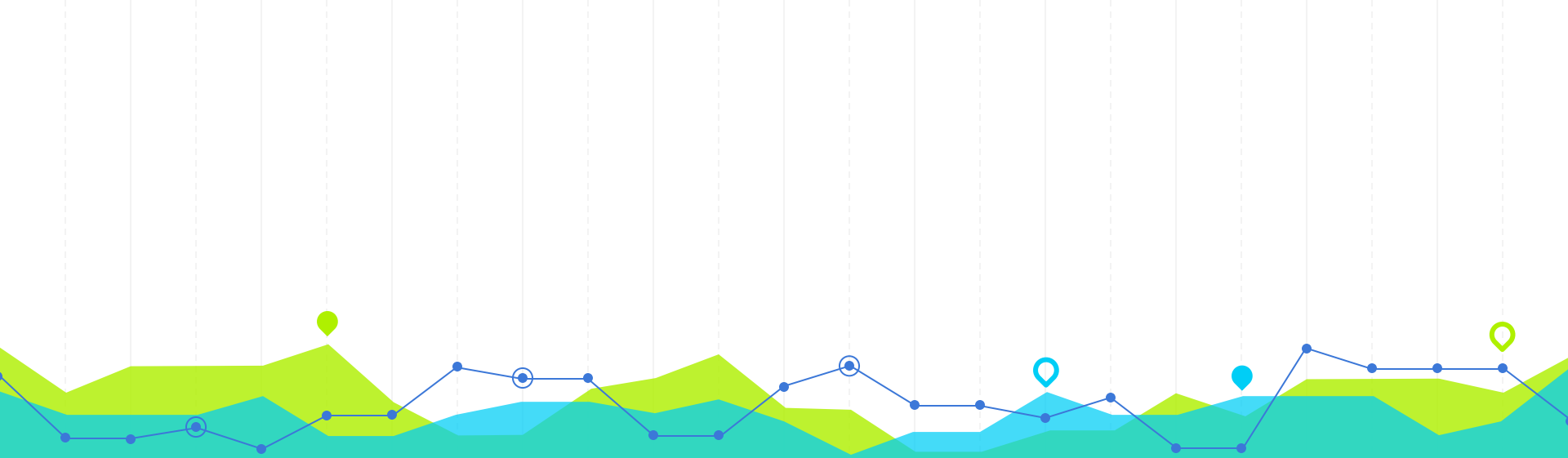
Button Detail is used to navigate to the Student Detail screen

UPDATE STUDENT LIST FUNCTION

The main purpose of this function is to “refresh” student data list

```
fun updateStudentList(newStudentList: ArrayList<Student>) {  
    studenList.clear()  
    studenList.addAll(newStudentList)  
    notifyDataSetChanged()  
}
```

tells the recycler view that every single item should be updated.



THE VIEW MODEL

5

CREATE VIEW MODEL

- The ViewModel class is designed to store and manage UI-related data in a lifecycle conscious way. View model provide a way to emit the data to observer.
- Create a new Kotlin class in “viewmodel” package, named it as “ListViewModel.kt”
- This ListViewModel is used as bridging operation between student model and fragment list

Extends View Model

LISTVIEW MODEL

```
class ListViewModel:ViewModel() {  
    val studentsLD = MutableLiveData<ArrayList<Student>>()  
    val studentLoadErrorLD = MutableLiveData<Boolean>()  
    val loadingLD = MutableLiveData<Boolean>()  
}
```

- studentsLD = live data that “emit” students data (in list form)
- studentLoadErrorLD = live data that “emit” error status (boolean)
- loadingLD = live data that “emit” data loading status (boolean)

MutableLiveData is type of LiveData that updateable

Copy from here:
<https://pastebin.com/GUtgT9hG>

REFRESH FUNCTION

Hardcoded data. In the next lecture will be replaced with actual data from external sources

```
fun refresh() {  
    val student1 =  
        Student("16055", "Nonie", "1998/03/28", "5718444778", "http://dummyimage.com/75x100.jpg/cc0000/ffffff")  
  
    val student2 =  
        Student("13312", "Rich", "1994/12/14", "3925444073", "http://dummyimage.com/75x100.jpg/5fa2dd/ffffff")  
  
    val student3 =  
        Student("11204", "Dinny", "1994/10/07", "6827808747", "http://dummyimage.com/75x100.jpg/5fa2dd/ffffff1")  
  
    val studentList: ArrayList<Student> = arrayListOf<Student>(student1, student2, student3)  
  
    studentsLD.value = studentList  
    studentLoadErrorLD.value = false  
    loadingLD.value = false  
}
```

Refresh function in real case used to load data from server/local and prepare livedata objects. Ready to be observed

USE VIEW MODEL

- Next step is to use the view model in StudentListFragment
- Open StudentListFragment.kt, create two object:

```
class StudentListFragment : Fragment() {  
    private lateinit var viewModel: ListViewModel  
    private val studentListAdapter = StudentListAdapter(arrayListOf())  
}
```

Override onCreateView, and then
initialize and call refresh function

INITIALIZE VIEW MODEL

```
viewModel = ViewModelProvider(this).get(ListViewModel::class.java)
viewModel.refresh()
```

```
recView.layoutManager = LinearLayoutManager(context)
recView.adapter = studentListAdapter
```

```
observeViewModel()
```

Initialize the adapter

OBSERVE VIEW MODEL FUNCTION

This function set actions for observer about how to handle the emitted data

```
fun observeViewModel() {  
}
```

OBSERVE VIEW MODEL FUNCTION

```
fun observeViewModel() {
```

```
    viewModel.studentsLD.observe(viewLifecycleOwner, Observer {  
        studentListAdapter.updateStudentList(it)  
    })  
}
```

This observe function is used to “observe” Students data

Any program under the Observer function will be executed if needed (ie. activity state changed, configuration changed, etc)

Students LiveData (observable) attached to this fragment (observer). Every time observer changes its state, the observable emit the data

OBSERVE VIEW MODEL FUNCTION

```
viewModel.studentLoadErrorLD.observe(viewLifecycleOwner, Observer {  
    if(it == true) {  
        txtError.visibility = View.VISIBLE  
    } else {  
        txtError.visibility = View.GONE  
    }  
}))
```

This observe “loading error” livedata.
For now its just use dummy data

OBSERVE VIEW MODEL FUNCTION

```
viewModel.loadingLD.observe(viewLifecycleOwner, Observer {  
    if(it == true) {  
        recView.visibility = View.GONE  
        progressLoad.visibility = View.VISIBLE  
    } else {  
        recView.visibility = View.VISIBLE  
        progressLoad.visibility = View.GONE  
    }  
}))
```

This observe “loading” LiveData. For now its just use dummy data



HOMEWORK 6

CREATE VIEW MODEL

- Create new ViewModel to handle the StudentDetailFragment
- Name it as “DetailViewModel.kt”

```
class DetailViewModel:ViewModel() {  
    val studentLD = MutableLiveData<Student>()  
  
}
```

This DetailViewModel only handle
single Student Data

FETCH FUNCTION

- Create fetch function (it has same function like “refresh”)

```
class DetailViewModel:ViewModel() {  
    val studentLD = MutableLiveData<Student>()  
  
    fun fetch() {  
        val student1 = Student("16055", "Nonie", "1998/03/28", "5718444778",  
                                "http://dummyimage.com/75x100.jpg/cc0000/ffffff")  
        studentLD.value = student1  
    }  
}
```

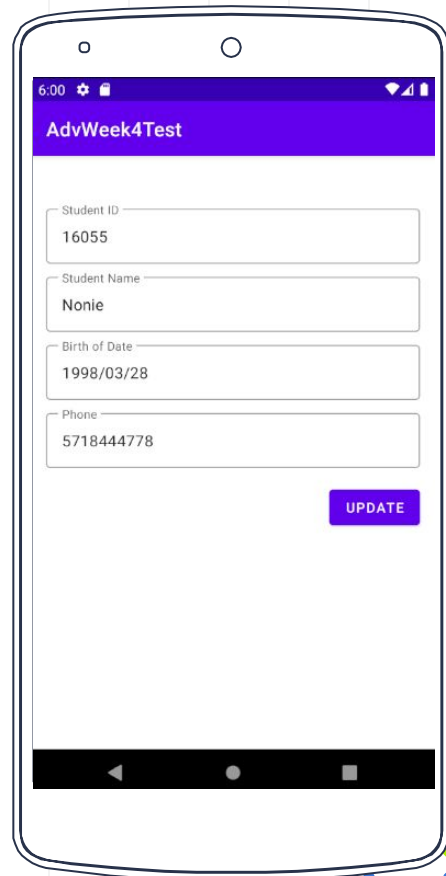
Prepare single dummy data student

USE THE DETAIL VIEW MODEL

Your job is to use this new view model in StudentDetailFragment

Hints:

- call fetch function in StudentDetailFragment.kt
- make observer for the “Student” livedata
- Ignore button update
- use setText method to change the edit text value



DUE DATE

- You must commit + push not exceeding the date (see the due date on ULS)
- Submit the github link to ULS



THANKS!

Any questions?

You can find me at
andre@staff.ubaya.ac.id

