Intro to Architecture Components

Recap

- We studied the Volley library.
- We saw the Singleton pattern, and used it to ensure that we had only a single request queue.
- We saw how to set up and use DownloadManager, a system Service, to do background transfers.
- We saw how to use JobScheduler to fetch data in the background.

Today

- We’ll start work on the Android Architecture Components- specifically, **ViewModel**, and **LiveData**.

Android Architecture Components

- They are a part of Android Jetpack, which is a huge and relatively new library separate from the platform APIs.
- Updated more frequently than the Android platform itself.
- These components were designed to save programmers time in writing tons of boilerplate code (for example, loaders).
- There are several components.

Android Architecture Components

- The Data Binding Library
  - Helps you avoid writing Java code using annotations in the xml file.
- Lifecycle and LifecycleOwner
  - Help you manage your app lifecycle.
- LiveData
  - An observable data holder, lifecycle-aware, can respond to changes in data.
- Navigation
  - Helps you manage navigation within your app.
- Paging Library
  - Helps you gradually load data into RecyclerView.

Android Architecture Components

- Room Persistence Library
  - Easy read/write to SQLite databases using annotations.
- ViewModel
  - Helps preserve data model through configuration changes.
- WorkManager
  - Abstraction layer over JobScheduler, AlarmManager, and Firebase JobDispatcher that automatically selects between these.
- The architecture components are constantly evolving.
- They are now the favored tool for writing nice Android apps.
- Today, we'll focus on ViewModel and LiveData.

LiveData

- LiveData is an observable wrapper around other data types.
- It is lifecycle-aware.
- Typical workflow for LiveData objects:
  - You wrap your datatype in LiveData.
  - Create an Observer object, define the onChanged() method.
  - Attach the Observer object to the LiveData object using observe().
- It’s impossible to discuss LiveData without discussing ViewModel.

ViewModel

- ViewModel decouples UI data from the UI controller (the Activity/Fragment).
- This allows data in the ViewModel to survive configuration changes.
- ViewModel is lifecycle-aware with regard to configuration changes.
- However, a ViewModel cannot survive the process being stopped! You still need onSaveInstanceState().

ViewModel

- A ViewModel is a great way to separate the data from any fragment or activity.
- It provides a fantastic way to share data between Fragments.
- You typically store your LiveData objects inside ViewModels.
- Testing is now easier, since you can make mock data sources to feed your ViewModel.

ViewModel and LiveData: Example 33

- Let’s take the Openweathermap API example and modify it.
- Recall: we used background threads and static classes to make things lifecycle-aware.
- We’ll now attempt to do the same using LiveData and ViewModel.
- The UI will need to pass in the location to our ViewModel.
- The ViewModel will hold a MutableLiveData<WeatherData>, which will contained the parsed WeatherData object.
- We’ll use a background thread inside ViewModel to get the data on a separate thread.

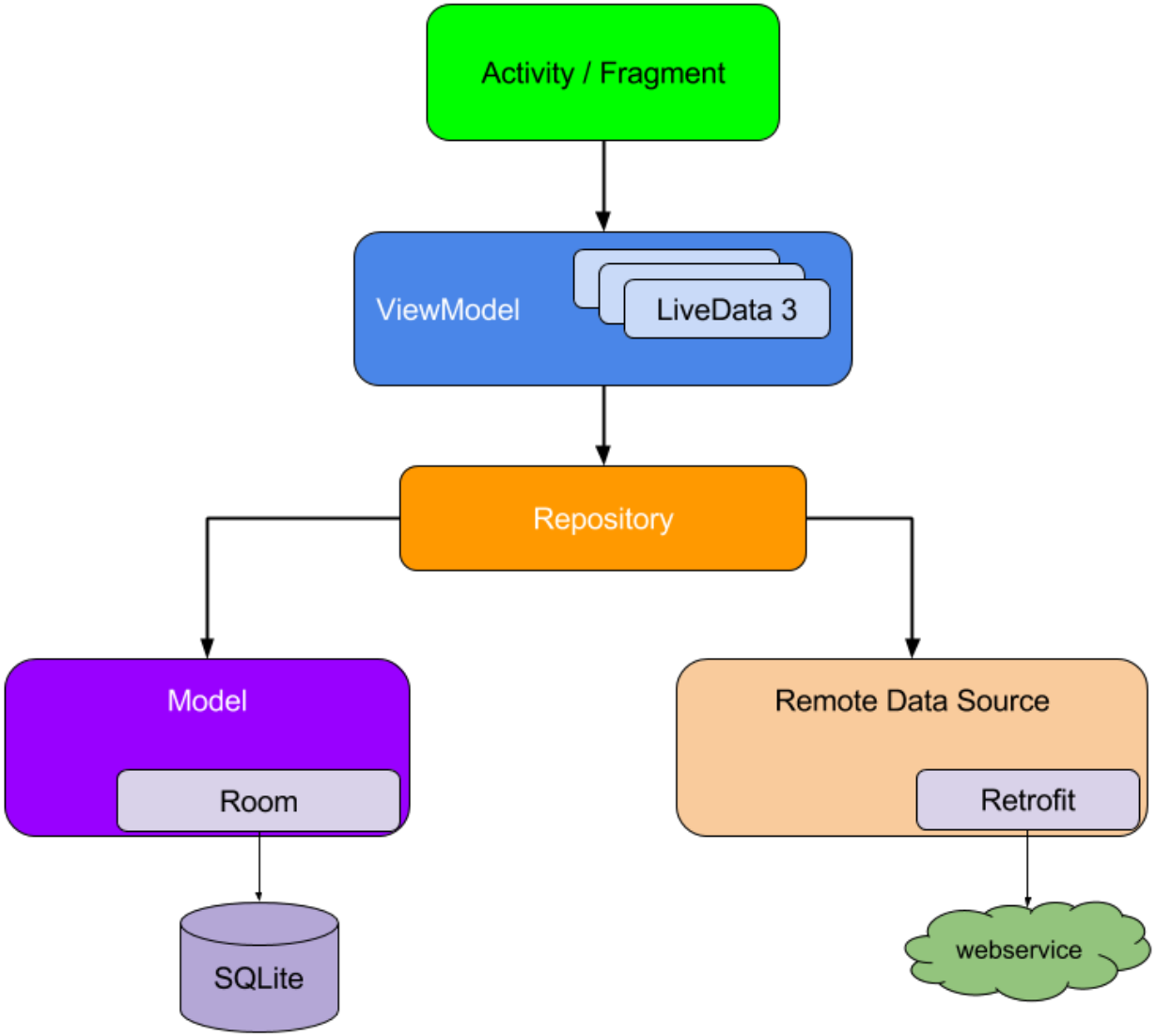
ViewModel and LiveData: Example 33

- Our code works pretty well!
- The data lives in a different place from the Activity, and is therefore safe.
- From the software engineering perspective, it’s still not perfect.
- The issue is that the ViewModel is fetching data from the network.
- While we’ve decoupled the data from the UI, it’s still coupled strongly to our ViewModel.
- We’d have to rewrite ViewModel if we wanted to fetch data from a database instead.

Repository Design Pattern

- Since the ViewModel is a data store for the view, it shouldn't really know whether the data is coming from the internet or a local database.
- The repository design pattern exists to solve this problem.
- So, the solution is that the ViewModel should get the data from a **repository**.
- The repository itself gets the data from either the internet or a database (or whatever).
- Let's see a sketch.

Repository Design Pattern



Repository Design Pattern: Example 34

- Let’s re-architect our app to follow this design pattern.
- The repository class will obtain the data from the network and parse it.
- The ViewModel will pass data onto the repository to help the latter fetch data.
- The ViewModel will be the only class to hold/create instances of the repository.
- The Activity won’t know anything other than the ViewModel.

Summary

- We wrapped up internet connectivity (for the most part).
- We saw how LiveData and ViewModel can work together to achieve lifecycle awareness.
- We also saw how to implement the repository design pattern.
- In the process, we implement our first MVVM architecture- Model, View, View Model.