Finishing up the Delivery Mechanism: App Layer

Kaushal Dhruw
APP DEVELOPER / AUTHOR

@drulabs linkedin.com/in/kaushal-dhruw/

Overview

Consuming the ViewModels

Implementing Dagger2

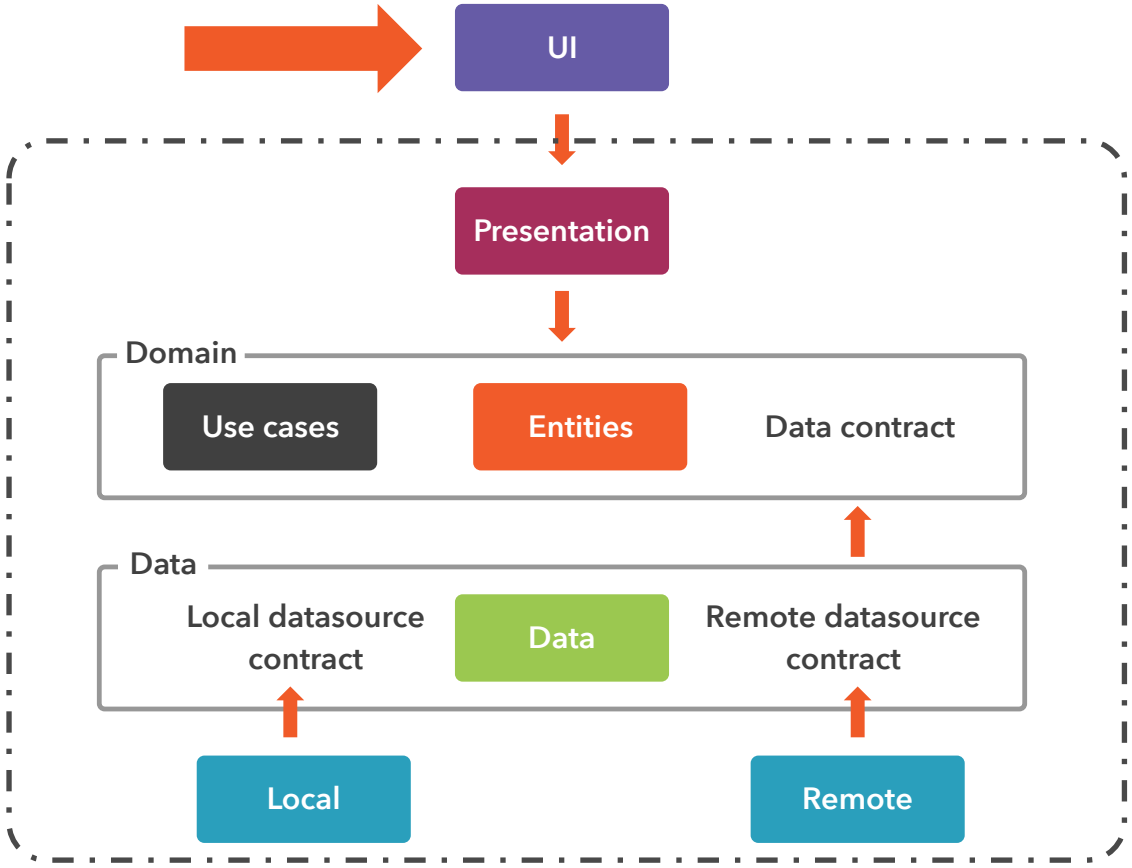
Testing the App

Revisiting CLEAN Architecture

Summary

Consuming the ViewModels

Layers of CLEAN Architecture



Demo

App module dependencies

Structure of the App module

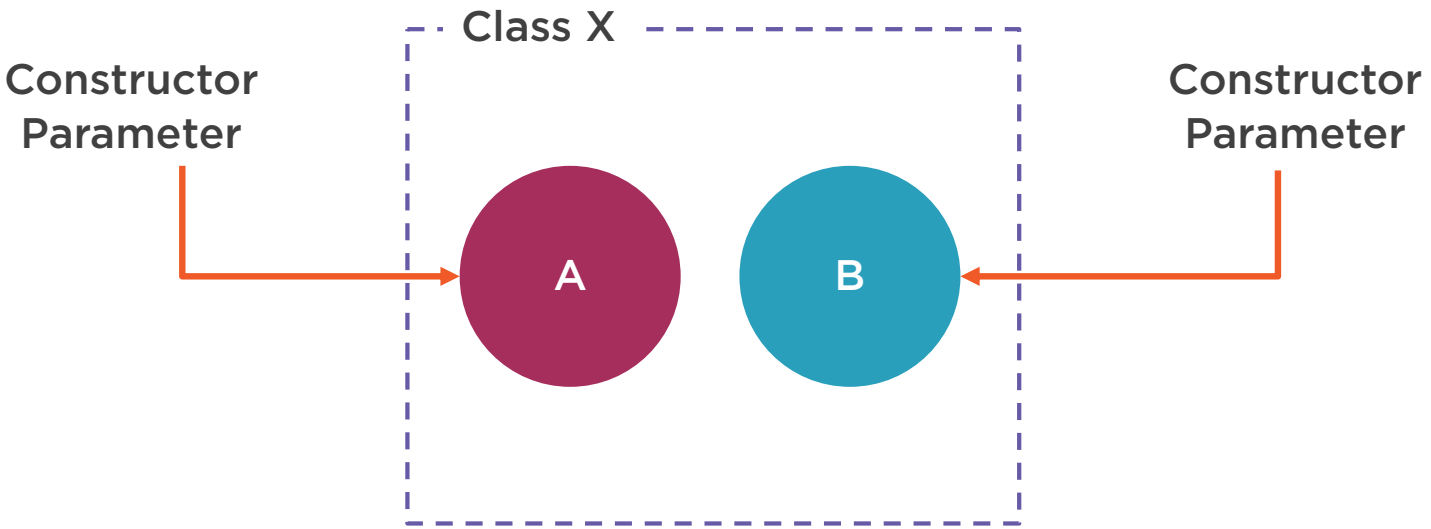
ViewModels consumption in Activities

Rendering the UI

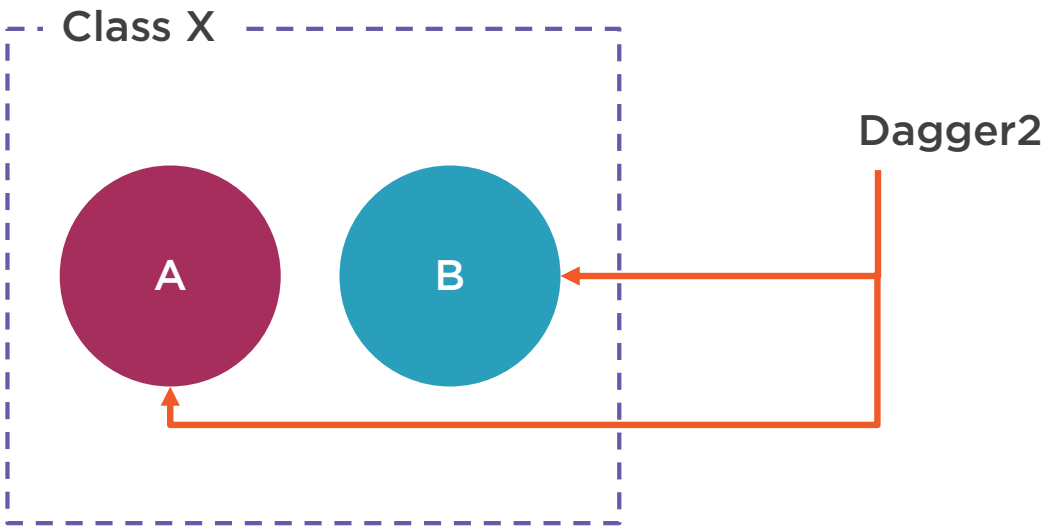
Banking app in action

Implementing Dagger2

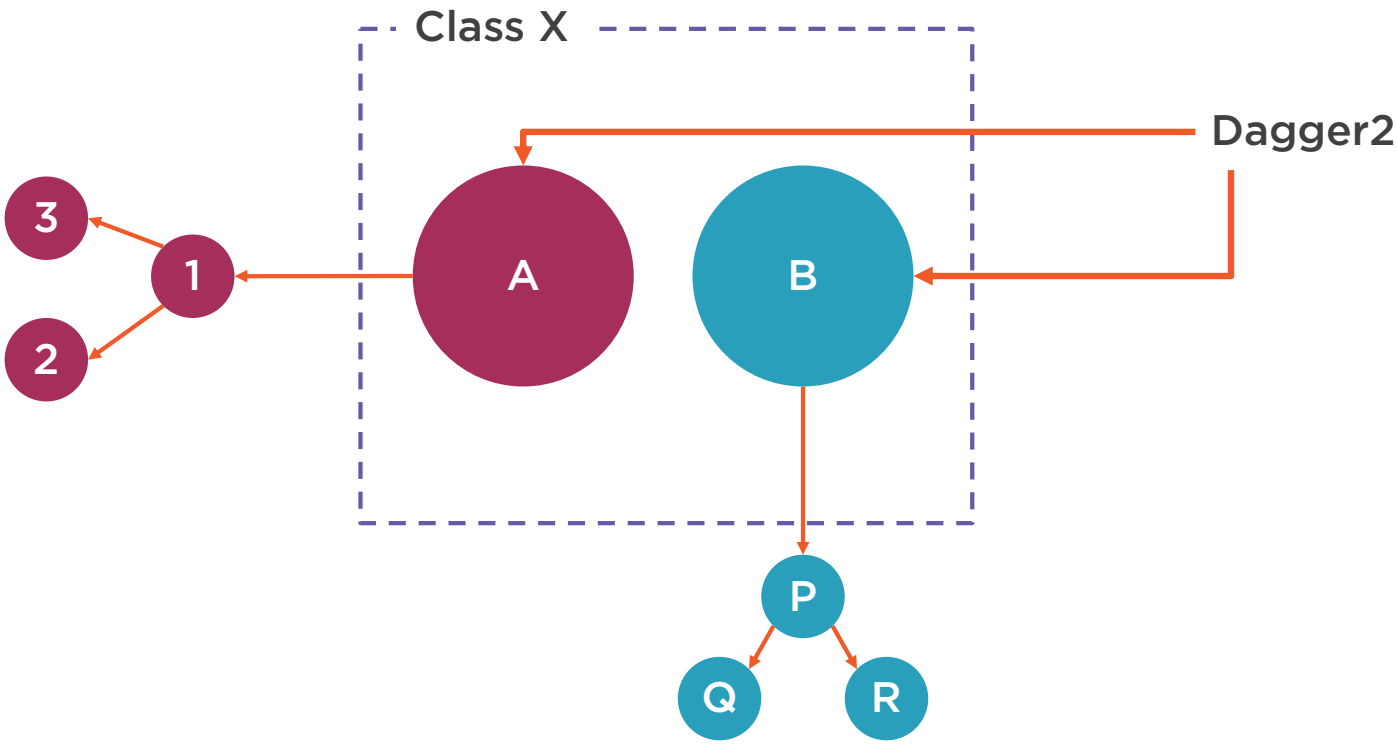
Dependency Injection with Dagger



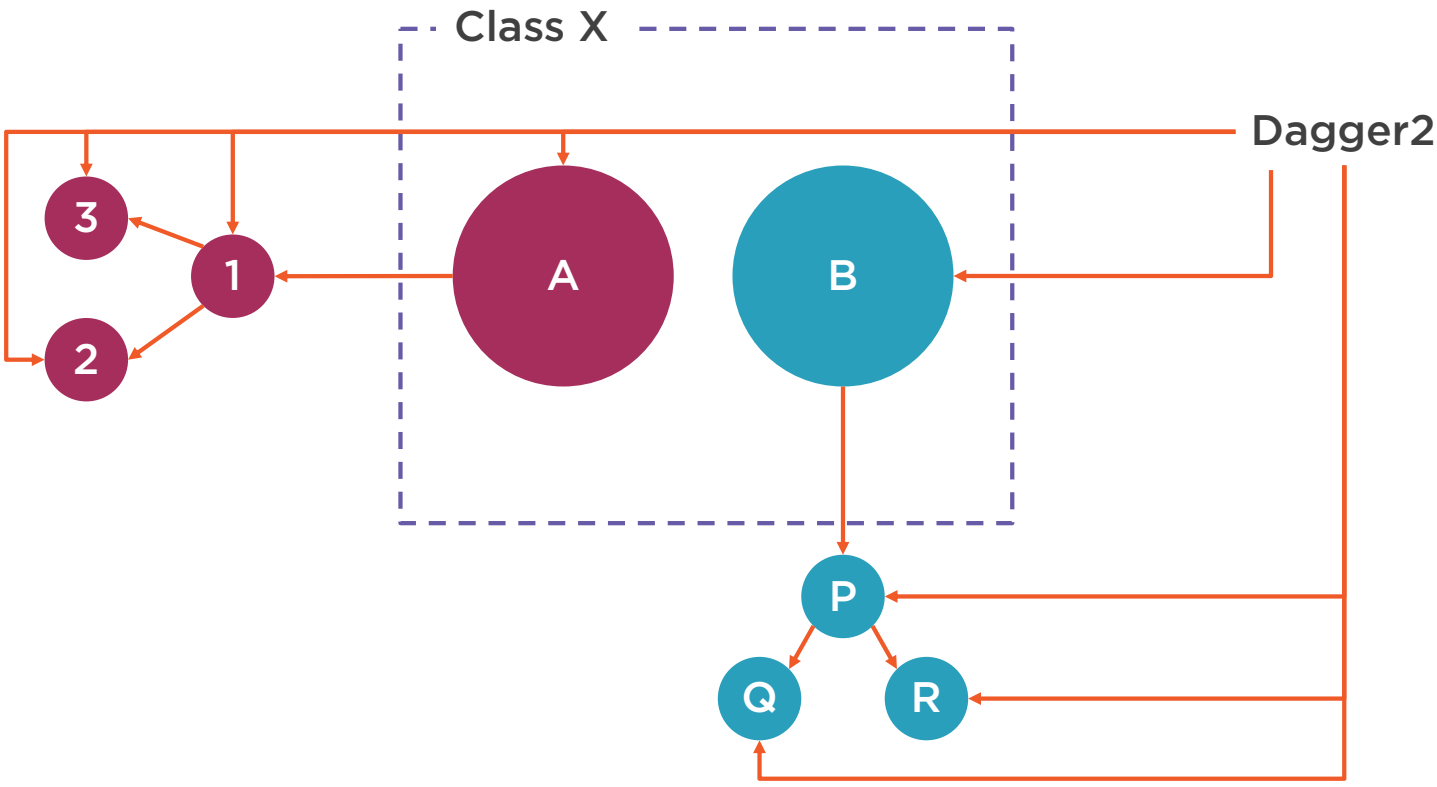
Dependency Injection with Dagger



Dependency Injection with Dagger



Dependency Injection with Dagger



Elements Required for Dagger

@Inject

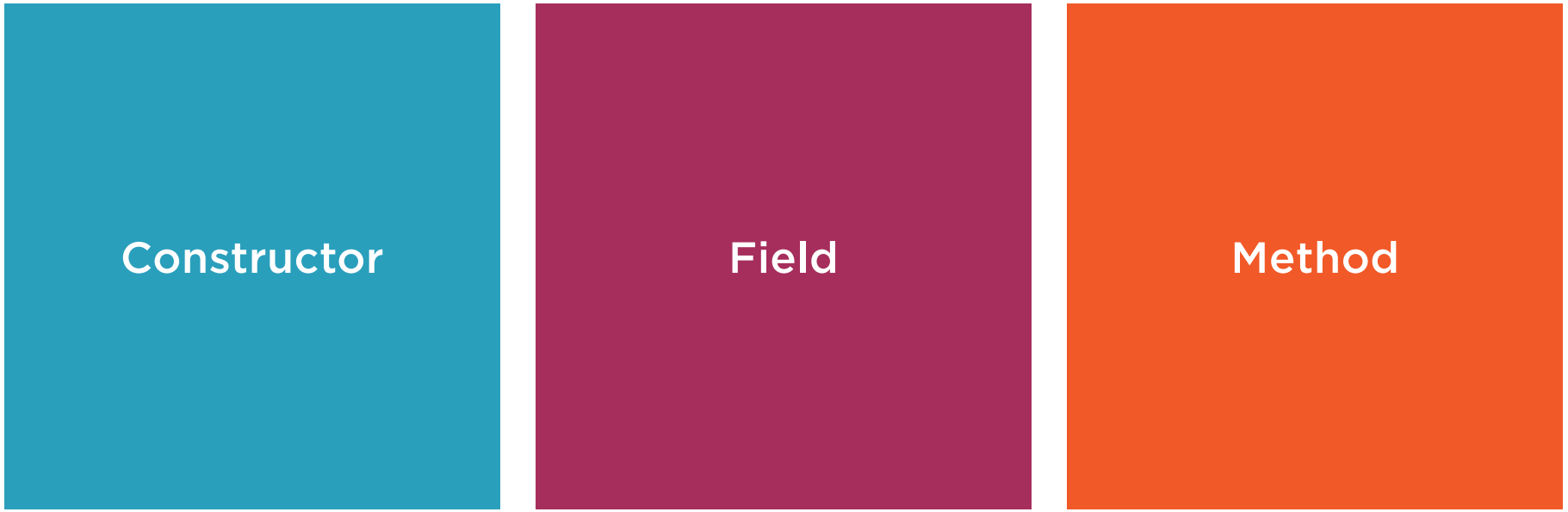
@Module

@Component

@Qualifier

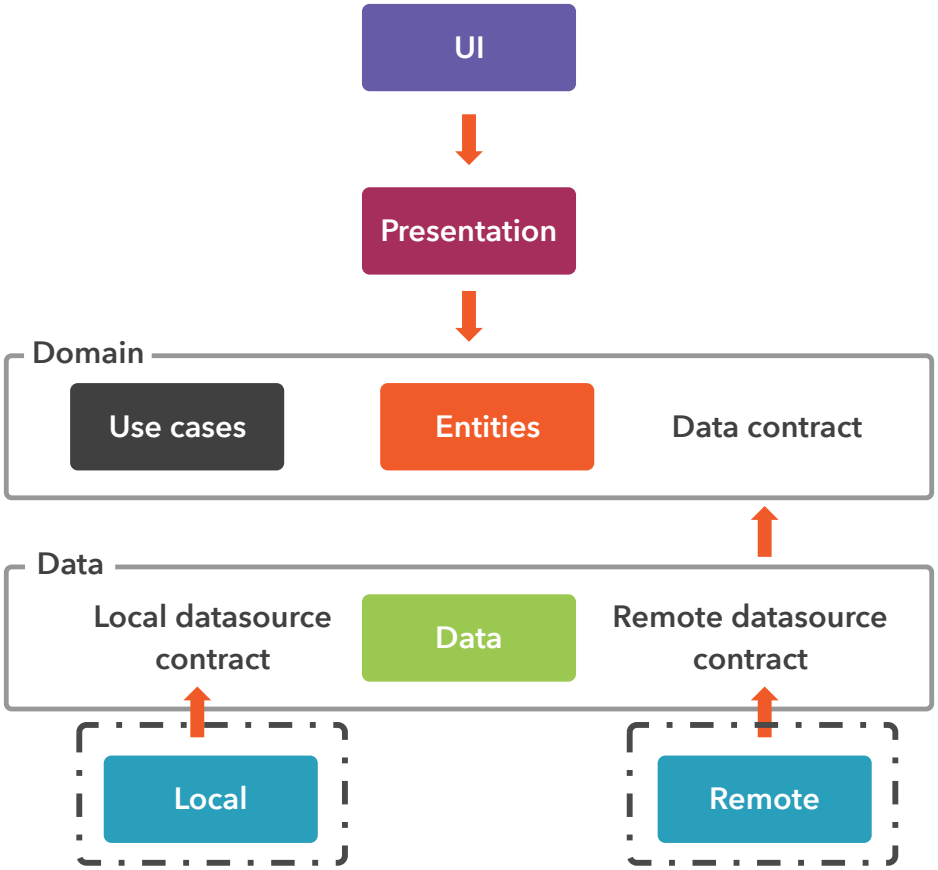
@Singleton

Types of Dependency Injection



Testing the App

Banking App


UI
Presentation
Domain
Use cases
Entities
Data contract
Data
Local datasource contract
Data
Remote datasource contract
Local
Remote

Demo

App layer's instrumentation tests

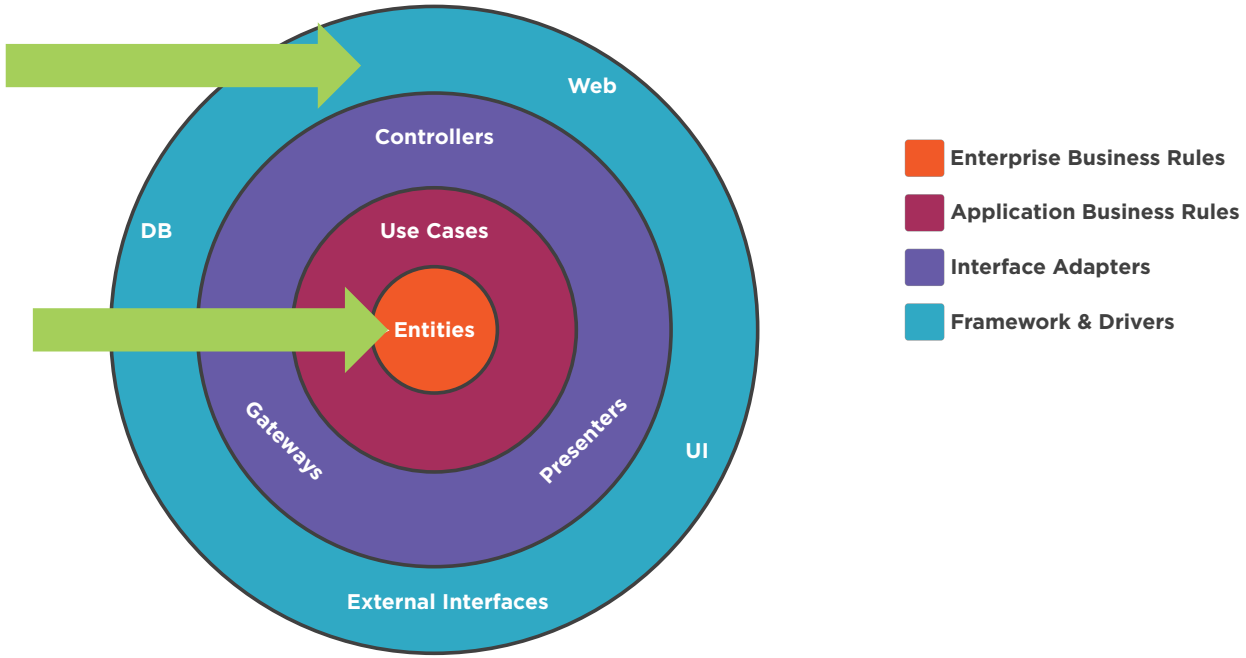
In-memory database for local module

Fake implementation of BankingService

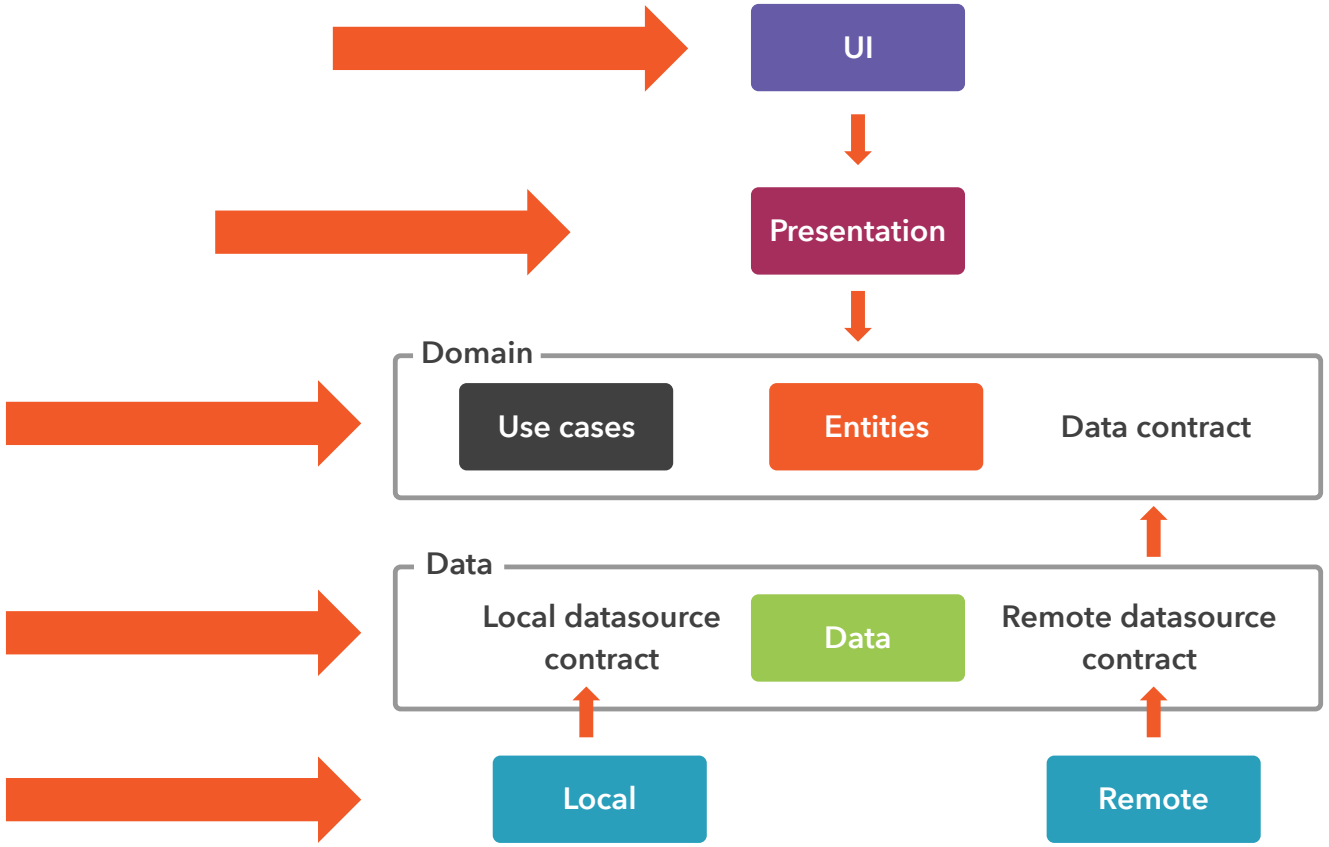
Test execution in emulator

Revisiting the CLEAN Architecture

Layers of CLEAN Architecture



Implementing the CLEAN Architecture



Concerns

Who maintains the view state?

Is there no state maintenance in the banking app?

Why the remote source invoke every time?

Why the frameworks are easily switchable?

Is there a feature related partitioning?

Summary

- Importance of architecture and testing
- CLEAN Architecture concept and implementation
- RxJava
- Room, ViewModel, LiveData - AAC
- Dagger2 dependency injection

Thank You

Kaushal Dhruw
APP DEVELOPER / AUTHOR

@drulabs linkedin.com/in/kaushal-dhruw/