

OR HOW I WAS REWRITING A  
CONFERENCE APP OVER WEEKENDS

---

KOTLIN MULTIPARTFORM  
ROLLER-COASTER

# CHALLENGE



DEUTSCHE TELEKOM AG

---

MICHAL  
HARAKAL

# ROLLER-COASTER

---

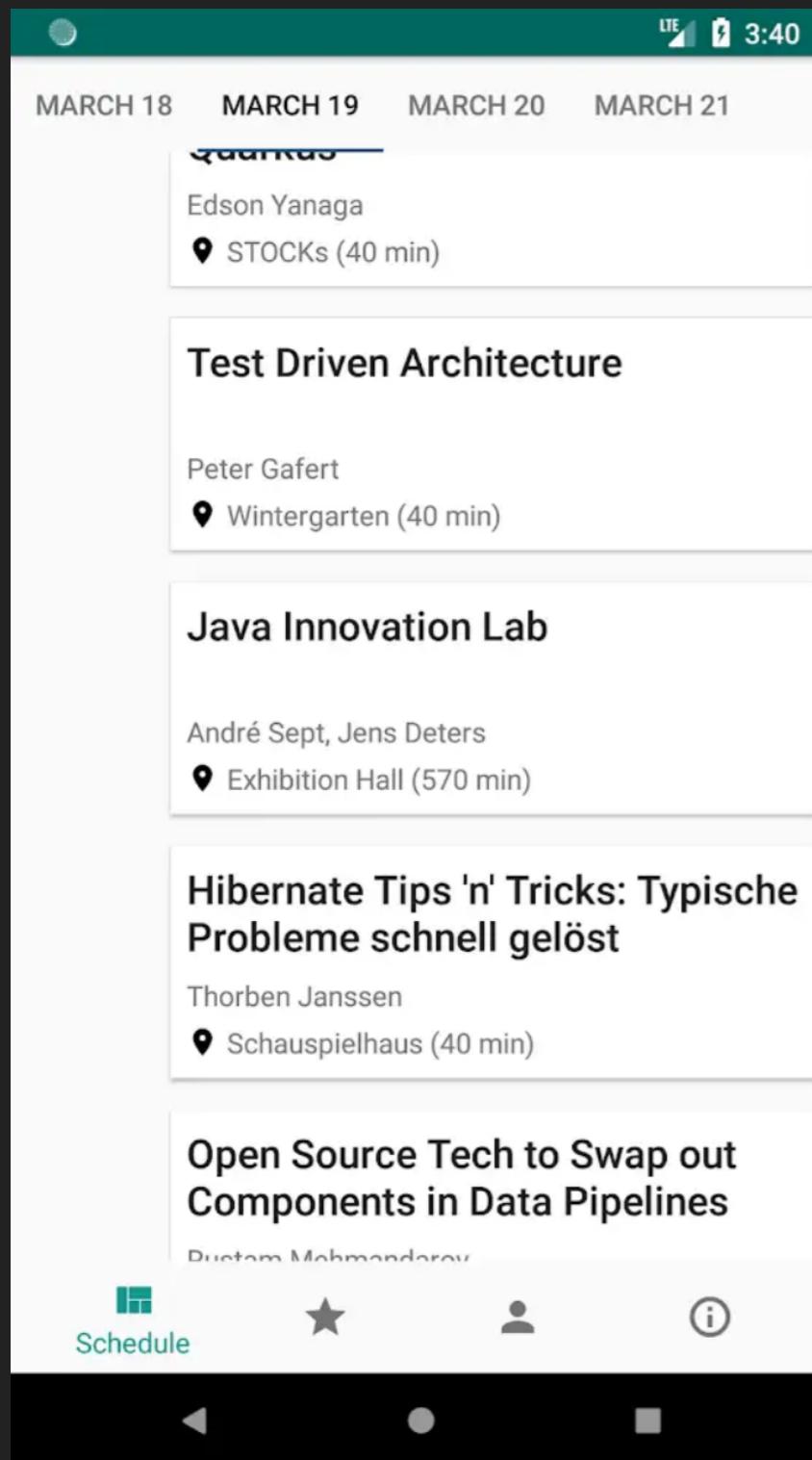
- ▶ an exciting entertainment in an amusement park, like a fast train that goes up and down very steep slopes and around very sudden bends
  
- ▶ a situation which changes from one extreme to another, or in which a person's feelings change from one extreme to another.



Cambridge Dictionary



# STARTING POSITION - JAVALAND NATIVE ANDROID CLIENT



- ▶ written 100% in Kotlin
- ▶ clean architecture approach with building blocks as separated gradle modules
- ▶ coroutines
- ▶ MVP for presentation layer
- ▶ as far possible, java only, to get rid of Android whenever you can
- ▶ finest selection of Android libs(okhttp, retrofit, picasso, gson)

---

# DUKECON.ORG



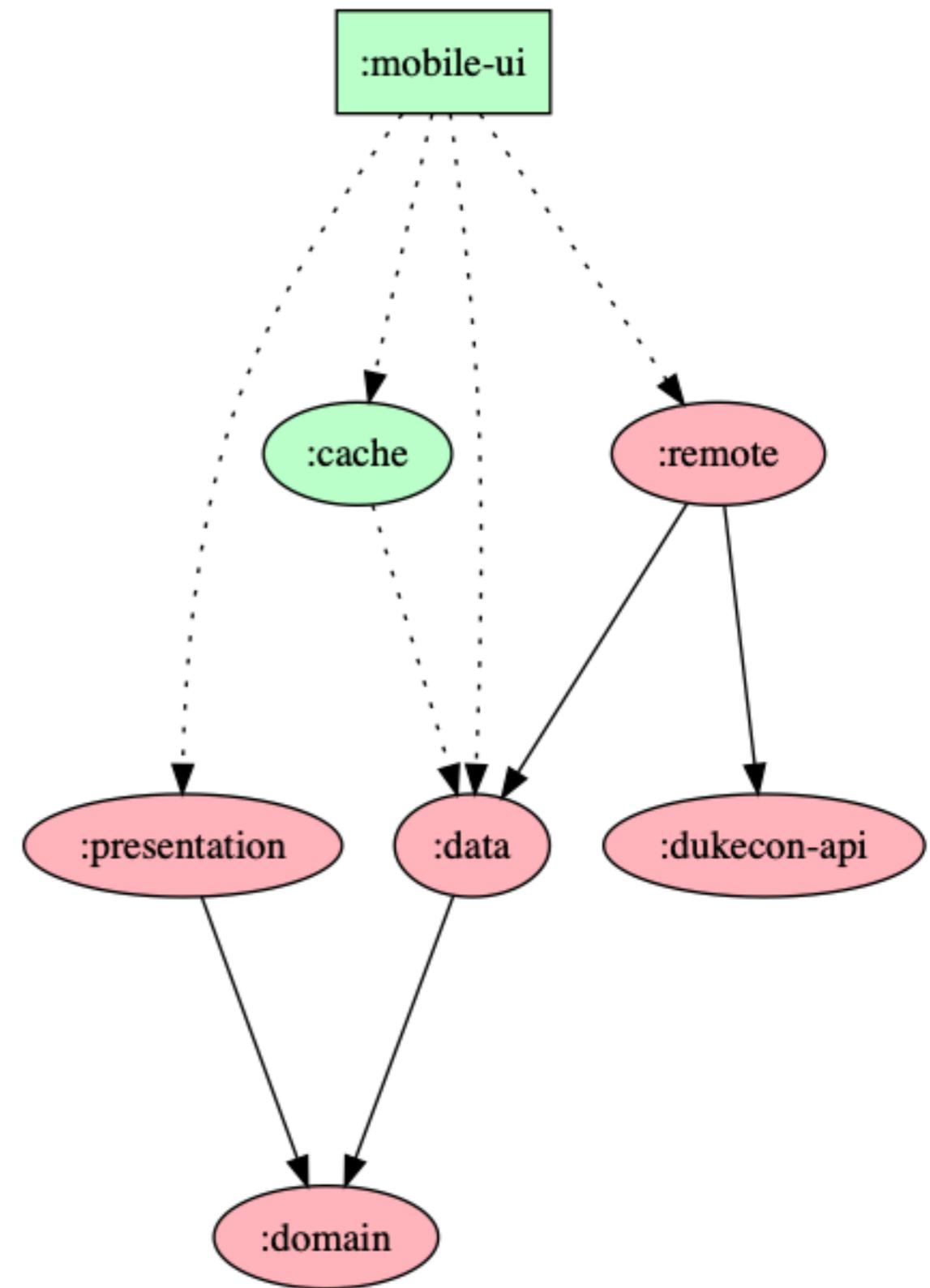
**DOAG**

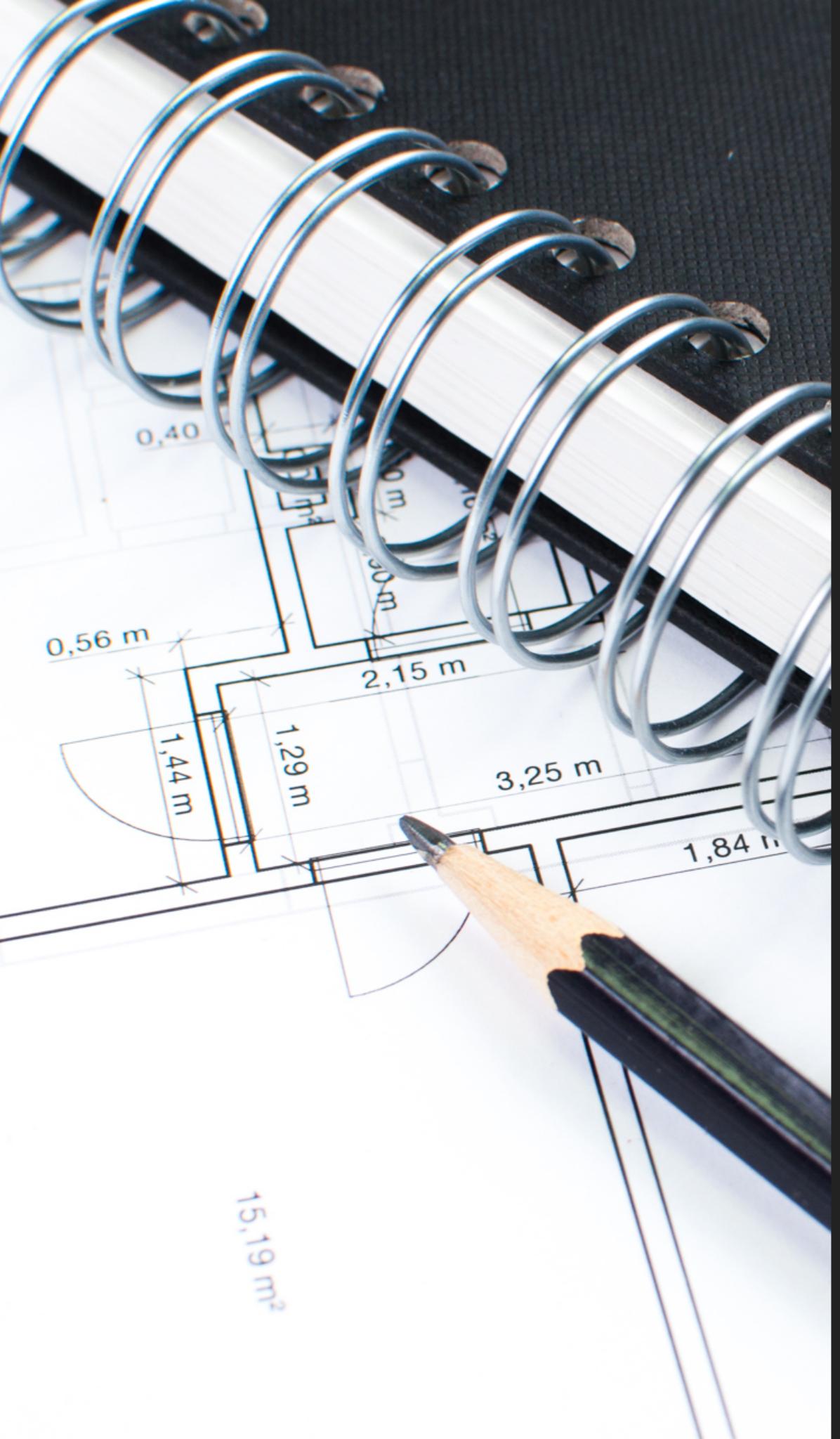
**APEX** connect

**APACHECON**

## INITIAL MODULES STRUCTURE

- ▶ only **cache** and **app** are Android specific
- ▶ java only modules
- ▶ **dukecon-api** - Retrofit client and DTOs generated completely with swagger





PLAN

---

# PLAN

- ▶ Fokus on Android first
- ▶ convert java modules to Kotlin Multiplatform
- ▶ migrate platform dependent to Kotlin Multiplatform  
**(remote, cache)**
- ▶ use **Kotlinkonf** app for iOS

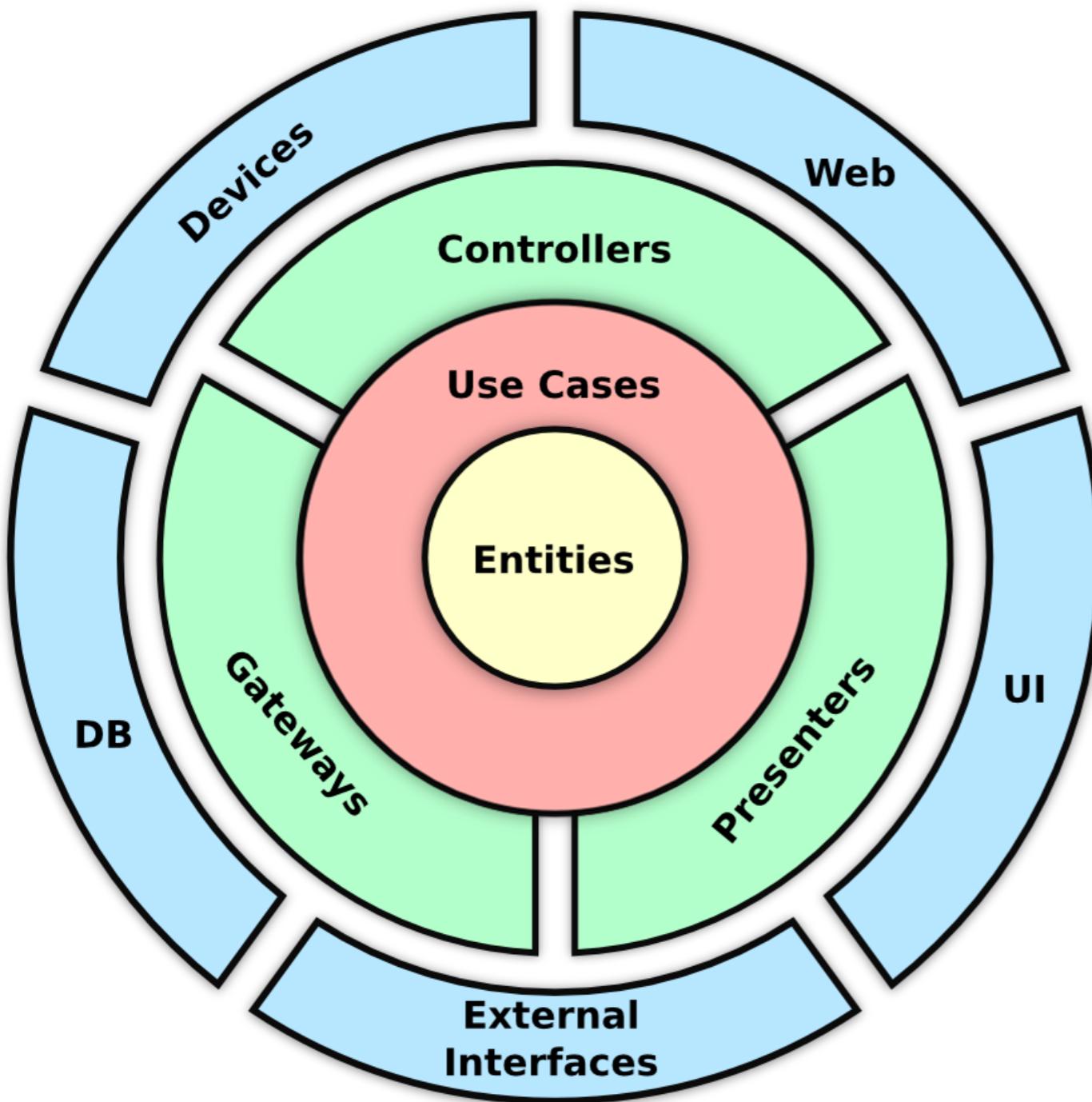
**STANDING ON THE  
SHOULDERS OF GIANTS**

---

# STANDING ON THE SHOULDERS OF GIANTS

- ▶ Chicago Roboto
- ▶ Kotlin clean architecture starter by **Joe Birch**
- ▶ Dukecon.org
- ▶ **Kotlinkonf**
- ▶ SDKSearch by **Jake Wharton**

# CLEAN ARCHITECTURE



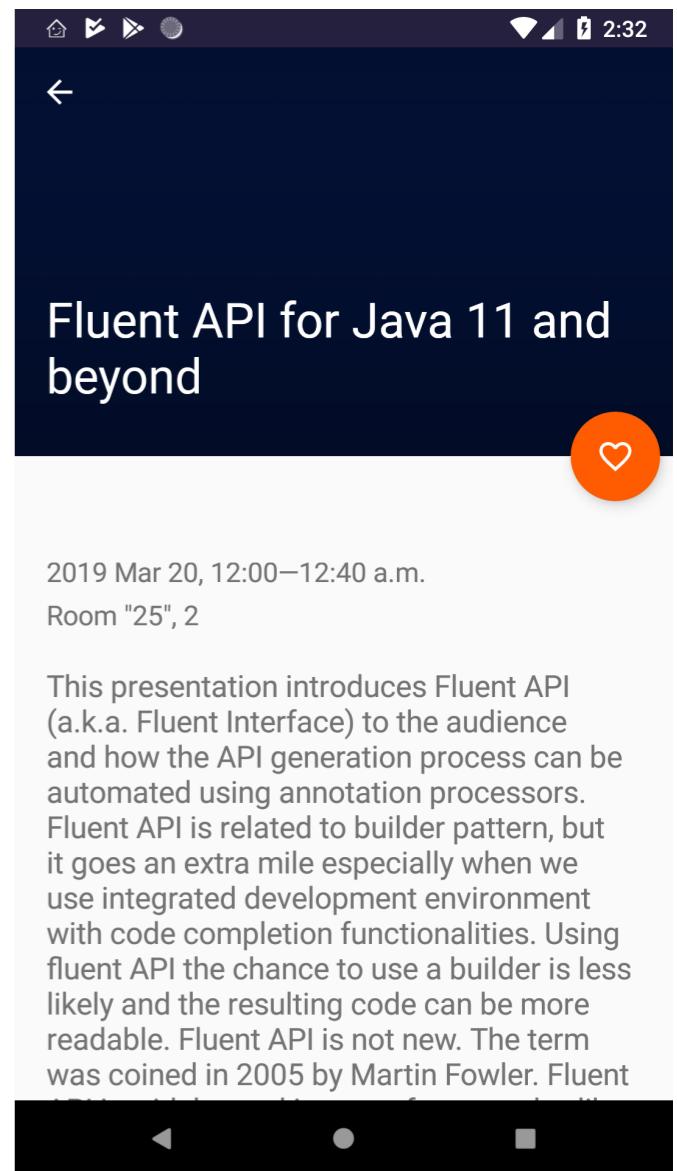
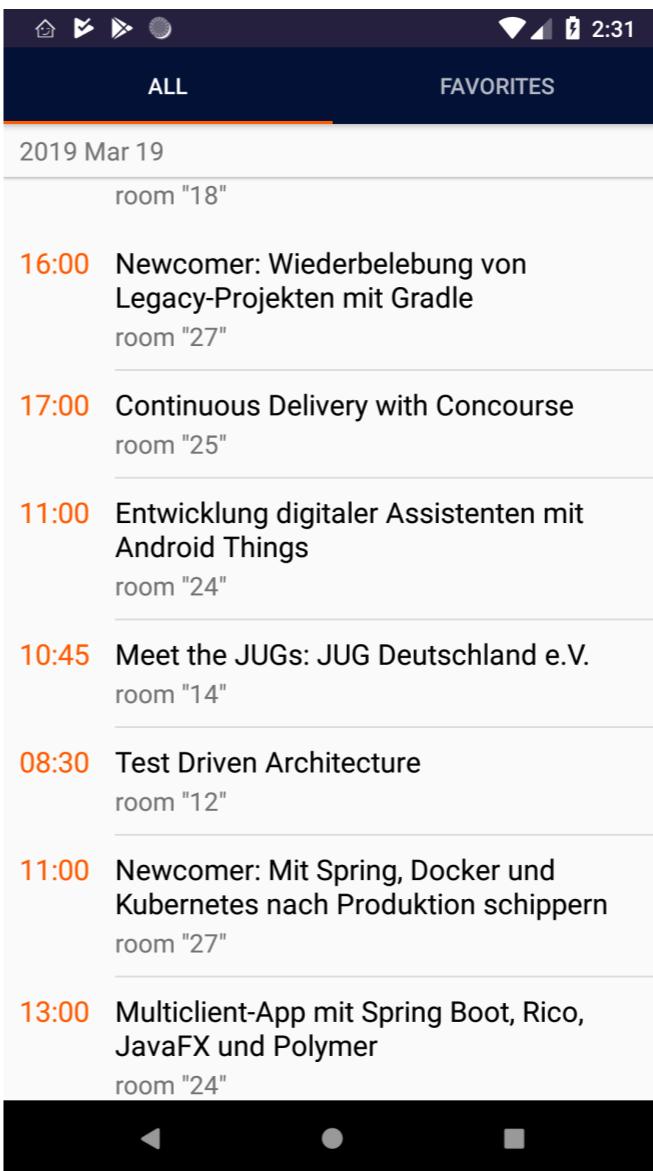
- Enterprise Business Rules
- Application Business Rules
- Interface Adapters
- Frameworks & Drivers

- ▶ layered
- ▶ dependencies points inwards
- ▶ keeps Android dependencies on outside

# CHALLENGE DIARY

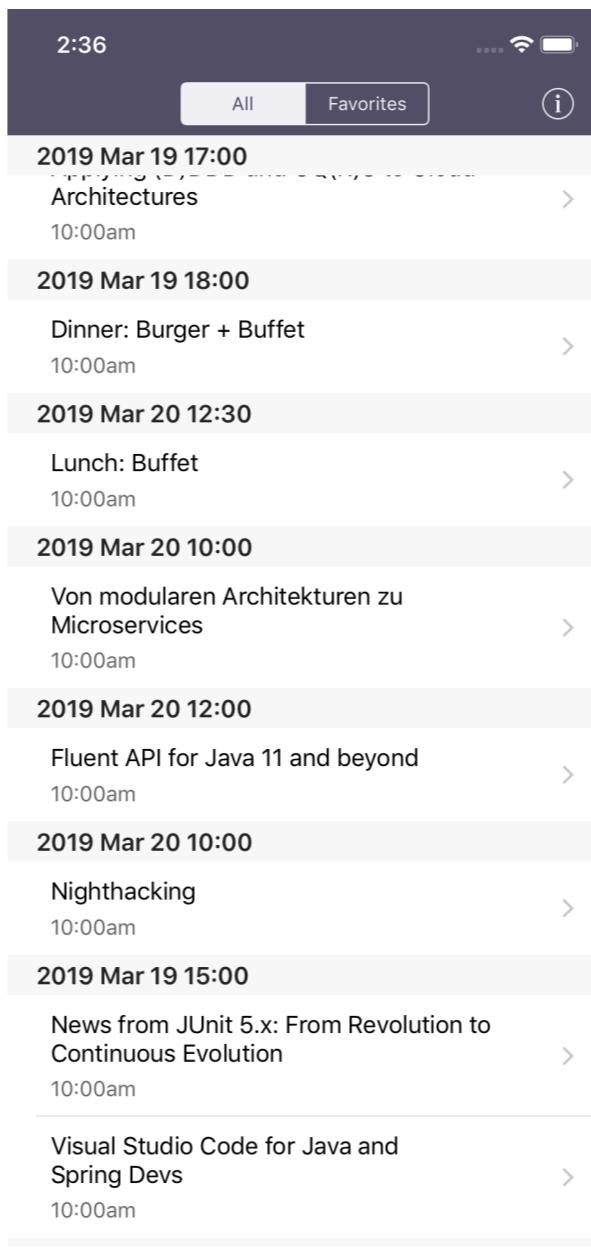
# WEEKEND 1 - KOTLINKONF SPEAKS DUKECON

- ▶ swagger generated remote client and data object
- ▶ Android running



# WEEKEND 1 - IOS SPEAKS DUKECON

- ▶ how to debug?
- ▶ iOS specifics  
wisdom required
- ▶ gradle vs. build.sh
- ▶ AppCode



This screenshot shows a detailed view of a session in the DukeCon app. The top bar indicates the time is 2:37. The left side has a back arrow labeled 'Sessions'. The right side shows the title 'Session' and a star icon. The speaker information is listed as 'Speaker Dave Ford'. The session title is '**Fluent API for Java 11 and beyond**'. The date and time are '2019 Mar 20, 12:00—12:40 a.m.' and the room is '25, 2'. A descriptive text block provides information about Fluent API, mentioning its introduction by Martin Fowler in 2005, its use in frameworks like Mockito and JOOQ, and its relation to builder patterns and code completion. Another text block discusses the new features in Java 8, 9, 10, and 11 related to fluent API. At the bottom, there is a rating section with three icons: a smiley face, a neutral face, and a sad face, with the text 'Tap to Rate:'.

Speaker  
Dave Ford

**Fluent API for Java 11 and beyond**

2019 Mar 20, 12:00—12:40 a.m.  
25, 2

This presentation introduces Fluent API (a.k.a. Fluent Interface) to the audience and how the API generation process can be automated using annotation processors. Fluent API is related to builder pattern, but it goes an extra mile especially when we use integrated development environment with code completion functionalities. Using fluent API the chance to use a builder is less likely and the resulting code can be more readable. Fluent API is not new. The term was coined in 2005 by Martin Fowler. Fluent API is widely used in many frameworks, like Mockito or JOOQ.

The new thing that this presentation focuses on how we can create fluent API using the features available in Java 8 (default methods and lambda), Java 9 (private methods in interfaces), Java 10 (var declarations) and Java 11 extending the var declarations to lambda expressions and JVM support for nested classes.

Tap to Rate:

Smiley face icon

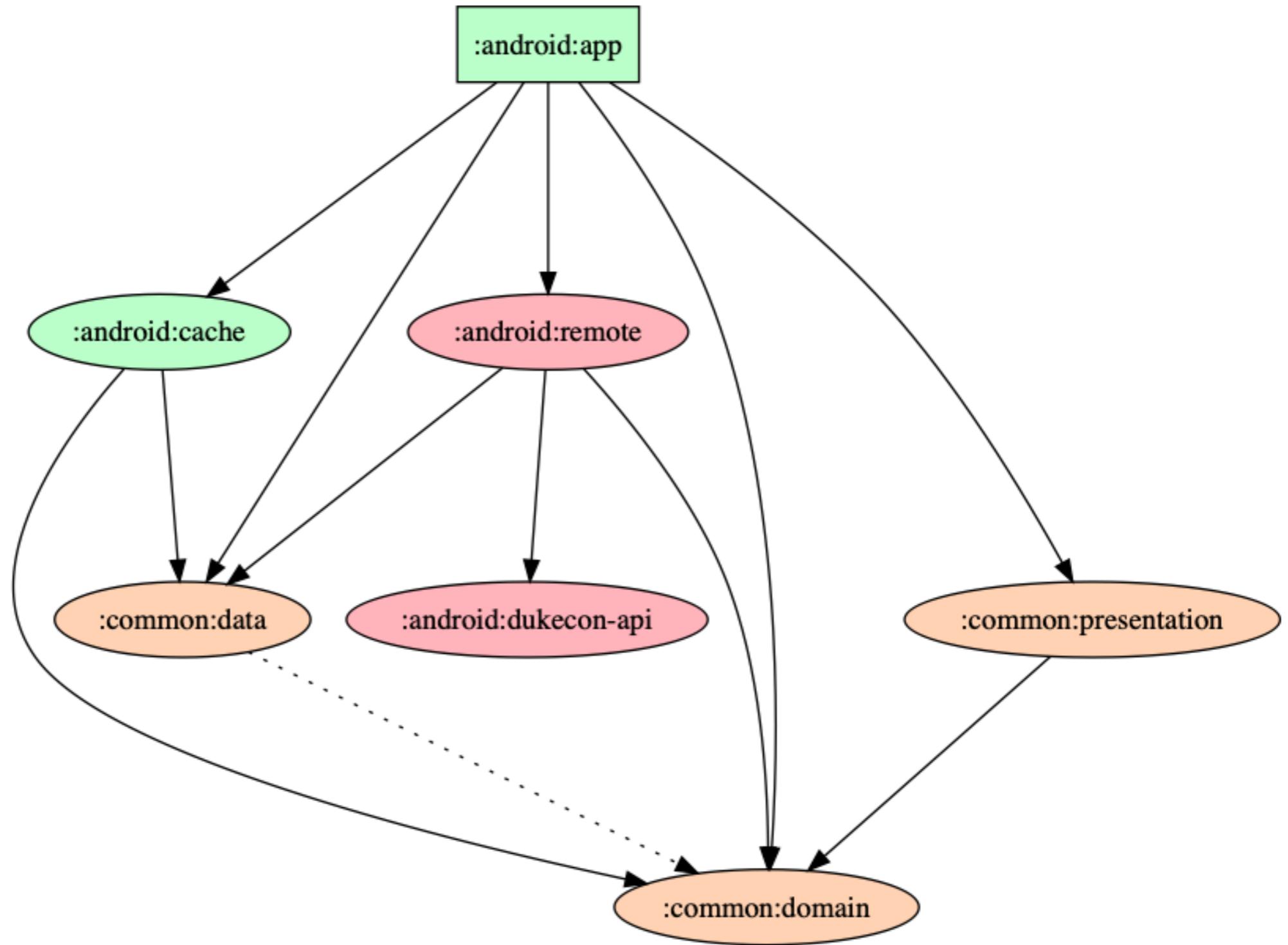
Neutral face icon

Sad face icon

## WEEKEND 2 - COMMON

- ▶ moving java only, platform independent classes into Kotlin MPP
- ▶ removing code related to a *network* awareness
- ▶ removing **@Inject** annotations

dukecon-mobile-mpp



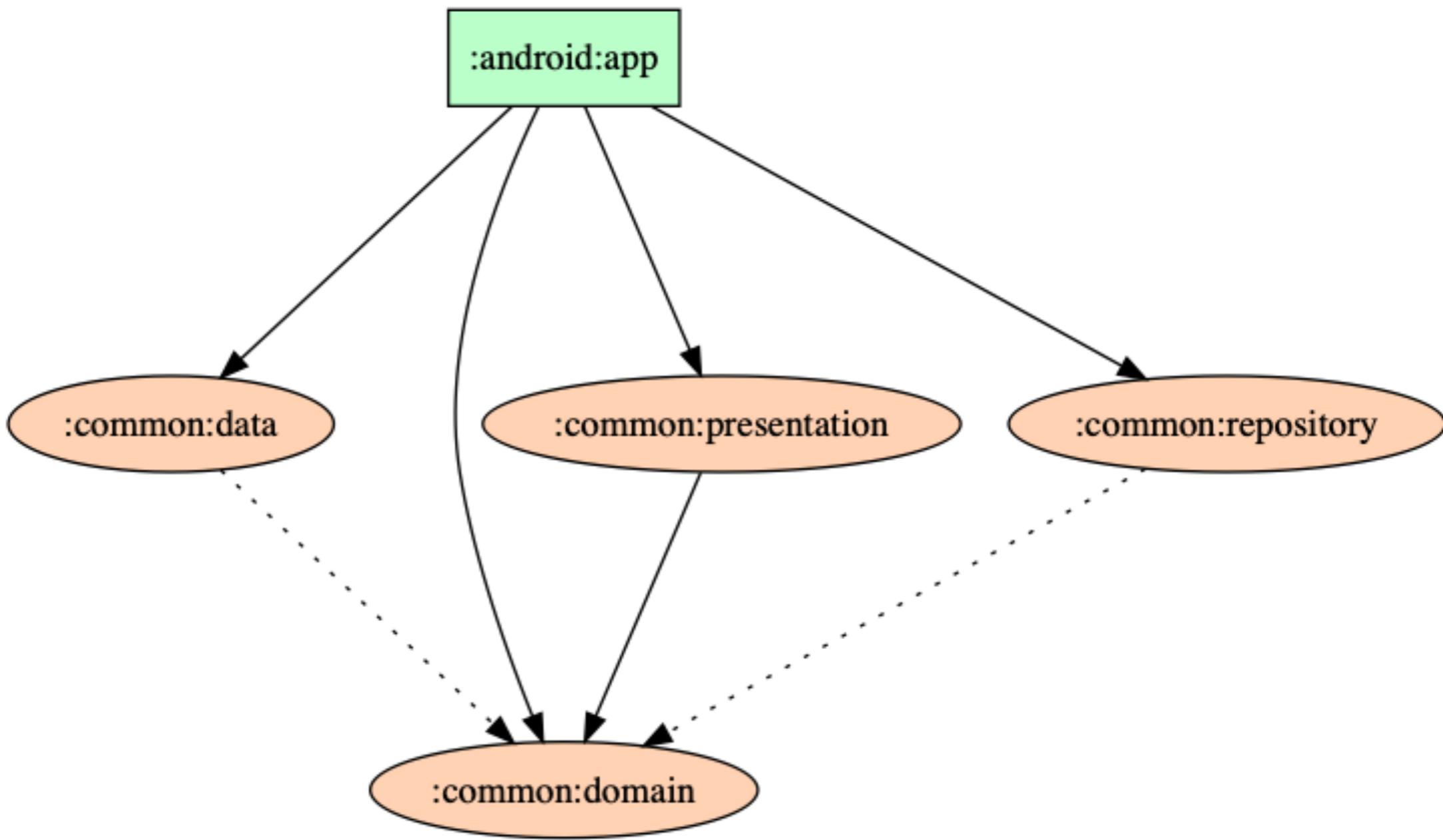
---

## WEEKEND 3 - FINAL WEEKEND

- ▶ Android App runs with Kotlin MPP Module
- ▶ still issues with Date Time
- ▶ original heavy usage of constructor injected parameters has to be replaced by manual calls in Dagger module classes

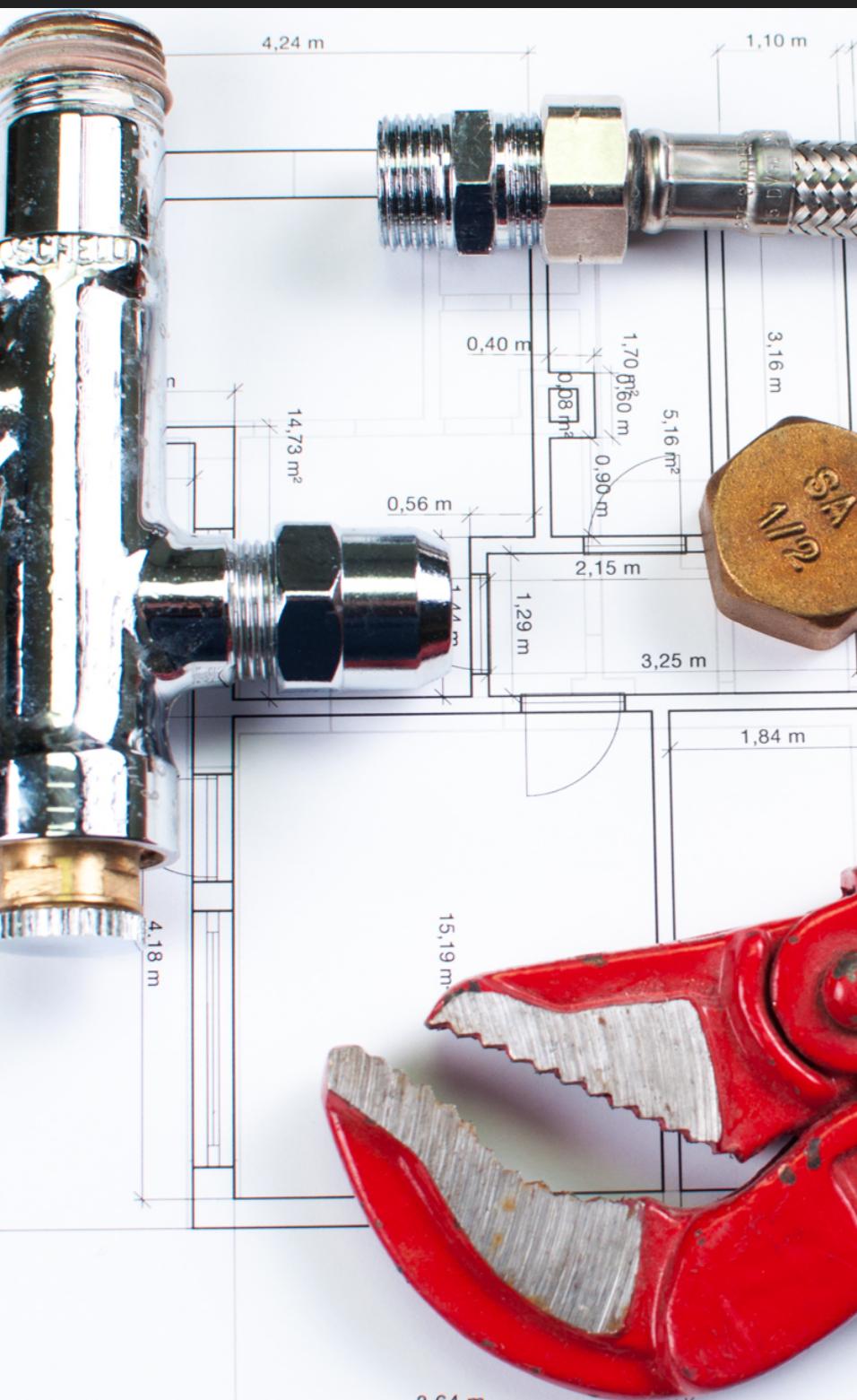
## WEEKEND 3

# dukecon-mobile-mpp



# TASK TO SOLVE

# PLUMBING



- ▶ Gradle multiplatform modules
  - ▶ common
  - ▶ android
  - ▶ iOS
- ▶ Dependency management
  - ▶ Spring Dependency Management
  - ▶ support for BOM

# NETWORKING OR KTOR FUN



IntelliJ **Ktor** plugin generates basic project structure, API service and DTO classes for serialisation

manual changes required for DTOs  
(@Optional)

hard to debug

logging not working

**Okhttp** by default, SSL not working  
on 4.4

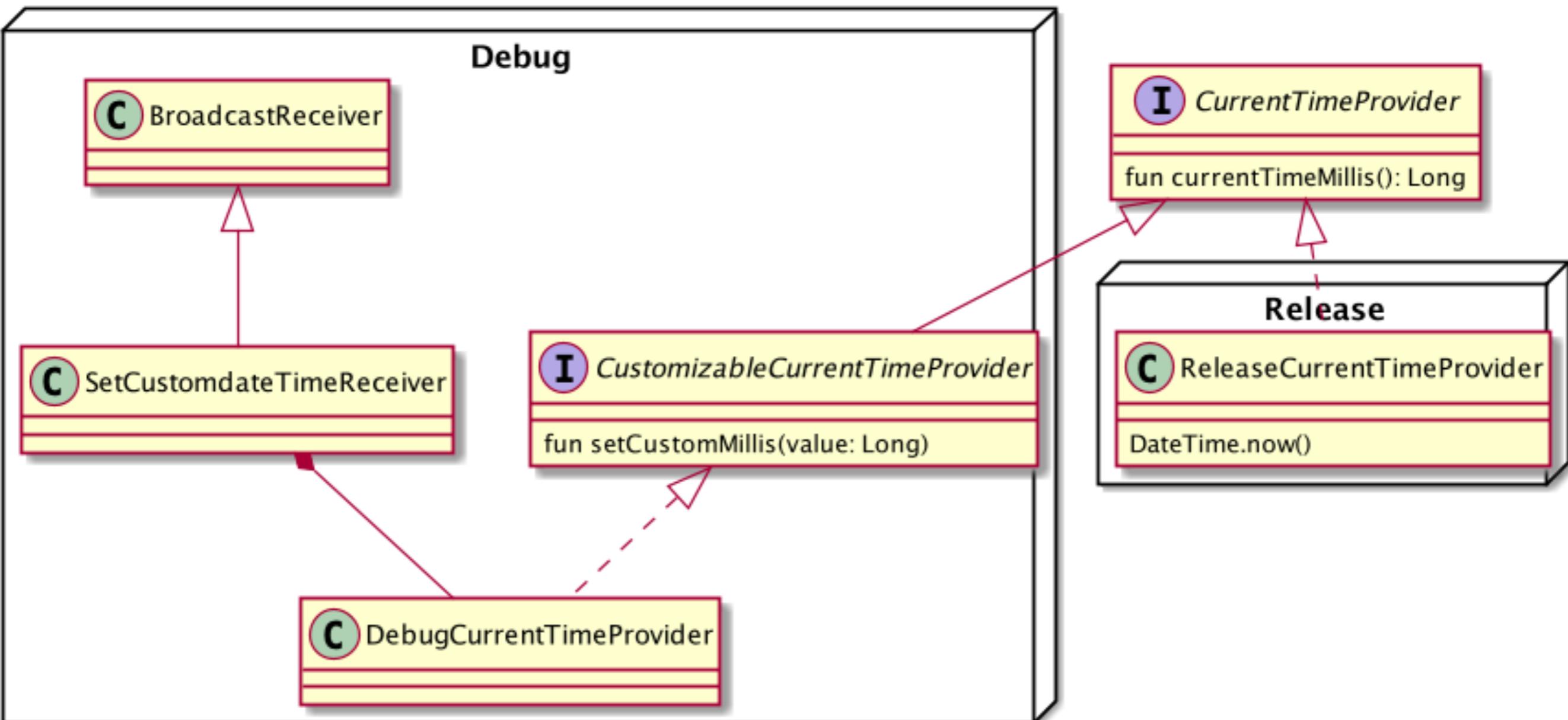
# DATE AND TIME



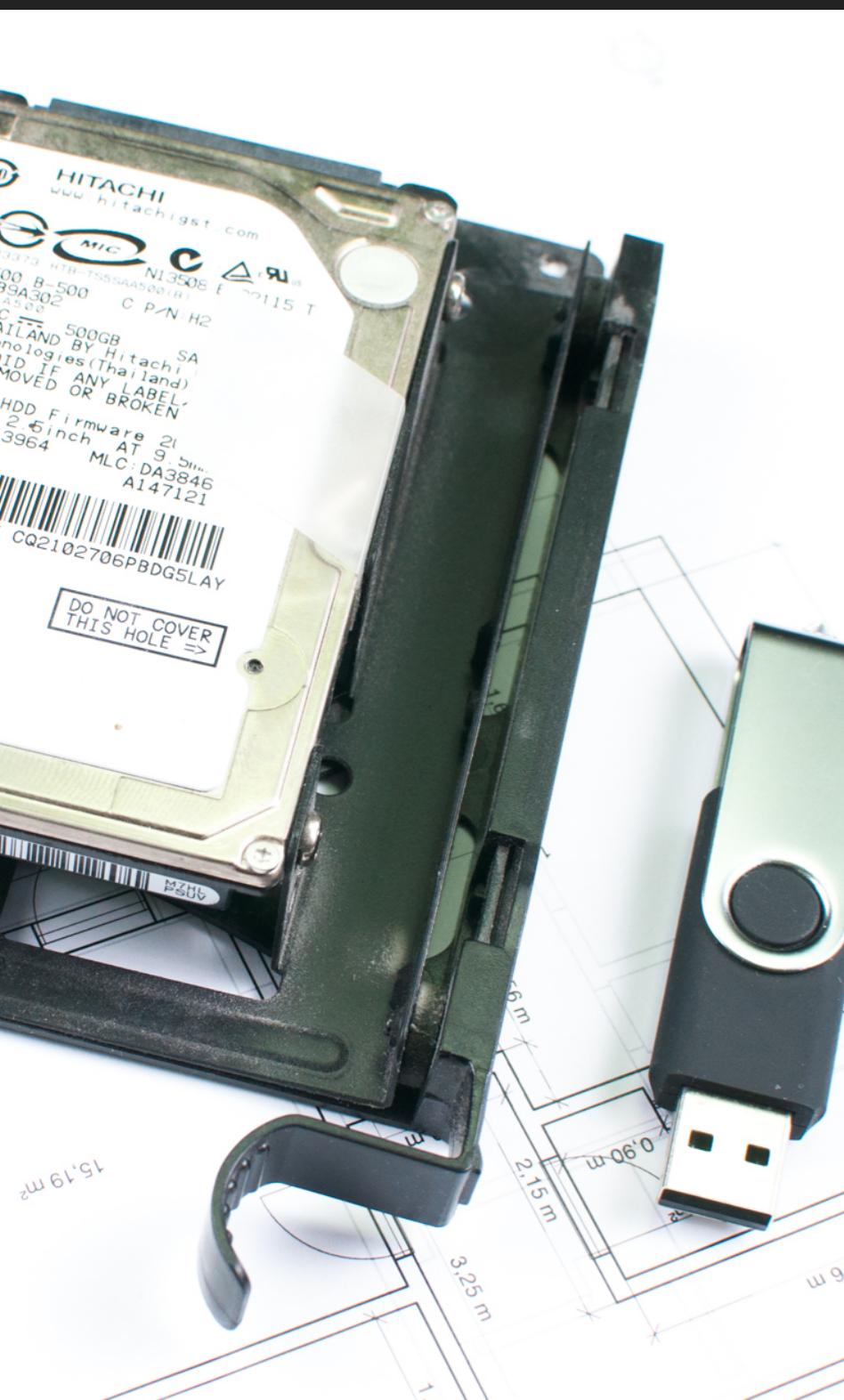
- ▶ **JodaTime** originally
- ▶ Migrated to **ThreeTen** later
- ▶ **GMTDate**
- ▶ *TimeProvider* for testing

```
$ adb shell am broadcast -a  
org.dukecon.android.ui.intent.TIME --es set_time "12:33:00"
```

# CURRENT TIME PROVIDER

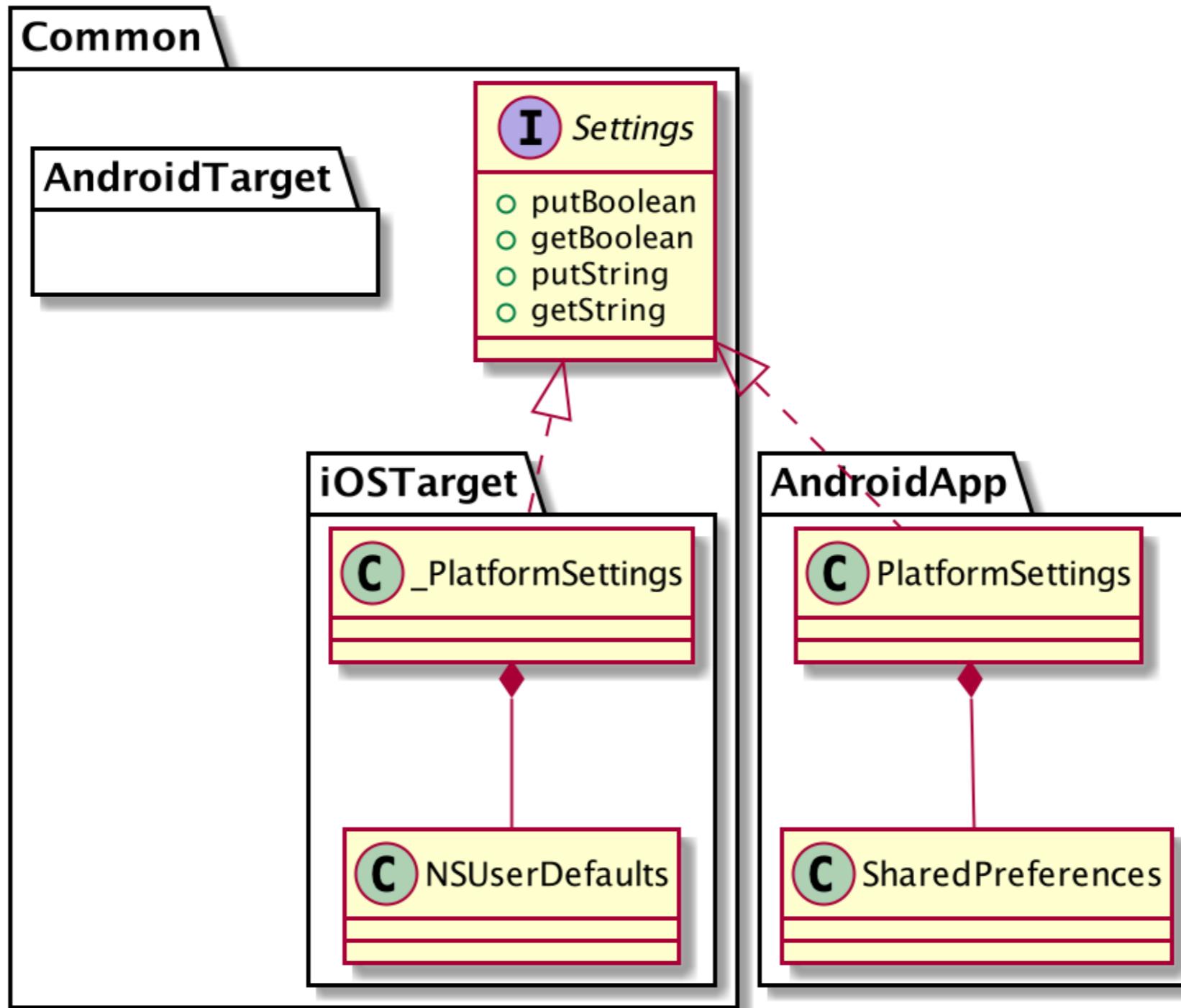


# LOCAL CACHE



- ▶ Ktor serialised gson objects
- ▶ shared preferences on Android
- ▶ NSUserDefaults

# SETTINGS



# IOS

- ▶ added support into AppCode 2019.1 via Plugin
- ▶ Xcode Kotlin Multiplatform by Touchlab



# RESULT

---

## LESSONS LEARNED

- ▶ Dagger
- ▶ iOS support
- ▶ Software architecture
- ▶ bringing Kotlin Multiplatform into real life projects, code sharing with iOS is the key

---

## DOWNS

- ▶ MPP project is hard to setup
  - ▶ RTFM
  - ▶ clear cache
  - ▶ restart IDE
- ▶ iOS support
  - ▶ no debugging
- ▶ DI

---

## UPS

- ▶ Android Prototype is running
- ▶ Promising current events on iOS
- ▶ Chance to use it in real projects already now
- ▶ Kotlin is just cool