Write an Android library

by Romain Rochegude

Introduction

Multiple purposes

- Database
- Networking
- JSON
- UI
- etc.

Multiple types

- Helper (ex.: retrofit, jackson, ButterKnife)
- Structural (ex. : mosby, Android Architecture Components)
- Complete feature, customizable with theme (ex.: ZXing, Android DirectoryChooser)
- UI (custom views or animations, ex.: MPAndroidChart)

Multiple implementations

- Pure code (classes and API, ex.: mosby)
- Annotation processing at compile time (ex.: ButterKnife)
- Annotation processing at runtime and dynamic proxy (ex.: retrofit)

1. Design it

1.1. Global approach

- Modeling with PlantUML
- Write immutable objects (because Objects Should Be Immutable)
    - https://github.com/google/auto/tree/master/value
- Failure strategy: fail safe vs. fail fast
    - Fail safe with resilience (recover, retry)
    - Fail fast with preconditions
- Lazy evaluation (native in Kotlin)

1.2. Reactive programming

- RxJava and RxAndroid

  > “ RxJava - Reactive Extensions for the JVM - a library for composing asynchronous and event-based programs using observable sequences for the Java VM. ”

  > “ RxAndroid - RxJava bindings for Android ”

- Observables, subscribers
- Asynchronous programming (schedulers)
- Functional operators

Benefits

- Responsive
- Resilient
- Message-driven

1.3. Annotations and compile-time processing

- Java module containing annotation(s)
- Java module containing processor
- Android application module to demonstrate it

Useful libraries

- JavaPoet (and now KotlinPoet)

> A Java API for generating .java source files.

- AutoService

> A configuration/metadata generator for java.util.ServiceLoader-style service providers

- Compile Testing

> Testing tools for javac and annotation processors

2. Check it

2.1. Testing strategy

- Fluent assertions (ex.: AssertJ, truth)
- BDD frameworks (ex.: JGiven, Cucumber)
- Code coverage and mutation testing (ex.: Zester)

2.2. Static analysis

- Sonar
- Lint
- FindBugs
- PMD/CPD
- Error Prone
- Android support annotations

3. Ship it

3.1. Extra information

- Documenting using Markdown (`README.md`)
- Generate Javadoc
- Provide a demo application

3.2. Publication/distribution

- The raw way: `svn externals`, `libs/*.jar`, `libs/*.aar`
- The modern way: upload files to a repository
  - Private repository (ex.: Nexus)
  - Public repository (ex.: JCenter)
  - Use of Gradle tasks (generate JARs/javadoc, sign, upload)

Conclusion

- Follow OOP principles
- Enjoy the ecosystem (RxJava, APT, libraries, etc.)
- Provide a robust set of tests...
- ...and a clear API/Javadoc and/or manual and/or demo application
- Automate whatever is possible with Gradle

Addendum: some helpful libraries

- https://github.com/android10/arrow *(Optional, Preconditions, etc.)*
- http://www.pojomatic.org/
- http://www.vavr.io/ *(Lazy, Option, Try, etc.)*
- https://github.com/jhalterman/failsafe
- and so on:
  - https://github.com/cxxr/better-java
  - https://github.com/KotlinBy/awesome-kotlin

Bibliography (1/2)

- http://www.yegor256.com/elegant-objects.html
- http://www.yegor256.com/2015/08/25/fail-fast.html
- http://liviutudor.com/2012/06/06/simplify-your-singletons/
- http://www.vogella.com/tutorials/RxJava/article.html
- http://www.slideshare.net/allegrotech/rxjava-introduction-context

Bibliography (2/2)

- http://hannesdorfmann.com/annotation-processing/annotationprocessing101
- https://jobs.zalando.com/tech/blog/zester-mutation-testing/
- https://www.virag.si/2015/01/publishing-gradle-android-library-to-jcenter/
- http://tutos-android-france.com/deployer-librairie-jcenter/
- http://slides.com/pivovarit/javaslang-functional-java-done-right#/

Logo credits

Social Media graphic by pixel_perfect from Flaticon is licensed under CC BY 3.0. Check out the new logo that I created on LogoMaker.com https://logomakr.com/7eyQap7eyQap