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. Common sense

Follow OOP principles

Write immutable objets

Why? Because Objects Should Be Immutable

- https://github.com/google/auto/tree/master/value
- https://github.com/gabrielittner/auto-value-with
- https://immutables.github.io/

Don't use `NULL` references

[Why NULL is Bad?]

- Use of Optional?
  - http://fernandocejas.com/2016/02/20/how-to-use-optional-on-android-and-java/
  - http://blog.jhades.org/java-8-how-to-use-optional/
  - http://www.vavr.io/vavr-docs/#_option
- or create new object instead of returning `null`

Lazy

- http://www.vavr.io/vavr-docs/#_lazy
- http://liviutudor.com/2012/06/06/simplify-your-singletons/
- Native in Kotlin

Failure strategy (fail fast vs. fail safe)

- Defensive programming
- Fail fast with preconditions
  - https://github.com/android10/arrow
- Fail safe with resilience (recover, retry)
  - http://www.vavr.io/vavr-docs/#_try
  - https://github.com/jhalterman/failsafe
- Need Robust Software? Make It Fragile

2. Improve code quality

Automate what’s possible

- Pojomatic
  - http://www.pojomatic.org/

> “ configurable implementations of the `equals(Object)`, `hashCode()` and `toString()` methods inherited from `java.lang.Object` ”

- Pojo-tester
  - http://www.pojo.pl

> “ test your POJO against `equals`, `hashCode`, `toString`, `getters`, `setters` and even `constructors` ”

Testing strategy

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

Static analysis

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

Embrace Java ecosystem with existing libraries

https://github.com/cxxr/better-java

3. 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

Make RxJava debugging easier

https://github.com/T-Spoon/Traceur

> “ Easier RxJava2 debugging with better stacktraces ”

A pretty good example of a rx-based library

https://github.com/tbruyelle/RxPermissions

And more...

https://android-arsenal.com/tag/38

4. Annotations and compile-time processing

Structure

- 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

Useful resources

- http://hannesdorfmann.com/annotation-processing/annotationprocessing101
- https://github.com/RoRoche/AnnotationProcessorStarter
- More implementation examples...
    - https://android-arsenal.com/tag/166

5. Extended toolkit

- Modeling using PlantUML
- Documenting using Markdown
- Generate Javadoc

6. 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, etc.)
- Provide a robust set of tests...
- ...and a clear JavaDoc and/or manual and/or demo application
- Automate whatever is possible with Gradle

Bibliography

- http://www.yegor256.com/elegant-objects.html
- http://slides.com/pivovarit/javaslang-functional-java-done-right#/
- https://jobs.zalando.com/tech/blog/zester-mutation-testing/
- http://www.slideshare.net/allegrotech/rxjava-introduction-context
- http://www.vogella.com/tutorials/RxJava/article.html
- https://www.virag.si/2015/01/publishing-gradle-android-library-to-jcenter/
- http://tutos-android-france.com/deployer-librairie-jcenter/

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