Advanced Programming

Ricard Solé Casas

February 11, 2018

Foreword

The source code for this report and app can be found on Github[1]. A production version of the application can be installed from Google's Play Store at http://bit.ly/ada-zooify. The minimum Android version required is 5.0 (Lollipop).

Some of the decisions taken in building this application will appear to not follow the suggested guidelines. A detailed explanation is provided in the *design choices* chapter.

Declaration

I confirm that the submitted coursework is my own work and that all material attributed to others (whether published or unpublished) has been clearly identified and fully acknowledged and referred to original sources. I agree that the College has the right to submit my work to the plagiarism detection service. TurnitinUK for originality checks.

[1] https://github.com/rcsole/zoo-coursework

Contents

[2]https://android.com
[3]https://www.wikiwand.com/en/JavaFX

Chapter 1

Design Choices

Some of the choice I have made when building this project deviate a little from the initially suggested approach and can therefore benefit from some extra explanation.

1.1 Android[1], not JavaFX[2]

The design of this assignment seems to be testing more of UI development than it does OOP and other concepts covered in the *Advanced Programming* module. While one lecture did briefly touch on what *JavaFX* is, and how it might be and improvement on Java's Swing[3] the lecture was not nearly in-depth enough to justify using a framework that has been put on maintenance mode[4].

Instead the more sensible choices seemed to use a web-framework with a UI built on HTML, CSS, and JavaScript, such as Spark[5]; or the most popular mobile platform's SDK, *Android*. I would argue an even better choice for the given requirements (building a UI) would have been a 100% TypeScript[6] (or plain *JavaScript*) application using the browser's LevelDB[7], LocalStorage[8], or Firebase[9] as a persistance layer but one of the requirements commanded *Java* as the main language so they were not valid options. My most recient professional experience in *Java* included building *Android* applications so that was, from my point view, the most sensible choice given all the previously stated arguments was to use the *Android Platform*:

1. Existing knowledge
2. Industry standards
3. Assignment requirements

1.2 Pens

The assignment's wording was very vague and it was specially so when referring to the *pens*. I will unpack the text presented in the specification and explain my interpretation, in an attempt to get the reader of this report and myself on the same page:

> *A single pen can only contain animals of the same type.*

[1] https://android.com
[2] https://www.wikiwand.com/en/JavaFX
[3] https://www.wikiwand.com/en/Swing_(Java)
[4] Last version is JavaFX 8, released nearly 4 years ago in March of 2014.
[5] http://sparkjava.com/
[6] https://www.typescriptlang.org/
[7] http://leveldb.org/
[8] https://html.spec.whatwg.org/multipage/webstorage.html
[9] https://firebase.google.com/

It's unclear what the specification refers to here when it comes to *type*. An animal type could be one of the provided *environments* (water, dry, hybrid, and air), or it could also mean a species (Sloth, Cat, Dog, etc.). For this implementation the assumption made is that type refers to *environment*, not *species*. Using the *species* as the type would be a trivial change with the existing implementation.

Each pen has a length, width, and temperature. Aquariums and aviaries also have a height.

There are two things worth noting in this particular part of the specification:

1. There is no need to store the length, the width, or the height. If a pen is dry its are will be land in m^2, if it's an aviary it'll be m^3 in air space, and in water for aquariums. If it's hybrid, to be able to accomodate, for example, a Hippo, the pen will have two measurements, land in m^2, and water in m^3.
2. The temperature is irrelevant with the provided specification. None of the examples provided by the person writing the spec made mention of an animal needing specific temperatures and is therefore irrelevant information that need not be stored. Were this requirement changed, storing temperature data would be a trivial change with this implementation.

Pens including water have a water volume and can be either salt or fresh water.

As with to the temperature data whether it is fresh or salt water is irrelevant. Examples only specify water, dry, petting, air, or part-dry, party-water. Were this requirement changed, storing water type data would be a trivial change with this implementation.

1.3 Species class as opposed to multiple Animal classes

The assignment presented looks like a more lengthy example of the canonical concrete `Dog` class extends from `abstract Animal` class. The difference being the need for a UI and some other entities, like `Keeper`, and `Pen`. If that were the case it would indicate that the coursework designer expected or suggested we create one `class` for each sample animal provided.

While that approach works, I believe it would be severely defficient in a real-world application when managing a zoo. The most obvious short-coming of the (TODO)afformentioned architecture is that any new species registered by the zoo would require actual source code change. Instead I propose this: treat each animal entity as having a relationship to a *species* entity. Akin to how in the game of *Pokémon* one would have a *Pokédex* with all known *Pokémon types* (*species*) and a separate compartment with all the actual Pokémon. One wouldn't create a new `class` for every new *Pokémon type* that appeared. Instead, one would treat each *species* as a simply data. This would allow new discoveries about the *species* to be changed as needed from a UI and without changing the source code itself.

1.4 Keepers don't have a type

The module specification makes each of the initial Keepers be responsible for a given type of pen. I have chosen not to do that. It seems to be a silly choice. To change that it's a trivial modification.

Chapter 2

Overview

In this chapter I will attempt to provide a medium-level overview of the application architecture. I will also mention the few libraries that are used on top of the already existing *Android Platform* framework.

2.1 Libraries

2.1.1 Retrofit/okHttp

2.1.2 Stream Light API

2.1.3 Gson

2.1.4 Guava

2.1.5 Dagger

2.2 Dependency Injection

From Wikipedia:

2.3 Data

2.3.1 Local entities

2.3.2 Local services

2.3.3 Remote services

2.4 UI

2.4.1 Base classes

2.4.2 Main

2.4.3 Create

Chapter 3

Critical Evaluation

Chapter 4

Test Plan

Chapter 5

Appendix A: UML Diagram