Elephant Book Contribution Wiki

Table of contents:

[About this Wiki](#_top)

Android Studio Basics

Unit Testing

**About this Wiki:**

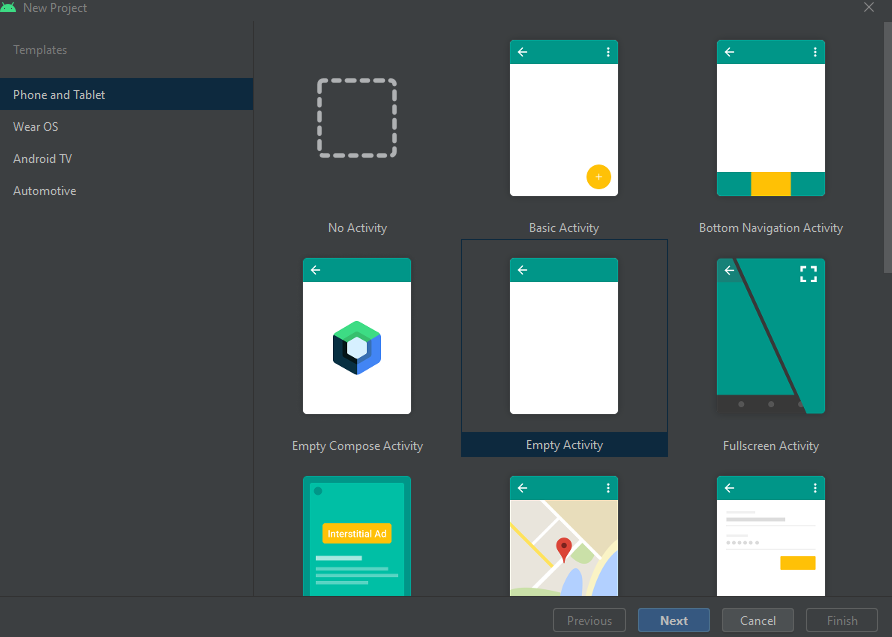
Apparently Android Studio is complicated; while kotlin is easy to learn if you know the standard OOP languages (Python, Java, C++), Android studio has tonnes of annoying behavior that prevents you from doing frank obvious stuff, and thus leads to much energy being expended bashing your head against the android studio brick wall.

My goal for this wiki is for this wiki is that it will document the potholes of the Android Studio road, with the wiki being readable enough so that people actually might read this before learning about the problem

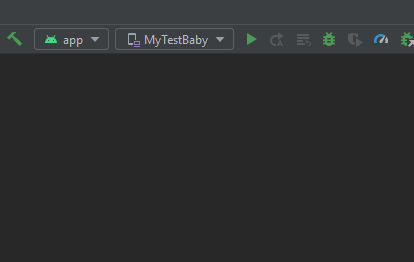
Oh, have solutions to problems, too.

**Android Studio Basics:**

If I remember correctly, when I was first starting out, I wanted to create a dummy Hello World app to get the hang of android studio and Kotlin.



I chose No Activity. From what I remember, No activity did not have enough stuff to run. Normally the way to run the app is to first create a virtual Android Device (Tools 🡺 AVD Manager🡺 Create Virtual Device button is in the bottom left), and then use the play button to run the app.



When I had a No Activity App, the is not part of the top bar of buttons. So I could not run the app.

My hypothesis is that the gradles files were not set up properly to allow the app to be run; the gradles are mysterious, yet important.

When working on an actual project that can, in fact, be run, you will probibally want to sync the gradle files (File🡺Synch project with gradle files)

**Unit Testing:**

Well, you want to start unit testing? Well, that’s quite a challenge for you, and let me tell you why. The following is a copy of my status updates on the subject.

Every time I so much as touch anything with a library object, Android studio rejects that test as illegitimate because it is not mocked. Looking up mocking on the web makes it seem like mocking creates a dummy version of said objects, which seems like not what I want.

There was something I found on one of the android studio testing pages that looked more powerful than mocking, but it failed at the #includes, so while I do not remember what said thing was, I do not remember what said thing is.

I have come to the conclusion (from both android studio documentation and the android developers reddit) that it seems like, given the way mockito works, it would be very hard to mock functions that change the member variables of a library object (like, for example, the JSONObject). This is because when creating a mock of a member function, the when .thenReturn syntax seems to be only for return values, and If I want to create a JSONObject to put into a constructor for one of the Elephant ID’s classes, I care much more about changing the member variables than the

Bonus points to JSONObject.put, which returns an updated JSONObject. But for me to preplan the function return, going to have to have it return a populated JSONObject, which… was the problem in the first place.

Mockito might have also shared Robolectric’s problem of not having all of the files for the testing framework be available through #includes, too. But the above problems still exist.

Copying the code for the JSONObject and putting it somewere nestled within the actual Eliphant ID code also was a dud; The JSONObject.java code found from the android studio navigator only contained function headers, so I would have to write the implementation. Which then would mean testing both the eliphantID Class as well as my implementation of the JSONObject at the same time. Woo.

Plus, it would lead to headaches down the road to use an entirely different “JSONObject” (which only really has “JSONObject”.put implimented) than the true, canon JSONObject.

https://medium.com/@june.pravin/mocking-is-not-practical-use-fakes-e30cc6eaaf4e

So anyways, I give up on unit testing.