Lappeenrannan teknillinen yliopisto

School of Business and Management

Sofware Development Skills

**Arto Savolainen, FITech-student**

**LEARNING DIARY, Software Development Skills: Mobile 2022-23**

**1.6.2023**

I registered my course enrollment in Moodle and read through the general course information. Started work on this diary. Checked out the environment setup tab - I already have VS Code set up with android emulator support (from playing around with Flutter development) and Git Desktop, so was free to skip this part. I then moved on to the Android mobile development tab. Change of plans - apparently these instruction videos use Android Studio, so I suppose I will do so as well. Luckily I already have it installed on my PC.

Downloaded and installed Java runtime and development kit. I've not written Java code since highschool so this should be an interesting course. Hopefully not a very frustrating one.

Created a new project in Android Studio and committed it. Watching the first video and I'm seeing a lot of XML files... not fond of that. Is this going to be a configuration nightmare?

**3.6.2023**

The layout editor is confusing. One can't just put down elements where one wants, they seem to have a life of their own. My first attempt at adding a simple text element below the toolbar in activity\_main has failed. The text element attaches itself on top of the toolbar and refuses to move. Not the start I wanted.

It appears the main activity view in this template is a CoordinatorLayout, which is a "super-powered FrameLayout" (obviously). Looks like this view contains the toolbar and space below it for content that is defined elsewhere. Apparently one is not supposed to just drag elements into such a layout. This template is not ideal for a beginner - there's too much going on here.

I started a fresh project from a simple "Hello world" template. Being prompted to extract hardcoded text into a string resource, I looked up Android naming conventions for files and resources. The Youtube video linked in the course material uses CamelCase for component IDs but googling suggest snake case might be the common way. I chose to use snake case for this project.

Finished the first instructional video and applied some of the techniques to my fledgling project. I now have a view with a couple of TextViews and an understanding of how to programmatically modify their contents. Learned the basics of debugging in Android Studio - it's the same as everywhere else.

I watched a couple more videos about Android development and figured out how to use Intents to start activities, i.e. transition between views in this case. I added a new Settings Views Activity into my project and created a button with a "settings" icon in MainActivity to navigate to it.

**4.6.2023**

Did more work on the settings screen of my app. Read the documentation and found out how to set an EditTextPreference to allow the user to input decimal numbers only. For an EditText this could simply be done with a single line of XML, but an EditTextPreference requires a more cumbersome approach through Java code. I wonder why that is? I could not find a reason for it, but I shan't assume laziness... hah.

Googled around and found that going back to the previous activity, i.e. popping the activity stack is very simple: just call finish(). Added a back button to SettingsActivity which calls finish() on click. Apparently this is called an "Up" button, and could be added to views automatically with an app bar. Considered configuring an app bar and a navigation system but decided against it. The app I'm developing only has three views, both of which are accessed from the main view, so the current method of navigation will suit my purposes perfectly.

I've spent most of this evening trying to figure out Android themes. Working with themes is absolutely terrible. The only way to edit a theme is to go through attributes in an XML file and give them values until you find the one you're looking for. And there are hundreds of attributes. You think iconTint might change the color of an icon in an ImageButton? No it doesn't, and nobody on the entire Internet knows what does. So in practice it is very time-consuming to define styles through themes, and many elements still had to be styled separately simply because I could not find the proper attribute. What I've learned tonight is that the Android theme system is trash, mostly because of scattered / non-existent documentation.

I examined the AndroidX Preference documentation and implemented some further validation for the EditTextPreference inputs on my Settings screen. I was expecting built-in methods for validating user input but the only thing I found was overriding the onPreferenceChange method and attaching that to the preference object. I will send an e-mail to the TA to ask about this problem as I would like to find a more elegant solution.

**5.6.2023**

Today I've been reading the Android developer guide's articles on background work. I need to figure out how to create a background worker that will periodically fetch data from a URL and persist even if the app instance is destroyed - this is to send notifications. For this purpose I've created a DataFetchWorker that is a subclass of Android's Worker class.

HttpUrlConnection looks cumbersome to use so I checked around for alternatives. My search brought me to <https://blog.codavel.com/android-http-libraries-landscape> which is a handy overview of HTTP query libraries for Android. I chose Volley for this since it looks simple enough and is recommended and maintained by Google.

Further examination - Volley works asynchronously which it makes it a bit more complicated to use within a Worker. OkHttp supports synchronous query execution so I will try that instead. A Worker runs in a background thread thus a blocking request is desired there. OkHttp docs had some syntax I'd not seen before - learned about the Java try-with-resources statement.

Preliminary implementation of a data-fetching Worker using OkHttp is functional. To check that the HTML it downloads is as expected I looked up how to write to a file on Android and extracted the .html file with the Android Studio Device File Explorer. The HTML looks correct, I can continue with implementing the actual business logic.

Spent time thinking about state management.. I started implementing my data store as a Singleton - looks like this is a valid and common approach. I researched and tested how arrays, lists, and unmodifiableLists work in Java. When I used the List.of() method to create an unmodifiableList Android Studio suggested I upgrade the "language level" of the project to 9. Is it a bad idea to use Java 9 features? I tested this using a Nexus 6 emulator with an image of Android 7.1.1 targeting API 25. I changed my application's minSdk in build.gradle and voilà it works! I decided to test this further and downloaded Android 5.1 which is the oldest image available in the Android Studio Device Manager. This version is over 8 years old and the app works on it.

Thinking about it, it's no surprise that this particular method would work on older machines, since its functionality requires no changes in the bytecode or the VM itself, just the program compiling Java code into bytecode. But do Android apps run in a Java VM? This prompted me to investigate how Android actually works. Android apps do not in fact run in a standard Java virtual machine; Android versions previous to Lollipop used Dalvik VM, while today Android Runtime is used. ART takes the same bytecode format as Dalvik and compiles it into machine code upon installation. Newer Java APIs are supported on older API versions through a process called "desugaring", where the new libraries are included by the compiler with the rest of the bytecode. I think what happened is that changing the language level to 9 caused Android Studio to include Java 9 features in the app through desugaring. Does this have a significant effect on file sizes? It's late so that investigation will have to wait.

**6.6.2023**

When handling the response from an HTTP request I encountered a warning about potentially producing a NullPointerException. Did some reading about null handling in Java and it looks like using the ternary operator when possible is the least verbose way to implement null checks. Java has also introduced the Optional type, which I already have experience using in Rust. It might be a useful tool for a larger architecture but in this project I will stick with the more traditional method in the interest of time.

When logging HTML retrieved by DataFetchWorker with Log.d() I could not find the string I was looking for using Logcat's find function. Again I wrote the HTML to a file and was able to verify the downloaded data was valid. This enabled me to identify a bug related to parsing said data. It looks like Logcat cuts off lines that are too long - the file was 260k characters spread across only a few lines. Now I need to split a string using a delimiter but escaping characters in strings is inconsistent in Java and will throw up errors with perfectly logical escape sequences.

I spent a long time debugging an error that was actually the result of a bug in the editor / compiler. Editing the original delimiter string to add a sequence of escape characters erroneously produced compiler warnings and errors. I'm certain it wasn't a mistake on my part as I made sure the string literal was written correctly. Typing out the same sequence of characters and assigning it to another variable worked. Finally I'm done with this. Edit: I was not done with it. String.split() takes a regular expression as the delimiter, of course. Oh my god googling how Java regex works when this should be an extremely straightforward problem to solve in any reasonable language is aggravating.

Finally. It finally works. What did the trick was using Pattern.split() instead of String.split(). I found this method after over an hour of googling. So what could have been, in a better world, simply responseBody.split(delimiter) ended up looking like this: Pattern.compile(Pattern.quote(delimiter)).split(responseBody); What an absolutely terrible experience.

Finished work on DataFetchWorker. App state is now correctly updated whenever fetching completes. Next I will have to research how linking state to display components is typically done in Android.

I've been reading the Android docs and for updating views they recommend ViewModel and LiveData, where a LiveData object holds the data and the ViewModel manages it. I will try this approach.

**7.6.2023**

The ViewModel for MainActivity is now working and the view is updated automatically when the MutableLiveData held by the view model changes. Now I need to figure out a way to update the MutableLiveData after UpdateWorker fetches new data. I think I will create a Mediator class to handle communication between StationsData (which holds app state) and the views.

Learned how to write and read SharedPreferences. I decided it was necessary to store the state of the currently selected station in SharedPreferences in case the app process is destroyed by the OS. Changed MainActivity's ViewModel to AndroidViewModel to be able to get context and thus retrieve data from SharedPreferences. Edit: As soon as I'd implemented this I realized it was not at all necessary to store state in SharedPreferences, and that data would be stale or at best redundant. All that's needed is storing the name of the currently selected station in SharedPreferences, to prevent the displayed station from changing every time the Activity or app process is destroyed.

Set API minSdk to 23 to be able to use TextView.setTextAppearance(), which changes the style of the text.

It took me a while to track down where my app failed. It didn't crash or anything, just didn't behave as excepted. I debugged the code and found an uncaught exception from MutableLiveData: "Cannot invoke setValue on a background thread from Coroutine". I googled the error and saw that MutableLiveData has a method called postValue() which should be used instead of setValue() when calling from a background thread - the source of the call chain was a Worker. I made the fix and it worked! It was a great relief and I'm pretty surprised I could fix it by just using a different method. Points to the API for that. This is the first time data is actually updated on the screen properly.

I think app architecture is mostly in place now. It took me a while to get here. Even a small and relatively simple Android app gets rather complicated to implement because of lifecycles. I had to think about the architecture of the program a lot more than in previous projects where I've written an equivalent app using other frameworks. Thankfully the Android API offers enough tools to get the job done. I haven't had to worry about threads or asynchronous execution in any significant detail which is pretty great.

Working with Java and IntelliJ has been pleasant when dealing with basic object-orientated programming. Code suggestions work very well and class interaction has been smooth to implement. Try-catch is still cumbersome though. The major pitfalls I've encountered so far are the complexity of solving some relatively simple tasks - here Java is showing its age - and the shortcomings of the Android API. Why is user input validation left to the developer to implement manually in Java, for example? Surely the API could provide some help in this regard.