C196 Project Reflection
Student Tracker Application
Developed by Brandon Lagasse

1. When considering the phone portrait/view, the first thing I would develop differently is the space used between fragments within the layout. For instance, if you have a fragment component that's used in multiple layouts, you may find yourself with too much whitespace if you're using a tablet, making for a less-than-desirable user experience. Tables also use more gestures like swiping than a phone, so that would have to be implemented. In the current app, only swiping would work. If you're working with constraints, these will have to be considered for each position, as the constraints may be attached to the screen edges.

2. My application was originally built for API 26. I didn't realize using an older API would cause so many issues, so I eventually ran APIs 34 and 30 for testing. This fixed an issue of data persistence between views. Knowing that 26 ran on most devices was my original thought process when picking the API. It wasn't clear to me when I started that the entire phone architecture would change as well. Fortunately, I discovered this early and was able to use different emulators to test between versions.

3. My biggest challenge came from not knowing the library while trying to plan out my system design. While javaFX was somewhat similar, a lot of state saving between layout changes was challenging to debug. Understanding the usage of emulators came as a new experience for me, because as I started building out my app, my focus changed to the code, rather than testing and the IDE. This was a big mistake in the long run. Recyclerviews gave me a lot of headaches as well. Along with saving state through layout changes, I also had to tackle issues with my math. I ran into several issues with saving terms, assessments, and courses, where they would save as -1 or show up for the wrong course/term.

4. Issues with my IDE and as well as the library were solved along the way, and not so proactively. It wasn't until I ran into issues with my normal channels of research that I really dug into understanding the library. Navigating the Android developer's website clarified a lot of my issues. When that didn't work, I worked with my instructor to work through the bugs and issues that were stopping me. The issue with Recyclerviews brought around my love-and-hate relationship with intents. This was a concept that I didn't cover when I first started learning how to develop on Android. Usually, I would pass between variables in Java, but this mechanism, when left out early, caused problems until I understood the concept; then the fixes were easy. The course, term, and assessment saving was a long process of backtracking, testing new variables, changing variable load location, and ultimately just knowing the order of operations. This was truly the most test-like area, as I spent probably a week on it until I figured out how to get the order correct.

5. I would do several things differently if I had to start the project over again. First, I would be able to make a much more comprehensive UML diagram knowing which parts of the android library and Room I had to incorporate. I would be able to make the system design make more sense, which would expedite the development significantly. Second, I would break up my code more to encapsulate small units of code, instead of bloated code that takes longer to load and is less efficient. And third, I would now be able to correctly build all of my layouts quickly. I didn't fully understand the relationships between layouts, menu, dao, and repository when I began. Ultimately, I'm looking forward to programming in Android Studio, just as I was with JavaFX when I was done. The process of learning for me was difficult, but I'm confident I could make an app in a much shorter time in the future.

6. The use of emulators vs a development device is very apparent the more you test your application. The more API versions you work with, you'll end up learning the intricacies of how the code works, as well as what best practices have been depreciated. The thing I love about emulators is the ability to wipe data easily, as well as try different API's at the same time. With a physical device, you're always going to be limited by screen size restrictions, API version limitations, and device hardware issues. The device hardware issues, if they occur, can cause another set of variables to be eliminated with bugs. With the emulators, it's nice and easy to delete what you need and create a new emulator within seconds.