Ryan Mitchell

CS 320

August 14, 2021

Summary and Reflections Report

**Summary**

This mobile application was an excellent introduction to the critical importance of JUnit testing and being able to confirm that all parts of a software are not only working as intended, but without exceptions. The two primary requirements that were focused on for this project were the field requirements of each class object, and the service’s ability to manipulate those objects as efficiently and simply as possible.

To address the first part of the requirements, I wrote each constructor with an internal boolean test to forcibly ensure that any object created would exist per the requirements. In the event a field was too long, contained the incorrect number of characters, or was null, an error message would be received and the object would not be created. To follow along with the requirements document, I also included methods that followed the naming conventions of exactly what each service required. The goal was to make absolutely sure that no additional methods were created, or that unnecessary parameters were used.

In testing the coverage of my services, I managed to achieve a 90%+ coverage rating on each test service and minimum 80 % on the object classes. The goal was to make sure that each aspect of every function was inputting and outputting exactly as expected with no exceptions that are mishandled. The JUnit tests had great assistance in their design thanks to a very cool and awesome professor, which I’m sure only further increases the credibility.

My experience writing JUnit tests independently was rather straightforward. Thanks to the very structured nature of their design and their goals, I was simply focused on imagining every potential output of every method and made sure to use an assertTrue/False to guarantee that not only every method, but every STEP within the methods was following expected logic. If you look at any ObjectTest.java, you can see that the very first step is to validate that every single field within an object contains the value it was constructed with. Since the primary goal of each service is the instantiation of objects and the local storage of them, every ServiceTest.java also tested the addition of newly created objects and their retrievability, as per the requirements. Even though I tested to ensure that objects constructed had all fields validated, I tested those same fields after they were added to a Map structure. To add onto the soundness of my tests, I also would mention that in addition to validating that input is correct, there are also tests to validate that a certain input would be handled if it was incorrect (such as in the testAddingDuplicateIdFail() test.

In terms of efficiency, some tests may not have been as efficient as they could have been. I believe this was in favor of thoroughness however, since testing every aspect is important. The efficiency from the JUnit tests depended on the efficiency of the code it was testing, and I *can* attest to this. Services used a singleton instance, a map data structure, and included methods to add, delete, or modify as requested. Object classes were setters/getters, constructors, and boolean methods to validate input. I am a little confused by this question to be honest, because while I feel the tests themselves were efficient, the primary goal was coverage which requires being extra thorough.

**Reflection**

The software testing techniques that were employed were ones that were both intuitive and demonstrated throughout my courses thus far. In the sense of what types of tests, the Assertions and Test packages provided by jupiter were the main sources. The biggest testing methods were assertTrue, assertFalse, and Assertions.assertThrows which were put to work to check every step of every method as expected. If we’re referring to more abstract software testing techniques, I tried to focus on consistency and simplicity. I used a lot of copy/paste within my programs for certain tests to make sure that I wouldn’t miss any small details due to typing errors. I tried to keep my tests concise and stick to what I was learning to prevent any misunderstandings or lone troubleshooting sessions. I also used the “step away” technique which seems to be a very underrated strategy. In the event I hit a roadblock, walking away from the computer to think about it would usually help me refresh and re approach the problem in a better state of mind. This can be easily applied to all aspects of life, in that sometimes you just need to take a break and destress your mind before you can truly unlock creativity for a solution. The last strategy I used was “test as you code”, which is also a common approach. You simply debug what you code as frequently as possible to prevent late troubleshooting over a 300 line application that has bugs which create more bugs.

For my mindset within this project, I would say that caution was definitely a key virtue that I focused on. As long as I was testing consistently and getting clean results, I generally developed my project pretty smoothly. Sometimes I would love an approach and try to incorporate it into my code, but only if it was something that I could fully understand. For things that are complex, I’d say it’s important to only use things that you absolutely understand. If an error arises within a technique or a method that you are just learning how to use, then troubleshooting becomes even more challenging. I have experienced this on several occasions in the past. In reviewing my own code, I always keep in mind that I am not an expert software engineer yet, and I always assume that some approach has been improved upon already by someone far more experienced than me. So between keeping my code simple and outcome oriented, I can be confident in what I write but always understand that there is most likely an even greater solution that I haven’t learned yet. As far as discipline, I know that coding sessions can be tedious or frustrating. Quality diminishes when a person becomes unmotivated or mentally strained, so constant breaks and quiet time to just think and imagine what might be possible one day are how one might help stay motivated. I will avoid technical debt by always working for a cause I can be proud of, and by focusing on the greater goal of serving to make the world a slightly better place. Routinely taking care of my mental health, and focusing on the big picture are how I maintain discipline. The best results stem from when a person enjoys what they are doing, and that applies to everything.