**Jonathan Santiago**

**CS-320**

**8/10/2022**

**Project Two**

1. **Describe your unit testing approach for each of the three features.**
   1. I tried to design all my components to the exact requirements provided to me. A large portion of the early SDLC stages revolves around understanding the requirements well and producing a design around that understanding. This means that a lot of time should be spent on trying to understand the customers’ needs as thoroughly as possible. Even if the team ends up producing a fully functional application at the end of the cycle, if it doesn’t accomplish the customers goals then it was just a waste of time and resources for the customer and could damage the developer’s reputations. An example of this would be the input validation within my class methods. The user had specific input validation requirements for each component, this usually was a requirement that input be a certain length and not a null input. I’ve seen examples of these types of validation that are incorporated into service classes, which are the classes that handle the tasks performed on the objects themselves. This is not out of the ordinary, but I chose to incorporate these validations at the lowest level possible, within the actual object’s methods. This increases security in my opinion because it makes the objects harder to manipulate from any sources outside of the service class. The entire application surrounds those lower-level objects so making them secure is a top priority and I found that this was another layer of protection we could offer those objects. I felt that this was more in line with the customers’ requirements than adding them at the service level.

I think my JUnit test had a very satisfactory coverage percentage. I wrote my tests in a way to make sure that I was getting the output I desired from my classes. I thoroughly tested every conditional branch to make sure I got not only proper output, but also thrown exceptions where input was not valid. As my screenshots show, I was able to get a very high coverage percentage, with the lowest still being above 80% overall but each class was covered 100%.

* 1. My experience writing JUnit tests was a very positive one. I never understood automated testing before I took this course, but it is not as intimidating as I once thought it was. What also helped was the simplicity of the components we were developing. JUnit has an assortment of tools within its arsenal, which made it very intimidating at first but as I did the reading and learned more about project one, I only needed a few of those tools to accomplish my goals. If you look at my code, you’ll see that the only two tools I use from JUnit’s library are assertTrue and assertEquals. These two tools allowed me to get complete coverage of all my classes while keeping the lines of code simple and straight forward. This same example can also be used to show how my code is efficient. Efficiency is not only about speed but also about how your code is structured. An application can perform great but if the code is structured in a way that makes it difficult or even impossible to understand by other developers, it’s useless. The applications of todays technical landscape have to always be built with future scaling in mind, so developers have to take that into consideration when they think about efficiency and the I feel that the simplicity of my code shows that.

1. **Reflection**
   1. I began my process by using static testing techniques. As I mentioned before, the beginning of a good application starts with having a good understanding of your customers requirements. If you can’t understand what they want to get out of the application you build for them, how can you build them a worthwhile product? Using this mindset, I made sure that I understood exactly what I needed my components to do. I also used the feedback from my instructor as another static testing technique to understand where my components can be improved. Because we aren’t working in a team for this project, my instructor’s feedback was the only type of peer review I could do so I took advantage of it wherever possible. I also used a dynamic testing technique called multiple condition coverage. My reasoning behind using this technique was the input validation that the customer requested. The components themselves were not too complex but the input validation done in each object method did have multiple conditions. Because of this, I made sure that I had written tests that cover each branch of all the conditionals along with some failure tests to make sure that exceptions would be thrown if the input is not valid. These are simpler techniques that fit my project because my application was very small but in the real world, you’re likely to be just one developer amongst many across many different teams so using this simple technique might not be practical due to the applications size and complexity of the code base. Always adjust your testing approach to the project you’re working on, don’t get stuck in the mindset that you have to perform tests a certain way every single time. Today’s technical landscape favors agile methodologies and flexibility so we should approach testing in the same manner.
   2. The mindset I tried to adopt for this project was the mindset of doing just enough. Sometimes as developers, we end up going down a rabbit hole when trying to build an application. At times, this can a good thing because some out of the box and brilliant ideas can come from it but often, it just leads to a huge waste of time. Developers can end up writing code that is way more complex than it needs to be and could be hard to for others on the team to understand which leads to confusion and stress. In previous SDLC books, I learned that when building an application, developers should do just enough to accomplish the goal. Time is money for the customer so doing the bare minimum to meet a requirement avoids the stress that can come from complex code and saves time. This makes the customer and rest of the team happy because it pushes the application closer to what is considered finished and helps save a lot of money. The best way to avoid bias is to try to take an outsider’s perspective. Will another developer that has no investment in your project, look at your JUnit tests and agree that they make sense? If you have to even think about that when looking at your test cases, then it’s best that you rethink them. A testing mindset requires that you poke holes in your application to make sure it meets the requirements of the customer, not your own personal requirements. Don’t try to write your tests in a way that you know will pass, write them in a way that will truly test the functionality of your application. Make sure you receive proper outputs, make sure that only validated input is allowed into your system, make sure that your application fails when it’s supposed to such as when receiving invalid input. Adopting this mindset when testing will allow you to build a more foundationally sound application from the beginning and improve the overall quality of your code base.

**References**

* Agarwal, T. (2020, February 11). Testing techniques : Types, advantages and disadvantages. ElProCus. Retrieved August 10, 2022, from <https://www.elprocus.com/what-are-testing-techniques-types-advantages-disadvantages/>
* Justice, J. (2020, December 23). Why write the minimum code to pass the test? CodingItWrong.com. Retrieved August 10, 2022, from <https://codingitwrong.com/2020/12/23/why-write-the-minimum-code-to-pass-the-test.html>
* Kuldeep, M, A., Founder, V. S. T. H., shete, N., Sahane, S., Jim, Mani, S., Kothari, R., Dunne, K., Winters, B., soni, K., &amp; Aram. (2022, August 7). 4 steps towards developing the agile testing mindset for successful transition to agile process. Software Testing Help. Retrieved August 10, 2022, from <https://www.softwaretestinghelp.com/developing-the-agile-testing-mindset/#:~:text=An%20Agile%20testing%20mindset%20means,added%20activities%20like%20finding%20defects>.
* Liebert, B. (2019, January 10). Why developing the bare minimum is the right strategy. Blackball Software. Retrieved August 10, 2022, from <https://blog.blackballsoftware.com/posts/2019/1/10/mvps-why-developing-the-bare-minimum-is-the-right-strategy>
* neeru360. (2021, March 1). Software testing techniques. GeeksforGeeks. Retrieved July 29, 2022, from <https://www.geeksforgeeks.org/software-testing-techniques/>