**William Wootton**

**SNHU**

**CS320**

**10.19.2025**

**Summary**

When working with the unit tests for the different packages involved in this system, I made sure to test that good values placed in would work and be added as intended. This also means during testing that when inputting bad values, errors would occur, meaning that only good values could be added into the system. When making JUnit tests I tried to condense them from feedback I had received over the course of the last few weeks, I was also trying to fit within the coverage of 90%. I was conflicted as to how many tests I should have and how complex they should be, I think in the scope of this project and feedback I have received that I fit this project decently well. However, if this was a project on my own, I would have liked to work on more tests and improve my system so that it would be more understandable and check more possibilities than I checked in this project. I tried to condense my tests down which in turn made them less robust and deep, maybe if I was to do another system with testing I would spend most of my time working on tests when coding as making sure I have enough tests to fit my own needs, and trying to fit possibilities of errors and issues inside these tests to make the overall report much more detailed.

Ever since I began this class, I have fallen in love with testing and testing code. I use it in all of my code now and wish to expand my knowledge and abilities in testing in the future. The coverage percentage for my testing was 92%, I think coverage can be quite misleading though in this case as I didn’t see it check for coverage on the else part of my if statements, which may have just been a me problem or a coverage problem. This means that I can only check if good values go in and get errors if I attempt to add a bad value. Again, this could be a personal error, and I will be looking more into it the farther I get in my java learning journey. I tackled all the problems within the rubric and guidelines and considered advice given from previous assignments, I remember advice that said my testing was too big or too long, so I condensed my tests to make the more readable and easier to understand. This was needed as reviewing some of my earlier projects gave me a headache at how much stuff was packed into one testing case.

**Reflection**

The testing techniques I used in this project were primarily white-box style code, myself being the main and sole developer wouldn’t be able to perform a high quality and traditional black-box testing style due to my intimate knowledge of the system. I think in a team approach, having a tester focused on black-box testing would help the system and team a lot in making a system that relates to the main idea as much as possible. White-box testing is great for evaluating the system internally, checking to make sure the functions I implemented work how I wanted, and the cases where I think an error may occur shouldn’t work. On the other hand black-box testing or another tester with no intimate knowledge of the system may come in and test something I never even thought of and it might ruin the entire system, this is invaluable when working with a system that may be accessed by hundreds of thousands of users, some of which may wish to cause harm to said system.

Testing code can be complex and confusing even in simple systems like the one in this project. The ability to create small, readable tests with easy-to-understand names and purposes allows this process to be much easier. When looking back at my old packages to understand what to implement in this project, I spent a lot of time just refactoring both the code and the tests in each package. I think the longer the term has gone on, and more feedback has allowed me to approach tests in a different way compared to how I was at the start. Computer science is a field with a wide range of different topics and specializations inside it; there aren’t a lot of professions with as many specializations as this one. From software engineer to software tester, to front-end developer, to systems architect, to data analyst, etc. All of which play a role in the development of any given system and to create a truly high-quality product each branch and level of development must be on point 100%, 100% of the time. This means that cutting corners can cause harm to not only your team, but also your department, your company, and any users wishing to use the systems you worked on.