**CS320 Project Two**

To ensure that the program/mobile application worked as intended, I read the specification document and tried to follow it to a tee. We were asked to ensure the class tests met requirements through Junit testing. So, the first and best place to start was with the classes themselves. I began by creating the class and the essential variables before moving on to the methods the class would require, such as getters and setters. The classes and their main variables had set requirements for how they would function or be built. One such example would, of course, be in the ContactClass. It was to contain a string that could not be longer than ten characters. I think it was pretty easy to do that, but to be sure, I did things manually, counting the string and so on. Regarding the unit tests, I would say that what I needed to do was pretty simple. Create a new object and fill in the required fields. But to ensure that this test properly runs, for the field being tested, it needed to fail so in other words, its character limit needed to surpass ten characters. Back when I talked about creating the string and making sure it met the requirements, an easy way to use it for the test would be to add any number of characters that would cause the limit to be exceeded. There were other requirements, for example the, stringID being not null and not updateable. Junit provides several frameworks or tools to help me check some of these. For instance, “assertNotNull” ensures that a parameter passed is not null. If the parameter happens to be null, it would cause the test to fail, which is what we need. So, for any null tests, I passed the parameter being checked as null, helping me check that off the box. The general approach to writing the tests was to ensure that they failed. I was intentionally passing failing parameters to check for functionality. I applied the same strategies and similar code for most tests with identical parameters. This gave me great success as I had good coverage in the test package. It surpassed the required 80%, with only the single class of AppointmentServiceTest falling under that mark amongst all classes in the test package. It was a pretty good experience writing the code for the Junit tests, as we had been building up to it with the milestones. It gave me time to familiarize myself with the code and how to write it.

I used several testing techniques, chief of which was unit testing. Unit testing involves testing specific parts or functions of a program to check its functionality. I did that because the course mainly wants us to use Unit testing. Testing methods separately, as described earlier. For example, testContactPhoneNotNull, which, judging by the name, was a test to ensure that a contact’s phone number isn’t null. Unit testing was applied to check all parts of the project individually. Another testing technique I employed was a combination of manual and automated testing. Seeing as I had to write the code, I needed to make sure it was correct as I was writing, so examining my code manually for any errors was important. The IDE helped point out little mistakes in my code and suggested fixes for them; some I followed, and some I didn’t, as I worried how they would affect my project. Automated testing was done by using the IDE to run the code. Whenever the code was run, I checked whether the tests passed. While I could check that manually, there is only so much I would be able to do. The IDE also helped display test coverage, showing areas I needed to improve or add more tests. All the Junit tests in and of themselves were a form of functional testing as we were developing software to do certain things. So, the Junit tests were a way to check for that functionality. A testing technique contained in the functional testing was boundary value testing. A clear example of this can be found in ContactTest, TaskTest, and AppointmentTest, checking to see if particular parameters and variables exceeded a limit and how the software would react in those situations. All these testing techniques ensure that users can use a product without too many issues. For example, a secretary trying to access a customer’s appointment data to disclose information on a call would pull up what they have from the record. However, the customer needs to put the correct information, like their phone number, in the proper format. Let’s say the default or expected number of characters for all phone numbers is 10, and a customer can somehow input 11, which would automatically make their contact info incorrect, presenting a flaw in the system and leaving the info undisclosed.

One important rule in software development is to ensure good quality in any products or projects one is involved in. Going into the project, I wanted to produce something that worked. If it didn’t work well, I wanted it to at least fulfill basic requirements. One way I did this was by exercising caution, following the rules laid out, and observing and implementing feedback received. This and the various methods, variables, classes, and applications form delicate interrelationships. If one class or method is changed or affected, classes or tests dependent on that test could be damaged or not work as intended. If I changed the name of the contactService class in the main package, the contactServiceTest class in the test package would not be able to pull methods or use functions from it anymore or how we need it to be used. Limiting bias was simple and complicated at the same time. One recommended way to limit bias would be the use of dedicated testers. However, as it was a solo project, I had to do the testing independently and follow the provided requirements and rubric to fight against bias.

Being disciplined, especially as a software engineer, is vital for specific reasons. First, a company places its trust in you to develop and deliver a product, giving you some freedom to work on said product. By extension, if the company puts its trust in you, so do its customers. This all comes together to form a third reason, which is your credibility as a professional. Cutting corners would drop your credibility and value as a pro, fostering distrust from companies and customers. When this happens, things like finding work, providing, and doing what you have spent time building on(your career) become harder. So, hard work, caution, and diligence are required to avoid these, as well as technical debt, which is the cost of reworking a project due to errors that might have come up through insufficient testing.