**7­2 Project Two**

**Ashley Baldwin**

**Southern New Hampshire University**

**CS­320­T4208**

**Professor Mike Prasad**

**April 21, 2023**

7-2 Project Two

**Summary**

In the beginning, my approach for each of the three features for the mobile application was simply to follow the tutorial and the examples from our reading on JUnit testing. At least this was my approach for the first feature of the application, the contact service feature. I then investigated other videos to help me understand what I was doing to do better on the second feature, the task service feature. While I did much better on this second feature and implemented what I learned and fixed up my first feature I still wasn’t confident in my work. I looked up other code like that that I was working on come the third feature, the appointment service feature. Again, I improved by my third feature and implemented these improvements in my past two features. By week 6, I realized I wasn’t happy with what I created. The code worked and the JUnit coverage was almost at 100%, but I thought I could do better. So, I focused more on each feature and how each part of the code was going to work together. I organized my code in a way that was easier for myself to read and understand, especially when things went wrong. From the start my main focus was on the software requirements. What this code was supposed to do. So, in my first feature, the contact service feature, I created a way to set and get things like contact ID, first name, last name, phone number, and address. The contact class holds the setters and getters along with checks that the entered information matches the requirements. For example, one requirement was for the phone number to be exactly 10 digits and cannot be null (it can be updated, but more on that for the contact service class). The requirement specifically stated “exactly 10 digits” meaning it cannot be just any 10 character, but only numerical characters (0-9) with no spaces and no dash, “-“ . So, after some researching on the internet, I found a way to limit the characters being anything between 0 and 9. It was a nested for loop. While this did work it felt like too much. Not to mention I had this nested for loop statement in my main constructor (the main constructor starting on line 90 in my final contact class) and in my setter for phone number. This was a lot of text to have to understand and go through multiple times. I learned about the regular expression, or “regex” in my code at line 47, can make limiting characters easier to read and shorter to code. Not only did I use this in my final code, but I also created validation checks that I could refer to within my setters, as shown in line 148 – 155. This cleaned my code and made it easier for me to read and understand later, when I needed to call back to specific methods I created. Then with the contact service class, this is where I did something simple, yet effective when it came to the requirements. The contact ID was to be no longer than 10 characters, cannot be null, must be unique, and cannot be updated. If you look at the contact class, you may think that I’ve not followed requirements because I have a setter method for contact ID. I felt, with how I coded this, that a setter would be necessary when creating a new contact. In contact class you will also see a constructor that only takes the contact ID as an argument. This was made for checking if the ID entered matches other IDs within the hash-map list. If the contact ID didn’t exist, then you can’t do things like update the phone number or delete the contact. This also helped with adding a contact in a similar manner. In the contact service class I have methods called updateSomeName(). These start at line 54 and finish out the code. I purposely left out making a method called updateContactID() for the reason stated within the requirements. You are not allowed to update or change the contact ID once it is set. Again, I was looking to keep my code easy to read so I wouldn’t get overwhelmed when trying to fix my code in any way. Careful coding and recoding were done specifically around the requirements. I didn’t want to over code or do anything super fancy. I stuck to the requirements how I understood them.

As for the JUnit testing, I was very confused about that. This was brand new to me and something I’ve never used before. Other than building and running my codes, I never tested my codes. Therefore, understanding the coverage was a challenge. So much so, that I specifically bought a cheap notebook to write down the changes I made in the class, line, and detailed explanation of what these changes were to do. I also wrote down the coverage percentage before and after these changes to know if I was making improvements in terms of coverage or if I messed it all up. I knew green was good and red was bad, but I was struggling to understand the lines highlighted in yellow along with understanding the message that came with it (for example, 2 of 4 branches missed). While coding, I was so overwhelmed by the coverage that I was mudding up my code and making it hard to read. Once I rewrote all my code, I also rewrote all my original tests, just to make things easier and to help myself with the coverage. I told myself that 100% coverage is not possible, but I at least wanted my main classes (i.e., contact and contactService classes) to be tested at 100% so that I know everything was tested for. I knew that my JUnit tests were effective based on the percentage and the coloration of the code highlighted. I am still a bit confused as to why the JUnit testing classes cannot be 100%, as some of the lines such as the assertThrows() tend to either give me red highlights (these were more prominent in my first drafts of my testing classes) or yellow highlights (which can be seen in any of the testing classes, but more specifically in the ContactTest class at line 113). When testing the full coverage for my contact service feature I got a 94.9% test coverage with my contact class and contactService class testing at 100%. Again, the percentage tells me how much has been covered by the tests, which is how I know my tests were effective. When looking at the contactService class, starting at line 26 you can see my method addContact(). There is an if-else statement. This statement has branches. Branches are execution paths. My current coverage test states that all 2 branches were checked, meaning that both execution paths were tested in my test class. This is also how I know that my code is technically sound. As far as the code being efficient, the tests for coverage and JUnit testing were completed within 0.339 seconds. With the ContactServiceTest with 0.001s and ContactTest at 0.058s.

**Reflection**

For this project, I didn’t really think as to how I was going to test my code. I was more concerned about creating the code and test code than I was testing it. When looking back, I suppose I used a bit of dynamic testing and white box testing with a bit of exploratory testing thrown in. While in a professional setting, these methods may not be the most ideal way to test, this was how I worked best as a solo engineer. I’m terrible at planning, so I typically get straight to coding basic things. For example, going back to the contact class, I knew I needed variables to hold information. I also knew the requirements for each. I also knew I needed to create setters and getters with the scope of the feature. I did create each class one at a time, however. I also created a test class for the class I just made. Again, I created the contact class and then once I finished, I created the contact test class before moving on to the contact service class. Since I had no real understanding as to what I was doing and why it needed to be done, looking back I felt most of my decisions were based on an exploratory testing. This type of testing is as it sounds, less planning and more of getting a feel for the software, adapting to findings on the fly. But I wasn’t so ignorant on what I was testing as I was creating the classes to be tested. Which is where white box testing comes in. In white box testing the QA engineer fully understands the internal structure of the software and its design. In my mind, it’s like the QA engineer is an inspector. Making sure everything is running and working as intended. The tests in white box testing are typically more deliberate and can be run in the early stages of development. Which is what I did. For example, in my contact class, I have the constructor first, followed by the getters and then the setters. In my contact test, I followed the same order as I wrote my contact class in. I tested the constructor followed by the setter and getters. I deliberately created this test in this way to test that my code was working as intended. This helped me tackle any problems that might have been in my code. When testing my contact service class, in the test I would write print statements, to print out sections of code to ensure that my code was adding and deleting as intended. This is part of the dynamic testing, where I would execute sections of code via test statements and even in my now deleted print statements. The print statements helped me understand how my codes were executing in ways that I couldn’t have progressed without. My print statements were showing me what my test codes were executing. My process was a beautiful chaotic mess that allowed me to understand how my code was working and that I was creating effective tests.

There are many other ways to test code. There is static testing, which tests code without ever executing the code. A way that my brain will not allow me to do. I cannot understand how to test something without seeing it work. However, static testing is the most cost-effective way to test, as when or if you find an error, you can fix them there before code is ever written. Automated testing is a common way to test as well. Instead of writing testing code, as I had to do, a software creates this code, doing more of the basic testing’s that QA engineers tend to spend most of their time on. There is also black box testing. This is where the design of the software being tested is unknown. This helps QA engineers test in ways that the end user would expect the software to work. In other words, it leaves the QA engineers to have the same expectations as the end user would have with the software. Being that I coded the software I was creating, this type of testing was impossible for me to do.

Finally, my mindset with this project. This was a somewhat stressful project to do simply because I was intimidated with the JUnit testing and the coverage. I used caution when it came to the requirements. For example, in the contact service feature, the requirement for the contact ID was to be no more than 10 characters, cannot be null, must be unique, and cannot be updated. When starting this project, I didn’t know how to go about creating something that could also be unique without checks of if that ID already existed. In the end I opted the checks as my best rout as the requirements stated that the ID needed to be no longer than 10 characters in length. My first creation I made a loop statement that generated a unique ID, however this generated ID was only using digits. I found that while not a bad idea, it also goes against the requirements in a way. Meaning that the ID can be no longer than 10, which means that one contact the ID can have a length of 3 while another could have 7 while another could have 10. It was too advanced coding for me to include an automated ID generator. While it almost guarantees a completely unique ID, I used caution to the requirements and kept my code simpler when it came to the requirements. I was also cautious when creating tests for methods that had statements, such as the if-else statement. The more statements a method or any code, the more branches you are creating and the more complete the test code needs to be. You need to be mindful of each branch that the code can execute and test that these branches also run as intended. While I was being cautious with my coding, I don’t think I was ever biased of my coding. I was never so proud of my code that I couldn’t change it or do something differently. Not at first. I’m proud of my final code, but I would add changes to make the final code even better. I had ideas for the code that were to advance for me at the time. I never looked as my code on the software developer side as my own, because I was using other ideas to better my base code. While I didn’t copy anyone’s code, I was inspired by some with their way of spacing the code or writing the code to make it more readable. I was inspired by others to use regular expression for my phone number requirement. Not all the ideas were mine originally, which is where I never saw my code as my own I suppose. Therefore, I never went easy on my testing. I tried to test everything, which is how I got my coverage percentages to be 100% for the three features (not the overall test coverage percentage). When turning in my project, the requirement was that the coverage needed to be at least 80%. I assume that was for total coverage. While all my features did in fact meet the minimum, I pushed myself to go the extra mile. I wanted my software developer codes to be at 100% coverage when I acted as the QA engineer and tested those codes. I worked long hours trying to understand my own code and how to test better or to change my code for better testing and efficiency. I was so frustrated at time I cried, but it mattered because I knew the tests needed to be fully fledged out these features. It’s important to not just take the minimum effort when writing or testing code. Especially if it’s code that goes out to end users to use. It’s one thing to cut corners in a personal project, but even then you are practicing bad habits that you could accidentally take into the work field. It’s better to stay disciplined with your code and to go the extra mile as to not do minimal effort and to cut corners. I was tempted to just give minimal coverage as required, but I knew I would be disappointed in myself because I know better than to provide the inimum effort. I also knew that my coverage wasn’t 100% within the main classes, and that was going to bother me as well. While I know I can’t test for everything, I can at least test for what is actually there. I can test for each path the code could take. This discipline will help me in future projects as to not always be so proud of my work that I give the minimum effort.