Project Two

CS 320

Justin Farquhar

6/22/2024

My approach directly aligned to the software requirements as I would constantly reference the requirements when I was working towards creating the software. By doing this, it allowed me to ensure that I was adding the features that the customer wanted out of the application. This also allowed for a better approach when it came to the testing side of things, letting me follow the guidelines already provided to ensure everything is working fine. Someone else could look over the guidelines of the application that was provided to us, and look through my code and directly see those reflected within it, typically in the same order as I would work through it in that same order. I generally used the same approach for each of the individual features, working off of the previous feature and adjusting my code so that it would work for the new feature. My quality of my Junit tests reflected this as well, increasing the overall quality by directly relating to the requirements. There is the potential room for including some of the additional junit tests, but while simple, I believe the ones that I chose to work effectively test my code and bring the correct end results. My code coverage could have been higher, but I believe that the testing requirements were met adequately.

I ensured that my code was technically sound by keeping it simple while still solving the requirements of the customer adequately. I believe my validation methods were done very well, ensuring that proper input validation was performed, I have included a snippet of code below:

A screen shot of a computer code

Description automatically generated

This snippet ensures that the appointmentID is not blank, the length is less than 10, and that there is not a duplicate ID that is already assigned. I used this general layout for all of my other validations, giving my code uniformity and keeping things easier on myself allowing for the reusing of code. This example also shows how efficient my code is as this verification code has the most checks, yet is still only 13 lines, which could likely be shorter if I was better at organizing my code.

For all of my tests, I was using either assertEquals or assertThrows. Using assertThrows ensured that my verification code was properly verifying each input and that they adhered to the requirements from the customer. These tests would work by creating an object with an incorrect value which should trigger an error code through the verification process. If the error was now thrown, the test would fail showing that something was coded wrong, or if the error was thrown, the test was pass showing that the inputs were being properly validated. The assertEquals was used to ensure each of the adding and deleting of objects was working properly. I was able to check using the specific ID’s for each object and compare with what I was adding and what should have been added. Similarly for the deleting, I would ensure that the specific ID was set to null, as it should no longer exist. There are a lot of different testing techniques that I did not find necessary for this project. AssertNotEquals would be used to ensure that two objects are not equals, which would not have been beneficial to this project, but could have been used when updating items within the objects, though equals works just fine for this. AssertNotNull would not have worked overly well with the objects used in this project as each part of the object could contain a null item, but the object itself could technically be null if it was deleted, and would be better off using if I were to be testing if an individual item was not null.

I employed a decent amount of caution throughout this project. Before this class I had never used junits before and I did not have the best understanding of java. So throughout the course I was being very careful when adding and looking up code to use to ensure that I was adding code that was appropriate as well as efficient. It is also extremely easy to use incorrect code when you do not understand what you are using, leading to incorrect results or even error ridden code. It was very important to appreciate the complexity and interrelationships of the code as this was directly for a customer. If I opted to ignore the complexity and try to wing it and make it as simple as possible, there is a very high likelihood that the code would not function appropriately and cause more issues for the customer. I tried to limit bias as a lot of this code was one of the first times I was using it, so if something wasn’t working, I simply chalked it up to me not understanding how it was supposed to be working fully and I would turn to researching it more rather than trying to brute force something to work. Bias can lead to a multitude of issues, if you are reviewing your own code, some people could take that as you could do no wrong or that since I wrote it, there will not be any errors present. This is almost never the case, and as soon as you begin expecting something to happen, it is less likely to be true. Discipline is an incredibly important skill to have for software developers. Being able to keep to the task and avoid cutting corners can drastically improve the quality of the code, leading to less errors in the end. At the end of the day, we are developing a product for a customer that they are paying for. If we are making a less than adequate product for them, we are truly cheating them out of their purchase, and every step should be taken to ensure that they are getting the highest quality that we can offer.