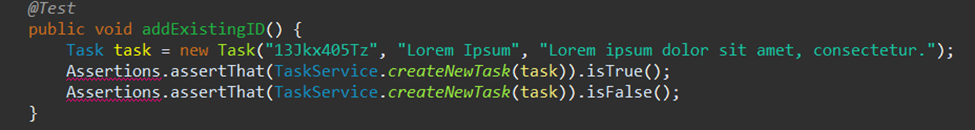
During this course we were tasked with building the back end of an application with no user interface required. The three core areas for the application were contact, tasks, and appointment information/creation. For that there were requirements for all three to be able to create, delete and update (contacts and tasks only) with information. More specifically there were restraints on the type and size of the inputs that the program should expect. Another example would be to ensure the same ID could not be input twice (code below),



I believe that my code was sufficient in meeting software requirements, however, I could have done a better job creating those rare edge cases to ensure the program would not fail. That said the intent of the application being created and what the customer asked for was met.

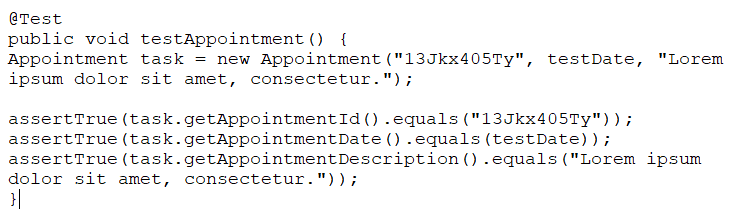
The overall quality of my work shows when running coverage for your Junit tests and the higher the coverage directly reflects the effectiveness of your tests. My coverage percentage across all three milestones was 96% or higher and this was due to testing every line of code created minus a few get() lines. This shows the effectiveness as when I used loops or if statements, I made sure to test the lines of code within them. One example of this was when I was required to update a contact I would first test that the contact existed, if it did, I would update and then test for the new updated contact information. The only thing that I could have done differently, as stated above, was to bring more extensive testing examples when it came to possible invalid user inputs to ensure a rare case would not cause failure within the code.

I made sure my code was technically sound by using appropriate data structures and keeping this data type through all three services. Switching from hash map to array or even a list would be confusing for other developers when reviewing my code and be harder to upkeep in the future. Since I chose hash map for all three, I made sure to use .containsKey (see below) and other tools associated with hash maps.

A screen shot of a computer code

Description automatically generated

Additionally, if you look at the example below when testing the creation of appointments making sure the data is assigned appropriately is another way to demonstrate the codes technical foundation.



To determine the efficiency of my code I refer to the example stated above when utilizing .containsKey. There was also the ability to use a for loop here to sift through the hash map however this would have been unnecessary code. This is because you would be adding longer runtimes and decisions within your program that are not necessarily needed. .equals is another efficiency choice as well because while an if statement could be used your introducing decisions within your code that is not needed which is why I believe efficiency and being technically sound go hand in hand when creating this project.

Many of the software techniques that I chose to employ during this project would be considered “Black Box Testing Techniques”. The main ones within that category being some boundary-based testing and then assessing user inputs through use case testing as well. From our test book use cases provide a way detect issues within the code or allowing unexpected outcomes to be observed (Software-testing, 2019). When it came to the contact service, would be if a user tried to update a non-existent contact. During the deletion of an appointment, if the appointment did not exist did the application respond correctly. Lastly, during task creation if a user updated with invalid data was it handled appropriately. These use cases that I have provided above are real world examples that I am guilting of doing as well at would be valid test to run to ensure your application can handle some common occurrences. Boundary test was also important within this project and there were data size limits across all three modules that needed to be confirmed.

Decision testing was a White Box technique that I did not use during this milestone. “Decision testing aims to ensure that the decisions in a program are adequately exercised.” (Software-testing, 2019) While I did test different statements within my code, I did not have very many decisions that required these white box techniques. For this technique of decision making, it would be applicable when deciding which decision tree needs to be executed based off a user response or data being input. One such example maybe if you were creating an administrative system and data required from employees would be very different than that of a customer or vendor.

The mindset I had to adapt for this project was not only that of someone learning testing techniques for the first time but also adopting more of a higher-level view of the program beyond just its creation and what my customer is expecting. I used caution within this project by using simple techniques and keeping my testing techniques at the beginner level as to not bight off more than I could chew. It was important to understand the complexity because you were creating code that was going to need to be read and provide testable lines. One such example was I needed to add code to check for a double ID being input or there would not be a test that

I tried to write tests to meet the requirements which helped me take personal bias out. However, in the real world I would be concerned that I may be too invested in the code that I create, and I would want to assume that it will function the way I intend it too. This could lead to areas being missed with my testing approaches. For example, if I needed to create a banking system and there should only be two decimal places or numbers should add and not multiply. Had I created the code I may have found that area too simple to mess up and not run code on it because my personal bias may let my ego get in the way when it comes to the simpler areas of the code.

Finally, to evaluate the importance of being disciplined when it comes to becoming a high functioning professional within the software industry, I look at accountability. Holding yourself to creating well written/robust testing code will be evident when others review and utilize your code. Cutting corners can lead to areas missed and reflect on the standards you hold as a developer. To avoid this, I will focus in on the documentation on languages and libraries to make sure I fully understand what capabilities are at my disposal and reviewing project documentation to meet the needs of the customers/employers that have put their faith in me.

Work Cited

Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). *Software testing : An istqb-bcs certified tester foundation guide - 4th edition*. BCS Learning & Development Limited.