CS 320 Project Two

In previous assignments, I was tasked to develop a mobile application which contained contact, task, and appointment services. These were all tested using JUnit tests to ensure the quality and efficiency of my code. I made sure to read through each software requirement and implement them in my code. One of the requirements that was given to me was a character limit for each attribute within the contact, task, and appointment classes. In the class, each attribute was developed to only accept entries which followed the requirements. Otherwise, an exception would be thrown to indicate that the entry was invalid. I also used JUnit tests to ensure that these attributes could be set appropriately without error and would throw the exception whenever necessary. JUnit tests are able to provide the coverage percentages for each class whenever the tests are run. Each of my classes had at least an 85% coverage level, indicating that the tests were effective, and the code will run properly.

During the past assignments, I gained an understanding of the importance of using JUnit testing. One way I ensured that the code was technically sound was to use proper annotations for each test. This is a necessary action to declare the method as a test. An example from my code is:Graphical user interface, application

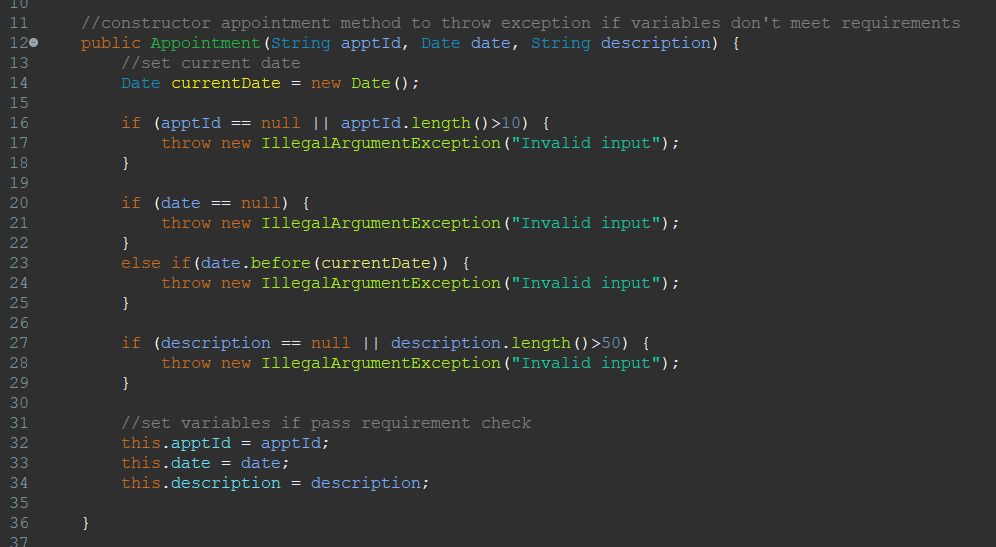
Description automatically generated

I also used in-line comments to make my code more technically sound and describe the code. An example is shown below:

Text

Description automatically generated

In regard to ensuring that my code was efficient, I used JUnit tests to make sure that each method provides the proper outcome and throws exceptions whenever necessary. I also used a constructor method in each service to ensure that all attributes meet the requirements given by the client. An example from my code is shown below for the appointment service class:



The software testing techniques that I used for this project include black box and white box testing techniques. In terms of black box, or specification-based testing, I considered the requirements of the program and tested those requirements within the code. Using black box testing ensures that the output of the program produces the expected result. The techniques that I implemented were equivalence partitioning and use case testing. For each service, I used equivalence partitioning to set the requirements for each attribute. For example, the variable “firstName” must not be null and must be less than 10 characters. Use case testing ensures that the user can perform the necessary actions within the program. In the program, the user must be able to update their information. I was able to implement JUnit testing in each service class to ensure that the behavior of the methods within the classes produces the proper outcome when the user edits their information. In regard to white box testing, I implemented statement testing and coverage to ensure that each statement in the program is executed at least one time. I did this by using JUnit testing in each milestone and checking that the coverage levels were 100% for each class and method within the class. I also implemented decision testing and coverage to ensure that every possible output from a decision point is executed, ensuring that various input behaves as expected. For example, in each class, I made sure to create a method which would output an exception if the user entered an invalid input. If the user entered a valid input, the variable would be updated.

Some other software testing techniques that I did not use include boundary value analysis, decision table testing, and state transitioning testing. Boundary value analysis is used to test input to ensure that it is within the boundaries. For example, if the lower boundary for a test case is 5 and the upper boundary is 15, the input must be between 5 and 15 in order for the program to accept the input. Decision table testing is used to define the functions of the system and the conditions under which each function operates. This ensures that each combination of those conditions produces the expected outcome. For example, there is a rule that states that all conditions must be true for an action to occur. If all conditions are not true, some other action occurs. State transitioning testing is used in systems where input conditions produce actions within the program. For example, a user can transition the program between time mode and activity mode. This testing ensures that the system performs the correct action for each combination of input.

Working on this project allowed me to gain a better understanding of the importance of testing during the software development process. It is important to appreciate the complexity and interrelationships of the code in order to fully understand how to properly conduct JUnit testing. One way I employed caution in my code was to implement public or private objects, depending on the way those objects are meant to be accessed. Specifically, I used public objects to identify the classes and methods, and I used private objects to identify variables such as name, id, etc. I also implemented caution by using JUnit testing to make sure the code deployed properly. When testing code, it is important to limit bias and have an open mind to the different ways that code is written. When testing your own code, it is very possible to be biased because you may not see the flaws in code that you wrote yourself. It is good to have another set of eyes be able to look through and review the code for any minor flaws which could cause issues in the future. As a software developer, it is extremely important to be disciplined when it comes to being committed to writing quality code. In doing this, you will ensure that each and every project works properly, is easily understandable, and can be re-used. Cutting corners can cause issues that aren’t immediately noticeable and may not show up until the program is deployed and used by real users. Doing this will hurt your reputation as a software engineer and can have a negative effect on clients and users. I plan to always make sure to practice proper coding techniques and make sure that my code is technically sound each and every time that I write a program. I will do this by using in line comments, making sure to keep security a priority, testing my code, and reviewing my code for any possible errors that could cause a defect.