Tessa Parker

CS-320

02/18/2023

Project Two: Summary Report

I was tasked with creating an application that would allow for the creation of contacts, tasks, and appointments. I used JUnit testing to ensure that the program was functioning as intended and successfully met all specified requirements. I made sure to follow the specification document (the rubric for project one) to build this program. For instance, let us look at the requirements for the Contact class. We were asked to create a new contact with a unique identification number, name, address, and phone number. We needed to ensure that we could update any of these fields except for the unique contact identifier and delete a contact using their identification number. I tested the functionality of all these requirements by creating JUnit test cases that accounted for every method written to accomplish these tasks. I also ensured that invalid input was properly handled by adding illegal argument exceptions and asserting that these exceptions were thrown using invalid test variables. For example, we can look at lines 44 and 45 in my appointment test class. In these lines, I assert that an illegal argument exception is thrown if the appointment identifier number is either null or not ten characters in length.

I ensured that my tests were effectively accounting for all cases that we might encounter by ensuring that every branch of every method was called. I was able to achieve 100% coverage with nearly all my test cases. The unit tests that I performed were all successfully completed, with no errors that prevented their completion. I ensured that my code was technically sound by testing all methods, including the creation, addition, and deletion of items from a list. This can be seen in my appointment service class in the test of the appointment deletion method (spanning lines 50 – 66). In this method, I added several items to the list of appointments, ensured that the method was properly searching the list for a specific appointment, and that the appointment had been successfully deleted.

I also made sure that I was trying to code efficiently. For instance, in the appointment service test class, I did not see the need to create a separate test case for the method used to search for an appointment within the list. We were instead able to provide full coverage of this method by calling it within other test cases. This can be seen in lines 50 – 64. In the test case for the method to delete an appointment, the method to search for an appointment is called and provided test coverage. By coding tests in this way, we can streamline our test classes for better legibility and leave less room for error.

For this project, I employed unit testing. To be more specific, I used JUnit 5 to test all the methods created for this project. Most of the testing methods I used for this project were assertions. This means that if I wanted to assert that a value was invalid, I would use an assertion that an exception was thrown. Likewise, I would use assertions to confirm that valid input was properly handled in the desired way. For instance, asserting that a contact with a valid identifier, name, address, and phone number is added to a contact list with all the appropriate parameters intact, updatable, and retrievable. I did not use many methods that are available through JUnit 5. As an example, while I did set up variables before each test case, I did not utilize before all, after each, or after all methods. I did not feel that we needed to perform a ‘tear down’ of information in this scenario and wanted to keep the code as efficient as possible. However, there are certainly scenarios in which these methods would have practical uses. A developer may wish to use an after all method in a large, complicated test class to ensure that no unnecessary data is being stored within the program.

In the creation of this project, I attempted to employ a cautious and detail-oriented mindset. The program was complex and there were a lot of interrelationships between different methods under test. By trying to understand how the methods nest into one another, I streamlined my testing approach and ensured that each part of the overall program was working as intended. As an example, I will refer to a test method I referenced earlier in this report. Through testing the delete appointment method in the appointment service class, seen in lines 50 – 64, we were able to call other methods and provide coverage, ensuring functionality in practice.

It is entirely possible that I was biased in my creation of this program and the program’s test cases. I knew how I intended for the program to work and tested according to that. I attempted to limit bias by accounting for outlier cases in any way that I could imagine. This can be seen in the creation of invalid parameter values for values that are either null, too long or too short in the contact test class. At first, I believed that the creation of invalid parameter values may have been extraneous because I could have tested exception branches taken for null values with relative ease. However, I found it important not to cut corners and practice discipline by making the extra effort to create test values that were populated inappropriately. This ensures that both conditions for an exception branch will be taken at some point, confirming that the program will catch these errors. If the program allowed for invalid input, it could cause issues with the overall functionality and incur technical debt as it may be difficult to pinpoint the issue further down the line.