Project 2 By Kirk Mashburn

The code I created in each of the milestones for Contact Service, Task Service and Appointment Service all followed the guidelines established in the rubric. Each milestone asked for code to validate inputs, this was done by creating a character limit and checking for null values. The service milestones also required the ability to add, delete, or change the data used for these services. The code for those services required standard error checking in the form of ensuring that no duplicate services could be added and that you could not delete something from the system that was not there. When updating a service, the character limit and null value error checking was also applied. This means that when updating a service, you could not enter values that the already established logic did not support. That means that all updated contacts, services and tasks followed the specified characteristics given in their class constructors.

Each test class used the Junit library for Java which allowed for the creation of test classes. These classes made use of the Test and assert features of Junit. Those features allow users to test if functions within the service class functioned correctly and used the asset feature to check if the error catching was working as expected. The code also made use of a custom thrown error to help assist with the assert function while testing. The testing cases included case limits, null inputs for the Contact, Assignment and Task class. These included the getters and setters methods for them as well as their constructor. The Services classes checked their add, delete and update features using the same assert and test features. Those classes also used a built-in hash map to assist with testing and part of the Junit testing was ensuring each service object was properly created by using the BeforeEach feature.

Best practices were followed to the best of my ability this was done by adhering to clean coding practices. Those practices were displayed by making user of smaller methods that were typically 2 to 6 lines in length and using the appropriate statements when necessary. No loops or other types of iteration statements were for these classes which help maintain a faster run time. Defensive programming was also used in the form of public methods to access other internal methods. This helps maintain proper encapsulation and security within the program.

The techniques employed for this assignment were creating methods in the testing class to cover a wide range of expected errors within the program based on the expected coverage within the guidelines and rubric. Since user input was not a part of the project that kind of error checking was not included. The tests were mainly centered around ensuring that the object created could only be created if they fit the mold so to speak. This brings the main error checking within the scope of the class itself which reduces conflicts in the overall program. Modularization of the code allows for an easier fix of the program if an error does occur. It also ensures that if an error does happen, then it mainly affects the scope of the class itself. This helps prevent crashing of the program. It is more acceptable for a program to spit out incorrect data than to hard crash. Ensuring that the overall function of the program is my mindset when performing Junit testing.

The major concern I have with testing is preventing a breakdown of a program. I seek to eliminate errors that would crash the program first. When building the program, I also work to eliminate the most obvious errors given in the outline. If a part of the code requires a specific input, I would work to make it wo where the input of bad data would redirect to some other portion of the program like a menu or back to the input field. The displayed error message should give the user some little chirp of how they are putting the data in incorrectly so they can reenter it properly.

I understand that this process of testing code kind of relies on the assumption that I will always catch the cases of incorrect input of a given program which can lead to a bias when testing. When performing review for larger programs it would help to think outside of the box as far as input cases are concerned. You will also need to review what data types of the functions in the classes are manipulating to minimize data conflicts. It helps to brainstorm about possibilities not outlined in a document early to eliminate the problems before they arise.

Mainly as a process of my testing I hope that by eliminating all the obvious answers early if an obscure error does come up it will not be due to obvious programing mistakes such as syntax errors, input errors, data mismatches and such. Another good discipline to keep is keeping your code clean and easy to read with comments explaining what each function does. For larger programs having some sort of header at the top explaining the purpose of the class you are working on also helps. I try to think about how the code is supposed to function if it is completed. I would like to have all the working parts though out and on paper before I start programming.

I want to avoid having already written most of the program and then having to go back and rebuild it form the ground up. This is another case where keeping the program segmented also helps. If something needs to be changed or worse case completely re written, then having the part of the program you are working on separated in it’s own class helps ensure that the other portions of the code still work over all and can be recovered by only making minor changes if they must accept data from a changed or entirely new class. If in of the task functions had to change their data, they accept then only minor changes to the service would have to be made to support it. The same thing would also occur with any other portion of the program.

I believe code in the project followed the best practices of code writing, keeping the code clean and segmented with defensive programming in mind. With Junit testing all the input cases were tested via the constructor and setter methods and each of the service functions were tested in a way that eliminated all the obvious scenarios when adding, deleting or changing data.