**Project Two Summary**

The client has provided me with software requirements of character length, not null, unique attributes, and the structure of each class with what actions the class needs to perform. Firstly, a means of creating the variables and retrieving them is required in set/get mutators/accessors, and to then test the expected output changes. Each JUnit test focused on one requirement at a time and was intended to be a concise test. For example, in the test files, the variable tests follow a repetitive cycle: length of first variable, empty first variable, null first variable. These requirements were on every variable and followed the same pattern, only a different length of the string was possible and tested. AppointmentTest.java tests apptID with a “testApptIDToLong()” followed by “testApptIDEmpty()” followed by "testApptIDNull()” for all software requirements of the apptID variable established by the client. This proceeds to be done again for “ApptDate” and “ApptDesc” in the exact order of ToLong(), Empty(), Null() except for ApptDate having an additional requirement of a date being entered in the past or in an invalid format. Therefore, an additional test is performed at the beginning for an early date and at the end for invalid format with a name scheme of the test to accurately describe the test.

At first glance, it appears that my JUnit tests do not provide enough coverage, however each test file has its linked file that is being tested. TaskTest.java is linked to Task.java like how TaskService.java is linked to TaskServiceTest.java. The test in TaskServiceTest.java will not test any of the functionality or requirements in Task.Java. Going through the element inspection of the JUnit test coverage test performed, the percentage is considerably low due to the evaluation reflecting the project as a whole and not the two linked files. The overall project shows a coverage of “35%” but in this example of TaskTest.java, the coverage of this element shows “88.1%” for Task.java and “66.7%” for TaskTest.Java. The percentage is derived from the number of missed instructions to the number of covered instructions. Upon further searching of the elements, Task misses 5 lines of instruction for each of the set mutators on the attributes which comes from not testing all possible branches of an OR statement. As the programmer however, I am aware that the testing covers both sides of the OR independently and it is not possible to test both the null and length for failing simultaneously. In the success test of the program, it is tested for both OR statement passing in the set mutator and therefore that branch of coverage is false. Therefore, I believe that coverage is properly above 80% under a detailed analysis of what the test performs in each line of code in the main program as this applies for all the Test programs including the Service files.

I ensured a technically sound program in consideration of software requirements by partitioning the possible inputs of the users into the possible ways of passing and failing the validation checks. A string of 10 characters and cannot be null needs testing for a test string of anywhere between 1-10 characters, greater than 10 characters, empty strings, and null variables. Under AppointmentTest.java, tests are structured and consistently used for each variable to check how the program handles good and bad data. And the tests are used minimally to avoid clutter and repetitive input checking. In the new Appointment constructor, the first variable being tested is the first variable in the constructor and tested for all the above listed outcomes. Going down the line of tests, it’s a repetition of the previous tests in the same order but for the next variable in the constructor with any additional tests based on the requirements of said variable. Since the constructor contains the same lines of code for input validation as the set mutator, only the constructor is tested with an assertion throw for invalid inputs and thus efficiently minimizes the size of each test.

**Project Two Reflection**

The main and only software testing technique applied to this project was Assertions. Assertions are designed to verify that the expected results are what occurred in the program. I my program I focused on the expected result of exception throwing for each invalid input since the client did not specify what needs to occur with each invalid input. This makes testing easier as each JUnit test is an assertion throw on every other input other than a valid input. I’m sure some form of dynamic runtime testing could have been useful for testing parameters and data validation, however, since this program does not require an instance being run of the program that would have been additional unused clutter. Having changed the approach of the program to request the user or data input to try again upon a failed input, a dynamic test would have been effective with the intent of being able to test all invalid input attempts back-to-back with a larger variety of string combinations in attempt to attack the code. This benefit of testing could help with testing and implementing security measures against attackers via injection.

Caution in testing was handled to an average extent as the complexity of the code was not extreme nor was security much of a concern in the current program. The most sensitive data transferred is first and last name, as well as the potential information provided in the description. Therefore, caution in testing for data injection, authentication, encryption was not implemented or focused on for these unit tests regarding the provided software requirements. The accuracy of data is tested via caution as assertions are done to verify that strings are being stored and overwritten correctly, as well as the format of certain variables being proper. Appointment class with the date type variable requires a very strict input format, and some testing is done to test for bad data formats but was not tested to the fullest for possible data corruption or injection attempts. Appreciating the complexity of the program helps the tester understand the extent of testing required on segments of the code. Knowing that the date variable type has to be of a specific format, I the tester should have provided more effort into ensuring that format is followed and labeled for the client or future programmers who work on the code in the future. Should the program have complex interrelationships, small mistakes in the program can result in even greater errors and bugs leading to unpredictability of functionality. The sooner these potentials are caught, a better future scalability can be achieved as the earlier versions of the code are soundproof.

Bias is unavoidable as a programmer and as the tester for my own program. Focusing on creating the code with frequent reminders of software requirements helped minimize bias as I ensured that testing was kept limited to the provided data validation. Being aware of bias allowed me to understand that it is possible for me to miss important aspects of the program as I have noticed with the AppointmentTest.java file. In earlier programs of ContactTest.java and TaskTest.java, I did not test for empty strings but became aware of the possibility of input in Appointment.java as I provided statements to handle that expectation. As a programmer I had confidence I was covering requirements to the best of my knowledge until I encountered the reflections, to which demonstrate to me that there should be concern when responsible for testing your own code. A second view of the program is essential towards eliminating possible bias and to catch errors in which the programmer may have missed in the construction of the program.

The client had a clear expectation of what the program needs regarding all the attributes of each class and data validation for each variable and method. Being disciplined to not cut corners and to properly test every aspect of the requirements is essential to meet the client’s satisfaction and reduce future costs and time for both the client and potential programmer’s side. For example, if the program that the client wishes to have created is found online in a similar structure for free use, it is important to fully understand the intent and purpose of the program used in reference to all methods applied. I was uncertain about how to store the date variable for appointment date and looked for resources online on the import functions on how to store and compare dates, however I had concluded that storing the variable as a date type was not realistic and properly stored and manipulated the variable as needed while still meeting the client’s requirements. There existed similar and complicated methods online on how to structure the storage of the date type, however I felt it unnecessary and risky for application of the data.