**Effective test and alignment to the software requirements**

As the first and one of the most important steps, I tried to prepare the requirements carefully and understand the expectations of the customer (here SNHU assignment). This step is very important as any misunderstanding could be very costly and the whole code may need to be revised or rewritten; I have personally experienced it in a VBA project and had to spend twice as much time as planned to fix issues that arose due to miscommunication, and not fully understanding the customers need, or not spending enough time to clarify ambiguities. Based on that experience, I learned that spending more time on asking clarifying questions at the beginning not only is not a waste of time, but also it saves a lot of time in the long term.

As a specific example in this assignment, I made a mistake at beginning for validation of the data entry in almost all fields. For instance, there was a requirement that contact ID String that cannot be longer than 10 characters, and at the beginning I used >10 which only allows IDs that are 9 characters long or less and not 10 characters maximum limit per requirement, instead of >=10 which allows 10 characters and less; (the correct code was: **if** (contactId == **null** || contactId.length() >= 10) {some code} and I fixed this issue for all other fields such as description, name, etc.).

To achieve more efficient tests, I used the JUnit Code Coverage Report and while the coverage from week five assignment improved from 67% to 88.2% for week 6 assignments (which is above 80%), I believe that there is still room to improve. Efficiency increases by experience and studying best practices such as removing redundant or cloned code, writing more tests, and looking at the tests that are authored by Junit developers and resources provided by them. The naming of test methods could also be chosen to show exactly what is the intention of the test (as one of the best practices), however I used shorter names to keep it succent.

**Efficient and technically sound code**

To achieve a technically sound code, I tried to provide descriptive exception error messages when errors are thrown which provides general and pre-defined messages. For instance, I used this message in in the Contact creation test unit when the address is longer than 30 characters or the field or input is left blank or empty "Address should be 30 characters or smaller and cannot be blank/empty; if it is, an exception should be thrown as in this example."); which not only makes it easier for other developers to follow the code when revising it, but it also provides better explanation to the users about the problem, and what they need to fix it (here the fix would be to make sure address is entered, and it is less than 30 characters). Although the 30 character limitation is a requirement by customer, I would have talked to them and explain that there may be some addressed that are longer than 30 characters and limiting the entry may not capture accurate information from user and would have suggested them to do more research and set a limit that fully covers the majority of situations; however for the purposes of this assignment, I just implemented what was asked in the requirements section.  
It has been one of my main goals to write short succent and efficient code but as a new developer, I know that there are a lot to learn and I can only succeed with lots of practice, study, and looking at the codes that have achieved to do the tasks in faster and better ways, with fewer lines. Also, in choosing variables and object names, I tried to be as descriptive as possible. For instance, in addAppointment() method (Boolean), I named the returning variable as idAlreadyExists so that it is easy to follow, and tried to be consistent in other classes and methods, however I during writing this reflection paper, I realized that I have missed to follow this practice is one or two methods and it was a learning opportunity for me to double check the code line by line, before deployment (here submission).

**Techniques that were used**

I used different approaches for testing. The first step was static testing in which I read the code line by line, paid attention to the warning or error messages, and strived to find any ambiguities, problems, or errors. Then I tried to understand the root cause of the issue. The most challenging part for me was the date comparison for setting the appointment date, which per requirement and logically, an appointment cannot be scheduled in a past time. First, I created a draft code and tried to compare the results using (which submitted in with week 5 assignment as DrafCode.java class but removed it for week 6/Project One assignment). I did not receive any error as I was expecting when used a past date such as 2023/04/01 not realizing that 04 stands for May and not April, as the months started with 0 to 11. During this process, I had to use different classes such as Calendar, simpleDateFormat, Date, GregorianCalendar, DateFormat and LocalTime and learned a lot about each of them, but finally used GregorianCalendar for the assignment. It took a while for me to understand the basics of each class and how to implement them for the purpose of the assignment, however the time spent in learning them well worthed it.

Junit was the main technique used in this assignment and basically, I entered what was expected in the unit to be compared against what the output of the testing method would be.  
the other method that seems to be applicable to this assignment was white box testing, as I was aware of the expectations and requirements, as well as how to implement the code to achieve those goals.   
One extra implementation that I performed was creating random 1o character IDs that can be used by any of other methods that use Id to create a new Task, Appointment or Contact. I really enjoyed it, and you can find details in RandomStrGenerator.java which was submitted with week five assignment.  
There are many other testing methods and techniques that I did not use in this assignment, mainly because I was not familiar with them, and I believe some of them were not applicable to this project. For instance, black box testing which is also known as functional testing, will conduct tests based on the information from the specification and developer in charge of testing should not (or does not) have access to the internal source code itself and that is why it is called black box testing in which the internal mechanisms of a system is not known to tester and the focus is only on the outputs generated in response to selected inputs and execution conditions (Nidhra et. al, 2012). This method was not applicable to our project, as all the internal mechanisms were known to the tester. I briefly mention some other methods and definitions that were not used in this project. Integration testing verifies the compatibility and consistency of some methods, classes, modules, or groups or when they combine to form a whole application was not used in this assignment. Security test was another method that was not employed; security tests are performed to identify weaknesses and vulnerabilities of the code, system or application.

**Mindset**  
The whole process of learning about software test automation was both challenging and satisfying for me. I am my own harshest critic and did not feel biased during designing tests, and despite getting frustrated when the test failed and I had to find out why, I never felt biased to defend my code. That being said, it could be substantially different when someone else is testing my code or in case my role is to test another developer’s code.

|

**Discipline**

This is an area that I need to improve and despite trying to be consistent and write easy to follow code. During the last year, my codes ended up to be spaghetti code and since then I have been working hard to create separate classes, functions, methods, and packages to keep the code clean and stay disciplined (depending the programming language that is being used).

References:

Nidhra, S., & Dondeti, J. (2012). Black box and white box testing techniques-a literature review. *International Journal of Embedded Systems and Applications (IJESA)*, *2*(2), 29-50.