Project 2

Santiago Ramirez

06/17/2022

Dr. Cross

**Project Resources:**

ContactServiceTest.java:

Figure(1-1)

Graphical user interface, text, application

Description automatically generated

ContactTest.java:

Figure(1-2)

Graphical user interface, text, application

Description automatically generated

Coverage Contact:

Figure(1-3)

Graphical user interface, application, table

Description automatically generated

Figure(1-4)

Graphical user interface, text, application

Description automatically generated

TaskServiceTest.java:

Figure(2-1)

Graphical user interface, text, application, email

Description automatically generated

TaskTest.java

Figure(2-2)

Graphical user interface, text, application

Description automatically generated

Coverage Task

Figure(2-3)

Graphical user interface, application, table

Description automatically generated

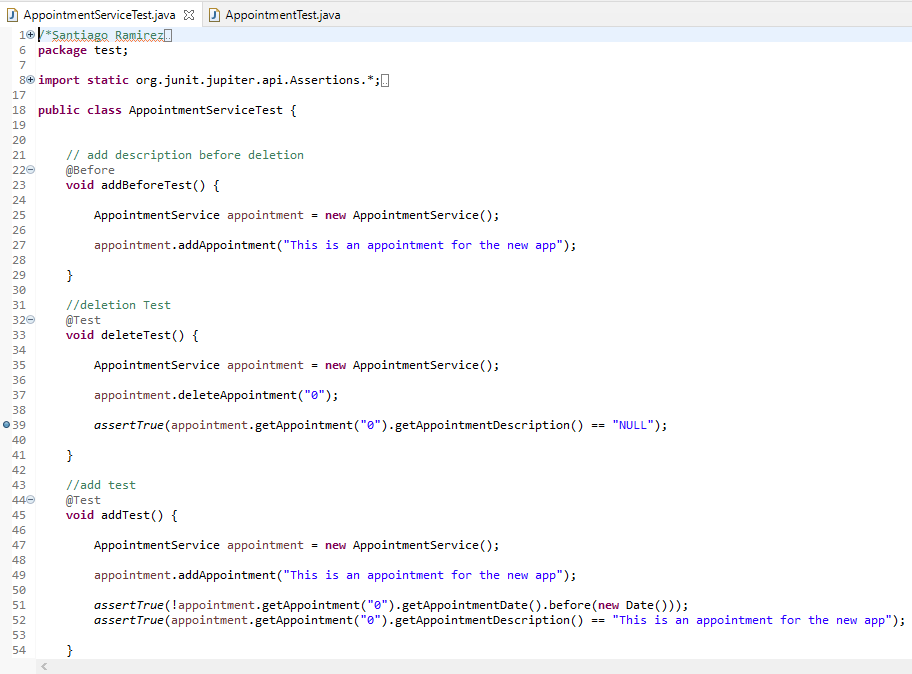
Figure(2-4)

Graphical user interface, text, application

Description automatically generated

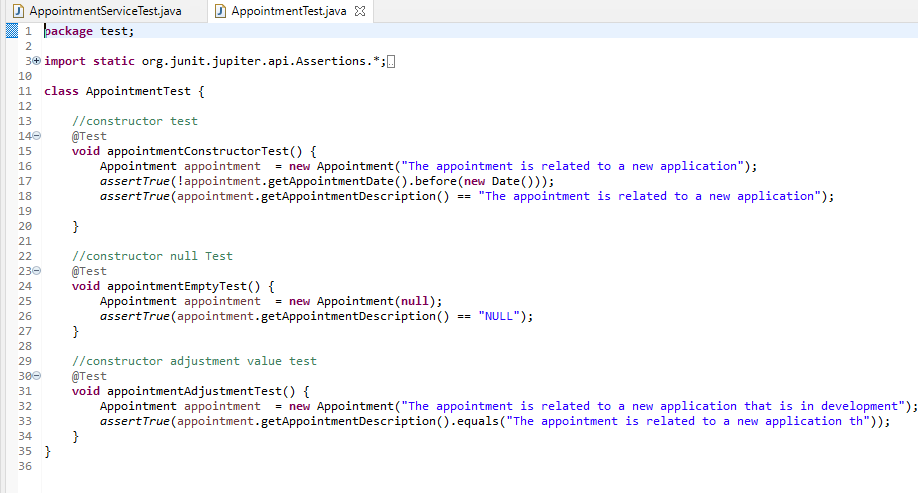
AppointmentServiceTest.java

Figure(3-1)



AppointmentTest.java

Figure(3-2)



Appointment Coverage

Figure(3-3)

Graphical user interface, text, application

Description automatically generated

Figure(3-4)

Graphical user interface, text, application

Description automatically generated

**Software Requirements**

My testing approach is fully aligned to the software requirements because as we can determine based on the previous pictures in TaskTest.java ContactTest.java and Appointment Test, they are all ensuring that the requirements of the quantity are met and these requirements are tested with the function taskContructorTest() in line 13 of TaskTest.java(2-2), in function validationContactTest() in line 17 of ContactTest.java(1-2) and in line 15 of AppointmentTest.java(3-2) we use the function appointmentConstructorTest() . Additionally, the requirements of deleting adding and updating are also met in both TaskServiceTest.java and ContactServiceTest.java, we can see that in ContactServiceTest.java line 30 it meets delete requirements with function deleteTest(), line 44 ensures that we check adding and updating contact by using function addAndUpdateTest()(1-1), meanwhile in the TaskServiceTest.java we check for delete requirement with the deleteTest() function in line 21, in line 32 the add and update requirements are met with addAndUpdateTest() function (2-2). Furthermore, we accomplish the requirements of deletion and adding of appointments in AppointmentServiceTest.java(3-1), by implementing the function deleteTest() in line 33 it allows the JUNIT test to check if deletion of appointments are working properly as well as the function addTest() in line 45 which purpose is to check if adding appointments are accurately set in the systems own database.

**Quality of The JUNIT Test.**

The overall quality of my Junit test is effective since we can see that in Contact Service Coverage picture there is somewhat of a high not perfect score on both the ContactServiceTest.java and the ContactTest.java and so understanding the percentage of each one is important as long as it has an 80% or higher which in this case ContactTest.java coverage(1-3) has 100% coverage which means it should have technically covered all the functionality of Contact.java , while the ContactServiceTest.java(1-4) has around 91% which is a high percentage that is qualify to ensure that the code is properly being tested and should cover most of the areas in ContactService.java. In the taskTest coverage(2-3) we can determined that the coverage has around 93% of the functions in task.java being properly tested while in the taskServiceTest coverage(2-4) the coverage was around 88% which will most likely cover most of the functions in taskService excluding some of the functionalities that taskTest.java had cover previously. Finally, the appointmentTest coverage(3-3) works similar to the other test but this time we cover 92% of the appointment.java functionality and 80% of the appointmentService with the appointmentServiceTest coverage(3-4). The reason some JUNIT test have less coverage then others is because some of the functionalities that are being use in a certain class have already been tested previously with another test so the coverage is basically being divided and therefor that means that the functions in every class are effectively working.

* How did you ensure that your code was technically sound? Cite specific lines of code from your tests to illustrate.

**Technical Sound**

I ensure that my code was technically sound in all my projects by allowing the projects to ensure that the code be adjusted if any requirements are not met and we can see this in TaskTest.java line 21 and line 29 which the functions are TaskConstructorFailedTest() and taskConstructorAdjustment() which ensures that adjustments are well develop in a way that the values are not over their limit and also to include information if there is no know information in the variable(2-2). Furthermore, in ContactTest.java line 26 function defaultValuesTest() and line 35 function adjustmentValuesTest() ensures that the information is adjusted based on quantity similar to the previous Task Service project(1-2). In AppointmentTest.java(3-2) we develop the same functionalities similar to previous testing which in this case will be in line 24-47 theappointmentEmptyTest() allows us to understand if empty values are being implemented with information as well as the appointmentAdjustmentTest() in line 31-34 which makes it possible for adjustments to happen when there is a value that is higher than the limit of characters.

**Efficient Code**

To ensure that my code was efficient in all projects I took into consideration trying to group up functions in the testing most of the similar functions that have basically the same attributes but with different values such as in within function addAndUpdateTest(), in lines 46-51 I took into account if the function addContact() worked currently then we can used that same function to update its values in lines 53-56 (1-1)we can see that I used all of the update functions that are being used in ContactService.java and in this way it makes things more efficient. Similarly, in TaskServiceTest.java we also took into account the grouping of similar functions that have the same attributes but with different values within the function addAndUpdateTest()(line 32), in lines 34- 37 we ensure that addObject() function is working and then we implement the update functions in lines 39-44 to check upon if every object has efficiently been updated in the object list(2-1). In appoitmentServiceTest.java(3-1) there are not that many functions similar to previous test that will required the collection of functions within one test function but we did do a step by step basis of testing in order to achieve a more efficient way of covering the effectiveness of the functions and we did this in line 21-27 of appointmentServiceTest.java(3-1) by implementing an addbeforetest() which purpose is to allow the system to understand that this function goes before the next function which is deleteTest() in line 33 and the reason we did this is so that the test doesn’t accidently erase the appointments being added in function addTest() in line 45. In conclusion the grouping of functions as well as the readability of the code helps it be efficient based on its quality.

**Testing Techniques Used**

All the milestones that I had built have used similar techniques and the techniques that were used in each milestone are:

Boundary Value analysis: which is a technique that will basically check if the value is at a correct range and if the functions are correctly adjusting itself to incorporate these limits that are part of the coding (Hambling, et al. 2015).

Decision Testing and Coverage: Is a technique that covers all testing parts of the if and else statements that are incorporated within my code and examples of this will be when there is an exception when the characters in a variable are not in range so the code itself will try to change to “NULL” if there was no value or if the range is too high it will try to test if the code adjust itself to the correct range (Hambling, et al. 2015)..

Structure-Based(White Box) Technique: Is a technique that basically incorporates the inner structure of the code and test if the particular elements within the structure are correctly exercised this is opposite to the Black Box technique that does not cover the structure of the code(Hambling, et al. 2015)..

Error Guessing Technique: Is a technique that uses the intuition of the tester and in this case my code does not have any error checking so its my job to identify any problematic errors within the code since my code does not necessarily give out any errors for any specific part instead it adjust itself in case of an errors but still because of this I need to identify any additional errors that might occur based on intuition(Hambling, et al. 2015).

**Testing Techniques not used**

The testing techniques that I did not incorporate in my testing are:

Input partitions: This technique testing will check upon the input validation of the code, and it will most likely make sure if the system is not incorporating wrong data types such as integers, characters, string, floats and/or Booleans. This technique was not incorporate in my code since most of the inputs are in string format but either way it is better to check if it is invalidating string format for all values (Hambling, et al. 2015)..

exploratory testing: This technique testing will check if there are other cases that are not cover within the testing itself so will use other alternatives to check upon errors or invalid output within the code and this technique was not incorporate in my code because I had no reason to check if the code had any alternative errors since it is more a simple and non-complex structure (Hambling, et al. 2015).

**Testing Technique Practical Uses and Implications**

Boundary Value analysis: The practical uses that this particular technique can be use in a project or situation will be if a project wants users(actors) to have a limited amount of character or value within a particular range in input and because of this it is important to use boundary value analysis which will check if the boundaries are been taken care of if the user for some reason ends up overriding the system with too many character or not providing any info with no-know values.

Decision Testing and Coverage: The particular use that this technique can be incorporate in a project will be if the structure has if/else statement, and/or for/while loops which in theory almost all structures are built upon these built-in functions so making sure that the test covers these particular parts are important in every project.

Structure-Based(White Box) Technique: Projects that are more center in testing that requires mostly the programmers over the tester will use this technique in a project because as explained before the white box technique purpose is to identify any issues within the overall inner structure of the code.

Error guessing technique: This technique will not work in most situations since a project that is built with many programmers as well as the magnitude of the code can make it challenging for error guessing techniques to work since it is better within a code that has less to analyze since it takes into account the intuition of a programmer or tester.

Input partitions: This technique can be used in a project in which a lot of data type inputs have to be validated such as identifying if the input has specifically used the correct type of data so the user(actor) does not accidently type the wrong data type or the programmer didn’t incorporate what data type he is referring to.

Exploration Testing: This technique can mostly be used in projects that can sometimes have future updates in the or even when the structure itself is huge that it needs to be understood in depth so that the program does not encounter any errors specifically logical errors.

**Employing Caution**

Acting as a software tester, I employ caution into the testing by ensuring that every important functional factor of the code has been properly inserted into the test and in that way, I was cautious of making sure that the code was consistently working with no errors an example of this would be the extend that I applied various methods in which the code doesn’t encounter an error through the means of adjusting the value and providing data if the data itself was empty . It is important to appreciate the complexity and interrelationships of the code because the complex connections that each function has in the code will define how properly the system will work and so removing errors in a complex system can help all code work properly. An example of ensuring that there is a respect to the complexity and interrelationship of a code will be to fully test each of these functionalities separate and see if they are coexisting together similar to how I tested the add functions with the delete function since add has an interrelationship with the delete function to work properly so appreciating these relations makes it possible to test the code correctly.

**Limit Bias**

As the software developer the way I can limit my bias while reviewing code is to ensure that not only the important code that make the system function are being tested but the small ones such as implementation of data limits or even if the dataset is being properly suited for the code but overall I believe that bias should not be a concern if I am the responsible for testing my own code since I understand my own code and I understand what makes it work correctly but an example of where bias review code could happen will be if we were to give our own code to a tester it will cause some biased to occur between the tester and the code since they are more likely to pick the most important code to review in the system and therefor undermined many other codes that might need the same treatment of error checking and in the end will be harder to find errors and time will reviewing code will be consuming.

**Importance of Discipline**

Being disciplined in the quality of developing a software is one of the most important aspects of being a software engineering professional because cutting corners when writing code can make code look too simple, less reliable, slow and potentially remove many features that might have made the system better. Additionally, when testing code, it is always important for the code to be fully tested because a code can’t be fully trusted with only a couple of code being tested because it might cause a logic error that might not appear the first time. To avoid technical debt as a practitioner in the field I will ensure to follow all the requirements that are needed for a system in future projects an example of this is how we had a task to do for each of our milestones and the projects that needed to be developed, we needed to follow every step and ensure that all requirements are met and because of disciplined I did not cut corners to make the code work but instead took action allowing the code to fulfill each requirements.

Reference:

Hambling, Brian Morgan, Peter Samaroo, Angelina Thompson, Geoff Williams, Peter. (2015). *Software Testing - An ISTQB-BCS Certified Tester Foundation Guide (3rd Edition).* BCS The Chartered Institute for IT. Retrieved from  
https://app.knovel.com/hotlink/toc/id:kpSTAIST01/software-testing-an-istqb/software-testing-an-istqb

Ramirez, S. (2022, June 17). JAVA. *AppointmentService*. Retrieved May 29, 2022. From Eclipse.

Ramirez, S. (2022, June 17). JAVA. *ContactService*. Retrieved May 29, 2022. From Eclipse.

Ramirez, S. (2022, June 17). JAVA. *TaskService*. Retrieved May 29, 2022. From Eclipse.