Grant Sorenson

10/23

CS-305

Project 2: System Summary and Reflection

## Summary

While creating the unit tests for each feature I thought it was important to maintain a good idea of the project requirements and necessary system functionality. It was also important to maintain good security practices like proper error handling and catching throughout the project which was enforced by certain tests throughout the main object classes Contact.java, Task.java, and Appointment.java. The tests written ensure 85% coverage throughout the project, but optimizations like not using arrays and further exception checks could improve the efficiency and security of the program.

The Contact.java and ContactService.java classes required more tests due to their increased complexity. The contact class held members which had length and content requirements as well as certain functionality methods that included the manipulation of certain values. This required tests to be created that checked the validity of these functions and maintained the desired output was within the required parameters or it produced an exception. The ContactTest.java tests included an extensive testing of the input parameters as well as add and delete method functionality. Besides the testCreationTestSucceed on line 23, The following lines 28-103 include tests that check for their specific invalid input requirements, like ContactIDTooLong in line 29 and ContactIDNotNull in line 37. These tests check that proper error handling for invalid input occurs and the program does not encounter any errors or hangups. Tests from lines 104-131 include assertions to ensure that the member variables were changed to the correct valid value and no errors were encountered. The ContactServiceTest.java test class included tests that ensured proper contact addition and deletion from lines 20-43. These tests could have been expanded to incorporate stressful environments and increases error handling. The main testing for input validity is done in Contact, so the only content checking the ContactServiceTest class has is from lines 45-64, which asserts that values are changed to the correct input values and no errors occur. This class could have also benefited from further error checking for invalid input data, but it would have been redundant due to the tests within the contact class. Overall I think the tests written for these classes show an acceptable amount of coverage and proper functionality.

The Task.java and TaskService.java classes were slightly easier to write tests for because they had less member variables that needed to have manipulator methods. These classes still required the proper validation and error handling, but on a smaller scale. This allowed me to implement more extensive add and delete tests to ensure the system would function and correctly handle errors in all situations. The TaskTest.java class begins with a creation test to ensure that the object is created. One member similar for each class is a unique identifier which cannot be changed, cannot be null, and must be retrievable. The next tests from lines 26-42 ensure that a valid sized ID is input and that it can be retrieved by a supplied member get function Contact.getTaskID(). The tests from lines 43-85 ensure data validation within the member variables that are able to be changed. These tests accurately cover the Task.java class and ensure that invalid input is handled correctly, and the program doesn’t run into any unwanted behavior. The TaskServiceTest.java class begins with a basic creation assertion to ensure that the object was created successfully. The following lines 26-56 fully test the functionality of the TaskService Object and ensure that additions and deletions to the TaskService object function correctly in all scenarios. For example, the testDeleteInvalidTask successfully returns an exception that a task that doesn’t exist cannot be deleted. The remaining tests from lines 56-102 ensure that the data fields description and name can be properly changed and no errors arise from invalid input data. These test do a good job of accurately covering the Task.java and TaskService.java classes by ensuring errors are handled properly input data is validated. The additional checks to the TaskService object’s delete and add methods increase the programs reliability which still manages a coverage of over 90%.

The Appointment.java and AppointmentService.java required the least amount of features. The AppointmentService object didn’t require any methods for manipulating the Appointment objects members, which reduced the need for extensive testing and saved time on the overall project. While creating tests for this class it was still important to ensure errors were handled properly and the input validation was correctly sending exceptions. The AppointmentTest.java class includes checks to ensure proper creation and data validation for the Appointment object from lines 15-43 in tests testAppointmentCreationSucceed and testContentCreationSucceed. To ensure invalid content isn’t accepted testInvalidContentSucceed ensures that exceptions are thrown for incorrect values and no changes to the object is made. This is important for maintaining correct data and ensuring all the data meets the requirements. Tests including the validation of updating the description and date fields from lines 66-85 maintain proper error handling standards. The AppointmentService class tests the creation of the objects and ensure proper content validity from lines 15-34 in tests testAppointmentServiceCreationSucceed and testAddAppointmentServiceSucceed which also asserts that input select values are not null. From lines 38-71 the tests check the add and delete methods for the AppointmentService object and ensure they function correctly in every situation. Overall the tests for the Appointment and AppointmentService object cover over 90% of the classes and do a good job of making sure data is valid and exceptions and errors are handled correctly.

## Reflection

The software testing techniques used during this project included a mix of White-box and experience testing. White box testing was used because white-box testing includes knowing the functionality inside of the system and providing input based on that functionality. There were very specific requirements for the project with regards to string data and values that could not be null. It was important to create tests to ensure these values were always valid when being manipulated by certain methods, and all expectations and errors were handled correctly. This allowed the program to run more efficiently and reduced the chance for crashes and hangups while running the program. Experience testing is testing based on experience with the system. After creating the initial Contact class and tests, it was easy to see that I would need to expand my testing reach in order to cover all areas of the program. I supplied tests like testAddInvalidAppointment in AppointmentServiceTest as well as tests in ContactServiceTest like addContactWithUIDandDeleteSucceed which would add a contact with a unique id and successfully delete it to ensure the add and delete methods were functioning properly. White-box testing has many practical used because it can help ensure all functionality within the program is being tested. This helps the developers to know the program is running correctly an can point out any areas in which errors or exceptions should be handled better. Experience testing has many uses because it simply is up to the skill of the tester to create goo experience-based tests. Whether it be stress test or validation, its important to cover all aspects of a programs logic and experience testing is sometimes a great way to come up with tests no one has thought of before. During this project I tried to adopt the mindset of defensive yet efficient programming. I employed caution by providing exceptions and error handling for values that had specific requirements. All of the classes had a variable that required it not be changeable and not be null, so this had to be handled correctly in order for the program to function correctly and encapsulate data that needed it. I tried to limit bias during my code review by pointing out improvements that could have been made or areas that could have been lacking in tests. I think a system can never be fully complete, it will only ever be on it’s release stage until no one is working on it anymore. This means that further rounds of revision are always necessary and improvement should be progressive. I think I could improve in the mindset area of bias and do better at taking advice from others and paying attention to important lessons that may not be readily apparent. With regards to my discipline during this project, I think it is possible to consistently maintain standards with enough practice, and its important master standards early on in order to ensure a solid foundation for development. Writing and testing code is important because it shows the quality and effort but in by the writer. Bad code will lead to bad systems, so its always important to uphold the highest standards when working with a team. I plan to avoid technical debt by keeping a solid grasp on whatever project I’m working on. Understanding the core of what your building makes it easier to write tests and improve it in the long run. Understanding that the final keyword prevented values from being augmented allowed me to make my program more secure and also more closely adhered to the requirements and saved me valuable time.