Dominic Drury

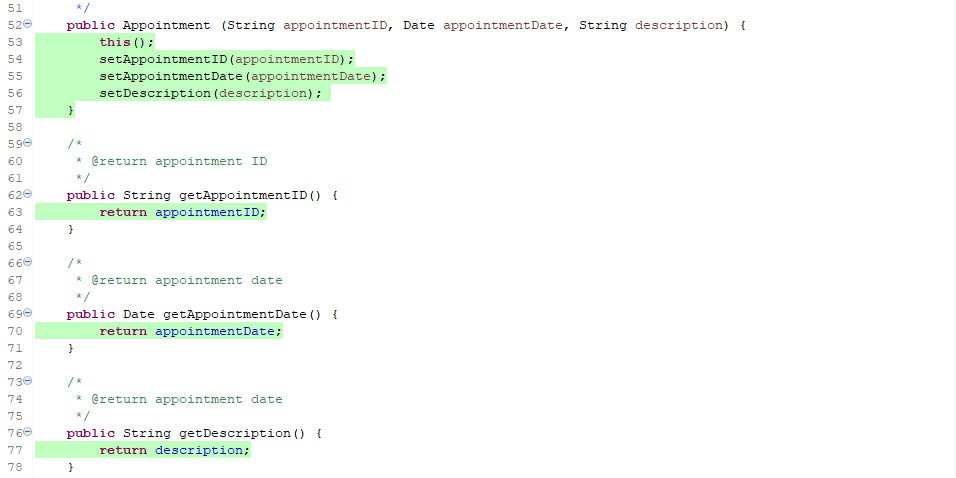
04/14/2023

CS320

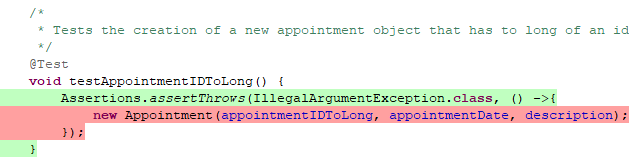
7-2 Project Two Submission

**Summary**:

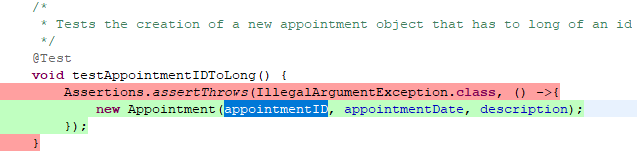
During my development of the unit testing for the three features I approached them in a linear fashion. What I mean by that is that I would follow the guideline and rubric, completing each set-in order from top to bottom. Luckily for me, I never really got stuck anywhere when testing, so this method worked really well for me, and since I was taking guidance solely from the software requirements, I was able to ensure I was aligned with them the entire time. Another advantage was since I wrote the program and the tests using the same list, as I was writing tests I was able to verify my coverage as teh file became more and more highlighted in green, showing it was being covered by the tests. I have included a snapshot below of the Appointment.java file as evidence.



I know that the overall quality of my JUnit tests are effective. The coverage for the files of task, task service, appointment, appointment service, contact, and contact service are all at 100% and I can verify that the checks to ensure that the only information added to the objects follow the requirements by altering the tests that make sure an error is thrown when information that fails the requirements is attempted. I include both a passing test and failing test below and highlighted the change.

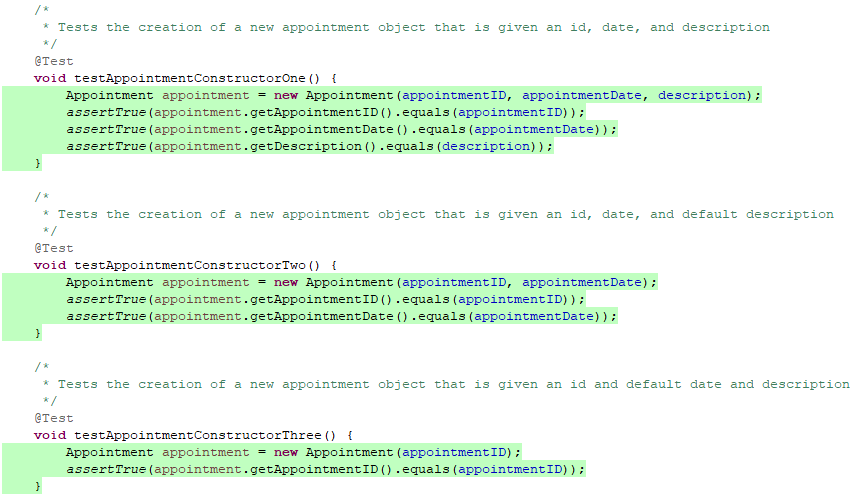




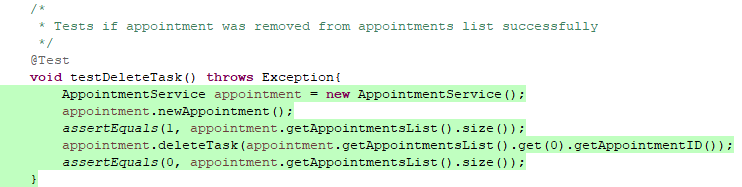


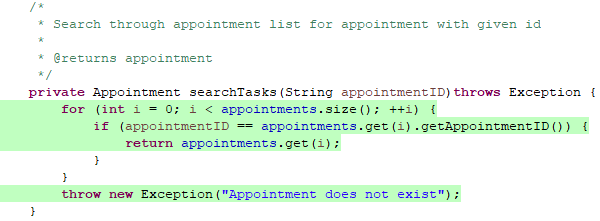


My general experience with writing these JUnit tests have been really positive. I enjoyed watching the progress fill as the coverage percentage increased with every additional test. I ensured that every test was technically sound by ensuring they were valid and reliable. An example of this is that rather than writing a single test that validated the construction of an object with every piece of information obtained by the user, I made a constructor that could make an object using any combination of information and tested every constructor to ensure that no matter the issue, there would be a constructor that could handle the situation. I have included a snapshot of the tests below as a citation of this.



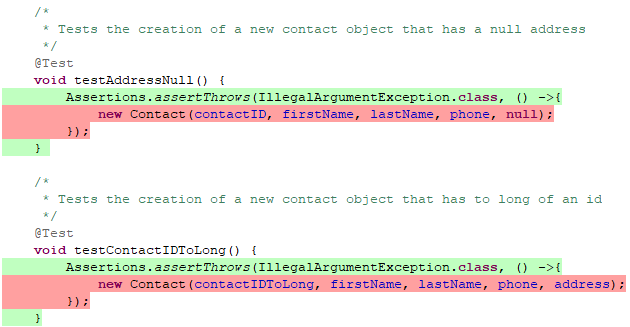
To ensure that my code was efficient I avoided redundant code. The best example of this is that while testing the deleteTask function, the searchTask function is used and therefore tested. With that knowledge there was no need to write a test for that function specifically. As citation for this I have included a snapshot of both the test that checks if an appointment was deleted as well as a screenshot of the search function being covered.





**Reflection**:

While working on this project I did notice a distinct difference in my mindset when developing and when testing. As I developed, I was trying to build something, and make something work. As I built and I ran my tests I felt as though I was actively trying to break the program, I had just built but being happy when I failed to do so. I employed a great deal of caution when writing both the program and the test. I ran the program often to make sure what I had written functioned, and I ran the tests and ran the coverage after each additional test. I wanted to make sure that the files I sent in either met or exceeded every requirement, going line by line through the guidelines and testing every limitation and expected functionality. An example of this is included below when I tested that the variables added met both the character max as well as being forbidden from being null.



I limited my bias by actively trying to break the code I had made using every test I could think of. I tried my best to get into the mentality that the only way to give the past me that wrote the program the respect I deserved was to do my best to test every line of code, every boarder variable, and every test methodology I had learned from the reading and practice this term. It seemed to help me not think of it as my program not working, but rather ensuring that it does when I sent it in as complete. If I had had thoughts of my program being broken if the tests managed to fail the program, I could absolutely understand the negative impact bias would have on testing your own code, and I understand why it is its own respected field. I was still sad when something I worked on for a week broke, I can’t imagine how it would feel if I had spent months working on it.

The importance of being disciplined in my commitment to quality as a software developer cannot be overstated. Slacking even a little can be detrimental. For me I got a reminder of that when I first sent in the project. I failed to verify the folder sent the files properly and had to rely on kindness to tell me files were missing. Without that my failure to be disciplined could have considerably negatively impacted my final grade. When writing and testing code, cutting corners can result in security failures, program failures, and could result in company's failing or people getting hurt or killed. I think the best way to avoid technical debt is to do two things, be diligent and hard working to keep to deadlines that have been set, and to understand that a quality product matters so much more than meeting a deadline. If you slack during a project, there is a good chance you will either have to cut corners or fail to meet a deadline resulting in a poor-quality product. For me the way I did show discipline is I had a constructor for any combination of information used to create an object, and I checked the folder I sent in three or four times to make sure the files were all there. I even used virtual studios instead of eclipse to make sure I was only opening the files I sent in and not the files I had saved locally.