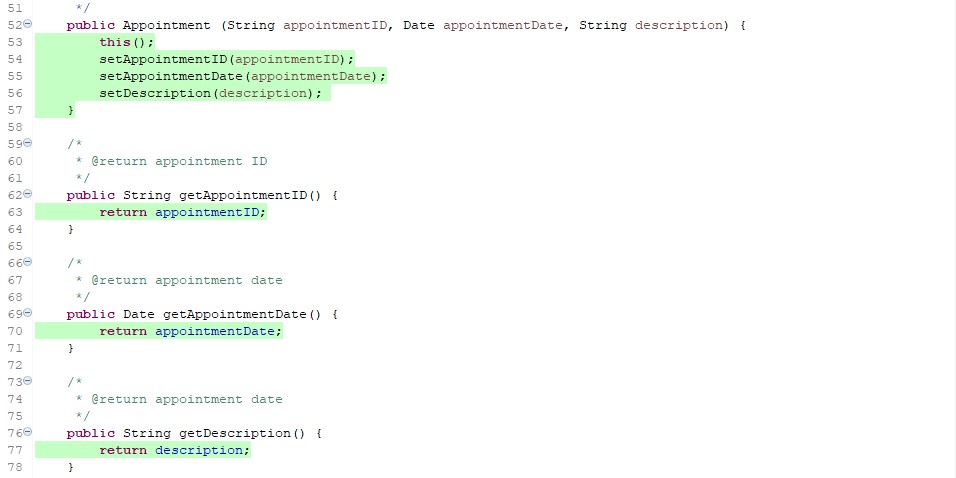
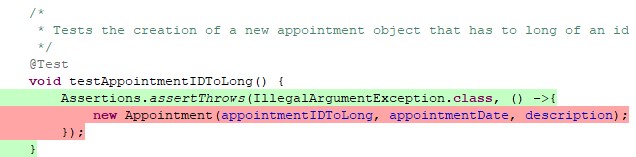
**Summary**:

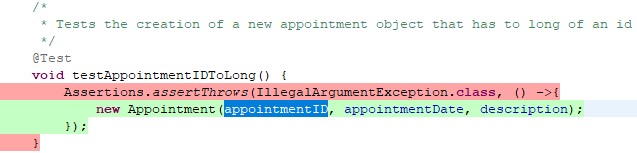
I have followed the guidelines and rubric, completing each set-in order from top to bottom. Most of the time when I get stuck, I have read all the instructions very carefully and solve the issues. Since I was taking guidance solely from the software requirements, I was able to ensure I was aligned with 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 the 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.



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.

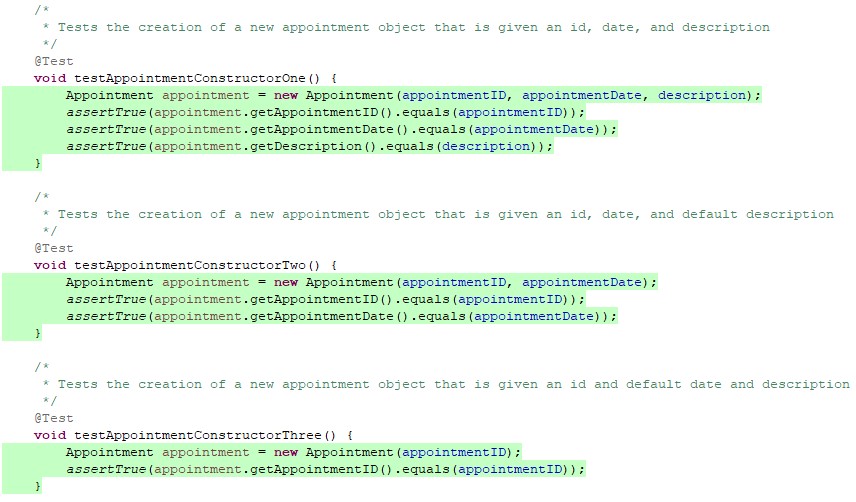




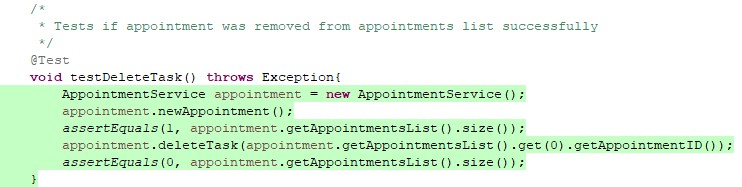


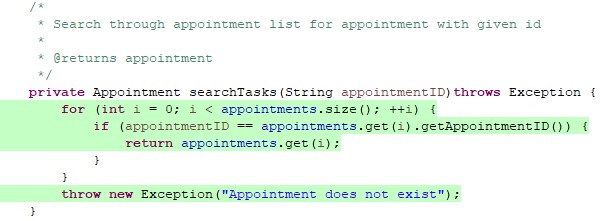


I ensured that every test was technically sound by ensuring they were valid and reliable. I have written a single test that validated the construction of an object with every piece of information obtained by the user, I made a constructor that could make an object using any combination of information 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.



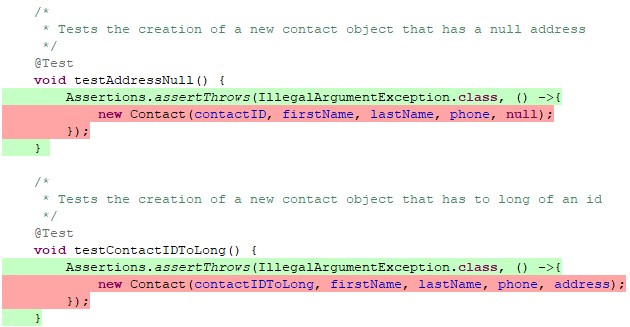
To ensure that my code was efficient I avoided redundant code. As citation for this I have included a snapshot of both the test that checks if an appointment was deleted as well as a screenshot of the search function being covered.





**Reflection**:

While working on this project I have noticed that there is a distinct difference when developing and 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.



In a software testing role, adopting a cautious mindset is crucial due to the inherent complexity of code and its interrelationships. Before running tests, it’s essential to grasp how different modules interact. For instance, if I’m testing a payment processing feature, I must consider how it integrates with user authentication and database transactions. Failure in one module can ripple through others, potentially leading to data loss or security issues.

Each feature has its own risk profile. While testing a new user registration feature, I approached it with caution, understanding that bugs could affect user data integrity. I performed thorough testing, including edge cases, to mitigate risks that could compromise user experience or security.

I employed a cautious approach by implementing incremental testing. After making changes to the codebase, I tested small sections before executing a full regression test. This way, I could isolate and address issues more effectively without impacting the entire system.

Understanding the interconnections within the codebase helped in predicting how changes might affect other components. For example, modifying a function in a library that handles data validation could inadvertently affect all features relying on that library, leading to unexpected behavior.

Engaging with developers and other testers enriched my perspective on the code's complexity. By discussing potential impacts of changes, I could tailor my testing approach to focus on high-risk areas, like a new algorithm for sorting data that could impact performance across the application.

When testing an API that aggregates data from multiple sources, I realized that changes in one source could lead to failures in data integrity or format. By understanding these interdependencies, I created comprehensive integration tests that simulated real-world scenarios, ensuring robustness against changes in individual components.

In summary, adopting a cautious mindset as a software tester is vital for navigating the complexities of code and ensuring a high-quality product. By being aware of interrelationships and dependencies, I was able to develop more effective testing strategies, identify potential issues early, and ultimately contribute to a more reliable software application.

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 cut corners or fail to meet a deadline resulting in a poor-quality product. For me the way I did show discipline is because I had a constructor for any combination of information used to create an object, and I checked the folder I sent 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.