Summary

My approach was entirely directed by the software requirements. IT was asked that the contact, task, and appointment packages were able to effectively create, edit, and delete instances of themselves so I created object, service, and testing classes to ensure that they all acted as expected.

I know that my testing was reasonable and effective as I included cases that would test that things went smoothly if they were provided the correct information and that the program would throw an error if it was asked to go against the service requirements. All the tests passed indicating that the program worked as intended.

I made sure my code was technically sound by focusing on clarity, being as modular as possible, and following best practices. In TaskTest.java testTaskValidation I was testing that the code would catch if the user tried to enter a task with unacceptable information. They are broken up and labeled by the variable being checked and what it is checking for. This makes it easier to add and remove qualifications as well as reference it when possibly building another object class.

I made sure my code was efficient by grouping tests that required the same setup. For instance, testUpdateAppointment in AppointmentServiceTest.java makes sure that multiple different variables can be updated at the same time.

Reflection

Testing Techniques

I used mostly positive, negative, and boundary testing when building since the requirements were relatively simple. There was a short list of things that each object needed to be able to do (create, edit, delete, etc) and qualities they couldn’t have (value not null, too short, or too long), so I just listed them out and checked each one individually and then modified the list for the new requirements when making the next object. This might not be the most efficient for larger projects but was worth the effort for something of this size.

I didn’t end up needing to any static testing as running the test code seemed to reveal any errors I was having. I could have ended up printing it out and writing notes and such, but it ended up not being necessary.

Mindset

I wasn’t necessarily cautious as I completed the task as I had the knowledge that there was no actual personal information being shared. I did keep in mind that the objects had similar requirements, so I made sure to keep it modular. This was done in the hopes it would make it easier to adapt pre-existing code to new situations/requirements and ended up being a productive effort.

Using the junit test cases helped me to limit the bias, but knowing what could go wrong with my code and what the character limits were helped. If I didn’t already have that framework, it would be more likely that there would be unintended consequences. There aren’t many systems that could conflict with each other this early in development, but as the program gets more and more complex, I imagine that more and more constraints will need to be put on things to make them work together. The difficulty comes when the developer only checks for mistakes they would make and it’s much easier to see your own bias when other people run into it.

I find it important not to cut corners when writing code because they multiply over time. If you are messy in one section and then use that code as a reference to do something else, you are referencing bad information. It would be like trying to level a table with an uneven plane, you would spend more effort fixing what you did wrong than doing anything new. I try to avoid creating mistakes in my code, by breaking it into easy-to-understand chunks and adding comments that say why I wrote something and what it is supposed to do.