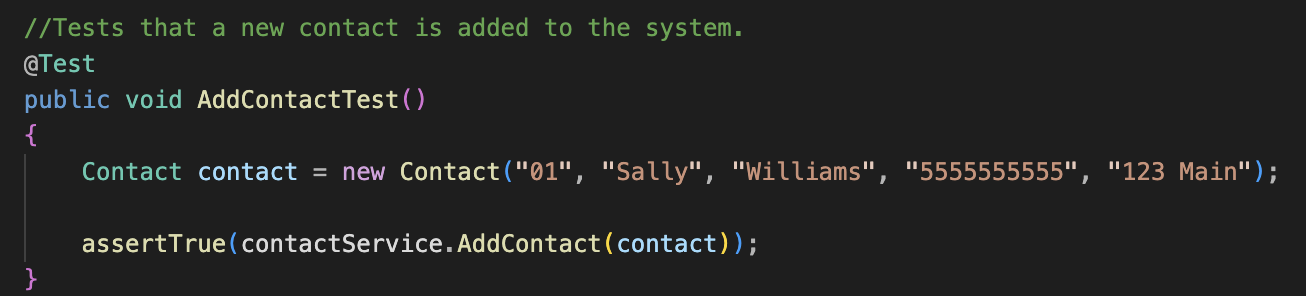
**Summary**

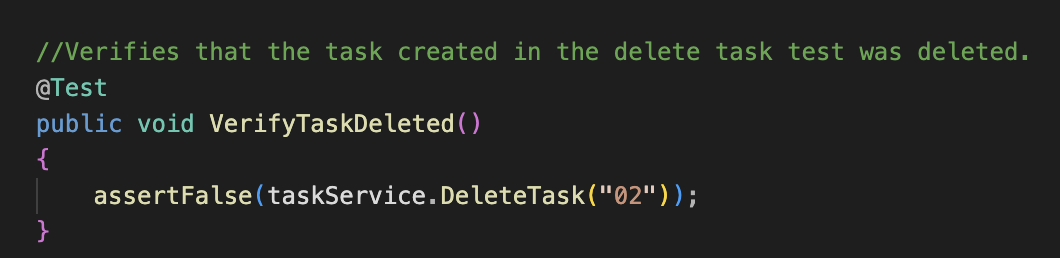
When it came to creating the tests, I chose to focus on breaking everything down to individual methods. This was done through the use of more tests, but allowed me to verify the different requirements. I was able to build my tests so that they grew from easiest to most complicated. A prime example of this can be seen from lines 22-28 in my ContactServiceTest class. That is the AddContactTest. The AddContact method needs to be first because we are using the add contact portion throughout the other tests. Here is a screenshot of that code:

(Screenshot of AddContactTest, 2024)

These unit tests went from this simple AddContactTest to the next screenshot below:(Screenshot of the UpdateLastNameTest code, 2024)

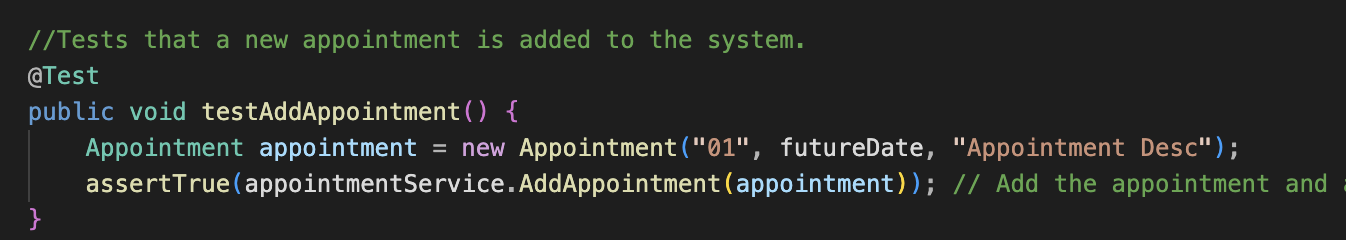
Since this came from the same ContactServiceTest, it is easier to see that this was a more complicated method because I had to change an element of the Contact. That was important because, if the add contact didn’t work, then it wouldn’t matter what I did otherwise. That approach allowed me to verify that every element worked.

However, I also had to think about the quality of these tests. I would say, among all of my services: Contact, Task, and Appointment, I covered everything that I could easily see. What I mean is that I made sure to test every element of my code. I didn’t look at one component and say that was okay, I made sure that each area worked. To do this, I actually built my code in smaller components. I also verified that these components worked through assertions or actual methods. One of these methods can be seen in the below example from the TaskServiceTest class:

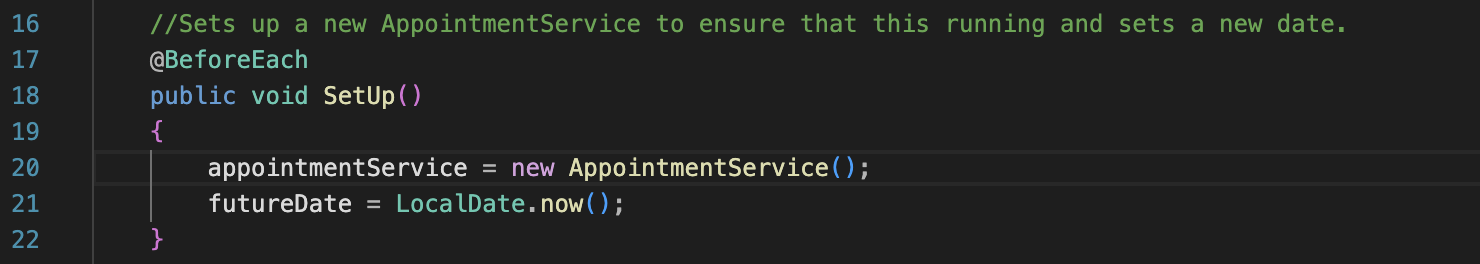
(Screenshot of the VerifyTaskDeleted test, 2024)

This allowed me to reach, at minimum, an 80% test coverage because I verified certain features, but also made sure to test everything.

In terms of writing JUnit tests, I can say that this was the first time I have done so. JUnit testing was new to me. To that end, I had to ensure that my code was sound. I had to create these tests that focused on portions of the main classes to work. An example can be found in my AppointmentServiceTest. On line 29, I use the code: “assertTrue(appointmentService.AddAppointment(appointment))” (Durr, 2024). This line of code ensured that, when I created a new appointment with the parameters of ID, date, and description, it did create that appointment. For the full test, here is a screenshot of that code:

(Screenshot of testAddAppointment, 2024)

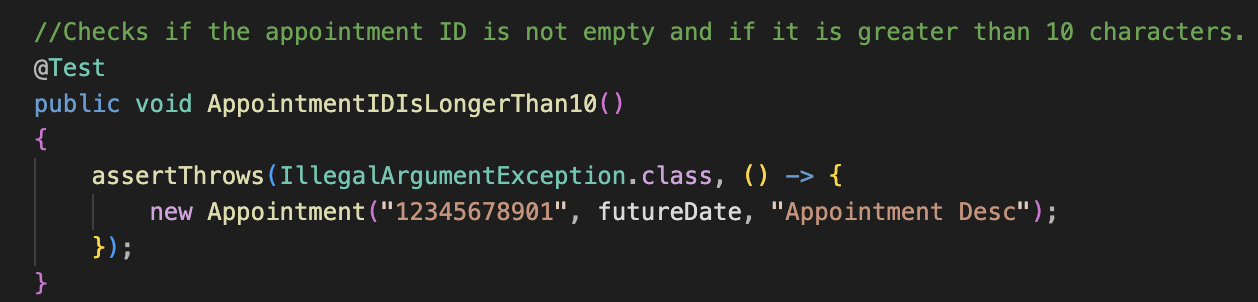
From there, I used a feature to ensure efficiency by creating a BeforeEach method that set up the common components. This is seen in the AppointmentServiceTest code. Here is a screenshot of that code:

 (Screenshot of SetUp, 2024)

This allowed me to be more efficient, because I wasn’t typing the same cover over and over.

**Reflection**

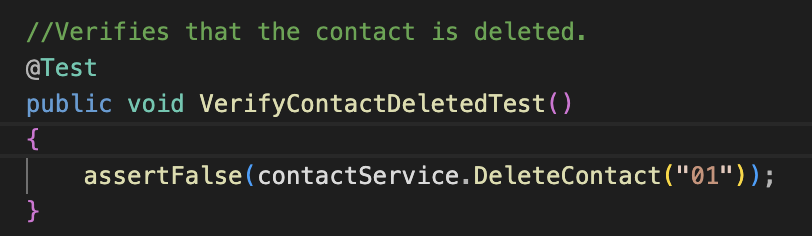
When it came to testing techniques, I employed two main techniques, unit testing and integration testing. The Unit tests can be seen in the above summary. Each method was designed to test one unit of the code. The add contact, task, or appointment was being tested with a method that called them. This is how unit tests work because “[u]nit testing is the process where you test the smallest functional unit of code” (*What Is Unit Testing? - Unit Testing Explained - AWS*, n.d.). I was testing a smaller unit that only did one thing. This is more apparent in the tests for contact, task, and appointment. Here is a screenshot for an example from the AppointmentTest class:

(Screenshot of AppointmentIDIsLongerThan10 Method, 2024)

This tested one requirement and threw an exception if the ID was over 10 characters. This also tied into the integration testing that took place. Integration testing is “a type of [software testing](https://katalon.com/resources-center/blog/software-testing) where components of the software are gradually integrated and then tested as a unified group” (Katalon, 2023). These services tested each other. I didn’t just call to one portion of my code, I called to other places. Here is a screenshot from the TaskServiceTest class that shows:

 (Screenshot of UpdateTaskNameTest, 2024)

I would add that I did a minor amount of functional testing. This is because functional testing “seeks to establish whether each application feature works as per the software requirements” (*What Is Functional Testing? Types & Examples | OpenText*, n.d.). I had to ensure that a contact, or task, or appointment was added. I had to verify that the desired results. An example of this verification is in this coding image:

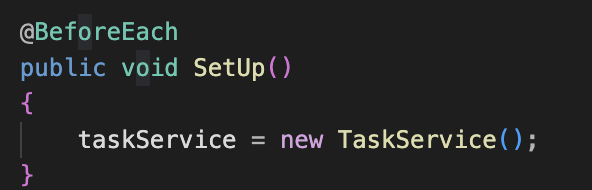
 (Screenshot of ContactServiceTest code, 2024)

The implications of all of these tests were that I was able to verify everything worked in small units, together, and as expected.

There were, of course, tests that I didn’t employ. I did not do any non-functional testing. Since the assignment was simple, I focused on coding and then testing. Some requirements of non-functional testing can be security and performance. I also didn’t do any security testing. Since this is meant to find vulnerabilities in the code, I could have done this, but I didn’t. That was not considered. These are the two main techniques that I could have employed, but did not.

The implication of these unused techniques is that I didn’t see what could have been done. I didn’t ensure that any database was secured against attacks. Since these are creating contacts, tasks, and appointments, security can be important. The use of security testing is to ensure that people from the outside can’t get in. The implication of not doing this means potential vulnerabilities within the code that could be exploited. This is also true in non-functional testing. Since a part of that can be performance, the practical usage is to make sure that the program performs. All I did was test that things work, that doesn’t mean it performs well.

In terms of mindset, I had to take on the view of, what needs to be looked at. I had to analyze my code and look for potential points of failure. I had to see what I might use consistently. An example of that was in my TaskServiceTest class where I did a single SetUp function that would create a new TaskService. Here is a screenshot of that:

(Screenshot of the TaskServiceTest SetUp, 2024)

I had to do this because the complexity of what I did could result in inefficient code. I had to ensure that everything integrated properly. The complexity of the working relationships of all four classes was something I learned to appreciate. An earlier example would be in creating a New Contact before I could even add a contact. This was found on line 25 in the AddContactTest code. I would never be able to make it work otherwise.

I am the programmer of this, I can look at my code and write unit tests that do what I want them to. As seen in earlier examples, I did what I could to ensure a lack of bias by verifying that a contact was deleted, that I used assertions, and that I looked over each line of code. The verification code was a way to try and avoid bias. That can be seen earlier in the VerifyDeletedContact screenshot or on lines 41-45 of the ContactServiceTest code.

None of this means anything if I wasn’t disciplined. As illustrated in earlier portions, I did what I could to cover as much as possible. I created methods that would set things up. I looked at each line of a test and made sure that it was necessary. Could I have done a better job, yes. I look back and see that I create the same Contact, Task, or Appointment multiple times. I could have added those to my SetUp methods. That was a lack of discipline in analyzing on my part, but I’m still learning. I need to ensure that I don’t cut corners. I have to analyze things. An example is as simple as making sure it’s possible to even get the test. That can be seen in this screenshot from the AppointmentServiceTest class:

(Screenshot of GetAppointmentTest, 2024)

However, in the end, I do feel that I did a good job at all of this.

**References/Citations**

Katalon. (2023, January 31). What is integration testing and its types? *katalon.com*. https://katalon.com/resources-center/blog/integration-testing

Software Testing Best Practices Checklist. (2023, October 25). *PractiTest*. https://www.practitest.com/resource-center/blog/software-testing-best-practices-checklist

Atlassian. (n.d.). *The different types of testing in software | Atlassian*. https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing

*What is Functional Testing? Types & Examples | OpenText*. (n.d.). OpenText. https://www.opentext.com/what-is/functional-testing

GeeksforGeeks. (2023, December 6). *Software testing techniques*. GeeksforGeeks. <https://www.geeksforgeeks.org/software-testing-techniques/>

*What is Unit Testing? - Unit Testing Explained - AWS*. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/what-is/unit-testing/#:~:text=Unit%20testing%20is%20the%20process,test%20for%20each%20code%20unit.

Durr, S. (2024, June 16). *AppointmentServiceTest* [Software].

Durr, S. (2024, June 20). Screenshot of the UpdateLastNameTest code [Screenshot].

Durr, S. (2024, June 20). Screenshot of ContactServiceTest code [Screenshot].

Durr, S. (2024, June 20). Screenshot of testAddAppointment [Screenshot].

Durr, S. (2024, June 20). Screenshot of SetUp [Screenshot].

Durr, S. (2024, June 20). Screenshot of AddContactTest [Screenshot].

Durr, S. (2024, June 20). Screenshot of the VerifyTaskDeleted test [Screenshot].

Durr, S. (2024, June 20). Screenshot of AppointmentIDIsLongerThan10 Method [Screenshot].

Durr, S. (2024, June 20). Screenshot of UpdateTaskNameTest [Screenshot].

Durr, S. (2024, June 20). Screenshot of the TaskServiceTest SetUp [Screenshot]

Durr, S. (2024, June 20). Screenshot of GetAppointmentTest [Screenshot].