Justin Angle

21 June 2024

CS 320

Project Two

There were three features I had to create and make tests for to ensure they met the requirements the customer had. Starting with the contact class, I had to meet five specific requirements. Many of these requirements dealt with how many characters were allowed in certain fields and the requirements these fields had. All of the fields on this contact class also could not be null. To verify this code met the requirements I tested if the correct exception would throw dependent on the input. For example, the Contact ID could not be longer than 10 characters and it could not be null. I utilized the three below tests to ensure the contact ID met the requirements presented. I utilized similar tests for the rest of the fields. This ensured all the fields would meet the requirements presented.

A computer screen shot of text

Description automatically generated

The next feature that was requested is the task class. This feature had three requirements for the task class. These requirements were similar to the contact requirements. They all dealt with the number of characters a field could have and that none of them could be null. With this feature I tested similar with the contact feature. This is because they had very similar requirements. One of these requirements that I will be showing is that the name field could not be longer than 20 characters and could not be null. The following Junit code is what I used to ensure this field met the requirements the customer requested.

A screen shot of a computer code

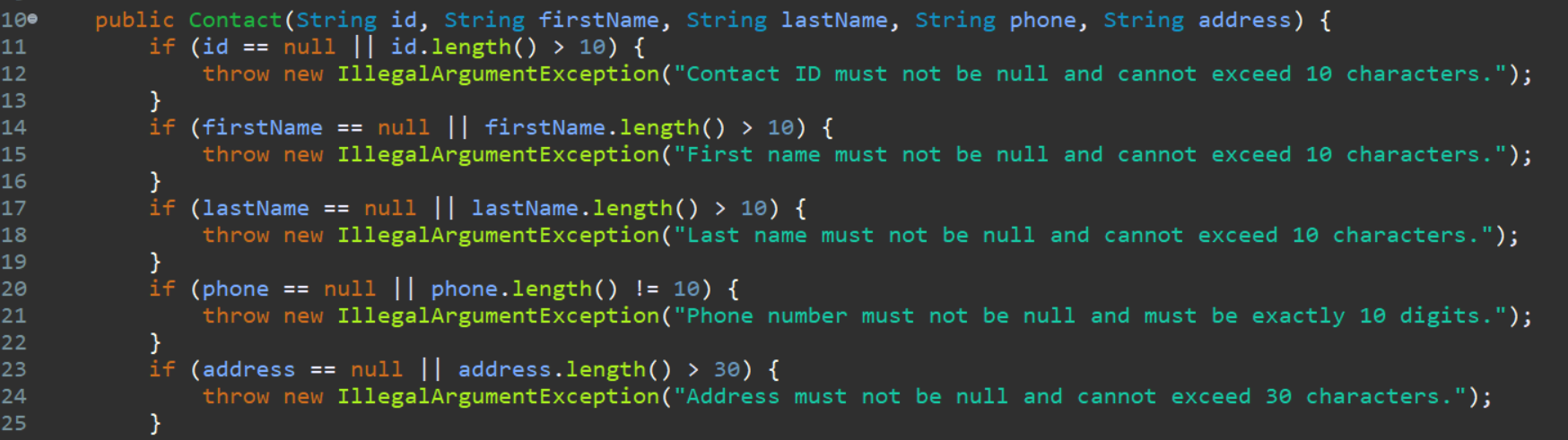
Description automatically generated

Moving onto the last feature that was requested by the customer is the appointment feature. This class had three requirements that had to met. I had the most problem with this feature as it dealt with dates and ensuring that the appointment date was not in the past. With the testing on this I used a different approach. Using the date field, rather than checking if an exception was thrown, I decided to check that the date met the requirements presented by the customer.



With these tests for all the features I was able to meet my minimum required percentage of coverage that I thought would be adequate. This was an 80% of coverage per feature. With the contact class, I met that with an 83% of coverage. For the task class, I was able to meet it with a 100% of coverage. And lastly, for the appointment class, I met the requirement with an 82.1% of coverage. Knowing that I ensured that I was covering the important aspects of the requirements and ensured the features were all tested to at least 80%, I was more confident that my coding was correct.

This was my first time coding in Java and utilizing Junit tests. As such, it was very difficult for me to learn how to code and how the Junit tests work with the Java code. However, I conducted research on how they all work together and learned the features that Junit provides. I ensured my code was technically sound by testing the entire code, ensuring that I met every requirement that was set forth. For example, with the contact class, I ensured that when entering information that information had to meet the requirements. Otherwise, and illegal argument exception would have been thrown and it would have told the user what the requirements were that needed to met. Below is that code.



I ensured my code was efficient by ensuring I covered everything I had to in order to meet the requirements. In the appointment test I created a setup for the rest of the tests. I initialized certain variables that I knew I was going to use multiple times. This way I was able to just input the variables in the tests rather than separately writing them each time I wanted to use it in a test. Below is an example of that code.

A screen shot of text

Description automatically generated

The software testing technique I employed throughout this project was unit testing. With unit testing, I tested each individual requirement to ensure the code worked as intended with certain inputs. This allowed me to test the functional unit of code to ensure the fields would have the intended results when inputted. Using these tests allowed me to isolate every input/requirement to ensure they are met no matter the input. Ensuring that if the user inputs something that is not accepted they can know why by telling them what was wrong.

A software testing technique that I did not use would be integration testing. Integration testing verifies the interaction between different code files of a software application. With integration testing you are ensuring that when these components of the program are combined there will be no problems or bugs that could arise from the interaction. This sort of testing helps identify any problems that might occur from those kinds of integrations.

The mindset I had when working this project was one of newness. As I mentioned earlier I had no previous experience with Java or Junit coding. So at first I was very overwhelmed, but after researching more and practicing I was able to understand the coding more. I employed caution by ensuring I did not write too much of complex code that might be hard to understand. I ensured that I kept the complexity down and use basic functions. It’s important to understand how the code would connect to each other and work together, to use the different java files you have to remember to import them. Forgetting that your code might rely on another java file might lead to errors in the code.

Eliminating bias when you are also responsible for testing your own code is very difficult. The way I attempted to eliminate bias was to pretend that I did not write the code and that I was testing it as an actual software tester. This helped me test it with less bias, which in turn helped in bettering my code as I noticed some things that could be better written. Often with bias, we think that our code is perfect and it’s very difficult to admit that the code may have mistakes or need improvement.

Being a disciplined software professional is important as the customers and the business you are working for are depending on you to produce quality code. We cannot cut corners when testing code as we need to ensure we check multiple avenues that a user might do in the code and ensure it still works. Cutting corners in testing usually means that there will be more bugs and errors once the final product is released. I’ve found that users have a way of finding bugs and errors that developers don’t even think about. To avoid technical debt for myself or the company that I start working for, I will ensure that it’s not only me looking through the code and testing it but I will ensure that multiple other testers and developers look over it and test it before even releasing it. I think another good practice is having alpha and beta tests. This allows users to interact with the software, provide feedback, and identify bugs that the developers and tester might not have seen.