Project Two

Dhiraj Gurung

Southern New Hampshire University

CS320

Toni Farley

February 20th, 2025

**Project Two**

**Summary**

When creating the code I made sure that each of the software requirements was met. For example, the 5 requirements in the Contact Class dictating the length and size of each object were followed thoroughly. The contact ID was made private to ensure that it could not be updated, and there were if statements in place to ensure that there was a value and that the length was not longer than 10 characters. As shown below there were three tests created one to ensure that the parameters were being passed and accepted, another to ensure that the setters were working correctly, and a third one to ensure that the program could handle the null, blank, and values of incorrect length (Figures 1-3). Additionally, I can say that my tests were highly effective as they covered a high percentage of the lines of written code. The ContactTest test set seems to have been less effective than the other tests as its coverage percentage is lower.

I made sure that my code was technically sound by using different edge cases to test for each of the given parameters as shown in Figure 5 with the Contact ID parameter. If the code could handle each of these scenarios then we can be certain that at least in the given scenarios the code was working properly. One way that I made sure that the code was efficient was by not repeating lines of code whenever it was possible. For example, as shown in Figure 6 I only wrote the code to test each parameter value once and used a CSV file to write all the edge cases. Instead of doing this, I could’ve written a different test for each parameter value and its own edge cases which would’ve resulted in more lines of code.

**Reflection**

While working on this project I ensured that my code was functioning properly by running tests on them. I used was using JUnit tests to test individual components of the software. For example, for the Appointment Class, I tested the setters separately to ensure they were functioning properly. I also created separate tests for each requirement like testing to see if the program could handle null values and past appointment dates. The practical use and implication of testing software at the individual level is to detect errors and pinpoint the source of those errors more easily when you test blocks of code rather than the whole program. In conjunction with unit testing, I also applied acceptance testing principles. I made sure that each of the requirements was implemented and working correctly. I tested null values for the invalid appointment ID, date, and description. I also tested to see if the program could handle errors when too many characters were given for the appointment ID and description by using realistic values. The practicality of acceptance testing is to emulate a real-world scenario and not just test for edge cases. By using values a user might enter you can check to see if the program would work in a real-world environment.

One software testing technique I did not use is security testing. Although I may have used private methods for the IDs to ensure that the IDs couldn’t be changed by the user, the code was not made with the intention of creating a secure program. Testing and coding for security would help to keep the databases of these IDs secure from harm in case someone with malicious intent wanted to access the appointment data. Additionally, there was no usability testing done to ensure a user-friendly program. Doing so would make it easier for consumers to create accounts and use the software in their day-to-day lives. If this was a program created for the masses by a private company then this may improve the customer’s impression of the company. However, the focus of this program and code was functionality. As a result, when I coded the program I only ensured that the program was functional and met the given requirements.

When working on this project I adopted the mindset of an agile principle I learned: “Just barely good enough”. This does not mean that I was settling for mediocrity but rather that I was focusing my time and efforts on creating a product that worked. My primary goal was to create a functional product, not a fancy one. In order to ensure that the code was functioning properly I had to be cautious, but this didn’t mean I couldn’t try new things. Instead, this meant that I was testing and checking my code for errors more frequently whilst trying new concepts. When I was using the CSV files to test for different parameter edge cases as shown in Figure 7, it was a new concept for me. As such I tested to see if the code was working with each line of the the first ID parameter before writing the code for the rest of the parameters. It was important to not only test the individual blocks of code but also the code as a whole at the end to ensure that each block of code works well with each other to ensure a functioning program.

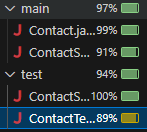
One way I tried to limit bias was by trying to look at the code objectively and testing each parameter value for all the same edge cases I could think of. This was done for the Contact class as shown in Figure 7, and it was repeated for the Task class as well (Figure 8).

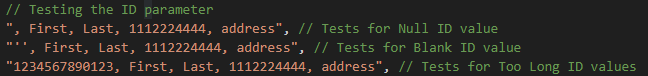
When writing and testing code it is important to be disciplined and take the work you’re doing seriously because mistakes in code could have significantly detrimental effects on the consumers. Coding defects or bugs have been responsible for data loss and vulnerabilities to malicious software which could lead to massive financial loss. One way I plan to avoid technical debt when it comes to coding is to try my best to write efficient code. For example, in this project, I tried to do this by using the CSV file to write all the test cases for the parameters and then only using one test block to test all of those cases. Additionally, I tried to reduce code length whenever possible like in Figure 9 where I wrote the getter for the phone number in one line of code.

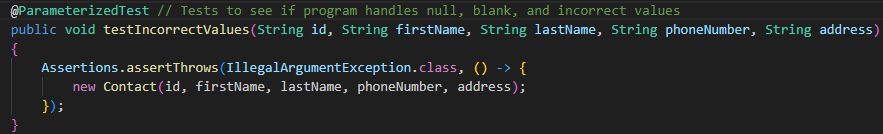
(**Figure 1**

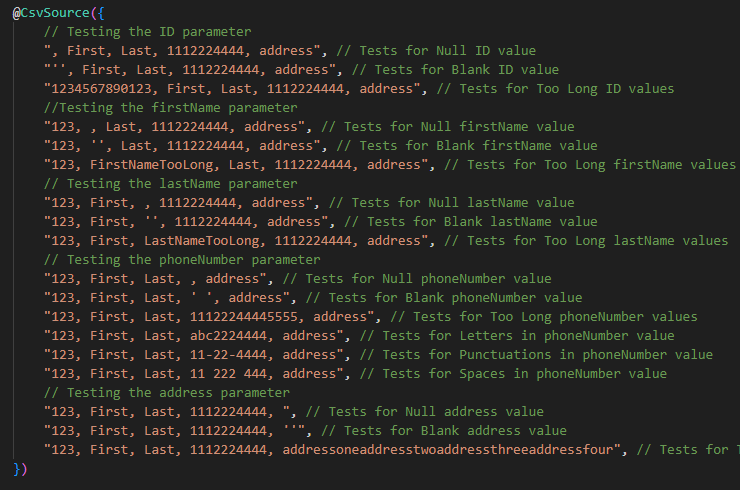
****(**Figure 2**)

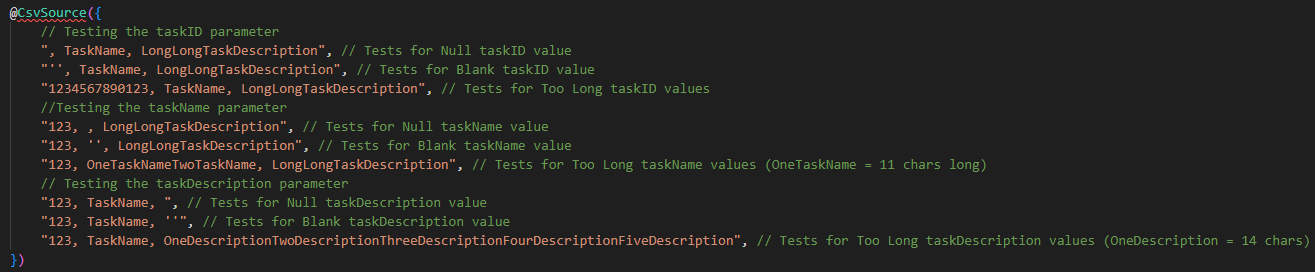
****(**Figure 3**)

(**Figure 4**)

(**Figure 5**)

(**Figure 6**)

(**Figure 7**)

(**Figure 8**)

(**Figure 9**)