Journal 4-2: Unit Testing Approach and Writing JUnit Tests

Dhiraj Gurung

Southern New Hampshire University

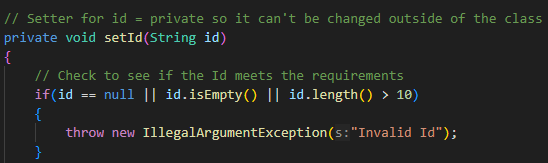
CS320

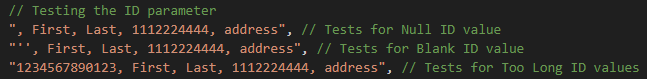
Toni Farley

January 28th, 2025

**Unit Testing Approach and Writing JUnit Tests**

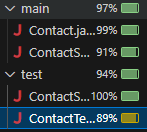
In my testing approach, I attempted to align the tests to be in line with the requirements as much as possible. For example, in the Contact class, I tried to make sure that each of the methods that I coded was trying to follow the requirements. Specifically, for the method to set the ID I created the method to be private to ensure that it couldn’t be updated. I also used an if statement to check if the ID field was null, longer than ten characters, or blank (Figure 1). When I was creating the tests I made sure to use different test values to test for each of the given requirements (Figure 2). Additionally, I tested to make sure the program would run with both the correct and incorrect values as well (Figure 3 and 2 respectively).

(**Figure 1**)

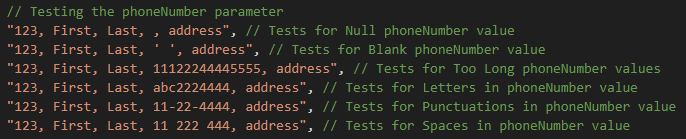
(**Figure 2**)

(**Figure 3**)

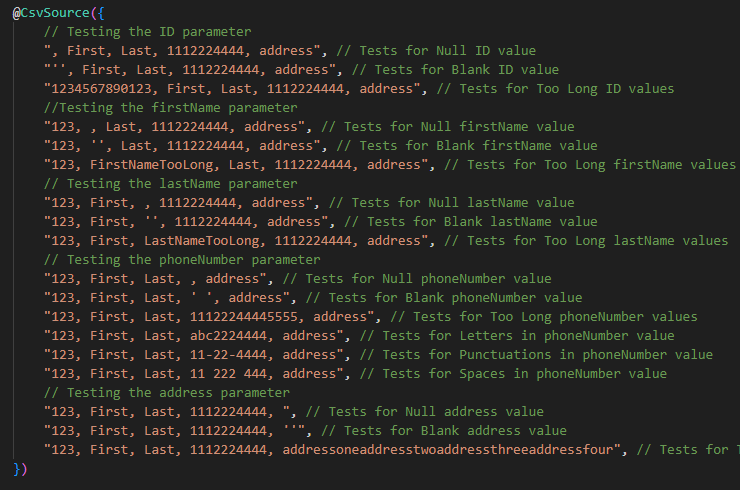
In terms of the quality of the JUnit tests for each of the classes, I tried my best to achieve 100% test coverage (Figure 4). Although the test coverage doesn’t tell us if our tests are correctly designed or not it does tell us if the tests cover all parts of the code. I know that my tests were mostly effective because they covered a high percentage of the lines of code that I had written. However, from the image below the ContactTest unit tests were seemingly less effective than the ContactService unit tests.

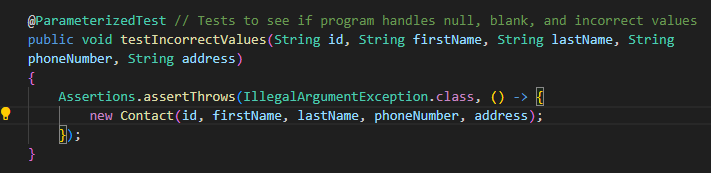
(**Figure 4**)

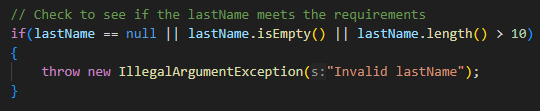
I made sure that my code was technically sound by setting up unit tests as I was done coding a chunk of code. Doing this helped me make sure that what I was coding was being tested for errors before introducing the code for the requirements I hadn’t yet coded. I was also compiling my code frequently to check for any errors and fixed them as needed. Additionally, as shown in Figure 5 I made sure to test for as many edge case scenarios as I could to ensure that the code was working properly.

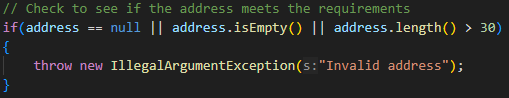
(**Figure 5**)

One way that I ensured that my code was efficient was by using a CSVSource file to write all potential test cases in one file and only having to use one assertion to test all of those cases instead of using assertions and repeating that same assertion for each and every line (Figures 6 &7). Another way I tried to be more efficient with my code was to try not to use and repeat the same exact lines of code in other parts of the program as well. However, I also reused similar logic statements when applicable instead of creating a different one from scratch. For example, the if statement logic for all the setters was reused because the requirements for the private member variables were all very similar (Figures 8 & 9).

(**Figure 6**)

(**Figure 7**)

(**Figure 8**)

(**Figure 9**)