Jaden Williams

SNHU - CS320

12/10/24

Project 2

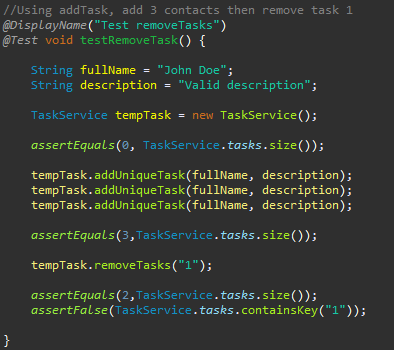
The task states that the full name is required, cannot be null, and cannot be larger than 20. Using the unit tests, four to be specific, I aimed to test the specifications themselves and where their boundary is.   
Here are screenshots testing setLastName:



I know my tests were effective because several of the videos that I watched stated that testing for everything is overly redundant and a waste of time in most cases. (Note: I think redundancy is great for testing purposes, the video simply stated it was a waste of time in an instance such as this) By ensuring I was able to test the specifications themselves, I did not need to run through every invalid/valid input. Instead, it was instructed that I aim for one valid input, one invalid input, and full testing of the get and set functions.

In testRemoveTask, I checked the functionality to remove task 1, and then checked it is removed. By testing for both if the task was deleted, and that the task pool size is correct, I can ensure that the correct task is removed always and thus prove the technical soundness of my program. If I had not tested for then there is a possibility of error.

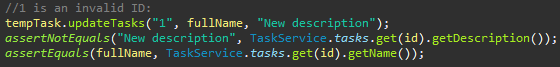
Here is a screenshot of the test:



Ensuring the deleted Task is actually deleted is an important part of code efficiency. I undertook testing to confirm that objects were created only in response to specific requests as shown below:



Making sure the original string is still intact was important as well:



In all three milestones I used Unit and Static testing. Static testing is inspecting code against the specifications to identify any flaws that may be within it. This came in handy a lot of times, especially when a JUnit test failed, and I had to identify why. There were a few times where I would have the wrong attribute assigned in the logic branch which threw an error and would cause the test to fail, this is where static testing really helped me excel.

False positives do exist, and in module 4 I became very aware of them, so I began having to really read into each exception to make sure I am testing for the correct errors. A bad input is a bad input, but if in the wrong spot or otherwise not assigned correctly I found that I can get the right exception for the wrong reason.

In milestone’s 3 and 4 I did not use any integration testing, and I used next to no system testing, whereas in milestone 5 I used some system tests. With each milestone, we created the base class as well as the service class. The service class is used to essentially control the base class so I could have tested the base class through the service class in a system arrangement, which is something I employed in milestone 5. With integration testing, if I had used it, I would test the program as a whole, testing the base classes through the service classes at the integration layer to see how they perform.

I also performed no automated testing and no security testing. Automation is nice, but I opted for manually run tests. Security on the other hand is not just a nicety. Security tests should be run on all components, libraries, etc. for any potential risks of threats. In the instance of milestone 3, 4, and 5 there weren’t any crazy usages of libraries, databases, etc. which put security tests on the back burner for me.

Automated testing is a very valuable tool, especially for large scale projects. CI tools are really what come to mind with automation. It’s useful for the entire life of development. When it comes to a project that has any data involved, security tests are also massively important, and very valuable. With these milestones, there is no risk involved so I found no need, but security tests would be used to test for protecting from buffer overflows, SQL injection, and other various potential threats and threat actors.

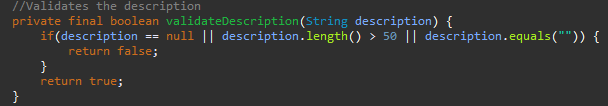
Unit testing is incredibly useful, it can be deployed to find how the program reacts to missing or null inputs, inputs that are way too long, and everything in between that is not supposed to happen. Used with static and system testing as the code grows, I can ensure the quality of the project that I am delivering.

My mindset throughout Project 1 changed constantly. Being both the programmer and the tester I had to ensure I was writing well thought out code, that fit the requirements and expected the unknown. As the tester I took the code and put it through a series of inputs to see how it would react, testing for any flaws that I may have created by accident.

In the requirements field of the Appointment Service in module 5, it reads:



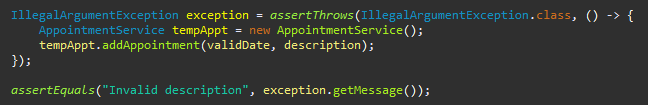
This seemed easy enough when I first read it. The string cannot be over 50 characters, or null. Easy. I will use a logic test to test for both. Then I thought about how an empty string would affect the test, it is not over 50 characters, and it is not null, so it would technically pass. A description is a requirement so therefore it should not be empty as well which caused for the test to look like this:



Through my research when I struggled hard with the module 4 assignment, I found a great tutorial explaining why a hash map rather than a list is significantly more efficient. It used the Hash Map method “contains” to directly find the ID rather than have a bunch of extra code that searches for an index. By using a hash map, it also simplified my JUnit tests as a whole. My code was as follows:



Bias is very important to remove when writing the code and testing it too. In my few years of experience programming, I have become all too familiar with my code appearing to be correct, but once I finally turn it in, a different set of eyes finds a lot of mistakes. I attempted to put myself in those “different set of eyes” and essentially grade my code and find the little mistakes. By employing the JUnit tests to test the correctness of the exception thrown, rather than just test for an exception to be thrown, I was able to find all the little mistakes. Here is an example:



This was used at the end of the test for an empty description to ensure I was getting the correct exception thrown.

Cutting corners may seem like the appropriate thing to do to reach a deadline, but it will cost you more time, effort and potentially money (If in the professional world). My first milestone assignment was late, I struggled really hard with it. I wanted to cut corners and get it done on time, but I knew that would not set me up for quality code and quality testing and in the end, I would not be happy with my product. Instead, I chose to not cut corners, take the late penalties and in turn created a program I am confident in. An important thing I have learned from this class is that technical debt comes in all forms. It can be a bug, or a missed use case, but regardless it is always important to continuously test for them, gather feedback on the program, and constantly improve upon the code.