Jeffrey McGinty

[Jeffrey.mcginty@snhu.edu](mailto:Jeffrey.mcginty@snhu.edu)

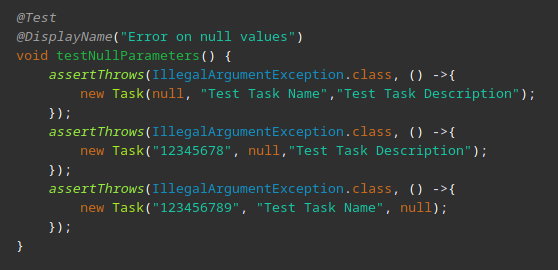
CS-320 Module 7 Project 2

April 16, 2023

**JUnit Testing Summary**

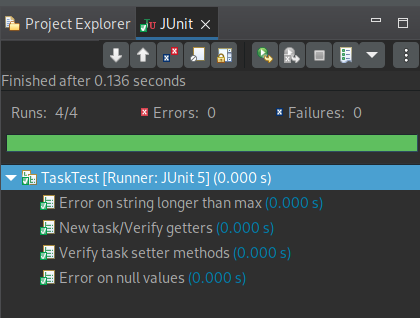
My approach as I was writing and testing each of the three features needed for the software system requested by the client was made up of three basic steps. I first carefully reviewed the requirements for each of the individual features. I then wrote the code to create appropriate classes which contained methods to accomplish each of the behaviors and functions specified in the requirements for each feature. Finally, I constructed JUnit tests that would systematically test the creation of each class as well as each method defined within the class. My goal was to make sure that each individual function and method executed as expected and returned correct output where applicable.

Overall, the tests I wrote were designed to check each of the points laid out in the requirements and confirm that my code met them all. For example, in the code block below, the test is designed to make sure that it is impossible to create a task with any of its attributes missing or set to null. Although the idea of not allowing null values is a singular concept, null values can be introduced into the process in multiple ways. Therefore, the method that creates tasks must be tested in multiple ways to ensure correct execution.



This test checks the creation of new tasks with a null value in each of the three required parameter fields. This approach ensures that the new task will be properly rejected if any parameter provided by the calling function is null.

The JUnit test functions also provide execution time information which allows me to see how long it takes for each of the class methods to complete. With this information it is easy to judge the efficiency of the code. None of the functions within the project are very complex. Nevertheless, the JUnit time reporting confirms that all methods can be run within milliseconds.

  
 In conclusion, I am confident that my tests were both thorough and effective because JUnit test coverage reports show that my tests evaluated more than 90% of the code written for each of the six Java classes that make up the three requested features. By observing the output of the coverage reports, I can see that every method and function in the code is evaluated and checked for correct and intended functionality.

**Reflection**

**Testing Techniques**

To ensure proper functioning, I implemented the JUnit testing framework to conduct static tests, also known as white box tests, on all my code. These tests are designed to evaluate the code without executing the entire product package. Individual internal functions and methods are tested one by one rather than compiling and testing the project.

Because I was working with just a part of an overall product, I did not conduct dynamic, or black box, tests. These types of tests are designed to compile and execute the code, evaluating the functionality of the completed product rather than its internal parts. This type of testing would undoubtedly be used later in the development process if this project were to continue.

I feel that both types of testing are an important part of the development process. In practice, I think that static testing is an excellent way to catch bugs and potential problems early on when they are cheaper and easier to fix. Dynamic testing, on the other hand, is more effective when a project is at a point where some or all of the product can be compiled and executed. At this stage, we can design tests that evaluate how the software would behave while being used. We would be looking more at the results of many functions working together to produce a desired outcome rather than the output returned by individual functions. By using both forms of testing throughout the development lifecycle, we can make sure that the product released functions properly and is easy to use.

**Mindset**

My mindset during the testing and test-writing process was to try to be sure I properly understood the requirements specified for the project and was designing tests that would ensure that all of them were met.

I think an understanding of the complexity and interrelationships within the code is essential to devising and implementing proper tests and testing methods. A tester needs to understand how one function may work with or depend on another to achieve a final result, or even how an external library is used to achieve some type of functionality. Without understanding these concepts, it is difficult, if not impossible, to put together tests that will not only show something is wrong but allow you to see where the problem actually lies.

For example, it is necessary to understand that the TaskService class calls and relies on the Task class to do its job. When testing the TaskService, we don’t just need to make sure that all of the functionality is tested, we also need to understand that any failed test could be the result of bad code in the underlying Task class as well. Along the same lines, we also need to make sure that as we are constructing tests, those tests cover the interactions between the two classes.

To make sure there was no bias in my tests, I simply took the approach of assuming nothing. I started with the premise that I could not be certain that any function or method within the code worked correctly. Neither did I assume that because one function worked, another would work. I treated each requirement and each code block as something separate and independent. I tried to make sure that my tests covered everything from the bottom up. In other words, I started with the lowest level methods, such as setters and getters in the Task class, making sure each of them functioned correctly and then worked my way up to more complex methods in the TaskService class that required more interaction between various parts of the code.

I believe discipline is an important quality for anyone that wants to consistently create the best code possible. A disciplined approach that involves always following best practices while coding and testing will help keep errors and mistakes to a minimum. As many problems as possible should be found and corrected by the developers, not by the clients. This means reviewing and testing for problems should be a constant and consistent part of development, even after you might think the product is finished.

For example, after I submitted my code for the TaskService and associated tests, I realized that I had forgotten to include a test for the delete task function. I could have ignored this and said nothing. Afterall, everything that was tested passed. I could have easily just assumed the delete function would work as well and the missing test probably wouldn’t be noticed. I, however, decided to go back and write the test to verify that the function did, in fact, work correctly. I then created a second submission, stating that I had made a mistake in the first version and that this newer version should be used in its place.