Summary and Reflections

Summary

**Describe your unit testing for each of the three features.**

To what extent was your approach aligned to the software requirements? Support your claims with specific evidence.

**A**: In this project, the requirements were very well laid out for me. In order to ensure that my program adhered to these requirements, I wrote methods that addressed and implemented them directly and then wrote JUnit tests to test them to make sure that they both accepted appropriate responses and rejected inappropriate responses at every opportunity presented. For example, in the Task class, the requirements for the task description must be no longer than 50 characters and must not be null. Therefore, this gave me three tests to make for those two requirements. For the first two, tests to make sure that the method throws an exception in the case of the description either being null or being longer than 50 characters, and the third test would make sure that when presented with a valid input, the program accepted it without fail every single time.

**Defend the overall quality of your tests. In other words, how do you know your JUnit tests were effective based on the coverage percentage?  
A:** I have good reason to believe that my testing coverage was very effective. With a coverage percentage (so far) of 89.7%, I believe that it’s a fantastic start towards making my code on point. While I still may have work to do, I’d like to think that my code was well worded and adhered to the proper practices of being a programmer, including a good word from my professor about the adequacy of the code.

**Describe your experience writing JUnit tests.**

How did you ensure that your code was technically sound? Cite specific lines of code from your tests to illustrate.

**A:** I made sure that all my code was technically sound by constant testing and debugging whenever possible. Doing so definitely had a small hand in slowing me down. For example, this test here:

A screenshot of a computer program

AI-generated content may be incorrect.

…took me a long time to come up with. It’s a test that validates that when a new task is created, it’s given a unique UUID number. It does this by creating three new tasks, grabbing their new uniquely generated UUID number convered to a 10-character string, then testing to see if any of them match with one another. It took me a while to find a method that did this and I’m glad that I spent the time I did to find it, as it helped me greatly improve my testing skills.

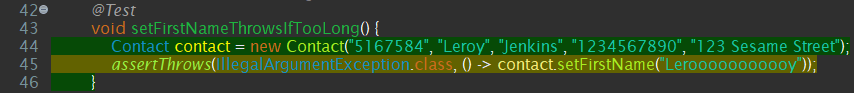
How did you ensure that your code was efficient? Cite specific lines of code from your tests to illustrate.

**A:** I was able to greatly improve the efficiency of my code, especially after the first milestone, by addressing specific concerns from my professor. For example, when writing tests that have multiple assert functions, it’s always best to include them in an assertAll() method in order to make sure that all tests inside the greater test are performed and passed, such as this code block:

A screenshot of a computer code

AI-generated content may be incorrect.

Doing this will ensure that the entirety of the contact’s information is validated through the system every single time and never misses. In addition to that, I have various tests that ensure that any invalid entries, such as “set first name throws if too long” tests like this:



It’s very efficient. It passes a possible input through the instantiation of a new contact and checks for an exception throw when the setFirstName() is called with the argument of a string that’s longer than 10 characters. The condition, naturally, is implemented within the setFirstName() method in the Contact class.

Reflection

**Testing Techniques**

What were the software testing techniques that you employed in this project? Describe their characteristics using specific details.

**A:** The first technique I employed in this project would be static testing. This doesn’t require actually running the code, it simply goes line by line and makes sure that the syntax errors and code smells are caught. JUnit testing then came next whereas you create a separate JUnit file, which contained the tests that would execute specific code blocks and methods and test their expected outcomes.

What were the other software testing techniques that you did not use for this project? Describe their characteristics using specific details.

**A:** One type of testing technique I did not implement would be security testing. This would involve running a Maven test, or a dependency report and would go through all of my code to find any potentially vulnerable dependencies and security risks associated with my API, my databases or anything of the sort.

For each of the techniques you discussed, explain the practical uses and implications for different software development projects and situations.

**A**: For starters, JUnit testing is very important when it comes to real world situations. For example, when dealing with a new startup tech company, if your main website is very new and is constantly undergoing maintenance, you want to make sure you have JUnit testing ensuring that all of your code is still doing what it’s supposed to be doing, even after extensive updating and changing of backend code. As an example for security testing, there will always be Maven dependency reports to ensure that you’re using the latest versions in your API, as anyone can manipulate a potential vulnerability in your code for their own purposes. Security testing can ensure that any errors like that are handled and fixed immediately.

**Mindset**

Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ caution? Why was it important to appreciate the complexity and interrelationships of the code you were testing? Provide specific examples to illustrate your claims.

**A:** It would take a while to explain just how much caution I employed while doing this project. The very moment I discovered how to properly run tests with coverage, my entire life goal for three days straight at one point was to try to achieve 100% coverage. I knew that in order to do that, my code was meant to be as efficient and well-written as possible. I’ll use my previous code as an example, the UUID test. I ran that test so many times in so many different configurations before I was able to see those green lights across the board, and even then, I was still working to make it better. Plenty of places within my work contain remnants of cautionary movements, such as this block:

A screenshot of a computer

AI-generated content may be incorrect.

In order for this test to work, I had to make sure that the ContactService() service object was created, then instantiate the new contact, then add that contact to the service, then delete that same contact, then test to make sure that the number that contact was given no longer exists within the system.

Assess the ways you tried to limit bias in your review of the code. On the software developer side, can you imagine that bias would be a concern if you were responsible for testing your own code? Provide specific examples to illustrate your claims.

**A:** I can say with confidence that I limit my bias in every way imaginable when I review my own code. I look at my code the same way that a bodybuilder looks at himself in the mirror. That is, I still notice every flaw, every inefficiency and want to improve those as quickly and as meaningfully as possible. I’m a bit of a perfectionist to a fault, so one of my favorite things to do, honestly is to go through even my old code and look for ways that I can improve upon it and make it better. My mindset kind of works like this: “Is it good code? No? Make it better. Yes? It could still be better. But always make sure first that it works like it’s supposed to.”

Finally, evaluate the importance of being disciplined in your commitment to quality as a software engineering professional. Why is it important not to cut corners when it comes to writing or testing code? How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.

**A:** As I said, I’m very careful with how I write my code. It may be messy at first, but when it’s time to refine it, I take plenty of time to refine it. It’s very important to beat this idea into your brain early on, so as to not be overwhelmed by it later. Drastic things can occur as a result of poorly tested code, like in the example I gave of Ariane 5 Flight 501 in 1996, where a single block of code that converted a 64-digit floating number to a 16-bit integer caused a major course correction during liftoff and caused the shuttle to break apart mid-atmospheric exit. The longer you take to test your code, the better, and history has proven that many times over.