**Project 2**

Vincent Messina

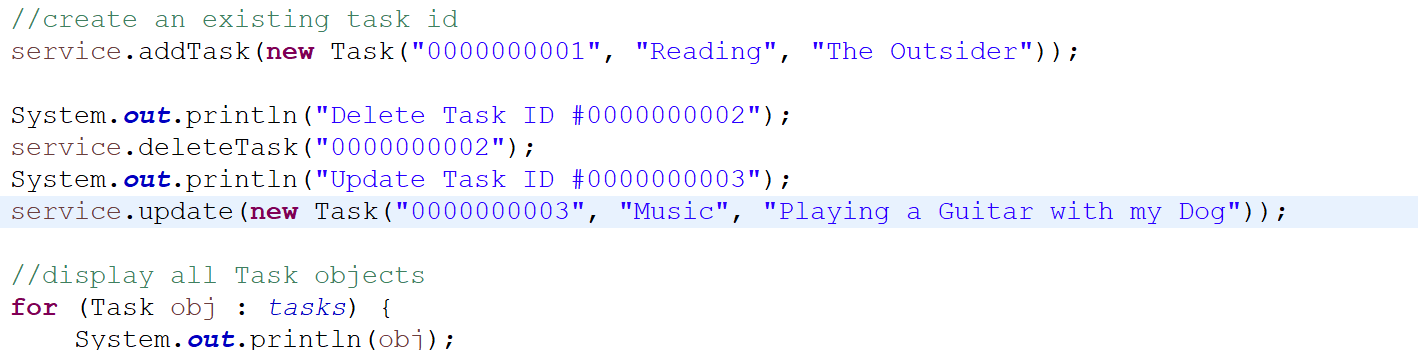
Southern New Hampshire University

CS 320

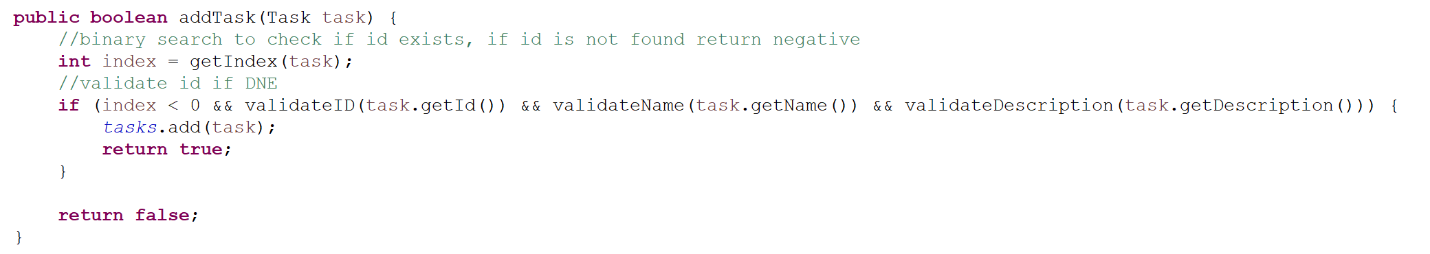
April 23, 2023

In all three milestones, I used JUnit and white box testing. Milestone One consisted of learning how to create a JUnit test. In Milestone Two, I used my knowledge of JUnit tests to test data structures. The data structure I tested in Milestone Two is a Binary Search Tree. I used white box testing by using JUnit. White box testing is when the tester has knowledge of the software being tested. I based my testing approach on the software requirements. For example, in the Contact Class the contactID string could not be longer than 10 characters. So, I tested the software based on the software requirements. Based on a percentage, my JUnit testing was about 75%. My tests did not always work.

Writing JUnit tests has been difficult. I am eager to learn more and be able to write more consistent and clean coded JUnit tests. Each time I wrote a new test, I needed to refactor my code many times before it would build properly. Using JUnit tests has been a helpful skill to learn. To ensure my code was technically sound, I made sure to write in correct syntax and write readable code. I used comments as needed to ensure whoever was reading or testing my code could easily understand what my thought process was.



To ensure my code was efficient, I wrote clean and industry correct code. I used powerful data structures such as a Binary Search Tree. I also tested and debugged lines of my code. For example, I used a Binary Search Tree to check if the inputs were validated to help the unit tests run more efficiently.



Some software testing techniques that I used in this project include Static testing, Dynamic testing, and JUnit testing. Static testing tests software and work products without executing code. Static testing can be used to test anything related to the project and can be sped up by using static testing tools. Dynamic testing tests software and work products by executing code. The purpose of Dynamic testing is to ensure the software functions properly. You provide input for the software and Dynamic testing validates that input. JUnit testing is used to write and execute automatic tests. JUnit testing is a form of White Box testing.

Some software testing techniques that I did not use for this project include Black Box Testing, Grey Box Testing, and Exploratory Testing. Black Box testing is useful because the tester needs no knowledge of programming languages. The tester uses the software from the prospective of the user. Grey Box testing involves a tester who has knowledge of programming languages but is testing from the user’s perspective. This is useful because the tester might be able to catch some mistakes with their knowledge of programming languages that a regular user might not. Exploratory testing is when a tester tests a software without any plan. It allows tester to adapt to the test without having to write a new test case. Black Box testing is practically useful in many different software and projects. For example, a user uses Black Box testing when testing a video game. The tester is playing the game from the users perspective and providing feedback with no knowledge of programming languages. Grey Box testing is very useful because the tester can use their knowledge to catch a mistake a regular user might not catch. However, they might look over some mistakes because of their familiarity of the software. Exploratory testing is useful to allow testers to adapt to their testing environment.

For this project, I neglected my programmer mindset and adopted the mindset of a tester. The testing mindset makes you focus on all parts of a program. It is a new way of thinking that makes you run the tests and apply it to your code. I used extreme caution to not pass over any implications in the code.

Bias can be a concern when testing your own code. Because you wrote the code, you already have a familiarity of it and could not notice simple mistakes. Someone who tests their own code is much more likely to miss a forgotten semicolon or curly bracket compared to someone who is testing code that is not theirs.

It is extremely important to be disciplined in your commitment to quality. Being lazy when it comes to quality code can cause the whole program to fail. One mistake is all a program needs to not build properly and fail. When cutting corners testing code, you will miss many mistakes that you would not have if you did not cut corners. This can lead to you actually spending more time testing a project or software because you will become familiar with the code. After you become familiar with the code it is much easier to miss mistakes.

**References**

Morgan, P., Hambling, B., Samaroo, A., Thompson, G., & Williams, P. (2019). *Software Testing: An ISTQB-BCS Certified Tester Foundation Guide*. BCS Learning and Development.