Project Two

Software requirements were provided in the project one guidelines in rubric. These guidelines stated the criteria that my program should meet and set the definition of done for the work I needed to complete. For example, it was stated that the first name variable couldn’t be longer than 10 characters. It was also stated that the first name variable couldn’t be null. In order to achieve this I used the code seen below.

A computer screen shot of a code

Description automatically generated

This example shows that the requirements seem to have been met. To ensure that this met the requirements I had to write tests for both the null and less than 10-character requirements. The code below shows how I tested that.

A screen shot of a computer code

Description automatically generated

A computer code on a black background

Description automatically generated

Both tests run through JUnit, and tests for success, failure or errors. They cover the requirements for adding, updating, deleting, and modifying. All tests ran and were completed with no errors. I tried ensuring that my code was both simple yet efficient. Ensuring my code was easy to read via comments and proper spacing has always been a straightforward process for me. The part that has usually been more difficult has been ensuring that my code is efficient. In order to make my code efficient, I attempted to ensure that I cut back on repeated code within my program. I feel like I especially did a good job of this in my task and taskService classes. I tried a different way of accomplishing the task of searching through my array list by have a specific function dedicated to it and simple calling it in all of my other functions. This helped make my code look cleaner and undoubtedly made it more efficient. The change can be seen below with how I handled it for deleting a task.

A computer screen shot of text

Description automatically generated

A screen shot of a computer

Description automatically generated

For the purposes of this project, objects within each milestone were individually tested to ensure the code functioned as intended. The results of each of these tests were then displayed within the console by JUnit as either pass, fail, or error. An example of a testing technique I used was the Equivalence Partition technique. This can be used to check that an input is within a certain range. An example of this within my program was testing for appointment description. An appointment description has a character limit of 50, as such I needed to ensure that any description that goes in is under that character limit. Below can be seen how I accomplished this.

A screen shot of a computer program

Description automatically generated

Once a description is past 50 characters it is cut off. There were several other testing techniques I didn’t use within my project. A prime example of this is decision table testing. Decision table testing is used to define functions and the conditions which the functions will use through testing combinations. Equivalence partition testing was used throughout several of the milestones to test character limits on variables such as descriptions, names, and IDs. Any situation where character or word limits need to be tested would greatly benefit from this testing. Another type of testing I didn’t cover within this project is State Transition testing. State transition tests for changes in state between current and past states. This can test to determine whether a certain feature of a program is “on” or “off” for example. Outside of this, any situation where certain variables or functions are changing from past to current states could benefit from this.

My perspective on programming has unquestionably changed to a perspective of considering more caution with how I program. Programming from the perspective of someone testing code has made me consider different areas which could result in more bugs/issues in the final product. Tests are directly connected with the code they are testing. While I tested my code, I tried to ensure I had as little bias as possible with my own code. Ensuring there is minimal bias while coding ensures that your code will be as effective as it could be and reduces leniency within your code to prevent you from cutting corners or not testing all that you should test. It is important to ensure that your approach towards your own code is similar to that which you would take towards another person’s code. I consider programming to be similar to other activities one might do. In order to maintain discipline with my coding, I have to ensure that I am constantly practicing the skill. This is done by avoiding shortcuts, and continuously learning enables me to maintain good habits when it comes to how I program. Another important aspect of maintaining disciple for me is taking breaks while coding. I’ve noticed that the longer programming session I have the less disciplined I am and the easier it is for me to overlook smaller mistakes. Generally I try to ensure that while programming I take “brain breaks” to keep my mind fresh, focused, and disciplined.

References:

Using equivalence partitioning in quality assurance tests. (n.d.). https://www.ranorex.com/blog/using-equivalence-partitioning/

GeeksforGeeks. (2023, December 6). *Software testing techniques*. https://www.geeksforgeeks.org/software-testing-techniques/

Ufimtsev, L. (2024, February 11). *Java code coverage in Eclipse*. Red Hat Developer. https://developers.redhat.com/blog/2017/10/06/java-code-coverage-eclipse