**Summary**

A screen shot of a computer

Description automatically generatedA computer screen shot of a computer code

Description automatically generated When developing the Unit testing it was important to align my approach with the requirements of the software. How I achieved this was by writing down the requirements of each and then first setting up the software with those assigned tasks by using private and public strings to set up each task. Some examples of this is in the images to the right. When assigning the reuirments before writing the rest fo the software it help me make sure I had all the requirments, and if I missed one requirment the IDE would let me know that There is an unsued item and then I would go back and look where to incorperate the item.

A screenshot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated The quality of myt Junit tests were all right I can see places for room for improvement whiel vat the same time they were effective in letting me know where The software messed up and how I either needed to incorperate a piece I missed or the software was reading the code wrongly due to me haivng writing errors. I was able to effectvly see these issues after I ran the JUnit test whichc an be seen with successful and nonsuccessful runs in the images to the left. Although there were numerious issues the Junit test made me aware of the issue and its location which would speed up the time needed to find the issue without it.

A computer screen shot of text

Description automatically generatedA screen shot of a computer program

Description automatically generated To ensure that my code was tehcnivally sound I first ensured that all requiremtns were in place then I used example from the internet specfically JUnits own website which had a good user guide to instruct me as I progressed through the testing code. There were issues that came up in the initial code but as ther assignemnts progressed I was able to devlop more tehcnically sound code. To the right is an example of Testing that came out tehcnically sound. Using the same tehcnique as above helped me also ensure the code was efficient I made the code as simple as I could but at the same time hit every objective which helped to lower time needed to right the code and cleanup of the code afterwards. An exmaple fo this can be seen to the left.

**Reflection**

Throughout this project an employed numerous software testing techniques to ensure the code ran as efficient as it could with different techniques happening at different parts of the code as well as some of the testing types happening numerous times through the life of the code’s development. Unit testing was a large portion of the testing I did but these testing techniques could only be implemented once the test was written. In the meantime, I used testing techniques such as manual testing where I would go through manually and make sure everything looked good. This technique was efficient at catching obvious mistakes such as spelling but missed more complex issues. Another testing technique was exploratory testing which I used to look at all aspects of the code which helped me find defects in the code. These testing techniques helped me to ensure the code was written properly and correct mistakes throughout the building of the code.

Some Software techniques I didn’t use during this project were integration testing as I didn’t need the different assignments to work together so integration wasn’t an issue. Some testing techniques weren’t chosen for us because I felt they didn’t apply to this situation, or it would have taken extra time which I didn’t feel was necessary to achieve the task that the testing techniques I did use were able to accomplish. These techniques, the ones I used and didn’t use all play an important role in testing and will present themselves at different items. For example, Unit testing such as Junit testing can be implemented in all sorts of software projects but it’s important to remember they must be built for specific projects which means extra time will need to be dedicated for that. On the other hand, exploratory testing can and should be used in all projects and can happen at any part of the project which can catch issues early and often preventing an excess amount of time needed for corrections later. In overall dependent on the project and its goals to assign different testing types to it with the idea of testing often and early and then having an automated test verify the work after wards to make sure nothing is missed.

A screenshot of a computer

Description automatically generated When working on this project my mindset was geared towards accomplishing the tasks set in the assignment’s requirements. My level of caution I feel was highest on the first assignment and went down as I progressed to our third major assignment. I will say that when my caution started to dwindle into the third assignment, I felt confident with what I was doing but I ended up with more errors due to much confidence in what I was doing. This can be shown in the image showing the issues between open project and the other in the images to the right showing the difference between two different classes. This showed me the importance of being cautious and not letting my confidence get int eh way of the complexity and interrelationships of the code. Limiting bias when developing code is especially important as it can cause issues. Limiting bias in these assignments was easier than if I was creating my own project due to having a specific requirements for both and have a specific outcome this means my goal was to accomplish the task to get all of the points in the assignment but in a project where I am in charge of the requirements I can see where the importance of limiting bias so I don’t miss anything. Lastly, being disciplined is a skill that is necessary for everyone but plays a specifically important role in software due to the catastrophic issues that may arise if there is a lack of discipline. When developing the code there was time I was copying and pasting lines back-to-back and then going back to make changes and I would miss things which caused issues down the line. This ultimately caused me to spend more time making corrections when I should have just gone line to line and not cut corners. This ultimately showed me that discipline and taking my time is important in preventing issues which can directly influence how people see my work or the company I’m working for.

**Citations**

Baskirt, O. (2022, August 22). *8 important software testing techniques*. Software Test Academy. https://www.swtestacademy.com/software-testing-techniques/

Brannen, S., Bechtold, S., Link, J., Merdes, M., Philipp, M., Rancourt, J., & Stein, C. (2023, April 22). *JUnit 5 User Guide*. JUnit 5 user guide. https://junit.org/junit5/docs/current/user-guide/