Nicholas Gorecki

21 March, 2018

Lab 2

https://github.com/msoe-2832/se2832-lab-2-Nickonnes.git

Introduction

With this lab, I am trying to make sure that there are no errors, faults, or flaws with the requirements of the CircularQueue. I must make sure that the functionality and conceptual requirements of the CircularQueue are met, and that there is no possible way to create an error in the system that does not meet said requirements.

Testing Strategy

I focused my testing strategy solely on Unit Testing. I did not do any of the UnsupportedOperationException methods, as they only had one line of code that was meant to throw an exception, no matter the parameters. I organized each of these unit tests by function, adding elements and then executing operations on the specified test for the particular function I was trying to test for almost all possible outcomes.

Fault Locations

Unfortunately, my Junit tests would not run, as I kept getting an error. I was just basing these tests off assumption and didn't really have the time to fix the issue. I noticed that there were some areas that I thought had issues, such as the offer method. If, lets say, the capacity and size were both five, it would enter the if statement and make the size six, while the capacity was still five. Therefore, to fix this, I removed the <= and changed it to <.

Right / Wrong

Most things went correct. There were some issues dealing with the circular procedure of the Queue, such as in the Offer method explained above. Separating tests into different units made the simplification of finding faults easier, as I was able to focus specifically on one function rather than a concept represented by multiple functions.

Conclusions

First, I learned that I do need to start a little earlier. I realize I wasn't as specific on the tests as I wish I could have been. I also want to understand why my Junit won't run any tests (I'll ask about that later). I did learn how to properly identify certain errors by analysis. I learned that writing tests simplifies the thinking process on what should and shouldn't be returned from a function, given a range or requirements for it. Documenting tests and knowing what you've done and what you haven't done makes it a lot easier to think in more detail when testing code. I also learned that code coverage does not define if a structure or chunk of code has been fully tested or not. Now I know that I must be more specific on testing and think in way more detail than I have in this lab if I want to make sure that I properly test code. Using different testing strategies may also be very useful.