Unit Testing with JUnit

By Team Voltron

Thomas Bassa, Gregory Carkin, Umar Idris, Michael Philotoff

Introduction

The goal of this mini project was to help understand how JUnit testing can be used to help find errors within the code and help fix the code. We were given a fully implemented circular queue class, however, it was not tested very well. It was quickly discovered to be very buggy, and our objective was to fully test and debug the code. To accomplish this, JUnit tests were written to uncover defects, which were then fixed and subsequently tested.

Testing Strategy

The testing strategy what to divide the code into sections, then try to get one sections of the code working at a time. The code was sectioned off by method, assigning a section for each substantial method of the code under test. Thus the tests were divided up by the methods they covered. Also, there is one testing method that covers all the functionality of the code, to make sure it all works together. Using this testing strategy allowed us to show our progress as we completed different portions of the project. This would be great if there were a customer who wanted a demonstration of our progress, as we would have functional parts that we could demonstrate to the customer. The only real issue with using this testing strategy it leaves some functionality of the code untestable until certain portions of the code have been fixed by prior testing and debugging.

Fault Locations

Function of CircularQueue	Issues and Resolutions
Constructor	CircularQueue was not accepting queue lengths of anything but 0. If statement condition was checking if maxQueueSize was not zero. Resolved by checking if maxQueueSize is less than or equal to zero
offer(E element)	offer was not putting the element in the next open position of the queue. The element being added was always added to the head+1 position, then head was set to itself. Resolved by adding the element to the tail, then updating the tail position by 1.
remove()	remove was causing an IndexIOutOfBounds exception to be thrown. The element being removed was from position tail - 1, then tail -1 was set to null, and tail was set to itself. This was resolved by removing the element at head, then setting head to null, then updating the position of head.
poll()	poll was causing indexoutofbounds exception to be thrown. The element being polled was at position tail-1, then that position was set to null. Resolved by polling element at head, then setting head to null.
element()	element was causing indexoutofbounds exception to be thrown. Element being returned was at tail-1. This was resolved by returning the element at head.
peek()	peek was causing indexoutofbounds exception to be thrown. The element being returned was at tail-1. This was resolved by returning element at head.
isEmpty()	isEmpty was returning true when there were elements in the queue. If statement was checking that the size of the queue was not zero. If it was not 0 then the queue was considered empty. This was resolved by comparing the queue size against 0.

Conclusions

Lessons Learned

Umar

This project built on the previous one, enforcing my knowledge on using JUnit and performing code coverage. Not only did we identify the bugs, we also had to fix them, which was a good experience as well. Introducing the use of version control was a good idea too. This will help us keep all our code in a central place. With Git, we assign tasks by creating new issues and assigning it to each member. When done, we close it, helping us keep track of our overall progress which is very crucial.

Michael

The lessons that I learned from mini project 2 were how to fix faults within code, how to use JUnit to help find the faults in code, and use the JUnit tests after the fix has be made to ensure the faults were resolved. Also, my knowledge of JUnit tests has expanded since last project on the usage of JUnit testing. This project also increased my knowledge of use Git with Eclipse, since most of the time I use the bash commands or the Windows GUI that can be downloaded from GitHub.

Greg

From this project I became more familiar with developing JUnit tests to account for most if not all possible cases. Additionally I learned how to implement JUnit tests to narrow down and find where bugs occur. Lastly, through this project I learned more about the capabilities of GitHub. Previously I used it similar to SVN as version control, but now I know about its ability to track issues and assign tasks.

Thomas

This project has further solidified my skill at JUnit testing, as well as my ability to judge what should be tested from an interface description. This project also taught me to use a number of tools for source code management: GitHub and its issue tracker, the Eclipse Git & GitHub integration plugins, and SourceTree. I believe I did well at coordinating the team's work after setting up the repositories and tools.

Future Improvements

For the next code project, it would be helpful to have a little more setup time to coordinate our workflow. The team could use more coordination in dividing the work, which GitHub Issues should help us improve, next time. A more challenging code assignment would be accepted; make it a little harder to debug and fix the faults within the code. The relative simplicity of this codebase could allow just one of us to complete everything in a few concentrated days.