John Mobarry

CMSC 204

3/14/2020

**Assignment 3 Reflection**

Implementing Assignment #3 was a beneficial experience as it resulted in a greater knowledge of the assigned topics and of the mercurial Intellij IDE. Double sided linked lists was one of the topics from before the midterm that I felt fairly hazy about and this project really helped me understand their implementation. I was also harshly reminded of how astray a project can get if the programmer is loose with their public vs static declarations. Also, in order to get the program to debug without notifications regarding depreciation I had to learn how to run my java project in an earlier version of the language. All together this project helped me gain competency in one of my weaknesses and increased my understanding of the IDE that I had been running in default mode.

During the implementation of assignment three I was tested on my knowledge of double ended linked lists. Conceptually I understood how to implement the nodes in order to make a linked list with next direction pointers, but it took some reflection to implement the double sided linked list. When creating the node inner class I had to quickly reflect on how I would need to adapt the data structure in order to be multidirectional. Each node needed to have a pointer implemented for their “next” and their “previous”. When a node was inserted to the front of the list the new node would need to next point to the current front of the list and the current front of the list would need to previous point to the inserted node. The reference to the“front node” would then need to be transferred over to the newly inserted node. Subsequently when inserting to the end of the double sided linked list you would need to set a next pointer from the current last element to the inserted element and the inserted element will need to set a previous pointer to the current last element. Then you would subsequently set the inserted element to the last element. Although I understood this concept in theory from my readings in the textbook, implementation was more difficult than previously expected as the logic needs to avoid the errors caused by edge cases.

In my final stretch towards completing the sorted double sided linked list I was experiencing a fatal error while completing the JUnit tests. My tests were failing on the generic types and was always attempting to handle the lists as if they were all for car objects. I reviewed my code, scoured the internet and could find no reason that the IDE would want to treat these objects as Cars instead of the generic types. The issue in my code is that I had declared the defined comparator within the class of static type in my code. So when the comparator was initialized in JUnit set up the comparator was defined as of type car and then statically remained that way for all the subsequent tests. For the longest time I was attempting to understand where things were going wrong through the debugger and could not get back to the source of the issue. I spent over an hour trying to see if there was an issue with my IDE or if I needed to load up another instance of Java. I then switched to trying to fix the issue within Oracle instead of IntelliJ and the problem remained. After taking a break and walking around the house I then realized that all of the issues in my code were simply due to a static vs. private declaration. After I made this change my program ran correctly and all my tests passed.

While completing the JUnit testing for the sorted doubly linked list my compiler kept creating an error regarding double boxing (for example (Double)8). My compiler stated that it was an error due to depreciation and it would not verify correctly if I passed the JUnit 4 tests. In order to remedy this issue I learned how to revert to a previous version of Java where the double boxing issue was not depreciated. By reading over the error message I came to the conclusion that the act of double boxing had been depreciated since Java “9”. In order to revert back to version 8 I had to go into the runtime and general settings and choose a different JDE to build and run my project in. By pivoting to running my java project in java 8 i was able to remove the issues with the double boxing and get my JUnit tests to pass in a green complete state instead of a yellow for potential issues.

This assignment ended up taking more time than I had thought to budget as I experienced multiple errors that I had to think outside of the box in order to remedy. Simple things such as a single word in a java project can prevent it from running correctly even if your logic checks out. In the future I will now be much more careful in determining which processes in my program warrant the static declaration. Also I now know how to quickly transition IntelliJ from one version to another version of Java if I am having issues with deprecated methods/processes. I learned valuable lessons from this assignment and feel much stronger in my ability to implement the numerous ADTs learned this semester.