Michael Wong
Professor Potika
CS 146 Sec. 02
September 20, 2018

Project 1 Shuffle Songs - LinkedList Report

For this project, I was tasked to solve two problems. The first problem was take a file containing a list of songs and shuffle them. Then, after they were shuffled, the shuffled list of songs would be outputted into a seperate file. The second problem was to simulate an elimination process used by a King in an ancient land. The process consisted of having a line of prisoners and then counting them off to get out of the line. The King does this until there is only one prisoner left and he/she would be granted freedom.

The first task regarding the shuffling of songs was a fairly simple and straightforward task. My approach was to read through the given file with the list of songs and input them into an array. I would then use the Fisher-Yates Algorithm to shuffle the array which includes swapping a random array index with later array elements. Once the array was shuffled, I would simply write the contents of the array back to a new output file. This is all done using a main method. For testing to see if the program shuffled the playlist correctly, I created a JUnit tester to test my program.

Some challenges that I faced with the song shuffling task regarded the equality of what the expected output was and my actual output. The playlist provided to me contained special characters that would not translate over to the new file output I created. To accommodate this issue, I created a new input playlist file, “playlist2.txt”, and a new expected output file, “Target2a.txt” and “Target1a.txt”, that replaced the special characters with regular characters. After this adjustment, my program cleared my JUnit test. The expected output equaled the actual output. The reading and writing functions were ran through a Java main method, not the JUnit test. I believe I was suppose to read, write and test my files through the JUnit test, so I was unable to run my code entirely through the JUnit test. Only the testing portion was done in JUnit. In addition, the “myRandGen” was also not done through the JUnit test. It was also implemented in the main method.

The second task of implementing a circular linked list was a much more challenging task then the song shuffling problem. The first problem I faced was creating the circular linked list. After creating the Node class, I had to construct the list itself. I was stuck between creating a collection of nodes or creating a linked list in the form of LinkedList<> list = new LinkedList();. First my linked list was just a collection of nodes, but afterwards I created a constructor for the class CircularLLGame that would hold the Nodes as a linked list, combining my two ideas. I used two instance variables, head and current to construct the list. Current would start at the head and

adding would be done after the head with current always being the tail. Once they were all added, I set current's next as the head, completing the circular linked list.

Another problem that I ran into was making sure the list was empty. Prior to my fix, I set the head equal to a new node with a value of 1 in the declaration. Since the head is a part of the list, is is impossible for the list to be empty. My solution was to make sure the constructor takes a parameter of at least 1 before creating the head and the rest of the list. If the parameter was 0 or less, the head would never be created nor would additional nodes be added resulting in an empty list.

For deleting the nodes, I originally had a simple remove function that deleted nodes based on their index. This function was not suited for a circular linked list since it could not loop back to the beginning. I tried to implement an if statement where if the index was bigger than the list's size, there would be a change variable that would loop it back. This method was not hard to follow, so I then tried traversing the list using the current node as a marker and this was much more logical. Current would traverse until the node before the node that was to be deleted. It would set current's next to current's second next, deleting current's next.

Lastly, implementing the deletion portion of the JUnit was a struggle. In the CircularLLGame class, I created methods to test if the linked list was empty and the size of the list. To insert into the list, I used its constructor. These test and calls passed with little difficulty. However, when I tried to call the delete method, the compiler threw me errors about static references with the tester and delete function. To solve this, I created a method, "eliminate", that basically just calls the delete method with the same parameter. It looks strange and odd to have, but once I called eliminate in the JUnit test, the code compiled and ran as expected. I think the problem resided in calling a static vs no-static method. Either way, I was provided test cases for this problem and utilized them in my JUnit test. Upon running it, the expected output matched the actual output.

In conclusion, I was successfully able to implement and solve the two problems presented to me in this project. The instructions and task of the problems were easy to understand and I was able to complete them. Most of my troubles came from the JUnits, the non-coding aspects, and creating a circular linked list. JUnits were a brand new topic for me, so it was difficult for me to understand their purpose and how to implement them. I eventually found out that they were a way to test if your code ran correctly and yielded the correct results by using assertions to check different steps in the runtime. I wasn't sure if I had to test each individual method or each test case. I decided to just test the ultimate result after the program finished for correctness. Overall during this project, my hardest task were implementing the JUnits, but I was able to complete the implementations of the problems presented to me in this project.