Chun-Wei Chen

Assignment #5 Written Report

2012/02/24

I approached the assignment by reviewing the slide and the sample program of LinkedList we used in class and doing the practice activities on the Practice-It. After doing some practices, I felt more comfortable with using LinkedList, and then I started coding. I first tried writing the JUnit test for the assignment, but I was not used write the test before the implementation; therefore, I still did the implementation part first. However, I was also thinking about what might be the case I needed to test while coding, so I wrote the test immediately after finishing coding each method.

Choosing the test cases wasn’t different from what I had done before, so it was not a big problem; nevertheless, writing the JUnit tests was a new thing for me. I took a look at the JUnit test sample code and the JavaDocs of the JUnit on the website to make sure how to use test methods like assertTrue and assertEquals. Even though I wrote the Unit test for each method after finishing the implementation of that method, sometimes I would still forgot to test some case I had already came up during coding the implementation. After finishing this assignment, I feel a little more comfortable with writing JUnit tests; however, I think I still need more practice to change my habit to write the JUnit test before the implementation.

I think my program meet the specification since it passed my entire JUnit tests. There are several special cases I encountered during the coding. The first one is in the equals. Since the parameter is Object, I need to consider the Object o passed in might be some Object not instance of Path or might reference to null; consequently, I check if the Object provided is a Path Object and not reference to null at the beginning of the equals method. The second special case is in deleteLocation method. Since calling a method should not break the class invariant of the Path class, a Path with only one location should not change the Path when calling the deleteLocation. I check if the size, the instance variable of the number of locations I used in Path, is one first. If the size is one, then return false instantly and do not change the Path. Another class invariant is two adjacent locations cannot be identical; therefore, there should still contains no two adjacent identical locations after calling deleteLocation. I took care that by using an if block to check if the locations before and after the one will be deleted are the same. If they are the same, then return false and do not change the Path. There are also some special cases in the insertLocation method. If the size of Path is one, insertLocation works the same as appendLocation. Another one is insert-at-the-end case. If inserting the new location at the end of the Path creates the smallest possible increase in total path length, then it should be inserted at the end; therefore, I checked if the distance between new location and the last location increase the least in total path length. The insertLocation method should ensure the Path after calling it still obey the class invariant, so the new location should not be insert if the new location intended to be inserted is already exist in the Path.

This entire assignment involves LinkedList and JUnit testing. It is really helpful for me to fully understand how to make use of LinkedList and JUnit test. Although where should use p and where should use p.next is sometimes confusing, it is worthy to figure it out during the assignment. I feel more confident with using LinkedList than before working the assignment. I find JUnit test is helpful to check if the method is correct or not. I don’t need to check the output myself; the JUnit test will tell me where my code does not meet my expected. For Example, I tested toString method and made use of toString method to test some other methods’ output. I used assertTrue to check if the output is in the correct order. If there is something different from what I expected, the test progress bar will show red and tell me what problem it has. I think if I get used to write JUnit test first, it will help me a lot with doing the implementation part.