Shahid Bilal Razzaq
U0996062
Assignment 7

Analysis Document

1. Have you worked with more than one partner yet? Remember, you are required to switch at least once this semester.

Yes, I have worked with two different partners so far, for the next paired assignment I will be working with a different partner.

2. In the LinkedListStack class, the stack data structure is implemented using a doubly-linked list. Would it be better to use a singly-linked list instead? Defend your answer.

In a DoublyLinkedList, each node has a reference to not only the next nod, but also the previous node. A SinglyLinkedList does not store a reference to a node's previous node. This double-linkage gives a DLL advantage over a SLL in operations where it would be useful to know what the previous node is, such as deletion of a node at a certain position. This advantage for a DLL comes at a slight cost, as it uses a bit more memory to store these extra references.

In reality, for such a short stack (in the case for this assignment), there is little to no advantage for using one over the other, however, as the stack would get larger with more items pushed into it, deletion (popping) performance would give advantage to the DLL, as it would be faster to peek at the item on the top of the list.

3. Would it be possible to replace the instance of DoublyLinkedList in the LinkedListStack class with an instance of Java's LinkedList? Why or why not?

Yes, it is possible to replace the instance of DoublyLinkedList with an instance of Java's LinkedList. This is because the operations for performing the basic tasks of manipulating a stack (push, pop, peek, clear, etc.) can be performed with Java's LinkedList as the backing list, because the needed methods are available. As an example, for peeking at an item on the top of the stack, we need to be able to return the last item in the stack (assuming the top item is the last item added), Java's linked list class provides a method to return the last item.

4. Comment on the efficiency of your time spent developing the LinkedListStack class.

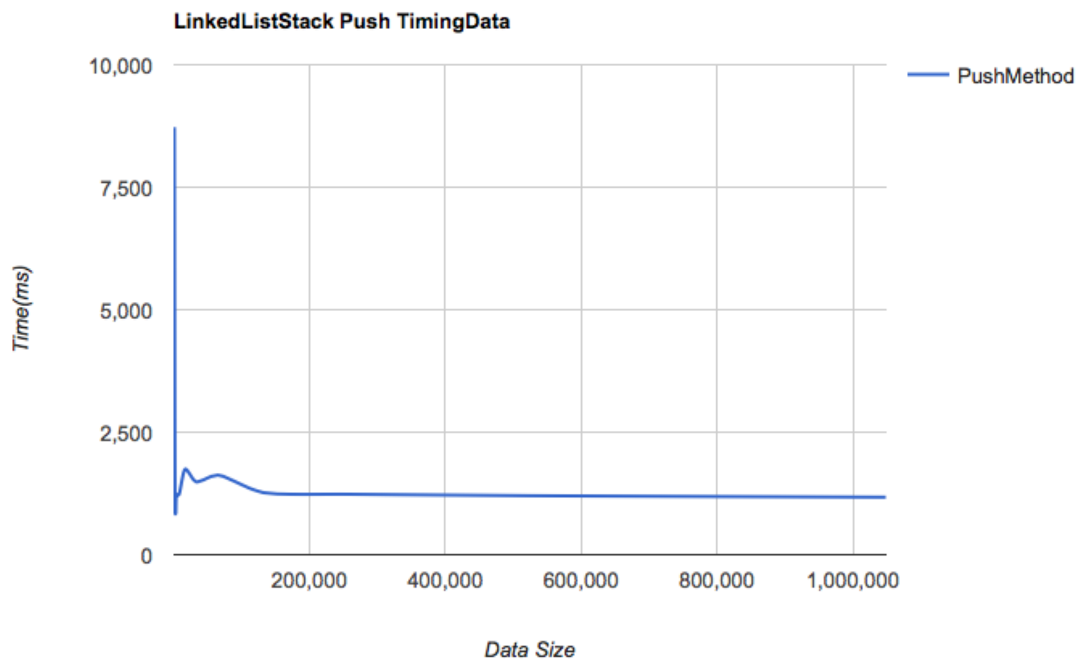
Creating this class was very quick and efficient, because the methods created using the backing doubly linked list class, and its methods were used. For example, in developing the .pop() method in the LinkedListStack class, the return for the method was the backing DoublyLinkedList's .removeLast() method, and so on and so forth for the other methods.

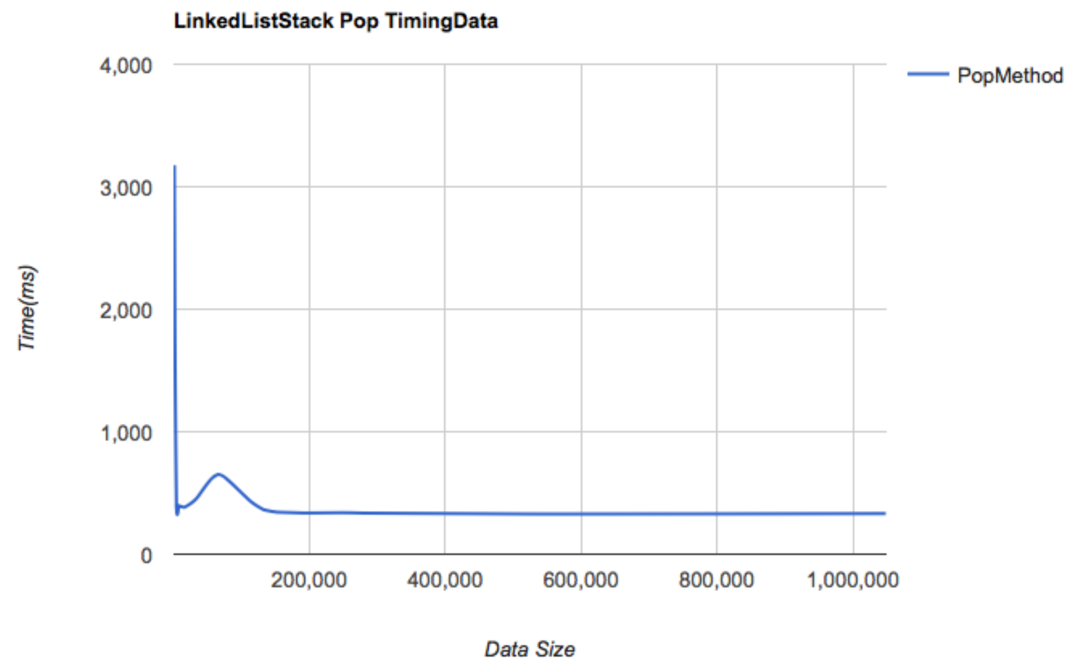
5. Note that the line and column number given by BalancedSymbolChecker indicate the location in a file where an unmatched symbol is detected (i.e., where the closing symbol is expected). Explain how you would also keep track of the line and column number of the unmatched opening symbol. For example, in Class1.java, the unmatched symbol is detected at line 6 and column 1, but the original '(' is located at line 2 and column 24.

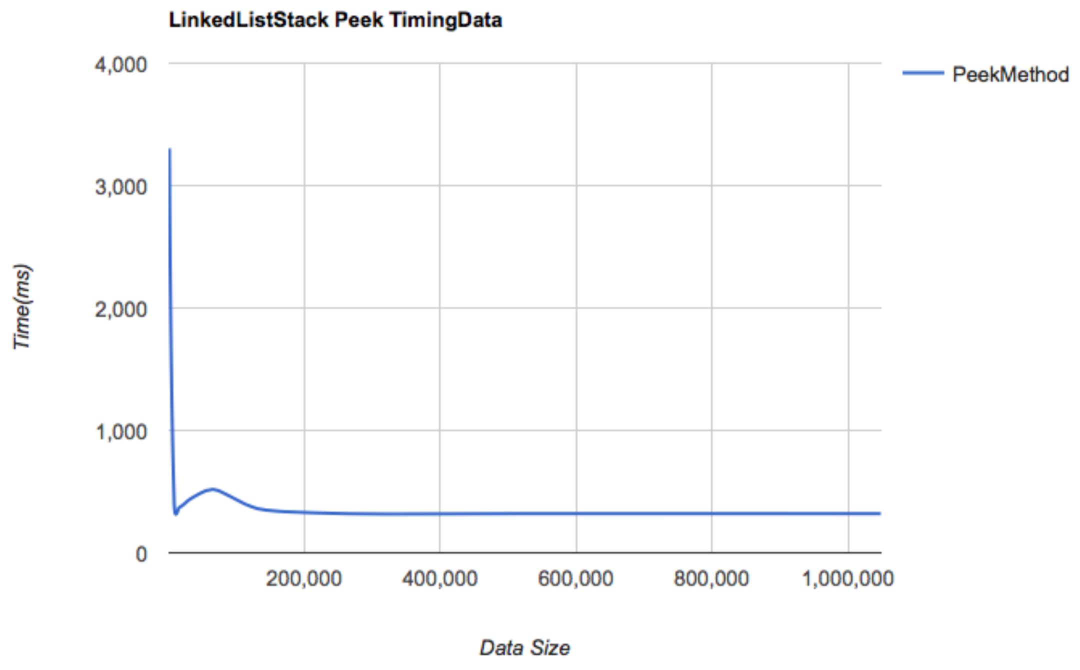
The methodology I used to scan through the file and examine symbols for matching involved creating a Scanner object from the input file, converting each line of the scanner into a String, and iterating through the string looking for characters to push and/ or pop onto the stack. Line count was accomplished by creating a counter variable that incremented each time the Scanner object of the file had another line. If an error, or mismatch was reported by the symbol checking statements, the while-loop incrementing through the lines (using the .hasNextLine() method) would break out, and line increment stopped on the line of the error.

Column increment was accomplished similarly using a counter variable, while iterating through a String made from the .nextLine(); method.. If an error and/or mismatch occurred, the loop would break and the incrementing would stop, thus giving the column number.

6. Collect and plot running times in order to determine if the running times of the LinkedListStack methods push, pop, and peek are O(1) as expected.







The experiments were set up to time pushing increasingly larger integers on to the stack, popping and peeking increasingly larger integers from increasingly large stacks. As expected all three operations show O(1) timing complexity as expected for these operations.

7. How many hours did you spend on this assignment?

About 10-12 (Most of it spent fixing up my DLL implementation)