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

Not yet, but my partner for the next assignment will be a different one.

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.

Because LinkedListStack only targets the last element in the stack and goes backward, the nodes would only have to keep track of their previous nodes, not their next ones. This means that using a singly-linked list would work with LinkedListStack. Using a singly-linked list would somewhat decrease the time it took to push and pop elements off the stack, but this decrease wouldn't be very big because the operations used would still have a complexity of O(c). The amount of code needed would also be less, so that is an advantage. Despite the improvements being small a singly-linked list would be the better choice just because there's no sense in having the functionality of double links if LinkedListStack only requires single links.

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 and it would still be fully functional. This is because Java's LinkedList is essentially the same as my DoublyLinkedList, only with more features and slightly better performance (this was shown in the last assignment's analysis). It has all the same methods that function in the same way, as well as being doubly-linked (not that that matters for LinkedListStack). This can be proven by replacing DoublyLinkedList with LinkedList in the LinkedListStack class file. It still compiled and passed all the tests I had written.

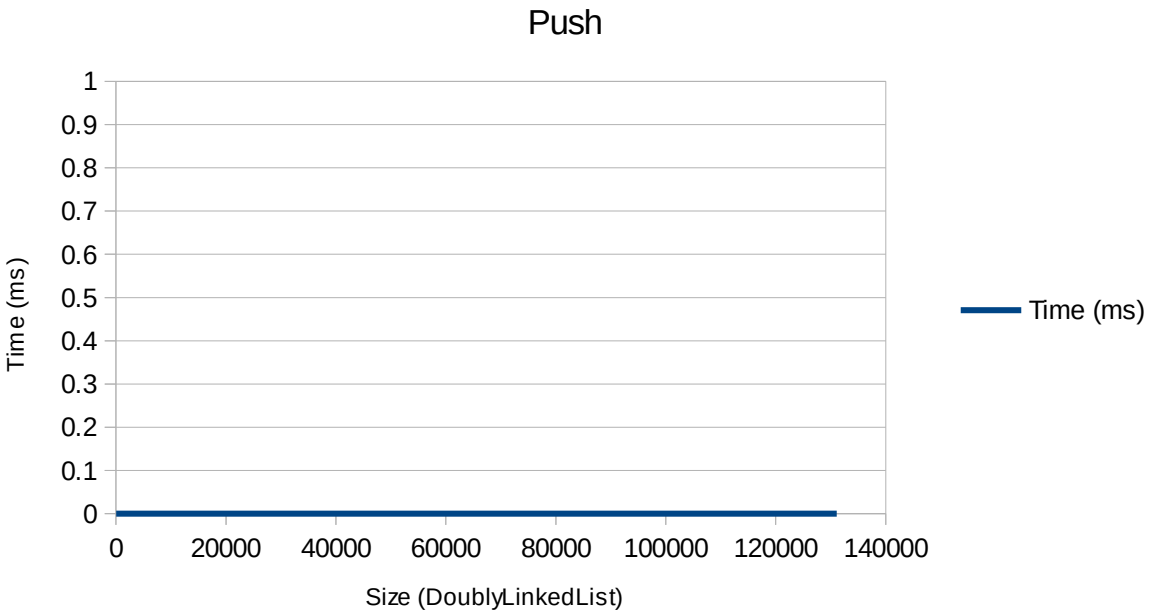
4. Comment on the efficiency of your time spent developing the LinkedListStack class.

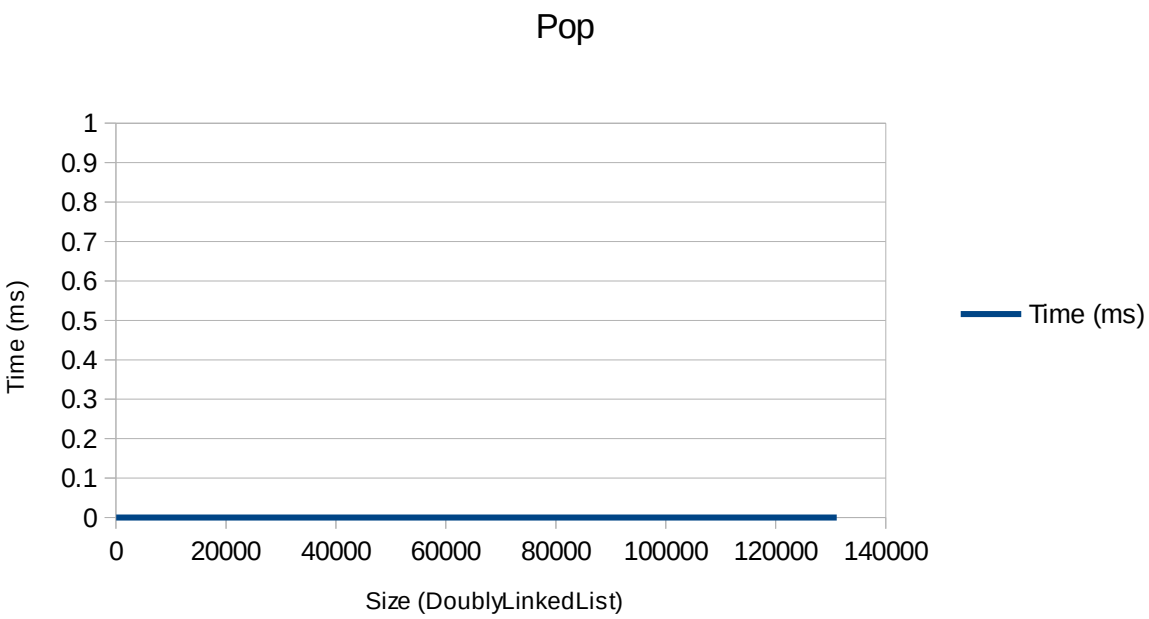
I felt like I was very efficient with my time. It was very easy and fast to code LinkedListStack because the real brains of the class are located in the previously coded DoublyLinkedList. Much of the LinkedListStack's functionality comes from simply calling on methods in DoublyLinkedList.

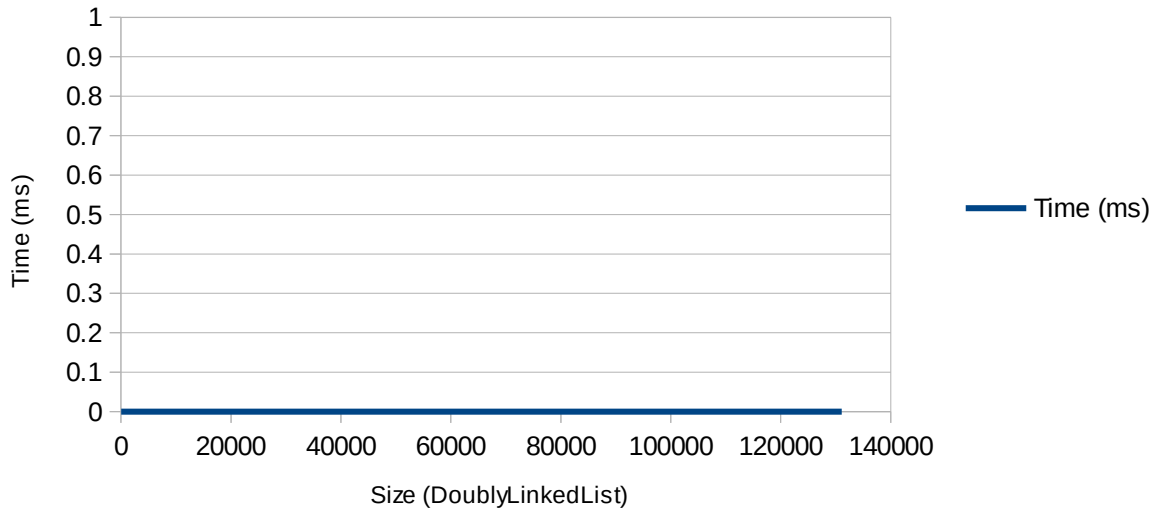
5. Note that the line and column number given by BalancedSymbolChecker indicate the location in a file where an unmatched symbol is detected (i.e., where the closing symbol is expected). Explain how you would also keep track of the line and column number of the unmatched opening symbol. For example, 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.

One way to do this would be to store the line and column number of every symbol that's pushed into the stack. This can be done by creating a LinkedListStack that uses a class that stores these three pieces of data in its nodes. Then when a mismatch occurs print to the screen not only the information about the closing symbol, but the opening one as well.

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.


Push
Time (ms)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
20000
40000
60000
80000
100000
120000
140000
Size (DoublyLinkedList)
Time (ms)


Pop
Time (ms)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
20000
40000
60000
80000
100000
120000
140000
Size (DoublyLinkedList)
Time (ms)



As seen by the three graphs, push, peek, and pop all had the exact same results. All three methods' times did not change at all as the size of the list increased. This indicates a complexity of O(c) (or O(1)) which is exactly what I expected. This is because all three methods will do the exact same thing no matter how many elements there are in the list.

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

About 12