Assignment 1 - Stacks document

By Aleksander Barczak

Matriculation number: 2497555

Introductory Statement

The assignment focuses on the Stack data structure and it’s use in the implementation of a Reverse Polish Notation calculator

Class Design

The classes necessary to be developed will be as follows will be as follows

* ListNode - a singular data entry within the stack structure with referenced by a previous ListNode or Stack head and referencing the following ListNode or null in the case there isn’t one
* Stack - the Stack data structure with methods to push and pop to and from the stack
* Tester - A class to perform basic tests on the Stack data Structure
* Exception classes - Exceptions to make the stack structure handle errors using exceptions. For instance if you were wanting to pop from the stack but it were empty

ListNode -

Fields:

* Double value - the data value held by the node, using the Double to allowing non-integer calculations or results
* ListNode nextNode - a reference to the next node in the stack structure

Methods:

* getValue - returns the data value of the listNode
* getNext - returns the reference to the next node in the data structure
* setNext - takes one argument of type ListNode and sets the reference to the next Node as that argument
* ListNode - initialising function, if no arguments passed data value is set to zero, otherwise value is set to the argument provided

Stack -

Fields:

* ListNode top - references the first data node in the structure

Methods:

* push - takes one argument of type double, creates a ListNode with that argument as the data value and sets it at the top of the data structure
* pop - returns the value of the top ListNode in the structure and removes it from the structure
* isEmpty - returns a boolean value whether or not the stack is emtpy
* Stack - initialising function

PsuedoCode -

No. I’ve got comments in my code, that is as far as my pseudocode goes, this is not worth my time.

TestPlan

|  |  |  |
| --- | --- | --- |
| Test | Expected result | Actual result |
| Entering an empty expression into the calculator | Calculator tells user expression is invalid | As Expected |
| Entering an expressing with too many operands or too few operands | Calculator tells user expression is invalid | As Expected |
| Entering an expression with a division by zero | Calculator responds with infinity | As Expected |
| Entering an expression with text instead of values or symbols | Calculator tells user expression is invalid | As Expected |

Self-Evaluation

The Solution I’ve presented is very robust and handles all user inputs very effectively, it meets all the criteria required by the marking scheme document aside from the use of the List class which I find baffling, it’s not necessary in any way whatsoever, aside from that I’ve gone beyond the expected work completed in my method of producing the infix version of the expression provided by the user, I found that making the Stack use Generics instead of set types allow me to very easily switch from the Doubles I used to allow floating point operations to a String so that the expression would be easy to store as I build it and calculate the answer simultaneously.

I understand that my test plan and pseudocode are lacking but I believe that they are very much unnecessary for this assignment.

Apologies for handing in late.