Lamess Kharfan
Student #: 10150607

CPSC 331 – Assignment 2

1) An algorithm for basic arithmetic in Lisp:
Beginning with an empty stack and the expression you'd like to evaluate, begin by pushing characters onto the stack starting from the right most end of the expression, push characters onto the stack while there are still characters in the expression to be pushed. After each push, check if the character now at the top of the stack is a '(', once a '(' character has been encountered, pop characters off of the stack until a corresponding ')' character has been encountered, appending each character to a base string as they are popped.
Once all characters between the corresponding brackets have been popped from the stack and we have an expression to be evaluated, check what operation needs to be performed by checking index 1 of the expression string. If the operation is addition, call the addition function, if it is subtraction, call the subtraction function with the expression to evaluate it.

In the addition function:

- Remove the brackets surrounding the expression
- Split everything in the expression by the " " (spaces) between them and place them into an array
- For every element in the array:
  - If the element is the operator "+", convert the string into a double, 0, and add it to the result.
  - Otherwise, convert the element into a double, and add it to the result.
- Catch any NumberFormatException or EmptyStackException or IllegalArgumentExceptions
- Return the result to where the addition function was called.

In the subtraction function:

- Remove the brackets surrounding the expression
- Split everything in the expression by the " " (spaces) between them and place them into an array
- If the array is of length 1, then – doesn't have at least one operand, and we have an invalid expression
- If the array is of length 2, then return the negative of the one operand as the result
- Otherwise, For every element in the array:
  - If the element if the subtraction operator, convert it to zero and add it to the result.
  - Otherwise, if weren't at the first operator of the subtraction, convert it to a double and add it to the result. Otherwise, convert the element into a double and subtract it from the current value of the result.
  - Catch any NumberFormatException or EmptyStackException or IllegalArgumentExceptions

- Return the result to where the subtraction function was called.

Now, if all characters in the expression have been pushed to the stack at some point, and the stack is empty, then we have our final result, return it to the calling code.

If every character in the expression has been pushed but and the stack is not empty, the expression was invalid, throw the appropriate exception.

If there are still characters in the expression to push to the stack, push the result of the last expression evaluation back onto the stack, continue to push characters to the stack from the point where we last left off.

This algorithm scans the expression once, pushes and pops each character exactly once, and also catches any invalid input errors.

2) See BoundedStack.java and cpsc331Stack.java
3) See BoundedStackArray.java
4) See LinkedListBoundedStack.java
5) See A2Q5.java
6) See A2Q6.java

PLEASE NOTE: Errors arise during compilation of a couple of these files, however they still run.

Black box and white box tests with input, expected output, and their purpose.

Test	Input	Expected Output	Purpose
Test how the program handles null strings as input	Expression = Null	NullPointerException	Invalid Input
Expression with only addition operator	(+)	0.0	Typical Case
Expression with only multiplication operator	(*)	1.0	Typical Case
Expression with only subtraction operator	(-)	InvalidArgumentException	Invalid Input
Expression with only division operator	(/)	InvalidArgumentException	Invalid Input
Normal addition with arbitrary number of operands	(+ x y z….)	x + y + z	Typical Case
Normal multiplication with arbitrary number of operands	(* x y z…)	x * y * z	Typical Case

Normal division with arbitrary number of operands	(/ x y z)	x / y / z	Typical case
Normal subtraction with arbitrary number of operands	(- x y z…)	x – y – z	Typical Case
Expression with an extra left bracket	((+ x y z)	EmptyStackException	Boundary Case
Expression with extra right bracket	(+ x y z))	InavlidArgumentException	Boundary Case
Expression formatted incorrectly	(+ - x)	NumberFormatException	Invalid Input
Subtraction with 1 operand	(- x)	-x	Typical Case
Division with 1 operand	(/ x)	1/x	Typical Case
Addition with 1 operand	(+ x)	x	Typical Case
Multiplication with 1 operand	(* x)	x	Typical Case
Expression with all operations	(+ (+ x y) (- z w) (* r s) (/q p))	N = result of addition of all inner functions	Typical Case
Expression with negative inputs	(+ -x –y)	-x + -y	Typical Case

Junit Tests in A2Q6.java implement all tests on both the array and linked list implementations of the bounded stack.

The JUnit Test suite A2Q6.java provides thorough testing of the algorithm because it the main types of invalid inputs, which includes extra brackets on either end of the expression, incorrect formatting of the expression, or when a null string for the expression is passed. It demonstrates which operators may be passed on their own into the expression and which may not. It demonstrates what happens when certain operators have only one operand. It also tests that normal expression evaluation works for each of the operators on their own as well as all together. Finally, it demonstrates all the exceptions appropriately thrown by the algorithm when invalid input is given to it. Also, it shows that the program has the ability to handle arbitrary integer values, such as negatives.