Assignment 3

Objectives

- Review command line input and String to Integer conversion
- Practice with Exceptions
- Exposure to prefix notation
- Practice using an ADT to solve a problem

Introduction

This assignment has two parts:

- implement the Stack ADT using a Linked List
- implement a program that uses a stack to evaluate prefix expressions

You've been provided with an array-based stack implementation so you can do either part first.

In this assignment you will implement a program that evaluates expressions written using prefix notation. For example, the result of evaluating: `- 9 7` is `2`
You are likely more familiar with expressions written using infix notation where the same expression would be written as: `9 - 7 = 3`

A lengthier example of a prefix expression…

```
- × ÷ 15 - 7 + 1 1 3 + 2 + 1 1
- × ÷ 15 - 7 2     3 + 2 + 1 1 =
- × ÷ 15 5         3 + 2 + 1 1 =
- × 3              3 + 2 + 1 1 =
- 9                  + 2 + 1 1 =
- 9                  + 2 2     =
- 9                  4         =
5
```

Part I

Create a class called `LLStack` in a file named `LLStack.java`. The class `LLStack` must implement the `Stack` interface specified in `Stack.java` using a linked list structure.

Create an appropriate `Node` class for your linked list implementation in a file named `Node.java`

Modify `StackTester.java` so that it tests your implementation of the `LLStack` class.

Part II

Implement a program in a file called `PrefixCalculator.java` that accepts prefix expressions on the command line and outputs the result of evaluating the expression. Your program must also handle invalid expressions gracefully.

While your StackTester tested your stack with Integers, your PrefixCalculator requires you to push not only operands on the stack but operators. A solution is to have your stack store Strings that will allow you to push both operands ie. "4" and operators ie. "+".
This will require some conversion when you go to perform the calculations. Check out the Integer API – specifically the `parseInt` method.

You must use an algorithm that processes the command line input from left to right such as follows:

```
try
      while there is more input
            value set to next argument of command line arguments
            if  value is an operator
                  push value on the stack
            endif
            else we know value is a operand
                  while (stack not empty AND top of stack is not an operator)
                        pop the poppedvalue from the stack
                        pop the operator from the stack
                        if last popped is not an operator
                              invalid expression
                              STOP evaluation
                        endif

                        apply the operator to value and poppedvalue
                        and store result in value
                        ie. value = poppedvalue OPERATOR value (4 + 5)
                  endwhile
                  push value on the stack
      endwhile
      if one element left on stack
            pop value and display it
      endif
      else
            invalid expression
      endelse
endtry
catch EmptyStack or NumberFormatException
      invalid expression
```

You should break down your PrefixCalculator program into functions. Solutions that have all the code in the main method will lose marks for poor style.

Your calculator only needs to support integer operands.

Your calculator should support the following binary operators:

+	addition
-	subtraction
/	division
x	multiplication (NOTE: this is a lower case x, NOT a *)

Your program MUST take input from the command line arguments and must produce the exact output shown in the table below. In particular, for an invalid expression the output must be:

```
invalid expression
```

If you output "invalid expression" or "INVALID expr" or anything that isn't an exact match to the assignment specification, you will lose marks.
The table below shows some of the test cases we will use and the exact output your program must produce for those inputs.

`java PrefixCalc + 1 2`	`3`
`java PrefixCalc + - 4 8 2`	`-2`
`java PrefixCalc / 10 5`	`2`
`java PrefixCalc / 9 - 7 4`	`3`
`java PrefixCalc x - 5 6 7`	`-7`
`java PrefixCalc x + 4 9 - + 2 5 8`	`-13`
`java PrefixCalc - x / 15 - 7 + 1 1 3 + 2 + 1 1`	`5`
`java PrefixCalc 2 9 +`	`invalid expression`
`java PrefixCalc + - + 1 2 9`	`invalid expression`
`java PrefixCalc words`	`invalid expression`

Submission

Submit only your `LLStack.java`, `Node.java` and `PrefixCalc.java` using conneX. **Please be sure you submit your assignment, not just save a draft**. Submit as many times as you want, your last submission will be the one graded.

A reminder that it is OK to talk about your assignment with your classmates, and you are encouraged to design solutions together, but each student must implement their own solution.

We will be using plagiarism detection software on your assignment submissions.

Grading

If you submit something that does not compile, you will receive a grade of 0 for the assignment. It is your responsibility to make sure you submit the correct files. You MUST implement an algorithm that reads the prefix expression from left to right and pushes both operators and operands on the stack. If you implement an alternative method you will receive 0 on the prefix calculator portion of your assignment.

Requirement	Marks
Passes stack tests	10
Prefix calculator passes test cases	10
Total	**20**