Computer Science 115
Assignment 4
Due March 20th at 11:59pm

Objectives

- Practice with Exceptions
- Understand postfix expressions
- Understand the Stack ADT

Introduction

In this assignment you will implement a program that evaluates postfix expressions. The program will accept the postfix expression on the command line and print the result of evaluating the expression.

One of the easiest ways to evaluate a postfix expression is to use a stack. You've been provided with a specification of the Stack ADT in the file `Stack.java`. You've also been provided with an implementation of the Stack ADT in the file `ArrayStack.java`.

This assignment has two parts: implement a stack using Linked List and implement a program that uses a stack to evaluate postfix expressions. Because you've been given an implementation of the stack using arrays, you can do either part first.

Part 1

You must 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. You will need to create an appropriate `Node` class. Notice that the Stack interface uses Generics.

You've also been provided with a set of test cases for the Stack ADT in the file `StackTester.java.` You will want to modify the tester so that it tests instances of the `LLStack` class, not the `ArrayStack` class.

Part 2

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

Your calculator only needs to support integer operands.

Your calculator should support the following binary operators:

+	addition
-	subtraction
/	division
x	multiplication

Note that we are using the lower case letter x to represent multiplication, not the *. (This is because of how some operating systems handle the * operator on the command line.)

The table below shows some invocations of the program and the expected output. Pay particular attention to the examples using the – and / operators.

Command line	***Output***
`java Calc 5 4 +`	`9`
`java Calc 5 4 -`	`1`
`java Calc 5 4 x`	`20`
`java Calc 10 2 /`	`5`
`java Calc 1 2 3 4 5 + + + +`	`15`
`java Calc 4 +`	`Invalid expression.`
`java Calc 4 5 + 6 3 / x`	`18`
`java Calc 1 2 + + +`	`Invalid expression.`
`java Calc what is this`	`Invalid expression.`

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

In order to gracefully handle invalid expressions, you will need to handle exceptions that are thrown by the Stack ADT and exceptions that are thrown by `Integer.parseInt`.

Here is some pseudo code which describes how you evaluate a postfix expression using a stack:

```
while there is more input
      if  next token is an operand (ie. Number)
            push value on the stack
      if next token is an operator
            pop two values from the stack
            apply the operator to the two values just popped
            push the result of applying the operator on the stack
```

When you are finished, there should be exactly one value on the stack, which is the result of evaluating the expression. If there is more than one value on the stack, the expression is invalid. If at any point you try to pop values from the stack and it is empty, the expression is invalid.

Submission

Submit your completed `LLStack.java` and `Calc.java` via the electronic submission web page.