CS 304 Homework Assignment 2

Due: 11:59pm, Thursday, September 22nd

This assignment is scored out of 66. It consists of 2 programming questions. When you submit, you are required to create a folder with your name (Last name first, then First name), CS304, HW2, e.g., LastName_FirstName_CS304_HW2. Put all your Java programs (`*.java`) as well as output files in the same folder. Zip this folder, and submit it as one file to Desire2Learn. Do not hand in any printouts. Triple check your assignment before you submit. **If you submit multiple times, only your latest version will be graded and its timestamp will be used to determine whether a late penalty should be applied.**

Programming Questions

In the folder that is provided with this homework assignment, there are five files:
`LNode.java`: a class that defines a node in linked lists
`LinkedStack.java`: a linked list based implementation of the **`Stack`** ADT
`PostFixCalculator.java`: a class that implements a postfix calculator using **`LinkedStack`**
`TestLinkedStack.java`: a driver class that tests the implementations of your **`LinkedStack`** and **`PostFixCalculator`** classes
`testExpressions.dat`: a binary file that contains a number of test expressions

P1 (28pts)

a. Completing the `LinkedStack` class

In the **`LinkedStack`** class, you are required to implement the following methods:

`push(int v)` – This method adds an integer to the top of the stack.
`pop()` – This method removes an element from the top of the stack and returns the element. It throws a **`RuntimeException`** "*Pop attempted on an empty stack*" if this operation is attempted on an empty stack.

`size()` – This method returns the number of elements on the stack.

Note that you are only supposed to touch the above three methods. You are NOT allowed to create any other methods, instance variables, or make any changes to methods other than these three methods or files other than "`LinkedStack.java`". Points will be taken off if you fail to follow this rule.

b. Code Testing

You are provided with a test driver implemented by "**`TestLinkedStack.java`**" (**Do not make any changes to this file!**) so there is no need to write your own.

Once you have completed the methods, you can run the test. You should create a plain text file named "**`output-P1.txt`**", copy and paste the output corresponding to **Problem 1** (if your code crashes or does not compile, copy and paste the error messages) to this file and save it.

P2 (38pts) Read book chapter *3.8 Case Study: Postfix Expression Evaluator*.

a. Completing the `PostFixCalculator` class

In the **`PostFixCalculator`** class, you are required to implement the **`calculate()`** method. This method reads in operands as well as operators from the **`m_postfix`** instance variable and evaluates the value of the expression.

Since not all postfix expression are legal, you need to check for invalid expressions, throwing correct **`RuntimeException`** exceptions accordingly. More specifically, you need to throw the following exceptions:

"*Unrecognized operator*" – An operator other than **+**, **-**, ***, **/** is detected.
"*Not enough operands*" – There are fewer operands than expected.
"*Too many operands*" – There are more operands than expected.

You may refer to the expected results printed by the test driver for exception formats and messages (e.g., "*Not enough operands*"). Your exception messages must be exactly the same as the expected ones to pass all tests.

Note that you are only supposed to touch the `calculate()` method. You are NOT allowed to create any other methods, instance variables, or make any changes to methods other than these three methods or files other than "`PostFixCalculator.java`". Points will be taken off if you fail to follow this rule.

b. Code Testing

In "**`TestLinkedStack.java`**" (**Again: do not make any changes to this file!**), a test driver for **`PostFixCalculator`** is provided. You are also given a data file "**`testExpressions.dat`**" that contains 33 test dates.

Depending on your programming environment, the data file might need to be placed in different folders so that you test driver can read it. For jGRASP, you can leave the data file in the same folder as your java files. For NetBeans, you should place it in your project folder in which you see directories like **`build`**, **`nbproject`**, and **`src`**, etc.

Once you have completed the **`calculate()`** method, you can run the test. You should create a plain text file named "**`output-P2.txt`**", copy and paste the output corresponding to **Problem 2** (if your code crashes or does not compile, copy and paste the error messages) to this file and save it.

Grading Rubrics:

Code does not compile: -10

P1.

Code compiles but crashes when executed: -5
Changes were made to things other than methods **`push`**, **`pop`**, and **`size`**: -5
Has output file: 5
Code passes 23 test cases: 23 (each test case worth 1 point)

P2.

Code compiles but crashes when executed: -5
Changes were made to things other than the **`calculate`** method: -5
Has output file: 5
Code passes 33 test cases: 33 (each test case worth 1 point)

Sample output for P1:

```
================ Problem 1 ================
Test 1: size() ==> [Passed]
 Expected: 0
 Yours: 0

Test 2: pop() ==> [Passed]
 Expected: a RuntimeException
 Yours: RuntimeException: "Pop attempted on an empty stack"

Test 3: push(10) and then isEmpty() ==> [Passed]
 Expected: false
 Yours: false

Test 4: toString() ==> [Passed]
 Expected (from top to bottom): 10
 Yours (from top to bottom): 10

Test 5: top() ==> [Passed]
 Expected: 10
 Yours: 10

Test 6: push(20) and then toString() ==> [Passed]
 Expected (from top to bottom): 20 10
 Yours (from top to bottom): 20 10

Test 7: top() ==> [Passed]
 Expected: 20
 Yours: 20
```

```
...

Test 23: pop() and then size() ==> [Passed]
 Expected: 0
 Yours: 0

Total test cases: 23
Correct: 23
Wrong: 0
================ End of Problem 1 ================
```

Sample output for P2:

```
================ Problem 2 ================
Test 1: (187) ==> [Passed]
 Expected: 187
 Yours: 187

Test 2: (+) ==> [Passed]
 Expected: RuntimeException: "Not enough operands"
 Yours: RuntimeException: "Not enough operands"

Test 3: (45 +) ==> [Passed]
 Expected: RuntimeException: "Not enough operands"
 Yours: RuntimeException: "Not enough operands"

Test 4: (4  5   +) ==> [Passed]
 Expected: 9
 Yours: 9

Test 5: (4  5+) ==> [Passed]
 Expected: RuntimeException: "Not enough operands"
 Yours: RuntimeException: "Not enough operands"

Test 6: (4  5  6   +) ==> [Passed]
 Expected: RuntimeException: "Too many operands"
 Yours: RuntimeException: "Too many operands"

...

Test 33: (45 5 3 4 + 8 8 * 4 2 / 1 + + - * +) ==> [Passed]
 Expected: -255
 Yours: -255

Total test cases: 33
Correct: 33
Wrong: 0
================ End of Problem 2 ================
```