Joanna Klukowska
joannakl@cs.nyu.edu

Programming Project 4: Calculator

Due date: November 18, 11:55PM EST.

You may discuss any of the assignments with your classmates and tutors (or anyone else) but all work for all assignments must be entirely your own. Any sharing or copying of assignments will be considered cheating. If you get significant help from anyone, you should acknowledge it in your submission (and your grade will be proportional to the part that you completed on your own). You are responsible for every line in your program: you need to know what it does and why. You should not use any data structures or features of Java that have not been covered in class (or the prerequisite class). If you have doubts whether or not you are allowed to use certain structures, just ask your instructor.

In this project you will work on a program that is a backend for a very simple calcular. Your calculator takes as input simple mathematical expressions like

```
23 - 2 * 5
4 - 9 * 11 + 155 * ( 21 - 17 )/ 3
( ( 15 / ( 7 - ( 1 + 1 )))- ( 2 + ( 1 + 1 ))
```

and determines their value.

The Program Input and Output

Input File Your program is given the name of the input text file on its command line (the first argument). The text file contains mathematical expressions (like the ones above). There is always only one expression per line. Each valid expression consists of tokens separated by one or more spaces. A token is either a positive integer, an operator (`+`, `-`, `*`, `/`), or an open or a closed parenthesis. The input file may contain empty lines.

If the filename is omitted from the command line, it is an error. The program should display an error message and terminate. The error message should indicate what went wrong (for example: "Error: missing name of the input file").

If the filename is given but the file does not exist or cannot be opened for reading by the program, for any reason, it is an error. The program should display an error message and terminate. The error message should indicate what went wrong (for example: "Error: file `calculations.txt` does not exist.", but make sure to replace the name with the name of the file with which the program was called).

Your program is NOT ALLOWED to hardcode the input filename in its own code. It is up to the user of the program to specify the name of the input file. Your program should not modify the name of the user-specified file (do not append anything to the name).

Your program may not read the input file more than once.

Your program may not modify the input file.

Output File Your program is given the name of the output text file on its command line (the second argument). As your program evaluates each of the expressions in the input file, the results of computations should be printed to the output file. If a corresponding expression in the input file is not valid or contain a blank line, your program should print INVALID instead of the result and continue with the next expression. The results should be printed one per line.

If the filename is omitted from the command line, it is an error. The program should display an error message and terminate. The error message should indicate what went wrong (for example: "Error: missing name of the output file").

If the filename is given but the file with a provided name cannot be created for any reason, it is an error. The program should display an error message and terminate. The error message should indicate what went wrong (for example: "Error: file `results.txt` could not be created.", but make sure to replace the name with the name of the file with which the program was called).

If the file with a given filename already exists, the program should overwrite it. If the file with a given filename does not exist, your program should create it.

Joanna Klukowska
joannakl@cs.nyu.edu

Your program is NOT ALLOWED to hardcode the input filename in its own code. It is up to the user of the program to specify the name of the input file. Your program should not modify the name of the user-specified file (do not append anything to the name).

User Input This program is not interactive.

Program Design

Your program must contain multiple classes (in multiple files) and must be developed in an object oriented way. You will need to develop the following classes:

- `Calculator` - the class that provides the `main()` method. This class is responsible for all input and output operations (the input file should be read by this class, and the class should write the results to the output file).
- `ExpressionTools` - the class that provides the methods for infix to postfix conversion and for postfix evaluation. This class provides only tools (like the `Math` class, so you should never need to create an instance of this class and all of its methods should be `static`).
- `MyStack` - the class that provides a reference based implementation of the interface provided below (you will not get any credit for this class if it is implemented using arrays or if it uses the `Stack` implementation from Java API).
- `PostFixException` - the class that represent the exception that should be thrown when errors in the expression occur.

The program should read each of the expressions from the input file, convert it to the postfix expression, evaluate the postfix expression and print the result to the output file .

Converting From Infix to Postfix

The algorithm for converting the infix expressions to postfix expressions is as follows.

`Algorithm: Conversion from Infix to Postfix Expressions`

```
for each token in the input infix string expression
  if the token is an operand
    append to postfix string expression
  else if the token is a left brace
    push it onto the operator stack
  else if the token is an operator
    if the operator stack is not empty
      while top element on the stack has precedence higher or equal
        pop the stack and append to postfix string expression
    push it (the current operator) onto the operator stack
  else if the token is a right brace
    while the operator stack is not empty
      if the top of the operator stack is not a matching left brace
        pop the operator stack and append to postfix string expression
      else
        pop the left brace and discard
        break
while the operator stack is not empty
  pop the operator stack and append to postfix string expression
```

Note that you will need to build error detection into this algorithm.

If any errors are encountered, your method should throw an exception of type `PostFixException` with an appropriate message. You should use the following definition of the exception class:

Joanna Klukowska
joannakl@cs.nyu.edu

```
1
2 public class PostFixException extends Exception {
3
4   public PostFixException() {
5     super();
6   }
7
8   public PostFixException(String message) {
9     super(message);
10  }
11 }
12
```

Evaluating the Expressions

The algorithm for evaluating the postfix expressions is as follows.

Algorithm: Evaluation of Postfix Expressions

```
scan the given postfix expression from left to right
for each token in the input postfix expression
  if the token is an operand
    push it (its value) onto a stack
  else if the token is an operator
    operand2 = pop stack
    operand1 = pop stack
    compute operand1 operator operand2
    push result onto stack
return top of the stack as result
```

Note that you will need to build error detection into this algorithm. If any errors are encountered, your method should throw an exception of type `PostFixException` with an appropriate message.

`MyStack` class

Both of the above algorithms need to use a stack. You should implement your own reference based stack that follows the following informal interface:

`Stack()` Creates an empty Stack.

`boolean empty()` Tests if this stack is empty.

`E peek()` Looks at the object at the top of this stack without removing it from the stack.

`E pop()` Removes the object at the top of this stack and returns that object as the value of this function.

`E push(E item)` Pushes an item onto the top of this stack.

`int search(Object o)` Returns the 1-based position where an object is on this stack.

For debugging purposes, you may also wish to implement a **`toString`** method so you can easily see what is on the stack. For details of the above methods see http://docs.oracle.com/javase/8/docs/api/java/util/Stack.html. Your implementation should follow the ideas that we discussed in class (using a **`head`** or **`top`** reference that points to the top of the stack and single linking between the nodes).

Joanna Klukowska
joannakl@cs.nyu.edu

Grading

If your program does not compile or crashes (almost) every time it is ran, you will get a zero on the assignment.

20 points design and implementation of the `Calculator` class (this includes validation the command line arguments, reading the input file, writing to the output file)

30 points design and implementation of the `MyStack` class

30 points design and implementation of the `ExpressionTools` class

20 points proper documentation

How and What to Submit

Your should submit all your source code files (the ones with .java extensions only) in a single **zip** file to NYU Classes. If you are unsure how to create a zip file or how to get to your source code files, you should ask long before the project is due.

If you wish to use your (one and only) freebie for this project (one week extension, no questions asked), then complete the form at `http://goo.gl/forms/YFDVBlscEB` **before the due date for the assignment**. All freebies are due seven days after the original due date and should be submitted to NYU Classes.