CSC 220 Data Structures

Program #5

Infix and Postfix

Description:

Your objective is to take an infix expression from the user, convert it into postfix form, then evaluate it to find an answer to the expression.

You will need your generic Stack, generic Queue, generic List and generic Node class for this assignment. The reason why you will need generic classes, is that in some cases you will need to use a Stack of Character objects and in other cases you will need a Stack of Integer objects.

Create a file called "Main.java" that would contain a public class called "Main". All work will be done inside this Main class. Here are the methods you must implement inside this class:

- `Entry point`
  - This should prompt the user for an infix expression, send that expression to the infixToPostfix method (described below), this will return a postfix expression (as a string), you will then send that to evalPostfix (described below) for evaluation, and then finally print out the summary shown in the sample output.

- `int getInfixPriority(char c)`
  - This takes in a char (a single character) and determines the infix priority as described in the notes. It then returns this priority.

- `int getStackPriority(char c)`
  - This takes in a char (a single character) and determines the stack priority as described in the notes. It then returns this priority.

- `boolean isOperand(char c)`
  - This will return true if c is an operand (basically a single digit from 0 to 9 inclusively). It will return false otherwise. To keep things simple, **only single digit numbers are allowed** in the input specified by the user. This means both the infix and the postfix notation will contain only single digit numbers. As you evaluate the expression, multi-digit numbers will naturally arise, and that's ok.

- `int eval(char operator, int a, int b)`
  - The `operator` character is one of the following operators: `+, -, *, /, or ^`. These are the only operators allowed (except for parentheses which is not handled by this method). Simply return a value of -1 if the operator is invalid.
  - `a` and `b` are your operands. The order goes, `a operator b`.
  - This method returns the result of applying the given operator, `operator`, to the operands, `a` and `b`

- `String infixToPostfix(String infixString)`
    - This method takes in an expression in infix form and produces the equivalent expression in postfix form. Here is where your stack and queue implementations will come into play. You must implement the algorithm from the notes. I don't want you grabbing some code from some online source and sticking it in here.

- `int evalPostfix(String postfixString)`
    - This takes an expression in postfix form and evaluates it, returning the evaluated result. All numbers are restricted to a single digit, no floating point numbers allowed, division symbols are handled as integer division, the final answer along with any intermediate answers are allowed to be multiple digits (i.e. it should return the mathematically correct answer, using integer division for any division).

No input validation is needed here. Assume the user will only enter single digit numbers and no floats. Also assume the user will only enter valid operators along with parentheses (if desired). No spaces are allowed in the input (this is so parsing the input is made a little easier), so you can safely assume no spaces are in the input. The valid operators you must handle are: + (addition), - (subtraction), / (integer division), * (multiplication), and ^ (exponentiation).

Here are some sample outputs. Your output and prompts should look **exactly** like this, obviously with respect to whatever the user enters.

```
$ javac Main.java
$ java Main
Enter infix expression: 2-3
Summary
-------
  Infix: 2-3
Postfix: 23-
 Result: -1

$ java Main
Enter infix expression: 5+2*9
Summary
-------
  Infix: 5+2*9
Postfix: 529*+
 Result: 23

$ java Main
Enter infix expression: (5^2)/9
Summary
-------
  Infix: (5^2)/9
Postfix: 52^9/
 Result: 2
```

About the demo .jar files:

The "demo.jar" file is for you to use. Run it in Git Bash with the following command:

```
java -jar demo.jar
```

Download Git Bash at: https://git-scm.com/downloads

This demo is provided to help in your understanding of the process of converting infix to postfix and evaluating postfix.

BONUS:

If you can add a conversion from postfix back into infix and then show the resulting infix form, I will add an **additional 5 points** to your assignment. To be clear, this means implementing the `postfixToInfix` method and calling it from main. Here are the details:

- `String postfixToInfix(String postfixString)`
  - This method takes in an expression in postfix form and produces the equivalent expression in infix form.

Here are some sample outputs with the bonus implementation. Your output for the infix form (converted back from postfix) may look a little different, but as long as it is correct (order of operations are correct as was originally specified by the user), then you are fine. Feel free to use extra parentheses as needed.

```
$ javac Main.java
$ java Main
Enter infix expression: 2-3
Summary
-------
  Infix: 2-3
Postfix: 23-
Back to Infix: (2-3)
 Result: -1

$ java Main
Enter infix expression: 5+2*9
Summary
-------
  Infix: 5+2*9
Postfix: 529*+
Back to Infix: (5+(2*9))
 Result: 23

$ java Main
Enter infix expression: (5^2)/9
Summary
-------
  Infix: (5^2)/9
Postfix: 52^9/
Back to Infix: ((5^2)/9)
 Result: 2
```

This is going to be more difficult since I haven't provided the pseudocode to accomplish this. Your stack and/or queue will be required for proper execution.

Submitting your assignment:

Submit only these files: **Stack.java, Queue.java, List.java** and **Main.java**. Do not submit any .class files. Zip up the .java files and submit the zip.

Rubric:

#	ITEM	POINTS
1	method: getInfixPriority	2
2	method: getStackPriority	2
3	method: isOperand	2
4	method: eval	3
5	method: infixToPostfix	16
6	method: evalPostfix	10
7	matches output	5
8	Bonus method: postfixToInfix	+5
	TOTAL	**40 (+5)**

#	PENALTIES	POINTS
1	Doesn't compile	-50%
2	Doesn't execute once compiled (i.e. it crashes)	-25%
3	Late up to 1 day	-25%
4	Late up to 2 days	-50%
5	Late after 2 days	-100%