Seoul National University

Data Structure

Fall 2019, U Kang

Programming Assignment 1: Lists, Stacks and Queues (Chapter 4)

Due: October 09, 11:59 pm, submit at eTL

Reminders

- The points of this homework add up to 100.
- Like all homework, this has to be done individually.
- Lead T.A.: Huiwen Xu (xuhuiwen33@snu.ac.kr)
- Write a program in Java.
- Do not use Java Collection Framework and the third-party implementation from the Internet.

1. How to submit the programming assignment

1) Create a ***JAR*** file including 'src' folder that contains your sources files. (Refer to '1 – Introduction.pptx' in the first lab session)
   - We will run your ***Main*** class in the JAR file to grade your programming assignments. Before submitting the JAR file, please check if your Main class in the JAR file works correctly.
   - You **MUST** obey the I/O specification of the programming assignment, and rules for the submission of the programming assignment.
   - Before submitting, check if your JAR file runs properly in your terminal with the following commands:

     ```
     java -classpath PA_XX_(studentid).jar com.dmlab.Main
     ```

     (e.g. java -classpath PA_01_201912345.jar com.dmlab.Main)

2) Submit the jar file to the eTL (http://etl.snu.ac.kr/) .

2. How to grade your programming assignment

1) We made a grading machine to automatically grade your programming assignment. The machine will run your program, and compare answers and outputs that your program generates for given inputs. If your program cannot generate correct answers for an input file, it will not give you the point corresponding to the input. Our machine will consider the following scenarios:
   - (***Accept***) When your program generates exact outputs for an input file, the machine will give you the point of the input.
   - (***Wrong Answer***) When your program runs normally, but generates incorrect outputs for an input file, including typos, the machine will not give you the point of the input.
   - (***Run Error***) When your program does not run, or is terminated suddenly for some reasons, the machine will not give you the point of an input file because it cannot generate any outputs.
   - (***Time Limit***) When your program runs over a predefined execution time for an input file, our machine will stop your program, and it will not give you the point of the input. The time limit of the execution is ***5 seconds***.

2) We will generate 10 input files, and assign 10 points for each input file. For example, if your program gets 9 accepts, and 1 wrong answer by the machine, the total point will be 90 points. Hence, before submitting your programming assignment, please be sure that your program makes correct answers in reasonable time for any input case.

3. Problem

Stack and **Queue** are simple but very powerful data structure when you make applications.

Stack has two principle operations: *push* and *pop*. With these operations, the order of element come off a stack is also called LIFO (Last In, First Out).

- *Push*: Adds an element to the collection
- *Pop*: Removes and returns the most recently added element

In case of Queue which has two principle operations: *add* and *poll*, the order of element come off a queue is also called FIFO (First In, First out).

- Add: Adds an element to the collection
- *Poll*: Removes and returns the oldest element in the collection

Stack Calculator is one of most famous application of the stack that handles the parenthesis in the arithmetic equation to calculate numbers with proper orders. To understand the mechanism of the stack calculator, we need to know about infix and postfix form of the equation. In the infix form, we place the operand in the middle of numbers to be calculated, e.g. (1 + 2). But the postfix form of the equation places the operand at the tail, e.g. (1 2 +). Our stack calculator will accept the input as the infix form of the equation including parenthesis and then transform it to the postfix form. When transforming, we will consider the order of calculation with regards to the parenthesis. So, any parenthesis is not included in the postfix form of the equation. Finally, the postfix form of the equation will be calculated using the stack. Specific algorithms for the transforming infix to postfix and calculation of postfix form is provided below.

To make problem simpler, we allow only +, – , *, / as operations and (,) as parenthesis.

- Transform the infix into the postfix

```
Prepare a stack and a queue
For each item in the infix form
        If item is operator +, -, *, / then
                while “stack is not empty” and “top of stack is not (“
                        if “Top of stack” ’s priority is greater than or same as item’s
```

```
                                    "Popped item" <- pop from the stack
                                    Add "Popped item" to the queue
                            else
                                    break
                    Push the item to the stack
            Else if item is parenthesis ( then
                    Push the item to the stack
            Else if item is parenthesis ) then
                    while "stack is not empty" and "top of stack is not ("
                            "Popped item" <- pop from the stack
                            Add "Popped item" to the queue
                    Pop an item from the stack if (
            Else if item is number then
                    Add the item to queue
    while "stack is not empty"
            "Popped item" <- pop from the stack
            Add "Popped item" to the queue
```

For example, the infix form of the equation "2/2+(3-4)*5" should be transformed like below

```
2/2+(3-4)*5 : stack[], queue[]
/2+(3-4)*5 : stack[], queue[2]
2+(3-4)*5 : stack[/], queue[2]
+(3-4)*5 : stack[/], queue[2,2]
(3-4)*5 : stack[+], queue[2,2,/]
3-4)*5 : stack[+,(], queue[2,2,/]
-4)*5 : stack[+,(], queue[2,2,/,3]
4)*5 : stack[+,(,-], queue[2,2,/,3]
)*5 : stack[+,(,-], queue[2,2,/,3,4]
*5 : stack[+], queue[2,2,/,3,4,-]
5 : stack[+,*], queue[2,2,/,3,4,-]
 : stack[+,*], queue[2,2,/,3,4,-,5]
 : stack[+], queue[2,2,/,3,4,-,5,*]
 : stack[], queue[2,2,/,3,4,-,5,*,+]
```

The postfix form of this equation is "2 2 / 3 4 – 5 * + "

- Calculate the postfix form of an equation

```
Prepare a stack
Prepare a queue with postfix form generated with the previous algorithm
Foreach polled item from the queue
        If the item is operator +, -, *, / then
                Pop two items from the stack
                Do the calculation and push back to the stack
        Else if the item is number then
                Push the item to the stack
Result of calculation should be on the top of stack
```

For example, the postfix form of the equation "2 2 / 3 4 – 5 * +" should be calculated like below.

```
queue[2,2,/,3,4,-,5,*,+] : stack []
queue[2,/,3,4,-,5,*,+] : stack [2]
queue[/,3,4,-,5,*,+] : stack [2,2]
queue[3,4,-,5,*,+] : stack [1]
queue[4,-,5,*,+] : stack [1,3]
queue[-,5,*,+] : stack [1,3,4]
queue[5,*,+] : stack [1,-1]
queue[*,+] : stack [1,-1,5]
queue[+] : stack [1,-5]
queue[] : stack [-4]
```

The result of this equation is "-4".

4. Specification

1) Stack

You can implement as limited sized array-based stack. When test your result, TA will not push over 128 items to the stack at the same time. But you should think of that TA can do the more pushes after pops. (e.g. 100 pushes -> 90 pops -> 100 pushes)

2) Queue

Like a stack, you can implement as limited sized array-based queue. When test your result, TA will not add over 128 items to the queue at the same time. But you should think of that TA can do the more adds after polls. (e.g.

100 adds -> 90 polls -> 100 adds) You may need to consider implementing circular iteration because of this condition.

3) Stack Calculator

We will test your project with only proper infix inputs. The proper means,

- There are only non-negative integers as number
- All of the items in the infix form will be separated by the one white-space. That means you can split it with " ". No need to consider exceptional cases.
- All pair of parentheses will be closed properly. After (appears, there should be a pair of) in the equation. Moreover, This will be no) before (appears. No need to consider the case of parentheses mismatch.

You need to fill the blanks of functions in the MyQueue.java, MyStack.java and StackCalculator.java from the skeleton codes. Check the interface files (Queue.java and Stack.java) to see the description of functions should be implemented.

5. Specification of I/O

We will provide the Main.java file that already implemented to fit the I/O format of this assignment. **You should not modify Main.java file and must not add additional prints to the stdout**. We will collect the prints of stdout from your submitted jar and compare with the current answer automatically. Only the exact match will give you a score.

6. Sample Input and Output

Sample Input	Sample Output
stack,clear stack,empty stack,push,1 stack,push,2 stack,push,3 stack,empty stack,peek	true false 3 3 2 2 1

stack,pop stack,peek stack,pop stack,pop stack,empty	true

Sample Input	Sample Output
queue,clear queue,empty queue,add,1 queue,add,2 queue,add,3 queue,empty queue,peek queue,poll queue,peek queue,poll queue,peek queue,poll queue,empty	true false 1 1 2 2 3 3 true

Sample Input	Sample Output
calc,1 + 2	3
calc,12 * 5 + 10 - (20 - 52) / 2	86
calc,(6 + 5) * 11 - 100 + 3	24
calc,7 + 8 + 8 - 5 * 3 / 5 - 4	16
calc,(10 / 2) * 3 + 4 * (9 - 6)	27