CSC1016S Assignment 9: Stacks and Queues

Assignment Instructions

This assignment concerns implementing ADTs using linked data structures and using queues/stacks to solve programming problems.

Question 1 [25 marks]

Queues are used extensively in situations where the input or output has to be delayed because of, for example, slow Internet speeds - these queues are called I/O buffers.

Write a program called `Question1.java` to simulate the operation of an I/O buffer using a linked queue data structure. You may use the built-in Java data structures.

Your program must accept a sequence of lines of text from the keyboard and process them as follows:

- If the first character is O (Oh), extract the next string from the buffer and output it. If there is no data in the queue, output "Buffer empty"
- If the first character is X, exit.
- Otherwise, store the string in the buffer.

Sample Input/Output

```
line1
line2
O
Data: line1
line3
line4
O
Data: line2
O
Data: line3
line5
O
Data: line4
O
Data: line5
O
Buffer empty
X
```

Question 2 [35 marks]

Write a program to identify mismatched brackets. This happens when there are too many opening brackets ('{', '[', '(' and '<') and not enough closing brackets ('}', ']', ')' and '>'), or vice versa; or when an opening bracket of one type corresponds to a closing bracket of another type, for example if a '(' is closed by a ']'.

CONTINUED

Write a program called `Question2.java` that takes a string as input. It should then scan the string, checking that all brackets match. If it encounters an opening bracket that does not match its corresponding closing bracket, it should print a message to that effect and then terminate any further checking as further errors will be ambiguous. If, at the end of the scan, there remain unclosed brackets or excess closing brackets, a message should be printed to that effect.

Hint: this problem is best solved using a stack, by popping brackets from the end off and storing them in another stack until an opening bracket is found. You may write your own stack implementation, or use the built-in java.util.Stack class.

It may be helpful to write other methods such as ones that check whether a bracket is an opening or closing bracket and ones that return the corresponding bracket to a given bracket. However, these are not essential.

Sample Input/Output

```
Enter a string to test:
( < [ { } ( { > ) ] >
error: '>' does not match with '{'.

Enter a string to test:
{ ( [ ] ) { ( ) ( ) } }
The string is correct! There are no mismatched brackets.

Enter a string to test:
{ ( [ ] ) { ( ) ( ) }
error at end: opening bracket '{' remains unclosed.

Enter a string to test:
( [ ] ) < > ] }
error at end: the close bracket ']' does not have a corresponding opening
bracket.
error at end: the close bracket '}' does not have a corresponding opening
bracket.
```

Question 3 [40 marks]

Reverse Polish Notation is an alternative way of writing arithmetic expressions where the operator is supplied after the 2 arguments. For example, "3 2 +" is equivalent to the traditional "3+2" and "3 2 * 1 +" is equivalent to the traditional "(3*2)+1". See Wikipedia for more details and examples.

Write a program called `Question3.java` to calculate the value of an integer expression in Reverse Polish Notation using a linked stack data structure to store intermediate values. You may use the built-in Java data structures.

The basic algorithm is to scan over the expression (a space-separated list of symbols) from left to right and:

- when an integer is encountered, push it onto the stack;

- when an operator is encountered, pop 2 integers off the stack, perform the operation and push the result back onto the stack; and
- when no more symbols are left, pop the answer off the stack.

The operators to be supported are +/*-. Note that "4 2 -" evaluates to 2 and "4 2 /" evaluates to 2.

Your program must also cater for the following errors:

- Insufficient arguments for <operator>, where less than 2 arguments are provided.
  For example, "3 +".
- Integer expected but not found, where a value is not an integer.
  For example, "3 f +".
- Extra symbols in expression, where the expression has been fully processed but more than one value remains on the stack.
  For example, "3 2 + 4".
- Insufficient symbols in expression, where the expression has been fully processed but there is no answer left on the stack.
  For example, "".

Sample Input/Output

```
3 3 * 4 2 - /
Answer: 4
Sample I/O
3 2 + *
Insufficient arguments for *
Sample I/O
Insufficient symbols in expression
Sample I/O
3 f +
Integer expected but not found
Sample I/O
3 2
Extra symbols in expression
```

Marking and Submission

Submit `Question1.java`, `Question2.java`, and `Question3.java` in a single .ZIP folder to the automatic marker.

The zipped folder should have the following naming convention:

yourstudentnumber.zip

END

CONTINUED