Assignment Instructions

This assignment is about the Queue and Stack data structures.

Question 0 [5]

A Queue may be considered as a linked list in which node insertion is done at the back of the list and deletion is done in front (analogous to the queue at a super market). Write a program that creates a queue, add elements to the queue and delete elements from the queue.

Sample I/O:

```
MENU: Add node(A), Delete node(D), Display(L), Exit(^z)
a
Enter something or ^z to return to main menu:
one
Enter something or ^z to return to main menu:
two
Enter something or ^z to return to main menu:
three
Enter something or ^z to return to main menu:
four
Enter something or ^z to return to main menu:
^z
MENU: Add node(A), Delete node(D), Display(L), Exit(^z)
l
one - two - three - four
MENU: Add node(A), Delete node(D), Display(L), Exit(^z)
d
Deletion done!
MENU: Add node(A), Delete node(D), Display(L), Exit(^z)
d
Deletion done!
MENU: Add node(A), Delete node(D), Display(L), Exit(^z)
d
Deletion done!
MENU: Add node(A), Delete node(D), Display(L), Exit(^z)
l
four
MENU: Add node(A), Delete node(D), Display(L), Exit(^z)
d
Deletion done!
MENU: Add node(A), Delete node(D), Display(L), Exit(^z)
d
Queue is empty
MENU: Add node(A), Delete node(D), Display(L), Exit(^z)
^z
Bye bye
```

Save and submit your solution as *Question0.java*

Question 1 [5]

A stack may be thought of as a linked list in which insertion/deletion is performed at the “top” of the list (similar to the way a stack of plates works, the last plate to go in is the first one to come out, conventionally). Write a menu driven program that adds elements to the stack and deletes elements from the stack. The program should printout *Stack is empty* when the user tries to delete from or display a Stack with no elements in it.

Sample I/O:

```
MENU: Add node(A), Delete node(D), Display(L), Exit(^z)
a
Enter something or ^z to return to main menu:
blue
Enter something or ^z to return to main menu:
green
Enter something or ^z to return to main menu:
red
Enter something or ^z to return to main menu:
^z
MENU: Add node(A), Delete node(D), Display(L), Exit(^z)
L
red
green
blue

MENU: Add node(A), Delete node(D), Display(L), Exit(^z)
D
Deletion done!
MENU: Add node(A), Delete node(D), Display(L), Exit(^z)
L
green
blue

MENU: Add node(A), Delete node(D), Display(L), Exit(^z)
A
Enter something or ^z to return to main menu:
purple
Enter something or ^z to return to main menu:
^z
MENU: Add node(A), Delete node(D), Display(L), Exit(^z)
l
purple
green
blue

MENU: Add node(A), Delete node(D), Display(L), Exit(^z)
^z
Bye bye
```

Save and submit your solution as *Question1.java*

Question 2 [20]

Queues are used extensively in situations where the input or output has to be delayed because of reasons such as limitations in data transfer rates or processing capabilities. These queues are called I/O buffers.

Write a program to simulate the operation of a first in first out I/O buffer using a linked queue data structure. You may use the built-in Java data structures.

Your program must allow the user to continually enter lines of text until they enter a line starting with the letter X. While the user enters the lines of text, process the input as follows:

- If the first character is O (Oh), extract the next string from the buffer and print it out prefixed with "*Data:*". If all the data has been extracted from the queue or 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 I/O

```
packet1
packet2
packet3
o
Data: packet1
o
Data: packet2
packet4
o
Data: packet3
o
Data: packet4
o
Buffer empty
x
```

Save and submit your solution as *Question2.java*

Question 3 [30]

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 ']'.

Your program must take 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 I/O:

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

Sample I/O:

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

Sample I/O:

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

Sample I/O:

```
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.
```

Save and submit your solution as *Question3.java*

Question 4 [40]

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 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 I/O

```
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
```

Important

Submit your solutions to question 0 through question 4 named as *Question0.java, Question1.java, Question2.java, Question3.java and Question4.java* respectively. Remember to sufficiently comment all your code.