COMP-10205 – Data Structure and Algorithms
Lab # 5 – Stacks and Queues

5% of course grade

Submission Requirements

Complete the following exercise and submit electronically in the dropbox on eLearn. Please refer the course Calendar for the exact date and time of the submission. This lab must be done individually.

Background

You are to write a program that will act as a simple calculator that supports addition, subtraction, multiplication and division operators. The program operates by taking in a pair of parameters (operation and number) that are separated by a space. For example a valid entry could consist of a + followed by 4.4. This would evaluate to 4.4. If the operation and value pair "* 2" in now entered the result would evaluate to 8.8.
The program will store all of the operations on a Queue. In addition to storing the operator and operand on the Queue, you must also be able to store Undo and Redo operations. These two operations will be represented by the letter 'U' for undo and the letter 'R' for redo. To evaluate all of the operations on the Queue the letter 'E' will be used. Once evaluated the Queue will be empty and ready to start a new calculation. To exit the program the letter 'X' will be used. On exit the contents of the Queue should be evaluated one last time.
This lab will require two classes. The first class will be used to store operation/value pairs. You will need to store the operation (use a char) and the value (use a double for the value)
The second class (Lab5) will have a main method and other methods to evaluate the contents of the queue. It will also require the use of one Stack for Undo operations and one stack for Redo operations. The stack and Queue used must be the one included below:

MQueue.java

```
import java.util.LinkedList;

public class MQueue<E>  {

    private LinkedList<E> q = new LinkedList<>();

    public void enqueue(E e) {
        q.addLast(e);
    }

    public E dequeue() {
        return q.removeFirst();
    }

    public E peek() {
        return q.getFirst();
    }

    public int size() {
        return q.size();
    }

    public boolean isEmpty() {
        return q.isEmpty();
    }
}
```

MStack.java

```
import java.util.LinkedList;

public class MStack<E> {

    private LinkedList<E> s = new LinkedList<>();

    public void push(E e) {
        s.addLast(e);
    }

    public E pop() {
        return s.removeLast();
    }

    public E peek() {
        return s.getLast();
    }

    public int size() {
        return s.size();
    }

    public boolean isEmpty() {
        return s.isEmpty();
    }
}
```

Suggested Steps:

1. Create a new project in NetBeans.
2. Add the code above to two new class files called MStack.java and MQueue.java for the stack and queue.
3. Add a new class to the project to hold the mathematical operation and value(ie. + 5). You will need a constructor to initiate the two read-only members of the class and accessors for two properties.
4. Override the toString method to display the contents of the object neatly (see output below for an example).
5. You may add other members to the class if required, but the two properties above must be declared final so that they cannot be modified after construction.
6. Inside the main class you will need to create an MQueue instance to store to store all of the calculator operations. It is suggested that you validate the information before adding to the Queue.
7. Evaluate the data stored on the Queue by applying the operations to the numbers.
8. Once your program can handle standard operations (+,-,*,/) off of the Queue you will need to add processing for the undo and redo operations. To do this you will need to track each successful operation in the Queue and save it to the Undo Stack. On an Undo command you will need to undo the operation to the current value (do the opposite of the operation, for example the opposite of + is -) and move the command to the Redo Stack. Upon Redo you need to apply the normal operation. Use the MStack push/pop operations for this. See the example test case below of Undo and Redo.

9. Your program must also manage cases where the Stack could be empty. In the case below the Undo was attempted when the undo stack as empty. Note this could also happen to Redo.

Example run:

Example program execution

Your program needs to produce the same numeric results given the input, but it can differ if format – Text highlighted in Yellow is the input to the program.

```
COMP10152 - Lab#5 - Calculator using Queues and Stacks ... by _______________
Enter tokens. Legal tokens are integers, +, -, *, /, U[ndo], R[edo], E[valuate] and [e]X[it]

+ 50
- 22
/ 5
* 10
U
R
U
U
U
U
R
R
R
R
R
W
+ 17
U
U
U
- 10000.77
X
+ 50
Total = 50.0
- 22
Total = 28.0
/ 5
Total = 5.6
* 10
Total = 56.0
U
Total = 5.6
R
Total = 56.0
U
Total = 5.6
U
Total = 28.0
U
Total = 50.0
U
Total = 0.0
R
Total = 50.0
R
Total = 28.0
R
Total = 5.6
R
Total = 56.0
R
ERROR --> Redo is empty - Can't Redo
Total = 56.0
W
```

```
ERROR --> Invalid Token - line ignored
+ 17
Total = 73.0
U
Total = 56.0
U
Total = 5.6
U
Total = 28.0
- 10000.77
Total = -9972.77
```

Useful classes and methods for this lab

- Scanner - nextLine,
- Double.parseDouble

Hints

- Use the isEmpty method of the MQueue/MStack to decide if queue or stack is empty (useful for input and error processing)

Marking Scheme

Program structure – Comments, follows best programming practices, designed using data structures and class as specified - 40%
Supports all operators Undo/Redo Stack - 40%
Error processing for invalid input – Empty conditions – 20%