Programming Project 3 – Stacks

Note: When you turn in an assignment to be graded in this class, you are making the claim that you neither gave nor received assistance on the work you turned in (except, of course, assistance from the instructor or teaching assistants). Any assistance from other sources, including the internet, is an honor code violation.

In this project you will implement the Stack ADT to work with the BRIDGES classes and implement an application to evaluate infix expressions. This project consists of 3 parts.

1. You will write a `MyStack<E>` class that implements the `StackInterface<E>` using a linked chain. Your `MyStack` class is required to use a private inner class implementation for the node of the linked chain called `MyNode`. The class will have a main method to demonstrate your implementation on a stack of integers; perform a sequence of push and pops and output the stack contents (you will need to write a `display()` method within the `MyStack<E>` class that will provide an output of the stack). Here is how your output should be formatted:

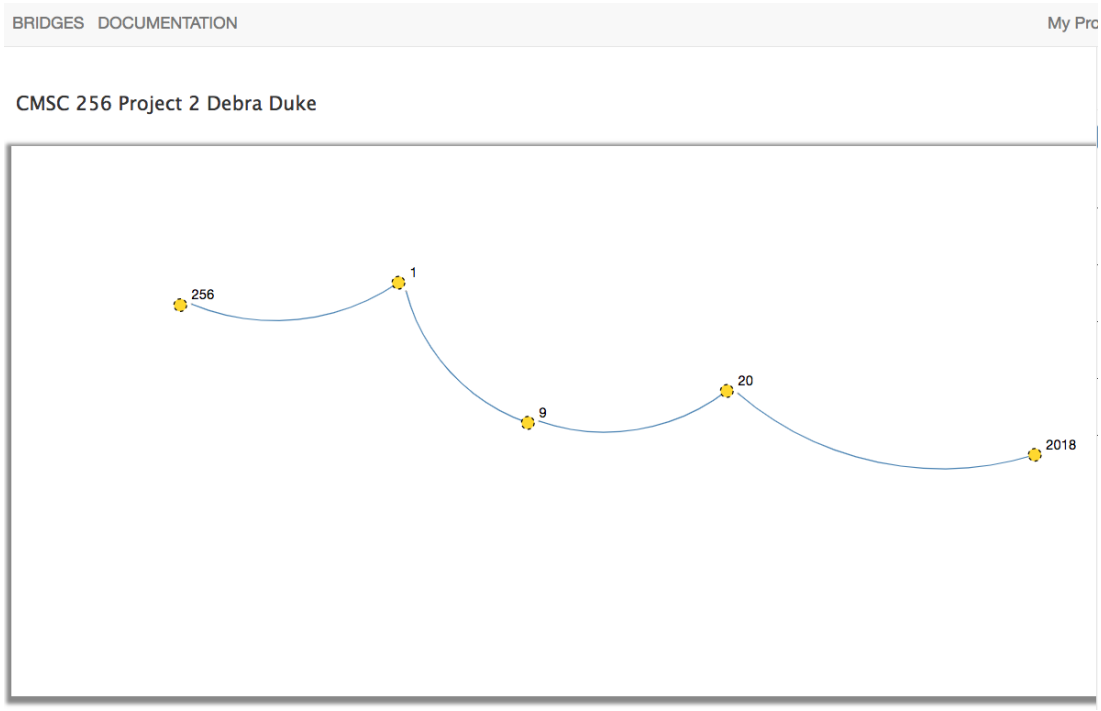
```
Pushed 256, 1, 9, 20, and 2018
Top of stack: 2018
              20
              9
              1
Stack bottom: 256

Called pop twice:
Top of stack: 9
              1
Stack bottom: 256

A call to peek returns 9
Top of stack: 9
              1
Stack bottom: 256

Called pop three times:
The stack is empty
```

Create another class called `InfixExpressionEvaluator`. This class consists of a main method and at least one other method to implement an infix expression evaluator using the MyStack class that you wrote for part 1. The pseudocode for this is located in Ch. 5 of your textbook. Implementation details for the `InfixExpressionEvaluator` class:

- Input. Several different infix expressions will be used to test your program. The input expressions are in symbolic form (using the letters 'a' through 'f', for instance, "a+b*c-9", without the double quotes) as the first command line argument to your program.
- Input. The characters, 'a' through 'f' will represent symbols with values provided for each symbol. You will read and parse this string. For example, "a=2 b=3 c=4" would be given as the second command line argument. All values for the symbols will be whole numbers.

- Output. The results of the expression.

2. You will adapt the implementation from part 1 to use the BRIDGES `SLelement` class, instead of the `MyNode` class. This class is named MyBridgesStack<E>. Test your implementation in a similar way as the MyStack class using a display method. In addition, send the stack contents to the Bridges visualizer.
- Output. Write a main method that will exercise the stack operations and produce the same standard output as was done for part 1. Use the BRIDGES visualizer to display the contents of the full stack, before any elements are popped off.



Follow the Coding Style Guideline in Blackboard and include a comment block that includes your name, the name of the project along with the file name and a brief description of the purpose of the class at the top of **every source code file that you submit**. Remove any package statements from your source code files. Your program must compile and execute correctly using the Command Prompt (Windows) or Terminal (Mac).

Submit your program by compressing the source code files into a single zipped/compressed folder and uploading it in Blackboard. You are only to submit a single compressed folder that contains only your source code. Double-check your submission to Blackboard. Submitting bytecode or other submission errors will result in a 0 for the project. Late projects are not accepted. You may submit partially completed project (make sure the code compiles) for partial credit.

Ask questions about any part of the programming project that is not clear!