**ITE7107: Laboratory Exercise 001 Answers**

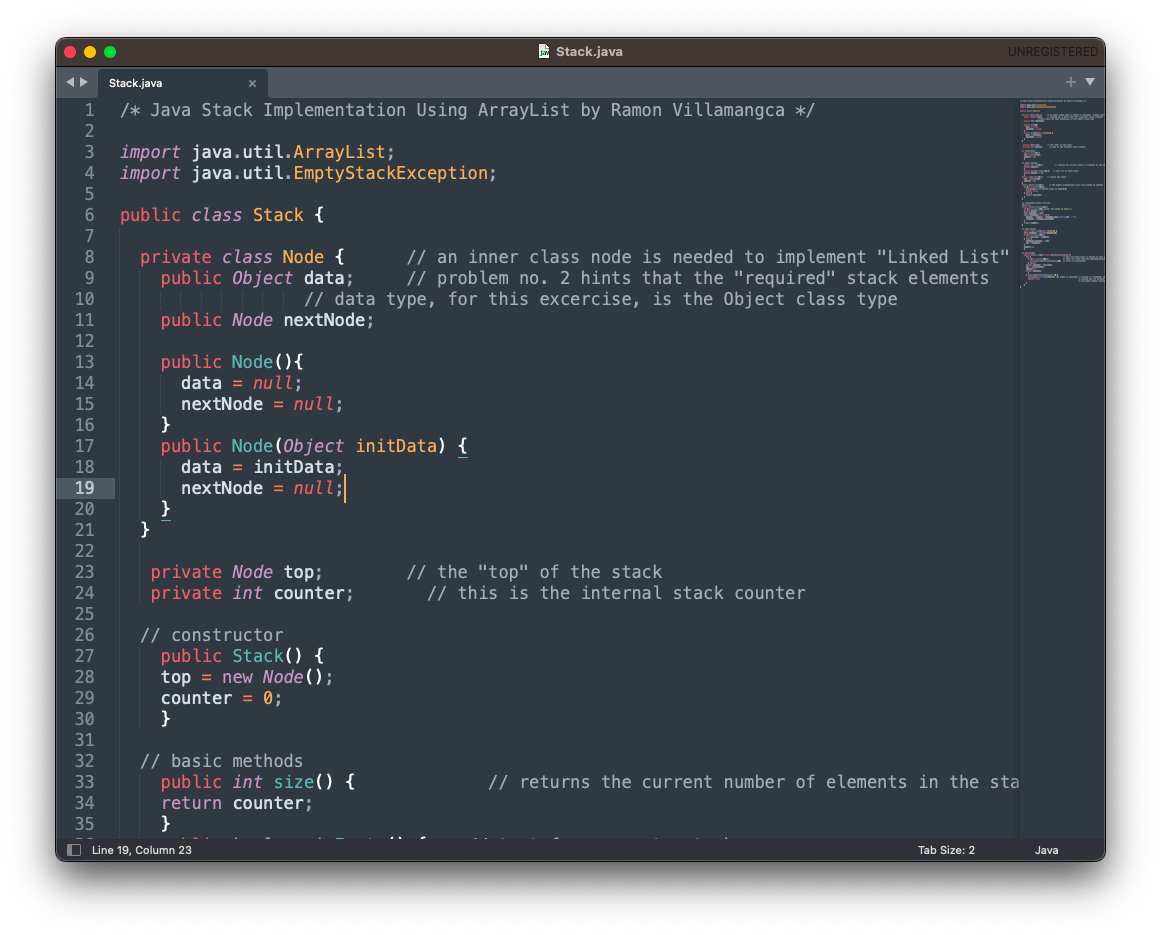
C++ Linked Lists, Stacks and Queues

Note: Source codes used for this exercise, together with this answer sheet are uploaded in the GitHub repository: <https://github.com/rvillamangca/AMA-ITE7107-DataSturctureAlgorithms>. Although the course is about C++ Data Structures implementation, it is noted that all exercises require Java language instead. Hence, the answers presented here are all written in Java.

Question 1: **Write the Stack constructor.**

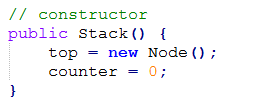
Answer:

Below figure shows the proposed Stack implementation. A Linked List Stack implementation is chosen as this is more efficient than the Array List implementation. This implementation affords an O(1) complexity in all the methods required and specified the Lesson Modules for this chapter.



* Stack Constructor

Since this is a Linked List implementation Stack Constructor code is as follows:



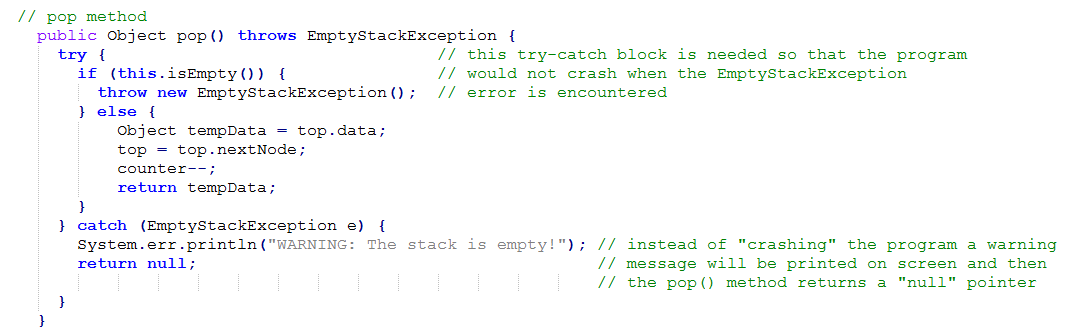
The constructor would be different if Array List is used.

Question 2: **Complete the pop method, using the following header:**



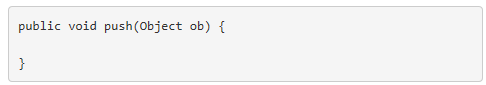
Answer:

Using the Linked List Stack implementation mentioned above below is the pop method:



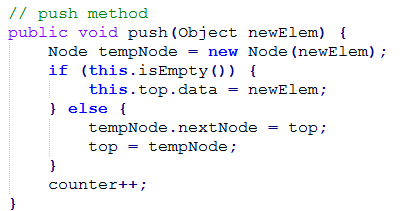
Throwing an exception could be actually uncalled for in this simple method. Without the use of a “try-catch” block, calling the pop method on an empty stack, will “crash” the program. It would have been better (and shorter) to just remove the “exception throwing” and just prints a warning string if the stack is empty. Note that the code would be different is Array List implementation is used.

Question 3: **Complete the push method, using the following header.**



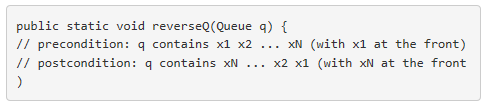
Answer:

Using the Linked List Stack implementation mentioned above below is the push method:



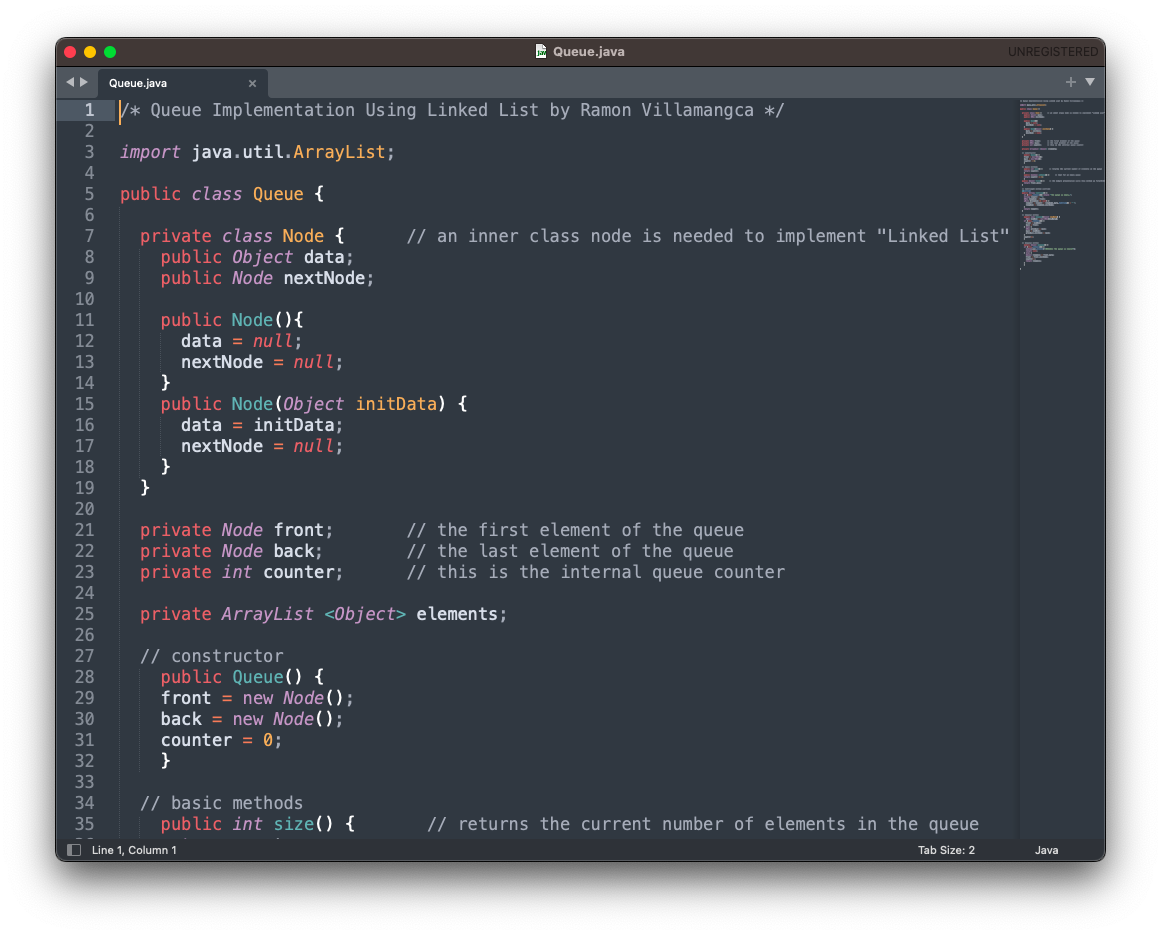
This code has constant O(1) complexity as new elements are added before the root node of the link. Again, the code would be different is Array List is used.

Question 4: **Complete method reverseQ, whose header is given below. Method reverseQ should use a Stack to reverse the order of the items in its Queue parameter.**



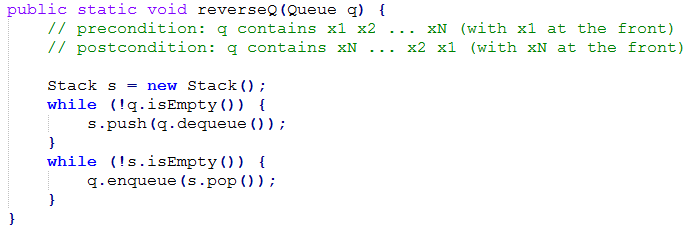
Answer:

Below figure shows the proposed Queue implementation proposed A Linked List Queue implementation is chosen. To ensure a constant O(1) complexity in all the methods required and specified the Lesson Modules for this chapter, the implementation monitors not only the “front” but also the “back” of the Queue. Doing this, there will be no need to traverse to the end of the linked list when the “enqueue” method is called.



* Queue Reversal Method

The Queue Reversal Method code is as follows:



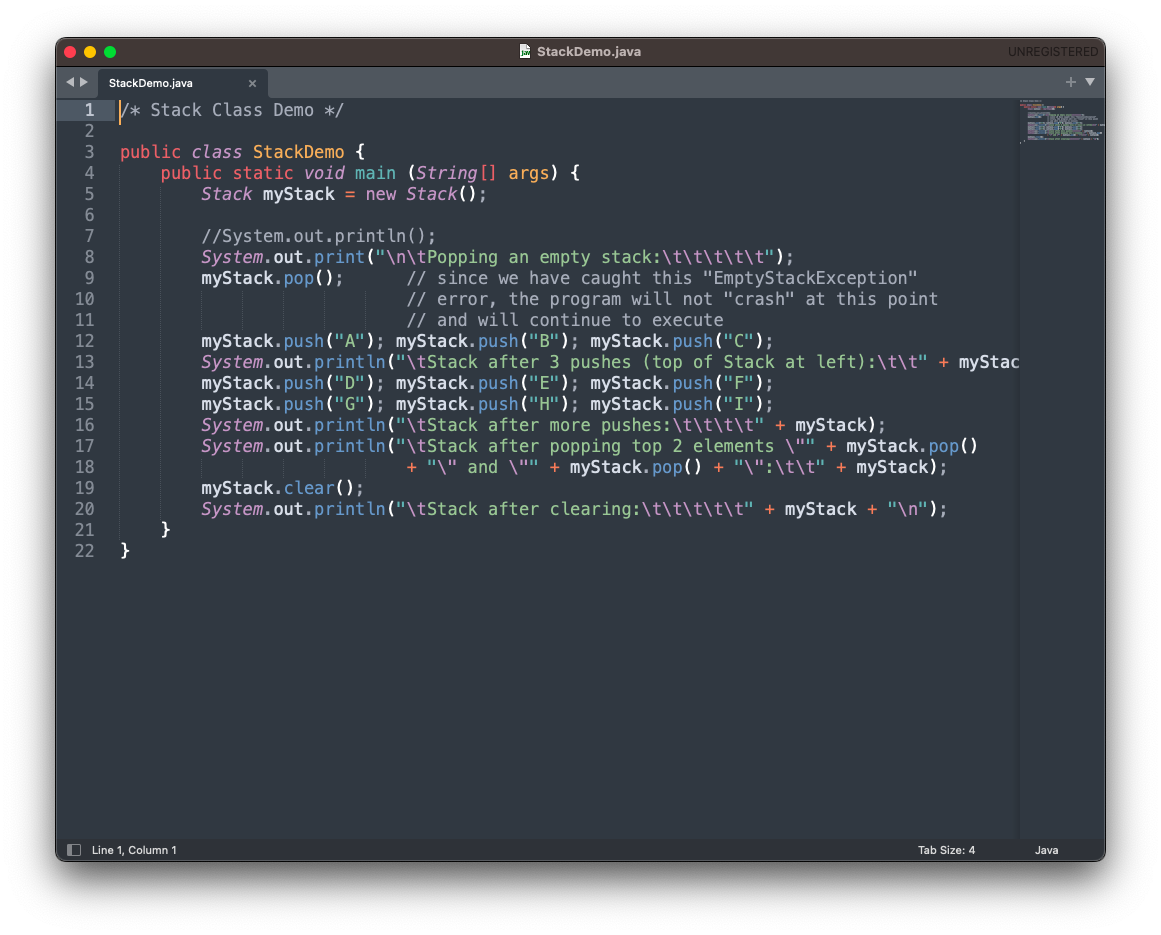
Note that the above code makes use of an instance of the Stack class previously implemented.

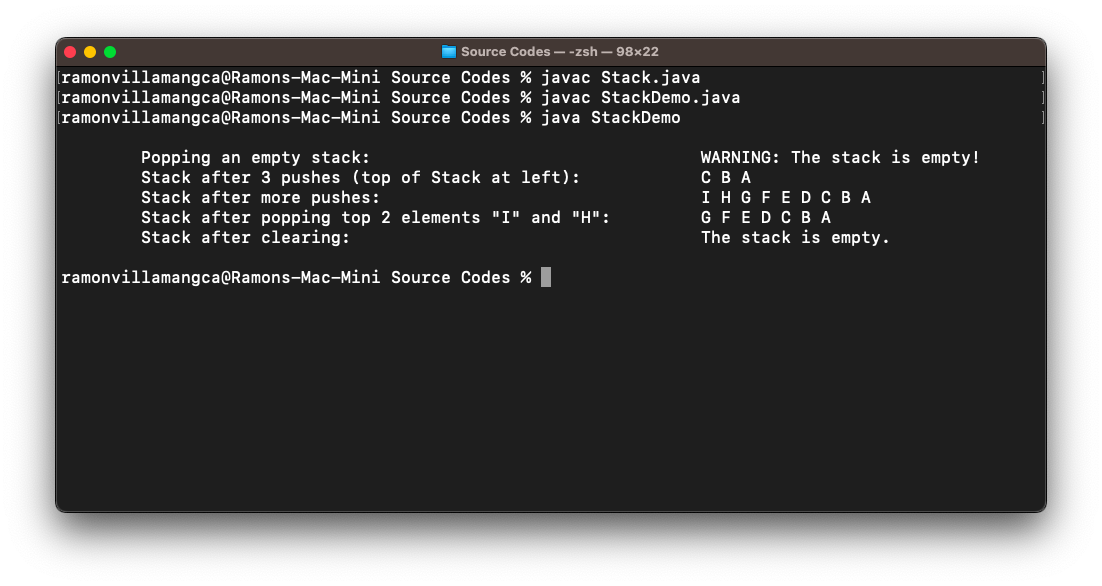
Furthermore, since this is a static method, this student elected to put this method inside the queue driver class.

**Implementation Testing:**

To test the Stack and Queue implementations, driver programs were also written with the following results.

* Stack Implementation Test:





* Queue Implementation Test:

