CSC172 LAB 6

STACKS

1 Introduction

The labs in CSC172 will follow a pair programming paradigm. Every student is encouraged (but not strictly required) to have a lab partner. Labs will typically have an even number of components. The two partners in a pair programming environment take turns at the keyboard. This paradigm facilitates code improvement through collaborative efforts, and exercises the programmers cognitive ability to understand and discuss concepts fundamental to computer programming. The use of pair programming is optional in CSC172. It is not a requirement. You can learn more about the pair programming paradigm, its history, methods, practical benefits, philosophical underpinnings, and scientific validation at http://en.wikipedia.org/wiki/Pair_programming

Every student must hand in their own work, but every student must list the name of their lab partner if any on all labs.

This lab has six parts. You and your partner(s) should switch off typing each part, as explained by your lab TA. As one person types the lab, the other should be watching over the code and offering suggestions. Each part should be in addition to the previous parts, so do not erase any previous work when you switch.

The textbook should present examples of the code necessary to complete this lab. However, collaboration is allowed. You and your lab partner may discuss the lab with other pairs in the lab. It is acceptable to write code on the white board for the benefit of other lab pairs, but you are not allowed to electronically copy and/or transfer files between groups.

The goal of this lab is to gain familiarity with linked list implementations of stacks.

2 Stacks

Stacks are specialized lists of data that only support inserting and deleting data from one end of the list. This means that stacks are LIFO data structures, where the Last In item is the First Out. This section of the lab will guide you through creating a stack using the singly linked list you made in Lab 2.

1. Begin this lab by typing in the code for the generic stack interface.

```
public interface Stack<AnyType> {
     public boolean isEmpty();
     public void push(AnyType x);
     public AnyType pop();
     public AnyType peek();
}
```

2. Create your own class that implements the `Stack` interface from part 1. Your stack should contain a reference to the singly linked list class you made in Lab 2 that is instantiated in the class constructor. Modify your linked list as necessary to support constant time insertions and deletions from the front of the list.
3. Implement the `push()` method in your stack which will insert the given object into the front of your linked list (top of the stack).
4. Implement the `pop()` method, which removes and returns the top (most recently inserted) item from the stack. Your method should return `null` if the stack is empty. Implement the `isEmpty()` method to check if the stack is empty and use it in your `pop()` method.
5. Sometimes we want to know what's on the top of the stack without having to pop it off the stack. Implement the `peek()` method to support this functionality.
6. Write a test program that instantiates an instance of your stack class and demonstrates that each of the required methods from the interface work. At a minimum, you should push a few objects onto the stack, check if the stack is empty, peek at the top item, and pop a few items off the stack.

3 Hand In

Hand in the source code from this lab at the appropriate location on the BlackBoard system at my.rochester.edu. You should hand in a single zip file (compressed archive) containing your source code, README, and OUTPUT files, as described below.

1. A plain text file named README that includes your contact information, your partner's name, a brief explanation of the lab (A one paragraph synopsis. Include information identifying what class and lab number your files represent.), and one sentence explaining the contents of all the other files you hand in.
2. All Java source code files representing the work accomplished for this lab. All source code files should contain author and partner identification in the comments at the top of the file.
3. A plain text file named OUTPUT that includes author information at the beginning and shows the compile and run steps of your code. The best way to generate this file is to cut and paste from the command line.

4 Grading

Your grade will be determined based on the correct implementation of the following (total 90%):

Stacks

10% generic interface, class, and constructor for the stack

20% push() method, with the necessary modifications to support inserting in the front of the list.

20% pop() method, with the necessary modifications to support removing from the front of the list.

10% peek() method

10% isEmpty() method

20% appropriate test cases in your test program

The README and OUTPUT files account for the remaining 10% of the lab grade.