Stack and Queue Lab

CS 331 — Fall, 2010

Revision 1.0

Assigned
Due

1 Introduction

As we discussed in lecture, stacks and queues are easy to code if you already have a working linked list implementation. In this lab, you will encapsulate a linked list (provided for you) to implement a stack class and a queue class. You will also write JUnit tests to make sure this thing really works.

1.1 Objectives

- Have experience using encapsulation to create new behaviors.
- Implement a stack.
- Implement an efficient queue.

2 Given Files

You are given 7 files for this lab. They are in the `stack-queue-lab` directory of your repository. Do a `git pull` to download them.

- `List.java` — A linked list implementation. Read it to see what kinds of methods are available. It should look pretty familiar to you.
- `Makefile` — the scripts needed to compile your files and run the tests. It supports the following targets:

  compile Compiles everything.

  compile-stack Just compiles the `Stack` and `List` classes (and tests).

  compile-queue Just compiles the `Queue` and `List` classes (and tests).

  tests Tests everything.

  test-stack Just tests the `Stack` and `List` classes.

  test-queue Just tests the `Queue` and `List` classes.

- `TestList.java` — These are the test cases for the `List` class.
- `Stack.java` — The empty `Stack` class.

- `Queue.java` — The empty `Queue` class.
- `TestStack.java` — The shell for the `Stack` class testing functions.
- `TestQueue.java` — The shell for the `Queue` class testing functions.

Note that the stack and queue code will not work out of the box.

3 Command Line Hint of the Day

Most people who use a command line have this habit: *every time* you change to a new directory, run `ls` to see what's in it. Typically, you already know what should be there, but if you are accidentally in the wrong directory, you'll see it right away, before you start adding files or changing things.

4 Your Work

For both of these classes, you must implement the methods listed below, and you must use the `List` class we've provided. We highly recommend that you write the test cases before you write the methods. It's okay if you write your test case, then after writing the method you decide to make changes to the test case. The point is that you should be making a good effort to write the test cases first.

4.1 The Stack Class

Stack<E> The constructor.

> **Inputs** none
>
> **Output** none

push

> **Inputs** `E data`
>
> **Output** none
>
> **Actions** Adds an element to the top of the stack.

pop

> **Inputs** none
>
> **Output** `E` — the element at the top of the stack. Return `null` if the stack is empty.
>
> **Actions** Removes the element at the top of the stack. No action if the stack is empty.

top

> **Inputs** none
>
> **Output** `E` — the element at the top of the stack. Return `null` if the stack is empty.
>
> **Actions** none

size

Inputs none

Output `int` — the number of elements in the stack.

Actions none

For testing, you will want to test several things.

- That an empty stack behaves properly. (Size returns zero, pop and top return `null`.)
- That `push` and `top` work together properly. After every `push`, the `top` method should return what was just `push`ed.
- Elements pushed onto the stack are popped out in the proper order, and the size is adjusted properly.

4.2 The Queue Class

Queue<E>

Inputs none

Output none

enqueue

Inputs `E` — the data to enqueue

Output none

Actions Adds the data to the back of the queue.

There were two ways given to implement this in class. For this version, it is probably best to use a `List` class which has a `last` pointer in it, a `insertAtEnd` method, and a `deleteFromFront` method. Then you can enqueue by inserting into the back of the list, and dequeue by removing from the front of the list.

If you prefer, you can use the two-list method.

dequeue

Inputs none

Output `E` — the data. Return `null` if the queue is empty.

Actions Removes the data from the queue.

front

Inputs none

Output `E` — the data. Return `null` if the queue is empty.

Actions none

size

Inputs none

Output `int` — the number of elements in the queue.

Actions none

For testing, you will want to test several things.

- That an empty queue behaves properly. (Size returns zero, dequeue and front return `null`.)
- That `enqueue` and `front` work together properly. After every `enqueue`, the `front` method should return the first thing enqueued, and after every `dequeue`, the `front` method should return what will be dequeued next.
- Elements enqueued are dequeued in the proper order, and the size is adjusted properly.

5 Handing In

To hand in your code, commit your final changes to the repository. The grading script (which will be turned on later) will grade your lab and put the results into your repository.