CS2401 – Weeks 13 & 14

In this lab assignment, we are going to practice stacks and queues. We hope you enjoy this assignment! Let's get started!

What is the scenario?

We have designed a tree structure that holds information about a network on helpful people. This is a limited network where each person is connected to two people: this person's two best advisors. Once we have this network populated, we would like to be able to access it in multiple ways. Among these many ways, we are interested in the following:

- Accessing the network level by level: to be able to answer questions like "who are the advisors to whom I am only distant by 2 (i.e., advisors at the 3rd level of the tree-network)?", "who are my most distant advisors?", etc.
- Accessing the network by branch: "who am I connected to through this given person?", etc.

Because in this lab, we do not want to use recursion, in order to do all of the above, we need new data structures, namely, stacks and queues. This is your job, in this lab. More details about the implementation are available below.

Let's build data structures

You are tasked with putting together UDT java files to handle Stacks and Queues. More specifically, you have to:

- Design a Stack data structure based on linked lists: we will call this file StackL.java.
- Design a Queue data structure based on arrays: we will call this file QueueA.java.

We, as users, should then be able to use your types seamlessly. To ensure that this happens, let's agree on the names of the class methods and how to use them.

Here are the details of each of these files: you have to abide by the descriptions below so that we can then use your types and methods in our own main method.

StackL.java:

- Attributes: GIVEN
    - Top
    - Size
- 2 constructors: TODO1
    - One default constructor
    - One constructor that take a node and starts the stack with this node in it

- Accessors / getters: TODO2
    - Only one
    - getSize
- Other methods: TODO3,4,5,6,7
    - Push: takes a person (see Person.java) and adds it to the stack
    - Pop: void method that removes the top element of the stack, if the stack is not empty
    - Peek: returns the top element of the stack (without removing it)
    - Clear: empties the stack
    - isEmpty: returns true if the stack is empty, false otherwise

QueueA.java:

- Attributes: GIVEN
    - Head (an integer index)
    - Tail (an integer index)
    - Size
    - Array of elements of generic type T
- 1 Constructor: GIVEN
    - Takes an integer as a parameter: it is the maximum size of your queue
- Accessors / getters: TODO 8
    - Only one
    - getSize
- Other methods: TODO 9,10,11,12,13,14
    - Enqueue: takes data and adds it to the queue if it is not full
    - Dequeue: method that removes the head of the queue, if the queue is not empty, and returns this element
    - Peek: returns the head element of the queue (without removing it)
    - Clear: empties the queue
    - isEmpty: returns true if the queue is empty, false otherwise
    - isFull: returns true if the queue is full, false otherwise

That's not all: in Execute.java, you now have to create tree traversal methods that allow the following:

- TODO 15: Level-order traversal (makes use of a QueueA type structure).
- TODO 16: Pre-order traversal (makes use of a StackL type structure).
- TODO 17: ExploreBranch: takes as input a string that is only made of letters L and R (for left and right respectively), indicating how to navigate the given branch.

Dummy stubs are provided within Execute.java for you to complete these methods. Also, comments are placed in the main for you to complete it and be able to run the methods you have designed (TODO 18).

Note: we provide:

- Node.java;
- BTNode.java;
- BTree.java;
- Person.java;
- Starter code for StackL.java;
- Starter code for QueueA.java;
- code in Execute.java to create our tree; and
- an example of file to populate your tree of advisors (code provided to create the tree).

You have to complete StackL.java and QueueA.java, and to modify Execute.java.

Grading: over 105 pts
10 pts Code is deemed to be of high quality
StackL.java: 20 pts
5 pts Correct constructors, getter
15 pts Correct other methods
QueueA.java: 20 pts
20 pts Correct other methods
BTree.java: 55 pts
15 pts Level-order traversal (using QueueA)
15 pts Pre-order traversal (using StackL)
10 pts ExploreBranch method
15 pts Correct main method

Due date: November 30th at 11:59pm

How to submit?
Check with your own TA for submission guidelines.

Failing to follow submission instructions and guidelines will result in up to 15 points off your overall grade in this lab. So please pay attention.