CS 342 **Project #1** **Fall 2019**

Generic Lists in Java

Due Date: Sunday, September 29th @11:59pm 2019

Description:

In this project you will implement your own versions of the stack and queue data structures. Although the Java API provides built-in support for them, you will write your own to practice the constructs and language details we have seen in class. That means you are NOT allowed to use any pre existing libraries or classes for this assignment.

In this simplified version, each data structure is a singly linked list. The stack is LIFO (last in first out) while the queue is FIFO (first in first out). Your implementation must be generic as to allow for different types when each data structure object is instantiated.

You will also implement the Iterator design pattern; allowing users access to a custom Iterator for your data structures.

Implementation Details:

You will download and use the Maven project template GLMavenFall19 from Blackboard. You will find a file called GLProject.java in the src folder. This class contains the main method. You should test this template with the Maven command “compile”. This command will both compile your src code and run exec:java; resulting in the line “hello generic lists” being printed and a successful build. The Maven command “test” will run one test case that fails all the time. You will create a new file, inside of src/main/java, for each outer class and interface. In comments at the top of the file GLProject.java, please include your name and netid and university email as well as a brief description of your project.

DO NOT add any folders or change the structure of the project in anyway. DO NOT alter the pom.xml file.

Create a generic abstract class called **GenericList<I>**:

This class will contain only two data fields:
Node<I> head (**t**his is the head of the list).
int length (the length of the list and should be private)

This class should include the following methods:
print(): prints the items of the list, one value per line.
add(I data): adds the value to the list. This is abstract since the implementation depends on what the data structure is.
delete(): returns the first value of the list and deletes the node.

This class should also define a generic inner class **Node<T>**: It will include two fields: **T data** and **Node<T> next**;

This class encapsulates a linked list. Defining a Node class and providing two methods that a queue and stack have in common while leaving adding a node to each implementation that inherits from it.

Create two more classes **GenericQueue<I>** and **GenericStack<I>**. They both should inherit from **GenericList<I>**.

The constructors for each class will take one parameter. That parameter will be a value that will go in the first node of the list encapsulated by each instance of the class. Each constructor should initialize the linked list **head**, with the value passed in by the constructor. Each class should also implement the method **add(I data)**, **GenericQueue** will add to the back of the list while **GenericStack** will add to the front. Each class must also keep track of the length of it's list using the **length** data field defined in the abstract class.

GenericQueue will have the methods **enqueue(I data)** and **dequeue()** which will call the methods **add(I data)** and **delete()** respectively. Enqueue and dequeue merely call **add(I data)** and **delete().** The reason for this is that a user would expect these calls to be implemented in each of those data structures. You will do the same with **GenericStack**

GenericStack will have the methods **push(I data)** and **pop()** which will call the methods **add(I data)** and **delete()** respectively.

Once implemented, you should be able to create instances of both GenericQueue and GenericStack in main with most primitive wrapper classes. You should be able to add and delete nodes to each data structure instance as well as print out the entire list and check for the length of the list. You must follow this implementation: meaning you can not add any additional data fields or classes. You may add getters/setters as need be.

Implementing Iterator Design Pattern:

You are to create an interface called **CreateIterator**. It will have one abstract method: **Iterator createIterator()**. You will want both GenericQueue and GenericStack classes to implement this interface.

You must also create a class to contain logic for iterating through your data structure. Call this class **GLIterator**.

It should implement the java interface Iterator (java.util.Iterator). You will have to implement two inherited methods: **hasNext()** and **next().**

You are expected to fully comment your code and use good coding practices.

Electronic Submission:

Zip the Maven project GLMavenFall19 and name it with your netid + Project1: for example, I would have a submission called mhalle5Project1.zip, and submit it to the link on Blackboard course website.

Assignment Details:

Late work **is accepted**. You may submit your code up to 24 hours late for a 10%

penalty. Anything later than 24 hours will not be graded and result in a zero.

We will test all projects on the command line using Maven 3.6.1. You may develop in any IDE you chose but make sure your project can be run on the command line using Maven commands. Any project that does not run will result in a zero. If you are unsure about using Maven, come see your TA or Professor.

Unless stated otherwise, all work submitted for grading *must* be done individually. While we encourage you to talk to your peers and learn from them, this interaction must be superficial with regards to all work submitted for grading. This means you *cannot* work in teams, you cannot work side-by-side, you cannot submit someone else's work (partial or complete) as your own. The University's policy is available here:

https://dos.uic.edu/conductforstudents.shtml.

In particular, note that you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance. Absolutely no transfer of program code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums. Other examples of academic dishonesty include emailing

your program to another student, copying-pasting code from the internet, working in a group on a homework assignment, and allowing a tutor, TA, or another individual to write an answer for you. It is also considered academic dishonesty if you click someone else's iClicker with the intent of answering for that student, whether for a quiz, exam, or class participation. Academic dishonesty is unacceptable, and penalties range from a letter grade drop to expulsion from the university; cases are handled via the official student conduct process described at https://dos.uic.edu/conductforstudents.shtml.