Homework 2: Linked Lists, Stacks, and Queues *Due: See T-Square*

Important

There are general homework guidelines you must always follow. If you fail to follow any of the following guidelines you risk receiving a **0** for the entire assignment.

1. All submitted code must compile under **JDK 8**. This includes unused code, so don't submit extra files that don't compile. (Java is backwards compatibile so if it compiles under JDK 7 it *should* compile under JDK 8.)
2. Do not include any package declarations in your classes.
3. Do not change any existing class headers, constructors, or method signatures.
4. Do not add additional public methods when implementing an interface.
5. Do not use anything that would trivialize the assignment. (e.g. don't import/use `java.util.LinkedList` for a Linked List assignment. Ask if you are unsure.)
6. You must submit your source code, the `.java` files, not the compiled `.class` files.
7. Code exactly what is asked for. No more and no less.
8. After you submit your files redownload them and run them to make sure they are what you intended to submit. You are responsible if you submit the wrong files.

Linked List

You are to code a simple doubly linked list with a head and a tail reference. A linked list is collection of nodes, each having a data item and a reference pointing to the next node and previous node. There are several different types of linked lists, so it is important to note that this assignment requires a doubly linked, non-circular linked list with a head and a tail reference. The list will have a head node that points to another node, which points to another and the previous node, etc. until you get to the tail node.

Your linked list implementation will implement the LinkedListInterface provided. It will have the public default constructor only - the one with no parameters. You will not write the default constructor, as it is automatically provided by Java. **So do not write any constructors.**

Adding

You will implement three add methods. One will add to the front, one will add to the back, and one will add at a given index. See the interface.

Removing

Removing, just like adding, can be done from anywhere in your linked list. When removing from the front, the head reference should point to the second node in the list. When removing from the back, the tail reference changes to point to the second to last node. Make sure that you set any pointers to the deleted nodes to null. See the interface.

Nodes

The linked list consists of nodes. Nodes have three instance variables: data, previous, and next. Do not modify this class.

Stacks and Queues

Once Linked List is coded and working, use it as the backend to implement both a stack and a queue. Notice that in the files Queue.java and Stack.java, there is a private LinkedListInterface reference pointing to a DoublyLinkedList object. Do not alter these declarations and instantiations. If you need private helper methods in those classes, you can add them. Do not add additional public methods beyond those demanded by the Queue and Stack interfaces respectively. Do not write any public getters or setters for these classes as that will completely violate encapsulation and the abstraction we are trying to enforce. You are to interact with the underlying linked list object via the public methods from the LinkedListInterface only. Do not try to access and/or manipulate the LinkedList directly. **DO NOT** use the `getHead()` and `getTail()` methods. These are for **grading purposes only.**

Stack

A stack is a data structure that follows the "LIFO, Last In First Out" set of rules. You can think of it as a stack of papers (or maybe tickets). If you lay three sheets of paper on top of each other and then want a sheet of paper, you will take the top sheet. The last one that you added to the stack is therefore the first one that leaves it. In your implementation, you will be required to implement push and pop methods. See the Javadoc for more instructions.

Your stack implementation will implement the StackInterface provided. Your implementation should override the default constructor.

Queues

A queue follows the "FIFO, First In First Out" set of rules. Think of it as a line (maybe at an amusement park). The first person in line expects to be the first one to get his or her ticket. For this homework you will implement an enqueue and dequeue method. See the Javadoc for more instructions.

Your queue implementation will implement the QueueInterface provided. Your implementation should override the default constructor.

Style and Formatting

It is important that your code is not only functional but is also written clearly and with good style. We will be checking your code against a style checker that we are providing. It is located in resources along with instructions on how to use it. We will take off a point for every style error that occurs. If you feel like what you wrote is in accordance with good style but still sets off the style checker, please email Jonathan Jemson (jonathanjemson@gatech.edu) and Matt Gruchacz (mattgruchacz@gatech.edu) with the subject header of "Style XML". **Please note that there is a new CheckStyle XML. All homework assignments starting with this one will use the new CheckStyle.**

Javadoc

Javadoc any helper methods you create in a style similar to the Javadoc for the methods in the interface.

JUnits

For this homework, no JUnit testing is required, but you may find it beneficial to unit test your linked list, stack, and queue as you write them to help avoid problems.

Provided

The following file(s) have been provided to you.

1. `LinkedListInterface.java` This is the interface you will implement. All instructions for what the methods should do are in the Javadoc. **Do not alter this file.**
2. `LinkedList.java` This is the class in which you will actually implement the interface. Feel free to add private helpers but **do not add any new public methods.**
3. `StackInterface.java` This is the interface for the stack you will implement. All instructions for what the methods should do are in the Javadoc. **Do not alter this file.**
4. `Stack.java` This is the class that will implement your stack. Look at the Javadoc for the given methods for information on how to implement them. Feel free to add private helpers but **do not add any new public methods.**
5. `QueueInterface.java` This is the interface for the queue you will implement. All instructions for what the methods should do are in the Javadoc. **Do not alter this file.**
6. `Queue.java` This is the queue implementation. Check the Javadoc for instructions and feel free to add additional methods (with Javadoc!). Feel free to add private helpers but **do not add any new public methods.**
7. `Node.java` This class encapsulates the data, the next reference, and the previous reference. **Do not alter this file.**
8. `HW2JUnitsStudent.java` These are basic JUnit tests for this assignment. You are encouraged to write additional tests to ensure you have accounted for all edge cases.

Deliverables

You must submit all of the following file(s). Please make sure the filename matches the filename(s) below. Be sure you receive the confirmation email from T-Square, and then download your uploaded files to a new folder, copy over the interfaces, recompile, and run. It is your responsibility to re-test your submission and discover editing oddities, upload issues, etc.

1. `DoublyLinkedList.java`
2. `Stack.java`
3. `Queue.java`

You may attach each file individually, or submit them in a zip archive.