## 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. Any compile errors will result in a 0.
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.
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. Always be very conscious of efficiency. Even if your method is to be $O(n)$, traversing the structure multiple times is considered non-efficient unless that is absolutely required (and that case is extremely rare).
7. You must submit your source code, the `.java` files, not the compiled `.class` files.
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.

## Instructions

You will be coding the following for this homework:

1. A Doubly-Linked List for your Stack and Queue implementations
2. Two Stack implementations, backed using an Array and a Linked List
3. Two Queue implementations, backed using an Array and a Linked List

## Stacks and Queues

A queue is a first-in, first-out (FIFO) data structure. A stack is a last-in, first-out (LIFO) data structure. These are ADTs, so there are multiple different implementations, two of which you will be coding for this homework.

All four of the stack/queue implementations should implement either `QueueInterface` or `StackInterface`. In the array implementations, the initial size of the array is specified in the interface. Use this variable in your code. For the linked list implementations, you are to use `LinkedListInterface` with its implementation `DoublyLinkedList` as the backing linked list. Do NOT use `java.util.LinkedList`, use your implementation. Do NOT change the instance variable type.

These implementations must be as efficient as possible. **Failure to do so will result in large point deductions.**

## Doubly-Linked List

A doubly-linked list is similar to a singly-linked list conceptually, except now, there is a previous reference. This allows for slightly faster expected Big-O for certain methods in the linked list at the expense of more memory use for the previous references.

Your implementation of a doubly-linked list should implement the `LinkedListInterface`, and be coded in `DoublyLinkedList`. Keep in mind that the primary portion of this assignment is the Stacks and Queues implementation, but if your linked list does not function properly, your Stack and Queue implementations will not function properly either.

While it is not expressly forbidden to reuse code from Homework 1, you may find it easier to simply code the Doubly-Linked List implementation from the beginning since many of the methods from Homework 1 are no longer there.

## Homework Notes

- The time complexities are specified in the javadocs, but keep in mind that there is also space efficiency to be concerned about! For example, for the array-backed implementation, be careful with when you resize.
- The array implementation operations for enqueuing and pushing are noted as *amortized* O(1). You can think of this as *on average* being O(1). That is, the intensive O(n) operation of resizing only happens around the `nth` call of the method, but since all of the other times the method is called it is an O(1) operation, it is called amortized O(1).

## A note on JUnits

We have provided a **very basic** set of tests for your code, in `StacksQueuesStudentTests.java`. These tests do not guarantee the correctness of your code (by any measure), nor does it guarantee you any grade. You may additionally post your own set of tests for others to use on the Georgia Tech GitHub as a gist. Do **NOT** post your tests on the public GitHub. There will be a link to the Georgia Tech GitHub as well as a list of JUnits other students have posted on the class Piazza (when it comes up).

If you need help on running JUnits, there is a guide, available on T-Square under Resources, to help you run JUnits on the command line, in IntelliJ, or in Eclipse.

## Forbidden Statements

You may not use these in your code at any time in CS 1332. If you use these, we will take off points.

- `break` may only be used in switch-case statements
- `continue`
- `package`
- `System.arraycopy()`
- `clone()`
- `assert()`
- `Arrays` class

- `Array` class
- `Collections` class
- `Collection.toArray()`
- Reflection APIs
- Inner or nested classes

Debug print statements are fine, but nothing should be printed when we run them. We expect clean runs - printing to the console when we're grading will result in a penalty.

## Provided

The following file(s) have been provided to you. There are several, but you will only edit one of them.

1. `LinkedListInterface.java`

   This is the interface you will implement in `DoublyLinkedList`. All instructions for what the methods should do are in the javadocs. **Do not alter this file.**

2. `DoublyLinkedList.java`

   This is the class in which you will implement `LinkedListInterface`. Feel free to add private helper methods but **do not add any new public methods, inner/nested classes, instance variables, or static variables**. You may reuse your code from homework 1 for this class.

3. `LinkedListNode.java`

   This class represents a single node in the linked list. It encapsulates the `data` and the `next` reference. **Do not alter this file.**

4. `StackInterface.java`

   This is the interface you will implement in `ArrayStack` and `LinkedListStack`. All instructions for what the methods should do are in the javadocs. **Do not alter this file.**

5. `ArrayStack.java`

   This is the class in which you will implement `StackInterface` with a backing array. Feel free to add private helper methods but **do not add any new public methods, inner/nested classes, instance variables, or static variables**.

6. `LinkedListStack.java`

   This is the class in which you will implement `StackInterface` with a backing linked list. Feel free to add private helper methods but **do not add any new public methods, inner/nested classes, instance variables, or static variables**.

7. `QueueInterface.java`

   This is the interface you will implement in `ArrayQueue` and `LinkedListQueue`. All instructions for what the methods should do are in the javadocs. **Do not alter this file.**

8. `ArrayQueue.java`

   This is the class in which you will implement `QueueInterface` with a backing array. Feel free to add private helper methods but **do not add any new public methods, inner/nested classes, instance variables, or static variables**.

9. `LinkedListQueue.java`

   This is the class in which you will implement `QueueInterface` with a backing linked list. Feel free to add private helper methods but **do not add any new public methods, inner/nested classes, instance variables, or static variables**.

10. `StacksQueuesStudentTests.java`

    This is the test class that contains a set of tests covering the basic operations on the `ArrayStack`, `LinkedListStack`, `ArrayQueue`, and `LinkedListQueue` classes. It is not intended to be exhaustive and does not guarantee any type of grade. **Write your own tests to ensure you cover 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. `ArrayStack.java`
3. `LinkedListStack.java`
4. `ArrayQueue.java`
5. `LinkedListQueue.java`