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.ArrayList` for an Array 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 inefficient 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.

Stacks and Queues

You are to code the following:

1. A stack backed by a linked list
2. A stack backed by an array
3. A queue backed by a linked list
4. A queue backed by an array

A queue is a first-in, first-out (FIFO) data structure. A stack is a last-in, first-out (LIFO) data structure.

All of your data structures should implement the appropriate interface, either `QueueInterface` or `StackInterface`. Both interfaces define an initial capacity that you must use in your array implementations; make sure to use the provided variable, not a magic number. Your linked list implementations should use the given head (and tail) pointer(s) to build the backing structure. Do NOT use Java's linked list classes.

As always, these implementations must be as efficient as possible. **Failure to do so will result in large point deductions.**

Circular Arrays

The backing array in your `ArrayQueue` implementation must behave circularly. This means that when the user dequeues an element, you should simply treat the next index in the array as the new front. **DO NOT SHIFT ANY ELEMENTS IN THE ARRAY.** This also means that if there are empty

spaces at the front of the array, the back of the queue should wrap around to the front of the array and make use of those spaces.

When regrowing the backing array, realign the queue with the front of the new array during transfer, so that the front of the queue is once again at index 0. This is the ONLY time that the front of the queue should be reset. When dequeueing the last element in the queue, simply increment front as you normally would and leave it there.

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.

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 or in IntelliJ.

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 T-Square, under 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 Carey MacDonald (careyjmac@gatech.edu) with the subject header of "CheckStyle XML".

Javadocs

Javadoc any helper methods you create in a style similar to the existing Javadocs. If a method is overridden or implemented from a superclass or an interface, you may use `@Override` instead of writing Javadocs. Any Javadocs you write must be useful and describe the contract, parameters, and return value of the method; random or useless javadocs added only to appease Checkstyle will lose points.

Exceptions

When throwing exceptions, you must include a message by passing in a String as a parameter. **The message must be useful and tell the user what went wrong**. "Error", "BAD THING HAPPENED", and "fail" are not good messages. The name of the exception itself is not a good message.

For example:

```
throw new PDFReadException("Did not read PDF, will lose points.");

throw new IllegalArgumentException("Cannot insert null data into data structure.");
```

Generics

If available, use the generic type of the class; do **not** use the raw type of the class. For example, use `new LinkedList<Integer>()` instead of `new LinkedList()`. Using the raw type of the class will result in a penalty.

Forbidden Statements

You may not use these in your code at any time in CS 1332.

- `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 your code. We expect clean runs - printing to the console when we're grading will result in a penalty. If you submit these, we will take off points.

Provided

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

1. `ArrayQueue.java`

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

2. `ArrayStack.java`

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

3. `LinkedNode.java`

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

4. `LinkedQueue.java`

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

5. `LinkedStack.java`

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

6. `QueueInterface.java`

   This is one of the interfaces you will implement. All instructions for what the methods should do are in the javadocs. **Do not alter this file.**

7. `StackInterface.java`

   This is one of the interfaces you will implement. All instructions for what the methods should do are in the javadocs. **Do not alter this file.**

8. `StacksQueuesStudentTests.java`

   This is the test class that contains a set of tests covering the basic operations of your implementations. 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. `ArrayQueue.java`
2. `ArrayStack.java`
3. `LinkedQueue.java`
4. `LinkedStack.java`