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, instance/global variables, 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 singly-linked list
2. A stack backed by an array
3. A queue backed by a singly-linked list with a tail reference
4. A queue backed by an array

A queue is a first-in, first-out (FIFO) data structure; the first item inserted is the first item to be removed. A stack is a last-in, first-out (LIFO) data structure; the last item to be inserted is the first item to be removed.

All of your data structures should meet the requirements stated in each of the method javadocs. The array classes have a provided `INITIAL_CAPACITY`; 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 and correspond to what was taught in class.

Circular Arrays

The backing array in your `ArrayQueue` implementation must behave circularly. For this assignment, the front variable in `ArrayQueue` should represent the index that holds the next element to dequeue. **Failure to adhere to this may result in large penalties.**

For enqueuing, add to the back of the queue. To access the back of the queue, you can add the size to the front variable to get the next index to add to, though you will have to account for the circular behavior yourself.

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 DURING A DEQUEUE.** 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.

Additionally, after removing the last element in the queue, reset the front variable to the beginning of the array, index 0.

Grading

Here is the grading breakdown for the assignment. There are various deductions not listed that are incurred when breaking the rules listed in this PDF, and in other various circumstances.

Methods:	
ArrayQueue enqueue	11pts
ArrayQueue dequeue	9pts
ArrayQueue peek	3pts
LinkedQueue enqueue	8pts
LinkedQueue dequeue	7pts
LinkedQueue peek	3pts
ArrayStack push	9pts
ArrayStack pop	8pts
ArrayStack peek	3pts
LinkedStack push	6pts
LinkedStack pop	5pts
LinkedStack peek	3pts
Other:	
Checkstyle	10pts
Efficiency	15pts
Total:	100pts

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 do they 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 Canvas under Files, 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 on Canvas, under Files, 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 Tim Aveni (tja@gatech.edu) with the subject header of "[CS 1332] 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.

Vulgar/Obscene Language

Any submission that contains profanity, vulgar, or obscene language will receive an automatic zero on the assignment. This policy applies not only to comments/javadocs but also things like variable names.

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:

Bad: `throw new IndexOutOfBoundsException(‘‘Index is out of bounds.’’);`

Good: `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.

- `package`
- `System.arraycopy()`
- `clone()`
- `assert()`
- `Arrays` class

- `Array` class
- `Thread` class
- `Collections` class
- `Collection.toArray()`
- Reflection APIs
- Inner or nested classes
- Lambda Expressions
- Method References (using the :: operator to obtain a reference to a method)

If you're not sure on whether you can use something, and it's not mentioned here or anywhere else in the homework files, just ask.

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. There are several, but we've noted the ones to edit.

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. `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, and that *only* the following file(s) are present. If you make resubmit, make sure only one copy of the file is present in the submission.

After submitting, double check to make sure it has been submitted on Canvas and then download your uploaded files to a new folder, copy over the support files, 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`