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 17**. 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. For example, do not add `throws` to the method headers since they are not necessary.
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 `ArrayList` 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 are expected to implement all of the methods in this homework. Each unimplemented method will result in a deduction.
8. You must submit your source code, the `.java` files, not the compiled `.class` files.
9. Only the last submission will be graded. Make sure your last submission has **all** required files. Resubmitting will void all previous submissions.
10. 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.

Collaboration Policy

Every student is expected to read, understand and abide by the Georgia Tech Academic Honor Code.

When working on homework assignments, you **may not** directly copy code from any source (other than your own past submissions). You are welcome to collaborate with peers and consult external resources, but you **must** personally write all of the code you submit. **You must list, at the top of each file in your submission, every student with whom you collaborated and every resource you consulted while completing the assignment**.

You may not directly share any files containing assignment code with other students or post your code publicly online. If you wish to store your code online in a personal private repository, you can use Github Enterprise to do this for free.

The only code you may share is JUnit test code on a pinned post on the official course Piazza. Use JUnits from other students at your own risk; **we do not endorse them**. See each assignment's PDF for more details. If you share JUnits, they **must** be shared on the site specified in the Piazza post, and not anywhere else (including a personal GitHub account).

Violators of the collaboration policy for this course will be turned into the Office of Student Integrity.

Style and Formatting

It is important that your code is not only functional, but written clearly and with good programming style. Your code will be checked against a style checker. The style checker is provided to you, and is located on Canvas. It can be found under Files, along with instructions on how to use it. A point is deducted for every style error that occurs. If there is a discrepancy between what you wrote in accordance with good style and the style checker, then address your concerns with the Head TA.

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.

JUnits

We have provided a **very basic** set of tests for your code. 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.

Stacks and Queues

You are to code the following:

1. A `Stack` backed by a singly-linked list without a tail reference.
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 `Stack` is a last-in, first-out (LIFO) data structure; the last item to be inserted is the first item to be removed. A `Queue` is a first-in, first-out (FIFO) data structure; the first item inserted is the first item to be removed.

All your data structures must follow the requirements stated in the javadocs of each method you must implement.

Note that the linked versions of the data structures should **NOT** be circular.

Capacity

The starting capacity of the `ArrayStack` and `ArrayQueue` should be the constant `INITIAL_CAPACITY` defined in the `.java` files. Reference the constant as-is. Do **not** simply copy the value of the constant. Do **not** change the constant. If, while adding an element, the backing array does not have enough space, you should regrow the backing array to **twice** its old capacity. Do **not** resize the backing array when removing elements.

Removing

With both array classes, when elements are deleted from the array, the index the element was removed from should be set to `null`. **All unused positions in the backing array must be set to `null`.**

Circular Arrays

The backing array in your `ArrayQueue` implementation **must behave circularly**. This means the front variable might wraparound to the beginning when you remove to take advantage of empty space while maintaining $O(1)$ efficiency for all operations (though, in particular, adding operations should be $O(1)$ amortized).

For this assignment, the front variable in `ArrayQueue` should represent the index that holds the next element to dequeue. **Failure to follow this convention will result in major loss of points.**

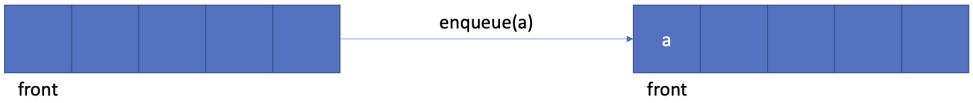
For enqueuing, add to the back of the queue. To access the back of the queue, you can add the size to the front variable to get the next index to add to, though you will have to account for the circular behavior yourself. If there are empty spaces at the front of the array, the back of the queue should wrap around to the front of the array and make use of those spaces.

For dequeuing, you should simply treat the next index in the array as the new front. **Do not shift any elements during a remove.**

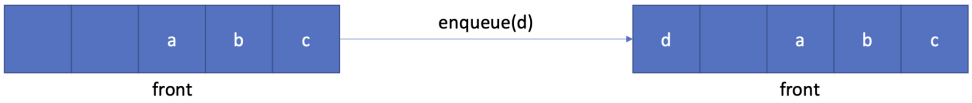
When resizing the backing array, "unwrap" the data. Realign the front of the queue with the beginning of the new array during the transfer. After unwrapping, the front variable of the queue is once again at index 0.

Additionally, after removing the last element in the queue, move the front variable like you normally would. Do **not** explicitly reset it to 0. This effectively means that going from size 1 to size 0 should not be a special case for your code.
The examples below demonstrate what the queue should look like at various states.

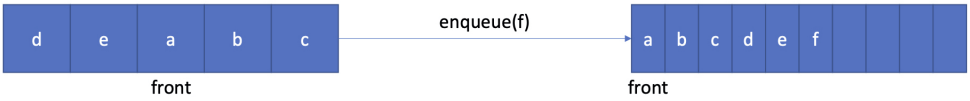
In the example below, the queue begins empty (initial state). An element is added.



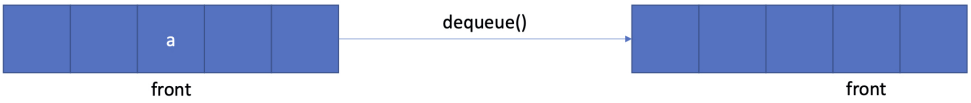
In the example below, adding to the back causes the newly added element to wrap around to the front.



In the example below, adding another element causes the queue to resize. The array capacity is doubled and the front element moves to index 0.



In the example below, the last element of the queue is removed, but front moves as expected. The front variable is not explicitly set to 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:	
ArrayStack constructor	1pts
ArrayStack push	7pts
ArrayStack pop	6pts
ArrayStack peek	4pts
LinkedStack push	6pts
LinkedStack pop	7pts
LinkedStack peek	4pts
ArrayQueue constructor	1pts
ArrayQueue enqueue	10pts
ArrayQueue dequeue	8pts
ArrayQueue peek	4pts
LinkedQueue enqueue	6pts
LinkedQueue dequeue	7pts
LinkedQueue peek	4pts
Other:	
Checkstyle	10pts
Efficiency	15pts
Total:	100pts

Provided

The following file(s) have been provided to you. There are several, but we've noted the ones to edit.

1. `ArrayStack.java`

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

2. `LinkedStack.java`

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

3. `ArrayQueue.java`

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

4. `LinkedQueue.java`

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

5. `LinkedNode.java`

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

6. `StackStudentTest.java`

   This is the test class that contains a set of tests covering the basic operations on the `Stack` 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.**

7. `QueueStudentTest.java`

   This is the test class that contains a set of tests covering the basic operations on the `Queue` 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). Make sure all file(s) listed below are in each submission, as only the last submission will be graded. Make sure the filename(s) matches the filename(s) below, and that *only* the following file(s) are present. If there are multiple files, do not zip up the files before submitting; submit them all as separate files.

Once submitted, double check that it has uploaded properly on Gradescope. To do this, download your uploaded file(s) to a new folder, copy over the support file(s), recompile, and run. It is your sole responsibility to re-test your submission and discover editing oddities, upload issues, etc.

1. `ArrayStack.java`
2. `LinkedStack.java`
3. `ArrayQueue.java`
4. `LinkedQueue.java`