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.

Circular Singly-Linked List

You are to code a circular singly-linked list with a head reference. A linked list is a collection of nodes, each having a data item and a reference pointing to the next node. Since it must be circular, the next reference for the last node in this list will point to the head node. Do **not** use a phantom node to represent the start or end of your list. A phantom or sentinel node is a node that does not store data held by the list and is used solely to indicate the start or end of a linked list. If your list contains n elements, then it should contain exactly n nodes.

Your linked list implementation will implement the `LinkedListInterface` provided. It will use the default constructor (the one with no parameter) which is automatically provided by Java. Do not write your own constructor.

Nodes

The linked list consists of nodes. A class `LinkedListNode` is provided to you. `LinkedListNode` has setter and getter methods to access and mutate the structure of the nodes.

Adding

You will implement three `add()` methods. One will add to the front, one will add to the back, and one will add anywhere in the list. See the interface for more details.

Removing

Removing, just like adding, can be done from the front, the back, or anywhere in your linked list. In addition, you will also be coding a method to remove the last instance of a piece of data. When removing from the front, the first node should be removed in such a way that the last node points to the new front of the list (mind the efficiency!). When removing from the back, the last node should be removed and

have the new last node point to the head. When removing from the middle, the previous node of the removed node should point to the next node of the removed node. Make sure that you set any remaining pointers to the deleted nodes to `null` since in order for the node to be garbage collected, there cannot be any way to access the node. See the interface for more details.

Equality

There are two ways of defining objects as equal: reference equality and value equality.

Reference equality is used when using the `==` operator. If two objects are equal by reference equality, that means that they have the exact same memory locations. For example, say we have a Person object with a name and id field. If you're using reference equality, two Person objects won't be considered equal unless they have the exact same memory location (are the exact same object), even if they have the same name and id.

Value equality is used when using the `.equals()` method. Here, the definition of equality is custom made for the object. For example, in that Person example above, we may want two objects to be considered equal if they have the same name and id.

Keep in mind which makes more sense to use while you are coding.

Grading

Here is the grading breakdown for the assignment:

Methods:	
addAtIndex	10pts
addToFront	5pts
addToBack	5pts
removeAtIndex	10pts
removeFromFront	5pts
removeFromBack	5pts
removeLastOccurrence	10pts
get	10pts
toArray	6pts
clear	5pts
isEmpty	4pts
Other:	
Checkstyle	10pts
Efficiency	15pts
Total:	100pts

Keep in mind that some functions are dependent on others to work, such as remove methods requiring the add methods to work. Also, the size function is used many times throughout the tests, so if the size isn't updated correctly, many tests can fail.

A note on JUnits

We have provided a **very basic** set of tests for your code, in `LinkedListStudentTests.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 Raymond Ortiz (rortiz9@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:

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.

- `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
- Lambda Expressions
- Method References

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 you will only edit one of them.

1. `LinkedListInterface.java`

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

2. `SinglyLinkedList.java`

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

3. `LinkedListNode.java`

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

4. `LinkedListStudentTests.java`

   This is the test class that contains a set of tests covering the basic operations on the `SinglyLinkedList` class. 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). Failure to submit all required files may result in a 0 for the assignment. 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. `SinglyLinkedList.java`