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.
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 when implementing an interface.
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.

Circular Doubly Linked List

You are to code a circular doubly-linked list with a head reference. A doubly linked list is collection of nodes, each having a data item and a reference pointing to the next node and a reference to the previous node. In the case of a circular linked list your last node next's reference will point to the head node rather than null as it would in a standard linked list. Likewise, the head node's previous reference will point to the last node in the list.

Your linked list implementation will implement the `LinkedListInterface` provided. It will implement two constructors. One constructor doesn't take in anything and creates an empty list. The other constructor takes in an array of items to add to the list.

Nodes

The linked list consists of nodes. A class `LinkedListNode` is provided to you. `LinkedListNode` has four methods to access and set the next and previous nodes, as well as an accessor method for the node's data.

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, a specific value from the list can be removed. When removing from the front, the head reference should point to the second node in the list. When removing from the back, the last node should be removed. 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 pointers to the deleted nodes to `null`. See the interface for more details.

A note on JUnits

We have provided a basic set of tests for your code, in `CircularLinkedListTestStudent.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 Jonathan Jemson (jonathanjemson@gatech.edu) with the subject header of "CheckStyle XML".

Javadocs

Javadoc any helper methods you create in a style similar to the Javadocs for the methods in the interface. If a method is overridden or implemented from a superclass or an interface, you may use `@Override` instead of writing Javadocs.

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.

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 List<Integer>()` instead of `new List()`. 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
- Reflection APIs

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. If you use 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. `CircularDoublyLinkedList.java` This is the class in which you will actually implement the interface. Feel free to add private helpers but **do not add any new public methods or instance variables**.
3. `LinkedListNode.java` This class represents a single node in the linked list. It encapsulates the data, next reference, and previous reference. **Do not alter this file.**
4. `CircularLinkedListTestStudent.java` This is the test class that contains a set of tests covering the basic operations on the `CircularDoublyLinkedList`. 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. `CircularDoublyLinkedList.java`

You may attach each file individually or submit them in a zip archive.