

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.

Doubly-Linked List

You are to code a non-circular doubly-linked list with head and tail references. A linked list is a collection of nodes, each having a data item and references to the previous and next nodes. The previous reference for the head/first node will point to null. The next reference for the tail/last node in this list will point to null. 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 use the default constructor (the one with no parameter) which is automatically provided by Java. Do not write your own constructor. Specifications for efficiency and what the method should do are provided in the javadocs.

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.

Traversing

When the `LinkedList` is doubly-linked like in this assignment, you are able to traverse the list from both ends depending on which makes more sense. For example, if you know the index you want is close to the end of the list, you'd start traversing from the tail and go backwards; the opposite applies if the index is close to the beginning of the list. **You should make this optimization wherever it applies.**

Adding

You will implement three `add()` methods. One will add to the front, one will add to the back, and one will add at a specific index. See the javadocs for more details.

Removing

Removing, just like adding, can be done from the front, the back, or from a specific index. Make sure that there is no longer any way to access the removed node so that the node will be garbage collected. See the javadocs 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. **You will want to use value equality in most cases in this course when comparing objects. Notable cases where you'd use reference equality include checking for null or comparing primitives (in this case, it's just the `==` operator being overloaded).**

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:	
<code>addAtIndex</code>	10pts
<code>addToFront</code>	5pts
<code>addToBack</code>	5pts
<code>removeAtIndex</code>	10pts
<code>removeFromFront</code>	5pts
<code>removeFromBack</code>	7pts
<code>lastOccurrence</code>	8pts
<code>get</code>	10pts
<code>toArray</code>	6pts
<code>clear</code>	5pts
<code>isEmpty</code>	4pts
Other:	
Checkstyle	10pts
Efficiency	15pts
Total:	100pts

Keep in mind that add functions are necessary to test other functions, so if an add doesn't work, remove tests might fail as the items to be removed were not added correctly. Additionally, the size function is used many times throughout the tests, so if the size isn't updated correctly or the method itself doesn't work, 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 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. `DoublyLinkedList.java`

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

2. `LinkedListNode.java`

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

3. `LinkedListStudentTests.java`

This is the test class that contains a set of tests covering the basic operations on the `DoublyLinkedList` 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). 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. `DoublyLinkedList.java`