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, 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.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.

Singly Linked List

You are to code a singly-linked list with a head and tail reference as well as a couple of operation methods. A linked list is a collection of nodes, each having a data item and a reference pointing to a next node (and, in the case of a doubly-linked list, a reference to the previous node). The next reference for the last node in this list would be `null`. Do **not** use a phantom node to represent the head or tail of your list. The head will either be null or be the first node in the list and tail will either be null or the last node in the list.

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 for you. `LinkedListNode` has `data` of type `T` and a `LinkedListNode` reference called `next`. See `LinkedListNode.java` for more information.

Adding

A regular add to a Linked List will append a new node to the end of the list with the new data to add. In this homework you will implement an `add` which will perform the previously stated action. You will also implement `addToFront()` which will add a node before the head and `addToIndex()` which will add a node to a specific index respectively. There is also an `addAll()` method which will add all elements in a `Collection`. See the interface for more details. **Remember to update head and tail references.**

Removing

A regular remove to a Linked List will remove the node at the end of the list and update the tail. In this homework you will implement `remove()` which will perform the previously stated action. You will also implement `removeFromFront()` which will remove the current head and set the next node as head and `removeAtIndex()` which will set the previous node's next to the current nodes next (hopping over the node). See the interface for more details. **Make sure that you set any references to the deleted nodes to `null`.**

Other Methods

See the interface for more details.

Homework Notes

- There may be duplicates in the list.
- Code with efficiency! Many of these methods can be completed with one pass through of the list. (Don't use crazy nested loops)
- If you have a reference to a node, setting the reference to null may not remove the node from the list.
  For example:
  ```
  LinkedListNode<T> curr = head;
  curr = null;
  ```
  `head` will still be pointing to the head.

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 (when it comes up).

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 Joonho Kim (jkim844@gatech.edu) with the subject header of "CheckStyle XML".

Javadocs

Javadoc any helper methods you create in a style similar to the existing Javadocs (remember to keep helper methods private). 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. The name of the exception itself is not a good message.

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 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. If you use these, we will take off points.

- `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

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.

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 the `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). 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`