# CSC 151: Project 3 – Stay in Line
# Due Thursday, May 14, 2020 at 10:00pm

## Objectives

- Reinforce how the behavior of an object is completely separate from its implementation (i.e. **information hiding**)

- Practice with the linked list data structure

You're not done with your Sequence class yet. If you look closely at the public interface (the prototypes of all of the public methods) of the Sequence class that you made for Project 2, you'll notice that none of them reveal that arrays are being used for the underlying implementation. That's the point. Despite the fundamental differences between arrays and sequences (arrays can't expand and contract, sequences can only be accessed at the "current" position), a programmer wishing to build upon the Sequence class doesn't need to know about arrays in order to use the class you made. That means that as long as you keep the same public interface for Sequence, you can alter/upgrade the underlying implementation and not affect in any way anybody else's code that happens to use your Sequence class. That's called **information hiding**, and it's a hallmark of good design. You'll see this at work as you switch from an array-based Sequence to a linked-list based one.

## Your Mission

Redo Project 2 in its entirety, but this time, use a linked list to hold the data members of a sequence instead of an array. The prototypes of all public methods need to be **exactly the same** as in Project 2. Here are the details:

1. You need to write two new classes: a ListNode class to define a single data node and a LinkedList class to model the list itself. Only instance variables in your ListNode class are allowed to be public. Remember that methods in your LinkedList class should be usable by anything that needs a general-purpose series of Strings. If a method in LinkedList is too specific to this project, then it's not reusable. For example, if there's a one-to-one correspondence between the methods in Sequence and the methods in LinkedList (so there's a `getCurrent` method, an `advance` method, etc. in LinkedList) you're doing something wrong. Sequences have a "current" element. Linked lists do not. While you could define a `getCurrent` method in LinkedList, doing so only makes sense for this project, unlike, say, a `search` method, which is something you'd like to do with any container.

2. Test your LinkedList class by writing a *LinkedListTester* class that uses JUnit testing. **Write the tests first, and test each LinkedList method after you write it. Don't write all the tests at the end.**

3. Rewrite your Sequence class. Be sure to re-examine your instance variables. Besides changing the array to a linked list, should other instance variables be modified? Do any

need to be added or removed? What is the invariant now? (And include it in the class description!)

4. Test your new implementation of Sequence. You should be able to reuse all of your own tests from Project 2 because the expected behavior has not changed. You should also upload your Sequence, LinkedList, and ListNode classes to Gradescope for testing.

Notice that I'm not telling you exactly what methods should be included in your LinkedList or ListNode classes. That's your job. Remember, each method in LinkedList should do a single task that manipulates the list as a whole (insertion, deletion, etc.) As always, you are not allowed to use Java's **LinkedList** class or any of Java's built-in container classes like **Vector, Arrays,** or **ArrayList** in your program.

Be modular. The LinkedList class we've seen contains methods for inserting a node into the list. This makes the class reusable by *any* class needing to insert nodes into a linked list. Therefore, when your Sequence class needs to add a new element, it should do so by invoking a method in the LinkedList class instead of having the `addBefore` or `addAfter` method access the linked list nodes directly. Remember, Sequence should not know about ListNodes!

As we've discussed in class, many of the methods are easier to write if you use the more "basic" methods as helpers. For example, if you used your `ensureCapacity` method as a helper method in `addBefore` and `addAfter`, you'll have less to change in `addBefore` and `addAfter` to get them working once you change your implementation of `ensureCapacity`. As you do this project, take note where you use those "basic" methods inside other methods. You should find that those methods that mainly call other methods to get their job done should require very little modification on your part. That's the idea behind reusability, modularity, and information hiding. If a given method's *behavior* doesn't change, then code that uses it will still work, regardless of how that method was *implemented*.

## A word about Size and Capacity

It's important to remember that a computer program is just a model for some system in the real world. In this case, your Sequence class is a model for a container, and all containers have a size and capacity. The distinction is easy to see when the sequence is array-based, but the line becomes blurred with a linked-list-based sequence. If the capacity is 10 but the size is 3, it does not mean you have 7 "empty" nodes. But if there are no "empty" nodes, does that mean you can get rid of methods like `ensureCapacity`? NO! If you did, any code that built upon your Sequence and used that method would now be broken. You still need to keep the concept of capacity around, even if your implementation doesn't, because we're trying to model a container here, and all containers have capacity. So you'll need a variable to keep track of capacity, even though you're not "using" it to manage the linked list. Come talk to me if this doesn't make sense!

## Grading

This project is worth 50 points divided up this way:

- 20 points for the output. Your score will come directly from Gradescope.
- 30 points for program design. This includes:
    - thorough testing of Sequence and LinkedList
    - good understandability
    - making complex methods reusable by using helper methods
    - good decisions about what methods should be included in what classes

As always, practice good programming skills:

- Use comments that reveal the purpose of your code. *Use Javadoc format for all comments. Include the (new) invariant in the class description for Sequence.*
- Test each method individually instead of trying to get the code to work all at once
- Use good debugging skills: tracing, echo printing, and checkpointing
- Be modular by using private auxiliary methods to do subtasks.

Please upload an electronic copy of your project to Nexus. Don't forget the SINGLE pdf file, comprised of all the code. SHARE that document with me on Google Drive as well as including it in your upload.

Having trouble? Don't wait until the last minute!

## Gentle Reminder

Programming assignments, like homework assignments, are *individual* projects. I encourage you to talk to others about the general nature of the project and ideas about how to pursue it. However, the technical work, the writing, and the inspiration behind these must be substantially your own. If any person besides you contributes in any way to the project, you must credit their work on your project. Similarly, if you include information that you have gleaned from other published sources, you must cite them as references. Looking at, and/or copying, other people's programs or written work is inappropriate, and will be considered cheating.