Data Structures (CSC 151) **Winter 2023**
Project 3: Stay in Line **Due: Tuesday, 02/14/2023 (11:59pm)**

> Unless specifically told otherwise, you may not import anything into your projects. That means you may not use any built-in data structures like ArrayList or Vector. This rule applies to all projects in this course.

Note: You are permitted to use the java.lang.Object class versions of primitive data types in this and future projects (e.g. Integer). For this project you are also allowed to use the Iterator and Iterable interfaces, which can be imported from java.util.Iterator. None of these are required to complete the project, but you are permitted to use them if you find it useful to do so.

Learning Objectives:

1. Reinforce how the behavior of an object is completely separate from its implementation
    a. i.e. information hiding
2. Practice with the linked list data structure

Reminder: Programming assignments are *individual* projects. I encourage you to talk to others about the general nature of the project and ways to approach it; however, the technical work (the code writing, and inspiration behind it) 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 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.

1 Introduction

We're not done with the Sequence class yet. If you look closely at the public interface (the prototypes of all 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 change size, sequences can only be directly 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. This means that as long as you keep the same public interface for Sequence, you can alter/upgrade the underlying implementation without affecting anyone else's code that just so happens to use your Sequence class. This is called information hiding and is a common feature in good program design. You'll see this in practice as you switch from an array-based Sequence to a linked-list-based one.

2 Your Mission

Redo Project 2 in its entirety, but this time you'll 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.

3 The Details

To get started, download the Project 3 starter code from Nexus.

1. You'll need to write two entirely new classes: *ListNode* to define a single data node and *LinkedList* to model the list itself. Remember that instance variables in your ListNode class are allowed to be public iff (if and only if) your LinkedList class never exposes ListNode objects to the public (i.e. how it should be implemented).

   Your job is to create a general-purpose linked list that can hold Strings. 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 (e.g. a getCurrent method, an advance method, etc. in LinkedList) you're doing it 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 particular project, unlike, for example, 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 reexamine 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 are the invariants now? Include a description of your invariants as a comment right above the instance variable declarations.

   Note: If you notice ways in which you can improve the implementation of your Sequence class, please go ahead and make these improvements, even if they don't have anything directly to do with the switch from an array-based to a linked-list-based implementation.

   In a separate document or on a piece of paper, keep track of the main changes you're making. Once you're done with this project, I'll ask you to submit a reflection on these changes.

4. Test your new implementation of *Sequence*. You should be able to reuse all of your tests from Project 2, because the expected behavior has not changed. If your set of tests for Project 2 had gaps, here is your chance to fill these gaps and improve your test suite.

5. Reflect on the modifications you had to make to your implementation of the Sequence class. This will be due on **Friday 2/17**. I'm letting you know now so that you can keep the necessary notes.
   a. Describe the main changes you had to make. Besides switching the array instance variable for a linked list, what changes, if any, did you make to the remaining instance variables and why?
   b. How did the switch impact your method implementations? How many methods did you have to modify? Given an example of a method where the design and implementation you

chose for Project 2 made the switch easy to implement. Give an example of a method where the switch would have been easier if you'd implemented it differently in Project 2.

c. For each public method of the Sequence class, compare the runtime of your array-based implementation to the runtime of your linked-list-based implementation (make sure to use proper big-O notation for this).

Although I have provided some starter code, notice that I'm not telling you exactly which other methods should be included in your LinkedList or ListNode classes. Figuring out which methods you can, should, or must implement to complete the project successfully is your job. Remember that each method in LinkedList should do a single task that manipulates the list as a whole (insertion, delection, etc.). As always, you're 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.

Keep your code modular. The partial implementations of a LinkedList that we've seen in class 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 ListNode objects directly. Sequence should not even be aware that the ListNode class exists!

When working on Project 2, you may have noticed that many of the methods are easier to write if you use more "basic" methods as helpers. For example, if you used ensureCapacity 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 work on this project, take note of where you use those "basic" methods inside other methods. You should find that the 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 note 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 gets blurred with a linked-list-based sequence. If the capacity is 10, but the size is 3, it doesn't mean that 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, then any code that built upon your Sequence and used that method will break. 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.
Please come and talk to me if this doesn't make sense to you.

4 Submission

Be sure to include the honor code affirmation in the comments of one of the classes:
I affirm that I have carried out the attached academic endeavors with full academic honesty, in accordance with the Union College Honor Code and the course syllabus.

Before you submit, check to make sure that your code satisfies the following requirements:

1. All files are properly commented
2. All files are formatted neatly and consistently
3. Your code has been cleaned up. Your code does not contain snippets that don't contribute to the purpose of the program (e.g. commented out code from earlier attempts or comments that aren't helpful).
4. Your variable and method names are informative.
5. There are no magic numbers (or other magic values).
6. Your code practices good information hiding
7. Your code makes use of existing public methods or private helper methods to make complex tasks more modular and to minimize the number of methods that access instance variables directly.

Once you've checked all these requirements, submit the following files to *Project 3: Sequence Again* on Gradescope:

- `Sequence.java`
- `SequenceTest.java`
- `ListNode.java`
- `LinkedList.java`
- `LinkedListTest.java`

5 Grading guidelines

- Correctness: programs do what the problems require. (Based on Gradescope tests)
- Testing: The evidence submitted shows that the program was tested thoroughly. The tests run every line of code at least once. Different input scenarios, and especially border cases, were tested.
- Documentation: Every public method and class is documented in the appropriate format (Javadoc) and provides the necessary information. The information provided would enable a user to use the class/method effectively in their code.
- Programming techniques: Programming constructions have been used appropriately in a way that makes the code efficient and easy to read and maintain. If the project required use of a specific technique or algorithm, this has been correctly implemented. For this project, I'll be paying special attention to good information hiding, modularity, robustness (e.g. no errors when someone uses removeCurrent on an empty Sequence), and that the stated invariants are coherently implemented.
- Style: The program is written and formatted to ensure readability. For example, naming conventions are obeyed, whitespace (blanks, line breaks, indentation) is used to help structure the code, formatting is consistent, and the code is well organized.