Data Structures (CSC 151)
Fall 2024
Project 3: Stay in Line **Due: Tuesday, 10/15/2024**

Learning Objectives:

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

Due dates:

- Implementation: Tuesday, 10/15/2024
- Reflection: Friday, 10/18/2024

1 Introduction

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.

2 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 signatures of all public methods in the Sequence class need to be exactly the same as in Project 2.

3 The Details

1. Start from the same starter code as Project 2.
2. *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.* Remember: instance variables in your ListNode class are allowed to be public, but your LinkedList class has to make sure that it never exposes ListNode objects to the public.

   Your job is to create a general-purpose generic linked list (i.e. `public class LinkedList<E>`). Your `LinkedList` class should implement the following public methods:

- A default constructor that creates an empty list.
- `int size()`
  which returns the number of elements in the linked list (i.e. the size of the linked list)
- `void addAtHead(E toAdd)`
  which adds a new item to the front of the list
- `void addAt(E toAdd, int index)`
  which inserts a new item into the list at the given index. If the index is greater than the size of the list, the method throws an `IndexOutOfBoundsException`.
- `E get(int index)`
  which returns the data that is contained in the node with the given index. If the index is not a valid index, the method throws an `IndexOutOfBoundsException`.
- `void set(String value, int index)`
  which sets the data of the node with the given index to the String value that is provided. If the index is not a valid index, the method throws an `IndexOutOfBoundsException`.
- `E remove(int index)`
  which removes the node with the given index from the linked list and returns the data in the removed node. If the index is not a valid index, the method throws an `IndexOutOfBoundsException`.
- `String toString()`
  which returns a string representation of the linked list. An empty linked list should have the string representation `"{}"`. A linked list containing the Strings `"A"` and `"B"` (in that order) should have the string representation `"{A, B"}`.

You may add additional public methods to your linked list implementation. However, remember that the task is to implement a *general purpose* linked list. If a method in LinkedList is too specific to this project, then it's not general purpose. 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 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 project, unlike, say, a search method, which is something you'd like to do with any container.

3. *Test your LinkedList class by writing a LinkedListTest 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.

4. *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 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 have not directly to do with the switch from an array based implementation to a linked list based implementation.

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

5. *Write tests your new implementation of Sequence in a class called SequenceTest.* You should be able to reuse all of your own 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.

   You should also upload your Sequence, LinkedList, and ListNode classes to Gradescope for testing.

6. *Reflect on the modifications you had to make to your implementation of the Sequence class.* **This will be due on Friday, 10/18.** I am letting you know now so that you can keep the necessary notes.

   (a) Describe the main changes you had to make. E.g. besides changing switching out the array instance variable against 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? Give 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 to implement had you 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.

Additional notes *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 partial impplementation of a 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 you noticed when working on Project 2, 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 see me if this doesn't make sense!

4 Submit

Be sure to include the honor code affirmation in the documentation for the Sequence and LinkedList 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.*

Submit the following files to Gradescope:

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

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

1. Are all of your files properly documented?
2. Are the invariants for the Sequence class explained in a comment?
3. Are your files formatted neatly and consistently?
4. Did you clean up your code once you got it to work? It should not contain any code snippets that don't contribute to the purpose of the program, commented out code from earlier attempts, or unnecessary comments.
5. Do you use variable and method names that are informative?
6. Did you get rid of all magic numbers?
7. Does your code practice good information hiding?
8. Does your code make use of already existing public methods or private helper methods to modularize complex tasks and to minimize the number of methods that have to access instance variables?

Finally, submit your files to Gradescope.

5 Gentle Reminder

Programming 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.

6 Grading guidelines

- Correctness: Your program does what the problem specifications require. (Based on Gradescope tests.)
- Testing: The evidence submitted shows that the program was tested thoroughly. The tests run every line of the 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. The invariants for the instance variables in your implementation are documented.
- Programming techniques: Programming constructions have been used appropriately in such a way that make the code efficient and easy to read and maintain. If the project required the use of a specific technique or algorithm, this has been correctly implemented. For this project, I will particularly look for good information hiding and modularity, robustness (e.g. no error when some uses `removeCurrent` on an empty Sequence), and that the stated invariant is 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 help structure the code, formatting is consistent and the code is well organized.