Data Structures (CSC 151)
Fall 2024
Project 2: Get in Line **Due: Tuesday, 10/01/2024**

Unless explicitly stated 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.

Learning Objectives:

1. To make your own abstract data type (ADT)
2. To learn how to separate the behavior of an object from its implementation
3. To practice thorough testing of your code

1 Introduction

We are now coming to the meat of the course: analyzing how language-independent data structures can help you create good code. One of the first data structures you'll be getting chummy with is called a *container* or *collection.* As its name implies, a container is simply a storage place for a potentially large number of items. One can add items to the container, remove items, search a container for a particular item, and many other operations.

You'll be building a container called a **Sequence.** Like an array, a sequence stores objects that are all of the same type. Unlike an array, a sequence's contents are not accessible by index; instead there is an element marked "current" that can be accessed, along with methods that allow you to move the "current" mark so that you can access other items by iterating through the items in a sequence. In addition, a sequence can grow and shrink to accommodate the insertion and deletion of items.

2 Your Mission

Your assignment is to create the public `Sequence` class that will allow you to construct and use sequence containers. Your Sequence class will be limited to storing Strings. I want you to implement your Sequence class using arrays. That is, each instance of a Sequence object should have its own array of Strings for storing the data contained within. (The Sequence container is similar to the Bag container we discussed in class. So you should review the implementation of the Bag ADT.)

Before you start to program anything, think carefully about the way you want to design your instance variables and formulate an invariant for your implementation of the ADT. Include the text of your invariant as part of your Sequence class Javadocs. Place it after the class description.

3 The Details

To get started, download the starter code from Nexus. You will complete the file named `Sequence.java`. Notice that the class Sequence is defined to be in the package `proj2`. Do not change the package name.

The starter code lists all the public methods you need to write. You must have only these public methods, and you may not change any of the prototypes listed, including the names of the methods. Of course, you may write as many private helper methods as you wish. In addition, your toString method should return a string that uses exactly the format specified in the comments.

As always, you are not allowed to use any of Java's built-in containers, like ArrayList or Vector. It is your job to figure out what instance variables are appropriate.

PITFALL ALERT: Avoid all side effects. For example, calling toString should not alter the current index! I've left notes in the Javadocs of places where you need to be extra careful about this, but you should avoid them everywhere unless they are part of the method's job.

4 Testing is part of the job

You should create one other class called `SequenceTest` in your project for JUnit testing. Each Sequence method will require several JUnit testing methods to make sure its behavior matches what it says in the Javadocs. The goal is to make sure your Sequence methods work under all conditions. This means that your Sequence should model the "real-world" concept of a container as closely as possible. For example, sequences have no limit as to how many items they can hold. That should make you think of 1 good test already: you should make sure that if you insert a new item to a full sequence using `addBefore` or `addAfter`, your program shouldn't crash or print an error message (that's poor design). It should simply make more room. Remember, you can't extend an array to make more room. You'll have to create a new array and copy the elements yourself. Do NOT use built-in methods like System.arraycopy to do this!

Good testing is an important part of good programming. Remember, you should write the tests before you write the Sequence code, which is a practice that professionals use. That way, your tests are not influenced by the way you have written Sequence. As before, my own tests on Gradescope are another means of testing, but you should write your own before trying mine.

5 Submit

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.*

Submit the following files to Gradescope:

- `Sequence.java`
- `SequenceTest.java`

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

1. Are all of your files properly commented?
2. Are your files formatted neatly and consistently?
3. 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.

4. Do you use variable and method names that are informative?
5. Did you get rid of all magic numbers?

Finally, submit your files to Gradescope (*Project 2: Sequence*).

6 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.

7 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 ADT's invariant(s) 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.