CSCI 1933 Project 3

List implementation: ArrayLists vs. LinkedLists

Due Date: Wednesday, April 1, 11:59 pm

Instructions

Please read and understand these expectations thoroughly. Failure to follow these instructions could negatively impact your grade. Rules detailed in the course syllabus also apply but will not necessarily be repeated here.

- **Due:** The project is due on **Wednesday, April 1** by **11:59 PM**.
- **Identification:** Place you and your partner's x500 in a comment in all files you submit. For example, `//Written by shino012 and hoang159`.
- **Submission:** Submit a `zip` or `tar` archive on Canvas containing all your `java` files. You are allowed to change or modify your submission, so submit early and often, and *verify that all your files are in the submission.*

  Failure to submit the correct files will result in a score of zero for all missing parts. Late submissions and submissions in an abnormal format (such as `.rar` or `.java`) will be penalized. Only submissions made via Canvas are acceptable.
- **Partners:** You may work alone or with *one* partner. **Failure to tell us who is your partner is indistinguishable from cheating and you will both receive a zero**. Ensure all code shared with your partner is private.
- **Code:** You must use the *EXACT* class and method signatures we ask for. This is because we use a program to evaluate your code (more on this later). Code that doesn't compile will receive a significant penalty. Code should be compatible with Java 8, which is installed on the CSE Labs computers.
- **Questions:** Questions related to the project can be discussed on Canvas in abstract. This relates to programming in Java, understanding the writeup, and topics covered in lecture and labs. **Do not post any code or solutions on the forum**. Do not e-mail the TAs your questions when they can be asked on Canvas.
- **Grading:** Grading will be done by the TAs, so please address grading problems to them *privately.*

  IMPORTANT: You are NOT permitted to use ANY built-in libraries, classes, etc. Double check that you have NO import statements in your code, except for those explicitly permitted.

Code Style

Part of your grade will be decided based on the "code style" demonstrated by your programming. In general, all projects will involve a style component. This should not be intimidating, but it is fundamentally important. The following items represent "good" coding style:

- Use effective comments to document what important variables, functions, and sections of the code are for. In general, the TA should be able to understand your logic through the comments left in the code.

  Try to leave comments as you program, rather than adding them all in at the end. Comments should not feel like arbitrary busy work - they should be written assuming the reader is fluent in Java, yet has no idea how your program works or why you chose certain solutions.
- Use effective and standard indentation.
- Use descriptive names for variables. Use standard Java style for your names: `ClassName`, `functionName`, `variableName` for structures in your code, and `ClassName.java` for the file names.

Try to avoid the following stylistic problems:

- Missing or highly redundant, useless comments. `int a = 5; //Set a to be 5` is not helpful.
- Disorganized and messy files. Poor indentation of braces (`{` and `}`).
- Incoherent variable names. Names such as `m` and `numberOfIndicesToCount` are not useful. The former is too short to be descriptive, while the latter is much too descriptive and redundant.
- Slow functions. While some algorithms are more efficient than others, functions that are aggressively inefficient could be penalized even if they are otherwise correct. In general, functions ought to terminate in under 5 seconds for any reasonable input.

The programming exercises detailed in the following pages will both be evaluated for code style. This will not be strict – for example, one bad indent or one subjective variable name are hardly a problem. However, if your code seems careless or confusing, or if no significant effort was made to document the code, then points will be deducted.

In further projects we will continue to expect a reasonable attempt at documentation and style as detailed in this section. If you are confused about the style guide, please talk with a TA.

1 Introduction

IMPORTANT: This project utilizes interfaces and generics. For brevity, this write-up omits discussions of such topics. For more information, please see a TA, review the lecture notes, or review other related literature (e.g. official Java documentation).

In this project you will implement a list in two different ways: as an ArrayList and as a LinkedList. You will then compare the time complexities of each List method when implemented as an ArrayList and as a LinkedList.

1.1 LinkedList Implementation

The first part of this project will be to implement a linked list. Create a class `LinkedList` that implements all the methods in `List` interface. Recall that to implement the `List` interface and use the generic compatibility with your code, `LinkedList` should have following structure:

```
public class LinkedList<T extends Comparable<T>> implements List<T> {
  ...
}
```

The underlying structure of a linked list is a node. This means you will have an instance variable that is the first node of the list. The provided `Node.java` contains a generic node class that you will use for your linked list implementation*.

Your `LinkedList` class should have a single constructor:

```
public LinkedList() {
  ...
}
```

that initializes the list to an empty list.

Implementation Details

- In addition to the methods described in the `List` interface, the `LinkedList` class should contain a private class variable `isSorted`. This should be initialized to `true` in the constructor (because it is sorted if it has no elements) and updated when the list is sorted, or more elements are added or set. For the purposes of this class, `isSorted` is only true if the list is sorted in ascending order.

*You may implement your linked list as a *headed* list, i.e., the first node in the list is a 'dummy' node and the second node is the first element of the list, or a non-headed list, i.e., the first node is the first element of the list. Depending on how you choose to implement your list, there will be some small nuances.

- Initially and after a call to `clear()`, the size should be zero and your list should be empty.
- In `sort()`, **do not use an array or ArrayList** to sort the elements. You are required to sort the values using only the linked list data structure. You can move nodes or swap values but you cannot use an array to store values while sorting.
- Depending on your implementation, remember that after sorting, the former first node may not be the current first node.

1.2 Array List Implementation

The second part of this project will be to implement an array list. Create a class `ArrayList` that implements all the methods in `List` interface. Recall that to implement the `List` interface and use the generic compatibility with your code, `ArrayList` should have following structure:

```
public class ArrayList<T extends Comparable<T>> implements List<T> {
  ...
}
```

The underlying structure of an array list is (obviously) an array. This means you will have an instance variable that is an array. Since our implementation is generic, the type of this array will be `T[]`. Due to Java's implementation of generics[†], you **CANNOT** simply create a generic array with:

```
T[] a = new T[size];
```

Rather, you have to create a `Comparable` (since `T` extends `Comparable`)[‡] array and *cast* it to an array of type `T`.

```
T[] a = (T[]) new Comparable[size];
```

Your `ArrayList` class should have a single constructor:

```
public ArrayList() {
  ...
}
```

that initializes the underlying array to a length of 2.

Implementation Details

- In addition to the methods described in the `List` interface, the `ArrayList` class should contain a private class variable `isSorted`. This should be initialized to `true` in the constructor (because it is sorted if it has no elements) and updated when the list is sorted, or more

[†] specifically because of type erasure

[‡] had `T` not extended `Comparable`, you would say `T[] a = (T[])new Object[size];`

elements are added or set. For the purposes of this class, `isSorted` is only true if the list is sorted in ascending order.

- When the underlying array becomes full, both `add` methods will automatically add more space by creating a *new* array that is **twice** the length of the original array, copying over everything from the original array to the new array, and finally setting the instance variable to the new array. *Hint: You may find it useful to write a separate* private *method that does the growing and copying*
- When calling either `remove` method, the underlying array should *no longer have that spot.* For example, if the array was `["hi", "bye", "hello", "okay", ...]` and you called `remove` with index 1, the array would be `["hi", "hello", "okay", ...]`. Basically, the only `null` elements of the array should be *after* all the data.
- Initially and after a call to `clear()`, the `size` method should return 0. The "size" refers to the number of elements in the *list* , NOT the length of the *array.* After a call to `clear()`, the underlying array should be reset to a length of 2 as in the constructor.

2 Unit Tests

As mentioned at the beginning of the project writeup we will be using a program to evaluate your code. Specifically, we will be using JUnit (unit tests) to test each method that you implement. We may release these tests for you to use when they are ready to so you can check that everything in your classes works properly. Expect to hear more information about this in the near future.

3 Analysis

Now that you have implemented and used both an array list and linked list, which one is better? Which methods in `List` are more efficient for each implementation?

For each of the 13 methods in `List`, compare the runtime (Big-O) for each method and implementation. Ignore any increased efficiency caused by the flag `isSorted`. Include an `analysis.txt` or `analysis.pdf` with your submission structured as follows:

Method	ArrayList Runtime	LinkedList Runtime	Explanation
`boolean add(T element)`	`O(...)`	`O(...)`	...
`boolean add(int index, T element)`	`O(...)`	`O(...)`	...
⋮	⋮	⋮	⋮

Your explanation for each method only needs to be a couple sentences briefly justifying your runtimes.

4 Honors

Note: This section is **required** for students in Honors section only. Optional for others but no extra credit.

You will have two additional methods to implement. Each method must be implemented in both the ArrayList class and the LinkedList class.

1. `getKSmallest(int k)` – This method returns a new List object (ArrayList or LinkedList depending on what type of object the method is called on) containting the k smallest elements in the given List. If k is less than 1 then null should be returned. If k is greater than or equal to the size of the list then the entire list should be returned.
2. `getKLargest(int k)` – This method returns a new List object (ArrayList or LinkedList depending on what type of object the method is called on) containing the k largest elements in the given List. If k is less than 1 then null should be returned. If k is greater than or equal to the size of the list then the entire list should be returned.

You should also include time complexities for these methods in your analysis.txt/analysis.pdf file.