Department Computer Science and Software Engineering
Concordia University

COMP 352: Data Structures and Algorithms
Winter 2025 – Programming Assignment 2

Due date and time: Friday February 28th, 2025 by 11:59 PM

Important Note: Please notice that you can NOT use any built-in Java data structures such as Stacks, Queues, ArrayList, HashMap, LinkedList, and so on. Using any of these will result in voiding your assignment. Stick with primitive arrays and strings. For what you need, you will be creating your own.

The List

In this programming assignment you will evaluate two implementations of *List* interface in terms of their performance for different operations.

List[1] interface is an ordered collection (also known as a *sequence*). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list.

The List interface provides four methods for index **access** to list elements, two methods to **search** for a specific object, and two methods to efficiently **insert** and **remove** multiple elements at an arbitrary location in the list. Note that the speed of these operations may depend on the implementation (e.g. Array or LinkedList).

You are required to write two implementations of `List` interface, one that uses array, and one that uses doubly-linked list. Then, you will have to test the performance of several operations when using your implementations.

Part 1:

Implement the following methods in the two implementations (called `MyArrayList` and `MyLinkedList`) of `List` interface:

```
Boolean   add(E e)                  // Appends the specified element to the end of this list
void add(int index,E element)       // Inserts the specified element at the specified position in this list
void  clear()                       // Removes all of the elements from this list
E  remove(int index)                // Removes the element at the specified position in this list
Boolean remove(Object o)            // Removes the first occurrence of the specified element from this list
int size()                          // Returns the number of elements in this list
```

Also implement the following method:

```
String  toString()                  // Returns a string representation of this list
```

[1] List (Java Platform SE 8)

Define your classes to be generics. The array implementation should have dynamic resizing (double the size when growing and halve the size when less than 25 % of the capacity is used); and the linked list implementation should use doubly linked list. Also, the behavior of these methods should be equivalent to that of Java Standard Library's classes `ArrayList` or `LinkedList`. Please refer to the corresponding descriptions online[2]/[3].

For the rest of the methods of the `List` interface, you may just throw an exception:

```
public type someUnneededMethod() {
        throw new UnsupportedOperationException();
}
```

Part 2:

Write your driver class `ListTester`. Use both of your list implementations and compare them to the corresponding Java library implementations (*`ArrayList`* and *`LinkedList`*)
For numbers $N = \{10, 100, 1000, 10000, 100000, 1000000\}$

a) Starting with *empty* lists of Number-s; measure how long it takes to insert N integer numbers (`int`, or `Integer`) with random values ranging from 0 to 2N into the lists, when inserting them at the *beginning*, at the *end*, and into a *random location* of the list (use indices to indicate where to do the insertion (e.g., `list.add (randomLocation, number)`).
b) Starting with *non-empty* lists of N items (e.g., from part *a*), measure how long it takes to remove N numbers from the lists when removing them from the beginning, from the end, and from a random location of the list (use indices to indicate the location).
c) Starting with *non-empty* lists of N items (same as part *b*), measure how long it takes to remove N random numbers (with values between 0 and 2N) from the four lists (some values might not exist in the list!).

- Produce the following table (the timing values below are just an illustration and do not relate to any real measurements):

N = 10	**Insert@**start(ms)	**Insert@**end (ms)	**Insert@**random (ms)
MyArrayList	15	201	603
ArrayList	15	201	603
MyLinkedList	15	201	603
LinkedList	15	201	603

N = 10	**Remove@**start(ms)	**Remove@**end(ms)	**Remove@**random(ms)	**Remove** byvalue (ms)
MyArrayList	15	201	603	121212
ArrayList	15	201	603	121212
MyLinkedList	15	201	603	121212
LinkedList	15	201	603	121212

- Repeat for all values of N = 100; N = 1000; ….etc.

- Save the result of your program execution in a file `testrun.txt` and submit it together with your other files.

[2] LinkedList (Java Platform SE 8)
[3] ArrayList (Java Platform SE 8)

Important Requirements:

1. Make sure you reset the timer (or save the intermediate time before the next measurement); i.e., make sure your measured time contains only the time to perform one set of operations that was supposed to be timed.
2. In case the operations for big N numbers take too long (e.g., more than 50s) you may reduce the number to a smaller one or eliminate it (so that you will have a range from, say, 1 to 100000).
3. **Do not use** any java abstract data type or packages when writing `MyArraylist` and `MyLinkedList` in these programming questions of your assignment.
4. Your code should handle boundary cases and error conditions. It is also imperative that you test your classes.

Submission: You will need to submit both **the Java programs**, together with your testrun text files.

- **A group of 2 (maximum) is allowed. No additional marks are given for working alone.**
- **If you are working alone, you need to zip and submit your zipped file under the submission folder: Programming Assignment 2. If working in a group, only one submission is to be made by either of the two members (do not submit twice). You need to zip (see below) and submit your zipped file, under the submission folder: Programming Assignment 2. No handwritten or scans are allowed for your pseudocode or other documentations.**

  Submission format: All assignment-related submissions that have more than one document must be adequately archived in a ZIP file using your ID(s) and last name(s) as file name. The submission itself must also contain your name(s) and student ID(s). Use your "official" name only - no abbreviations or nick names; capitalize the usual "last" name. Inappropriate submissions will be heavily penalized. If working in a group, the file name must include both IDs and last names.

  IMPORTANT: A demo for about 5 to 10 minutes will take place with the marker. THERE WILL BE A DEADLINE TO BOOK YOUR DEMO SLOT, AND YOU MUST BOOK YOUR DEMO PRIOR TO THAT TIME. You (or **both** members if working in a group) **must** attend the demo and be able to explain their program to the marker. Different marks may be assigned to teammates based on this demo. The schedule of the demos will be determined and announced by the markers, and students must reserve a time slot for the demo (only one time-slot per group; or zero mark is given).

  Now, please read very carefully:
  - **If you fail to demo, a zero mark is assigned regardless of your submission.**
  - **If you book a demo time, and do not show up, for whatever reason, you will be allowed to reschedule a second demo but a penalty of 50% will be applied.**
  - **Failing to demo at the second appointment will result in zero marks and no more chances will be given under any conditions.**
  - **If you book more than one time-slot for the demo (notice, it is one slot per team), a zero mark will be given (not even 50%).**

NOW, WHICH IS EXTREMELY IMPORTANT, please notice very carefully the warning/disclaimer at the start of the assignment concerning the publication of the assignment, its text, or its solution (i.e. by having the solution publicly online, on GitHub, etc.). This will immediately constitute an academic misconduct, and such act will not be tolerated even after graduation.