Opeyemi Adesina, PhD
Assistant Professor
School of Computing
University of the Fraser Valley
O: C2435, Abbotsford Campus
Tel: (604) 504-7441 (ext: 4931)
opeyemi.adesina@ufv.ca

Assignment 1 — List Implementation & Analysis

COMP 251 : Algorithms & Data Structures

(100 points)

When Due: May 28, 2023 – 23:59:00 (PDT) [Submission via Blackboard]

A. Background

The goal of this assignment is to assess your skills on developing implementations for the List abstract data type while understanding their complexities through executions. This assignment amounts to **10%** of the entire course grade. Whatever you obtain as a score will be scaled to this value for final grade computation. Specifically, for final computation your earned points x is converted using this: [x (points) = $10x/100$] %. No late submission will be permitted (see deadline above), unless approved or granted by the course instructor. This is required to be done **ALONE**.

In the real-world, there are several solutions to most problems. This is equally true for the computational world. For example, in the Java programming language the development of List has been realized with array-based (aka ArrayList) and linked-based (aka LinkedList) implementations. To avoid conflicts, we shall be referring to these implementations as **ArrayBasedList** (self-expanding or auto-growing) and **SinglyLinkedList** in our work. This leaves us with analyzing these implementations to determine which is best for the problem at hand (particularly, understanding implications of our design for significant operations). Of course, we understand there is no one-size-fits-all for problems of this nature, notwithstanding - understanding the implications of our designs could help us make informed decisions.

B. List Abstract Data Type (ADT)

The Java API is a detailed list of permissible operations (Javadoc) on ArrayList and LinkedList. However, in Table 1 we present a list of operations (and their descriptions) that are most interesting to us (which is a proper subset of the list referenced for the original Java implementations for both cases) for the purpose of this assignment. Similarly, in Figure 1, we present a class model of the internal structure of the expected implementation. You are required to base your implementation on generics, so that your implementations will be type-independent.

C. Tasks To Be Completed

You are required to complete the following tasks:

50 points Implement ArrayBasedList and the SinglyLinkedList with the specification (or requirements) presented in Table 1. Implementation of other relevant or helper methods should have private acces-

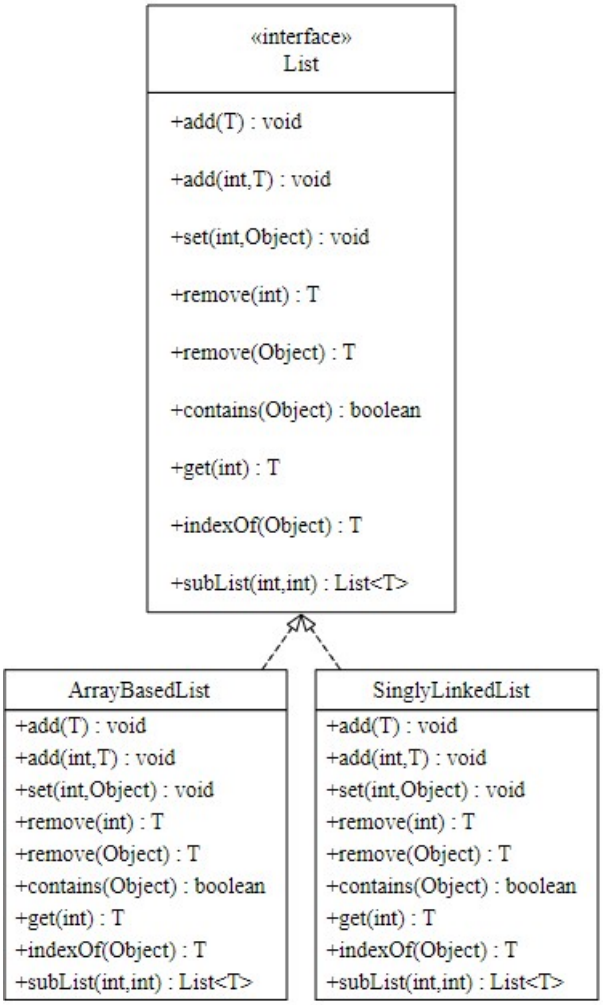

Figure 1: List specification model

sibility specifications while all APIs shall have public access modes. I will be developing test cases to assess correctness of your implementations.

40 points For each pair of accessor and modifier methods, conduct a performance study (using different input sizes like, 100, 1,000, 10,000, 100,000, 250,000).

(a) For each input size, generate at least two test cases.
(b) For each test case, run the program for at least three times (preferably in milliseconds) and collect the time value and take the average of all the runs.
(c) Plot the results (average execution time against the input size) on the same graph.

10 points Discuss your findings and results. Particularly, which of the implementations do you think would work best for any particular situations you could think of assuming you are a Solution Architect.

D. Academic Integrity Statement

By submitting this assignment, you pledge to have abide by the statements of Policy 70 of University of the Fraser Valley (UFV) including every other policy of the University that might be relevant to this subject matter. You agree that this submission is entirely yours but not a solution copied from the internet, or written by a tutor (either paid or unpaid), or written by a friend or a senior student. You acknowledge to seek help from the instructor as often as may be required (either through office hour or intermittent drop-by or a scheduled appointment).

Table 1: Required application programming interfaces - APIs

Operations	ArrayBasedList	SinglyLinkedList
Constructors	**ArrayBasedList()** Constructs an empty list with an initial capacity of ten. **ArrayList(int initialCapacity)** Constructs an empty list with the specified initial capacity.	**SinglyLinkedList()** Constructs an empty list.
Modifiers	**void add(T element)** Appends the specified element to the end of this list. **void add(int index, T element)** Inserts the specified element at the specified position in this list. **void set(int index, Object element)** Replaces the element at the specified position in this list with the specified element. **T remove(int index)** Returns the element at the specified position in this list. **T remove(Object element)** Removes the first occurrence of the specified element from this list, if it is present.	**void add(T element)** Appends the specified element to the end of this list. **void add(int index, T element)** Inserts the specified element at the specified position in this list. **void set(int index, Object element)** Replaces the element at the specified position in this list with the specified element. **T remove(int index)** Returns the element at the specified position in this list. **T remove(Object element)** Removes the first occurrence of the specified element from this list, if it is present.
Accessors	**boolean contains(Object element)** Returns true if this list contains the specified element. **T get(int index)** Returns the element at the specified position in this list. **T indexOf(Object element)** Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element. **List< T > subList(int fromIndex, int toIndex)** Returns a view of the portion of this list between the specified $fromIndex$, inclusive, and $toIndex$, exclusive.	**boolean contains(Object element)** Returns true if this list contains the specified element. **T get(int index)** Returns the element at the specified position in this list. **T indexOf(Object element)** Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element. **List< T > subList(int fromIndex, int toIndex)** Returns a view of the portion of this list between the specified $fromIndex$, inclusive, and $toIndex$, exclusive.