EECS 2011: Assignment 1

Due: as set in *eClass*

May be done in groups of up to three students

> *TL;DR: implement 7 methods of the `List` interface, submit a java file for one class. Cheating not allowed.*

Motivation

The purpose of this assignment is to implement a variation of a linked list and to investigate its time complexity for some of the list operations.

Introduction

Like the Assignment 0, this assignment will involve (partially) implementing a `List`[1] interface. The `List` interface provides four methods for positional (indexed) **access** to list elements, two methods to **search** for a specified object, and two methods to **insert** and **remove** multiple elements at an arbitrary point in the list.

While some of these operations often execute in constant amount of time, the others may execute in time proportional to the index value, depending on the implementation (the `LinkedList` class, for example, is not very efficient when accessing elements in the middle of the list using an index, and `ArrayList` does not allow for efficient insertion or deletion, unless the element is at the end of the list).

Implementation

In this assignment, you will have to write a quadruply-linked list-based partial implementation of the `List` interface. Unlike the existing `java.util.LinkedList`, your implementation will allow for faster traversal of the list. Specifically, the list nodes should contain references to elements ten positions forwards and backwards – if these exist, in addition to the common one position in each direction:

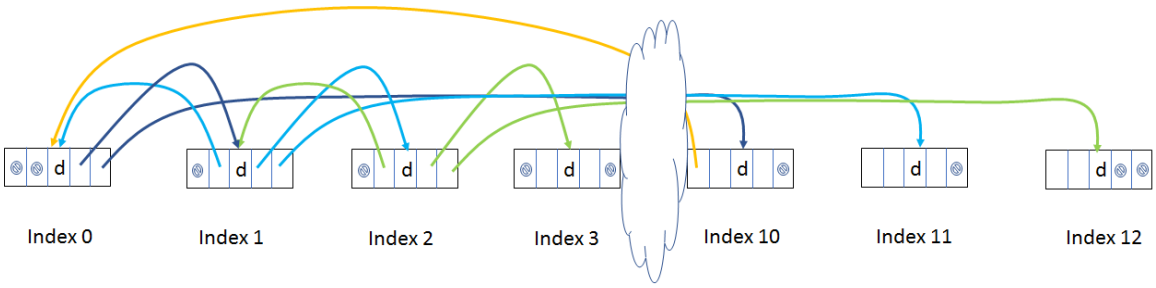

Figure 1: Illustration of the list you are required to implement. Nodes between indices 4 and 9 are not shown. For clarity, only some of the links are shown. ⊘ symbols indicate null references, and "d" are user data references.

[1] https://docs.oracle.com/javase/8/docs/api/java/util/List.html

You may review the following LinkedList implementation as an example how a linked list is implemented in Java (you may not copy or re-use the code from there):

https://developer.classpath.org/doc/java/util/LinkedList-source.html

Ultimately, you should implement your class from scratch. Also, you need to ensure the best possible performance for all the implemented operations. E.g., to access the 85th element of a 100-element list, one should not need to access more than six elements in addition to the requested one.

Part 1

Implement the following public methods in your implementation of the `List` interface, called `TenLinkedList`:

```
1. boolean     add(E e)
2. void        add(int index, E element)
3. E           remove(int index)
4. E           get(int index)
5. int         size()
6. void        clear()
7. String      toString() (see Java API: AbstractCollection[2])
```

One public constructor should exist in your implementation: one that takes no parameters and creates an empty list when the class is instantiated.

The class should use generics. The behaviour of the methods in your implementation should be *equivalent* to that of Java Standard Library's classes (e.g., `LinkedList;` please refer to the class API online). For the methods of the interface that you do not need to implement, you may either leave them empty or throw an exception

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

Of course, you are free to implement any private or protected methods and classes as you see fit. However, the methods mentioned above (or the ones present in the `List` interface, or in the class' superclasses) are the only public methods your class should contain. Furthermore, your code should not have any side effects, such as printing to the console.

Part 2

Nothing needs to be submitted for this part; however, completing this part will provide you an opportunity to investigate if your implementation is correct and to solidify your understanding of linked lists.

Create some tester class and use it with your list implementation to investigate its running time complexity as the number of items in the list increases and as you vary the indices of the elements you are trying to operate on. Try measuring the running time when inserting (deleting, getting…) 10, 100, 1000 … 1,000,000 elements at various positions. Confirm

[2] https://docs.oracle.com/javase/7/docs/api/java/util/AbstractCollection.html#toString()

that the running time follows the expected behaviour for `add`, `remove`, and for other methods.

Furthermore, it is recommended that you practice writing unit test cases for your implementation. In fact, it is often suggested that the test cases are written *before* the implementation commences.

NOTES

1. Do not use *package*-s in your project (put your classes in the `default` package). Using packages will cost you a 10 % deduction from the assignment mark.

2. Some aspects of your code will be marked automatically (e.g., how it handles boundary cases and error conditions), using a custom tester class. It is also imperative you test your classes. If any of the java files that you submit do not compile, the whole submission will be given a grade of zero, regardless of how trivial the compiler error is. The code you submit should compile using

```
javac *.java
```

command in either Windows, Linux, or MacOS.

3. Your code should include Javadoc comments.

4. Using any `java.util` implementations of `Collection`[3] or `Map`[4] interfaces is not allowed (this includes, but is not limited to, `ArrayList`, `LinkedList`, `Stack` and others; importing the `java.util.List` interface is of course fine).

Submission

Find all the `java` files in your project and submit them electronically via eClass (do not zip, tar, rar, or 7z them). Only one file is expected, but you may write other classes, if you deem them necessary.

If working in a group, make only one submission per group and include a `group.txt` file containing the names and the student numbers of the group members. The deadline is firm. Contact the instructor *in advance* if you cannot meet the deadline explaining your circumstances. The extension may or may not be granted, at instructor's discretion.

Grading

The assignment will be graded using *the Common Grading Scheme for Undergraduate Faculties*[5]. We look at whether the code passes the unit tests, satisfies the requirements of this document, and whether it conforms to the code style rules.

---

[3] https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html

[4] https://docs.oracle.com/javase/8/docs/api/java/util/Map.html

[5] https://secretariat-policies.info.yorku.ca/policies/common-grading-scheme-for-undergraduate-faculties/

Academic Honesty

Academic honesty will be strictly enforced in this course. Specifically, direct collaboration (e.g., sharing code or answers) is not permitted, and plagiarism detection software will be employed. You are, however, allowed to discuss the questions, ideas, or approaches you take.

Cite all sources you use (online sources – including web sites, old solutions, books, etc.) in your code comments; using textbook examples is allowed, but these must be cited as well.

E.g.,

```
1) /**
     * Constructs an empty list with the specified initial capacity.
     *
     * @param   initialCapacity   the initial capacity of the list
     * @exception IllegalArgumentException if the specified initial
capacity
     *            is negative
     * uses code from www.xxx.yyy.edu/123.html for initialization
     *
     */

2) //formula based on Jane Smart's thesis, page XX, available at
https://...
       a = b + c;
```

Overall, although using outside sources is *sometimes* allowed, unless specified otherwise – with proper citing, if the amount of non-original work is excessive, your grade may be reduced. You might find the following article useful to have an idea what kind of collaboration is appropriate: https://en.wikipedia.org/wiki/Clean_room_design

You may not post the contents of this assignment, nor solicit solutions on any external resources.