TCSS 342 - Data Structures
Assignment 1 - Lists

Time Estimate: 6-8 hours

Learning Outcomes

The purpose of this assignment is to:

- Build your own linked list data structure.
- Build your own dynamically sized array data structure.
- Build a generic data structure in Java.
- Gain experience testing and debugging data structures.
- Build data structures exactly to an API's specifications.

Overview

As Java programmers with some experience under your belts you have had a chance to use the common types of Java lists, LinkedLists and ArrayLists. In order to master their use it is important to build them yourself. In this assignment we will build our own specialized linked list and dynamically sized array list data structures for use in our assignments in this course.

To complete this you will implement two public classes and one private class:

- MyLinkedList<Type> (with a private Node)
- MyArrayList<Type>

Formal Specifications

MyArrayList<Type>
-list : Type[] -capacity : int -size : int
+insert(item : Type, index : int) +remove(index : int) : Type +contains(item : Type) : boolean +indexOf(item : Type) : int +get(index : int) : Type +set(index : int, item : Type) +size() : int +isEmpty() : boolean +toString() : String -resize()

Field summary

- `list` - We store the elements of the list in this array.
    - You can initialize a generic array like this:

```
list = (Type[]) new Object[capacity];
```

- `capacity` - The length of the array `list` and the current maximum `size`. Initialized to 16.
- `size` - The number of elements stored in the `list`.

Method summary

- `insert` - Inserts the `item` at position `index`. Any elements after the inserted element shuffle down one position. If the `index` is greater than the `size` then this method does nothing. This is the primary means of adding elements into a dynamically sized array. This method should run in O(i) time where i is the number of elements shuffled.
- `remove` - Removes the element at position `index` and returns the element. Any elements after the removed element shuffle down to fill the empty position. If the `index` is out of bounds this method does nothing and returns `null`. This is the primary means of deleting elements from a dynamically sized array. This method should run in O(i) time where i is the number of elements shuffled.
- `contains` - Searches the `list` for the `item` and returns true if found (and false otherwise). This is one of two standard searches in a list. This method should run in O(n) time.
- `indexOf` - Searches the `list` for the `item` and returns the index if found (and -1 otherwise). This is one of two standard searches in a list. This method should run in O(n) time.
- `get` - Returns the element stored at `index` and `null` if the `index` is out of bounds. This method should run in O(1) time.
- `set` - Updates the element stored at `index` and does nothing if the `index` is out of bounds. This method should run in O(1) time.
- `size` - Returns the field `size`. This method should run in O(1) time.
- `isEmpty` - Returns true if the `size` is 0 and false otherwise. This method should run in O(1) time.
- `toString` - Returns a string that has the contents of the `list` separated by commas and spaces and enclosed in square brackets. This method should run in O(n) time.
    - Example: [1, 2, 3, 4]
- `resize` - Called by `insert` when the `list` is full. Doubles the `capacity` of the `list` and copies the elements into a new array. This method should run in O(n) time.

MyLinkedList<Type>
-first : Node -current : Node -previous : Node -size : int
+addBefore(item : Type) +addAfter(item : Type) +current() : Type +first() : Type +next() : Type +remove() : Type +contains(item : Type) : boolean +size() : int +isEmpty() : boolean +toString() : String

Field summary

- `first` - A reference to the first `Node` in the list. Is `null` if the list is empty.
- `current` - A reference to the current `Node` in the list. Initialized to be the `first` of the list. Used to iterate over the list. Is only `null` if the `current` `Node` moves off the end of the list.
- `previous` - A reference to the previous current `Node` in the list. Used to iterate over the list. Is only `null` if `current` is equal to `first`.
- `size` - The number of elements stored in the `list.`

Method summary

- `addBefore` - Adds the `item` before the `current` `Node`. If the `current` `Node` is `null` the new element is added in the last position. This is a common means of adding elements into a linked list. This method should run in O(1) time.
- `addAfter` - Adds the `item` after the `current` `Node`. If the `current` `Node` is `null` this method does nothing. This is a common means of adding elements into a linked list. This method should run in O(1) time.
- `remove` - Removes the `current` `Node` and returns the element. Any elements after the removed element shuffle down to fill the empty position. If the `current` `Node` is

`null` this method does nothing. After this method the `current` `Node` will be equal to the node after the removed node. This is a common means of deleting elements from a linked list. This method should run in O(1) time.

- `current` - Returns the `item` stored in the `current` `Node`. This method returns `null` if the `current` `Node` is `null`. This method should run in O(1) time.
- `first` - Sets the `current` `Node` to be the `first` `Node` and returns the `item` stored in it. This method returns `null` if the `first` `Node` is `null`. This method should run in O(1) time.
- `next` - Sets the `current` `Node` to be the next node in the list and returns the `item` stored in it. This method returns `null` if the `current` `Node` is `null`. This method should run in O(1) time.
- `contains` - Searches the `Nodes` for the `item` and returns true if found (and false otherwise). This is the standard search in a linked list. This method should run in O(n) time.
- `size` - Returns the field `size`. This method should run in O(1) time.
- `isEmpty` - Returns true if the `size` is 0 and false otherwise. This method should run in O(1) time.
- `toString` - Returns a string that has the contents of the `Nodes` separated by commas and spaces and enclosed in square brackets. This method should run in O(n) time.
    - Example: [1, 2, 3, 4]

Node
+item : Type +next : Node
+toString() : String

Note: Node should be a private class within MyLinkedList and should not be accessible outside of this class.

Field summary

- `item` - The item stored in this node.
- `next` - A reference to the next `Node` in the list. Is `null` if there is no next `Node`.

Method summary

- `toString` - Returns the `toString` of `item`.

Testing

It is important that you test your code to ensure it works on all potential inputs. Please do this in a separate Main class, and without using additional tools (like JUnit). You will not submit your test code. Instead, your data structures will be tested by code developed by the instructor and/or grader. This code will not be provided to you, as you should test your code before submitting it. If your code fails our tests we will let you know which tests it fails so you can find and correct your errors.

Here is some example output from my Linked List test code:

```
Linked List Contents []
addBefore(1)
Linked List Contents [1]
addBefore(2)
Linked List Contents [1, 2]
addBefore(3)
Linked List Contents [1, 2, 3]
first() 1
addAfter(4)
Linked List Contents [1, 4, 2, 3]
current() 1
Linked List Contents [1, 4, 2, 3]
next() 4
Linked List Contents [1, 4, 2, 3]
current() 4
Linked List Contents [1, 4, 2, 3]
next() 2
Linked List Contents [1, 4, 2, 3]
current() 2
Linked List Contents [1, 4, 2, 3]
addBefore(5)
Linked List Contents [1, 4, 5, 2, 3]
addBefore(6)
Linked List Contents [1, 4, 5, 6, 2, 3]
remove() 2
Linked List Contents [1, 4, 5, 6, 3]
```

****Important****: You should test your data structures more than this! This is just an example to give you an idea of how to begin testing. This test does not test all the methods, nor does it test all the edge cases for possible method calls. Your testing should do both.

Submission

You will submit a .zip file containing:

- MyLinkedList.java - your linked list class.
- MyArrayList.java - your dynamically sized array class.

Grading Rubric

In order to count as complete your submission must:

1. Implement both the linked list and dynamically sized array list efficiently and without errors.
2. Follow the API exactly so that your classes will work with any testing code.
3. Pass all tests in the testing code. If you fail any tests, you will be told which test failed.

Reminder: Incomplete assignments can always be corrected and resubmitted. If they are completed within 7 days of the due date they will count as late and after that period they will count as missed. Please review the grading matrix for the number of permitted late and missed assignments.