TCSS 342 - Data Structures
Assignment 1 - Lists

Version 1.2 (June 20, 2022)

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 protected class:

- MyLinkedList<Type> (with a protected 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 to make room for the new element.
    - If the `index` is greater than the `size` or is negative then this method does nothing.
    - This method calls `resize` if there is not enough room in the array for the new element.
    - This method should run in O(i) time where i is the number of elements shuffled.
- `remove` - Removes the element at position `index.`
    - Returns the element that was removed.
    - 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 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 method should run in O(n) time.
- `indexOf` - Searches the `list` for the `item` and returns the index if found (and -1 otherwise).
    - 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.
    - Example: [1, 2, 3, 4]
    - This method should run in O(n) time.
- `resize` - Doubles the `capacity` of the `list.`
    - Creates a new array of twice the size and copies the old elements into the new list.
    - Called by `insert` when the `list` is full.
    - 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.
    - This reference should be updated whenever the first node is changed.
- `current` - A reference to the current node in the list.
    - The current node of the list is used to traverse.
    - The current node should only be changed by the methods `first`, `next` and `remove`.

    - Initialized to be the `null`.
    - When this node is `null` the current node has fallen off the end of the list.
- `previous` - A reference to the node before the `current` node in the list.
    - Whenever the `current` node is updated you should update this node.
    - This node is only `null` if `current` is equal to `first`.
    - If `current` is `null` then this node should be the last node in the list.
- `size` - The number of elements stored in the `list.`

Method summary

- `addBefore` - Adds the `item` before the `current` node.
    - This method adds the `item` between the `previous` node and the `current` node.
    - If the `current` node is `null` the new element is added in the last position.
    - If the `current` node is the `first` node then the new element becomes the new `first` node.
    - This method should run in O(1) time.
- `addAfter` - Adds the `item` after the `current` node.
    - This method adds the `item` between the `current` node and its next node.
    - If the `current` node is `null` this method does nothing.
    - This method should run in O(1) time.
- `remove` - Removes the `current` node and returns the element.
    - The link between the `previous` node and the node after the `current` node must be reconnected.
    - If the `current` node is `null` this method does nothing and returns `null`.
    - After this method the `current` node will be updated to the node after the removed node.
    - 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.
    - This method returns the `item` stored in the `current` node after the update.
    - 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.

    - This method returns the `item` stored in the `current` node after the update.
    - 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 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.
    - Example: [1, 2, 3, 4]
    - This method should run in O(n) time.

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

Note: Node should be a protected class within MyLinkedList.

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`.

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 pass all JUnit tests. In order to do that:

1. All class, field and method names must match these specifications exactly.
2. All method signatures must match these specifications exactly.

3. Your methods must function in the way specified and only in this way (do not perform extra functions).
4. Your methods must be free of bugs.

If you are struggling to pass the tests, have bugs you can't find or error messages you don't understand then please reach out to me or your learning group.

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.