MyLinkedList<E> Design Document

Description

The MyLinkedList<E> object can store data of data type E, where E is replaced by a specific data type or object. MyLinkedList<E> objects can be used for lists of objects where repetition is allowed, order is important, and the number of elements in the list changes, but changing the size of the list is more important than quick random access of the elements.

Services

The constructor MyLinkedList<E>() can be used to construct a new MyLinkedList<E> object. This runs in O(1) time.

The method size() can be used to return the size of the MyLinkedList<E> object. This method runs in O(1) time.

The method get(ind) returns the object in the list at index ind, where the index is based from 0. A precondition for this method is that the parameter ind must be an integer greater than equal to 0 but less than the size of the list. This directly implies the second precondition that the list must contain at least 1 element. This method runs in O(n) time.

The method set(ind,obj) sets the element at index ind (based from 0) to the given parameter obj. The two preconditions for this method are that the parameter ind must be an integer greater than equal to 0 but less than the size of the list, and the parameter obj must be the same data type or object as the other elements in the list. This method runs in O(n) time.

The method add(obj) appends the element obj to the end of the list. One precondition for this method is that the parameter obj must be the same data type or object as the other elements in the list. This method increases the size of the list by one. This method runs in O(1) time.

The method add(ind,obj) appends the element obj to the index ind in the list. The preconditions for this method is that the parameter obj must be the same data type or object as the other elements in the list and that the parameter ind must be an integer greater than equal to 0 but less than or equals to the size of the list. This method increases the size of the list by one. This method runs in O(n) time.

The method remove(ind) removes the element at index ind (based from 0) from the list. The precondition for this method is that the parameter ind must be an integer greater than equal to 0 but less than the size of the list. This method decreases the size of the list by one and shifts all the elements from index ind+1 to the end of the list one index to the left (ie index--). This method runs in O(n) time.

The method toString() returns the array as a string, with the elements encased in [] and separated by commas and spaces. This method runs in O(n) time.

The method `addFirst(obj)` appends the element `obj` to the end of the list. One precondition for this method is that the parameter `obj` must be the same data type or object as the other elements in the list. This method increases the size of the list by one. This method runs in `O(1)` time.

The method `addLast(obj)` appends the element `obj` to the front of the list. One precondition for this method is that the parameter `obj` must be the same data type or object as the other elements in the list. This method increases the size of the list by one. This method runs in `O(1)` time.

The method `getFirst()` returns the first element of the list (at index `0`), where the index is based from `0`. A precondition for this method is that the list must contain at least `1` element. This method runs in `O(1)` time.

The method `getLast()` returns the last element of the list (at index `size-1`), where the index is based from `0`. A precondition for this method is that the list must contain at least `1` element. This method runs in `O(1)` time.

The method `removeFirst()` removes the first element (at index `0` – based from `0`) from the list. A precondition for this method is that the list must contain at least `1` element. This method decreases the size of the list by one and shifts all the elements from index `1` to the end of the list one index to the left (ie `index--`). This method runs in `O(1)` time.

The method `removeLast()` removes the first element (at index `0` – based from `0`) from the list. A precondition for this method is that the list must contain at least `1` element. This method decreases the size of the list by one and shifts all the elements from index `ind+1` to the end of the list one index to the left (ie `index--`). This method runs in `O(1)` time.

The method `iterator()` creates an iterator for the `MyLinkedList<E>` object. This method runs in `O(1)` time.

Internal Data Structures and State

The `MyLinkedList<E>` objects use the private variables `first`, `last`, and `size`.

The variable `first` is used to remember the first node in the list. The variable `last` is used to remember the last node in the list. These two instance variables change the node which they point to whenever the first and/or last node in the list is changed by the methods `add(obj)`, `add(ind,obj)`, `remove(ind)`, `addFirst(obj)`, `addLast(obj)`, `removeFirst()`, or `removeLast()`.

The variable `size` is used to remember the size of the list. This instance variable increases by one each time `add(obj)`, `add(ind,obj)`, `addFirst(obj)`, or `addLast(obj)` is called and decreases each time `remove(ind)`, `removeFirst()`, or `removeLast()` is called.

Test Plan

The method `toString()` can be tested by printing multiple linked lists and checking that the output matches the expected contents.

The method `size()` can be tested by printing the output of this method for many linked lists and checking that output matches the length of the linked lists.

The method `get(ind)` can be tested by printing the output of this method for linked lists and checking that output matches the actual element of each linked lists at index `ind`.

The method `set(ind,obj)` can be tested by calling this method on many linked lists and checking that the resulting linked lists matches the expected one.

The method `add(obj)` can be tested by calling this method on many linked lists and checking that the resulting linked lists matches the expected one.

The method `add(ind,obj)` can be tested by calling this method on many linked lists and checking that the resulting linked lists matches the expected one.

The method `remove(ind)` can be tested by calling this method on many linked lists and checking that the resulting linked lists and the returned element are correct.

The method `addFirst(obj)` can be tested by calling this method on many linked lists and checking that the resulting linked lists matches the expected one.

The method `addLast(obj)` can be tested by calling this method on many linked lists and checking that the resulting linked lists matches the expected one.

The method `getFirst()` can be tested by printing the output of this method for linked lists and checking that output matches the actual element of each linked lists at index `0`.

The method `getLast()` can be tested by printing the output of this method for linked lists and checking that output matches the actual element of each linked lists at index `size() - 1`.

The method `removeFirst()` can be tested by calling this method on many linked lists and checking that the resulting linked lists and the returned element are correct.

The method `removeLast()` can be tested by calling this method on many linked lists and checking that the resulting linked lists and the returned element are correct.