CSE 222 HOMEWORK #3

LISTS AND COLLECTIONS

Orhan Aksoy

09104302

1. Problem

Instead of a regular doubly-linked list of a generic type that is composed of nodes containing a single data entry, a new list is to be designed whose nodes contain arrays of data items. However, the list and its ListIterator interface should behave exactly the same way as its original one-data-item-per-node list.

2. Analysis

Having a number of data items per node has a significant impact on the operations on the list.

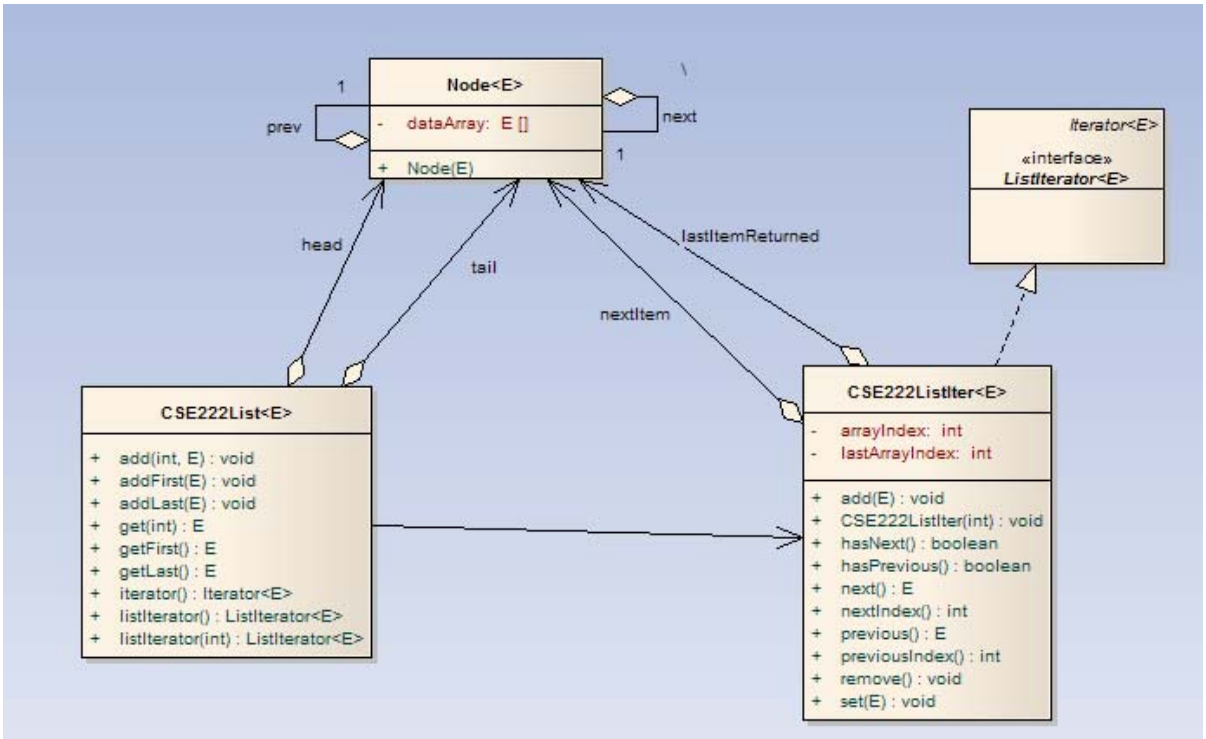
In a regular linked list, the addition operations normally create a new node and establish its connections to appropriate nodes in the list. In this case, however, an addition of a data item is a copy operation into an array location in a node. Prior to this operation, a check has to be made to see whether or not there's an available node with empty array cells in the appropriate location in the list. After addition, the restructuring of the list is necessary: Probably moving some array items, creating new nodes, etc.

The removal operation has a similar impact. Rather than simply removing a node and interconnect its neighbors, some operations need to be made within the node: copying array elements, deleting nodes if necessary, etc.

For the purpose of iteration in a regular linked list, a single node pointer would suffice to point to a specific location. In this case, however, both a node pointer and an index into the array are required to point to a specific data item. For every iterator operation, these members are to be updated to keep the list state consistent.

3. Design

The class diagram of the solution is shown below:



The CSE222List class provides 3 versions of add functions: The first one for addition at a specific location, second and third versions for adding to the front and to the end of the list respectively. Similarly, the get functions return data items from a specific location, from the front and the back of the list.

The List class also has three functions to return an iteratior into its list. These functions return a CSE222ListIterator object.

The complexity of movements within the list is handled by the CSE222ListIter class. All of the functions in the list class call their counterparts in the iterator class.

The iterator class keeps a reference to the pointed node and an index into the array within the node. It also has a reference named ‘lastItemReturned’ for the users of the class to refer to during remove and add operations.

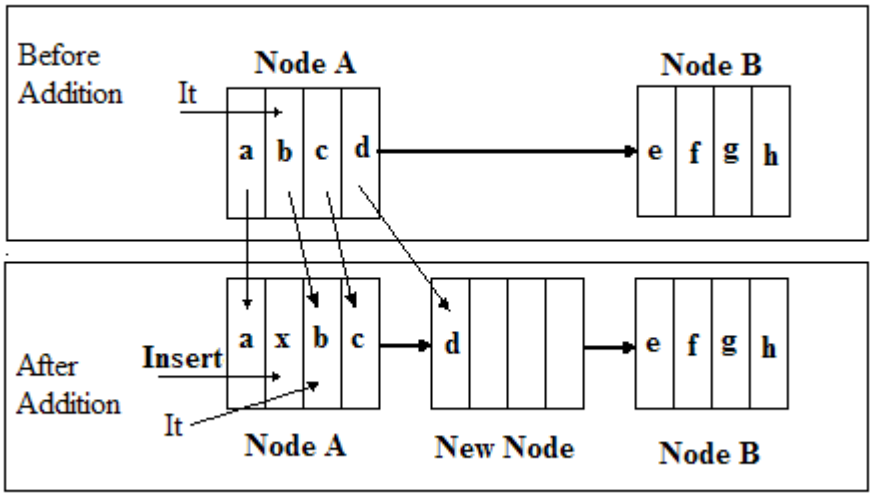
The details of the class member functions can be found on javadoc files.

4. Implementation

4.1. Addition

If a new item is to be added into the list at any location, the existing items on the array are shifted to their next locations. If the array was full before, one item is pushed out of the array. In this case, a new node is created and placed between the head and the head's old neighbor, holding only this single element.

The addition operation is done using the CSE222ListIterator member 'add'. The iterator adds the new item at the position it points and increments itself. This operation is shown on the figure below:



If an item is to be added at the end of the list and the last node is full, a new node is created and the item is put in the first array location in the node. The node interconnections are made exactly the same way as the regular linked list.

4.2. Removal

The removal is accomplished using the ListIterator 'remove' function. If the removed item is at the end of the array of a node, the member count of that node is decremented. Otherwise, the elements are shifted to keep the array in a consistent state.

If no items are left after the removal of an array item, the node is removed from the list. In this case, the interconnections of its neighbors are made exactly the same way as the regular doubly linked list.

After the removal, the iterator points to the next item in the list.

4.3. List traversal

The traversal is accomplished through the iterators. A list iterator can be created at a specific location in the list designated by the 'item count'. Then, next() and previous() methods are called to move around the list.

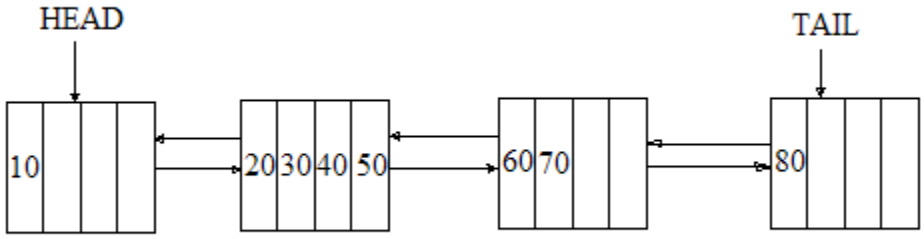
next() operation:

The next operation checks if it is pointing to the last array cell in the current node. If not, points to the next array location and returns the previously pointed item. If it is the last cell in the current node, proceeds to the next node in the list and points to the first array location. Again, returns the previous item in the array.

previous() operation:

The previous operation checks if it is pointing to the first array cell in the current node. If not, points to the previous array location and returns the new pointed item. If it is the first cell in the current node, proceeds to the previous node in the list and points to the last array location. Again, returns the newly pointed in the array.

After various additions and removals, a traversal on the list is shown on the figure below:



5. Program Output