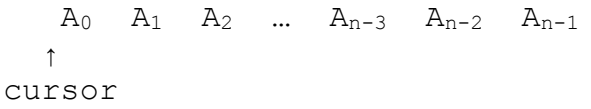
List Iterator Assignment

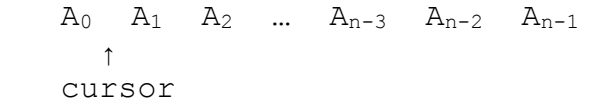
Overview

An iterator is a tool to efficiently access each value in a list, one at a time. In class, we discussed the Iterator interface and its implementation within the DoublyLinkedList class. The iterator object for a linked list can only move its cursor in one direction, forward, from the first value in the list to the last value in the list. Hence, the methods hasNext() and next(). The ListIterator interface provides the ability for an iterator to move in both directions, forward and backward. Hence, the additional methods hasPrevious() and previous().

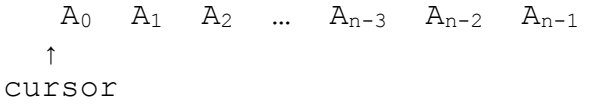
In the abstract view of an iterator, its cursor doesn't point to any value but between values in the list. At the list iterator's creation, its cursor points to the space right before the first value in the list.



Now, a call to the list iterator's hasNext() method would return true since the value A_0 is immediately to the right of the list iterator's cursor. A call to the list iterator's next() method will return the value A_0 and advance the cursor immediately after the value A_0.



Now, a call to the list iterator's hasPrevious() method would return true since the value A_0 is immediately to the left of the list iterator's cursor. A call to the list iterator's previous() method will return the value A_0 and advance the cursor immediately before the value A_0.



Unlike an iterator object, the list iterator object has an add method. A call to the list iterator's add method inserts the new value immediately before the value that would be returned by a call to next(), if any, and after the value that would be returned by a call to previous(), if any. If the list contains no values, the new value becomes the sole value in the list. The new value inserts before the list iterator's cursor such that a subsequent call to next() would be unaffected, and a subsequent call to previous() would return the new value.

An example: a call to the list iterator's add method, add(NV) would place NV (the new value) between A_{i-1} and A_i in the list as shown right below.

The list before the call to the add method:

A_0 A_1 A_2 … A_{i-1} A_i A_{i+1} … A_{n-3} A_{n-2} A_{n-1}

↑

cursor

The list after the call to the add method:

A_0 A_1 A_2 … A_{i-1} NV A_i A_{i+1} … A_{n-3} A_{n-2} A_{n-1}

↑

cursor

Like an iterator object, the list iterator object has a remove method. Though, the remove method of the list iterator object works a bit different than the remove method of an iterator object. A call to the list iterator's remove method removes from the list the last value returned by either the next method or the previous method, whichever method was called right before the call to remove(). This call to remove() can only be made once per a call to next() or previous(), which means that another call to the list iterator's remove method should be preceded by another call to either next() or previous(). If another call to either next() or previous() doesn't precede the next call to the list iterator's remove method then that call to the remove method should generate an exception. Also, if a call to the list iterator's add method is made after the last call to either next() or previous() and before the call to the list iterator's remove method then that call to the remove method should generate an exception.

Example 1:
The list before the call to the next method:

A_0 A_1 A_2 … A_{i-1} A_i A_{i+1} … A_{n-3} A_{n-2} A_{n-1}

↑

cursor

The list after the call to the next method:

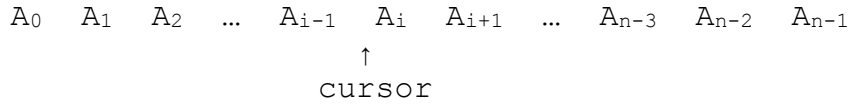
A_0 A_1 A_2 … A_{i-1} A_i A_{i+1} … A_{n-3} A_{n-2} A_{n-1}

↑

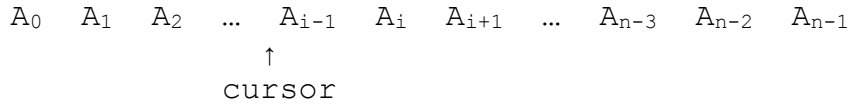
cursor

Then, a call to the list iterator's remove method will remove the value A_i from the list.

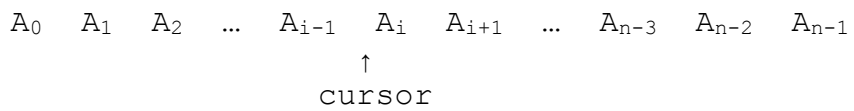
Example 2:
The list before the call to the previous method:



The list after the call to the previous method:



Then, a call to the list iterator's remove method will remove the value A_{i-1} from the list.



Design

You will write the implementation for the LinkedListIterator class which implements the ListIterator interface nested within the DoublyLinkedList class. Download the files List.java, ListIterator.java and DoublyLinkedList.java needed for the assignment.

You will write code for all the methods of the LinkedListIterator class. You will definitely have to write the code for the hasPrevious, previous and add methods. You can add any additional data members to the LinkedListIterator class but you **cannot** change or remove the existing ones: cursor and expectedModCount. You should not have to write or rewrite any of the code for the hasNext method. You will probably have to add additional code to the LinkedListIterator class constructor, and the next and remove methods. You will not need to create any additional methods for the LinkedListIterator class. You **cannot** add any nested classes to the LinkedListIterator class.

You **cannot** modify any of the code for the DoublyLinkedList class itself. You **cannot** add any nested classes to the DoublyLinkedList class. You **cannot** modify the nested Node class.

You do not need to write any other code for the assignment but you will need to thoroughly test your code implementation for the LinkedListIterator.

Grading Criteria

The total project is worth 20 points, broken down as follows:

1. If your DoublyLinkedList.java file does not compile successfully then the grade for the assignment is zero.
2. If your DoublyLinkedList.java file produces runtime errors which prevents the grader from determining if your code works properly then the grade for the assignment is zero.

If the program compiles successfully and executes without significant runtime errors then the grade computes as follows:
Followed proper submission instructions, 2 points:

1. Was the file submitted a zip file.
2. The zip file has the correct filename.
3. The contents of the zip file are in the correct format.

List Iterator execution:

4. The hasPrevious method works and executes properly, 3 points.
5. The hasNext method works and executes properly, 3 points.
6. The previous method works and executes properly, 3 points.
7. The next method works and executes properly, 3 points.
8. The add method works and executes properly, 3 points.
9. The remove method works and executes properly, 3 points.

Late submission penalty: assignments submitted after the due date are subjected to a 2 point deduction for each day late.

Submission Instructions

You'll place the DoublyLinkedList.java file containing your implementation of the LinkedListIterator class in a Zip file. The file should **NOT** be a **7z** or **rar** file! You can follow the directions below for creating a zip file depending on the operating system running on the computer containing your assignment's DoublyLinkedList.java file.

Creating a Zip file in Microsoft Windows (any version):

1. Right-click the DoublyLinkedList.java file to display a pop-up menu.
2. Click on **Send to**.
3. Click on **Compressed (zipped) Folder**.
4. Rename your Zip file as described below.
5. Follow the directions below to submit your assignment.

Creating a Zip file in Mac OS X:

1. Click **File** on the menu bar.
2. Click on **Compress "DoublyLinkedList.java"**.
3. Mac OS X creates the file **DoublyLinkedList.java.zip**.
4. Rename **DoublyLinkedList.java.zip** as described below.
5. Follow the directions below to submit your assignment.

Save the Zip file with the filename having the following format:
your last name,
followed by an underscore _,
followed by your first name,
followed by an underscore _,
followed by the word **Assignment1**.
For example, if your name is John Doe then the filename would be: **Doe_John_Assignment1**

Once you submit your assignment you will not be able to resubmit it!
Make absolutely sure the assignment you want to submit is the assignment you want graded.
There will be **NO** exceptions to this rule!

You will submit your Zip file via your CUNY Blackboard account.
Follow these instructions:

Log onto your CUNY BlackBoard account.
Click on the CSCI 313 course link in the list of courses you're taking this semester.
Click on **Content** in the green area on the left side of the webpage.
You will see the **Assignment 1 – List Iterator Assignment**.
Click on the assignment.
Upload your Zip file and then click the submit button to submit your assignment.

Due Date: Submit this assignment by Wednesday, March 28, 2018.