CSC205AB
MinilabDLL - Doubly Linked Lists

For this Minilab, you will change the singly linked list that we developed in class (SLList.java) to a doubly linked list (DLList.java). You'll then change some of the code to take advantage of the "prev" link and then test it.

Background: Make sure that you understand the SLList that was developed in class. The following methods were implemented in our SLList:

- toString() – returns a representation of the list and its elements as a String.
- isEmpty() – returns whether or not the linked list is empty.
- getFirst() – returns the first element on the list. If the list is empty, throw a NoSuchElementException. Be sure and import java.util.*; so it understands this exception.
- getLast() – returns the last element on the list. If the list is empty, throw a NoSuchElementException.
- addFirst(E el) – adds the element that is passed in to the head of the list and returns nothing.
- addLast(E el) – adds the element that is passed in to the tail of the list and returns nothing.
- add(int index, E elt) – adds the element that is passed in at the "position" indicated by index
- removeFirst() – deletes the first node from the list and returns its element. If the list is empty, throw a NoSuchElementException
- removeLast() – deletes the last node from the list and returns its element. If the list is empty, throw a NoSuchElementException.
- remove(E trash) – deletes the first occurrence of the element that is passed in; returns true or false depending on whether it is successful.
- contains(E elt) – returns true if the list contains the given element
- size() – returns the length of the list

Please do the following:

1. Make a copy of the SLList.java **that is given on Canvas** and rename it DLList.java. Be sure it compiles (rename things as needed).
2. Change the code so that it implements a doubly linked list. Change every name that contains SLL to contain DLL instead. Your DLLNode<E> class should have links that point to the previous and next nodes (the links are generally named prev and next). Please be sure that comments no longer say that it is a singly linked list.
3. The Canvas version of SLList.java is the same as the one that we developed in class, but has an additional method called backwards(). Currently, this method is just a copy of toString().
4. Change the method called backwards() so that it traverses the list from tail to head, following the prev link. That way, you will be able to tell if the prev links are set correctly.
5. Change the addLast method to correctly update the prev link. This should be done before the other methods since the tester uses it to generate test cases and the test doubly linked list should be correct.
6. Change the methods in DLList that add or remove nodes so that the prev and next links are used and updated correctly. In our removeLast method, we had to traverse the SLList list to find node before the tail so we could update the link. In DLList, there is a .prev link so **you must change the code to just use the prev reference** (the easier way) instead of traversing the list.

Testing: You can test your DLList with the DLLTesterHG that is provided. If your prev links are not set correctly, the calls to backwards() could result in the wrong answer, an infinite loop, or your program could crash (probably with a NullPointerException). As always, the tester simply catches the exception and prints its name; if you want to see where it occurred, you have to uncomment the line in the catch block that tells the exception to .printStackTrace();

Please submit: your DLList.java file via HyperGrade.