Homework 5

The purpose of this exercise is to practice further implementing a List interface and a `ListIterator`.

`ListIterator`: We will use a definition of the `ListIterator` which is close to the definition of the `ListIterator` in `java.util`. In particular, the iterator will allow to traverse the list forward using `hasNext()` and `next()` and backward using `hasPrevious()` and `previous()`. Below is an excerpts from the documentation for `java.util.ListIterator`.

```
public interface ListIterator<E>
extends Iterator<E>
```

An iterator for lists that allows the programmer to traverse the list in either direction, modify the list during iteration, and obtain the iterator's current position in the list. A `ListIterator` has no current element; its *cursor position* always lies between the element that would be returned by a call to `previous()` and the element that would be returned by a call to `next()`. An iterator for a list of length n has n+1 possible cursor positions, as illustrated by the carets (^) below:

```
                     Element(0)   Element(1)   Element(2)   ... Element(n-1)
 cursor positions:  ^           ^           ^           ^                   ^
```

Note that the cursor does not point to one element but is between two consecutive elements.

Export into Intellij Idea the GitHub Homework_5 folder. The program contains the interfaces `List` and `ListIterator`, the class `DoublyLinkedList` and class `Main` that includes the `main` method. You need only modify the class `DoublyLinkedList`. You may add additional tests to the main methods or comment out some existing test, but do not remove any.
You must:

(1) Read the definition of the `Node` private inner class. It is complete. You will need to understand and use the `Node` class.

(2) Complete the incomplete methods in the `DoublyLinkedList` (All incomplete methods include the comment MUST COMPLETE). Using calls to methods already written can greatly simplify your code. Comments gives specifications for each incomplete .

(3) Complete the `LinkedListIterator` (a private inner class of `DoublyLinkedList`).
This class has three fields: `cursor` is an integer that keep track of the position of the iterator. If a list has size n, then cursor can take values from 0 to n. There are two `Node` fields, one is for the node before the cursor (`previousNode`) and the other one is for the node after the cursor (`nextNode`). When `cursor` is 0, then `previousNode` is `null`, when the `cursor` is n, then `nextNode` is null. The code for `hasNext()` and

`next()` has been completed. You must complete `hasPrevious()` and `previous()`.
The first constructor starts the iterator at the head of the list; the second iterator start at the tail if the argument takes the value true.