Homework: List Implementation with Singly-Linked List and Two Smart Nodes

This homework is necessary preparation for the lab. Even though you need to turn in your answers only to the first two questions, it is essential that you spend some time on all the questions if you want to be able to complete the upcoming lab.

1. Review the `ListKernel` and `List` interfaces and make sure you are comfortable with the `List` mathematical model and the contracts of all the kernel methods (and some of the secondary ones). To aid in this task, we provide the following exercise. Complete (and print with your homework) the following tracing table. Recalling how much we care that you use correct punctuation when you write objects' values, we urge you to practice doing so. For example, after the 7th statement the value of `list` is `(<2, -1>, <0, 1>)`. Note the use of rounded parentheses, commas, and angle brackets.

Statement	Variable Values
`List<Integer> list = new List1L<>();`	
	list = `(<>, <>)`
`list.addRightFront(1);`	
	list = `(<>, <1>)`
`list.addRightFront(0);`	
	list = `(<>, <0, 1>)`
`list.addRightFront(2);`	
	list = `(<>, <2, 0, 1>)`
`list.advance();`	
	list = `(<2>, <0, 1>)`
`list.addRightFront(-1);`	
	list = `(<2>, <-1, 0, 1>)`
`list.advance();`	
	list = `(<2, -1>, <0, 1>)`
`list.moveToFinish();`	
	list = `(<2, -1, 0, 1>, <>)`
`list.retreat();`	
	list = `(<2, -1, 0>, <1>)`

`int i = list.rightFront();`	

	list = `(<2, -1, 0>, <1>)`
	i = `1`

`list.moveToStart();`	

	list = `(<>, <2, -1, 0, 1>)`
	i = `1`

2. Implement the `List<T>` secondary *instance* method `retreat` declared below. This must be a *layered* implementation using only the `ListKernel` methods.

```
/**
 * Retreats the position in {@code this} by one.
 *
 * @updates this
 * @requires this.left /= <>
 * @ensures <pre>
 * this.left * this.right = #this.left * #this.right  and
 * |this.left| = |#this.left| - 1
 * </pre>
 */
public void retreat() {...}
```

3. Review the Linked Data Structures II slides. In particular, make sure you completely understand the pictures of the representation of `List` implemented as a singly-linked list of nodes with one "smart" node at the start (Slides 14-16) and the modified representation with two "smart" nodes, one at the start and another at the end (Slides 19-20).
4. Review the `List2` implementation of `List` kernel represented as a singly-linked list of `Node`s with one smart node. Pay particular attention to the representation fields, the convention, and the correspondence. In the lab, you will be asked to modify this implementation so that it uses an additional smart node at the end of the singly-linked list.
5. Study the Implementing an Iterator slides and review the iterator implementation, `List2Iterator`, a nested class in `List2`. If you have any questions, make sure to ask them in class during the lab.