Lab 4 The List ADT

Goal

In this lab you will implement a `reverse` method that will reverse the order of the items in the linked implementation of the List ADT.

Resources

- Appendix A: Creating Classes from Other Classes
- Chapter 12: Lists
- Chapter 14: A List Implementation That Links Data

In `javadoc` directory

- *`ListInterface.html`*—Interface documentation for the interface `ListInterface`

Java Files

- *`ListInterface.java`*
- *`LList.java`*
- *`LinkedListExtensionsTest.java`*

In the `reverse` method of `LList`, implement your algorithm from the pre-lab exercises. Iteration is needed.

Checkpoint: Compile and run LinkedListExtensionsTest. The checkReverse tests should pass. If not, debug and retest.

Post-Lab Follow-Ups

1. Implement the `reverse` method using only the public methods in `ListInterface`. Compare the performance with what you did in the lab.

2. Consider a method
   ```
   void randomPermutation()
   ```
   that will randomly reorder the contents of the list. Create two versions of the method and compare the performance. In the first version, only use the methods from `ListInterface`. In the second version, always work directly with the linked chain in the `LList` implementation.

3. Consider a method
   ```
   void moveToBack(int from)
   ```
   that will move the item at position `from` to the end of the list. Create two versions of the method and compare the performance. In the first version, only use the methods from `ListInterface`. In the second version, always work directly with the linked chain in the `LList` implementation.

4. Consider a method
   ```
   void interleave()
   ```
   that will do a perfect shuffle. Conceptually, you split the lists into halves and then alternate taking items from the two lists. For example, if the original list is `[a b c d e f g h i]`, the splits would be `[a b c d e]` and `[f g h i]`. (If the length is odd, the first list gets the extra item.) The result of the interleave is `[a f b g c g d i e]`. Create two versions of the method and compare the performance. In the first version, only use the methods from `ListInterface`. In the second version, always work directly with the linked chain in the `LList` implementation.