Day 22 – The List Interface and Implementations

Chapter 06, 6.1 – The List Interface, Review Problem(s): ***3, 4, 5***

Chapter 06, 6.2 – List Implementations, Review Problem(s): ***7, 13***

Chapter 06, 6.4 – Sorted Array-Based List Implementation, Review Problem(s): ***31***

Chapter 06, 6.5 – List Variations, Review Problem(s): ***34, 36***

Previously, we used linked-lists, composed of nodes, to implement ADTs such as Stacks and Queues.

Today, we consider the List as an ADT itself. And, in a potentially amazing (or confusing) twist, we will implement the List ADT two ways, using arrays and linked-lists.

Just remember, before today the linked-list was a structure we used to implement other ADTs, and today we discuss a more general List ADT.

The List ADT is a collection of elements, with a linear relationship existing among its elements, each element has a position, called its ***index***. All the standard Collection operations are supported:

- `add`
- `get`
- `contains`
- `remove`
- `isFull`
- `isEmpty`
- `size`

But in addition, implementations of the List ADT support index-related operations and iteration.

The elements of a list are indexed sequentially, from 0 to list-size – 1. We can add, retrieve, change, or remove an element at an indicated index. In addition, we can determine the index of an element.

Chapter 06 PPT, pg 01-06.

Our lists implement the library's `Iterable` interface.

`Iterable` requires a single method, `iterator`, that creates and returns an Iterator object.

`Iterator` objects provide three operations: `hasNext`, `next`, and `remove`.

Chapter 06 PPT, pg 07-09.

Iterator lab.

We discuss an array-based and a link-based implementation of the List ADT. Take careful note of the anonymous inner class used in the array-based implementation.

Chapter 06 PPT, pg 10-19.

In the discussion of the Sorted Array-Based List Implementation, note the discussion of the Comparator Interface.

Chapter 06 PPT, pg 24-45.

Highlight slide 45:

Important Concept Revisited: Abstraction Hierarchy

- Here we saw another example of an abstraction hierarchy.
- Applications use the LargeInt class, which uses the LargeIntList class, which uses the DLLNode class.
- When working at any level in this hierarchy as a programmer we need to know only how to use the next lower level – we do not need to know how it is implemented nor need we worry about the details of lower levels.
- Abstraction is indeed the key to conquering complexity.