Day 21 – Collection Implementations and Variations

Chapter 05, 5.5 – The Collection Interface, Review Problem(s): ***23, 25, 27***

Chapter 05, 5.6 – Link-Based Collection Implementation, Review Problem(s): ***28, 29***

Chapter 05, 5.7 – Collection Variations

The ArrayCollection class, which of course is an implementation of the Collection ADT, has a fast `add` operation, ***O(1)***, but slow `get`, `contains`, and `remove` operations, ***O(N)***.

Many applications also require fast retrieval, especially if they constantly access the collection. If we do a little work to maintain a sorted array, then we can use a Binary Search algorithm for `find`, which in turn will speed up both `get` and `contains`.

Chapter 05 PPT, pg 29-33.

When designing an ADT, we can choose whether the elements are "***by-copy***" or "***by-reference***." As it sounds, in by-copy, the ADT manipulates copies of the data, while by-reference manipulates references to the actual elements. By-reference reminds us of the problems we had with aliasing.

Chapter 05 PPT, pg 34-41, taking special note of the "weight" objects example.

We can also implement a Link-Based Collection of the Collection ADT.

Chapter 05 PPT, pg 42-46.

There are additional variations on implementing the Collection ADT. The Java library provides a "robust" collections framework. He we present the Bag ADT and Set ADT.

Chapter 05 PPT, pg 47-53.