Day 20 – The Collection Interface and Applications

Chapter 05, 5.1 – The Collection Interface, Review Problem(s): ***3***

Chapter 05, 5.2 – Array-Based Collection Implementation, Review Problem(s): ***6, 7***

Chapter 05, 5.3 – Application: Vocabulary Density

Chapter 05, 5.4 – Comparing Object Revisited, Review Problem(s): ***15, 16, 19***

Before we start on Chapter 05, let's review a concept from HW 04. Specifically, there are several options when tracking Exceptions in Java. Previously we simply threw an Exception. In HW 04, although it wasn't explicitly asked for, it seems to make sense to try to do something with the Exception, or at least continue with executing if an error we anticipate happening actually happens.

Recall in Day 09, we extended our Car.java class so that it threw an Exception if a Car object was attempted to be instantiated with negative initial mileage.

```
// instantiate several CarSafe objects, demonstrating that a try-catch
// allows execution to continue after an error is caught

// first, create arrays to hold the work we do
int[] miles = new int[] {-20,100};
String[] colors = new String[] {"red","blue"};
CarSafe[] fleet = new CarSafe[2];

for (int i=0; i<2; i++)
{
        try
        {
                fleet[i] = new CarSafe(miles[i],colors[i]);
                System.out.println( fleet[i] );
        }
        catch (CarMileageNegativeException e)
        {
                System.out.println(e.getMessage());
        }
}
```

Now, on to Chapter 05. The ADTs we discussed last chapter, Stacks and Queues, restrict access to the stored data based on the order in which it was stored.

When we need to retrieve information regardless of the order in which it is stored, this is called ***content based access***. The Collection ADT we present in Chapter 05 is the most basic ADT for this purpose.

In a Collection, we have functionality to

- Add
- Remove

- Retrieve

Content based access requires an `equals` method to find objects. Duplicate elements are allowed, but `null` elements are not. The `add` and `remove` operations return boolean values, indicating success.

Now, we transition from considering the Collection ADT to various implementations and applications. The first implementation of the Collection ADT will use an array as the underlying data structure. Note the helper `find` method, and it's order of operations is ***O(N)***, so the entire array may need to be searched to find an element.

Chapter 05 PPT, pg 01-14.

We've seen that Collection operations requires testing objects for equality. As we transition to consider an implementation of the Collection ADT using a ***sorted*** array, we need to compare objects for order, answering the question, "which is bigger?"

The comparison == operator returns ***`true`*** if two objects are pointing to the same memory location. It gives a ***`false`*** if objects have the exact same values for all instance variables, but are in different memory locations.

The `.equals` method is exported from the `Object` class, and it can be redefined to compare the attribute variables of objects which are in different memory locations. Objects are identified and compared by their "key."

To support a sorted Collection we need to be able to tell when one object is less than, equal to, or greater than another object.

The Java library provides an interface, called `Comparable`, which can be used to ensure that a class provides this functionality. We call the order established by a class `compareTo` method the ***natural order*** of the class.

`compareTo` in class programming lab.

Chapter 05 PPT, pg 18-28.

Vocabulary density of a text is the total number of words in the text, divided by the number of unique words. How would you use a Collection to determine vocabulary density?

Chapter 05 PPT, pg 15-17.

HW 05 expands on this, creating a challenge game.