Day 10 – Data Structures and Structuring Mechanisms

Chapter 01, 1.4 – Data Structures, Review Problem(s): ***27, 28, 29***

Chapter 01, 1.5 – Basic Structuring Mechanisms, Review Problem(s): ***30, 31, 32, 34***

As you work through the below, put comments in your code describing what you're doing and the connections.

We will explore Aliasing and Garbage Collection using the Car class.

You may use the Car class you created on Day 07, or use the code from Moodle, your choice. In this case, we will only modify the TDCar class.

Create two different Car objects, but using the same parameters, and then output them (just sending the object to System.out.println will use the toString method):

```
    Car Highlander = new Car(22000, "red");
    System.out.println( Highlander );
    Car Highlander2 = new Car(22000, "red");
    System.out.println( Highlander2 );
```

Now, compare the two objects using the == operator. Does the if statement indicate these objects are equal, or not?

```
if (Highlander == Highlander2)
    {
      System.out.println("they are ==");
    } else {
      System.out.println("they are NOT ==");
    }
```

So, even though the objects contain the exact same values, the == tests the location of the object.

Now, compare the two objects using the equals method. Does the if statement indicate these objects are equal, or not?

This also indicates that the objects are not equal, primarily because we did not define an equals method in Car.java. ***Just for this example, let's add an equals method to Car.java***. *Normally we would NOT change an existing class, but for expediency in this example we will.*

```
public boolean equals (Car check)
   {
         boolean returnVal = true;
         if (this.color!=check.color) returnVal=false;
         if (this.miles!=check.miles) returnVal=false;
         return returnVal;
   }
```

If you re-run the code, the == will still show the objects are not equal, but now the equals method, having checked each of the attributes, will show they are equal.

Now, assign one of the objects to the other, using the = sign:

```
Highlander2 = Highlander;
```

Re-run both if statements. What happens?

```
they are ==
they are equal()
```

Both references now point to the same memory location, so the == now shows them as being equal.

The unreferenced memory location is now ***garbage***, and Java will need to free that memory. In some programming languages, when unused memory is not freed up, errors occur when the application runs out of memory. These are called ***memory leaks***.

Note that Java treats primitive data, like int and float differently. Tests for equality are as you would expect, and are on the values in the memory locations, not the memory locations themselves.