06.04 Virtual Lecture Notes (Part 2)

You are accustomed to using arrays that contain either a single primitive data type or single object like a **`String`**. For example, the following shows two ways to declare and initialize a small numeric array of type **`double`**:

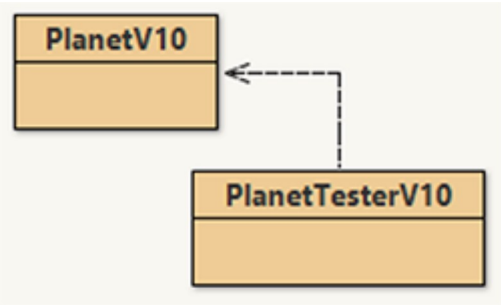
```
double[] numbers = {23.5, -0.43, 8.75};

double[] numbers = new double[3];
numbers [0] = 23.5;
numbers [1] = -0.43;
numbers [2] =  8.75;
```

This representation is easy to grasp because each index position represents one storage location in memory, and one storage location can only hold one value.

On the other hand, a statement like the following, which creates an array of objects and assigns them to index positions `[0]` and `[1]`, is a little harder to grasp:

```
PlanetV10[] planets = {new PlanetV10("Jupiter", 142984),
                       new PlanetV10("Mars", 6794)};
```

Open the PlanetV10 and PlanetTesterV10 classes side by side and examine the code. Do a quick desk check of the code.



Examining the constructor of the PlanetV10 class reveals that there will be four private instance variables associated with each object in each index position, as well as one to calculate radius.

```
private String n;
private double d;

//two parameter constructor
public PlanetV10(String name, double diam){
   n = name;
   d = diam;
}
```

Multiple values are associated with each object assigned to an index position. Java assigns a reference value to each index position that indicates where the four values for an object can be found in memory.

Without getting too technical, it might help simply to envision the instantiation of each object in the following manner:

```
planets[0]

          .n = "Jupiter"
          .d = 142984
          .radius = 0.0

planets[1]

          .n = "Mars"
          .d = 6794
          .radius = 0.0
```

This representation shows that there are two objects in the **`planets`** array at index positions `[0]` and `[1]` and that each object's private instance variables are closely associated with it. There is also an inference that the instance variables in a specific object can be accessed with the now familiar dot notation. For example, can you predict the outcome of the following statements?

```
int i = 1;
double diam1 = planets[i-1].getDiam();
planets[i].setDiam(diam1);
planets[i].calcRadius(planets[i].getDiam());
```

Statements like these are the key to many of the remaining assignments in this module. It would be good practice for you to type this in at the end of the PlanetTesterV10 class, add some print statements, and see if your predictions are correct.

Print