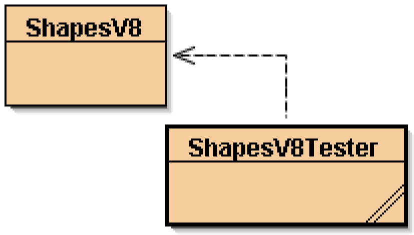
08.08 Virtual Lecture Notes (Part 2)

You are accustomed to using arrays that contain either a single primitive data type or single object like a **`String`**. For example, the following statement declares and initializes a small numeric array of type **`double`** which can be envisioned as three elements in an indexed list.

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

Now open the ShapesV8 and ShapesV8Tester classes side-by-side and examine the code. Do a quick desk check of the code and be sure you explored the interactive version of the ShapesV8 class in the lesson.



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

```
ShapesV8[] shapes = {new ShapesV8(10, 5), new ShapesV8(7, 13)};
```

Examining the constructor of the ShapesV8 class reveals that there will actually be four private instance variables associated with each index position, not just the two shown as arguments!

```
         private int mySide1, mySide2;
         private double myArea, myHypoteneuse;

         ShapesV8(int s1, int s2)
         {
             mySide1 = s1;
             mySide2 = s2;
             myArea = 0.0;
             myHypoteneuse = 0.0;
         }
```

Clearly each object represents too much information to be assigned to a single index position. But Java resolves this problem by assigning a reference value to each index position that indicates where the four values for these 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.

shapes[0]　　　　　　　　　　　　　　shapes[1]

.mySide1 = 10	.mySide1 = 7
.mySide2 = 5	.mySide2 = 13
.myArea = 0.0	.myArea = 0.0
.myHypoteneuse = 0.0	.myHypoteneuse = 0.0

This representation shows that there are two objects in the **`shapes`** 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 sideA = shapes[0].getSide1();
shapes [1].setSide1( shapes[0].getSide2());
shapes[1].calcTriArea();
shapes[1].calcHypoteneuse();
```

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 ShapesV8Tester class, add some print statements, and see if your predictions are correct.