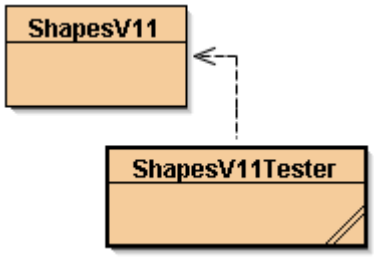
08.11 Virtual Lecture Notes

Print copies of the ShapesV11 and ShapesV11Tester Classes so you can lay them out side by side and do a quick desk check. Compare the ShapesV8 class to the Shapes V11 class and you will notice that they are virtually identical.



Next, examine the **`ShapesV11Tester`** class. The first key difference between this and the earlier version that dealt with an array of objects is shown in the following segment of code. The new **`shapes ArrayList`** is declared, but the type must also be included between angle brackets.

```
ArrayList<ShapesV11> shapes = new ArrayList<ShapesV11>();

shapes.add(new ShapesV11(10, 5));
shapes.add(new ShapesV11(7, 13));
shapes.add(new ShapesV11(2, 4));
shapes.add(new ShapesV11(28, 3));
```

Next, notice that the **`add()`** method is used to add individual objects (records) into the **`shapes Arraylist`**. Examining the constructor of the **`ShapesV11`** class reveals that the two arguments for each triangle are passed to corresponding parameters (**`s1`** and **`s2`**), which in turn are reassigned to the private instance variables **`mySide1`** and **`mySide2`**. In addition, 0s are assigned to the two remaining private instance variables (**myArea** and **myHypotenuse**) because their values will be calculated.

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

ShapesV11(int s1, int s2)
{
    mySide1 = s1;
    mySide2 = s2;
    myArea = 0.0;
    myHypotenuse = 0.0;
}
```

After this segment of code is executed, the structural organization of the data in each record of the **`ArrayList`** may be represented as shown here.

ShapesV11

index	mySide 1	mySide 2	myArea	myHypoteneuse
1	10	5	0.00	0.00
2	7	13	0.00	0.00
3	2	4	0.00	0.00
4	28	3	0.00	0.00

With the data structure defined, the calculations can be performed. But, notice that another **`ShapesV11 ArrayList`** object, **`dataRecord`**, has been declared. The for loop iterates through the **`ArrayList`**, getting one record at a time and assigning it to **dataRecord**. Then each method is invoked on the **`dataRecord`** object to calculate the area and the hypotenuse of each individual triangle object.

```
ShapesV11 dataRecord;

for(int index = 0; index < shapes.size(); index++)
{
    dataRecord = shapes.get(index);
    dataRecord.calcTriArea();
    dataRecord.calcHypoteneuse();
}
```

The second for loop goes through another series of iterations, getting each **`dataRecord`** of the **`shapes ArrayList`** and invoking the getter methods to print the results.

```
for(int index = 0; index < shapes.size(); index++)
{
    dataRecord = shapes.get(index);

    System.out.printf("%12d %9d %15.2f %12.2f%n",
                      dataRecord.getSide1(),
                      dataRecord.getSide2(),
                      dataRecord.getHypoteneuse(),
                      dataRecord.getTriArea());
}
```

Spend some time experimenting with the demo program until you are comfortable with the way it operates. Try adding some new triangles to the ArrayList and observe the output.