Virtual Lecture Notes

Remember that a class's attributes and behaviors are represented by instance variables and methods, respectively. Instance variables are initialized by a constructor and they are usually private. When a parameterless default constructor is used, instance variables that are primitive data types are assigned a value of 0 or 0.0, and objects are assigned the **`null`** reference. When a constructor that takes parameters is used, the values of the local variables in the parameter list are re-assigned to the private instance variables.

Version 5 of the Shapes class illustrates a key ingredient in object-oriented programming: private instance variables. The class diagram on the left (**`ShapesV3`**) relies on a default constructor, while the diagram on the right (**`ShapesV5`**) illustrates the use of a constructor that takes parameters. Both versions are equally valid.

ShapesV3
<< In main method >> + int side1 + int side2 + double triArea + double hypoteneuse
<< Constructor >> + ShapesV3() < Methods >> + double calcTriArea(int s1, int s2) + double calcHypoteneuse(int s1, int s2)

ShapesV5
<<Private Instance Variables>> - int mySide1 - int mySide2 < In main method >> + int side1 + int side2 + double triArea + double hypoteneuse
<< Constructor >> + ShapesV5(int s1, int s2) < Methods >> + double calcTriArea() + double calcHypoteneuse()

In the **`ShapesV5`** class, notice the use of two private instance variables (**`mySide1`** and **`mySide2`**), the parameter list for the constructor, and the lack of a parameter list for the methods. When you examine the source code shown below, pay close attention to the corresponding sections in the class diagram.

Notice that values are no longer passed directly to local variables in a method's parameter list. Instead, values are passed to the constructor and the values in the parameter list variables are then re-assigned to the private instance variables. Consequently, the private instance variables are used in the arithmetic statements, not the local variables in the constructor's parameter list. Values may still be passed to methods, depending on the context of the situation.

Does it seem like an unnecessary extra step, or a detour, for values to pass through the constructor to the private instance variables? This data hiding feature is characteristic of OOP design, as you will see in the second half of the course.

Study the source code of the **`ShapesV5`** class very carefully, and run the program to observe the output. Use the desk check procedure to trace the flow of control through the program. You will need to be able to use a constructor that takes parameters in

the programming assessent for this lesson, so if you have questions, please contact your instructor for assistance.

```
public class ShapesV5
{
    //private instance variables
    int mySide1, mySide2;
    //default constructor
    ShapesV5()
    {
    }
    //constructor with two parameters
    ShapesV5 (int s1, int s2)
    {
        mySide1 = s1;
        mySide2 = s2;
    }
    //calculate area of a triangle
    public double calcTriArea()
    {
        return mySide1 * mySide2 * .5;
    }
    //calculate the hypoteneuse of a right triangle
    public double calcHypoteneuse()
    {
        return Math.sqrt(Math.pow(mySide1, 2) + Math.pow(mySide2, 2));
    }
    //main method
    public static void main(String[] args)
    {
        //declaration of variables
        int side1, side2, radius;
        double triArea, circArea, hypoteneuse, circumference;
        //initialization of variables
        side1 = 10; side2 = 5;
        triArea = 0; hypoteneuse = 0;
        ShapesV5 shapes = new ShapesV5(side1, side2);
        //call methods
        triArea = shapes.calcTriArea();
        hypoteneuse = shapes.calcHypoteneuse();
        //print results
        System.out.printf(" Triangle Area = %8.2f%n", triArea);
        System.out.printf("   Hypoteneuse = %8.2f%n", hypoteneuse);
    }
}
```