08.03 Virtual Lecture Notes (Part 1)

Java programs can be written in several styles, which can be categorized as **`main()`** method, procedural, and object-oriented design programming. Three variations of the **`Shapes`** class are described to illustrate the progression through these styles. Print this document so you can spread it out on your desk and trace through each program as you refer to the version notes. Be sure you run the corresponding program, which was downloaded previously, and make the final modifications suggested. You will not be prepared for the next assignment unless you practice adding new methods in the OOP style. Request help if you need it.

Version 1 of the program is an example of placing all of the code in the **`main()`** method. Although the sequential flow of control line-by-line through the **`main()`** method is easy to follow and interpret, this simplistic style is ill-suited for anything but testing small segments of code.

Version 2 is an example of the procedural programming style. **Static methods** contain small functional units of code and each method is called, as needed, with messages from within the **`main()`** program. While this design works in the program, there are more appropriate uses for static methods, which will be dealt with later. The key point is that static methods do not require the explicit instantiation of objects, which has implications for the design of object-oriented programs.

Version 3 illustrates basic object-oriented programming in which a default constructor is used to create an object **`(shapes`**) of type **`ShapeV3`**. The actual default constructor, located immediately below the class definition, is easily missed.

```
ShapesV3()
{
}
```

It may seem odd that this simple statement, which contains no code between the curly braces, could really hold the key to object-oriented programming!

Next, turn your attention to the **`ShapesV3`** class's two methods: **`calcTriArea()`** and **`calcHypoteneuse()`**. Notice that, except for no longer including the **`static`** modifier, the headers of the methods in the V2 and V3 classes are identical.

Some important modifications have been made in the **`main()`** method, which identify the object-oriented nature of the **`ShapesV3`** class. The following statement is a significant new addition to the program, but should look familiar because of the keyword **`new`**.

```
ShapesV3 shapes = new ShapesV3()
```

This is the same format previously used to construct **`PrintWriter`** and **`Scanner`** objects and is the standard format for invoking the constructor of a class to create an object. It consists of two parts separated by the equal sign.

The code on the left of the equal sign declares that there will be a new object (**shapes**) of type **ShapesV3**, the right side invokes the constructor to actually create the new object. This statement can also be written in two lines as shown below.

```
ShapesV3 shapes;
shapes = new ShapesV3();
```

Once a **shapes** object is constructed, the methods of the **ShapesV3** class can be invoked to perform their calculations. Invoking a method is accomplished using dot notation to call the method on the object, as illustrated in the following statements.

```
//call methods
triArea = shapes.calcTriArea(side1, side2);
hypoteneuse = shapes.calcHypoteneuse(side1, side2);
```

How does this differ from invoking static methods? Refer to the **ShapesV2** class and you will notice that naming the object is not required, but everything else is the same.

When these statements are executed, flow of control jumps to the appropriate method and value of each variable listed as an argument is passed to the corresponding variable in the method's parameter list. The method then performs its task and returns an answer to be assigned to the variable on the left side of the equal sign.

To make the transition from procedural programs with static methods to object-oriented programs that explicitly construct an object requires four modifications:

1. Include a statement that invokes the constructor to create a **new** object.
2. Write the default constructor.
3. Write method headers, but do not include the static modifier.
4. Write statements in the **main()** program using dot notation to call methods for the object constructed.

Create a new class called **ShapesV4** and copy the **ShapesV3** class to it. (Be sure to change the name.) Modify the class to calculate the area of a rectangle, the perimeter of a rectangle, and the circumference of a circle. Use the following method headers when you make your modifications:

```
public double calcAreaRectangle(int w, int l)
public int calcRectPerimeter(int w, int l)
public double calcCircumference(int r)
```

The three versions of the **Shapes** class are shown below. Study them carefully.

```
public class ShapesV1
{
  //main method
  public static void main(String[] args)
  {
     //declaration of variables
     int side1, side2;
     double triArea, hypoteneuse;

     //initialization of variables
     side1 = 10; side2 = 5;
     triArea = 0; hypoteneuse = 0;

     //calculations
     triArea = side1 * side2 * .5;
     hypoteneuse = Math.sqrt(Math.pow(side1, 2)
                   + Math.pow(side2, 2));

     //print results
     System.out.printf(" Triangle Area = %8.2f%n", triArea);
     System.out.printf("   Hypoteneuse = %8.2f%n", hypoteneuse);

  }
}
```

```java
public class ShapesV2
{
  //calculate area of a triangle
  public static double calcTriArea(int s1, int s2)
  {
      return s1 * s2 * .5;
  }

  //calculate hypoteneuse
  public static double calcHypoteneuse(int s1, int s2)
  {
      return Math.sqrt(Math.pow(s1, 2) + Math.pow(s2, 2));
  }

  //main method
  public static void main(String[] args)
  {
     //declaration of variables
     int side1, side2, radius;
     double triArea, hypoteneuse;

     //initialization of variables
     side1 = 10; side2 = 5;
     triArea = 0; hypoteneuse = 0;

     //call methods
     triArea = calcTriArea(side1, side2);
     hypoteneuse = calcHypoteneuse(side1, side2);

     //print results
     System.out.printf(" Triangle Area = %8.2f%n", triArea);
     System.out.printf("   Hypoteneuse = %8.2f%n", hypoteneuse);

  }
}
```

```
public class ShapesV3
{
  //default constructor
  ShapesV3()
  {
  }

  //calculate area of a triangle
  public double calcTriArea(int s1, int s2)
  {
      return s1 * s2 * .5;
  }

  //calculate the hypoteneuse of a right triangle
  public double calcHypoteneuse(int s1, int s2)
  {
      return Math.sqrt(Math.pow(s1, 2) + Math.pow(s2, 2));
  }

  //main method
  public static void main(String[] args)
  {
     //declaration of variables
     int side1, side2;
     double triArea, hypoteneuse;

     //initialization of variables
     side1 = 10; side2 = 5;
     triArea = 0; hypoteneuse = 0;

     ShapesV3 shapes = new ShapesV3();

     //call methods
     triArea = shapes.calcTriArea(side1, side2);
     hypoteneuse = shapes.calcHypoteneuse(side1, side2);

     //print results
     System.out.printf(" Triangle Area = %8.2f%n", triArea);
     System.out.printf("   Hypoteneuse = %8.2f%n", hypoteneuse);
  }
}
```