AP Computer Science A – Unit 05: Writing Classes English Name: ____________________

Worksheet: Introduction to Writing Classes, part 1

In our lecture, we discussed using a class to organize code by separating methods that do a particular job. We discussed the implementation of a `Math` class using the example code given below.

***Code Block 1**: Code implementing partial functionality of the Java `Math` class*

```
public class Math {
   public static final double PI = 3.14159265358979323846;
   public static double abs(double a) {
      if(a >= 0) {
         return a;
      } else {
         return -a;
      }
   }
}
```

A student wished to program a class to contain useful constants and methods that he will later use in his program that will perform Chemistry calculations. To start, he decides the class should contain a constant named `AVOGADROS_NUMBER` with the value `6.02214076e23`, and a method `molesToParticles` that will take in as a parameter the number of moles of the substance as a `double` value, and return the number of particles of the substance, also as a `double` value. Note that Avagadro's number is the number of particles in a mole of substance, so one need only multiply the number of moles by Avagadro's number to get the number of particles.

1. Using the `Math` class code, above, as a template, write a class named `Chemistry` that contains the constant `AVAGADROS_NUMBER` and method `molesToParticles` as described above.

```
public class Chemistry {
   public static final double AVOGADROS_NUMBER = 6.02214076e23;
   public static double molesToParticles(double moles) {
      return moles * AVOGADROS_NUMBER;
   }
}
```

2. Write a class, named `TestChemistry` that calls `molesToParticles` with a reasonable parameter value, then prints the results returned by the method. Recall how `Math` class methods are called.

```
public class TestChemistry {
   public static void main(String[] args) {
      double particles = Chemistry.molesToParticles(3.0);
      System.out.println("Three moles contains " +
                          particles + " particles.");
   }
}
```

When ***fields*** (*attributes*) are modified with the `static` modifier, there is only one value stored per class, and when ***methods*** (*operations*) are modified with the `static` modifier, they can only read and modify the `static` fields, and not read nor modify any non-`static` fields.

Recall we discussed an example `Vector` class, for which the example code is given below.

Code Block 2*: Third draft of the* `Vector` *class*

```
public class Vector {
   public double x;
   public double y;
   public Vector(double x, double y) {
      this.x = x;
      this.y = y;
   }
   public void add(Vector v) {
      this.x += v.x;
      this.y += v.y;
   }
}
```

English Name: ____________________

3. Write a class named `Coordinate3d` that contains three fields of type `double` that will store the coordinates of a point in three dimensions using the *fields*: `x`, `y` and `z`. Write a constructor that will initialize the class, and a method named `distance` that takes no parameters and returns the distance of the point from the origin at `(x, y, z) = (0, 0, 0)`. Recall that distance, `d`, is given by the formula, $d^2 = x^2 + y^2 + z^2$. Also write a separate class named `TestCoord3d` that will test both the constructor and the `distance` method.

```
// Write the Coordinate3d class here:


public class Coordinate3d {
   private double x;
   private double y;
   private double z;
   public Coordinate3d(double x, double y, double z) {
      this.x = x;
      this.y = y;
      this.z = z;
   }
   public double distance() {
      return Math.sqrt(x*x + y*y + z*z);
   }
}
```

```
// Write the TestCoord3d class here:


public class TestCoord3d {
    public static void main(String[] args) {
        Coordinate3d p = new Coordinate3d(6, 4, 4);
        System.out.println("Distance = " + p.distance());
    }
}
```