Classes, Version 2

```
public class Pointv2
{
        Pointv2() // default constructor, if parameters are not specified with key-word new
        {
        x = 0;
        y = 0;
        }

        Pointv2(int a, int b)
        {
        x = a;
        y = b;
        }

        public void moveTo(int newX, int newY) // moveTo() is a mutator method
        {
        x = newX;
        y = newY;
        }


        // A better way to provide output -- think MVC; MVC => Model View Controller

        public String toString()
        {
        return "(" + x + ", " + y + ")";
        }

protected int x; // Data member
protected int y; // Data member
}



public class usePointv2 // driver program
{
        public static void main(String[] args)
        {
        Pointv2 p = new Pointv2(5,5);
        Pointv2 p1 = new Pointv2(4,4);

        System.out.println("Point p: " + p);   // implicitly calls p.toString()
        System.out.println("Point p1: " + p1);


        System.out.println("Move p1 to (2,2)... \n");
        p1.moveTo(2,2);
        System.out.println("Point p1 (after moveTo()): " + p1 + '\n');
        }
}
```

Sample run:

```
$ java usePointv2
Point p: (5, 5)
Point p1: (4, 4)
Move p1 to (2,2)...

Point p1 (after moveTo()): (2, 2)
```

Notes:

1. At this juncture, the above seems to be a step backward…

   a. Encapsulation is compromised, via ***protected*** instead of ***private***

2. However, the need for accessors has been removed by using toString()

3. In addition, the base class Pointv2 is better designed to support inheritance

4. Finally a satisfactory level of encapsulation can be performed, provided that the Java **package** concept is leveraged...

   In order to best perform encapsulation, when using *protected* (in Java) the base class Pointv2 and the derived class Ptv2 should be in one package – the driver program, usePtv2 should be in a separate package – many IDEs do this by default... More on this later in the semester

Speaking of inheritance, see example based on the PointV2 on the next page…

Inheritance, without accessors

```
public class Ptv2 extends Pointv2
{
        public Ptv2()
        {
        super(0,0);
        }

        public Ptv2(int a, int b)
        {
        super(a,b);
        }

        private double square(int k)
        {
        return k*k;
        }

        public double distance(Pointv2 p)
        {
        double dist = Math.sqrt( square(x - p.x) + square(y - p.y) );  // p.x and p.y are permitted w/protected
        return dist;
        }
}



public class usePtv2
{
        public static void main(String[] args)
        {
        Pointv2 p = new Pointv2(5,5);
        Ptv2 p1 = new Ptv2(4,4);
        Ptv2 p2 = new Ptv2(9,9);

        // With accessors: System.out.println("Point p: " + "(" + p.getX() + "," + p.getY() + ")");
        // With accessors: System.out.println("Point p1: " + "(" + p1.getX() + "," + p1.getY() + ")");

        System.out.println("Pointv2 p: " + p);
        System.out.println("Ptv2 p1: " + p1);
        System.out.println("Ptv2 p2: " + p2 + '\n');

        p1.moveTo(3,1);

        // With accessors: System.out.println("Point p1, after the move: " + "(" + p1.getX() + "," + p1.getY() + ")");

        System.out.println("Point p1, after moveTo() message to new 3,1: " + p1);

        double theDistance = p1.distance(p2);
        System.out.println("The distance between p1 and p2 is: " + theDistance);
        }
}
```

Sample run:

```
    $ java usePtv2
    Pointv2 p: (5, 5)
    Ptv2 p1: (4, 4)
    Ptv2 p2: (9, 9)

    Point p1, after moveTo() message to new 3,1: (3, 1)
    The distance between p1 and p2 is: 10.0
```

Notes:

The above approach is a compromise...

- To achieve satisfactory encapsulation requires more advanced packaging

- However, accessors are unnecessary and inheritance better facilitated