More on Classes… Consider the following Point class, as previously discussed:

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

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

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

        public void print()
        {
        System.out.println("(" + x + ", " + y + ")");
        }

private int x;
private int y;
}
```

Next a simple driver program:

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

        System.out.print("Point p: ");
        p.print();

        System.out.print("Point p1: ");
        p1.print();

        p1.moveTo(2,2);
        System.out.print("After the moveTo() call, Point p1: ");
        p1.print();

        // Is this line below legal?????  Hint: Violates encapsulation

        // System.out.println("Point p1: " + "(" + p1.x + ", " + p1.y + ")");
        }
}
```

Notes:

An improvement from the original Classes entry on the class website, but could be better, e.g.:

1. The accessors getX() and getY() have been removed... Why?

   a. Violates the concept of Information Hiding

2. The print() method could be improved... Current version requires extra System.out.print()/println()s and violates other principles -- see next page for details…

Sample run:

```
% java usePoint
Point p: (5, 5)
Point p1: (4, 4)
After the moveTo() call, Point p1: (2, 2)
```

A Revised/Better Approach…

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

        Point_Revised(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 + ")";
        }

private int x;  // Data member
private int y;  // Data member
}
```

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

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

        p1.moveTo(2,2);
        System.out.print("Point p1 (after moveTo()): " + p1);
        }
}
```

Sample Run:

```
% java usePoint_Revised
Point p: (5, 5)
Point p1: (4, 4)
Point p1 (after moveTo()): (2, 2)
```

Notes:

1. Use of MVC provides the following advantages:
    a. In Java this means use of toString() instead of a print() method
    b. Processing is INDEPENDENT from I/O
    c. In this example, there are fewer calls to System.out.print()/println()
    d. Promotes reuse, since it is unnecessary to modify processing modules to accommodate modifications to I/O, e.g.
        i. Command Prompt
        ii. Terminal Window
        iii. Windows Graphical User Input (GUI) program
        iv. GUI program on other systems, e.g. Macintosh, tablets, phones, etc.
        v. Others…
    e. Note use of toString(), how it relates to ***inheritance***, ***overriding*** (as opposed to ***overloading***) and MVC