

Lab 5: Creating Java Class Methods

Lab 5 Due Date: Monday October 10 at 11:59 pm

Background:

Lab 5 will be an exercise in creating the methods for a Java class and processing numbers. You will implement a class called **Point**. The class models the x and y coordinate of a position on a Cartesian plane: the internal representation (the measurement stored in the private data members) are two integers, representing the coordinate values, while the external representation (the measurement printed in the console window) is the same.

There is an existing class, **PointTest** that is already written and complete. Once you successfully finished the **Point** class, the program should run correctly, generating the output displayed below. You will also have access to the **InteractiveIO** class for console input and output.

Design:

The class has the following methods that need to be completed:

- **Default constructor** — sets the x and y values to 0
- **Second constructor (with parameters)** — accepts int parameters for an x and y and sets the passed values as appropriate to create the proper internal representation
- **set** — receives two String arguments for an x and a y; the point's internal representation is derived from these arguments
- **getX, getY** — two methods that return the x-value and the y-value of the Point object, respectively
- **add** — receives a Point object as an argument, adds the x and y of the passed object to the x and y of the Point object that invoked the method, and returns a new Point object with the total of the two times.

For example: If **point1** represents x = 5, y = 20 and **point2** represents x = 7, y = -10, then **point1.add(point2)** returns a Point object that represents x = 12, y = 10

- **distance** — receives a Point object as an argument, and returns a double value that represents the distance between two points, using the distance formula (distance = square root of $(x_2 - x_1)^2 + (y_2 - y_1)^2$)

For example: If **point1** represents x = 3, y = 4 and **point2** represents x = 1, y = 2, then **point1.distance(point2)** returns the double value 2.82843

- **toString** — returns a string with an external representation (x and y) of the Point object. The format for this output is "X = xx, Y = yy" (where xx is the x value and yy is the y value).
-

Lab 5: Model a Point

Download the [lab5.zip](#) file by clicking on this link. Save the zip file to your C:\Temp directory and unzip the files. Open the directory **lab5**. You should see the following files:

- PointModel.mcp (the project file for this Java application)
- PointTest.java (the source file that tests this Java application, already finished)
- Point.java (the source file for the Clock object)
- InteractiveIO.java (the class that supports console input and output)

Open the PointModel.mcp file and look at the filenames listed inside. PointModel.mcp is not a Karel or Java program file - it's project file CodeWarrior requires as part of every Java program. A project file keeps track of what files are parts of a Java program. Typically there is one project file for each Java program -- however there can be many Java source files comprising a single Java program. The program we are going to work with today has 3 source file: *PointTest.java*, *Point.java*, and *InteractiveIO.java*.

The task

Upon successfully completing the Point class, the program should run, generating output that corresponds to the sample output displayed below (assuming the same input values are used.)

Writing the PointModel program

For this exercise, all of your work will be done in **Point.java**. You must complete the **Point** class. The **PointTest** class and **InteractiveIO** have already been completed..

Sample Solution

Output of a correctly implemented Point class should look something like this:

```
Enter the x value: 2
Enter the y value: -3

p1: X = 2, Y = -3

p2: X = 5, Y = -5

p3: X = 7, Y = -8

Distance from p1 to p3 is 7.0710678118654755

Press Enter to continue
```

Handing in your Solution

Your solutions should be in the form of .zip files. When we grade your solution we will unzip the folder and execute the project. If your project does not run you will lose the execution points for that lab.

Your Solution zip file must contain all the files in your projects

Your teaching assistants will show you how to zip up the files for submission. Instructions are also in the [Tutorial](#) document that is available

Click on this link to [Submit](#) your zip file