Object-Oriented Programming

Exercise 1: Cuboids

MyPoint3D.java

MyPoint3D class consists of three data members: the integer x-, y- and z-coordinates of a 3D point. These three members of MyPoint3D class have been included in the provided partial code, and you should **NOT** change them. Neither should you add new data members.

You should add these methods:

- Two constructors
  - MyPoint3D(): To construct a point (0,0,0).
  - MyPoint3D(int, int, int): To construct a point with the x-, y- and z-coordinates specified in the parameters.

- Mutators setX, setY and setZ to set the respective coordinator of a point.
- Accessors getX, getY and getZ to retrieve the respective coordinator of a point.
- Overriding toString method to convert a point object to a string representation.

MyCuboid.java

MyCuboid class defines a right-angled cuboid where its edges are parallel to one of x-, y- or z-axis. The class consists of two data members: the two opposite vertices of a cuboid. Each vertex is an object of class MyPoint3D. Figure 1 below shows a cuboid with its two opposite vertices highlighted.

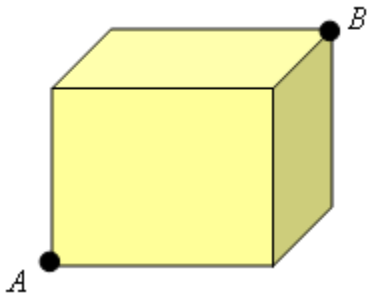

Figure 1. Cuboid with opposite vertices *A* and *B*.

The data members of MyCuboid class have been included in the provided partial code, and you should **NOT** change them. Neither should you add new data members.

You should add these methods:

- Two constructors
  - MyCuboid(): To construct a cuboid with opposite vertices (0,0,0) and (1,1,1).
  - MyCuboid(int,int,int,int,int,int): To construct a cuboid with vertices whose coordinates are specified by the parameters. The first three integer parameters are

the x-, y- and z-coordinates of the first vertex, and the next three integer parameters are the coordinates of the second vertex.

- Mutators setVertex1 and setVertex2 to set the respective vertex of a cuboid.
- Accessors getVertex1 and getVertex2 to retrieve the respective vertex of a cuboid.
- Overriding toString method to convert a MyCuboid object to a string representation.

Lab6Ex1.java

The program Lab6Ex1.java is to create two MyCuboid objects and computes the volume of their common space.

The main() method and the signatures of the two methods readCuboid() and commonVolume() are given in the provided partial code, and you should **NOT** change them.

Please note that you should create only one instance of Scanner. (If multiple instances of Scanner are created, you will encounter problem with CourseMarker.) As illustrated in the partial code, a single scanner object is created in the main() method, and passed into readCuboid() method.

Sample run using interactive input (user's input shown in blue; output shown in **bold purpl**

```
$ javac MyPoint3D.java
$ javac MyCuboid.java
$ javac Lab5Ex1.java
$ java Lab5Ex1
Enter 1st cuboid: -10 20 10 20 0 -5
Enter 2nd cuboid: 10 30 -10 60 18 -3
1st cuboid: [ (-10,20,10); (20,0,-5) ]
2nd cuboid: [ (10,30,-10); (60,18,-3) ]
Volume of common space = 40
```

Another sample run:

```
$ java Lab5Ex1
Enter 1st cuboid: 1 2 3 4 5 6
Enter 2nd cuboid: 6 5 4 3 2 1
1st cuboid: [ (1,2,3); (4,5,6) ]
2nd cuboid: [ (6,5,4); (3,2,1) ]
Volume of common space = 3
```

Exercise 2: Overlapping Rectangles

You are given this code:

- MyRectangle.java (partial code)

Two rectangles *A* and *B*, whose sides are parallel to the x-axis and y-axis, can be found to be in one of the following 3 cases:

- Case 1: Rectangles *A* and *B* overlap each other.
- Case 2: Rectangles *A* and *B* touch each other.
- Case 3: Rectangles *A* and *B* are disjoint.

Figure 2 below shows the various positions of the rectangles in the 3 cases.

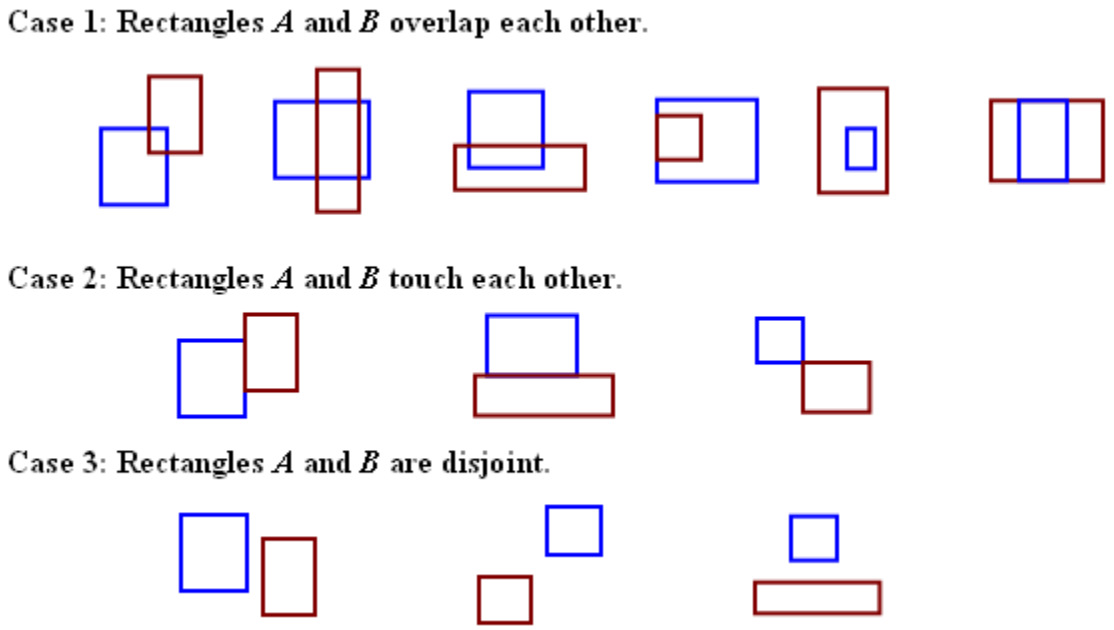

Figure 2. Two rectangles that overlap, touch, or are disjoint.

Write a program MyRectangle.java to create the MyRectangle class that includes two data members: the bottom-left corner and the top-right corner of the rectangle, both of which are objects of the Point2D.Double class. You are also to design relevant methods for the following requirements.

The **main method** in your program should create 2 MyRectangle objects from user's inputs, print the rectangles' information, and output a conclusion that states whether the two rectangles **overlap**, **touch**, or are **disjoint**. Please refer to the 3 sample runs below.

If the two rectangles overlap each other, you need also to compute and display the overlapped area. Please refer to the first sample run below for an example.

You are to note that in the creation of a rectangle, the user may enter ANY 2 opposite corners of the rectangle, not necessarily the bottom-left corner and top-right corner. Therefore, you need to check the values and make the necessary adjustments before creating the rectangle.

You may assume that the user enters two distinct points for the corners, and that the rectangle has a positive area.

You may write any additional methods you deem necessary. A more modular program will receive more credit than a less modular one.

2.3 Sample runs

Sample run using interactive input (user's input shown in blue; output shown in **bold purple**):

```
$ javac MyRectangle.java
$ java MyRectangle
Enter one corner of 1st rectangle: 7.6 2.2
Enter opp corner of 1st rectangle: 1.5 0.3
Enter one corner of 2nd rectangle: 9.4 -2.5
Enter opp corner of 2nd rectangle: 4 4.2
Rectangle A = ([1.5, 0.3], [7.6, 2.2])
Rectangle B = ([4.0, -2.5], [9.4, 4.2])
Rectangles A and B overlap each other.
Overlapped area = 6.84
```

Second sample run:

```
$ java MyRectangle
Enter one corner of 1st rectangle: 33.3 -1.1
Enter opp corner of 1st rectangle: 20.5 3.6
Enter one corner of 2nd rectangle: 20.5 8.6
Enter opp corner of 2nd rectangle: 10.3 -5.2
Rectangle A = ([20.5, -1.1], [33.3, 3.6])
Rectangle B = ([10.3, -5.2], [20.5, 8.6])
Rectangles A and B touch each other.
```

Third sample run:

```
$ java MyRectangle
Enter one corner of 1st rectangle: 4 0
Enter opp corner of 1st rectangle: 9 5
Enter one corner of 2nd rectangle: 0 11
Enter opp corner of 2nd rectangle: 5 6
Rectangle A = ([4.0, 0.0], [9.0, 5.0])
Rectangle B = ([0.0, 6.0], [5.0, 11.0])
Rectangles A and B are disjoint.
```

--- The end ---