Problem Set 4 Exercise #34: Overlapping Rectangles [Challenge]

Reference: Lecture 10 OOP Unit 3 notes

Learning objectives: Object-oriented programming; Algorithm design

Estimated completion time: 120 minutes

Problem statement:

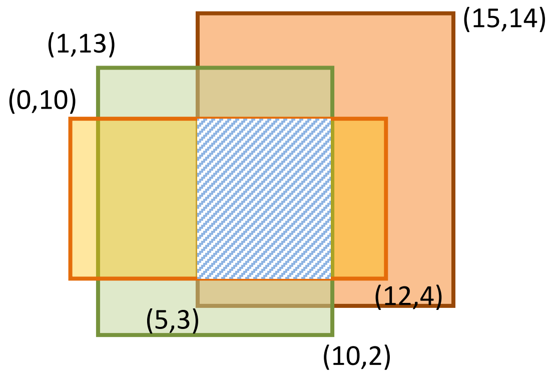
[CS1020 home assignment]

We can represent a rectangle (whose sides are parallel to the *x*-axis or *y*-axis) with its two opposite vertices, where each vertex is a **`java.awt.Point`** object. For example, a rectangle with vertices at (3, -1), (3, 6), (15, 6) and (15, -1) may be represented by any of these 4 pairs of points: (3, -1) and (15, 6); (15, 6) and (3, -1); (3, 6) and (15, -1); or (15, -1) and (3, 6).

In **PS4_Ex34_Rectangle.java**, write a **`Rectangle`** class that contains two attributes: **`vertex1`** and **`vertex2`** which are the two opposite vertices of a rectangle. Fill this class with appropriate constructor and member methods.

Write a user program **PS4_Ex34_OverlapRectangles.java** to read in an integer indicating the number of rectangles. You may assume that this value is at least 1. Your program then reads the value of each rectangle, and computes the overlap area of all the rectangles.

For example, given the 3 rectangles shown below (not drawn to scale), the final overlap, which is also a rectangle (in grey colour), has vertices at (5, 4) and (10, 10), with an area of 30.



Sample run #1:

```
How many rectangles? 2
Enter 2 opposite vertices of first rectangle: 6 6 10 10
Enter 2 opposite vertices of next rectangle: 7 7 8 8
Overlap rectangle: [(7, 7); (8, 8)]
Overlap area = 1
```

Sample run #2:

```
How many rectangles? 3
Enter 2 opposite vertices of first rectangle: 1 13 10 2
Enter 2 opposite vertices of next rectangle: 15 14 5 3
Enter 2 opposite vertices of next rectangle: 12 4 0 10
Overlap rectangle: [(5, 4); (10, 10)]
Overlap area = 30
```

Sample run #3:

```
How many rectangles? 2
Enter 2 opposite vertices of first rectangle: 6 6 10 10
Enter 2 opposite vertices of next rectangle: 1 1 8 5
No overlap
```

Challenge yourself by not reading the tips at the bottom of this page ☺

Useful tips:

1. When you read the second rectangle, compute the overlap of it with the first rectangle. This overlap is itself a rectangle. When you read the third rectangle, compute the overlap of it with the previous overlap. Hence, each time a next rectangle is read, you determine the new overlap rectangle. After reading all the rectangles, you have the final overlap and you can then easily compute the area of this overlap.
2. Note that the user may enter any pair of opposite corners of a rectangle and thus the logic to compute the overlap area becomes rather complex. Hence it pays to rearrange the two vertices entered by the user such that the two corners stored in the **`Rectangle`** class are the bottom-left and top-right corners. In this way, the rest of the program is much easier to write. Such **pre-processing** step is very common in algorithmic problem solving. Once the data are arranged in a certain desired manner, subsequent computation can be greatly simplified.
3. Use **`Math.min()`** and **`Math.max()`** methods in place of long-winded if-else statements.
4. Be aware of the *degenerate case* where two rectangles have no overlap, i.e. the overlap is an "empty" rectangle. See sample run #3 for an example. How would you want to represent such a rectangle object in your program so that the final computation is correct?