Study the Intersect Program

Obtain a copy of the program Intersect.java. Try it out. Figure out how it works. PLEASE NOTE: this program has errors ("bugs").

Draw a UML Diagram

Just so you don’t forget how to do it, neatly draw the UML diagram for the Intersect class. Remember that a UML diagram shows the instance variables and the methods, including the constructor. For methods, it shows the types of parameters and the type of value returned. Please imitate the style of the UML diagrams in the book.

Note that the UML diagram includes only methods and instance variables that are defined in the class Intersect; it does not include methods of other classes that are called from within the Intersect class.

A Few Words about Methods

This program has been designed specifically to show off how you can use methods to build up a complex program from a series of simpler pieces. The basic idea is as follows:

- If two rectangles intersect, then a point of one rectangle must lie inside the other rectangle. It is not practical to test every point inside each rectangle (there are theoretically an infinite number of points, but even if we limit ourselves to pixel values there are still 10,000 points in a 100-by-100-pixel rectangle). So instead of checking every point, we will just check the corners. We have to consider two cases: either a corner of rectangle 1 is inside rectangle 2, or vice-versa. (Of course, both could be true). Thus, we make two calls to “cornerInside”.

- To see if a corner of a rectangle lies inside another, we have to check several corners. Therefore, we have to make several calls to the method pointInside.

- Method pointInside just checks to see if a given point lies inside a given rectangle.

There are easier ways to write this program! But this shows off the technique of breaking a problem up into smaller problems and solving the problem by a series of method calls.
Generate Test Data

To thoroughly test a computer application you must try out all the possibilities that can occur. For instance, it would not be enough to try just the following two cases:

- Rectangle 1: location 50 50, height 100, width 100
  Rectangle 2: location 75 75, height 100, width 100

- Rectangle 1: location 50 50, height 100, width 100
  Rectangle 2: location 200 200, height 100, width 100

It is true that one of these cases has intersecting rectangles and the other does not, but as I mentioned above, this program has errors in it that may cause it to sometimes give the wrong answer, even if it works correctly in some cases.

Construct a complete set of test cases. Consider all possibilities — e.g., two rectangles share a single corner, one rectangle is completely inside the other, etc. Draw your cases on paper, with the locations and sizes indicated. For each test case, indicate what the correct answer should be and what the program says the answer is.

Fix the Program

Make this program work for all possible rectangles! Hand in your UML diagram, your test data, and your modified code.

Checklist:

- Hand in a neatly drawn UML diagram of the original program. Note that your final program might have a different UML diagram if you change or add methods; I’m asking for the UML of the original program.

- Hand in a neatly written up set of test data showing a complete set of test cases (drawn as rectangles, showing locations and sizes), the correct answers, and the original program’s answers.

- Hand in the corrected program. Header comments should include your name, the lab number, the date, and a brief description of what you modified.

- Descriptive comments about any code you added or changed.

- Sign your name on the program for the honor code.