NAME: **login ID:**
precept:

_

COS 126 Midterm 2 Programming Exam, Fall 2010

This exam is like a mini-programming assignment. You will create two classes, and a client, compile them with provided files, and run it with the provided test data on your laptop. Debug your programs as needed. This exam is open book, open browser. You may use code from your assignments or code found on the COS126 website. When you are done, submit your program via the course website using the submit link for Precept Exam 2 on the Assignments page.

Grading. Each programs will be graded on correctness, clarity (including comments), design, and efficiency. You will lose a substantial number of points if your programs do not compile or if they crash on typical inputs.

Print your name, login ID, and precept number on this page (*now*), and write out and sign the Honor Code pledge before turning in this paper. *Note*: It is a violation of the Honor Code to discuss this midterm exam question with anyone until after everyone in the class has taken the exam. You have 50 minutes to complete the exam.

"I pledge my honor that I have not violated the Honor Code during this examination."

_____

Signature

1	/21
2A	/14
2B	/5
Total	/40

December 16, 2010

Part 1 (21 points)

Files. This part of the assignment uses `StdIn.java`, `StdOut.java`, and the data file `tiny1D.txt`, so *you must have a directory for this exam and download these three files (two .java files and the data file) from the course assignments page before you begin.*

Your first task. Write a Java class `Range1D.java` for 1-dimensional ranges (intervals). Specifically, you must implement the following public API:

```
public class Range1D
```

`public Range1D(double lo, double hi)`	Constructor for the range (*lo*, *hi*).
`public boolean intersects(Range1D that)`	Does this range intersect `that`?
`public double mid()`	Midpoint of this range.
`public double size()`	Length of this range.
`public String toString()`	String representation of this range.

You may assume that `lo` is less than `hi` and that both are greater than 0 and less than 1. Two ranges *intersect* if they have more than a point in common. Note that the ranges are open intervals (endpoints are not included) so that, for example (.1, .2) and (.2, .3) do not intersect.

Test client. Develop a test client `main()` for `Range1D` that reads ranges from standard input in pairs and prints whether the pair of ranges is disjoint or whether they intersect. For example, the input file `tiny1D.txt` specifies six pairs of ranges on six lines, as follows:

```
.1 .3  .7 .9          (.1, .3) and (.7, .9)
.1 .7  .3 .9          (.1, .7) and (.3, .9)
.2 .3  .2 .3          (.2, .3) and (.2, .3)
.1 .2  .2 .3          (.1, .2) and (.2, .3)
.3 .9  .1 .7          (.3, .9) and (.1, .7)
.7 .9  .1 .3          (.7, .9) and (.1, .3)
```

For this input, your program should produce the following output:

```
% javac Range1D.java
% java Range1D < tiny1D.txt
(0.1, 0.3) and (0.7, 0.9) are disjoint
(0.1, 0.7) intersects (0.3, 0.9)
(0.2, 0.3) intersects (0.2, 0.3)
(0.1, 0.2) and (0.2, 0.3) are disjoint
(0.3, 0.9) intersects (0.1, 0.7)
(0.7, 0.9) and (0.1, 0.3) are disjoint
```

This test client does not test `mid()` and `size()`, but you must include implementations of these methods to get full credit for Part 1. Submit your program `Range1D.java` via the link on the Assignments page. DO NOT GO TO THE NEXT PAGE UNTIL YOU HAVE SUBMITTED.

Part 2A. (14 points) *Do not attempt this part unless you have completed Part 1 successfully.*

Files. This part of the assignment uses `StdIn.java`, `StdOut.java`, `StdDraw.java` and the data files `tiny2D.txt` and `large2D.txt`, so *you must have a directory for this part and download these six files (three .java files and two data files) from the course assignments page before you begin. You will not need Queue.java and large2D.txt until Part 2B, but download them now.*

Write a Java class `Range2D.java` for 2-dimensional ranges. Specifically, you must implement the following public API:

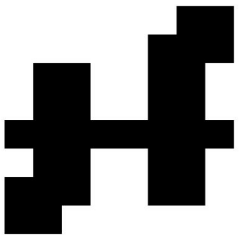
```
public class Range2D
```

`public Range2D(Range1D h, Range1D v)`	Constructor for the set of points (*x*, *y*) with *x* in `h` and *y* in `v`.
`public boolean intersects(Range2D that)`	Does this range intersect `that`?
`public void draw()`	Draw a rectangle that shows this range.
`public String toString()`	String representation of this 2D range.

Again, two 2D ranges intersect if they have a point in common. For `toString()`, print the two 1D ranges within parentheses, separated by a comma.

Test client. Develop a test client `main()` for `Range2D` that reads 2D ranges from standard input and draws them with `StdDraw`. Use `StdDraw.filledRectangle()` to draw the rectangles. It takes four arguments, the coordinates of the center, half the width, and half the height (this is where you use `mid()` and `size()` from Part 1).

The data files for 2D ranges will have four numbers per line, two for the horizontal range and two for the vertical range. For example, `tiny2D.txt` specifies five 2D ranges on five lines, as shown at right below. Your goal for this part is to produce the drawing shown at left.



`.1 .3 .1 .3`	*x* in (.1, .3) and *y* in (.1, .3)
`.7 .9 .7 .9`	*x* in (.7, .9) and *y* in (.7, .9)
`.6 .8 .2 .8`	*x* in (.6, .8) and *y* in (.2, .8)
`.2 .4 .2 .7`	*x* in (.2, .4) and *y* in (.2, .7)
`.1 .9 .4 .5`	*x* in (.1, .9) and *y* in (.4, .5)

This test client does not test `intersects()` and `toString()`, but you must include implementations of these methods to get full credit for Part 2A. Submit your completed program `Range2D.java` via the link on the Assignments page.

Part 2B. (5 points) *Do not attempt this part unless you have completed Part 2A successfully.*

Write a `Range2D` client named `Intersections.java` that includes a single static method `main()` that tests all of the methods in `Range2D`, as follows: Read and draw ranges from standard input, as before (you may use your code from Part 2A as a starting point) , but then print all pairs of intervals that intersect (if the number of ranges on standard input is less than 70), and the number of pairs that intersect. To accomplish this task, save the ranges on a (generic) `Queue`, since you do not know how many of them will appear on standard input, and use the *foreach* construct to iterate through them. For `tiny2D.txt` your program should produce the output below.

```
% java Intersections < tiny2D.txt
((0.1, 0.3), (0.1, 0.3)) intersects ((0.2, 0.4), (0.2, 0.7))
((0.7, 0.9), (0.7, 0.9)) intersects ((0.6, 0.8), (0.2, 0.8))
((0.6, 0.8), (0.2, 0.8)) intersects ((0.7, 0.9), (0.7, 0.9))
((0.6, 0.8), (0.2, 0.8)) intersects ((0.1, 0.9), (0.4, 0.5))
((0.2, 0.4), (0.2, 0.7)) intersects ((0.1, 0.3), (0.1, 0.3))
((0.2, 0.4), (0.2, 0.7)) intersects ((0.1, 0.9), (0.4, 0.5))
((0.1, 0.9), (0.4, 0.5)) intersects ((0.6, 0.8), (0.2, 0.8))
((0.1, 0.9), (0.4, 0.5)) intersects ((0.2, 0.4), (0.2, 0.7))
4 intersections
```

Note that each intersection appears twice, so that the count is half the length of the list.

For the input file `inputFun.txt`, you should get the drawing to the left and 77 intersections.

Submit your completed program `Intersections.java` via the link on the Assignments page.