CIS 36A, Spring 2015
Berkeley City College, Nate Titterton

LAB 1 (part B)

Create a project named `lab1` for your work. (You can copy your files from part A into this project so that you have one thing to submit for lab 1 work). Make a package to contain your Java files.

Framework code and this document are available at in the course moodle site. The `Line1` class is included on the last page.

(As you do these problems, don't worry about 'divide-by-zero' errors. You'll learn to handle situations like these later in the course.)

1. A line

Recall that a unique line can be defined by a pair of numbers `(x,y)` for one point and a pair of numbers for the another point; there is only one line that passes through those points.

Consider the `Line1` class in your framework code. This classes defines a line using four integers. The methods have been written for you, but you need to test them. In fact, there just might be a bug!

Fill in the `Line1.main()` method as directed by the inline comments.

2. A line stored differently

A line can also be defined by a y-intercept `b` (the value on the y-axis that it passes through) and a slope `m`, or *rise over run*. In this step you are going to write a Line2 class that uses *slope* and *intercept* to define a line. Use an integer for the intercept, but not for slope, which will need a fractional (decimal) part.

Consider the `Line2` class in your framework code, which is a straight duplicate of `Line1`. You'll need to change the instance variables to use a slope and intercept. This will break the methods in the class: you'll need to fix these to use your new instance variables. Some notes:

- In your `angleTo()` method, use a parameter of type `Line1` rather than the new `Line2`. That is, you'll figure out an angle between lines stored in two different ways. It should be *very* easy to convert the `angleTo()` method from the `Line1` class because it doesn't directly use a line's instance variables, but rather uses only the `slope()` method!
- With `pointInLine()`, you can still use the same logic as in `Line1`: comparing two slopes to make sure they are equal. One point that is on the line is the y-intercept—that is, the point `(0,b)` where `b` is the y-intercept—you can use that point to calculate one slope. For the other slope, why not just use the slope of the line encoded in your `Line2` object

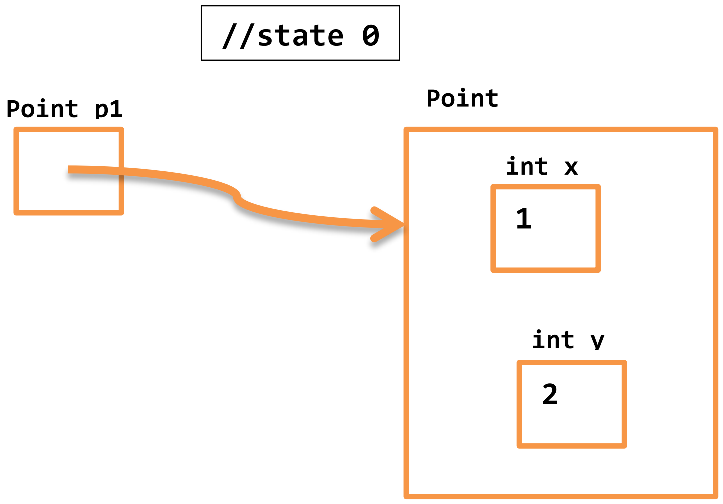
Test your class with the `Line2.main()` method, filling in as directed by the inline comments.

3. Points

Check out the Javadoc page for **`java.awt.Point`**: http://docs.oracle.com/javase/7/docs/api/java/awt/Point.html. We'll use it to make another version of the Line1 class. But first, consider the following Java code snippet (in the **`PointTest`** class in your framework code):

```
Point p1 = new Point();
p1.x = 1;
p1.y = 2;
// state 0
Point p2 = new Point();
p2.x = 3;
p2.y = 4;
// state 1
p2.x = p1.y;
// state 2
p1 = p2;
// state 3
p2.x = p1.y;
// state 4
System.out.println(p1.x + " " + p1.y
        + " " + p2.x + " " + p2.y);
```

There are five comments that start with "state"; you need to sketch the box-and-pointer diagram of the variables in this snippet at each of the comments. (We'll figure out how to turn this in later) Here is the first one to get you started:



4. Getting to the point now.

Fill in the **`Line3`** class. Note that it uses two **`Point`** objects instead of four **`int`**s, which will slightly change the bodies of its methods relative to what was done in **`Line1`**.

Test your class with the **`Line3.main()`** method.

5. What's in a name?

Write a class **`Name`** with the following methods:

- **`void setName(String firstName, String secondName, int gender)`**

  Sets the name stored in this object. A gender of 1 is male, 2 is female (think of this as encoding the number of X chromosomes!)

- **`String name()`**

  Returns the name (first name followed by last name). The first letter only will be capitalized in each name part.

- **`String rollCallName()`**

  Returns the last name, followed by a comma, followed by first name. The first letter only will be capitalized in each name part.

- **`String initials()`**

  Returns the initials as a single, two-character, capitalized String.

- **`String pigLatinName()`**

  Returns the pig latin version of the name, all lower case. See http://en.wikipedia.org/wiki/Pig_Latin#Rules for rules. For instance,

  - **`Nate Titterton`** should turn to **`atenay ittertontay`**
  - **`Evan Johnson`** should turn to **`evanway ohnsonjay`**

In your static **`main()`** method, include enough test calls to your **`Name`** class to ensure it works correctly.

Line1:

```
package lines;

// Line class with points stored
public class Line1 {

    int x1, y1, x2, y2;

    // returns the slope of the line.  This is a ratio of rise over run.
    double slope() {
        double slope = (y1 - y2)/(x1 - x2);
        return slope;
    }

    // returns true if the point given as parameters is on the line
    // algorithm: check that the lines between this point and our
    //   other points have the same slope.
    boolean pointOnLine(int x, int y) {
        // first slope is between the passed in point and x1,y1
        double slope1 = (y-y1)/(x-x1);
        // slope between parameter point and x2,y2
        double slope2 = (y-y2)/(x-x1);
        // if the slopes are equal, then the point is on the line
        return (slope1==slope2);
    }


    // who remembers this stuff?  http://planetmath.org/anglebetweentwolines does.
    double angleTo(Line1 otherLine) {
        double differenceInSlopes = slope() - otherLine.slope();
        double denominator = 1 + ( slope() * otherLine.slope() );
        double angleInDegrees = Math.abs( differenceInSlopes / denominator );
        return angleInDegrees;
    }



    // Use main() to test your Line1 class
    public static void main(String[] args) {
        System.out.println("testing Line1");
        // Set myLine to contain a reference to a new line object.
        Line1 myLine = new Line1();
        // Initialize myLine's x1 and y1 to the point (5, 10),
        // and initialize myLine's x2 and y2 to the point (45, 50).

        // Print the line's slope, which should be 1 (or, 45 degrees).

        // check that (10,15) is on the line

        // check that the angle between this line and another line
        //  defined by (5,10) and (10,20) is about 1/3
    }
}
```