Project 4: Lines and More Lines

1 Objective

Now that we've created and used basic objects, we're moving forward to deeper object interactions. We're adding in two new concepts as well: reading data from a file and storing objects in an array.

This program has no user interface except for drawing panel output (in the extra credit portion) but will certainly have unit tests that ensures all objects/methods are behaving as intended.

2 Lines

2.1 Equations

As you may remember from your math studies, lines can be represented by the equation:

$$y = mx + b$$

where *m* represents the slope of the line and *b* represents the intercept (the place at which the line crosses the y axis). See this site for a review of the basic concepts.

2.2 Intersections

To find the point of intersection of two lines, substitute one line's *mx+b* value for the other equation's y value, then solve for x. For example, find the intersection of lines y = 4x – 1 and y = –x +19:

$$4x - 1 = -x + 19$$
$$5x = 20$$
$$x = 4$$

You then find the y coordinate by substituting the now-known x: y = 4x – 1 → y = 4(4) – 1 → 15. So, the intersection of the two lines is at (4,15). See this site for a review of finding the intersection of two lines. You'll need to find a way to do this programmatically, translating the manual steps into code.

3 Brightness

3.1 Calculating Based on RGB

Brightness is calculated based on the red, green, and blue components of a color. The equation for calculating it is as follows; *note that the multiplier isn't squared, only the R, G, and B values are*.

$$B = \sqrt{0.241R^2 + 0.691G^2 + 0.068B^2}$$

3.2 Non-decreasing Brightness Sequence

Looking at a sequence of brightness values, you'll be asked to find and return the longest sequence of non-decreasing values. For example, given this sequence: 67.8, 91.3, **56.3, 87.3, 87.3, 90.4**, 55.0, the sequence that would be returned is the sequence of values shown in boldface. Important: in this project, you'll be asked to return an *array of Line object references*—not values—representing this sequence.

4 Input File

4.1 File

The input file will contain the following elements, in order:

- On the first line, an integer indicating how many Lines are represented in the file.
- One additional data line per Line, each having doubles representing slope and y intercept followed by three integers representing the red, green, and blue components of the line's color. The color elements are in the range of 0 to 255.
- Data elements are separated by spaces.

4.2 Example Contents

```
2
4.0 -2.0 240 56 33
1.0 5.0 0 200 175
```

The example file's contents are interpreted as follows:

- There are two Lines represented in the file
- The first line's equation is y = 4x – 2 and has color R=240, G=56, and B=33.
- The second line's equation is y = x + 5 and has color R=0, G=200, B=175.

4.3 Notes

- Create the file in notepad or in another text editor. If on the Mac and using TextEdit, make sure you set the editor to Plain Text mode (a setting, not a save option) to ensure the file is only simple text.
- If you rely on text files for testing (this is expected), don't forget to include them in your submission.
- You are not responsible for bad data; if the file isn't created correctly and it blows up your program, that's okay for now.

5 Objects You'll Create

5.1 Objects

Here are UML Class Diagrams for the classes you are to create. Create *exactly* these methods, with *exactly* the parameters and return types shown. See the next page for other important notes.

Line

You decide what goes here, but remember this is class global data, so use only what is necessary

Constructors
+Line()
+Line(slope : Double, intercept : Double,
color : Color)
Accessors
+getSlope() : Double
+getIntercept() : Double
+getXIntercept() : Double
+getColor() : Color
+brightness() : Double
+toString() : String
Mutators
+setSlope(slope : Double)
+setIntercept(intercept : Double)
+setColor(color : Color)
Other Methods
+findIntersect(otherLine : Line) : Point

LineSet

You decide what goes here, but remember this is class global data, so use only what is necessary

Constructors
+LineSet(File lineFile)
Accessors
+lineCount() : Integer
+maxYIntercept() : Double
+minYIntercept() : Double
+maxXIntercept() : Double
+minXIntercept() : Double
+avgBrightness() : Double
+getLine(lineIndex : Integer) : Line
+toString() : String
Mutators
(none)
Other Methods
+nonDecrBrightSeq() : Line[]
+sortBySlope()
+drawLines() (extra credit)

Point

-x : Double
-y : Double

Constructors
+Point()
+Point(x : Double, y : Double)
Accessors
+getX() : Double
+getY() : Double
+toString() : String
Mutators
+setX(newX : Double)
+setY(newY : Double)
Other Methods
(none)

5.2 Notes

5.2.1 Line

- findIntersect() should throw an IllegalArgumentException if the two lines are parallel.
- For the empty constructor, use slope = 0.0, intercept = 0.0, and Color.BLACK for defaults.

5.2.2 LineSet

- The constructor throws a FileNotFoundException if the file isn't found or an IllegalStateException if there are fewer than 2 lines in the set. It reads in the lines, creates Line objects, and fills in the array of Line object references.
- Think carefully about the private member data you choose to store; just because you can compute it doesn't mean you need to store it. Consider that later we might add, delete, or manipulate individual lines, so your code would need to keep current all private data.
- Use a Scanner to read the text file; see the Hints section for more information.
- For sorting, find the class samples for the Insertion sort and adapt it for use here. You'll manually be moving around array elements during the sort operation.
- You don't need preconditions for array indexes, e.g., those passed to LineSet's getLine method. Why? Because the exception that the client will get is exactly right for the scenario; you can't improve on it.
- For **extra credit**, implement the drawLines() method. Use the provided methods for finding the minimum and maximum X and Y intercepts, add some padding (say 20% on each side), then draw the lines using a DrawingPanel. Also draw an X and Y axis for reference, though you don't need to label these in any way. This must be generic, working for *any* specified line set.

5.2.3 Point

You'll need a Point class that uses floating point values for x and y. The CSC142Point class in the course materials already provides that; I suggest adapting that for use here. There is no need to write additional test code for this unless you extend the functionality.

6 Code Implementation

Follow the Course Style Guide.

7 Testing

- Use the JUnit unit test framework as demonstrated in class and in videos.
- The testing required here is significant and shouldn't be an afterthought. I recommend writing the test function early and adding to it as you write each method.
- Don't forget to test all constructors as well as other methods.
- Don't forget to test preconditions and other code that is *supposed* to throw exceptions.
- You'll need to spot check that array contents before and after sorting are what you expect.
- To properly test LineSet's nonDecrBrightSeq you'll need a variety of test files. Create files with different lengths of brightness sequences and files with the longest sequence at the start and the end of the file. This is important because we know as programmers we often create off-by-one errors; these can only be found through careful debugging and testing.

8 Submitting Your Work

- If your test methods depend on test files, be sure and put them in your project folder.
- Use BlueJ to create a .jar file. Be sure and specify "include source" so your code will be included. Submit the .jar file.

9 Hints

- **Draw and refer to an object diagram!** Capture the moment where a file has been read and all the objects are created and related. This will be extremely helpful in writing code that navigates deep into the model.
- I suggest building objects from the ground up, e.g., start with the Point class coding, then move on to Line coding and testing, then finally LineSet code and testing. Incrementally develop your solution, building small additions on the solid foundation you've already laid.
- Don't duplicate code; avoid this wherever possible. As an example, don't forget it's legal for one constructor to call another constructor; this is good practice and helps avoid duplication and the creation of extra code paths to test.
- The algorithm for LineSet's nonDecrBrightSeq function requires a good deal of thought before coding. I highly recommend doing work on paper to see how it should work, then writing pseudocode, then turning that pseudocode into Java code.
- The text file is to be a *tokenized* file; that means that items can be separated by white space (spaces, tabs, carriage returns) and the code should work with any of those. When you use a Scanner to read the file, the work of reading the next token is done for you automatically.

10 Grading Matrix and Achievement Levels

Achievement	**Max Points**
Compile/runtime errors	50%
Line and Point present and working; no test method	65%
Add Line test method	70%
LineSeries present and working; no test method or brightness seq.	80%
Add LineSeries test method	85%
Add brightness sequence	90%
Documentation/Style	100%
Add drawLines()	105%