CS 145: Homework 3

Paragraph

1 Description

Your primary goals in this homework are to learn about 2-D dimensional arrays and gain more experience with loops, conditional statements, and file I/O. You will do all this in context of writing a graphing calculator for a special class of mathematical equations called parametric equations.

You will write two new classes for this homework: `Paragraph` and `Image`. Place these two classes in a package named `hw3`. Additionally, you will need a working `PostOps` class from preassignment 3. If you do not have one, entreat someone who got a perfect score to give you their copy. `PostOps` must stay in package `pre3`—do not move it to `hw3`.

2 Background

2.1 Parametric equations

1-D parametric equations express curves as functions of a single parameter. We will call this parameter p. To model a circle, we can treat p as an angle. It's x- and y-coordinates can be then expressed with two familiar parametric equations (in postfix form):

$$\begin{aligned} x(p) &= p\ \cos \\ y(p) &= p\ \sin \end{aligned}$$

We want to walk along the entire perimeter, so we say that p has domain $[0, 2\pi]$ in radians, or $[0, 360]$ in degrees.

Projectile motion is also modeled with parametric equations. Suppose every 1 second, an object travels 1 unit on the x-axis, starting from the origin. We can express it's x-coordinate at time p as its x-velocity (1 unit/second) times the time p. It's y-position at time p is $-4.9p^2$. All together we have:

$$\begin{aligned} x(p) &= p \\ y(p) &= p\ p\ *-4.9\ * \end{aligned}$$

We can define the domain of p to be anything we want, but we probably shouldn't let time go negative: $[0, <$ your-favorite-number-here $>]$.

In this homework, you will read from a file one or more $(x(p), y(p))$ function pairs like this. It is your job to evaluate these functions at various values of p and color in the corresponding pixels in an image.

2.2 Digital images

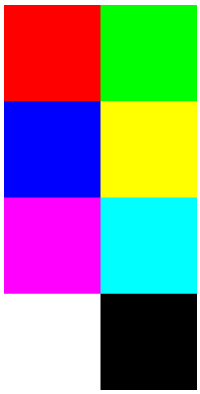
Digital images are often stored by programmers as 2-D arrays of pixel values. In a color image, each pixel is represented as a triplet of values, a blending of red, green, and blue intensities. Java provides a class for encapsulating this triplet: `Color`. We can assign the color orange (which is red at full intensity, green at half, and blue quietly sleeping in the corner) to the pixel in row 5, column 3 of a 2-D `Color` array with:

```
pixels[5][3] = new Color(128, 255, 0);
```

2.3 PPM

When it comes time to store our 2-D array in a file so that we can see it, we have to write the data out in a format that image editors can read. There are a few very popular formats, like JPEG and PNG. They are popular because they compress the data very compactly with mathematics beyond the scope of this course. We will instead use a very readable format called PPM. Humans can open a PPM file in a text editor and read its contents directly. See the example text view of an image below-left:

```
P3
2 4
255
255   0   0
  0 255   0
  0   0 255
255 255   0
255   0 255
  0 255 255
255 255 255
  0   0   0
```



The first line of all PPM images is "P3". This is the PPM standard's *magic number*, which alerts image editors as to the kind of file that it is. The second line is the image's width and height. The third line is the maximum intensity found in any color. A color whose red, green, and blue components are this number will display as white. For our purposes, assume this number is always 255. The next lines are the individual pixel triplets, row by row. First we've got red and green. Then blue and yellow. Then magenta and cyan. And finally white and black. The first pixel in the file is shown at the top-left of the image.

Paste the text into a file and view it with an image editor. Not all editors can read PPM. Try using The GIMP. Tweak color values in the text file and see what happens. Make sure you understand the file structure before moving on.

3 Image

Your `Image` class encapsulates a 2-D grid of colors and its operations, like coloring a pixel, filling with a background color, and writing an image out to a file in the PPM format. It must meet this specification:

- Has a constructor that accepts these parameters, in order: an `int` width and an `int` height. It creates a 2-D array of `Color`s.
- Has a method `fill` that accepts a `Color` as its only parameter. It fills the entire image with the `Color`.
- Has a method `dot` that accepts these parameters, in order: a `Color`, an `int` column index, and an `int` row index. The pixel at the indexed location is assigned the given color.
- Has a method `writeToPPM` that accepts a `String` path (e.g., "C:/Users/goo/Desktop/test.ppm") to a file to write the image to. The file need not already exist. This method writes the image to the file in the PPM format described above. This method does not know how to handle and therefore should not handle any `FileNotFoundException`s. Mark it with a `throws` clause.

Test these methods in isolation before moving on.

4 Paragraph

Your `Paragraph` class reads a series of curves described by parametric equations from a file, plots them on an image, and writes the image out to a file. It must meet this specification:

- Has a method `plot` that accepts these parameters, in order: an `Image` object and a `String` having this form:

  ```
  x-equation,y-equation,starting p,ending p,step between ps,red,green,blue
  ```

  The method steps along the domain of p, evaluates the equations at each p value, and fills the pixels identified by the equations with the color specified by the last three values (which you may assume to be `int`s in $[0, 255]$). Suppose p has domain $[0, 10]$ with step 2. Then we'd evaluate the equations six times, for $p \in \{0, 2, 4, 6, 8, 10\}$.

  To plot a green circle with radius 50 and centered at (25, 10), we'd pass this `String`:

  ```
  p cos 50 * 25 +,p sin 50 * 10 +,0,6.28,0.1,0,255,0
  ```

Use your `PostOps.evaluate` method to do the evaluating. However, notice that the equations likely contain a p term. Your `PostOps` class knows nothing about p. Before passing it the expression, replace all occurrences of it with the actual numeric p value as you step along.

Test this method before moving on.

- Has a `main` method that reads a file of the following format:

```
width height red green blue
curve1
curve2
curve3
...
```

The width, height are `int`s; red, green, and blue are `int`s between 0 and 255; and each curve is a `String` having the form described in the previous method. It creates an `Image` with the specified width and height, fills the image with the background color whose triplet is specified on the first line, plots each curve to the image, and writes the image out to a PPM file. If a curve line has a '#' as its first character or if it is completely blank, it is ignored. How this method handles `FileNotFoundException`s is unspecified—do what you please.

We suggest you place your test files in your `hw3` package. When you create a `File` to open them, prepend "src/hw3/" onto the file name.

Feel free to share any testing code, curve files, and resulting images on Piazza. (Please convert to PNG or JPG before posting.) You may also vote exactly once for your favorite image (not your own, however) with a "+1" reply in the followup discussion. The student with the most votes will be given up to half my kingdom.

5 Files

- SpecChecker: twodee.org/teaching/cs145/2012A/homework/speccheck_hw3.jar
- Example input: twodee.org/teaching/cs145/2012A/homework/paragraph_eg.txt
- Example output: twodee.org/teaching/cs145/2012A/homework/paragraph_eg.png

6 Other expectations and submission

See homework 1 and preassignment 1 to see what else you are graded on and how to submit your work.