

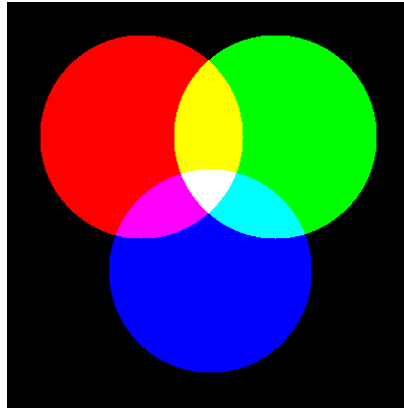
CSCI 136 Programming Exam #2
Fundamentals of Computer Science II
Spring 2012

This part of the exam is like a mini-programming assignment. You will create a program, compile it, and debug it as necessary. This part of the exam is open book and open web. You may use code from the course web site or from your past assignments. When you are done, submit all your Java source files to the Moodle exam #2 dropbox. Please ***double check you have submitted all the required files.***

You will have 100 minutes. No communication with any non-staff members is allowed. This includes all forms of real-world and electronic communication.

Grading. Your program will be graded on correctness and to a lesser degree on clarity (including comments) and efficiency. You will lose a substantial number of points if your code does not compile or crashes on typical inputs.

Overview. Your task is to implement a program that visualizes how a number of colored spots light up a square box. For example, here is an image generated from three spots, one that is completely red, one that is complete green, and one that is completely blue:



Recall that a `java.awt.Color` object can be represented by three integers (R, G, B) where R, G, and B are integers between 0 and 255 (inclusive of 0 and 255). Pure black is (0, 0, 0) while pure white is (255, 255, 255). The color at each pixel position of the square is the sum of the red, green, and blue components of all spots within range of a given pixel, subject to a maximum value of 255 for any RGB component.

Getting stated. Download the file <http://katie.mtech.edu/classes/csci136/spot.zip>. This file contains empty files for the three programs you will develop: `Spot.java` (part 1), `Spots.java` (part 2), and `MakePicture.java` (part 3). It also contains `Picture.java` class and sample input you can use for testing.

Part 1. Write a simple class `Spot` that represents a single colored spot of light. The `Spot` object knows things like its x- and y-location, its radius, and its color. Your class should have the following API:

```
public class Spot
-----
    Spot(double x, double y, double r, int red, int green, int blue)
    double getX()
    double getY()
    double getR()
    Color inRange(double x, double y)
```

The constructor should simply use the passed in parameters to setup the object's state in its instance variables.

The `inRange()` method should return the `Color` this `Spot` would add if the specified point is within the radius of the spot (strictly within the circle of radius `r` centered at (x, y)), returning `null` otherwise. To compute the distance between (x_1, y_1) and (x_2, y_2) , take the square root of $(x_2 - x_1)^2 + (y_2 - y_1)^2$, as usual.

Note. Do not forget to use the keyword `this` if your instance variables have the same name as the arguments passed to your constructor or method.

Note. Do not worry if you do not use your accessor methods in the subsequent parts.

Note. You will need to `import java.awt.*` in order to use the built-in Java `Color` class.

Part 2. Create a class `Spots` that represents a collection of `Spot` objects. The `Spots` object can construct itself based on a text file that specifies the details of 0 or more colored spots. Here is the input file specifying the three spots shown on the previous page:

```
0.33 0.33 0.25 255 0 0
0.66 0.33 0.25 0 255 0
0.50 0.66 0.25 0 0 255
```

The first line specifies a spot at (0.33, 0.33) of radius 0.25 that is red. The second line specifies a spot at (0.66, 0.33) of radius 0.25 that is green. The third line specifies a spot at (0.5, 0.66) of radius (0.25) that is blue. You can assume the input file is well-formed.

Your `Spots` class should implement the following API:

```
public class Spots
-----
    Spots(String filename)
    int getSize()
    Color getColorMix(double x, double y)
```

Your constructor should parse the input file using Java file I/O, creating and storing all the `Spot` objects specified in the input file. The `getSize()` method returns the number of spots in the collection.

The `getColorMix()` method computes the mix of colors present at the specified (x, y) location. The color returned has a red component equal to the sum of the red components of any `Spot` within the range of (x, y). This is subject to a maximum value of 255. A similar calculation is done for the green and blue components.

Part 3. Create a command-line program `MakePicture`. `MakePicture` takes three arguments: the input filename, the resolution of the square output image, and the filename for the output image. If fewer than three arguments are passed to the program, it terminates after displaying the following error message:

```
% java MakePicture 3spots.txt
MakePicture <input file> <resolution> <output file>
```

You do not need to do any additional error handling, i.e. you can assume the input file exists and is well-formed, the resolution is a positive integer, and the output file can be written to.

You will be using the `Picture.java` class to: create an initial blank image of the specified size, change the color of each individual pixel, and finally write the image to the output file. Here is all of the `Picture` API that you need to know:

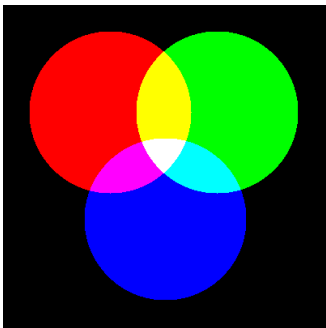
```
public class Picture
-----
    Picture(int w, int h) // Create a blank w-by-h picture, where each pixel is black
    void set(int i, int j, Color c) // Set color of pixel (i, j) to c, note (0,0) is upper-left
    void save(String name) // Save the picture to a file in a standard image format
```

Your program must do the following:

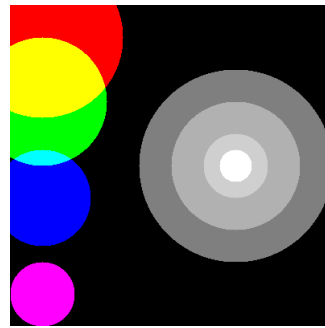
- Print an error message if fewer than 3 command-line arguments are given.
- Create a Spots object based on the filename specified on the command-line.
- Print out the number of spots loaded from the file and the input filename.
- Create a blank square Picture object of the resolution specified on the command-line.
- Set the color of each pixel (i, j) based on the mixture color at the location (i/resolution, j/resolution). Thus a Spot object at (0.0, 0.0) will appear at the upper-left corner of the Picture while a Spot at (1.0, 1.0) will appear at the lower-right corner.
- Print out the image resolution and the output filename.
- Write the image to the output filename
- Print percentage (to 2 decimal places) of pixels that were completely black (RGB of (0,0,0)).
- Print percentage (to 2 decimal places) of pixels that were completely white (RGB of (255,255,255)).

Note. Your program does not display anything graphically itself. It simply calls the save() method in the Picture class to write the computed color mixture to a *.png or *.jpg file. You can view your image in the Windows explorer. Here is the console output and our resulting images for the sample input files:

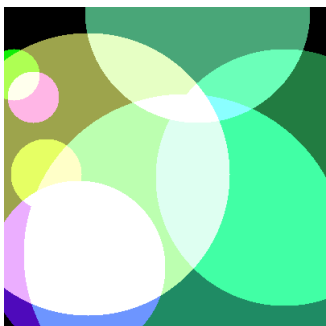
```
% java MakePicture 3spots.txt 400 3spots.png
Loaded 3 spots from 3spots.txt
Writing 400 x 400 pixel image to 3spots.png
Completely black: 50.73%
Completely white: 0.87%
```



```
% java MakePicture 8spots.txt 400 8spots.png
Loaded 8 spots from 8spots.txt
Writing 400 x 400 pixel image to 8spots.png
Completely black: 47.81%
Completely white: 0.78%
```



```
% java MakePicture 10spots.txt 400 10spots.png
Loaded 10 spots from 10spots.txt
Writing 400 x 400 pixel image to 10spots.png
Completely black: 3.59%
Completely white: 17.20%
```



```
% java MakePicture 1000spots.txt 500 1000spots.png
Loaded 1000 spots from 1000spots.txt
Writing 500 x 500 pixel image to 1000spots.png
Completely black: 35.24%
Completely white: 6.59%
```

