CS 5004 Object-Oriented Design

Assignment 9: An Image-Processing Application Controller

Due Date: Monday, April 10th, 11:59 pm.

This homework should be done with your partner from part 1.

1 Introduction

In this part of this project, you will add some more operations on images. You will also create a complete program that takes "commands" from a user through a file, and then process them as a batch in your program. The result of this iteration will be a program that a user can run like any other program, but not have back-and-forth interaction.

The objective of this iteration is to enhance your model, and add a controller to your program.

2 Color Transformations

If you did not complete color transformations as part of the previous assignment, you are now required to do it. Please refer to Assignment 8 description for details.

As you are enhancing the model, review the SOLID principles from Lecture 10. Think about changes that your model needs in order to add operations, and the best way to do this. Changing existing methods in the interface is not a good idea. Leaving interfaces completely intact, or adding to it without changing anything that already exists are better ideas...

3 Dithering

Older newspaper printers could only print black and white (remember the loud dot-matrix printers?). This posed a problem when trying to print photographs. This was done by printing the image using only black and white dots, such that a reader could still clearly make out what the image was showing. This operation of breaking down an image that has many colors into an image that is made of dots from just a few colors is known as dithering. Although dot-matrix printers are a thing of the past, and newspaper printers have vastly improved, dithering still has its applications. One example is to compress an image: reduce all colors in an image to just a few colors so that storing them requires less space. This is known as an "indexed color scheme" (thinking of an image being made out of fixed colors in a table), and is used by GIF images.

A popular technique to dither an image is the *Floyd-Steinberg algorithm* (see `https://en.wikipedia.org/wiki/Floyd%E2%80%93Steinberg_dithering` for more information). Although it can be applied to create colored dithered images, we will use it to create "black and white" images. That is, our images will only have two colors: black and white!

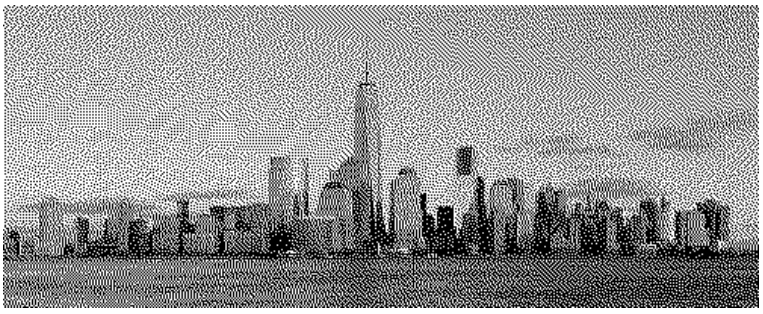
We first convert our image to greyscale. Then the algorithm proceeds as follows:

for each position (r, c) in image (traversing row-wise) **do**
  `oldColor` = red-component of pixel (r, c) // or green or blue
  `newColor` = 0 or 255, whichever is closer to `oldColor`
  `error = oldColor - newColor`
  set color of pixel (r, c) to `(newColor, newColor, nwColor)`

  add $(7/16 * error)$ to pixel on the right $(r, c + 1)$
  add $(3/16 * error)$ to pixel on the next-row-left $(r + 1, c - 1)$
  add $(5/16 * error)$ to pixel below in next row $(r + 1, c)$
  add $(1/16 * error)$ to pixel on the next-row-right $(r + 1, c + 1)$
end

The picture below illustrates the effect of dithering.

Original Image



Dithered Image

4 Image Mosaic

Many image processing applications support an effect that gives an image a "stained glass window" effect. Stained glass windows are popular in many ancient architectures, especially older churches. They create pictures by joining smaller irregularly-shaped pieces of stained glass.

An image can be "broken down" into such stained glass pieces, by choosing a set of points in the image (called seeds). Each pixel in the image is then paired to the seed that is closest to it (by distance). This creates a cluster of pixels for each seed. Then the color of each pixel in the image is replaced with the average color of its cluster. This may be thought of as an "easier cousin" of k-means clustering (easier because a pixel is clustered simply to its nearest seed, and the clustering is a one-step (instead of iterative) process).

The seeds can be chosen in a number of ways. The simplest method (that you will implement) is to choose the seeds randomly (i.e. choose random pixel locations from the image). The picture below illustrates the effect of mosaic.

Original Image

Mosaic with 1000 seeds

Mosaic with 4000 seeds

Mosaic with 8000 seeds

Mosaic with 15000 seeds

5 Image Manipulation by Batch Commands

Until now the "driver" of your program was a meaningless main method, or a test. A better way would be to allow the user to load an image, apply various effects to it and save the results. One way to accomplish this is if the user can provide input in a "script", as follows:

```
load manhattan.png
dither
save manhattan-dither.png
load manhattan.png
blur
save manhattan-blur.png
sepia
save manhattan-blur-sepia.png
...
```

A user could (for example) type the above in a file, and then provide it to the program. The program would then read the script one line at a time, process it and create several files as requested by the user.

6 What To Do

In this assignment you must enhance the **model** of this image processing application, and add a controller to your program that supports a script like above.

6.1 Required Features

Your program must be able to:

1. convert an image to greyscale and sepia
2. dither an image to black-and-white
3. support the user entering commands in a file with your program executing them (although in future it may not just be a file!). The file must be provided to the program using command-line arguments (the parameter of the main method). To do this when running your main method from IntelliJ: go to **Run ... Edit Configurations...**. Now in the configuration that you are running, you will see a field Program arguments. If you type "input.txt" there, save changes and run your program, then "input.txt" is available as `args[0]` in your `public static void main(String[] args)` method. For this to work, this file should be inside your project folder (where `src/` and `test/` are).

You are not expected to support exactly the commands shown in the script above. The above was just an illustrative example of the kinds of things a user should be able to do. You may choose different words, or syntax. Whatever words and syntax you choose should be reasonable for a user to provide. **Do not try to support expressive or intuitive syntax: start with basic, easy-to-parse commands and enhance if you have time.**

6.2 Extra Credit

For extra credit, your program should be able to convert images into mosaics. The user of your model should be able to control mosaic by specifying the number of seeds to use. This will be a required feature of your program later, so you can earn extra credit for implementing future features now. Think of it as delivering on features initially promised to the user in Version 3.0!

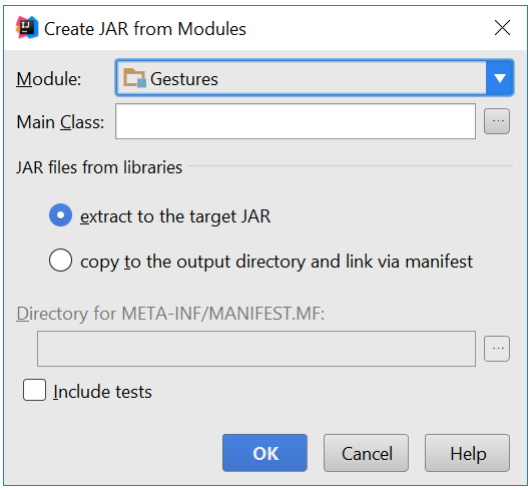
7 Design Enhancements and Changes

You should document and justify all design changes in the README.md file. You are allowed to change your design. However the bigger the change, the more convincing we expect your explanation to be.

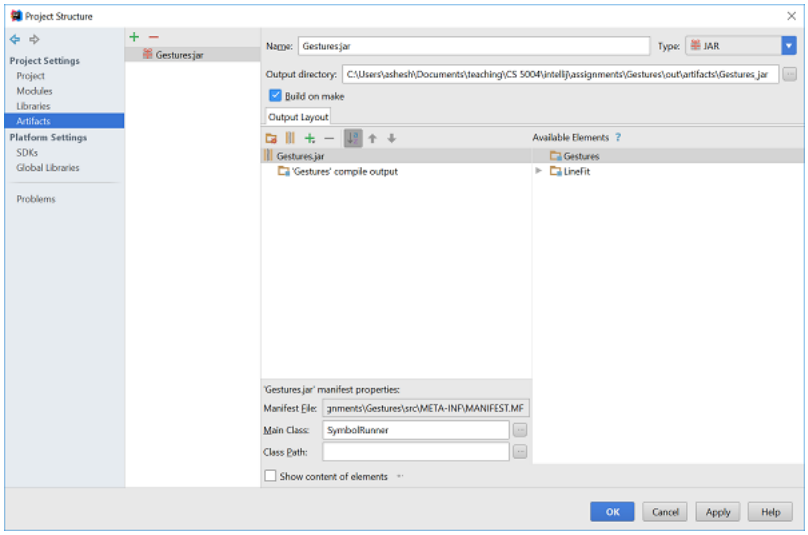
8 Create a JAR file of your program

To create a JAR file, do the following:

- To to **File ... Project Structure ... Project Settings ... Artifacts**
- Click on the plus sign
- Choose **JAR ... From Modules with dependencies**. You should now see the module in your project that you are working on (may be different from what is shown in the image below depending on your operating system):



- Select the main class of your program (where you defined the `main(String[] args)` method)
- If you see a checkbox labelled "Build on make", check it.
- Click **Ok**
- You should now see something like:



  If now you see a checkbox labelled "Build on make", check it now.

- Make your project (the button on the left of the run configurations dropdown, with the ones and zeroes and a down-arrow on it). Your `.jar` file should now be in `<projectRoot>/out/artifacts`.
- **Verify that your jar file works.** To do this, copy the jar file to another folder. Now open a command-prompt/terminal and navigate to that folder. Now type

  `java -jar NameOfJarFile.jar any-command-line-arguments-separated-by-spaces`

  and press **Enter**. The program should behave accordingly. If instead you get errors, review the above procedure create the JAR file correctly.

9 Criteria for grading

You will be graded on:

1. Your design (interfaces, classes, method signatures in them, etc.)
2. Whether your code looks well-structured and clean (i.e. not unnecessarily complicating things, or using unwieldy logic).

3. Correctness of your implementations, evidenced in part by the images you submit.
4. Whether you have written enough comments for your classes and methods, and whether they are in proper Javadoc style.
5. Whether you have used access modifiers correctly: `public` and `private`.
6. Whether your code is formatted correctly (according to the style grader).

10 What to submit

Your zip file should contain three folders: `src`, `test`, and `res` (even if empty).

- All your code should be in `src/`.
- All your test code should be in `test/`.
- Submit at least two images and their greyscaled, sepia-toned, dithered, and mosaic-ed results in the `res/` folder.
- Submit a correct JAR file in the `res/` folder. We should be able to run your program from this jar file.
- Submit at least two example files in the `res/` folder that your program can take as input, to communicate to us how to specify commands.
- Submit a README.md file that documents how to use your program, which parts of the program are complete, and design changes and justifications. The file should also include citations for each image you have used. It is your responsibility to ensure that you are legally allowed to use any images, and your citation should be detailed enough for us to access the image and read its terms of usage. If an image is your own photograph or other original work, specify that you own it and are authorizing its use in the project.