File I/O, Image Processing, API Connectivity, JSON

Important Dates:

- Assigned: November 6, 2024
- Deadline: December 13, 2024 at 11:59 PM EST

Objectives:

- Students learn about a common raw image file format.
- Students examine various basic image-manipulation algorithms.
- Students add functionality to an existing GUI-based program.
- Students design a program that connects to an API, sends a GET request, and parses the returned data using JSON.

What To Do:

Design classes with the given specification in each problem. This problem set requires a bit of creativity and thinking outside the box. Work with other people to get this done and do not wait until the last minute to start!

The problem set has two required problems worth 70 and 30 points respectively. There are 30 possible extra credit points, meaning the highest score you can earn is 130/100%.

Problem 1 (20 points)

In this question, you will take what you've learned throughout the class to design a primitive image editor. Don't worry, you won't be required to work with any GUI components or design the UI yourself—all of that is taken care of.

First, we provide a bit of background. Most often, computers store colors as (alpha)-red-green-blue integers. Namely, a 32-bit `int` integer stores the data associated with a color where the most-significant byte stores the alpha (transparency) channel, the next byte stores the red channel, the next stores the green, and the least-significant byte stores the blue channel. A *channel*, as suggested, is a value between 0 and 255 inclusive. Often times, colors are represented using hexadecimal as a means of shortening the notation. E.g., `0xffff00ff` represents a color whose alpha channel is 255, red is 255, green is 0, and blue is 255. Fortunately, Java provides a `Color` class that allows us to circumvent the need to, say, manipulate the bits of a color directly, as we might in a lower-level language.

Jef Poskanzer invented the PPM image file format in the late 1980's. Its advantages over other image file formats include its listing of pixel data as explicit RGB values. PPM files also specify the image dimensions. For example, the following lines describe a 2×3 (pixel) image with red, green, and blue pixels in the first row, followed by blue, green, and red pixels in the second. Note the exclusion of the alpha channel. To view this image, copy the text into a `.ppm` file and open it at this link: `https://www.cs.rhodes.edu/welshc/COMP141_F16/ppmReader.html`.

```
P3
2 3
255
255 0 0 0 255 0 0 0 255
0 0 255 0 255 0 255 0 0
```

Java provides the handy `BufferedImage` class for working with images. Some of its methods, with descriptions, are listed below.

- `new BufferedImage(int w, int h, int type)` creates a new `BufferedImage` object with width w, height h, and a "pixel type." For your purposes, this value can be hard-coded as `BufferedImage.TYPE_INT_RGB`.
- `void setRGB(int x, int y, int C)` sets the color of a pixel at (x, y) to C. To convert a `Color` object to its integer representation, call `getRGB()` on the `Color` object.
- `int getWidth()` returns the width of the `BufferedImage`.
- `int getHeight()` returns the height of the `BufferedImage`.

We also provide helpful methods in the `ColorOperations` class that you should acquaint yourself with. You do not need to worry about the bodies *of* these methods; just assume that they work as intended, and know their inputs and output values.

Once you are fairly acclimated with the `BufferedImage` class (and the `Color` class), head to the `ImageEditor` class. This is where you will begin writing your own code.

(a) First, finish implementing the `void readPpmImage(String in)` method. When given a file name that contains a PPM image specification, you should open the file (using whatever input mechanism you please, e.g,. `BufferedReader` or `Scanner`), read the data into a new `BufferedImage` object, then return that image. At the end of the `try` block is some template code that we ask you to not delete. We assume that your image is called `img`.

(b) Next, finish designing the `void writePpmImage(String out)` method. When given a file name, you should open the file, write out the PPM header data, then the image (pixel) data. Ensure that you correctly follow the PPM file format specification!

Problem 2 (50 points)

For the following questions, you will implement the image transformations listed under "Tools" in the UI. Each of these transformations corresponds to a `static` method inside the `ImageOperations` class. By default the transformations under "Tools" are disabled. As you implement each one, head into the `ToolsMenu` class and enable it for testing.

For each of the examples that we provide, we assume the following base image of a praying mantis (of which is also provided in `.ppm` format in the project, alongside examples of what each method/image operation should produce):

Figure 1: Praying Mantis

(a) Design the `static BufferedImage zeroRed(BufferedImage img)` method, which removes the red channel from the colors of the image.

Figure 2: No Red Channel

(b) Design the `static BufferedImage grayscale(BufferedImage img)` method, which converts the image to grayscale.

Figure 3: Grayscale Image

(c) Design the `static BufferedImage invert(BufferedImage img)` method, which inverts the pixel data.

Figure 4: Negative Image

(d) Design the `static BufferedImage mirror(BufferedImage img, MirrorMenuItem.MirrorDirection dir)` method, which mirrors the image either vertically or horizontally. The second parameter `dir` can be one of `MirrorMenuItem.MirrorDirection.HORIZONTAL` or `MirrorMenuItem.MirrorDirection.VERTICAL` for horizontal and vertical mirroring respectively. Consider the following examples of images being mirrored.

Figure 5: Mirroring Vertically

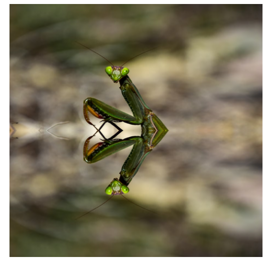

Figure 6: Mirroring Horizontally

(e) Design the `static BufferedImage repeat(BufferedImage img, int n, RepeatMenuItem.RepeatDirection dir)` method, which creates an image with the image repeated either side-by-side or top-to-bottom, depending on the given argument, *n* times. The `dir` parameter can be one of `RepeatMenuItem.RepeatDirection.HORIZONTAL` or `RepeatMenuItem.RepeatDirection.HORIZONTAL` for horizontal or vertical repetition respectively.

(f) Design the `static BufferedImage rotate(BufferedImage img, RotateMenuItem.RotateDirection dir)` method, which rotates an image 90 degrees either clockwise or counterclockwise. The `dir` parameter can be one of `RotateMenuItem.RotateDirection.CLOCKWISE` or `RotateMenuItem.RotateDirection.COUNTERCLOCKWISE`.

Figure 7: Rotate Clockwise

Figure 8: Rotate Counter-clockwise

For each operation, you should thoroughly test (note: **not** JUnit test) not only the transformation, but also saving and opening an image with applied transformations. Note that the "Help" menu exists to give you pointers on how to use the program, including zooming, undo/redo operations, and opening/saving.

Extra Credit (1-10 points) Implement a new image manipulation operation that is not listed. Points are awarded based on difficulty and creativity. So, an operation that simply "zeroes the green" will likely only receive 1 extra credit point, but a "pixelation filter" may receive closer to 10.

Problem 3 (30 points)

For this final question, you will work with retrieving data from a URL using HTTP GET requests. A GET request is simply a request to a web-server for some data. In particular, our program will contact a weather API with some parameters to retrieve the weather report for the upcoming week in Bloomington.[1] Therefore there are no test cases to write for this question because the data will be different on a per-person basis. Follow these steps. Note that this is very "hand-hold-y" because it is new content. You will need to investigate the Java documentation for specific methods.

(a) Create the `WeatherForecast` class containing only the `main` method.

(b) First, to make a GET request, we need to make use of the `URL` and `HttpUrlConnection` classes. We construct a new `URL` that contacts `https://api.open-meteo.com/v1/forecast?` with some extra data.

(c) Such extra data acts as the *parameters* to the GET request. Namely, we need to pass three data fields as key/value pairs: `latitude`, `longitude`, the rate at which our information is retrieved (i.e., whether we want an hourly forecast), the temperature unit, and finally the time zone. We separate the parameters to an HTTP request via ampersands. For example:

`https://api.open-meteo.com/v1/forecast?latitude=`*X*`&longitude=`*Y*`&hourly=`*Z*`&`
`temperature_unit=`*W*`&timezone=`*V*

Where X is the latitude, Y is the longitude, Z is `"temperature_2m"`, W is either `"celsius"` or `"fahrenheit"`, and V is `"EST"`.

(d) Wrap the `new URL` call inside a new instance of `HttpUrlConnection`. We then must set the request method to `"GET"`, followed by a call to check the response code of the HTTP request. HTTP requests return several codes to represent the status of the response, one of which is `200`, designating a successful connection. So, if the response code is `200`, we can continue. Otherwise, we will throw an `IOException`, stating that the exception failed.

(e) We now need to read the response back from the request. Open a new `BufferedReader` that reads from the connection's `InputStream`. Follow this up with a loop that reads the request data into a single string. You should *not* use standard string concatenation; instead, use `StringBuilder`.

[1] The latitude/longitude coordinates for Bloomington, IN are 39.168804/−86.536659. Note that Google incorrectly reports that the longitude is 86.53669.

(f) The data that we read back from the `GET` request is in the *JSON* format, standing for "JavaScript Object Notation". JSON, according to its Wikipedia page, is an "open standard file format and data interchange format that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and arrays." While we will (obviously) not be using JavaScript to solve this problem, we will take advantage of the Google JSON parsing library: `gson`. In doing so we will learn about a popular build system called *Maven*, which is a popular dependency manager for Java projects. Dependency managers allow the programmer to easily integrate new libraries into their code without having to manually download the JAR files, configure them to work on their system, and so forth. If you go further into Java development, you may also hear about *Gradle*, which is a similar such manager.

The starter code in Canvas already has these dependencies handled for you, so you should be able to just use the Gson library methods.

(g) To parse the string retrieved by the request, initialize a `JsonElement` to store the result of invoking the static `parseString` method on the response string builder. From here, it is up to you to read through the documentation to figure out how to extract the two `JsonArray` objects that store the times and temperatures respectively. This can be done with exactly three lines of code. You'll need to take advantage of the `.get`, `getAsJsonObject`, and `getAsJsonArray` methods.

(h) Print out the weather report for the next seven days, starting from the current hour. Print the report in the following format, using 3-hour intervals:

```
Bloomington 7-Day Forecast in Fahrenheit:
Forecast for 2023-11-10:
    00:00: 47.4°F
    03:00: 55.5°F
    06:00: 53.7°F
    09:00: 43.2°F
    12:00: 38.1°F
    15:00: 34.3°F
    18:00: 37.2°F
    21:00: 37.2°F
Forecast for 2023-11-11:
    00:00: 49.4°F
    03:00: 54.6°F
    ...
Forecast for 2023-11-17:
    ...
```

(i) **Extra Credit (10-20 points)** Use terminal arguments to make this a more useful program. In particular, I should be able to set the latitude, longitude and temperature unit via flags:

```
./WeatherForecast --latitude 36.0689 --longitude -79.8102 --unit C
```

This would then print out the 7-day forecast for Greensboro, NC. Make sure you modify the forecast header to print the accurate location instead of `"Bloomington 7-Day Forecast"`! Should you want to go even further above and beyond, we will award 1 point per extra "feature", up to a maximum of 10, that you implement. Examples include precipitation percentages, high/low temperatures, visibility, sunrise/sunset times for dates, and so forth. Explore the API to get ideas.