HW 02: Binary I/O

Due Feb 15, 2022 by 2:55pm **Points** 1 **Submitting** a file upload **File Types** zip
Available until Feb 15, 2022 at 2:55pm

This assignment was locked Feb 15, 2022 at 2:55pm.

CS-2013 Programming with Data Structures

Homework 02: Binary I/O and Image Manipulation

Required Files / Software / And Resources

- **PPM Images** (http://csns.cysun.org/download?fileId=7538426) : Used for testing.
  - **NOTE: Some images may be too large to test with this program. Try out all the images in the zip file.**
- **IrfanView** (http://www.irfanview.com/) : (For Windows Users) Program that lets you view PPM Images.
- **GIMP** (https://www.gimp.org/) : (For MAC Users) Program that lets you view PPM Images.

General Description

For this assignment you will be implementing a program which can read a PPMImage and alter its pixel data to manipulate how the image looks. You will use Binary IO to process the image.

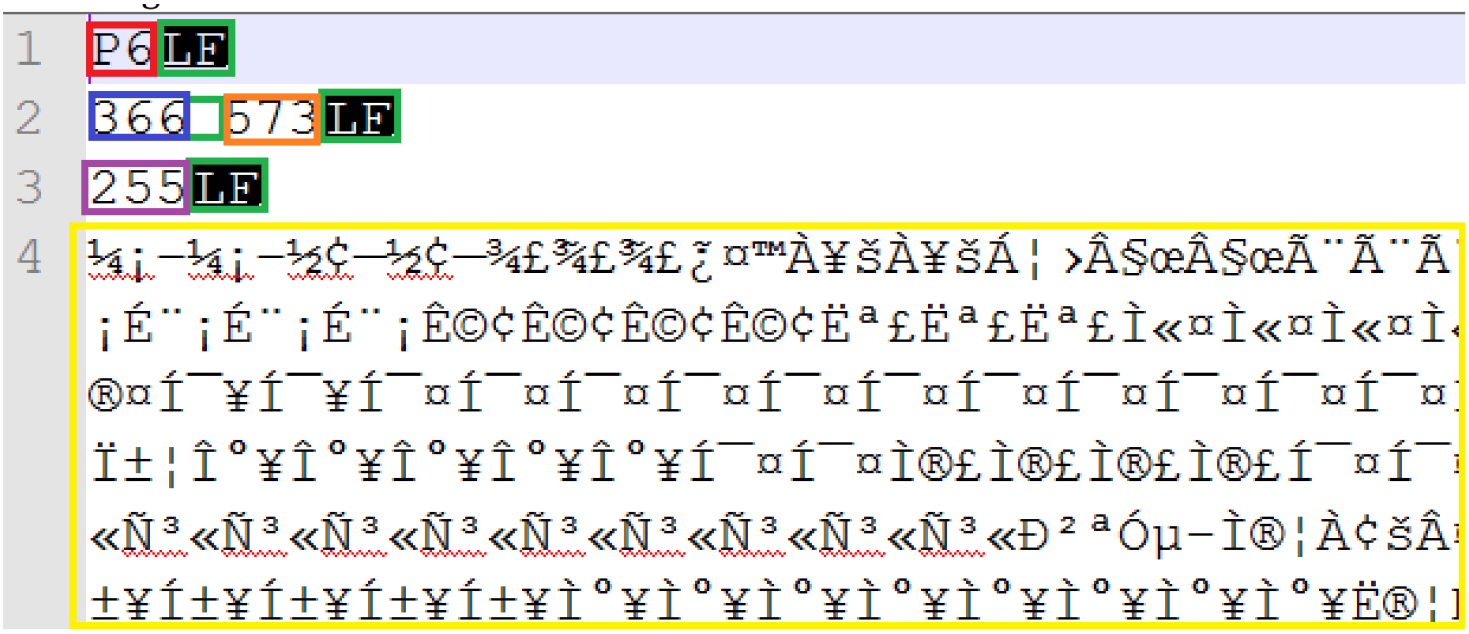
The PPM Image Format

Digital images are represented as a sequence of elements known as pixels. The pixels are arranged in a series of rows and columns. Several file formats have been created over the years to store images for later viewing. Some of these file formats are PNG, JPEG, GIF, and so on. Another such format is the PPM file format. The PPM file format provides a fairly straightforward way of accessing image data for manipulation and is an uncompressed format. A PPM Image has the following format:

- A "magic number" for identifying the file type. A PPM image's magic number is the two characters "P6".
- A whitespace character (this can be an actual space, or line feed, for us we will use the linefeed "\n" character).
- A width formatted as ASCII characters in decimal.
- A whitespace character (an actual space).
- A height, again in ASCII decimal.
- A whitespace character (the linefeed "\n" character).

- The maximum color value (maxval), again in ASCII decimal. For our purposes the maxval will always be 255.
- A single whitespace character (the linefeed "\n" character).
- A raster of height rows, in order from top to bottom. Each row consists of width pixels, in order from left to right. Each pixel is a triplet of red, green, and blue channels, in that order. Each channel is represented in pure binary by either 1 or 2 bytes. If the maxval is less than 256, it is 1 byte. Otherwise, it is 2 bytes. The most significant byte is first. For our purposes we will be working with values between 0 ~ 255, so each channel of a pixel (red, green, or blue) will only take up one byte.

If we were to look at a PPM image file inside of a text editor such as Notepad++, we would see something like the following:



- **The red box is the magic number.**
- **The green boxes are the white space characters, in this case they should be spaces or line feeds NOT carriage returns.**
- **The blue box is the width of the image.**
- **The orange box is the height of the image.**
- **The purple box is the maxvalue of each pixel channel.**
- **The yellow box is the actual pixel data. it looks like a jumbled mess because the pixel data is written to the file using binary file I/O and the text editor will try to render the byte values as their ASCII character equivalent.**

PPMImage Class

- **Data Fields:**
  - **`magicNumber`**: a String to store the magic number.
  - **`width`**: an integer value for the width of the picture.

- **`height`**: an integer value for the height of the picture.
- **`maxColorValue`**: an integer value for the maxColorValue value.
- **`raster[]`**: This is a 1D **char** array. The size of this array should be computed with width * height * 3 (the image has width * height pixels and each pixel is made up of 3 color channels Red, Green, Blue.) We use an array of char because Java does not have an unsigned byte datatype (bytes that are only positive values) and char is 2 bytes which allows us to store the pixel data with the correct (positive only) values. If you do not use a char array, then the math to manipulate the image will not be correct.

- **Constructor:**
  - **`PPMImage(String imageFileName)`**: Takes a String that is the name of a PPM image. Loads the image data into the object. The image must be a PPM image. If the file is not a PPM image then your program should throw an appropriate exception. This method will call your private readImage method.
- **I/O Methods**:
  - **`writeImage(String outputImageFileName)`**: Takes a String which is the name of the file to write the image data to. This will create a new file with the given name and write all of the image data into the file. If you only give the file name, then image will be written to the current working directory of the project. If the file name is not a .ppm file name, you should also throw an exception. When writing the image header information (magic number, width, height, maxColorValue), you will need to write the data character by character (including the whitespace characters). The method which writes a byte at a time is able to take a char value and correctly write it to the file.
  - **`readImage(String imageFileName)`**: This method should be a private method and should be called by the constructor to read the image data from the image file. HINT: You will need to read the image data byte by byte. For the image header information (magic number, width, height, maxColorValue), you will need to read these byte by byte (including the whitespace characters), but convert each byte to a char in order to store the data in your program correctly.) Remember that multiple char values can be concatenated into a string and if necessary, a string can always be converted into an integer or other data type using the correct parsing method.
  - **`grayscale()`**: This method should alter the data of each pixel using the given formula to turn the image from color to grayscale.
    - To convert an image to grayscale, use the following formula. NOTE: R, G, B stands for the original red, green, and blue values respectively, R', G', B' are the resulting new values.

```
R' = (R * .299) + (G * .587) + (B * .114)
G' = (R * .299) + (G * .587) + (B * .114)
B' = (R * .299) + (G * .587) + (B * .114)
```

  - **`sepia()`**: This method should alter the data of each pixel using the given formula to turn the image from color to sepia.

- To convert an image to sepia, use the following formula. NOTE: R, G, B stands for the original red, green, and blue values respectively, R', G', B' are the resulting new values.

```
R' = (R * .393) + (G *.769) + (B * .189)
G' = (R * .349) + (G *.686) + (B * .168)
B' = (R * .272) + (G *.534) + (B * .131)
```

NOTE: If a resulting value is greater than 255, then just set it to be 255.

Image Manipulation Methods:

`negative()`: This method should alter the data of each pixel using the given formula to turn the image from color to negative().

- To convert an image to negative, use the following formula. NOTE: R, G, B stands for the original red, green, and blue values respectively, R', G', B' are the resulting new values.

```
R' = 255 - R
G' = 255 - G
B' = 255 - B
```

Main Class -> Console UI

- Create a nice console menu which allows the user to do the following:
  - Choose an image to manipulate.
  - Change the image to sepia, greyscale, or negative.
  - Save the new image with a user chosen file name.
  - Exit the program.

Restrictions

- You may not use any Java classes which have to do with working with images
- All pixel manipulation must be coded by you, using the given formulas.

Video Presentation Checklist:

The following is a checklist of everything I would like you to demonstrate / explain in your video.

1. **Show** the code of your PPMImage Class.
2. **Explain** how the code for your writeImage() method works.
3. **Explain** how the code for your readImage() method works.
4. **Demonstrate** that your program is able to read an image in and write the same image back without any issues. Do this for at least 3 of the sample images.

1. NOTE: I will test your program will all images given so make sure they all work.

5. **Demonstrate** that your program is able to read an image, turn it into greyscale, negative, and sepia, and then save it to a new file correctly. Do this for at least 3 of the sample images.

You should be able to do all of this in no more than 5-7 minutes. Less than 5 minutes is OK.

Deliverables

- Compress your **src** into a .zip file.
  - If done correctly, this .zip file will contain all the necessary files that I need to grade your assignment.
  - Upload your .zip folder to Canvas before the due date and time.
- Be sure to upload a .zip file of your **src** folder to Canvas by the due date and time.
- **DOCUMENTATION:** You are required to document your code for this assignment according to the canvas page and using /**Javadoc comments*/. See the example Canvas page on how to write proper javadoc comments.
- **VIDEO:** You should record a video presentation of your project according to the checklist given above.
  - Upload your video to your canvas studio space. (If using Canvas Screen and Webcam Recorder, this should be done for you automatically.)
  - Go into the video on Canvas and choose the **Share** option, then choose **"Create public link".**
  - Copy and paste the public link as a comment on the Canvas upload homework interface.