CSCI165 Computer Science II
Discussion Problem
Drawing a "Bitmap" from a matrix of pixel data

Objectives:

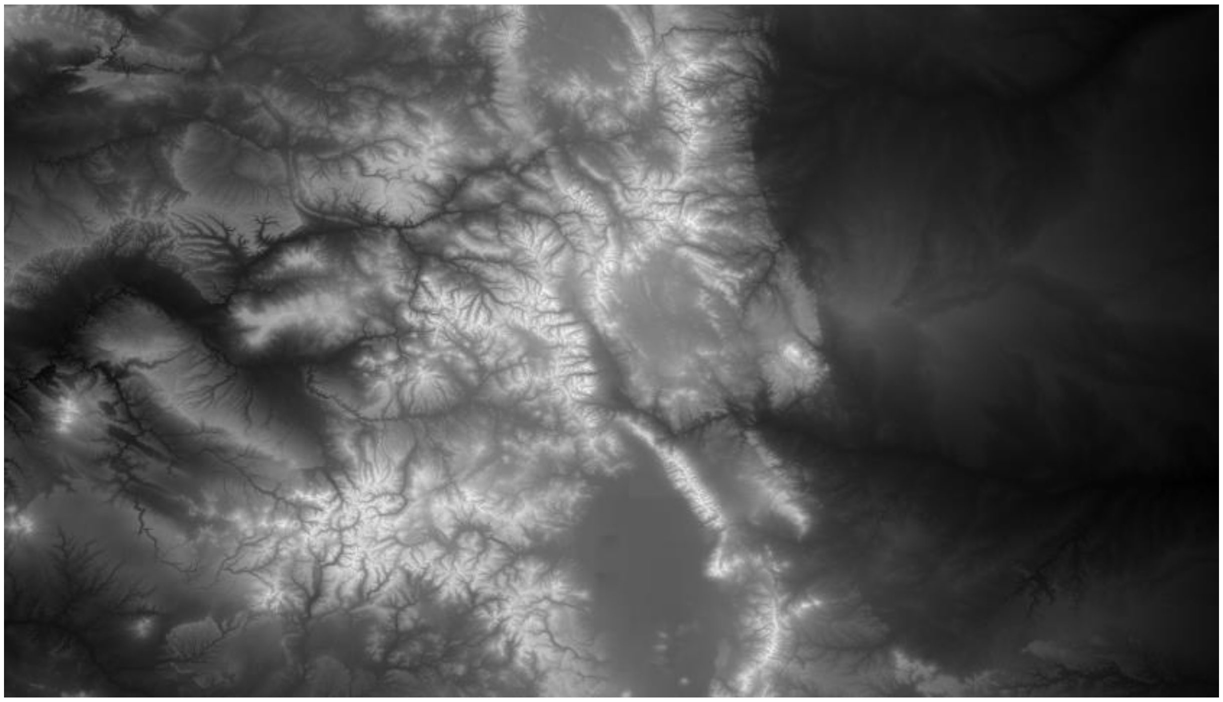
- Matrix iteration and processing
- RGB Color manipulation
- Simple drawing using Java 2D graphics

Discussion: Share screen shots of code and the resulting map. Have well documented and legible code. Provide insight to your problem solving and assist class mates in need. In order to get full credit on a discussion you must interact meaningfully with your class mates.

Submission: Push finished code to a **discussion** folder in your repo.

Problem: This problem is the first step in your lab for next week. I'd like to spend the discussion energy on making sure everyone can nail this step.

In this problem you will read a set of topographic (land elevation) data from the National Oceanic and Atmospheric Information into a 2D array (matrix). You will then use that matrix to generate a bit mapped visualization using the Java 2D graphics library. The file of topographic data that you have been given will generate the following map of the Rocky Mountains in Colorado.



This file is a simulation of a "bit map", where each pixel in the image maps to a single color. The image above has 480x840 "pixels". Each pixel is drawn using 1x1 rectangles and the **fillRect** method from the Java2D Graphics object, as shown in lecture.

Step One: Loading the Data

1. Read data from the file ***Colorado_844x408.dat.*** The data is a plain ascii text file from NOAA representing the average elevations of patches of land. Each value in this file represents roughly 700x700 meters in the US. The ***Colorado_844x408.dat***, file is the elevation data for most of the state of Colorado. The data comes as one large, space-separated list of integers. There are 403,200 integers representing a 480-row by 844-column grid. Each integer is the average elevation in meters of each cell in the grid. The integers should be read in ***row-major order.***
2. **WARNING:** The data are given as a continuous stream of numbers - there are no line breaks in the file, so you can't use nextLine(). You need to use Scanner's nextInt() method to read each subsequent integer. You also probably want to write nested loops to cover the exact rows/cols (480/844) needed. Use well named constants to store these numbers so they can be easily manipulated.
3. After you've read the data**, test it with the debugger.** Make sure that you're actually reading data into the 2D array. Do a sanity check by looking at a specific row and column and comparing with a classmate in the discussion forum, or against the original data itself. Let's work together and help each other out here.

Step Two: Finding the highest and lowest elevation points

1. In order to draw the map you need to find the min and max values in the map first. Implement the **findMinimum()** and **findMaximum()** methods. These should return the smallest and largest values respectively in the 2D array. You'll need to write code that scans the entire array and keeps track of the smallest and largest things found so far. Don't use any short cuts here. Demonstrate that you can construct logic to traverse the matrix and find the appropriate values. Don't use API calls to handle this task for you.
2. Test these functions independently first to make sure you're getting decent values. Maybe check with a classmate to see if they are getting the same thing. A good way to build test cases for this is to define a ***much smaller*** matrix; one that is easy to determine the min and max values.

Step 3 – Draw the map

1. Implement the ***drawMap(Graphics)*** method of the provided class. The method will be passed the Graphics object you should use to do the drawing. The method "draws" the 2D array of integers as a "bit map" with the value in each cell in the array defining the color gradient to color the "pixel". This can be viewed as a series of filled 1x1 rectangles with the color set to a gray scale value between white and black.

   RGB Colors: https://www.rapidtables.com/web/color/RGB_Color.html

   The shade of gray should be scaled to the elevation of the map. If each of the RGB values are the same, you'll get a shade of gray. Thus, there are 256 possible shades of gray from black (0,0,0) to middle gray (128,128,128), to white (255,255,255)

   To make the shade of gray**,** you should use the min and max values in the 2D array to scale each int to a value between 0 and 255 inclusive. (**Note:** this requires math) Once you have found your value between 0 and 255, set the fill color just before drawing the rectangle. This was shown in the lecture demo.

```java
int c = //calculated grayscale value
g2d.setColor(new Color(c, c, c)); // same value for RGB
g2d.fillRect(x, y, 1, 1); // 1x1 filled rectangle
```

WARNING: the 2D array represents the data in row-major order (that’s how it came out of the data file). But the graphics are drawn in column-major order x,y format. Thus, remember to flip row and col (or x and y) if using them as variables to access the array or draw the graphics