CS 142, Spring 2024
Programming Project #4: Mountain Pass (40 points)
Due: Wednesday, May 8, 2024, 11:59 PM

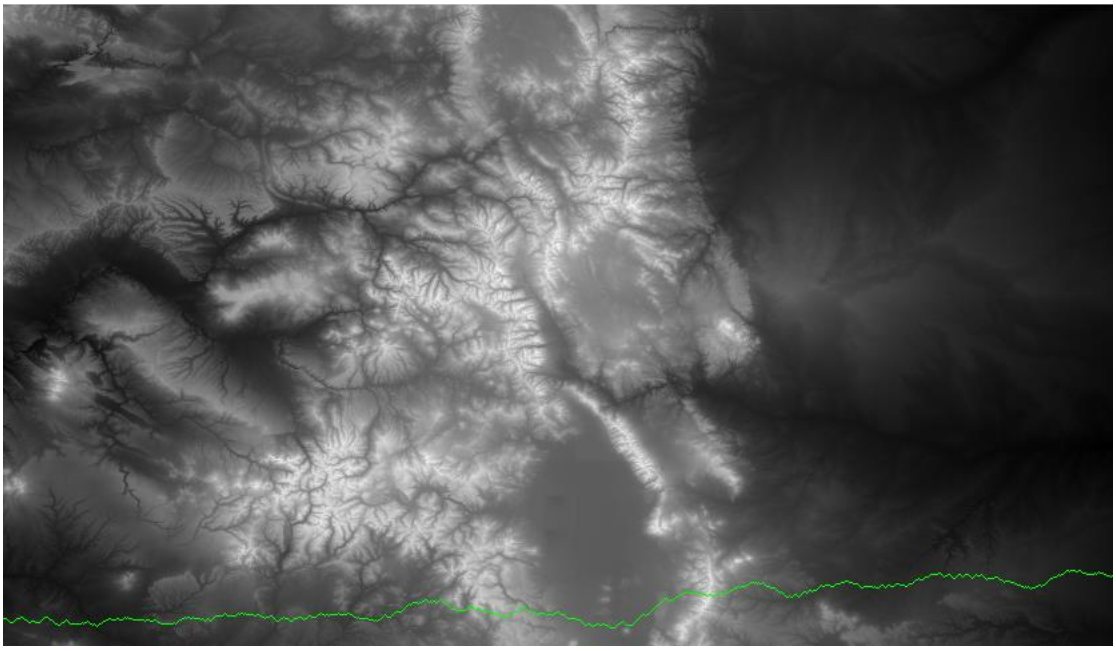
Thanks to Baker Franke for the assignment idea and to Marty Stepp and Stuart Reges for parts of this document.

This assignment focuses on working with arrays of arrays. Turn in a file named `MountainPass.java` and `Location.java`.

Program Description:

When people plan roads, rail roads or other routes over mountain ranges they spend a lot of time working with elevation data to chart a good path. A good path is one that has a minimal change in elevation. When traveling over a mountain range it is impossible to avoid a bunch of elevation gain and then fall but the more that can be avoided the better.

For this assignment you will read in a CSV file of topographic (land elevation) data into a 2D array, find the highest point, chart a low elevation change path through the land shown and output an image of the path like the one below.



Input Data and Files:

Your program should prompt the reader for an input file. Several example data files are provided on the class website. Open these files with jGrasp, VSCode or some other text editor. Do NOT open with Excel.

The first line of the data file will contain the *width* and *height* of the area. There will be *height* following rows. Each of them will contain *width* numbers. You can assume all numbers will be integers and that there will be no spaces or tabs in the file. For example, a small input file might look like the following:

```
10,5
100,200,100,100,100,200,200,200,100,100
200,100,100,100,100,200,200,200,200,1
1,1,200,200,500,400,300,100,1,100
200,200,1,200,1,200,1,1,300,100
200,200,200,1,100,100,100,100,100,1
```

Console Output:

Your program should print out the x, y coordinate of the highest elevation in the data file and the lowest elevation in the data file. If there is a tie it should print the location closest to the top of the file. If two are equally close to the top of the file it should print the one furthest left. It should also output the coordinates visited on the lowest elevation change route from the left side to the right and the total change in elevation. Example file output:

```
File name? mountain data.csv
Mountain Peak: (4, 2)
Lowest Point: (9, 1)
Lowest Elevation Change Route: (0, 2), (1, 2), (2, 3), (3, 4), (4, 3), (5, 4), (6, 4), (7, 4), (8, 4), (9, 3)
Total Elevation Change: 99
Steepest Elevation Change: 99
```

Graphical Output:

Your program should output the elevation data, peak, lowest point and path as an image. To do this, we will be using the `DrawingPanel` class as shown in lecture.

Remember that to use `DrawingPanel` you will need to create a `DrawingPanel` object, ask it for its `Graphics` and then use that `Graphics` object to draw 1 by 1 rectangles, one for each elevation measurement. To create your own color from red, green and blue values, just pass an integer for red, one for green and one for blue to the constructor of `Color`.

Your image should contain the location of the highest peak in red (255, 0, 0), the locations of each step on the lowest elevation change found path in green (0, 255, 0) and gray values to represent the elevation for all of the other locations in the original image.

We suggest that you alter your array of arrays as you look for the path and mark each step as you travel it. That way, once you have computed the path you can just output the contents of the array of arrays. You will need the elevation data for later computations, so we suggest you store each path cell in a `Location` object so you can access its data later. Since your path is longer than one step, store these `Location`s in an array.

To translate an elevation to grayscale compute its percentage of 255 (the maximum number of colors). For example, `int gray_value = 255 - (255 * (elevations[i][j] - min_elevation) / (max_elevation - min_elevation));`. Use this value for the red, green and blue values for that pixel.

Finding the Path:

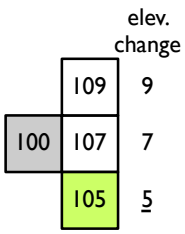
For this assignment we will use a "greedy" algorithm to find the path. A “greedy” algorithm is one in which, in the face of too many possible choices, you make a choice that seems best at that moment. For the maps we are dealing with there are 7.24×10^{405} possible paths you could take starting from western side of the map and taking one step forward until you reach the eastern side.

3011	2900	2852	2808	2791	2818
2972	2937	2886	2860	2830	2748
2937	2959	2913	2864	2791	2742
2999	2888	2986	2910	2821	2754
2909	2816	2893	2997	2962	2798

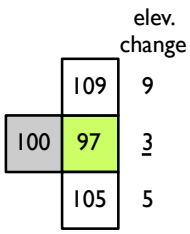
Shows a portion of the data.
Greedy path shown in green.

Since our map is in a 2D grid, we can envision a "walk" as starting in some cell at the left-most edge of the map (column 0) and proceeding forward by taking a "step" into one of the 3 adjacent cells in the next column over (column 1). Our "greedy walk" will assume that you will choose the cell whose elevation is closest to the elevation of the cell you're standing in. (NOTE: this might mean walking uphill or downhill). At the start you have no current cell so nothing to compare to. Start your walk by choosing a random row in the first column of the grid.

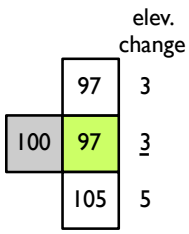
The diagrams below show a few scenarios for choosing where to take the next step. In the case of a tie with the forward position, you should always choose to go straight forward. In the case of a tie between the two non-forward locations, you should flip a coin to choose where to go.



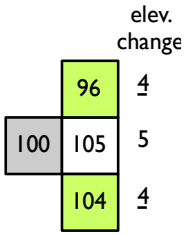
Case 1: smallest change is 5, go fwd-down



Case 2: smallest change is 3, go fwd



Case 3: smallest change is a tie (3), fwd is an option, so go fwd



Case 4: smallest change is a tie (4), choose randomly between fwd-up or fwd-down

There are other ways to choose a path through the mountains. Note that this algorithm will find a path but it will not be guaranteed to be the best path.

The total elevation change is the sum of the absolute values of the difference between the elevation in each cell pair on the path summed together. The steepest elevation change is the largest of the absolute value of these differences. For example, if the path had elevations of `100, 101, 105, 100`, to compute the total elevation change you would do the following: `abs(100 - 101) + abs(101 - 105) + abs(105 - 100) = 1 + 4 + 5 = 9`. Therefore, `9` is the total elevation change and the steepest slope is `5`.

Implementation Guidelines:

We strongly suggest that you start by working with one of the very small provided files. This will make it easier to find the cause of any bugs you run into as the data will be small enough to look through easily. However, note that if you try to view the image on the `DrawingPanel` the small input files will be so small the image will be hard to see. Zoom in to see these more clearly.

We suggest that you first make sure that you can read your data in to an array of arrays correctly. When you have done that, write code to find the peak. Only then move on to finding the path. Start by just outputting the path to the console. Once you can output it to the console correctly then move on to outputting the image.

Stylistic Guidelines:

For full credit you must implement and use a `Location` type that stores an x, y coordinate and an elevation. It is up to you what state it stores and what methods it contains. However, keep in mind, if an operation is highly connected to the type (comparison of locations, distances, output format, etc), it should be done by the class.

You do not need to store `Location`s in your array of arrays but you should use them to keep track of x, y locations and elevations throughout your program, like the peak and the coordinates of the path.

Your methods should be well-structured and avoid redundancy, and your `main` function should be a concise summary of the overall program. Avoid "chaining," which is when many functions call each other without ever returning to `main`.

For this assignment you are limited to the parts of the Java language we have covered in class and you used in 141. If you are unsure if you are allowed to you something, please check with Allison.

Follow past stylistic guidelines about indentation, line lengths, identifier names, and localizing variables, and commenting at the beginning of your program, at the start of each method, and on complex sections of code. Describe your parameter and return values in your comments. **You may not have global variables (except constants) in** `MountainPath.java`.