Welcome to the Mountain Valleys project. This is an interesting use of authentic two-dimensional data to predict the flow of water moving downstream.

This is another "Pair Programming" Assignment

While you are working on this project, you will use the "Pair Programming" approach. In pair programming, two programmers share one computer. One student is the "driver," who controls the keyboard and mouse. The other is the "navigator," who observes, asks questions, suggests solutions, and thinks about slightly longer-term strategies. You should switch roles about every 20 minutes. For this assignment each partner will receive the same grade.

Be sure that both of you have duplicate saved copies of your work each day. You need to have something to work with tomorrow in the event your partner is absent.

---

acknowledgements to Baker Franke for this assignment

There is a wealth of authentic data that can provide fascinating insights when properly analyzed and presented. For example, the National Oceanic and Atmospheric Administration (NOAA) has accurately mapped the land elevations of most of the United States, and freely provides that data to anyone who wants it. Consider a subset of that elevation data representing a portion of Colorado (mountains!), presented as a file of integer values. Each value represents an elevation (in meters above sea level) spaced about 100 meters horizontally apart from each other. The file data can be read into a two-dimensional array, which can be used to determine, for example, relative highs and lows that could be used to graphically display the topography of mountains and valleys. Imagine a raindrop falling somewhere on the land represented by the data. In which direction would the drop of water flow (downhill, obviously). Where would it eventually wind up? Could this data be used to model the risk of potential storm water flows?

Your assignment is to use the provided elevation data to display graphically on a map, then to draw the likely path of water drops as they flow downhill from higher elevations. A graphic display is provided for you, but you must determine how the information will be displayed.

Your task is to finish the partially completed `DataPlotter` methods `readValues()`, `plotData()`, `plotDownhillPath(int x, int y)`, and `plotAllPaths()`.

`readValues()`

Begin by reading in the values from the data file into a two-dimensional array. The first two integers in the file represent the number of rows and columns in the data. Instantiate the instance variable array `grid` to the proper size (rows and columns), and then read all of the data into the array (using nested for loops). The scanner object has already been set up for you, as well as the statements that read the first two numbers (rows and columns). The statement `fileReader.nextInt()` will (obviously) read the next integer from the file. Once you have placed all the values into the array, the values in the four corners of the `grid` array should be as follows:

upper left:	2564	upper right:	1104
lower left:	1446	lower right:	1110

Test your work (print the four corner values of your `grid` array to the terminal window) before continuing.

`plotData()`

Here, iterate thought the entire array again, finding the highest and lowest values of all the data. The highest elevations are going to be displayed in white, and the lowest in black, and everything else in the grey scale colors in between. The graphic display uses an 8-bit color scheme, so a color of (255, 255, 255) is white, and a color of (0, 0, 0) is black. Use the highest and lowest values to create a scale factor that will be used to constrain the values between 0 and 255.

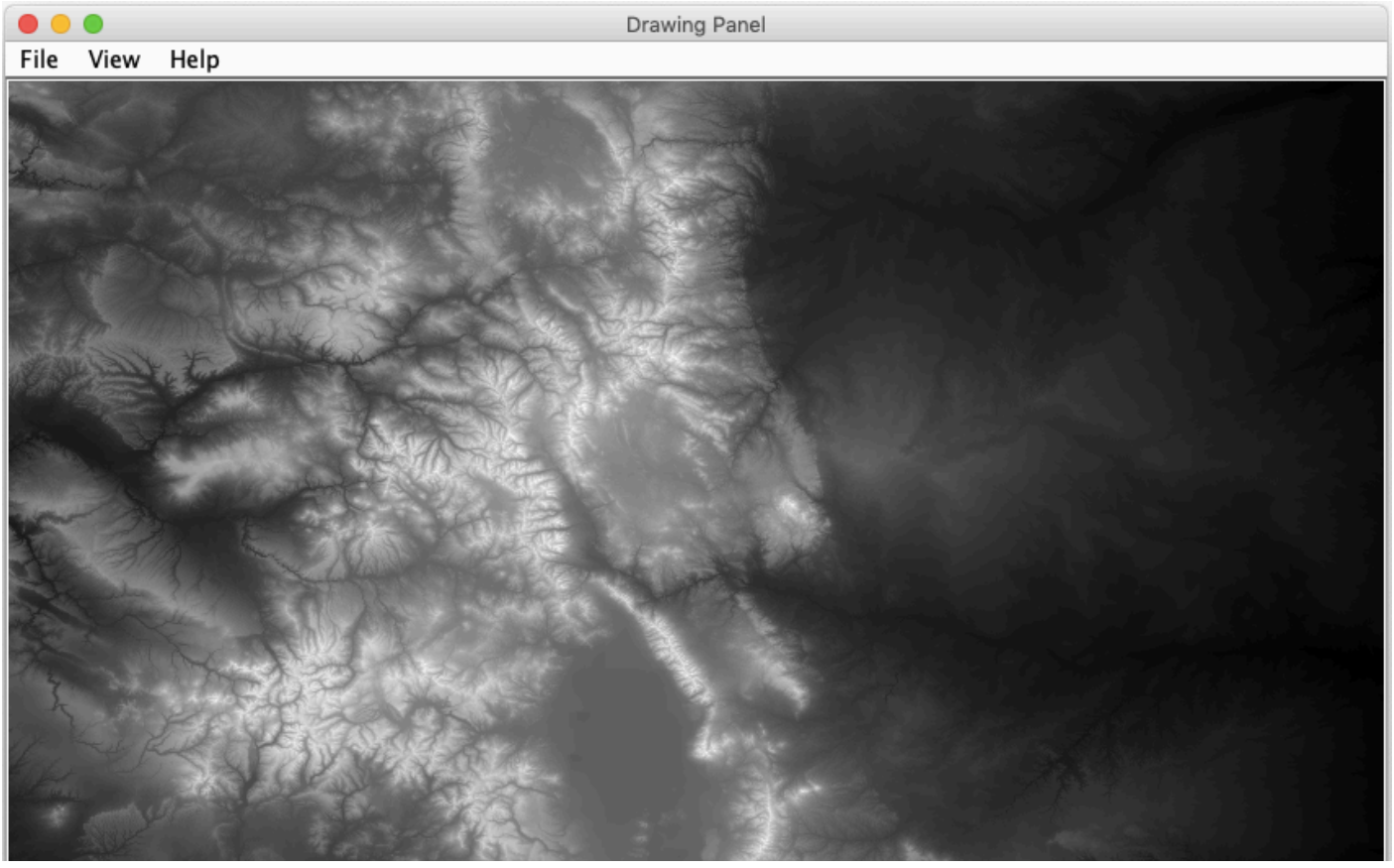
Think about this carefully. What data type should the scale factor variable be? What formula will you use to calculate the scale factor? For example, say that the highest data value was 1000 and the lowest 400. We'd therefore need to use a scale factor of (value – 400) * 0.425 to translate those values between 255 and 0. (Of course your scale factor will be different, because the actual file data values are different than this example.) Be sure to use variables in your formula based on the file data so that your calculation will work correctly for other files with different elevations. Test your calculation by making sure the highest value in the file indeed converts to 255, and the lowest to zero.

Once you have determined your scale factor, iterate though the array once again, scaling each entry and assigning it to an integer variable. Then use that value to then set the right shade of gray and draw a pixel dot on the graphic display for each elevation in the following manner.

```
Color color = new Color(value, value, value);
panel.setPixel(c, r, color);
```

Note that the graphic display is referenced as *column, row* rather than *row, column*. That is because while the array data is row major (row, column), computer graphics are displayed using x, y coordinates, which means (column, row).

When displayed properly, the image should look exactly like the following.



`plotDownhillPath(int x, int y)`

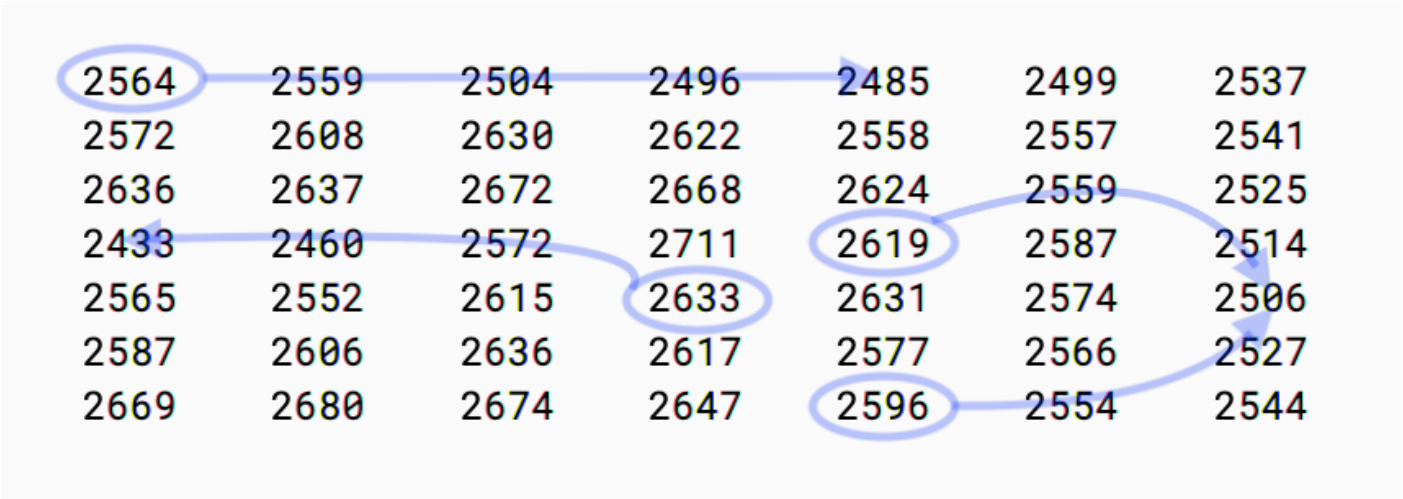
Imaging a drop of water falling from the sky onto a given point on the map. Where would that drop of water flow (downhill, of course)? Specifically, to which one of the 8 locations (North, Northeast, East, Southeast, South, Southwest, West, Northwest) that surround the location will the water flow to? Your task is to check all of the eight surrounding locations of the given point (x, y) for the lowest elevation. If that lowest elevation is lower than your current location, then color it blue (for water), and test again for even lower elevations from this new location (think recursion). Continue in this fashion until you reach a location where every location surrounding it is higher. You should now have a thin blue line from the starting to the ending location.

Note that for a given location (x, y), you can display a blue dot there with the following statement.

```
panel.setPixel(x, y, Color.blue);
```

Be careful not to go outside the map, however. Not every point on the map has 8 neighbors. Specifically, the point where x and y are zero (upper left corner) has no neighbors to the Southwest, West, Northwest, North, or Northeast.

Consider a subset of the actual elevation data as shown below limited to just a 7 x 7 grid. Four example locations are marked with an oval, with arrows depicting the downhill flow from each until it comes to a stop. Note that for our purposes, water does not fall from a single location toward all downhill directions, but rather in the single steepest downhill direction from that location.



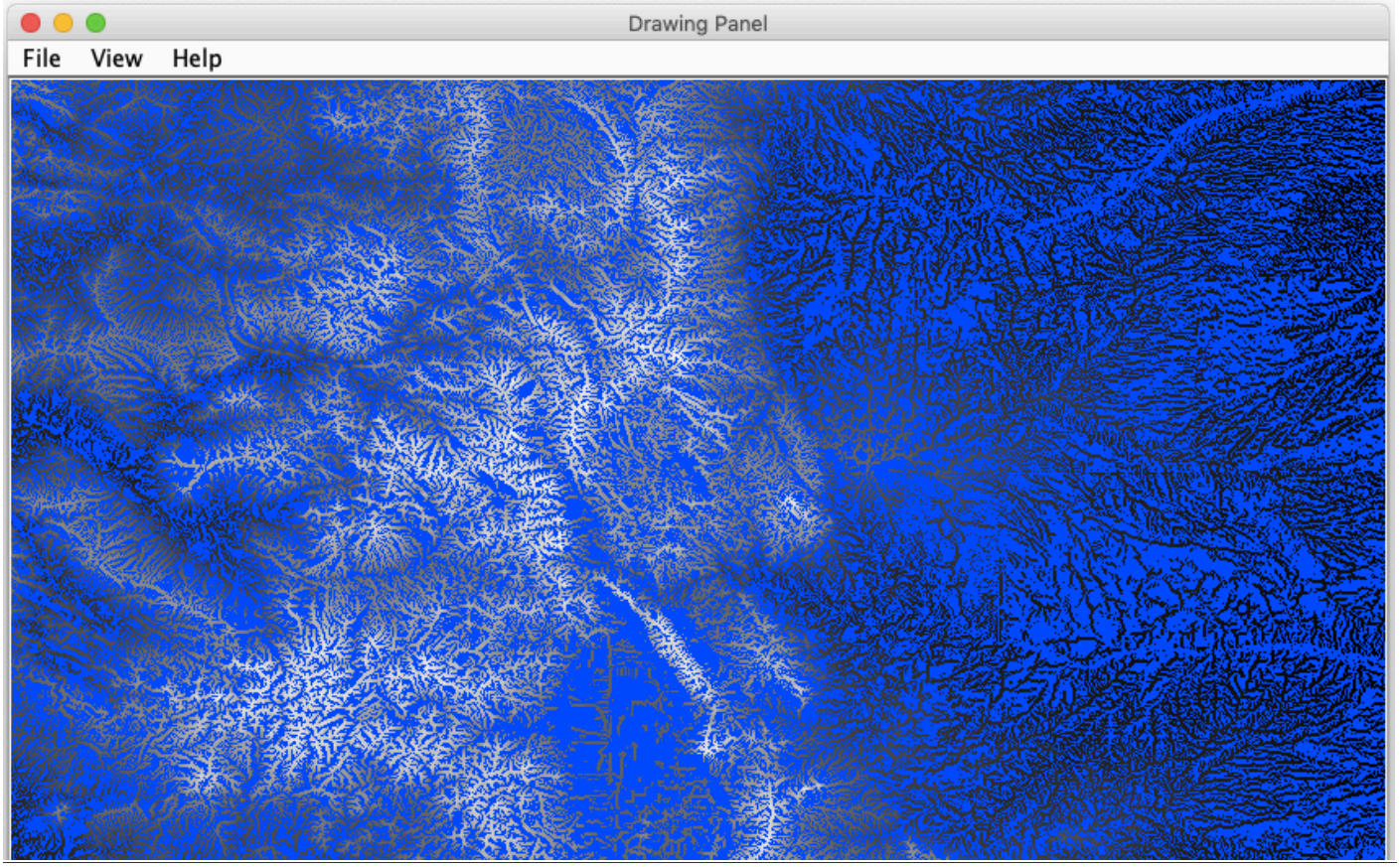
The elevations on the outside edges of the grid have limited options for water flow in order to stay inside the grid. In other words, row and column values to search will never be less than zero, and row values to search will always be less than the total number of rows, and column values to search will always be less than the total number of columns.

Write a routine that will find the flow to the lowest adjacent neighbor and then plots that point blue. Start at row 0 and column 0, and verify that the flow goes four spots to the right and then stops. Print out the chosen elevations to check your work, comparing it against the data listed above (note the actual data set has many, many more rows and columns than those listed above). Then test for row 5, and column 4. Make sure your routine works for any given value of x and y inside the grid. It is difficult to check your work for just one point by looking at the map, however, as a single blue line is very hard to see.

Remember that in your two-dimensional array of integers, values are accessed in *row major* form (rows, columns). The graphic display, however is exactly the opposite (x, y), or (columns, rows).

`plotAllPaths()`

Now, modify this method so that it calls `plotDownhillPath(int x, int y)` for all values of x and y inside the grid. When it is all done, the graphic display should look something like the following.



Notice how the graphic display indicates where water is likely to pool when rainfall is severe. It is obvious that it drains away quickly from the mountain peaks, but then begins to pool in the lowlands. Imagine how data like this could be used to predict flood plains in order to protect against loss of life and property.

Congratulations. You have now finished the Mountain Valleys project. Please demonstrate your work for me to observe. I'll also be giving you another data file to test.