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

This is another "Pair Programming" Assignment

While 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 something to work with tomorrow if your partner is absent.

acknowledgments to Baker Franke for this assignment

Authentic data 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. It 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 to display the topography of mountains and valleys graphically. 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 stormwater flows?

Your assignment is to use the provided elevation data to display graphically on a map and then to draw the likely path of rainwater (and snowmelt) flowing 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 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. The scanner object has already been set up for you, along with a statement that reads the first number (`rows`). Do something similar for the number of columns (`cols`).

Then, instantiate the instance variable array `grid` to the proper size (rows and columns) and read all the data into the array (using nested for-loops). The statement `fileReader.nextInt()` will (obviously) read the next elevation integer from the file.

Once you have placed all the values into the array, test your work by running `DataPlotterTest` to verify that `readValues` is written correctly.

`plotData()`

Here, iterate through the entire array again, finding the data's highest and lowest values. In the display, the highest elevations will be displayed in white, the lowest in black, and everything else in grayscale colors. 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.

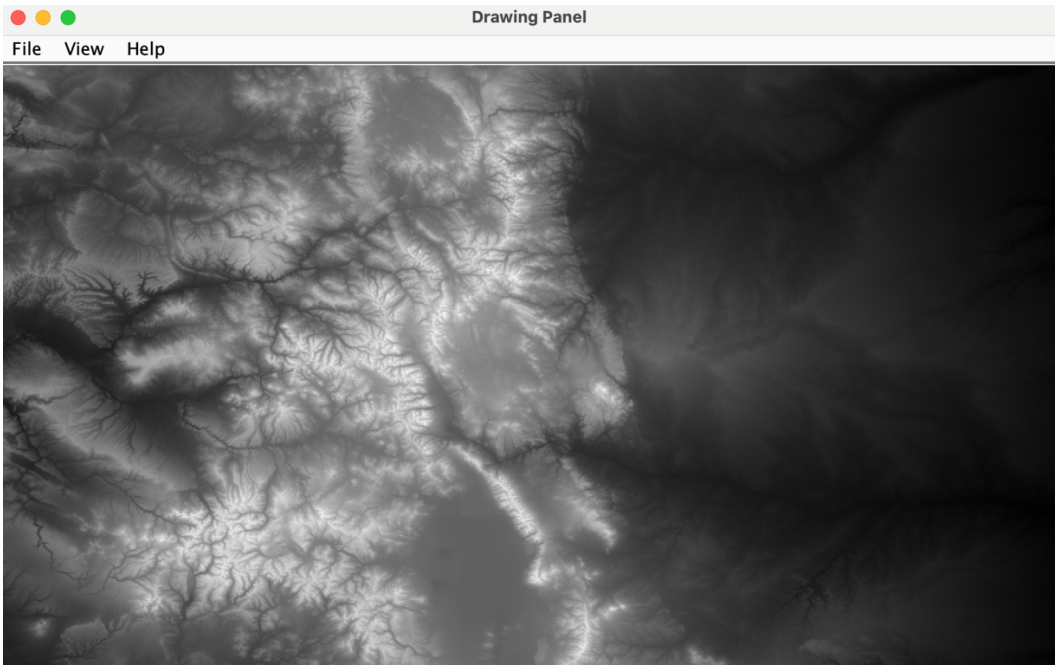
Think about this carefully. What data type should the scale factor variable be? What formula will you use to calculate the scale factor? For example, say that the highest data value was 1000 and the lowest 400. Therefore, we'd need to use a scale factor of 255 / (1000 - 400) or 0.425. Then, we'd use the formula (value – 400) * 0.425 to translate those values between 255 and 0. (Of course, your scale factor will differ because the actual file data values differ from 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.

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

```
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). The top-left corner of a computer display is (0, 0). Like a Cartesian coordinate system, as x values increase, the location coordinate moves to the right (columns). Counterintuitively, however, as y values increase, the location moves *down* (rows).

Run the `main` method in `DataPlotter` and observe the image displayed. When displayed correctly, the image should look exactly like the following.



Test your work by running `DataPlotterTest` to verify that `plotData` is written correctly.

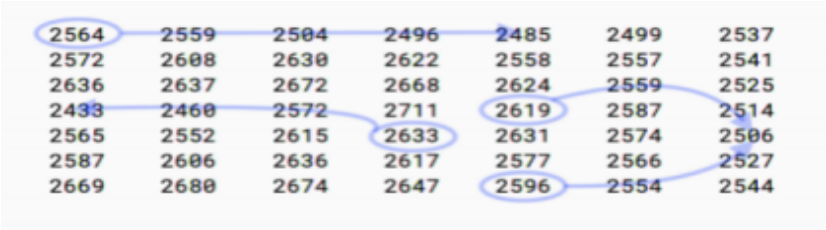
`plotDownhillPath(int x, int y)`

Imagine a drop of water falling from the sky onto a given point on the map. Where would that drop of water flow? Precisely, to which one of the eight locations (North, Northeast, East, Southeast, South, Southwest, West, Northwest) that surround the location will the water flow? Your task is to check the eight surrounding locations of the given point (x, y) for the lowest elevation.

(x - 1, y - 1)	(x, y - 1)	(x + 1, y - 1)
(x - 1, y)	**(x, y)**	(x + 1, y)
(x - 1, y + 1)	(x, y + 1)	(x + 1, y + 1)

Be careful not to go outside the grid, however. Not every position on the grid has eight neighbors. Specifically, the location where x and y are zero (upper left corner) has no neighbors to the SW, W, NW, N, or NE.

Consider a subset of the actual elevation data as shown below, limited to 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 to stay inside the grid. In other words, row and column values to search will never be less than zero, 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.

One way to resolve this issue is to declare six local variables.

```
int xMinus1 = Math.max(0, x - 1);
int yMinus1 = Math.max(0, y - 1);
int xPlus1 = Math.min(cols - 1, x + 1);
int yPlus1 = Math.min(rows - 1, y + 1);
int lowestX = x;
int lowestY = y;
```

Then, using nested for-loops and the loop variables `testX` and `testY`, iterate from `xMinus1` to `xPlus1` (inclusive) and `yMinus1` to `yPlus1` (inclusive). Check to see if `grid[testY][testX]` is lower than `grid[lowestY][lowestX]`.

After you have checked the eight surrounding locations, see if the lowest elevation is lower than your current location, and if so, 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 (`lowestX`, `lowestY`), you can display a blue dot with the following statement.

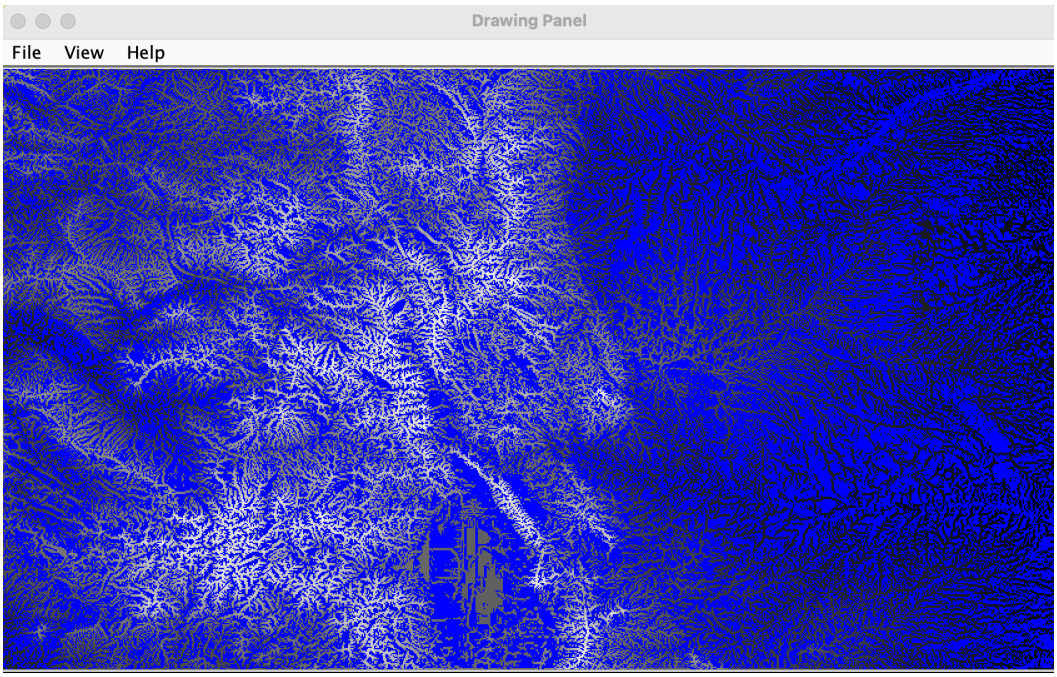
```
panel.setPixel(lowestX, lowestY, Color.blue);
```

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

Test your work by running `DataPlotterTest` to verify that `plotDownhillPath` is written correctly.

`plotAllPaths()`

Now, modify this method to call `plotDownhillPath(int x, int y)` for all values of x and y inside the grid. When done, run the `main` mnethod in `DataPlotter` to observe the graphic display, which should look like the following.



Notice how the graphic display indicates where water is likely to concentrate when rainfall is severe. It is apparent 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 to protect against loss of life and property.

But wait, there's more

Ah, not all elevations are positive numbers. What if we looked at a different elevation data set, say Death Valley (California), where many elevation measurements are below sea level? Would your program still work correctly? The tester will check these values as well.

Test your work by running `DataPlotterTest` to verify that `plotAllPaths` is written correctly. When finished with the entire project, please submit a screenshot of the tester's success message.