CSCI 136 Programming Exam #1
Fundamentals of Computer Science II
Spring 2015

This part of the exam is like a mini-programming assignment. You will create a program, compile it, and debug it as necessary. This part of the exam is open book and open web. You may use code from the course web site or from your past assignments. When you are done, submit all your Java source files to the Moodle exam #1 dropbox. Please ***double check you have submitted all the required files***.

You will have 100 minutes. No communication with any non-staff members is allowed. This includes all forms of real-world and electronic communication.

Grading. Your program will be graded on correctness and to a lesser degree on clarity (including comments) and efficiency. Partial credit is possible, so strive to provide a solution that demonstrates you know how to do as many parts of the problem as possible (even if there are bugs tripping up the total solution).

Overview. You are developing an interactive flood simulator. Your simulator loads a file containing elevation data about a square area of land. You can then simulate a flood starting from any grid location on the map with a given flood height. The water spreads until it hits land that is at least as high or higher than the flood height. To get started, download the zip file containing a stub class as well as a variety of test data files: http://katie.mtech.edu/classes/csci136/flood.zip

Part 1: ElevationData. This is the workhorse of the flood simulator. This data type stores the elevation data for every location on a 2D square grid. Here is the API:

```
public class ElevationData
```

```
        ElevationData(String filename) throws FileNotFoundException
    int getSize()
boolean validLocation(int x, int y)
 double getElevation(int x, int y)
   void setElevation(int x, int y, double elevation)
 double getMaxElevation()
 double getAvgElevation()
    int flood(int x, int y, double depth)
    int mouseToIndex(double mousePosition)
   void draw()
```

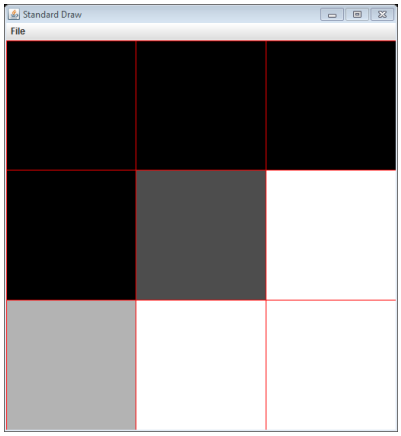
You need to implement all the methods except for `draw` (this method has been completed for you). Here are further details about each method you need to implement (also there are comments in the stub code):

`getSize`	Returns the size dimension of the map, e.g. for `tiny.txt` it would return 3.
`validLocation`	Returns true if the specified x- and y-index are a valid location given the map that was loaded, returns false otherwise.
`getElevation`	Returns the elevation at the given x- and y-index location. If the location is off the map, returns Double.NaN
`setElevation`	Attempts to set the elevation at the given x- and y-index location. If the location is off the map, does not do anything (but does not crash).
`getMaxElevation`	Returns the highest elevation currently in the data. Before any flooding, this is the highest point of land. After flooding, it could be the height of the biggest flood event.
`getAvgElevation`	Computes the average elevation of the currently stored data.
`flood`	Simulates a flood starting at the specified x- and y-index location. The flood spreads in all directions including diagonally. The flood stops if it hits the edge of the map or if it encounters a location with an elevation greater than or equal to the flood depth. The method returns the total number of grid locations that were flooded by the call.
`mouseToIndex`	Used by the interactive flood simulator to convert a floating-point mouse x- or y-location in the unit box to the corresponding x- and y- index on the map grid.

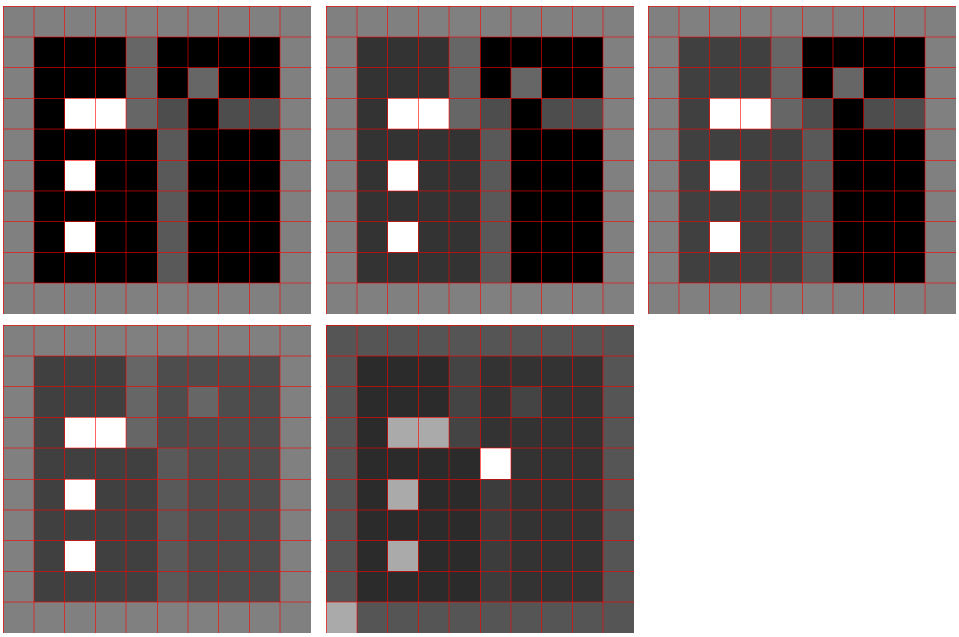
The elevation data is loaded from a text file that starts with an integer specifying the dimension of the grid. This is followed by zero or more 3-tuples. Each 3-tuple gives the x- and y-index locations of a grid location and a non-negative floating-point elevation of the land at that location. Locations not appearing in the file are assumed to have an elevation of 0.0 (sea level). You can assume the 3-tuples are well-formed and valid locations on the map. For example here is `tiny.txt`:

```
3
0 0 0.7
1 0 1.0
2 0 1.0
2 1 1.0
1 1 0.3
```

The `draw` method provides a visualization of the loaded elevation data. The maximum elevation is colored pure white, elevations of 0.0 are black. NOTE: the grid location (0, 0) is in the lower-left corner.



We have provided a test `main` method that loads the file `map10.txt`. The program prints out various information as well as displays a graphical progression of the map as things change. Here is our output:



```
% java ElevationData
BEFORE
Map size      : 10
Max elevation : 2.0000
Avg elevation : 0.5250
( 0,  0) 1.0000 true
( 3,  4) 0.0000 true
( 5,  5) 0.7000 true
(10, 10)    NaN false

flood(5, 5, 0.2) = 0
flood(3, 1, 0.4) = 25
flood(3, 1, 0.5) = 25
flood(3, 1, 0.5) = 0
flood(7, 3, 0.6) = 23

AFTER
Map size      : 10
Max elevation : 3.0000
Avg elevation : 0.8210
( 0,  0) 2.0000 true
( 3,  4) 0.5000 true
( 5,  5) 3.0000 true
(10, 10)    NaN false

mouse (0.00, 0.15) -> (0, 1)
mouse (0.05, 0.25) -> (0, 2)
mouse (0.10, 0.51) -> (1, 5)
mouse (0.95, 0.99) -> (9, 9)
```

Part 2: FloodSim. Create an interactive client program `FloodSim` that makes use of your `ElevationData` class. You need to create this from scratch (no stub code is provided). The program requires two command line arguments, the filename of the elevation data to load and the floating-point flood depth to simulate. If fewer than two arguments are given, the program prints an error message and terminates:

```
% java FloodSim
FloodSim <filename> <depth>
```

If the filename specified as the first command line argument is not found, the program prints an error message indicating the missing filename and terminates:

```
% java FloodSim bogus.txt 1.0
Failed to load: bogus.txt
```

The program displays the current elevation data and waits for the user to hit the spacebar. When they hit the spacebar, a flood is simulated starting from the location under the user's mouse pointer. The new visualization of the map is shown. Additionally, the location and number of flooded squares is printed to standard output. The user can carrying on, triggering floods at additional locations. If the user hits the 'q' key, the program prints out the total number of floods triggered by the user and says goodbye:

```
% java FloodSim map10.txt 0.5
flood at (7, 3), 23 squares flooded
flood at (3, 5), 25 squares flooded
flood at (5, 5), 0 squares flooded
flood at (3, 6), 0 squares flooded
4 flood(s), bye bye!
```

Here is another example run:

```
% java FloodSim strange.txt 1.0
flood at (5, 28), 30 squares flooded
flood at (4, 19), 6 squares flooded
flood at (15, 18), 31 squares flooded
flood at (16, 31), 83 squares flooded
flood at (23, 31), 45 squares flooded
flood at (37, 24), 34 squares flooded
6 flood(s), bye bye!
```