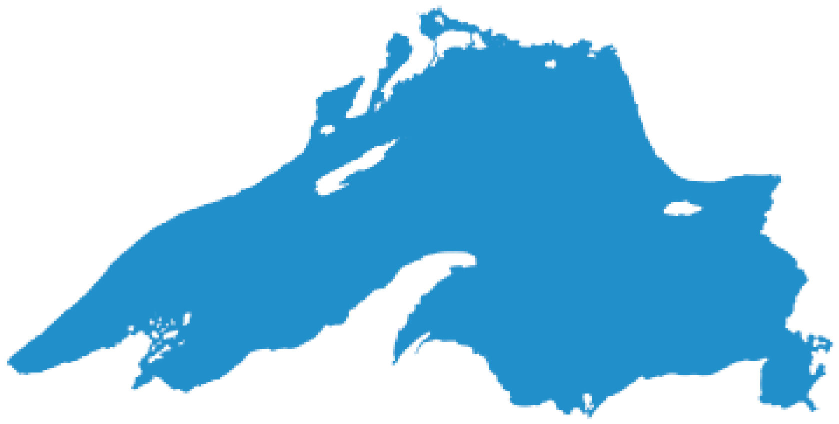
CSCI 136 Programming Exam #0
Fundamentals of Computer Science II
Spring 2015

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 #0 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 searching for a shipwreck with the help of one or more sets of sonar data recorded by a surface ship. You are to develop the classes need to store and query this collection of data. To get started, download the zip file containing stub classes as well as a variety of test data files:
http://katie.mtech.edu/classes/csci136/sonar.zip

Part 1: SonarData. This class loads a grid of sonar data from a text file. It keeps track of all the readings. Clients can ask for the size of the grid or for a reading at a specific location. It can also generate a ASCII representation of its data. Here is the API:

```
public class SonarData
-----------------------------------------------------------------------------------
       SonarData(String filename) throws FileNotFoundException
   int getWidth()
   int getHeight()
double getReading(int x, int y)
String toString()
```

See the provided stub code in `SonarData.java` for details about what each method does. The constructor of SonarData loads data from a text file, for example here is `data-tiny1.txt`:

```
10 5
-1.0 -1.0 -1.0 -1.0  0.0  0.0  0.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0  0.0  0.2  0.0  0.6  0.4 -1.0 -1.0
-1.0  0.0  0.0  0.1  0.0  0.0  0.5  1.0  0.1 -1.0
 0.0  0.3  0.0  0.0  0.0  0.0  0.0  0.9  0.0 -1.0
 0.0  0.0  0.0 -1.0 -1.0  0.0  0.0  0.0  0.0  0.0
```

This file represents a very low resolution scan of Lake Superior. It has 10 readings in the x-dimension (width) and 5 in the y-dimension (height). The width and height are specified by the first two integers in the file. This is followed by the readings at each location, starting with (x=0, y =0), (x=1, y=0), etc. A reading with the value -1.0 indicates land. Positive values are the strength of the sonar return.

The `toString` method in `SonarData` visualizes the data using ASCII text. The above file look like this:

```
XXXX...XXX

XXX.+.@+XX

X..+..+@+X

.+.....@.X

...XX.....
```

Here are the rules for how readings appear:

X	readings below 0.0.
.	readings greater than or equal to 0.0 and strictly less than 0.1
+	readings greater than or equal to 0.1 and strictly less than 0.6
@	readings greater than or equal to 0.6

The methods `getWidth` and `getHeight` return the size of the loaded data. `getReading` returns the reading at the specified (x, y) location. You can assume the caller to `getReading` has specified a valid location.

Part 2: SonarDataCollection. This class holds a collection of sonar data sets. Here is the API:

```
public class SonarDataCollection
-------------------------------------------------------------------------------
       SonarDataCollection(String filename) throws FileNotFoundException
   int getWidth()
   int getHeight()
   int getSize()
double getMinReading(int x, int y)
```

See the provided stub code in `SonarDataCollection.java` for details about what each method does. The constructor of `SonarDataCollection` loads its data sets from a series of files specified in the filename given to the constructor. For example, here is the file set-tiny2.txt:

```
data-tiny1.txt
data-tiny2.txt
```

This file specifies the collection should load two SonarData objects from the files `data-tiny1.txt` and `data-tiny2.txt`. As the constructor loads the data sets, it also prints out an ASCII representation of each set. For example if given `set-tiny2.txt`, the constructor would print:

```
XXXX...XXX
XXX.+.@+XX
X..+..+@+X
.+.....@.X
...XX.....

XXXX...XXX
XXX.+.++XX
X+....+@.X
.+.....@+X
...XX...+.
```

If the file containing the list of data files is not found, or if any of the data files is not found, the constructor throws a `FileNotFoundException`. If loading multiple data sets, any set with a size different from the first data set is not added to the collection and the error message shown below is printed. However, ***subsequent data sets should still be loaded***. Here is the output of the constructor for the file `set-tiny-mismatch.txt`:

```
XXXX...XXX
XXX.+.@+XX
X..+..+@+X
.+.....@.X
...XX.....

Filename data-small1.txt has a different size, skipping!

XXXX...XXX
XXX.+.++XX
X+....+@.X
.+.....@+X
...XX...+.
```

The methods `getWidth` and `getHeight` return the size of the loaded data set(s). If no data sets were loaded successfully, these methods return -1.

`getSize` returns the number of loaded data sets.

`getMinReading` returns the minimum reading in the loaded data sets at a specified (x, y) location. If any of the data sets has land at the specified location, it returns -1.0. If no data sets are loaded, it returns `Double.POSITIVE_INFINITY`. You can assume the caller to `getMinReading` has specified a valid location.

Part 3: Sonar. This is the main program that performs detection based on a set of sonar data sets. The program takes two command line arguments, the first specifies the file containing the list of data set filenames. The second is a floating-point threshold.

The program outputs the data visualization (as provided by the `SonarDataCollection` constructor). The total number of files loaded successfully is then printed. This is followed by all (x, y) locations where the reading is greater than or equal to the threshold value along with the minimum value at that location (rounded to two decimal places). The locations are checked starting at (0, 0), preceding horizontally checking (1, 0), (2, 0), and so on. After completely the top row, it checks (0, 1), (1, 1), (2, 1), and so on. Finally the program outputs the total count of locations that met the threshold. Here is an example run:

```
% java Sonar set-tiny1.txt 0.5
XXXX...XXX
XXX.+.@+XX
X..+..+@+X
.+.....@.X
...XX.....

Loaded files: 1

(6, 1) = 0.60
(6, 2) = 0.50
(7, 2) = 1.00
(7, 3) = 0.90

Total hits: 4
```

If the filename provided to `Sonar` is not found, or any of the files contained in it are not found, `Sonar` prints the following error message an the program exits:

```
% java Sonar bogus.txt 0.5
Failed to load data!

% java Sonar set-tiny-missing.txt 0.5
Failed to load data!
```

If the 2nd argument is not parseable as a floating-point, `Sonar` prints the following error message and exits:

```
% java Sonar set-tiny1.txt five
Threshold must be a floating-point value!
```

Here are some other example runs:

```
% java Sonar set-small2.txt 0.8
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX.....XXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXX...........XXXXXXXXXXXX
XXXXXXXXXXXXXXXXX.@........++XXXXXXXXXXX
XXXXXXXXXXXXXX...++.........+XXXXXXXXXXX
XXXXXXXXXXXXXX...+......+....XXXXXXXXXXX
XXXXXXXXXXXXXX++.........+....XXXXXXXXXX
XXXXXXXXXXXXX..++.............XXXXXXXXXX
XXXXXXXXXXXX...+...................XXXXX
XXXXXXXXX...++........+............XXXXX
XXXXXXX..+..+..+.....+++...........XXXXX
XXXXXX...+....++......+.........@...XXXX
XXXXX.........@.+...........+.......XXXX
XXXX....+.+.......X........+@.......XXXX
XXX..............X..........++......XXXX
XX.+.+..@.......XX...........+.......XXX
X...........@.XXXX.X..................XX
X...XX......XXXXXXXXXX...............XXX
XXXXXX....XXXXXXXXXXXX.......XXXX....XXX
XXXXXXXXXXXXXXXXXXXXXXX....XXXXXXX...XXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX.....XXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXX....+......XXXXXXXXXXXX
XXXXXXXXXXXXXXXXX.@.......+..XXXXXXXXXXX
XXXXXXXXXXXXXX...++.........+XXXXXXXXXXX
XXXXXXXXXXXXXX...+......+....XXXXXXXXXXX
XXXXXXXXXXXXXX.+...+.....+....XXXXXXXXXX
XXXXXXXXXXXXX..++..++.........XXXXXXXXXX
XXXXXXXXXXXX...+...................XXXXX
XXXXXXXXX...++.....................XXXXX
XXXXXXX..+..+..+.....+@+........@..XXXXX
XXXXXX...+.....@......+.........@...XXXX
XXXXX...........+...........+.......XXXX
XXXX....+.+.......X.....+..+@.......XXXX
XXX..............X.....+@.......+@..XXXX
XX.+........+...XX..............+....XXX
X..........+@+XXXX.X..................XX
X...XX......XXXXXXXXXX......+........XXX
XXXXXX....XXXXXXXXXXXX......+XXXX....XXX
XXXXXXXXXXXXXXXXXXXXXXX....XXXXXXX...XXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Loaded files: 2

(32, 11) = 0.80
(28, 13) = 0.90
(12, 16) = 0.90

Total hits: 3
```

```
% java Sonar set-tiny-mismatch.txt 0.4
XXXX...XXX
XXX.+.@+XX
X..+..+@+X
.+.....@.X
...XX.....

Filename data-small1.txt has a different size, skipping!

XXXX...XXX
XXX.+.++XX
X+....+@.X
.+.....@+X
...XX...+.

Loaded files: 2

(6, 2) = 0.40
(7, 2) = 0.90
(7, 3) = 0.80

Total hits: 3

% java Sonar set-large.txt 0.5
...(output omitted)...

Loaded files: 2

(45, 11) = 0.50
(46, 11) = 0.60
(29, 12) = 0.70
(28, 13) = 0.70
(29, 13) = 0.70
(28, 14) = 0.80
(30, 14) = 0.50
(29, 15) = 0.50
(14, 23) = 0.70
(15, 23) = 0.60
(15, 24) = 0.90
(16, 24) = 0.60
(15, 25) = 1.00
(16, 25) = 0.80
(15, 26) = 0.90
(68, 31) = 0.50

Total hits: 16
```