CSC 110: Fundamentals of Programming I
Assignment #6: 2D arrays

Due date

(Note the change in usual day-of-week & time!)

November 21st, 2016 (Monday) at 11:30 am (i.e., before noon) via submission to conneX.

How to hand in your work

Submit the requested files for Part (b) (see below) through the Assignment #6 link on the CSC 110 conneX site. Please make sure you follow all the required steps for submission (including confirming your submission). Part (a) is not marked.

Learning outcomes

When you have completed this assignment, you should understand:

- How to create and work with *two-dimensional (i.e., 2D) arrays*.
- How to design code that is tolerant of *error cases*.

Part (a): Problems from the Textbook

Complete Chapter 7 self-check problems 27, 28 and 29 and compare your answers with those given at:

`http://www.buildingjavaprograms.com/self-check-solutions-4ed.html`

Part (b): Minesweeper!

Problem description

Your task in this assignment is to create *MineSweeper.java*. This will be a somewhat simplified version of the original "Minesweeper" game. You can find the rules for the full game at:

- *https://en.wikipedia.org/wiki/Minesweeper_(video_game)*

and you can try out one version of the full game at:

- *http://minesweeperonline.com*

Your version will be simpler than these. In particular:

1. Your grid will be eight by eight in size with ten mines. *The mines are to be placed randomly.* (Be sure to use the *Random* class which is discussed on page 320 of the text. You are not permitted to use *Math.random*).

2. Your input will be from the computer keyboard instead of via the mouse. You will output all information to the console window. *If the user enters an invalid grid cell location, you must indicate the error and ask until you get a valid cell.*

3. The method for clearing blank grid cells will be much simpler. If the user uncovers a blank cell, you must automatically uncover only the eight adjacent cells (and your method does not cascade any further than this).

4. There will be no score and no timing. *The user wins* if they uncover all cells except for those containing mines. *The user loses* if they directly uncover a cell containing a mine.

The input and output of your program must match the sample shown in *SampleMineSweeperOutput.pdf*. (Note: In this PDF the text in red boxes is meant to explain how the program output is related to actual game play. You are not meant to produce the text in these boxes.)

Methods

Your solution **must** implement and use at least the following methods (but your solution may use more). You are to decide some of what is to passed in as parameters and what is to be returned. I have used "???" to refer to some type of 2D array; you are free to choose an appropriate type, but the parameter must be a 2D array (i.e., ArrayLists, etc. are not permitted).

1. Write the following method:

   ```
   public static ??? initializeFullGrid(???)
   ```

   where "???" corresponds to types/parameters you must specify. The method will place ten bombs at random in the game grid. For each grid cell that does not contain a bomb, record the number of bombs adjacent to the cell.

2. Write the following method:

   ```
   public static ??? revealGridCell(int row, int col, ???)
   ```

   where "???" corresponds to types/parameters you must specify. If the cell is blank then also uncover all neighbouring cells. Note: You must be careful of edge cases when uncovering neighbours. If the grid cell is at an edge, it will have fewer than eight neighbours.

3. Write the following method:

   ```
   public static ??? drawFullGrid(???)
   ```

   where "???" corresponds to types/parameters you must specify. The method will draw the current state of the game board to the console window.

Ensure that your code handles invalid row or column indices throughout the program so that invalid array locations are never accessed (i.e., will not result in the program crashing with an *ArrayIndexOutOfBoundsException*).

Implementation ideas

- Use a 2D array of some type to keep track of the game board. In each cell you could indicate the number of bombs neighbouring the cell. You'll want some way to indicate that the cell itself has a bomb.

- If needed, you can use a second 2D array to keep track of which grid cells have or haven't been uncovered.

What the marker will look for

- Compiles & runs on the lab machines.

- Implements the expected gameplay and produces the expected output (minor variations in visual appearance are acceptable if they do not change the game

play).

- Correctly implements and uses the requested methods.
- Handles possible error cases appropriately (in particular, invalid grid cell locations and grid edge cases).
- Is well-structured through the use of meaningful methods.
- Is appropriately documented and matches the style guidelines.

Files to submit: ***MineSweeper.java*** *via the Assignment #6 link on conneX.*

Grading scheme

- "A" grade: An exceptional submission demonstrating creativity and initiative going above and beyond the assignment requirements. The program runs without any problems using two-dimensional arrays and is decomposed into methods. Any extra work appears in a file named *MineSweeperExtra.java*, and identified within the file (i.e., Class comment) is how you have extended the assignment to demonstrate creativity and initiative.
- "B" grade: A submission completing the requirements of the assignment. The program runs without any problems using two-dimensional arrays and is decomposed into meaningful methods.
- "C" grade: A submission completing most of the requirements of the assignment. The program runs with some problems but uses two-dimensional arrays and is decomposed into methods.
- "D" grade: A serious attempt at completing requirements for the assignment. The program runs with major problems, or does not use two-dimensional arrays or is not decomposed into methods.
- "F" grade: Either no submission given, the submission does not compile, or submission represents very little work.