CS 110 Final Project

Due Monday, May 10th, end of day (11:59pm). No late submissions will be accepted.
Early Due Date (+5 points) Friday, May 7th, end of day.
You may reach a maximum score of 105 (out of 100).

All remaining class time and lab time will be dedicated office hours to answer questions on this project. No meeting will be started, contact Jackie via Teams chat with questions.

Read through the instructions carefully. Only concepts learned in CS110 may be employed in this project. No break/continue statements are permitted.

Minesweeper

Minesweeper is a single player puzzle game. The game board consists of a two-dimensional grid of cells. Within the grid, is a number of mine cells. The object of Minesweeper is to find all the mines without uncovering any of them. If you're not familiar with the game, visit http://minesweeperonline.com/. We will be implementing a text-only version of the game.

There will be a core requirement for this project, worth 90% of the total grade. Beyond the 90% you will have options to gain additional points to reach 105%. Your grade for the final project may not exceed 105.

Minesweeper game core requirements

- Create a 10x12 (10 rows, 12 columns) board containing 10 randomly placed mines.
- Initially, all squares are covered and represented with an "x"
    - When a square is uncovered, it will have the following representation:
        - Mine: *
        - A square which is not a mine and has 0 neighboring mines: _ (underscore)
        - A square which is not a mine and has 1 or more neighboring mines: n (the number of neighboring mines)
- Present the user with the following options:
    - `(U)ncover r c, (F)lag r c, (Q)uit`
    - If the user selects *uncover*:
        - If the square selected is a mine, the game is over and the player loses
        - If the square selected has no neighboring mines, uncover the square (change the representation to an underscore and reveal the 5x5 neighborhood), do not reveal mines.
        - If the square selected has 1 neighboring mine, change the representation to the number of neighboring mines and reveal the 3x3 neighborhood, do not reveal mines.
        - If the square selected has more than 1 neighboring mine, uncover only the square.
    - If the user selects flag:
        - Mark the covered cell with an 'f'. This is done when the user wants to mark a cell that they believe is a mine. If a cell is already marked, flagging that cell, removes the flag. A flagged cell cannot be uncovered (you must unflag it first).

Note: any option other than U, F or Q is ignored.

Redisplay the grid after processing each user command.
The game continues until the user chooses to quit, uncovers a mine (loses), or uncovers all cells that are not mines (wins).
You are only required to play one game.
Full javadocs documentation is required on all classes.

Credit beyond core requirements:

IMPORTANT If you wish to receive extra credit, you must add the following (completed with your info) at the top of your Driver.java file. List each each extra credit item.**

```
/**  EXTRA CREDIT CONSIDERATION
     I have added the following functionality, I would like evaluated for extra credit:
                ** item #1
                ** item #2
*/
```

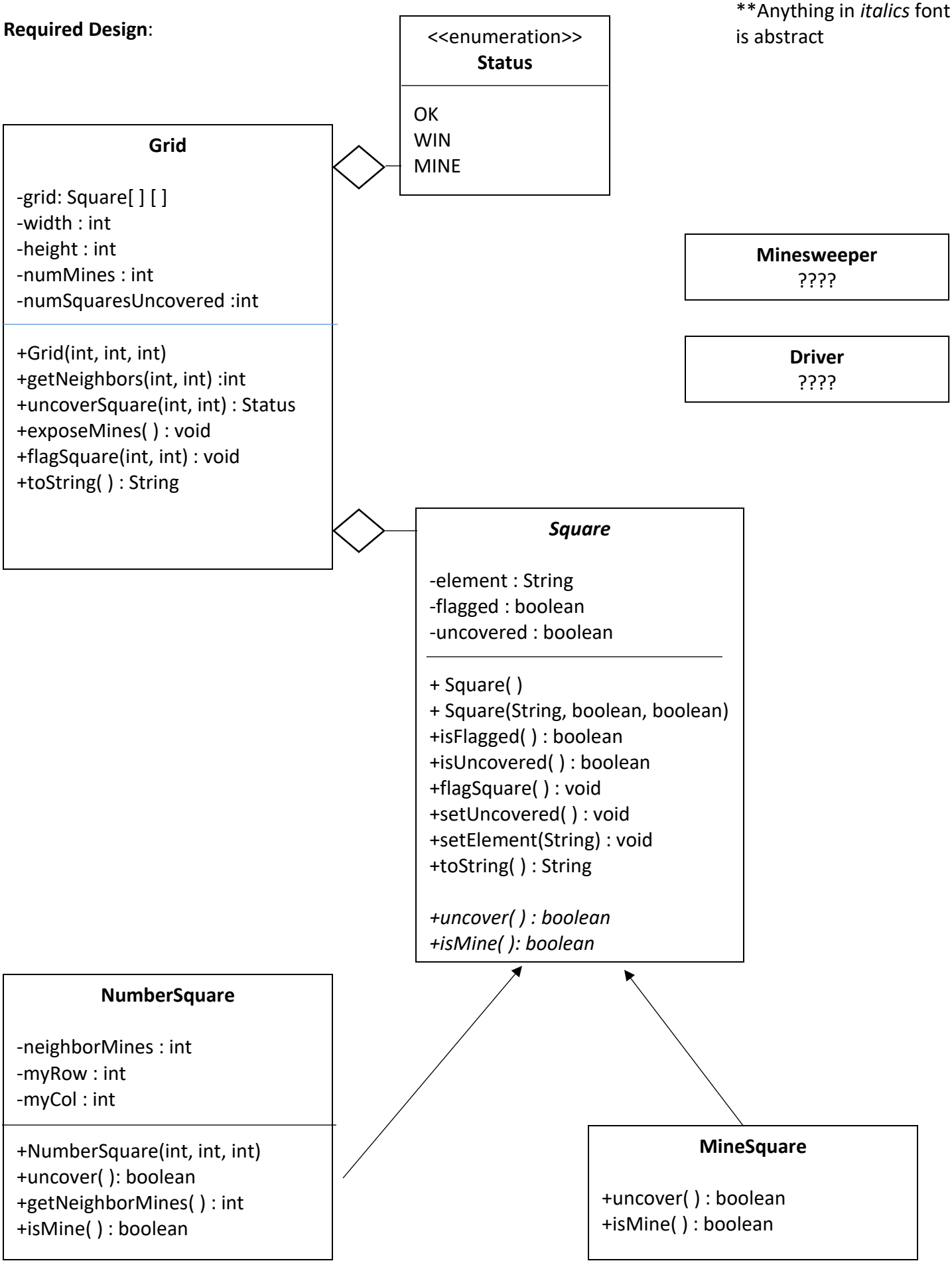
- First selection can't be a mine. In the real game of Minesweeper, the first square selected is never a mine. Implement this functionality in your game. (5 points)
- Throw an exception when the user attempts to uncover a flagged square. You must handle this exception within your code. (5 points)
- Allow the user to select a level when they start. (5 points)
    - Beginner – 8 x 8 grid with 8 mines
    - Intermediate 10 x 12 grid with 10 mines
    - Expert 16 x 20 grid with 50 mines
- Allow the user to restart the game (3 points)
- Early submission (5/7 end of day) (5 points)
- Use recursion and implement the true game of Minesweeper (disregard the 3x3, 5x5 exposure and expose all squares, in each direction until a non-zero `NumberSquare` is exposed.) This must be fully functional to receive the credit. (15 points)

You must use the classes specified in the UML on the next page. For classes that have a complete UML, you must use the listed instance variables and methods. You are free to design the `Minesweeper` class and the `Driver` class anyway you'd like. <u>You will be graded on good OOP principles.</u> Hint: `Minesweeper.java` should contain a `Minesweeper` class that uses the classes provided and provides the methods to play the game. `Driver.java` contains the `main( )` method that will interact with the `Minesweeper` class.

Any work you submit for this assignment should be authored entirely by yourself. Assistance is permitted from the instructor or teaching assistants only. Submitting online code, in whole or part, is strictly prohibited. All submitted programming assignments are subject to originality verification through software designed and used for the Measure Of Software Similarity (MOSS).

Required Design:

**Anything in *italics* font is abstract



`Status` is an enumerated type that may be declared in a file by itself, or within the `Grid` class.

The `Square` class is an abstract class that is meant to be the basis for all `Square`s in the grid.

`element` – the current representation of the `Square`
`flagged` – is the `Square` flagged or not
`uncovered` – is the `Square` uncovered or not

The methods in this class are constructor and getters/setters.

`NumberSquare` extends `Square` and is not abstract. This is a square that is not a mine.

`neighborMines` – the number of adjacent `Square`s that are mines
`myRow`, `myCol` – this `Square`'s location within the grid

`uncover`() method will call `setUncovered`( ) to change the `covered` instance variable and set the `element` instance variable represent what should be displayed in the grid (hint, either "-" or "1", "2", etc). If the square is flagged, this method does nothing.

`MineSquare` extends `Square` is not abstract. This square contains a mine.

No additional instance data, simply implement the inherited abstract methods.

`Grid` is the two dimensional collection of `Square` objects. The default grid size (if not completing 'levels') is 10 rows and 12 columns, with 10 mines randomly placed.

`height` is defined by the number of rows
`width` is defined by the number of columns
`numMines` is 10 (unless specified differently in extra credit work)
`numSquaresUncovered` will start at 0, and increase as mines are uncovered. This variable can be used to determine if the user has won the game.

`getNeighbors(r, c)` will return the number of `Square`s, adjacent to r, c that contain mines. Caution -- make sure you don't go off the edge of the grid.

`uncoverSquare(r, c)`:

- determine if square is a mine → return Status of MINE
- otherwise
  - if number of neighbors is 0
    - expose 5x5 neighborhood. Do not expose mines. Do not go off the end of the grid
  - else if number of neighbors is 1
    - expose 3x3 neighborhood. Do not expose mines. Do not go off the end of the grid
  - else
    - expose on square at r,c
  - update numSquaresUncovered
  - Check if user won (if so, return Status of WIN)
  - Otherwise return Status of OK

`exposeMines( )` – uncover all mines

`flagSquare(r,c )` – tell the appropriate `Square` it has been flagged

Some sample moves:

The board to start (mines indicated in red – this is just for your information, not part of the display)

```
    0  1  2  3  4  5  6  7  8  9 10 11
 0  x  x  x  x  x  x  x  x  x  x  x  x
 1  x  x  x  x  x  x  x  x  x  x  x  x
 2  x  x  x  x  x  x  x  x  x  x  x  x
 3  x  x  x  x  x  x  x  x  x  x  x  x
 4  x  x  x  x  x  x  x  x  x  x  x  x
 5  x  x  x  x  x  x  x  x  x  x  x  x
 6  x  x  x  x  x  x  x  x  x  x  x  x
 7  x  x  x  x  x  x  x  x  x  x  x  x
 8  x  x  x  x  x  x  x  x  x  x  x  x
 9  x  x  x  x  x  x  x  x  x  x  x  x
```

```
What next?
Options: (U)ncover r c,  (F)lag r c, (Q)uit
U 1 1
```

```
    0  1  2  3  4  5  6  7  8  9 10 11
 0  _  _  _  _  x  x  x  x  x  x  x  x
 1  _  _  _  _  x  x  x  x  x  x  x  x
 2  _  _  _  _  x  x  x  x  x  x  x  x
 3  _  _  _  1  x  x  x  x  x  x  x  x
 4  x  x  x  x  x  x  x  x  x  x  x  x
 5  x  x  x  x  x  x  x  x  x  x  x  x
 6  x  x  x  x  x  x  x  x  x  x  x  x
 7  x  x  x  x  x  x  x  x  x  x  x  x
 8  x  x  x  x  x  x  x  x  x  x  x  x
 9  x  x  x  x  x  x  x  x  x  x  x  x
```

The board to start (mines indicated in red)

```
    0  1  2  3  4  5  6  7  8  9 10 11
 0  x  x  x  x  x  x  x  x  x  x  x  x
 1  x  x  x  x  x  x  x  x  x  x  x  x
 2  x  x  x  x  x  x  x  x  x  x  x  x
 3  x  x  x  x  x  x  x  x  x  x  x  x
 4  x  x  x  x  x  x  x  x  x  x  x  x
 5  x  x  x  x  x  x  x  x  x  x  x  x
 6  x  x  x  x  x  x  x  x  x  x  x  x
 7  x  x  x  x  x  x  x  x  x  x  x  x
 8  x  x  x  x  x  x  x  x  x  x  x  x
 9  x  x  x  x  x  x  x  x  x  x  x  x
```

```
What next?
Options: (U)ncover r c,  (F)lag r c, (Q)uit
U 3 2
```

```
    0  1  2  3  4  5  6  7  8  9 10 11
 0  x  x  x  x  x  x  x  x  x  x  x  x
 1  _  _  _  _  _  x  x  x  x  x  x  x
 2  _  _  _  _  _  x  x  x  x  x  x  x
 3  _  _     1  1  x  x  x  x  x  x  x
 4  _  _  _  2  x  x  x  x  x  x  x  x
 5  _  _  _  2  x  x  x  x  x  x  x  x
 6  x  x  x  x  x  x  x  x  x  x  x  x
 7  x  x  x  x  x  x  x  x  x  x  x  x
 8  x  x  x  x  x  x  x  x  x  x  x  x
 9  x  x  x  x  x  x  x  x  x  x  x  x
```

The board to start (mines indicated in red)

```
    0  1  2  3  4  5  6  7  8  9 10 11
 0  x  x  x  x  x  x  x  x  x  x  x  x
 1  x  x  x  x  x  x  x  x  x  x  x  x
 2  x  x  x  x  x  x  x  x  x  x  x  x
 3  x  x  x  x  x  x  x  x  x  x  x  x
 4  x  x  x  x  x  x  x  x  x  x  x  x
 5  x  x  x  x  x  x  x  x  x  x  x  x
 6  x  x  x  x  x  x  x  x  x  x  x  x
 7  x  x  x  x  x  x  x  x  x  x  x  x
 8  x  x  x  x  x  x  x  x  x  x  x  x
 9  x  x  x  x  x  x  x  x  x  x  x  x
```

```
What next?
Options: (U)ncover r c,  (F)lag r c, (Q)uit
u 6 4
    0  1  2  3  4  5  6  7  8  9 10 11
 0  x  x  x  x  x  x  x  x  x  x  x  x
 1  x  x  x  x  x  x  x  x  x  x  x  x
 2  x  x  x  x  x  x  x  x  x  x  x  x
 3  x  x  x  x  x  x  x  x  x  x  x  x
 4  x  x  x  x  x  x  x  x  x  x  x  x
 5  x  x  x  x  x  x  x  x  x  x  x  x
 6  x  x  x  x  3  x  x  x  x  x  x  x
 7  x  x  x  x  x  x  x  x  x  x  x  x
 8  x  x  x  x  x  x  x  x  x  x  x  x
 9  x  x  x  x  x  x  x  x  x  x  x  x
```

The board to start (mines indicated in red)

```
    0  1  2  3  4  5  6  7  8  9 10 11
 0  x  x  x  x  x  x  x  x  x  x  x  x
 1  x  x  x  x  x  x  x  x  x  x  x  x
 2  x  x  x  x  x  x  x  x  x  x  x  x
 3  x  x  x  x  x  x  x  x  x  x  x  x
 4  x  x  x  x  x  x  x  x  x  x  x  x
 5  x  x  x  x  x  x  x  x  x  x  x  x
 6  x  x  x  x  x  x  x  x  x  x  x  x
 7  x  x  x  x  x  x  x  x  x  x  x  x
 8  x  x  x  x  x  x  x  x  x  x  x  x
 9  x  x  x  x  x  x  x  x  x  x  x  x

What next?
Options: (U)ncover r c,  (F)lag r c, (Q)uit
u 7 9
    0  1  2  3  4  5  6  7  8  9 10 11
 0  x  x  x  x  x  x  x  x  x  x  x  x
 1  x  x  x  x  x  x  x  x  x  x  x  x
 2  x  x  x  x  x  x  *  x  x  x  *  x
 3  x  x  x  x  x  x  x  x  x  x  x  x
 4  x  x  x  x  *  x  x  x  x  x  *  x
 5  x  x  x  x  *  x  x  x  x  x  x  x
 6  x  x  x  x  x  *  x  x  x  x  x  x
 7  x  x  x  *  x  x  x  x  x  *  x  x
 8  x  x  x  x  *  x  x  x  x  x  x  x
 9  x  *  x  x  x  x  x  x  x  x  x  x
Better luck next time!
```

The board to start (mines indicated in red)

```
    0  1  2  3  4  5  6  7  8  9 10 11
 0  x  x  x  x  x  x  x  x  x  x  x  x
 1  x  x  x  x  x  x  x  x  x  x  x  x
 2  x  x  x  x  x  x  x  x  x  x  x  x
 3  x  x  x  x  x  x  x  x  x  x  x  x
 4  x  x  x  x  x  x  x  x  x  x  x  x
 5  x  x  x  x  x  x  x  x  x  x  x  x
 6  x  x  x  x  x  x  x  x  x  x  x  x
 7  x  x  x  x  x  x  x  x  x  x  x  x
 8  x  x  x  x  x  x  x  x  x  x  x  x
 9  x  x  x  x  x  x  x  x  x  x  x  x
```

```
What next?
Options: (U)ncover r c,  (F)lag r c, (Q)uit
f 1 1
    0  1  2  3  4  5  6  7  8  9 10 11
 0  x  x  x  x  x  x  x  x  x  x  x  x
 1  x  f  x  x  x  x  x  x  x  x  x  x
 2  x  x  x  x  x  x  x  x  x  x  x  x
 3  x  x  x  x  x  x  x  x  x  x  x  x
 4  x  x  x  x  x  x  x  x  x  x  x  x
 5  x  x  x  x  x  x  x  x  x  x  x  x
 6  x  x  x  x  x  x  x  x  x  x  x  x
 7  x  x  x  x  x  x  x  x  x  x  x  x
 8  x  x  x  x  x  x  x  x  x  x  x  x
 9  x  x  x  x  x  x  x  x  x  x  x  x
```