Program 3: Maze **Due date:** Tuesday, *03/17/20 @ 11:59pm*

(Maze Traversal Using Recursive Backtracking) The grid of #s and dots (.) in Figure 1 is a two-dimensional array representation of a maze. The #s represent the walls of the maze, and the dots represent locations in the possible paths through the maze. A move can be made only to a location in the array that contains a dot.

```
# # # # # # # # # # # #
# . . . # . . . . . . #
. . # . # . # # # # . #
# # # . # . . . . # . #
# . . . . # # # . # . .
# # # # . # . # . # . #
# . . # . # . # . # . #
# # . # . # . # . # . #
# . . . . . . . . # . #
# # # # # # . # # # . #
# . . . . . . # . . . #
# # # # # # # # # # # #
```

Figure 1. Two-dimensional array representation of a maze.

Write a recursive method (`mazeTraversal`) to walk through mazes like the one in the Figure. The method should receive as arguments a 12-by-12 character array representing the maze and the current location in the maze (the first time this method is called, the current location should be the entry point of the maze). As `mazeTraversal` attempts to locate the exit, it should place the character x in each square in the path. There's a simple algorithm for walking through a maze that guarantees finding the exit (assuming there's an exit). If there's no exit, you'll arrive at the starting location again. The algorithm is as follows: From the current location in the maze, try to move one space in any of the possible directions (down, right, up or left). If it's possible to move in at least one direction, call `mazeTraversal` recursively, passing the new spot on the maze as the current spot. If it's not possible to go in any direction, "back up" to a previous location in the maze and try a new direction for that location (this is an example of recursive backtracking). Program the method to display the maze after each move so the user can watch as the maze is solved. The final output of the maze should display only the path needed to solve the maze—if going in a particular direction results in a dead end, the x's going in that direction should not be displayed. [Hint: To display only the final path, it may be helpful to mark off spots that result in a dead end with another character (such as '0').]

Use the java files provided by the instructor. The java file "Maze.java" contains a skeleton class definition of *Maze*, use this skeleton to complete the `mazeTraversal` method.

IMPORTANT

Follow the various style conventions we've discussed in class (variable naming, constants, spaces, etc) including putting comments in your program.

1. The program must compile without errors.
2. Your program must have the following comments at the top. Don't forget to include them because they will count toward the grade of this lab.

```
/*************************************************************
 * First and Last Name   CSCI 6305.01 Spring 2020    Program 3: Maze          *
  ************************************************************/
```

3. Rename your classes and files as Maze_lastnameFirstname.java and MazeTest_lastnameFirstname.java
4. If you want to submit a solution for the extra credit then name your files as Maze_extra_lastnameFirstname.java and MazeTest_extra_lastnameFirstname.java
5. When done, submit your files via blackboard.