Module 8 Assignment (30 points)

Consider a maze made up of rectangular array of squares, such as the following one:

```
X X X X X X X X X X X X X
      X       X X X X X
X X       X         X   X
X X   X X   X   X   X   X
X X     X       X     X
X X X X X X X
X X X X X X X X X X X X X
```

Figure 1--- A sample maze

The “X” blocks represent a blocked square and form the walls of the maze. Let’s consider mazes that have only one entrance and one exit, as in our example. Beginning at the entrance at the top left side of the maze, fund a path to the exit at the bottom right side. You can move only up, down, left, or right. Square is either clear or blocked.

Let a two dimensional array represent the maze. You should write two alternative methods to find a path through the maze. One of the methods should find the path recursively while the other method is non-recursive and must use a stack data structure to find a path. Some mazes might have more than one successful path, while others might have no path.

Hints:

- For the non-recursive implementation, the primary consideration is that if you reach a dead end, you need to be able to backtrack along the path to the point where you can try a different path. The stack data structure makes it possible to store the current path and backtrack if necessary. Every clear position visited in the current path is pushed to the stack and if a dead end is reached, the previous position is popped from the stack to backtrack. You need to have a loop that begins with the start position and pushes it into the stack then moves to a neighboring cell until either the goal position is reached or the stack becomes empty.

- Make sure to mark each clear array cell you move to as ”visited” to avoid checking it again and getting stuck in an infinite loop. For example, each array cell can have three different values: “X” for blocked, “V” for visited, and “ “ for clear. A dead end is an array cell whose all neighbors are either blocked or already visited.

- The recursive method is pretty straightforward, you can call the method recursively on the left, right, up, or down neighbor (if they are clear) and return the first solution you find. Similar to the iterative solution, you need to mark each cell you visit as “v” and avoid calling the recursive method on a neighbor that is already visited. You can consider the dead-end and reaching the goal as stopping criteria.

What you need to do:

I have attached three files with this assignment spec:

- **Position.java:** a class to store the position of an array cell. It has two attributes: row and column along with their accessor and mutator methods. It also has an equals method that checks for equality of two Position objects.
- **Maze.**java: a class to store and traverse a maze. It has one attribute: a two dimensional character array which represents the maze. It also has two methods “traversewithStack” which finds a solution to the maze using stack and “traverseRecursive” which finds the solution recursively.
- **MazeTester:** a sample driver class which calls the traverse method on the maze shown in figure 1. To ensure that your program works correctly for all mazes, try to test your program for at least a couple more mazes in addition to the one in figure 1.

Your job is to implement the two methods : “traverseWithStack” and “traverseRecursive” in the Maze.java class. Both of these methods receive the start and goal positions as parameters and return an array of Position which stores a solution to the maze if such solution exists; otherwise, they return null.

Note: There might be more than one possible solution to a maze, depending on the ordering in which the neighboring cells are visited at each location. **Please follow the following order in visiting the neighbor positions:**

First move left, if the left neighbor is not clear, then move right, if the right neighbor is not clear, then move up, if it is not clear, then move down.

To get a full credit, it is very important that you comply with this ordering rules, so that your method produces a solution that matches the correct solution in my test harness

You also need to validate the parameters passed to the traverse method. For example, the start and end positions should be valid array positions and contain blank. You also need to check for a null or an empty array (array with all null elements).

What you need to turn in:

You need to submit the Maze.java file containing your implementation of traversewithStack and traverseRecursive methods.

Grading Rubric:

TraverseRecursive returns a correct solution to all test cases	5
The recursive calls and stopping criteria are correct5	5
The traverseWithStack returns a correct solution to all test cases	15
Stack data structure is appropriately used to simulate recursion	5
Total	30