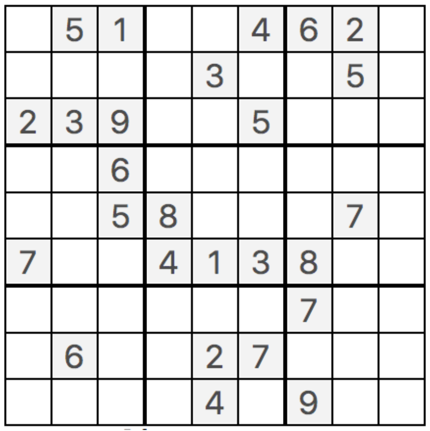
SudokuSolver.java

Objective: To solve a Sudoku puzzle using two-dimensional arrays and recursion.

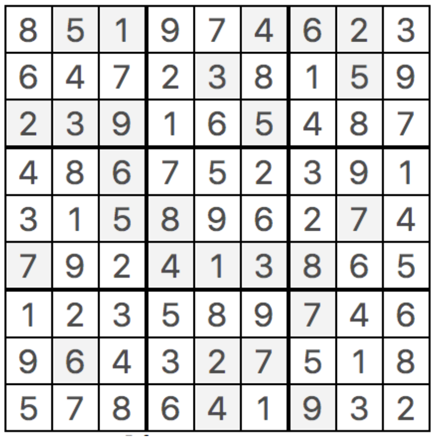
Background:
In your **SudokuMaker** program you generated complete, correct Sudoku puzzles. In the real world, most Sudoku puzzles have random grids blanked out so you can only view it partially completed. For example, you may get one like that in figure 1 below.



	5	1			4	6	2	
				3			5	
2	3	9			5			
		6						
		5	8				7	
7			4	1	3	8		
						7		
	6			2	7			
				4		9		

Fig. 1 - Sudoku puzzle

There are numerous strategies for filling in a puzzle. In the end, all of the blanks are replaced with numbers 1 through 9 such that each column, row, and 3x3 grid have no duplicate numbers. (See figure 2 below)



8	5	1	9	7	4	6	2	3
6	4	7	2	3	8	1	5	9
2	3	9	1	6	5	4	8	7
4	8	6	7	5	2	3	9	1
3	1	5	8	9	6	2	7	4
7	9	2	4	1	3	8	6	5
1	2	3	5	8	9	7	4	6
9	6	4	3	2	7	5	1	8
5	7	8	6	4	1	9	3	2

Fig. 2 - Completed Sudoku puzzle

In **SudokuSolver**, you will input a partially completed (valid) puzzle and generate a completed solution. Luckily for us, the leap from **SudokuMaker** to **SudokuSolver** is very small. Much of the code used in the former can be used in the latter. Even better, the algorithm for solving a puzzle is similar in most ways to creating a puzzle.

Algorithm:
You will implement a "brute-force" method for completing a partially solved Sudoku puzzle. Like **SudokuMaker**, **SudokuSolver** uses recursion and backtracking. The idea is to start in the upper-left corner and move row-major order searching for "empty" spaces (spaces with 0's). You leave the "filled" spaces alone (having 1 to 9 already in it). When you find an empty space, you try a random number in that spot and recursively solve the rest of the puzzle. If all numbers 1 through 9 fail for that space, then the program discards this partial solution and rolls back to the previous empty grid

location to try another random number. Eventually, the program will fill all the empty spaces and terminates with a solution.

Just as before, you will use a two-dimensional array of integers for the puzzle. All of the fields, constructor, and methods for loading the partially completed puzzle are included in Mr Greenstein's files. All you have to complete is the **solvePuzzle** method along with any "helper methods." The pseudocode for the algorithm has been left out. The algorithm is so close to the **SudokuMaker** pseudocode that you should be able to modify the algorithm on your own.

Assignment:

1. Download **SudokuSolver.zip** from Mr Greenstein's web site and unzip. It will create a **SudokuSolver** directory to do your work. It contains **SudokuSolver.java** and two puzzle files.

2. In **SudokuSolver.java**, complete the `solvePuzzle()` method. Do not change the any of the other methods since their operation will be used for grading.

3. Test your code using the **puzzle1.txt** and **puzzle2.txt** files.

4. Create your own puzzle file to test your code. You can generate a puzzle using your **SudokuMaker** program, then blank out values.

Note: Some partial puzzles may have more than one complete, valid solution. In grading, any valid solution will be accepted.

When completed, the program will print something like the following output.

```
% java SudokuSolver puzzle1.txt

Original Sudoku puzzle

  +-----------+-----------+-----------+
  |  4        |     7  8  |        3  |
  |        9  |  4     2  |  1  7     |
  |     3     |  5        |        4  |
  +-----------+-----------+-----------+
  |  7        |     2  5  |           |
  |     2     |           |  4  3     |
  |           |     4     |  2  5     |
  +-----------+-----------+-----------+
  |  1     4  |  2     3  |        9  |
  |     5     |           |           |
  |     8     |  1  5     |     4     |
  +-----------+-----------+-----------+

Solved Sudoku puzzle

  +-----------+-----------+-----------+
  |  4  1  5  |  9  7  8  |  6  2  3  |
  |  8  6  9  |  4  3  2  |  1  7  5  |
  |  2  3  7  |  5  1  6  |  8  9  4  |
  +-----------+-----------+-----------+
  |  7  4  1  |  3  2  5  |  9  6  8  |
  |  5  2  6  |  7  8  9  |  4  3  1  |
  |  3  9  8  |  6  4  1  |  2  5  7  |
  +-----------+-----------+-----------+
  |  1  7  4  |  2  6  3  |  5  8  9  |
  |  6  5  3  |  8  9  4  |  7  1  2  |
  |  9  8  2  |  1  5  7  |  3  4  6  |
  +-----------+-----------+-----------+
```