Sudoku Solver (Backtracking 2)

Java

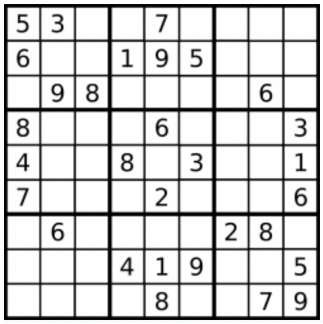
Write a program to solve a Sudoku puzzle by filling the empty cells.

A sudoku solution must satisfy all of the following rules:

1. Each of the digits 1-9 must occur exactly once in each row.
2. Each of the digits 1-9 must occur exactly once in each column.
3. Each of the digits 1-9 must occur exactly once in each of the 9 3x3 sub-boxes of the grid.

The '.' character indicates empty cells.

Sample Input



```
board =
[["5","3",".",".","7",".",".",".",".",".],["6",".",".","1","9","5",".",".",".",".],["
.","9","8",".",".",".",".",".","6",".],["8",".",".",".",".","6",".",".",".","3"],["4",
".",".","8",".","3",".",".",".","1"],["7",".",".",".",".","2",".",".",".",".","6"],[".","6"
,".",".",".",".",".","2","8","."],[".",".",".","4","1","9",".",".",".","5"],[".",".",".
",".","8",".",".",".","7","9"]]
```

Sample Output

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

```
[["5","3","4","6","7","8","9","1","2"],["6","7","2","1","9","5","3","4","8"],["
1","9","8","3","4","2","5","6","7"],["8","5","9","7","6","1","4","2","3"],["4",
"2","6","8","5","3","7","9","1"],["7","1","3","9","2","4","8","5","6"],["9","6"
,"1","5","3","7","2","8","4"],["2","8","7","4","1","9","6","3","5"],["3","4","5
","2","8","6","1","7","9"]]
```

Code

a. StartingRow = 3*(row/3) & StartingCol = 3*(col/3)
b. StartingRow = row - row%3 & StartingCol = col - col%3

Code

```
class Solution {
   public boolean isSafe(char[][] board, int row, int col, int number) {
       //column
       for(int i=0; i<board.length; i++) {
           if(board[i][col] == (char)(number+'0')) {
               return false;
           }
       }

       //row
       for(int j=0; j<board.length; j++) {
           if(board[row][j] == (char)(number+'0')) {
               return false;
           }
       }

       //grid
       int sr = 3 * (row/3);
       int sc = 3 * (col/3);

       for(int i=sr; i<sr+3; i++) {
           for(int j=sc; j<sc+3; j++) {
               if(board[i][j] == (char)(number+'0')) {
                   return false;
               }
           }
       }
```

```
        return true;
    }

    public boolean helper(char[][] board, int row, int col) {
        if(row == board.length) {
            return true;
        }

        int nrow = 0;
        int ncol = 0;

        if(col == board.length-1) {
            nrow = row + 1;
            ncol = 0;
        } else {
            nrow = row;
            ncol = col + 1;
        }

        if(board[row][col] != '.') {
            if(helper(board, nrow, ncol)) {
                return true;
            }
        } else {

            //fill the place
            for(int i=1; i<=9; i++) {
                if(isSafe(board, row, col, i)) {
                    board[row][col] = (char)(i+'0');
                    if(helper(board, nrow, ncol))
                        return true;
                    else
                         board[row][col] = '.';
                }
            }
        }

        return false;
    }

    public void solveSudoku(char[][] board) {
        helper(board, 0, 0);
    }
}
```