Computer Science 282
Programming Assignment #1

You are to write a program that will solve sudoku puzzles. To learn about sudoku, you can visit the wikipedia page here: http://en.wikipedia.org/wiki/Sudoku. All you really need to read, though, is the first paragraph where the rules are explained and then look at the example on the right..

You must implement the two classes and the methods shown in the code fragments below and use them to solve the sudoku problems. Do not make any changes to the classes, variables, names, etc. Just add code. Also, no backtracking or recursion should be used. Pay attention to the many hints and additional requirements found in the comments.

This assignment is as much about following instructions as it is about programming. Be sure your methods do exactly what is expected of them. What follows are some more general rules that you must follow for this and, as applicable, all other programs you write in this class. Failure to follow these rules will cost you points.

1. I will eventually post a final test program. A small sample test program is already posted to help get you started. Don't wait for my final test program to appear to debug your program. You should begin testing with your own test programs. Also, the final test program I post may not be the same test program I run on your programs. You cannot assume your program is correct just because it works on the posted test program.

2. Submit your program through Canvas as a file upload with the name prog1.java. Be sure that my main (test) program is in a different file from your class(es) and you should not submit my test program. Also, do not make your classes "public". If you deviate from any of these rules you will lose points because doing so makes it much more difficult for me to manage and test your programs.

3. The top of your prog1.java file (and all java code that you turn in to me in the future) should begin with comments that show: (i) Your name; (ii) Comp 282 and the time you class "meets"; (iii) The assignment number; (iv) The date the assignment was handed in (which is not necessarily the same as the due date); (v) A brief description of what is contained in the file.

4. For all Java code you turn in be sure to:
   a) Choose clear and suggestive variable names.
   b) No part of your printed output should have lines wrapped to the next line or disappearing off the end of the page.
   c) Use comments generously to describe how your methods work. Comments should appear inside your methods, too – not just at the top. That said, don't overdo the comments.
   d) There should be no more than one *return* statement in any method.
   e) Do not use any *break* statements to exit loops.
   f) Do not use global variables. If you are not sure if you are using them, ask.
   g) Reminder: Programs should be 100% your own work, as stated in the syllabus.

```
// One square in the Sudoku grid

class Spot {
    private int row, col;

    public Spot(int row, int col) {    }

    public void setRow(int row) {    };

    public void setCol(int col) {    };

    public int getRow() {    };
```

```
    public int getCol() {    };
}


class Sudoku {

    // The Sudoku grid
    //    0 corresponds to an empty spot
    //    other values correspond to solution values

    private int grid[][];


    // default constructor -- I never seem to use it....

    public Sudoku() {    }


//    Construct a Sudoku instance from a 2-d array of int's

//    public sudoku(int data[][]) {
//        board = new int[9][9];
//        for (int row = 0; row < 9; row++)
//            for (int col = 0; col < 9; col++)
//                board[row][col] = data[row][col];
//    }


    // Construct a Sudoku puzzle from an array of strings
    // There is one string per row and each row has a space
    //    between each group of 3 int's
    // ***  Looking at the data at the top of the test program will make this clear  ***

    // charAt is a String method that returns a char from a string
    //     This is similar to using [] to access an element in an array

    // if ch is a char with a value between '0' and '9' then
    //    (int) ch - 48   will cast it to the actual int value

    // Because of the 2 extra spaces in each row string we need to be a little tricky
    //    extracting the int's from those strings

    // This is the code that extracts the int from the string:
    //         (int) (s[row].charAt(col + col/3)) - 48

    public Sudoku(String s[]) {    }


    // Copy constructor

    // Creates a new grid for a new Sudoku problem.
    // Be sure to create a new grid. Do not just point to p's grid.

    public Sudoku(Sudoku p) {    }


    // Hint: use String.valueOf(i) to convert an int to a String

    // Your goal is to make your output look exactly like mine.

    public String toString() {    }


    // This is for me to use to make it easier to check your answers

    public String toString2() {
        String result = new String();
        for (int row = 0; row < 9; row++) {
            for (int col = 0; col < 9; col++) {
```

```
            result = result + String.valueOf(grid[row][col]);
        }
    }
    return result;
}


// Rotate the sudoku puzzle. Again, this is for me to use in my test program.

public void rotate() {
    int[][] temp = new int[9][9];
    int row, col;
    for (row = 0; row < 9; row++) {
        for (col = 0; col < 9; col++) {
            temp[col][8-row] = grid[row][col];
        }
    }
    for (row = 0; row < 9; row++) {
        for (col = 0; col < 9; col++) {
            grid[row][col] = temp[row][col];
        }
    }
}


// Does the current grid satisfy all the Sudoku rules?
// In other words, checks to see that no row, col, or box contains some
//    value more than once.

public boolean isValid() {    }


// Does val appears in this row of the grid?

private boolean doesRowContain(int row, int val) {    }


// Does val appears in this col of the grid?

private boolean doesColContain(int col, int val) {    }


// Does val appears in this box of the grid?

private boolean doesBoxContain(int rowbox, int colbox, int val) {    }


// Return n if n is the only legal value for this spot
// Return 0 otherwise

private int uniqueSpotValue(Spot sq) {    }


// If there is only one legal spot to put val in this row of the grid
//    then return that Spot
// Otherwise return null

private Spot onlyOneSpotinRowForVal(int row, int val) {    }


// If there is only one legal spot to put val in this col of the grid
//    then return that Spot
// Otherwise return null

private Spot onlyOneSpotinColForVal(int col, int val) {    }
```

```java
    // If there is only one legal spot to put val in this box of the grid
    //    then return that Spot
    // Otherwise return null

    private Spot onlyOneSpotinBoxForVal(int rowInBox, int colInBox, int val) {    }


    // Put as many values as possible into the grid using only the above methods

    // Do this by repeated calling the 4 previous methods until the grid no longer changes

    public void solve() {    }


    // I haven't decided yet if I want to talk about this -- or if I want you
    // to program it. Using this method allows virtually any Sudoku to be solved.
    // What do you think?  :)

    public void solveWithRandom() {    }


    // who are you?  Be sure to put your name here.

    public static String myName() {
        return "Fred Flintstone";
    }
}
```