CMPS 12A
Introduction to Programming
Spring 2010

Programming Assignment 5

Due Sunday May 23, 10:00 pm

This is part one of a two-part programming assignment in which you will write a java program to solve Sudoku puzzles. If you are unfamiliar with Sudoku, see `http://en.wikipedia.org/wiki/Sudoku`, or any of the hundreds of websites devoted to the puzzle. Briefly, Sudoku is a logic puzzle in which one fills a 9×9 grid with numbers in such a way that every row, column, and 3×3 "box" contains each of the numbers 1 through 9 exactly once. An instance of the puzzle consists of a partially filled 9×9 grid, and it is the solver's task to fill in the remaining cells using logical deduction.

A Sudoku Puzzle

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

Solution

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

It has been shown that there are 6,670,903,752,021,072,936,960 distinct solution grids. If one takes into account symmetries in the puzzle such as rotations, reflections, and permutations, then there are just 5,472,730,538 essentially different ways to fill in a grid. Each filled grid however, gives rise to a large number of distinct puzzles by removing the contents of different sets of cells. Notice that if there are not enough filled cells to begin with, then the puzzle solution is not uniquely determined. To be a valid Sudoku puzzle, there can be only one correct way to fill in the empty cells.

In this assignment you will write a program called `SudokuIO.java`, which performs input/output operations and checks to see if the grid is already filled. Your program will read in a 9×9 grid from a file whose name is given on the command line, then print a text representation of that grid to stdout. Your program will also determine if the grid is filled or not, and will print a message with that information to

stdout. Each of the methods you write in this assignment will be re-used in programming assignment 6, which constitutes part two of the project. There you will write a complete Sudoku solver capable of solving many (but not all) Sudoku puzzles. (These are the puzzles sometimes categorized as "easy Sudoku".)

An input file for `SudokuIO.java` will consist of a space separated list of 81 numbers giving the nine rows of the puzzle from left to right, and top to bottom. Extra intervening white space (spaces, tabs, and newlines) will be ignored by your program. In this file, a blank cell will be represented by the number zero. Thus an input file for the above puzzle would look like:

Input file:

```
0 0 0  5 3 0  0 0 4
0 8 3  9 0 1  0 0 0
6 1 0  0 2 8  7 0 0

0 0 0  0 0 0  2 0 7
0 4 0  1 0 7  0 8 0
7 0 9  0 0 0  0 0 0

0 0 1  7 4 0  0 5 8
0 0 0  6 0 3  9 7 0
2 0 0  0 9 5  0 0 0
```

Notice that extra lines and spaces may be included in order to make the "box" structures more visible, but this is not part of the file format. A single line of 81 numbers would look the same to your program. You are not required to check the validity of the input file in any way, and you may therefore assume that your program will be tested only on files representing valid Sudoku puzzles.

Program output will be formatted in a similar manner, except that blank cells will be represented by dashes "`-`" rather than zeros. Extra lines and spaces will be included to reveal the box structure. Thus the corresponding output (printed to stdout) will be:

Output:

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

- - -  - - -  2 - 7
- 4 -  1 - 7  - 8 -
7 - 9  - - -  - - -

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

This puzzle is not filled
```

Your program will print out one blank line after the last line of the grid, followed by a one-line message stating whether or not the grid is filled, as above. If the input file contains no zeros, and therefore represents a filled grid, then the message would of course be "`This puzzle is filled`".

Your program will implement the puzzle grid as a 2-dimensional `int` array. It is recommended (though not required) that this array be of size 10×10, so if you call the array `grid` say, you can ignore index zero, and thus refer the seventh row, fifth column of the puzzle as `grid[7][5]` rather than `grid[6][4]`. A small amount of memory is lost in doing this, but it's probably worth the gain in clarity.

Required Features

You will include a `Makefile` with this project that creates an executable jar file called `SudokuIO`. See lab assignment 5 for instructions on how to do this. Your Makefile will include a phony target called `clean` that removes `SudokuIO.class`, and another target called `spotless` that removes both `SudokuIO.class` and the jar file `SudokuIO` itself. Your Makefile need not contain a submit utility, as was required in lab5. Read the Makefiles included with the sequence of "Person" examples on the webpage as a guide.

If too many or too few command line arguments are given, your program will respond with a usage message, then quit.

```
% SudokuIO
Usage: SudokuIO [InputFile]

% SudokuIO xyz abc
Usage: SudokuIO [InputFile]
```

If the file specified on the command line does not exist or cannot be opened, your program will print the system message associated with `FileNotFoundException`, print the same usage message as before, then quit.

```
% SudokuIO xyz
xyz (The system cannot find the file specified)
Usage: SudokuIO [InputFile]
```

All output in the above examples will be printed to stderr. Read the example `ReadIntegers.java` on the class website for hints on how to accomplish all this. Your program will define and use static methods with the following names and signatures:

```
static void usage()
static void getGrid(int[][] G, Scanner sc)
static void printGrid(int[][] G)
static boolean isFilled(int[][] G)
```

Method `usage()` will print a usage message to stderr, as above, then exit the program. Method `getGrid()` will extract integers from the `Scanner` object `sc` and place them in the `int` array `G`, row by row. A typical call to `getGrid()` from within `main()` might look like

```
getGrid(grid, sc);
```

This assumes that `grid` is a 2-dimensional array allocated to be of the proper size, and `sc` is a `Scanner` object pointing to the input file. You may of course give these function arguments any names you wish. Method `printGrid()` will print the contents of its array argument to stdout in the format described above. In particular, zeros will be printed as dashes, and the box structure will be apparent from the

spacing. Method `isFilled()` will return `true` if all entries in its array argument are non-zero, and `false` otherwise.

Remarks

Again, each of these methods will be used in pa6, along with some other methods that implement a specific Sudoku strategy. Your source file for this project will be called `SudokuIO.java`, and should be submitted, along with your `Makefile`, to the assignment name `pa5`.