DM550/DM857
Introduction to Programming
Fall 2017 Re-exam Project (Java)

Department of Mathematics and Computer Science
University of Southern Denmark

February 23, 2018

Introduction
The purpose of this project is to try in practice the use of programming techniques and knowledge about the programming language Java. Please make sure to read this entire note before starting your work on this project. Pay close attention to the three sections sections below.

Exam Rules
This project is part of the re-exam for DM550 and DM857. To pass the re-exam a Python project (from the re-exam or from the ordinary exam) and this Java project (or the one from the ordinary exam) have to be passed in order to pass the overall exam. This Java project has to done individually.

Deliverables
A project report (at least 6 pages without front page and appendix) containing the following 6 sections has to be delivered:

- **front page** (course number, name, section, date of birth)
- **specification** (what the program is supposed to do)
- **design** (how the program was planned)
- **implementation** (how the program was written)
- **testing** (what tests you performed)
- **conclusion** (how satisfying the result is)

The report has to be delivered as a PDF file (suffix .pdf) together with the Java source code files (suffix .java) and any needed supporting data files electronically using Blackboard's SDU Assignment functionality. No printed copies are required. **Do not forget to include the complete source code!**

Deadlines

March, 23, 2018, 23:59

Late deliveries cannot be accepted, so please plan your time accordingly.

The Problem

Your task in this part of the project is to write a solver for Sudoku puzzles. Sudoku puzzles are a kind of crossword puzzles with numbers where the following two conditions have to be met:

- In each row or column, the nine numbers have to be from the set $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and they must all be different.
- For each of the nine non-overlapping 3x3 blocks, the nine numbers have to be from the set $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and they must all be different.

The following two figures show a Sudoku puzzle and its solution.

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

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

The Input

For input to your program, the Sudoku puzzles are represented as text files. The cells in these two-dimensional "matrices" can be one of the following two types:

- A letter `x` signifies an empty cell, i.e., a cell that your solver needs to find a number for.
- A numeric term signifies a cell whose value is fixed, i.e., the corresponding cell needs to have this number in the solution that you find.

Thus, for our example above we obtain the following text file:

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

The home page of the course contains a number of possible inputs to test your program on.

The Output

The output of your solver should be a corresponding number of lines. The representation is similar to the one for the input except for all `x`s being replaced by the appropriate numbers.

Thus, for our example above we obtain the following output:

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

The Task

Implement a class `Sudoku` that represents a Sudoku puzzle. Write methods for reading in a Sudoku puzzle from a text file and for converting a solved Sudoku puzzle into a text file. The implement a method `solve` that fills in the missing numbers.

Keep in mind, that there are many different ways how to implement such a `solve` method. Explain your approach in the design section of your report. Then implement and test it.

You could for example choose one of the two following approaches:

- Use a generate-and-test approach, i.e., enumerate solution candidates and test them until you find a solution. This can be done recursively or iteratively.
- Implement solving rules as used by human players.

Example Output

After reading in the text file described above, printing a string representation of the Sudoku instance should output the following:

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

After using `solve`, the string representation of the Sudoku instance should be as above:

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