© Manfred Kerber
Alexandros Evangelidis

School of Computer Science
University of Birmingham

# Worksheet 5

## MSc/ICY Software Workshop

Assessed Exercise: 9% of the continuous assessment mark.

### Submission: Thursday 1 December 2016 2pm

**5% late submission penalty within the first 24 hours. No submission after 24 hours.**

**JavaDoc comments are mandatory. Follow the submission guidelines on**

`https://canvas.bham.ac.uk/files/3242907`**.**

Assessment via viva. In your submission write all classes in a single directory, called WS1-5, from which you create your zip file.

**Exercise 1: (Basic, 20%)** A Sudoku is a $9 \times 9$ quadratic structure with some numbers given in which the digits from 1 through 9 have to be filled so that in every row, in every column, and in 9 sub-squares of size $3 \times 3$ each digit occurs exactly once. For more details, see `https://en.wikipedia.org/wiki/Sudoku`.

Let a fully solved Sudoku (of size $9 \times 9$) be given as a two-dimensional array of numbers between 1 and 9 so that each number occurs exactly once in each row, each column, and each of the 9 sub-squares. For instance, the following is a fully solved Sudoku:

```
++===+===+===++===+===+===++===+===+===++
|| 1 | 2 | 3 || 4 | 5 | 6 || 7 | 8 | 9 ||  0
++---+---+---++---+---+---++---+---+---++
|| 4 | 5 | 6 || 7 | 8 | 9 || 1 | 2 | 3 ||  1
++---+---+---++---+---+---++---+---+---++
|| 7 | 8 | 9 || 1 | 2 | 3 || 4 | 5 | 6 ||  2
++===+===+===++===+===+===++===+===+===++
|| 2 | 3 | 4 || 5 | 6 | 7 || 8 | 9 | 1 ||  3
++---+---+---++---+---+---++---+---+---++
|| 5 | 6 | 7 || 8 | 9 | 1 || 2 | 3 | 4 ||  4
++---+---+---++---+---+---++---+---+---++
|| 8 | 9 | 1 || 2 | 3 | 4 || 5 | 6 | 7 ||  5
++===+===+===++===+===+===++===+===+===++
|| 3 | 4 | 5 || 6 | 7 | 8 || 9 | 1 | 2 ||  6
++---+---+---++---+---+---++---+---+---++
|| 6 | 7 | 8 || 9 | 1 | 2 || 3 | 4 | 5 ||  7
++---+---+---++---+---+---++---+---+---++
|| 9 | 1 | 2 || 3 | 4 | 5 || 6 | 7 | 8 ||      8
++===+===+===++===+===+===++===+===+===++
```

Create a class Sudoku with a single field variable `private int[][] array;`. For a fully filled Sudoku the values are all between 1 and 9 inclusively. For a partially filled Sudoku the unfilled fields should be represented by `0`.

Write the class with a constructor and a getter. Furthermore write a `public String toString()` method that prints a Sudoku in the format provided. Unfilled field should be left empty rather than being filled with `0`.

Write also a method `public boolean isFilled()`, which checks whether all elements are filled in a Sudoku.

**Exercise 2: (Basic, 20%)** Write in a class `SudokuRead` a method to read in a potentially only partially filled Sudoku (given as below) from a file. That is, write a `public Sudoku readSudoku(String fileName) throws IllegalArgumentException, IOException` method. Your method should throw exceptions: if the file is not accessible; and if there is an input that is not a number between 1 and 9 (inclusively) or an empty space, or if the input is incomplete. Note that in each line the first 9 characters must be correct, but that the line may contain trailing input of any kind (typically further empty spaces). Likewise in the whole file the first 9 lines must be correct, but there may be trailing lines (typically further empty lines).

```
 234567 9
45678912
78912345
234 67891
567891 34
8912 4567
34 678912
6 8912345
 12345678
```

**Exercise 3: (Medium, 20%)** Write in a `SudokuCheck` class a method that checks whether a given Sudoku of size $9 \times 9$ satisfies the conditions. Concretely, write a method `public static boolean[][] check(Sudoku sudoku)` that returns a two-dimensional `result` array of `boolean` values such that `result[0]` is a one-dimensional array of boolean values indicating for each of the rows 0 through 8 whether it satisfies the condition that each of the values 1 through 9 occurs exactly once in it or not. Likewise `result[1]` states this for the columns, and `result[2]` for the nine bigger squares. List the rows top-down, the columns left-right, and the $3 \times 3$ squares as follows:

| | | |
|---|---|---|
| 0 | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |

For the example from Exercise 1 all 27 values in the `result` array are `true`.
If you changed the 8 in the lower right corner from an 8 to a 9, e.g., the values of `result[0][8]`, `result[1][8]`, and `result[2][8]` would all be false, all the other entries in the `result` array would be true.
If, however you changed in the original Sudoku the lower left corner from a 9 to an 8, the values of `result[0][8]`, `result[1][0]`, and `result[2][6]` would all be false, and all the other entries in the `result` array would be true.

**Exercise 4: (Medium, 20%)** Write a sub-class `SudokuInteractive` of the `Sudoku` class which allows users to play a Sudoku interactively. In particular write a method `public static void play(String file)`, which reads in a Sudoku represented in the `file` in the form given in Exercise 2. The user should be able to enter from the command line their input by something like `d4:5` in order to indicate that in field `d4` should go a `5`. If that field is part of the originally given Sudoku the action should be ignored; if, however, the user wants this way to update a previously entered value, then the current value in the Sudoku should be overwritten. Except for these types of input only two other inputs should be accepted: first, `reset` to clear all the fields that the user has filled in, that is, the game is reset to the start situation; second, `exit` to terminate the game.

Make sure that your program does not allow for any other input and that all exceptions are caught appropriately.
Override the `public String toString()` method so that the user sees whether a field is given in the original Sudoku, or filled in by them, e.g. by displaying the Sudoku by something such as the following, where originally given fields such as `a2` are displayed by something like `*2*`, whereas user filled fields such as `a1` by something like `1` .

```
      1   2   3    4   5   6    7   8   9
   ++===+===+===++===+===+===++===+===+===++
a  || 1 |*2*|*3*||*4*|*5*|*6*||*7*|   |*9*||
   ++---+---+---++---+---+---++---+---+---++
b  ||*4*|*5*|*6*||*7*|*8*|*9*||*1*|*2*|   ||
   ++---+---+---++---+---+---++---+---+---++
c  ||*7*|*8*|*9*||*1*|*2*|*3*||*4*|*5*| 6 ||
   ++===+===+===++===+===+===++===+===+===++
d  ||*2*|*3*|*4*||   |*6*|*7*||*8*|*9*|*1*||
   ++---+---+---++---+---+---++---+---+---++
e  ||*5*|*6*|*7*||*8*|*9*|*1*||   |*3*|*4*||
   ++---+---+---++---+---+---++---+---+---++
f  ||*8*|*9*|*1*||*2*|   |*4*||*5*|*6*|*7*||
   ++===+===+===++===+===+===++===+===+===++
g  ||*3*|*4*|   ||*6*|*7*|*8*||*9*|*1*|*2*||
   ++---+---+---++---+---+---++---+---+---++
h  ||*6*|   |*8*||*9*|*1*|*2*||*3*|*4*|*5*||
   ++---+---+---++---+---+---++---+---+---++
i  ||   |*1*|*2*||*3*|*4*|*5*||*6*|*7*|*8*||
   ++===+===+===++===+===+===++===+===+===++
```

**Exercise 5: (Advanced, 20%)** Write a class `SudokuSolve` with a method `public void solve()` that can solve easy Sudokus. Essentially there are two steps that your solver should apply:

- The fact that every number may occur only once in each row, column, or big square restricts the (a priori nine) possibilities for each single field. If there is only one possibility left for a particular field the solver enters that possibility.
- For a particular number check whether in a particular row/column/big square it has only one single place where it can go. If there is only one place the solver enters the number there.

After each action there may be a knock-on effect on other fields. These are acted upon until there are no more changes. Easy Sudokus can be solved this way. For difficult Sudokus other rules may be necessary, or a completely different approach can be taken such as trying out values in a systematic way. All this is of no concern for this exercise, however.