Informatics 1: Object Oriented Programming

Tutorial 03

Week 4: 04/02 - 08/02

Volker Seeker (volker.seeker@ed.ac.uk)
Naums Mogers (naums.mogers@ed.ac.uk)

1 Introduction

In this tutorial you will need to write some more **functions** and **iterate through arrays** to create a way of validating the correctness of numbers put into the Sudoku by the user.

You should extend the solution to your previous tutorial exercise with the new content introduced here. Alternatively, you can extend the sample solution provided for the previous tutorial.

As a good rule of thumb, three things are necessary to improve a skill:

1. Regular practice
2. Exercises outside your comfort zone
3. Feedback on your progress

Therefore, please finish these exercises at home as best you can and send the solutions to your tutors before the tutorial so they can give you specific feedback. Even though a solution is working correctly, it is not necessarily good code. Your tutor will be able to point those things out to you.

2 Exercises

In a regular Sudoku you are only allowed to place numbers between 1 and 9. We also have zeros but they are simply placeholders for empty fields. We already validate the user's input in this regard by accepting only those numbers as command line input.

There are three additional rules to the game illustrated in Figure 1 which you will implement in the following tasks:

1. A number can only be placed in a field if there is not already the same number in the same row.
2. A number can only be placed in a field if there is not already the same number in the same column.
3. A number can only be placed in a field if there is not already the same number in its corresponding 3x3 subgrid / block.

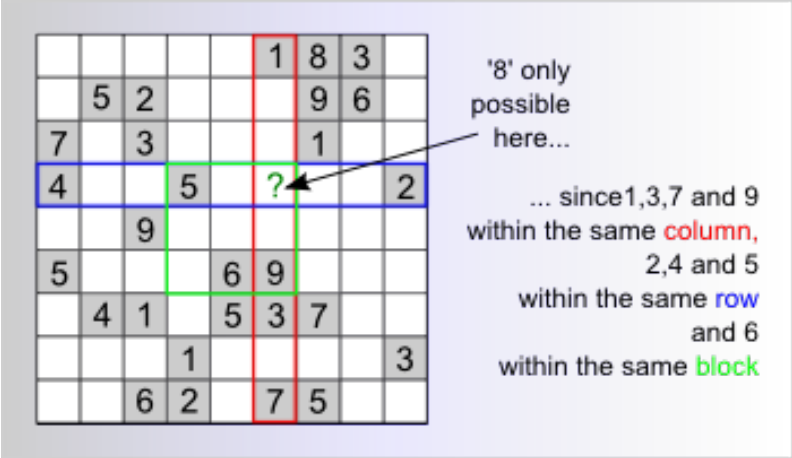

Figure 1: Sudoku rules for placing numbers.
source: http://sudoku.becher-sundstroem.de/help.html

Task 1 - Check the Row

◁ **Task**

Add a `public static` function to your Sudoku `class` called `checkRow` which gets four arguments: an integer x, an integer y, an integer value and a two-dimensional integer array grid. **X** and **y** specify the coordinates in the grid where the given value should be added, **value** is the value intended to be added by the user and **grid** is the Sudoku game grid.

This function should check whether the same value passed in as an argument is already present in the grid's row specified by the given y value. The calculated result should be returned as a `boolean`.

Note that this function should only check if the given value is allowed in the specified row and does not actually set the value into the grid.

Testing your Code This function is not integrated into your actual program yet, i.e. you are not using it anywhere. In order to test it, place it at the beginning of your `main` function after the grid has been initialised and observe the correctness of its output by using `System.out.println`. Task 4 will tell you how it will be integrated at which point you can get rid of your test code. In general, you should always run your code against some test values while you develop it even if you have to temporarily place it in a location where it won't remain. In the future, you can then consider writing Unit tests as a permanent test location for only a particular function.

Next to all this, you should make sure that the provided basic tests run through to validate the correctness of your function headers.

Task 2 - Check the Column

◁ **Task**

Similar to the previous task, you should now add a `public static` function to your Sudoku `class` called `checkColumn` which again gets the same four arguments.

This function should check whether the same value passed in as an argument is already present in the grid's column specified by the given x value. Same as before, the calculated result should be returned as a `boolean`.

Task 3 - Check the Subgrid / Block

◁ **Task**

Now implement a similar function to check the third rule by adding a `public static` function to your Sudoku `class` called `checkSubGrid` which also gets the same four arguments.

Each 9x9 Sudoku is subdivided into nine 3x3 subgrids aka. blocks. This is illustrated in Figure 2. The given **x** and **y** coordinates specify a field in one of those blocks and your function needs to find out if the given **value** is not yet present anywhere else inside this block and return `true` if this is the case. To do that, you need to find out how to iterate through the arrays (calculate start and end of your loop(s)) to only cover the corresponding block.

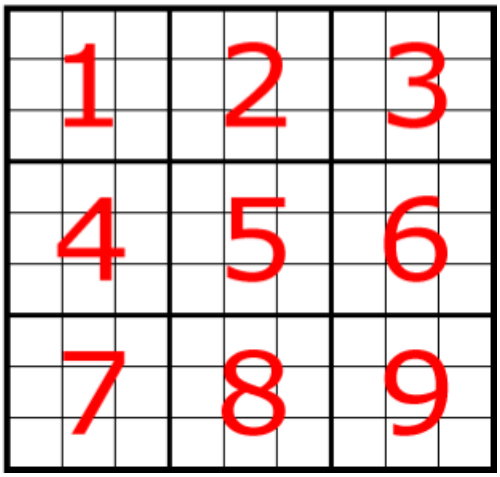

Figure 2: Each 9x9 Sudoku is subdivided into nine blocks.
source: https://www.brainbashers.com/sudokuhelpbasics.asp

Task 1 - Integration

◁ **Task**

As a final task you should make use of your three new functions by integrating them into your game. To do this, implement a fourth `public static` function called `isValid` which receives the same arguments as the other three, calls each of them in turn with those argument values and finally returns their combined results. Consider, how the corresponding boolean expression needs to look like.

After testing that your functions work properly it is time to integrate them into your game. To do this, first make sure you got rid of any test code in your `main` function. Then extend your handler for the "set field" action. After you have received x and y coordinates and a corresponding value from the user, pass them together with the grid into the `isValid` function. Based on the result, you should either set the value into the grid or print an error indicating that this value is not allowed for the specified position.

3 Optional Extra Tasks

If you have finished all of the above tasks and look to do more, please consider the following suggestions or come up with your own ideas:

Number clashes If the user provides an invalid value, indicate where the clashing number can be found (row, column, block) with additional command line output and/or by marking it in the printed grid.

Parsing a Sudoku from Command Line Arguments Your program would be a bit dull if you could only ever play a single Sudoku. As a quick workaround, you could provide a Sudoku grid via the command line arguments. Admittedly, this is still not a good solution but definitely a good exercise. We will implement a better solution in a later tutorial.

For now, provide all 81 numbers separated by space as a command line argument. Write a new function `parseArgs` which parses `String[] args` and returns the corresponding two dimensional game grid.

To make this a bit less tedious, here are a few Sudokus for copy pasting:

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