**Workshop 6**

**Subject**: 2D Arrays

**Concept**: Sudoku

**Workshop duration**: 90 Minutes

**Workshop brief**:

The main purpose of this workshop is to continue practicing two-dimensional Arrays and help students feel more comfortable using them in code. We will do so by implementing various functions in the Sudoku Class, which is based on the original game.

Note the skeleton java file that will be sent to the students before or in the beginning of the workshop.

For your convenience, we also added the solution.

If you desire to edit the presentation, you can do it [here](https://docs.google.com/presentation/d/1E6-8tHcAADxKrv1CPCTM725e-VY1bJlmdRW_0KR8M9Y/edit?usp=sharing), but please **make your own copy** if you would like to do that.

**Workshop structure**:

1. **Introduction**

**Duration:** 10~15 minutes

The objective of the original game is to fill a 9×9 square with digits, so that every column, every row, and every one of the nine 3×3 sub-squares within the square contains all of the digits from 1 to 9. In order to make the solution easier, the puzzle maker normally provides some of the digits, and the player has to complete the missing digits. Here is an example of a valid solution:

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

1. **Main workshop task student briefing**

**Duration:** 15~25 minutes

**2.1 what’s the task**

In order to make the problem more general, we will consider Sudoku games in which the size of the square can be any number N that has an integer square root. That is, N = 1, 4, 9, 16, 25, etc.

In this general case, the rules of the game are exactly the same, except that we have to fill the N-by-N square with the numbers 1,...,N (in the example that we saw above N=9).

We wish to write a Sudoku class that does one thing only:

**checks if a given N-by-N square contains a valid Sudoku solution**.

Once again, note that we don’t worry here about playing the game. Instead, we wish to verify that a proposed solution is valid. That’s all.

**2.2 Approaching the Task**

Before writing code, allow the students to think of how they can split the problem into sub-problems. What functions would they need to implement in order to make the solution simple and elegant?

First of all, we will need a “validation function” which, for a given array of integers of length N, checks if it contains all the numbers from 1 to N. That way, we will know that a specific row/column/sub-square is valid.

So, how would we use the above function for our purpose?

Specifically: Ask the students - how can we get the i-th row from a 2D array m as a 1D array?

Answer: It's simple: m[i] (why?).

What about getting a specific column? This is more complicated. We will need another function for this one.

In addition, we will need a function that "transforms" a sub-square to an array, to validate it also, using our validation function.

Propose the following implementation strategy, and mention that other approaches are also possible:

// Given: an array of size N.

// Checks if the given array contains all the numbers 1,2,3...,N.

private static boolean **containsAllNumbers**(int[] arr)

// Given a 2-dimensional array and a column number,

// returns an array containing the values in this column.

// Assumes that col is a valid column number -- no need to check it.

private static int[] **getColumn**(int[][] arr, int col)

// Given a square 2-dimensional array, an index (row,col), and a size n,

// returns the n-by-n sub-array whose top-left coordinate is (row,col).

// Assumes that row, col, and n are all valid -- no need to check it.

private static int[] **getSquare**(int[][] arr, int row, int col, int n)

// Checks if the given 2D-array is a Sudoku square

private static boolean **sudoku**(int[][] arr)

1. **Sub-Functions Implementation**

**Duration:** 15~25 minutes

1. For each sub-function give a proper explanation regarding its role to the main task

2. Then, give the students time slot to program it and then review it together

**3.1** private static boolean **containsAllNumbers**(int[] arr)

The following method checks that a given array of size N contains *all* the numbers 1,2,3...,N

(assume that all the numbers in the array are between 1 and N, inclusive).

Remind the students that the array index runs from 0 to N-1 and

ask them how this should help us in the implementation.

What could help us keep track of the array entries?

For example, containsAllNumbers({2, 1, 4, 3}) will return true,

while containsAllNumbers({1, 2, 3, 1}) will return false.

**3.2** private static int[] **getColumn**(int[][] arr, int col)

The following method gets a 2D array and a column number, and returns a 1D array that contains all the values in that column. The method assumes that the column number is valid, and there is no need to write code that checks it.

For example, getColumn(m,4) where m is the above matrix, will return the array [7, 9, 4, 6, 5, 2, 3, 1, 8]

**3.3** private static int[] **getSquare**(int[][] arr, int row, int col, int n)

The following method gets a 2D array and a sub-square within it.

The method returns a 1D array which is the “flattening” of the nxn sub-square whose

top-left coordinate is (row, col).

The method assumes that row, col, and n are all valid, and there is no need to write code that checks it.

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

Discuss with the students about what will be the range of the loops and how we can

"flatten" the sub-matrix into a one-dimensional array.

How should we manage the indexes of the new array in order to set

the entries in the right place?

For example, getSquare(m,6,6,3) will return the array [2, 8, 4, 6, 3, 5, 1, 7, 9]

1. **Suduku function Implementation**

**Duration:** 15~25 minutes

Here,

We’ll introduce the main function to the students and before revealing the implementation,

Give them the opportunity to program it by themselfs.

1. Discuss on how we should use the above functions in order to help us in solving the problem.
2. Tell the students that the test can be executed in several steps:
3. First, check that every row is valid (we already mentioned it).
4. Next, check that every column is valid (again, how does it differ from validating the rows?).
5. Finally, check that every sub-square is valid (how should we increment the indexes of the loops?).
6. Remind them that the square root of a number N can be obtained using the method call Math.sqrt(N).

Function Signature:

private static boolean **sudoku**(int[][] arr)

The following method checks if the values of the given 2D square array

form a valid Sudoku solution. Note that if the size of the given square array is *n*-by-*n*,

then according to the Sudoku rules, the size of every sub-square is by

(the method assumes that *n* has an integer root , and there is no need to check it).

1. **Workshop summary**

**Duration:** A few minutes

It would be useful in a case you have a spare time to sum up the main goal we wanted to deliver here.

guide throw the whole components that assemble the main function (suduko),

And of course,

**Q &A!!!!**

**I’m available to any question would come up 🙂**

**Yam**