sOFTWARE 2 PRACTICAL

## Design

Week 11 – Additional Exercises

The aim of this week additional exercises is to think about generative AI for coding, software design and working within a team (however you can do them alone too). Although you are not allowed to use generative AI for assessments, it is important to know how to use such tools and understand their limitations. Remember to review the code generated, and only use code that you understand.

We want to create a library to create new Sudoku puzzle, solve existing ones automatically, and make a command line Sudoku game.

# What is Sudoku?

Sudoku, originally known as Number Place, is a captivating logic-based and combinatorial number-placement puzzle. The goal is to fill a 9 × 9 grid with digits so that each column, each row, and each of the nine 3 × 3 sub-grids (also called “boxes” or “regions”) contains all the digits from 1 to 9. The puzzle starts with a partially completed grid, which should have a unique solution when properly posed.

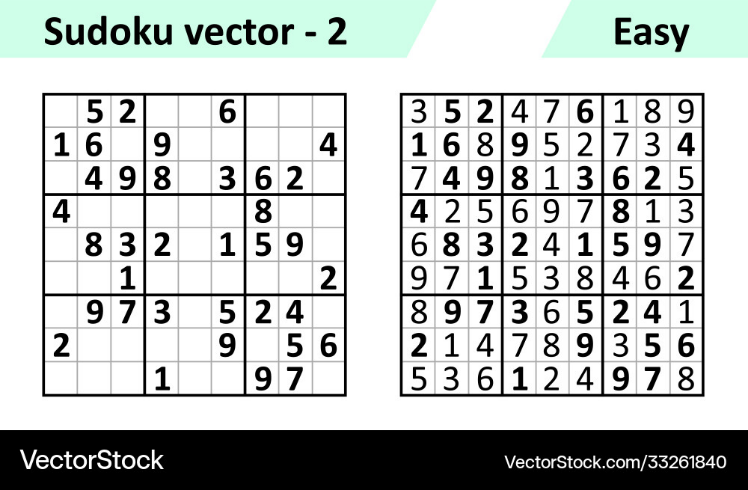


Figure 1: Starting Sudoku grid (left) and its solution (right).

The puzzle starts with a partially filled grid as shown in Figure 1. The objective is to fill in the entire grid with numbers from 1 to 9. Each row, column, and square (9 spaces each) must contain all the numbers from 1 to 9 without repetition. The puzzle is completed when there are no more blank spaces.

Note: there are several variants of the game, like Sudoku Kids () or HexaDoku ().

# To make things more interesting!

To complete these exercises, I would recommend that you find one (or two) of your peers and work together by using the code of the other person. For example, you both design your interfaces (Exercise 1), then swap your code to implement the concrete implementation of the interface (Exercise 2). Then swap your code again to implement the Sudoku solver (Exercise 3). By doing so, you will see how documentation and following the given requirements (interface) is essential in the production of software. Good luck!

# Exercise 1:

The first step is to design the ISudokuGrid interface that contains all the necessary methods to interact with the grid. The interface must be sufficiently flexible so that it can be used by:

1. SudokuSolver class, to solve the game using an algorithm,
2. SudokuGenerator class, to generate new puzzles,
3. SudokuGame class, to play the game from the command line.

These three classes should interact with the interface ISudokuGrid only, not one of its concrete class implementations (except when creating an instance of a sudoku grid). Try using a generative AI of your choice (Copilot, Blackbox AI, ChatGPT) to create the interface. Analyse the validity of the code generated and adapt it to your need. In addition, create the Javadoc for the entire interface.

### Exercise 2:

Implement the class SudokuGrid which implements ISudokuGrid. The class should have at least a default constructor that create an empty grid. Any methods not part of the interface and should be private (apart from the constructor). This is important to ensure your code does not depend on the implementation of the class, but only on its public interface. Try using a generative AI of your choice (Copilot, Blackbox AI, ChatGPT) to implement the class. Analyse the validity of the code generated and adapt it to your need. Write the Javadoc for the entire class where needed. Again, you could try to use generative AI to write the documentation.

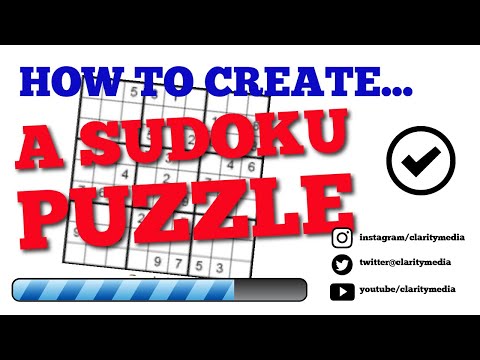
# Exercise 3:

Implement the SudokuSolver class. The SudokuSolver must use the ISudokuGrid interface from a concrete implementation of the said interface, and not implement its own grid or implement the ISudokuGrid. interface. One of the common techniques used to solve the problem is backtracking. The solver should be able to check if a puzzle is solvable, if the solution is unique, and give the solution(s). Try using a generative AI of your choice (Copilot, Blackbox AI, ChatGPT) to create the solver. Analyse the validity of the code generated and adapt it to your need. Should you create a java interface before? Provide the necessary Javadoc.

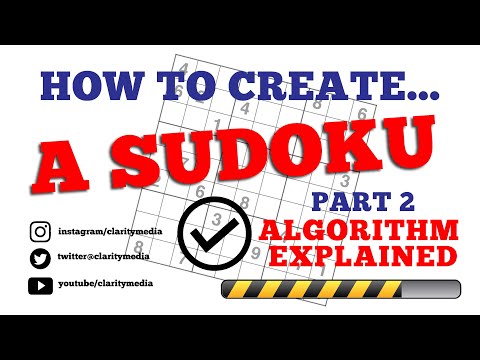
# Exercise 4:

Implement the SudokuGenerator class. The SudokuGenerator must use the ISudokuGrid interface from a concrete implementation of the said interface, and not implement its own grid or implement the ISudokuGrid. interface.

A recursive algorithm could be used to create the starting puzzle. The YouTube video below on how to fill the Sudoku grid explain the first part of the algorithm.

[](https://www.youtube.com/embed/iSdW8OM_b3E?feature=oembed)

Once we have a full grid, we need to remove values to create some blank spaces. The following video shows you a possible approach to do that.

[](https://www.youtube.com/embed/DpmTbMQFHaI?feature=oembed)

# Exercise 5:

Implement a Sudoku game that can be played using the command prompt. Again, the SudokuGame must use the ISudokuGrid interface from a concrete implementation of the said interface, and not implement its own grid or implement the ISudokuGrid.