DOCUMENTATION.

Design Patterns.

The code was not very long, but still there was a canPlace function that was being used twice, once in the Generator class and once in the Solve class. Hence, we pulled that function out and placed it into an interface, by using the Strategy Design pattern. It helped in making the code more readable, compact, and flexible.

Algorithm For Solving Sudoku.

BackTracking

Sudoku may be solved by assigning numbers to vacant cells one by one. Before you assign a number, make sure it's safe to do so. Verify that the same number does not appear in the current row, column, or 3X3 subgrid. After you've double-checked for safety, assign the number and recursively examine whether or not this assignment leads to a solution. If the assignment doesn't yield a result, attempt the next number for the current blank cell. Return false and output is no solution if none of the numbers (1 to 9) leads to a solution.

Inspired from computerphile

Checking Functions.

1.isEmpty:-checks whether the grid is empty or not. if empty, proceeds to fill in the number.

2.canPlace:-checks whether can we place that number in a particular grid and the same number doesn’t appear in the same column, row or 3X3 grid it belongs to.

OOP Principles.

1. There isn’t anything that varies across classes.
2. There isn’t any instance of inheritance in our code.

3. The canPlace method was common for both the Generator and the Solve class. A Placeable interface has been created with the common canPlace function and has been implemented in both the classes.
4. No instance of objects interacting with each other.
5. The canPlace method can be extended to any other class that needs it by simply implementing the Placable interface.
6. We don’t have many classes of the same type to depend on abstraction instead of concrete classes.

SANYAM GARG
2020A7PS1514P
VEDANT THAKKAR
2020A7PS0059P
https://github.com/Sanyam-Garg/OOP-Project-Submission