Assignment 3: Sudoku

2IP90, Programming - Q1 (2023)

1 | Sudoku

Startup

1. Create a new directory `sudoku` and open it in VSCode.
2. Download the files `SudokuGrid.java` and `SudokuSolver.java` and put them in the same folder.
3. Open the file and fill in your names and student IDs.

When you're finished with the assignment, submit `SudokuSolver.java` and `SudokuGrid.java` to Canvas.

Problem Description

Write a solver for Sudoku. Sudoku is a variation on the Sudoku puzzle. A Sudoku puzzle contains a grid of three by three boxes. These boxes contain three by three squares. Thus, the puzzle consists of nine by nine squares. Initially, some squares are filled with a number. The challenge for the puzzler is to fill in all squares with the numbers one (1) to nine (9) such that each row, each column, and each box contains each number one through nine exactly once.

Terminology:

- A grid is a matrix of nine by nine squares. Each square is either empty or contains a number between one and nine.
- A full grid is a grid without empty squares.
- We call a set of squares conflict-free if every number between one and nine occurs at most once in that set.
- We call a grid conflict-free if every row, column, and box is conflict-free.
- A grid h is an extension of a grid g if each non-empty square in g contains the same number as in grid h.
- A solution of a grid g is a full grid that is conflict-free and an extension of g.
- A puzzle is a grid that is conflict-free and has exactly one solution.

Input

No user input.

Output

If the solver finds a solution, your program prints that solution. Otherwise, your program prints a message indicating that no solution was found.

Example Run

If you run your program on the given Sudoku, you should get the following output:

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

Approach

- Use the given templates in files `SudokuGrid.java` and `SudokuSolver.java`.
- The `SudokuGrid` class should contain the following:
  - Constants for the grid size and digit range (e.g., `SIZE` and `DIGIT_RANGE`).
  - A method `SudokuGrid copy()` to create a copy of the grid.
  - A method `java.awt.Point findEmptyCell()` to find the next empty cell (use `null` to indicate when no empty cell was found). The method tries to find the next empty cell in reading order. That is, first left to right, then top to bottom. If this method succeeds, it returns the coordinates of the cell as an array {row, column}, and otherwise as the array {-1, -1}. Instead of {-1, -1}, you can also return `null`. Keep track of the last found empty square using instance variables `rEmpty` and `cEmpty`.
  - A `void print()` method to print the Sudoku grid.
  - Method to fill cells `fillCell(int r, int c)`.
  - Implement a method `boolean givesConflict(int r, int c, int d)` within the `SudokuGrid` class to determine whether filling in the number `d` in the square with position `r` (for row) and `c` (for column) will give a conflict. Separate this method further into three methods `rowConflict`, `colConflict` and `boxConflict`.
  - Constructors
- The `SudokuSolver` class should have a `SudokuGrid grid` and a boolean `solve()` method to check if the grid is solvable.
- In the `solve(SudokuGrid grid)` method, use a recursive strategy to find a solution:
  - If the recursive call returns `true`, you have a solution and return `true`.
  - Otherwise, undo what was filled in and continue with the next digit.
- Ensure that every Sudoku puzzle has at most one solution.