TP 1: Sudoku Solver

IFT2015 - Data Structures – A24

November 17, 2024

Objective

Develop an object-oriented program to solve a Sudoku puzzle. Given a 9x9 grid that may contain cells pre-filled with numbers between 1 and 9, your task is to complete the grid so that each row, each column, and each of the nine 3x3 sub-grids contain the numbers from 1 to 9 exactly once.

Problem Description

You will implement a Sudoku solver that reads one or more 9x9 puzzles with certain cells already filled with numbers from 1 to 9, while others are empty (represented by 0). Your program should solve the Sudoku by filling in the empty cells and outputting the solution to the puzzle.

Sudoku Rules

- Each number from 1 to 9 must appear exactly once in each row.
- Each number from 1 to 9 must appear exactly once in each column.
- Each number from 1 to 9 must appear exactly once in each 3x3 sub-grid.

We provide a series of tests (`SudokuApp`) that run and verify the solution for multiple puzzles. You must pass these tests in the main class, `SudokuApp.java`, to earn full execution points (see grading criteria).

Input-Output Formats

The tests consist of two-dimensional int arrays of exactly 9 rows by 9 columns (see `SudokuApp.java`). Inputs are numbers from 0 to 9, with 0 indicating an unfilled cell in the puzzle.

If the puzzle has a solution, output a line indicating "Puzzle Solved," followed by the solution of 9 rows containing numbers from 1 to 9, separated by spaces. If the puzzle has no solution, output a message indicating this, such as "No

solution exists." You must also check if the puzzle size is correct, 9x9. If it is not the correct size, you may print a message such as "Illegal dimensions."

Example Input

```
Integer[][] puzzle = {
    {5, 3, 0, 0, 7, 0, 0, 0, 0},
    {6, 0, 0, 1, 9, 5, 0, 0, 0},
    {0, 9, 8, 0, 0, 0, 0, 6, 0},
    {8, 0, 0, 0, 6, 0, 0, 0, 3},
    {4, 0, 0, 8, 0, 3, 0, 0, 1},
    {7, 0, 0, 0, 2, 0, 0, 0, 6},
    {0, 6, 0, 0, 0, 0, 2, 8, 0},
    {0, 0, 0, 4, 1, 9, 0, 0, 5},
    {0, 0, 0, 0, 8, 0, 0, 7, 9}
};
```

Example Output

```
Test Case 1: Sudoku Solved:
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
```

Structure and Requirements

You must follow object-oriented programming (OOP) principles and Abstract Data Types (ADT) for this project. Required interfaces and classes and their contracts are described below:

Mandatory Interfaces and Classes

⇒ **Interface `GameBoard<T>`:** Represents the abstraction of a game grid, a matrix with integer indices and generic type values `<T>`.

- Required methods (see `GameBoard.java`):
  - `T getCell(int x, int y)`
  - `void setCell(int x, int y, T value)`

  - `int getWidth()`
  - `int getHeight()`
  - `void display()`

- The implementation of `GameBoard` should be called `IntegerBoard` as the Sudoku game uses an integer grid:
  - Represents the 9x9 Sudoku grid
  - Provides methods from the `GameBoard` interface to:
    * Obtain and set grid dimensions
    * Obtain and set values in individual cells
    * Display the grid
    * Clone the current grid for encapsulation purposes

⇒ **Interface `GameSolver`:**
Represents the abstraction of a player capable of solving and printing the solution.

Requires the following methods (see GameSolver.java):

- `boolean solve()`
- `void printSolution()`

The implementation of `GameSolver` is called `SudokuSolver`, which contains the logic to solve the Sudoku puzzle, implementing the methods required by `GameSolver`. It solves and ensures the solution adheres to the game rules. Your program should solve all Sudoku puzzles within a file `SudokuApp.java`, which will be modified during the grading process.

⇒ **Class `SudokuApp`:**
Implements a test suite and verifies the functionality of the solver.

Makefile

A `Makefile` is provided to simplify the compilation and execution of the program and initial tests if you are using Unix.

Code and Submission

The deadline is November 17, 2024, at 11:59 PM. Submit your solution on StudiUM as a ZIP file with the following considerations:

- This project can be completed in teams of up to two. Collaboration is encouraged for designing, implementing, and testing the solution.
- Include all source code files in separate `.java` files in a ZIP archive:

  - `GameBoard.java`
  - `IntegerBoard.java`
  - `GameSolver.java`
  - `SudokuSolver.java`
  - `SudokuApp.java`

- Submit the archive with the format and file names: `TP1_f1_f2.tgz` or `TP1_f1_f2.zip`, where `f1` and `f2` are the last names of the two team members. If working alone, use `TP1_f.tgz` or `TP1_f.zip`, where `f` is your last name.
- Late submissions will incur a 20% penalty per 24-hour period starting at 12:00 AM on November 18, 2024.

Suggested Steps

To ensure that your program works correctly, follow these steps:

1. Navigate to the directory containing your files and make sure it includes the Makefile, `GameBoard.java`, `IntegerBoard.java`, `GameSolver.java`, `SudokuSolver.java`, `SudokuApp.java`, `Position.java`, `Tree.java`, and `AbstractTree.java`.
2. Compile the files by executing the `make` command in the terminal. This will compile all Java files and generate .class files.
3. Run the program by typing the command `make run`. This will display the results in the terminal (stdout).

Framework for `SudokuSolver`

```
public class SudokuSolver implements GameSolver {
    IntegerBoard board;
    IntegerBoard solution;
    Tree<IntegerBoard> ...;

    public SudokuSolver(GameBoard board) {
        ...
    }

    // mandatory interface methods
    public boolean solve() { ... }
    public void printSolution() { ... }
```

```
    // validate an insertion in the board
    public boolean isValidPlacement(int row, int col,
        Integer value) {
         ...
    }

    // actual solver
    private boolean solveBoard() { ... }
}
```

Your player must use a general tree structure of `IntegerBoard` to solve the puzzles, `Tree<IntegerBoard>`.

Grading System

Your work will be graded based on the following criteria:

- Correct code (solves puzzles) 20%
- Object-oriented design 20%
- Passes all tests 50%
- Cleanliness and readability 10%
- **BONUS 10%** - if you generalize your program to solve problems of any size, known as Sudoku variants, for example, 4x4, 16x16, you will receive 10 bonus points. Note, however, that you should return a solution instead of an error message for illegal dimensions. Note that there may also be puzzles with illegal dimensions, for instance, if the dimensions are not perfect squares since the number of cells on each side must be the square root of an integer.

Detailed Grading Criteria

Correct code: The program solves all formats but may not necessarily find the correct values.

Object-oriented design: The program should follow object-oriented programming principles. For example, minimal separation of information between classes, adherence to interfaces, encapsulation of classes, etc.

Passes all tests: It should pass the tests provided by the `App` as well as the test used for grading.

Cleanliness and readability: The code should be clean and properly commented.

Questions

For questions, please post on the TP1 forum on StudiUM or contact the teaching assistants or professor directly:

- Francois Major: `francois.major@umontreal.ca`
- Simon Guy: `simon.guy@umontreal.ca`
- Mohamed Elyes Kanoun: `mohamed.elyes.kanoun@umontreal.ca`
- Morteza Mahdiani: `Morteza.mahdiani@umontreal.com`

Game on!