

HW4 – Leveraging Reuse with Template Methods

Estimated time: 24-30 hours

Objectives

- Become more familiar with UML modeling, particularly when your algorithms dependent on how your structure data and conversely.
- Become familiar with applying the Template Method pattern appropriately
- Gain practice applying other patterns
- Learn to considering performance when selecting between design alternatives
- Improve unit testing skills

Overview

In this assignment, you will build a Sudoku puzzle solver (not an interactive player) and in the process make use of Template Method pattern and other appropriate patterns that we have studied to date.

For information about the Sudoku game, look at <https://en.wikipedia.org/wiki/Sudoku> and other online resources about the game, its rules, and strategies for solving it.

Functional Requirements

1. The solver should accept as input: a) the name of file containing a puzzle and b) the name of an output file that will contain the results.

- 1.1. Your program should be able to handle the following command line arguments

-h	displays a help message about the valid command line arguments
<input file name>	reads puzzle from the specified input file and writes the output to the console (or displays it in a GUI)
<input file name> <output file name>	reads puzzle from the specified input file and writes the output to the specify output file

- 1.2. [Optional]: The solver may have a GUI. Even if there is an GUI, it needs to except command line arguments.
 - 1.3. [Optional]: You may choose to allow a whole directory to be specified instead of single input files
 - 1.4. [Optional]: You may an output directory to be specified instead of a single file.

2. The solver must be able to solve puzzles of the following sizes: 4x4, 9x9, 16x16, 25x25, 36x36.
 - 2.1. You do not need to consider puzzles that are not perfect squares.
3. Your solver should be able to solve all solvable puzzles, as well as detect and report invalid puzzles as such
 - 3.1. A puzzle is a “valid” Sudoku puzzle if it has one and only one answer
 - 3.2. A puzzle is “invalid” if it is a) not formatted correctly or is not a 4x4, 9x9, 16x16, 25x25, or 36x36, b) doesn’t contain the right symbols, c) doesn’t have a solution, or d) has more than one solution.
4. Your solver should keep track of which techniques it uses to solve a puzzle, how much time it spends in doing each technique, and how much time takes to solve the whole puzzle.
5. The input file will be a text file containing the grid size, n , on the first row, followed by row containing the n different symbols that will be used in the puzzle. Each symbol must eventually appear in every row, column, or block. Lines 3 to $n+2$ represent the n rows the puzzle, and each should contain n symbols or dashes. The dashes represent the places (i.e., cells) where the solver needs to figure out which symbol belongs there. Here are some examples:

```

4
1 2 3 4
4 2 - 1
- - - 2
3 - 2 -
- 4 - 3

```

```

16
1 2 3 4 5 6 7 8 9 A B C D E F G
4 9 - 1 3 6 7 - 8 - - - D - -
- 6 3 5 - - - 9 - - A - - - -
5 - - - 2 9 3 6 4 - - - B - - -
- 2 - 3 1 - - 4 - - - - - - -
- 7 4 - - - 2 1 - - - F - - - -
- - 1 - 6 4 - 8 - - - - - 2 -
1 8 6 9 - - - 2 5 - - - - - -
- 4 - - 5 1 8 3 - D - - 2 - - -
3 - 9 4 8 - - 7 - - - - - - -
4 9 - 1 3 6 7 - 8 - - - D - -
- 6 3 5 - - - 9 - - A - - - -
5 - - - 2 9 3 6 4 - - - B - - -
- 2 - 3 1 - - 4 - - - - - - -
- 7 4 - - - 2 1 - - - F - - - -
- - 1 - 6 4 - 8 - - - - - 2 -
1 8 6 9 - - - 2 5 - - - - - -

```

6. The solver must write its results to the specific output file.

- 6.1. If a puzzle is solvable with one and only one solution, then solver save the solution into the output text file, using in the same format described for the input file, but with all of the dashes replaced by their correct symbols. For example, the output for the first puzzle shown above would be as follows:

```
4
1 2 3 4
2 - 3 1
1 3 - 4
3 1 4 -
- 2 1 3
```

```
Solved
2 4 3 1
1 3 2 4
3 1 4 2
4 2 1 3
```

Total time: 00:00:03.9630000

Strategy	Uses	Time
Apply Changes	2	00:00:00.0110000
Only One Possibility	1	00:00:00.0020000
Only One Place	0	00:00:00
Twins	0	00:00:00
Guess	0	00:00:00

Note: The actual strategies in your program may be different than those shown in the above example.

- 6.2. If the puzzle is not formatted correctly. The solver should output the original puzzle, followed by the line with the word "Bad Puzzle".

```
4
1 2 3 4
2 -1
1 3 - 4
```

Bad Puzzle

- 6.3. If the puzzle is not solvable. The solver should output the original puzzle, followed by the line with the word “Unsolvable”.

```
4
1 2 3 4
1 2 3 4
2 - 1 -
3 1 - 2
4 - 2 -
```

Bad Puzzle

- 6.4. If the puzzle has multiple solutions. The solver should output the original puzzle, followed by the line with the word “Multiple Solutions”, and then each solution (including size row and symbol rows)

```
4
1 2 3 4
- - - -
2 - 1 -
- - - -
4 - - -
```

Multiple Solutions

Instructions

To build this system, you will need to do the following:

1. Research different techniques for determining blank (dash) cells in a puzzle. We'll call these techniques “cell-solution algorithms”.
2. Spend some time modeling Sudoku puzzles from a structural perspective. For example, you could think about it as simple $n \times n$ array. Or, you could think about as rows, columns, and blocks, each of which contain n cells and where each cell is in exactly 1 row, 1 column, and 1 block. Or, you could think about them in term of some other data structure. Note that how you choose to think about the puzzle will have a direct impact on what cell-solution algorithms you can envision. With this kind of program, how think object structure directly constrains how you solve problems. Don't hesitate to experience with multiple structures.
3. After settling on an object structure, try to find at least three such cell-solution algorithms. Depending on how you break the up or classify them, you may have many more.
4. Determine the steps in each algorithm

5. Generalize the steps so you can extract a common template for all of the algorithms, this will become your template method.
6. Design the rest of your system.
 - 6.1. Feel free to use other patterns, but do not use a pattern without good justification. The emphasis will be on **appropriate application** and **not blind or forced use**. Hint: the template method and strategy patterns often work well together.
7. Implement your solver based on your design. Your implementation will be evaluate based on your design.
8. Test all non-GUI components using thorough executable test cases.
9. Test the system against the functional requirements, using test cases provided by the instructor and your own test cases.
10. Pay attention to performance. Think about how you can make your performance faster, without comprising design principles.

Submission Instructions

Zip up your entire solution, including test cases and sample input files, in an archive file called CS5700_hw4_<fullname>.zip, where fullname is your first and last names. Then, submit the zip file to the Canvas system.

Grading Criteria

Criteria	Max Points
A clear and concise designs consisting of UML class diagrams for the structural components of system, plus sequence diagrams that describe several key scenarios (e.g., the loading of a puzzle, the solving a solvable puzzle, the creation of an output file afterwards, etc.) The design must include at least one application of the Template method pattern. The design should include applications of other pattern, if problems that they solve exist and their context is applicable.	30
A working implementation that can solve puzzles with an easy or medium difficulty level. The implementation must follow the design and have good encapsulations, abstractions, appropriate use of inheritance, and aggregation.	40
Thorough executable unit test cases for core application logic	40
Reasonable systems test using ad hoc methods	10