

Homework #3

due Tuesday, October 16, 12:30 PM

For this assignment, you will write both a Java application and an applet to solve Sudoku puzzles. A Sudoku puzzle requires logic alone to solve and is very amenable to computer solution. In case you haven't seen one before, consider the following example (courtesy Milwaukee Journal Sentinel): (ignore the letters *A*, *B* and *C*)

			3	9	5			
	<i>C</i>	5			8	9		2
				2		<i>B</i>		5
6		2						7
	8	4				5	3	<i>A</i>
7						1		6
3				6				
5		6	2			7		
			8	3	1			

The puzzle is to fill in the empty squares so that every 3x3 square, every row and every column has each of the digits 1 through 9 once. You can use logic to figure out what digits must go where. For example:

- A* The box marked *A* must include a “9” because the digits “1,3,5,6,7” are already in the box, “4, 8” are already in the row and “2” is already present in the column. Thus, by a process of elimination, “9” must belong here. This is the easiest strategy to use to solve puzzles and is the only one needed to solve this example, but this strategy is insufficient for many puzzles, including the one published in the Milwaukee Journal Sentinel for today, October 4th.
- B* The upper-right 3x3 box (with *B* in it) does not currently have a “3” in it, but a simple inspection will show that a “3” cannot occur on the top row of the box because of the “3” in the box at the top of the puzzle, nor can it be in the middle column because of the “3” in the middle box on the right. Thus, the only place where a “3” could be placed is at location *B*.
- C* The second row (with *C* in it) also does not have a “3” in it. But the 8th space is in the same column with the “3” that prevented “3” in the middle row of the upper-right 3x3 box, and thus it can't have a “3”. Neither can the 4th or 5th spaces, because they occur in a 3x3 box (the top box) that already has a “3.” The 1st space is not possible because of the “3” in the lower-right 3x3 box, and thus again, this leaves the second space (marked *C*) as the only place where a “3” can occur on this row.

These three strategies (where strategy C applies to looking for where a digit can go in a column too) are sufficient for most of the Sudoku problems published. (I provide one Sudoku puzzle from The London Times for which these strategies are insufficient.)

For this assignment you will write a Sudoku solver that uses at least these strategies to solve problems. The Sudoku solver will have two interfaces: one command-line and textual and the other graphical.

Undergraduate students may work in teams of 2 for this homework. Use AFS to give both of you permission to access a homework directory *before you start work*.

1 Sudoku Solver

The first task for this homework is to write a Sudoku solver that works as a Java command-line application. It will read in a Sudoku problem file (named on the command-line) which uses a simple format to express the boards. An example of the format is the following

```

--- 395 ---
__5 __8 9_2
--- _2_ __5

6_2 ___ __7
_84 ___ 53_
7__ ___ 1_6

3__ _6_ ___
5_6 2__ 7__
--- 831 ---

```

which is for the board shown on the first page of this homework.

The solver will consist of applying *strategies* to the board (in order, the easiest first) until one is able to find a spot to solve. A strategy thus has the following definition:

```

public interface SudokuCellStrategy {
    /**
     * Try a Sudoku strategy. Return a solution for
     * a cell, or null if nothing can be done
     * @return cell solution if done, or null if no work possible
     */
    public abstract SudokuCellSolution work(SudokuBoard board);

    /**
     * Return the "intelligence" level of this strategy
     * @return intelligence level of strategy.
     */
    public abstract int getLevel();
}

```

(The “intelligence” level is one, two or three respectively for strategies *A*, *B* and *C*, but is irrelevant for the textual Sudoku solver.) This is an example of the STRATEGY design pattern from the textbook. Each of the three strategies (*A*, *B*, *C*) implements this interface. Additionally, strategy *B* and both kinds of strategy *C* (rows and columns) share a lot of code and should be implemented using the TEMPLATE design pattern.

Your `SudokuSolver` class should be neutral with respect to whether the program is textual or an applet. The specifically textual part of the program (opening files and printing boards) should be done in a separate class `TextSudokuSolver`. This class should print the board at the beginning and after every step.

2 Sudoku Solver Applet

The second part of your homework is writing an animation applet that accepts a Sudoku problem as an applet parameter and then solves it graphically. It should also handle the other parameters (such as “delay”) to indicate the speed of doing the puzzle.

You will need to use the refactoring of the animation applet hierarchy done in lecture. These files will be available in the `$CLASSHOME/src/lecture/` directory as usual. Delay starting on this part until you have finished the textual Sudoku solver.

As well as solving the problem step by step, you should also use color to indicate the “intelligence” level of the strategy that was used to fill in the digit. It is also desirable, but not required, that alternate 3x3 boxes be shaded darker. The method `darker()` in the `Color` class is useful here. (The pale green used in the solution is defined as `Color(192,255,192)`.)

An important aspect of this part of the Homework is ensuring that the two solvers share all code that is used for solving problems and only need to provide the medium-specific aspects.

3 Files

Please put your Java files for your solution in a package named `sudoku` in your `homework3` directory, except for the strategies themselves which should be in a package `sudoku.strategy`.

Some major parts of the Homework are already done for you: the Sudoku board class itself, as well as the two helper classes `SudokuCellStrategy` (already shown in this homework) and `SudokuCellSolution`. In particular, you will not need to write the code to parse the Sudoku board format. We also provide several Sudoku problems for your textual solver (files with a `prb` extension) and for your applet solver (files with a `html` extension). These files are all in the `$CLASSHOME/src/homework3` directory.

Both projects must be runnable from within their respective directories. In other words, on a grid, it must be possible to

```
grid.cs: cd homework3/bin
grid.cs: java sudoku/TextSudokuSolver ../2005-10-4.prb
...
grid.cs: appletviewer Sudoku95.html
```

As usual, all work must be completed by 12:30pm, Tuesday, October 16th.