Tutorial: Sudoku

Programming II : Objects

Autumn Term 2017

Rules

From `http://www.sudoku.com/`:

The goal of Sudoku is to fill in a 9 by 9 grid with digits so that each column, row, and 3 by 3 block contain the numbers between 1 to 9. At the beginning of the game, the 9 by 9 grid will have some of the squares filled in. Your job is to use logic to fill in the missing digits and complete the grid. A move is incorrect if:

- Any row contains more than one of the same number from 1 to 9
- Any column contains more than one of the same number from 1 to 9
- Any 3 by 3 grid contains more than one of the same number from 1 to 9

When people play Sudoku they frequently put in the squares using a very small font possible numbers that a given square could take. These are called *pencil marks*. As they fill in the squares they remove pencil marks that are no longer possibilities. The way this problem suggests you solve a Sudoku puzzle is the following:

> Put into each empty square (using pencil marks) all the possible numbers that could be in the square. After each move remove those pencil marks that are no longer possible. For easy games there should always be at least one square where there is only one pencil mark. Make that move and start again.

What to implement

Because a Soduko game is by its definition 9 by 9 you may put this literal in the code itself. The classes you need to implement are the following:

- `Square` – This will be a single square. It may have a value or be empty. It may have pencil marks of possible numbers that could be in the square.
- `Grid` – This will be a 9 by 9 array of `Square`s.
- `Solver` – This should contain the `main` method and solve a given puzzle.

1. Start by implementing `class Square` (remembering to put it into a file called Square.java).
   (a) You need first to declare the `private` fields. These should hold the value (probably 0 for empty, 1-9 for values) and the pencil marks. Think how you wish to store the pencil marks. It also may be easier to write the methods if you store the number of pencil marks as well as the pencil marks – but this isn't essential.
   (b) Write the following methods. You may write any helper methods that will make your code clearer which should be made `private`.

i. `int value()` – returns the value in the square, what to do if the square is empty is up to you

ii. `boolean isEmpty()` – true iff it doesn't contain a number between 1 and 9

iii. `boolean hasUniquePMark()` – a square has exactly one pencil mark

iv. `int uniquePMark()` – for a square that has exactly one pencil mark, what its value is

v. `void removePMarkIfThere(int n)` – if a square has a pencil mark n then after this method has executed it no longer has that pencil mark

(c) You may wish to write a `toString` method for printing a square. (It could have borders or not, but it only should contain the value, not the pencil marks).

(d) Write two constructors for `Square`. One should be for an empty square and the other for one that holds a number.

2. Next implement `class Grid`, a 9 by 9 array of `Square`s. This should be a `private` field.

(a) Implement either a `toString` or a `print` method so that grids can be printed.

(b) Write the following methods. You may write any helper methods that will make your code clearer which should be made `private`.

i. `boolean hasFreeSpace()` – returns true iff there is a space on the grid without a number between 1 and 9

ii. `boolean isGoodMove(int row, int col)` – returns true iff there is a possible move at grid[row][col]

iii. `void makeMove(int row, int col)` – fill in grid[row][col] with the correct number (pre `isGoodMove`

iv. `void removePMarks(int row, int col)` – at grid[row][col] get rid on any wrong pencil marks

(c) Implement the constructor for `Grid`. It should take a two dimensional array of numbers as an input parameter. The constructor needs to initialise each of the squares, both with values and with pencil marks.

3. Finally implement `Solver`. You can do it with your implemented classes or add some additional methods to them. Here is a game that it should work for:

```
static int[][] game = {
     {0, 6, 0, 3, 0, 0, 8, 0, 4},
     {5, 3, 7, 0, 9, 0, 0, 0, 0},
     {0, 4, 0, 0, 0, 6, 3, 0, 7},
     {0, 9, 0, 0, 5, 1, 2, 3, 8},
     {0, 0, 0, 0, 0, 0, 0, 0, 0},
     {7, 1, 3, 6, 2, 0, 0, 4, 0},
     {3, 0, 6, 4, 0, 0, 0, 1, 0},
     {0, 0, 0, 0, 6, 0, 5, 2, 3},
     {1, 0, 2, 0, 0, 9, 0, 8, 0}
 };
```

There has been a vast amount of research into finding ways to solve Sudoku puzzles and once you have finished this solver you may wish to make your solver stronger. A simple improvement to the *look for the single pencil mark* algorithm would be in each unit (row, column, block) *look for any pencil mark that only appears in exactly one square* and make that move. For example in a row, a square could have pencil marks 3 and 4. If there are no other pencil mark 4s in the row then even though there is a 3 as well as a 4 in the pencil marks, the square must have value 4. This algorithm includes the simpler one.