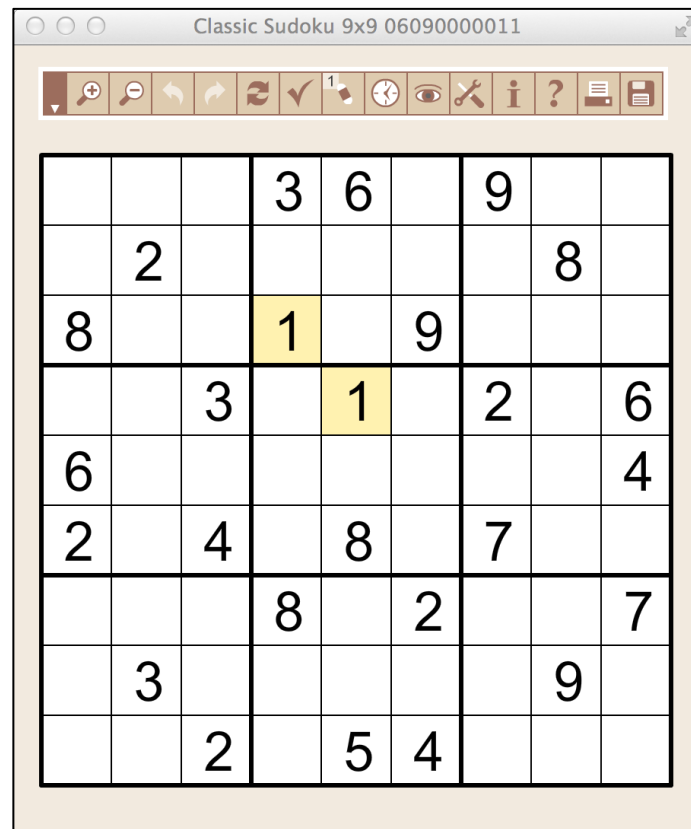


CS 46B
Spring 2017
Homework 5: Sudoku



Due: 11:59PM, Tuesday April 4. THAT'S SOONER THAN YOU THINK!

INTRODUCTION: Sudoku is just a puzzle, but the backtracking technique for solving it is used in many important application domains. To solve a Sudoku without backtracking, you could generate every possible solution, then evaluate all of them and collect the legal ones. But there are 9^{81} ways to fill in a Sudoku grid. 9^{81} is $\approx 2 \cdot 10^{77}$. If you could evaluate a billion grids per second, it would take $2 \cdot 10^{68}$ seconds. The universe is less than 10^{18} seconds old. For this homework, you will learn the power of backtracking by writing a Sudoku solver that finds answers in just a few seconds.

The puzzle on page 1, and all puzzles that appear in the code, are copyright 1997-2016 Conceptis Ltd.

SUDOKU Rules: A puzzle looks like the one on the previous page: a 9x9 grid with some numbers filled in. The challenge is to fill in each empty space with a digit 1 through 9, so that no digit is repeated in any row, column or “block”. “Block” is my name for the nine 3x3 blocks outlined in thicker black lines in the picture.

Your Assignment: In the Eclipse workspace of your choice, create a new Java project containing package “sudoku”. Import the 4 starter files that you downloaded with this assignment: Evaluation.java, Grid.java, Solver.java, and TestGridSupplier.java. You won't need to change Evaluation.java or TestGridSupplier.java. Your assignment is to finish Grid.java and Solver.java.

Grid: This class models a Sudoku puzzle that is unsolved, partially solved, or completely solved. The class has a 9x9 array of ints, called “values”. If a Sudoku square is empty, the corresponding cell in “values” is zero; otherwise the cell in “values” contains the number in the Sudoku square. The starter class has a ctor and a toString() method that you should not change. It also has 4 empty methods that you need to write: next9Grids(), isLegal(), isFull(), and equals(). Do not change their names, arg lists, or return types. The comments on the unfinished methods tell you all you need to know about what they should do. It's ok to add more methods to this class.

You don't need to provide this class with hashCode() or compareTo() methods. The equals() method is just so that you can compare your puzzle solutions to solutions in TestGridSupplier. (That is, you won't be collecting Grid instances into a hash set a tree set.)

Solver: Most of this class has already been written. Complete the solveRecurse() method using the backtracking technique you saw in lecture and lab. Also complete the evaluate() method. The main() method is for you to use while testing your code with the puzzles and solutions in TestGridSupplier.

Evaluation: You saw this enum in lecture and lab. It contains 3 values that represent the 3 possible outcomes of the `evaluate()` method of the `Grid` class. Look at the source code. Sometimes enums can be complicated, but this one is not. You might need to write a simple enum for the next midterm.

TestGridSupplier: This class contains static methods that return `Grid` instances. Some of them are puzzles, some are solutions, and some are for testing your code.

SUBMISSION: As usual, export your sources from Eclipse. Be sure your jar contains all your sources. You will receive zero points if your jar doesn't contain sources. No work will be accepted after the deadline except for documented emergencies.