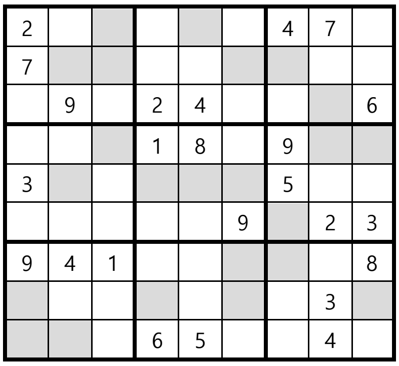
Objectives

- Write a Java program to solve an even/odd Sudoku puzzle.
- Solve logical requirements using Boolean satisfiability solving (SAT).
- Perform automated testing using JUnit5, involving File I/O.

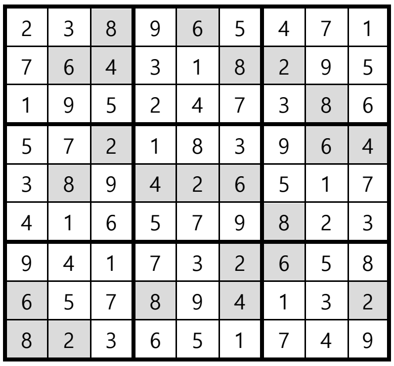
Even/Odd Sudoku

- Even/odd Sudoku is a variant of Sudoku. The goal of even/odd Sudoku is to fill numbers from 1 to 9 in the empty cells of a 9x9 grid such that
    - 1~9 appear exactly once in each row, column and 3x3 box.
    - All grey cells has even numbers, and the others has odd numbers.



2		▨		▨		4	7	
7	▨	▨			▨	▨		
	9		2	4			▨	6
		▨	1	8		9	▨	▨
3	▨		▨	▨	▨	5		
					9	▨	2	3
9	4	1			▨	▨		8
▨			▨		▨		3	▨
▨	▨		6	5			4	

Unsolved



2	3	8	9	6	5	4	7	1
7	6	4	3	1	8	2	9	5
1	9	5	2	4	7	3	8	6
5	7	2	1	8	3	9	6	4
3	8	9	4	2	6	5	1	7
4	1	6	5	7	9	8	2	3
9	4	1	7	3	2	6	5	8
6	5	7	8	9	4	1	3	2
8	2	3	6	5	1	7	4	9

Solved

- In this assignment you write a Java program to solve even/odd Sudoku. The input and the output of the program are plane text files.
    - An input file contains 9 lines of 9 characters (digits, "*", or "."). Unknown numbers are represented by either "*" (even) or "." (odd).
    - Output files must be in a similar format, where the file names have the form: "[input_file_name]_[number_of_solution].solution".
    - E.g., if the input file is "test" and there are 3 solutions, the output files are "test_1.solution", "test_2.solution", and "test_3.solution".

```
2.*.*.47.
7**..**..
.9.24..*6
..*18.9**
3*.***5..
*.*..9*23
941..**.8
*..*.*.3*
**.65..4.
```

Input Format

```
238965471
764318295
195247386
572183964
389426517
416579823
941732658
657894132
823651749
```

Output Format

Boolean Satisfiability Problem

- The requirements of even/odd Sudoku can be encoded as Boolean formulas, composed of Boolean variables and operators: AND (&), OR (|), NOT (!).
- The Boolean satisfiability problem (SAT) is to find an assignment of truth values to the variables that makes a given formula *True*. For example:
    - "(x_1 | !x_2) & x_3" is satisfiable by: x_1 = True, x_2 = True, x_3 = True.
    - "(x_1 | x_2) & !x_1 & !x_2" has no satisfiable truth assignments.
- SAT solving often considers a restricted form of Boolean formulas in conjunctive normal form (CNF).
    - A CNF formula is a conjunction of clauses.
    - A clause is a disjunction of literals.
    - A literal is a variable or its negation.
- E.g., consider the CNF formula "(x_1 | !x_2 | x_3) & (!x_1 | !x_2) & x_2".
    - This formula has three clauses "x_1 | !x_2 | x_3", "!x_1 | !x_2", and "x_2".
    - The clause "x_1 | !x_2 | x_3" has three literals "x_1", "!x_2", and "x_3".
    - "x_1" and "x_3" are variable literals, and "!x_2" is a negated literal.
- See the following Wikipedia article for more information:
    - https://en.wikipedia.org/wiki/Conjunctive_normal_form

SAT Solver

- A SAT solver is a program that solves the satisfiability problem. For example, Sat4j (http://www.sat4j.org) is a SAT solving library for Java.
- SAT solvers, including Sat4j, normally consider formulas in CNF. In this case, a CNF formula is written as a set of sets of integers, where:
    - A positive number denotes a variable literal.
    - A negative number denotes a negated literal.
- For example, consider the following CNF formulas:
    - "(x_1 | x_2) & (x_2 | !x_3)" is written as the set: {{1,2}, {2,-3}}.
    - "(x_1 | !x_2 | x_3) & (!x_1 | !x_2) & x_2" is written as: {{1, -2, 3}, {-1, -2}, {2}}.
- We provide a simple example to show how to use the Sat4j library. For more information, please see the links below:
    - https://sat4j.gitbooks.io/case-studies/content/
    - http://www.sat4j.org/maven234/apidocs/index.html

Logical Encoding of Even/Odd Sudoku

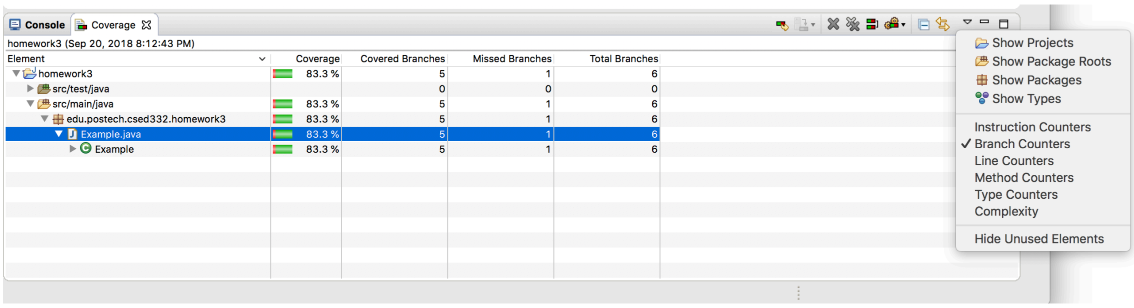
- In this assignment **you must use a SAT solver** to solve an even/odd sudoku puzzle. The requirements of the puzzle are encoded as a formula in CNF.
- The encoding involves 729 = 9*9*9 variables, namely, $x_{i,j,k}$ for 1 ≤ i, j, k ≤ 9.
  - Variable $x_{i,j,k}$ is assigned *True* iff the cell in row i and column j is assigned number k. That is, each cell involves 9 variables.
  - For example, $x_{1,2,3}$ = True means that the cell in row 1 and column 2 has the number 3.
  - In Sat4j, each variable $x_{i,j,k}$ must be expressed as a single number, so you need to relate each triple (i, j, k) to a certain (unique) number.
- The encoding asserts the following requirements, where each satisfiable truth assignment of 729 Boolean variables gives a solution of the puzzle:
  - There is at least one number in each cell.
  - Each number appears at most once in each row.
  - Each number appears at most once in each column.
  - Each number appears at most once in each 3x3 box.
  - Each even cell has an even number.
  - Each odd cell has an odd number.
  - A solution must be consistent with the starting grid.
- Your program will generate an encoding of an input even/odd Sudoku puzzle, and invoke the Sat4J SAT solver to find a satisfiable assignment.
  - Remember that the encoding is a CNF formula, which is a conjunction of clauses, where each clause is a disjunction of literals.
  - You need to formulate the above requirements in this way. The following encoding examples can be helpful.
- Encoding examples:
  - "Cell (1, 2) has number 3" can be encoded the clause: { $x_{1,2,3}$ }
  - "Cell (1, 2) cannot have number 3" can be encoded as the clause: { !$x_{1,2,3}$ }
  - "Cell (1, 2) has at least one number" can be encoded as the clause: { $x_{1,2,1}$, $x_{1,2,2}$, $x_{1,2,3}$, $x_{1,2,4}$, $x_{1,2,5}$, $x_{1,2,6}$, $x_{1,2,7}$, $x_{1,2,8}$, $x_{1,2,9}$ }.
  - "Cells (1, 2) and (2, 2) cannot have number 3 at the same time" can be encoded as the clause: { !$x_{1,2,3}$, !$x_{2,2,3}$ }.

Skeleton Code

- The src folder containing the skeleton for the source code. There are 3 classes in the source code (Game, Sudoku, Solution).
- You should implement all methods with TODO. In addition, you can freely add more member variables, methods, and classes, if needed.
- **Do not modify the API of existing methods, since they will be used for grading.** (You can add a throws clause to existing methods, if you want).
- You need to modify pom.xml to declare extra dependencies for Maven (such as junit5, Sat4j, etc.).

Tests and Coverage

- You should implement Junit5 test methods for each method in the provided classes (Game, Sudoku, Solution).
- The test classes contain test methods to check the functionality of each method in the classes in the source code (i.e., Game, Sudoku, Solution).
- Your tests must achieve **90%** branch coverage. Please add more tests to achieve desired coverage. (You will get bonus points for higher coverage).
- We will use "mvn test" for grading. Please make sure that "mvn test" runs correctly (otherwise, you may get 0 points).



homework3 (Sep 20, 2018 8:12:43 PM)

Element	Coverage	Covered Branches	Missed Branches	Total Branches
homework3	83.3 %	5	1	6
src/test/java		0	0	0
src/main/java	83.3 %	5	1	6
edu.postech.csed332.homework3	83.3 %	5	1	6
Example.java	83.3 %	5	1	6
Example	83.3 %	5	1	6

Turning in

- Create a private project "homework3" in http://gladius.postech.ac.kr, and clone the project on your machine.
- Commit your changes in your "homework3" project and push them to the remote repository that includes your pom.xml file and your src folder.
- You have to tag the project with "submitted" when you finish your homework (see https://docs.gitlab.com/ee/university/training/topics/tags.html).