Assignment 1

CS 1027
Computer Science Fundamentals II

Due date: Thursday, October 5 at 11:55 pm

Learning Outcomes

In this assignment, you will get practice with:

- Creating classes and their methods
- Arrays and 2D arrays
- Conditionals and loops
- Working with objects that interact with one another
- Creating a subclass and using inheritance
- Programming according to specifications

Introduction

Sudoku is a popular number puzzle in which the numbers 1 through *n* have to be placed into an *n* by *n* grid of cells such that each row and column contains each of the *n* digits exactly once each. Most sudokus have *n* = 9 so the numbers 1 through 9 are placed into a 9 by 9 grid. In this popular 9 digit sudoku, there is an additional rule that each 3 by 3 box must contain each of the 9 digits exactly once. Sudokus generally have some given digits from which the solver should be able to correctly complete the remaining cells using deduction and other strategies. A sudoku must have only one possible solution based on the given digits and to be valid it must follow **all** the rules mentioned above. In Figure 1 below, the left image is an example of a valid sudoku solution. On the right side, the sudoku is invalid because there is a 7 in row 2 column 8 (highlighted in yellow) which conflicts with the other 7s in its row, column, and box. All these 7s are shown in red text.

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

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

Figure 1. An example of a valid 9x9 sudoku (left) and an invalid 9x9 sudoku (right).

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

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

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

Figure 2. Depictions of a row (left), column (middle), and 3x3 box (right) in a valid 9x9 sudoku

Figure 2 above shows examples of a valid row (left), column (middle), and 3x3 box in which each contains all the digits from 1 through 9 exactly once.

In recent years, variations of sudokus have emerged that combine the standard rules of sudoku with additional rules or restrictions for the placement of digits in different parts of the grid. One such variation is a unique diagonal sudoku in which the numbers along one or both of the main diaginals (top left to bottom right OR bottom left to top right) must also contain each of the *n* digits exactly once each without repeating any. The regular rules of sudoku **must** still be followed in addition to the new diagonal rule. The sudoku shown below in Figure 3 is an example of a valid unique diagonal sudoku because it satisfies the rules of a regular sudoku **and** it has each of the digits 1 through 9 exactly once along its bottom-left to top-right diagonal.

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

Figure 3. An example of a valid unique diagonal 9x9 sudoku.

Assignment 1

CS 1027
Computer Science Fundamentals II

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

Figure 4. An example of an invalid unique diagonal 9x9 sudoku. This is a valid regular sudoku (the numbers 1-9 are placed once each in each row, column, and 3x3 box) but does not have all unique numbers along either of its diagonals so it is not a valid Unique Diagonal sudoku.

The sudoku shown in Figure 4 is not a valid unique diagonal sudoku. Although it is a valid regular sudoku, it does not have all the digits 1 through 9 exactly once along either of its diagonals. The digits 2 and 7 are repeated along the bottom-left to top-right diagonal (highlighted in yellow). The other diagonal also contains repeated digits (not highlighted).

For this assignment, we will be using a 2D (2-dimensional) int array to represent the grid of a Sudoku puzzle and implementing various methods to check if a given sudoku is a valid solution based on the rules explained above. We will also work with the unique diagonal variant of sudoku and check its validity as well. The assignment does not involve solving sudoku puzzles or making an interactive sudoku playing system.

For simplicity, we use 0-based indexing for rows and columns in the sudoku. For example, in the sudoku shown in Figure 4 above, we would say that the top-left corner cell (containing the digit 9) is at row 0 column 0. The 4 in the cell adjacent to it is at row 0 column 1. The bottom-right cell (containing the digit 5) is at row 8 column 8.

Assignment 1

CS 1027
Computer Science Fundamentals II

Classes to Implement

For this assignment, you must implement two (2) Java classes: **Sudoku** and **UniqueDiagonalSudoku**. Follow the guidelines for each one below.

In these classes, you may implement more private (helper) methods if you want. However, you may not implement more public methods **except** public static void main(String[] args) for testing purposes (this is allowed and encouraged).

You may **not** add instance variables other than the ones specified in these instructions nor change the variable types or accessibility (i.e. making a variable public when it should be private). Penalties will be applied if you implement additional instance variables or change the variable types or modifiers from what is described here.

You may **not** import any Java libraries such as java.util.Arrays.

Sudoku.java

This class is used to represent a sudoku containing integers in a 2D array.

The class must have exactly (no more and no less) the following private instance variables:

- private int size (this represents the size of the grid, assuming it will always be a square)
- private int[][] grid (this is the grid containing all the digits)

The class must have the following public methods:

- public Sudoku (int[][] numbers): constructor
  - Initialize the instance variables size and grid using the given parameter numbers.
  - Note that grid must be copied as a shallow copy, meaning the same object reference from the parameter numbers will be used for this instance variable grid
- public int getSize ()
  - Return the size variable
- public int[][] getGrid ()
  - Return the grid variable
- public int getDigitAt (int row, int col)
  - Return the digit stored in the grid at the given row and col indices
  - If either of the indices are out of range (i.e. less than 0 or larger than size-1), return -1
- public boolean isValidRow (int row)
  - Determine if the row at index row in the sudoku is valid, i.e. that it contains all the digits from 1 through size (inclusive) exactly once each. Return true if it's valid, or false otherwise. If the row index is out of range, return false.
- public boolean isValidCol (int col)

    - Determine if the column at index col in the sudoku is valid, i.e. that it contains all the digits from 1 through size (inclusive) exactly once each. Return true if it's valid, or false otherwise. If the col index is out of range, return false.
- public boolean isValidBox (int row, int col)
    - Determine if the 3x3 box whose top-left corner is at index row, col in the sudoku is valid, i.e. that it contains all the digits from 1 through 9 (inclusive) exactly once each. Return true if it's valid, or false otherwise. If the row or col indices are out of range, return false.
    - NOTE: this method will **only** be called for a 3x3 box in a 9-sized sudoku so it does **not** need to work for any other sized boxes
- public boolean isValidSolution ()
    - Determine if the entire sudoku is valid by calling isValidRow() and isValidCol() for every row and column and ensuring they are all true. If the sudoku is size 9, you must also call isValidBox for all nine 3x3 boxes (this is **only** needed for size 9 sudokus) and ensure that they are all true as well. If **all** conditions are true, then return true to indicate the entire sudoku is valid, or false if at least one condition is not met.
- public boolean equals (Sudoku other)
    - Determine if the 'this' sudoku is identical to the other Sudoku. To be identical, the size of each must be equal **and** the digits in the grid of each must **all** me equal in the same positions. Return true if they are identical, or false otherwise.
- public String toString ()
    - Build and return a string containing all the digits in the grid with a single space after each digit and a newline character (\n) at the end of each row (after the space that follows the final digit in the row) so that the printed string looks like a grid structure.

UniqueDiagonalSudoku.java

This class is used to represent a sudoku in which the digits along one or both of the main diagonals must all be unique (no repeats) to be considered a valid solution. Since this is a special type of sudoku, this class **must inherit** from the Sudoku class.

No new instance variables are needed in this class.

The class must have the following public methods:

- public UniqueDiagonalSudoku (int[][] numbers): constructor
    - Use super() to call the Sudoku constructor.
- public boolean isValidSolution ()
    - This method is overriding the isValidSolution() method in the Sudoku class

- The regular sudoku rules still apply for a UniqueDiagonalSudoku, so call the parent isValidSolution() to check that it follows the standard rules (hint: how can you call the parent isValidSolution() method from inside the overridden one?)
- The additional rule is that one or both of the main diagonals (from top-left to bottom-right and from bottom-left to top-right) must contain all the digits from 1 through size (inclusive) exactly once each.
- If both conditions are true (regular sudoku rules are followed **and** at least one diagonal contains all the valid digits once each without repeats), return true. Otherwise, return false.

Provided files

TestSudoku.java is a tester file to **help** check if your java classes are implemented correctly.

Similar files will be incorporated into Gradescope's auto-grader. Additional tests will be run that will be hidden from you. **Passing all the tests within the provided files does not necessarily mean that your code is correct in all cases.**

Marking Notes

Functional Specifications

- Does the program behave according to specifications?
- Does it produce the correct output and pass all tests?
- Are the classes implemented properly?
- Does the code run properly on Gradescope (even if it runs on Eclipse, it is **up to you** to ensure it works on Gradescope to get the test marks)
- Does the program produce compilation or run-time errors on Gradescope?
- Does the program fail to follow the instructions (i.e. changing variable types, etc.)

Non-Functional Specifications

- Are there comments throughout the code (Javadocs or other comments)?
- Are the variables and methods given appropriate, meaningful names?
- Is the code clean and readable with proper indenting and white-space?
- Is the code consistent regarding formatting and naming conventions?
- Submission errors (i.e. missing files, too many files, etc.) will receive a penalty.
- Including a "package" line at the top of a file will receive a penalty.

Assignment 1

CS 1027
Computer Science Fundamentals II

Remember **you must do** all the work on your own. **Do not copy** or even look at the work of another student. All submitted code will be run through similarity-detection software.

Submission (due Thursday, October 5 at 11:55 pm)

Assignments must be submitted to Gradescope, not on OWL. If you are new to this platform, see these instructions on submitting on Gradescope.

Rules

- Please only submit the files specified below.
- Do not attach other files even if they were part of the assignment.
- Do not upload the .class files! Penalties will be applied for this.
- Submit the assignment on time. Late submissions will receive a penalty of 10% per day.
- Forgetting to submit is not a valid excuse for submitting late.
- Submissions must be done through Gradescope. **If your code runs on Eclipse but not on Gradescope, you will NOT get the marks! Make sure it works on Gradescope to get these marks.**
- You are expected to perform additional testing (create your own tester class to do this) to ensure that your code works for a variety of cases. We are providing you with some tests but we may use additional tests that you haven't seen for marking.
- Assignment files are NOT to be emailed to the instructor(s) or TA(s). They will not be marked if sent by email.
- You may re-submit code as many times as you wish, however, re-submissions after the assignment deadline will receive a late penalty.

Files to submit

- Sudoku.java
- UniqueDiagonalSudoku.java

Grading Criteria

Total Marks: 20

15 marks	Autograder Tests (some tests are hidden from you)
5 marks	Non-functional Specifications (code readability, comments, correct variables and functions, etc.)