CSIS215 Object Oriented Programming **Project 1** **Spring 20-21**

The aim of this project is to review the first version of the same project and implement it using a class.

In this project, the class, the methods and the entire code must be documented:

- An introductory documentation showing the title of the project, the date, the author, the purpose, the list of the methods with brief description.
- Each method has to be preceded by a multi-line comment stating clearly: the **purpose** of the method, the **pre-condition** (what it takes as parameter(s) and if any precondition(s) should be respected in order to make the method functioning as it should be and the **post-condition** what will happens when the method ends and if there is a returned value.
- Variables, initial values, important code blocks must be also documented.

Write a code that respects the code standards of the Java programming language regarding:

- Variables' and method' names
- Space use
- Use of empty lines,
- Indentation
- ….

Your project is assessed based on the rubrics that you are encourage to download from the Moodle course page and keep it in mind while developing your code.

Define a class **Sudoku** that has the following data fields:

- **initialGrid**: reference to a 2D array that represents the initial gird of the game. 0 value in an element (cells) of this array means that element (cell) was not assigned at the beginning. The player cannot change the values of these elements (cells). The player has to assign a value for these unassigned elements (cells) according the rules described in the method **addValueInACell**.
- **nbrOfErrors**: represents the number of occurred errors committed by the player and initialize it to zero. An error occurs when the player adds a value that already exists in the same row, or same column or in the same block. When the player commits more errors than the number of allowed errors (**nbrOfAllowedErrors**), the game ends with failure.
- **nbrOfAllowedErrors**: represents the number of errors allowed per game. We suggest to initialize it to 5.
- **emptyElementsArray:** reference to a 2D array with elements having the value 0 if its corresponding element (cell), in the **initialGrid** is to be input by the user and the value 1 if its value is initially set.
- **nbrOfMissingDigits**: an integer that represents the number of digits to be guessed by the player. When its value becomes 0, the player wins the game. This data field is assigned by the member method **countMissedValues**.

The class has the following member methods (for all these methods, row and col are between 0 and 8):

1. The public constructor **Sudoku** that takes a reference to a 2D array of integer values (between 1 and 9). This constructor must create the 2d array of integers, referenced by the data field **initialGrid**, with the same dimensions as its parameter. It also assigns the values of parameter array to the elements of the array **initialGrid**.

2. The private member method **displaySudoku** that displays the filled elements of the data field **initialGrid** in a tabular format. Empty elements should be replaced by spaces. The output of this method should be the same as follows:

	col1	col2	col3	col4	col5	col6	col7	col8	col9
Row1	9		3	5	6		7		4
Row2		5	4	1		7		8	6
Row3	6	8	7			2	5	1	
Row4			5	9	2		1	7	
Row5	7	2		8		1	9	4	3
Row6		9			7	3	6		2
Row7	1		2		3	4	8		
Row8	4		9	6					1
Row9	5	3	8	2		9		6	7

2. The private member method **generateEmptyElementsArray** that returns a 2D array that has the elements equal to 1 if tis corresponding elements in the parameter is different from 0, otherwise equal to 0.

   Assume that the parameter's values satisfy the constraints of values of Sudoku game: all values are between 0 and 9 and there is no repeated value in rows, columns and blocks.

3. The private member method **countMissedValues** that returns the number of the 0 values in **initialGrid**. In fact, the returned value represents the number of values that must be guessed by the player.

4. The private member method **int addValueInACell** that receives a value **num** to be added and the coordinates **row** and **col** of the cell of **initialGrid** in which the value is to be stored. **Assume** that the received coordinates **row** and **col** are valid (between 0 and 8) and the value (between 1 and 9) to be added in the element with the same coordinates in the **intialGrid. T**he validation of the arguments of this method must be done in the **runGame** method.

   The method **addValueInACell** calls the methods **existInRow**, **existInCol** and **existInBlock** that check the unicity of this value in the related row and the related column and in the related block (3 × 3). If the value **num** doesn't exist in the same row and the same column and the same block, it is stored in the cell **initialGrid[row][col]**, otherwise no change occurs in the array **initialGrid**.

   The returned value is:

   - 0: if the value **num** is stored in a cell that wasn't assigned yet by the user.
   - 1: if the value **num** is stored in a cell that was previously assigned by the user and that

the user wants to change its value.

- 2: if the value **num** exists already in another cell of the same row.
- 3: if the value **num** exists already in another cell of the same column.
- 4: if the value **num** exists already in another cell of the same block.

5. The private member method **existInRow** that checks if its parameter **num** exists or not in the row **row** of **initialGrid**.

6. The private member method **existInCol** that checks if its parameter **num** exists or not in the column **col** of the parameter **initialGrid**.

7. The private member method **existInBlock** that checks if its parameter **num** exists or not in the block related to the parameters **row** and **col**. This method must calculate internally the coordinates of the upper-left corner of the block to witch the element belongs. (Hint: use the operator modulo %).

8. The method **runGame** that should do the following:

   - Variables:
     1. **theValue**: declare this variable of type int to read in it the value to be added to the grid.
     2. **rowNumber**: declare this int variable that represents the row number entered by the user. The row number must be between 1 and 9.
     3. **colNumber**: declare this int variable that represents the column number entered by the user. The column number must be between 1 and 9.
     4. **resultOfAddValueInACell**: declare this int variable to store the return value of the method **addValueInACell**.

After declaring the above-mentioned variables, the method **runGame** must:

1. call the method **generateEmptyElementsArray** and assign the data field reference **emptyElementsArray.**
2. call the method **countMissedValues** and assign the data field **nbrOfMissingDigits.**
3. display the array **initialGrid**,
4. in a loop, prompt the user to
   - 4.1. enter the row number until a valid value between 1 and 9 is entered.
   - 4.2. enter the column number until a valid value between 1 and 9 is entered.
   - 4.3. check that the cell is not already assign ed in the initial grid. If it is, repeat steps 4.1 and 4.2 must be repeated until valid values are entered.
   - 4.4. enter the value to be added to the grid until a valid one between 1 and 9 is entered.
   - 4.5. use the entered values to call the method **addValueInACell**
   - 4.6. act according the returned value of the method **addValueInACell** to either update the variable(s) that need to be updated or display an appropriate message that shows the user his error (same value in the same row or column or block) and update the variable(s) that need to be updated.
   - 4.7. test if the number of committed errors is greater than the number of allowed errors, or all the missed values are added in the right way and display the appropriate message

For your information, you can find, below, the initial grid and its corresponding **emptyElementsArray** and the final solution that you have to use in order to test your code.

The initial grid:

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

The grid for given values:

1	0	1	1	1	0	1	0	1
0	1	1	1	0	1	0	1	0
1	1	1	0	0	1	1	1	0
0	0	1	1	1	0	1	1	0
1	1	0	1	0	1	1	1	1
0	1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	0	0
1	0	1	1	0	0	0	0	1
1	1	1	1	0	1	0	1	0

The solution

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

Sample Run:

```
      col1  col2  col3  col4  col5  col6  col7  col8  col9
Row1  9           3     5     6           7           4
Row2        5     4     1           7           8
Row3  6     8     7                 2     5     1
Row4              5     9     2           1     7
Row5  7     2           8           1     9     4     3
Row6        9                 7     3     6           2
Row7  1           2           3     4     8
Row8  4           9     6                             1
Row9  5     3     8     2           9           6     7
Enter the row(between 1 and 9): 10
Invalid row number.
Enter the row(between 1 and 9): -5
Invalid row number.
Enter the row(between 1 and 9): 1
Enter the column(between 1 and 9): 56
Invalid column number.
Enter the column(between 1 and 9): -7
Invalid column number.
Enter the column(between 1 and 9): 45
Invalid column number.
Enter the column(between 1 and 9): 1
The element[1[1] is initially assigned.
Enter the row(between 1 and 9): 1
Enter the column(between 1 and 9): 2
Enter the value you want to add(between 1 and 9): 2
The value exists in the same column
Number of errors is 1 out of 3

      col1  col2  col3  col4  col5  col6  col7  col8  col9
Row1  9           3     5     6           7           4
Row2        5     4     1           7           8
Row3  6     8     7                 2     5     1
Row4              5     9     2           1     7
Row5  7     2           8           1     9     4     3
Row6        9                 7     3     6           2
Row7  1           2           3     4     8
Row8  4           9     6                             1
Row9  5     3     8     2           9           6     7
Enter the row(between 1 and 9): 1
Enter the column(between 1 and 9): 2
Enter the value you want to add(between 1 and 9): 7
The value exists in the same column
Number of errors is 2 out of 3
```