Course code: CS 201
Course Name: Java 1

Project description

Select one of the following projects, the project will be submitted in three phases.

Phase one (select project):

Select one of the following projects, and submit an initial report with the following part

- Project title
- Abstract
- Team members

Phase two (submit the code):

Print the code and test your running code with your teaching assistant.

Phase three (oral Exam):

Create a final report with the following parts:

- Abstract
- Introduction (Problem definition, how you tackled it in your project).
- Definitions of technical terms used.
- Algorithm or flowchart of your code.
- UML class diagram
- Conclusion: Points of strengths and weaknesses for your implemented solution.
- References in IEEE conference format

Bring **final report** and the **printed code** checked from your teaching assistant and submit them to your instructor to discuss your solution.

Due Dates:

Phase 1	29/11/2018
Phase 2	During the week 10/12/2018 – 15/12/2018
Phase 3	A day Will be assigned during the week 15/12/2018 – 20/12/2018

List of projects:

Project 1 (calculate student grades):

Maximum team members: 3

Write a complete program that reads and displays information about the students registered in a course. The program contain a class names **CsStudent** which holds the following information about the students (assignments grades, bonus grades, midterm grades and final exam grades).

Attributes of the class:

- First name
- Last name
- Id
- List of assignment grades out of 100
- List of bonus grades out of 100
- Midterm exam grade out of 100
- Final exam grades out of 100
- Final grade out of 100

Methods of the class:

- Constructor
- CompareTo : it compares the student object with the other passed object based on their full name, it returns, -1, 0, +1 if the full name of the student object occurs before, equal or after the full name of the other student.
- ComputeFinalGrade: which computes the final grade for the student according to the following formula:
  *Final grade= 0.5*final exam grade+0.3*Average assignment garde+0.2*Midterm grade*

Note:

- The lowest assignment grade is not counted in the average, such that if there are 10 assignments, only the best 9 assignment grades are included in the average.
- The bonus points affect the final grade in the following manner:
  - 0-2 bonus points -> No change
  - 3-4 bonus points -> final grade increases by 1
  - 5 bonus points -> final grade increases by 2
  - 6 or more points -> final grades increases by 3

Bonus part:

Create a class names **LoadeStudents**, which contains only one static method called load(), which opens a file whose path is passed to it as parameter and returns an ArrayList of CSStudent objects created with the information read from the file.

In the file the information associated with each student is listed on different line, within a line. The information is listed in the following order:

- Last name
- First name
- Id
- Midterm garde
- Final grade
- Pairs of numbers representing assignment and bonus grades (assume there are only 6 pairs)

These values are separated by white spaces in a single line.

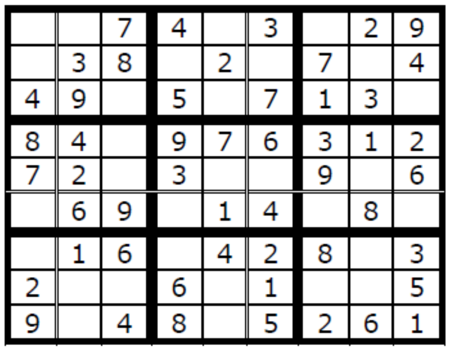
Example of a single line in the file:

Aly Mohamed 123456789 95 98 100 1 98 0 85 1 99 1 98 0 100 1

<u>Project 2 (</u>Sudoku):

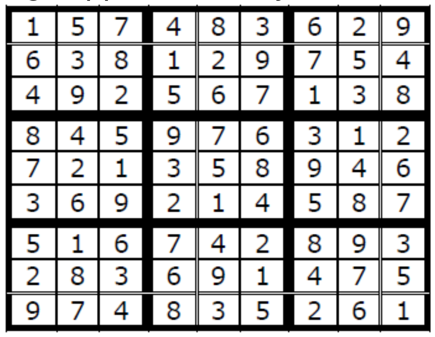
Maximum team members: 5

In the last several years, a new logical puzzle from Japan called Sudoku has become quite a hit in Europe and the United States. In Sudoku, you start with a 9x9 grid of numbers in which some of the cells have been filled in with digits between 1 and 9, as shown:



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

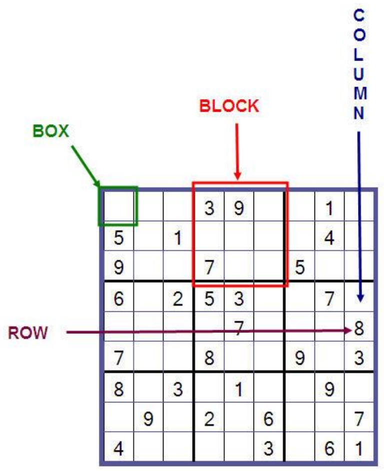
Your job in the puzzle is to fill in each of the empty spaces with a digit between 1 and 9 so that each digit appears exactly once in each row, each column, and each of the smaller 3x3 squares, as shown:



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

Each Sudoku puzzle is carefully constructed so that there is only one solution. Suppose that you wanted to write a class called **Sudoku**, which displays a Sudoku, ask the player to solve the displayed Sudoku, and checks the player solution.
Methods of the class:

- Constructor: takes as input a 2D array for the Sudoku game, and created Sudoku object.
- Display: prints the Sudoku on the screen.
- Play: allows the player to solve the displayed Sudoku, by entering the row number, column number and value, the method will continue to ask the player to enter the missed parts until all the empty cells are filled or the player enters -1 to stop playing.
- Check: this method checks whether the player solution is valid according to the following rules:
    - Each column must contain all of the numbers 1 through 9 and no two numbers in the same column of a Sudoku puzzle can be the same.
    - Each row must contain all of the numbers 1 through 9 and no two numbers in the same row of a Sudoku puzzle can be the same.
    - Each block must contain all of the numbers 1 through 9 and no two numbers in the same block of a Sudoku puzzle can be the same.



Bonus part:

Create a new class called **GenerateSudoku** which contains only one static method called generate(),which randomly generates a Sudoku puzzle taking in consideration that there is only one solution and returns an object of type Sudoku.

Project 3 (Stack):

Maximum team members: 4

Design a class called **Stack** for modeling a stack, a stack is a data structure used to store a collection of objects. Individual items can be added and stored in a stack using a push operation. Objects can be retrieved using a pop operation, which removes an item from top of the stack, applying the rule LIFO (last in first out).

Methods of the class:

- Constructor
- isEmpty : returns true if the stack is empty
- peek : returns the value of the element on the top of the stack.
- push: adds an element to the stack.
- pop : removes and returns the element on the top of the stack.
- getsize: returns the number of elements in the stack.
- sortAscending: sort the stack in ascending order with the help of 2 extra stacks.

Bonus:

Create a class names **LoadeStack**, which contains only one static method called load(), which opens a file whose path is passed to it as parameter, reads list of numbers from a file and push it to the stack and returns an object of type Stack.