**Exercise 1:**



Class ***Researcher***

* + Attributes:
    - ***name***: the name of the researcher.
  + Methods:
* ***Researcher (name:*** *String****)***: constructor
* ***getSalary()***: this method does the following:
  + ***For Faculty***: it returns the salary which is computed as follows:

*salary = 5600 SAR + (500 \* the number of years of experience).*

* + ***For Student***: it returns the salary which is computed as follows: *salary = 300 \* gpa.*

Class ***Faculty***

* + Attributes:
    - ***position***: the position of the faculty.
    - ***nbYearsOfExperience:*** the number of years of experience of the faculty.
  + Methods:
* ***Faculty (name:*** *String****, position:*** *String****, nbYears:*** *int****)***: constructor.
* ***getNbYearsOfExperience ():*** returns the number of years of experience of the Faculty.

Class ***Student***

* + Attributes:
    - ***gpa***: the gpa of the student.
  + Methods:
* ***Student (name:*** *String****, gpa:*** *double****):*** constructor.
* ***getGPA():*** returns the gpa of the student.

**QUESTION**: Translate into Java code the classes ***Researcher*** and ***Faculty***.

**Exercise 2:**

Let’s consider the same Researcher class and its subclasses described in exercise 1.



Class ***Project***

* + Attributes:
    - ***name***: the name of the project.
    - ***duration***: the duration of the project.
    - ***budget***: the budget of the project
  + Methods:
* ***Project(name:*** *String****, duration:*** *int****, budget:*** *double****, size:*** *int****)***: constructor
* ***addResearcher (r:*** *Researcher****):*** this method adds the researcher ***r*** to the project. It raises the following:
  + ***ArrayIndexOutOfBoundException*** if the array ***arRes*** is full.
  + ***Exception*** with the following message “*Not allowed to join a project.*” if ***r*** is a Student with a GPA less than 2.0.
* ***addFromArray(ar:*** *Researcher[ ]****, s:*** *double****):*** this method reads researchers from the array ***ar*** and adds to the project the Faculty members having a salary greater than ***s***. It returns the number of Faculty members successfully added to the project.
* ***getFaculty():*** this method returns the most experienced Faculty member (the Faculty having the highest number of years of experience).
* ***getName()*** and ***getBudget():*** getters of the class Project.

**QUESTION**: Translate into Java code the class ***Project.***

**Exercise 3:**

Let’s consider the same Project class described in exercise 2.



Class ***LinkedListOfProjects***

* + Methods:
* ***LinkedListOfProjects*** ***( )***: constructor.
* ***getAverageBudget ( )*:**this method returns the average budget of the Projects.
* ***remove (p:*** *Project***):**this method removes the first Project having the same name as ***p***. If the project is successfully removed it returns true. Otherwise, it returns false.
* ***size( ):***this method returns the number of projects of the list.
* ***get (i:*** *int****):*** This method returns the Project that is at position ***i***.

Notice that the project in the first Node corresponds to position 0. The project in second Node corresponds to position 1 etc.

**QUESTION**: Translate into Java code the class ***LinkedListOfProjects.***

**Exercise 4:**

Let’s consider the same ***LinkedListOfProjects*** class described in exercise 3.



Class ***College***

* + Attributes:
    - ***name***: the name of the College.
  + Methods:
* ***College(name:*** *String****)***: constructor
* ***HighestBudgetProject (* ):** this method returns the project having the maximum budget .
* ***saveIntoFile(fileName:*** String, ***b***: double**):** this method saves into the file ***fileName*** all projects having a budget greater than ***b***. It returns the number of saved projects.
* ***readFromFileAndRemove(fileName:*** String**):** this method reads projects from the file ***fileName*** and removes them from the list. It returns the number of removed projects.

**QUESTION**: Translate into Java code the class ***College.***