Langara
THE COLLEGE OF HIGHER LEARNING.

Department of Computing Science & Information Systems
CPSC 1181

Lab#2
May 25, 2022

Objectives:

Use ArrayList object.

Design and implement classes

Preparation:

Read chapter 7 of the textbook.

Due date:

Due Date: 11:00 PM on Wednesday, Sep 28, 2022

Where to upload:

Zip your files into yourstudentID.zip where yourstudentID is your student number, and upload it to dropbox in D2L.

Part A: Tutorial

Do Tutorial2 . Due date 11:00 PM Sep 22, 2022. Late submission will not be accepted.

Part B:

Develop a program that stores and manages students of a course. Course may have many students. This lab assignment is based on your previous lab assignment, and it should support the following operations.

Add a new student to course.

Delete students from course.

Look up an existing student.

Add quiz grades to students. Each quiz has a different marking scheme.

For simplicity, assume once the quiz is assigned, it cannot be modified or deleted.

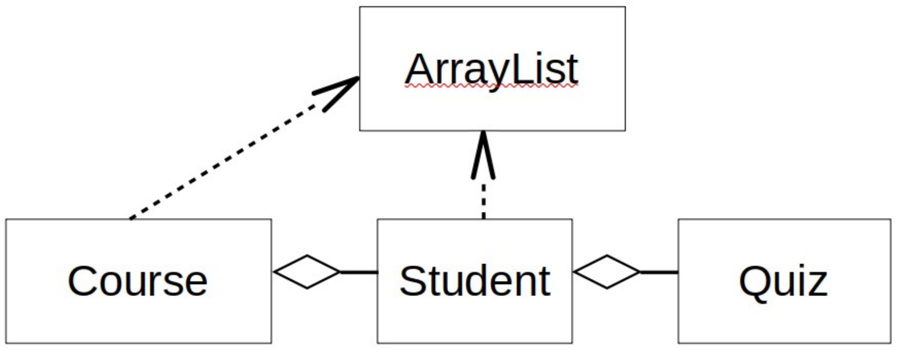
Find a student with the highest grade average.

Find average of the course.

Part1 1: [35 ,marks]

Create the following three classes:

The UML diagram of set of the classes is shown below:



Students.java : Modify class Student you have already created in Labt #1. Remove class Address from your implementation since you are not going to use it in this lab assignment.

Each Student object stores the name, surname, student number, loginId, and quizzes.

- The class Student assigns a unique StudentId (student number) for each student starting from 10000001 during construction (Refer to example BackAccount in your notes).
- Modify the Student constructor as:
  `public Student(String name, String surname){   }`
  Note that we are not passing a student number to each student during creation. The class itself assigns a unique student number to each student.
- Modify addQuiz() method to
  `public void addQuiz(double maxGrade, double studentGrade){}`
  Note that each quiz has a different scale

Student class should provide following public interface:
```
public Student(String name, String surname);
public void setName(String firstName, String surname);
public String getName();
public long getStudentNumber();
public String getLoginId();
String getInfo(){
      return name+","+surname+"("+studentNumber+","+loginId+")";
}
public String toString(){
      return "[name: "+name+", surname:"+surname+
        "(StudnetNumber :"+studentNumber+", loginId:"+loginId+")]";
}
```

To implement following two methods, refer to part 2 of the assignment

```
public void addQuiz(double maxGrade, double grade);
public double getAverage(); // returns the average of the quiz for the student
```

Course.java : This class uses an ArrayList of Student objects internally to manage Student's objects.

The class Course should provide the following public interface :

```
// Constructor
public Course( )

// Add a new student to the list of the students, and return reference of the
// newly created student. In case of failure it return null
Student addStudent (String name, String familyName)

// Return a reference of the student. In  case student is not found it return null
Student findStudent (long studentNumber);



// Remove the student from the list and return reference of the student
// being removed. In case student is not found it returns null
Student deleteStudent (long studentNumber);
```

To implement following two methods, refer to **part 2** of the assignment

```
// adds a quiz to a student given student with student Numbetr
// maxGrade: the grade of the quiz
// studenGrade: the grade student achieved
// return true is case succeed, or false in case of failure
boolean addQuiz(long StudentNumber, double maxGrade, double studentGrade);

// Return reference of the student with the highest average quiz
Student findTopStudent();

// Return the average of the course. (average of the average of the students
grades)
double getAverage ( );
```

CourseTester.java: This class provides test routines to test class **Course**. You should provide enough test cases to be sure that class **Course** works correctly. However, unlike Lab#1, you do need to document the test cases (no extra testCase.doc for this part).

Part2 : [25 marks]

Unlike lab1, each quiz has its own max value. In lab1 all quizzes were out of 10.

For example if the quiz grades of a student are

12 out of 15 for the first quiz (maxGrade = 15, studentGrade = 12),
9 out of 12 for the second quiz,
4 out of 8 for the third quiz,
then the average of the quiz for this student is calculated from the following formula:

$$studentAvg=\frac{\sum \frac{sutudentGrade}{maxGrade}}{number\ of\ quizzes}=\frac{\frac{12}{15}+\frac{9}{12}+\frac{4}{8}}{3}*100=\frac{0.8+0.75+0.5}{3}*100=0.683*100=68.3$$

Quiz.java :

Design and implement class **Quiz** with two instance variables:

```
double maxGrade;        //  quiz max grade
double studentGrade;    //  the grade student achieved in the quiz
```

Modify Student class to :

```
public class Student{
    private ArrayList<Quiz> quiz;
    …
}
```

Now each student has an ArrayList of Quiz as its instance field, and all quizzes for each student are stored in the ArrayList.

Implement the following methods of the class Student.

```
public void addQuiz(double maxGrade, double grade);
public double getAverage(); // returns the average of quiz for the student
```

```
and the following methods of the class Course
boolean addQuiz(long StudentNumber, double maxGrade, double studentGrade);
Student findTopStudent();
double getAverage ( );
```

Develop appropriate constructors and methods. For simplicity, assume that once the grade of a quiz is assigned it cannot be modified or deleted.

Add test cases to **CourseTester.java** to test this part.

TOTAL MARK: 60

Bonus: [15 marks]

Preparation: Redirected input/output

You need to enter the test data every time you are going to debug your program. Your program is going to crash many times until you fix all the problems. Entering the sample data every time you want to test the program is a nightmare. However, there is an alternative for this problem. Save your test data in a text file, and let the program reads the sample data form your file instead of reading them from keyboard without modifying your program. Note that this is a different functionality to read data from a file. We call it redirect commands.
Use redirect commands **<** and **>** to redirect the input and output.
< redirects the input
> redirects the output

Download program redirect.zip and save it in your local drive, and then run it. You should enter zero for both x and y to terminate the program.
Sample run of the program:

```
input X : 23
input Y : 45
X + Y = 68.0

input X : 56
input Y : 22
X + Y = 78.0

input X : 0
input Y : 0
X + Y = 0.0
```

Now use redirect command to redirect input of the program shown below:

```
java InOut < input.txt
```

Output of the program:

```
input X : 23.0
input Y : 45.0
X + Y = 68.0

input X : 56.0
input Y : 22.0
X + Y = 78.0

input X : 0.0
input Y : 0.0
X + Y = 0.0
```

Note that the program takes input from input.txt file, and we do not need to use keyboard to input data every time we run the program.
To capture the output of the program, run it as

```
java InOut < input.txt  > ouput.txt
```

The program reads data from input.txt file, and saves the output in output.txt file.

What to do:

Now you are sure that the class Course works correctly. In this section you are going to add a runner program to prompt user, take commands, call appropriate methods, and display the result.

The class CourseRunner.java uses a loop to prompt user for an option, takes input from keyboard, calls the Course methods, and displays the result. You do not need to verify if input is valid. Assume user inputs valid inputs.

Sample run of the class:

```
College directory commands:
1. add a new Student
2. find Student
3. delete Student
4. add quiz
5. find student with the highest quiz score
```

```
6. get course average
7. quit
select option: 1
Add a new Student:
student's name: gary
student's surname: zagreb
student added: gary,zagreb(10000001,gzagr01)

Course directory commands:
1. add a new Student
2. find Student
3. delete Student
4. add quiz
5. find student with the highest quiz score
6. get course average
7. quit
select option: 2
Find student:
student's number(id): 10000340
student not found!
```

*The user response are shown in red font for clarification purpose only. In your program, all fonts are the same.

Download CourseRunner.java , input.txt , and output.txt files.
The class CourseRunner.java file is partially developed. Note that it will not compile until you develop Course class. Run the CourseRunner using redirected commands as shown below:

```
java CourseRunner < input.txt > output.txt
```

The file input.txt file is provided. Your program should created the same output.txt file using the sample input.txt file.
Note that your marker has his own file to use instead of input.txt file to test your program.

What to submit

1. Comment your classes and methods appropriately using the javadoc notation.
2. The source files: Quiz.java, Student.java, Course.java, CourseTester.java, CourseRunner.java, your own sample input.txt file, the output.txt file generated by your sample input.
3. Comments about your assignment if needed. These comments are not the comments documenting your code but rather something you need to convey to us about your assignment
4. Submit to D2l Dropbox: lab2