Langara
THE COLLEGE OF HIGHER LEARNING.

Department of Computing Science & Information Systems
CPSC 1181-002

Lab#2
Sep 17, 2019

Objectives:

Use ArrayList object.

Design and implement classes

Preparation:

Read chapter 7 of the textbook.

Due date:

Due Date: 11:00 PM on Monday September 23, 2019

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

Download tutorial 1

You should finish tutorial and upload your files by 11:00 PM today.

Part B:

Develop a program that stores and manages students of a course. Course may have many students. The 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:

You should create the following five classes:

Quiz.java :

Unlike Lab1, each quiz has its own scale.

Design and implement a class called Quiz. Each quiz has two instance variables:

```
double scale;           //  quiz scale
double studentGrade;    //  the grade student achieved in the quiz
```

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

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. Quizzes are arranged using an ArrayList.

- Each student has an ArrayList of Quiz.
- The class Student assigns a unique StudentId (student number) for each student starting from 10000001 during construction.
- Modify the Student constructor as:
  `public Student(String name, String surname){   }`
  Note that we are not passing a studentId to each student during creation. The class itself assigns a unique studentId to each student.
- Modify addQuiz() method to
  `public void addQuiz(double scale, 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 void addQuiz(double scale, double studentGrad);
public String getName();
public long getStudentNumber();
public String getLoginId();
String getInfo(){
      return name+","+surname+"("+studentId+","+loginId+")";
}
public String toString(){
      return "[name: "+name+", surname:"+surname+"(StudnetId:"+studentId+",
      loginId:"+loginId+")]";
}
```

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
Studdent findStudent (long studnetId);

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

// adds a quiz to a student given studentId
// scale: the scale of the quiz
// studenGrade: the grade student achieved
// return true is case succeed, or false in case of failure
boolean addQuiz(long StudentId, double scale, 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 Student. You should provide enough test cases to be sure that class Student works correctly. However, unlike Lab#1, you do need to document the test cases (no extra testCase.doc for this part).

Part2 :

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 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:

```
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: 1
Add a new Student:
student's name: hossein
student's surname: darbandi
student added: hossein,darbandi(10000001,hdarb01)

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!
```

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

TOTAL MARK: 60