Lab 5: Computing Course Grades

Due: 18:00, 08 Mar 2012 (Fri) Full marks: 100

Introduction

In a university course, the grading criteria for three types of students are listed below:

1. *Normal students* are assessed using five assignments (1 to 5). They fail if their average assignment score is below 50.
2. *Graduate students* are assessed using five assignments (65%) and also a quiz (35%). They fail if the total score is below 60.
3. *Part-time graduate students* are assessed using three (out of five) assignments with highest scores (75%) and also a quiz (25%), they fail only if the total score is below 40.

Please take note that the full marks for each assignment and quiz are 100. You will write four classes: **Student**, **GraduateStudent**, **PartTimeGraduateStudent,** and **GradeCalculator** to read the assignments and quiz (if any) scores of the students from a data file and compute whether each student passes or fails.

Program Specification

Create a new project in NetBeans named **`GradeCalculator`**; you will apply inheritance and polymorphism to write a program to compute grades of different students according to the rules above. The following gives the details of the four classes: **`Student`**, **`GraduateStudent`**, **`PartTimeGraduateStudent`**, and **`GradeCalculator`**. In all of them, you are ***not allowed to add any extra* public *contents***.

`Class Student`

This class models normal students in the course.

Instance Variables

- **`private String sid;`**
  A string denoting the 10-digit student ID of the student

- **`private double[] assignment;`**
  An array of `double` for storing the five assignment scores of the student

- **`private double totalScore;`**
  The total score of the student

- **`private String courseGrade;`**
  A string denoting the course grade (i.e. "Pass" or "Fail") of the student

Constructor and Instance Methods

- **`public Student(String id)`**
  The parameter `id` is the student ID of the student. You should use it to initialize the corresponding instance variable `sid`. You can assume that: (1) `id` *contains only digit characters* and (2) *its length is at most 10*. If the length is smaller than 10, you should add '0's to its front so that its length becomes 10. For example, "09876543" becomes "0009876543". You also have to create an array of size 5 for `assignment` and initialize `courseGrade` as "****" and `totalScore` as 0.

- **`public String getID()`**
  This method returns the student ID of the student.

- **`public double getAssignmentScore(int i)`**
  This method returns the `i`-th assignment score of the student. You can assume that the parameter `i` *is always from 1 to 5 inclusively*.

- **`public void setAssignmentScore(int i, double score)`**
  This method sets the `i`-th assignment score of the student as `score`. If `score` is smaller than 0, then set it as 0. If `score` is greater than 100, then set it as 100. You can assume parameter `i` *is always from 1 to 5 inclusively*.

- **`protected void sortAssignmentScore()`**
  This method sorts the assignment score in ascending order. This method is *useful to its subclasses*. **Hints**: the standard Java class **`Arrays`** is useful[1], use the following example as a guide on how to sort an array of numbers.

```
double[] numbers = { 4.5, 2.3, 3.7, 1.6 };
Arrays.sort(numbers);
System.out.println(Arrays.toString(numbers));
```

[1] See API specification http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html

- **`public double getTotalScore()`**
  This method returns the value of the instance variable `totalScore` of the student.

- **`protected void setTotalScore(double total)`**
  This `protected` method sets the instance variable `totalScore` as parameter `total`. This method is *useful to its subclasses*.

- **`public void computeCourseGrade()`**
  This method performs two tasks: (1) compute the average assignment score (i.e. based on the rule of normal student) and use it to update the instance variable `totalScore`, and (2) use the total score to update the instance variable `courseGrade` to either "Pass" or "Fail".

- **`public String getCourseGrade()`**
  This method returns the value of the instance variable `courseGrade` of the student.

- **`protected void setCourseGrade(String grade)`**
  This `protected` method sets the instance variable `courseGrade` to `grade`. This method is *useful to its subclasses*.

Class GraduateStudent extends Student

This class is a subclass of **`Student`** and models a graduate student in the course.

Instance Variables

- **`private double quiz;`**
  The quiz score of the graduate student

Constructor and Instance Methods

- **`public GraduateStudent(String id)`**
  This constructor (1) calls the constructor of its superclass and (2) initializes the `quiz` to 0.

- **`public double getQuiz()`**
  This method returns the value of the instance variable `quiz` of the graduate student.

- **`public void setQuiz(double score)`**
  This method sets the instance variable `quiz` to parameter `score`.

- **`public void computeCourseGrade()`**
  This method *overrides* the method in its superclass. It performs two tasks: (1) compute the total score using the assignment and quiz scores (i.e. based on the rule of graduate student) and update the instance variable `totalScore` in its superclass, and (2) use the total score to update the instance variable `courseGrade` in the superclass to either "Pass" or "Fail".

Class PartTimeGraduateStudent extends GraduateStudent

This class is a subclass of **`GraduateStudent`** and models a part-time graduate student.

Constructor and Instance Methods

- **`public PartTimeGraduateStudent(String id)`**
  This constructor calls the constructor of its superclass.

- **`public void computeCourseGrade()`**
  This method *overrides* the method in its superclass. It performs two tasks: (1) compute the total score using the assignment and quiz scores (i.e. based on the rule of part-time graduate student) and update the instance variable `totalScore` in the superclass, and (2) use the total score to update the instance variable `courseGrade` in the superclass to either "Pass" or "Fail".

Class GradeCalculator

This class models the process of computing the course grades for all the students in the course.

Instance Variables

- **`private int numStudents;`**
  An integer denoting the number of students in the course

- **`private Student[] students;`**
  An array of `Student` for storing all the students in the course

- **`private Scanner scanner;`**

  The scanner for reading data inputs

Constructor and Instance Methods

- **`public GradeCalculator(Scanner sc)`**

  The parameter `sc` is the scanner for reading data input. You should initialize (1) `scanner` as `sc`, (2) `numStudents` as `0`, and (3) `students` as an array of size 50 (i.e. the course can admit at most 50 students).

- **`public void computeResult()`**

  This method computes and prints out the total scores and pass/fail results of all the students. This method is given to you and should be exactly like this.

```
public void computeResult() {
    System.out.println("Results of " +
                numStudents + " students:" );
    for(int i = 0; i < numStudents; i++) {
        students[i].computeCourseGrade();
        System.out.println(students[i].getID() + " " +
                    String.format("%.2f",
                        students[i].getTotalScore()) +
                    " " + students[i].getCourseGrade());
    }
}
```

- **`public void readData()`**

  This method reads the data from the input file using the scanner. Each line of the input file denotes the information of a student; and contains seven or eight fields (delimited by spaces) depending on the type of the student.

  - The first field denotes the student type, which is either "NOS", "GRS", or "PTS", denoting normal, graduate, or part-time graduate students respectively.
  - The second field is the student ID.
  - The third to seventh fields are the scores of assignments 1 to 5.
  - The eighth field exists for graduate students only (including part-time), denoting the quiz score.

  You can assume that (1) *the first field must be either "NOS", "GRS", or "PTS"*, (2) *the second field contains only digit characters the length is at most 10* and (3) *the remaining fields must be numbers*. For simplicity, you can assume the

students in the file are unique and the file contains at most 50 lines. This method performs four main tasks: (1) read the data from the input file, (2) create the **`Student`**, **`GraduateStudent`** or **`PartTimeGraduateStudent`** objects accordingly with the corresponding assignment and quiz (if any) scores, (3) update the array instance variable `students` to store the objects, and (4) update the instance variable `numStudents` according to the number of lines of data in the input file. **Hints**: read the *Java Tutorials* article on how to use the Scanner object to read data from a simple text file:

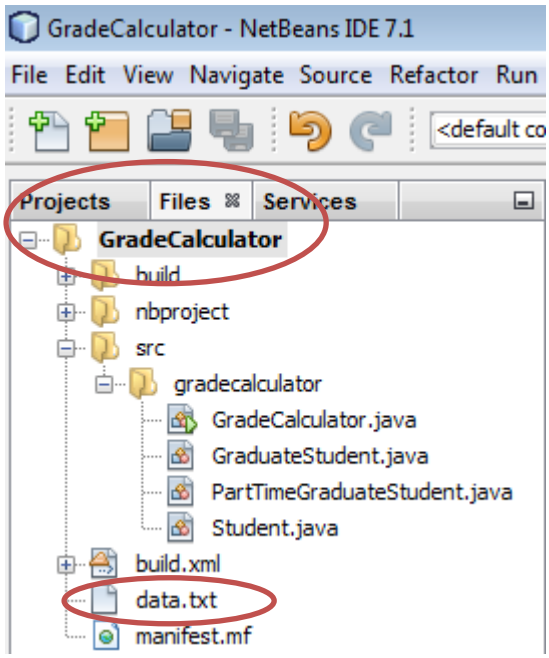
URL: http://docs.oracle.com/javase/tutorial/essential/io/scanning.html

- **`public static void main(String[] args)`**

This *static* main method is given to you and should be exactly like this.

```
public static void main(String[] args) {
    try {
        Scanner sc = new Scanner(new BufferedReader(
                             new FileReader("data.txt")));
        GradeCalculator gc = new GradeCalculator(sc);
        gc.readData();
        gc.computeResult();
} catch (FileNotFoundException exp) {
    System.err.println("Error, Input File Not Found!");
}
```

In NetBeans, please be reminded that you have to put the input file `data.txt` under the Project root folder. You can check this using "File Window" (Ctrl-2).



Sample Run

The following shows the content of the sample input file `data.txt`.

```
NOS 46709394 78 90 92 64 70.5
PTS 5201314 95 87 50 59 64 89
GRS 25277177 66 39 53 69 80 43
NOS 25266366 46 40 55 57 42.1
NOS 1234567890 95 80 91 92 82
```

The following shows a sample run of the program given the above input file.

```
Results of 5 students:
0046709394 78.90 Pass
0005201314 83.75 Pass
0025277177 54.96 Fail
0025266366 48.02 Fail
1234567890 88.00 Pass
```

The above sample data input file and sample run are also available in CU e-Learning System.

Submission

Preparing a ***zip file*** called `gradecalculator.zip`, which should contain only the four Java source files: (1) `Student.java`, (2) `GraduateStudent.java`, (3) `PartTimeGraduateStudent.java`, and (4) `GradeCalculator.java`. Submit the zip file to CU e-Learning System (i.e. the entry "**lab05 – Computing Course Grades**" under "**Lab Works**").

== END ==

