CS1083 Assignment #1 - Fall 2024

Due: Wednesday, 18 September before 4:30 pm in the Desire2Learn dropbox. (See submission instructions below).

The purpose of this assignment is:

- Draw a UML Diagram
- Program an ArrayList
- Review 2D Arrays

Grade Management System

You have been given the following description of a program needed to help a school manage the grades of their students. This assignment involves 3 objects and 1 interface.

The first object is the **`CourseMatrix`** object. This object stores 2, 2D arrays of varying types relating to a student's performance/completion of courses. The first 2D array should store the codes of the courses and the second array should store the GPA received for the courses (if -1, it means the student has not completed the course). You may assume that the columns of the 2D array are the terms and that every course code will be some series of capital letters followed by exactly 4 numbers. Both arrays should have accessor methods. You may assume that a student will take at most 5 courses per term and that if a course code is empty, then the student is not taking a full course load. Both arrays should be passed in as parameters to the constructor. For example, if a student studies for two terms, and takes CS1073 (3.7), CS1203 (4.0), and MATH1003 (2.5) in the first term, and they take CS1083 (3.3), CS1103 (3.0), CS1303 (4.0), MATH1013 (2.5), and MATH1503 (2.7), in the second term, this would be their matrices:

"CS1073"	"CS1083"
"CS1203"	"CS1103"
"MATH1003"	"CS1303"
""	"MATH1013"
""	"MATH1503"

3.7	3.3
4.0	3.0
2.5	4.0
-1	2.5
-1	2.7

The second object is the **`Student`** object. This object has multiple children (we will only be programming one for this assignment). The constructor of this object should take all instance variables except ID. All students should store a student ID (everyone should be automatically assigned a unique one starting at 1000), their name, and a course matrix. Students should not exist independently as only their children should be directly instantiated. This class must include accessors for the name, ID, and course matrix, but no additional methods are required. All types of students should implement the **`Gradable`** interface (see below).

The third object is the **`UndergradStudent`** object. The constructor of this object should take all instance variables except ID. Undergraduate students should have everything that a typical **`Student`** has; however, they should also have a degree program which is represented as a code: "CS" (computer science), "MATH" (mathematics), "POLS" (political science), etc. where each of these codes should match with the course codes from the **`CourseMatrix`** objects. To calculate an undergraduate student's GPA at this institution we will assume that all courses that match the student's degree program are worth 4 credit hours, while all others are 3 credit hours. If we assume that the student whose course matrix that we see above is a CS student, their GPA would be calculated as follows:

Total number of credit hours = 4 + 4 + 3 + 4 + 4 + 4 + 3 + 3 = 29
GPA (not-normalized) =
4 * 3.7 + 4 * 4.0 + 3 * 2.5 + 4 * 3.3 + 4 * 3.0 + 4 * 4.0 + 3* 2.5 + 3 * 2.7 = 91.5

GPA = 91.5 / 29 = 3.27…

Finally, **`UndergradStudent`** should have a **`toString()`** method that returns a string where the first line contains the student name, ID, degree program, and their overall GPA. The rest of the lines in this string should be the list of courses.

Finally, create a **`Gradable`** interface. This interface requires two methods (1) a **`calculateGPA()`** method which takes no parameters and returns a student's calculated GPA overall and (2) a **`listCourses()`** method which returns a String containing a list of all courses the students has taken.

For any student, the **`listCourses()`** method should return a String where each line of output should list the GPA they received in a **completed** course and the course code. The order here matters as all the courses for their first term should be printed before second, all second before third, and so on. Take time to consider which class the **`listCourses()`** method should be.

Before programming, draw a UML diagram either on paper or digitally and must be submitted as part of your PDF submission. This UML diagram should follow the rules we covered in class and be legible.

Driver Program for Testing

Write a driver program to test the classes you have written. The driver has the following requirements:

- create at least 4 undergraduate students with different courses, GPAs, etc.
- add all of them to a singular waitlist which is an ArrayList
- print all using a for-each loop

Your electronic submission (submitted via Desire2Learn) will consist of two files. Name your files YourName-fileName.extension, e.g. JohnSmith-as1.zip, JohnSmith-as1.pdf:

1. A single pdf file containing a listing of the source code for the **`Gradable`**, **`Student`**, **`UndergradStudent`**, **`CourseMatrix`**, and **driver classes**, and a discussion of each test case along with output captured from the terminal window. The pdf must also include the UML diagram.
2. A zip file containing your five Java classes.