CSE 1141 - COMPUTER PROGRAMMING I
Programming Assignment # 6
DUE DATE: 30/12/2017 - 23:00 (No extension)

In this homework, you are expected to implement a simple course registration program. As a student, you should be familiar with the process of registering for courses. A full simulation of the process of online registration would be very difficult. Therefore, a simplified version of the process will be simulated by your program in the sense of creating your own classes and having object interactions. In this homework, you are required to add `Student`, `Course`, `Faculty`, `RegisterInfo`, `AssignInfo` and `Test` classes into your implementation. The details of those classes are given below.

1. Implement a `Student` class with the following UML diagram.

	Student
-	name: String
-	surname: String
-	major: String
-	ID: int
-	year: int
-	registerList: RegisterInfo[]
+	Student(name: String, surname: String, major: String, ID: int, year: int)
+	addRegisterInfo(registerInfo: RegisterInfo): void
+	removeRegisterInfo(course: Course): boolean
+	printCourseList(): void
+	getTotalCredit(): int
+	getter/setter methods

- Students will attempt to register for courses.
- Each student should have a name, a surname, a major, a student ID, a year, and a list of courses attempted to be registered.
- `major` field should indicate the department of a student such as “Computer Engineering”.
- `year` field should indicate the student’s year of study. For example, the values of 1, 2, 3 and 4 should be used for freshman, sophomore, junior and senior students.
- Each student should keep the list of courses attempted to be registered and this information is kept in `registerList` array which is in type of `RegisterInfo`.
- Each student should be created with a given name, surname, major, ID and year values in the constructor.
- `addRegisterInfo` method should add the given `registerInfo` to the `registerList` of the student. If the `registerList` is `null` in the beginning, then create an array with the length of 1, and assign the given `registerInfo` to the first

index of the array. Otherwise, increase the length of the array by 1, add the `registerInfo` to the last index of the array.

- `removeRegisterInfo` method should remove the corresponding `registerInfo` (get this data from the argument `course`) from the `registerList`. You should decrease the size of `registerList` by 1 and copy the content of old `registerList` to the new `registerList`. Do not forget that the removed `registerInfo` can be located at any index of `registerList`.
- `printCourseList` method should print the courses which are attempted to be registered by the student. This method does not take any input. You should print the course list by traversing `registerList` array of the student.
- You should implement getter and setter methods for the private data fields of the class.

2. Implement a `Course` class with the following UML diagram.

Course	
-	department: String
-	name: String
-	credits: int
-	prereqYear: int
-	maxEnrollment: int
-	reservedSeats: int
-	studentList: Student[]
-	replacementList: Student[]
-	instructor: Faculty
+	Course (department: String, name: String, credits: int, prereqYear: int,
+	maxEnrollment: int, reservedSeats: int)
+	registerCourse(std: Student): RegisterInfo
+	assignInstructor(instructor: Faculty, force: boolean): AssignInfo
+	registerReplacementList(): void
+	printStudentList(): void
+	getter/setter methods

- Each course has a department, a name, course credits, a prerequisite year, a maximum number of enrollment, a number of reserved seats, a list of registered students, a list of students in replacement list (which is used for reserved seats), and an instructor.
- As a simplification of the concept of "prerequisites", each course has an integer value for the minimum student year that will allow a student to register for a course. For example, if a course called "Operating Systems" has a student year of 3, then only the students who are seniors (4) or juniors (3) will be allowed to register for that course.
- `maxEnrollment` indicates the quota of the course.
- `reservedSeats` indicates that a subset of `maxEnrollment` is reserved to register students based on instructor decisions.

    - For example, if `maxEnrollment` has the value of 20, and the `reservedSeats` has the value of 2; then, at most 18 students can be registered for that course. For remaining 2 positions, the instructor will decide which students in the `replacementList` will be registered for the course.
- A student will be allowed to register if the course is not full and the student meets the prerequisites.
- `studentList` should keep the currently registered students to the course.
- `replacementList` should keep the candidate students to register for the course.
- If the course does not have enough space for a student; then, the student is stored in `replacementList` array of the course. Once an instructor is assigned to a course, s/he will decide that whether the students in `replacementList` should be registered to the course based on the given number of reserved seats.
- Each course should be created with a given name, credits, prerequisite year, maximum number of enrollment (or quota), and the number of reserved seats values in its constructor.
- `registerCourse` method takes an input in type of `Student`, and try to register the corresponding student (`std`) to the course.
    - This method should register the students, if s/he meets prerequisite requirement and if there is enough space in the course quota.
    - It should register the students also from other departments if the rate of the currently registered students is less than the **70%** of the course capacity. Otherwise, you should reject this request. This rule is not valid for the students has the same major as the course department.
        - For example, suppose that there is a course with the capacity of 100. Until the number of registered students reaches the value of 69, there is no rule based on department check. However, if this value is greater than or equal 70; then, only the students having the same major as the course department can register to the course.
    - If the course is full and there is available place for `reservedSeats`, then add the student (even in the same major or not) to the `replacementList` of the course and the instructor will decide whether this student will be registered to the course or not.
    - In this method you should create a `RegisterInfo` object, fill the required data fields of the object and return its reference.
- `assignInstructor` method takes two inputs in type of `Faculty` and `boolean`, and assigns the corresponding instructor to the course.
    - In this method, you should assign the corresponding instructor to the course if the department names match.
    - If the `force` value is `false`, then do not assign the instructor to the course if the course has already an instructor assigned.
    - On the other hand, you should change the instructor with the given one, if the `force` value is `true`.

        - When you changed the instructor of a course, you should update the information in the corresponding `AssignInfo` object of the `AssignInfoList` array of the instructors.
    - After the assignment, this method should create an `AssignInfo` object, fill the required data fields of the object and return its reference.
- `registerReplacementList` method should register the students in replacement list based on the number of reserved seats of the course once an instructor assigned to the course.
    - Firstly, this method should search for the `replacementList` array and register the student in the list if s/he has a major in the department of the course as well as if there is available place in the course quota.
    - Secondly, if there are still remaining places in the course quota; then, search for the `replacementList` array one more time, and register the remaining students in the list even if they have different majors until the quota becomes full or the `replacementList` becomes empty.
    - You should update the size and content of `replacementList` and `studentList` after the method registers a student to the course. Indeed, you should copy some of the students in `replacementList` to the `studentList`. Additionally, you should update the `registerList` of the corresponding student.
    - This method can be invoked once an instructor is assigned to the course; otherwise, you should inform the user with an error message.
- `printStudentList` method should print the students registered to the course. This method does not take any input. You should print the student list by traversing `studentList` array of the course.
- You should implement getter and setter methods for the private data fields of the class.

3. Implement a `Faculty` class with the following UML diagram.

	Faculty
-	ID: int
-	name: String
-	surname: String
-	departmentName: String
-	assignInfoList: AssignInfo[]
+	Faculty(id: int, name: String, surname: String, departmentName: String)
+	withdrawAssignInfo(course: Course): boolean
+	addAssignInfo(assignInfo: AssignInfo): void
+	printCourseList(): void
+	getter/setter methods

- Each instructor has an ID, a name, a surname, a department name and a list of courses given by the instructor.
- Each instructor should keep the list of course assignment information in `assignInfoList` which is an array in type of `AssignInfo`.
- Each instructor should be created with a given ID, name, surname, department name values in its constructor.
- `withdrawAssignInfo` method takes a course value as argument and withdraws the corresponding course from the instructor's `assignInfoList`. You should decrease the size of `assignInfoList` by 1 and copy the content of old `assignInfoList` to the new `assignInfoList`. Do not forget that the removed course can be located at any index of `assignInfoList`.
- `addAssignInfo` method should add the given `AssignInfo` to the instructor's `assignInfoList`. If the `assignInfoList` is `null` in the beginning, then create an array with the length of 1, and assign the given `AssignInfo` to the first index of the array. Otherwise, increment the length of the array by 1, and add the `AssignInfo` to the last index of the array.
- `printCourseList` method should print the courses assigned to the instructor. This method does not take any input. You should print the course list by traversing `assignInfoList` array of the instructor.
- You should implement getter and setter methods for the private data fields of the class.

4. Implement a `RegisterInfo` class with the following UML diagram.

RegisterInfo	
- - - -	course: Course result: String registerID: int registerMessage: String
+	getter/setter methods

- `RegisterInfo` class is used to interact `Student` class with the `Course` class.
- Once a student attempts to register a course, the course object creates a new `RegisterInfo`, fills the data fields of it and adds it to the `RegisterInfoList` of the student in the `registerCourse` method.
- The data fields store the following information:
    - o `course` keeps the course that the student attempts to register for,
    - o `result` keeps the whether the registration request "APPROVED", "REJECTED" or "WAITING".
    - o `registerID` shows the information that the student is registered to the course in which order.

   - `registerMessage` stores a message such as “REQUEST APPROVED”, “REQUEST WAITING - REPLACEMENT LIST”, “REQUEST REJECTED – PREREQUISITE”, or “REQUEST REJECTED – QUATO”.

5. Implement a `AssignInfo` class with the following UML diagram.

AssignInfo	
- - -	assignResult: String assignMessage: String course: Course
+	getter/setter methods

- `AssignInfo` class is used to interact `Faculty` class with the `Course` class.
- Once an instructor attempts to be assigned to a course, the course object creates a new `AssignInfo`, fills the data fields of it and adds it to the `AssignInfoList` of the instructor in the `assignInstructor` method of the Course class.
- The data fields store the following information:
   - `course` keeps the course assigned,
   - `assignResult` keeps the whether the assignment request “APPROVED” or “REJECTED”.
   - `assignMessage` stores a message such as “INSTRUCTOR *NAME SURNAME* ASSIGNED” or “ANOTHER INSTRUCTOR HAS ALREADY ASSIGNED” or “DEPARTMENT MISMATCH”.

6. Implement a test class for your program to create new courses, students, and instructors. Then register students to the courses, assign instructors to the courses and move the students in the replacements list of a course to the registered list once an instructor has been assigned to the course. A simple test class file will be given to you with a sample output file.

This is a simple scenario to test your class implementations. There might be other test cases too. Therefore, please pay attention to use the same class, method and variable names in your implementations. You are allowed to increase the number of methods in the classes; however, you cannot decrease the number of them. *You are not allowed to use ArrayLists in the homework! You can only use Arrays.*

Submission Instructions:

Please zip and submit your files using filename YourNumberHW6.zip
(ex: 150713852HW6.zip) to Canvas system (under Assignments tab).
Your zip file should contain the following 10 files:

1. 6 Java source files: Student.java, Course.java, Faculty.java, RegisterInfo.java, AssignInfo.java and Test.java
2. 6 Java class files: Student.class, Course.class, Faculty.class, RegisterInfo.class, AssignInfo.class, Test.class

Notes:

1. Write a comment at the beginning of your program to explain the purpose of the program.
2. Write your name and student ID as a comment.
3. Include necessary comments to explain your actions.
4. Select meaningful names for your variables and class name.
5. You are allowed to use the materials that you have learned in lectures & labs.
6. Do not use things that you did not learn in the course.
7. **Program submissions** should be done through the Canvas class page, under the assignments tab. Do not send program submissions through e-mail. E-mail attachments will not be accepted as valid submissions.
8. You are responsible for making sure you are turning in the right file, and that it is not corrupted in anyway. We will not allow resubmissions if you turn in the wrong file, even if you can prove that you have not modified the file after the deadline.
9. In case of any form of **copying and cheating** on solutions, all parts will get **ZERO** grade. You should submit your own work. In case of any forms of cheating or copying, both giver and receiver are equally culpable and suffer equal penalties.
   All types of plagiarism will result in zero grade from the homework.
10. No late submission will be accepted.

<u>Grading:</u>

- RegisterInfo Class → 5 points
- AssignInfo Class → 5 points
- Student Class → 12 points
- Faculty Class → 12 points
- Course Class → 25 points
- Test Class → 6 Points
- The correctness of test cases → 25 points
- Submission Format → 10 points
  - 6 java files + 6 class files
  - Make sure that your class files can be executed on another computer.
  - Make sure that the input/output of your program must be the same with the examples above (all informative strings & spaces).
  - Comments are necessary!