ECE373 Assignment 3

DEVELOPING AN OBJECT -ORIENTED MODEL FOR A UNIVERSITY

1 Objectives

In this assignment you will extend earlier work to implement an object oriented model of a University.

Learning objectives include

- Inheritance, Abstract classes along with Overloading/Overriding
- Organizing java classes in multilevel packages and exposure to a larger scale project
- Further Familiarizing with programming concepts introduced in lectures including (but not limited to) Strings, ArrayLists and their methods

Special Notes:

- For this assignment, **2 Drivers to test your program are provided**. These are provided to clarify how your program should run. We will use these drivers to test the programs, but we will also use additional drivers to do additional testing of the functionality specified in this assignment description.
- Before you start to program, make sure to go through the code provided in the Drivers (found on d2l)) and understand them. Understanding the drivers will make things clearer.
- START EARLY. DO NOT START LAST MINUTE.

2 The University Model

In this assignment we will develop an object-oriented model for a university which extends our earlier work. The requirements from earlier work for students, courses, and departments, which are relevant to this assignment, are reproduced at the end of this document for reference.

Having developed a working model of a university for the previous assignment, in this assignment (i) we will use inheritance to enhance code reuse and improve structure of code (ii) add new classes OnlineCourse, CampusCourse, Employee, Staff with unique functionalities for each.

In developing the university program, you need to carefully note all the requirements below to identify what you need to implement (and what changes should be made to the previous classes for students, courses, and departments). The extended requirements include:

Person:

- Each Person will have a name, a schedule, a campus course list, an online course list.
- There are 2 kinds of Persons: Students and Employees
- Person class will
  - be the parent class of the Student, and Employee classes
  - Implement the concrete detectConflict(Course aCourse) method that returns a boolean true if a schedule conflict is detected with the course aCourse.
  - Implement the concrete printSchedule() method to print this person's schedule.
  - declare abstract methods addCampusCourse(CampusCourse cCourse) and addOnlineCourse(OnlineCourse oCourse) methods which should be implemented by its subclasses

Student:

In addition to the before mentioned, students will

- belong to a department
- be a subclass of Person
- have additional fields: units completed, total units needed for degree completion, currently enrolled credits, and tuition fee.
- have a method to calculate remaining credit for degree completion
- implement the addCampusCourse and addOnlineCourse methods.
    - `addCampusCourse` need to use detectConflict from the person class to check if there is a conflict and use the availableTo(aStudent) from CampusCourse class to check if this course is available for this student (see the CampusCourse class description). This method, if the course is added, also adds the course's number of credits to the student's enrolled credits.
    - `addOnlineCourse` use the availableTo(astudent) from OnlineCourse class to check does this course available for this student (see the OnlineCourse class description). This method, if the course is added, also adds the course's number of credits to the student's enrolled credits.
- have a method to calculate the total tuition fee. Tuition fee depends on the courses taken by student. (On campus fee plus online fee)
    - On campus fee = Number of credits enrolled on campus * 300
    - Online fee = Flat $2000 for every online course that is 3 credits and $3000 if the online course is 4 credits.
- have a method a dropCampusCourse(CampusCourse cCourse) which should first check to see if the course is in the student's course list.
  If it is, then only in dropCampusCourse method, you should check if the student has one or more online courses. This student needs to have at least 6 credits for campus course after student drop this campus course to hold the online courses. If student will not have at least 6 campus course credits after dropping this campus course, you need to print "'student name' can't drop this CampusCourse, because student don't has enough campus course credit to hold online course".

After checking the condition in dropCampusCourse method. If we have enough credits, the course should be removed from their schedule, otherwise an error message should be printed out: "The course '**course name**' could not be dropped because '**student name**' is not enrolled in '**course name**'." (check output files for more details).

- have a method a dropOnlineCourse(OnlineCourse oCourse) which should first check to see if the course is in the student's course list. Then the course should be removed from their schedule, otherwise an error message should be printed out: "The course '**course name**' could not be dropped because '**student name**' is not enrolled in '**course name**'." (check output files for more details).

Employee:

- will be a subclass of Person (as mentioned).
- Employees can be either Professor or Staff.
- will have an **abstract** method called earns() that needs to be implemented in the children
- will have an **abstract** method called raise (double percent) needs to be implemented in the children.

Professor:

- have a filed salary, salary is how much professor earn for each year.
- Professors can sign up to teach courses
    - when a professor signs up, he/she should be listed under the course info, and reciprocally he/she should have the course in his schedule (and have access to course info).
    - there can be only one professor for each course, but professors can teach multiple courses
    - If a professor is assigned to a course that already has a professor, an error message should be printed: "The professor cannot be assigned to this course, because professor '**professor name**' is already assigned to the course '**course name**'." (check outputs file for more detail of print).

- professors are associated with a university department
- professors earn return salary/26 for every pay period.
- In the raise method, salary is raised by a percentage.
- implement the addCampusCourse and addOnlineCourse methods.
    - addCampusCourse needs to use detectConflict from person class to check for possible conflicts.

Staff:

The staffs are also university employees, but they may be working part time. Staffs

- are associated with a university department
- have a field for payRate (hourly rate in $).
- have hoursWorked field (monthly) as they can be part time
- A staff earns hoursWorked * payRate (per pay period).
- have a field for tuition fee.
- have a method to calculate the total tuition fee.

  Tuition fee depends on the course taken by the staff. (On campus fee or online fee)

  i. On campus fee = Number of credits enrolled on campus * 300

  ii. Online fee is flat $2000 if online course is 3 credits and $3000 if the online course is 4 credits.
- In raise method, payRate is raised by percentage.
- can sign up to take a class, but **allowed to take no more than one course at a time** and the last course signed up overwrites any previously signed-up course (You should report a warning when this happens check output1 for more detail).

Course:

- Each course has a name, a department it belongs to, a course number, roster of students taking the course and a Professor who teaches the course, number of credit units for this course.
- It is the parent class of CampusCourse and OnlineCourse classes. The university offers 2 kinds of courses: On campus courses, and online courses.
- A concrete method **addStudentToRoster()** to add a student or a staff to the course's

student roster. This class type should determine what the student roster's ArrayList type should be.

- An abstract method availableTo(Student astudent) will return true or false to indicate whether this course is available for this student. Needs to be implemented in CampusCourse and OnlineCourse classes.

CampusCourse:

- Will have fields, max number of student in this course, schedule, and a classroom where the course meets.
- CampusCourse will have a **Classroom** assigned; therefore, before this assignment is done, it should be checked to make sure that no two classes which take place at the same time are assigned to the same classroom as this will cause a scheduling conflict for that classroom. Such classroom assignment conflicts should be detected and prevented. CampusCourses with overlapping meeting times should have different classrooms assigned. A classroom's printSchedule method should print the times and the courses scheduled in that room chronologically day by day (see driver outputs).
- CampusCourse's printSchedule should print what times the course meets.
- CampusCourse is available to a student if the course has not reached its max number of students (that can enroll) limit.

OnlineCourse:

- **Will not have schedule or classrooms** associated with it. It will **also not have** printSchedule() method.
- OnlineCourse will be **available to student only if the student has enrolled for at least 6 credit units of on campus courses**. If the student has not signed up for 6 campus course units and tries to sign up for an OnlineCourse, an error message should be printed saying "Student '**student name'** has only '**no. campus credits**' credits enrolled. Students should have at least 6 credits registered before registering online courses." (check outputs files).

ClassRoom:

- A classroom's printSchedule method should print the times and the courses scheduled in that room chronologically day by day (see driver outputs).

Additional Requirements (Departments, Classrooms, Etc.):

Students, professors, and staff should all have printSchedule() method. It should print the schedule for that person chronologically day by day for campus courses and list the online courses they are enrolled in (see the driver outputs). You will need to write code to convert the meeting times from the integers, such as 102, to descriptive format, like "Mon 9:30am to 10:45am" and print them, (recall that in previous assignment you did schedules with integers only).

The university also has departments, and **classrooms**. Note the requirements below:

- Each department will offer Courses.
- A department can report its students, professors, staff, and courses, using its printStudentList, printProfessorList printStaffList, and printCourseList (Prints both campus and online courses) methods.

The University class:

- You also need to create a class called University which keeps track of the departments and the class rooms in the University (may consider ArrayLists of relevant classes/objects mentioned above).
- The University class will also have methods named printStudentList, printProfessorList, , printStaffList, and printCourseList (Prints both campus and online courses) to report enrolled students, professors, working staff, and the list of courses at the university.

Create Three Packages to Group Classes as below

Create three packages to organize the classes in the university program. Name these packages as follows and place the classes accordingly:

- org.university.hardware (multi-level package) will contain

- Classroom, Department classes

- org.university.people package will contain
  - Person, Employee, Student, Professor and Staff classes
- org.university.software package will contain
  - Course, CampusCourse, OnlineCourse, University and one of the Drivers (mentioned below)

3 Drivers and Testing

To guide you and make the assignment more clear, a couple of Driver classes and their expected execution results are provided. These drivers should be used for testing. We also use them as part of our grading (but the grading will not be limited to these two test drivers alone). You need to make sure your implementation will run against both of these drivers and produces same results as shown. Print schedules in the day and time format (Monday 8:00am to 9:15am, See the expected outputs from the drivers). The drivers are to help you but every minor detail might not be in these Driver classes, so you must think and deduce based on the specified requirements for the model what fields and methods should be implemented for each class.

4 Submission

- **Make sure** you have done the following before you submit:
  - Name your project as University_yourLastname. (_ is underscore, substitute "yourLastname" with your last name, for example Smith)
  - Name your zip file as Assignment3_yourLastname.zip (for example, Assignment3_Lahiru.zip).
  - Export the project as an Archive zip file as discussed previously.

You need to submit the following:

1) The archive file Assignment3_yourLastname.zip
2) A readme file in pdf containing the UML description for your classes **and their associations (check class notes for more detail)**, which should be

named as Assignment3_readme_ yourLastname.pdf.

APPENDIX: Previous Model Requirements

1. The Model

The model must observe the following rules:

- Each Student will have a name, and is enrolled in a university department, and has a course list.
- Students can sign up for courses.
  - When a student signs up for a course, the course should appear in his list of courses, and reciprocally he/she should appear on the course roster.
  - Students can sign up for multiple courses

- Each Department has a department name, a list of courses it offers, and a list of students enrolled (majoring) in that department
  - Departments add new students: the Student is added to the department's list of Students, and also the student's department should be set to that specific department.
  - Departments add Courses: the Course will be added to the Department's list of Courses, and also the Course's department should be set to that specific department.
  -

- Each course should at least have a student roster, a course number, a course name, the department, to which it belongs, and a schedule, when it meets (see below for schedule).

- When queried, the university should be able to list all departments, and the courses, students, professors, and staffs in each department.

2. Schedule

The schedule for each course is set as a set of numbers, one for each class meeting time. In each number, the first digit is for the day and the second and third digits are for the time slot of that day. For example, "101" is the first time slot of Monday. "203" is the third time slot of Tuesday. The following string arrays can be used to map numbers to relevant dates and slots.

```
1  String[] Week = {"Mon","Tue","Wed","Thu","Fri"};
2  String[] Slot = {"8:00am to 9:15am",
3                   "9:30am to 10:45am",
4                   "11:00am to 12:15pm",
5                   "12:30pm to 1:45pm",
6                   "2:00pm to 3:15pm",
7                   "3:30pm to 4:45pm"
8  };
```

Example : "205" will be mapped to the second element of Week array (Week[1] not Week[2]) and fifth element of the Slot array to get the time "Tue 2:00pm to 3:15pm", "402" will be mapped to the fourth element of Week array and second element of the Slot array to get the time "Thu 9:30am to 10:45am".

FAQ:

Q: What is meant by "UML description for your classes and their associations"?

A: Show inheritance. Show aggregations and other associations and their multiplicities.

Q: I have errors in the driver files, should I edit them?

A: No, your errors are due to the files you created, don't edit the driver files unless it is announced to do so by everyone.

Q: Do we need to add staff objects to the studentRoster() field?

A: Yes, because the staff can also be students in a course (although they can only take one course at a time). When a staff takes a course, he also becomes a student in that course and should be on the course's roster.

EXTRA CREDIT (10 percent)

To receive this extra credit, you should be able to substitute a dropCourse method instead of the two methods dropCampusCourse, and dropOnlineCourse such that it has the same functionality as both of these later methods, but works seamlessly across both kinds of courses. In addition you need to be able to substitute a addCourse method instead of the two methods addCampusCourse and addOnlineCourse such that it has the same functionality as both of these later methods, but works seamlessly across both kinds of courses. You should be able to report the number of total online course credits and total campus course credits being taken when asked. You must still maintain two subclasses OnlineCourse and CampusCourse for the class Course class, so Course will still be an abstract parent class for both of them.