Programming 2
COSC2136 OUA – Study Period 1, 2015

Academic Management System (AMS)

Assignment 1: *OO Design and Implementation* (25 marks)

The objective of this assignment is to introduce you to a real-life development practices, and also extend your understanding of OO design and development processes (utilising basic UML and Java programming language). As such, the emphasis is on adhering to a specification, which was designed to test various aspects of OO development, rather than developing a complete commercial grade product with realistic business rules.

Your task is to *implement a simple "Academic Management System (AMS)"* outlined below. The solution should be *developed according to a supplied UML class diagram, and also adhere to a purposively built test harness*. The class diagram itself will be outlined during our online chat session next Tuesday and a final class diagram will be made available on Blackboard for your reference. You may use this diagram to whatever extent you consider appropriate. The test harness can be found in the *ams.test* package of the provided source code.

The system will consist of a collection of interacting classes/interfaces to store information about:

1. The **university** and its **program**;
2. The **courses** offered as part of the program;
3. The **student** who enrols/withdraws into/from courses;
4. The **results** obtained by the student.

The system will also include a unified "front end", through the provision of an *AMSModel* façade interface (located in the *ams.model.facade* package) that will offer a single point of entry for all high-level operations. This front end will support the testing of your program, independently of a graphical user interface, using the test harness.

1. AMS Scenario

The system will need to be modelled using a number of OO classes/interfaces. The main entities involved in the system, and their behaviour, are detailed below.

University
The *university* should capture information about the student and the program. Note that a typical university will contain multiple students and programs, but for simplicity reasons the decision was made to 'restrict' the university to **one student and one program only**.

Program
The *program* has a program code (e.g. "BP062"), a title (e.g. "Bachelor of IT"), and a collection of courses. The program is dynamic and unbounded i.e. you can add any number of courses to a program.

Course
Each *course* has a course code (e.g. "COSC2136"), a title (e.g. "Adv Prog Techniques"), an associated number of credit points (see below), and any number of prerequisite courses. Note that a course can be added to a program only if all the prerequisite courses already exist. Also, the system needs to cater for two distinct types of Courses, `Core Courses` and `Electives`.

- `Core courses` have a fixed (constant) number of credit points (12 points), whereas `Elective` courses can be allocated 6 or 12 credit points.

Student
The *student* has a student ID, a full name, a maximum allowable study load (defined in terms of number of credit points, see below), a collection of currently enrolled courses, and a collection of results for the previously completed courses. The student can enrol and withdraw into/from offered courses. Also, the system needs to cater for two distinct types of Students, `Undergraduate (UG)` and `Postgraduate (PG)`.

- The maximum allowable study load is fixed (constant) at 60 credit points for UG students, and 48 credit points for PG students.
- The course enrolment procedure will differ based on a particular student type as described further in section 3 below.

Result
Each *result* represents a grade obtained by the student for a given course. For simplicity reasons, the result is defined in terms of a boolean value - true (PA), or false (NN).

2. Base Functionality

At a minimum, the AMS system should provide the following functionality (check the provided *AMSModel* facade interface for a complete list of required functions):

- Create new university
- Add a program and a student to the university (as stated previously, the university will have one program and one student only)
- Add/delete a course to/from a program
- Get a list (collection) of all the courses in the program
- Get a count of all the core courses in the program
- Get a count of all the elective courses in the program
- Enrol a student into course
- Withdraw a student from course
- Get a list (collection) of all currently enrolled courses for the student
- Calculate the current study load (number of credit points for all *currently enrolled* courses) of a student
- Add a result for a completed course
- Get a result for a completed course
- Get a list (collection) of all the results for all the completed courses i.e. student's past academic history
- Calculate total career points (number of credit points for all *passed* courses) of a student

3. Additional Implementation Details and Constraints

- Your primary goal is to implement the provided `AMSModel` interface, in a class called `AMSFacade` in order to provide the behaviour specified as comments in the provided `AMSModel` source file and tested by the provided `TestHarness.java`. In Assignment 2 you will be asked to write a graphical user interface to more effectively utilise `AMSModel`.

- You have freedom in how you choose to implement your solution; however, you must implement it in such a way that the `TestHarness` is **NOT** modified. You should use inheritance, polymorphism, abstract classes and interfaces effectively, as taught in this course.

- Even though the `AMSModel` interface returns array types, you should use data structures from the Java collection API (`List, Map` etc.) in your implementation and convert them for return as necessary. You should choose structures that are appropriate to the task, for example if lookups are frequently performed a `Map` would be a suitable data structure.

- System Exceptions must extend the provided `AMSException` class (located in the *ams.model.exception* package). Note: at the minimum you should implement and use two exception sub-classes: `ProgramException` and `EnrollmentException`.

- You must provide appropriate constructors and methods as required by the `TestHarness` in order to ensure that your solution can be complied and tested **without modifying** the `TestHarness`. Example1. `Program` class must implement the `Course[] getAllCourses()` method as used in the `TestHarness` tests.
Example 2. `UGStudent`, which represents one of the concrete `Student` classes that you need to implement according to `TestHarness`, must have the following constructor:

```
public UGStudent (int studentID, String studentname)
```

The rest of the requirements can be determined from the `TestHarness` source file.

Program Structure Rules:

The program is dynamic and unbounded. That is, you can have an unlimited number of courses added to the program. When calling `addCourse(…)` and `removeCourse(…)` methods to build up the program structure, the following rules **must** be maintained:

1. Upon adding course A to a program, you need to check that all prerequisite courses for course A exist in the program;
2. You cannot remove course A from the program, if course A is currently specified as one of the prerequisites for the other course/s in the program.

The `ProgramException` should be thrown, when the above rules are violated.

Enrolment Rules:

1. Duplicate enrolments are not allowed i.e. the student cannot enrol in the course if this course is in the list of currently enrolled courses, or if it was *completed previously with a pass grade*.
2. An UG student cannot be overloaded i.e. the maximum allowable course load cannot be exceeded.
3. A PG student *can be overloaded by up to 6 credit points, but only if he/she has never failed any of the previously studied courses* (as indicated by the previous Results of this student).
4. An UG student must pass all required pre-requisite courses prior to enrolling into a Course.
5. A PG student must pass any one of the pre-requisite courses prior to enrolling into a Course.

The `EnrollmentException` should be thrown, when the above rules are violated.

4. String Representations

Certain classes must provide specific string representations by implementing a `public String toString()` method which conforms to the format described below. This is required to enable correct operation of the `TestHarness` and is used as an alternative to data access methods, thereby allowing greater flexibility in how you implement your solution. Examine the `TestHarness` and the specification below to see how they are used.

Program

`program_code:program_title`

e.g. BP062:Bachelor of Computer Science

Student

`student_id:full_name:maximum_load`

e.g. 3000001:Joe Bloggs:60

Course

`course_code:course_name:credit_points:prereqs_courseCodes:type`

e.g. COSC2136:Programming 2:12:COSC1073:CORE

Note:
(1) `type` must be either “CORE” or “ELECTIVE”;
(2) `prereqsCourseCodes` are optional since not all courses have pre-requisites.
e.g. COSC1073:Programming 1:12:CORE
(3) multiple `prereqsCourseCodes` should be separated with comas (no space in between)
e.g. COSC1107:Computing Theory:12:MATH1074,COSC1076:CORE

Result
`courseCode:grade`
e.g. COSC2136:pass
Note: grade can be either “pass” or “fail”.

`Program`, `Student`, and `Course` codes/ids and names/titles are assumed to be unique and case insensitive, and must not contain any colons (`:`) or commas (`,`) that would otherwise lead to invalidly formatted string representations.

Regulations and Submission Instructions

You are free to refer to textbooks, notes, work in study groups etc. to discover approaches to problems; however, the assignment must be your own individual work.

Where you do make use of other references, please cite them in your work. Note that you will only be assessed on your own work so the use of third party code is discouraged.

The source code for this assignment (i.e. **Eclipse** project[1]) should be submitted through **Weblearn** by the listed due date (see Blackboard->Course Information->Study Calendar).

Please note that Weblearn only accepts a single file for submission so you should compress your whole project (i.e. the project folder in Eclipse workspace) into a .zip archive and submit the archive.

Late submissions are handled as per usual RMIT regulations - 10% deduction (2.5 marks in Assignment 1) per day. You are only allowed to have 5 late days maximum.

[1] You can develop your system using any IDE, but will need to create an Eclipse project using your source code files for submission purposes. You can then zip the complete project (found in your workspace), and submit the corresponding zip archive via Weblearn.