King Abdulaziz University
Faculty of Computing and Information Systems
Spring 2016 – 2nd Term 2016

Course Code: CPCS203 Course Name: Programming II

Assignment # 4 (FCIT-eAA)

Assigned Date	**: Sunday, 17th April 2016 (17/04/2016)**
Delivery Date & time:	**Sunday, 1st May 2016 (01/05/2016) @ 11:59PM**

WARNING:

- This program must ONLY be submitted on the Blackboard!
- This is an Individual assignment. You must solve it by yourself. Any form of plagiarism will result in receiving **-4** (less than zero) in the project.
- This project worth 6% of the overall module marks (100%).
- There will be a Quiz on assignment 4 and for further information please check with you lab instructor.

Objectives

- Continue practicing the use and implement of the Inheritance concept (which supports reuse), dynamic binding.
- Learn how to use and implement Polymorphism and Object Explicit casting.
- Learn to use and implement ArrayList class.
- To explore the equals method in the Object class.
- To design and use abstract classes.

- Learn to use interfaces and define them. Also, define classes that implement interfaces.

Delivery

- Submit your assignment on the Blackboard ONLY.
- **Make sure to add your names / IDs / Section / course name / Assignment number, as comment at the beginning of your program**.

Description

FCIT Academic Affairs Electronic System (FCIT-eAA)

The FCIT faculty is customizing its Academic Affairs Electronic System (FCIT-eAA). This customization will help Academic Affairs in fulfilling new Faculty requirements and keeping track of student academic activities. First, the system must continue with providing the basic learning management systems functionalities. For example, students registering courses, printing tables or adding dropping, etc… Moreover, faculty can choose courses to teach, print course student lists. Typically Academic Affairs will continue having functionalities such as printing sorted lists, search for students information or faculty, etc..

In addition, to the basic functionality the updated FCIT-eAA system must generate validation reports on registration (adding courses) activities to help in addressing problems quickly.

The validity reports generate reports on students' cases that need attention. These cases are students registering courses under 12 hours, students registering courses cross-level and graduate students that must register Senior project 1 but did not. The Cross-level students are students not following the department plan accurately, which are considered delayed students. The requirements of every needed command will be described below.

More Details are as follows:

Your program will use File I/O to read input data from the given file with name [infciteaa.txt]. Make sure that the file exists or display a message that it does not exist. Once the file is open all FCIT-eAA information must be read into the application. This means the Course, Instructor, Student and Sections data must be read and stored in the appropriate data structure in the FCIT-eAA application. All data entry commands will start with the word "input". This is followed by an integer indicating the number of records that need to be read. The description of every data entry command is given in the coming sections.

The commands you will have to implement are as follows:

- **InputDepartmentPlan**– following this command the file consists of an integer indicating the total number of courses (52 courses) in the CS department plan. Thus, students are required to complete all these courses in order to graduate with a bachelor's degree in Computer Science.

The Course information stored in the input file is organized as follows:

1) If the line begins with the word "Level" then it will be followed by the level number. This is used to group the courses according to their levels in the plan. When you read this line you must read the integer that follows and store it in all courses of this level.

2) If the line begins with a Course code then you must read all remaining course information Course Number, Course Name, Course Credit, Theory and Lab. Moreover, the Course Name is stored with "?" marks instead of blanks to ensure that the complete course name is read. This must be replaced by blanks after it is read. For example:

In the input file the course name is: "Writing?Skills"

In the Application FCITEAA it must be replaced with "Writing ▽Skills", where ▽mean blank.

Furthermore, the words theory and lab if they are read from the input file then the Boolean variables in the Course class are set to true otherwise they are set to false. The level number is read from the beginning and should be repeated with every course in the same level. [see input file]

In the output file that must be created by the application with name “outfciteaa.txt” must include the following message after all Course data is inserted. Also, see sample outputfile.

The Message: “Department Plan(Courses) Data has been Inserted”

==”

- **InputInstructorData** – Again this command is followed by an integer indicating the number of instructors teaching in the Faculty. Their information includes Instructor’s name, Instructor’s ID, Beginning of work year, Qualification (PhD, MSc,etc..) and Instructor’s Schedule (List of Sections). The instructors Schedule will be chosen in a separate command (coming later). Similar to Course the name of the Instructor’s name and her qualifications will include “?” mark. Thus, the “?” mark must be replaced with a blank once read into the application. [see input file]

In the output file that must be created by the application with name “outfciteaa.txt” must include the following message after all Instructors data is inserted. Also, see sample output file.

The Message: “Department Instructor's Data has been Inserted”

- **InputStudentData –** Here the Faculty Students’ information is read into the application. The information includes Students name (String), student id (Integer), student starting (Registration) date at faculty (Integer), the student’s current schedule, GPA (double) and Total Number of hours studied. Note that the student Schedule will be registered in a separate command (coming later). [see input file]

  In the output file that must be created by the application with name “outfciteaa.txt” must include the following message after all Students data is inserted. Also, see sample output file.

  The Message: “Department Student’s Data has been Inserted”

- **InputSectionData-** Finally, the Department Schedule information is read from the input file. The Schedule is built out of Sections information (current available courses). These sections are the ones registered by students and they are added into their schedules. Same thing goes with the instructor's; their schedules include the sections they teach. The Section information includes ID, Name, Course Information (determined by the pair course code and course num in the input file), Size(No. of students in section), Slot for Theory part (Day & Time), Slot for Lab part (Day & Time), Room and Lab numbers, Instructor name and finally, list of students (Student class) Registered. Note that if a Theory or Lab part of the course is not given then the term "N/A" is added to indicate that it's not taught in the course. Also, the Course information is stored in the Sections array as a reference to the class course described above.

In the output file that must be created by the application with name "outfciteaa.txt" must include the following message after all Sections data is inserted. Also, see sample output file.

The Message: "Department Schedule (Sections) has been Inserted"

- **RegisterCourse –** This command allows the students to register the courses they want. Then it checks the validity of their choices. First check made is verifying that the given Student ID is actually a student's ID. Then, check for **the five cases** listed below. The result of the request and checks will be the registered student schedule. This will be followed by a report stating the problems with the requested courses, if any. In the input file the requests start with the student number then it is followed by a list of requested section IDs, which means all input data are integers. The request line ends with a "-1" indicating no more courses are requested. [see input file]. Keep in mind that once a student registers a course and it is added to its Schedule then AT THE SAME TIME it must be added to the Section Student list found in the Department Schedule.

The cases that the system should check for are as follows:

- If all courses requests are from the same level then the request is valid and it is accepted.
- If a student is a graduate [TotalHourse > 100 & Studying Years >= 4] and did not register CPCS498 then a warning must be added to the registration report.
- If a student has registered less than 12 hours then this must be highlighted in the report.

- If a student requested courses from two different levels then this must be highlighted in the report. Normally, this happens to students who are delayed students. This can happen to students who finished 4 years or more and their totalCredit is less than 100. Their status will be "Fourth Year Delayed".
- If two courses contradict in slots then the one from the lower level is registered and a warning is added to the registration report. This means the second section will not be registered. [see in put file]

  For example: there is a conflict in the lab between sections DGR & GGR then the first section DGR is registered and the second one GGR will not be registered.

The output will be the registered schedule of the student followed by a report on the validity of the requests.[see output file]

- **InstructorLoadRequest**- Up to this point no instructor has requested a course to teach. Thus, under this command the total number of requests is given. This is followed by a list of integers the first one will indicate the Instructors ID then the remaining integers are the list of Section IDs that the instructor wants to teach. For ease of development it is GRANTEED that the requests comply with all regulations. This means just read them into the system and generate the required Schedules for every instructor. <u>It's important to keep track of class aggregations and similar to Student Registration, once a section is added to Instructor load her name (instructor) must also be added in the Sections array.</u>

- **Print All InstructorsLoads-** This is the final request, which will only printout all the instructors' schedules. However, the constraint here is to print them SORTED in ascending order based on the instructors starting date. Typically, you will need to implement the interface Comparable and its method compareTo() in order to utilize the sort method from Arrays.
- **Print Section Student List-** The command reads the Sections ID and prints all its registered students.

Further Details are as follows:

1. The Application must define three main arrays

```
private static Course[] deprtCourses;

private static ArrayList<FCITMember> deprtMember =  new ArrayList<FCITMember>();

private static ArrayList<Section> deprtSchedule = new ArrayList<Section>();
```

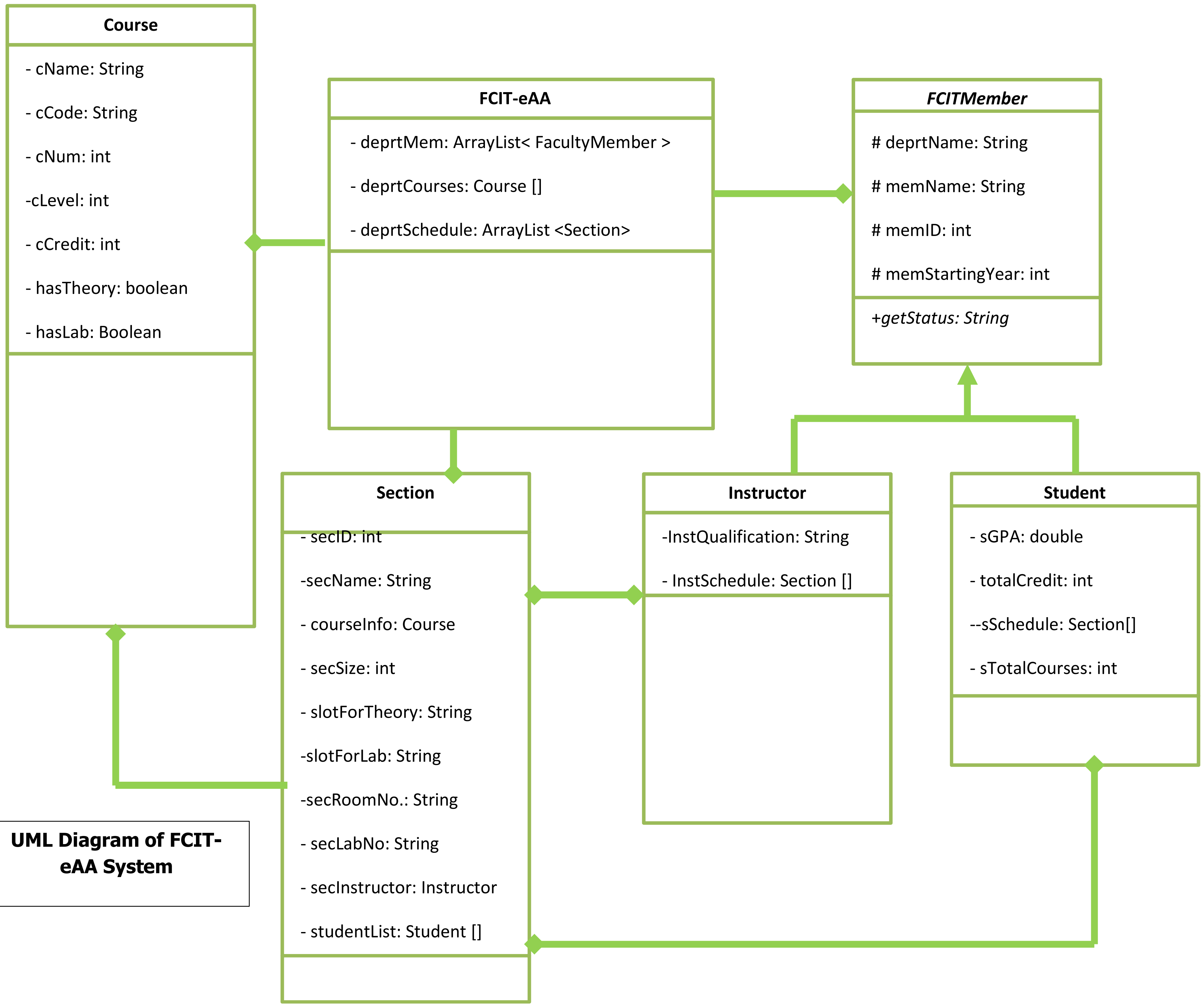
2. The application must define six classes (FCIT-eAA/ Course/ FCITMember/ Section/ Instructor/ Student) follow the relations and attributes given in the UML diagram below. Note that FCITMember is and abstract class that contains an abstract method get status with signature as given below:

```
public abstract String getStatus();
```

3. The class Instructor must implement interface Comparable based on the memStartingYear attribute. Hence, the older members come first in the list.

4. The FCIT-eAA must include the main() method.
5. Your program output must be exactly same as given sample output files.
6. Your display should be in a readable form.
7. Document your code with comments.
8. Use meaningful variables.
9. Use dash lines between each method (when used).

See the **UML Diagram underneath** to know basic class details.



UML Diagram of FCIT-eAA System

Zoom word file, if you are unable to see the above UML Class diagram.

Important Notes:

- **(Delayed submission will not be accepted and there will not be any extension of the project) .**

Deliverables:

- You should submit **one zip file containing all java codes**:

  BA1110348P34fciteaa.java where BA is your section, **1110348 your ID and p4 is program 4.**

NOTE: your name, ID, and section number should be included as comments in all files!

Input and Output Format

Your program must generate output in a similar format to the sample run provided.

Sample input: See sample input file.

Sample output : See sample output files.

Good Luck and Start Early!