OMSCS 6310 – Software Architecture & Design
Assignment #2 [100 points]: Course Management System – Individual Design (v2)
Summer Term 2017 – Prof. Mark Moss
Due Date: Tuesday, May 30, 2017, 11:59 pm (AOE)

Submission:
- This assignment must be completed as an individual, not as part of a group.
- Submit your answers via T-Square.
- You must notify us via a private post on Piazza BEFORE the Due Date if you are encountering difficulty submitting your project. You will not be penalized for situations where T-Square is encountering significant technical problems. However, you must alert us before the Due Date – not well after the fact.

Scenario: You are part of a project team that has been directed to help design and implement a new course management system for a local university. This university is experimenting with allowing their students to take a very flexible blend of on-campus and online courses, and they feel that the current automated systems used to manage traditional "brick-and-mortar" institutions will not suffice. This phase of the project involves developing initial design documents to ensure that the team has a solid grasp of the problem domain in general, along with the client's more specific requirements. Your initial design will be updated and refined through numerous design and implementation cycles during this course.

Disclaimer: *This scenario has been developed solely for this course. Any similarities or differences between this scenario and any of the programs at Georgia Tech programs are purely coincidental.*

Deliverables: This assignment requires you to submit the following items:
1. ***Object Oriented Analysis [25 points]:*** The main intent of this portion of the assignment is for you to take the time to read the Client's Problem Description below, and be sure that you have an understanding of the problem domain. You must apply the OOA techniques as described in P2L2 of the course Udacity videos to the Client's Problem Description. Produce your answer in the form of (Candidate) Classes, Attributes, Operations and Potential Relationships as follows:

   CLASSES, ATTRIBUTES and OPERATIONS:
   Class #1: <supporting sentence from the problem description>
   - **Attribute #1 <supporting sentence>**
   - **Attribute #2 <supporting sentence>**
   - **...**
   - **Operation #1**
   - **Operation #2**
   - **...**

   Class #2 <supporting sentence>
   ...
   RELATIONSHIPS:
   Relationship #1 – type {generalization, aggregation or association} between Class A and Class B <supporting sentence>
   ...

As an example, consider the following partial description of a flight scheduling scenario:

"… Airline passengers can purchase one or more tickets. Passengers must have a unique identifier assigned to them to prevent name conflicts. Each ticket will list a flight number, departure airport, departure time and destination airport. Also, each ticket will have the name of one specific passenger for security purposes."

You might produce the following OOA results based on this scenario:

CLASSES, ATTRIBUTES and OPERATIONS:
Class #1: Passengers <Airline passengers can purchase...>
- **Unique Identifier attr: <Passengers must have a unique identifier...>**
- **Name attr: <...each ticket will have the name of one specific passenger...>**
- **Purchase() op: <Airline passengers can purchase one or more tickets.>**
- **...**

Class #2: Tickets <Each ticket will...>
- **Flight Number attr: <Each ticket will list a flight number...>**
- **Departure Airport attr: <Each ticket will list a...departure airport,...>**
- **...**

...
RELATIONSHIPS:
Purchases: Association between Passengers and Tickets <Airline passengers can purchase one or more tickets.>
...

These results are not necessarily comprehensive for the scenario, and are shown to present a clear example of the format. ***Key factors that will affect your score:***

- A key part of the OOA technique involves identifying nouns, adjectives and verbs in the problem description to help generate the classes, attributes and relationships. Part of developing your skills as a software architect/designer involves using some discretion in selecting the components that most appropriately reflect the problem domain. Said another way, simply selecting every single noun and translating it into a class will generally lead you to create some "unnecessarily messy" and overly complicated results. There is no "exact/right" number of classes for the problem description below; however, a reasonable result will likely have somewhere between three to ten (3 – 10) classes.
- Approximately 10 points will be granted based on identifying a reasonable set of classes;
- Approximately 10 points will be granted based on identifying the corresponding attributes and operations for those classes;
- Approximately 5 points will be granted based on identifying supported relationships between your classes; and,
- Up to 5 points might be deducted for answers have significant formatting, grammatical or similar types of errors.

- Ensure that your classes, attributes, etc. are supported by the client's problem description. Be careful not to inject your perspectives on the problem, especially since your additions might fundamentally change the structure and/or nature of the problem (and proposed solution).
- You are permitted to add a few sentences to explain any aspects of your analysis that you feel need extra clarification. These sentences are optional: non-submissions will not be penalized.

2. ***Unified Modeling Language (UML) Class Model Diagram [75 points]:*** The main intent of this portion of the assignment is for you to transform the OOA results that you generated above into a valid UML diagram. You must apply the UML diagramming techniques as initially described P2L2, and further expanded upon in P2L3 through P2L5. The Library Exercise example in the "Final Considerations" section in P2L5 sets the standard for the level of diagram you are expected to produce for the Client's Problem Description below. ***Key factors that will affect your score:***
   - You must generate your class model diagram using an automated UML diagramming tool so that it is as clear & legible as possible. The choice of tool is yours; however, you should do a "sanity check" to make sure that the final diagram is readable/legible when exported to PDF; or, if necessary, some reasonable graphical format (PNG, JPG or GIF).
   - Approximately 25 points will be granted based on correctly displaying the classes;
   - Approximately 25 points will be granted based on correctly displaying the corresponding attributes and operations for those classes;
   - Approximately 15 points will be granted based on correctly displaying supported relationships between your classes, to include clearing showing generalization, aggregation and cardinalities for associations; and,
   - Approximately 10 points will be granted based on correctly assigning reasonable data types for the values stored in the attributes and produced by the operation results. You are welcome to use fairly generic data types as shown in the example such as Integer, String, Boolean, Float/Real, Date, Money/Currency, etc.
   - You must designate which version of UML you will be using – either 1.4 (the latest ISO-accepted version) or 2.0 (the latest OMG-accepted version). There are significant differences between the versions, so your diagram must be consistent with the standard you've designated. Either version is acceptable at this point in the course, though (be warned) we might require a specific version for some of the other UML style components later in the course, especially considering the activity-based diagrams.
   - You are permitted to add a few sentences to explain any aspects of your design that you feel need extra clarification. These sentences are optional: non-submissions will not be penalized.

Writing Style Guidelines: The style guidelines can be found on the course Udacity site, and at: https://s3.amazonaws.com/content.udacity-data.com/courses/gt-cs6310/assignments/writing.html
Your deliverables should be submitted to T-Square in Adobe PDF format with the file names ***oo_analysis.pdf*** and ***uml_class_model.pdf***. If your UML diagramming tool does not allow you to produce a single readable/legible diagram file, then you are permitted to use multiple files as necessary – just label the files clearly so that we know how to reassemble to overall diagram. The one file approach for your UML diagram is preferred, but we value readability over the number of submitted files. Please be aware that points will likely be deducted for unreadable submissions.

Client's Problem Description: There are a number of movements in education to leverage the potential of online education. Online courses can be offered in a variety of formats, and with content that is comparable to on-campus courses. However, many recent studies have recognized that advantages of each general approach – on-campus and online – can also be highly complementary. Our clients think that this combined approach has tremendous potential, and so they've decided to modify their Computer Science program to offer these types of courses to help encourage and increase enrollment. Our focus in this project, however, will be on developing a system to manage the online course enrollments and grade recordings.

From an operational perspective, the clients are envisioning that this system will be used to help determine the instructor allocations on a term-by-term basis. The system will also be needed to keep track of student course requests that are granted or refused, with the goal of satisfying as many requests as possible. They are refining their instructor hiring and course assignment processes, and this system will help them simulate these processes and (ultimately) make better decisions.

Students, Courses and Instructors

Their students will be given the opportunity to take a wide variety of courses. They represent a smaller institution, but their system must be able to maintain the historical records for all of these students. Persistent storage after a single execution sequence (run) of the system that you develop for them is not required at this time – that is, the information that is entered during any single run of your program does not have to be stored for later runs. Later phases of this project will require a persistent storage solution, but you are welcome to implement your solution using traditional Java data structures at this point.

Each student will have a "UUID" (University Unique Identifier) so that they can be distinguished from other students, past or present. The system must be able to track some basic identifying information for each of the students, to include names, home addresses, and phone numbers at a minimum. The system must also allow students to request enrollment in certain courses during each term, and to request disenrollment as needed.

The university will have a significant number of working students; and, the students will select courses that are most appropriate for their current job/position and future goals. The clients aren't establishing specific (and potentially limiting) course sequences that will be needed for degree completion. Instead, students will be allowed to construct and request approval for a sequence of courses that best fit their interests and skill requirements for their current or future employment opportunities. During any single term, students will be permitted to take one, two or three courses. Also, students can take courses as their work schedule allows to encourage flexibility. Students can remain enrolled in their degree program – and aren't automatically disenrolled – even if they don't take classes during a given term.

Students will select courses from the most current Course Catalog. Each course must have a distinct Course ID, along with the Course Title and Description. Also, some of these courses might only be

offered during certain times of the year – for example, a course might be offered only during the Fall term, while another course might be offered during the Winter, Spring and Summer terms. A small example of the Computer Science courses from our current Course Catalog is shown here:

ID	***Course Title***	***Terms Offered***
2	Programming I	Fall
3	Computer Architecture	Fall, Spring, Summer
4	Programming II	Fall, Winter, Summer
5	Algorithms and Data Structures	Winter, Spring
8	Software Architecture and Design	Fall, Spring, Summer
10	Machine Learning	Fall
11	Programming Languages	Winter, Summer
14	Database Systems	Fall, Winter, Spring, Summer
19	Operating Systems	Fall, Spring

This is only a small subset of the Computer Science classes they offer, minus the Course Descriptions. In a similar vein, the continuing growth of Computer Science has stimulated a growing number of sub-areas and new course topics. The clients will also need to be able to update our courses to make sure that they can address new topics, technologies and techniques. To that end, the system must be able to support the creation of new courses, along with the update and occasional removal of existing courses.

These courses must be created, taught and managed by qualified instructors. For each term, a list of instructors who have been hired for that term will be provided, where each instructor has the skills and competencies to teach some of the courses offered in the Course Catalog. Each instructor will select one of their eligible courses to teach during the term, which will determine the total number of available seats for that course. The personal information that must be maintained for instructors is similar to the information is maintained for students: names, addresses and phone numbers at a minimum. Instructors will also have a single UUID to distinguish them from other instructors and students. Instructors must be able to select the course(s) that they will teach during a given term.

For each course that is taken by a student, an academic record must be maintained that records the student's performance during that session of the course. Since a student might take a course multiple times, the term and year must also be maintained for each attempt. The student's performance will be measured by a final letter grade (e.g. A, B, C, D or F) and might also include comments from the instructor. Each record with comments, therefore, should also include a link to the instructor that taught that session of the course, just in case those comments are referenced later during an Academic Board, etc. to avoid any confusion. Instructors must be able to enter new academic records and update previously created records as well.

Finally, our clients have also observed that some of the more passionate and successful students have volunteered to become instructors in the degree program in later terms. And some instructors have been interested in taking some of the courses being offered – especially in some of the more unique disciplines. Consequently, the instructors have decided to take some of these courses, effectively

making them students as well. Given this "blending" aspect of the student and instructor groups, the client has found it easier to use the same ID for a single person for the duration of that person's time with the university, rather than issuing and tracking two different "student ID" and "instructor ID" values.

Requesting Courses versus Being Permitted to Take a Course

The clients want to offer a wide variety of courses to their students. They've also identified requirements and constraints, primarily based on the limitations of supporting student course requests. Their main areas of concern are: course prerequisites and passing scores; available seats for a given course based on instructor allocations; and, allowances for retaking courses for poor/failing scores.

There's a strong desire to allow students to take the courses that they feel will benefit them the most; however, the client's experiences have been that certain courses are very difficult to complete successfully without a very strong background, most often ensured by having the students take one or more prerequisites. Therefore, each (main) course in the catalog can also have one or more other prerequisite courses; and, all of the prerequisite courses must be completed successfully before that student is allowed to request and take the main course. Successful completion of a course means that the student must earn an A, B or C. A grade of D or F will not be considered successful completion for prerequisite purposes. A course is not required to have prerequisites, as is the case for many introductory/"101"-level courses, or some special topic courses.

Also, the number of available seats for a course is determined by the instructors who have been assigned to teach the course. The list of available seats for a course can change from term to term. Your system must track the number of available seats for a course based on the number of instructors that have agreed to teach the course during the term, and the number of students who have already been assigned a seat. More details about the specific number of seats that each instructor will make available for a course will be provided with the implementation details.

Finally, students might want to retake a course to improve their overall grades and grade point average. This is a good idea in numerous situations; however, we need to limit this to ensure that a large number of students have the opportunity to take courses for their first time as well. Therefore, the policy is that a student may retake a course if, and only if, they have earned only Ds or Fs only in that course so far.

In summary, for a student to be able to take a course, certain preconditions must be true:

- The student must have successfully completed all prerequisites courses;
- The student must not have taken the course before and earned an A, B or C; and,
- There must be an available seat to support the student's request.

UML Class Model Diagram: Classes, Attributes and Operations

Your UML class model diagram should describe the structure of student, instructor and course classes at a minimum, along with any other classes that you feel are warranted based on the client's

descriptions. Each of the classes should include the appropriate attributes to record required information, along with an appropriate data type. The data types don't have to match the specific syntax of any particular programming language such as C, Java, etc. but they should be reasonably clear, and should also distinguish between single (scalar) and composite (lists, arrays, etc.) values. Your diagram should include the relevant operations that will be needed to manage course offering and enrollment activities from term to term, to include:

- selecting which course that will be taught by an instructor;
- allowing students to request a course;
- checking course requests to ensure they are valid (e.g. all prerequisites taken);
- enrolling a student in a course based on a valid request; and,
- recording a final grade for the student once the course has been completed.

Closing Comments & Suggestions: This is the information that has been provided by the customer so far. We (the OMSCS 6310 Team) will likely conduct an Office Hours where you will be permitted to ask us questions in order to clarify the client's intent, etc. We will answer some of the questions, but we will not necessarily answer all of them. The clients will also add, update, and possibly remove some of the requirements over the span of the course. One of your main tasks will be to ensure that your architectural documents and related artifacts remain consistent with the problem requirements – and with your system implementations – over time.

Quick Reminder on Collaborating with Others: Please use Piazza for your questions and/or comments, and post publicly whenever it is appropriate. If your questions or comments contain information that specifically provides an answer for some part of the assignment, then please make your post private first, and we (the OMSCS 6310 Team) will review it and decide if it is suitable to be shared with the larger class. Best of luck on to you this assignment, and please contact us if you have questions or concerns.