OMSCS 6310 – Software Architecture & Design
Assignment #4 [100 points]: Course Management System – Individual Implementation (v3)
Fall Term 2017 – Prof. Mark Moss
Due Date: Wednesday, October 4, 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: The clients at the university are pleased with your progress in the initial design. In the previous phase of the project, you were asked to generate fundamental design documents that capture some of the main entities, attributes, and the relationships between the elements of the problem space – students, courses, instructors, academic records, etc. In this phase, you will implement the core business functionality for your system. Your task for this assignment will be to ensure that your prototype implementation functions consistently with the given requirements; and, to modify your UML diagram to maintain consistency with your implementation.

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. ***Implementation Source Code [50 points]:*** Your program must perform the following basic tasks: (a) read and store the information contained in eight files – ***courses.csv***, ***instructors.csv***, ***listings.csv***, ***prereqs.csv***, ***programs.csv***, ***records.csv***, ***requests.csv*** and ***students.csv***; and, (b) analyze the file contents and display the results. The clients will provide examples of the input and output file formats, along with an explanation of the desired analytical tasks. You must provide the actual Java source code, along with all references to any external packages that you used to develop your system, in a ZIP file named ***source_files.zip***. You must also provide an executable JAR file named ***working_system.jar*** to support testing and evaluation of your system.

   Key factors that will affect your score:
   - We will evaluate the correctness of your system against multiple test cases. The clients have provided five (5) basic test cases that will be used to evaluate the correctness of your system. These five cases have been provided to you, and you should use them to ensure that your code is producing the correct output in terms of function and formatting.
   - Each test case folder includes eight **`*.csv`** files: **`courses, instructors, listings, prereqs, programs, records, requests`** and **`students`**. Each test case also includes two **`*.txt`** files: a **`readme`** file with a brief explanation of the intent for that test case, a file named **`answer`** containing the expected output with correct content and formatting.

- We will also generate more complex test cases that will NOT be provided to you in advance. The file structures within the test cases will not change, but the file contents will, so please don't "hardcode" any of the file contents into your program.
- The five basic cases that you've been provided will be worth 4 points each. The remaining test cases will cover the remaining 30 points.

Please see the Submission Details section below for more information about how to send us your source code, etc. We will evaluate the structure and design of your source code, and its consistency with the UML and design-related artifacts that you provided in the earlier project phases. We will also evaluate the correctness of your system against multiple test cases.

2. ***Program Design & UML Diagram Consistency [50 points]:*** As mentioned earlier, the actual implementation of a working solution – even if only for a subset of the original problems requirements – might have prompted you to make changes to your underlying design. The main intent for this portion of the assignment is to update your UML Class Model from the previous assignment as needed to ensure it is consistent with the program you've just developed. And, even though it is important to generate programs that function correctly, it is also important to ensure that your design is consistent with the actual system. While Part 1 of this assignment focuses on the correctness of the system, Part 2 focuses on ensuring that you've made reasonable and consistent design choices.

   Key factors that will affect your score:
   - In the earlier design assignments, we were checking your UML diagrams to ensure consistency with your object-oriented analysis (explicitly or implicitly) of the client's requirements. In this portion of the assignment, we will be looking for consistency with your program's source code.
   - Your program code and the UML diagrams must match as precisely as possible. You must remove any unimplemented features – classes, attributes, operations, etc. – from your UML Class Model as required to make them consistent.
   - One key exception is that "simple" set() and get() operations may be omitted from your UML diagrams for clarity. The term simple in this case means set() and get() operations that simply modify or retrieve (respectively) an attribute's direct value based on the input parameter. If your setter/getter operations perform any other significantly complex actions or side effects, however, you must declare them in the UML diagram.
   - 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).
   - ***The "automated" aspect is NOT intended to mean that the diagramming tool you select must generate the diagrams and/or code for you automatically.*** Though these kinds of capabilities exist, it is much more desirable – at least during this course – that you take the time to update your UML diagram manually after you have completed your program, so that you develop a better understanding of the connections between your code and your model.
   - Approximately 10 points will be granted based on correctly displaying the classes;

- Approximately 15 points will be granted based on correctly displaying the corresponding attributes and operations for those classes;
- Approximately 20 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 5 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. Also, be clear about using composite data structures such as arrays, lists, sets, etc. versus scalar (single) values.
- As before, please clearly 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.

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 The deliverables should be submitted in the appropriate formats (ZIP, JAR, PDF) with the file names ***source_code.zip***, ***working_system.jar***, and ***uml_class_model.pdf***. Ensure that all files are clear and legible; points may be deducted for unreadable submissions.

Client's Problem Description: We were pleased with the initial design, and it helped us refine and further develop our requirements. Our main goal at this time is to ensure that our system manages the data in a reasonable way that allows us to answer certain basic questions about the data:

- Which courses are being taught during this current term?
- Which students have passed courses and spent money on our courses?
- Which requests should be granted or denied; and, why should they be denied?

During later phases of the project, this system will be developed to support operations over multiple sessions, to include all of the activities such as assigning the final course grades and creating the corresponding academic records that normally accompany transitioning from one session to the next. In this phase (assignment), however, we will only focus on the current session, and calculations will be done based on the data that is currently in the files for a given test case. Therefore, we would like you to develop a prototype system that will process the data in the input files (test cases), and then provide answers for our questions above. To better clarify our needs, we will provide more details about some of the input files and desired output formats below.

Students

students.csv: Our student data is contained in the ***students.csv*** file. This file (like the others) will be presented in a basic Comma Separated Values (CSV) format. Each record in this file will have five fields: (1) a unique identifier over the student domain; (2) the student's name; (3) the student's address of record; (4) the student's official phone/contact number; and, (5) the ID for the program that the student has selected. Note that some of the fields do contain spaces; however, we have

scrubbed the files as much as possible to ensure that there are no "embedded commas" which might complicate your processing. Here is an example of a ***students.csv*** file:

```
1004,CAROL TAYLOR,215 4th Avenue 27517,516479061,3
1009,GARY ALLEN,128 Pine Street 83866,8304231126,3
1012,JULIE TURNER,927 6th Avenue 78553,2587799053,2
1014,RENEE CARNEY,840 Main Street West 28729,8118091235,2
1015,JAMES FISHER,231 Windsor Court 12288,4477500021,2
1016,TRACEY WHITE,387 Canterbury Drive 49531,4952312905,1
1020,LILLIE LEWIS,373 Magnolia Court 55751,566944369,0
1021,JEFFREY CLAYTON,600 Bridle Lane 70941,6222277693,1
1022,SUE VELASQUEZ,204 Riverside Drive 72894,7543928902,0
1024,DWIGHT WILCOX,981 Laurel Street 49148,7571682264,0
```

The student's address is given as a house number, street name and zip code – our automated mailing systems obviate the need for us to store the city and state information explicitly. And, as you can imagine, some of the data that we are providing you has been "anonymized" (e.g. randomized) to protect the student's privacy rights – however, the formats are exactly accurate in accordance with the actual data files. Finally, please notice that a program ID of "0" means that the student has not selected a specific program yet.

Courses & Prerequisites

courses.csv: Our course data is contained in the ***courses.csv*** file. Like the records in the students.csv file, the records here are fixed length, and have three fields each. The fields are: (1) unique identifier over the course domain; (2) the course's (short) name; and, (3) the cost of the course per student per session. Please note that some courses will still be listed in the file even if they are not being offered currently – this happens occasionally due to various administrative reasons: lack of instructors, budgeting priority, etc. The Course Catalog from the original design discussions is given here:

Course ID	*Course Title*	*Prereqs*	*Cost*
2	Programming I		$100
3	Computer Architecture		$150
4	Programming II	2	$100
7	Algorithms and Data Structures		$100
8	Software Architecture and Design	4	$100
10	Machine Learning	2, 7	$200
13	Programming Languages		$100
14	Database Systems	7	$300
16	Operating Systems	3, 4, 7	$300

Here is an example of how this same information would be presented in a ***courses.csv*** file:

```
2,Programming I,100
3,Computer Architecture,150
4,Programming II,100
7,Algorithms and Data Structures,100
```

```
8,Software Architecture and Design,100
10,Machine Learning,200
13,Programming Languages,100
14,Database Systems,300
16,Operating Systems,300
```

Course descriptions with more detail might be important in the future, but we can do without them for now. Also, you might notice that the prerequisite information is missing from this file. It is contained in the related ***prereqs.csv*** file, which uses the course IDs to establish the "main course to prerequisite course" relationship.

One more key observation – you might notice that the course IDs and names have changed in comparison to the Course Catalog presented in earlier assignments. Please understand that our course listings are very flexible – names and IDs can be changed between test cases, so please not "hardcode" this information directly into your system. This is also true for student data, etc. Once we develop a more solid, running system, the IDs for courses, students, etc. will be more persistent over time. For now, though, the information submitted to your system in different test cases will likely change – treat every test case (set of files) independently.

prereqs.csv: The data provided in the ***prereqs.csv*** file provides information about which courses serve as prerequisites for other courses. Each record in this file has two fields of integers, and both field values represent the course identifiers for courses selected from the catalog. The course ID value in the first field means that that course requires the course ID represented in the second field as a prerequisite. All of the course ID values must refer to valid courses as listed in the ***courses.csv*** file. For example, the first line below represents the fact that Course 4 (Programming II) requires Course 2 (Programming I) as a prerequisite. Here is an example of the ***prereqs.csv*** file corresponding to the Course Catalog above:

```
4,2
8,4
10,2
10,7
14,7
16,3
16,4
16,7
```

Instructors

instructors.csv: Our instructor data is contained in the ***instructors.csv*** file. The format of the instructor file includes five fields: (1) unique identifier over the instructor domain; (2) the instructor's name; (3) the instructor's office hours; (4) the instructor's e-mail address; and, (5) the course that the instructor is teaching during this session. If the instructor is not teaching a course, then the value for the fifth field will be "0". Here is an example of an ***instructors.csv*** file:

```
2,EVERETT KIM,Fri_10_12,ekim2@learnondemand.edu,2
3,JOSEPH LE,Mon_Tue_9_10,jle17@learnondemand.edu,10
```

```
5,LAUREN HALEY,Thu_14_16,lhaley10@learnondemand.edu,4
6,JOSEPH LAWSON,Tue_Thu_14_15,jlawson31@learnondemand.edu,16
8,REBECCA CURRY,Wed_16_18,rcurry6@learnondemand.edu,3
```

Academic Records

records.csv: Academic records data is contained in the ***records.csv*** file. This data captures the final grades and evaluations for students who have taken our courses. Students are also allowed to take courses multiple times as needed, so incorporating the year and term information for each record is helpful for deconflicting multiple grades for the same course. Each record in this file will have five fields: (1) a unique identifier over the student domain; (2) a unique identifier over the course domain; (3) the student's final grade for that session of the course; (4) the year that the course was conducted; and, (5) the term that the course was conducted. The final grade will be represented in the file as a single-character string from the set {A, B, C, D, F}. The year will be an integer between 1950 and 2017, and the term will be represented by an integer from 1 through 4 (inclusively) to represent the Winter, Spring, Summer and Fall terms, respectively. Here is an example of the ***records.csv*** file:

```
1009,2,A,2016,3
1016,13,A,2016,2
1012,3,D,2016,2
1021,2,F,2016,2
1015,2,B,2016,3
1014,3,F,2016,3
1014,3,C,2017,1
1015,3,A,2016,3
1004,7,A,2017,1
1015,4,B,2017,1
1004,2,B,2017,2
1009,7,D,2016,4
1021,2,C,2016,4
```

Degree Programs & Course Requests

programs.csv: We have established degree programs for the students. Each degree program has a clear and distinct ID and name, and consists of a set of courses that must be successfully passed. A student may only enroll in one degree program at a time, but is permitted to switch between programs subject to certain restrictions. You will not need to be concerned about managing program selections and/or changes during this phase of the project. Each record in this file will have two fields: (1) a unique ID across the program domain; and, (2) the name of the program. Here is an example of a ***programs.csv*** file:

```
1,Software_Developer
2,Systems_Engineer
3,Data_Engineer
```

listings.csv: Since each program consists of one or more courses, this file represents the sets of courses that make up the different programs. Each degree program has a clear and distinct ID and name, and consists of a set of courses that must be successfully passed. Each record in this file will

have two fields: (1) a unique ID across the program domain; and, (2) the ID of a course that is a part of the program's requirements. Here's an example of the ***listings.csv*** file:

```
1,2
1,4
1,7
1,8
1,13
2,2
2,3
2,4
2,7
2,14
2,16
3,2
3,7
3,10
3,14
```

requests.csv: Finally, this last file represents the attempts of students to register for courses for this term. Each record in this file will have two fields: (1) a unique ID across the student domain; and, (2) the ID of the course in which the student wants to enroll. Here's an example of the ***requests.csv*** file:

```
1024,2
1012,3
1015,2
1009,10
1009,7
1004,10
1021,4
```

Analyzing the Input File Data

After the input files have been ingested, then your system shall analyze the data that has been collected and provide answers to the following questions:

- Which courses are being taught during this current term?
- Which students have passed courses and spent money on our courses?
- Which requests should be granted or denied; and, why should they be denied?

More specifically, your system shall display the following:

- A list titled "courses being taught" that consists of the courses, and only those courses, that are being taught by one or more instructors during this term. The list must be arranged in ascending order of the course IDs, and must include the instructor ID and name of at least one of the people teaching the course. You may select any one of the instructors for display if there are two or more instructors teaching the course.
- A list titled "student status and costs" that consists of the students, and only those students, who have paid any money taking courses. The list should also include the number of distinct courses that the student has passed, such that passing the same course multiple times should only be

counted once for that course ID. A student must have paid the cost for course, however, every time it was taken, regardless of passing or failing that specific session.

- A list titled "request processing" that consists of one line for each of the requests in the ***requests.csv*** file. The request shall be accompanied by the message "denied: missing prerequisites" if the student is missing one or more prerequisites; or, by the message "denied: not being offered" if there are no instructors teaching the course. Otherwise, the request must be considered valid, and accompanied by the message "granted". In the case that student is missing prerequisites and the course is also not being offered, then the "denied: missing prerequisites" message takes priority and must be displayed.

Remember the policies from the design discussions about passing grades: namely, that 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. As an example, the example below is based on the eight example files above, and lists the desired output for both content and format:

```
courses being taught
2: Programming I by 2: EVERETT KIM
3: Computer Architecture by 8: REBECCA CURRY
4: Programming II by 5: LAUREN HALEY
10: Machine Learning by 3: JOSEPH LE
student status and costs
1004: CAROL TAYLOR passed 2 course(s) for $200
1009: GARY ALLEN passed 1 course(s) for $200
1012: JULIE TURNER passed 0 course(s) for $150
1014: RENEE CARNEY passed 1 course(s) for $300
1015: JAMES FISHER passed 3 course(s) for $350
1016: TRACEY WHITE passed 1 course(s) for $100
1021: JEFFREY CLAYTON passed 1 course(s) for $200
request processing
1024, 2: granted
1012, 3: granted
1015, 2: granted
1009, 10: denied: missing prerequisites
1009, 7: denied: not being offered
1004, 10: granted
1021, 4: granted
```

The output shown above is captured in the ***answer.txt*** file in test case #1. The example files listed in these instructions are captured in test case #1, which is used as a sort of "base case" for the others as well. Test cases #2 through #5 are similar to test case #1, with some small changes made (in each case) to some of the files to exercise the different functional and policy requirements. You should match your output against these test cases to ensure that your output is in the correct format.

Other Requirements & Relevant Points:

- Our administrative assistants did a reasonable job of hastily assembling this test data for you, along with a few other test cases to ensure that you have reasonable examples of the formats. They were a bit "overzealous" in their efforts, though – they sorted the records for many of the

files in increasing order of the unique ID field. There is no guarantee that the records in the test files will be sorted in ascending ID (or any particular) order, so ***your program must be designed to handle records in any of the files in any order.***

- All IDs (at this point) are positive integers to simplify processing, and we will let you know in a future session if this policy/format changes.

- The test cases that we've provided so far are all fairly small, which has the advantage of allowing you to more easily verify the correct answers by hand. More importantly, they are not comprehensive, and they don't cover all of the requirements. We recommend that you develop your own test cases from scratch and/or by modifying the tests we've provided.
- To that end, you are permitted to create and share test cases with you fellow students. Strong emphasis here: you may share the test case files as described above along with a very basic statement describing the tests. However, do not include extra information about design and/or implementation specific details, etc. at the risk of sharing solution details in violation of academic conduct policies. If you are unclear if your descriptive test comments are "crossing the line", then feel free to share them privately with the TAs on Piazza for review before sharing them with other students.
- You will not have to test for empty files – all test files will have at least one record. And though it is generally a good idea, you are not required to check for formatting errors in the test files.

- We have discussed some of the expected growth rates for our program over the next few years with the school administrators and IT staff, and agree that we will probably need a more robust data storage solution for the future. For now, however, we will keep the sizes of the student, course and instructor content files under 500 records each; and, we will keep the academic records content files under 2000 records. Bottom line, you aren't required to implement a major database or data management system, etc. at this phase of the development process.

Submission Details:

- You must submit your source code in a ZIP file named ***source_code.zip*** with a project structure as described below. You must include all of your source *.java code – you do not need to include the (compiled) *.class files. Also, you must submit (separately) a non-obfuscate, runnable JAR file of your program named ***working_system.jar*** to aid in evaluating correctness.

- We will test your program by copying it into a directory with an existing test case, and then executing it at a (Linux) Terminal prompt with the command:
  `java -jar working_system.jar`

- One possible way to test your program is to use the **`diff()`** function. The diff function compares two files to identify the differences between them, and is usually implemented in most Linux/Unix and Mac OS systems. You can store the results of running your program on the test case, and then compare your results to the answers using the following commands:

```
java -jar working_system.jar > results.txt
diff system_output.txt results.txt
```

Windows OS system also normally include the **`FC()`** command that performs a similar function, as well as the opportunity to import the **`cgywin`** package or similar libraries. Please feel free to use the Linux **`man()`** command, or other Internet resources, to learn more.

- Your program should display the output directly to "standard output" on the Linux Terminal – not to a file, special console, etc. Also, your program may safely assume that the files will be correctly named, and will be located in the local/working directory where your JAR is currently located, as shown in the above examples. More importantly, we will copy your JAR file to different locations during our testing processes, so ***be sure that your program reads the test files from the current local directory***, and that your program is NOT hard-coded to read from a fixed directory path or other structure.

- We've provided a virtual machine (VM) for you to use to develop your program, but you are not required to use it for development if you have other preferences. We do recommend, however, that you test your finished system on the VM before you make your final submission. ***The VM will be considered the "execution environment standard" for testing purposes***, and sometimes-heard explanations of "…it worked on my (home) computer…" will not necessarily be sufficient.

- On a related note, it's good to test your finished system on a separate machine if possible, away from the original development environment. Unfortunately, we do receive otherwise correct solutions that fail during our tests because of certain common errors:
    - forgetting to include one or more external libraries, etc.
    - having your program read test files embedded files in its own JAR package;
    - having your program read files from a specific, hardcoded (and non-existent) folder

- Many modern Integrated Development Environments (IDEs) such as Eclipse offer very straightforward features that will allow you to create a runnable JAR file fairly easily. Also, please be aware that we might (re-)compile your source code to verify the functionality & evaluation of your solution compared to your submitted designs and source code.

But Wait… If You Order Now… Free "Utility" Code…

We've also include sample ***main.java*** and ***scratchpad.java*** files. You are welcome to compile, build and generate a JAR file from these files. Their main purpose of this program is to read the input files and display their contents to assure you that you are getting the complete and correct lines of data. You should create a package named ***edu.gatech*** under a new JAVA project in your IDE, and then copy the files to that location. Then you should be able to compile and build the project, and create an executable JAR as desired. As an example, if you run your JAR on the files given here (from test case #1), then this should be the output:

```
courses.csv
2, Programming I, 100
3, Computer Architecture, 150
4, Programming II, 100
7, Algorithms and Data Structures, 100
8, Software Architecture and Design, 100
```

```
10, Machine Learning, 200
13, Programming Languages, 100
14, Database Systems, 300
16, Operating Systems, 300
instructors.csv
2, EVERETT KIM, Fri_10_12, ekim2@learnondemand.edu, 2
3, JOSEPH LE, Mon_Tue_9_10, jle17@learnondemand.edu, 10
5, LAUREN HALEY, Thu_14_16, lhaley10@learnondemand.edu, 4
6, JOSEPH LAWSON, Tue_Thu_14_15, jlawson31@learnondemand.edu, 16
8, REBECCA CURRY, Wed_16_18, rcurry6@learnondemand.edu, 3
listings.csv
1, 2
1, 4
1, 7
1, 8
1, 13
2, 2
2, 3
2, 4
2, 7
2, 14
2, 16
3, 2
3, 7
3, 10
3, 14
prereqs.csv
4, 2
8, 4
10, 2
10, 7
14, 7
16, 3
16, 4
16, 7
programs.csv
1, Software_Developer
2, Systems_Engineer
3, Data_Engineer
records.csv
1009, 2, A, 2016, 3
1016, 13, A, 2016, 2
1012, 3, D, 2016, 2
1021, 2, F, 2016, 2
1015, 2, B, 2016, 3
1014, 3, F, 2016, 3
1014, 3, C, 2017, 1
1015, 3, A, 2016, 3
1004, 7, A, 2017, 1
1015, 4, B, 2017, 1
1004, 2, B, 2017, 2
1009, 7, D, 2016, 4
1021, 2, C, 2016, 4
requests.csv
```

```
1024, 2
1012, 3
1015, 2
1009, 10
1009, 7
1004, 10
1021, 4
students.csv
1004, CAROL TAYLOR, 215 4th Avenue 27517, 516479061, 3
1009, GARY ALLEN, 128 Pine Street 83866, 8304231126, 3
1012, JULIE TURNER, 927 6th Avenue 78553, 2587799053, 2
1014, RENEE CARNEY, 840 Main Street West 28729, 8118091235, 2
1015, JAMES FISHER, 231 Windsor Court 12288, 4477500021, 2
1016, TRACEY WHITE, 387 Canterbury Drive 49531, 4952312905, 1
1020, LILLIE LEWIS, 373 Magnolia Court 55751, 566944369, 0
1021, JEFFREY CLAYTON, 600 Bridle Lane 70941, 6222277693, 1
1022, SUE VELASQUEZ, 204 Riverside Drive 72894, 7543928902, 0
1024, DWIGHT WILCOX, 981 Laurel Street 49148, 7571682264, 0
```

This output is purely diagnostic – your system should not print these lines upon submission. You are welcome to use these files as you see fit, modify them, or not use them at all. They are "utility files" only – you must still implement all of the other java files representing the classes corresponding to your design of you proposed solution. Do not simply place all of your "solution" code into these two files as some sort of "monolithic" and unwieldy submission.

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.
Mark

Mark Moss, Lecturer
OMSCS 6310 – Software Architecture & Design
Instructor of Record
Georgia Tech, College of Computing