CCPROG3 MCO1 Program Specifications

Program Overview – Hotel Reservation System

Your task for MCO1 is to create a Hotel Reservation System (HRS). An actual HRS would have several functionalities that would be too complicated to implement for the course or would not be feasible within the given time frame; hence, the required HRS will be a scoped-down version that includes the following major functionalities:

1. **Create Hotel**
   In this feature, the user can input configurations needed to initialize a hotel. A hotel must have a name, which must be unique in relation to other hotels in the system. The system starts off with no hotels present and may have zero or multiple hotels at any given time.

   A hotel may have a minimum of one room and a maximum of fifty rooms. A room must have a name (e.g. 301, 302, A1, A2) to distinguish it from other rooms in the hotel. The naming convention can be left to the developers (i.e. developers can set their own automated naming scheme when initializing the rooms), but note that a room's name should be unique from other rooms in the hotel. A room must also have a base price per night, which is set to a default of 1,299.0. As there is only one type of room, the base price per night should be consistent across rooms. The base price across all rooms may be changed in the Manage Hotel feature.

   There may be zero or many reservations at any given time for a hotel, although when first created, a hotel should have zero reservations. For additional context, a reservation has a guest name, check-in date, check-out date, and have a link to the room's information. The reservation should also have information on the total price for the booking and a breakdown of the cost per night.

2. **View Hotel**
   In this feature, the user can view the current information found in a selected hotel. This would include high-level information of the hotel and low-level information based on what the user would like to see. See the following for more information:
   a. High-level hotel information should include the name of the hotel, total number of rooms, estimate earnings for the month (i.e. sum of total price across all reservations)
   b. Available low-level information should include the following:
      i. Total number of available and booked rooms for a selected date
      ii. Information about a selected room, such as the room's name, price per night, and availability across the entire month
      iii. Information about a selected reservation, such as the guest information, room information, check-in and -out dates, the total price for the booking, and the breakdown of the price per night

3. **Manage Hotel**
   In this feature, the user can modify the different configurations of the hotel. The user should be able to perform the following:
   a. Change the name of the hotel
      i. Unique naming convention should still be upheld
   b. Add room(s)
      i. Unique naming convention should still be upheld
   c. Remove room(s)
      i. A room can only be removed if it does not have an active reservation
      ii. For example, if there are ten rooms and rooms one to three have reservations, only rooms four to ten can be removed
   d. Update the base price for a room
      i. Updating the base price for a room can only be done if there are currently no reservations in the entire hotel since the room's price should be consistent across rooms.
      ii. The new price must be >= 100.0.
   e. Remove reservation
   f. Remove hotel

Developers are free to design the input process for any of the configurations to cater to unique characteristics of the system's design (i.e. how the system was modeled). However, users must be prompted to confirm a modification or else the modification would be discarded.

4. **Simulate Booking**
   In this feature, the user can simulate the process of booking a room. The user should be able to select a hotel and specify their check-in and check-out dates. Developers are free to design the mechanism that selects a room. This mechanism may be automated (e.g. select the first room, if available, return room, if not, check the next room) or manual (e.g. user selects from a list of rooms). The system should validate that the time frame does not violate any constraints (e.g. check-out on the 1st of the month, check-in on the 31st of the month). Once a reservation is made, the room's status should be updated, and the reservation details should be stored in the system and be viewable in the View Hotel feature.

Here are a few clarifications to help better understand the limitations/scope of the specifications:

- Floors in a hotel, the number of guests in a reservation, and the maximum number of guests per room do not need to be accounted for in this project.
- In real life, the check-in and check-out times are typically separate times in the day. For example, check-out might be at 11:00 am while check-in might be at 2:00 pm. The actual time-in and time-out values in our HRS aren't as important, but the system should allow for the reservation of a room to start (i.e. check-in) on the same day that another reservation of the same room ends (i.e. check-out).
- To make the calendar simpler, assume that we're working with a single month with 31 days. No reservations can be made when the check-out is on the 1st of the month or when the check-in is on the 31st of the month. Bookings cannot be made outside of the defined period for the month.
- Lastly, some items (particularly with how input is handled) are purposely left vague to allow students the freedom to design a solution that addresses the concern. Your initial model of the system would also affect the way your system would ask for input. You're encouraged to consult with your instructor.

Other Requirements

1. The design and implementation of the solution should…
   - Conforms to the specifications described above
   - Exhibit proper object-based concepts, like encapsulation and information-hiding
2. Interaction with the user (i.e. input and output) should be through a command line interface (CLI). No graphical interface is expected for MCO1.
3. To allow for an easier time to validate the program, usage of libraries outside of what is available in the Java API is not allowed.

Please also note that MCO2 will look to extend some mechanisms of MCO1; however, MCO2's specifications will be delayed in order to simulate additional mechanics or modifications requests by a client after development of the base system. Additionally, a graphical user interface should be implemented only for MCO2.

General Instructions

Deliverables

The deliverables for both MCOs include:

1. Signed declaration of original work (declaration of sources and citations may also be placed here)
   - See Appendix A for an example
2. Softcopy of the class diagram following UML notations
   - Kindly ensure that the diagram is easy to read and well structured
3. Javadoc-generated documentation for proponent-defined classes with pertinent information
4. Zip file containing the source code with proper internal documentation
   - The program must be written in Java
   - Test script following the format indicated in Appendix A
5. A video demonstration of your program
   - While groups have the freedom to conduct their demonstration, a demo script will be provided closer to the due date to help with showing the expected functionalities.
   - The demonstration should also quickly explain key aspects of the program's design found in the group's class diagram
   - Please keep the demo as concise as possible and refrain from adding unnecessary information

Submission
All deliverables for the MCO are to be submitted via **Canvas**. Submissions made in other venues will not be accepted. Please also make sure to take note of the deadlines specified on Canvas. No late submissions will be accepted.

Grading
For grading of the MCO, please refer to the MCO rubrics indicated in the syllabus or the appendix.

Collaboration and Academic Honesty
This project is meant to be worked on as a pair (i.e. max of 2 members in a group). In exceptional cases, a student may be allowed by their instructor to work on the project alone; however, permission should be sought as collaboration is a key component of the learning experience. Under no circumstance will a group be allowed to work on the MCO with more than 2 members.

A student cannot discuss or ask about design or implementation with other persons, with the exception of the teacher and their groupmate. Copying other people's work and/or working in collaboration with other teams are not allowed and are punishable by a grade of 0.0 for the entire CCPROG3 course and a case may be filed with the Discipline Office. In short, do not risk it; the consequences are not worth the reward[1]. Comply with the policies on collaboration and AI usage as discussed in the course syllabus.

Documentation and Coding Standards
Do not forget to include internal documentation (comments) in your code. At the very least, there should be an introductory comment and a comment before every class and every method. This will be used later to generate the required External Documentation for your Machine Project. You may use an IDE or the appropriate command-based instructions to create the documentation, but it must be PROPERLY constructed.

Please note that we're not expecting you to add comments for each and every line of code. A well-documented program also implies that coding standards are adhered to in such a way that they aid in the documentation of the code. Comments should be considered for more complex logic.

Bonus Points
No bonus points will be awarded for MCO1. Bonus points will only be awarded for MCO2. To encourage that usage of version control, please note that a small portion of the bonus points for MCO2 will be the usage of version control. Please consider using version control as early as MCO1 to help with collaborating within the group.

Resources and Citations
All sources should have proper citations. Citations should be written using the APA format. Examples of APA-formatted citations can be seen in the References section of the syllabus. You're encouraged to use the declaration of original work document as the document to place the citations.

Demonstration Delivery
The mode of delivery of the demo for MCO1 is left to the discretion of your instructor. All members are expected to be present in the video demonstration and should have relatively equal parts in terms of the discussion. Any student who is not present during the demo will receive a zero for the phase.

During the MP demo, it is expected that the program can be compiled successfully and will run. If the program does not run, the grade for that phase is 0. However, a running program with complete features may not necessarily get full credit, as implementation (i.e., code) will still be checked.

Other Notes
You are also required to create and use methods and classes whenever possible. Make sure to use Object-Based (for MCO1) and Object-Oriented (for MCO2) Programming concepts properly. No brute force solution. When in doubt, consult with your instructor.

Statements and methods not taught in class can be used in the implementation. However, these are left for the student to learn on his or her own.

---

[1] What is a measly passing grade compared to a life-long burden on your conscience?

Appendix A. Template for Declaration of Original Work

Declaration of Original Work

We/I, [Your Name(s)] of section [section], declare that the code, resources, and documents that we submitted for the [1st/2nd] phase of the major course output (MCO) for CCPROG3 are our own work and effort. We take full responsibility for the submission and understand the repercussions of committing academic dishonesty, as stated in the DLSU Student Handbook. We affirm that we have not used any unauthorized assistance or unfair means in completing this project.

[*In case your project uses resources, like images, that were not created by your group.*] We acknowledge the following external sources or references used in the development of this project:

1. Author. Year. Title. Publisher. Link.
2. Author. Year. Title. Publisher. Link.
3. Author. Year. Title. Publisher. Link.

By signing this declaration, we affirm the authenticity and originality of our work.

Signature and date	*Signature and date*
Student 1 Name ID number	Student 2 Name ID number

[*Note to students: Do not submit documents where your signatures are easily accessible. Ideally, submit a flattened PDF to add a layer of security for your digital signatures*]

Appendix B. Example of Test Script Format

Class: MyClass						
Method	**#**	**Test Description**	**Sample Input Data**	**Expected Output**	**Actual Output**	**P/F**
isPositive	1	Determines that a positive whole number is positive	74	true	true	P
	2	Determines that a positive floating point number is positive	6.112	true	true	P
	3	Determines that a negative whole number is not positive	-871	false	false	P
	4	Determines that a negative floating point number is not positive	-0.0067	false	false	P
	5	Determines that 0 is not positive	0	false	false	P

Appendix C. Rubrics for MCO1

Total: 100 points

Criteria	Exemplary	Satisfactory	Developing	Beginning	None
[Prerequisite] **100%**					Late submission of deliverables. OR Part or all of deliverable is plagiarized or not a product of student's output. OR No significant contribution to the group output. OR Did not appear during the demo. **0**
Program Correctness and Completeness	Program executes without errors, and properly provides all the functionalities of the object-based implementation. **40**	Program executes without errors, but is missing some minor features. **30-35**	Program executes with minor errors, or is missing significant features. **20 - 25**	Program executes with minor errors and is missing significant features. **10 - 15**	Program does not run OR Program does not produce any output. **0**
Designing Classes/Objects	Design decisions comply completely with the specs and all classes, attributes, behaviors, and relationships identified and shown in the UML class diagram are logical.	Design decisions comply completely with the specs as shown in the UML class diagram, but include unnecessary or redundant classes OR there is incomplete information on the attributes, behaviors, or relationships.	The design provides for most but not all of the original specs as shown in the UML class diagram. Most likely, include unnecessary or redundant classes AND there is incomplete information on the attributes, behaviors, or relationships.	The UML class diagram does not include most information based on the specs.	No submitted UML class diagram. OR Design of classes are not in compliance to a logical object-based design.

	20	15	10	5	0
Consistency of design and code	Program implementation corresponds correctly with the presented Object-based design. **10**	There are minor inconsistencies in design or implementation of attributes or methods, but all necessary features are still provided. **8**	There are minor inconsistencies in design or implementation of attributes or methods, which leads to missing or unexpected features. **5**	There are significant inconsistencies between design and implementation at the class level, but most features are still apparent. **3**	No submitted class diagram to compare with. **0**
Program Readability and Documentation	Coding standards prescribed in the course is followed. AND In-line comments are included for long codes, apart from proper documentation of methods (inclusive of pre-conditions, post-conditions, method parameters, and return data) via Javadoc. **10**	Coding standards prescribed in the course is followed. AND No in-line comments are included for long codes. But, there is proper documentation of methods (inclusive of pre-conditions, post-conditions, method parameters, and return data) via Javadoc. **8**	Coding standards prescribed in the course is followed. AND No in-line comments are included for long codes. There are method documentation via Javadoc, but are missing some information. **5**	Coding standards prescribed in the course is followed. AND No in-line comments are included for long codes. AND Method documentation is not via Javadoc annotation. **3**	No internal documentation. **0**
Test Case Design and Documentation	Apart from getters and setters, all methods in all classes are tested with at least 3 documented unique test cases. **10**	All methods in all classes are tested, but not all have at least 3 documented unique test cases. **8**	Most methods in all classes are tested with at least 3 documented unique test cases. **5**	Many methods do not have at least 3 documented unique test cases. **3**	No test script submitted. **0**
Delivery and Presentation	All members are present in the video presentation and have relatively equal parts to present. All members exhibit mastery of their project throughout the demo.	All members are present in the video presentation and have relatively equal parts to present. All members exhibit familiarity of their project throughout the demo.	All members are present in the video presentation; however, some members have little contribution to the demo. Most members exhibit familiarity of their project throughout the demo.	All members are present in the video presentation; however, some members have little contribution to the demo. Most members exhibit some knowledge of their project throughout the demo.	Did not appear in the demo. OR Member did not exhibit ample knowledge about the design AND implementation of

	10	**8**	**5**	**3**	the project. **0**