MCO2 Program Specifications

Program Overview – Extended Hotel Reservation System

Your task for MCO2 is to extend the current Hotel Reservation System (HRS) – described in the MCO1 specifications – by implementing the following mechanisms:

1. **Multiple Types of Rooms: Standard, Deluxe, and Executive**
   Instead of offering just a single type of room, customers can now select from three types of room when booking - Standard, Deluxe, and Executive based on availability. Each type differs in price:
   a. Standard room's rate is equivalent to the hotel's base rate,
   b. Deluxe is 20% more from the base rate, and
   c. Executive is 35% higher from the base rate.

   A hotel still has a minimum of one room and maximum of fifty rooms regardless of their types. Groups are free to design how a user would go about specifying rooms either when adding rooms or when creating a hotel.

2. **Discount Code for Reservations**
   When making a reservation, the user may input a discount code which would affect the computation of the total price in different manners. Here is a list of discount codes to implement:

   - **`I_WORK_HERE`** → Flat 10% discount to the final price of a reservation
   - **`STAY4_GET1`** → If the reservation has 5 days or more, the first day of the reservation is given for free
   - **`PAYDAY`** → This gives a 7% discount to the overall price if reservation covers (but not as checkout) either day 15 or 30. Can be used in bookings such as:
     - Check-in: 15, Check-out: 16 **BUT NOT** Check-in: 14, Check-out: 15
     - Check-in: 28, Check-out: 31 **BUT NOT** Check-in: 28, Check-out: 30

   These codes are not automatically applied just because a condition was satisfied (e.g. reservation is 5 days OR includes day 15 OR 30). The code must still be provided. For simplicity, there is no limit to the number of times a discount code can be used but only one can be used per reservation. Discount codes are also case sensitive.

3. **Date Price Modifier**
   It is important to note that the price for a single night is the base price of the room multiplied by the price rate. The Data Price Modifier feature allows the system to make certain days cost more or less than others – as if to give them a premium or to make them more enticing.

   By default, the hotel sets the price rate of all dates in the month to 100% implying the total price of the room (depending on the type) for one night would be the base price of the room (i.e. base_rate * 1.0). Through the date price modifier, the system may change a date to have 50% to 150% the base price of a room.

   To better understand the modifier, let's imagine a scenario where the management would like to bump the prices for staying the nights of the 5th and 6th days of the month to 110%. We also modify the 8th day to 90%. Here are a few examples of reservations on specific days of the month using this scenario:

Example 1	**Example 2**	**Example 3**
Stay: 3rd day to 6th day	**Stay**: 5th day to 9th day	**Stay**: 5th day to 6th day
Breakdown: • 3rd-4th -> 100% • 4th-5th -> 100% • 5th-6th -> 110%	**Breakdown**: • 5th-6th -> 110% • 6th-7th -> 110% • 7th-8th -> 100% • 8th-9th -> 90%	**Breakdown**: • 5th-6th -> 110%

Groups are free to design how the user selects days and modifies their date price modifiers.

4. **Graphical User Interface**
   Developing a graphical user interface (GUI) is one way to offer an application that is easier for users to understand and navigate. This includes providing window frames that can contain buttons, text input fields, and other onscreen components for each operation of your application. You may utilize any GUI components readily available in Java. All graphical components in Java belong to a common inheritance hierarchy, so they share a common set of methods that can be used to get and set properties such as background color, size, and font.

   In case you decide to design a more artistic GUI requiring more complex artworks, remember that it is not required to be created from scratch. You may retrieve or use already existing digital artworks as long as you cite them accordingly.

   Kindly take note that some IDEs provide drag-and-drop functionality for creating GUIs. While you're allowed to explore these tools, note that these tools are known for creating spaghetti code – making it harder to understand and manipulate the underlying code. Also, some IDEs (e.g. Netbeans) provide GUI builders that utilize their own GUI APIs. You're asked to refrain from using such arrangements mainly because of the requirement to stick to what is available in the Java API.

   Note: Your program must be implemented with GUI elements. Failure to do so will result in the program not being checked. The GUI component in the rubric is with respect to *usability* – not generally implementation of GUI.

5. **Model-View-Controller Architecture**
   The Model-View-Controller or MVC is an architectural design pattern that allows the application to be divided based on defined roles. To promote a more organized and easily maintainable source code, MVC architecture should be followed.

You're expected to use your output for MCO1 as a basis for your MCO2; however, depending on how you approached, MCO1, you might be forced to overhaul portions of your design and/or implementation. This is intended to get you to think about how design might accommodate new features in an existing system. While there is no MCO3 (i.e. no further extensions), everyone is encouraged to approach MCO2 by looking at how object-oriented concepts could make the system more flexible to change.

Other Requirements

1. Implementation should simulate real-world objects.
   - That is, the number of hotel rooms created belonging to a hotel is not just an attribute quantity but rather actual instances.
2. Design and implementation should maximize (but not over use) object-oriented concepts, like inheritance, polymorphism, and method overloading/overriding.
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.

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
   - The class diagram should exhibit the appropriate object-oriented programming principles
   - For MCO1, the class diagram only needs to depict the requirements for MCO1
   - For MCO2, the class diagram should be complete and only depict the Model components
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

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].

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 command prompt command javadoc to create this 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
An MCO2 submission can receive at most 10 bonus points based on additional features if and only if the submission has met the basic requirements (i.e. exemplary for program correctness and design of classes and their relationships). Bonus points will only apply to MCO2.

Awarded points may vary based on the complexity of the additional feature. Assets that make the MP more engaging or that add to the overall user experience may also be awarded points. Groups have the freedom to add bonus features as they see fit. However, added features must not permanently alter the base requirements of the project. There should always be a quick and easy way to test the minimum requirements in the final submission, despite any added features. If a bonus feature needs to alter the base requirements, the group must provide a setting that will momentarily disable such alterations. Not being able to test the expected base requirements will most likely result in deductions to the submission's score, as well as the bonus points not being counted.

To encourage the usage of version control (such as Git), up to 4 bonus points may be awarded. While the usage of version control will not be taught in this course, you are encouraged to organize and store your code via a Git repository, like services offered by GitHub. Utilizing some form of version control will make it easier for the members of the group to collaborate with each other and the commit history also helps in providing some form of accountability. Awarded points may vary based on how the group was able to leverage the usage of version control (e.g. small and often commits, descriptive commit comments, contributions from all members, branching).

Note that bonus points are capped at 10 points. For example, if a group was awarded 10 points for the implementation of additional features and 4 points for effective usage of version control, only 10 points will be awarded. Lastly, do not waste your effort on additional points when the project requirements have not been met!

Resources and Citations
We would just like to emphasize that you DO NOT need to create your own sprites or images (background pictures, item/product pictures, etc.) for the simulation. You can just use what's available on the Internet and just include them in your project. If you wish to do so, we would also like to remind you to please cite where you got your sprites.

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.

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

Demonstration
The MCO2 demo is expected to be conducted live (whether F2F or live). During the demo, all members of the group should be present and follow the instructions given by the instructor. Apart from Q&A, a demo problem will be given to the group as part of the demo and the problem is to be solved individually.

A student or a group who is not present during the demo or who cannot answer questions regarding the design and implementation of the submitted project convincingly will incur a grade of 0 for that project 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 and Object-Oriented Programming concepts properly. No brute force solution.

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.

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**
isPositiv e	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. Rubric for MCO2
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-oriented 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. AND Design is following Model-View Controller. **15**	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. AND Design is following Model-View Controller. **10**	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. OR Design is not following Model-View Controller. **6**	The design provides some of the original specs and includes unnecessary or redundant classes. AND Design is not following Model-View Controller. **3**	No submitted UML class diagram. OR Design of classes are not in compliance to a logical object-oriented design. **0**
Designing Class Relationships	Design decisions comply completely with the specs and all class relationships	Design decisions comply completely with the specs and all class relationships are	The design provides for most but not all of the original specs, or some some information is	The design provides for most but not all of the original specs or some	No submitted UML class diagram OR

	are logical and provide scalability to the system, as depicted in all information in the UML class diagram. **5**	logical, as depicted in complete information in the UML class diagram. **4**	missing in the UML class diagram. **3**	class relationships are unnecessary or redundant. **2**	Design of relationships are not in compliance to a logical object-oriented design. **0**
Consistency of design and code	Program implementation corresponds correctly with the presented OO design. **5**	There are minor inconsistencies in design or implementation of attributes or methods, but all necessary features are still provided. **4**	There are minor inconsistencies in design or implementation of attributes or methods, which leads to missing or unexpected features. **3**	There are significant inconsistencies between design and implementation at the class level, but most features are still apparent. **2**	No submitted class diagram to compare with. **0**
GUI Usability	The Graphical User Interface is user-friendly and shows mastery and proper use of the API. **5**	The Graphical User Interface is usable and shows mastery and proper use of the API. **4**	The Graphical User Interface is usable but features some improper use of a few API components. **3**	The Graphical User Interface works properly but is difficult to use. Also, it exhibits some improper use of a few API components. **2**	Some of the GUI does not work properly. **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	All methods in all classes are tested, but not all have at least	Most methods in all classes are tested with at least 3 documented unique test cases.	Many methods do not have at least 3	No test script submitted.

	least 3 documented unique test cases. **10**	3 documented unique test cases. **8**	 **5**	documented unique test cases. **3**	 **0**
Program Debugging	Solved the demo problem successfully in the given duration. **10**	Minor errors or discrepancies in expected output resulted after testing the solution to the demo problem. **8**	Some cases were not considered in the solution to the demo problem, thus resulted to errors in the result. **5**	Many cases were not considered in the solution to the demo problem. **3**	Did not appear during the demo. OR Cannot solve the demo problem in the given duration. **0**