GREEN PROPERTY EXCHANGE

A Property Management Simulation

I. Introduction

In the bustling digital economy of the Philippines, sustainable property sharing has emerged as both an environmental initiative and a lucrative business opportunity. Green Property Exchange represents the next generation of property rental platforms, where environmentally conscious property owners connect with eco-minded travelers and long-term renters.

II. The Challenge

The student assumes the role of a software developer tasked with creating an innovative property management system for Green Property Exchange, a fictional start-up focused on sustainable tourism in the Philippines. The challenge requires developing a system that demonstrates mastery of object-oriented design principles. The student will evolve the system into a GUI application that leverages inheritance and polymorphism, starting with a text-based prototype using basic object-based concepts. Throughout this development process, the student must architect a solution that meets current specifications and anticipates future extensions through proper encapsulation and abstraction, proving their ability to create maintainable, quality software.

III. Minimum Requirements

MCO1 Requirements

1. Create Property Listing.

   In this feature, the user can input configurations needed to initialize a property listing. A property must have:

   - A unique name relative to other properties in the system. The system starts with no properties present and may have zero or multiple properties at any given time.
   - A property must have at least one available date and a maximum of thirty available dates (representing a whole month).
   - A date must have a number (e.g., 1, 2, 3... 30) to distinguish it from other dates in the property's availability calendar.
   - A date must also have a base price per night, which is set to PHP 1,500.00 by default. There is only one type of property in MCO1, so the base price per night should be consistent across all dates. However, the Manage Property feature may change the base price across all dates.
   - A property may have zero or many reservations at any given time, although it should have zero when first created.

2. View Property

This feature allows the user to view current information about a selected property, including high-level and detailed information, based on what the user wants. These include the following:

- Calendar View - The visual month calendar should display:
  - Available dates (clearly marked as bookable)
  - Reserved dates (showing guest name or "BOOKED" status)
  - Date numbers (1-30) in a grid format similar to a planner
  - Current base price per night for reference

- High-level property information as follows:
  - Property's name
  - Total number of available dates
  - Estimated earnings for the month (i.e., sum of total price across all reservations).

- Detailed information as follows:
  - Total number of available and booked dates for a selected date range.
  - Information about a selected date, such as the date number, price per night, and availability status.
  - Reservation information, such as the guest's and property's information, check-in and check-out dates, the total price for the booking, and the price breakdown per night.

3. Manage Property

In this feature, the user can modify the property's different configurations. The user should be able to perform the following:

- Change the property's name (The system should still enforce the rule of maintaining a unique naming convention).
- Update the base price per night.
- Updating the base price can only be done if there are no reservations on the entire property, since the price should be consistent across dates.
- The new price must be >= PHP 100.00.
- Remove the reservation.
- Remove property. (The system should only allow removal of properties with no active reservations.)

4. Simulate Booking

In this feature, the user can simulate booking a property.

- The user should be able to select a property and, at a minimum, specify the following:
  - Guest name
  - Check-in date
  - Check-out date

- The system should validate that the reservation time frame does not violate any constraints:
    - No check-out on the 1st of the month
    - No check-in on the 30th of the month
    - Booking dates do not conflict with existing reservations.

- The reservation should also have information on:
    - The total price for the booking.
    - The breakdown of the cost per night.

- Upon successful reservation, the system should:
    - Update the property's availability
    - Record reservation details.
    - Make the reservation viewable (see View Property feature above).

Clarifications for MCO1:

- *To simplify the calendar, assume we're working with a month with 30 days.*
- *Reservations cannot have check-out on the 1st day or check-in on the 30th day.*
- *In real life, check-in and check-out times are typically separate times in the day. Our system should allow a property reservation to start (i.e., check-in) on the same day that another reservation of the same property ends (i.e., check-out).*

MCO 2 Requirements

The task of the student for MCO2 is to extend the current Property Management System – described in the MCO1 specifications – by implementing the following mechanisms:

1. Multiple Types of Properties

   Instead of offering just a single property type, guests can now select from four types of properties, namely:
   - Eco-Apartment
   - Sustainable House
   - Green Resort
   - Eco-Glamping

   Each type differs in price.
   - Eco-Apartment's rate is equivalent to the property's base rate.
   - Sustainable House is 20% more than the base rate.
   - Green Resort is 35% higher than the base rate.
   - Eco-Glamping is 50% higher than the base rate.

A property owner or user can now list multiple property types. The student can design how a user would specify property types when adding or creating a listing.

2. Environmental Impact Pricing Modifier

   The Environmental Impact Pricing Modifier feature allows the system to make specific dates cost more or less than others – as if to reward eco-friendly behaviors or account for environmental costs.

   By default, the property sets the environmental impact rate for all dates in the month to 100%, implying that the total price for one night would be the property's base price (i.e., base_rate * 1.0).

   Through the environmental impact modifier, the system may change a date to have 80% to 120% the base price of a property.

   To better understand the modifier, let's imagine a scenario where management would like to reduce prices for staying the nights of Earth Day (22nd) and World Environment Day (5th) to 90%. We also modify high-pollution days (15th and 30th) to 110%. Here are a few examples of reservations on specific days of the month using this scenario:

   Example 1: Stay from the 4th day to the 7th day (Calendar days)

   Breakdown:
   4th-5th → 100% (Day 1)
   5th-6th → 90% (Day 2, Environment Day)
   6th-7th → 100% (Day 3)

   Example 2: Stay from 21st day to the 23rd day

   Breakdown:
   21st-22nd → 100% (Day 1)
   22nd-23rd → 90% (Day 2, Earth Day)

   Example 3: Stay from 28th day to the 30th day

   Breakdown:
   28th-29th → 100% (Day 1)
   29th-30th → 110% (Day 2, Hi-pollution days)

   The student can design how users select dates and modify their environmental impact pricing modifiers.

3. Enhanced Calendar View with Environmental Impact Pricing

The calendar view from MCO1 is enhanced to display environmental impact pricing modifiers and implemented using a GUI. The visual monthly calendar now shows:

- Available dates with color-coded environmental impact indicators:
  - Green dates: Reduced environmental impact (80-89% of base price)
  - White dates: Standard environmental effects (100% of base price)
  - Yellow dates: Increased environmental impact (101-120% of base price)
- Reserved dates continue to show booking status
- Price per night displayed for each date reflects the environmental modifier
- Environmental impact legend explaining the color coding and pricing

IV. Milestones (see section V for Deliverables)

a. Initial Design – due 9:00 pm, October 3 (F)

1. Draft UML Diagram
2. Screen Navigation prototype (Wireframe)

b. MCO1 – due 9:00 pm, October 24, 2025 (F)

1. UML Class Diagram covering MCO 1 requirements (Model Layer).
2. Java implementation of the MCO 1 specifications
3. No GUI display is expected. However, displays must be made in the Console.

c. MCO2 – due 12:00 nn, November 24, 2026 (M)

1. Complete UML Diagrams (Object-Oriented) covering MCO 1 and MCO 2 requirements.
2. GUI with Mouse-Controlled Inputs.
3. Complete implementation of the application
4. Implementation following the SOLID design principles

V. Deliverables

The deliverables for both MCOs include:

1. The design and implementation of the solution should:
   - Conform to the specifications described above
   - Exhibit proper object-based / object-oriented concepts, like encapsulation and information-hiding, etc.
   - **NOT** be derived or influenced from any use of Generative AI tools or applications
2. Signed declaration of original work (declaration of sources and citations may also be placed here)
   - See **Appendix A** for an example
3. Softcopy of the class diagram following UML notations (in pdf or png)
   - Kindly ensure that the diagram is easy to read and well structured
4. For CCPROG3 students, Javadoc-generated documentation for proponent-defined classes with pertinent information
5. Zip file containing the source code with proper internal documentation
   - The program must be written in Java (for CCPROG3 students) or C++ (for GDPROG3 students).

- Include external libraries that were used (e.g., JavaFX)

6. Test script following the format indicated in **Appendix B**
   - In general, there should be at least 3 categories (as indicated in the description) of test cases per method (except for setters and getters).
   - There is no need to test user-defined methods which are ONLY for screen design (i.e., no computations/processing; just print/println).
7. For MCO1 only: 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
8. Groups should back-up their projects. A softcopy of the final unmodified files (for each phase) should be sent to their own emails, apart from regular submissions of progress in AnimoSpace and/or Git.

VI. Submission

All deliverables for the MCO are to be submitted via AnimoSpace. Submissions made in other venues will not be accepted. Please also make sure to take note of the deadlines specified on AnimoSpace. Late submissions will not be accepted.

VII. Grading

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

VIII. Collaboration and Academic Honesty

This project is meant to be developed as a pair for both phases (i.e., MCO1 and MCO2) barring any reports of freeloading or decision by the pair to split. 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. Questions about the project specifications should be raised in the Discussion page in AnimoSpace. 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 subject and a case may be filed with the Discipline Office. In short, do not risk it; the consequences are not worth the reward. Comply with the policies on collaboration and AI usage as discussed in the syllabus.

IX. 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. For those implementing using Java, this will be used later to generate the required External Documentation for your project via javadoc. You may use an IDE or

the appropriate command-based instructions to create the documentation, but it must be PROPERLY constructed.

Please note that you're not expected to provide 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.

X. Bonus Points

Bonus points will only be awarded for MCO2. The above description of the program is the basic requirement. Any additional feature will be left to the creativity of the group. Bonus points would be awarded depending on the additional implemented features. These additional features could include:

- Implementation of a program feature using a Strategy Design Pattern.
- Add a Guest Loyalty Program to allow different price points to different types of guests.
- Implement an object-level data persistence feature.

Depending on the scale of the new feature, additional points will be awarded to the group. However, make sure that all the minimum requirements are completely and correctly met first; if this is not the case then no additional points will be credited despite the additional features. To encourage the 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 group collaboration.

XI. 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.

Further, this is to emphasize that you DO NOT need to create your own sprites (background pictures, walls, characters, bombs, etc.) for the game. You can just use what's available on the Internet and just include these into your project, just make sure to cite your sources.

XII. Demo

Demo for MCO1 is via a video submission. 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.

In MCO2, demo is live and will include an individual demo problem. Schedule for the demo will generally be during class time, but there may be other schedules opened by the faculty in case there is not enough time to accommodate everyone. Sequence (of who goes first in the class to do the demo) is determined by the faculty. Thus, do not be absent or late during announced demo days. 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 demo, it is expected that the program can be compiled successfully in the command prompt 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.

XIII. 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.

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 GD/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:*

1. *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*
2. *You may use the eSignature feature of Google Docs. See this template.*]

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