Assignment 2 - Hotel Management System

Programming Fundamentals

Professor Areej Abdulfattah

Hamad Alkhazraji

202205473

Introduction

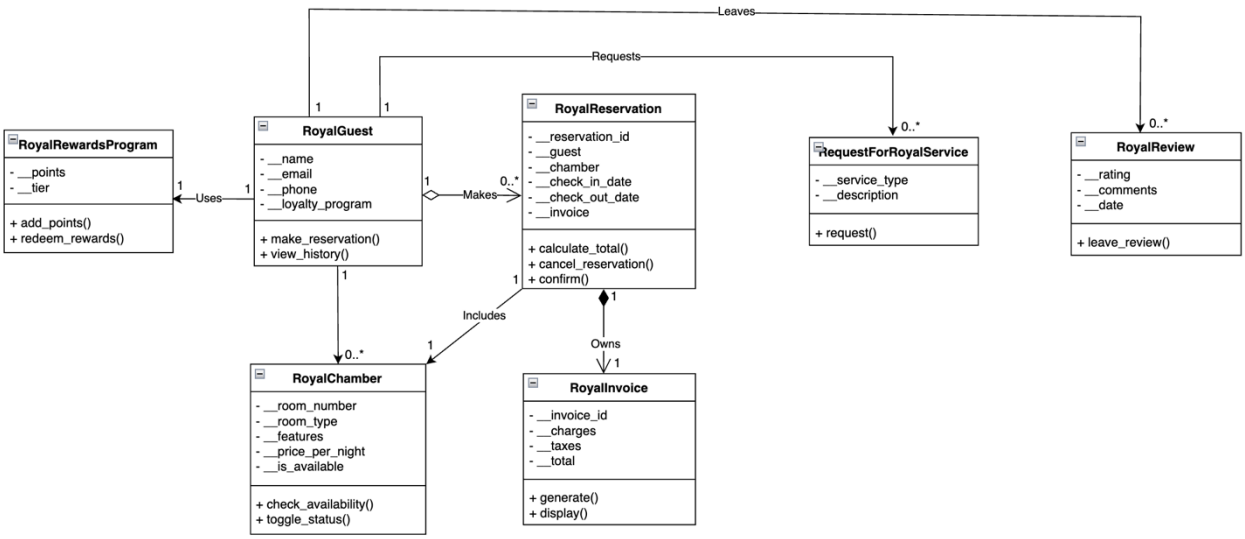
This assignment is about designing and building a software system for a hotel called Royal Stay. The goal is to help the hotel improve how it manages rooms, bookings, guest information, payments, loyalty rewards, service requests and reviews. As part of the Programming Fundamentals course, this project is a way to apply what we have learned about object-oriented programming and UML in a real-world context.

The system must be able to handle important hotel features like adding and managing rooms, creating guest accounts, booking rooms, generating invoices, tracking loyalty points, making service requests and collecting feedback. To build it well, we are expected to follow OOP concepts like classes, objects, encapsulation, inheritance and good code structure. The assignment also requires that we clearly show the design using a UML class diagram, write clean and error-free Python code, test the system with real examples and prepare a well-written report.

To make the project creative and fun, I decided to follow a royal luxury theme. So, instead of plain names like “Guest” or “Room,” I used special names like RoyalGuest, RoyalChamber and RoyalInvoice. The methods also follow this style, for example, earn_loyalty_crowns() adds points to a guest’s loyalty program in a way that matches the royal theme.

In this report, I will explain how I planned the system, designed the UML diagram, wrote the code, created test cases and documented everything. The steps I followed include understanding the problem, choosing the right classes, showing their relationships, coding each part, testing all the features and putting everything together clearly.

UML Relationship Notations – Explained



This UML class diagram represents the object-oriented design of the Royal Stay Hotel Management System. It visually represents the structural relationships among the primary classes like RoyalGuest, RoyalChamber, RoyalReservation, RoyalInvoice, RoyalRewardsProgram, RequestForRoyalService and RoyalReview.

Symbol	Type	Description
◇	Aggregation (Has-A)	A weaker whole-part relationship. The whole can exist without the part and the part can also exist independently.
◆	Composition (Owns-A)	A stronger whole-part relationship. The part cannot exist without the whole. When the whole is destroyed, so is the part.
▲	Inheritance (Is-A)	A generalization-specialization relationship. The child class inherits attributes and methods of the parent (Not applicable to mine).
→	Association	A general link between classes where objects interact but exist independently.
+	Public	Attribute or method is visible and accessible from anywhere.
-	Private	Attribute or method is only accessible within the class. Follows the encapsulation principle.
#	Protected	Attribute or method is accessible within the class and its subclasses.

Class Descriptions & Relationships

Each class in the Royal Stay Hotel Management System was carefully planned using key ideas from Object-Oriented Analysis and Design (OOAD). These include important OOP concepts like:

- **Abstraction**: hiding complex details and showing only what is needed
- **Encapsulation**: protecting data by using private or protected attributes
- **Association**: linking different classes together
- **Inheritance**: letting one class get features from another
- **Modularity**: breaking the system into smaller and reusable parts

Below is a table that describes the main classes in the system their roles and how they fit into the overall design.

Class Name	Role & Description
RoyalGuest	This class represents a hotel guest. It stores personal details like name and contact info and also tracks loyalty points. Guests can update their profile, make bookings and earn rewards through this class.
RoyalChamber	This is the hotel room class. Each room called a chamber has a type like deluxe or suite, special features, price per night and availability status. This helps the system manage which rooms are open for booking.
RoyalReservation	This class handles room bookings. It stores booking dates, guest and room info and connects to invoices and cancellations. It plays a role in managing how guests reserve rooms.
RoyalInvoice	An invoice is created when a reservation is confirmed. This class keeps a breakdown of charges like room cost, service fees and taxes. It helps guests know exactly what they are paying for.
RoyalRewardsProgram	This class manages the hotel's loyalty program. It lets guests earn and use loyalty points called Royal Crowns and connects with reservations to update rewards automatically after a stay.

RequestForRoyalService	Guests can use this class to make extra service requests like ordering room service, requesting cleaning or asking for transport. The requests are handled by hotel staff.
RoyalReview	After a stay, guests can leave feedback and ratings. This class stores those reviews which the hotel can use to improve services and future guests can use to read experiences.

Class Relationships

In Object-Oriented Programming and Design, class relationships show how different classes in a system connect and work together. These relationships help make the design more organized and realistic by mapping how real-world entities interact. In this system, we've used several key types of relationships:

Types of Relationships Explained

- **Inheritance**: This is when one class (child) shares the properties and behaviors of another class (parent). It is used when there is a clear hierarchy.
- **Aggregation**: This is a has-a relationship where one class is made up of other classes but those parts can still exist on their own.
- **Composition**: This is a stronger part-of relationship where the part cannot exist without the whole.
- **Association**: This is a general connection between two classes where they are linked but there is no ownership.
- **Multiplicity** (One-to-One, One-to-Many, etc.): This shows how many objects of one class can relate to another.

Key Relationships in the System

Relationship Type	Example	Explanation
Inheritance	None	No inheritance is used because each class is unique and does not inherit any properties from other classes.
Aggregation	RoyalGuest ◇—— RoyalReservation	A guest can have many reservations but even if a reservation is canceled, the guest still exists. This is a has-a but independent relationship.
Composition	RoyalReservation ◆—— RoyalInvoice	A reservation owns its invoice. If the reservation is deleted, the invoice is also deleted. This is a strong relationship with full dependency.
Association	RoyalGuest ↔ RoyalChamber (via reservation)	A guest stays in a chamber but this connection is managed through a reservation. It is a basic link without ownership.
One-to-Many	RoyalGuest → Multiple RoyalReservation	One guest can make many reservations. This is a common setup for booking systems.
One-to-One	RoyalReservation → One RoyalInvoice	Each reservation creates exactly one invoice. They are connected.

Design Justification

Aspect	Explanation
Modularity & Structure	Each class is placed in a separate file to illustrate real-world responsibilities, following professional code design. This improves readability, makes debugging easier and allows for collaboration.
Encapsulation	All data is protected using private/protected attributes. Controlled access using getter/setter methods ensures data is used safely and correctly, following the principles of encapsulation.

Naming & Themes	The project uses royal-themed names and consistent formatting across all classes, methods and printed outputs. This creative touch enhances user engagement and aligns with the chosen theme.
Object-Oriented Principles Used	The system applies abstraction by simplifying complex hotel operations into clear objects, encapsulation by data protection, modularity by clear separation of concerns and association/aggregation/composition to represent real-world relationships.
Scalability & Reusability	The current design allows for easy extension, new features like a restaurant module or spa services can be added without rewriting the original system. Reusability is achieved by keeping methods general and flexible.
Real-World	Each class closely models a real-world hotel component like invoices are linked to bookings, service requests are tied to guests and loyalty rewards reflect repeat customer behavior. This makes the code realistic and practical.
Readability & Documentation	Each class and method includes docstrings, comments and clean formatting. This ensures any future coder or instructor can easily read, understand and work with the code.
Input Validation & Error Handling	Input is checked and validated where appropriate. If errors occur, messages are displayed rather than crashing the program which enhances the code and its flexibility.

Python Codes

Class 1: RoyalGuest

```
# Guest class to hold personal info and their reservations
class RoyalGuest:
    def __init__(self, name, email, phone):
        self.__name = name
        self.__email = email
```

```
        self.__phone = phone

        # Composition: has a loyalty program
        self.__loyalty_program = RoyalRewardsProgram()

        # List of reservations made by the guest
        self.__reservations = []

    # Adds a new reservation to the guest's history
    def make_reservation(self, reservation):
        self.__reservations.append(reservation)

    # View list of all past reservations
    def view_history(self):
        return self.__reservations

    # Get guest's name
    def get_name(self):
        return self.__name

    # Access guest's loyalty program
    def get_loyalty(self):
        return self.__loyalty_program

    # Displays guest information
    def __str__(self):
        return f"👑 Guest: {self.__name} | {self.__email} | Loyalty: {self.__loyalty_program}"
```

Class 2: RoyalChamber

```
# Defines room types in a clear way as pre-determined choices
from enum import Enum

# Room types that are available in the hotel
class RoomType(Enum):
    SINGLE = "Single"
    DOUBLE = "Double"
    SUITE = "Suite"

# RoyalChamber represents a hotel room
class RoyalChamber:
    def __init__(self, room_number, room_type: RoomType, features, price_per_night):

        # Private attributes to hold room details
        self.__room_number = room_number
        self.__room_type = room_type

        # List of features/amenities
        self.__features = features
        self.__price_per_night = price_per_night

        # Starts as available
        self.__is_available = True

    # Getter for room number
    def get_room_number(self):
```

```
        return self.__room_number

    # Getter for room price
    def get_price(self):
        return self.__price_per_night

    # Check if room is available
    def is_available(self):
        return self.__is_available

    # Alternative method for availability check
    def check_availability(self):
        return self.__is_available

    # Changes availability status from True to False or False to True
    def toggle_status(self):
        self.__is_available = not self.__is_available

    # Returns formatted room information
    def __str__(self):
        return f"Chamber {self.__room_number} [{self.__room_type.value}] – AED
{self.__price_per_night}/night"
```

Class 3: RoyalReservation

```
# Handles check-in and check-out dates to ensure its similar to an actual real life scenario
from datetime import date
```

```
# Manages reservation process: guest, room, dates and invoice
class RoyalReservation:
    def __init__(self, reservation_id, guest: RoyalGuest, chamber: RoyalChamber, check_in_date: date, check_out_date: date):
        self.__reservation_id = reservation_id
        self.__guest = guest
        self.__chamber = chamber
        self.__check_in_date = check_in_date
        self.__check_out_date = check_out_date
        self.__invoice = None

    # Calculates number of nights * room price
    def calculate_total(self):
        nights = (self.__check_out_date - self.__check_in_date).days
        return nights * self.__chamber.get_price()

    # Cancels the reservation and makes the room available again
    def cancel_reservation(self):
        self.__chamber.toggle_status()

    # Confirms booking, generates invoice and adds loyalty points for the guest
    def confirm(self):

        # Marks room as booked
        self.__chamber.toggle_status()
        charges = self.calculate_total()
        taxes = round(charges * 0.1, 2)
```

```
        self.__invoice = RoyalInvoice(f"INV-{self.__reservation_id}", charges, taxes)
        self.__invoice.generate()
        self.__guest.make_reservation(self)
        self.__guest.get_loyalty().add_points((self.__check_out_date - self.__check_in_date).days)

    # Returns the invoice linked to this reservation
    def get_invoice(self):
        return self.__invoice

    # Summary of reservation
    def __str__(self):
        return f"📅 Reservation {self.__reservation_id} | Guest: {self.__guest.get_name()} | Room: {self.__chamber.get_room_number()}"
```

Class 4: RoyalInvoice

```
# Represents the invoice generated for each reservation
class RoyalInvoice:
    def __init__(self, invoice_id, charges, taxes):
        self.__invoice_id = invoice_id
        self.__charges = charges
        self.__taxes = taxes

        # Total will be calculated later
        self.__total = 0

    # Generates total by adding charges + taxes
    def generate(self):
```

```
        self.__total = self.__charges + self.__taxes

    # Shows invoice details
    def display(self):
        return f"🧾 Invoice #{self.__invoice_id}: Charges: AED {self.__charges}, Taxes: AED {self.__taxes}, Total: AED {self.__total}"

    # Returns the total bill
    def get_total(self):
        return self.__total
```

Class 5: RoyalRewardsProgram

```
# Manages guest loyalty points and reward tier
class RoyalRewardsProgram:
    def __init__(self):

        # Start with 0 points
        self.__points = 0
        self.__tier = "Bronze"

    # Adds points based on number of nights stayed
    def add_points(self, nights):
        self.__points += nights * 10
        self.__update_tier()

    # Redeem rewards if the guest has 100 or more points
    def redeem_rewards(self):
```

```
        if self.__points >= 100:
            self.__points -= 100
            return True
        return False

    # Updates guests tier based on total points
    def __update_tier(self):
        if self.__points >= 300:
            self.__tier = "Gold"
        elif self.__points >= 150:
            self.__tier = "Silver"
        else:
            self.__tier = "Bronze"

    # Get the guest's current tier
    def get_tier(self):
        return self.__tier

    # Get current points balance
    def get_points(self):
        return self.__points

    # String to show points and tier
    def __str__(self):
        return f"Tier: {self.__tier}, Points: {self.__points}"
```

Class 6: RequestForRoyalService

```
# For guests to request services like room cleaning, spa and food
class RequestForRoyalService:
    def __init__(self, service_type, description):
        self.__service_type = service_type
        self.__description = description

    # Generates formatted request
    def request(self):
        return f"🛎 Service Requested: {self.__service_type} – {self.__description}"
```

Class 7: RoyalReview

```
# Handles dates to ensure its similar to an actual real life scenario
from datetime import date

# Guest can leave a review after their stay
class RoyalReview:
    def __init__(self, rating, comments, date_of_review):
        self.__rating = rating
        self.__comments = comments
        self.__date = date_of_review

    # Format the review message
    def leave_review(self):
        return f"⭐ {self.__rating}/5 | '{self.__comments}' on {self.__date}"
```

Test Cases & Sample Outputs

This section provides a series of real-life-inspired scenarios to test and showcase the key features of the Royal Stay Hotel Management System. Each test simulates a practical use case and highlights how the classes work together to provide a seamless, modular and professional experience. From managing guest information to booking rooms, handling payments, loyalty rewards and service requests, these examples reflect how the object-oriented model fulfills the assignment's technical requirements.

Test Case 1: Royal Guest Books a Luxurious Stay

This test case demonstrates how a royal guest interacts with the hotel system to book a premium room. The guest, Prince Hamad of Al Jadi, books a Royal Suite featuring amenities like a Private Jacuzzi, Champagne and Butler on Call. The process involves creating a guest profile, selecting a room and confirming a reservation for a 4-night stay. Upon confirmation, the system generates an invoice, adds loyalty points to the guest's profile and records the reservation in their booking history. This test shows the core functionalities of the system including room booking, invoice calculation, guest history tracking and loyalty tier updates.

Code

```
# Creating a royal guest
guest_royal = RoyalGuest("Prince Hamad of Al Jadi", "Hamad@royal.ae", "+971-50-3838370")

# Creating a royal suite
suite_room = RoyalChamber(81, RoomType.SUITE,
["Private Jacuzzi", "Champagne", "Butler on Call"], 5143.81)
```

```
# Making a reservation
reservation_royal = RoyalReservation("R001", guest_royal, suite_room,
date(2025, 5, 14), date(2025, 5, 18))
reservation_royal.confirm()

# Displaying booking details
print(guest_royal)
print(reservation_royal)
print(reservation_royal.get_invoice().display())

# Showing reservation history
print("\n📜 Reservation History:")
for i in guest_royal.view_history():
    print(i)

# Showing loyalty points and tier
print("\n🏵️ Loyalty Program:")
print(guest_royal.get_loyalty())
```

Output

```
👑 Guest: Prince Hamad of Al Jadi | Hamad@royal.ae | Loyalty: Tier: Bronze, Points: 40
📅 Reservation R001 | Guest: Prince Hamad of Al Jadi | Room: 81
🧾 Invoice #INV-R001: Charges: AED 20575.24, Taxes: AED 2057.52, Total: AED 22632.760000000002

📜 Reservation History:
📅 Reservation R001 | Guest: Prince Hamad of Al Jadi | Room: 81

🎖️ Loyalty Program:
Tier: Bronze, Points: 40
```

The system successfully created the guest and suite room, confirmed the reservation and generated an invoice with taxes calculated at 10%. The output shows all expected details: the guest's full profile with loyalty status, the booking confirmation with reservation ID and room

number and the correctly calculated invoice total. The reservation history reflects the booking and the guest has earned 40 loyalty points, displayed under the Bronze tier which proves that all parts of the system are working in sync as designed. The clean and thematic output also illustrates user-friendliness and system clarity.

Test Case 2: Guest Requests Services and Leaves Feedback

This test case simulates a scenario where a guest, after making a luxurious reservation, requests a premium in-room service and then leaves a feedback review upon checkout. It highlights two essential features of the Royal Stay Hotel Management System: the RequestForRoyalService and RoyalReview classes. These modules allow for real-time service communication with staff and collection of guest feedback which helps with enhancing guest satisfaction and operational quality.

Code

```
# Royal Service request
service_request = RequestForRoyalService("Spa & Wellness", "Full body massage with oil in the Royal Spa Suite")
print("\n🛎️ Guest Service Request:")
print(service_request.request())

# Leaving a royal review
royal_review = RoyalReview(5, "Truly majestic! The suite, the food, the people, everything was perfect!", date(2025, 5, 18))
print("\n💬 Guest Feedback:")
print(royal_review.leave_review())
```

Output

```
🛎️ Guest Service Request:
🛎️ Service Requested: Spa & Wellness — Full body massage with oil in the Royal Spa Suite

💬 Guest Feedback:
⭐ 5/5 | 'Truly majestic! The suite, the food, the people, everything was perfect!' on 2025-05-18
```

The simulation begins with a request for a spa service in the Royal Suite. The service request is processed and printed with a styled message. After enjoying the stay, the guest submits a 5-star review describing the experience. The review is timestamped and clearly formatted. The output confirms both the request and the review were captured and displayed as expected, validating the behavior of both classes. This shows how the system handles post-booking guest interaction and it successfully reflects real-world hotel operations.

Test Case 3: Guest Makes Multiple Bookings and Redeems Rewards

This test case shows how the system handles a guest who stays at the hotel many times. The guest, Sheikha Marya, books five different rooms on different dates. The goal is to show how the system keeps track of these bookings, gives loyalty points for each night stayed, upgrades the guest's tier and finally lets the guest use those points to redeem a reward. It tests how well the loyalty and booking systems work together.

Code

```
# Booking chambers
Deluxe1 = RoyalChamber(252, RoomType.DOUBLE, ["Balcony View", "Mini Bar"], 1212)
Standard1 = RoyalChamber(331, RoomType.SINGLE, ["Smart TV", "Complimentary
Breakfast"], 777)
Deluxe2 = RoyalChamber(122, RoomType.DOUBLE, ["Balcony View", "Mini Bar"], 1921)
Standard2 = RoyalChamber(291, RoomType.SINGLE, ["Smart TV", "Complimentary
Breakfast"], 917)
```

```
# Creating a guest
Loyalguest = RoyalGuest("Sheikha Marya", "Marya@royalmail.com", "+971-56-1920178")

# First booking
reservation1 = RoyalReservation("R002", Loyalguest, Deluxe1, date(2025, 4, 5), date(2025, 4, 10))
reservation1.confirm()

# Second booking
reservation2 = RoyalReservation("R003", Loyalguest, Standard1, date(2025, 4, 12), date(2025, 4, 16))
reservation2.confirm()

# Third booking
reservation3 = RoyalReservation("R004", Loyalguest, suite_room, date(2025, 4, 18), date(2025, 4, 22))
reservation3.confirm()

# Fourth booking
reservation4 = RoyalReservation("R005", Loyalguest, Standard2, date(2025, 4, 23), date(2025, 4, 26))
reservation4.confirm()

# Fifth booking
reservation5 = RoyalReservation("R006", Loyalguest, Deluxe2, date(2025, 4, 27), date(2025, 4, 30))
reservation5.confirm()
```

```
# Printing guest info
print("\n👑 Guest Status After Multiple Bookings:")
print(Loyalguest)

# Showing reservation history
print("\n📜 All Reservations:")
for i in Loyalguest.view_history():
    print(i)

# Showing loyalty status
print("\n🎖️ Loyalty Program Status:")
print(Loyalguest.get_loyalty())

# Trying to redeem a reward
print("\n🎁 Redeeming Rewards:")
if Loyalguest.get_loyalty().redeem_rewards():
    print("✅ Reward redeemed! Enjoy a complimentary Royal Service!")
else:
    print("❌ Not enough points for reward redemption.")
```

Output

```
👑 Guest Status After Multiple Bookings:
👑 Guest: Sheikha Marya | Marya@royalmail.com | Loyalty: Tier: Silver, Points: 190

📜 All Reservations:
📅 Reservation R002 | Guest: Sheikha Marya | Room: 252
📅 Reservation R003 | Guest: Sheikha Marya | Room: 331
📅 Reservation R004 | Guest: Sheikha Marya | Room: 81
📅 Reservation R005 | Guest: Sheikha Marya | Room: 291
📅 Reservation R006 | Guest: Sheikha Marya | Room: 122

🎖️ Loyalty Program Status:
Tier: Silver, Points: 190

🎁 Redeeming Rewards:
✅ Reward redeemed! Enjoy a complimentary Royal Service!
```

After the five bookings, Sheikha Marya earns a total of 190 points and her loyalty tier changes from Bronze to Silver. The system correctly records every booking and shows them when asked. Then, when she tries to redeem her reward, the system checks that she has enough points and confirms the redemption with a clear message. This proves that the loyalty system works properly and rewards returning guests as designed.

Test Case 4: Guest Cancels a Booking

This test case is designed to show how the system handles cancellations. In this example, Sheikh Hassan makes a reservation for Room 414 at the Royal Hotel. After confirming the booking, he decides to cancel it. The goal is to make sure that once the reservation is canceled, the room becomes available again for future guests. It tests how well the cancellation function updates the system.

Code

```
# Guest
Guestcancels = RoyalGuest("Sheikh Hassan", "hassan@royalmail.ae", "+971-50-223344")

# Room
Roomcancels = RoyalChamber(414, RoomType.DOUBLE, ["Rain Shower", "Mini Fridge"],
1900)

# Booking
cancelreservation = RoyalReservation("R007", Guestcancels, Roomcancels, date(2025, 4, 20),
date(2025, 4, 23))
cancelreservation.confirm()
```

```
# Canceling the reservation
cancelreservation.cancel_reservation()

# Printing cancellation status
print("\n❌ Reservation Cancelled:")
print(cancelreservation)
print("Room available again?", Roomcancels.is_available())
```

Output

```
❌ Reservation Cancelled:
📅 Reservation R007 | Guest: Sheikh Hassan | Room: 414
Room available again? True
```

The reservation was successfully canceled. The output clearly shows the reservation information and confirms that the room is now available again (True). This means the system correctly reset the room's availability status after cancellation. This feature is very important in hotel systems to ensure real-time room availability is accurate.

Summary of Learnings

This assignment for the Royal Stay Hotel Management System was a comprehensive and practical opportunity to apply everything I have learned in the Programming Fundamentals course to a real-life scenario. The task was to design and build a complete hotel management system that performs real-world hotel operations, including room bookings, guest management, invoicing, loyalty programs, service requests and reviews. To do this, I applied concepts learned in the Programming Fundamentals course like encapsulation, abstraction, association, aggregation, composition, modularity and class relationships, all modeled using UML and implemented in

Python. I began with a clear UML class diagram to organize the system structure visually and then wrote the code in Python following proper OOP practices, by using private and protected attributes, getters and setters, constructors and well-structured methods. I ensured each class was meaningful, aligned with real-world hotel operations and added personal touches to make it unique and creative. This assignment allowed me to effectively meet all the course learning outcomes: I designed software that maps real-world entities using UML (LO1_OOAD), implemented well-structured and error-free object-oriented code (LO2_OOProgramming) and documented my work clearly and professionally (LO4_SWDocumentation). This assignment really pushed me to think in both critical and creative ways. It helped strengthen everything I learned in class by giving me the opportunity to apply those ideas to a real-world software scenario. I was able to experience what it is like to go through a full development process, starting from planning and analysis, then moving into implementation and testing. It allowed me to connect theory with practicality and meet the course goals. I now feel much more confident in my ability to design systems, handle complex problems using object-oriented programming and build clean, organized and flexible system for real-life needs.

GitHub Link