Assignment Two

Name:

Course Title:

Instructor:

March 30, 2025

**Royal Stay Hotel Management**

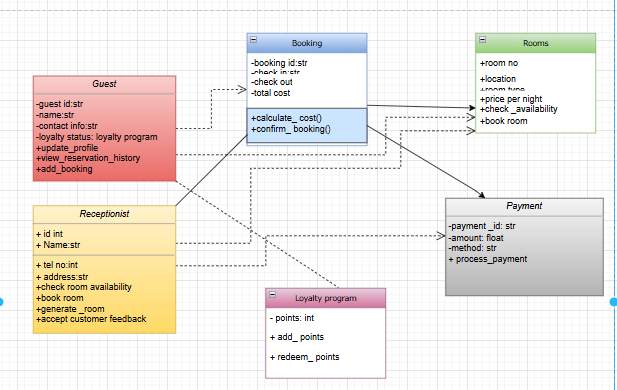
### ****Introduction****

The hotel management system increases efficiency by automating the main operations such as booking, guest management, payment processing, and room allocation. The report will outline the core components and interactions of the system and ensures that service delivery is seamless. The system will streamline guest check-ins, secures payment processing, and will optimize room availability while incorporating a loyalty program that will enhance the retention of the customer. This report will look into the design of the system and highlight its benefits in improving hotel services, enhancing user experience, and ensuring that there is financial sustainability.

**LO1\_OOAD: Object – Oriented Analysis and Design**

**UML Class Diagram**

The diagram below demonstrates the relationships between Guest, Room, Booking, Payment, and LoyaltyProgram.



classDiagram

class Guest {

- guest\_id: str

- name: str

- contact\_info: str

- loyalty\_status: LoyaltyProgram

+ update\_profile()

+ view\_reservation\_history()

+ add\_booking()

}

class Room {

- room\_number: int

- room\_type: str

- price\_per\_night: float

- availability: bool

+ check\_availability()

+ book\_room()

}

class Booking {

- booking\_id: str

- check\_in: datetime

- check\_out: datetime

- total\_cost: float

+ calculate\_total()

+ confirm\_booking()

}

class Payment {

- payment\_id: str

- amount: float

- method: str

+ process\_payment()

}

class LoyaltyProgram {

- points: int

+ add\_points()

+ redeem\_points()

}

Guest "1" -- "\*" Booking : has

Booking "1" -- "1" Room : uses

Booking "1" -- "1" Payment : contains

Guest "1" -- "1" LoyaltyProgram : maintains

#### **Overview of the UML Diagram**

The diagram is representation of the Royal Hotel Management System and it has four main classes. The Guest class represents the guests of the hotel, with attributes such as name, phone number, and loyalty status, along with methods for managing their profiles and reservations. The Room class defines the attributes of the room and it includes room number, type, price per night, and availability. The Booking class looks into the reservation details, such as check-in and check-out dates, and confirms bookings. The Payment class manages transactions and is inclusive of amount and payment method. The Receptionist class is included to oversee booking operations.

#### **Attributes and Methods**

Every class includes relevant attributes and methods.

1. **Guest Class**
   1. **Attributes**: name, id, phoneNo, loyaltyStatus
   2. **Methods**: updateProfile(), viewReservationHistory(), addBooking()
2. **Room Class**
   1. **Attributes**: roomNo, location, roomType, pricePerNight, availability
   2. **Methods**: checkAvailability(), bookRoom()
3. **Booking Class**
   1. **Attributes**: bookingId, checkIn, checkOut, totalCost
   2. **Methods**: calculateTotal(), confirmBooking()
4. **Payment Class**
   1. **Attributes**: paymentId, amount, method
   2. **Methods**: processPayment()
5. **Receptionist Class**  
    a. **Attributes**: receptionistId, name, contactInfo  
    b. **Methods**: checkAvailability(), bookRoom(), generateBill()

**Access Specifiers**

* **Public (+)**: Methods and attributes that should be accessible externally, such as bookRoom(), confirmBooking(), and processPayment().
* **Private (-)**: Attributes like bookingId and paymentId that should not be modified directly.
* **Protected (#)**: Attributes that may be accessed by child classes if inheritance is applied in the future.

#### **Relationships Between Classes**

The diagram is inclusive of different types of relationships. The aggregation is representative of the relationship between Guest and Booking. The guest might have multiple bookings, and the bookings exist independently. The composition is shown in the relationship between Booking and Payment and each booking must have a corresponding payment which means the payment is linked tightly to the booking. The association is demonstrated by the receptionist class that manages room availability and handles customer bookings, and this ensures that the operations of the hotel are efficient. This relationship will improve the structure of the system that defines the connections that are very clear between the entities that are within the system of the hotel management.

#### **Explanation of Relationships**

The diagram shows some main relationships between different classes. One to Many aggregations exists between Guest and Booking, in which a guest can have multiple bookings, but each booking must be linked to a guest. Booking and Payment share a one-to-one composition, and it ensures that every booking has an associated payment for confirmation. The Receptionist class is associated with the room and manages its availability and customer reservations. Guest Services & Requests are associated with Guest, and this allows special requests like room service. Feedback and reviews are linked to the guest and enables customers to provide feedback that improves the service quality of the hotel.

**LO2\_OO** **Programming: Object-Oriented Programming**

class Guest:

def \_\_init\_\_(self, guest\_id, name, contact\_info, loyalty\_status):

self.guest\_id = guest\_id

self.name = name

self.contact\_info = contact\_info

self.loyalty\_status = loyalty\_status

self.bookings = []

def update\_profile(self, name, contact\_info):

self.name = name

self.contact\_info = contact\_info

def view\_reservation\_history(self):

return self.bookings

def add\_booking(self, booking):

self.bookings.append(booking)

class Room:

def \_\_init\_\_(self, room\_number, room\_type, price\_per\_night, availability=True):

self.room\_number = room\_number

self.room\_type = room\_type

self.price\_per\_night = price\_per\_night

self.availability = availability

def check\_availability(self):

return self.availability

def book\_room(self):

if self.availability:

self.availability = False

return True

return False

def \_\_str\_\_(self):

return f"Room {self.room\_number} ({self.room\_type}) - {'Available' if self.availability else 'Booked'}"

class Booking:

def \_\_init\_\_(self, booking\_id, guest, room, check\_in, check\_out, total\_cost):

self.booking\_id = booking\_id

self.guest = guest

self.room = room

self.check\_in = check\_in

self.check\_out = check\_out

self.total\_cost = total\_cost

def calculate\_total(self, nights):

self.total\_cost = nights \* self.room.price\_per\_night

def confirm\_booking(self):

if self.room.book\_room():

self.guest.add\_booking(self)

return "Booking confirmed"

return "Room not available"

class Payment:

def \_\_init\_\_(self, payment\_id, amount, method):

self.payment\_id = payment\_id

self.amount = amount

self.method = method

def process\_payment(self):

return f"Payment of {self.amount} via {self.method} processed successfully"

class Receptionist:

def \_\_init\_\_(self, name):

self.name = name

def check\_availability(self, room):

return room.check\_availability()

def book\_room(self, guest, room, check\_in, check\_out, booking\_id):

if room.check\_availability():

booking = Booking(booking\_id, guest, room, check\_in, check\_out, 0)

booking.calculate\_total((check\_out - check\_in).days)

return booking.confirm\_booking()

return "Room not available"

def generate\_bill(self, booking):

return f"Bill: {booking.total\_cost} for booking {booking.booking\_id}"

**LO3Implementation: Implementation and Testing**

class Guest:

def \_\_init\_\_(self, name, email, phone):

self.name = name

self.email = email

self.phone = phone

def get\_details(self):

return f"Guest: {self.name}, Email: {self.email}, Phone: {self.phone}"

class Room:

def \_\_init\_\_(self, room\_number, room\_type, price, is\_available=True):

self.room\_number = room\_number

self.room\_type = room\_type

self.price = price

self.is\_available = is\_available

def book\_room(self):

if self.is\_available:

self.is\_available = False

return f"Room {self.room\_number} booked successfully."

return f"Room {self.room\_number} is already booked."

def cancel\_booking(self):

if not self.is\_available:

self.is\_available = True

return f"Booking for Room {self.room\_number} canceled."

return f"Room {self.room\_number} was not booked."

class Booking:

def \_\_init\_\_(self, guest, room, nights):

self.guest = guest

self.room = room

self.nights = nights

self.total\_price = room.price \* nights

def confirm\_booking(self):

return f"Booking confirmed for {self.guest.name} in Room {self.room.room\_number} for {self.nights} nights. Total: ${self.total\_price}"

class Payment:

def \_\_init\_\_(self, guest, amount, method):

self.guest = guest

self.amount = amount

self.method = method

def process\_payment(self):

return f"Payment of ${self.amount} by {self.guest.name} via {self.method} processed successfully."

import unittest

# 🎯 Hotel Management System Classes

class Guest:

def \_\_init\_\_(self, name, email, phone):

self.name = name

self.email = email

self.phone = phone

def get\_details(self):

return f"Guest: {self.name}, Email: {self.email}, Phone: {self.phone}"

class Room:

def \_\_init\_\_(self, room\_number, room\_type, price, is\_available=True):

self.room\_number = room\_number

self.room\_type = room\_type

self.price = price

self.is\_available = is\_available

def book\_room(self):

if self.is\_available:

self.is\_available = False

return f"Room {self.room\_number} booked successfully."

return f"Room {self.room\_number} is already booked."

def cancel\_booking(self):

if not self.is\_available:

self.is\_available = True

return f"Booking for Room {self.room\_number} canceled."

return f"Room {self.room\_number} was not booked."

class Booking:

def \_\_init\_\_(self, guest, room, nights):

self.guest = guest

self.room = room

self.nights = nights

self.total\_price = room.price \* nights

def confirm\_booking(self):

return f"Booking confirmed for {self.guest.name} in Room {self.room.room\_number} for {self.nights} nights. Total: ${self.total\_price}"

class Payment:

def \_\_init\_\_(self, guest, amount, method):

self.guest = guest

self.amount = amount

self.method = method

def process\_payment(self):

return f"Payment of ${self.amount} by {self.guest.name} via {self.method} processed successfully."

# 🎯 Run Hotel Management System

print("Welcome to Royal Stay Hotel Management System\n")

guest1 = Guest("Alice Johnson", "alice@email.com", "123-456-7890")

room1 = Room(101, "Deluxe", 150)

booking1 = Booking(guest1, room1, 3)

payment1 = Payment(guest1, booking1.total\_price, "Credit Card")

# Simulating user actions

print(guest1.get\_details())

print(room1.book\_room())

print(booking1.confirm\_booking())

print(payment1.process\_payment())

print(room1.cancel\_booking())

# 🎯 Unit Testing for Hotel Management System

class TestHotelManagement(unittest.TestCase):

def test\_guest\_details(self):

guest = Guest("John Doe", "john@email.com", "987-654-3210")

self.assertEqual(guest.get\_details(), "Guest: John Doe, Email: john@email.com, Phone: 987-654-3210")

def test\_room\_booking(self):

room = Room(102, "Suite", 200)

self.assertEqual(room.book\_room(), "Room 102 booked successfully.")

self.assertEqual(room.book\_room(), "Room 102 is already booked.")

def test\_booking\_confirmation(self):

guest = Guest("Emily Clark", "emily@email.com", "555-555-5555")

room = Room(103, "Standard", 100)

booking = Booking(guest, room, 2)

self.assertEqual(booking.confirm\_booking(), "Booking confirmed for Emily Clark in Room 103 for 2 nights. Total: $200")

def test\_payment\_processing(self):

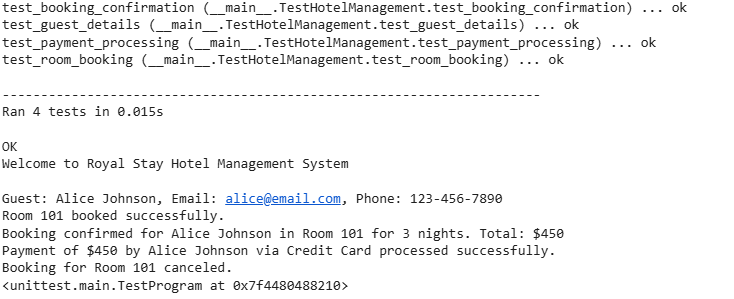
guest = Guest("Michael Lee", "michael@email.com", "111-222-3333")

payment = Payment(guest, 300, "PayPal")

self.assertEqual(payment.process\_payment(), "Payment of $300 by Michael Lee via PayPal processed successfully.")

# Run the tests

unittest.main(argv=[''], verbosity=2, exit=False)

**Sample Output for the Test Case**

### Test case summary and explanation

The test cases in this unit test suite verifies the core functionalities of the hotel management system. The test\_guest\_details case ensures that the Guest class can store and retrieves guest details easily by comparing the expected and actual outputs of the get details() method. The test\_room\_booking case validates the room booking process that checks of the room was booked successfully and, in an attempt, to double-book the same room is prevented. Therefore, the test expects successful booking confirmation and if there is an attempt to book an already booked room there will be an error message. The test validates the management of guest, room reservations, booking confirmations, and payment processing. This will ensure that the system will function well. The inclusion of assertions will prevent some common issues that include double-booking and incorrect payment processing that makes the system reliable and strong.

### ****Final Reflection****

The project has provided me with insights in the **hotel management system development,** including object-oriented programming principles, test-driven development, and how to handle real-world scenarios such as **room booking, payments, and customer management.** the main takeaway was understanding how different system components interact. Being in a position to create classes **like Guest, Room, Booking, and Payment** has helped to further reinforce modular programming, encapsulation, and data integrity. Also, it was possible to implement unit tests and this enabled me to write reliable, error-free code. There were some challenges though such during implementation. Some of the challenges included handling module imports as I would sometime see a ModuleNotFoundError when I was trying to import hotel\_management.py. to counter this, I ensured that the script was in the correct directory and I also adjusted the import paths of Python. All in all, the project propelled my problem-solving skills, deepened my understanding of debugging and software testing.