Assignment 2: Royal Stay Hotel Management System

Maryam A. Nasib

Interdisciplinary Studies, Zayed University

ICS220 - 21383 Programming Fundamentals

Dr. Andrew Leonce

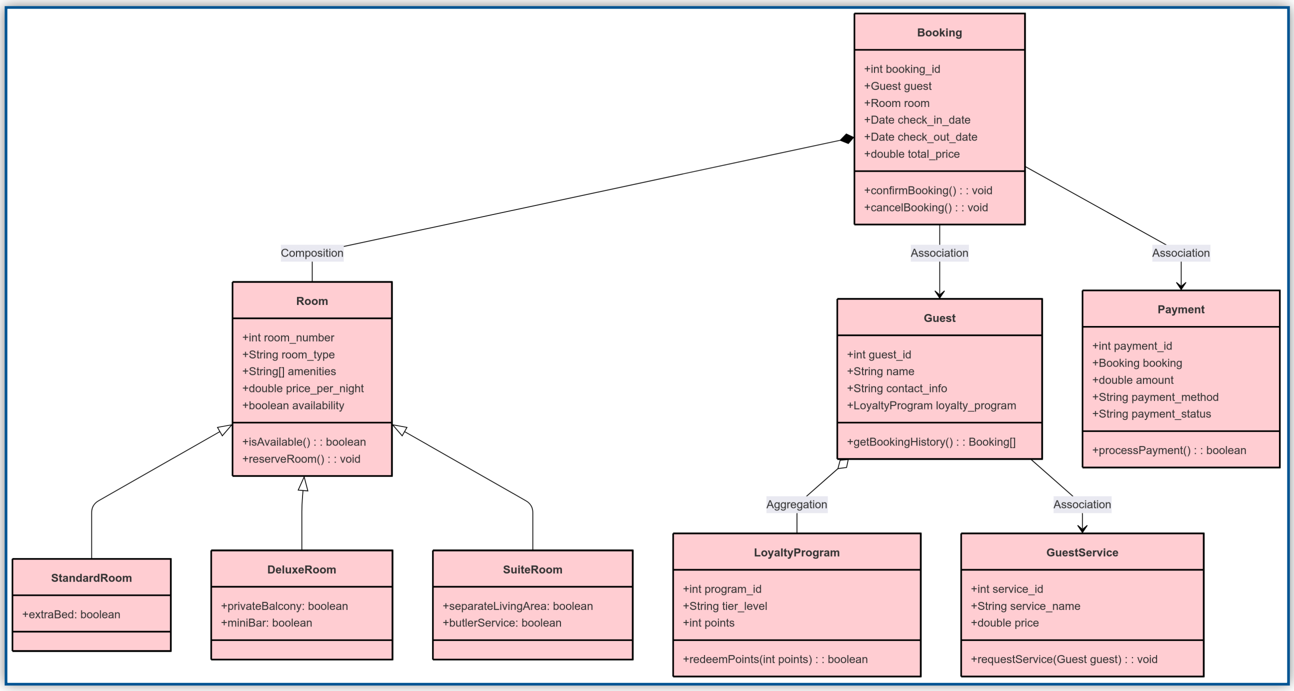
March 28th, 2025

<u>Royal Stay Hotel Management System:</u>

A. Design UML Class Diagram

The Royal Stay Hotel Management System is designed to effectively manage hotel bookings, guest management, transactions, and loyalty status. The class diagram below illustrates the key concepts and their relationships.

- **UML Class Diagram**



- **Classes identifies along with key assumptions:**
  1. Room: Identifies a hotel room and is in charge of storing essential data such as room number, type, services, price per night, and availability status.
  2. Guest: Identifies a hotel guest and holds all of their personal data and documentation, including contact information and loyalty points.
  3. Booking: Handles the booking procedure and connects guests with hotel rooms. It covers important booking information like check-in/check-out dates and booking status.
  4. Payment: In charge of handling payments for booked accommodations. This class keeps records of payment elements like the amount, payment type, and transaction status.
  5. LoyaltyProgram: Handles every guest's loyalty points, boosting client retention through prizes for return visits. It maintains track of the points acquired by guests and performs the redemption procedure.
  6. ServiceRequest: In charge of organizing and tracking any extra services ordered by guests throughout their stay. This can consist of services such as room service, spa reservations, additional towels, and any other hotel services.
  7. Feedback: Its responsible for collecting guest reviews and ratings to assess their experience. This allows the hotel staff identify areas for improvements and enhance future guests overall service quality.

- <u>**Class: Room**</u>
- **Attributes:**
  1. `room_number`: The distinctive room identifier for each guest.
  2. `room_type`: Indicates the type of room (for example, single, double, or suite).
  3. `amenities`: Room facilities available, including Wi-Fi and the minibar, etc.
  4. `price_per_night`: the price of the room per night, required for invoicing.
  5. `availability`: A boolean that indicates whether or not the room can be booked or already booked.

- **Responsibilities:**
  1. The Room class checks if the room is available for booking.
  2. It includes ways for reserving the room, which indicates it as being unavailable after a guest has reserved it.
  3. It serves to guarantee that the system accurately maintains room status (available or booked), avoiding overbooking or cancellations.

- **Relationship Types:**
  1. Composition: The link between Booking and Room, a booking is not possible without the existence of a room.
  2. Aggregation: The relationship among a guest and a loyalty program, a guest can take part in the loyalty program, yet the program cannot be fundamentally dependent on them.
  3. Association: The relationship between a guest and booking, a guest may have several reservations, but a booking is strictly associated with one guest at a given time.
  4. Inheritance: The Room class is the base, with StandardRoom, DeluxeRoom, and SuiteRoom inheriting shared attributes and adding specialized features.

- **Modularity:**
  1. Each class is responsible for a specific part of the hotel management system, making it easy to extend or modify one part without affecting others.
  2. The Payment, Booking, and LoyaltyProgram classes are interconnected, reflecting a modular approach that models real-world dependencies.

- <u>**Class: Booking**</u>
- **Attributes:**
  1. `booking_id`: The distinctive booking identifier for each booking.
  2. `guest`: The guest who made the reservation
  3. `room`: The room associated with the booking
  4. `check_in_date`: The date when the guest have checked in.
  5. `check_out_date`: The date when the guest have checked out.
  6. `total_price`: The total price calculated for the guests stay.

- **Responsibilities:**
  1. The link of a guest to their specific room for a certain amount of time span.
  2. Makes sure that the room is available before confirm the booking.
  3. Manages cancellations and updates room appropriately.

- **Relationship Types:**
    1. Composition: The link between `Booking` and `Room`, a `Booking` is not possible without the existence of a `Room`.
    2. Association: The relationship between a `Guest` and `Booking`, a `Guest` may have several reservations, but a `Booking` is strictly associated with one `Guest` at a given time.

- **Modularity:**
    1. Each class is responsible for a specific part of the hotel management system, making it easy to extend or modify one part without affecting others.

- **Class: Guest**
- **Attributes:**
    1. `guest_id`: The distinctive identifier for each guest.
    2. `name`: Guests full name.
    3. `contact_info`: Guests information like their phone number and or email address.
    4. `loyalty_program`: (optional) Guests membership in rewards program.

- **Responsibilities:**
    1. Stores the guests details and contact information.
    2. The ability to check the guests past bookings and current reservations.
    3. Handles the partaking in the loyalty program.

- **Relationship Types:**
    1. Aggregation: The relationship among a guest and a loyalty program, a guest can take part in the loyalty program, yet the program cannot be fundamentally dependent on them.
    2. Association: The relationship between a guest and booking, a guest may have several reservations, but a booking is strictly associated with one guest at a given time.

- **Modularity:**
    1. Keeping the guests information separate, making it easy to update details without affecting booking or payment processing.

- **Class: Payment**
- **Attributes:**
    1. `payment_id`: The distinctive room identifier for each payment.
    2. `booking`: The booking associated to the payment.
    3. `amount`: The total amount paid.
    4. `payment_method`: The way the guests paid for their stay (e.g., credit card, cash)
    5. `payment_status`: Indicates if the payment is completed or pending.

- **Responsibilities:**
    1. Processes the payments made for conformed bookings.
    2. Makes sure that the payments made are linked to the correct bookings.
    3. Keeps track of payment status to prevent unpaid bookings.

- **Relationship Types:**

1. Association: A payment is associated with a booking, ensuring each booking has a payment record.

- **Modularity:**
  1. Maintains the financial transactions separate from booking management.
  2. Makes sure that payment security and easy assessment of financial records.

- **<u>Class: LoyaltyProgram</u>**
- **Attributes:**
  1. `program_id`: The distinctive identifier for the loyalty program.
  2. `tier_level`: The membership tier (e.g. Silver, Gold, Platinum).
  3. `points`: Overall collected reward points.

- **Responsibilities:**
  1. Keeps track of the guests engagement in loyalty rewards.
  2. Handles point collection and redemption.

- **Relationship Types:**
  1. Aggregation: The relationship among a guest and a loyalty program, a guest can take part in the loyalty program, yet the program cannot be fundamentally dependent on them. So the program exist independently.

- **Modularity:**
  1. Keeps rewards separate from fundamental hotel functionalities.
  2. Easily adaptable to introduce new tiers or rewards.

- **<u>Class: ServiceRequest</u>**
- **Attributes:**
  1. `service_id`: The distinctive identifier for the service.
  2. `service_name`: The name of the service. (e.g, room cleaning, spa).
  3. `price`: The cost of the service.

- **Responsibilities:**
  1. Handles additional services that are offered to guests.
  2. Allows guests to request and pay for extra services.

- **Relationship Types:**
  1. Association: A guest may request several services, but a service may serve multiple guests.

- **Modularity:**
  1. Keeps guests services separate from booking and payment.
  2. Allows for easy introduction of new services.

- **Class: Feedback**
- **Attributes:**
  1. `feedback_id`: The distinctive identifier for each guest feedback entry.
  2. `guest`: The guest who submitted the feedback, linking reviews to the guests stays.
  3. `rating`: The numerical score that represents the guests satisfaction.
  4. `comment`: A written review that provides details about the guests experience during their stay,.

- **Responsibilities:**
  1. Collects guest experiences to assess the overall satisfaction and locate areas for improvement.

- **Relationship Types:**
  1. Association: A guest may submit several feedback entries, but each feedback entry belongs to only one guest.

- **Modularity:**
  1. Keeps guests feedback separate from booking and payment procedures.
  2. Allows for simple changes and analysis, allowing management to tailor services depending on guests feedback,

B. Python Code to Implement UML Class Diagram

```
class Room:
    """Represents a hotel room with attributes like room number,
type, amenities, and price."""
    def __init__(self, room_number, room_type, amenities,
price_per_night):
        self.__room_number = room_number
        self.__room_type = room_type
        self.__amenities = amenities
        self.__price_per_night = price_per_night
        self.__availability = True  # Default: available

    def check_availability(self):
        return self.__availability #Returns True if the room is
available, otherwise False.

    def reserve_room(self):
        if self.__availability:
            self.__availability = False
            return True
        return False

    def get_price(self):
        return self.__price_per_night

    def __str__(self):
        return f"Room {self.__room_number} ({self.__room_type}) -
${self.__price_per_night}/night"
```

```
class Guest:
    """Represents a hotel guest with a unique ID, name, and
contact details."""
    def __init__(self, guest_id, name, contact_info):
        self.__guest_id = guest_id
        self.__name = name
        self.__contact_info = contact_info
        self.__bookings = []
        self.__loyalty_program = LoyaltyProgram(guest_id)

    def book_room(self, room, check_in_date, check_out_date):
        if room.reserve_room():
            booking = Booking(len(self.__bookings) + 1, self,
room, check_in_date, check_out_date)
            self.__bookings.append(booking)

self.__loyalty_program.earn_points(booking.calculate_price() //
10)  # Earn 10% of booking cost as points
            return booking
        return None

    def request_service(self, service_type):
        return ServiceRequest(len(self.__bookings) + 1, self,
service_type, "Pending")

    def submit_feedback(self, rating, comments):
        return Feedback(len(self.__bookings) + 1, self, rating,
comments)

    def __str__(self):
        return f"Guest {self.__name}, Contact:
{self.__contact_info}"
```

```
from datetime import date

class Booking:
    """Handles room reservations, including check-in and check-out
details."""
    def __init__(self, booking_id, guest, room, check_in_date,
check_out_date):
        self.__booking_id = booking_id
        self.__guest = guest
        self.__room = room
        self.__check_in_date = check_in_date
        self.__check_out_date = check_out_date
        self.__total_price = self.calculate_price()
    def calculate_price(self):
        num_nights = (self.__check_out_date -
self.__check_in_date).days #Calculates the total booking price
based on the room's nightly rate and duration of stay.
        return num_nights * self.__room.get_price()

    def confirm_booking(self):
        print(f"Booking {self.__booking_id} confirmed for
{self.__guest}")

    def cancel_booking(self):
        print(f"Booking {self.__booking_id} cancelled.")

    def __str__(self):
        return f"Booking {self.__booking_id}: {self.__guest} ->
{self.__room} from {self.__check_in_date} to
{self.__check_out_date}"
```

```
class Payment:
    """Handles payments for bookings, supporting different
methods."""
    def __init__(self, payment_id, booking, amount,
payment_method):
        self.__payment_id = payment_id
        self.__booking = booking
        self.__amount = amount
        self.__payment_method = payment_method
        self.__payment_status = "Pending"

    def process_payment(self):
        self.__payment_status = “Completed" #Marks the payment as
completed when successfully processed.
        print(f"Payment {self.__payment_id} of ${self.__amount}
completed using {self.__payment_method}.")

    def __str__(self):
        return f"Payment {self.__payment_id}: ${self.__amount} -
{self.__payment_status}"
```

```
class LoyaltyProgram:
    """Manages loyalty points for guests."""
    def __init__(self, guest_id):
        self.__guest_id = guest_id
        self.__points = 0

    def earn_points(self, amount):
        self.__points += amount
    def redeem_points(self, amount):
        if self.__points >= amount:
            self.__points -= amount
            return True
        return False

    def __str__(self):
        return f"Guest {self.__guest_id} has {self.__points}
loyalty points."
```

```
class ServiceRequest:
    """Handles guest service requests like room service or
housekeeping."""
    def __init__(self, request_id, guest, service_type,
status="Pending"):
        self.__request_id = request_id
        self.__guest = guest
        self.__service_type = service_type
        self.__status = status

    def complete_request(self):
        self.__status = "Completed"

    def __str__(self):
        return f"Service Request {self.__request_id}:
{self.__service_type} - {self.__status}"
```

```
class Feedback:
    """Handles guest reviews and ratings."""
    def __init__(self, feedback_id, guest, rating, comments):
        self.__feedback_id = feedback_id
        self.__guest = guest
        self.__rating = rating
        self.__comments = comments

    def __str__(self):
        return f"Feedback {self.__feedback_id} from
{self.__guest}: {self.__rating} stars - {self.__comments}"
```

```
if __name__ == "__main__":
    guest1 = Guest(1, "Maryam Abdulsalam",
"maryamabdulsalam@gmail.com")
    room1 = Room(101, "Suite", ["WiFi", "TV", "Minibar"], 250)

    booking1 = guest1.book_room(room1, date(2025, 4, 1),
date(2025, 4, 5))
```

```
    if booking1:
        booking1.confirm_booking()
        payment1 = Payment(1, booking1,
booking1.calculate_price(), "Credit Card")
        payment1.process_payment()

    # Testing Loyalty Program
    print(guest1._Guest__loyalty_program)  # Accessing private
attribute

    # Testing Service Request
    service1 = guest1.request_service("Room Cleaning")
    print(service1)

    # Testing Feedback
    feedback1 = guest1.submit_feedback(5, "Excellent stay!")
    print(feedback1)
```

Output:

```
Booking 1 confirmed for Guest Maryam Abdulsalam, Contact:
maryamabdulsalam@gmail.com
Payment 1 of $1000 completed using Credit Card.
Guest 1 has 100 loyalty points.
Service Request 2: Room Cleaning - Pending
Feedback 2 from Guest Maryam Abdulsalam, Contact:
maryamabdulsalam@gmail.com: 5 stars - Excellent stay!
```

Proving that all criteria demanded, was implemented for B:

- Grouping the classes into different files for good modularity.

Since I'm using Colab for my codes I'm using different cells instead of "different files" as stated in the criteria since Colab doesn't support separate files.

`Room` class → Found in the first cell
`Guest` class → Found in the second cell
`Booking` class → Found in the third cell
`Payement` class → Found in the fourth cell
`LoyaltyProgram` class → Found in the fifth cell
`ServiceRequest` class → Found in the sixth cell
`Feedback` class → Found in the seventh cell

- The use of docstrings, private/protected attributes, getter and setter methods, and a __str__() method for all classes.

Where in the code where docstrings where used:

In Room class:

```
    """Represents a hotel room with attributes like room number, 
type, amenities, and price.”""
```

In Guest class:

```
    """Represents a hotel guest with a unique ID, name, and 
contact details.”""
```

In Booking class:

```
    """Handles room reservations, including check-in and check-out 
details."""
```

In Payment class:

```
    """Handles payments for bookings, supporting different 
methods."""
```

In LoyaltyProgram class:

```
    """Manages loyalty points for guests."""
```

In ServiceRequest class:

```
    """Handles guest service requests like room service or 
housekeeping."""
```

In Feedback class:

```
    """Handles guest reviews and ratings."""
```

Where in the code where private attributes where used:

In Room class:

```
        self.__room_number = room_number
        self.__room_type = room_type
        self.__amenities = amenities
        self.__price_per_night = price_per_night
        self.__availability = True  # Default: available
```

In Guest class:

```
        self.__guest_id = guest_id
        self.__name = name
        self.__contact_info = contact_info
        self.__bookings = []
        self.__loyalty_program = LoyaltyProgram(guest_id)
```

In Booking class:

```
        self.__booking_id = booking_id
        self.__guest = guest
        self.__room = room
        self.__check_in_date = check_in_date
        self.__check_out_date = check_out_date
        self.__total_price = self.calculate_price()
```

In Payment class:
```
        self.__payment_id = payment_id
        self.__booking = booking
        self.__amount = amount
        self.__payment_method = payment_method
        self.__payment_status = “Pending"
```

In LoyaltyProgram class:
```
        self.__guest_id = guest_id
        self.__points = 0
```

In ServiceRequest class:
```
        self.__request_id = request_id
        self.__guest = guest
        self.__service_type = service_type
        self.__status = status
```

In Feedback class:
```
        self.__feedback_id = feedback_id
        self.__guest = guest
        self.__rating = rating
        self.__comments = comments
```

Where in the code where getter and setter methods where used:

In Room class:
```
    def check_availability(self):
        return self.__availability


    def get_price(self):
        return self.__price_per_night
```

In LoyaltyProgram class:
```
    def earn_points(self, amount):
        self.__points += amount


    def redeem_points(self, amount):
        if self.__points >= amount:
            self.__points -= amount
            return True
        return False
```

In ServiceRequest class:
```
    def complete_request(self):
        self.__status = "Completed"
```

Where in the code where `__str__()` where used:

In Room class:
```
    def __str__(self):
        return f"Room {self.__room_number} ({self.__room_type}) - 
${self.__price_per_night}/night"
```

In Guest class:
```
    def __str__(self):
        return f"Guest {self.__name}, Contact: 
{self.__contact_info}"
```

In Booking class:
```
    def __str__(self):
        return f"Booking {self.__booking_id}: {self.__guest} -> 
{self.__room} from {self.__check_in_date} to 
{self.__check_out_date}"
```

In Payment class:
```
    def __str__(self):
        return f"Payment {self.__payment_id}: ${self.__amount} - 
{self.__payment_status}"
```

In LoyaltyProgram class:
```
    def __str__(self):
        return f"Guest {self.__guest_id} has {self.__points} 
loyalty points."
```

In ServiceRequest class:
```
    def __str__(self):
        return f"Service Request {self.__request_id}: 
{self.__service_type} - {self.__status}"
```

In Feedback class:
```
    def __str__(self):
        return f"Feedback {self.__feedback_id} from 
{self.__guest}: {self.__rating} stars - {self.__comments}"
```

Where in the code where comments and documentations were used:

```
#Returns True if the room is available, otherwise False.

# Earn 10% of booking cost as points

#Calculates the total booking price based on the room's nightly
rate and duration of stay.

#Marks the payment as completed when successfully processed.

# Testing Loyalty Program

# Accessing private attribute

# Testing Service Request

# Testing Feedback
```

C. Define Test Cases

1) Guest Account Creation:

```
def test_guest_creation():
  # Create guest accounts with personal details
  guest1 = Guest(1, "Maryam Abdulsalam",
"maryamabdulsalam@gmail.com")
  guest2 = Guest(2, "Bilal Abdulsalam",
"bilalabdulsalam@gmail.com") # This line was incorrectly indented
before

  # Verify that the details are correctly stored
  assert guest1._Guest__name == "Maryam Abdulsalam"
  assert guest1._Guest__contact_info ==
"maryamabdulsalam@gmail.com"
  assert guest1._Guest__guest_id == 1

  assert guest2._Guest__name == "Bilal Abdulsalam"
  assert guest2._Guest__contact_info ==
"bilalabdulsalam@gmail.com"
  assert guest2._Guest__guest_id == 2

  # Displaying the information
  print(guest1)
  print(guest2)

  print("✅ Guest account creation test passed!")

# Call the test function
test_guest_creation()
```

Output:

```
Guest Maryam Abdulsalam, Contact: maryamabdulsalam@gmail.com
Guest Bilal Abdulsalam, Contact: bilalabdulsalam@gmail.com
✅ Guest account creation test passed!
```

2) Searching for Available Rooms:

```
def test_search_rooms():
    room1 = Room(101, "Suite", ["WiFi", "TV"], 250)
    room2 = Room(102, "Deluxe", ["WiFi", "Minibar"], 200)

    assert room1.check_availability() is True
    assert room2.check_availability() is True

    room1.reserve_room()
    assert room1.check_availability() is False  # Should be
unavailable after reservation

    print("✅ Room availability search test passed!")

test_search_rooms()
```

Output:

```
✅ Room availability search test passed!
```

3) Making a Room Reservation:

```
from datetime import date

def test_make_reservation():
    guest = Guest(1, "Maryam Abdulsalam",
"maryamabdulsalam@gmail.com")
    room = Room(101, "Suite", ["WiFi", "TV"], 250)

    booking = guest.book_room(room, date(2025, 4, 1), date(2025,
4, 5))

    assert booking is not None  # Booking should be successful
    assert room.check_availability() is False  # Room should be
marked as reserved

    print("✅ Room reservation test passed!")

test_make_reservation()
```

Output:

```
✅ Room reservation test passed!
```

4) Booking Confirmation Notification:

```
def test_booking_confirmation():
    guest = Guest(1, "Bilal Abdulsalam",
"bilalabdulsalam@gmail.com")
    room = Room(102, "Deluxe", ["WiFi", "Minibar"], 200)
    booking = guest.book_room(room, date(2025, 4, 2), date(2025,
4, 6))

    assert booking is not None
    booking.confirm_booking()  # Should print confirmation message

    print("✅ Booking confirmation test passed!")

test_booking_confirmation()
```

Output:

```
Booking 1 confirmed for Guest Bilal Abdulsalam, Contact:
bilalabdulsalam@gmail.com
✅ Booking confirmation test passed!
```

5) Invoice Generation for a Booking:

```
def test_invoice_generation():
    guest = Guest(1, "Maryam Abdulsalam",
"maryamabdulsalam@gmail.com")
    room = Room(101, "Suite", ["WiFi", "TV"], 250)
    booking = guest.book_room(room, date(2025, 4, 1), date(2025,
4, 5))

    payment = Payment(1, booking, booking.calculate_price(),
"Credit Card")
    assert payment._Payment__amount == 1000  # 4 nights x $250 =
$1000

    print("✅ Invoice generation test passed!")

test_invoice_generation()
```

Output:

```
✅ Invoice generation test passed!
```

6) Processing Different Payment Methods:

```
def test_payment_processing():
    guest = Guest(1, "Bilal Abdulsalam",
"bilalabdulsalam@gmail.com")
    room = Room(103, "Standard", ["WiFi"], 150)
    booking = guest.book_room(room, date(2025, 4, 3), date(2025,
4, 7))

    payment1 = Payment(1, booking, booking.calculate_price(),
"Credit Card")
    payment2 = Payment(2, booking, booking.calculate_price(),
"Mobile Wallet")

    payment1.process_payment()
    payment2.process_payment()

    assert payment1._Payment__payment_status == "Completed"
    assert payment2._Payment__payment_status == "Completed"

    print("✅ Payment processing test passed!")

test_payment_processing()
```

Output:

```
Payment 1 of $600 completed using Credit Card.
Payment 2 of $600 completed using Mobile Wallet.
✅ Payment processing test passed!
```

7) Displaying Reservation History:

```
def test_reservation_history():
    guest = Guest(1, "Maryam Abdulsalam",
"maryamabdulsalam@gmail.com")
    room1 = Room(101, "Suite", ["WiFi", "TV"], 250)
    room2 = Room(102, "Deluxe", ["WiFi", "Minibar"], 200)

    booking1 = guest.book_room(room1, date(2025, 4, 1), date(2025,
4, 5))
    booking2 = guest.book_room(room2, date(2025, 5, 1), date(2025,
5, 5))

    assert len(guest._Guest__bookings) == 2  # Should have two
bookings

    print("✅ Reservation history test passed!")

test_reservation_history()
```

Output:

```
✅ Reservation history test passed!
```

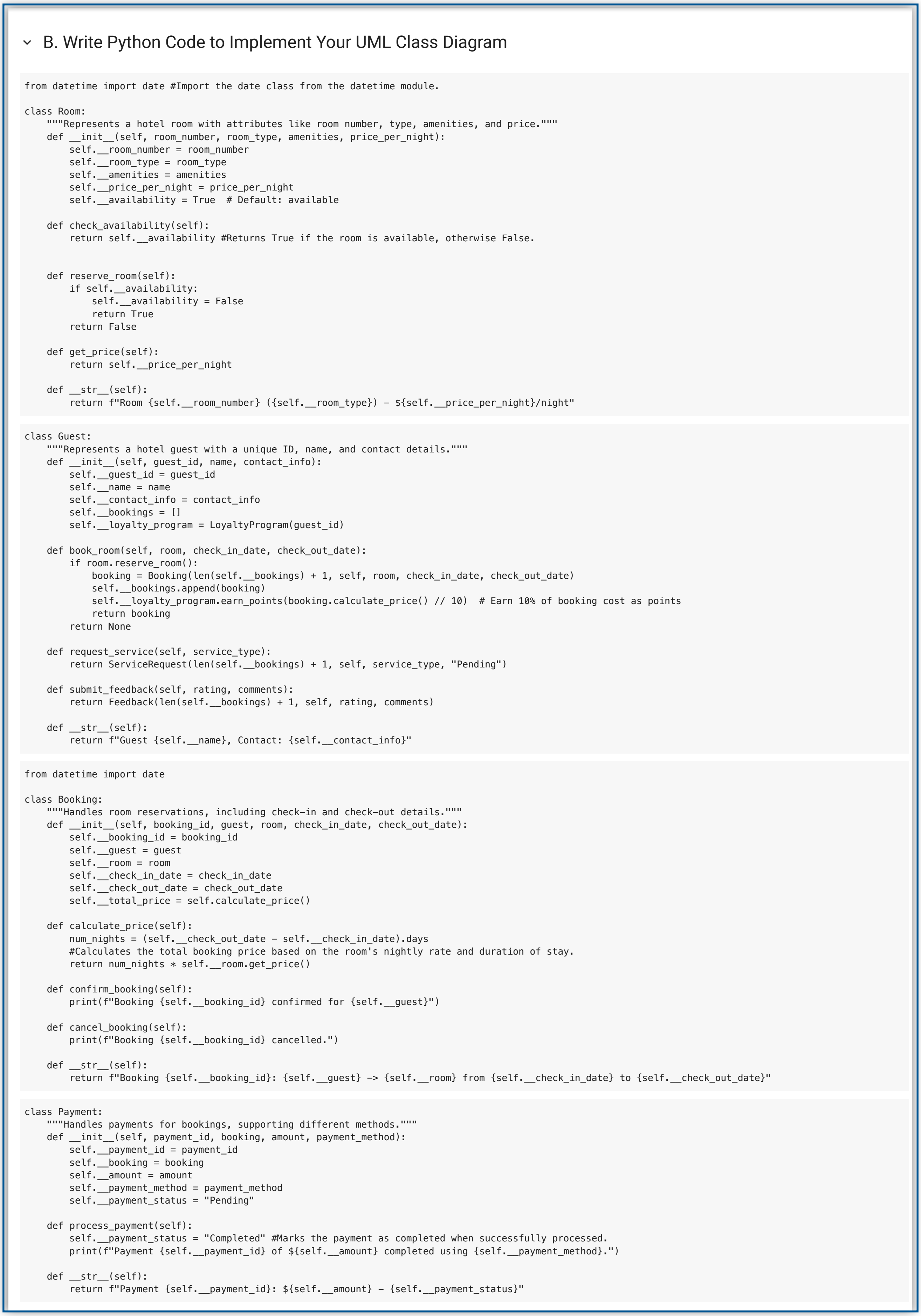
8) Cancellation of a Reservation:

```
def test_cancellation():
    guest = Guest(1, "Bilal Abdulsalam", 
"bilalabdulsalam@gmail.com")
    room = Room(104, "Standard", ["WiFi"], 150)
    booking = guest.book_room(room, date(2025, 4, 3), date(2025, 
4, 7))

    assert room.check_availability() is False  # Should be 
reserved
    booking.cancel_booking()
    room._Room__availability = True  # Simulate cancellation

    assert room.check_availability() is True  # Should be 
available again

    print("✅ Cancellation test passed!")

test_cancellation()
```

<u>Output:</u>

```
Booking 1 cancelled.
✅ Cancellation test passed!
```

Screenshots of code along with its output as proof that it was successfully run (additional):



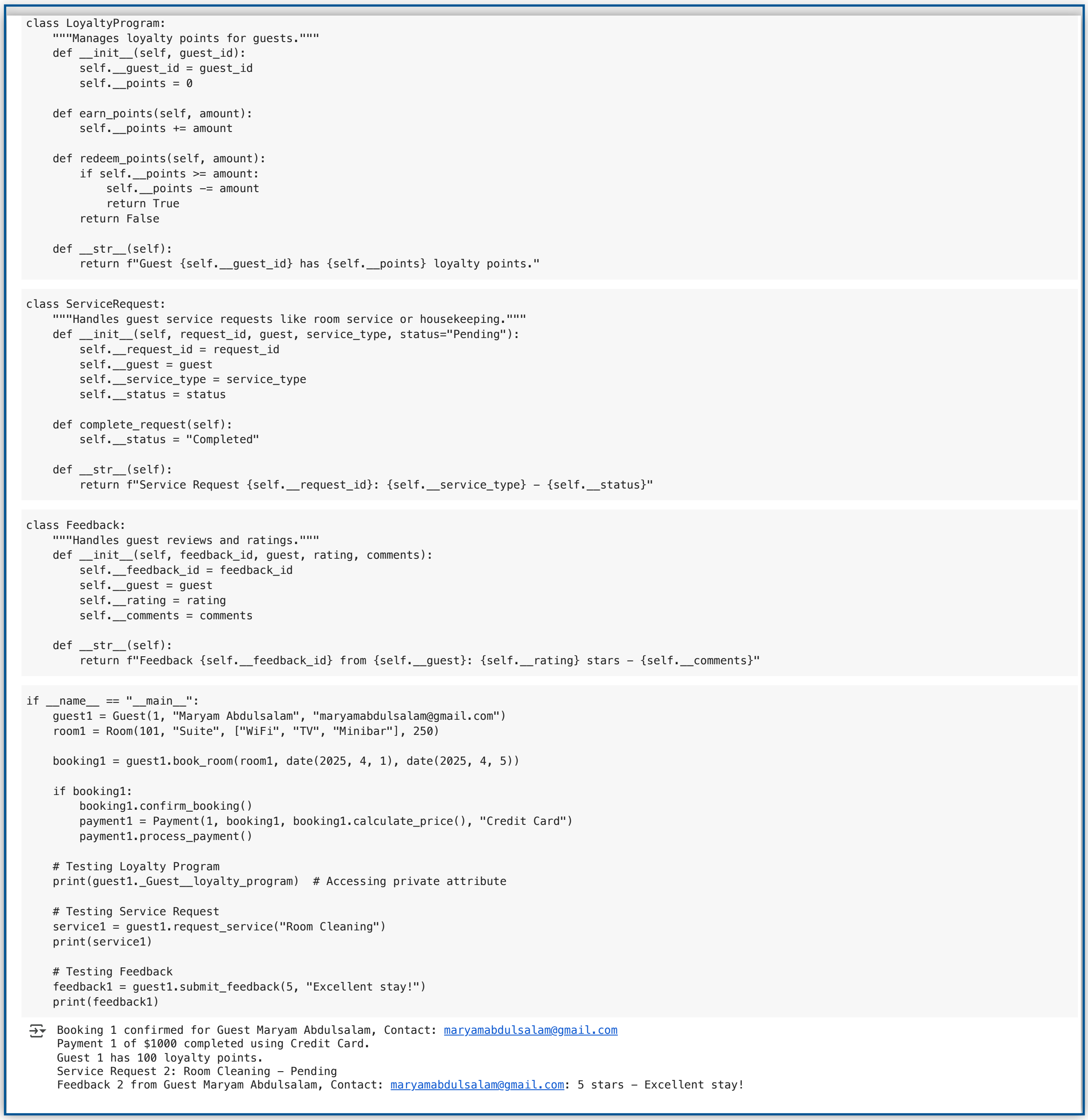
B. Write Python Code to Implement Your UML Class Diagram

```python
from datetime import date #Import the date class from the datetime module.

class Room:
    """Represents a hotel room with attributes like room number, type, amenities, and price."""
    def __init__(self, room_number, room_type, amenities, price_per_night):
        self.__room_number = room_number
        self.__room_type = room_type
        self.__amenities = amenities
        self.__price_per_night = price_per_night
        self.__availability = True  # Default: available

    def check_availability(self):
        return self.__availability #Returns True if the room is available, otherwise False.


    def reserve_room(self):
        if self.__availability:
            self.__availability = False
            return True
        return False

    def get_price(self):
        return self.__price_per_night

    def __str__(self):
        return f"Room {self.__room_number} ({self.__room_type}) - ${self.__price_per_night}/night"
```

```python
class Guest:
    """Represents a hotel guest with a unique ID, name, and contact details."""
    def __init__(self, guest_id, name, contact_info):
        self.__guest_id = guest_id
        self.__name = name
        self.__contact_info = contact_info
        self.__bookings = []
        self.__loyalty_program = LoyaltyProgram(guest_id)

    def book_room(self, room, check_in_date, check_out_date):
        if room.reserve_room():
            booking = Booking(len(self.__bookings) + 1, self, room, check_in_date, check_out_date)
            self.__bookings.append(booking)
            self.__loyalty_program.earn_points(booking.calculate_price() // 10)  # Earn 10% of booking cost as points
            return booking
        return None

    def request_service(self, service_type):
        return ServiceRequest(len(self.__bookings) + 1, self, service_type, "Pending")

    def submit_feedback(self, rating, comments):
        return Feedback(len(self.__bookings) + 1, self, rating, comments)

    def __str__(self):
        return f"Guest {self.__name}, Contact: {self.__contact_info}"
```

```python
from datetime import date

class Booking:
    """Handles room reservations, including check-in and check-out details."""
    def __init__(self, booking_id, guest, room, check_in_date, check_out_date):
        self.__booking_id = booking_id
        self.__guest = guest
        self.__room = room
        self.__check_in_date = check_in_date
        self.__check_out_date = check_out_date
        self.__total_price = self.calculate_price()

    def calculate_price(self):
        num_nights = (self.__check_out_date - self.__check_in_date).days
        #Calculates the total booking price based on the room's nightly rate and duration of stay.
        return num_nights * self.__room.get_price()

    def confirm_booking(self):
        print(f"Booking {self.__booking_id} confirmed for {self.__guest}")

    def cancel_booking(self):
        print(f"Booking {self.__booking_id} cancelled.")

    def __str__(self):
        return f"Booking {self.__booking_id}: {self.__guest} -> {self.__room} from {self.__check_in_date} to {self.__check_out_date}"
```

```python
class Payment:
    """Handles payments for bookings, supporting different methods."""
    def __init__(self, payment_id, booking, amount, payment_method):
        self.__payment_id = payment_id
        self.__booking = booking
        self.__amount = amount
        self.__payment_method = payment_method
        self.__payment_status = "Pending"

    def process_payment(self):
        self.__payment_status = "Completed" #Marks the payment as completed when successfully processed.
        print(f"Payment {self.__payment_id} of ${self.__amount} completed using {self.__payment_method}.")

    def __str__(self):
        return f"Payment {self.__payment_id}: ${self.__amount} - {self.__payment_status}"
```



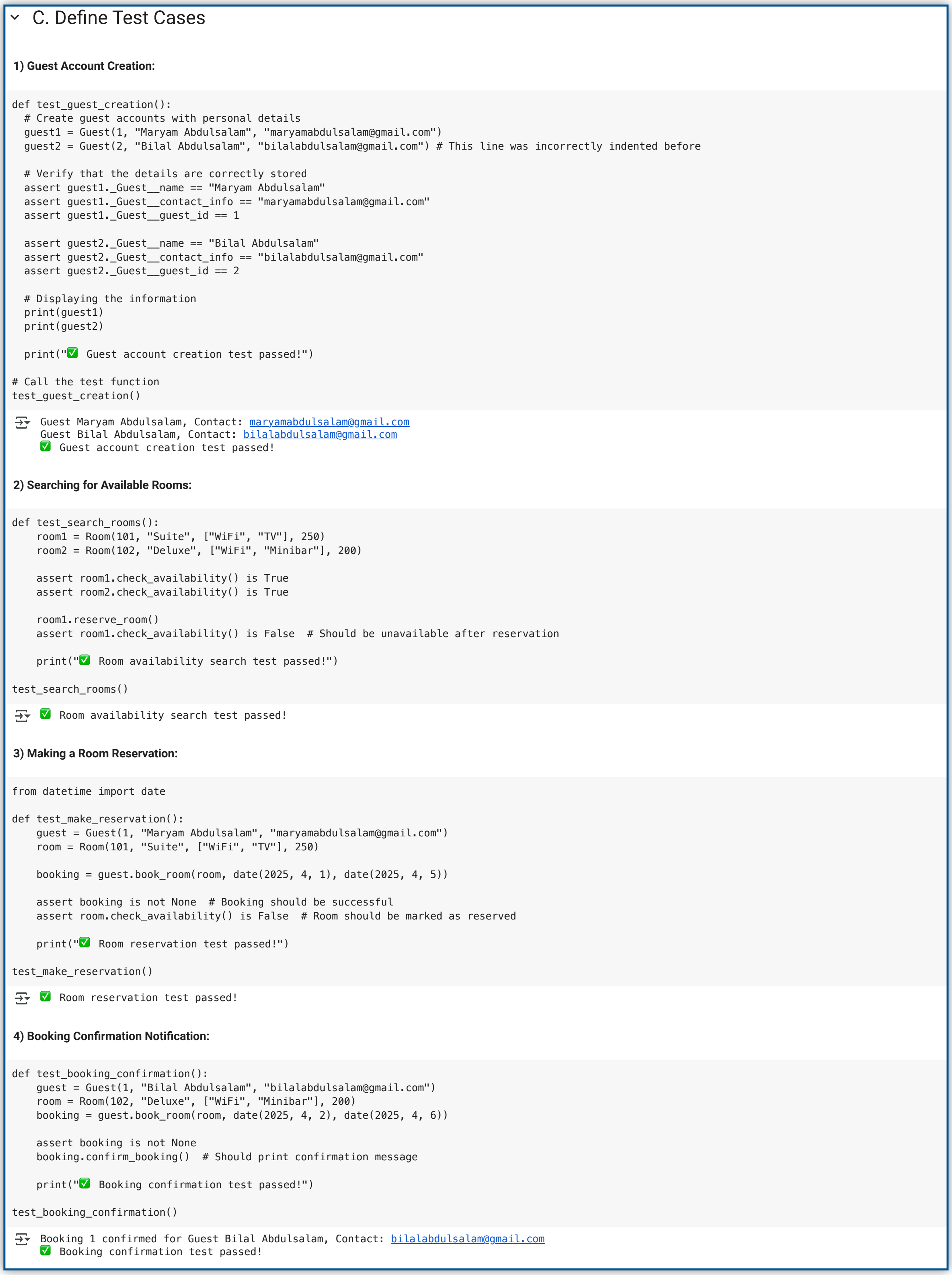
```python
class LoyaltyProgram:
    """Manages loyalty points for guests."""
    def __init__(self, guest_id):
        self.__guest_id = guest_id
        self.__points = 0

    def earn_points(self, amount):
        self.__points += amount

    def redeem_points(self, amount):
        if self.__points >= amount:
            self.__points -= amount
            return True
        return False

    def __str__(self):
        return f"Guest {self.__guest_id} has {self.__points} loyalty points."
```

```python
class ServiceRequest:
    """Handles guest service requests like room service or housekeeping."""
    def __init__(self, request_id, guest, service_type, status="Pending"):
        self.__request_id = request_id
        self.__guest = guest
        self.__service_type = service_type
        self.__status = status

    def complete_request(self):
        self.__status = "Completed"

    def __str__(self):
        return f"Service Request {self.__request_id}: {self.__service_type} - {self.__status}"
```

```python
class Feedback:
    """Handles guest reviews and ratings."""
    def __init__(self, feedback_id, guest, rating, comments):
        self.__feedback_id = feedback_id
        self.__guest = guest
        self.__rating = rating
        self.__comments = comments

    def __str__(self):
        return f"Feedback {self.__feedback_id} from {self.__guest}: {self.__rating} stars - {self.__comments}"
```

```python
if __name__ == "__main__":
    guest1 = Guest(1, "Maryam Abdulsalam", "maryamabdulsalam@gmail.com")
    room1 = Room(101, "Suite", ["WiFi", "TV", "Minibar"], 250)

    booking1 = guest1.book_room(room1, date(2025, 4, 1), date(2025, 4, 5))

    if booking1:
        booking1.confirm_booking()
        payment1 = Payment(1, booking1, booking1.calculate_price(), "Credit Card")
        payment1.process_payment()

    # Testing Loyalty Program
    print(guest1._Guest__loyalty_program)  # Accessing private attribute

    # Testing Service Request
    service1 = guest1.request_service("Room Cleaning")
    print(service1)

    # Testing Feedback
    feedback1 = guest1.submit_feedback(5, "Excellent stay!")
    print(feedback1)
```

```
Booking 1 confirmed for Guest Maryam Abdulsalam, Contact: maryamabdulsalam@gmail.com
Payment 1 of $1000 completed using Credit Card.
Guest 1 has 100 loyalty points.
Service Request 2: Room Cleaning - Pending
Feedback 2 from Guest Maryam Abdulsalam, Contact: maryamabdulsalam@gmail.com: 5 stars - Excellent stay!
```



C. Define Test Cases

1) Guest Account Creation:

```
def test_guest_creation():
  # Create guest accounts with personal details
  guest1 = Guest(1, "Maryam Abdulsalam", "maryamabdulsalam@gmail.com")
  guest2 = Guest(2, "Bilal Abdulsalam", "bilalabdulsalam@gmail.com") # This line was incorrectly indented before

  # Verify that the details are correctly stored
  assert guest1._Guest__name == "Maryam Abdulsalam"
  assert guest1._Guest__contact_info == "maryamabdulsalam@gmail.com"
  assert guest1._Guest__guest_id == 1

  assert guest2._Guest__name == "Bilal Abdulsalam"
  assert guest2._Guest__contact_info == "bilalabdulsalam@gmail.com"
  assert guest2._Guest__guest_id == 2

  # Displaying the information
  print(guest1)
  print(guest2)

  print("✅ Guest account creation test passed!")

# Call the test function
test_guest_creation()
```

```
Guest Maryam Abdulsalam, Contact: maryamabdulsalam@gmail.com
Guest Bilal Abdulsalam, Contact: bilalabdulsalam@gmail.com
✅ Guest account creation test passed!
```

2) Searching for Available Rooms:

```
def test_search_rooms():
    room1 = Room(101, "Suite", ["WiFi", "TV"], 250)
    room2 = Room(102, "Deluxe", ["WiFi", "Minibar"], 200)

    assert room1.check_availability() is True
    assert room2.check_availability() is True

    room1.reserve_room()
    assert room1.check_availability() is False  # Should be unavailable after reservation

    print("✅ Room availability search test passed!")

test_search_rooms()
```

```
✅ Room availability search test passed!
```

3) Making a Room Reservation:

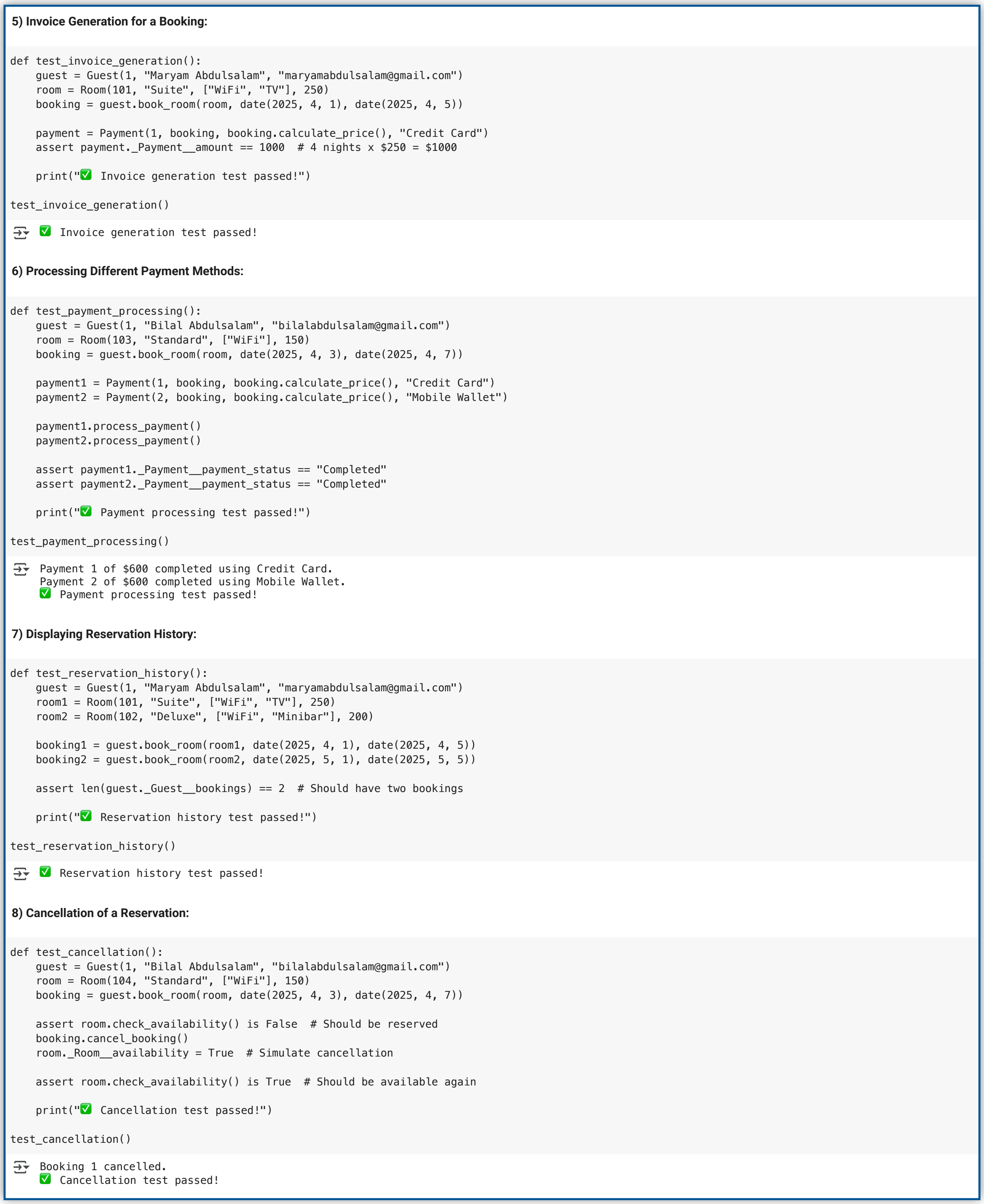
```
from datetime import date

def test_make_reservation():
    guest = Guest(1, "Maryam Abdulsalam", "maryamabdulsalam@gmail.com")
    room = Room(101, "Suite", ["WiFi", "TV"], 250)

    booking = guest.book_room(room, date(2025, 4, 1), date(2025, 4, 5))

    assert booking is not None  # Booking should be successful
    assert room.check_availability() is False  # Room should be marked as reserved

    print("✅ Room reservation test passed!")

test_make_reservation()
```

```
✅ Room reservation test passed!
```

4) Booking Confirmation Notification:

```
def test_booking_confirmation():
    guest = Guest(1, "Bilal Abdulsalam", "bilalabdulsalam@gmail.com")
    room = Room(102, "Deluxe", ["WiFi", "Minibar"], 200)
    booking = guest.book_room(room, date(2025, 4, 2), date(2025, 4, 6))

    assert booking is not None
    booking.confirm_booking()  # Should print confirmation message

    print("✅ Booking confirmation test passed!")

test_booking_confirmation()
```

```
Booking 1 confirmed for Guest Bilal Abdulsalam, Contact: bilalabdulsalam@gmail.com
✅ Booking confirmation test passed!
```



5) Invoice Generation for a Booking:

```
def test_invoice_generation():
    guest = Guest(1, "Maryam Abdulsalam", "maryamabdulsalam@gmail.com")
    room = Room(101, "Suite", ["WiFi", "TV"], 250)
    booking = guest.book_room(room, date(2025, 4, 1), date(2025, 4, 5))

    payment = Payment(1, booking, booking.calculate_price(), "Credit Card")
    assert payment._Payment__amount == 1000  # 4 nights x $250 = $1000

    print("✅ Invoice generation test passed!")

test_invoice_generation()
```

```
✅ Invoice generation test passed!
```

6) Processing Different Payment Methods:

```
def test_payment_processing():
    guest = Guest(1, "Bilal Abdulsalam", "bilalabdulsalam@gmail.com")
    room = Room(103, "Standard", ["WiFi"], 150)
    booking = guest.book_room(room, date(2025, 4, 3), date(2025, 4, 7))

    payment1 = Payment(1, booking, booking.calculate_price(), "Credit Card")
    payment2 = Payment(2, booking, booking.calculate_price(), "Mobile Wallet")

    payment1.process_payment()
    payment2.process_payment()

    assert payment1._Payment__payment_status == "Completed"
    assert payment2._Payment__payment_status == "Completed"

    print("✅ Payment processing test passed!")

test_payment_processing()
```

```
Payment 1 of $600 completed using Credit Card.
Payment 2 of $600 completed using Mobile Wallet.
✅ Payment processing test passed!
```

7) Displaying Reservation History:

```
def test_reservation_history():
    guest = Guest(1, "Maryam Abdulsalam", "maryamabdulsalam@gmail.com")
    room1 = Room(101, "Suite", ["WiFi", "TV"], 250)
    room2 = Room(102, "Deluxe", ["WiFi", "Minibar"], 200)

    booking1 = guest.book_room(room1, date(2025, 4, 1), date(2025, 4, 5))
    booking2 = guest.book_room(room2, date(2025, 5, 1), date(2025, 5, 5))

    assert len(guest._Guest__bookings) == 2  # Should have two bookings

    print("✅ Reservation history test passed!")

test_reservation_history()
```

```
✅ Reservation history test passed!
```

8) Cancellation of a Reservation:

```
def test_cancellation():
    guest = Guest(1, "Bilal Abdulsalam", "bilalabdulsalam@gmail.com")
    room = Room(104, "Standard", ["WiFi"], 150)
    booking = guest.book_room(room, date(2025, 4, 3), date(2025, 4, 7))

    assert room.check_availability() is False  # Should be reserved
    booking.cancel_booking()
    room._Room__availability = True  # Simulate cancellation

    assert room.check_availability() is True  # Should be available again

    print("✅ Cancellation test passed!")

test_cancellation()
```

```
Booking 1 cancelled.
✅ Cancellation test passed!
```

D. Summary of learnings:

This assignment improved my grasp of UML class diagrams for displaying system interactions, as well as my programming abilities with objects and classes. I learnt how to define system activities using use-case diagrams and form relationships with class diagrams. This is consistent with the first learning outcome, since I used UML notations to map real-world items.

One problem was accurately specifying UML relationships (such as "Include" and "Extend") and making sure access specifiers were right. I handled this by going over and editing diagrams and codes as necessary as seen in my Github repository uploads. Another problem was arranging classes based on criteria such as introducing placeholder methods and reflecting.
Part C was the so insightful for me, I worked on developing test cases to check the performance of guest creation, room reservations, payments, and other features, which improved my abilities to test and validate object-oriented programs. I also wrote clear, well-documented code that adhered to learning outcome regarding documentation to help with readability. Finally, GitHub helped me stay organized and up-to-date.

GitHub repository link:

https://github.com/MaryamAbdulsalam09/Royal-Stay-Hotel-Management-System-Maryam-Abdulsalam

References:

Minerva Project. (n.d.). LO1_OOAD: Analyze and design software that map real-world entities and relationships using Unified Modelling Language (UML) notations. Retrieved March 28, 2025, from https://forum.uae.minervaproject.com/app/outcome-index/learning-outcomes/LO1_OOAD?course_id=2165

Minerva Project. (n.d.). LO2_OOProgramming: Create working object-oriented programs in a computer language that are well-structured, error-free, and can solve computational problems. Retrieved March 28, 2025, from https://forum.uae.minervaproject.com/app/outcome-index/learning-outcomes/LO2_OOProgramming?course_id=2165

Minerva Project. (n.d.). LO4_SWDocumentation: Communicate with a clear and precise style that is suited to an appropriate audience to produce well-documented code, design documents, and presentations that are readable and understandable. Retrieved March 28, 2025, from https://forum.uae.minervaproject.com/app/outcome-index/learning-outcomes/LO4_SWDocumentation?course_id=2165

VDEngineering. (2022, January 23). *Python OOP (Object Oriented Programming) Project - A Hotel Reservation System - Complete explanation*. YouTube. https://www.youtube.com/watch?v=KhklWqco8W0

Chattergoon, B. (2023, February 16). *Working with payments data in Python and SQL*. Medium. https://medium.com/better-programming/working-with-payments-data-in-python-and-sql-44439a701a6b

QuantEcon. (n.d.). *Lecture-python-programming.myst*. GitHub. Retrieved March 28, 2025, from https://github.com/QuantEcon/lecture-python-programming.myst

Sargent, T. J., & Stachurski, J. (n.d.). *OOP II: Building classes*. Python Programming for Economics and Finance. https://python-programming.quantecon.org/python_oop.html

Mermaid Live. (n.d.). *Mermaid live editor.* https://mermaid.live/

Google Colab. (n.d.). *Collaboratory.* https://colab.google/