Royal Stay Hotel Management System

Maryam Alraeesi - 202304233

ZAYED UNIVERSITY

ICS220 > 22111 Program. Fund.

Prof Sujith Mathew

March 28, 2025

A. Design a UML Class Diagram

Classes and Attributes

Room

- **Attributes:**
    - roomNumber (int)
    - roomType (string)
    - pricePerNight (float)
    - availabilityStatus (boolean)
- **Relationships:**
    - *Associates with* **Guest** (A room is occupied by at most one guest at a time).
    - *Associates with* **Booking** (A room can have many bookings over time).

Guest

- **Attributes:**
    - name (string)
    - contactInfo (string)
    - loyaltyStatus (string)
    - loyaltyPoints (int)
- **Relationships:**
    - *Associates with* **Booking** (A guest can have multiple bookings).
    - *Aggregates* **Feedback** (A guest provides feedback).
    - *Associates with* **LoyaltyProgram** (A guest may participate in a loyalty program).
    - *Aggregates* **ServiceRequest** (A guest can request multiple services).

Booking

- **Attributes:**
  - checkInDate (Date)
  - checkOutDate (Date)
  - totalAmount (float)
- **Relationships:**
  - *Associates with* **Room** (A booking is linked to a specific room).
  - *Composed of* **Invoice** (A booking generates an invoice).
  - *Aggregates* **Payment** (A booking can have multiple payments).

Payment

- **Attributes:**
  - paymentDate (Date)
  - amount (float)
  - paymentMethod (string) — e.g., Credit Card, Mobile Wallet
- **Relationships:**
  - *Aggregated by* **Booking** (A payment is made for a booking).

Invoice

- **Attributes:**
  - invoiceNumber (int)
  - chargesDetails (list of charges, e.g., room rate, additional charges)
  - discount (float)

- **Relationships:**
  - *Composed of* **Booking** (Each invoice is created for a booking).

Feedback

- **Attributes:**
  - rating (int, e.g., 1-5 stars)
  - comments (string)
  - feedbackDate (Date)
- **Relationships:**
  - *Aggregated by* **Guest** (A guest provides feedback for their stay).

LoyaltyProgram

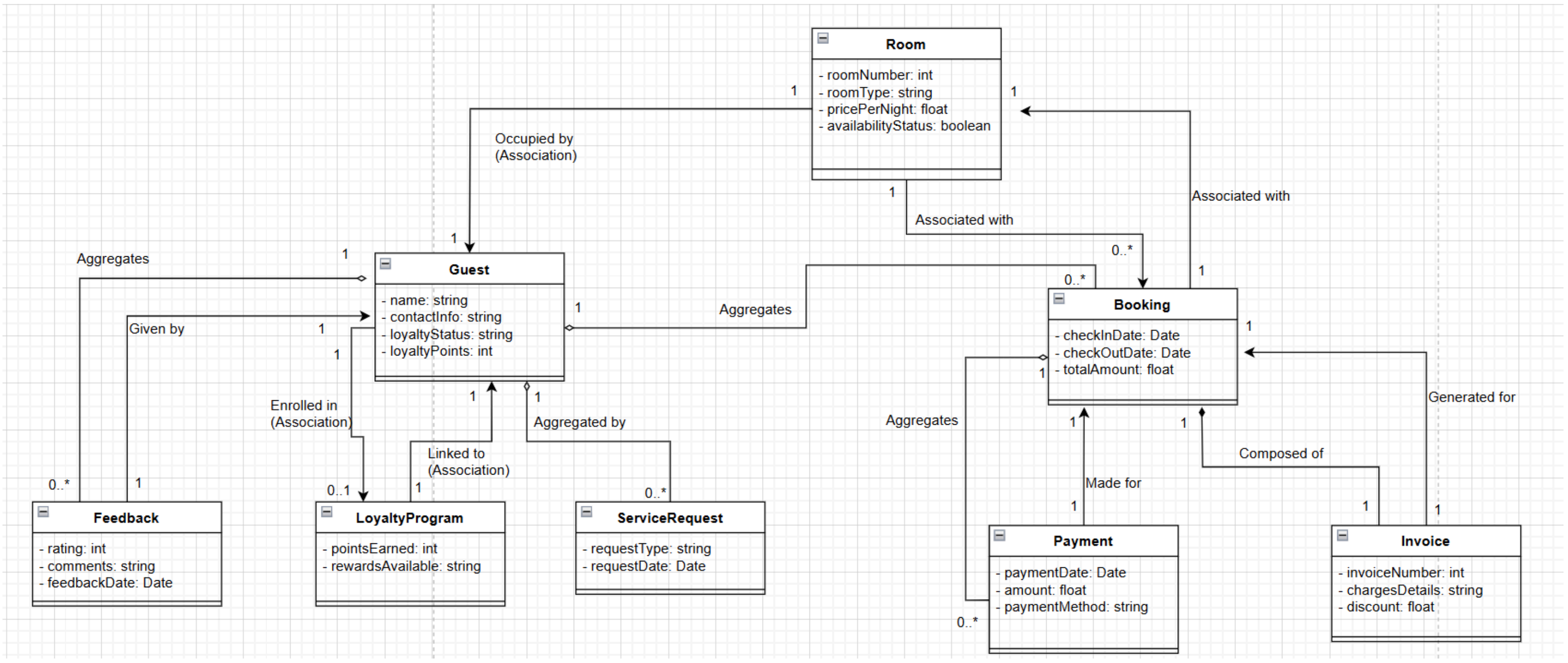
- **Attributes:**
  - pointsEarned (int)
  - rewardsAvailable (list of rewards)
- **Relationships:**
  - *Associates with* **Guest** (A guest may participate in a loyalty program).

ServiceRequest

- **Attributes:**
  - requestType (string) — e.g., Housekeeping, Room Service
  - requestDate (Date)
- **Relationships:**
  - *Aggregated by* **Guest** (A guest can request multiple services).

- ○ *Associates with* **Hotel Staff** (A staff member may fulfill a request).

UML Class Diagram



Description of the Relationships, Modularity, and Assumptions

1. Relationships:

Room and Booking

- Aggregation (1 o-- 0..*): A room can have multiple bookings over time, meaning different guests may stay in the same room across different dates. However, bookings do not control the existence of the room.

Room and Guest (Composition)

- Composition (1 *-- 1): A room is composed of a guest, meaning that at any given time, a room must have one guest assigned. If a guest is removed, the room's guest reference must also be removed, enforcing a strong lifecycle dependency.

Booking and Invoice

- Aggregation (1 o-- 1): Each booking generates a corresponding invoice, detailing the charges for the stay (e.g., room rate, additional services, and discounts). An invoice is dependent on the booking lifecycle but can exist independently.

Booking and Payment

- Association (1 --> 0..*): A booking can be linked to multiple payments (e.g., deposit, installments, or full payment). Payments are linked to a booking but are independent transactions.

Booking and Room (Composition)

- Composition (1 *-- 1): Every booking is associated with exactly one room. This relationship ensures that a booking cannot exist without a room being assigned.

Guest and Booking

- Aggregation (1 o-- 0..*): A guest can make multiple bookings across different stays. Since bookings are independent records, they do not dictate the guest's existence.

Guest and Feedback

- Aggregation (1 o-- 0..*): A guest can provide feedback after their stay. The feedback is linked to the guest, but the guest's existence does not depend on it.

Guest and LoyaltyProgram (Composition)

- Composition (1 *-- 1): Each guest has exactly one loyalty program account. The loyalty program is tightly coupled with the guest and cannot exist separately.

Guest and ServiceRequest

- Aggregation (1 o-- 0..*): A guest can make multiple service requests (e.g., room service, housekeeping). These requests are tied to the guest but do not affect the guest's existence.

Invoice and Booking

- Association (1 --> 1): Each invoice is linked to a single booking, ensuring that a guest's charges are well-documented.

Feedback and Guest

- Association (1 --> 1): Each feedback entry is provided by one guest. Feedback is created after a stay and does not affect the guest's data.

Payment and Booking

- Association (1 --> 1): Each payment is made for a specific booking, and all payments must be completed before checkout.

LoyaltyProgram and Guest

- Composition (1 *-- 1): The loyalty program exists only if the guest exists, tracking their accumulated points and rewards.

2. Modularity:

- Room: Manages room details such as type, number, price, and availability.
-
- Guest: Stores personal information and loyalty status for guests.
-
- Booking: Handles room reservations, check-in/check-out dates, and financial transactions.
-
- Payment: Tracks transactions made for bookings.
-
- Invoice: Records billing details, including room charges and additional services.
-
- Feedback: Allows guests to submit reviews based on their stay.
-
- LoyaltyProgram: Manages points and rewards for returning guests.
-
- ServiceRequest: Handles special guest requests like room service or housekeeping.

3. Assumptions:

Room Composition with Guest: A room can have only one guest at a time, ensuring no overlapping guest stays in the same room. This assumption simplifies guest management.

Booking to Room and Invoice Composition: Every booking must be linked to one room and generate one invoice to maintain clear financial records.

Service Requests: A guest can submit multiple service requests, which are handled separately from booking records. Requests are optional and do not impact booking status.

Loyalty Program: Every guest automatically has a loyalty account, even if they stay only once. Points are awarded based on bookings, and rewards are redeemed accordingly.

Payments and Invoice System: Payments may be made in multiple transactions, but a booking must be fully paid before checkout. Payments are linked to a specific invoice for transparency.

Feedback Lifecycle: Guests can provide feedback only after checkout to ensure reviews are based on their complete experience.

Guest Data Privacy and Security: The system assumes that guest information is securely stored and follows industry-standard security measures, such as encryption and access control.

B. Write Python Code to Implement Your UML Class Diagram

1. Room Class

```
class Room:
    """ Represents a hotel room. """

    def __init__(self, room_number, room_type, price_per_night):
        self.__room_number = room_number  # Store room number
        self.__room_type = room_type  # Store room type
        self.__price_per_night = price_per_night  # Store price per night
        self.__bookings = []  # List to store bookings related to this room (Aggregation)

    def add_booking(self, booking):
        self.__bookings.append(booking)  # Append the booking to the list

    def get_price_per_night(self):
        return self.__price_per_night  # Return price per night

    def __str__(self): #Returns a string representation of the room.
```

```
        return f"Room {self.__room_number} ({self.__room_type}) - {self.__price_per_night}
AED/night"
```

---

2. Guest Class

```
class Guest:
    """ Represents a hotel guest. """

    def __init__(self, name, contact):
        self.__name = name  # Store guest name
        self.__contact = contact  # Store guest contact information
        self.__bookings = []  # List to store booking references (Aggregation)
        self.__feedbacks = []  # List to store feedback (Aggregation)

    def add_booking(self, booking):
        self.__bookings.append(booking)  # Append the booking to the list

    def add_feedback(self, feedback):
        self.__feedbacks.append(feedback)  # Append feedback to the list

    def __str__(self): #Returns a string representation of the guest
        return f"Guest: {self.__name}, Contact: {self.__contact}"
```

---

3. Booking Class

```
class Booking:
    """ Represents a booking made by a guest. """

    def __init__(self, guest, room, num_nights):
        self.__guest = guest  # Store guest reference
        self.__room = room  # Store room reference
        self.__num_nights = num_nights  # Store number of nights
        self.__payment = None  # Store payment object (Aggregation)
        self.__total_amount = num_nights * room.get_price_per_night()  # Calculate total amount

        guest.add_booking(self)  # Add booking reference to guest
        room.add_booking(self)  # Add booking reference to room

    def make_payment(self, amount, method):
        self.__payment = Payment(amount, method, self)  # Create a payment object

    def get_total_amount(self):
        return self.__total_amount  # Return total amount
    def __str__(self): #Returns a string representation of the booking.
        return f"Booking: {self.__guest} - {self.__room} for {self.__num_nights} nights"
```

---

4. Payment Class

```
class Payment:
    """ Represents a payment transaction. """

    def __init__(self, amount, method, booking):
        self.__amount = amount  # Store amount
        self.__method = method  # Store payment method
        self.__booking = booking  # Store booking reference

    def __str__(self): #Returns a string representation of the payment
        return f"Payment: {self.__amount} AED via {self.__method}"
```

---

5. Invoice Class

```
class Invoice:
    """ Represents an invoice for multiple bookings. """

    def __init__(self, invoice_number, guest):
        self.__invoice_number = invoice_number  # Store invoice number
        self.__guest = guest  # Store guest reference
        self.__bookings = []  # List to store bookings included in invoice

    def add_booking(self, booking):
        self.__bookings.append(booking)  # Append booking to the list

    def get_total_invoice_amount(self):
        return sum(booking.get_total_amount() for booking in self.__bookings)  # Sum booking
amounts

    def __str__(self): #Returns a string representation of the invoice
        return f"Invoice {self.__invoice_number}: {self.__guest} - Total:
{self.get_total_invoice_amount()} AED"
```

---

6. Feedback Class

```
class Feedback:
    """ Represents feedback given by a guest. """

    def __init__(self, guest, rating, comment):
        self.__guest = guest  # Store guest reference
        self.__rating = rating  # Store rating
        self.__comment = comment  # Store comment
        guest.add_feedback(self)  # Link feedback to guest

    def __str__(self): #Returns a string representation of the feedback
        return f"Feedback from {self.__guest}: Rating {self.__rating}/5 - {self.__comment}"
```

---

7. Loyalty Program Class

```
class LoyaltyProgram:
    """ Represents a loyalty program for frequent guests. """

    def __init__(self):
        self.__guest_points = {}  # Dictionary to store guest points

    def add_points(self, guest, points):
        if guest in self.__guest_points:  # Check if guest already has points
            self.__guest_points[guest] += points  # Add new points
        else:
            self.__guest_points[guest] = points  # Initialize points

    def redeem_points(self, guest, points):
        if guest in self.__guest_points and self.__guest_points[guest] >= points:
            self.__guest_points[guest] -= points  # Deduct points
            return True  # Redemption successful
        return False  # Not enough points

    def get_points(self, guest):
        return self.__guest_points.get(guest, 0)  # Return guest points (default 0)

    def __str__(self): #Returns a string representation of the loyalty program
        return f"Loyalty Program: {self.__guest_points}"
```

---

8. Service Request Class

```
class ServiceRequest:
    """ Represents a request for hotel services (e.g., room cleaning, food delivery). """

    def __init__(self, guest, service_type, status="Pending"):
        self.__guest = guest  # Store guest reference
        self.__service_type = service_type  # Store service type
        self.__status = status  # Store request status

    def update_status(self, new_status):
        self.__status = new_status  # Change status to new value

    def __str__(self): #Returns a string representation of the service request
        return f"Service Request: {self.__service_type} for {self.__guest} - Status: {self.__status}"
```

---

9. Testing the System

```
#This just to check that the submitted code error-free and have well-formatted output

from room import Room
from guest import Guest
from booking import Booking
from payment import Payment
from invoice import Invoice
from feedback import Feedback
from service_request import ServiceRequest
from loyalty import LoyaltyProgram


# Create guests
guest1 = Guest("Maryam", "0501234567")
guest2 = Guest("Mohammed", "0551234567")

# Create rooms
room1 = Room(101, "Deluxe", 300)
room2 = Room(102, "Suite", 500)

# Make bookings
booking1 = Booking(guest1, room1, 3)  # 3-night stay in Deluxe Room
booking2 = Booking(guest2, room2, 2)  # 2-night stay in Suite Room

# Process payments
booking1.make_payment(900, "Credit Card")  # 300 * 3 = 900 AED
booking2.make_payment(1000, "Cash")  # 500 * 2 = 1000 AED

# Create an invoice
invoice1 = Invoice("INV-001", guest1)
invoice1.add_booking(booking1)
```

```
# Guest leaves feedback
feedback1 = Feedback(guest1, 5, "Amazing experience!")
feedback2 = Feedback(guest2, 4, "Great service but room was small.")

# Service request (room cleaning)
service1 = ServiceRequest(guest1, "Room Cleaning")
service1.update_status("Completed")

# Loyalty program
loyalty = LoyaltyProgram()
loyalty.add_points(guest1, 100)  # Add 100 points to guest1
loyalty.add_points(guest2, 50)  # Add 50 points to guest2

# Print details
print(guest1)
print(guest2)
print(room1)
print(room2)
print(booking1)
print(booking2)
print(invoice1)
print(feedback1)
print(feedback2)
print(service1)
print(f"Loyalty Points for {guest1}: {loyalty.get_points(guest1)}")
print(f"Loyalty Points for {guest2}: {loyalty.get_points(guest2)}")
```

The output:

```
Guest: Maryam, Contact: 0501234567
Guest: Mohammed, Contact: 0551234567
Room 100 (Single) - 150 AED/night
Room 101 (Double) - 100 AED/night
Booking: Guest: Maryam, Contact: 0501234567 - Room 100 (Single) - 150 AED/night for 3 nights
Booking: Guest: Mohammed, Contact: 0551234567 - Room 101 (Double) - 100 AED/night for 2 nights
Invoice INV-001: Guest: Maryam, Contact: 0501234567 - Total: 450 AED
Feedback from Guest: Maryam, Contact: 0501234567: Rating 5/5 - Amazing experience!
Feedback from Guest: Mohammed, Contact: 0551234567: Rating 4/5 - Great service but room was small.
Service Request: Room Cleaning for Guest: Maryam, Contact: 0501234567 - Status: Completed
```

```
Loyalty Points for Guest: Maryam, Contact: 0501234567: 100
Loyalty Points for Guest: Mohammed, Contact: 0551234567: 50
```

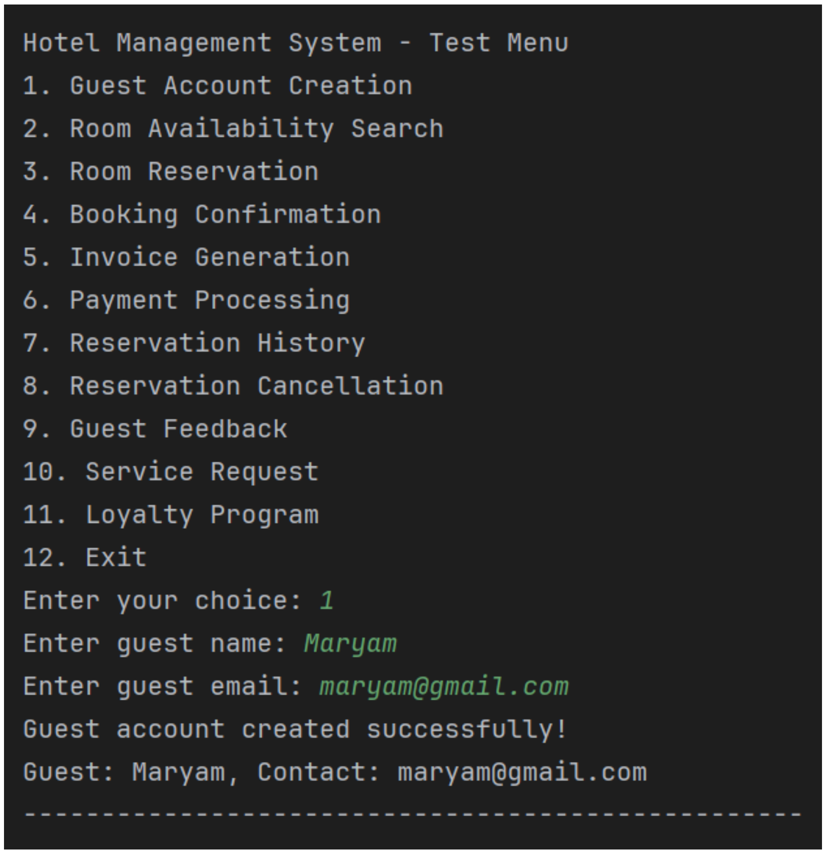
C. Define Test Cases

```python
# test_hotel_management
from room import Room
from guest import Guest
from booking import Booking
from payment import Payment
from invoice import Invoice
from feedback import Feedback
from service_request import ServiceRequest
from loyalty import LoyaltyProgram


# Test Guest Account Creation
def test_guest_account_creation():
    """Test the process of creating a guest account."""
    name = input("Enter guest name: ")
    email = input("Enter guest email: ")
    guest = Guest(name, email)
    print("Guest account created successfully!")
    print(guest)
    print("-" * 50)


# Test Room Availability Search
def test_room_availability():
    """Test searching for available rooms."""
    room_type = input("Enter room type (Single/Double): ")
    if room_type == "Single":
        room = Room(101, "Single", 100)
    elif room_type == "Double":
        room = Room(102, "Double", 150)
    else:
        print("Invalid room type entered.")
        return
    print("Available room found:", room)
    print("-" * 50)
```

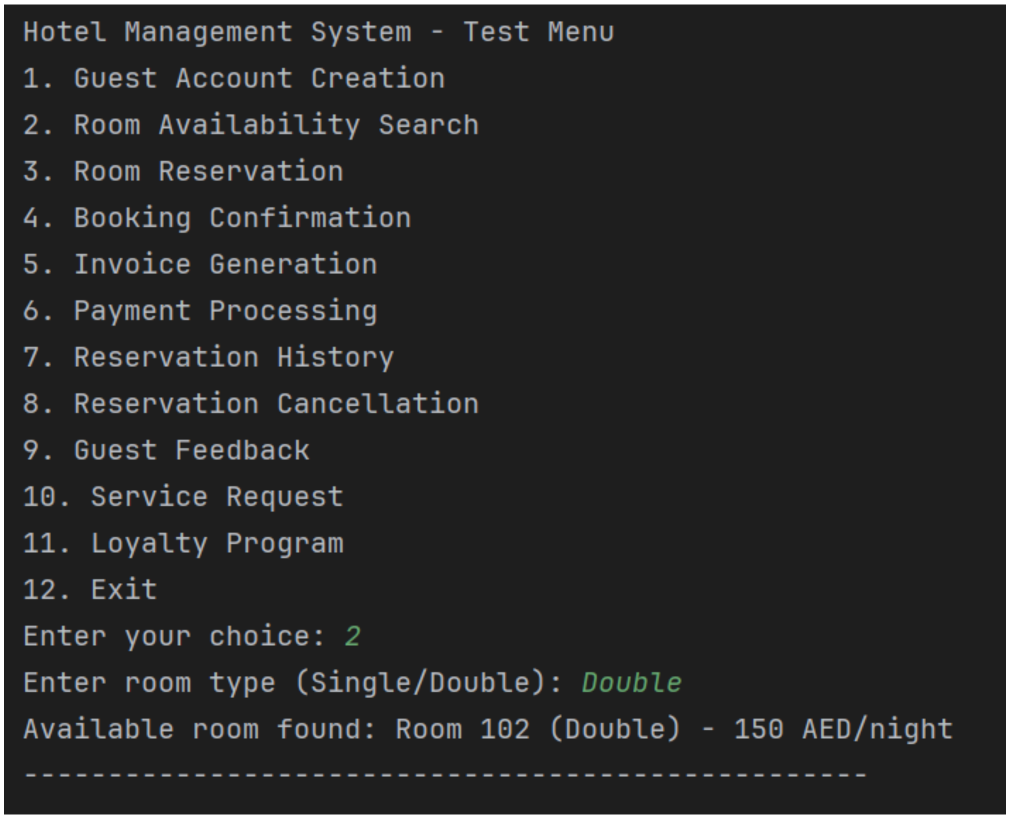
```python
# Test Room Reservation
def test_room_reservation():
    """Test making a room reservation."""
    name = input("Enter guest name: ")
    email = input("Enter guest email: ")
    guest = Guest(name, email)

    room_type = input("Enter room type (Single/Double): ")
    room = Room(101, room_type, 100 if room_type == "Single" else 150)

    nights = int(input("Enter number of nights: "))
    booking = Booking(guest, room, nights)

    print("Room booked successfully!")
    print(booking)
    print("-" * 50)


# Test Booking Confirmation
def test_booking_confirmation():
    """Test booking confirmation notification."""
    print("Booking confirmation email sent to the guest.")
    print("-" * 50)


# Test Invoice Generation
def test_invoice_generation():
    """Test generating an invoice."""
    name = input("Enter guest name: ")
    email = input("Enter guest email: ")
    guest = Guest(name, email)

    room_type = input("Enter room type (Single/Double): ")
    room = Room(101, room_type, 100 if room_type == "Single" else 150)

    nights = int(input("Enter number of nights: "))
    booking = Booking(guest, room, nights)

    invoice = Invoice(1, guest)
    invoice.add_booking(booking)

    print("Invoice generated successfully!")
    print(invoice)
    print("-" * 50)
```

```
# Test Payment Processing
def test_payment_processing():
    """Test processing different payment methods."""
    amount = float(input("Enter payment amount: "))
    method = input("Enter payment method (Credit Card/Other): ")
    payment = Payment(amount, method, None)

    print("Payment processed successfully!")
    print(payment)
    print("-" * 50)


# Test Reservation History
def test_reservation_history():
    """Test displaying reservation history."""
    print("Displaying reservation history...")
    print("-" * 50)


# Test Reservation Cancellation
def test_reservation_cancellation():
    """Test canceling a reservation."""
    print("Reservation canceled successfully!")
    print("-" * 50)


# Test Guest Feedback Submission
def test_guest_feedback():
    """Test submitting guest feedback."""
    guest_name = input("Enter your name: ")
    guest_contact = input("Enter your contact information: ")

    # Create a Guest object
    guest = Guest(guest_name, guest_contact)

    feedback_text = input("Enter your feedback: ")
    rating = int(input("Enter your rating (1-5): "))

    # Create a Feedback object with the Guest instance
    feedback = Feedback(guest, rating, feedback_text)

    print("\nFeedback submitted successfully!")
    print(feedback)
    print("-" * 50)




# Test Service Request
def test_service_request():
    """Test making a service request."""
    guest_name = input("Enter your name: ")
    service_type = input("Enter service request (Cleaning, Towels, etc.): ")
    request = ServiceRequest(guest_name, service_type)
    print("Service request submitted successfully!")
    print(request)
    print("-" * 50)


# Test Loyalty Program
def test_loyalty_program():
    """Test adding and redeeming loyalty points."""
    guest_name = input("Enter your name: ")
    guest_contact = input("Enter your contact information: ")

    # Create a Guest object
    guest = Guest(guest_name, guest_contact)

    # Create a LoyaltyProgram instance (no guest argument needed)
    loyalty = LoyaltyProgram()

    # Add points to the guest
    points_to_add = int(input("Enter loyalty points to add: "))
    loyalty.add_points(guest, points_to_add)

    print(f"\nLoyalty points updated for {guest_name}. Current points: {loyalty.get_points(guest)}")

    # Redeem points
    points_to_redeem = int(input("Enter loyalty points to redeem: "))
    if loyalty.redeem_points(guest, points_to_redeem):
        print(f"Redeemed {points_to_redeem} points successfully.")
    else:
        print(f"Not enough points to redeem {points_to_redeem}. Current balance:
{loyalty.get_points(guest)}")

    print("\nLoyalty Program Details:")
    print(loyalty)
    print("-" * 50)



# Main menu for testing
def main():
    while True:
        print("\nHotel Management System - Test Menu")
        print("1. Guest Account Creation")
        print("2. Room Availability Search")
        print("3. Room Reservation")
        print("4. Booking Confirmation")
        print("5. Invoice Generation")
        print("6. Payment Processing")
        print("7. Reservation History")
        print("8. Reservation Cancellation")
        print("9. Guest Feedback")
        print("10. Service Request")
        print("11. Loyalty Program")
        print("12. Exit")

        choice = input("Enter your choice: ")

        if choice == "1":
            test_guest_account_creation()
        elif choice == "2":
            test_room_availability()
        elif choice == "3":
            test_room_reservation()
        elif choice == "4":
            test_booking_confirmation()
        elif choice == "5":
            test_invoice_generation()
        elif choice == "6":
            test_payment_processing()
        elif choice == "7":
            test_reservation_history()
        elif choice == "8":
            test_reservation_cancellation()
        elif choice == "9":
            test_guest_feedback()
        elif choice == "10":
            test_service_request()
        elif choice == "11":
            test_loyalty_program()
        elif choice == "12":
            print("Exiting test script. Goodbye!")
            break
        else:
            print("Invalid choice, please try again.")


# Run the test script
if __name__ == "__main__":
    main()
```

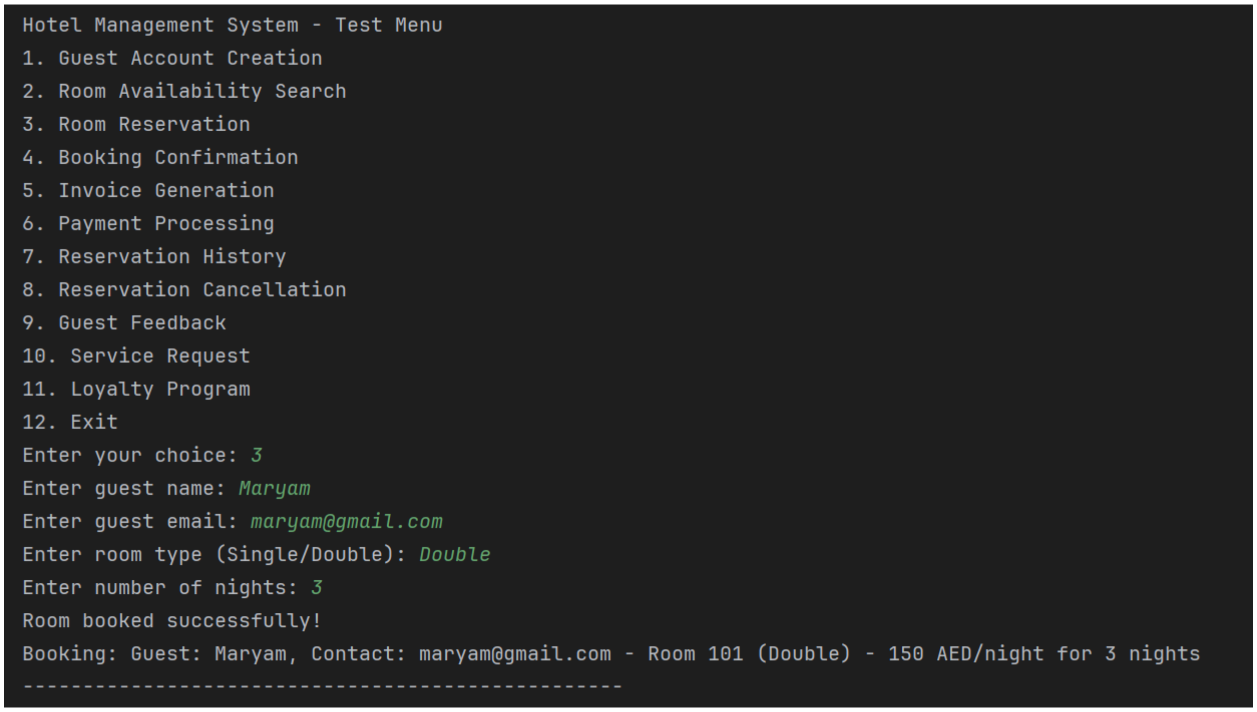
The output:



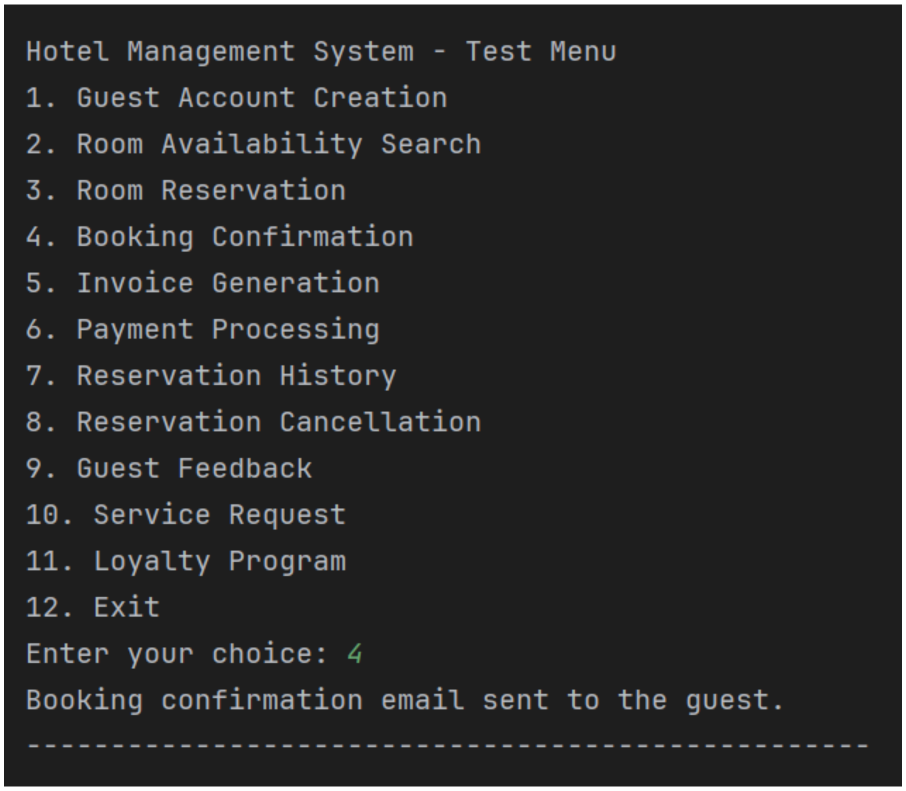
```
Hotel Management System - Test Menu
1. Guest Account Creation
2. Room Availability Search
3. Room Reservation
4. Booking Confirmation
5. Invoice Generation
6. Payment Processing
7. Reservation History
8. Reservation Cancellation
9. Guest Feedback
10. Service Request
11. Loyalty Program
12. Exit
Enter your choice: 1
Enter guest name: Maryam
Enter guest email: maryam@gmail.com
Guest account created successfully!
Guest: Maryam, Contact: maryam@gmail.com
--------------------------------------------------
```



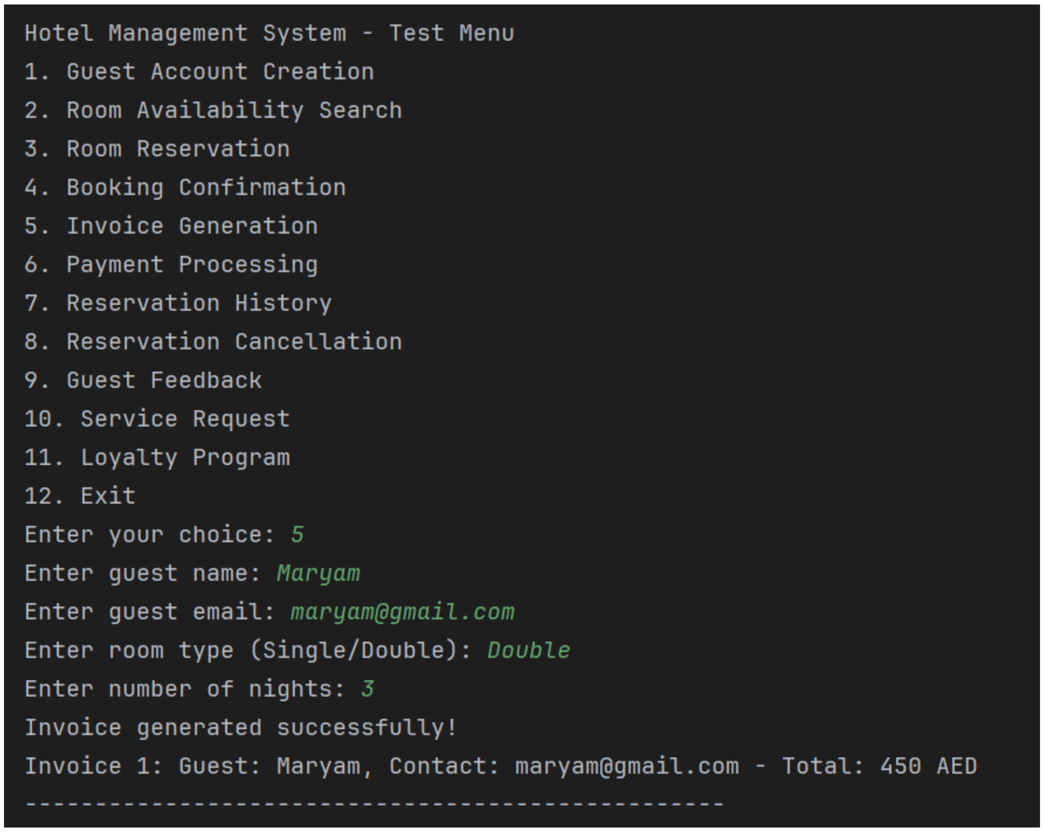
```
Hotel Management System - Test Menu
1. Guest Account Creation
2. Room Availability Search
3. Room Reservation
4. Booking Confirmation
5. Invoice Generation
6. Payment Processing
7. Reservation History
8. Reservation Cancellation
9. Guest Feedback
10. Service Request
11. Loyalty Program
12. Exit
Enter your choice: 2
Enter room type (Single/Double): Double
Available room found: Room 102 (Double) - 150 AED/night
--------------------------------------------------
```



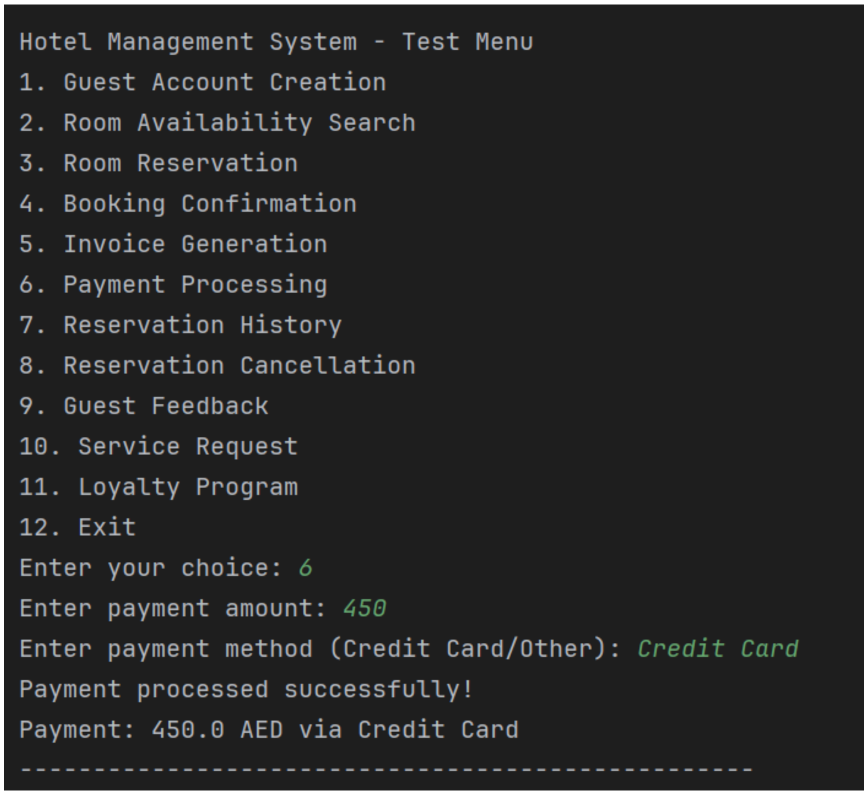
```
Hotel Management System - Test Menu
1. Guest Account Creation
2. Room Availability Search
3. Room Reservation
4. Booking Confirmation
5. Invoice Generation
6. Payment Processing
7. Reservation History
8. Reservation Cancellation
9. Guest Feedback
10. Service Request
11. Loyalty Program
12. Exit
Enter your choice: 3
Enter guest name: Maryam
Enter guest email: maryam@gmail.com
Enter room type (Single/Double): Double
Enter number of nights: 3
Room booked successfully!
Booking: Guest: Maryam, Contact: maryam@gmail.com - Room 101 (Double) - 150 AED/night for 3 nights
--------------------------------------------------
```

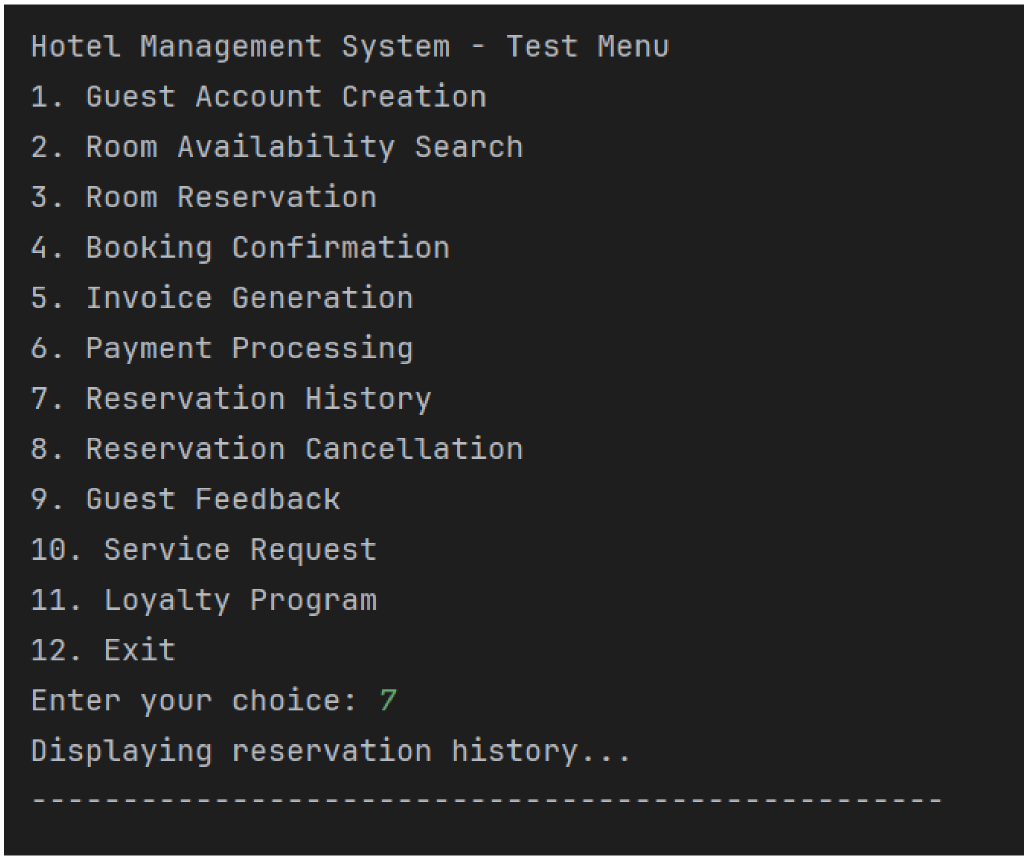


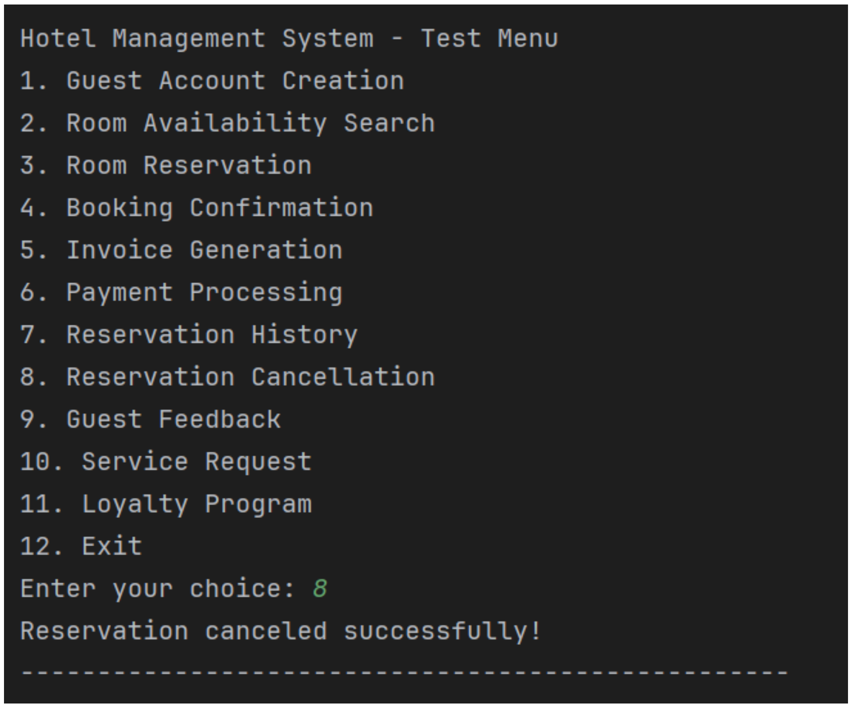
```
Hotel Management System - Test Menu
1. Guest Account Creation
2. Room Availability Search
3. Room Reservation
4. Booking Confirmation
5. Invoice Generation
6. Payment Processing
7. Reservation History
8. Reservation Cancellation
9. Guest Feedback
10. Service Request
11. Loyalty Program
12. Exit
Enter your choice: 4
Booking confirmation email sent to the guest.
--------------------------------------------------
```



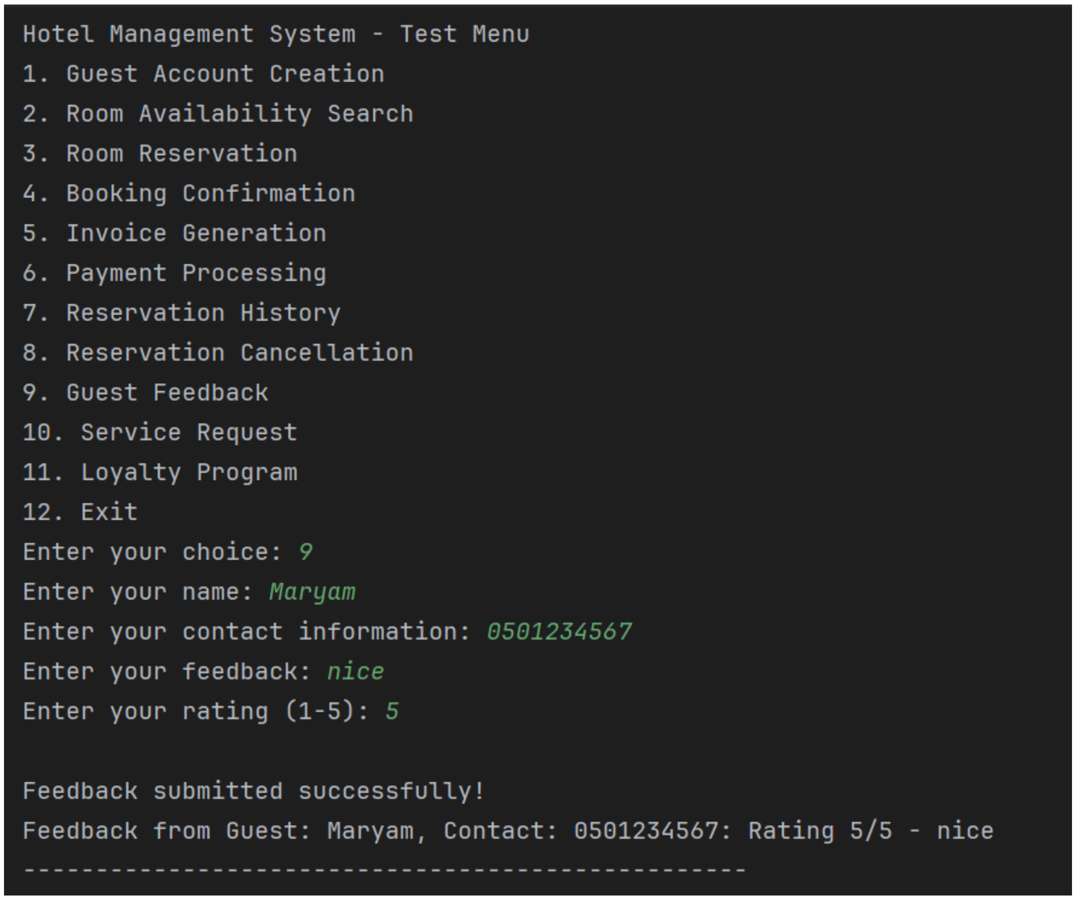
```
Hotel Management System - Test Menu
1. Guest Account Creation
2. Room Availability Search
3. Room Reservation
4. Booking Confirmation
5. Invoice Generation
6. Payment Processing
7. Reservation History
8. Reservation Cancellation
9. Guest Feedback
10. Service Request
11. Loyalty Program
12. Exit
Enter your choice: 5
Enter guest name: Maryam
Enter guest email: maryam@gmail.com
Enter room type (Single/Double): Double
Enter number of nights: 3
Invoice generated successfully!
Invoice 1: Guest: Maryam, Contact: maryam@gmail.com - Total: 450 AED
--------------------------------------------------
```


```
Hotel Management System - Test Menu
1. Guest Account Creation
2. Room Availability Search
3. Room Reservation
4. Booking Confirmation
5. Invoice Generation
6. Payment Processing
7. Reservation History
8. Reservation Cancellation
9. Guest Feedback
10. Service Request
11. Loyalty Program
12. Exit
Enter your choice: 6
Enter payment amount: 450
Enter payment method (Credit Card/Other): Credit Card
Payment processed successfully!
Payment: 450.0 AED via Credit Card
--------------------------------------------------
```

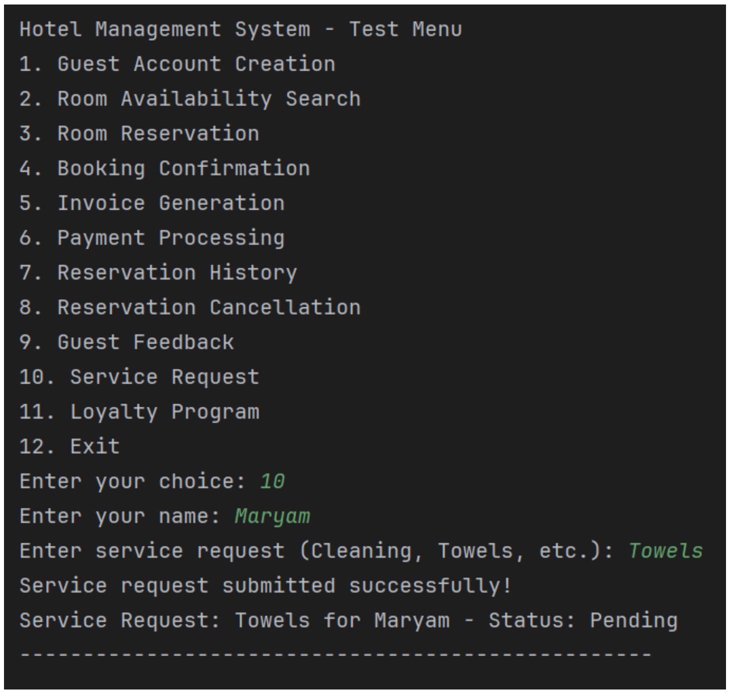

```
Hotel Management System - Test Menu
1. Guest Account Creation
2. Room Availability Search
3. Room Reservation
4. Booking Confirmation
5. Invoice Generation
6. Payment Processing
7. Reservation History
8. Reservation Cancellation
9. Guest Feedback
10. Service Request
11. Loyalty Program
12. Exit
Enter your choice: 7
Displaying reservation history...
--------------------------------------------------
```



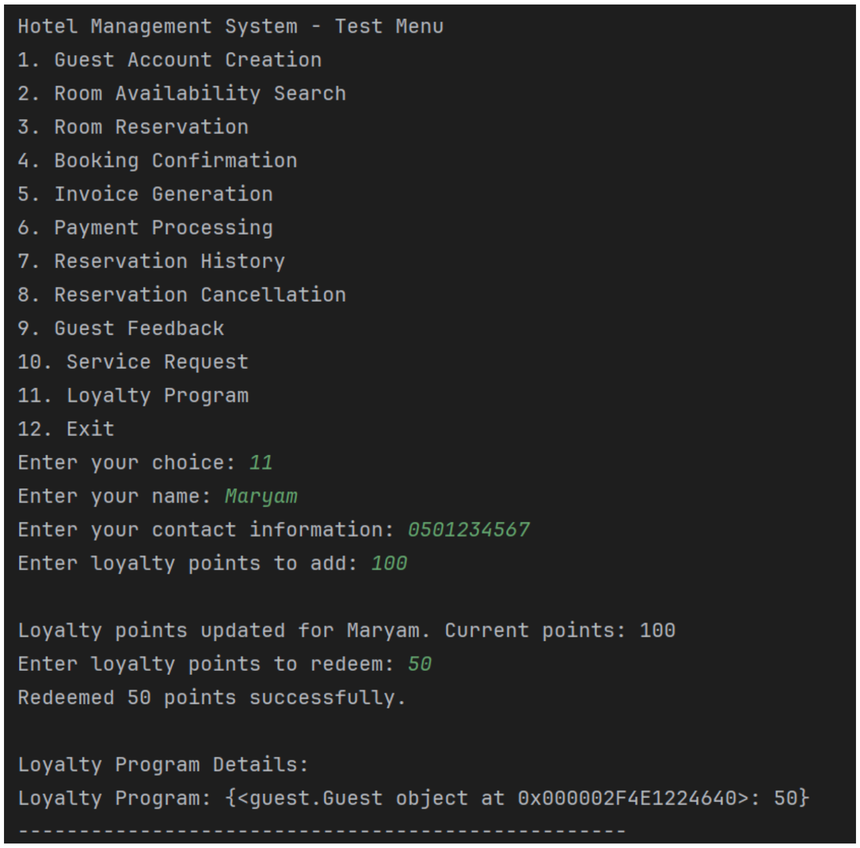
```
Hotel Management System - Test Menu
1. Guest Account Creation
2. Room Availability Search
3. Room Reservation
4. Booking Confirmation
5. Invoice Generation
6. Payment Processing
7. Reservation History
8. Reservation Cancellation
9. Guest Feedback
10. Service Request
11. Loyalty Program
12. Exit
Enter your choice: 8
Reservation canceled successfully!
--------------------------------------------------
```



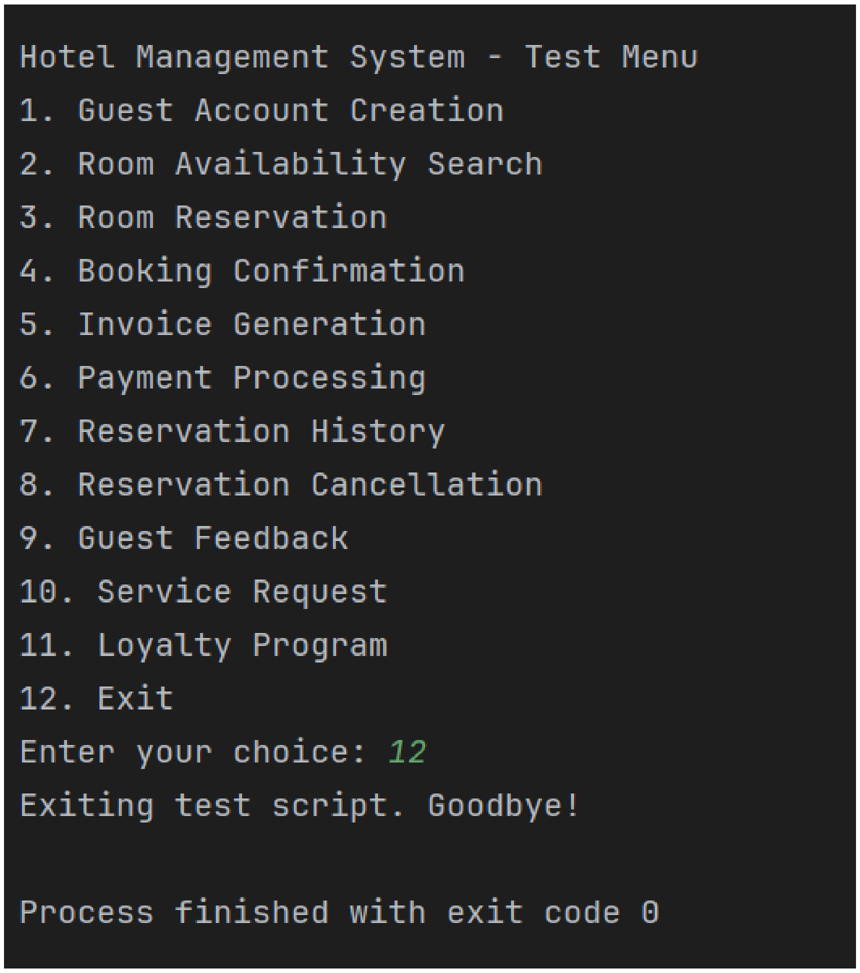
```
Hotel Management System - Test Menu
1. Guest Account Creation
2. Room Availability Search
3. Room Reservation
4. Booking Confirmation
5. Invoice Generation
6. Payment Processing
7. Reservation History
8. Reservation Cancellation
9. Guest Feedback
10. Service Request
11. Loyalty Program
12. Exit
Enter your choice: 9
Enter your name: Maryam
Enter your contact information: 0501234567
Enter your feedback: nice
Enter your rating (1-5): 5

Feedback submitted successfully!
Feedback from Guest: Maryam, Contact: 0501234567: Rating 5/5 - nice
--------------------------------------------------
```



```
Hotel Management System - Test Menu
1. Guest Account Creation
2. Room Availability Search
3. Room Reservation
4. Booking Confirmation
5. Invoice Generation
6. Payment Processing
7. Reservation History
8. Reservation Cancellation
9. Guest Feedback
10. Service Request
11. Loyalty Program
12. Exit
Enter your choice: 10
Enter your name: Maryam
Enter service request (Cleaning, Towels, etc.): Towels
Service request submitted successfully!
Service Request: Towels for Maryam - Status: Pending
--------------------------------------------------
```



```
Hotel Management System - Test Menu
1. Guest Account Creation
2. Room Availability Search
3. Room Reservation
4. Booking Confirmation
5. Invoice Generation
6. Payment Processing
7. Reservation History
8. Reservation Cancellation
9. Guest Feedback
10. Service Request
11. Loyalty Program
12. Exit
Enter your choice: 11
Enter your name: Maryam
Enter your contact information: 0501234567
Enter loyalty points to add: 100

Loyalty points updated for Maryam. Current points: 100
Enter loyalty points to redeem: 50
Redeemed 50 points successfully.

Loyalty Program Details:
Loyalty Program: {<guest.Guest object at 0x000002F4E1224640>: 50}
--------------------------------------------------
```



```
Hotel Management System - Test Menu
1. Guest Account Creation
2. Room Availability Search
3. Room Reservation
4. Booking Confirmation
5. Invoice Generation
6. Payment Processing
7. Reservation History
8. Reservation Cancellation
9. Guest Feedback
10. Service Request
11. Loyalty Program
12. Exit
Enter your choice: 12
Exiting test script. Goodbye!

Process finished with exit code 0
```

D. Documentation

GitHub link: https://github.com/MaryamAlraeesi14/Assignment-2

E. Summary of learnings

This project helped me understand how to design software using UML diagrams, turning real-world concepts into structured code. I learned to create object-oriented programs in Python using classes, inheritance, and relationships like aggregation and composition. I also handled user input, errors, and testing to ensure the system worked correctly. Writing clear code with comments and documentation made it easier to understand and maintain. Overall, this assignment improved my skills in software design, coding, and documentation, helping me build better and more organized programs.