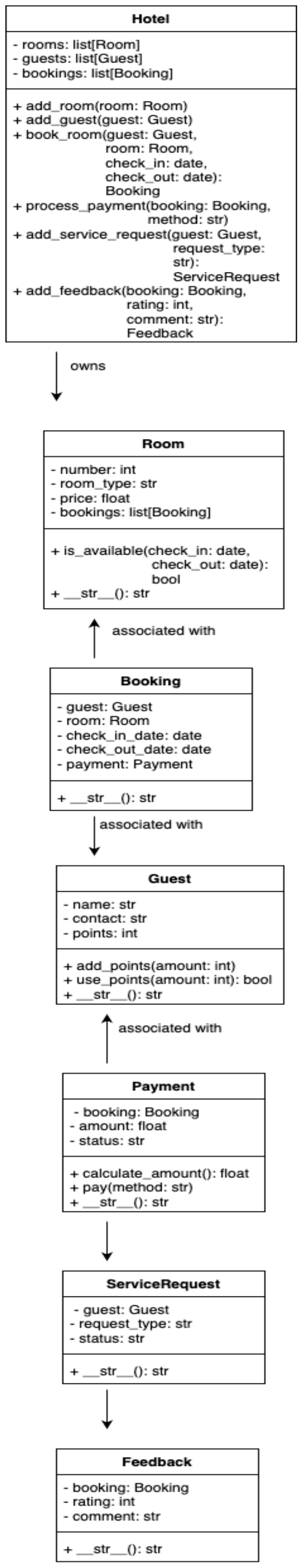
Royal Stay Hotel Management System

Abdulrahman Alzaabi

ICS220 Program. Fund.

Areej Abdulfattah

A. UML case diagram:



B. Description

The UML class diagram consists of seven classes:

- **Hotel**: Central class managing rooms, guests, and bookings.
- **Room**: Represents hotel rooms with availability functionality.
- **Guest**: Stores guest information and loyalty points.
- **Booking**: Links guests to rooms for specific dates.
- **Payment**: Manages payment details for bookings.
- **ServiceRequest**: Handles guest service requests.
- **Feedback**: Records guest feedback for bookings.

Relationships:

- **Composition**: Hotel owns Room, Guest, and Booking.
- **Association**: Booking connects to Guest, Room, and Payment; ServiceRequest to Guest; Feedback to Booking.

C.

Code:

```
from datetime import date



class Room:

    """Represents a hotel room with number, type, price, and bookings."""

    def __init__(self, number, room_type, price):

        self.number = number

        self.room_type = room_type

        self.price = price

        self.bookings = []



    def is_available(self, check_in, check_out):

        """Check if the room is available between the given dates."""
        for booking in self.bookings:
            if (check_in < booking.check_out_date and check_out > booking.check_in_date):
                return False
        return True


    def __str__(self):
        """Return a string representation of the room."""
        return f"Room {self.number} ({self.room_type}), ${self.price}/night"


class Guest:
    """Represents a guest with name, contact, and loyalty points."""
    def __init__(self, name, contact):
        self.name = name
        self.contact = contact
        self.points = 0


    def add_points(self, amount):
        """Add loyalty points to the guest's account."""
        self.points += amount


    def use_points(self, amount):
        """Use loyalty points if sufficient are available."""
        if self.points >= amount:
            self.points -= amount
            return True
        return False
    def __str__(self):
        """Return a string representation of the guest."""
        return f"Guest: {self.name}, Points: {self.points}"


class Booking:
    """Represents a booking made by a guest for a room."""
    def __init__(self, guest, room, check_in_date, check_out_date):
        self.guest = guest
        self.room = room
        self.check_in_date = check_in_date
        self.check_out_date = check_out_date
        self.room.bookings.append(self)
        self.payment = None


    def __str__(self):
        """Return a string representation of the booking."""
        return f"Booking for {self.guest.name} in Room {self.room.number} from {self.check_in_date} to {self.check_out_date}"


class Payment:
    """Represents a payment for a booking."""
    def __init__(self, booking):
        self.booking = booking
        self.amount = self.calculate_amount()
        self.status = "Pending"


    def calculate_amount(self):
        """Calculate the total amount based on the number of nights."""
        nights = (self.booking.check_out_date - self.booking.check_in_date).days

        return nights * self.booking.room.price



    def pay(self, method):

        """Process the payment and update the status."""

        self.status = f"Paid via {method}"

        self.booking.guest.add_points(int(self.amount))



    def __str__(self):

        """Return a string representation of the payment."""

        return f"Payment for {self.booking}: ${self.amount}, {self.status}"



class ServiceRequest:

    """Represents a service request made by a guest."""

    def __init__(self, guest, request_type):

        self.guest = guest

        self.request_type = request_type

        self.status = "Pending"



    def __str__(self):

        """Return a string representation of the service request."""

        return f"Service Request from {self.guest.name}: {self.request_type} ({self.status})"



class Feedback:

    """Represents feedback for a booking."""

    def __init__(self, booking, rating, comment):

        self.booking = booking
        self.rating = rating
        self.comment = comment

    def __str__(self):
        """Return a string representation of the feedback."""
        return f"Feedback for {self.booking}: Rating {self.rating}/5, Comment: {self.comment}"

class Hotel:
    """Represents the hotel managing rooms, guests, and bookings."""
    def __init__(self):
        self.rooms = []
        self.guests = []
        self.bookings = []

    def add_room(self, room):
        """Add a room to the hotel."""
        self.rooms.append(room)

    def add_guest(self, guest):
        """Add a guest to the hotel."""
        self.guests.append(guest)

    def book_room(self, guest, room, check_in, check_out):
        """Book a room if available and create a payment."""
        if room.is_available(check_in, check_out):
            booking = Booking(guest, room, check_in, check_out)
            self.bookings.append(booking)
            booking.payment = Payment(booking)
            return booking
        else:
            print(f"Error: Room {room.number} is not available.")
            return None

    def process_payment(self, booking, method):
        """Process the payment for a booking."""
        if booking.payment:
            booking.payment.pay(method)

    def add_service_request(self, guest, request_type):
        """Add a service request for a guest."""
        return ServiceRequest(guest, request_type)

    def add_feedback(self, booking, rating, comment):
        """Add feedback for a booking."""
        return Feedback(booking, rating, comment)

if __name__ == "__main__":
    hotel = Hotel()
    room1 = Room(101, "Single", 100)
    room2 = Room(102, "Double", 150)
    hotel.add_room(room1)
    hotel.add_room(room2)
    abdulrahman = Guest("Abdulrahman", "abdulrahman@gmail.com")
    fatima = Guest("Fatima", "fatima@gmail.com")
```

```
hotel.add_guest(abdulrahman)

hotel.add_guest(fatima)

booking1 = hotel.book_room(abdulrahman, room1, date(2025, 4, 1), date(2025, 4, 3))

hotel.process_payment(booking1, "Credit Card")

booking2 = hotel.book_room(fatima, room2, date(2025, 4, 1), date(2025, 4, 3))

hotel.process_payment(booking2, "Cash")

print(booking2.payment)

print(f"Fatima's points: {fatima.points}")

print("Abdulrahman tries to use 100 points:", abdulrahman.use_points(100))

print(f"Abdulrahman's points after: {abdulrahman.points}")

service = hotel.add_service_request(abdulrahman, "Room Service")

print(service)

feedback = hotel.add_feedback(booking1, 5, "Great stay!")

print(feedback)
```

Test Cases

The test cases are implemented in the if __name__ == "__main__": block and include:

- **Guest Creation**: Creating guests Abdulrahman and Fatima.
- **Room Booking**: Booking Room 101 for Abdulrahman and Room 102 for Fatima.
- **Payment Processing**: Processing payments via Credit Card and Cash.
- **Loyalty Points**: Earning points from payments and using 100 points by Abdulrahman.
- **Service Request**: Adding a "Room Service" request for Abdulrahman.
- **Feedback**: Submitting feedback for Abdulrahman’s booking.

Output:

```
Payment for Booking for Fatima in Room 102 from 2025-04-01 to 2025-04-03: $300, Paid via Cash

Fatima's points: 300

Abdulrahman tries to use 100 points: True

Abdulrahman's points after: 100

Service Request from Abdulrahman: Room Service (Pending)

Feedback for Booking for Abdulrahman in Room 101 from 2025-04-01 to 2025-04-03: Rating 5/5, Comment: Great stay!
```

GitGub Link:

https://github.com/Aboood2220/Royal-Stay-Hotel-Management-System

Summary:

In this project I learned to use Python classes and objects to build a hotel system. I found out how a hotel links to rooms and guests. Drawing a UML diagram helped me plan before coding which made things easier.

Writing the code was hard sometimes especially making all the classes work together. But I learned why keeping code neat.

This project showed me how to use object-oriented programming to make a system step by step and now I feel better about using these skills later whenever needed.