Assignment #2

Reem Alhamami

202302133

Programming Fundamentals - ICS220

Areej Abdulfattah - 23018

28 March 2025

#LO1_OOAD, #LO2_OOProgramming, #LO4_SWDocumentation

Part A: Classes (Attributes, Methods, Relationships)

1. Room
    a. **Responsibility:** Which represents hotel rooms, storing information like room type, price, and availability.
    b. **Attributes:** room number, room type, amenities, price per night, availability.
    c. **Methods:** mark available, mark unavailable.
    d. **Relationship:** Composition with Reservation (A reservation cannot exist without a room).
2. Guest
    a. **Responsibility:** Stores guest information like personal details and loyalty status.
    b. **Attributes:** guest ID, guest name, guest email, guest phone number, feedback points.
    c. **Methods:** update profile, view reservation history.
    d. **Relationship:**
        i. Association with Reservation: A guest can make multiple reservations.
        ii. Aggregation with Loyalty Program: A guest may have a loyalty program account.
3. Reservation
    a. **Responsibility:** Manages room bookings, including check-in and check-out dates.
    b. **Attributes:** reservation ID, guest, room, check in date, check out date, status.
    c. **Methods:** confirm the reservation, cancel the reservation, update the dates.
    d. **Relationship:**
        i. Composition with Room: A reservation is linked to a specific room.

        ii. Association with Guest: A guest can have multiple reservations.
        iii. Association with Invoice: Each reservation generates an invoice.

4. Invoice
    a. **Responsibility:** Generates an invoice for completed reservations.
    b. **Attributes:** invoice ID, reservation, total amount, additional charges, discount.
    c. **Methods:** generate the invoice, apply any discount, calculate the total.
    d. **Relationship:**
        i. Association with reservation: Each reservation generates an invoice.
        ii. association with payment: A payment is linked to an invoice.
5. Payment
    a. **Responsibility:** Handles payment processing.
    b. **Attributes:** Payment info, invoice, payment method, payment status.
    c. **Methods:** process payment, refund payment.
    d. **Relationship:**
        i. Aggregation with Invoice: Payments are linked to invoices.
6. Loyalty Program
    a. **Responsibility:** Manages guest loyalty points and rewards.
    b. **Attributes:** Guest, point balance, rewards available
    c. **Methods:** Earn points, redeem points, view rewards.
    d. **Relationship:**
        i. Aggregation with Guest: A guest can participate in a loyalty program.
7. Service Request
    a. **Responsibility:** Handles guest requests for additional services.
    b. **Attributes:** Request info, guest, request type, status.

c. **Methods**: submit request, update status.

d. **Relationship:**

   i. Association with Guest: Guests can make service requests.

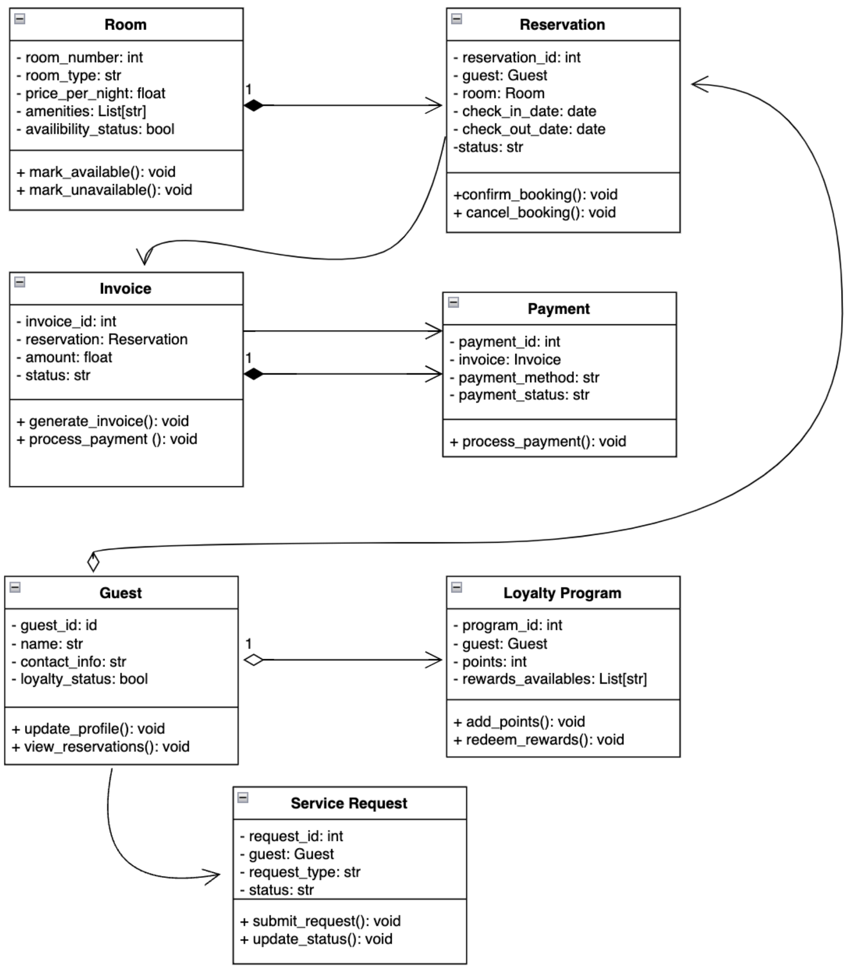


figure 1: *UML Diagram for the Hotel System, Using Draw.io*

Part B: Copy of the Code

This is a copy of the code, the original code is uploaded in Github

```
#Room Class
```

```
class Room:

   def __init__(self, room_number, room_type, price_per_night,
amenities):

       self.room_number = room_number          # int
       self.room_type = room_type                    # str
       self.price_per_night = price_per_night        # float
       self.amenities = amenities                    # List[str]
       self.availability_status = True               # bool (default:
available)


   def mark_available(self):
       """Set room status to available."""
       self.availability_status = True


   def mark_unavailable(self):
       """Set room status to unavailable."""
       self.availability_status = False
```

```
#Guest Class
class Guest:
   def __init__(self, guest_id, name, contact_info):
       self.guest_id = guest_id                      # int
       self.name = name                              # str
       self.contact_info = contact_info               # str
       self.loyalty_status = False                   # bool (default: not
enrolled)
       self.reservations = []                        # List[Reservation]
(track bookings)


   def update_profile(self, new_name, new_contact):
       """Update guest details."""
       self.name = new_name
       self.contact_info = new_contact


   def view_reservations(self):
       """Return all reservations linked to this guest."""
       return self.reservations
```

```
#Reservation Class
class Reservation:
    def __init__(self, reservation_id, guest, room, check_in_date,
check_out_date):
        self.reservation_id = reservation_id     # int
        self.guest = guest                       # Guest object
(composition)
        self.room = room                         # Room object
(composition)
        self.check_in_date = check_in_date       # date
        self.check_out_date = check_out_date     # date
        self.status = "Pending"                  # str
(Pending/Confirmed/Cancelled)


    def confirm_booking(self):
        """Confirm reservation if room is available."""
        if self.room.availability_status:
            self.status = "Confirmed"
            self.room.mark_unavailable()

            self.guest.reservations.append(self)   # Link to guest's 
booking history

        else:

            raise ValueError("Room is already booked.")


    def cancel_booking(self):

        """Cancel reservation and free the room."""

        if self.status == "Confirmed":

            self.status = "Cancelled"

            self.room.mark_available()

            self.guest.reservations.remove(self)

        else:

            raise ValueError("Cannot cancel a non-confirmed booking.")
```

```
#Invoice Class

class Invoice:

    def __init__(self, invoice_id, reservation):

        self.invoice_id = invoice_id             # int

        self.reservation = reservation           # Reservation object 
(composition)

        self.amount = self.calculate_amount()    # float

        self.status = "Unpaid"                   # str (Unpaid/Paid)


    def calculate_amount(self):
        """Compute total cost based on reservation duration."""

        nights = (self.reservation.check_out_date -
self.reservation.check_in_date).days

        return nights * self.reservation.room.price_per_night



    def generate_invoice(self):

        """Return invoice details as a formatted string."""

        return f"Invoice #{self.invoice_id}: ${self.amount} (Status:
{self.status})"



    def process_payment(self):

        """Mark invoice as paid."""

        self.status = "Paid"
```

```
#Payment class

class Payment:

    def __init__(self, payment_id, invoice, payment_method):

        self.payment_id = payment_id                # int

        self.invoice = invoice                      # Invoice object
(composition)

        self.payment_method = payment_method        # str (e.g., "Credit
Card")

        self.payment_status = "Pending"             # str
(Pending/Completed/Failed)



    def process_payment(self):

        """Process payment and update invoice status."""
```

```python
        try:
            self.invoice.process_payment()        # Calls Invoice's method
            self.payment_status = "Completed"
        except Exception as e:
            self.payment_status = "Failed"
            raise Exception(f"Payment failed: {str(e)}")
```

```python
#Loyalty Program
class LoyaltyProgram:
    def __init__(self, program_id, guest):
        self.program_id = program_id             # int
        self.guest = guest                       # Guest object (composition)
        self.points = 0                          # int (default: 0)
        self.rewards_available = ["Free Night", "Upgrade"]  # List[str]

    def add_points(self, points_earned):
        """Add points to the guest's loyalty account."""
        if self.guest.loyalty_status:
            self.points += points_earned
        else:
            raise ValueError("Guest is not enrolled in the loyalty program.")

    def redeem_rewards(self, reward):
        """Redeem a reward if points are sufficient."""
        if reward in self.rewards_available:
            if reward == "Free Night" and self.points >= 100:
                self.points -= 100
                return "Free Night reward claimed!"
            elif reward == "Upgrade" and self.points >= 50:
                self.points -= 50
                return "Room upgrade claimed!"
            else:
                raise ValueError("Not enough points for this reward.")
        else:
            raise ValueError("Invalid reward.")
```

```python
#Service Request Class
class ServiceRequest:
    def __init__(self, request_id, guest, request_type):
        self.request_id = request_id            # int
        self.guest = guest                      # Guest object
(composition)
        self.request_type = request_type        # str (e.g., "Room
Cleaning")
        self.status = "Open"                    # str (Open/In
Progress/Closed)


    def submit_request(self):
        """Submit a new service request."""
        if self.status == "Open":

            self.guest.service_requests.append(self)  # Link to guest's
requests

        else:

            raise ValueError("Request already submitted.")



    def update_status(self, new_status):

        """Update the request's status."""

        valid_statuses = ["Open", "In Progress", "Closed"]

        if new_status in valid_statuses:

            self.status = new_status

        else:

            raise ValueError("Invalid status.")
```

Part C: Test Cases

```
# --- Workflow Execution (Part C) ---
print("=== Hotel Management System Simulation ===")

# 1. Initialize Objects
room_101 = Room(101, "Deluxe", 200.00, ["WiFi", "Mini-Cafe"])
guest_john = Guest(1, "Reem Alhamami", "Reem@Alhamami.com")
reservation = Reservation(1, guest_john, room_101, date(2025, 3, 15),
date(2025, 4, 1))
invoice = Invoice(1, reservation)
payment = Payment(1, invoice, "Credit Card")
loyalty_program = LoyaltyProgram(1, guest_john)
```

```python
guest_john.loyalty_status = True  # Activate loyalty program
service_request = ServiceRequest(1, guest_john, "Extra Towels")

# 2. Execute Workflow
print("\n=== Booking Phase ===")
reservation.confirm_booking()
print(f"Reservation Status: {reservation.status}")
print(f"Room 101 Available: {room_101.availability_status}")

print("\n=== Payment Phase ===")
print(invoice.generate_invoice())
payment.process_payment()
print(f"Payment Status: {payment.payment_status}")
print(f"Invoice Status: {invoice.status}")

print("\n=== Loyalty Program ===")
loyalty_program.add_points(150)
print(f"Loyalty Points: {loyalty_program.points}")
print(loyalty_program.redeem_rewards("Free Night"))

print("\n=== Service Request ===")
service_request.submit_request()
service_request.update_status("In Progress")
print(f"Service Status: {service_request.status}")
```

Code Outputs:

```
=== Hotel Management System Simulation ===

=== Booking Phase ===
Reservation Status: Confirmed
Room 101 Available: False

=== Payment Phase ===
Invoice #1: $3400.00 (Status: Unpaid)
Payment Status: Completed
Invoice Status: Paid

=== Loyalty Program ===
Loyalty Points: 150
Free Night reward claimed!

=== Service Request ===
Service Status: In Progress
```

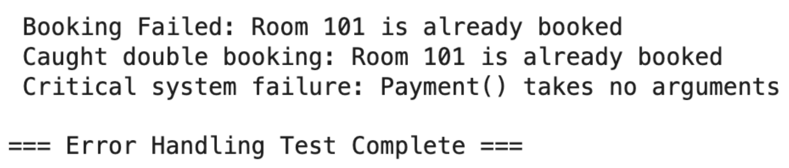
Testing by Error handling:

```python
# --- Testing the Error Handling ---
```

```
try:

   # Force error scenarios:

   room_101 = Room(101, "Deluxe", 200, ["WiFi"])

   guest = Guest(1, "Reem", "Reem@gmail.com")



   # 1. Test invalid dates

   try:

       bad_res = Reservation(1, guest, room_101, date(2025,3,20),
date(2025,3,30))

   except InvalidDateError as e:

       print(f" Caught invalid dates: {e}")



   # 2. Test double booking

   res1 = Reservation(1, guest, room_101, date(2025,3,20),
date(2025,3,30))

   res1.confirm_booking()



   try:

       res2 = Reservation(2, guest, room_101, date(2025,3,23),
date(2025,4,1))

       res2.confirm_booking()

   except RoomNotAvailableError as e:

       print(f" Caught double booking: {e}")



   # 3. Test payment failure

   invoice = Invoice(1, res1)
```

```
    payment = Payment(1, invoice, "Cryptocurrency")  # Unsupported method

    try:
        payment.process_payment()
    except PaymentProcessingError as e:
        print(f" Caught payment error: {e}")

except Exception as e:
    print(f" Critical system failure: {e}")
finally:
    print("\n=== Error Handling Test Complete ===")
```

Code Outputs:



```
 Booking Failed: Room 101 is already booked
 Caught double booking: Room 101 is already booked
 Critical system failure: Payment() takes no arguments

=== Error Handling Test Complete ===
```

- System Workflow testing is showing how the program works when everything goes as planned.
- Error Handling focuses on errors and problems, like when a room is already booked or payment fails, and it's like having a backup plan.

<u>**Summary**</u>

In this assignment, I started by identifying the main classes needed for the Hotel management system, along with their attributes, methods, and relationships. After that, I created a UML class diagram to visually represent how these classes are connected. In Part B, I implemented the design by writing Python code for each class, making sure to include all the attributes, methods, and relationships I had defined earlier. Finally, in Part C, I tested the code in two different ways. First, I checked the workflow to ensure that the system functions as expected when the classes interact with each other. Then, I tested error handling to make sure the program can handle invalid inputs or unexpected situations without crashing. These tests are important because they help verify that the system works correctly and can handle different scenarios smoothly.