Assignment 2

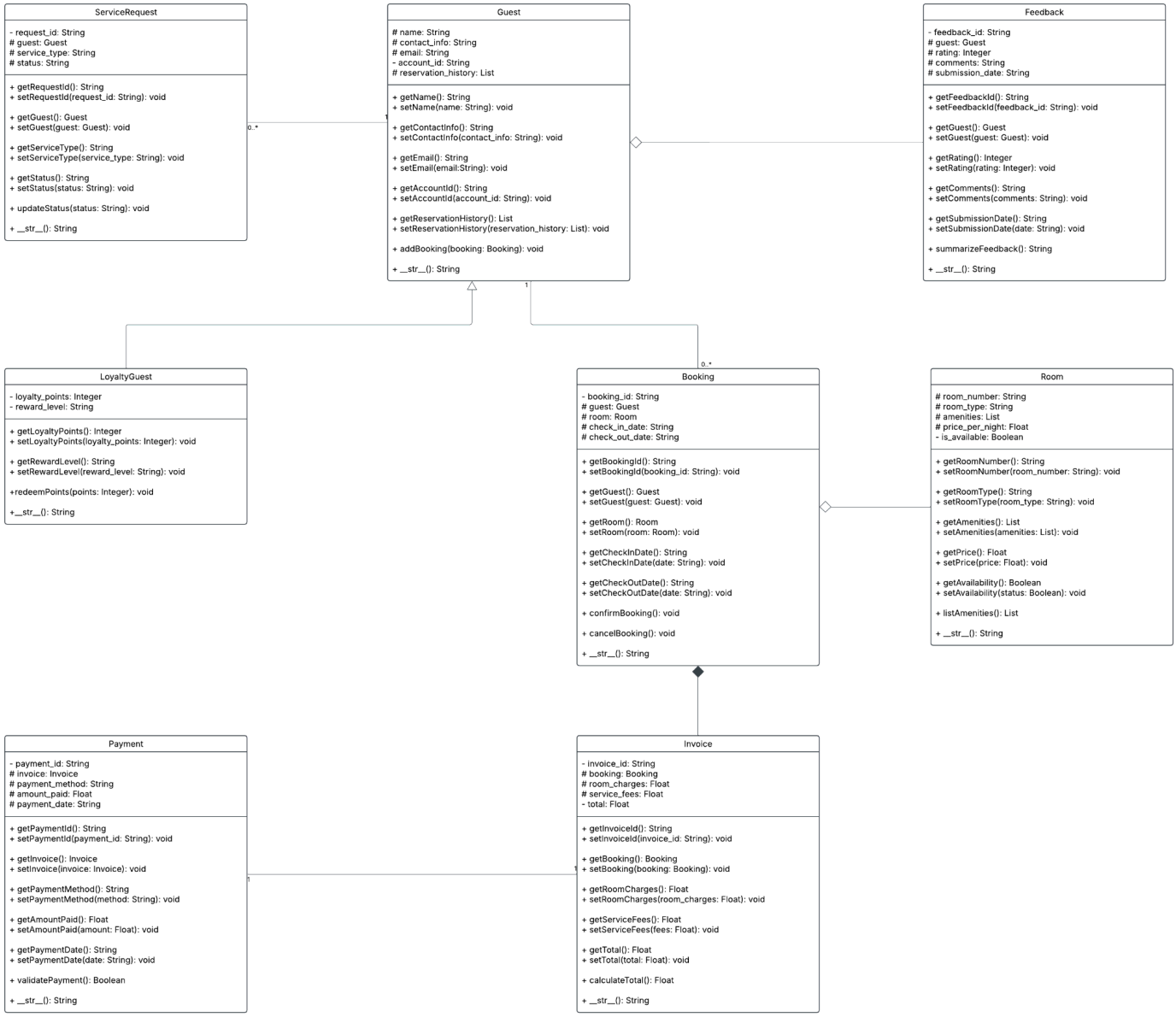
Ahmed Alremeithi - 202221113

ICS220 - Program. Fund.

Professor Afshan Parkar

March 28, 2025

UML Class Diagram:



Link:

https://lucid.app/lucidchart/c8fee318-9155-4c77-9b26-df7383760d47/edit?viewport_loc=-2366%2C-196%2C6369%2C3328%2C0_0&invitationId=inv_d4e7663e-ef7e-40a4-8b76-f177021666e7

Explaining UML Class Diagram:

1st Class	2nd Class	Association Type	Cardinality (if present)	Explanation of Relationship
LoyaltyGuest	Guest	Inheritance	None	LoyaltyGuest inherits from Guest. This extends the base functionality to include loyalty points and reward levels. This allows reuse of common guest attributes and methods with the added loyalty features.
Guest	Booking	Association (Binary)	1 (Guest): * (Booking	Each guest can make multiple bookings. The system links the guest's identity to their reservation history, allowing tracking of all past and current bookings.
Guest	Feedback	Aggregation	None	A guest is associated with one or mode feedback entries. The feedback exists independently and is not owned by the guest. If a guest is deleted, their feedback can still remain in the system.
Booking	Room	Aggregation	None	A booking is associated with exactly one room. The room exists independently and is not owned by the booking. If a booking is deleted, the room remains available in the system.
Booking	Invoice	Composition	None	Every booking generates exactly one invoice, which contains detailed billing for the guest's stay. The invoice cannot exist independently of its booking and is deleted if the booking is removed.
Payment	Invoice	Association (Binary)	1 (Payment): 1(Invoice)	A payment is made for a specific invoice. This association ensures payment data (method, date, amount) is connected to a particular transaction represented by the invoice.
Guest	ServiceRequest	Association (Binary)	1 (Guest): * (ServiceRequest)	A guest may submit multiple service requests (like housekeeping, room service). Each request is linked to one guest and allows staff to respond promptly through the system.

Classes.py:

```
# classes.py

class Guest:
    """Represents a hotel guest."""
    def __init__(self, name, contact_info, email, account_id):
        # Protected and private attributes
        self._name = name
        self._contact_info = contact_info
        self._email = email
        self.__account_id = account_id
        self._reservation_history = []  # List of Booking objects

    def getName(self):
        return self._name

    def setName(self, name):
        self._name = name

    def getContactInfo(self):
        return self._contact_info

    def setContactInfo(self, contact_info):
        self._contact_info = contact_info

    def getEmail(self):
        return self._email

    def setEmail(self, email):
        self._email = email

    def getAccountId(self):
        return self.__account_id

    def setAccountId(self, account_id):
        self.__account_id = account_id

    def getReservationHistory(self):
        return self._reservation_history
```

```
    def setReservationHistory(self, reservation_history):
        self._reservation_history = reservation_history

    def addBooking(self, booking):
        # Add a booking to the guest's reservation history
        self._reservation_history.append(booking)

    def __str__(self):
        return "Guest: " + self._name


class LoyaltyGuest(Guest):
    """Represents a loyal guest with reward points."""
    def __init__(self, name, contact_info, email, account_id,
loyalty_points, reward_level):
        super().__init__(name, contact_info, email, account_id)
        self.__loyalty_points = loyalty_points
        self.__reward_level = reward_level

    def getLoyaltyPoints(self):
        return self.__loyalty_points

    def setLoyaltyPoints(self, loyalty_points):
        self.__loyalty_points = loyalty_points

    def getRewardLevel(self):
        return self.__reward_level

    def setRewardLevel(self, reward_level):
        self.__reward_level = reward_level

    def redeemPoints(self, points):
        # Redeem loyalty points
        if self.__loyalty_points >= points:
            self.__loyalty_points -= points

    def __str__(self):
        return "Loyalty Guest: " + self.getName()
```

```
class Room:
    """Represents a hotel room."""
    def __init__(self, room_number, room_type, amenities, price_per_night,
is_available):
        self._room_number = room_number
        self._room_type = room_type
        self._amenities = amenities  # List
        self._price_per_night = price_per_night
        self.__is_available = is_available

    def getRoomNumber(self):
        return self._room_number

    def setRoomNumber(self, room_number):
        self._room_number = room_number

    def getRoomType(self):
        return self._room_type

    def setRoomType(self, room_type):
        self._room_type = room_type

    def getAmenities(self):
        return self._amenities

    def setAmenities(self, amenities):
        self._amenities = amenities

    def getPrice(self):
        return self._price_per_night

    def setPrice(self, price):
        self._price_per_night = price

    def getAvailability(self):
        return self.__is_available

    def setAvailability(self, status):
        self.__is_available = status
```

```
    def listAmenities(self):
        # Return list of amenities
        return self._amenities

    def __str__(self):
        return "Room: " + self._room_number


class Booking:
    """Represents a hotel booking."""
    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

    def getBookingId(self):
        return self.__booking_id

    def setBookingId(self, booking_id):
        self.__booking_id = booking_id

    def getGuest(self):
        return self._guest

    def setGuest(self, guest):
        self._guest = guest

    def getRoom(self):
        return self._room

    def setRoom(self, room):
        self._room = room

    def getCheckInDate(self):
        return self._check_in_date
```

```
    def setCheckInDate(self, date):
        self._check_in_date = date

    def getCheckOutDate(self):
        return self._check_out_date

    def setCheckOutDate(self, date):
        self._check_out_date = date

    def confirmBooking(self):
        return "Booking Confirmed"

    def cancelBooking(self):
        return "Booking Cancelled"

    def __str__(self):
        return "Booking ID: " + self.__booking_id


class Invoice:
    """Represents an invoice for a booking."""
    def __init__(self, invoice_id, booking, room_charges, service_fees,
total):
        self.__invoice_id = invoice_id
        self._booking = booking
        self._room_charges = room_charges
        self._service_fees = service_fees
        self.__total = total

    def getInvoiceId(self):
        return self.__invoice_id

    def setInvoiceId(self, invoice_id):
        self.__invoice_id = invoice_id

    def getBooking(self):
        return self._booking

    def setBooking(self, booking):
        self._booking = booking
```

```
    def getRoomCharges(self):
        return self._room_charges

    def setRoomCharges(self, room_charges):
        self._room_charges = room_charges

    def getServiceFees(self):
        return self._service_fees

    def setServiceFees(self, fees):
        self._service_fees = fees

    def getTotal(self):
        return self.__total

    def setTotal(self, total):
        self.__total = total

    def calculateTotal(self):
        self.__total = self._room_charges + self._service_fees
        return self.__total

    def __str__(self):
        return "Invoice ID: " + self.__invoice_id


class Payment:
    """Represents a payment made for an invoice."""
    def __init__(self, payment_id, invoice, payment_method, amount_paid,
payment_date):
        self.__payment_id = payment_id
        self._invoice = invoice
        self._payment_method = payment_method
        self._amount_paid = amount_paid
        self._payment_date = payment_date

    def getPaymentId(self):
        return self.__payment_id
```

```
    def setPaymentId(self, payment_id):
        self.__payment_id = payment_id

    def getInvoice(self):
        return self._invoice

    def setInvoice(self, invoice):
        self._invoice = invoice

    def getPaymentMethod(self):
        return self._payment_method

    def setPaymentMethod(self, method):
        self._payment_method = method

    def getAmountPaid(self):
        return self._amount_paid

    def setAmountPaid(self, amount):
        self._amount_paid = amount

    def getPaymentDate(self):
        return self._payment_date

    def setPaymentDate(self, date):
        self._payment_date = date

    def validatePayment(self):
        # Check if payment is sufficient
        return self._amount_paid >= self._invoice.getTotal()

    def __str__(self):
        return "Payment ID: " + self.__payment_id


class ServiceRequest:
    """Represents a service request made by a guest."""
    def __init__(self, request_id, guest, service_type, status):
        self.__request_id = request_id
        self._guest = guest
```

```
        self._service_type = service_type
        self._status = status

    def getRequestId(self):
        return self.__request_id

    def setRequestId(self, request_id):
        self.__request_id = request_id

    def getGuest(self):
        return self._guest

    def setGuest(self, guest):
        self._guest = guest

    def getServiceType(self):
        return self._service_type

    def setServiceType(self, service_type):
        self._service_type = service_type

    def getStatus(self):
        return self._status

    def setStatus(self, status):
        self._status = status

    def updateStatus(self, status):
        # Update request status
        self._status = status

    def __str__(self):
        return "Service Request ID: " + self.__request_id



class Feedback:
    """Represents feedback provided by a guest after their stay."""
    def __init__(self, feedback_id, guest, rating, comments,
submission_date):
```

```
        self.__feedback_id = feedback_id
        self._guest = guest
        self._rating = rating
        self._comments = comments
        self._submission_date = submission_date

    def getFeedbackId(self):
        return self.__feedback_id

    def setFeedbackId(self, feedback_id):
        self.__feedback_id = feedback_id

    def getGuest(self):
        return self._guest

    def setGuest(self, guest):
        self._guest = guest

    def getRating(self):
        return self._rating

    def setRating(self, rating):
        self._rating = rating

    def getComments(self):
        return self._comments

    def setComments(self, comments):
        self._comments = comments

    def getSubmissionDate(self):
        return self._submission_date

    def setSubmissionDate(self, date):
        self._submission_date = date

    def summarizeFeedback(self):
        # Return the feedback summary as a string
        return "Rating: " + str(self._rating) + ", Comment: " +
self._comments
```

```
def __str__(self):
    return "Feedback ID: " + self.__feedback_id
```

Testing.py:

```
# testing.py

from classes import Guest, LoyaltyGuest, Room, Booking, Invoice, Payment,
ServiceRequest, Feedback

# Guest Account Creation
print("--- Guest Account Creation ---")
guest1 = Guest("Ahmed Alremeithi", "0509999999", "ahmed@email.com",
"G001")
guest2 = LoyaltyGuest("Ghanem Alremeithi", "0508888888",
"ghanem@email.com", "G002", 150, "Gold")
print(guest1)
print(guest2)

# Searching for Available Rooms
print("\n--- Searching for Available Rooms ---")
room1 = Room("101", "Single", ["Wi-Fi", "TV"], 300, True)
room2 = Room("102", "Double", ["Wi-Fi", "Mini-Bar"], 500, False)
room3 = Room("103", "Suite", ["Wi-Fi", "TV", "Jacuzzi"], 800, True)

available_rooms = []
for room in [room1, room2, room3]:
   if room.getAvailability():
       available_rooms.append(room)

for room in available_rooms:
   print("Available Room:", room.getRoomNumber(), "Type:",
room.getRoomType(), "Amenities:", room.listAmenities())

# Making a Room Reservation
print("\n--- Making a Room Reservation ---")
booking1 = Booking("B001", guest1, room1, "2025-04-01", "2025-04-05")
booking2 = Booking("B002", guest2, room3, "2025-04-10", "2025-04-15")
guest1.addBooking(booking1)
guest2.addBooking(booking2)
print(booking1.confirmBooking())
print(booking2.confirmBooking())
```

```
# Booking Confirmation Notification (simulated with print)
print("\n--- Booking Confirmation Notification ---")
print("Confirmation sent to:", guest1.getEmail())
print("Confirmation sent to:", guest2.getEmail())

# Invoice Generation for a Booking
print("\n--- Invoice Generation ---")
invoice1 = Invoice("I001", booking1, room1.getPrice() * 4, 50, 0)
invoice2 = Invoice("I002", booking2, room3.getPrice() * 5, 100, 0)
print("Total for Invoice 1:", invoice1.calculateTotal())
print("Total for Invoice 2:", invoice2.calculateTotal())

# Processing Different Payment Methods
print("\n--- Payment Processing ---")
payment1 = Payment("P001", invoice1, "Credit Card", 1250, "2025-04-01")
payment2 = Payment("P002", invoice2, "Mobile Wallet", 4100, "2025-04-10")
print("Payment 1 valid:", payment1.validatePayment())
print("Payment 2 valid:", payment2.validatePayment())

# Displaying Reservation History
print("\n--- Reservation History ---")
for booking in guest1.getReservationHistory():
   print("Guest 1 Booking:", booking.getBookingId(), "Room:",
booking.getRoom().getRoomNumber())

for booking in guest2.getReservationHistory():
   print("Guest 2 Booking:", booking.getBookingId(), "Room:",
booking.getRoom().getRoomNumber())

# Cancellation of a Reservation
print("\n--- Cancel Reservation ---")
print("Before cancellation, Room 103 availability:",
room3.getAvailability())
room3.setAvailability(True)
print(booking2.cancelBooking())
print("After cancellation, Room 103 availability:",
room3.getAvailability())

# Service Request Test
print("\n--- Service Request Test ---")
```

```
request1 = ServiceRequest("SR001", guest1, "Housekeeping", "Pending")
request2 = ServiceRequest("SR002", guest2, "Room Service", "Pending")
print("Request 1 status:", request1.getStatus())
request1.updateStatus("Completed")
print("Request 1 new status:", request1.getStatus())
print("Request 2 status:", request2.getStatus())

# Feedback and Reviews
print("\n--- Feedback and Reviews ---")
feedback1 = Feedback("F001", guest1, 5, "Amazing service and clean room!",
"2025-04-06")
feedback2 = Feedback("F002", guest2, 4, "Comfortable stay, but room
service was slow.", "2025-04-16")
print("Feedback 1 Summary:", feedback1.summarizeFeedback())
print("Feedback 2 Summary:", feedback2.summarizeFeedback())
```

Output:

--- Guest Account Creation ---

Guest: Ahmed Alremeithi

Loyalty Guest: Ghanem Alremeithi

--- Searching for Available Rooms ---

Available Room: 101 Type: Single Amenities: ['Wi-Fi', 'TV']

Available Room: 103 Type: Suite Amenities: ['Wi-Fi', 'TV', 'Jacuzzi']

--- Making a Room Reservation ---

Booking Confirmed

Booking Confirmed

--- Booking Confirmation Notification ---

Confirmation sent to: ahmed@email.com

Confirmation sent to: ghanem@email.com

--- Invoice Generation ---

Total for Invoice 1: 1250

Total for Invoice 2: 4100

--- Payment Processing ---

Payment 1 valid: True

Payment 2 valid: True

--- Reservation History ---

Guest 1 Booking: B001 Room: 101

Guest 2 Booking: B002 Room: 103

--- Cancel Reservation ---

Before cancellation, Room 103 availability: True

Booking Cancelled

After cancellation, Room 103 availability: True

--- Service Request Test ---

Request 1 status: Pending

Request 1 new status: Completed

Request 2 status: Pending

--- Feedback and Reviews ---

Feedback 1 Summary: Rating: 5, Comment: Amazing service and clean room!

Feedback 2 Summary: Rating: 4, Comment: Comfortable stay, but room service was slow.

Github Repository Link: https://github.com/AhmedAlremeithi/Codingforzu122/tree/main

Summary of Learnings:

#LO1_OOAD:

In this project, I applied UML concepts by designing a class diagram that clearly shows the real-world entities in a hotel system like Guest, Room, Booking, Invoice, and more. I used correct relationships such as inheritance, composition, and aggregation, with proper attributes, access specifiers, and methods. The diagram helped me visualize how the system works and how each part is connected.

#LO2_OOProgramming:

I created well-structured Python code that matches the UML diagram. I used object-oriented programming principles like encapsulation, inheritance, and modularity. The code is organized into separate files and includes all classes with appropriate attributes, methods, and access control. I also wrote test cases to make sure all features work properly and the program runs without errors.

#LO4_SWDocumentation:

Throughout the project, I focused on writing clear and understandable code. Each class and method includes docstrings and comments that explain what it does. The naming is consistent and readable, and I kept the structure simple for anyone reviewing the code. This helped make the program easier to follow and maintain.