Royal Stay Hotel Management System

Nasser Nabeel Lootah

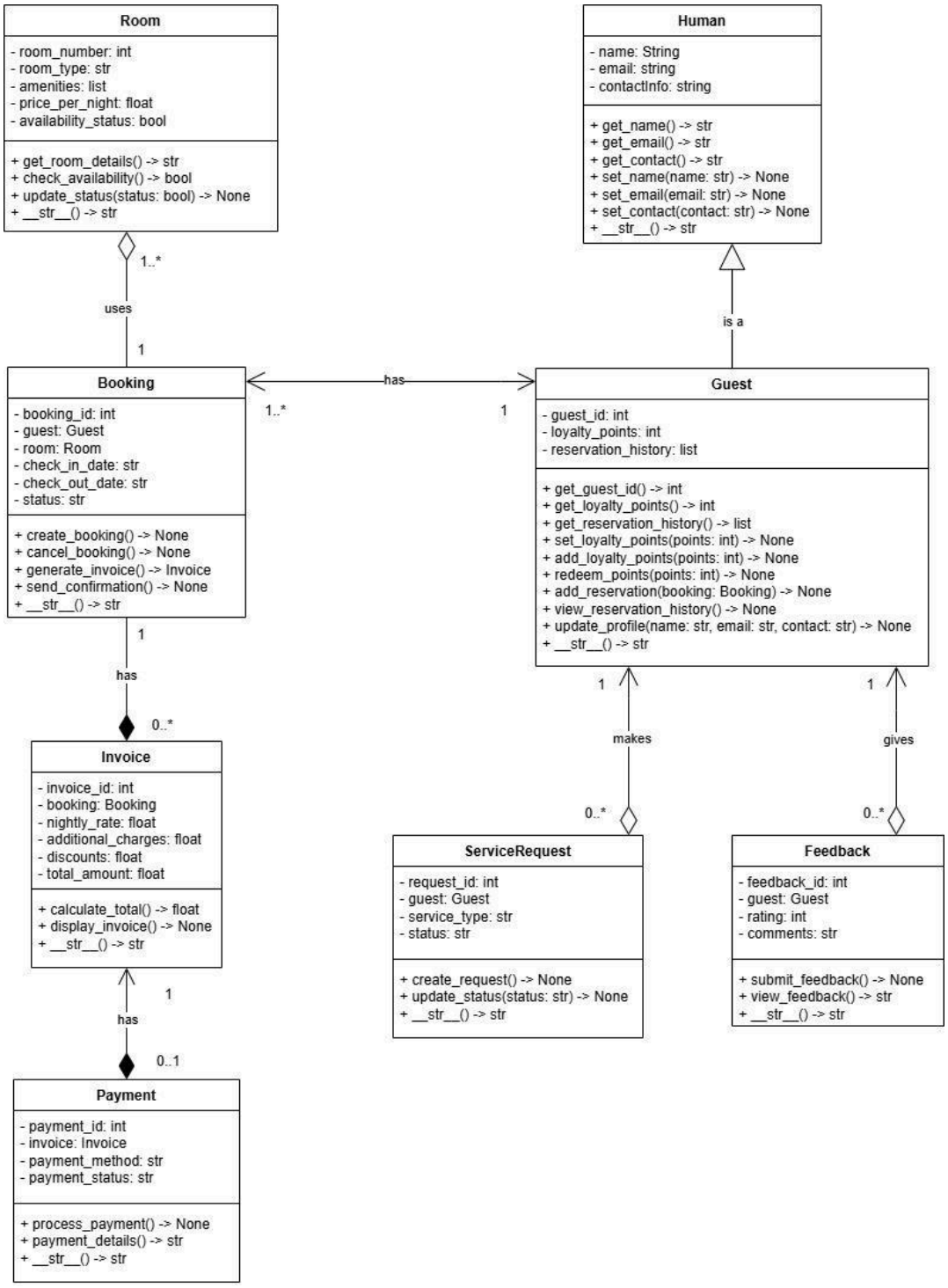
Zayed University

ICS220 > 21383 Program. Fund.

February 28, 2025

Professor Leonce

UML Class Diagram



UML Class Description

The UML class diagram of the Royal Stay Hotel Management System consists of 8 main classes and defines their attributes, methods, and relationships. The system is designed to efficiently manage room bookings, guest information, payments, service requests and feedback.

1. Human

The Human class is a base class that contains common personal details:

- **Attributes:** name, email, contactinfo
- **Methods:** Getters and setters for attributes, and a __str__() method.

2. Guest

The Guest class inherits from Human and represents the hotel guest.

- **Attributes:** guest_id, loyalty_points, reservation_history
- **Methods:** Profile update, loyalty points management, and reservation history management.
- **Relationship:**
    - ✔ **Inheritance:** Guest is a Human.
    - ✔ **Association:** Guest has 1..* Booking.
    - ✔ **Aggregation:** Guest makes 0..* ServiceRequest and gives 0..* Feedback.

3. Room

The Room class represents the hotel rooms.

- **Attributes:** room_number, room_type, amenities, price_per_night, availability_status
- **Methods:** Room details retrieval, availability check, and status update.

- **Relationship:**
    - **Aggregation:** Booking uses 1..* Rooms.

4. Booking

The Booking class manages guest bookings.

- **Attributes:** booking_id, guest, room, check_in_date, check_out_date, status
- **Methods:** Booking creation, cancellation, invoice generation, and confirmation notification.
- **Relationship:**
    - ✔ **Association:** Booking belongs to exactly 1 Guest.
    - ✔ **Aggregation:** Booking uses 1..* Room.
    - ✔ **Composition:** Booking has 0..* Invoice.

5. Invoice

The Invoice class handles the billing information for bookings.

- **Attributes:** invoice_id, booking, nightly_rate, additional_charges, discounts, total_amount
- **Methods:** Total calculation and display of invoice.
- **Relationship:**
    - ✔ **Composition:** Invoice belongs to exactly 1 Booking.
    - ✔ **Composition:** Invoice has 0..1 Payment.

6. Payment

The Payment class processes payments.

- **Attributes:** payment_id, invoice, payment_method, payment_status
- **Methods:** Payment processing and displaying payment details.
- **Relationship:**

    ✔ **Composition:** Payment belongs to exactly 1 Invoice.

7. ServiceRequest

The ServiceRequest class manages additional services requested by guests.

- **Attributes:** request_id, guest, service_type, status
- **Methods:** Request creation and status update.
- **Relationship:**

    ✔ **Aggregation:** ServiceRequest is made by exactly 1 Guest.

8. Feedback

The Feedback class manages guest feedback and ratings.

- **Attributes:** feedback_id, guest, rating, comments
- **Methods:** Feedback submission and viewing.
- **Relationship:**

    ✔ **Aggregation:** Feedback is given by exactly 1 Guest.

Aggregation Reason

Aggregation means weak relationships. The part can exist without the whole.

- Guest → ServiceRequest, Feedback:
  Requests and feedback can stay even if guest is deleted.

- Booking → Room:
  Room exists even if booking is canceled.

Composition Reason

Composition means strong relationship. The part cannot exist without the whole.

- Booking → Invoice:
  Invoice is part of booking. If booking is deleted, invoice is useless.
- Invoice → Payment:
  Payment belongs to invoice. No invoice → no payment.

Python Classes Codes

Human Class

```
class Human:
    """
    Human class - Base class for common human attributes
    """

    def __init__(self, name: str, email: str, contact: str):
        self.__name = name
        self.__email = email
        self.__contact = contact

    # Getter Methods
    def get_name(self) -> str:
        return self.__name

    def get_email(self) -> str:
        return self.__email
```

```
    def get_contact(self) -> str:
        return self.__contact

    # Setter Methods
    def set_name(self, name: str):
        self.__name = name

    def set_email(self, email: str):
        self.__email = email

    def set_contact(self, contact: str):
        self.__contact = contact

    def __str__(self) -> str:
        return f"Name: {self.__name}, Email: {self.__email}, Contact:
{self.__contact}"
```

Guest Class

```
from human import Human


class Guest(Human):
    """
    Guest class - Inherits from Human and adds guest-specific attributes
    """

    def __init__(self, guest_id: int, name: str, email: str, contact: str,
loyalty_points: int = 0):
        super().__init__(name, email, contact)
        self.__guest_id = guest_id
        self.__loyalty_points = loyalty_points
        self.__reservation_history = []  # List to store past bookings
```

```
# Getter Methods
def get_guest_id(self) -> int:
    return self.__guest_id

def get_loyalty_points(self) -> int:
    return self.__loyalty_points

def get_reservation_history(self) -> list:
    return self.__reservation_history

# Setter Methods
def set_loyalty_points(self, points: int):
    self.__loyalty_points = points

# Functional Methods
def add_loyalty_points(self, points: int):
    self.__loyalty_points += points

def redeem_points(self, points: int):
    if points <= self.__loyalty_points:
        self.__loyalty_points -= points
    else:
        print("Not enough loyalty points to redeem.")

def add_reservation(self, booking):
    self.__reservation_history.append(booking)

def view_reservation_history(self):
    if not self.__reservation_history:
        print("No reservations found.")
    else:
        for res in self.__reservation_history:
```

```
                print(res)


    def update_profile(self, name: str, email: str, contact: str):
        self.set_name(name)
        self.set_email(email)
        self.set_contact(contact)


    def __str__(self) -> str:
        return f"{super().__str__()}, Guest ID: {self.__guest_id}, Loyalty Points:
{self.__loyalty_points}"
```

Room class

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


    def __init__(self, room_number: int, room_type: str, amenities: list,
price_per_night: float, availability_status: bool = True):
        self.__room_number = room_number
        self.__room_type = room_type
        self.__amenities = amenities
        self.__price_per_night = price_per_night
        self.__availability_status = availability_status


    # Getter methods
    def get_room_number(self) -> int:
        return self.__room_number


    def get_room_type(self) -> str:
```

```
        return self.__room_type


    def get_amenities(self) -> list:
        return self.__amenities


    def get_price_per_night(self) -> float:
        return self.__price_per_night


    def is_available(self) -> bool:
        return self.__availability_status


    # Setter methods
    def update_status(self, status: bool) -> None:
        self.__availability_status = status


    # Functional method
    def get_room_details(self) -> str:
        return f"Room {self.__room_number}: {self.__room_type}, Amenities: {',
'.join(self.__amenities)}, Price: {self.__price_per_night}, Available:
{self.__availability_status}"


    def check_availability(self) -> bool:
        return self.__availability_status


    def __str__(self) -> str:
        return f"Room {self.__room_number} ({self.__room_type}) - AED
{self.__price_per_night} - Available: {self.__availability_status}"
```

Booking class

```
from room import Room
from guest import Guest


class Booking:
```

```
    """
    Booking class - Manages room bookings
    """

    def __init__(self, booking_id: int, guest: Guest, room: Room, check_in_date: str,
check_out_date: str, status: str = "Confirmed"):
        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.__status = status
        self.__invoice = None  # Will be set later

    # Getter methods
    def get_booking_id(self) -> int:
        return self.__booking_id

    def get_guest(self) -> Guest:
        return self.__guest

    def get_room(self) -> Room:
        return self.__room

    def get_status(self) -> str:
        return self.__status

    # Functional methods
    def create_booking(self) -> None:
        if self.__room.is_available():
            self.__room.update_status(False)
            print(f"Booking {self.__booking_id} created for Guest
{self.__guest.get_guest_id()}")
        else:
```

```
            print("Room not available!")


    def cancel_booking(self) -> None:
        if self.__status != "Cancelled":
            self.__status = "Cancelled"
            self.__room.update_status(True)
            print(f"Booking {self.__booking_id} has been cancelled.")
        else:
            print("Booking already cancelled.")


    def generate_invoice(self) -> "Invoice":
        from invoice import Invoice
        self.__invoice = Invoice(self, self.__room.get_price_per_night(), 0.0, 0.0)
        print(f"Invoice generated for Booking {self.__booking_id}")
        return self.__invoice


    def send_confirmation(self) -> None:
        print(f"Confirmation sent for Booking {self.__booking_id} to Guest
{self.__guest.get_guest_id()}")


    def __str__(self) -> str:
        return f"Booking ID: {self.__booking_id}, Guest:
{self.__guest.get_guest_id()}, Room: {self.__room.get_room_number()}, Status:
{self.__status}"
```

Invoice class

```
 from booking import Booking


class Invoice:
    """
    Invoice class - Handles billing information for a booking
```

```
    """

    def __init__(self, booking: Booking, nightly_rate: float, additional_charges:
float, discounts: float):
        self.__invoice_id = f"INV-{booking.get_booking_id()}"
        self.__booking = booking
        self.__nightly_rate = nightly_rate
        self.__additional_charges = additional_charges
        self.__discounts = discounts
        self.__total_amount = 0.0

    # Getter methods
    def get_invoice_id(self) -> str:
        return self.__invoice_id

    def get_total_amount(self) -> float:
        return self.__total_amount

    # Functional methods
    def calculate_total(self) -> float:
        self.__total_amount = (self.__nightly_rate + self.__additional_charges) -
self.__discounts
        return self.__total_amount

    def display_invoice(self) -> None:
        print(f"Invoice ID: {self.__invoice_id}")
        print(f"Booking ID: {self.__booking.get_booking_id()}")
        print(f"Nightly Rate: AED {self.__nightly_rate}")
        print(f"Additional Charges: AED {self.__additional_charges}")
        print(f"Discounts: AED {self.__discounts}")
        print(f"Total Amount: AED {self.__total_amount}")

    def __str__(self) -> str:
        return f"Invoice {self.__invoice_id} - Total: AED {self.__total_amount}"
```

Payment Class

```
from invoice import Invoice


class Payment:
    """
    Payment class - Processes payment for an invoice
    """

    def __init__(self, payment_id: int, invoice: Invoice, payment_method: str):
        self.__payment_id = payment_id
        self.__invoice = invoice
        self.__payment_method = payment_method
        self.__payment_status = "Pending"

    # Getter methods
    def get_payment_id(self) -> int:
        return self.__payment_id

    def get_payment_status(self) -> str:
        return self.__payment_status

    # Functional methods
    def process_payment(self) -> None:
        total = self.__invoice.calculate_total()
        if total > 0:
            self.__payment_status = "Completed"
            print(f"Payment of AED {total} completed using {self.__payment_method}.")
        else:
            print("Invalid total amount. Payment failed.")
```

```
    def payment_details(self) -> str:
        return f"Payment ID: {self.__payment_id}, Method: {self.__payment_method},
Status: {self.__payment_status}"

    def __str__(self) -> str:
        return f"Payment {self.__payment_id} - Status: {self.__payment_status}"
```

Service_request class

```
from guest import Guest

class ServiceRequest:
    """
    ServiceRequest class - Handles additional services requested by guests
    """

    def __init__(self, request_id: int, guest: Guest, service_type: str, status: str =
"Pending"):
        self.__request_id = request_id
        self.__guest = guest
        self.__service_type = service_type
        self.__status = status

    # Getter methods
    def get_request_id(self) -> int:
        return self.__request_id

    def get_status(self) -> str:
        return self.__status

    # Functional methods
    def create_request(self) -> None:
```

```
        print(f"Service request '{self.__service_type}' created for Guest ID
{self.__guest.get_guest_id()}.")


    def update_status(self, status: str) -> None:
        self.__status = status
        print(f"Service request {self.__request_id} status updated to {status}.")


    def __str__(self) -> str:
        return f"Request ID: {self.__request_id}, Guest:
{self.__guest.get_guest_id()}, Service: {self.__service_type}, Status:
{self.__status}"
```

Feedback class

```
from guest import Guest


class Feedback:
    """
    Feedback class - Handles guest feedback and ratings
    """


    def __init__(self, feedback_id: int, guest: Guest, rating: int, comments: str):
        self.__feedback_id = feedback_id
        self.__guest = guest
        self.__rating = rating
        self.__comments = comments


    # Functional methods
    def submit_feedback(self) -> None:
        print(f"Feedback submitted by Guest {self.__guest.get_guest_id()} with rating
{self.__rating}/5.")


    def view_feedback(self) -> str:
```

```
        return f"Rating: {self.__rating}/5, Comments: {self.__comments}"


    def __str__(self) -> str:
        return f"Feedback ID: {self.__feedback_id}, Guest:
{self.__guest.get_guest_id()}, Rating: {self.__rating}, Comments: {self.__comments}"
```

Test_casses class

```
# Import required classes
from guest import Guest
from room import Room
from booking import Booking
from invoice import Invoice
from payment import Payment
from service_request import ServiceRequest
from feedback import Feedback


# ------------------------------
# Class: Tester
# Handles all hotel system operations
# ------------------------------
class Tester:
    def __init__(self):
        # Lists to store all objects
        self.guests = []
        self.rooms = []
        self.bookings = []
        self.service_requests = []
        self.feedbacks = []
        self.payments = []


        # Add dummy UAE-based Guests
```

```
        guest1 = Guest(1, "Ahmed Al Mansoori", "ahmed.mansoori@gmail.com",
"+971501234567", 100)
        guest2 = Guest(2, "Fatima Al Nuaimi", "fatima.nuaimi@yahoo.com",
"+971552223344", 250)
        self.guests.append(guest1)
        self.guests.append(guest2)

        # Add dummy Rooms
        room1 = Room(1001, "Single", ["WiFi", "TV", "Mini-bar"], 350.0)
        room2 = Room(1002, "Double", ["WiFi", "TV", "Mini-bar", "Balcony"], 550.0)
        room3 = Room(2001, "Suite", ["WiFi", "TV", "Mini-bar", "Sea View", "Jacuzzi"],
950.0)
        self.rooms.append(room1)
        self.rooms.append(room2)
        self.rooms.append(room3)

        print("\n--- Dummy Data Loaded ---")

    # Create a new Guest Account
    def create_guest(self):
        try:
            guest_id = int(input("Enter Guest ID: "))
            name = input("Enter Name: ")
            email = input("Enter Email: ")
            contact = input("Enter Contact: ")
            points = int(input("Enter Loyalty Points: "))
            guest = Guest(guest_id, name, email, contact, points)
            self.guests.append(guest)
            print("Guest account created successfully.")
            print(guest)
        except Exception as e:
            print("Error:", e)

    # Add a new room
```

```
def add_room(self):
    try:
        room_no = int(input("Enter Room Number: "))
        room_type = input("Enter Room Type: ")
        amenities = input("Enter Amenities (comma separated): ").split(",")
        price = float(input("Enter Price per Night: "))
        room = Room(room_no, room_type, amenities, price)
        self.rooms.append(room)
        print("Room added successfully.")
        print(room)
    except Exception as e:
        print("Error:", e)

# Display all available rooms
def search_rooms(self):
    print("\nAvailable Rooms:")
    for r in self.rooms:
        if r.is_available():
            print(r)

# Make a room reservation
def make_reservation(self):
    try:
        booking_id = int(input("Enter Booking ID: "))
        guest_id = int(input("Enter Guest ID: "))

        # Search for Guest
        guest = None
        for g in self.guests:
            if g.get_guest_id() == guest_id:
                guest = g
        if not guest:
            print("Guest not found!")
```

```
            return

        # Search for Room
        room_no = int(input("Enter Room Number: "))
        room = None
        for r in self.rooms:
            if r.get_room_number() == room_no:
                room = r
        if not room:
            print("Room not found!")
            return
        if not room.is_available():
            print("Room not available!")
            return

        # Create booking
        check_in = input("Enter Check-in Date: ")
        check_out = input("Enter Check-out Date: ")
        booking = Booking(booking_id, guest, room, check_in, check_out)
        booking.create_booking()
        self.bookings.append(booking)
        guest.add_reservation(booking)
        print("Booking created successfully.")
        print(booking)
    except Exception as e:
        print("Error:", e)

# Cancel an existing reservation
def cancel_reservation(self):
    try:
        booking_id = int(input("Enter Booking ID to cancel: "))
        booking = None
        for b in self.bookings:
```

```
            if b.get_booking_id() == booking_id:
                booking = b
        if not booking:
            print("Booking not found!")
            return
        booking.cancel_booking()
    except Exception as e:
        print("Error:", e)

# Generate invoice for a booking
def generate_invoice(self):
    try:
        booking_id = int(input("Enter Booking ID to generate invoice: "))
        booking = None
        for b in self.bookings:
            if b.get_booking_id() == booking_id:
                booking = b
        if not booking:
            print("Booking not found!")
            return
        invoice = booking.generate_invoice()
        invoice.calculate_total()
        invoice.display_invoice()
    except Exception as e:
        print("Error:", e)

# Process payment for an invoice
def process_payment(self):
    try:
        booking_id = int(input("Enter Booking ID to process payment: "))
        booking = None
        for b in self.bookings:
            if b.get_booking_id() == booking_id:
```

```
                booking = b
        if not booking:
            print("Booking not found!")
            return
        if not booking._Booking__invoice:
            print("No invoice found. Generate invoice first.")
            return
        payment_id = int(input("Enter Payment ID: "))
        method = input("Enter Payment Method (Card/Wallet): ")
        payment = Payment(payment_id, booking._Booking__invoice, method)
        payment.process_payment()
        self.payments.append(payment)
    except Exception as e:
        print("Error:", e)

# Make a service request
def make_service_request(self):
    try:
        request_id = int(input("Enter Request ID: "))
        guest_id = int(input("Enter Guest ID: "))
        guest = None
        for g in self.guests:
            if g.get_guest_id() == guest_id:
                guest = g
        if not guest:
            print("Guest not found!")
            return
        service_type = input("Enter Service Type: ")
        request = ServiceRequest(request_id, guest, service_type)
        request.create_request()
        self.service_requests.append(request)
    except Exception as e:
        print("Error:", e)
```

```
# Submit guest feedback
def submit_feedback(self):
    try:
        feedback_id = int(input("Enter Feedback ID: "))
        guest_id = int(input("Enter Guest ID: "))
        guest = None
        for g in self.guests:
            if g.get_guest_id() == guest_id:
                guest = g
        if not guest:
            print("Guest not found!")
            return
        rating = int(input("Enter Rating (1-5): "))
        comments = input("Enter Comments: ")
        fb = Feedback(feedback_id, guest, rating, comments)
        fb.submit_feedback()
        self.feedbacks.append(fb)
    except Exception as e:
        print("Error:", e)

# View all past bookings of a guest
def view_reservation_history(self):
    try:
        guest_id = int(input("Enter Guest ID: "))
        guest = None
        for g in self.guests:
            if g.get_guest_id() == guest_id:
                guest = g
        if not guest:
            print("Guest not found!")
            return
        print(f"Reservation history for Guest {guest_id}:")
```

```
            guest.view_reservation_history()
        except Exception as e:
            print("Error:", e)


# ------------------------------
# Class: MainMenu
# Displays options and links to Tester functions
# ------------------------------
class MainMenu:
    def __init__(self):
        self.tester = Tester()

    # Display the main menu
    def display_menu(self):
        while True:
            print("\n--- Royal Stay Hotel Management System ---")
            print("1. Create Guest Account")
            print("2. Add Room")
            print("3. Search Available Rooms")
            print("4. Make Room Reservation")
            print("5. Cancel Reservation")
            print("6. Generate Invoice")
            print("7. Process Payment")
            print("8. Make Service Request")
            print("9. Submit Feedback")
            print("10. View Reservation History")
            print("11. Exit")

            try:
                choice = int(input("Enter your choice: "))
            except ValueError:
                print("Invalid input! Please enter a number.")
```

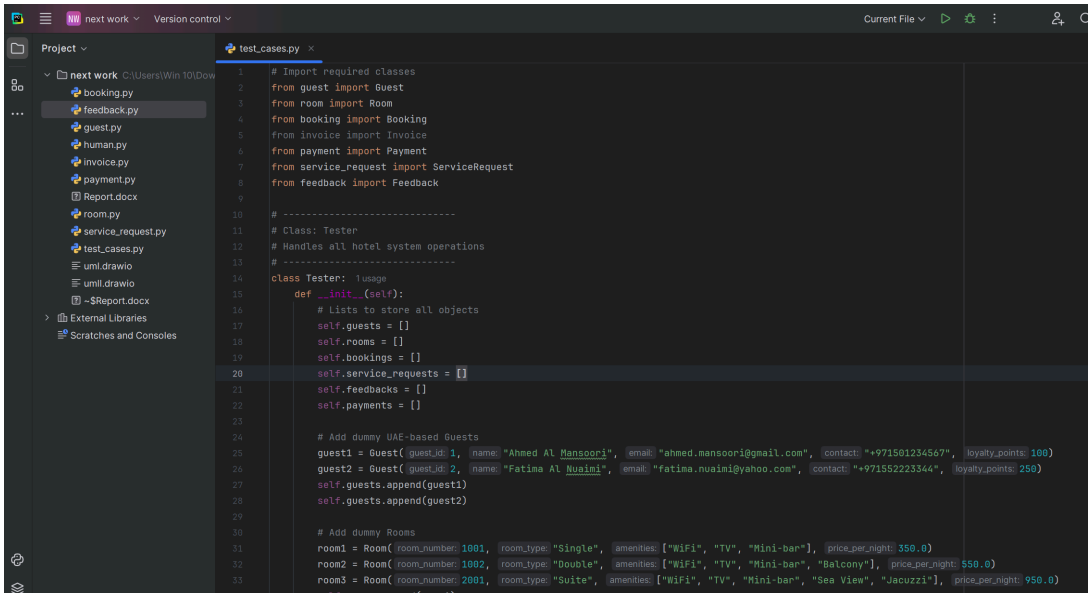
```
                continue

            # Menu options
            if choice == 1:
                self.tester.create_guest()
            elif choice == 2:
                self.tester.add_room()
            elif choice == 3:
                self.tester.search_rooms()
            elif choice == 4:
                self.tester.make_reservation()
            elif choice == 5:
                self.tester.cancel_reservation()
            elif choice == 6:
                self.tester.generate_invoice()
            elif choice == 7:
                self.tester.process_payment()
            elif choice == 8:
                self.tester.make_service_request()
            elif choice == 9:
                self.tester.submit_feedback()
            elif choice == 10:
                self.tester.view_reservation_history()
            elif choice == 11:
                print("Thank you for using Royal Stay Hotel System!")
                break
            else:
                print("Invalid choice. Please try again.")


# ------------------------------
# Program Starting Point
# ------------------------------
```

```
if __name__ == "__main__":
    menu = MainMenu()
    menu.display_menu()
```

Screenshots for Testing



Before testing we added some dummy data like rooms and some guests and other data we can add while testing using input.

```
--- Dummy Data Loaded ---

--- Royal Stay Hotel Management System ---
1. Create Guest Account
2. Add Room
3. Search Available Rooms
4. Make Room Reservation
5. Cancel Reservation
6. Generate Invoice
7. Process Payment
8. Make Service Request
9. Submit Feedback
10. View Reservation History
11. Exit
Enter your choice: 1
Enter Guest ID: 909
Enter Name: Dana
Enter Email: dana@rmail.com
Enter Contact: 051234566789
Enter Loyalty Points: 100
Guest account created successfully.
Name: Dana, Email: dana@rmail.com, Contact: 051234566789, Guest ID: 909, Loyalty Points: 100
```

Other than dummy guests we added 1 more guest.

```
--- Royal Stay Hotel Management System ---
1. Create Guest Account
2. Add Room
3. Search Available Rooms
4. Make Room Reservation
5. Cancel Reservation
6. Generate Invoice
7. Process Payment
8. Make Service Request
9. Submit Feedback
10. View Reservation History
11. Exit
Enter your choice: 2
Enter Room Number: 111
Enter Room Type: VIP
Enter Amenities (comma separated): Wifi, SPA
Enter Price per Night: 450
Room added successfully.
Room 111 (VIP) - AED 450.0 - Available: True
```

1 room is also added other than the dummy rooms.

```
Enter your choice: 3

Available Rooms:
Room 1001 (Single) - AED 350.0 - Available: True
Room 1002 (Double) - AED 550.0 - Available: True
Room 2001 (Suite) - AED 950.0 - Available: True
Room 111 (VIP) - AED 450.0 - Available: True
```

We can see all available rooms by pressing 3 from the choice.

```
--- Royal Stay Hotel Management System ---
1. Create Guest Account
2. Add Room
3. Search Available Rooms
4. Make Room Reservation
5. Cancel Reservation
6. Generate Invoice
7. Process Payment
8. Make Service Request
9. Submit Feedback
10. View Reservation History
11. Exit
Enter your choice: 4
Enter Booking ID: 1
Enter Guest ID: 9
Guest not found!
```

```
Enter your choice: 4
Enter Booking ID: 1
Enter Guest ID: 1
Enter Room Number: 111111
Room not found!
```

While room booking if you enter wrong guest id or wring wrong room id which is not there it will tell you.

```
--- Royal Stay Hotel Management System ---
1. Create Guest Account
2. Add Room
3. Search Available Rooms
4. Make Room Reservation
5. Cancel Reservation
6. Generate Invoice
7. Process Payment
8. Make Service Request
9. Submit Feedback
10. View Reservation History
11. Exit
Enter your choice: 4
Enter Booking ID: 1
Enter Guest ID: 1
Enter Room Number: 1001
Enter Check-in Date: 21 April 2023
Enter Check-out Date: 22 April 2023
Booking 1 created for Guest 1
Booking created successfully.
Booking ID: 1, Guest: 1, Room: 1001, Status: Confirmed
```

By entering correct details room is successfully booked.

```
Enter your choice: 3

Available Rooms:
Room 1002 (Double) - AED 550.0 - Available: True
Room 2001 (Suite) - AED 950.0 - Available: True
Room 111 (VIP) - AED 450.0 - Available: True
```

After booking room 1001 we can see its not available in list because its booked.

```
Enter your choice: 6
Enter Booking ID to generate invoice: 11111
Booking not found!
```

For generate receipt if you enter wrong booking id it will gives you error.

```
Enter your choice: 6
Enter Booking ID to generate invoice: 1
Invoice generated for Booking 1
Invoice ID: INV-1
Booking ID: 1
Nightly Rate: AED 350.0
Additional Charges: AED 0.0
Discounts: AED 0.0
Total Amount: AED 350.0
```

By entering correct booking id invoice is generated.

```
Enter your choice: 7
Enter Booking ID to process payment: 12
Booking not found!
```

Process payment option if you write wrong booking id it will give you error message.

```
Enter your choice: 7
Enter Booking ID to process payment: 1
Enter Payment ID: 77
Enter Payment Method (Card/Wallet): Card
Payment of AED 350.0 completed using Card.
```

With correct details it will show correctly process the payment.

```
Enter your choice: 8
Enter Request ID: 31
Enter Guest ID: 1
Enter Service Type: SPA
Service request 'SPA' created for Guest ID 1.
```

We can made a service request by entering details.

```
Enter your choice: 9
Enter Feedback ID: 99
Enter Guest ID: 1
Enter Rating (1-5): 5
Enter Comments: nice rooms
Feedback submitted by Guest 1 with rating 5/5.
```

You can gives feedback by entering correct guest id.

```
Enter your choice: 10
Enter Guest ID: 1
Reservation history for Guest 1:
Booking ID: 1, Guest: 1, Room: 1001, Status: Confirmed
```

View reservation history by pressing correct guest id.

```
Enter your choice: 5
Enter Booking ID to cancel: 2
Booking not found!
```

```
Enter your choice: 5
Enter Booking ID to cancel: 1
Booking 1 has been cancelled.
```

Cancel a reservation by entering correct booking id.

Summary of Learning

In this project, I learned how to use Object-Oriented Programming to build a real hotel management system. I understood how to create different classes like Guest, Room, Booking, Invoice, and more. I also learned how to connect these classes using relationships like Inheritance, Aggregation, and Composition. I practiced writing clean code with private attributes, getter and setter methods, and comments to make the program easy to understand.

I also learned how to test the program using a menu-based system. I added dummy data to make testing easy. I handled errors and user inputs so the program doesn't crash. This project helped me understand how real hotel systems work and how to organize code properly.