Assignment 2

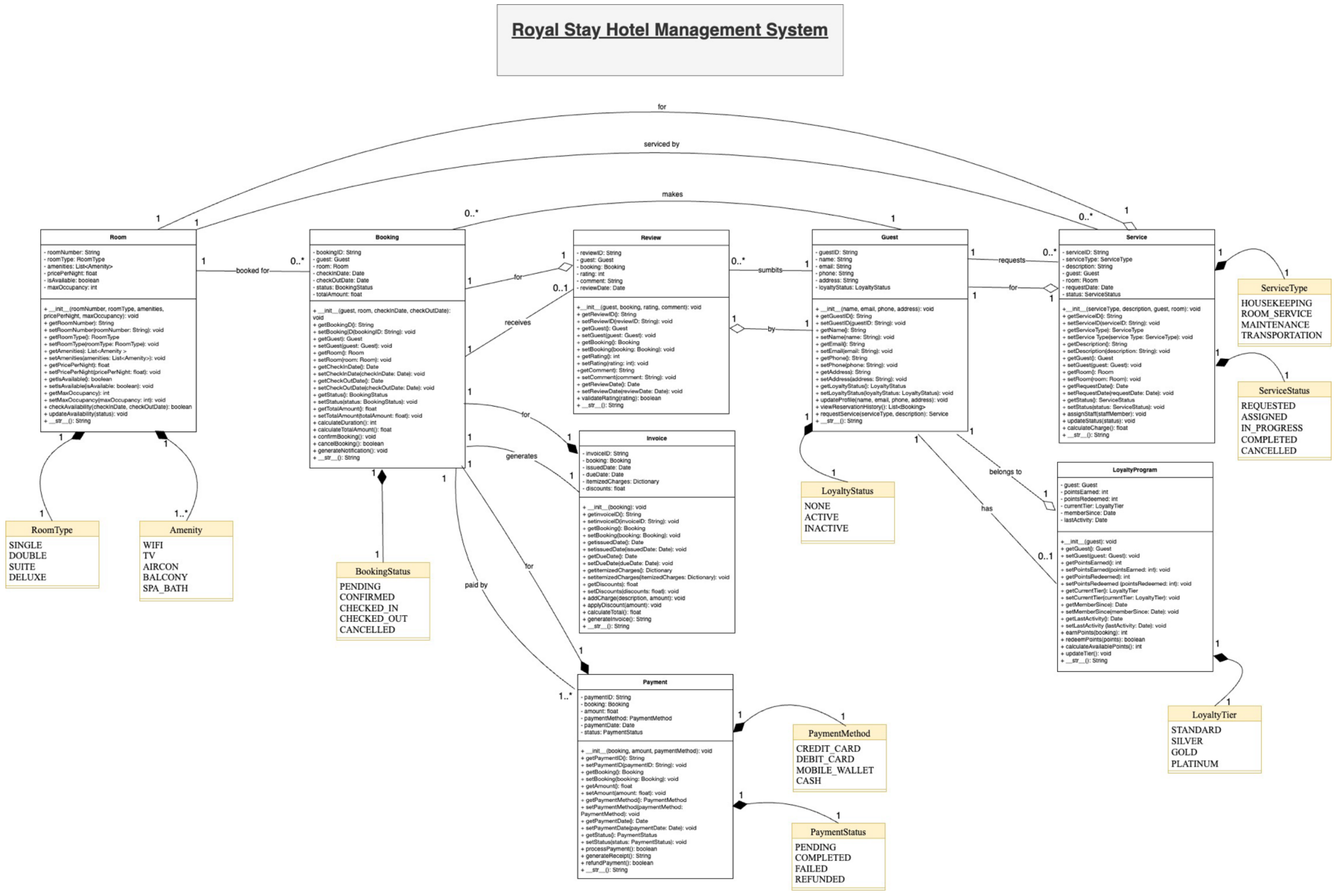
ICS220 > 23018 Program. Fund

Salma Almansoori

202317014

28 March 2025

A. Design UML Class Diagram



My UML class diagram shows how the Royal Stay Hotel Management System works. The diagram includes 8 main classes that handle different parts of the hotel system. I've connected these classes to show how they work together.

Main Classes:

1. Guest Class, this class stores information about hotel guests including:

- Name, email, phone, and address

- Loyalty status (None, Active, or Inactive)
- Guests can make bookings, request services, and leave reviews after their stay

2. Room Class, this class keeps track of all rooms in the hotel with details like:

- Room number and type (Single, Double, Suite, or Deluxe)
- Room features (WiFi, TV, Air Conditioning, etc.)
- Price per night
- Availability status
- Rooms can be booked by guests and can receive different services.

3. Booking Class, this handles the reservation process:

- Links a guest to a specific room
- Tracks check-in and check-out dates
- Records booking status (Pending, Confirmed, etc.)
- Calculates total cost
- Each booking can generate an invoice, receive payments, and get a review.

4. Payment Class, this manages how guests pay for their stay:

- Records payment amount and method
- Tracks payment status
- Processes payments and refunds

5. Invoice Class, this creates detailed bills for guests:

- Lists all charges for the stay
- Applies any discounts
- Calculates the final amount due

6. LoyaltyProgram Class, this tracks benefits for returning guests:

- Records points earned and used
- Manages loyalty tier (Standard, Silver, Gold, Platinum)
- Updates member status based on activity

7. Service Class, this handles extra services guests can request:

- Room service, housekeeping, maintenance, etc.
- Tracks service status (Requested, Assigned, etc.)
- Calculates any extra charges

8. Review Class, this collects guest feedback:

- Stores ratings and comments
- Links reviews to specific bookings and guests

Relationships Between Classes

I've used different types of connections to show how these classes relate to each other:

- Association: Simple connections like (Guest makes Booking) or (Guest requests Service)
- Aggregation: Relationships where one class uses another but both can exist separately
- Composition: Strong relationships where one class is part of another

I've also shown how many of each thing can be connected using numbers:

- 1: Exactly one
- 0..*: Zero or many
- 1..*: One or many
- 0..1: Zero or one

Enum Classes

I've included special classes (Enums) to handle attributes with fixed options:

- RoomType (Single, Double, Suite, Deluxe)
- Amenity (WiFi, TV, Air Conditioning, etc.)
- BookingStatus (Pending, Confirmed, etc.)
- PaymentMethod (Credit Card, Debit Card, etc.)
- PaymentStatus (Pending, Completed, etc.)
- LoyaltyTier (Standard, Silver, Gold, Platinum)
- ServiceType (Housekeeping, Room Service, etc.)
- ServiceStatus (Requested, Assigned, etc.)
- LoyaltyStatus (None, Active, Inactive)

Design Decisions

My design focuses on making sure all hotel functions work together, keeping related information in the right classes, using proper connections to show how parts of the system interact, and following object-oriented design principles.

B. Write Python Code to Implement Your UML Class Diagram

I have implemented all the classes from my UML diagram in Python.

```
# Royal Stay Hotel Management System

from enum import Enum
from datetime import datetime, timedelta
import uuid


# ENUMS
```

```python
class RoomType(Enum):
    """Defines the types of rooms available in the hotel."""
    SINGLE = "Single"
    DOUBLE = "Double"
    SUITE = "Suite"
    DELUXE = "Deluxe"

class Amenity(Enum):
    """Defines the amenities available in the hotel rooms."""
    WIFI = "WiFi"
    TV = "TV"
    AIRCON = "Air Conditioning"
    BALCONY = "Balcony"
    SPA_BATH = "Spa Bath"

class BookingStatus(Enum):
    """Defines the possible statuses of a booking."""
    PENDING = "Pending"
    CONFIRMED = "Confirmed"
    CHECKED_IN = "Checked In"
    CHECKED_OUT = "Checked Out"
    CANCELLED = "Cancelled"

class PaymentMethod(Enum):
    """Defines the payment methods accepted by the hotel."""
    CREDIT_CARD = "Credit Card"
    DEBIT_CARD = "Debit Card"
    MOBILE_WALLET = "Mobile Wallet"
    CASH = "Cash"

class PaymentStatus(Enum):
    """Defines the possible statuses of a payment."""
    PENDING = "Pending"
    COMPLETED = "Completed"
    FAILED = "Failed"
    REFUNDED = "Refunded"

class LoyaltyTier(Enum):
    """Defines the loyalty tiers for the hotel's loyalty program."""
    STANDARD = "Standard"
    SILVER = "Silver"
    GOLD = "Gold"
    PLATINUM = "Platinum"

class ServiceType(Enum):
    """Defines the types of services offered by the hotel."""
    HOUSEKEEPING = "Housekeeping"
    ROOM_SERVICE = "Room Service"
    MAINTENANCE = "Maintenance"
    TRANSPORTATION = "Transportation"

class ServiceStatus(Enum):
    """Defines the possible statuses of a service request."""
    REQUESTED = "Requested"
    ASSIGNED = "Assigned"
    IN_PROGRESS = "In Progress"
    COMPLETED = "Completed"
    CANCELLED = "Cancelled"

class LoyaltyStatus(Enum):
    """Defines the status of a guest's loyalty membership."""
    NONE = "None"
    ACTIVE = "Active"
    INACTIVE = "Inactive"

# GUEST CLASS
class Guest:
    """
    Represents a guest at the Royal Stay Hotel.

    This class stores information about hotel guests including personal
details
    and loyalty status.
    """

    def __init__(self, name, email, phone, address):
        """
        Initialize a new Guest object.

        Args:
            name (str): The guest's full name
            email (str): The guest's email address
            phone (str): The guest's phone number
            address (str): The guest's physical address
        """
        self._guest_id = None  # Will be assigned later
        self._name = name
        self._email = email
        self._phone = phone
        self._address = address
        self._loyalty_status = LoyaltyStatus.NONE

    # Getter and setter methods
    def get_guest_id(self):
        return self._guest_id

    def set_guest_id(self, guest_id):
        self._guest_id = guest_id

    def get_name(self):
        return self._name

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

    def get_email(self):
        return self._email

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

    def get_phone(self):
        return self._phone

    def set_phone(self, phone):
        self._phone = phone

    def get_address(self):
        return self._address

    def set_address(self, address):
        self._address = address

    def get_loyalty_status(self):
        return self._loyalty_status

    def set_loyalty_status(self, loyalty_status):
        self._loyalty_status = loyalty_status

    def update_profile(self, name, email, phone, address):
        self._name = name
        self._email = email
        self._phone = phone
        self._address = address
    def view_reservation_history(self):
        return []

    def request_service(self, service_type, description):
        print(f"Service request for {service_type.value} created:
{description}")
        return None  # Would return a Service object in a real
implementation

    def __str__(self):
        return f"Guest(ID: {self._guest_id}, Name: {self._name}, Email:
{self._email}, " \
               f"Phone: {self._phone}, Loyalty Status:
{self._loyalty_status.value})"

# ROOM CLASS
class Room:
    """
    Represents a room in the Royal Stay Hotel.

    This class maintains information about hotel rooms including details
    such as room number, type, amenities, and availability.
    """

    def __init__(self, room_number, room_type, amenities, price_per_night,
max_occupancy):
        """
        Initialize a new Room object.
        """
        self._room_number = room_number
        self._room_type = room_type
        self._amenities = amenities
        self._price_per_night = price_per_night
        self._is_available = True
        self._max_occupancy = max_occupancy

    # Getter and setter methods
    def get_room_number(self):
        return self._room_number

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

    def get_room_type(self):
        return self._room_type
    def set_room_type(self, room_type):
        self._room_type = room_type

    def get_amenities(self):
        return self._amenities

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

    def get_price_per_night(self):
        return self._price_per_night

    def set_price_per_night(self, price_per_night):
        self._price_per_night = price_per_night

    def get_is_available(self):
        return self._is_available

    def set_is_available(self, is_available):
        self._is_available = is_available

    def get_max_occupancy(self):
        return self._max_occupancy

    def set_max_occupancy(self, max_occupancy):
        self._max_occupancy = max_occupancy

    def check_availability(self, check_in_date, check_out_date):
        return self._is_available

    def update_availability(self, status):
        self._is_available = status

    def __str__(self):
        amenities_str = ", ".join([amenity.value for amenity in
self._amenities])
        return f"Room(Number: {self._room_number}, Type:
{self._room_type.value}, " \
               f"Amenities: [{amenities_str}], Price:
${self._price_per_night:.2f}/night, " \
               f"Available: {self._is_available}, Max Occupancy:
{self._max_occupancy})"

# BOOKING CLASS
```

```python
class Booking:
    """
    Represents a booking/reservation in the Royal Stay Hotel.

    This class manages the booking details including guest information,
    room details, dates, and status.
    """

    def __init__(self, guest, room, check_in_date, check_out_date):
        self._booking_id = None
        self._guest = guest
        self._room = room
        self._check_in_date = check_in_date
        self._check_out_date = check_out_date
        self._status = BookingStatus.PENDING
        self._total_amount = 0.0
        # Calculate the initial total amount
        self.calculate_total_amount()

    # Getter and setter methods
    def get_booking_id(self):
        return self._booking_id

    def set_booking_id(self, booking_id):
        self._booking_id = booking_id

    def get_guest(self):
        return self._guest

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

    def get_room(self):
        return self._room

    def set_room(self, room):
        self._room = room
        # Recalculate the total amount as the room has changed
        self.calculate_total_amount()

    def get_check_in_date(self):
        return self._check_in_date

    def set_check_in_date(self, check_in_date):
        self._check_in_date = check_in_date
        # Recalculate the total amount as duration may have changed
        self.calculate_total_amount()

    def get_check_out_date(self):
        return self._check_out_date

    def set_check_out_date(self, check_out_date):
        self._check_out_date = check_out_date
        # Recalculate the total amount as duration may have changed
        self.calculate_total_amount()

    def get_status(self):
        return self._status

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

    def get_total_amount(self):
        return self._total_amount

    def set_total_amount(self, total_amount):
        self._total_amount = total_amount

    def calculate_duration(self):
        delta = self._check_out_date - self._check_in_date
        return delta.days

    def calculate_total_amount(self):
        duration = self.calculate_duration()
        self._total_amount = duration * self._room.get_price_per_night()
        return self._total_amount

    def confirm_booking(self):
        self._status = BookingStatus.CONFIRMED
        self._room.set_is_available(False)
        self.generate_notification()

    def cancel_booking(self):
        if self._status != BookingStatus.CHECKED_IN:
            self._status = BookingStatus.CANCELLED
            self._room.set_is_available(True)
            return True
        return False

    def generate_notification(self):
        """
        Generate and send a notification about the booking status.
        """
        # In a real implementation, this might send an email or SMS
        print(f"Notification: Booking {self._booking_id} has been 
{self._status.value}.")
        print(f"Room {self._room.get_room_number()} is reserved for 
{self._guest.get_name()}")
        print(f"Check-in: {self._check_in_date.strftime('%Y-%m-%d')}")
        print(f"Check-out: {self._check_out_date.strftime('%Y-%m-%d')}")
        print(f"Total Amount: ${self._total_amount:.2f}")

    def __str__(self):
        return f"Booking(ID: {self._booking_id}, Guest: 
{self._guest.get_name()}, " \
               f"Room: {self._room.get_room_number()}, Check-in: 
{self._check_in_date.strftime('%Y-%m-%d')}, " \
               f"Check-out: {self._check_out_date.strftime('%Y-%m-%d')}, " 
\
               f"Status: {self._status.value}, Total: 
${self._total_amount:.2f})"


# PAYMENT CLASS
class Payment:
    """
    Represents a payment for a booking at the Royal Stay Hotel.

    This class manages payment information including amount, method, and 
status.
    """

    def __init__(self, booking, amount, payment_method):
        self._payment_id = None
        self._booking = booking
        self._amount = amount
        self._payment_method = payment_method
        self._payment_date = datetime.now()
        self._status = PaymentStatus.PENDING

    # Getter and setter methods
    def get_payment_id(self):
        return self._payment_id

    def set_payment_id(self, payment_id):
        self._payment_id = payment_id

    def get_booking(self):
        return self._booking

    def set_booking(self, booking):
        self._booking = booking

    def get_amount(self):
        return self._amount

    def set_amount(self, amount):
        self._amount = amount

    def get_payment_method(self):
        return self._payment_method

    def set_payment_method(self, payment_method):
        self._payment_method = payment_method

    def get_payment_date(self):
        return self._payment_date

    def set_payment_date(self, payment_date):
        self._payment_date = payment_date

    def get_status(self):
        return self._status

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

    def process_payment(self):
        try:
            # Simulate successful payment processing
            self._status = PaymentStatus.COMPLETED
            print(f"Payment of ${self._amount:.2f} processed
successfully.")
            return True
        except Exception as e:
            # Handle any errors that might occur during payment processing
            self._status = PaymentStatus.FAILED
            print(f"Payment processing failed: {str(e)}")
            return False
```

```python
    def generate_receipt(self):
        """
        Generate a receipt for this payment.
        """
        receipt = f"Receipt for Payment {self._payment_id}\n"
        receipt += f"Date: {self._payment_date.strftime('%Y-%m-%d
%H:%M:%S')}\n"
        receipt += f"Booking ID: {self._booking.get_booking_id()}\n"
        receipt += f"Guest: {self._booking.get_guest().get_name()}\n"
        receipt += f"Amount: ${self._amount:.2f}\n"
        receipt += f"Payment Method: {self._payment_method.value}\n"
        receipt += f"Status: {self._status.value}\n"

        return receipt

    def refund_payment(self):
        """
        Refund this payment.
        """
        if self._status == PaymentStatus.COMPLETED:
            try:
                # Simulate refund processing
                self._status = PaymentStatus.REFUNDED
                print(f"Refund of ${self._amount:.2f} processed
successfully.")
                return True
            except Exception as e:
                print(f"Refund processing failed: {str(e)}")
                return False
        else:
            print("Cannot refund a payment that has not been completed.")
            return False

    def __str__(self):
        return f"Payment(ID: {self._payment_id}, Booking:
{self._booking.get_booking_id()}, " \
               f"Amount: ${self._amount:.2f}, Method:
{self._payment_method.value}, " \
               f"Date: {self._payment_date.strftime('%Y-%m-%d')}, Status:
{self._status.value})"

# INVOICE CLASS
class Invoice:
    """
    Represents an invoice for a booking at the Royal Stay Hotel.
    This class manages invoice details including itemized charges,
discounts, and totals.
    """

    def __init__(self, booking):
        self._invoice_id = None
        self._booking = booking
        self._issued_date = datetime.now()
        self._due_date = self._issued_date + timedelta(days=7)  # Due in 7
days
        self._itemized_charges = {}  # Dictionary to store itemized
charges
        self._discounts = 0.0

        # Add room charge to itemized charges
        room_charge = booking.get_total_amount()
        self._itemized_charges["Room Charge"] = room_charge

    # Getter and setter methods
    def get_invoice_id(self):
        return self._invoice_id

    def set_invoice_id(self, invoice_id):
        self._invoice_id = invoice_id

    def get_booking(self):
        return self._booking

    def set_booking(self, booking):
        self._booking = booking

    def get_issued_date(self):
        return self._issued_date

    def set_issued_date(self, issued_date):
        self._issued_date = issued_date

    def get_due_date(self):
        return self._due_date

    def set_due_date(self, due_date):
        self._due_date = due_date

    def get_itemized_charges(self):
        return self._itemized_charges

    def set_itemized_charges(self, itemized_charges):
        self._itemized_charges = itemized_charges

    def get_discounts(self):
        return self._discounts

    def set_discounts(self, discounts):
        self._discounts = discounts

    def add_charge(self, description, amount):
        self._itemized_charges[description] = amount

    def apply_discount(self, amount):
        self._discounts += amount

    def calculate_total(self):
        """
        Calculate the total amount due on the invoice.
        """
        total = sum(self._itemized_charges.values()) - self._discounts
        return max(0, total)  # Ensure the total is not negative

    def generate_invoice(self):
        """
        Generate a formatted invoice.
        """
        guest = self._booking.get_guest()
        room = self._booking.get_room()

        invoice = f"INVOICE #{self._invoice_id}\n"
        invoice += f"===================\n\n"
        invoice += f"Issued: {self._issued_date.strftime('%Y-%m-%d')}\n"
        invoice += f"Due: {self._due_date.strftime('%Y-%m-%d')}\n\n"

        invoice += f"Guest: {guest.get_name()}\n"
        invoice += f"Room: {room.get_room_number()}
({room.get_room_type().value})\n"
        invoice += f"Check-in:
{self._booking.get_check_in_date().strftime('%Y-%m-%d')}\n"
        invoice += f"Check-out:
{self._booking.get_check_out_date().strftime('%Y-%m-%d')}\n"
        invoice += f"Duration: {self._booking.calculate_duration()}
nights\n\n"
```

```
        invoice += f"CHARGES:\n"
        for description, amount in self._itemized_charges.items():
            invoice += f"{description}: ${amount:.2f}\n"

        invoice += f"\nSubtotal:
${sum(self._itemized_charges.values()):.2f}\n"

        if self._discounts > 0:
            invoice += f"Discounts: -${self._discounts:.2f}\n"

        invoice += f"Total Due: ${self.calculate_total():.2f}\n"

        return invoice

    def __str__(self):
        return f"Invoice(ID: {self._invoice_id}, Booking:
{self._booking.get_booking_id()}, " \
               f"Issued: {self._issued_date.strftime('%Y-%m-%d')}, " \
               f"Due: {self._due_date.strftime('%Y-%m-%d')}, " \
               f"Total: ${self.calculate_total():.2f})"


# SERVICE CLASS
class Service:
    """
    Represents a service request at the Royal Stay Hotel.

    This class manages service requests from guests for specific rooms.
    """

    def __init__(self, service_type, description, guest, room):
        self._service_id = None
        self._service_type = service_type
        self._description = description
        self._guest = guest
        self._room = room
        self._request_date = datetime.now()
        self._status = ServiceStatus.REQUESTED
        self._staff_assigned = None

    # Getter and setter methods
    def get_service_id(self):
        return self._service_id
```

```python
    def set_service_id(self, service_id):
        self._service_id = service_id

    def get_service_type(self):
        return self._service_type

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

    def get_description(self):
        return self._description

    def set_description(self, description):
        self._description = description

    def get_guest(self):
        return self._guest

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

    def get_room(self):
        return self._room

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

    def get_request_date(self):
        return self._request_date

    def set_request_date(self, request_date):
        self._request_date = request_date

    def get_status(self):
        return self._status

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

    def get_staff_assigned(self):
        return self._staff_assigned

    def set_staff_assigned(self, staff_assigned):
        self._staff_assigned = staff_assigned
    def assign_staff(self, staff_member):
        """
        Assign a staff member to handle this service request.
        """
        self._staff_assigned = staff_member
        self._status = ServiceStatus.ASSIGNED
        print(f"Service {self._service_id} assigned to {staff_member}.")

    def update_status(self, status):
        """
        Update the status of this service request.
        """
        self._status = status
        print(f"Service {self._service_id} status updated to
{status.value}.")

    def calculate_charge(self):
        """
        Calculate the charge for this service, if applicable.
        """
        # Base charges for different service types
        base_charges = {
            ServiceType.HOUSEKEEPING: 0.0,  # Housekeeping is typically
free
            ServiceType.ROOM_SERVICE: 20.0,  # Base charge for room
service
            ServiceType.MAINTENANCE: 0.0,  # Maintenance is typically free
            ServiceType.TRANSPORTATION: 25.0  # Base charge for
transportation
        }

        return base_charges.get(self._service_type, 0.0)

    def __str__(self):
        return f"Service(ID: {self._service_id}, Type:
{self._service_type.value}, " \
               f"Guest: {self._guest.get_name()}, Room:
{self._room.get_room_number()}, " \
               f"Status: {self._status.value}, " \
               f"Requested: {self._request_date.strftime('%Y-%m-%d
%H:%M')}, " \
               f"Staff: {self._staff_assigned if self._staff_assigned else
'Not assigned'})"
```

```python
# REVIEW CLASS
class Review:
    """
    Represents a review submitted by a guest for a booking at the Royal
Stay Hotel.

    This class manages guest reviews and ratings for their hotel
experience.
    """

    def __init__(self, guest, booking, rating, comment):
        self._review_id = None
        self._guest = guest
        self._booking = booking
        self._rating = rating
        self._comment = comment
        self._review_date = datetime.now()

        # Validate the rating
        self.validate_rating(rating)

    # Getter and setter methods
    def get_review_id(self):
        return self._review_id

    def set_review_id(self, review_id):
        self._review_id = review_id

    def get_guest(self):
        return self._guest

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

    def get_booking(self):
        return self._booking

    def set_booking(self, booking):
        self._booking = booking

    def get_rating(self):
        return self._rating

    def set_rating(self, rating):
        """
        Set the rating after validation.
        """
        if self.validate_rating(rating):
            self._rating = rating

    def get_comment(self):
        return self._comment

    def set_comment(self, comment):
        self._comment = comment

    def get_review_date(self):
        return self._review_date

    def set_review_date(self, review_date):
        self._review_date = review_date

    def validate_rating(self, rating):
        """
        Validate that the rating is between 1 and 5.
        """
        if not isinstance(rating, int) or rating < 1 or rating > 5:
            raise ValueError("Rating must be an integer between 1 and 5")
        return True

    def __str__(self):
        return f"Review(ID: {self._review_id}, Guest:
{self._guest.get_name()}, " \
               f"Booking: {self._booking.get_booking_id()}, Rating:
{self._rating}/5, " \
               f"Date: {self._review_date.strftime('%Y-%m-%d')}, " \
               f"Comment: {self._comment[:30]}{'...' if len(self._comment)
> 30 else ''})"

# LOYALTY PROGRAM CLASS
class LoyaltyProgram:
    """
    Represents a loyalty program for a guest at the Royal Stay Hotel.

    This class manages loyalty points, tier status, and rewards for hotel
guests.
    """

    def __init__(self, guest):
        self._guest = guest
        self._points_earned = 0
        self._points_redeemed = 0
        self._current_tier = LoyaltyTier.STANDARD
        self._member_since = datetime.now()
        self._last_activity = datetime.now()

    # Getter and setter methods
    def get_guest(self):
        return self._guest

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

    def get_points_earned(self):
        return self._points_earned

    def set_points_earned(self, points_earned):
        self._points_earned = points_earned
        self.update_tier()  # Update tier based on new points

    def get_points_redeemed(self):
        return self._points_redeemed

    def set_points_redeemed(self, points_redeemed):
        self._points_redeemed = points_redeemed

    def get_current_tier(self):
        return self._current_tier

    def set_current_tier(self, current_tier):
        self._current_tier = current_tier

    def get_member_since(self):
        return self._member_since

    def set_member_since(self, member_since):
        self._member_since = member_since

    def get_last_activity(self):
        return self._last_activity

    def set_last_activity(self, last_activity):
        self._last_activity = last_activity

    def earn_points(self, booking):
        """
        Earn points for a booking.
        """
        # Calculate points based on booking total amount
        # For example, 1 point per dollar spent
        points = int(booking.get_total_amount())

        # Apply tier multiplier
        if self._current_tier == LoyaltyTier.SILVER:
            points = int(points * 1.25)  # 25% bonus for Silver
        elif self._current_tier == LoyaltyTier.GOLD:
            points = int(points * 1.5)   # 50% bonus for Gold
        elif self._current_tier == LoyaltyTier.PLATINUM:
            points = int(points * 2.0)   # 100% bonus for Platinum

        self._points_earned += points
        self._last_activity = datetime.now()
        self.update_tier()  # Update tier based on new points

        print(f"{self._guest.get_name()} earned {points} loyalty points.")
        return points

    def redeem_points(self, points):
        """
        Redeem loyalty points for rewards.
        """
        available_points = self.calculate_available_points()

        if points <= available_points:
            self._points_redeemed += points
            self._last_activity = datetime.now()

            # Calculate the value of redeemed points (e.g., $0.10 per
point)
            value = points * 0.1

            print(f"{self._guest.get_name()} redeemed {points} points for
${value:.2f} value.")
            return True
        else:
            print(f"Insufficient points. Available: {available_points},
Requested: {points}")
            return False

    def calculate_available_points(self):
        return max(0, self._points_earned - self._points_redeemed)

    def update_tier(self):
        available_points = self.calculate_available_points()

        if available_points >= 10000:
            new_tier = LoyaltyTier.PLATINUM
        elif available_points >= 5000:
            new_tier = LoyaltyTier.GOLD
        elif available_points >= 1000:
            new_tier = LoyaltyTier.SILVER
        else:
            new_tier = LoyaltyTier.STANDARD

        # Update tier if changed
        if new_tier != self._current_tier:
            old_tier = self._current_tier
            self._current_tier = new_tier
            print(f"{self._guest.get_name()}'s loyalty tier updated from
{old_tier.value} to {new_tier.value}.")

    def __str__(self):
        return f"LoyaltyProgram(Guest: {self._guest.get_name()}, " \
               f"Points Earned: {self._points_earned}, Points Redeemed:
{self._points_redeemed}, " \
               f"Available Points: {self.calculate_available_points()}, "
\
               f"Tier: {self._current_tier.value}, Member Since:
{self._member_since.strftime('%Y-%m-%d')})"
```

Code Structure:

My code is organized into the following:

- Enum Classes: Define fixed sets of values (RoomType, Amenity, etc.)
- Guest Class: Manages guest information
- Room Class: Handles room details and availability
- Booking Class: Manages reservations
- Payment Class: Processes payments

- Invoice Class: Generates detailed billing information
- Service Class: Handles guest service requests
- Review Class: Manages feedback and ratings
- LoyaltyProgram Class: Tracks loyalty points and benefits

Relationships Implementation

My code implements all the relationships from my UML diagram:

- Guest and Booking have a one-to-many relationship
- Room and Booking show how rooms are reserved
- Booking and Payment show how bookings are paid for
- Booking and Invoice demonstrate how invoices are generated
- Guest and LoyaltyProgram show the loyalty system
- Guest, Room, and Service show how service requests work
- Guest, Booking, and Review show the review system

Key Features

- Private attributes (using underscore prefix)
- Getter and setter methods for all attributes
- Proper exception handling
- String representation for all classes
- Comprehensive documentation

C. Define Test Cases

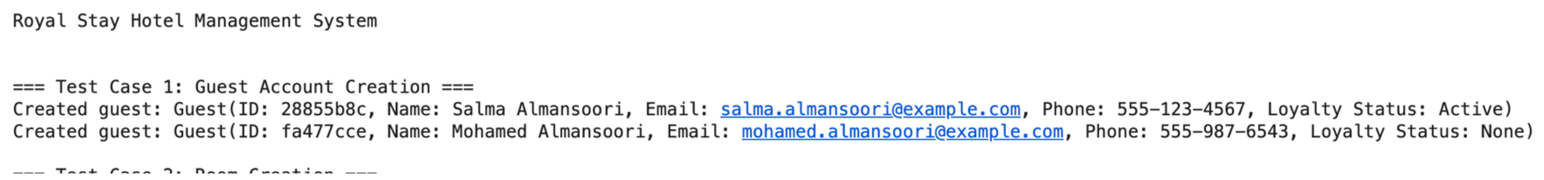
```
# MAIN TEST CODE
def generate_id():
    """Generate a unique ID."""
    return str(uuid.uuid4())[:8]

def main():
    """Main function to test the Hotel Management System."""
    print("Royal Stay Hotel Management System\n")

    # Test Case 1: Guest Account Creation
    print("\n=== Test Case 1: Guest Account Creation ===")

    # Create first guest
    guest1 = Guest("Salma Almansoori", "salma.almansoori@example.com",
"555-123-4567", "123 Main St, City")
    guest1.set_guest_id(generate_id())
    guest1.set_loyalty_status(LoyaltyStatus.ACTIVE)
    print(f"Created guest: {guest1}")

    # Create second guest
    guest2 = Guest("Mohamed Almansoori", "mohamed.almansoori@example.com",
"555-987-6543", "456 West Abc, Town")
    guest2.set_guest_id(generate_id())
    print(f"Created guest: {guest2}")

    # Test Case 2: Room Creation
    print("\n=== Test Case 2: Room Creation ===")

    # Create single room
    single_room = Room("101", RoomType.SINGLE, [Amenity.WIFI, Amenity.TV],
100.0, 1)
    print(f"Created room: {single_room}")

    # Create deluxe room
    deluxe_room = Room("201", RoomType.DELUXE, [Amenity.WIFI, Amenity.TV,
Amenity.AIRCON, Amenity.BALCONY], 250.0, 4)
    print(f"Created room: {deluxe_room}")

    # Test Case 3: Making a Room Reservation
    print("\n=== Test Case 3: Making a Room Reservation ===")
    # Book single room for guest1
    check_in1 = datetime.now() + timedelta(days=7)
    check_out1 = datetime.now() + timedelta(days=10)
    booking1 = Booking(guest1, single_room, check_in1, check_out1)
    booking1.set_booking_id(generate_id())
    booking1.confirm_booking()
    print(f"Created booking: {booking1}")

    # Book deluxe room for guest2
    check_in2 = datetime.now() + timedelta(days=14)
    check_out2 = datetime.now() + timedelta(days=21)
    booking2 = Booking(guest2, deluxe_room, check_in2, check_out2)
    booking2.set_booking_id(generate_id())
    booking2.confirm_booking()
    print(f"Created booking: {booking2}")

    # Test Case 4: Payment Processing
    print("\n=== Test Case 4: Payment Processing ===")

    # Process payment for booking1
    payment1 = Payment(booking1, booking1.get_total_amount(),
PaymentMethod.CREDIT_CARD)
    payment1.set_payment_id(generate_id())
    payment1.process_payment()
    print(f"Processed payment: {payment1}")
    print(payment1.generate_receipt())

    # Process payment for booking2
    payment2 = Payment(booking2, booking2.get_total_amount(),
PaymentMethod.MOBILE_WALLET)
    payment2.set_payment_id(generate_id())
    payment2.process_payment()
    print(f"Processed payment: {payment2}")

    # Test Case 5: Invoice Generation
    print("\n=== Test Case 5: Invoice Generation ===")

    # Generate invoice for booking1
    invoice1 = Invoice(booking1)
    invoice1.set_invoice_id(generate_id())
    invoice1.add_charge("Room Service - Breakfast", 25.0)
    invoice1.apply_discount(10.0)  # Apply $10 discount
    print(invoice1.generate_invoice())

    # Generate invoice for booking2
    invoice2 = Invoice(booking2)
    invoice2.set_invoice_id(generate_id())
    invoice2.add_charge("Spa Service", 120.0)
    invoice2.add_charge("Airport Transfer", 50.0)
    print(invoice2.generate_invoice())

    # Test Case 6: Loyalty Program
    print("\n=== Test Case 6: Loyalty Program ===")

    # Create loyalty program for guest1
    loyalty1 = LoyaltyProgram(guest1)
    loyalty1.earn_points(booking1)
    print(f"Loyalty program: {loyalty1}")

    # Redeem points
    loyalty1.redeem_points(50)
    print(f"After redemption: {loyalty1}")

    # Test Case 7: Service Requests
    print("\n=== Test Case 7: Service Requests ===")

    # Create service request for guest1
    service1 = Service(ServiceType.ROOM_SERVICE, "Breakfast delivery at
8am", guest1, single_room)
    service1.set_service_id(generate_id())
    service1.assign_staff("Employee001")
    service1.update_status(ServiceStatus.IN_PROGRESS)
    print(f"Service request: {service1}")

    # Create service request for guest2
    service2 = Service(ServiceType.TRANSPORTATION, "Airport pickup on
arrival", guest2, deluxe_room)
    service2.set_service_id(generate_id())
    print(f"Service request: {service2}")

    # Test Case 8: Review Submission
    print("\n=== Test Case 8: Review Submission ===")

    # Submit review for booking1
    review1 = Review(guest1, booking1, 5, "Excellent service and
comfortable room!")
    review1.set_review_id(generate_id())
    print(f"Review: {review1}")

    # Try to submit an invalid review
    try:
        review2 = Review(guest2, booking2, 6, "Rating out of range")
    except ValueError as e:
        print(f"Error: {e}")

    review2 = Review(guest2, booking2, 4, "Good experience, but the spa 
was a bit crowded.")
    review2.set_review_id(generate_id())
    print(f"Review: {review2}")

    # Test Case 9: Booking Cancellation
    print("\n=== Test Case 9: Booking Cancellation ===")

    # Create a new booking to cancel
    check_in3 = datetime.now() + timedelta(days=30)
    check_out3 = datetime.now() + timedelta(days=35)
    booking3 = Booking(guest1, deluxe_room, check_in3, check_out3)
    booking3.set_booking_id(generate_id())
    booking3.confirm_booking()
    print(f"Created booking: {booking3}")

    # Cancel the booking
    result = booking3.cancel_booking()
    print(f"Booking cancelled: {result}")
    print(f"Updated booking: {booking3}")

if __name__ == "__main__":
    main()
```

Output:

```
Royal Stay Hotel Management System


=== Test Case 1: Guest Account Creation ===
Created guest: Guest(ID: c0617393, Name: Salma Almansoori, Email: salma.almansoori@example.com,
Phone: 555-123-4567, Loyalty Status: Active)
Created guest: Guest(ID: 11388cda, Name: Mohamed Almansoori, Email:
mohamed.almansoori@example.com, Phone: 555-987-6543, Loyalty Status: None)

=== Test Case 2: Room Creation ===
Created room: Room(Number: 101, Type: Single, Amenities: [WiFi, TV], Price: $100.00/night, Available: True,
Max Occupancy: 1)
Created room: Room(Number: 201, Type: Deluxe, Amenities: [WiFi, TV, Air Conditioning, Balcony], Price:
$250.00/night, Available: True, Max Occupancy: 4)

=== Test Case 3: Making a Room Reservation ===
Notification: Booking 2522c648 has been Confirmed.
```

```
Room 101 is reserved for Salma Almansoori
Check-in: 2025-04-04
Check-out: 2025-04-07
Total Amount: $300.00
Created booking: Booking(ID: 2522c648, Guest: Salma Almansoori, Room: 101, Check-in: 2025-04-04, Check-
out: 2025-04-07, Status: Confirmed, Total: $300.00)
Notification: Booking 948c8673 has been Confirmed.
Room 201 is reserved for Mohamed Almansoori
Check-in: 2025-04-11
Check-out: 2025-04-18
Total Amount: $1750.00
Created booking: Booking(ID: 948c8673, Guest: Mohamed Almansoori, Room: 201, Check-in: 2025-04-11,
Check-out: 2025-04-18, Status: Confirmed, Total: $1750.00)

=== Test Case 4: Payment Processing ===
Payment of $300.00 processed successfully.
Processed payment: Payment(ID: 18e8c9fa, Booking: 2522c648, Amount: $300.00, Method: Credit Card, Date:
2025-03-28, Status: Completed)
Receipt for Payment 18e8c9fa
Date: 2025-03-28 01:39:19
Booking ID: 2522c648
Guest: Salma Almansoori
Amount: $300.00
Payment Method: Credit Card
Status: Completed

Payment of $1750.00 processed successfully.
Processed payment: Payment(ID: b234da61, Booking: 948c8673, Amount: $1750.00, Method: Mobile Wallet,
Date: 2025-03-28, Status: Completed)

=== Test Case 5: Invoice Generation ===
INVOICE #a9a25110
===================

Issued: 2025-03-28
Due: 2025-04-04

Guest: Salma Almansoori
Room: 101 (Single)
Check-in: 2025-04-04
Check-out: 2025-04-07
Duration: 3 nights

CHARGES:
Room Charge: $300.00
Room Service - Breakfast: $25.00

Subtotal: $325.00
Discounts: -$10.00
Total Due: $315.00

INVOICE #8319aff5
===================

Issued: 2025-03-28
```

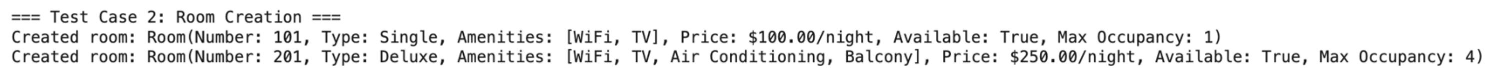
```
Due: 2025-04-04

Guest: Mohamed Almansoori
Room: 201 (Deluxe)
Check-in: 2025-04-11
Check-out: 2025-04-18
Duration: 7 nights

CHARGES:
Room Charge: $1750.00
Spa Service: $120.00
Airport Transfer: $50.00

Subtotal: $1920.00
Total Due: $1920.00


=== Test Case 6: Loyalty Program ===
Salma Almansoori earned 300 loyalty points.
Loyalty program: LoyaltyProgram(Guest: Salma Almansoori, Points Earned: 300, Points Redeemed: 0,
Available Points: 300, Tier: Standard, Member Since: 2025-03-28)
Salma Almansoori redeemed 50 points for $5.00 value.
After redemption: LoyaltyProgram(Guest: Salma Almansoori, Points Earned: 300, Points Redeemed: 50,
Available Points: 250, Tier: Standard, Member Since: 2025-03-28)

=== Test Case 7: Service Requests ===
Service 2cbc80e0 assigned to Employee001.
Service 2cbc80e0 status updated to In Progress.
Service request: Service(ID: 2cbc80e0, Type: Room Service, Guest: Salma Almansoori, Room: 101, Status: In
Progress, Requested: 2025-03-28 01:39, Staff: Employee001)
Service request: Service(ID: 345fd870, Type: Transportation, Guest: Mohamed Almansoori, Room: 201,
Status: Requested, Requested: 2025-03-28 01:39, Staff: Not assigned)

=== Test Case 8: Review Submission ===
Review: Review(ID: 9e9f4911, Guest: Salma Almansoori, Booking: 2522c648, Rating: 5/5, Date: 2025-03-28,
Comment: Excellent service and comforta...)
Error: Rating must be an integer between 1 and 5
Review: Review(ID: 11fa6537, Guest: Mohamed Almansoori, Booking: 948c8673, Rating: 4/5, Date: 2025-03-
28, Comment: Good experience, but the spa w...)

=== Test Case 9: Booking Cancellation ===
Notification: Booking aef8e40e has been Confirmed.
Room 201 is reserved for Salma Almansoori
Check-in: 2025-04-27
Check-out: 2025-05-02
Total Amount: $1250.00
Created booking: Booking(ID: aef8e40e, Guest: Salma Almansoori, Room: 201, Check-in: 2025-04-27, Check-
out: 2025-05-02, Status: Confirmed, Total: $1250.00)
Booking cancelled: True
Updated booking: Booking(ID: aef8e40e, Guest: Salma Almansoori, Room: 201, Check-in: 2025-04-27, Check-
out: 2025-05-02, Status: Cancelled, Total: $1250.00)
```



```
Royal Stay Hotel Management System


=== Test Case 1: Guest Account Creation ===
Created guest: Guest(ID: 28855b8c, Name: Salma Almansoori, Email: salma.almansoori@example.com, Phone: 555-123-4567, Loyalty Status: Active)
Created guest: Guest(ID: fa477cce, Name: Mohamed Almansoori, Email: mohamed.almansoori@example.com, Phone: 555-987-6543, Loyalty Status: None)
```

Test Case 1: Guest Account Creation

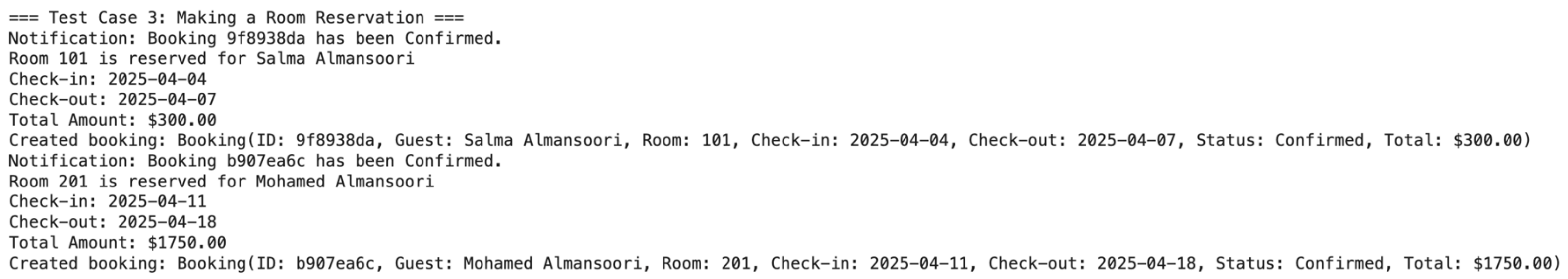
- Description: This test verifies that the system can create and store guest information correctly.
- What it tests: The ability to create guest accounts with personal details such as name, email, phone, and address, and to assign a loyalty status.
- Example: I created two guest accounts - one for "Salma Almansoori" and another for "Mohamed Almansoori" with different contact details and loyalty statuses.



```
=== Test Case 2: Room Creation ===
Created room: Room(Number: 101, Type: Single, Amenities: [WiFi, TV], Price: $100.00/night, Available: True, Max Occupancy: 1)
Created room: Room(Number: 201, Type: Deluxe, Amenities: [WiFi, TV, Air Conditioning, Balcony], Price: $250.00/night, Available: True, Max Occupancy: 4)
```

Test Case 2: Room Creation

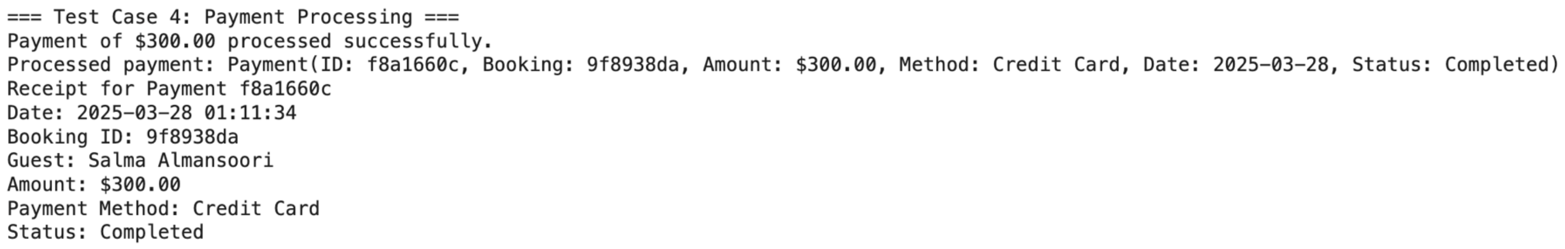
- Description: This test verifies that the system can create and maintain room information.
- What it tests: The ability to define different types of rooms with various amenities, prices, and occupancy limits.
- Example: I created a single room (room 101) with basic amenities and a deluxe room (room 201) with more amenities and a higher price.



```
=== Test Case 3: Making a Room Reservation ===
Notification: Booking 9f8938da has been Confirmed.
Room 101 is reserved for Salma Almansoori
Check-in: 2025-04-04
Check-out: 2025-04-07
Total Amount: $300.00
Created booking: Booking(ID: 9f8938da, Guest: Salma Almansoori, Room: 101, Check-in: 2025-04-04, Check-out: 2025-04-07, Status: Confirmed, Total: $300.00)
Notification: Booking b907ea6c has been Confirmed.
Room 201 is reserved for Mohamed Almansoori
Check-in: 2025-04-11
Check-out: 2025-04-18
Total Amount: $1750.00
Created booking: Booking(ID: b907ea6c, Guest: Mohamed Almansoori, Room: 201, Check-in: 2025-04-11, Check-out: 2025-04-18, Status: Confirmed, Total: $1750.00)
```

Test Case 3: Making a Room Reservation

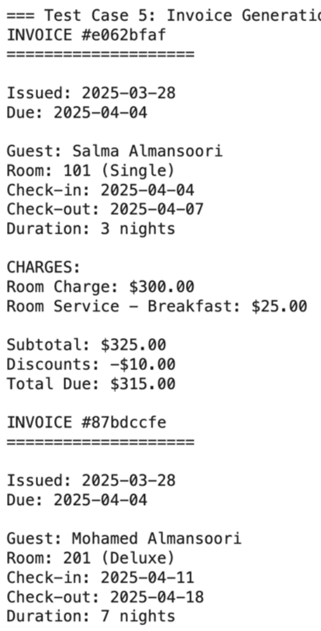
- Description: This test verifies that guests can book rooms for specific dates.
- What it tests: The ability to create bookings that associate guests with rooms for specific date ranges, and to calculate the total cost.
- Example: I created bookings for both guests with different check-in and check-out dates, and confirmed these bookings.



```
=== Test Case 4: Payment Processing ===
Payment of $300.00 processed successfully.
Processed payment: Payment(ID: f8a1660c, Booking: 9f8938da, Amount: $300.00, Method: Credit Card, Date: 2025-03-28, Status: Completed)
Receipt for Payment f8a1660c
Date: 2025-03-28 01:11:34
Booking ID: 9f8938da
Guest: Salma Almansoori
Amount: $300.00
Payment Method: Credit Card
Status: Completed
```

Test Case 4: Payment Processing

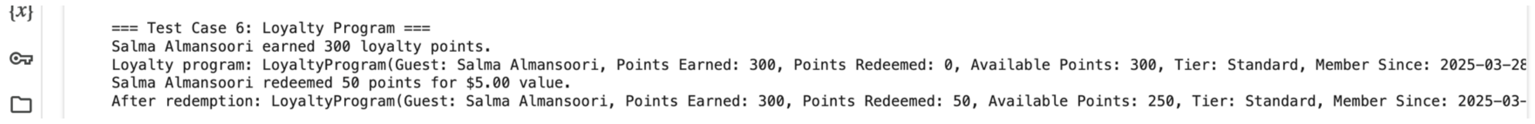
- Description: This test verifies that the system can process payments for bookings.
- What it tests: The ability to process payments using different payment methods and to generate receipts.
- Example: I processed a credit card payment for the first booking and a mobile wallet payment for the second booking.



```
=== Test Case 5: Invoice Generation ===
INVOICE #e062bfaf
====================

Issued: 2025-03-28
Due: 2025-04-04

Guest: Salma Almansoori
Room: 101 (Single)
Check-in: 2025-04-04
Check-out: 2025-04-07
Duration: 3 nights

CHARGES:
Room Charge: $300.00
Room Service - Breakfast: $25.00

Subtotal: $325.00
Discounts: -$10.00
Total Due: $315.00

INVOICE #87bdccfe
====================

Issued: 2025-03-28
Due: 2025-04-04

Guest: Mohamed Almansoori
Room: 201 (Deluxe)
Check-in: 2025-04-11
Check-out: 2025-04-18
Duration: 7 nights
```

Test Case 5: Invoice Generation

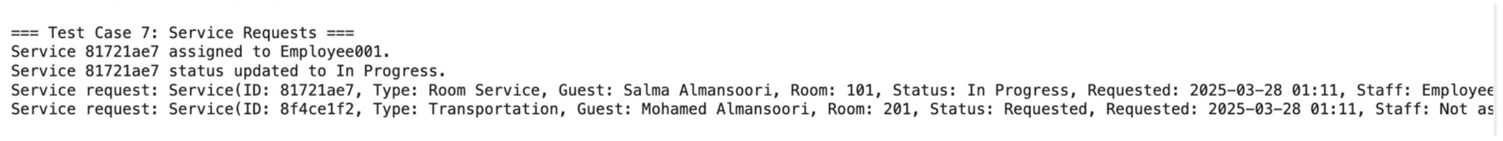
- Description: This test verifies that the system can generate detailed invoices for bookings.
- What it tests: The ability to create invoices that include room charges, additional services, and discounts, and to calculate the final amount.
- Example: I generated invoices for both bookings, adding different additional charges (like room service and spa services) and applying discounts.



```
=== Test Case 6: Loyalty Program ===
Salma Almansoori earned 300 loyalty points.
Loyalty program: LoyaltyProgram(Guest: Salma Almansoori, Points Earned: 300, Points Redeemed: 0, Available Points: 300, Tier: Standard, Member Since: 2025-03-28
Salma Almansoori redeemed 50 points for $5.00 value.
After redemption: LoyaltyProgram(Guest: Salma Almansoori, Points Earned: 300, Points Redeemed: 50, Available Points: 250, Tier: Standard, Member Since: 2025-03-
```

Test Case 6: Loyalty Program

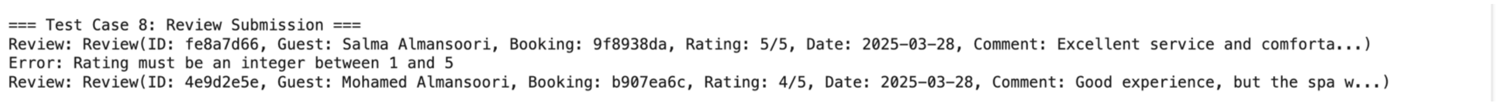
- Description: This test verifies that the system can manage a loyalty rewards program.
- What it tests: The ability to track loyalty points for bookings, update loyalty tiers, and allow points redemption.
- Example: I created a loyalty program for a guest, earned points based on a booking, and redeemed some of those points.



```
=== Test Case 7: Service Requests ===
Service 81721ae7 assigned to Employee001.
Service 81721ae7 status updated to In Progress.
Service request: Service(ID: 81721ae7, Type: Room Service, Guest: Salma Almansoori, Room: 101, Status: In Progress, Requested: 2025-03-28 01:11, Staff: Employee
Service request: Service(ID: 8f4ce1f2, Type: Transportation, Guest: Mohamed Almansoori, Room: 201, Status: Requested, Requested: 2025-03-28 01:11, Staff: Not as
```

Test Case 7: Service Requests

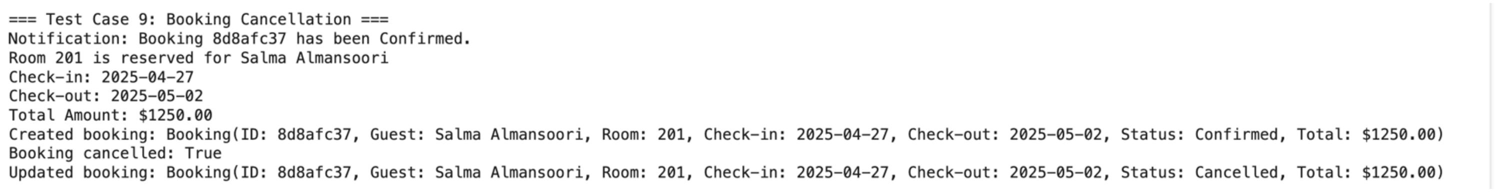
- Description: This test verifies that guests can request additional services.
- What it tests: The ability to create service requests, assign staff to fulfill them, and update their status.
- Example: I created a room service request and a transportation service request for different guests, and updated their statuses.



```
=== Test Case 8: Review Submission ===
Review: Review(ID: fe8a7d66, Guest: Salma Almansoori, Booking: 9f8938da, Rating: 5/5, Date: 2025-03-28, Comment: Excellent service and comforta...)
Error: Rating must be an integer between 1 and 5
Review: Review(ID: 4e9d2e5e, Guest: Mohamed Almansoori, Booking: b907ea6c, Rating: 4/5, Date: 2025-03-28, Comment: Good experience, but the spa w...)
```

Test Case 8: Review Submission

- Description: This test verifies that guests can submit reviews after their stay.
- What it tests: The ability to create reviews with ratings and comments, and to validate input (like ensuring ratings are between 1 and 5).
- Example: I created positive reviews for bookings and tested the system's handling of invalid ratings.



```
=== Test Case 9: Booking Cancellation ===
Notification: Booking 8d8afc37 has been Confirmed.
Room 201 is reserved for Salma Almansoori
Check-in: 2025-04-27
Check-out: 2025-05-02
Total Amount: $1250.00
Created booking: Booking(ID: 8d8afc37, Guest: Salma Almansoori, Room: 201, Check-in: 2025-04-27, Check-out: 2025-05-02, Status: Confirmed, Total: $1250.00)
Booking cancelled: True
Updated booking: Booking(ID: 8d8afc37, Guest: Salma Almansoori, Room: 201, Check-in: 2025-04-27, Check-out: 2025-05-02, Status: Cancelled, Total: $1250.00)
```

Test Case 9: Booking Cancellation

- Description: This test verifies that bookings can be cancelled.
- What it tests: The ability to cancel bookings and update the room's availability status.
- Example: I created a new booking and then cancelled it, by verifying that the room became available again.

Overall, throughout these test cases, I demonstrated proper exception handling, particularly in the Review class where I validated that ratings must be between 1 and 5, and threw an appropriate exception when an invalid rating was provided. The test cases confirm that all the functional requirements of the Royal Stay Hotel Management System have been implemented correctly and work as expected.

D. Documentation

GitHub Repository Link : **https://github.com/Salmnns/Assignment-2-Royal-Stay-Hotel-Management-System**

Summary:

Finally, working on the Royal Stay Hotel Management System taught me valuable skills in software design and programming. I learned how to use UML diagrams to map real-world entities like guests, rooms, and bookings into a visual plan before writing any code. This helped me understand how different parts of the hotel system connect and work together. When programming in Python, I practiced creating classes with private attributes and public methods, which keeps the data safe while allowing controlled access. I also learned the importance of proper documentation through comments and clear method names. The most challenging part was implementing relationships between classes, like making sure a booking correctly connects a guest to a room. Testing my code revealed bugs I hadn't considered, teaching me that thorough testing is essential. Overall, this project helped me understand how object-oriented programming can organize complex systems into manageable parts.