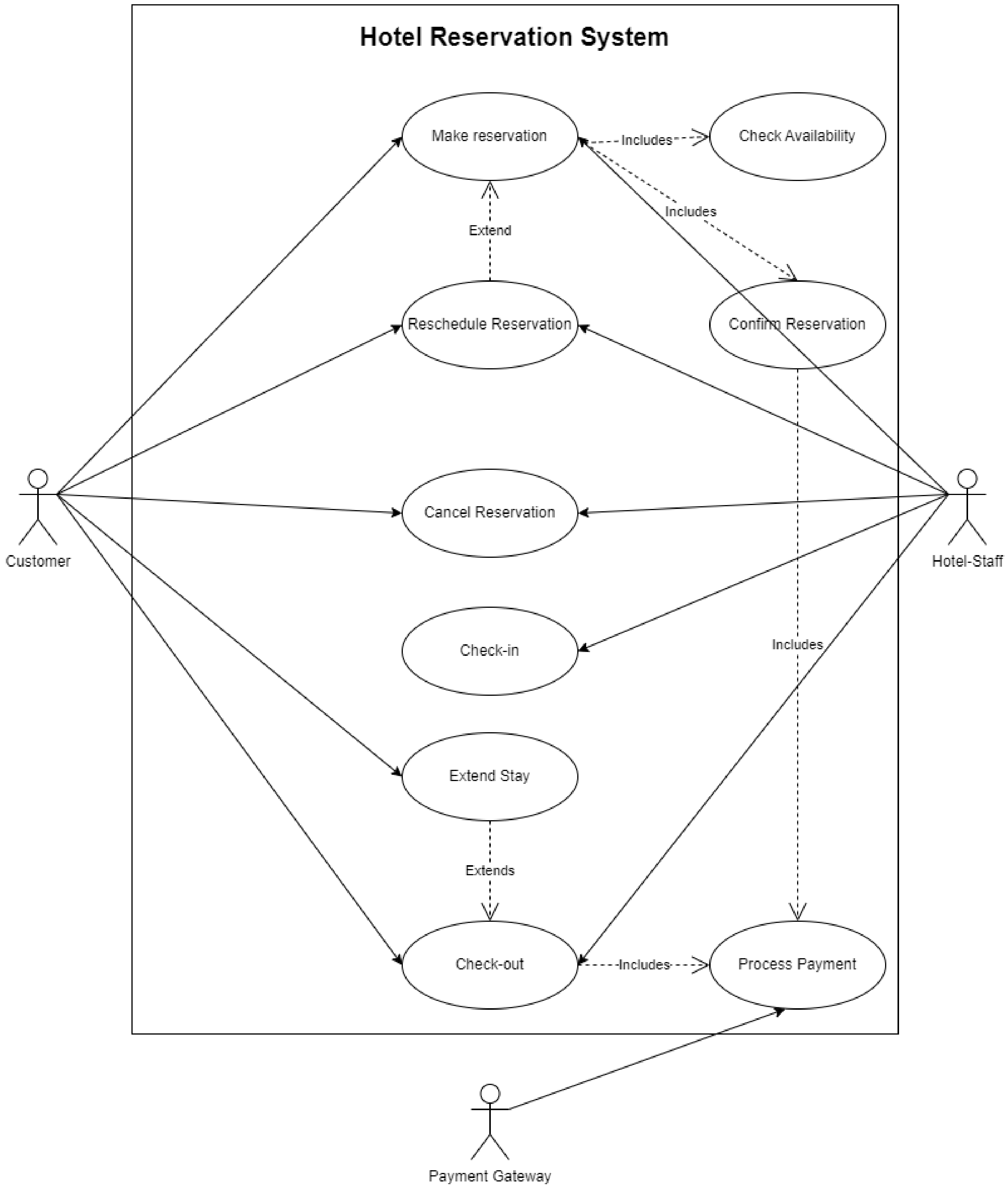
Hotel Reservation System

UML Use-Case Diagrams and Descriptions



Description:

The hotel reservation system facilitates the process of making reservations, managing check-ins and check-outs, processing payments, and handling customer requests such as extending stays, canceling, or rescheduling reservations. The diagram involves two main actors, Customer and Hotel Staff, and an external Payment Gateway.

Actors:

1. Customer:
    - Initiates reservations.
    - Can cancel or reschedule reservations.
    - Checks in and checks out.
    - May extend their stay by a few hours.
2. Hotel Staff:
    - Assists customers by making reservations on their behalf.
    - Manages customer check-ins, check-outs, and handles cancellations and rescheduling.
3. Payment Gateway:
    - Processes payments during reservation confirmation and at check-out.

Main Use Cases:

1. Make Reservation:
    - This is the primary use case where a customer or hotel staff initiates the process of booking a room.
    - The system will check room availability and, if available, confirm the reservation and process the payment.
    - This use case includes:
        - Check Availability: The system must verify if the room is available for the desired dates before proceeding with the booking.
        - Confirm Reservation: Once the availability is confirmed, the system requests customer details and finalizes the reservation.
        - Process Payment: This is triggered after confirmation to ensure that the payment is successfully processed to secure the reservation.
2. Cancel Reservation:
    - The customer or hotel staff can cancel a reservation if needed.
    - A reservation can only be canceled without penalty if it's done 3 days before the check-in date.
3. Reschedule Reservation:
    - The customer or hotel staff can modify the reservation by rescheduling it before the check-in date.
    - This is dependent on room availability.

4. Check-In:
    - Customers check in on the day of their reservation. Hotel staff will update the system to mark the customer as checked in.
5. Check-Out:
    - When a customer leaves the hotel, hotel staff or the customer initiates the check-out process.
    - Any outstanding payment (such as the balance for additional services) must be processed.
6. Extend Stay (Extend Relationship):
    - This is an optional use case that allows the customer to extend their stay by a few hours.
    - It is connected as an extend to the Check-Out use case, meaning it happens if the customer chooses to stay longer before checking out.

Include Relationships:

- Check Availability → Make Reservation:
    - The Check Availability use case is included in the Make Reservation process. This means that checking availability is a mandatory step that always occurs when a reservation is made, ensuring that the room is available before proceeding.
- Confirm Reservation → Make Reservation:
    - The Confirm Reservation use case is included in the Make Reservation process after availability is confirmed. This step ensures that the reservation is finalized by capturing necessary details and securing the booking.
- Process Payment → Confirm Reservation:
    - The Process Payment use case is included in Confirm Reservation, meaning it is required to finalize the reservation after confirming availability. The reservation is only confirmed if payment is successfully processed.

Extend Relationships:

- Extend Stay → Check-Out:
    - The Extend Stay use case is connected as an extend relationship to the Check-Out use case. This means that extending a stay is an optional action that happens before the check-out process is completed. It allows customers to prolong their stay by additional hours if required.

Use-Case Description Tables

Use case	Make Reservation
Actors	Customer, Hotel Staff
Trigger	Customer requests to book a room via the system or phone.
Preconditions	Customer or hotel staff has access to the reservation system.
Main Scenario	1. Customer selects check-in/check-out dates. 2. System checks availability of rooms for the selected dates. 3. If available, the customer enters personal details (name, email, etc.). 4. System generates a reservation summary. 5. Customer chooses a payment method. 6. System processes payment. 7. Customer receives an email/SMS confirmation.
Exceptions	1. No rooms available: System displays an error and suggests alternate dates or room types. 2. Payment fails: System notifies customer to retry or use a different payment method.

Use Case	Confirm Reservation
Actors	Customer, Hotel Staff
Trigger	Availability check shows rooms are available and the customer proceeds with booking.
Preconditions	Rooms must be available for the selected dates.
Main Scenario	1. System asks for customer details (e.g., name, email, phone). 2. System calculates the total cost, including taxes and fees. 3. Customer verifies the reservation summary. 4. System generates a reservation number.
Exceptions	1. Room unavailable after confirmation step: Notify customer, and suggest alternatives. 2. Incorrect personal information: Customer is prompted to correct details.

Use Case	Process Payment
Actors	Customer, Payment Gateway
Trigger	Customer confirms the reservation.
Preconditions	Reservation is confirmed and requires payment to secure.
Main Scenario	1. Customer selects a payment method (credit card, etc.). 2. System forwards payment details to the Payment Gateway. 3. Payment Gateway processes the payment. 4. System confirms payment success and updates the reservation status.
Exceptions	1. Payment declined: System informs customer and prompts retry or alternate payment method. 2. Payment gateway failure: System logs error and requests customer to retry later.

Use Case	Cancel Reservation
Actors	Customer, Hotel Staff
Trigger	Customer decides to cancel the reservation.
Preconditions	Reservation must be made and must be cancelable (within the allowed time frame).
Main Scenario	1. Customer accesses the reservation system. 2. Customer selects the reservation they want to cancel. 3. System cancels the reservation and refunds the advance payment (if within the allowed period). 4. Customer receives a cancellation confirmation.
Exceptions	1. Reservation cannot be canceled (less than 3 days before check-in): System displays an error message, and no refund is processed.

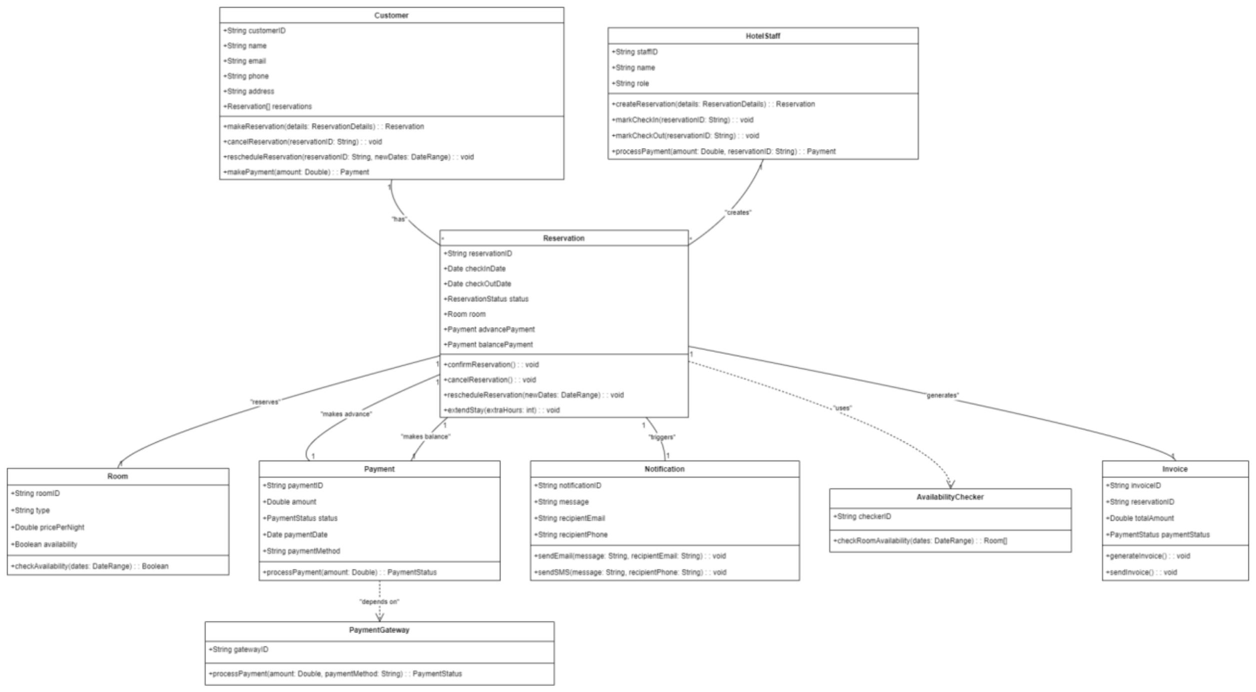
Use Case	Reschedule Reservation
Actors	Customer, Hotel Staff
Trigger	Customer requests to change the reservation dates.
Preconditions	A reservation is already made, and the new dates must be within the allowed rescheduling period.
Main Scenario	1. Customer accesses their reservation. 2. Customer selects new check-in and check-out dates. 3. System checks room availability for the new dates. 4. System confirms the updated reservation and sends a notification to the customer.
Exceptions	1. New dates unavailable: System displays an error message and suggests alternate dates. 2. Reschedule period expired (less than 3 days before check-in): System denies the rescheduling request.

Use Case	Check-In
Actors	Customer, Hotel Staff
Trigger	Customer arrives at the hotel for check-in.
Preconditions	A confirmed reservation exists.
Main Scenario	1. Hotel staff accesses the customer’s reservation. 2. Staff marks the reservation as "checked in" in the system. 3. Customer is granted access to the room.
Exceptions	1. No valid reservation: Staff informs the customer, and check-in is denied.

Use Case	Check-Out
Actors	Customer, Hotel Staff, Payment Gateway
Trigger	Customer checks out of the hotel.
Preconditions	Customer has checked in and stayed in the hotel.
Main Scenario	1. Hotel staff accesses the customer’s reservation. 2. Customer settles any outstanding payments. 3. Hotel staff marks the reservation as "checked out". 4. System generates and sends an invoice to the customer.
Exceptions	1. Outstanding balance not paid: System prevents check-out until payment is completed. 2. Payment processing failure: System prompts retry or alternate payment method.

Use Case	Extend Stay
Actors	Customer, Hotel Staff
Trigger	Customer requests to extend their stay by a few hours.
Preconditions	Customer is checked in, and the room is available for the extended time.
Main Scenario	1. Customer requests an extended stay. 2. System checks room availability for the requested extension period. 3. If available, the system calculates the additional cost. 4. Customer confirms the extended stay. 5. System updates the reservation and processes the payment for the extended time.
Exceptions	1. Room not available for extension: System displays an error message, and no extension is allowed. 2. Payment failure for the extension: System prompts retry or alternate payment method.

UML Class Diagrams and Description


Customer
+String customerID
+String name
+String email
+String phone
+String address
+Reservation[] reservations
+makeReservation(details: ReservationDetails) : : Reservation
+cancelReservation(reservationID: String) : : void
+rescheduleReservation(reservationID: String, newDates: DateRange) : : void
+makePayment(amount: Double) : : Payment
HotelStaff
+String staffID
+String name
+String role
+createReservation(details: ReservationDetails) : : Reservation
+markCheckIn(reservationID: String) : : void
+markCheckOut(reservationID: String) : : void
+processPayment(amount: Double, reservationID: String) : : Payment
"has"
"creates"
Reservation
+String reservationID
+Date checkInDate
+Date checkOutDate
+ReservationStatus status
+Room room
+Payment advancePayment
+Payment balancePayment
+confirmReservation() : : void
+cancelReservation() : : void
+rescheduleReservation(newDates: DateRange) : : void
+extendStay(extraHours: int) : : void
"reserves"
"makes advance"
"makes balance"
"triggers"
"uses"
"generates"
Room
+String roomID
+String type
+Double pricePerNight
+Boolean availability
+checkAvailability(dates: DateRange) : : Boolean
Payment
+String paymentID
+Double amount
+PaymentStatus status
+Date paymentDate
+String paymentMethod
+processPayment(amount: Double) : : PaymentStatus
Notification
+String notificationID
+String message
+String recipientEmail
+String recipientPhone
+sendEmail(message: String, recipientEmail: String) : : void
+sendSMS(message: String, recipientPhone: String) : : void
AvailabilityChecker
+String checkerID
+checkRoomAvailability(dates: DateRange) : : Room[]
Invoice
+String invoiceID
+String reservationID
+Double totalAmount
+PaymentStatus paymentStatus
+generateInvoice() : : void
+sendInvoice() : : void
"depends on"
PaymentGateway
+String gatewayID
+processPayment(amount: Double, paymentMethod: String) : : PaymentStatus

Customer:

- This class includes the personal details of the customers, such as customerID, name, email, and phone. The customer can have multiple reservations, establishing a one-to-many association with the Reservation class. This relationship implies that a Customer "HAS" Reservations.
- Key methods include functionalities like makeReservation(), cancelReservation(), rescheduleReservation(), and makePayment().

HotelStaff:

- The HotelStaff class stores information about hotel employees such as staffID, name, and role. It has a one-to-many association with the Reservation class, meaning a staff member can manage multiple reservations. This association indicates that HotelStaff "MANAGES" Reservations.
- Key methods include createReservation(), markCheckIn(), markCheckOut(), and processPayment().

Reservation:

- The Reservation class contains key details related to booking, such as reservationID, checkInDate, checkOutDate, status, and room. It is associated with both Customer and HotelStaff through a many-to-one relationship, representing that Reservations "BELONG TO" Customers and Hotel Staff. It also has a one-to-one relationship with the Room class, indicating each reservation is linked to a specific room.
- Main methods include confirmReservation(), cancelReservation(), and extendStay().

Room:

- The Room class encapsulates the properties of a hotel room, including roomID, type, pricePerNight, and availability. It is associated with the Reservation class in a one-to-one relationship, which signifies that a Room "IS BOOKED IN" a Reservation.
- The key method in this class is checkAvailability().

Payment:

- The Payment class represents the payment details related to the reservation process. Attributes include paymentID, amount, status, and paymentMethod. It has an association with the Reservation class and can initiate payments via external payment gateways.
- The key method is processPayment().

Notification:

- This class deals with sending notifications (emails and SMS) to the customer when reservations are confirmed or canceled. It is associated with the Reservation class and includes attributes such as notificationID, message, recipientEmail, and recipientPhone.
- The methods include sendEmail() and sendSMS().

Invoice:

- The Invoice class stores the payment summary for each reservation, with attributes like invoiceID, totalAmount, and paymentStatus. It is linked to Reservation and generates and sends invoices after payment confirmation.
- The key methods are generateInvoice() and sendInvoice().

Source Code

```
from datetime import datetime
from typing import List

class DateRange:
    def __init__(self, start_date: datetime, end_date: datetime):
        self.start_date = start_date
        self.end_date = end_date

    def is_valid(self) -> bool:
        """Check if the date range is valid (start date should be before
end date)."""
        return self.start_date < self.end_date
```

```
class Customer:
    """
    Represents a customer in the hotel reservation system.

    Attributes:
        customer_id (str): Unique identifier for the customer.
        name (str): Full name of the customer.
        email (str): Email address of the customer.
        phone (str): Phone number of the customer.
        address (str): Residential address of the customer.
        reservations (List[Reservation]): List of reservations made by the
customer.
    """

    def __init__(self, customer_id: str, name: str, email: str, phone:
str, address: str):
        """Initialize a Customer object with provided details."""
        self.customer_id = customer_id
        self.name = name
        self.email = email
        self.phone = phone
        self.address = address
        self.reservations: List['Reservation'] = []
    def getCustomerID(self):
        """Return the customer's unique identifier."""
        return self.customer_id

    def setCustomerID(self, customer_id):
        """Set the customer's unique identifier."""
        self.customer_id = customer_id

    def getName(self):
        """Return the customer's name."""
        return self.name

    def setName(self, name):
        """Set the customer's name."""
        self.name = name

    def getEmail(self):
        """Return the customer's email address."""
        return self.email

    def setEmail(self, email):
        """Set the customer's email address."""
        self.email = email

    def getPhone(self):
        """Return the customer's phone number."""
        return self.phone

    def setPhone(self, phone):
        """Set the customer's phone number."""
        self.phone = phone

    def getAddress(self):
        """Return the customer's address."""
        return self.address

    def setAddress(self, address):
        """Set the customer's address."""
        self.address = address
    def make_reservation(self, details: 'ReservationDetails') ->
'Reservation':
        """Create a new reservation with the given details.

        Args:
            details (ReservationDetails): Details of the reservation.

        Returns:
            Reservation: The created reservation object.
        """
        pass  # This will be implemented to create and add a new
reservation

    def cancel_reservation(self, reservation_id: str):
        """Cancel a reservation based on the reservation ID.

        Args:
            reservation_id (str): The ID of the reservation to cancel.
        """
        pass  # This will be implemented to cancel the reservation

    def reschedule_reservation(self, reservation_id: str, new_dates:
'DateRange'):
        """Reschedule an existing reservation with new dates.

        Args:
            reservation_id (str): The ID of the reservation to reschedule.
            new_dates (DateRange): The new dates for the reservation.
        """
        pass  # This will be implemented to reschedule the reservation

    def make_payment(self, amount: float) -> 'Payment':
        """Process payment for a reservation.

        Args:
            amount (float): The amount to process for the payment.

        Returns:
            Payment: The payment object created.
        """
        pass  # This will be implemented to handle payment processing


class HotelStaff:
    """
    Represents a hotel staff member.

    Attributes:
        staff_id (str): Unique identifier for the staff member.
        name (str): Full name of the staff member.
        role (str): Role of the staff member in the hotel.
    """

    def __init__(self, staff_id: str, name: str, role: str):
        """Initialize a HotelStaff object with provided details."""
        self.staff_id = staff_id
        self.name = name
        self.role = role

    def get_staff_id(self):
        """Return the staff member's unique identifier."""
        return self.staff_id

    def set_staff_id(self, staff_id):
        """Set the staff member's unique identifier."""
        self.staff_id = staff_id

    def get_name(self):
        """Return the staff member's name."""
        return self.name

    def set_name(self, name):
        """Set the staff member's name."""
        self.name = name

    def get_role(self):
        """Return the staff member's role."""
        return self.role

    def set_role(self, role):
        """Set the staff member's role."""
        self.role = role

    def create_reservation(self, details: 'ReservationDetails') ->
'Reservation':
        """Create a new reservation with the given details.

        Args:
            details (ReservationDetails): Details of the reservation.

        Returns:
            Reservation: The created reservation object.
        """
        pass  # This will be implemented to create a reservation

    def mark_check_in(self, reservation_id: str):
        """Mark a reservation as checked in.

        Args:
            reservation_id (str): The ID of the reservation to check in.
        """
        pass  # This will be implemented to mark check-in

    def mark_check_out(self, reservation_id: str):
        """Mark a reservation as checked out.

        Args:
            reservation_id (str): The ID of the reservation to check out.
        """
        pass  # This will be implemented to mark check-out

    def process_payment(self, amount: float, reservation_id: str) ->
'Payment':
        """Process payment for a specific reservation.

        Args:
            amount (float): The amount to process for the payment.
            reservation_id (str): The ID of the reservation to process
payment for.
```

```
        Returns:
            Payment: The payment object created.
        """
        pass  # This will be implemented to handle payment processing


class Reservation:
    """
    Represents a reservation made by a customer.

    Attributes:
        reservation_id (str): Unique identifier for the reservation.
        customer (Customer): The customer who made the reservation.
        check_in_date (datetime): The check-in date for the reservation.
        check_out_date (datetime): The check-out date for the reservation.
        status (str): The status of the reservation (e.g., Pending).
        room (Room): The room associated with the reservation.
        advance_payment (Payment): The advance payment made for the
reservation.
        balance_payment (Payment): The balance payment for the
reservation.
        taxes (float): The total taxes applicable to the reservation.
        room_cost (float): The total room cost for the reservation.
        total_amount (float): The total amount for the reservation.
    """

    def __init__(self, reservation_id: str, customer: Customer
,check_in_date: datetime, check_out_date: datetime):
        """Initialize a Reservation object with provided details."""
        self.reservation_id = reservation_id
        self.customer: 'Customer' = customer
        self.check_in_date = check_in_date
        self.check_out_date = check_out_date
        self.status = 'Pending'  # ReservationStatus can be an Enum
        self.room: 'Room' = None
        self.advance_payment: 'Payment' = None
        self.balance_payment: 'Payment' = None
        self.taxes: float = 0.0
        self.room_cost: float = 0.0
        self.total_amount: float = 0.0
```

```python
def get_reservation_id(self):
    """Return the reservation's unique identifier."""
    return self.reservation_id

def set_reservation_id(self, reservation_id):
    """Set the reservation's unique identifier."""
    self.reservation_id = reservation_id

def get_check_in_date(self):
    """Return the check-in date of the reservation."""
    return self.check_in_date

def set_check_in_date(self, check_in_date):
    """Set the check-in date of the reservation."""
    self.check_in_date = check_in_date

def get_check_out_date(self):
    """Return the check-out date of the reservation."""
    return self.check_out_date

def set_check_out_date(self, check_out_date):
    """Set the check-out date of the reservation."""
    self.check_out_date = check_out_date

def get_status(self):
    """Return the status of the reservation."""
    return self.status

def set_status(self, status):
    """Set the status of the reservation."""
    self.status = status

def get_room(self):
    """Return the room associated with the reservation."""
    return self.room

def set_room(self, room):
    """Set the room associated with the reservation."""
    self.room = room
```

```
def get_advance_payment(self):
    """Return the advance payment made for the reservation."""
    return self.advance_payment

def set_advance_payment(self, advance_payment):
    """Set the advance payment made for the reservation."""
    self.advance_payment = advance_payment

def get_balance_payment(self):
    """Return the balance payment for the reservation."""
    return self.balance_payment

def set_balance_payment(self, balance_payment):
    """Set the balance payment for the reservation."""
    self.balance_payment = balance_payment

def get_taxes(self):
    """Return the total taxes applicable to the reservation."""
    return self.taxes

def set_taxes(self, taxes):
    """Set the total taxes applicable to the reservation."""
    self.taxes = taxes

def get_room_cost(self):
    """Return the total room cost for the reservation."""
    return self.room_cost

def set_room_cost(self, room_cost):
    """Set the total room cost for the reservation."""
    self.room_cost = room_cost

def get_total_amount(self):
    """Return the total amount for the reservation."""
    return self.total_amount

def set_total_amount(self, total_amount):
    """Set the total amount for the reservation."""
    self.total_amount = total_amount
```

```python
    def confirm_reservation(self):
        """Confirm the reservation."""
        pass  # This will be implemented to confirm the reservation

    def cancel_reservation(self):
        """Cancel the reservation."""
        pass  # This will be implemented to cancel the reservation

    def reschedule_reservation(self, new_dates: 'DateRange'):
        """Reschedule the reservation with new dates.

        Args:
            new_dates (DateRange): The new dates for the reservation.
        """
        pass  # This will be implemented to reschedule the reservation

    def extend_stay(self, extra_hours: int):
        """Extend the stay by additional hours.

        Args:
            extra_hours (int): The number of extra hours to extend the
stay.
        """
        pass  # This will be implemented to extend the stay


class Room:
    """
    Represents a hotel room.

    Attributes:
        room_id (str): Unique identifier for the room.
        type (str): Type of the room (e.g., Single, Double).
        price_per_night (float): Price per night for the room.
        availability (bool): Availability status of the room.
    """

    def __init__(self, room_id: str, room_type: str, price_per_night:
float, availability: bool):
        """Initialize a Room object with provided details."""
        self.room_id = room_id
        self.type = room_type
        self.price_per_night = price_per_night
        self.availability = availability

    def get_room_id(self):
        """Return the room's unique identifier."""
        return self.room_id

    def set_room_id(self, room_id):
        """Set the room's unique identifier."""
        self.room_id = room_id

    def get_type(self):
        """Return the room type."""
        return self.type

    def set_type(self, room_type):
        """Set the room type."""
        self.type = room_type

    def get_price_per_night(self):
        """Return the price per night for the room."""
        return self.price_per_night

    def set_price_per_night(self, price_per_night):
        """Set the price per night for the room."""
        self.price_per_night = price_per_night

    def get_availability(self):
        """Return the availability of the room."""
        return self.availability

    def set_availability(self, availability):
        """Set the availability of the room."""
        self.availability = availability

    def check_availability(self, dates: 'DateRange') -> bool:
        """Check if the room is available for the given dates."""
        pass  # This will be implemented to check room availability


class Payment:
    """
    Represents a payment.

    Attributes:
        payment_id (str): Unique identifier for the payment.
        amount (float): Value of the payment.
        status (str): Status of the payment (e.g., Pending).
        payment_date (datetime): Date and time of the payment.
        payment_method (str): Method used for the payment (e.g., Credit
Card).
        payment_note (str): Additional note or details about the payment.
    """
    def __init__(self, payment_id: str, amount: float, status: str,
payment_date: datetime, payment_method: str, payment_note: str):
        """Initialize a Payment object with provided details."""
        self.payment_id = payment_id
        self.amount = amount
        self.status = status  # PaymentStatus can be an Enum
        self.payment_date = payment_date
        self.payment_method = payment_method
        self.payment_note = payment_note

    def get_payment_id(self):
        """Return the payment identifier."""
        return self.payment_id

    def set_payment_id(self, payment_id):
        """Set the payment identifier."""
        self.payment_id = payment_id

    def get_amount(self):
        """Return the payment amount."""
        return self.amount

    def set_amount(self, amount):
        """Set the payment amount."""
        self.amount = amount

    def process_payment(self, amount: float) -> str:
        """Process the payment and return the payment status."""
        pass  # This will be implemented to process the payment

class Invoice:
    """
    Represents an invoice.

    Attributes:
        invoice_id (str): Unique identifier for the invoice.
        reservation (Reservation): The reservation associated with the
invoice.
        total_amount (float): Total amount of the invoice.
        payment_status (str): Status of the payment (e.g., Pending).
    """
    def __init__(self, invoice_id: str, reservation: Reservation,
total_amount: float, payment_status: str):
        """Initialize an Invoice object with provided details."""
        self.invoice_id = invoice_id
        self.reservation: 'Reservation' = reservation
        self.total_amount = total_amount
        self.payment_status = payment_status  # PaymentStatus can be an
Enum

    def get_invoice_id(self):
        """Return the invoice identifier."""
        return self.invoice_id

    def set_invoice_id(self, invoice_id):
        """Set the invoice identifier."""
        self.invoice_id = invoice_id

    def get_reservation(self):
        """Return the reservation."""
        return self.reservation

    def set_reservation(self, reservation):
        """Set the reservation."""
```

```python
        self.reservation = reservation

    def get_total_amount(self):
        """Return the total amount"""
        return self.total_amount

    def set_total_amount(self, total_amount):
        """Set the total amount."""
        self.total_amount = total_amount

    def get_payment_status(self):
        """Return the payment status"""
        return self.payment_status

    def set_payment_status(self, payment_status):
        """Set the payment status."""
        self.payment_status = payment_status

    def generate_invoice(self):
        """Generate an invoice for the reservation."""
        print("Your Reservation Is Confirmed")
        print("Thank you for your reservation. Please print your hotel
receipt and show it at checkin\n")

        print(f"Your Name: {self.reservation.customer.name}")
        print(f"Your Email: {self.reservation.customer.email}")
        print(f"Priceline Trip Number:
{self.reservation.advance_payment.payment_id}")
        print(f"Hotel Confirmation Number:
{self.reservation.reservation_id}\n")

        print("Comfort Inn & Suites Los Alamos")
        print(f"{self.reservation.customer.address}")
        print(f"Phone {self.reservation.customer.phone}")
        print(f"{self.reservation.room.room_id}:
${self.reservation.customer.name}")
        print(f"Check-In: {self.reservation.check_in_date}")
        print(f"Check-Out: {self.reservation.check_out_date}")
        print(f"Number of Nights: {(self.reservation.check_out_date -
self.reservation.check_in_date).days }")
```

```
        print(f"Number of Rooms: 1")
        print(f"Room Type: {self.reservation.room.type}")

        print("Summary of Charges")
        print(f"Billing Name: {self.reservation.customer.name}")
        print(f"{self.reservation.advance_payment.payment_method}:
{self.reservation.advance_payment.payment_note}")
        print(f"Room Cost: {self.reservation.room.price_per_night}")
        print(f"Rooms: 1")
        print(f"Nights: {(self.reservation.check_out_date -
self.reservation.check_in_date).days }")
        print(f"Room Subtotal: ${self.reservation.room_cost:.2f}")
        print(f"Taxes and Fees: ${self.reservation.taxes:.2f}")
        print(f"Total Charges: ${self.reservation.total_amount:.2f}")
        print("prices are in US dollars")


    def send_invoice(self):
        """Send the generated invoice to the customer."""
        pass  # Implement invoice sending logic


class Notification:
    def __init__(self, notification_id: str, message: str,
recipient_email: str, recipient_phone: str):
        self.notification_id = notification_id
        self.message = message
        self.recipient_email = recipient_email
        self.recipient_phone = recipient_phone

    def send_email(self):
        """Send an email notification."""
        pass  # Implement email sending logic

    def send_sms(self):
        """Send an SMS notification."""
        pass  # Implement SMS sending logic
```

```
def main():
    # Hotel rooms
    room1 = Room("Room001", "2 Queen Beds No Smoking Desk/Safe/Coffee 
Maker in Room/Hair Dryer", 89.95, True)
    room2 = Room("Room002", "2 Queen Beds No Smoking Desk/Safe/Coffee 
Maker in Room/Hair Dryer", 89.95, True)
    room3 = Room("Room003", "2 Queen Beds No Smoking Desk/Safe/Coffee 
Maker in Room/Hair Dryer", 89.95, True)
    room4 = Room("Room004", "2 Queen Beds No Smoking Desk/Safe/Coffee 
Maker in Room/Hair Dryer", 89.95, True)
    room5 = Room("Room005", "2 Queen Beds No Smoking Desk/Safe/Coffee 
Maker in Room/Hair Dryer", 89.95, True)

    # Create a customer
    customer = Customer("C001", "Ted Vera", "tedivera@mac.com", "505-661-
1110", "2455 Trinty Drive, Los Alamos, NM 87544")

    # Create a reservation
    reservation = Reservation("52523687", customer, datetime(2010, 8, 22), 
datetime(2010, 8, 24))

    # Check whether the room is available or not
    isRoomAvailable = room1.check_availability(DateRange(datetime(2023, 
10, 10), datetime(2023, 10, 15)))
    # Let's set room availability to true manually
    isRoomAvailable = True

    if isRoomAvailable:
        room1.set_availability(False)
        reservation.room = room1
        reservation.confirm_reservation()

        taxes = 21.58
        room_cost = reservation.room.price_per_night * 
(reservation.check_out_date - reservation.check_in_date).days
        reservation.set_taxes(taxes)
        reservation.set_room_cost(room_cost)
        reservation.set_total_amount(room_cost+taxes)
    # Let's set reservation status to Confirmed manually
    reservation.status = 'Confirmed'
    # Create an adavance payment
    if reservation.status == 'Confirmed':
        advance_payment_amount = reservation.total_amount * 0.2
        payment = Payment("15549850358", advance_payment_amount,
"Pending", datetime.now(), "Credit Card", "Mastercard (ending in 1904)")
        payment.process_payment(advance_payment_amount)
        reservation.set_advance_payment(payment)

        # Send a notification to the user
        notification = Notification("No123456789", "Your reservation is
confirmed", "tedivera@mac.com", "505-661-1110")
        notification.send_email()
        notification.send_sms()

    invoice = Invoice("INV001", reservation, reservation.total_amount,
"Paid")
    invoice.generate_invoice()
    invoice.send_invoice()
```

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

Summary of Learnings:

The assignment involved translating a real-world scenario into structured system models and implementing the corresponding functionality in Python. The process began with the creation of UML use case diagrams to represent user interactions, followed by detailed use case description tables to outline scenarios and workflows. This approach ensured a comprehensive understanding of the system requirements and how to capture them visually.

Subsequently, UML class diagrams were developed to define the relationships between system entities, with particular attention to associations, dependencies, and multiplicities. These diagrams were crucial in structuring the system's components, aligning with the scenario's requirements.

Finally, the system design was implemented using Python. The classes and methods were created based on the UML diagrams, incorporating key functionalities such as reservations, payments, and notifications. This phase enhanced proficiency in object-oriented programming and addressing challenges related to class dependencies, output formatting, and debugging.

Overall, the assignment provided a valuable opportunity to bridge the gap between system design and implementation, reinforcing proficiency in system modeling, diagramming, and coding, with practical insights into real-world application development.