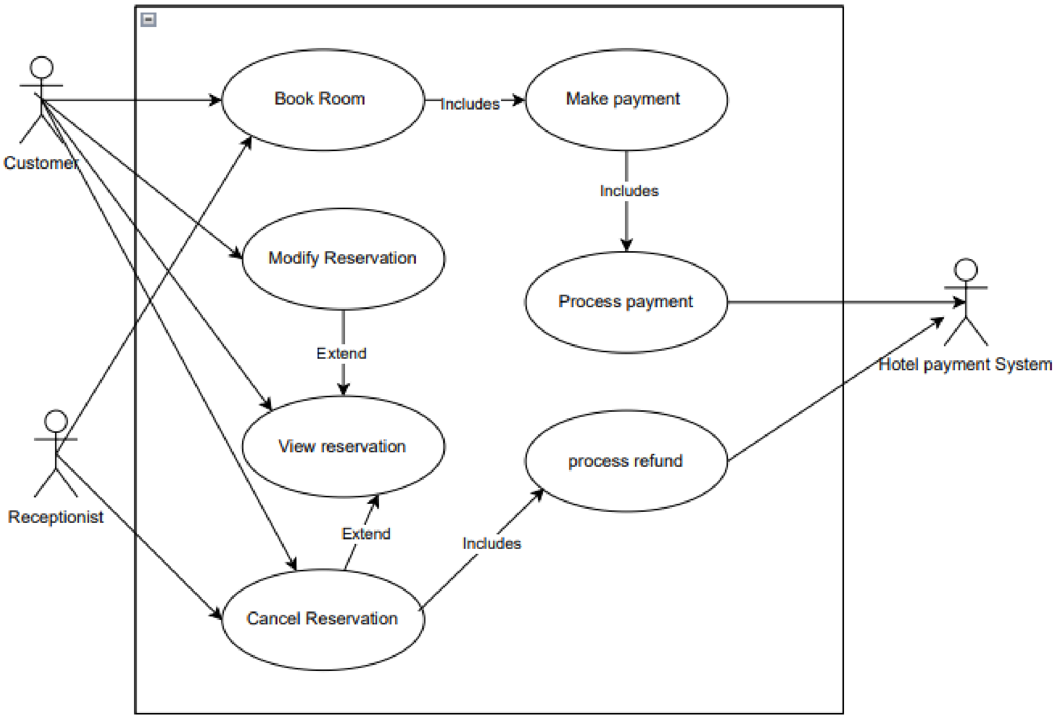
Hotel Reservation System



Description: This diagram illustrates principal activity of the Customer, Receptionist, and Payment System and the use cases concerned with hotel bookings and hotel reservations. The primary use cases are:

1. Book Room: This use case enables a customer or a receptionist to make a booking of a hotel room.
2. View Reservation: An existing reservation can be viewed by the customer or the receptionist.
3. Modify Reservation: Some actions like update, meaning a change in data such as changing the dates are possible by the customer.
4. Cancel Reservation: This use case permits the customer or the receptionist to cancel a reservation.
5. Make Payment: This is part of booking process that need a customer to input card details.
6. Process Payment: When payment information is provided, the payment gateway acts on the payment.
7. Process Refund: In case of cancellation, the payment system takes into account necessary refunds.

Use Case:	**Book a Room**
Actors:	**1. Customer: The person who wants to book a hotel room. 2. Receptionist: Hotel staff handling the room booking process. 3. Hotel Payment System: Processes the payment for the room booking.**
Trigger:	**A customer wants to book a room through the hotel's reservation system.**
Preconditions:	**- The customer can use the website or contact a receptionist. - Rooms are available in the hotel. - The customer has a valid payment method.**
Main Scenario:	**1. The customer visits the hotel website or calls the receptionist. 2. The customer selects an available room. 3. The customer provides booking details (e.g., check-in/check-out). 4. The customer enters personal details (e.g., name, contact). 5. The customer chooses extra services. 6. The customer proceeds to payment. 7. Payment is processed and booking acknowledgment is communicated.**
Exception:	**1a. If no rooms are available, a message informs the customer. 2a. If payment is declined, the customer is asked to try another method. 3a. If the system is down, the customer is asked to try again later.**

Use Case:	**Modify Reservation**
Actors:	**1. Customer** **2. Receptionist**
Trigger:	**The customer wants to make changes to an existing reservation.**
Preconditions:	**- The customer has a valid existing reservation.** **- Changes are allowed within the hotel's policy.**
Main Scenario:	**1. The customer accesses their reservation online or contacts the receptionist.** **2. The customer specifies the desired changes (e.g., room type, check-in date).** **3. The system checks the availability of requested modifications.** **4. The system updates the reservation and confirms the changes.**
Exception:	**1a. If the requested changes cannot be accommodated, the customer is informed.** **2a. If the system is down, the customer is asked to try again later.**

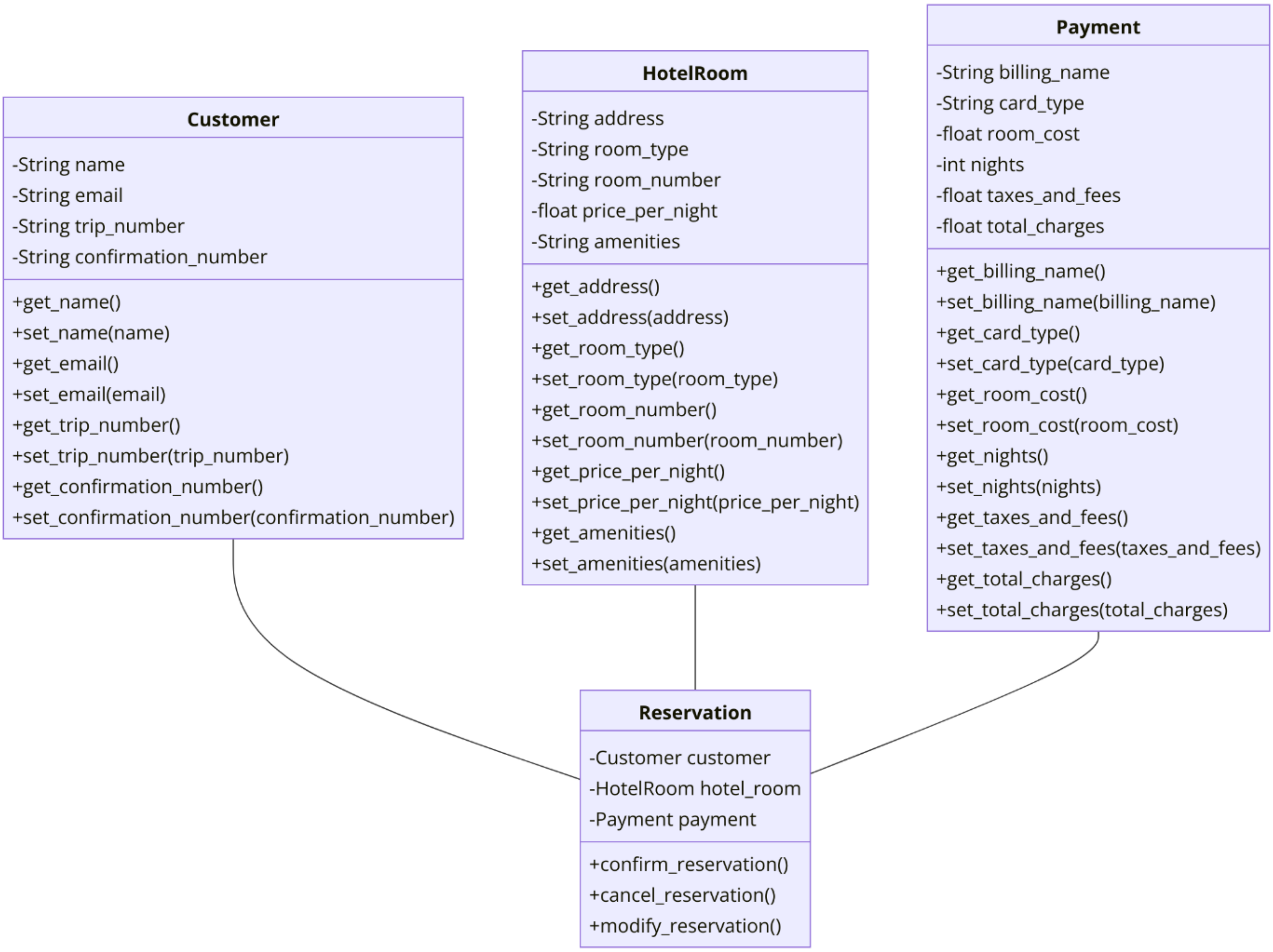
Use Case:	**View Reservation**
Actors:	**1. Customer** **2. Receptionist**
Trigger:	**The customer wants to view the details of their reservation.**
Preconditions:	**- The customer has an existing reservation.**
Main Scenario:	**1. The customer accesses their account or contacts the receptionist.** **2. The system retrieves the reservation details.** **3. The customer reviews the reservation information (e.g., room type, dates).**
Exception:	**1a. If the reservation does not exist, the customer is informed.**

Use Case:	**Cancel Reservation**
Actors:	**1. Customer** **2. Receptionist**
Trigger:	**The customer wants to cancel their reservation.**
Preconditions:	**- The customer has an existing reservation.** **- The cancellation is within the allowed policy timeframe.**
Main Scenario:	**1. The customer accesses the reservation system or contacts the receptionist.** **2. The system retrieves the reservation details.** **3. The customer requests to cancel the reservation.** **4. The system processes the cancellation and updates the status.** **5. Refund processing is initiated if applicable.**
Exception:	**1a. If the cancellation is outside the allowed timeframe, the customer is informed.** **2a. If the system is down, the customer is asked to try again later.**

Use Case:	**Make Payment**
Actors:	**1. Customer** **2. Hotel Payment System**
Trigger:	**The customer needs to make a payment for a reservation.**
Preconditions:	**- The customer has selected a room.** **- The customer has a valid payment method.**
Main Scenario:	**1. The customer proceeds to the payment page after booking a room.** **2. The system prompts the customer to enter payment details.** **3. The system processes the payment.** **4. Payment confirmation is displayed to the customer**
Exception:	**1a. If the payment is declined, the customer is asked to use a different method.**

Use Case:	**Process Payment**
Actors:	**1. Hotel Payment System**
Trigger:	**A payment is initiated by the customer for booking.**
Preconditions:	**- The customer has entered valid payment details.**
Main Scenario:	**1. The hotel payment system validates the payment information. 2. The system processes the transaction. 3. The customer receives a confirmation of the payment.**
Exception:	**1a. If the payment fails, the system notifies the customer.**

Use Case:	**Process Refund**
Actors:	**1. Hotel Payment System**
Trigger:	**A refund needs to be processed for a canceled reservation.**
Preconditions:	**- The reservation has been canceled within the refund policy.**
Main Scenario:	**1. The system retrieves the reservation and calculates the refund amount. 2. The payment system processes the refund. 3. The customer is notified that the refund has been processed.**
Exception:	**1a. If the refund fails, the system informs the customer and retries.**

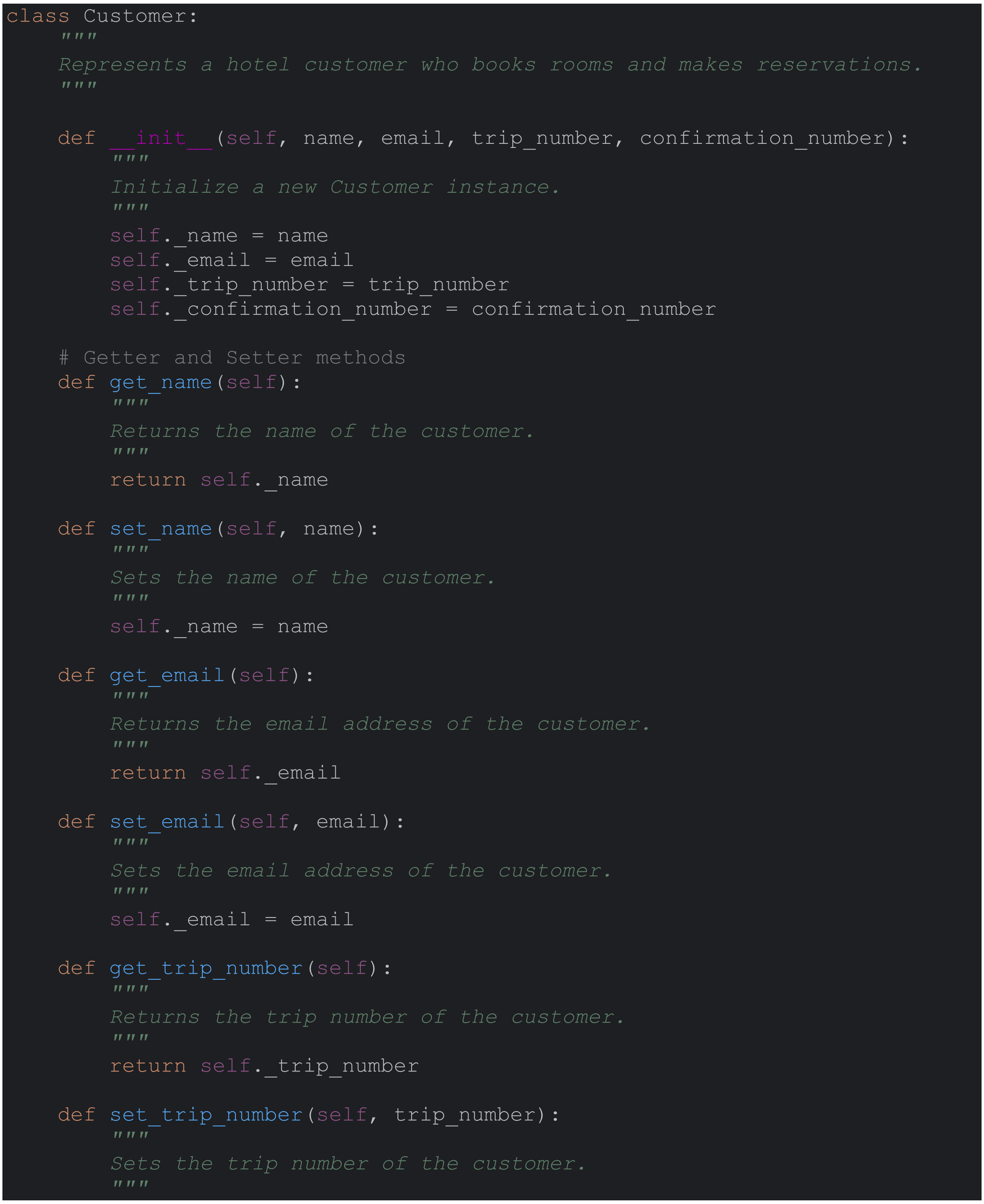


Description: This class diagram represents the key components of a hotel reservation system, focusing on the interactions between several classes:

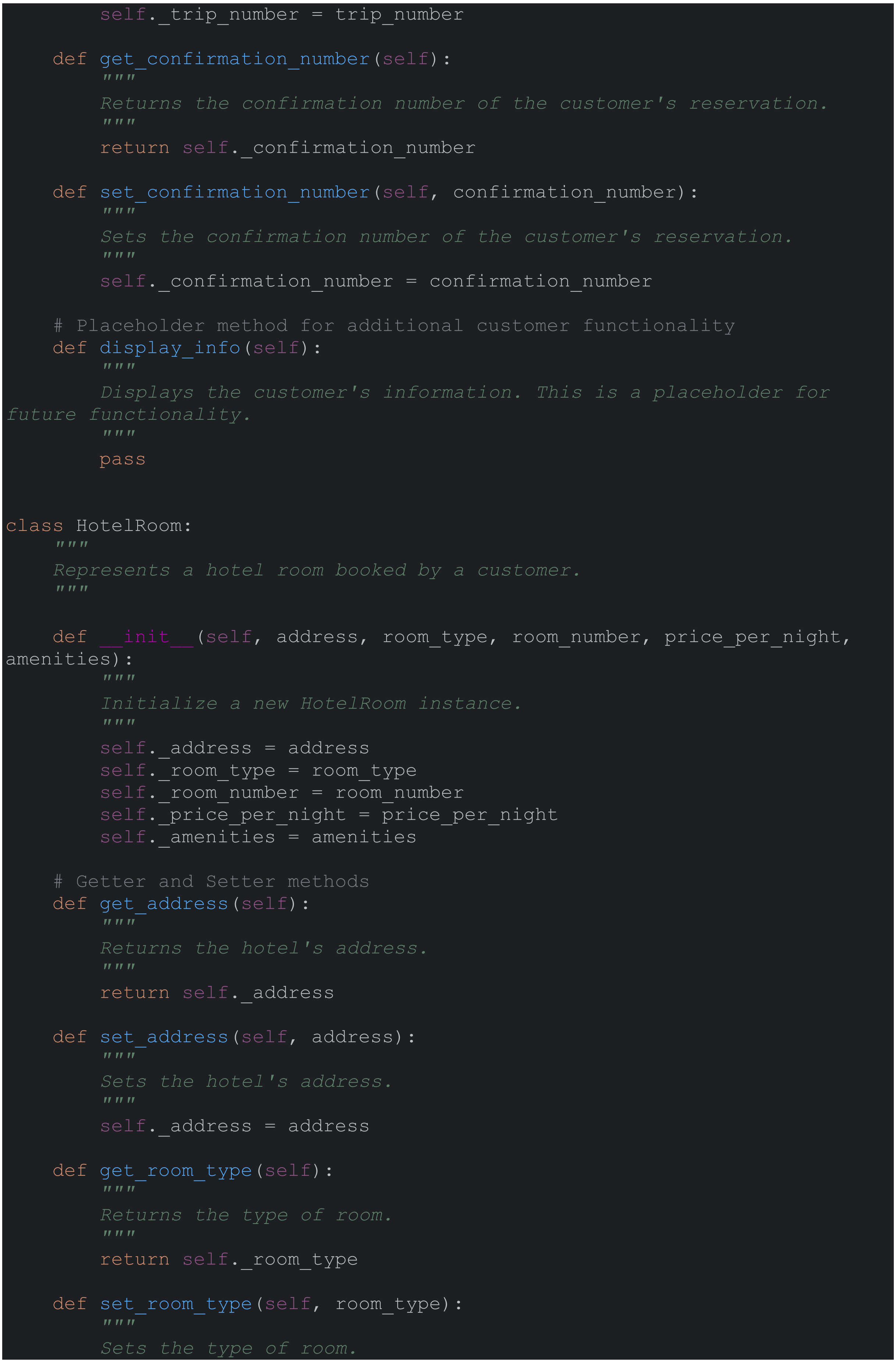
1. Customer Class: Stores identifiable information about the customer under attributes that include the name, the email, and the trip, number, and confirmation number. It comprises getter and setter methods for each of the attributes.
2. Hotel Room Class: Comprises data about the specific hotel room, such as its address, the type of space, the number of rooms, the price per night, and the services provided. It also offers getter and setter methods for every attribute.
3. Payment Class: Handles the payment details which are billing name, the type of card used, the cost of the room, the number of nights the client spent, and the taxes and fees as well as the total charges. It has methods to read and modify these values as required.
4. Reservation Class: This class relates to the Customer, Hotel Room, and Payment classes. It has features for checking status, modifying, or even deleting a reservation. This class serves as the middleman for the reservation system.

The connections between the classes show that a Reservation is created by a Customer for a specific hotel room and involves Payment. Every class has its role in the reservation process, which also explains how the system is organized schematically.

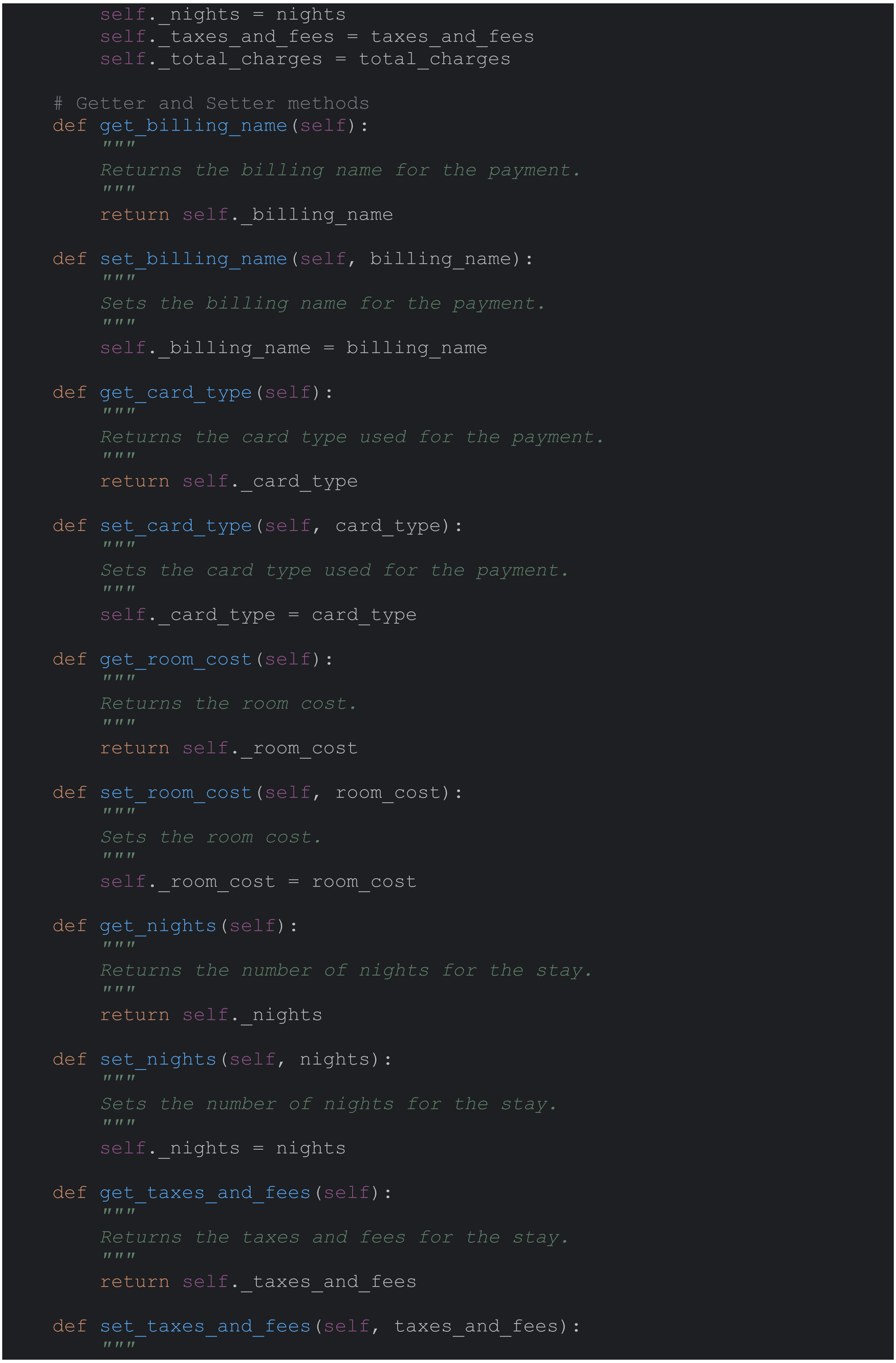
Code:



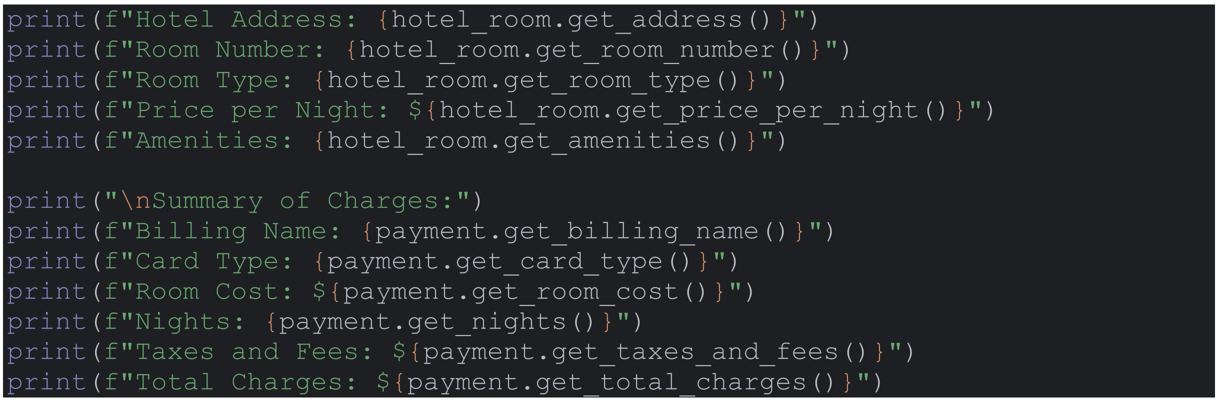
```
class Customer:
    """
    Represents a hotel customer who books rooms and makes reservations.
    """

    def __init__(self, name, email, trip_number, confirmation_number):
        """
        Initialize a new Customer instance.
        """
        self._name = name
        self._email = email
        self._trip_number = trip_number
        self._confirmation_number = confirmation_number

    # Getter and Setter methods
    def get_name(self):
        """
        Returns the name of the customer.
        """
        return self._name

    def set_name(self, name):
        """
        Sets the name of the customer.
        """
        self._name = name

    def get_email(self):
        """
        Returns the email address of the customer.
        """
        return self._email

    def set_email(self, email):
        """
        Sets the email address of the customer.
        """
        self._email = email

    def get_trip_number(self):
        """
        Returns the trip number of the customer.
        """
        return self._trip_number

    def set_trip_number(self, trip_number):
        """
        Sets the trip number of the customer.
        """
```



```python
        self._trip_number = trip_number

    def get_confirmation_number(self):
        """
        Returns the confirmation number of the customer's reservation.
        """
        return self._confirmation_number

    def set_confirmation_number(self, confirmation_number):
        """
        Sets the confirmation number of the customer's reservation.
        """
        self._confirmation_number = confirmation_number

    # Placeholder method for additional customer functionality
    def display_info(self):
        """
        Displays the customer's information. This is a placeholder for
future functionality.
        """
        pass


class HotelRoom:
    """
    Represents a hotel room booked by a customer.
    """

    def __init__(self, address, room_type, room_number, price_per_night,
amenities):
        """
        Initialize a new HotelRoom instance.
        """
        self._address = address
        self._room_type = room_type
        self._room_number = room_number
        self._price_per_night = price_per_night
        self._amenities = amenities

    # Getter and Setter methods
    def get_address(self):
        """
        Returns the hotel's address.
        """
        return self._address

    def set_address(self, address):
        """
        Sets the hotel's address.
        """
        self._address = address

    def get_room_type(self):
        """
        Returns the type of room.
        """
        return self._room_type

    def set_room_type(self, room_type):
        """
        Sets the type of room.
        """
        self._room_type = room_type

    def get_room_number(self):
        """
        Returns the room number.
        """
        return self._room_number

    def set_room_number(self, room_number):
        """
        Sets the room number.
        """
        self._room_number = room_number

    def get_price_per_night(self):
        """
        Returns the price per night of the room.
        """
        return self._price_per_night

    def set_price_per_night(self, price_per_night):
        """
        Sets the price per night of the room.
        """
        self._price_per_night = price_per_night

    def get_amenities(self):
        """
        Returns the list of amenities available in the room.
        """
        return self._amenities

    def set_amenities(self, amenities):
        """
        Sets the list of amenities available in the room.
        """
        self._amenities = amenities

    # Placeholder method for checking room availability
    def display_room_info(self):
        """
        Displays the room's information. This is a placeholder for future
functionality.
        """
        pass


class Payment:
    """
    Represents payment information for a hotel booking.
    """

    def __init__(self, billing_name, card_type, room_cost, nights,
taxes_and_fees, total_charges):
        """
        Initialize a new Payment instance.
        """
        self._billing_name = billing_name
        self._card_type = card_type
        self._room_cost = room_cost
```



```python
        self._nights = nights
        self._taxes_and_fees = taxes_and_fees
        self._total_charges = total_charges

    # Getter and Setter methods
    def get_billing_name(self):
        """
        Returns the billing name for the payment.
        """
        return self._billing_name

    def set_billing_name(self, billing_name):
        """
        Sets the billing name for the payment.
        """
        self._billing_name = billing_name

    def get_card_type(self):
        """
        Returns the card type used for the payment.
        """
        return self._card_type

    def set_card_type(self, card_type):
        """
        Sets the card type used for the payment.
        """
        self._card_type = card_type

    def get_room_cost(self):
        """
        Returns the room cost.
        """
        return self._room_cost

    def set_room_cost(self, room_cost):
        """
        Sets the room cost.
        """
        self._room_cost = room_cost

    def get_nights(self):
        """
        Returns the number of nights for the stay.
        """
        return self._nights

    def set_nights(self, nights):
        """
        Sets the number of nights for the stay.
        """
        self._nights = nights

    def get_taxes_and_fees(self):
        """
        Returns the taxes and fees for the stay.
        """
        return self._taxes_and_fees

    def set_taxes_and_fees(self, taxes_and_fees):
        """
        Sets the taxes and fees for the stay.
        """
        self._taxes_and_fees = taxes_and_fees

    def get_total_charges(self):
        """
        Returns the total charges for the stay.
        """
        return self._total_charges

    def set_total_charges(self, total_charges):
        """
        Sets the total charges for the stay.
        """
        self._total_charges = total_charges

    # Placeholder method for processing the payment
    def display_payment_info(self):
        """
        Displays the payment information. This is a placeholder for future
functionality.
        """
        pass

# Example to populate and display the information shown in Figure 1

# Creating customer object
customer = Customer(
    name="Ted Vera",
    email="tedvera@mac.com",
    trip_number="1568450968",
    confirmation_number="52532687"
)

# Creating hotel room object
hotel_room = HotelRoom(
    address="2455 Trinity Drive, Los Alamos, NM, 87544",
    room_type="2 Queen Beds",
    room_number="1",
    price_per_night=89.55,
    amenities="No Smoking/Desk/Safe/Coffee Maker in Room/Hair Dryer"
)

# Creating payment object
payment = Payment(
    billing_name="Ted H Vera",
    card_type="Mastercard (ending in 9604)",
    room_cost=89.55,
    nights=2,
    taxes_and_fees=21.58,
    total_charges=201.48
)

# Displaying the reservation confirmation information
print("Reservation Confirmation:")
print(f"Customer: {customer.get_name()}")
print(f"Email: {customer.get_email()}")
print(f"Trip Number: {customer.get_trip_number()}")
print(f"Hotel Confirmation Number: {customer.get_confirmation_number()}")

print("\nHotel Details:")
```



```python
print(f"Hotel Address: {hotel_room.get_address()}")
print(f"Room Number: {hotel_room.get_room_number()}")
print(f"Room Type: {hotel_room.get_room_type()}")
print(f"Price per Night: ${hotel_room.get_price_per_night()}")
print(f"Amenities: {hotel_room.get_amenities()}")

print("\nSummary of Charges:")
print(f"Billing Name: {payment.get_billing_name()}")
print(f"Card Type: {payment.get_card_type()}")
print(f"Room Cost: ${payment.get_room_cost()}")
print(f"Nights: {payment.get_nights()}")
print(f"Taxes and Fees: ${payment.get_taxes_and_fees()}")
print(f"Total Charges: ${payment.get_total_charges()}")
```

https://github.com/NaserM7/Naser

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.