Hotel Reservation System

Hessa O. Alqamzi

202219944

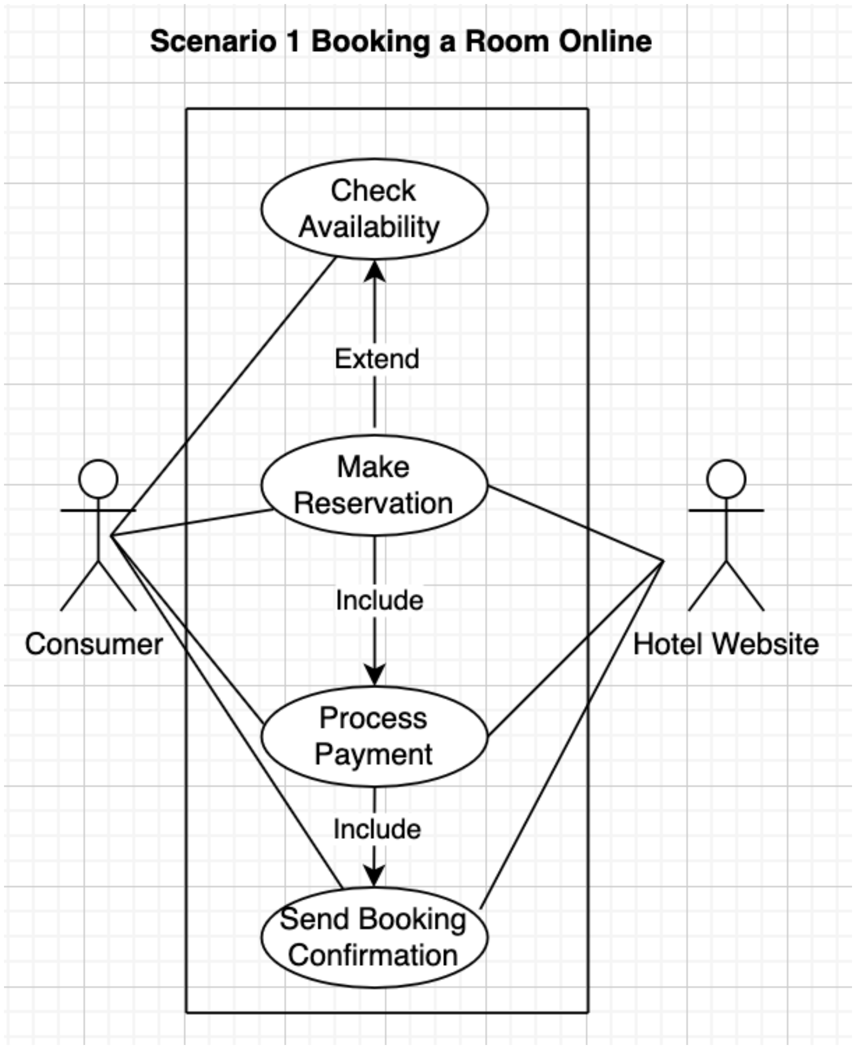
College of Interdisciplinary Studies, Zayed University

Dr. Mathew

September 30, 2024

Scenario 1:

UML Use-Case Diagram:

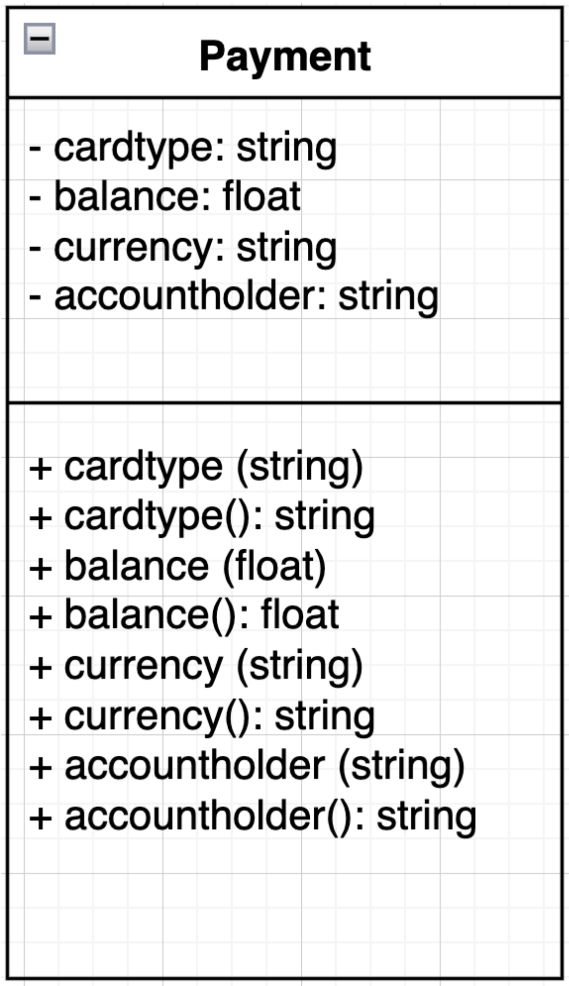


Use-Case Description:

Case	Process Payment
Actors	Consumer Hotel Website
Trigger	The customer initiates payment by entering their bank card information, and the hotel website validates the information.

Precondition	The customer has selected a room and is ready to pay. The customer has provided valid card information. The hotel has verified the room’s availability and confirmed the reservation.
Main Scenario	The customer views available rooms for the desired date. The customer selects a room to reserve. The customer enters their bank card information to complete the reservation. The hotel website verifies the card details and checks if the customer has sufficient funds. The payment is processed and the reservation is confirmed. The customer receives a confirmation of their reservation.
Exceptions	Invalid Card Information: If the card details are incorrect or incomplete, the hotel website displays an error message and prompts the customer to re-enter their information. Insufficient Funds: If the card has insufficient funds, the hotel website notifies the customer and requests a different payment method. Technical Issues: If there are technical problems with the payment gateway or website, the customer is informed of the issue and asked to try again later or contact support.

UML Class Diagram:



UML Class Description:

<u>Class Name:</u> Payment

The Payment class represents a payment method associated with an account. It is designed to handle essential payment information, making it easier to manage transactions and user account details.

<u>Attributes:</u>

1. cardType: string (private)

Description: Represents the type of card being used for payment (e.g., Visa, MasterCard). This attribute helps identify the payment method and may be relevant for processing transactions.

2. balance: float (private)

   Description: Indicates the available balance in the account associated with the payment method. This value is crucial for ensuring that sufficient funds are available for transactions.

3. currency: string (private)

   Description: Specifies the currency in which the payment is made (e.g., USD, EUR). This attribute is important for conversions and ensuring that transactions are processed in the correct currency.

4. accountHolder: string (private)

   Description: Contains the name of the individual or entity that holds the account. This attribute is essential for verification and record-keeping purposes.

<u>Methods:</u>

1. cardType(): string (public)

   Description: A public getter method that returns the type of card. It provides access to the cardType attribute, allowing other parts of the program to retrieve this information.

2. balance(): float (public)

   Description: A public getter method that returns the current available balance. This method is used to check the amount available for transactions.

3. currency(): string (public)

   Description: A public getter method that returns the currency type. It allows other components of the system to access and display the currency associated with the payment.

4. accountHolder(): string (public)

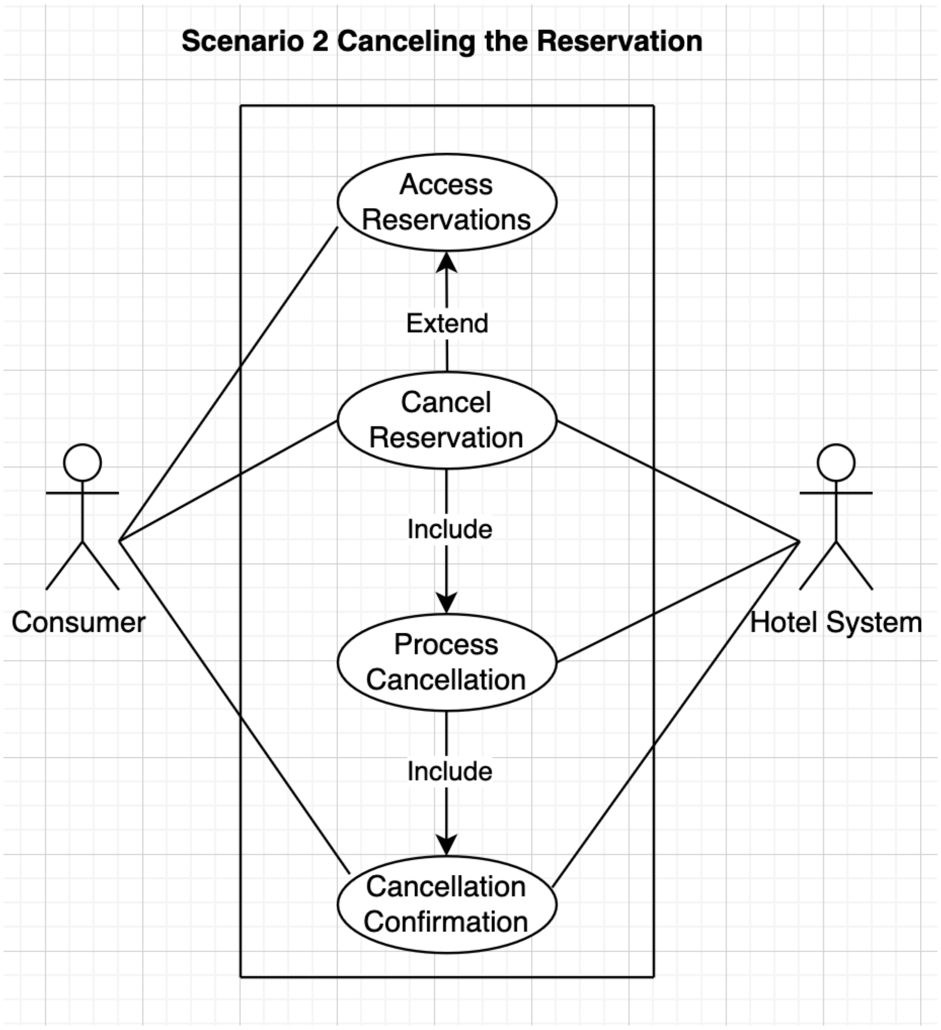
Description: A public getter method that returns the name of the account holder. This method provides access to the accountHolder attribute for verification and record purposes.

Python Class:

```
class Payment:
   """Class to represent a payment method."""

   # Constructor
   def __init__(self, cardtype="", balance=0.0, currency="",
accountholder=""):
       self.__cardtype = cardtype
       self.__balance = balance
       self.__currency = currency
       self.__accountholder = accountholder

   # Setter & Getter for cardtype
   def get_cardtype(self):
       return self.__cardtype

   def set_cardtype(self, cardtype):
       self.__cardtype = cardtype

   # Setter & Getter for balance
   def get_balance(self):
       return self.__balance

   def set_balance(self, balance):
       self.__balance = balance

   # Setter & Getter for currency
   def get_currency(self):
       return self.__currency

   def set_currency(self, currency):
       self.__currency = currency

   # Setter & Getter for accountholder
   def get_accountholder(self):
       return self.__accountholder

   def set_accountholder(self, accountholder):
       self.__accountholder = accountholder

   # Display function
   def __str__(self):
        return f"Card Type: {self.__cardtype}, Balance: {self.__balance},
Currency: {self.__currency}, Account Holder: {self.__accountholder}"


# Creating an Object
payment1 = Payment("Visa", 2500.0, "USD", "John Doe")
print(payment1)
```

Scenario 2:

UML Use-Case Diagram:

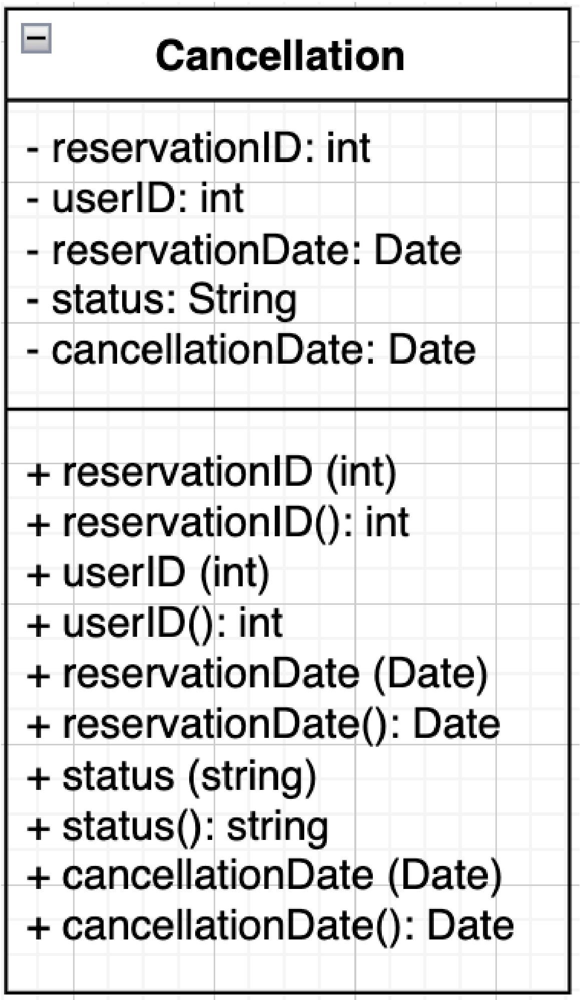


Use-Case Description:

Case	Cancel Reservation
Actors	Consumer Hotel System
Trigger	The consumer requests to cancel their reservation through the hotel system.
Precondition	The consumer has an existing reservation with the hotel. The consumer is logged into their account or provides a valid reservation reference.
Main Scenario	The consumer selects the reservation they wish to cancel. The hotel system verifies the reservation details and checks if the cancellation request meets the hotel's cancellation policy. The hotel system processes the cancellation request and updates the reservation status. The consumer receives a confirmation of the cancellation, and any applicable refund or cancellation fee is processed according to the policy. The reservation is removed from the consumer’s booking list, and the room becomes available for other guests.

Exceptions	Invalid Reservation Details: If the consumer provides incorrect reservation details, the hotel system displays an error message and prompts the consumer to re-enter or verify their information. Cancellation Policy Violation: If the cancellation request does not meet the hotel's policy (e.g., within a non-refundable period), the hotel system notifies the consumer of any applicable penalties or fees. Technical Issues: If there are technical problems with the hotel system, the consumer is informed of the issue and advised to try again later or contact customer support.

UML Class Diagram:


Cancellation
- reservationID: int
- userID: int
- reservationDate: Date
- status: String
- cancellationDate: Date
+ reservationID (int)
+ reservationID(): int
+ userID (int)
+ userID(): int
+ reservationDate (Date)
+ reservationDate(): Date
+ status (string)
+ status(): string
+ cancellationDate (Date)
+ cancellationDate(): Date

UML Class Description:

Class Name: Cancellation

The Cancellation class keeps track of details related to canceling a reservation. It stores important information to manage cancellations effectively.

Attributes:

1. reservationID: int (private)

   Description: A unique number for the reservation being canceled.

2. userID: int (private)

   Description: A unique number for the user who made the reservation.

3. reservationDate: Date (private)

   Description: The date when the reservation was originally made.

4. status: String (private)

   Description: The current status of the reservation, like "Cancelled" or "Pending."

5. cancellationDate: Date (private)

   Description: The date when the cancellation took place.

Methods:

1. reservationID(): int (public)

    Description: Gets the unique number for the reservation.

2. userID(): int (public)

    Description: Gets the unique number for the user.

3. reservationDate(): Date (public)

    Description: Gets the date of the original reservation.

4. status(): String (public)

    Description: Gets the current status of the reservation.

5. cancellationDate(): Date (public)

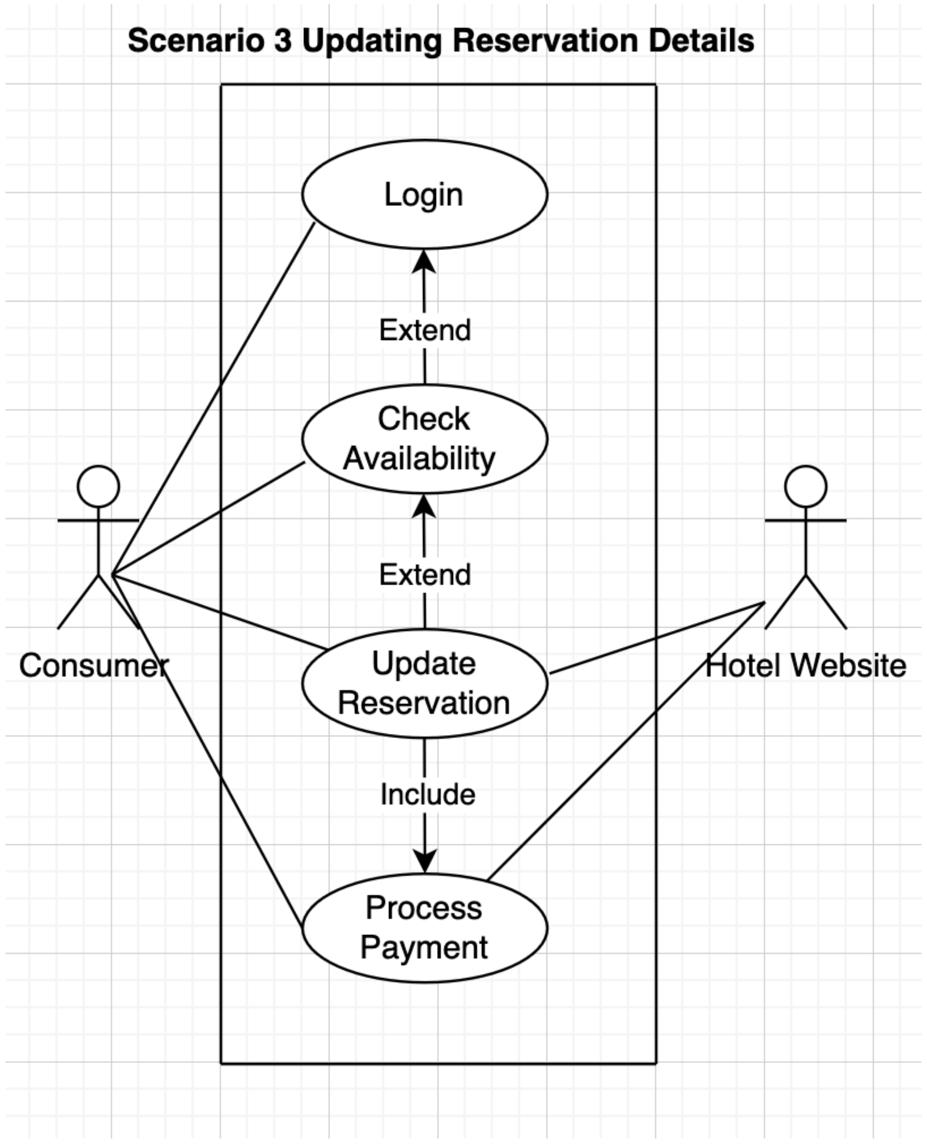
    Description: Gets the date of the cancellation.

Python Class:

```python
from datetime import date


class Cancellation:
    """Class to represent a reservation cancellation."""

    # Constructor
    def __init__(self, reservation_id=0, user_id=0, reservation_date=None,
status="", cancellation_date=None):
        self.__reservation_id = reservation_id
        self.__user_id = user_id
        self.__reservation_date = reservation_date if reservation_date is not
None else date.today()
        self.__status = status
        self.__cancellation_date = cancellation_date if cancellation_date is
not None else date.today()

    # Setter & Getter for reservationID
    def get_reservation_id(self):
        return self.__reservation_id

    def set_reservation_id(self, reservation_id):
        self.__reservation_id = reservation_id
    # Setter & Getter for userID
    def get_user_id(self):
        return self.__user_id

    def set_user_id(self, user_id):
        self.__user_id = user_id

    # Setter & Getter for reservationDate
    def get_reservation_date(self):
        return self.__reservation_date

    def set_reservation_date(self, reservation_date):
        self.__reservation_date = reservation_date

    # Setter & Getter for status
    def get_status(self):
        return self.__status

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

    # Setter & Getter for cancellationDate
    def get_cancellation_date(self):
        return self.__cancellation_date

    def set_cancellation_date(self, cancellation_date):
        self.__cancellation_date = cancellation_date

    # Display function
    def __str__(self):
        return (f"Reservation ID: {self.__reservation_id}, User ID:
{self.__user_id}, "
                f"Reservation Date: {self.__reservation_date}, Status:
{self.__status}, "
                f"Cancellation Date: {self.__cancellation_date}")


# Creating an Object
cancellation1 = Cancellation(101, 202, date(2024, 9, 15), "Cancelled",
date(2024, 9, 20))
print(cancellation1)
```

Scenario 3:

UML Use-Case Diagram:



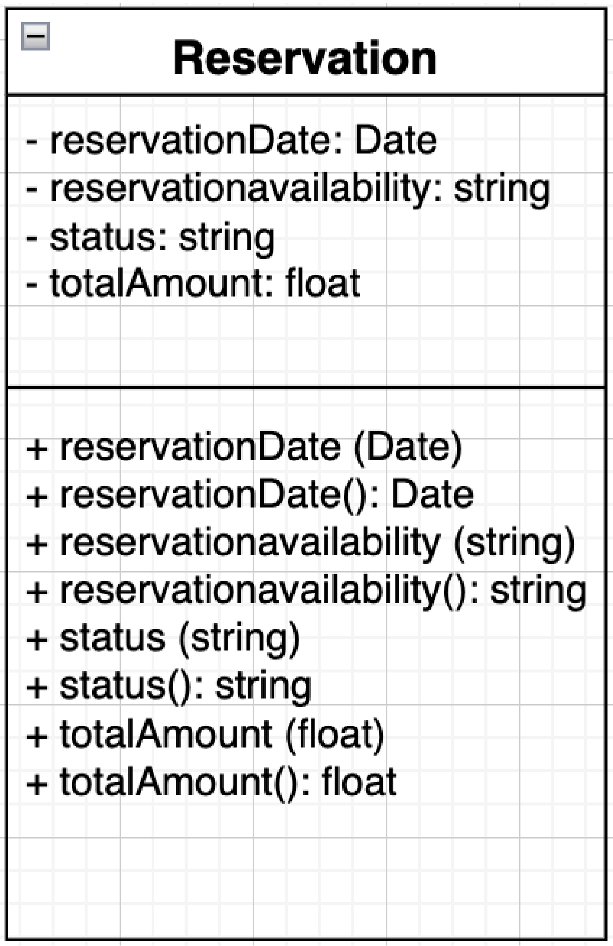
Use-Case Description:

Case	Update Reservation
Actors	Consumer Hotel Website
Trigger	The consumer requests to update their existing reservation through the hotel website.

Precondition	The consumer has an active reservation with the hotel. The consumer is logged into their account or provides a valid reservation reference.
Main Scenario	The consumer logs into their account or accesses the reservation management page. The consumer selects the reservation they wish to update. The hotel website displays the current reservation details and provides options for modification (e.g., changing dates, room type, or adding special requests). The consumer makes the desired updates and submits the changes. The hotel website verifies the updated reservation details and checks for availability or policy constraints. The hotel website processes the updates and confirms the changes to the reservation. The consumer receives a confirmation of the updated reservation, reflecting the new details.
Exceptions	Invalid Reservation Details: If the consumer provides incorrect reservation details or references, the hotel website displays an error message and prompts for correct information. Unavailability of Updated Options: If the updated request (e.g., new dates or room type) is not available, the hotel website informs the consumer and may offer alternative options or suggest re-selecting available options. Policy Violation: If the requested updates do not adhere to the hotel's policies (e.g., changing dates outside allowed timeframes), the website notifies the consumer of any restrictions or additional fees.

	Technical Issues: If there are technical problems with the hotel website, the consumer is informed of the issue and advised to try again later or contact customer support.

UML Class Diagram:



UML Class Description:

Class Name: Reservation

The Reservation class holds information about a reservation. It tracks important details such as dates, availability, status, and costs.

Attributes:

1. reservationDate: Date (private)

   Description: The date when the reservation is made.

2. reservationAvailability: string (private)

   Description: Indicates if the reservation is available or not.

3. status: string (private)

   Description: The current status of the reservation (e.g., "Confirmed," "Pending").

4. totalAmount: float (private)

   Description: The total cost of the reservation.

Methods:

1. reservationDate(): Date (public)

   Description: Gets the date of the reservation.

2. reservationAvailability(): string (public)

   Description: Gets the availability status of the reservation.

3. status(): string (public)

   Description: Gets the current status of the reservation.

4. totalAmount(): float (public)

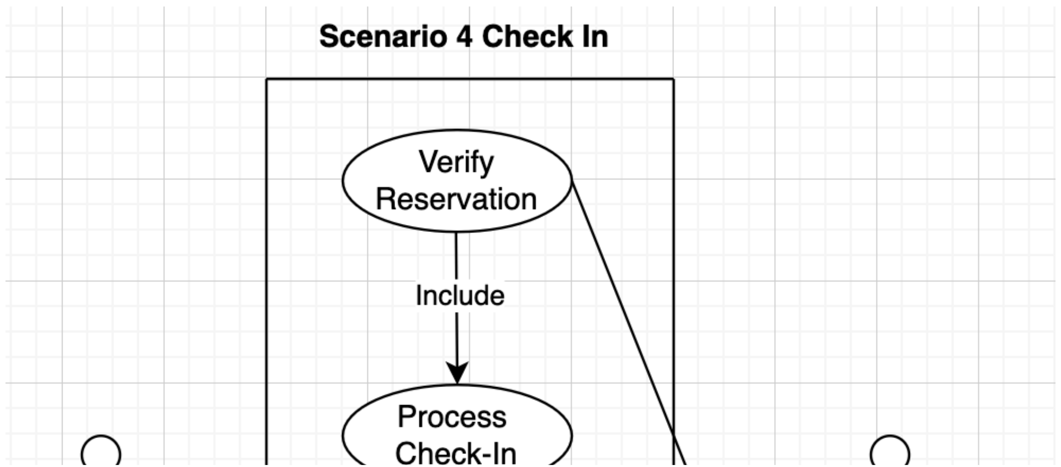
   Description: Gets the total cost of the reservation.

Python Class:

```python
from datetime import date


class Reservation:
    """Class to represent a reservation."""

    # Constructor
    def __init__(self, reservation_date=None, reservation_availability="",
status="", total_amount=0.0):
        self.__reservation_date = reservation_date if reservation_date is not
None else date.today()
        self.__reservation_availability = reservation_availability
        self.__status = status
        self.__total_amount = total_amount

    # Setter & Getter for reservationDate
    def get_reservation_date(self):
        return self.__reservation_date

    def set_reservation_date(self, reservation_date):
        self.__reservation_date = reservation_date

    # Setter & Getter for reservationAvailability
    def get_reservation_availability(self):
        return self.__reservation_availability

    def set_reservation_availability(self, reservation_availability):
        self.__reservation_availability = reservation_availability

    # Setter & Getter for status
    def get_status(self):
        return self.__status

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

    # Setter & Getter for totalAmount
    def get_total_amount(self):
        return self.__total_amount

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

    # Display function
    def __str__(self):
        return (f"Reservation Date: {self.__reservation_date}, "
                f"Availability: {self.__reservation_availability}, "
                f"Status: {self.__status}, Total Amount:
{self.__total_amount:.2f}")


# Creating an Object
reservation1 = Reservation(date(2024, 10, 5), "Available", "Confirmed",
150.75)
print(reservation1)
```

Scenario 4:

UML Use-Case Diagram:

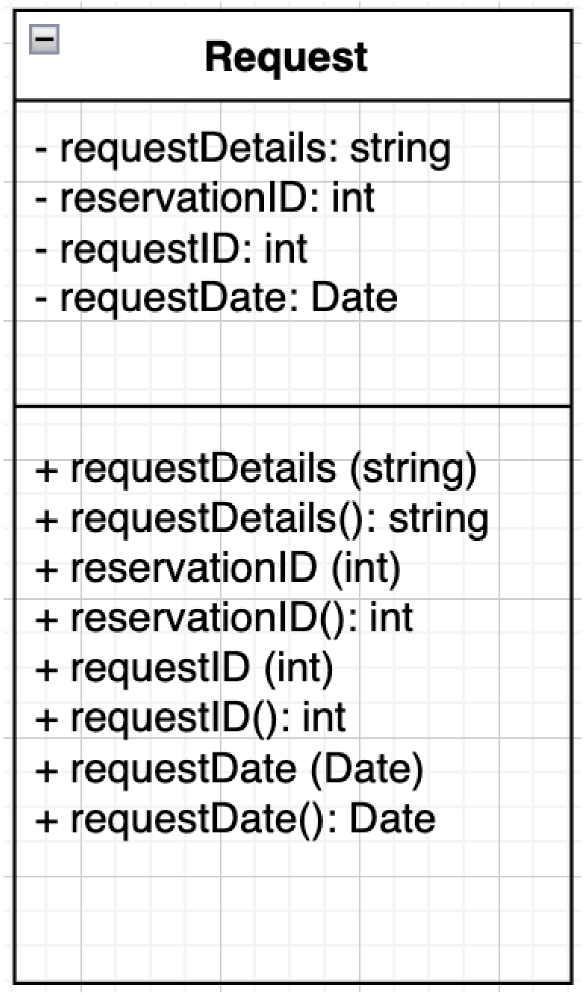


Use-Case Description:

Case	Handel Special Requests
Actors	Consumer Hotel Costumer Service
Trigger	A consumer submits a special request to the hotel either before or during their stay.
Precondition	The consumer has an existing reservation with the hotel. The hotel has a system or procedure in place to handle special requests.
Main Scenario	The guest contacts the hotel with a special request (e.g., extra pillows, late check-out). Hotel staff enter the request details into their system. Staff check if the request can be fulfilled based on availability and policies.

	Staff notify the guest about the request status and any additional costs. The hotel arranges to meet the guest's request. Staff follow up to ensure the guest is happy with how the request was handled. The request is marked as completed, and any feedback from the guest is recorded.
Exceptions	Request Cannot Be Fulfilled: If the request cannot be accommodated due to unavailability or other constraints, hotel staff inform the guest and offer alternative solutions if possible. Additional Costs: If fulfilling the request involves extra charges, staff notify the guest about the costs and obtain their confirmation before proceeding.

UML Class Diagram:



UML Class Description:

Class Name: Request

The Request class stores information about a user request related to reservations. It captures details necessary for processing the request.

Attributes:

1. requestDetails: string (private)

   Description: A description of the request made by the user.

2. reservationID: int (private)

    Description: A unique identifier for the associated reservation.

3. requestID: int (private)

    Description: A unique identifier for the request itself.

4. requestDate: Date (private)

    Description: The date when the request was made.

<u>Methods:</u>

1. requestDetails(): string (public)

    Description: Gets the details of the request.

2. reservationID(): int (public)

    Description: Gets the unique number for the associated reservation.

3. requestID(): int (public)

    Description: Gets the unique number for the request.

4. requestDate(): Date (public)

    Description: Gets the date the request was made.

Python Class:

```python
from datetime import date


class Request:
   """Class to represent a request related to a reservation."""

   # Constructor
   def __init__(self, request_details="", reservation_id=0, request_id=0,
request_date=None):
       self.__request_details = request_details
       self.__reservation_id = reservation_id
```

```python
        self.__request_id = request_id
        self.__request_date = request_date if request_date is not None else
date.today()

    # Setter & Getter for requestDetails
    def get_request_details(self):
        return self.__request_details

    def set_request_details(self, request_details):
        self.__request_details = request_details

    # Setter & Getter for reservationID
    def get_reservation_id(self):
        return self.__reservation_id

    def set_reservation_id(self, reservation_id):
        self.__reservation_id = reservation_id

    # Setter & Getter for requestID
    def get_request_id(self):
        return self.__request_id

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

    # Setter & Getter for requestDate
    def get_request_date(self):
        return self.__request_date

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

    # Display function
    def __str__(self):
        return (f"Request ID: {self.__request_id}, "
                f"Reservation ID: {self.__reservation_id}, "
                f"Request Details: {self.__request_details}, "
                f"Request Date: {self.__request_date}")


# Creating an Object
request1 = Request("Change reservation date", 101, 1001, date(2024, 9, 24))
print(request1)
```

Summary of learnings:

Use-Case Diagrams

Use-Case Diagrams are simple visual tools that show how users interact with the hotel system. They help identify different functions like payment processing and cancellations, making it clear who does what.

Use-Case Descriptions

Use-Case Descriptions outline the specific steps users take when using the system. They detail what triggers each action, who is involved, and what could go wrong, helping developers understand user needs better.

Class Diagrams

Class Diagrams illustrate the structure of the system by showing classes, their attributes, and methods. This helps organize data and functions, ensuring everything is in the right place and works well together.

Python Class Implementation

Python Class Implementation gives real coding examples for creating and managing objects in the hotel system, such as payments, reservations, and cancellations. For instance, an object could represent a payment method, including attributes like card type and balance. This hands-on approach helps developers see how to turn ideas into actual code, allowing them to understand the behavior and properties of each object in the system.

Link of Github:

https://github.com/hubhub12345/assig1