

Hamda Sami Bukattara

College of Interdisciplinary Studies, Zayed University

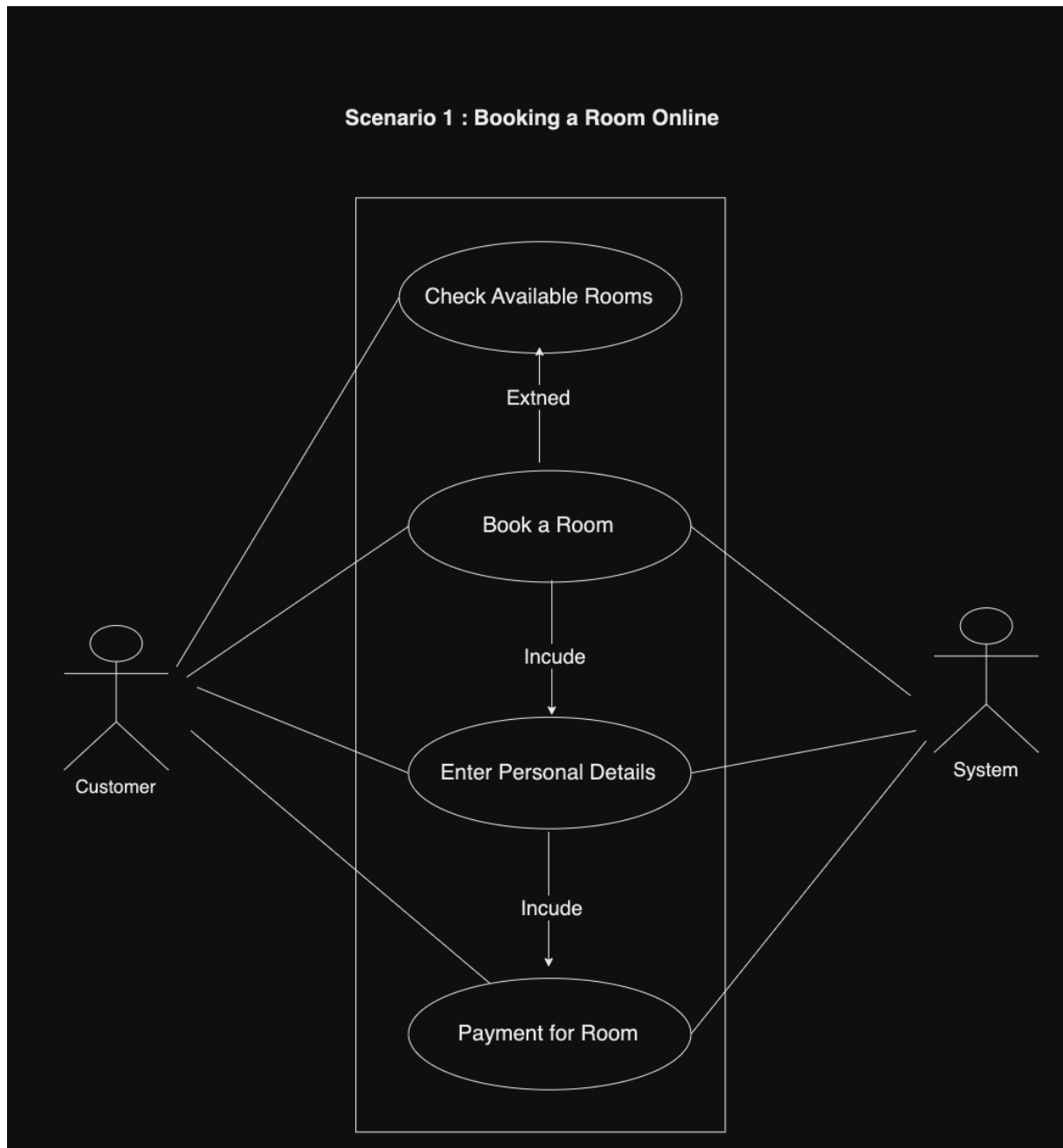
ICS220: Program. Fund.

Dr. Sujith Mathew

September 30, 2024

Scenario 1:

Step 1 : UML Use-Case Diagram



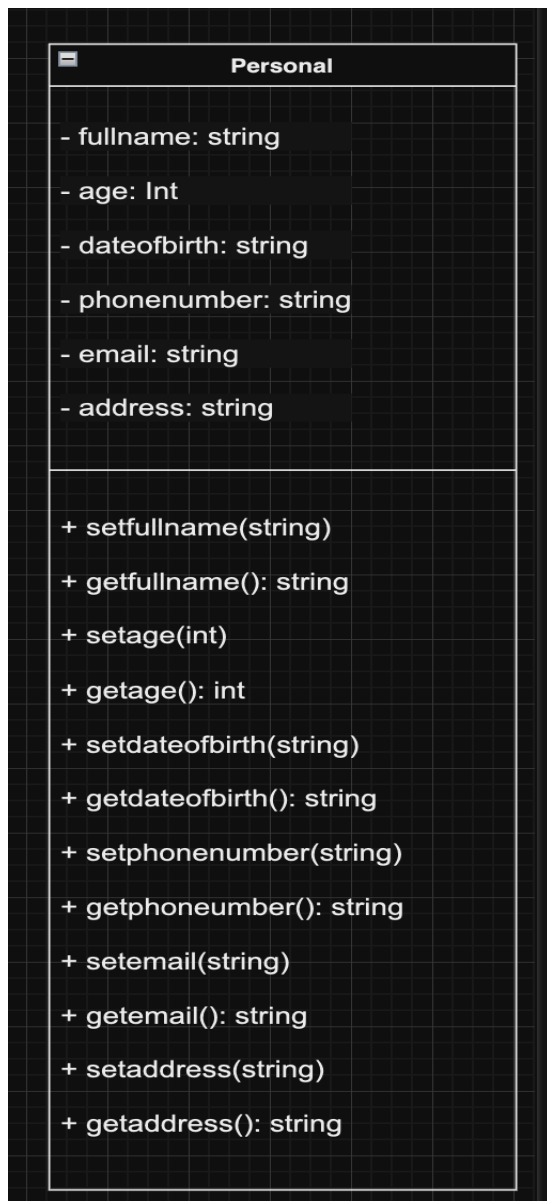
Step 2 : Use-Case Description:

The Use Case	Enter Personal Details
Actors	<ul style="list-style-type: none">- Customer- System

Triggers	<ul style="list-style-type: none"> - The customer needs to enter their personal details during the booking process
Pre-conditions	<ul style="list-style-type: none"> - The customer has selected a room and is in the process of booking - The system is ready to receive personal details - The customer is logged into their account (if the booking process requires an account)
Main Scenario	<ol style="list-style-type: none"> 1. The system prompts the customer to enter their personal details 2. The customer enters their name, email, phone number, and other required information 3. The system validates the entered details 4. The system stores the customer's personal details in the database 5. The system confirms that the personal details are saved and proceeds to the next step in the booking process (payment)
Expectations	<ul style="list-style-type: none"> - If the details are missing or invalid For example the user entered the wrong format for phone number or email, the system displays an error message and asks the customer to re-enter the correct details - If the system fails to store the details due to a technical issue, it notifies the

	<p>customer and asks them to try again</p> <ul style="list-style-type: none"> - If the session times out due to customer inactivity, the system logs out the customer and prompts them to re-enter their details after logging in again
--	--

Step 3 : UML Case Diagram:



UML Class Description:

Class Name: Personal

The Personal class represents an individual's personal information in a system. It stores basic details like the person's name, age, date of birth, phone number, email, and address. This class allows for easy access to and updating of these details through getter and setter methods. It is often used in applications like customer management or user registration, where tracking and managing personal data is important. This class ensures the information is organized and can be retrieved or modified when needed.

Attributes:

1) fullname: string (private)

Stores the full name of a person as text or string, such as "Ahmad Ali". It is private, meaning it can't be directly accessed or changed from outside the class. This privacy ensures the full name is managed properly and prevents unintended modifications.

2) age: Int (private)

It represents the person's age as a whole number, like 29. Since it's private, the value can't be directly accessed or changed from outside the class, ensuring it stays accurate and controlled.

3) dateofbirth: string (private)

Stores the person's birth date as a string, typically in the format "YYYY-MM-DD". Being private, it keeps the birth date secure and prevents any unintended or unauthorized changes.

4) phonenumber: string (private)

Holds the person's phone number, stored as string, for example, "97150778544". Keeping this information private ensures that it remains protected and prevents unauthorized access or changes.

5) email: string (private)

Stores the person's email address, such as "ahmadali@gmail.com". It remains private to ensure that sensitive information like the email is secured from outside access or changes.

6) address: string (private)

Represents the person's home or mailing address as text, like "Al Mizhar 2". Keeping the address private protects the person's location details from being exposed or altered.

Behaviors/ methods :

The set_fullname(string) (public)

method allows you to update the person's full name by accepting a string value that represents the person's new name. This updated name is then stored in the private fullname attribute of the class.

The get_fullname(): string (public)

method retrieves the person's full name from the private attribute and returns it as a string, providing controlled access to the stored name.

The set_age(int) (public)

method updates the person's age by taking an integer value that represents their current age and storing it securely in the private age attribute of the class.

The get_age(): int (public)

method retrieves the person's current age from the private attribute and returns it as an integer, ensuring that the value can be accessed safely.

The set_dateofbirth(string) (public)

method allows you to update the person's date of birth by accepting a string value, usually formatted as "YYYY-MM-DD", and storing it in the private dateofbirth attribute.

The get_dateofbirth(): string (public)

method retrieves the stored date of birth from the private attribute and returns it as a string, giving controlled access to this information.

The set_phonenumber(string) (public)

method updates the person's phone number by accepting a string value that represents the number and storing it securely in the private phonenumber attribute.

The get_phonenumber(): string (public)

method retrieves the person's phone number from the private attribute and returns it as a string, allowing access to this data while keeping it protected.

The set_email(string) (public)

method updates the person's email address by accepting a string value and storing it in the private email attribute to ensure the email is secure and correctly stored.

The get_email(): string (public)

method retrieves the person's email address from the private attribute and returns it as a string, providing controlled access to this sensitive information.

The set_address(string) (public)

method allows you to update the person's home or mailing address by accepting a string value representing the address and securely storing it in the private address attribute.

The get_address(): string (public)

method retrieves the person's address from the private attribute and returns it as a string, allowing controlled access to the stored location details.

Step 4 : Python Code

```
# I am creating a class to handle personal information
class Personal:
    """This is the class I am using to store personal details."""

    # I define the attributes for personal information and mention if they
    # are string or int
    def __init__(self, fullname="", age=0, dateofbirth="", phonenumber="",
    email="", address=""):
        # These are my private attributes for storing personal details
        self.__fullname = fullname
        self.__age = age
        self.__dateofbirth = dateofbirth
        self.__phonenumber = phonenumber
        self.__email = email
        self.__address = address

    # I use this method to get the full name
    def get_fullname(self):
        return self.__fullname

    # This method allows me to get the current age
    def get_age(self):
        return self.__age

    # This method helps me retrieve the date of birth
    def get_dateofbirth(self):
        return self.__dateofbirth

    # Here, I can get the phone number using this method
    def get_phonenumber(self):
        return self.__phonenumber

    # I use this method to get the email address
    def get_email(self):
        return self.__email

    # This method allows me to retrieve the home address
    def get_address(self):
        return self.__address

    # I use this method to set or update the full name
    def set_fullname(self, fullname):
        self.__fullname = fullname

    # I use this method to update the age
    def set_age(self, age):
        self.__age = age
```

```

# This method allows me to set the date of birth
def set_dateofbirth(self, dateofbirth):
    self.__dateofbirth = dateofbirth

# I use this method to update the phone number
def set_phonenumber(self, phonenumber):
    self.__phonenumber = phonenumber

# I use this method to update the email
def set_email(self, email):
    self.__email = email

# This method allows me to set the home address
def set_address(self, address):
    self.__address = address

# I created this method to display the personal details neatly
def display_personal_details(self):
    # I'm using f-strings here to format and display all the personal
details
    print(f"Full Name: {self.__fullname}")
    print(f"Age: {self.__age}")
    print(f>Date of Birth: {self.__dateofbirth}")
    print(f"Phone Number: {self.__phonenumber}")
    print(f>Email: {self.__email}")
    print(f"Address: {self.__address}")

# Here I create an object with the personal information I have
person_info = Personal(
    fullname="Ahmad Ali",
    age=29,
    dateofbirth="1995-05-19",
    phonenumber="97150778544", # I added the phone number
    email="ahmadali@gmail.com",
    address="Al Mizhar 2"
)

# Now, I'll display the personal details using the method I defined earlier
person_info.display_personal_details()

# I use the setter methods to update the full name if needed
# In this case, I'm keeping the full name unchanged
person_info.set_fullname("Ahmad Ali")

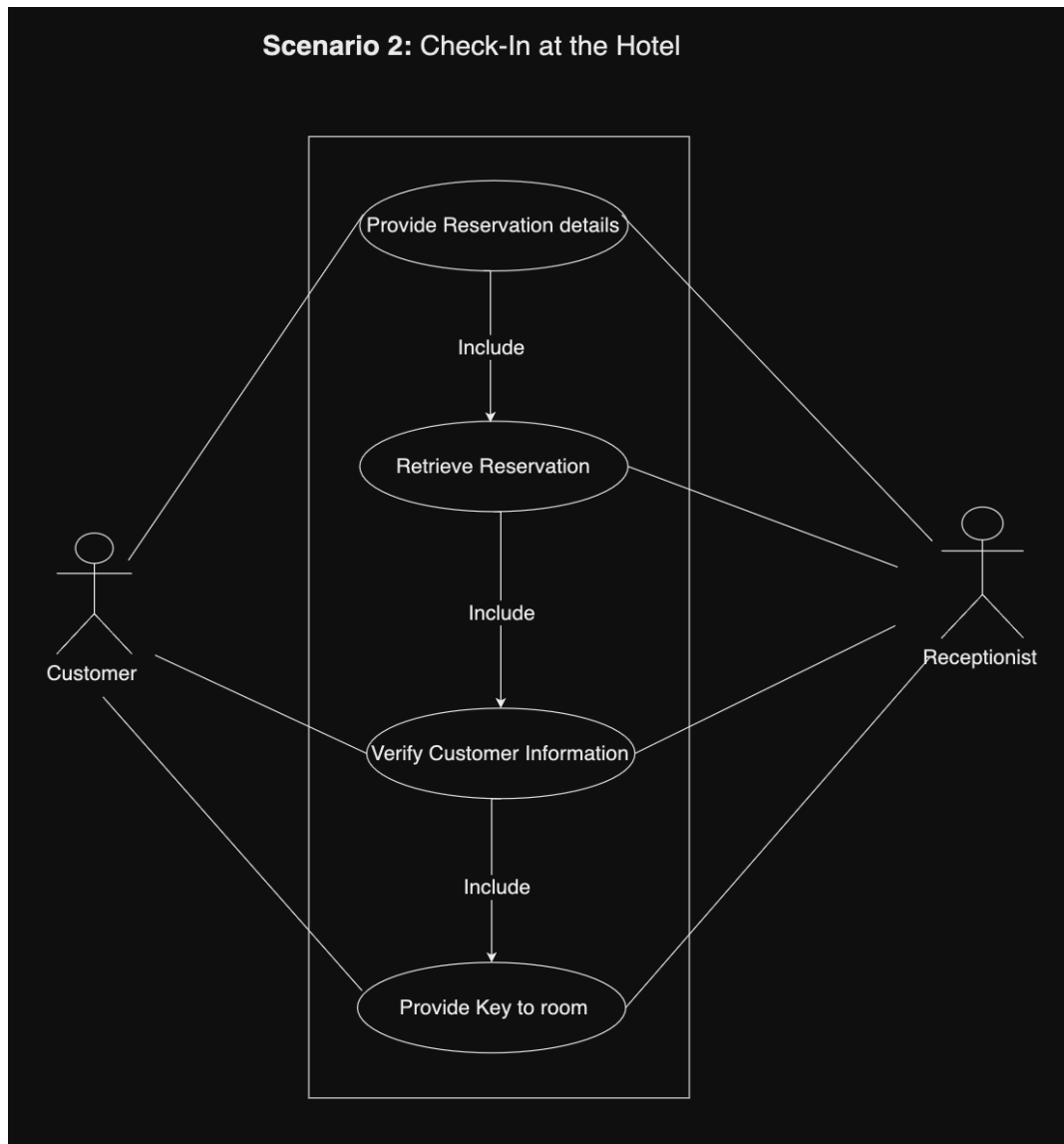
# I'm displaying the updated personal details to confirm the details worked
print("\nUpdated Personal Details:")
person_info.display_personal_details()

```

The code Works

Scenario 2:

Step 1 : UML Use-Case Diagram



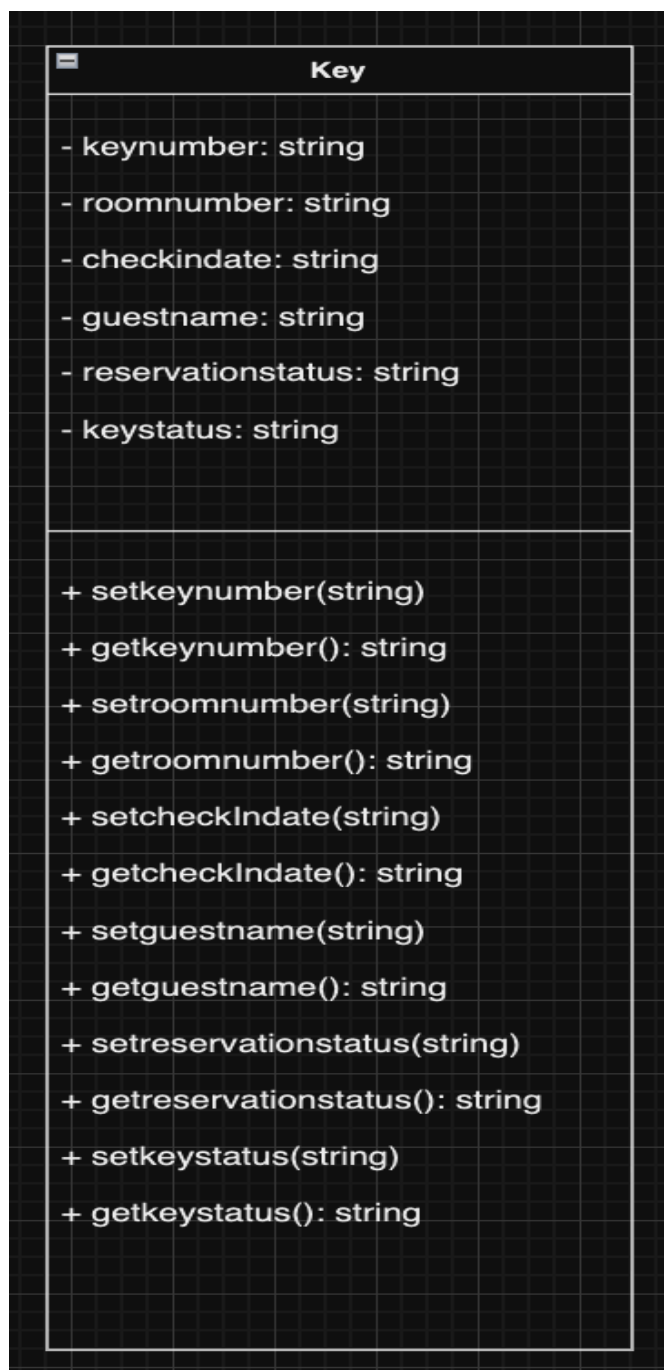
Step 2 : Use-Case Description:

The Use Case	Provide Key to Room
Actors	<ul style="list-style-type: none">- Customer- Receptionist
Triggers	<ul style="list-style-type: none">- The receptionist provides the room key to the customer after the check in

	process is completed.
Pre-conditions	<ul style="list-style-type: none"> - The customer has a confirmed reservation in the system - The customer's personal details have been verified - The payment and booking process is successfully completed - The receptionist is logged into the system and ready to provide the key
Main Scenario	<ol style="list-style-type: none"> 1. The receptionist retrieves the customer's reservation in the system 2. The system verifies the customer's identity and reservation details 3. The receptionist requests the system to generate or provide the room key 4. The system confirms the customer is checked in and ready to receive the key 5. The receptionist hands over the room key to the customer
Expectations	<ul style="list-style-type: none"> - If the reservation is not found, the system prompts the receptionist to ask for updated customer details or confirmation

	<ul style="list-style-type: none"> - If the payment has not been completed, the system notifies the receptionist and customer to resolve the payment - If the system encounters a technical issue in generating the room key, it alerts the receptionist and provides instructions for manual intervention
--	--

Step 3 : UML Case Diagram:



UML Class Description:

Class Name: Key

The Key class represents a hotel room key in a hotel management system. It stores important information such as the key number, room number, check-in date, guest name, reservation status, and key status. This class allows for efficient management of room key data through getter and setter methods. It is typically used in hotel systems to track guest check-ins and ensure that room keys are properly assigned and managed. This class ensures that the key information is secure and can be retrieved or modified as needed, depending on the current status of a guest's stay.

Attributes

- 1) **keynumber: string (private)**
Stores the unique key number assigned to a room. It is private, meaning it can't be accessed or changed from outside the class, ensuring that key information is secure
- 2) **roomnumber: string (private)**
Represents the number of the room associated with the key. Since it's private, the room number can only be modified within the class, keeping the room assignment controlled.
- 3) **checkindate: string (private)**
Stores the check-in date for the guest, typically in the format "YYYY-MM-DD". Keeping this private ensures that check-in details are handled securely.
- 4) **guestname: string (private)**
Holds the name of the guest who is assigned the room key. This is private to ensure that personal details are not accessible from outside the class.
- 5) **reservationstatus: string (private)**
Represents the status of the guest's reservation, such as "confirmed" or "pending". The private status ensures that reservation details are only updated internally.
- 6) **keystatus: string (private)**
Stores the current status of the key, such as "active" or "deactivated". Keeping this attribute private ensures that the key status is only modified through controlled access.

Behaviors/ methods :

set_keynumber(string) (public)

This method allows you to update the key number by accepting a string value representing the key's unique identifier. The updated key number is stored in the private keynumber attribute

get_keynumber(): string (public)

This method retrieves the key number from the private attribute and returns it as a string, providing controlled access to the stored key number

set_roomnumber(string) (public)

This method updates the room number by accepting a string value that represents the number of the room. The room number is stored securely in the private roomnumber attribute

get_roomnumber(): string (public)

This method retrieves the room number from the private attribute and returns it as a string, ensuring controlled access to the room data

set_checkindate(string) (public)

This method allows you to update the check-in date by accepting a string value, usually in the format "YYYY-MM-DD", and storing it in the private checkindate attribute

get_checkindate(): string (public)

This method retrieves the check-in date from the private attribute and returns it as a string, giving controlled access to the check-in information

set_guestname(string) (public)

This method updates the guest's name by accepting a string value that represents the guest staying in the room. The guest name is securely stored in the private guestname attribute

get_guestname(): string (public)

This method retrieves the guest's name from the private attribute and returns it as a string, allowing access to this personal data while keeping it protected.

set_reservationstatus(string) (public)

This method updates the reservation status of the guest by accepting a string value for example is it "confirmed" or "pending" and securely stores it in the private reservationstatus attribute.

get_reservationstatus(): string (public)

This method retrieves the reservation status from the private attribute and returns it as a string, ensuring controlled access to the reservation data.

set_keystatus(string) (public)

This method updates the status of the key for example is it "active" or "deactivated" by accepting a string value and securely storing it in the private keystatus attribute.

get_keystatus(): string (public)

This method retrieves the key's current status from the private attribute and returns it as a string, providing controlled access to the key's state.

Step 4 : Python Code

```
# I am creating a class to handle key information
class Key:
    """This is the class I am using to store key and room details."""

    # In this constructor, I define the attributes for key information
    def __init__(self, keynumber="", roomnumber="", checkindate="",
guestname="", reservationstatus="", keystatus=""):
        # These are my private attributes for storing key details
        self.__keynumber = keynumber
        self.__roomnumber = roomnumber
        self.__checkindate = checkindate
        self.__guestname = guestname
        self.__reservationstatus = reservationstatus
        self.__keystatus = keystatus

    # I use this method to get the key number
    def get_keynumber(self):
        return self.__keynumber

    # I use this method to get the room number
    def get_roomnumber(self):
        return self.__roomnumber

    # I use this method to get the check-in date
    def get_checkindate(self):
        return self.__checkindate

    # This method helps me get the guest name
    def get_guestname(self):
        return self.__guestname

    # I use this method to get the reservation status
    def get_reservationstatus(self):
        return self.__reservationstatus

    # This method allows me to get the key status
    def get_keystatus(self):
        return self.__keystatus

    # I use this method to set or update the key number
    def set_keynumber(self, keynumber):
        self.__keynumber = keynumber

    # I use this method to set or update the room number
    def set_roomnumber(self, roomnumber):
        self.__roomnumber = roomnumber

    # This method allows me to set the check-in date
    def set_checkindate(self, checkindate):
```

```

        self.__checkindate = checkindate

# I use this method to update the guest name
def set_guestname(self, guestname):
    self.__guestname = guestname

# I use this method to update the reservation status
def set_reservationstatus(self, reservationstatus):
    self.__reservationstatus = reservationstatus

# This method allows me to set or update the key status
def set_keystatus(self, keystatus):
    self.__keystatus = keystatus

# I created this method to display the key details neatly
def display_key_details(self):
    # I'm using f-strings here to format and display all the key details
    print(f"Key Number: {self.__keynumber}")
    print(f"Room Number: {self.__roomnumber}")
    print(f"Check-In Date: {self.__checkindate}")
    print(f"Guest Name: {self.__guestname}")
    print(f"Reservation Status: {self.__reservationstatus}")
    print(f"Key Status: {self.__keystatus}")

# Now, I create an object of the Key class with details for a guest
key_info = Key(
    keynumber="K12345",
    roomnumber="101",
    checkindate="2024-09-22",
    guestname="Ahmad Ali",
    reservationstatus="Confirmed",
    keystatus="Active"
)

# I display the key details using the method I defined earlier
key_info.display_key_details()

# I use the setter methods to update the key number and room number if
needed
# For example, I'm changing the key number to "K54321" and the room number
to "102"
key_info.set_keynumber("K54321")
key_info.set_roomnumber("102")

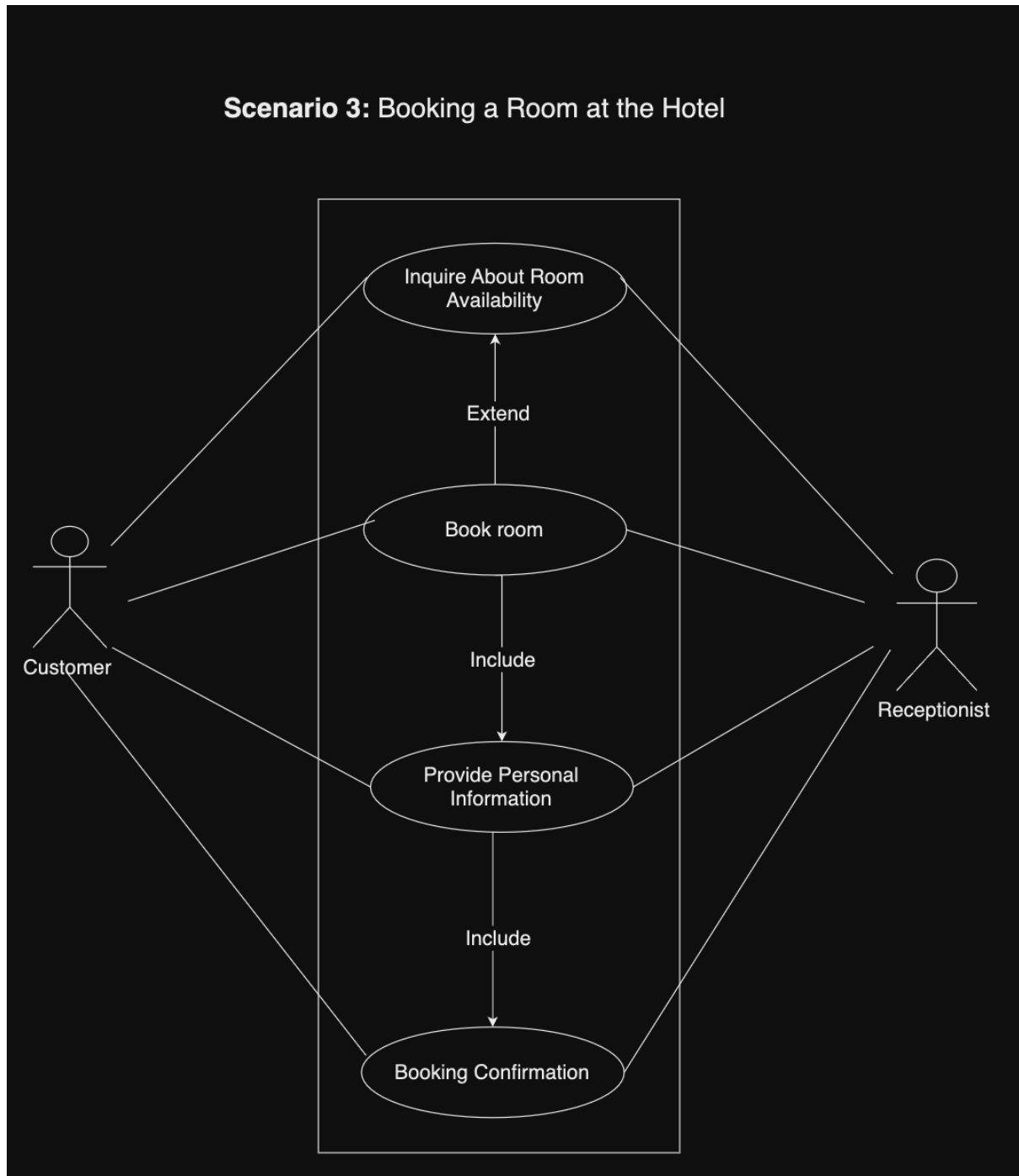
# I'm displaying the updated key details to confirm the changes worked
print("\nUpdated Key Details:")
key_info.display_key_details()

```

The code Works

Scenario 3:

Step 1 : UML Use-Case Diagram

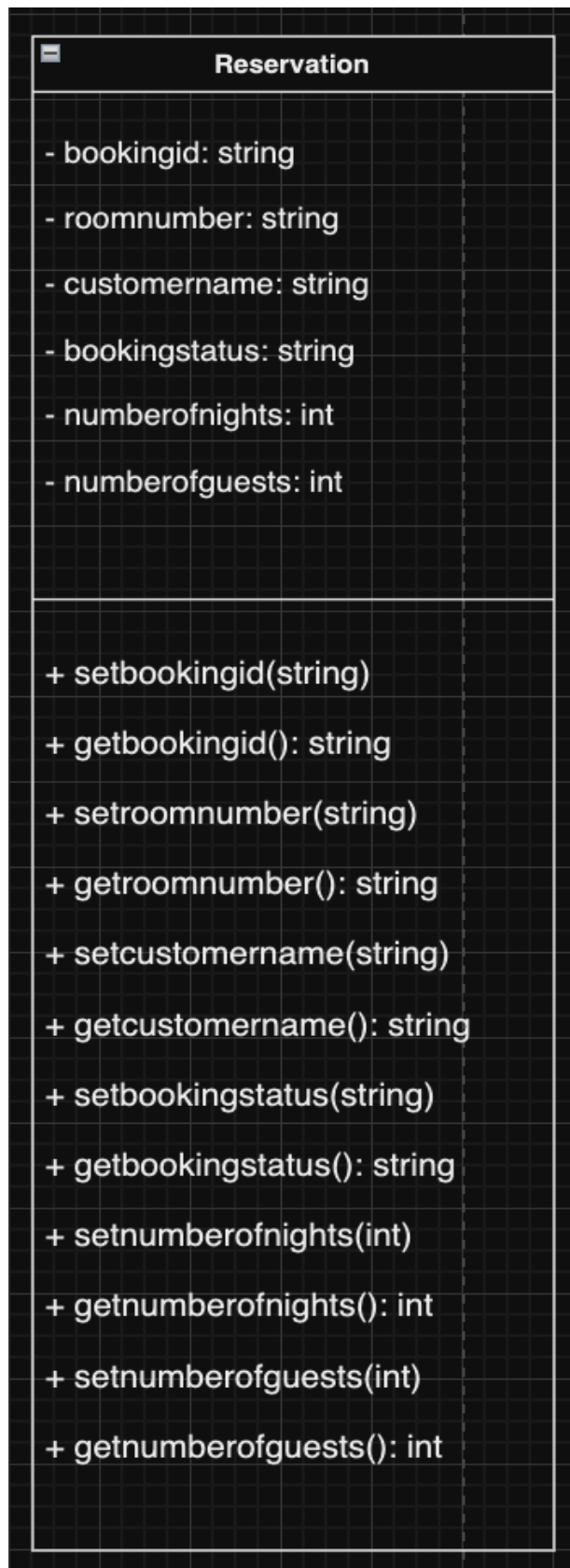


Step 2 : Use-Case Description:

The Use Case	Booking Confirmation
Actors	<ul style="list-style-type: none">- Customer- Receptionist
Triggers	<ul style="list-style-type: none">- The customer has successfully provided their personal information and the room booking has been confirmed by the system and the receptionist
Pre-conditions	<ul style="list-style-type: none">- The customer has selected a room and provided the necessary personal information.- The system has verified room availability and all customer details.- Payment details, if required, have been entered, and the booking process is ready for confirmation- The receptionist is logged into the system and ready to confirm the booking.
Main Scenario	<ol style="list-style-type: none">1. The receptionist retrieves the customer's booking details in the system.2. The system verifies room availability, customer details, and payment information.

	<ol style="list-style-type: none"> 3. The receptionist confirms the booking in the system. 4. The system processes the booking confirmation and updates the reservation status. 5. The system notifies the customer that the booking is confirmed and generates a booking confirmation receipt.
Expectations	<ul style="list-style-type: none"> - If the room is no longer available, the system notifies the receptionist and suggests alternative options. - If customer details or payment information are missing or incorrect, the system asks the receptionist and customer to update the details. - If the system encounters any technical issues during the booking confirmation, it alerts the receptionist and provides options to retry or manually confirm the booking.

Step 3 : UML Case Diagram:



UML Class Description:

Class Name: Reservation

The Reservation class represents a hotel room booking in a hotel management system. It stores essential information such as the booking ID, room number, customer name, booking status, number of nights, and number of guests. This class allows for effective management of bookings through getter and setter methods. It is used in hotel systems to track reservations and ensure that bookings are properly managed and updated. The class ensures that booking details are secure and can be retrieved or modified when necessary.

Attributes

1. **bookingid: string (private)**
Stores the unique identifier for each booking, such as "B12345".
This ensures the booking information is private and only accessible through specific methods.
2. **roomnumber: string (private)**
Holds the number of the room assigned to the booking, for example, "Room 101".
Keeping this private ensures the room assignment is managed and not changed unexpectedly.
3. **customername: string (private)**
Stores the name of the customer who made the booking, for example, "Ahmad Ali".
This attribute remains private to ensure customer details are secure and not exposed.
4. **bookingstatus: string (private)**
Represents the status of the booking, such as "confirmed" or "pending".
Keeping this private ensures the booking status is controlled internally in the system.
5. **numberofnights: int (private)**
Stores the total number of nights the customer will be staying, such as 3 nights.
This attribute is private to ensure accurate tracking and control of stay duration.
6. **numberofguests: int (private)**
Represents the number of guests for the booking, for example, "2 guests".
Keeping this private ensures the system manages the guest count effectively.

Behavior /Methods:

set_bookingid(string) (public)

This method allows you to update the booking ID by accepting a string value representing the unique booking identifier. The updated value is stored in the private bookingid attribute.

get_bookingid(): string (public)

This method retrieves the booking ID from the private attribute and returns it as a string, providing controlled access to the stored booking ID.

set_roomnumber(string) (public)

This method allows you to update the room number by accepting a string value representing the assigned room. The room number is securely stored in the private roomnumber attribute.

get_roomnumber(): string (public)

This method retrieves the room number from the private attribute and returns it as a string, allowing access to the assigned room number.

set_customername(string) (public)

This method updates the customer's name by accepting a string value representing the customer staying in the room. The name is stored in the private customername attribute.

get_customername(): string (public)

This method retrieves the customer's name from the private attribute and returns it as a string, allowing controlled access to the customer's name.

set_bookingstatus(string) (public)

This method updates the booking status by accepting a string value, such as "confirmed" or "pending". The status is securely stored in the private bookingstatus attribute.

get_bookingstatus(): string (public)

This method retrieves the current booking status from the private attribute and returns it as a string, providing controlled access to the status.

set_numberofnights(int) (public)

This method allows you to update the number of nights for the booking by accepting an integer value representing the duration of stay. The value is stored in the private numberofnights attribute.

get_numberofnights(): int (public)

This method retrieves the number of nights from the private attribute and returns it as an integer, allowing access to the booking's stay duration.

set_numberofguests(int) (public)

This method allows you to update the number of guests for the booking by accepting an integer value representing the number of guests. The updated value is stored in the private numberofguests attribute.

get_numberofguests(): int (public)

This method retrieves the number of guests from the private attribute and returns it as an integer, allowing controlled access to the number of guests.

Step 4 : Python Code

```
# I am creating a class to handle booking information  
  
class Reservation:
```

```
"""This is the class I am using to store booking and room details."""

# In this constructor, I define the attributes for booking information

def __init__(self, bookingid="", roomnumber="", customername="",
bookingstatus="", numberofnights=0, numberofguests=0):

    # These are my private attributes for storing booking details

    self.__bookingid = bookingid

    self.__roomnumber = roomnumber

    self.__customername = customername

    self.__bookingstatus = bookingstatus

    self.__numberofnights = numberofnights

    self.__numberofguests = numberofguests


# I use this method to get the booking ID

def get_bookingid(self):

    return self.__bookingid


# I use this method to get the room number

def get_roomnumber(self):

    return self.__roomnumber


# This method helps me retrieve the customer's name

def get_customername(self):

    return self.__customername


# I use this method to get the booking status

def get_bookingstatus(self):

    return self.__bookingstatus
```

```
# This method allows me to get the number of nights for the booking

def get_numberofnights(self):

    return self.__numberofnights


# I use this method to get the number of guests

def get_numberofguests(self):

    return self.__numberofguests


# I use this method to set or update the booking ID

def set_bookingid(self, bookingid):

    self.__bookingid = bookingid


# I use this method to set or update the room number

def set_roomnumber(self, roomnumber):

    self.__roomnumber = roomnumber


# This method allows me to set or update the customer's name

def set_customername(self, customername):

    self.__customername = customername


# I use this method to update the booking status

def set_bookingstatus(self, bookingstatus):

    self.__bookingstatus = bookingstatus


# This method allows me to set the number of nights for the booking

def set_numberofnights(self, numberofnights):

    self.__numberofnights = numberofnights
```

```

# I use this method to update the number of guests for the booking

def set_numberofguests(self, numberofguests):

    self.__numberofguests = numberofguests


# I created this method to display the booking details neatly

def display_booking_details(self):

    # I'm using f-strings here to format and display all the booking
details

    print(f"Booking ID: {self.__bookingid}")

    print(f"Room Number: {self.__roomnumber}")

    print(f"Customer Name: {self.__customername}")

    print(f"Booking Status: {self.__bookingstatus}")

    print(f"Number of Nights: {self.__numberofnights}")

    print(f"Number of Guests: {self.__numberofguests}")


# Now, I create an object of the Booking class with details for a guest
booking_info = Reservation(

    bookingid="B12345",

    roomnumber="101",

    customername="Ahmad Ali",

    bookingstatus="Confirmed",

    numberofnights=3,

    numberofguests=2

)


# I display the booking details using the method I defined earlier

```



```

booking_info.display_booking_details()

# I use the setter methods to update the booking ID and room number if
needed

# For example, I'm changing the booking ID to "B54321" and the room number
to "102"

booking_info.set_bookingid("B54321")

booking_info.set_roomnumber("102")

# I'm displaying the updated booking details to confirm the changes worked

print("\nUpdated Booking Details:")

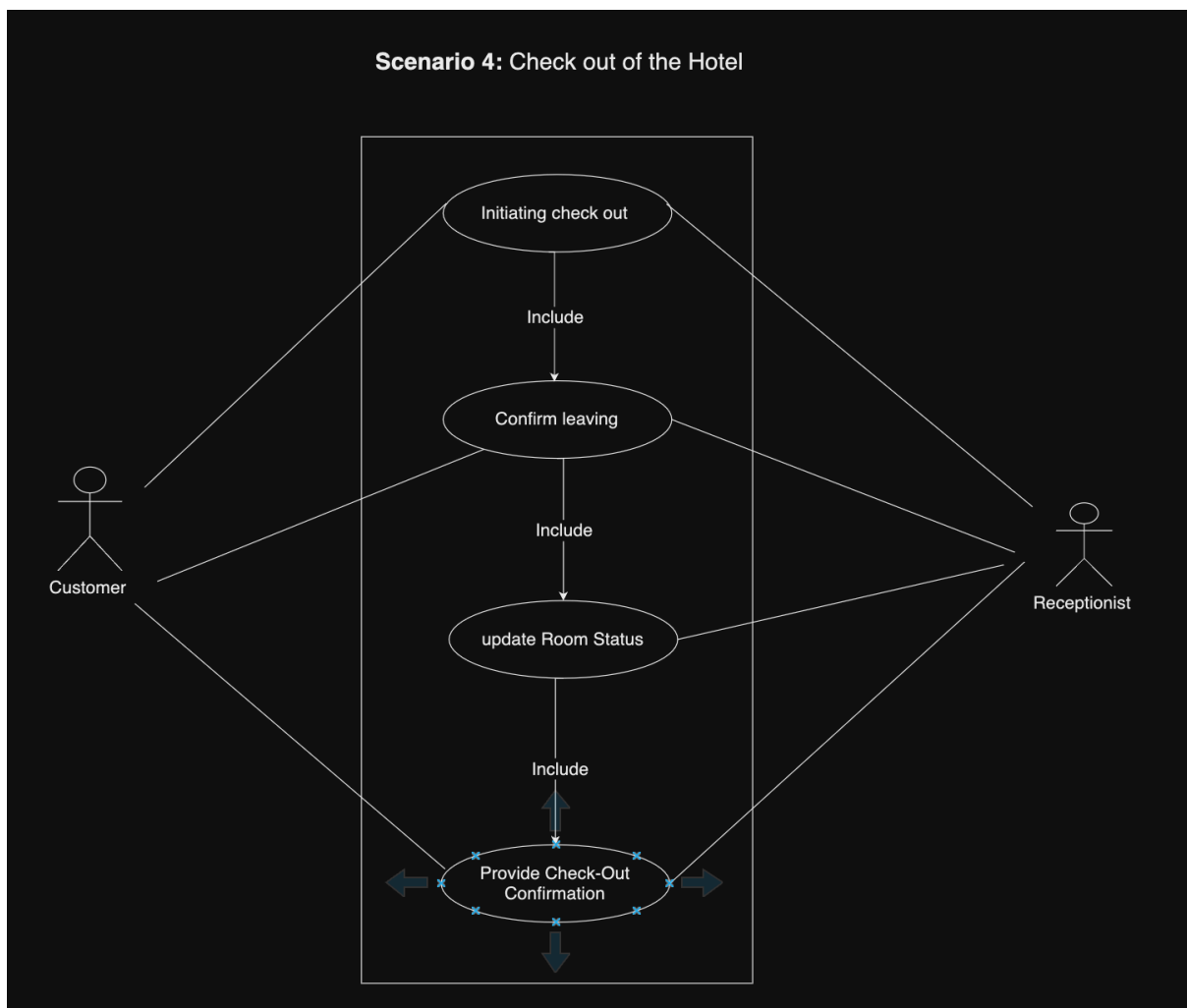
booking_info.display_booking_details()

```

The code Works

Scenario 4:

Step 1 : UML Use-Case Diagram




Step 2 : Use-Case Description:

The Use Case	Provide Check-Out Confirmation
Actors	<ul style="list-style-type: none">- Customer- Receptionist
Triggers	<ul style="list-style-type: none">- The customer has confirmed their departure, and the system has updated the room status, causing the system to provide the final checkout confirmation to the customer.
Pre-conditions	<ul style="list-style-type: none">- The customer has checked out and their stay has officially ended.- The receptionist is logged into the system and has access to the customer's booking and check-out information- All payments and outstanding balances have been settled and paid by the customer.
Main Scenario	<ol style="list-style-type: none">1. The receptionist retrieves the customer's check-out details in the system.2. The system verifies that the customer has completed the check-out process and that all balances are cleared3. The receptionist confirms that the check-out process is completed.

	<ol style="list-style-type: none"> 4. The system generates a check-out confirmation receipt for the customer 5. The receptionist hands over the confirmation receipt to the customer, acknowledging the completion of their stay
Expectations	<ul style="list-style-type: none"> ● If the customer has outstanding payments or unresolved issues, the system notifies the receptionist and prompts the customer to settle the balance before providing confirmation ● If the system encounters technical difficulties while generating the check-out confirmation, it alerts the receptionist and provides instructions for manual confirmation ● If the customer requests a digital receipt instead of a physical one, the system offers an option to email the check-out confirmation to the customer

Step 3 : UML Case Diagram:

 Customercheckout
<ul style="list-style-type: none">- confirmationid: string- customername: string- roomnumber: string- checkoutdate: string- totalamount: float- extracharges: float- durationofstay: int
<ul style="list-style-type: none">+ setconfirmationid(string)+ getconfirmationid(): string+ setcustomername(string)+ getcustomername(): string+ setroomnumber(string)+ getroomnumber(): string+ setcheckoutdate(string)+ getcheckoutdate(): string+ settotalamount(float)+ gettotalamount(): float+ setextracharges(float)+ getextracharges(): float+ setdurationofstay(int)+ getdurationofstay(): int

UML Class Description:

Class Name: Customercheckout

The Customercheckout class is designed to manage the guest departure process in a hotel system. It handles essential information such as the confirmation ID, customer name, room number, check-out date, total amount, extra charges, and duration of stay. The class ensures that the guest's departure is processed smoothly and securely, updating the room status and providing necessary confirmations for a complete check-out. Additionally, it tracks any remaining balance or extra charges that the guest may need to settle before departure.

Attributes:

1. **confirmationid**: string (private)
This attribute stores a unique identifier for each check-out confirmation, like "C12345". It ensures that each check-out process is tracked separately.
2. **customername**: string (private)
Holds the name of the customer who is checking out. It's private to protect customer information.
3. **roomnumber**: string (private)
Stores the room number of the guest. This helps link the check-out to the correct room.
4. **checkoutdate**: string (private)
Represents the date when the guest checks out, formatted as "YYYY-MM-DD". It's private to track this event securely.
5. **totalamount**: float (private)
Holds the total payment amount for the guest's stay, ensuring secure handling of financial data.
6. **extracharges**: float (private)
Stores any extra charges incurred by the guest during their stay, such as room service or other fees.
7. **durationofstay**: int (private)
Tracks the total number of days the guest stayed. This attribute is kept private for security.

Behavior/ Methods:

setconfirmationid(string) (public)

Allows updating the confirmation ID with a new value to securely track the check-out process.

getconfirmationid(): string (public)

Retrieves the current confirmation ID, providing controlled access to this unique identifier.

setcustomername(string) (public)

Updates the customer's name securely.

getcustomername(): string (public)

Retrieves the customer's name, ensuring controlled access to guest information.

setroomnumber(string) (public)

Updates the room number for the guest's check-out process.

getroomnumber(): string (public)

Retrieves the room number, providing secure access to this information.

setcheckoutdate(string) (public)

Updates the check-out date securely.

getcheckoutdate(): string (public)

Retrieves the check-out date to track when the guest left.

settotalamount(float) (public)

Updates the total amount due for the stay, ensuring it is securely stored.

gettotalamount(): float (public)

Retrieves the total amount paid by the guest, providing access to financial details.

setextracharges(float) (public)

Updates any extra charges incurred by the guest during their stay.

getextracharges(): float (public)

Retrieves the extra charges, allowing controlled access to this information.

setdurationofstay(int) (public)

Updates the duration of the guest's stay in days.

getdurationofstay(): int (public)

Retrieves the duration of the stay, ensuring access to this important detail.

Step 4 : Python Code

```
# I am creating a class to handle checkout confirmation information
class Customercheckout:
    """This is the class I am using to store checkout confirmation and room
    details."""

    # In this constructor, I define the attributes for checkout confirmation
    def __init__(self, confirmationid="", customername="", roomnumber="",
checkoutdate="", totalamount=0.0, extracharges=0.0, durationofstay=0):
        # These are my private attributes for storing checkout confirmation
        details
        self.__confirmationid = confirmationid
        self.__customername = customername
        self.__roomnumber = roomnumber
        self.__checkoutdate = checkoutdate
        self.__totalamount = totalamount
        self.__extracharges = extracharges
        self.__durationofstay = durationofstay

    # I use this method to get the confirmation ID
    def get_confirmationid(self):
        return self.__confirmationid

    # I use this method to get the customer name
    def get_customername(self):
        return self.__customername

    # This method helps me retrieve the room number
    def get_roomnumber(self):
        return self.__roomnumber

    # I use this method to get the checkout date
    def get_checkoutdate(self):
        return self.__checkoutdate

    # This method allows me to get the total amount
    def get_totalamount(self):
        return self.__totalamount

    # I use this method to get the extra charges
    def get_extracharges(self):
        return self.__extracharges

    # This method allows me to retrieve the duration of stay
    def get_durationofstay(self):
        return self.__durationofstay
```

```

# I use this method to set or update the confirmation ID
def set_confirmationid(self, confirmationid):
    self.__confirmationid = confirmationid

# I use this method to set or update the customer name
def set_customername(self, customername):
    self.__customername = customername

# I use this method to set or update the room number
def set_roomnumber(self, roomnumber):
    self.__roomnumber = roomnumber

# This method allows me to set or update the checkout date
def set_checkoutdate(self, checkoutdate):
    self.__checkoutdate = checkoutdate

# I use this method to update the total amount
def set_totalamount(self, totalamount):
    self.__totalamount = totalamount

# I use this method to set or update the extra charges
def set_extracharges(self, extracharges):
    self.__extracharges = extracharges

# This method allows me to update the duration of stay
def set_durationofstay(self, durationofstay):
    self.__durationofstay = durationofstay

# I created this method to display the checkout confirmation details
neatly
def display_checkout_details(self):
    # I'm using f-strings here to format and display all the checkout
confirmation details
    print(f"Confirmation ID: {self.__confirmationid}")
    print(f"Customer Name: {self.__customername}")
    print(f"Room Number: {self.__roomnumber}")
    print(f"Checkout Date: {self.__checkoutdate}")
    print(f"Total Amount: {self.__totalamount}")
    print(f"Extra Charges: {self.__extracharges}")
    print(f"Duration of Stay: {self.__durationofstay}")

# Now, I create an object of the Customercheckout class with details for a
guest
checkout_info = Customercheckout(
    confirmationid="C12345",
    customername="Ahmad Ali",
    roomnumber="101",
    checkoutdate="2024-09-22",
    totalamount=500.00,
    extracharges=50.00,
    durationofstay=3

```



```

)

# I display the checkout confirmation details using the method I defined
earlier
checkout_info.display_checkout_details()

# I use the setter methods to update the confirmation ID and room number if
needed
# For example, I'm changing the confirmation ID to "C54321" and the room
number to "102"
checkout_info.set_confirmationid("C54321")
checkout_info.set_roomnumber("102")

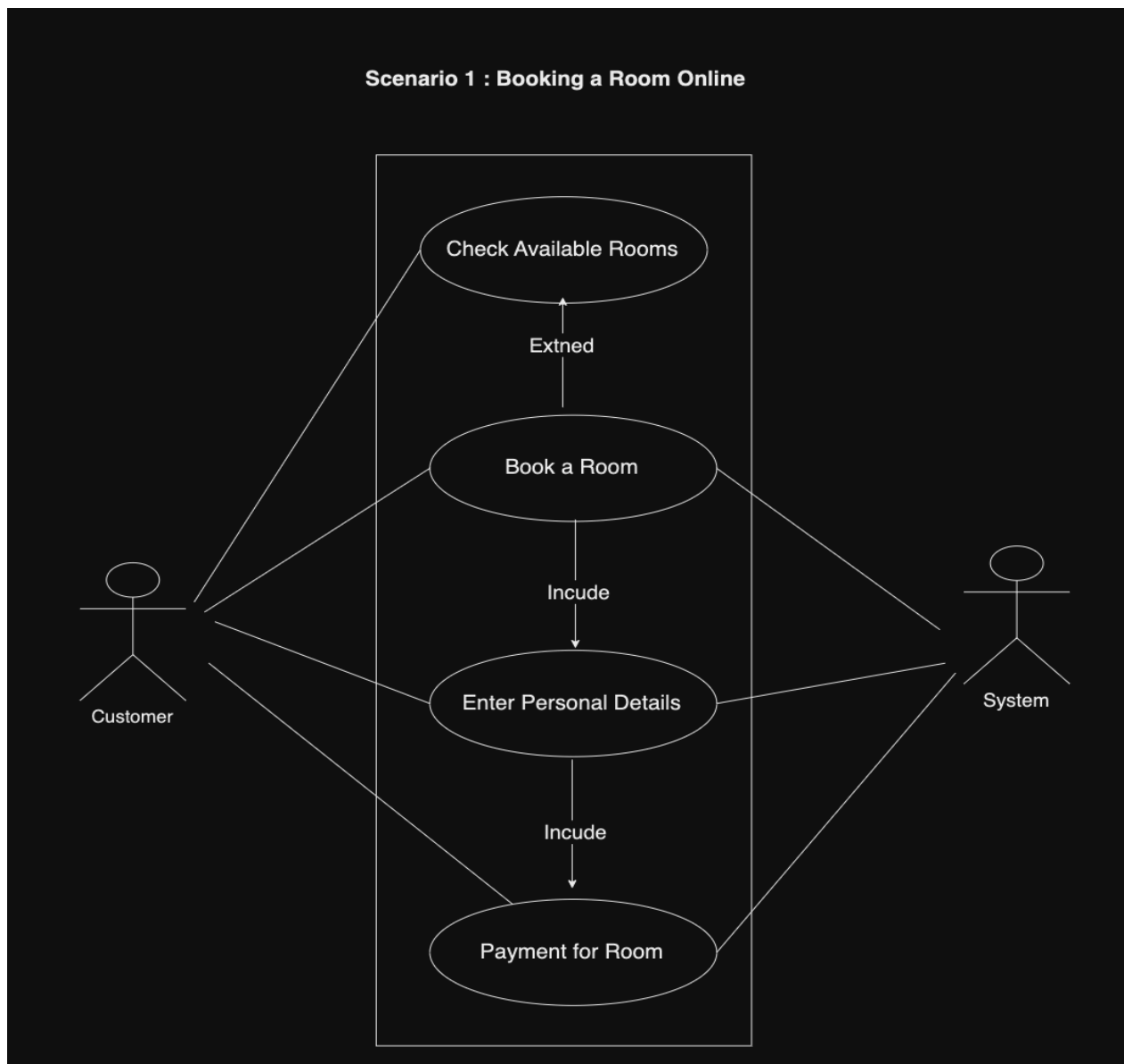
# I'm displaying the updated checkout confirmation details to confirm the
changes worked
print("\nUpdated Checkout Confirmation Details:")
checkout_info.display_checkout_details()

```

The code Works

Scenario 1: The second Use Case, so scenario 5

Step 1 : UML Use-Case Diagram



Step 2 : Use-Case Description:

The Use Case	Payment for Room
Actors	<ul style="list-style-type: none">- Customer- System
Triggers	<ul style="list-style-type: none">- The customer has entered their personal details, and the system is ready to process the payment for the room booking.
Pre-conditions	<ul style="list-style-type: none">- The customer has provided valid personal and booking details.- The system has confirmed that the room is available for booking.- The system is connected to a secure payment gateway to process the payment.
Main Scenario	<ol style="list-style-type: none">1. The system prompts the customer to proceed with the payment for the room2. The customer enters their payment information, such as credit card details or online payment options3. The system verifies the payment details

	<p>and initiates the transaction</p> <ol style="list-style-type: none"> 4. The payment gateway processes the transaction and confirms whether it was successful 5. The system updates the booking status to "confirmed" once the payment is successfully processed 6. The system generates a payment receipt and sends a confirmation notification to the customer.
Expectations	<ul style="list-style-type: none"> - If the payment fails or is declined, the system notifies the customer and provides options to retry with different payment methods - If the system encounters technical issues while processing the payment, it alerts the customer and gives them the option to retry or use an alternative payment method - If the customer exits the payment process without completing it, the system marks the booking as "pending" and sends a reminder notification to complete the payment later

Step 3 : UML Case Diagram:

Payment
- paymentid : int
- paymentstatus : string
- paymentamount : float
- paymentdate : string
- cardtype : string
- currency : string
- taxamount : float
+ setpaymentid(int)
+ getpaymentid(): int
+ setpaymentstatus(string)
+ getpaymentstatus(): string
+ setpaymentamount(float)
+ getpaymentamount(): float
+ setpaymentdate(string)
+ getpaymentdate(): string
+ setcardtype(string)
+ getcardtype(): string
+ setcurrency(string)
+ getcurrency(): string
+ settaxamount(float)
+ gettaxamount(): float

UML Class Description:

Class Name: Payment

The Payment class is responsible for managing payment-related details in the system. It stores critical data such as the payment ID, payment status, payment amount, payment date, card type, currency, and tax amount. This class securely handles these attributes and ensures that all operations related to payments, such as updates and retrievals, are managed efficiently and safely.

Attributes:

1) paymentid: int (private)

Stores a unique identifier for each payment transaction, like "P12345." This ensures that each payment is tracked individually and securely.

2) paymentstatus: string (private)

Represents the current status of the payment, such as "completed" or "pending." This is kept private to ensure that only authorized updates are made.

3) paymentamount: float (private)

Holds the total amount paid by the customer for their stay. It is private to secure financial details.

4) paymentdate: string (private)

Stores the date of the payment, formatted as "YYYY-MM-DD." This helps in tracking when the payment was made.

5) cardtype: string (private)

Holds the type of card used for payment, like "Visa" or "Mastercard." It is kept private for the secure handling of sensitive payment information.

6) currency: string (private)

Stores the currency used in the transaction. In your scenario, this will be "AED" (Emirati Dirhams).

7) taxamount: float (private)

Stores the tax amount applied to the payment, calculated as a percentage of the total amount. This attribute is private to ensure secure financial purpose.

Behaviors/ Methods:

setpaymentid(int) (public)

Allows the payment ID to be updated with a new value, ensuring that each payment is securely identified.

getpaymentid(): int (public)

Retrieves the unique payment ID, providing controlled access to this identifier.

setpaymentstatus(string) (public)

Updates the current status of the payment to reflect changes, such as from "pending" to "completed."

getpaymentstatus(): string (public)

Retrieves the current status of the payment, ensuring controlled access.

setpaymentamount(float) (public)

Updates the total payment amount securely.

getpaymentamount(): float (public)

Retrieves the total amount paid for the room stay, allowing access to financial details.

setpaymentdate(string) (public)

Updates the date when the payment was made, ensuring the information is stored correctly.

getpaymentdate(): string (public)

Retrieves the payment date to track when the payment was processed.

setcardtype(string) (public)

Updates the card type used for payment securely.

getcardtype(): string (public)

Retrieves the card type used for the transaction, ensuring controlled access to sensitive information.

setcurrency(string) (public)

Updates the currency used in the transaction, ensuring it is stored securely (in this case, it should be "AED").

getcurrency(): string (public)

Retrieves the currency type which is AED for the payment.

settaxamount(float) (public)

Updates the tax amount applied to the payment, ensuring it is securely tracked.

gettaxamount(): float (public)

Retrieves the tax amount applied to the payment, providing access to this financial detail

Step 4 : Python Code

```
# I am creating a class to handle payment information
class Payment:
    """This is the class I am using to store payment and transaction
    details."""

    # In this constructor, I define the attributes for payment information
    def __init__(self, paymentid=0, paymentstatus=" ", paymentamount=0.0,
paymentdate=" ", cardtype=" ", currency=" ", taxamount=0.0):
        # These are my private attributes for storing payment details
        self.__paymentid = paymentid
        self.__paymentstatus = paymentstatus
        self.__paymentamount = paymentamount
        self.__paymentdate = paymentdate
        self.__cardtype = cardtype
        self.__currency = currency # it could be USD or AED but I picked
AED
        self.__taxamount = taxamount

    # I use this method to get the payment ID
    def get_paymentid(self):
        return self.__paymentid

    # This method allows me to get the current payment status
    def get_paymentstatus(self):
        return self.__paymentstatus

    # This method helps me retrieve the total payment amount
    def get_paymentamount(self):
        return self.__paymentamount

    # Here, I can get the payment date using this method
    def get_paymentdate(self):
        return self.__paymentdate

    # I use this method to get the card type used for the payment
    def get_cardtype(self):
        return self.__cardtype

    # This method allows me to retrieve the currency used for the payment
    def get_currency(self):
        return self.__currency
```

```

# This method lets me get the tax amount applied to the payment
def get_taxamount(self):
    return self.__taxamount

# I use this method to set or update the payment ID
def set_paymentid(self, paymentid):
    self.__paymentid = paymentid

# I use this method to update the payment status
def set_paymentstatus(self, paymentstatus):
    self.__paymentstatus = paymentstatus

# This method allows me to set the payment amount
def set_paymentamount(self, paymentamount):
    self.__paymentamount = paymentamount

# I use this method to update the payment date
def set_paymentdate(self, paymentdate):
    self.__paymentdate = paymentdate

# I use this method to update the card type used for the payment
def set_cardtype(self, cardtype):
    self.__cardtype = cardtype

# This method allows me to update the currency used in the payment
def set_currency(self, currency):
    self.__currency = currency

# I use this method to set the tax amount for the payment
def set_taxamount(self, taxamount):
    self.__taxamount = taxamount

# I created this method to display the payment details neatly
def display_payment_details(self):
    # I'm using f-strings here to format and display all the payment
information
    print(f"Payment ID: {self.__paymentid}")
    print(f"Payment Status: {self.__paymentstatus}")
    print(f"Payment Amount: {self.__paymentamount}")
    print(f"Payment Date: {self.__paymentdate}")
    print(f"Card Type: {self.__cardtype}")
    print(f"Currency: {self.__currency}")
    print(f"Tax Amount: {self.__taxamount}")

# Creating an object with the payment information I have
payment_info = Payment(
    paymentid="P12345",
    paymentstatus="Completed",
    paymentamount=200.00,
    paymentdate="2024-09-22",
    cardtype="Visa",

```



```
    currency="AED", # I set the currency to AED
    taxamount=15.00
)

# Displaying the payment details using the method I defined earlier
payment_info.display_payment_details()

# I use the setter methods to update the payment ID if needed
# For example, I'm changing the payment ID to "P54321"
payment_info.set_paymentid("P54321")

# Displaying the updated payment details to confirm the changes worked
print("\nUpdated Payment Details:")
payment_info.display_payment_details()
```

The code works

Github Link:

<https://github.com/hamdaSami/Hamda-Assignment-/tree/main>