Full Code

```
# Importing datetime and enum modules

# Handles check-in and check-out dates to ensure its similar to an
actual real life scenario
from datetime import date

# Defines room types in a clear way as pre-determined choices
from enum import Enum

# Room types that are available in the hotel
class RoomType(Enum):
    SINGLE = "Single"
    DOUBLE = "Double"
    SUITE = "Suite"

# RoyalChamber represents a hotel room
class RoyalChamber:
    def __init__(self, room_number, room_type: RoomType, features,
price_per_night):

        # Private attributes to hold room details
        self.__room_number = room_number
        self.__room_type = room_type

        # List of features/amenities
        self.__features = features
        self.__price_per_night = price_per_night

        # Starts as available
        self.__is_available = True

    # Getter for room number
    def get_room_number(self):
        return self.__room_number

    # Getter for room price
    def get_price(self):
        return self.__price_per_night

    # Check if room is available
    def is_available(self):
        return self.__is_available

    # Alternative method for availability check
    def check_availability(self):
        return self.__is_available

    # Changes availability status from True to False or False to
True
    def toggle_status(self):
```

```
        self.__is_available = not self.__is_available

    # Returns formatted room information
    def __str__(self):
        return f"Chamber {self.__room_number}
[{self.__room_type.value}] — AED {self.__price_per_night}/night"

# Manages guest loyalty points and reward tier
class RoyalRewardsProgram:
    def __init__(self):

        # Start with 0 points
        self.__points = 0
        self.__tier = "Bronze"

    # Adds points based on number of nights stayed
    def add_points(self, nights):
        self.__points += nights * 10
        self.__update_tier()

    # Redeem rewards if the guest has 100 or more points
    def redeem_rewards(self):
        if self.__points >= 100:
            self.__points -= 100
            return True
        return False

    # Updates guests tier based on total points
    def __update_tier(self):
        if self.__points >= 300:
            self.__tier = "Gold"
        elif self.__points >= 150:
            self.__tier = "Silver"
        else:
            self.__tier = "Bronze"

    # Get the guest's current tier
    def get_tier(self):
        return self.__tier

    # Get current points balance
    def get_points(self):
        return self.__points

    # String to show points and tier
    def __str__(self):
        return f"Tier: {self.__tier}, Points: {self.__points}"

# Guest class to hold personal info and their reservations
class RoyalGuest:
    def __init__(self, name, email, phone):
        self.__name = name
        self.__email = email
        self.__phone = phone
```

```
        # Composition: has a loyalty program
        self.__loyalty_program = RoyalRewardsProgram()

        # List of reservations made by the guest
        self.__reservations = []

    # Adds a new reservation to the guest's history
    def make_reservation(self, reservation):
        self.__reservations.append(reservation)

    # View list of all past reservations
    def view_history(self):
        return self.__reservations

    # Get guest's name
    def get_name(self):
        return self.__name

    # Access guest's loyalty program
    def get_loyalty(self):
        return self.__loyalty_program

    # Displays guest information
    def __str__(self):
        return f"□ Guest: {self.__name} | {self.__email} | Loyalty:
{self.__loyalty_program}"

# Represents the invoice generated for each reservation
class RoyalInvoice:
    def __init__(self, invoice_id, charges, taxes):
        self.__invoice_id = invoice_id
        self.__charges = charges
        self.__taxes = taxes

        # Total will be calculated later
        self.__total = 0

    # Generates total by adding charges + taxes
    def generate(self):
        self.__total = self.__charges + self.__taxes

    # Shows invoice details
    def display(self):
        return f"□ Invoice #{self.__invoice_id}: Charges: AED
{self.__charges}, Taxes: AED {self.__taxes}, Total: AED
{self.__total}"

    # Returns the total bill
    def get_total(self):
        return self.__total

# Manages reservation process: guest, room, dates and invoice
class RoyalReservation:
```

```
    def __init__(self, reservation_id, guest: RoyalGuest, chamber:
RoyalChamber, check_in_date: date, check_out_date: date):
        self.__reservation_id = reservation_id
        self.__guest = guest
        self.__chamber = chamber
        self.__check_in_date = check_in_date
        self.__check_out_date = check_out_date
        self.__invoice = None

    # Calculates number of nights * room price
    def calculate_total(self):
        nights = (self.__check_out_date - self.__check_in_date).days
        return nights * self.__chamber.get_price()

    # Cancels the reservation and makes the room available again
    def cancel_reservation(self):
        self.__chamber.toggle_status()

    # Confirms booking, generates invoice and adds loyalty points
for the guest
    def confirm(self):

        # Marks room as booked
        self.__chamber.toggle_status()
        charges = self.calculate_total()
        taxes = round(charges * 0.1, 2)
        self.__invoice = RoyalInvoice(f"INV-
{self.__reservation_id}", charges, taxes)
        self.__invoice.generate()
        self.__guest.make_reservation(self)
        self.__guest.get_loyalty().add_points((self.__check_out_date
- self.__check_in_date).days)

    # Returns the invoice linked to this reservation
    def get_invoice(self):
        return self.__invoice

    # Summary of reservation
    def __str__(self):
        return f" Reservation {self.__reservation_id} | Guest:
{self.__guest.get_name()} | Room:
{self.__chamber.get_room_number()}"

# For guests to request services like room cleaning, spa and food
class RequestForRoyalService:
    def __init__(self, service_type, description):
        self.__service_type = service_type
        self.__description = description

    # Generates formatted request
    def request(self):
        return f" Service Requested: {self.__service_type} –
{self.__description}"
```

```
# Guest can leave a review after their stay
class RoyalReview:
    def __init__(self, rating, comments, date_of_review):
        self.__rating = rating
        self.__comments = comments
        self.__date = date_of_review

    # Format the review message
    def leave_review(self):
        return f"□ {self.__rating}/5 | '{self.__comments}' on
{self.__date}"
```

Test Case 1: Royal Guest Books a Luxurious Stay

The goal is to demonstrate room booking, invoice, loyalty point addition and reservation history.

```
from datetime import date
from enum import Enum

class RoomType(Enum):
    SINGLE = "Single"
    DOUBLE = "Double"
    SUITE = "Suite"

class RoyalChamber:
    def __init__(self, room_number, room_type: RoomType, features,
price_per_night):
        self.__room_number = room_number
        self.__room_type = room_type
        self.__features = features
        self.__price_per_night = price_per_night
        self.__is_available = True

    def get_room_number(self):
        return self.__room_number

    def get_price(self):
        return self.__price_per_night

    def is_available(self):
        return self.__is_available

    def check_availability(self):
        return self.__is_available

    def toggle_status(self):
        self.__is_available = not self.__is_available

    def __str__(self):
```

```
        return f"Chamber {self.__room_number}
[{self.__room_type.value}] — AED {self.__price_per_night}/night"

class RoyalRewardsProgram:
    def __init__(self):
        self.__points = 0
        self.__tier = "Bronze"

    def add_points(self, nights):
        self.__points += nights * 10
        self.__update_tier()

    def redeem_rewards(self):
        if self.__points >= 100:
            self.__points -= 100
            return True
        return False

    def __update_tier(self):
        if self.__points >= 300:
            self.__tier = "Gold"
        elif self.__points >= 150:
            self.__tier = "Silver"
        else:
            self.__tier = "Bronze"

    def get_tier(self):
        return self.__tier

    def get_points(self):
        return self.__points

    def __str__(self):
        return f"Tier: {self.__tier}, Points: {self.__points}"

class RoyalGuest:
    def __init__(self, name, email, phone):
        self.__name = name
        self.__email = email
        self.__phone = phone
        self.__loyalty_program = RoyalRewardsProgram()
        self.__reservations = []

    def make_reservation(self, reservation):
        self.__reservations.append(reservation)

    def view_history(self):
        return self.__reservations

    def get_name(self):
        return self.__name

    def get_loyalty(self):
        return self.__loyalty_program
```

```
    def __str__(self):
        return f"🧾 Guest: {self.__name} | {self.__email} | Loyalty:
{self.__loyalty_program}"

class RoyalInvoice:
    def __init__(self, invoice_id, charges, taxes):
        self.__invoice_id = invoice_id
        self.__charges = charges
        self.__taxes = taxes
        self.__total = 0

    def generate(self):
        self.__total = self.__charges + self.__taxes

    def display(self):
        return f"🧾 Invoice #{self.__invoice_id}: Charges: AED
{self.__charges}, Taxes: AED {self.__taxes}, Total: AED
{self.__total}"

    def get_total(self):
        return self.__total

class RoyalReservation:
    def __init__(self, reservation_id, guest: RoyalGuest, chamber:
RoyalChamber, check_in_date: date, check_out_date: date):
        self.__reservation_id = reservation_id
        self.__guest = guest
        self.__chamber = chamber
        self.__check_in_date = check_in_date
        self.__check_out_date = check_out_date
        self.__invoice = None

    def calculate_total(self):
        nights = (self.__check_out_date - self.__check_in_date).days
        return nights * self.__chamber.get_price()

    def cancel_reservation(self):
        self.__chamber.toggle_status()

    def confirm(self):
        self.__chamber.toggle_status()
        charges = self.calculate_total()
        taxes = round(charges * 0.1, 2)
        self.__invoice = RoyalInvoice(f"INV-
{self.__reservation_id}", charges, taxes)
        self.__invoice.generate()
        self.__guest.make_reservation(self)
        self.__guest.get_loyalty().add_points((self.__check_out_date
- self.__check_in_date).days)

    def get_invoice(self):
        return self.__invoice
```

```
    def __str__(self):
        return f" Reservation {self.__reservation_id} | Guest:
{self.__guest.get_name()} | Room:
{self.__chamber.get_room_number()}"

class RequestForRoyalService:
    def __init__(self, service_type, description):
        self.__service_type = service_type
        self.__description = description

    def request(self):
        return f" Service Requested: {self.__service_type} –
{self.__description}"

class RoyalReview:
    def __init__(self, rating, comments, date_of_review):
        self.__rating = rating
        self.__comments = comments
        self.__date = date_of_review

    def leave_review(self):
        return f" {self.__rating}/5 | '{self.__comments}' on
{self.__date}"

# Creating a royal guest
guest_royal = RoyalGuest("Prince Hamad of Al Jadi",
"Hamad@royal.ae", "+971-50-3838370")

# Creating a royal suite
suite_room = RoyalChamber(81, RoomType.SUITE, ["Private Jacuzzi",
"Champagne", "Butler on Call"], 5143.81)

# Making a reservation
reservation_royal = RoyalReservation("R001", guest_royal,
suite_room, date(2025, 5, 14), date(2025, 5, 18))
reservation_royal.confirm()

# Displaying booking details
print(guest_royal)
print(reservation_royal)
print(reservation_royal.get_invoice().display())

# Showing reservation history
print("\n Reservation History:")
for i in guest_royal.view_history():
    print(i)

# Showing loyalty points and tier
print("\n Loyalty Program:")
print(guest_royal.get_loyalty())
```

```
 Guest: Prince Hamad of Al Jadi | Hamad@royal.ae | Loyalty: Tier:
Bronze, Points: 40
 Reservation R001 | Guest: Prince Hamad of Al Jadi | Room: 81
```

```
 Invoice #INV-R001: Charges: AED 20575.24, Taxes: AED 2057.52,
Total: AED 22632.760000000002

 Reservation History:
 Reservation R001 | Guest: Prince Hamad of Al Jadi | Room: 81

 Loyalty Program:
Tier: Bronze, Points: 40
```

Test Case 2: Guest Requests Services and Leaves Feedback

The goal is to demonstrate service requests and post-stay review.

```
from datetime import date
from enum import Enum

class RoomType(Enum):
    SINGLE = "Single"
    DOUBLE = "Double"
    SUITE = "Suite"

class RoyalChamber:
    def __init__(self, room_number, room_type: RoomType, features,
price_per_night):
        self.__room_number = room_number
        self.__room_type = room_type
        self.__features = features
        self.__price_per_night = price_per_night
        self.__is_available = True

    def get_room_number(self):
        return self.__room_number

    def get_price(self):
        return self.__price_per_night

    def is_available(self):
        return self.__is_available

    def check_availability(self):
        return self.__is_available

    def toggle_status(self):
        self.__is_available = not self.__is_available

    def __str__(self):
        return f"Chamber {self.__room_number}
[{self.__room_type.value}] — AED {self.__price_per_night}/night"

class RoyalRewardsProgram:
```

```
    def __init__(self):
        self.__points = 0
        self.__tier = "Bronze"

    def add_points(self, nights):
        self.__points += nights * 10
        self.__update_tier()

    def redeem_rewards(self):
        if self.__points >= 100:
            self.__points -= 100
            return True
        return False

    def __update_tier(self):
        if self.__points >= 300:
            self.__tier = "Gold"
        elif self.__points >= 150:
            self.__tier = "Silver"
        else:
            self.__tier = "Bronze"

    def get_tier(self):
        return self.__tier

    def get_points(self):
        return self.__points

    def __str__(self):
        return f"Tier: {self.__tier}, Points: {self.__points}"

class RoyalGuest:
    def __init__(self, name, email, phone):
        self.__name = name
        self.__email = email
        self.__phone = phone
        self.__loyalty_program = RoyalRewardsProgram()
        self.__reservations = []

    def make_reservation(self, reservation):
        self.__reservations.append(reservation)

    def view_history(self):
        return self.__reservations

    def get_name(self):
        return self.__name

    def get_loyalty(self):
        return self.__loyalty_program

    def __str__(self):
        return f" Guest: {self.__name} | {self.__email} | Loyalty:
{self.__loyalty_program}"
```

```
class RoyalInvoice:
    def __init__(self, invoice_id, charges, taxes):
        self.__invoice_id = invoice_id
        self.__charges = charges
        self.__taxes = taxes
        self.__total = 0

    def generate(self):
        self.__total = self.__charges + self.__taxes

    def display(self):
        return f"🧾 Invoice #{self.__invoice_id}: Charges: AED
{self.__charges}, Taxes: AED {self.__taxes}, Total: AED
{self.__total}"

    def get_total(self):
        return self.__total

class RoyalReservation:
    def __init__(self, reservation_id, guest: RoyalGuest, chamber:
RoyalChamber, check_in_date: date, check_out_date: date):
        self.__reservation_id = reservation_id
        self.__guest = guest
        self.__chamber = chamber
        self.__check_in_date = check_in_date
        self.__check_out_date = check_out_date
        self.__invoice = None

    def calculate_total(self):
        nights = (self.__check_out_date - self.__check_in_date).days
        return nights * self.__chamber.get_price()

    def cancel_reservation(self):
        self.__chamber.toggle_status()

    def confirm(self):
        self.__chamber.toggle_status()
        charges = self.calculate_total()
        taxes = round(charges * 0.1, 2)
        self.__invoice = RoyalInvoice(f"INV-
{self.__reservation_id}", charges, taxes)
        self.__invoice.generate()
        self.__guest.make_reservation(self)
        self.__guest.get_loyalty().add_points((self.__check_out_date
- self.__check_in_date).days)

    def get_invoice(self):
        return self.__invoice

    def __str__(self):
        return f"🏨 Reservation {self.__reservation_id} | Guest:
{self.__guest.get_name()} | Room:
{self.__chamber.get_room_number()}"
```

```
class RequestForRoyalService:
    def __init__(self, service_type, description):
        self.__service_type = service_type
        self.__description = description

    def request(self):
        return f" Service Requested: {self.__service_type} –
{self.__description}"

class RoyalReview:
    def __init__(self, rating, comments, date_of_review):
        self.__rating = rating
        self.__comments = comments
        self.__date = date_of_review

    def leave_review(self):
        return f" {self.__rating}/5 | '{self.__comments}' on
{self.__date}"

# Royal Service request
service_request = RequestForRoyalService("Spa & Wellness", "Full
body massage with oil in the Royal Spa Suite")
print("\n Guest Service Request:")
print(service_request.request())

# Leaving a royal review
royal_review = RoyalReview(5, "Truly majestic! The suite, the food,
the people, everything was perfect!", date(2025, 5, 18))
print("\n Guest Feedback:")
print(royal_review.leave_review())
```

```
 Guest Service Request:
 Service Requested: Spa & Wellness – Full body massage with oil in
the Royal Spa Suite

 Guest Feedback:
 5/5 | 'Truly majestic! The suite, the food, the people, everything
was perfect!' on 2025-05-18
```

Test Case 3: Guest Makes Multiple Bookings and Redeems Rewards

The goal is to demonstrate multiple bookings, cumulative points and redeeming rewards.

```
from datetime import date
from enum import Enum

class RoomType(Enum):
    SINGLE = "Single"
```

```
    DOUBLE = "Double"
    SUITE = "Suite"

class RoyalChamber:
    def __init__(self, room_number, room_type: RoomType, features,
price_per_night):
        self.__room_number = room_number
        self.__room_type = room_type
        self.__features = features
        self.__price_per_night = price_per_night
        self.__is_available = True

    def get_room_number(self):
        return self.__room_number

    def get_price(self):
        return self.__price_per_night

    def is_available(self):
        return self.__is_available

    def check_availability(self):
        return self.__is_available

    def toggle_status(self):
        self.__is_available = not self.__is_available

    def __str__(self):
        return f"Chamber {self.__room_number}
[{self.__room_type.value}] — AED {self.__price_per_night}/night"

class RoyalRewardsProgram:
    def __init__(self):
        self.__points = 0
        self.__tier = "Bronze"

    def add_points(self, nights):
        self.__points += nights * 10
        self.__update_tier()

    def redeem_rewards(self):
        if self.__points >= 100:
            self.__points -= 100
            return True
        return False

    def __update_tier(self):
        if self.__points >= 300:
            self.__tier = "Gold"
        elif self.__points >= 150:
            self.__tier = "Silver"
        else:
            self.__tier = "Bronze"
```

```
    def get_tier(self):
        return self.__tier

    def get_points(self):
        return self.__points

    def __str__(self):
        return f"Tier: {self.__tier}, Points: {self.__points}"

class RoyalGuest:
    def __init__(self, name, email, phone):
        self.__name = name
        self.__email = email
        self.__phone = phone
        self.__loyalty_program = RoyalRewardsProgram()
        self.__reservations = []

    def make_reservation(self, reservation):
        self.__reservations.append(reservation)

    def view_history(self):
        return self.__reservations

    def get_name(self):
        return self.__name

    def get_loyalty(self):
        return self.__loyalty_program

    def __str__(self):
        return f" Guest: {self.__name} | {self.__email} | Loyalty:
{self.__loyalty_program}"

class RoyalInvoice:
    def __init__(self, invoice_id, charges, taxes):
        self.__invoice_id = invoice_id
        self.__charges = charges
        self.__taxes = taxes
        self.__total = 0

    def generate(self):
        self.__total = self.__charges + self.__taxes

    def display(self):
        return f" Invoice #{self.__invoice_id}: Charges: AED
{self.__charges}, Taxes: AED {self.__taxes}, Total: AED
{self.__total}"

    def get_total(self):
        return self.__total

class RoyalReservation:
    def __init__(self, reservation_id, guest: RoyalGuest, chamber:
RoyalChamber, check_in_date: date, check_out_date: date):
```

```
        self.__reservation_id = reservation_id
        self.__guest = guest
        self.__chamber = chamber
        self.__check_in_date = check_in_date
        self.__check_out_date = check_out_date
        self.__invoice = None

    def calculate_total(self):
        nights = (self.__check_out_date - self.__check_in_date).days
        return nights * self.__chamber.get_price()

    def cancel_reservation(self):
        self.__chamber.toggle_status()

    def confirm(self):
        self.__chamber.toggle_status()
        charges = self.calculate_total()
        taxes = round(charges * 0.1, 2)
        self.__invoice = RoyalInvoice(f"INV-
{self.__reservation_id}", charges, taxes)
        self.__invoice.generate()
        self.__guest.make_reservation(self)
        self.__guest.get_loyalty().add_points((self.__check_out_date
- self.__check_in_date).days)

    def get_invoice(self):
        return self.__invoice

    def __str__(self):
        return f" Reservation {self.__reservation_id} | Guest:
{self.__guest.get_name()} | Room:
{self.__chamber.get_room_number()}"

class RequestForRoyalService:
    def __init__(self, service_type, description):
        self.__service_type = service_type
        self.__description = description

    def request(self):
        return f" Service Requested: {self.__service_type} –
{self.__description}"

class RoyalReview:
    def __init__(self, rating, comments, date_of_review):
        self.__rating = rating
        self.__comments = comments
        self.__date = date_of_review

    def leave_review(self):
        return f" {self.__rating}/5 | '{self.__comments}' on
{self.__date}"

# Booking chambers
Deluxe1 = RoyalChamber(252, RoomType.DOUBLE, ["Balcony View", "Mini
```

```
Bar"], 1212)
Standard1 = RoyalChamber(331, RoomType.SINGLE, ["Smart TV",
"Complimentary Breakfast"], 777)
Deluxe2 = RoyalChamber(122, RoomType.DOUBLE, ["Balcony View", "Mini
Bar"], 1921)
Standard2 = RoyalChamber(291, RoomType.SINGLE, ["Smart TV",
"Complimentary Breakfast"], 917)

# Creating a guest
Loyalguest = RoyalGuest("Sheikha Marya", "Marya@royalmail.com",
"+971-56-1920178")

# First booking
reservation1 = RoyalReservation("R002", Loyalguest, Deluxe1,
date(2025, 4, 5), date(2025, 4, 10))
reservation1.confirm()

# Second booking
reservation2 = RoyalReservation("R003", Loyalguest, Standard1,
date(2025, 4, 12), date(2025, 4, 16))
reservation2.confirm()

# Third booking
reservation3 = RoyalReservation("R004", Loyalguest, suite_room,
date(2025, 4, 18), date(2025, 4, 22))
reservation3.confirm()

# Fourth booking
reservation4 = RoyalReservation("R005", Loyalguest, Standard2,
date(2025, 4, 23), date(2025, 4, 26))
reservation4.confirm()

# Fifth booking
reservation5 = RoyalReservation("R006", Loyalguest, Deluxe2,
date(2025, 4, 27), date(2025, 4, 30))
reservation5.confirm()

# Printing guest info
print("\n Guest Status After Multiple Bookings:")
print(Loyalguest)

# Showing reservation history
print("\n All Reservations:")
for i in Loyalguest.view_history():
    print(i)

# Showing loyalty status
print("\n Loyalty Program Status:")
print(Loyalguest.get_loyalty())

# Trying to redeem a reward
print("\n Redeeming Rewards:")
if Loyalguest.get_loyalty().redeem_rewards():
    print(" Reward redeemed! Enjoy a complimentary Royal Service!")
```

```
else:
    print("□ Not enough points for reward redemption.")


□ Guest Status After Multiple Bookings:
□ Guest: Sheikha Marya | Marya@royalmail.com | Loyalty: Tier:
Silver, Points: 190

□ All Reservations:
□ Reservation R002 | Guest: Sheikha Marya | Room: 252
□ Reservation R003 | Guest: Sheikha Marya | Room: 331
□ Reservation R004 | Guest: Sheikha Marya | Room: 81
□ Reservation R005 | Guest: Sheikha Marya | Room: 291
□ Reservation R006 | Guest: Sheikha Marya | Room: 122

 Loyalty Program Status:
Tier: Silver, Points: 190

□ Redeeming Rewards:
□ Reward redeemed! Enjoy a complimentary Royal Service!
```

Test Case 4: Guest Cancels a Booking

The goal is to demonstrate reservation cancellation and room status reset.

```
from datetime import date
from enum import Enum

class RoomType(Enum):
    SINGLE = "Single"
    DOUBLE = "Double"
    SUITE = "Suite"

class RoyalChamber:
    def __init__(self, room_number, room_type: RoomType, features,
price_per_night):
        self.__room_number = room_number
        self.__room_type = room_type
        self.__features = features
        self.__price_per_night = price_per_night
        self.__is_available = True

    def get_room_number(self):
        return self.__room_number

    def get_price(self):
        return self.__price_per_night

    def is_available(self):
        return self.__is_available

    def check_availability(self):
```

```
        return self.__is_available

    def toggle_status(self):
        self.__is_available = not self.__is_available

    def __str__(self):
        return f"Chamber {self.__room_number}
[{self.__room_type.value}] — AED {self.__price_per_night}/night"

class RoyalRewardsProgram:
    def __init__(self):
        self.__points = 0
        self.__tier = "Bronze"

    def add_points(self, nights):
        self.__points += nights * 10
        self.__update_tier()

    def redeem_rewards(self):
        if self.__points >= 100:
            self.__points -= 100
            return True
        return False

    def __update_tier(self):
        if self.__points >= 300:
            self.__tier = "Gold"
        elif self.__points >= 150:
            self.__tier = "Silver"
        else:
            self.__tier = "Bronze"

    def get_tier(self):
        return self.__tier

    def get_points(self):
        return self.__points

    def __str__(self):
        return f"Tier: {self.__tier}, Points: {self.__points}"

class RoyalGuest:
    def __init__(self, name, email, phone):
        self.__name = name
        self.__email = email
        self.__phone = phone
        self.__loyalty_program = RoyalRewardsProgram()
        self.__reservations = []

    def make_reservation(self, reservation):
        self.__reservations.append(reservation)

    def view_history(self):
        return self.__reservations
```

```
    def get_name(self):
        return self.__name

    def get_loyalty(self):
        return self.__loyalty_program

    def __str__(self):
        return f"□ Guest: {self.__name} | {self.__email} | Loyalty:
{self.__loyalty_program}"

class RoyalInvoice:
    def __init__(self, invoice_id, charges, taxes):
        self.__invoice_id = invoice_id
        self.__charges = charges
        self.__taxes = taxes
        self.__total = 0

    def generate(self):
        self.__total = self.__charges + self.__taxes

    def display(self):
        return f"□ Invoice #{self.__invoice_id}: Charges: AED
{self.__charges}, Taxes: AED {self.__taxes}, Total: AED
{self.__total}"

    def get_total(self):
        return self.__total

class RoyalReservation:
    def __init__(self, reservation_id, guest: RoyalGuest, chamber:
RoyalChamber, check_in_date: date, check_out_date: date):
        self.__reservation_id = reservation_id
        self.__guest = guest
        self.__chamber = chamber
        self.__check_in_date = check_in_date
        self.__check_out_date = check_out_date
        self.__invoice = None

    def calculate_total(self):
        nights = (self.__check_out_date - self.__check_in_date).days
        return nights * self.__chamber.get_price()

    def cancel_reservation(self):
        self.__chamber.toggle_status()

    def confirm(self):
        self.__chamber.toggle_status()
        charges = self.calculate_total()
        taxes = round(charges * 0.1, 2)
        self.__invoice = RoyalInvoice(f"INV-
{self.__reservation_id}", charges, taxes)
        self.__invoice.generate()
        self.__guest.make_reservation(self)
```

```
        self.__guest.get_loyalty().add_points((self.__check_out_date
- self.__check_in_date).days)

    def get_invoice(self):
        return self.__invoice

    def __str__(self):
        return f" Reservation {self.__reservation_id} | Guest:
{self.__guest.get_name()} | Room:
{self.__chamber.get_room_number()}"

class RequestForRoyalService:
    def __init__(self, service_type, description):
        self.__service_type = service_type
        self.__description = description

    def request(self):
        return f" Service Requested: {self.__service_type} –
{self.__description}"

class RoyalReview:
    def __init__(self, rating, comments, date_of_review):
        self.__rating = rating
        self.__comments = comments
        self.__date = date_of_review

    def leave_review(self):
        return f" {self.__rating}/5 | '{self.__comments}' on
{self.__date}"

# Guest
Guestcancels = RoyalGuest("Sheikh Hassan", "hassan@royalmail.ae",
"+971-50-223344")

# Room
Roomcancels = RoyalChamber(414, RoomType.DOUBLE, ["Rain Shower",
"Mini Fridge"], 1900)

# Booking
cancelreservation = RoyalReservation("R007", Guestcancels,
Roomcancels, date(2025, 4, 20), date(2025, 4, 23))
cancelreservation.confirm()

# Canceling the reservation
cancelreservation.cancel_reservation()

# Printing cancellation status
print("\n Reservation Cancelled:")
print(cancelreservation)
print("Room available again?", Roomcancels.is_available())


 Reservation Cancelled:
 Reservation R007 | Guest: Sheikh Hassan | Room: 414
Room available again? True
```