**Royal Stay Hotel Management System**

**Student Name:**

**Student ID:**

**Table of Contents**

[**UML Class Diagram 3**](#_wiyrblxsy5cn)

[**Python Classes 5**](#_30qrfff47osr)

[Classes 5](#_sc13nz7iiib0)

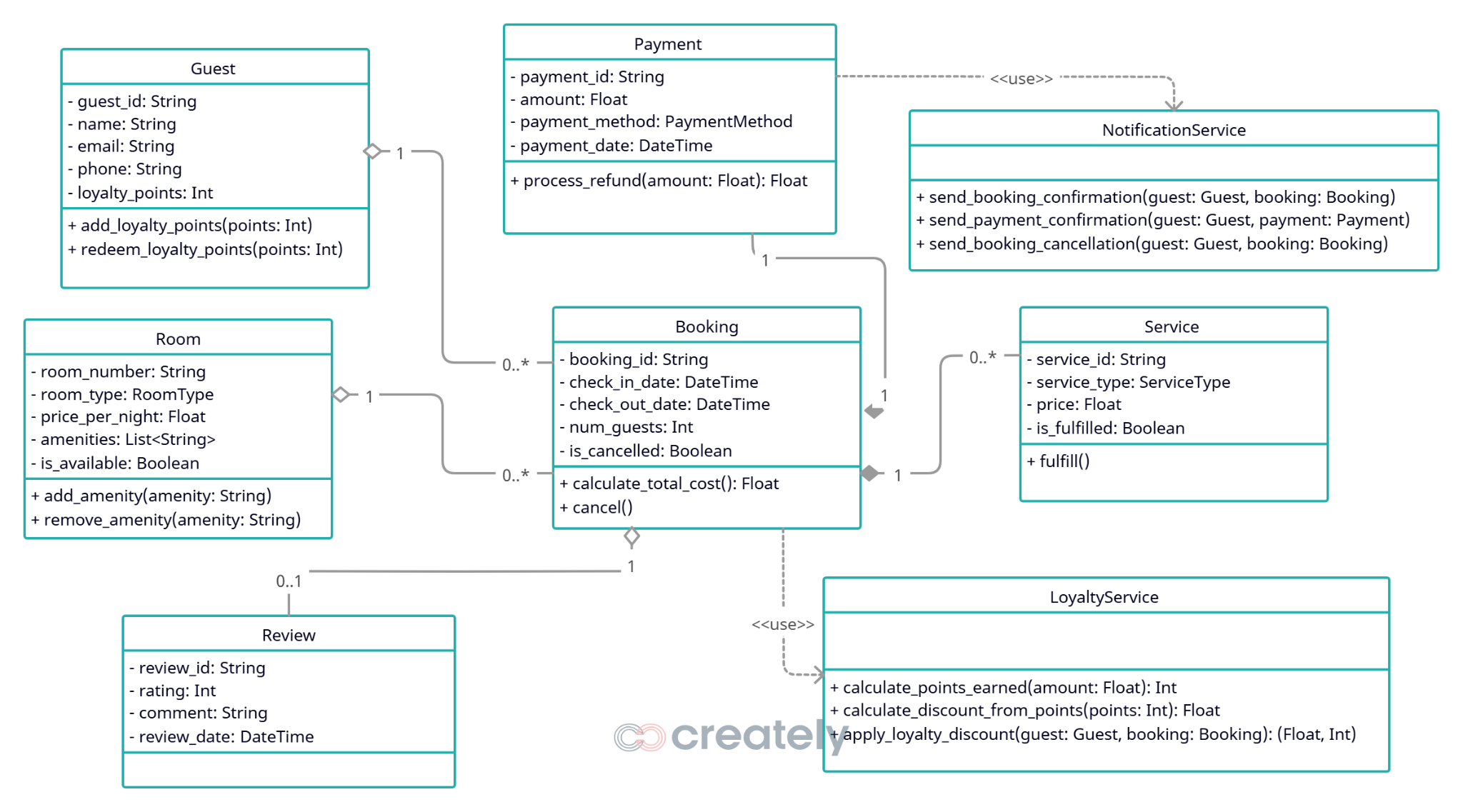
[Main File 5](#_pvno5zf0g25x)

[Test Cases 5](#_y40s50jqloud)

[**Github Repository 5**](#_smf9i8xdm9cp)

[**Summary of Learnings 5**](#_k0e5t6n70x2x)

# **UML Class Diagram**



The aforementioned UML class diagram outlines the structure of a Hotel Management System, detailing the key classes, their attributes, and methods. The Guest class captures guest information, including personal details and loyalty points, with methods to manage these points. The Room class defines room properties such as type, price, and amenities, along with methods to modify amenities. The Booking class handles reservation details, including check-in/check-out dates and guest count, and provides methods for cost calculation and cancellation.

The Payment class manages transaction details, including payment method and amount, with a method to process refunds. The Service class tracks additional services requested by guests, their status, and includes a method to mark services as fulfilled. The Review class stores feedback from guests, including ratings and comments.

Utility classes like NotificationService and LoyaltyService support system operations. The NotificationService sends booking and payment confirmations, as well as cancellation notices, while the LoyaltyService calculates points, discounts, and applies loyalty benefits.

The diagram also reveals key relationships between classes, each with specific cardinality to define how objects interact. Beginning with the Guest and Booking classes, there is a one-to-many relationship where a single Guest can make multiple Bookings, while each Booking is associated with exactly one Guest. This reflects the real-world scenario where a customer may reserve rooms on different occasions, but each reservation is tied to a specific individual.

Moving to the Room and Booking relationship, we see another one-to-many connection. A Room can be part of numerous Bookings over time (as it gets reserved for different dates), but each Booking refers to one specific Room. This ensures the system accurately tracks which rooms are booked and when. The Booking class also has a one-to-one composition relationship with Payment, meaning every Booking must have exactly one Payment, and the Payment cannot exist independently of its Booking. This design enforces that all reservations are properly paid for.

For additional services, the Booking class maintains a one-to-many composition relationship with the Service class. A single Booking may include multiple Services (like room service or laundry), but each Service is exclusively linked to one Booking. Similarly, the Booking has a one-to-zero-or-one relationship with Review, as a guest may optionally leave one Review per Booking. This optionality allows for flexibility in feedback collection.

The utility classes, NotificationService and LoyaltyService, interact through dependency relationships. The Booking class depends on LoyaltyService to calculate points and discounts, while Payment relies on NotificationService to send confirmation messages. These dependencies ensure modularity, keeping business logic separate from operational functions like notifications and loyalty programs. Together, these relationships create a cohesive system for managing reservations, payments, and guest services efficiently.

# **Python Classes**

## **Classes**

from **datetime** import **datetime**, **timedelta**

from **typing** import **Optional**

class **Booking**:

"""Class representing a room booking/reservation"""

def **\_\_init\_\_**(self, booking\_id: **str**, guest, room, check\_in\_date: **datetime**,

check\_out\_date: **datetime**, num\_guests: **int** = 1):

"""

Initialize a Booking object

Args:

booking\_id (str): Unique booking identifier

guest (Guest): Guest making the booking

room (Room): Room being booked

check\_in\_date (datetime): Check-in date

check\_out\_date (datetime): Check-out date

num\_guests (int, optional): Number of guests. Defaults to 1.

"""

self.\_booking\_id = booking\_id

self.\_guest = guest

self.\_room = room

self.\_check\_in\_date = check\_in\_date

self.\_check\_out\_date = check\_out\_date

self.\_num\_guests = num\_guests

self.\_booking\_date = **datetime**.**now**()

self.\_additional\_services = []

self.\_is\_cancelled = False

self.\_cancellation\_date = None

*# Validate the booking*

self.**\_validate\_booking**()

*# Mark room as unavailable*

self.\_room.is\_available = False

*# Add to guest's reservation history*

self.\_guest.add\_reservation(self)

def **\_validate\_booking**(self):

"""Validate the booking parameters"""

if self.\_check\_in\_date **>=** self.\_check\_out\_date:

raise **ValueError**("Check-out date must be after check-in date")

if self.\_num\_guests > self.\_room.\_max\_occupancy:

raise **ValueError**(f"Room can only accommodate {self.\_room.\_max\_occupancy} guests")

if not self.\_room.is\_available:

raise **ValueError**("Room is not available for booking")

**@property**

def **booking\_id**(self):

return self.\_booking\_id

**@property**

def **guest**(self):

return self.\_guest

**@property**

def **room**(self):

return self.\_room

**@property**

def **check\_in\_date**(self):

return self.\_check\_in\_date

**@property**

def **check\_out\_date**(self):

return self.\_check\_out\_date

**@property**

def **num\_nights**(self):

"""Calculate number of nights for the stay"""

return (self.\_check\_out\_date **-** self.\_check\_in\_date).days

def **add\_service**(self, service):

"""Add an additional service to the booking"""

self.\_additional\_services.**append**(service)

def **calculate\_total\_cost**(self):

"""Calculate the total cost of the booking"""

base\_cost = self.num\_nights \* self.\_room.price\_per\_night

services\_cost = **sum**(service.price for service in self.\_additional\_services)

return base\_cost + services\_cost

def **cancel**(self):

"""Cancel the booking"""

if not self.\_is\_cancelled:

self.\_is\_cancelled = True

self.\_cancellation\_date = **datetime**.**now**()

self.\_room.is\_available = True

return True

return False

def **\_\_str\_\_**(self):

status = "Cancelled" if self.\_is\_cancelled else "Confirmed"

return (f"Booking(ID: {self.\_booking\_id}, Guest: {self.\_guest.name}, "

f"Room: {self.\_room.room\_number}, Status: {status}, "

f"Dates: {self.\_check\_in\_date.**strftime**('%Y-%m-%d')} to "

f"{self.\_check\_out\_date.**strftime**('%Y-%m-%d')}, "

f"Nights: {self.num\_nights}, Total: ${self.**calculate\_total\_cost**():.2f})")

class **Guest**:

"""Class representing a hotel guest"""

def **\_\_init\_\_**(self, guest\_id: **str**, name: **str**, email: **str**, phone: **str**, address: **str** = None):

"""

Initialize a Guest object

Args:

guest\_id (str): Unique identifier for the guest

name (str): Full name of the guest

email (str): Email address of the guest

phone (str): Contact phone number

address (str, optional): Physical address. Defaults to None.

"""

self.\_guest\_id = guest\_id

self.\_name = name

self.\_email = email

self.\_phone = phone

self.\_address = address

self.\_loyalty\_points = 0

self.\_reservations = []

**@property**

def **guest\_id**(self):

return self.\_guest\_id

**@property**

def **name**(self):

return self.\_name

**@name.setter**

def **name**(self, value):

if not **isinstance**(value, **str**) or **len**(value.**strip**()) == 0:

raise **ValueError**("Name must be a non-empty string")

self.\_name = value

**@property**

def **email**(self):

return self.\_email

**@email.setter**

def **email**(self, value):

if "@" not in value or "." not in value:

raise **ValueError**("Invalid email format")

self.\_email = value

**@property**

def **phone**(self):

return self.\_phone

**@phone.setter**

def **phone**(self, value):

if not **isinstance**(value, **str**) or **len**(value) < 7:

raise **ValueError**("Phone number must be at least 7 digits")

self.\_phone = value

**@property**

def **loyalty\_points**(self):

return self.\_loyalty\_points

def **add\_loyalty\_points**(self, points: **int**):

"""Add loyalty points to the guest's account"""

if points <= 0:

raise **ValueError**("Points must be positive")

self.\_loyalty\_points += points

def **redeem\_loyalty\_points**(self, points: **int**):

"""Redeem loyalty points from the guest's account"""

if points <= 0:

raise **ValueError**("Points must be positive")

if points > self.\_loyalty\_points:

raise **ValueError**("Not enough points to redeem")

self.\_loyalty\_points -= points

def **add\_reservation**(self, reservation):

"""Add a reservation to the guest's history"""

self.\_reservations.**append**(reservation)

def **get\_reservation\_history**(self):

"""Return the guest's reservation history"""

return self.\_reservations

def **\_\_str\_\_**(self):

return (f"Guest(ID: {self.\_guest\_id}, Name: {self.\_name}, Email: {self.\_email}, "

f"Phone: {self.\_phone}, Loyalty Points: {self.\_loyalty\_points})")

from **enum** import **Enum**

from **datetime** import **datetime**

class **PaymentMethod**(**Enum**):

"""Enumeration of payment methods"""

CREDIT\_CARD = "Credit Card"

DEBIT\_CARD = "Debit Card"

MOBILE\_WALLET = "Mobile Wallet"

BANK\_TRANSFER = "Bank Transfer"

CASH = "Cash"

class **Payment**:

"""Class representing a payment for a booking"""

def **\_\_init\_\_**(self, payment\_id: **str**, booking, amount: **float**,

payment\_method: **PaymentMethod**, payment\_date: **datetime** = None):

"""

Initialize a Payment object

Args:

payment\_id (str): Unique payment identifier

booking (Booking): Booking being paid for

amount (float): Payment amount

payment\_method (PaymentMethod): Method of payment

payment\_date (datetime, optional): Date of payment. Defaults to current datetime.

"""

self.\_payment\_id = payment\_id

self.\_booking = booking

self.\_amount = amount

self.\_payment\_method = payment\_method

self.\_payment\_date = payment\_date if payment\_date else **datetime**.**now**()

self.\_is\_refunded = False

self.\_refund\_date = None

**@property**

def **payment\_id**(self):

return self.\_payment\_id

**@property**

def **booking**(self):

return self.\_booking

**@property**

def **amount**(self):

return self.\_amount

**@property**

def **payment\_method**(self):

return self.\_payment\_method

**@property**

def **payment\_date**(self):

return self.\_payment\_date

def **process\_refund**(self, amount: **float** = None):

"""Process a refund for this payment"""

if self.\_is\_refunded:

raise **ValueError**("Payment is already refunded")

refund\_amount = amount if amount is not None else self.\_amount

if refund\_amount > self.\_amount:

raise **ValueError**("Refund amount cannot exceed original payment")

self.\_is\_refunded = True

self.\_refund\_date = **datetime**.**now**()

return refund\_amount

def **\_\_str\_\_**(self):

status = "Refunded" if self.\_is\_refunded else "Completed"

return (f"Payment(ID: {self.\_payment\_id}, Amount: ${self.\_amount:.2f}, "

f"Method: {self.\_payment\_method.value}, Status: {status}, "

f"Date: {self.\_payment\_date.**strftime**('%Y-%m-%d %H:%M:%S')})")

from **datetime** import **datetime**

class **Review**:

"""Class representing a guest review/feedback"""

def **\_\_init\_\_**(self, review\_id: **str**, guest, booking, rating: **int**,

comment: **str** = None, review\_date: **datetime** = None):

"""

Initialize a Review object

Args:

review\_id (str): Unique review identifier

guest (Guest): Guest making the review

booking (Booking): Booking being reviewed

rating (int): Rating (1-5)

comment (str, optional): Review comment. Defaults to None.

review\_date (datetime, optional): Date of review. Defaults to current datetime.

"""

if rating < 1 or rating > 5:

raise **ValueError**("Rating must be between 1 and 5")

self.\_review\_id = review\_id

self.\_guest = guest

self.\_booking = booking

self.\_rating = rating

self.\_comment = comment

self.\_review\_date = review\_date if review\_date else **datetime**.**now**()

**@property**

def **review\_id**(self):

return self.\_review\_id

**@property**

def **guest**(self):

return self.\_guest

**@property**

def **booking**(self):

return self.\_booking

**@property**

def **rating**(self):

return self.\_rating

**@property**

def **comment**(self):

return self.\_comment

**@comment.setter**

def **comment**(self, value):

self.\_comment = value

**@property**

def **review\_date**(self):

return self.\_review\_date

def **\_\_str\_\_**(self):

return (f"Review(ID: {self.\_review\_id}, Guest: {self.\_guest.name}, "

f"Rating: {self.\_rating}/5, Date: {self.\_review\_date.**strftime**('%Y-%m-%d')}, "

f"Comment: {self.\_comment[:30] + '...' if self.\_comment and **len**(self.\_comment) > 30 else self.\_comment})")

from **enum** import **Enum**

class **RoomType**(**Enum**):

"""Enumeration of room types"""

SINGLE = "Single"

DOUBLE = "Double"

SUITE = "Suite"

DELUXE = "Deluxe"

PRESIDENTIAL = "Presidential"

class **Room**:

"""Class representing a hotel room"""

def **\_\_init\_\_**(self, room\_number: **str**, room\_type: **RoomType**, price\_per\_night: **float**,

amenities: **list** = None, max\_occupancy: **int** = 2):

"""

Initialize a Room object

Args:

room\_number (str): Unique room number/identifier

room\_type (RoomType): Type of the room

price\_per\_night (float): Base price per night

amenities (list, optional): List of amenities. Defaults to None.

max\_occupancy (int, optional): Maximum number of guests. Defaults to 2.

"""

self.\_room\_number = room\_number

self.\_room\_type = room\_type

self.\_price\_per\_night = price\_per\_night

self.\_amenities = amenities if amenities else []

self.\_max\_occupancy = max\_occupancy

self.\_is\_available = True

**@property**

def **room\_number**(self):

return self.\_room\_number

**@property**

def **room\_type**(self):

return self.\_room\_type

**@property**

def **price\_per\_night**(self):

return self.\_price\_per\_night

**@price\_per\_night.setter**

def **price\_per\_night**(self, value):

if value <= 0:

raise **ValueError**("Price must be positive")

self.\_price\_per\_night = value

**@property**

def **amenities**(self):

return self.\_amenities

def **add\_amenity**(self, amenity: **str**):

"""Add an amenity to the room"""

if amenity not in self.\_amenities:

self.\_amenities.**append**(amenity)

def **remove\_amenity**(self, amenity: **str**):

"""Remove an amenity from the room"""

if amenity in self.\_amenities:

self.\_amenities.**remove**(amenity)

**@property**

def **is\_available**(self):

return self.\_is\_available

**@is\_available.setter**

def **is\_available**(self, value: **bool**):

self.\_is\_available = value

def **\_\_str\_\_**(self):

return (f"Room(Number: {self.\_room\_number}, Type: {self.\_room\_type.value}, "

f"Price: ${self.\_price\_per\_night:.2f}/night, Available: {self.\_is\_available}, "

f"Amenities: {', '.**join**(self.\_amenities)})")

from **enum** import **Enum**

class **ServiceType**(**Enum**):

"""Enumeration of service types"""

ROOM\_SERVICE = "Room Service"

HOUSEKEEPING = "Housekeeping"

LAUNDRY = "Laundry"

TRANSPORTATION = "Transportation"

SPA = "Spa"

OTHER = "Other"

class **Service**:

"""Class representing an additional service request"""

def **\_\_init\_\_**(self, service\_id: **str**, service\_type: **ServiceType**, description: **str**,

price: **float**, request\_time: **str** = None):

"""

Initialize a Service object

Args:

service\_id (str): Unique service identifier

service\_type (ServiceType): Type of service

description (str): Description of the service

price (float): Price of the service

request\_time (str, optional): Time the service is requested. Defaults to None.

"""

self.\_service\_id = service\_id

self.\_service\_type = service\_type

self.\_description = description

self.\_price = price

self.\_request\_time = request\_time

self.\_is\_fulfilled = False

**@property**

def **service\_id**(self):

return self.\_service\_id

**@property**

def **service\_type**(self):

return self.\_service\_type

**@property**

def **description**(self):

return self.\_description

**@property**

def **price**(self):

return self.\_price

**@price.setter**

def **price**(self, value):

if value < 0:

raise **ValueError**("Price cannot be negative")

self.\_price = value

**@property**

def **is\_fulfilled**(self):

return self.\_is\_fulfilled

def **fulfill**(self):

"""Mark the service as fulfilled"""

self.\_is\_fulfilled = True

def **\_\_str\_\_**(self):

status = "Fulfilled" if self.\_is\_fulfilled else "Pending"

return (f"Service(ID: {self.\_service\_id}, Type: {self.\_service\_type.value}, "

f"Description: {self.\_description}, Price: ${self.\_price:.2f}, "

f"Status: {status})")

class **LoyaltyService**:

"""Service for managing guest loyalty points"""

POINTS\_PER\_DOLLAR = 10 *# 10 points for every $1 spent*

POINTS\_REDEMPTION\_RATE = 100 *# 100 points = $1 discount*

**@classmethod**

def **calculate\_points\_earned**(cls, amount\_spent: **float**):

"""Calculate loyalty points earned based on amount spent"""

return **int**(amount\_spent \* cls.POINTS\_PER\_DOLLAR)

**@classmethod**

def **calculate\_discount\_from\_points**(cls, points: **int**):

"""Calculate discount amount from loyalty points"""

return points / cls.POINTS\_REDEMPTION\_RATE

**@classmethod**

def **apply\_loyalty\_discount**(cls, guest, booking):

"""

Apply loyalty discount to a booking if guest has enough points

Returns:

tuple: (discount\_amount, points\_used) if applied, (0, 0) otherwise

"""

if guest.loyalty\_points <= 0:

return 0, 0

*# Calculate maximum possible discount*

max\_discount = cls.**calculate\_discount\_from\_points**(guest.loyalty\_points)

booking\_total = booking.calculate\_total\_cost()

*# Can't discount more than the booking total*

discount\_amount = **min**(max\_discount, booking\_total)

points\_used = **int**(discount\_amount \* cls.POINTS\_REDEMPTION\_RATE)

*# Apply the discount*

guest.redeem\_loyalty\_points(points\_used)

return discount\_amount, points\_used

class **NotificationService**:

"""Service for sending notifications to guests"""

**@staticmethod**

def **send\_booking\_confirmation**(guest, booking):

"""Send booking confirmation to guest"""

subject = "Booking Confirmation"

message = (f"Dear {guest.name},\n\n"

f"Your booking has been confirmed.\n"

f"Booking ID: {booking.booking\_id}\n"

f"Room: {booking.room.room\_number} ({booking.room.room\_type.value})\n"

f"Check-in: {booking.check\_in\_date.strftime('%Y-%m-%d')}\n"

f"Check-out: {booking.check\_out\_date.strftime('%Y-%m-%d')}\n"

f"Total cost: ${booking.calculate\_total\_cost():.2f}\n\n"

f"Thank you for choosing Royal Stay Hotel!")

*# In a real implementation, this would send an email or push notification*

**print**(f"=== Email Notification ===\nTo: {guest.email}\nSubject: {subject}\n\n{message}\n")

**@staticmethod**

def **send\_payment\_confirmation**(guest, payment):

"""Send payment confirmation to guest"""

subject = "Payment Confirmation"

message = (f"Dear {guest.name},\n\n"

f"Your payment of ${payment.amount:.2f} has been received.\n"

f"Payment ID: {payment.payment\_id}\n"

f"Method: {payment.payment\_method.value}\n"

f"Date: {payment.payment\_date.strftime('%Y-%m-%d %H:%M:%S')}\n\n"

f"Thank you for choosing Royal Stay Hotel!")

**print**(f"=== Email Notification ===\nTo: {guest.email}\nSubject: {subject}\n\n{message}\n")

**@staticmethod**

def **send\_booking\_cancellation**(guest, booking, refund\_amount=None):

"""Send booking cancellation notification to guest"""

subject = "Booking Cancellation"

message = (f"Dear {guest.name},\n\n"

f"Your booking (ID: {booking.booking\_id}) has been cancelled.\n")

if refund\_amount is not None:

message += f"A refund of ${refund\_amount:.2f} will be processed.\n"

message += "\nWe hope to serve you again in the future."

**print**(f"=== Email Notification ===\nTo: {guest.email}\nSubject: {subject}\n\n{message}\n")

## **Main File**

from **datetime** import **datetime**, **timedelta**

from **models**.**guest** import **Guest**

from **models**.**room** import **Room**, **RoomType**

from **models**.**booking** import **Booking**

from **models**.**payment** import **Payment**, **PaymentMethod**

from **models**.**service** import **Service**, **ServiceType**

from **models**.**review** import **Review**

from **services**.**notification\_service** import **NotificationService**

from **services**.**loyalty\_service** import **LoyaltyService**

def **main**():

*# Create a guest*

guest1 = **Guest**("G001", "John Doe", "john.doe@email.com", "1234567890")

**print**("Guest created:", guest1)

*# Create a room*

room1 = **Room**("101", **RoomType**.DOUBLE, 150.00, ["Wi-Fi", "TV", "Mini-Bar"])

**print**("\nRoom created:", room1)

*# Create a booking*

check\_in = **datetime**.**now**() **+** **timedelta**(days=7)

check\_out = check\_in **+** **timedelta**(days=3)

booking1 = **Booking**("B001", guest1, room1, check\_in, check\_out, num\_guests=2)

**print**("\nBooking created:", booking1)

*# Add a service to the booking*

service1 = **Service**("S001", **ServiceType**.ROOM\_SERVICE, "Breakfast in bed", 25.00)

booking1.**add\_service**(service1)

**print**("\nService added to booking:", service1)

*# Calculate total cost*

**print**(f"\nTotal booking cost: ${booking1.**calculate\_total\_cost**():.2f}")

*# Apply loyalty discount*

guest1.**add\_loyalty\_points**(5000) *# Add some points*

discount, points\_used = **LoyaltyService**.**apply\_loyalty\_discount**(guest1, booking1)

if discount > 0:

**print**(f"\nApplied loyalty discount: ${discount:.2f} (using {points\_used} points)")

**print**(f"New booking total: ${booking1.**calculate\_total\_cost**():.2f}")

**print**(f"Remaining loyalty points: {guest1.loyalty\_points}")

*# Process payment*

payment1 = **Payment**("P001", booking1, booking1.**calculate\_total\_cost**(), **PaymentMethod**.CREDIT\_CARD)

**print**("\nPayment processed:", payment1)

*# Send notifications*

**NotificationService**.**send\_booking\_confirmation**(guest1, booking1)

**NotificationService**.**send\_payment\_confirmation**(guest1, payment1)

*# Add a review after stay*

review1 = **Review**("R001", guest1, booking1, 5, "Excellent stay, would recommend!")

**print**("\nReview submitted:", review1)

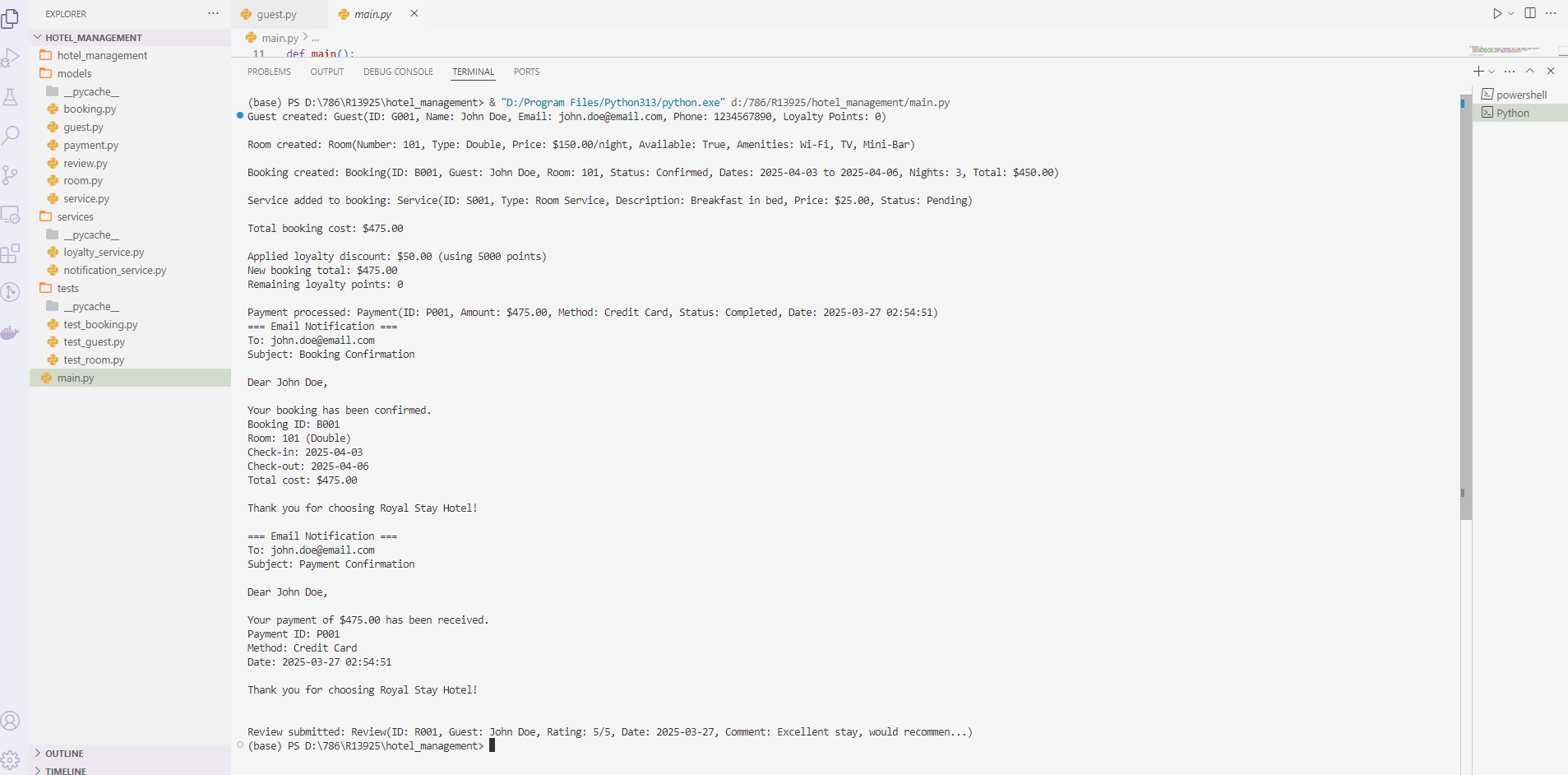
*# Cancel a booking example*

*# booking1.cancel()*

*# NotificationService.send\_booking\_cancellation(guest1, booking1, booking1.calculate\_total\_cost())*

if \_\_name\_\_ == "\_\_main\_\_":

**main**()



## **Test Cases**

import **unittest**

from **datetime** import **datetime**, **timedelta**

from **models**.**guest** import **Guest**

from **models**.**room** import **Room**, **RoomType**

from **models**.**booking** import **Booking**

from **models**.**service** import **Service**, **ServiceType**

class **TestBooking**(**unittest**.**TestCase**):

def **setUp**(self):

self.guest = **Guest**("G002", "Bob Johnson", "bob@example.com", "9876543210")

self.room = **Room**("301", **RoomType**.DELUXE, 200.00)

self.check\_in = **datetime**.**now**() **+** **timedelta**(days=14)

self.check\_out = self.check\_in **+** **timedelta**(days=5)

self.booking = **Booking**("B002", self.guest, self.room, self.check\_in, self.check\_out, 2)

def **test\_booking\_creation**(self):

self.**assertEqual**(self.booking.guest.name, "Bob Johnson")

self.**assertEqual**(self.booking.room.room\_number, "301")

self.**assertEqual**(self.booking.num\_nights, 5)

self.**assertFalse**(self.booking.\_is\_cancelled)

def **test\_invalid\_dates**(self):

with self.**assertRaises**(**ValueError**):

**Booking**("B003", self.guest, self.room, self.check\_out, self.check\_in)

def **test\_room\_availability**(self):

self.**assertFalse**(self.room.is\_available)

def **test\_service\_addition**(self):

service = **Service**("S002", **ServiceType**.TRANSPORTATION, "Airport pickup", 50.00)

self.booking.**add\_service**(service)

self.**assertEqual**(**len**(self.booking.\_additional\_services), 1)

self.**assertEqual**(self.booking.**calculate\_total\_cost**(), 200.00 \* 5 + 50.00)

def **test\_cancellation**(self):

self.**assertTrue**(self.booking.**cancel**())

self.**assertTrue**(self.booking.\_is\_cancelled)

self.**assertTrue**(self.room.is\_available)

*# Test duplicate cancellation*

self.**assertFalse**(self.booking.**cancel**())

if \_\_name\_\_ == "\_\_main\_\_":

**unittest**.**main**()

import **unittest**

from **datetime** import **datetime**, **timedelta**

from **models**.**guest** import **Guest**

from **models**.**room** import **Room**, **RoomType**

from **models**.**booking** import **Booking**

class **TestGuest**(**unittest**.**TestCase**):

def **setUp**(self):

self.guest = **Guest**("G001", "Alice Smith", "alice@example.com", "1234567890")

self.room = **Room**("101", **RoomType**.DOUBLE, 150.00)

check\_in = **datetime**.**now**() **+** **timedelta**(days=7)

check\_out = check\_in **+** **timedelta**(days=3)

self.booking = **Booking**("B001", self.guest, self.room, check\_in, check\_out)

def **test\_guest\_creation**(self):

self.**assertEqual**(self.guest.name, "Alice Smith")

self.**assertEqual**(self.guest.email, "alice@example.com")

self.**assertEqual**(self.guest.phone, "1234567890")

self.**assertEqual**(self.guest.loyalty\_points, 0)

def **test\_loyalty\_points**(self):

self.guest.**add\_loyalty\_points**(1000)

self.**assertEqual**(self.guest.loyalty\_points, 1000)

with self.**assertRaises**(**ValueError**):

self.guest.**add\_loyalty\_points**(-50)

self.guest.**redeem\_loyalty\_points**(500)

self.**assertEqual**(self.guest.loyalty\_points, 500)

with self.**assertRaises**(**ValueError**):

self.guest.**redeem\_loyalty\_points**(600) *# Not enough points*

def **test\_reservation\_history**(self):

self.**assertEqual**(**len**(self.guest.**get\_reservation\_history**()), 1)

self.**assertIsInstance**(self.guest.**get\_reservation\_history**()[0], **Booking**)

def **test\_invalid\_email**(self):

with self.**assertRaises**(**ValueError**):

self.guest.email = "invalid-email"

def **test\_invalid\_phone**(self):

with self.**assertRaises**(**ValueError**):

self.guest.phone = "123"

if \_\_name\_\_ == "\_\_main\_\_":

**unittest**.**main**()

import **unittest**

from **models**.**room** import **Room**, **RoomType**

class **TestRoom**(**unittest**.**TestCase**):

def **setUp**(self):

self.room = **Room**("201", **RoomType**.SUITE, 250.00, ["Wi-Fi", "TV", "Mini-Bar", "Jacuzzi"])

def **test\_room\_creation**(self):

self.**assertEqual**(self.room.room\_number, "201")

self.**assertEqual**(self.room.room\_type, **RoomType**.SUITE)

self.**assertEqual**(self.room.price\_per\_night, 250.00)

self.**assertEqual**(**len**(self.room.amenities), 4)

self.**assertTrue**(self.room.is\_available)

def **test\_amenity\_management**(self):

self.room.**add\_amenity**("Safe")

self.**assertIn**("Safe", self.room.amenities)

self.room.**remove\_amenity**("TV")

self.**assertNotIn**("TV", self.room.amenities)

def **test\_price\_validation**(self):

with self.**assertRaises**(**ValueError**):

self.room.price\_per\_night = -100.00

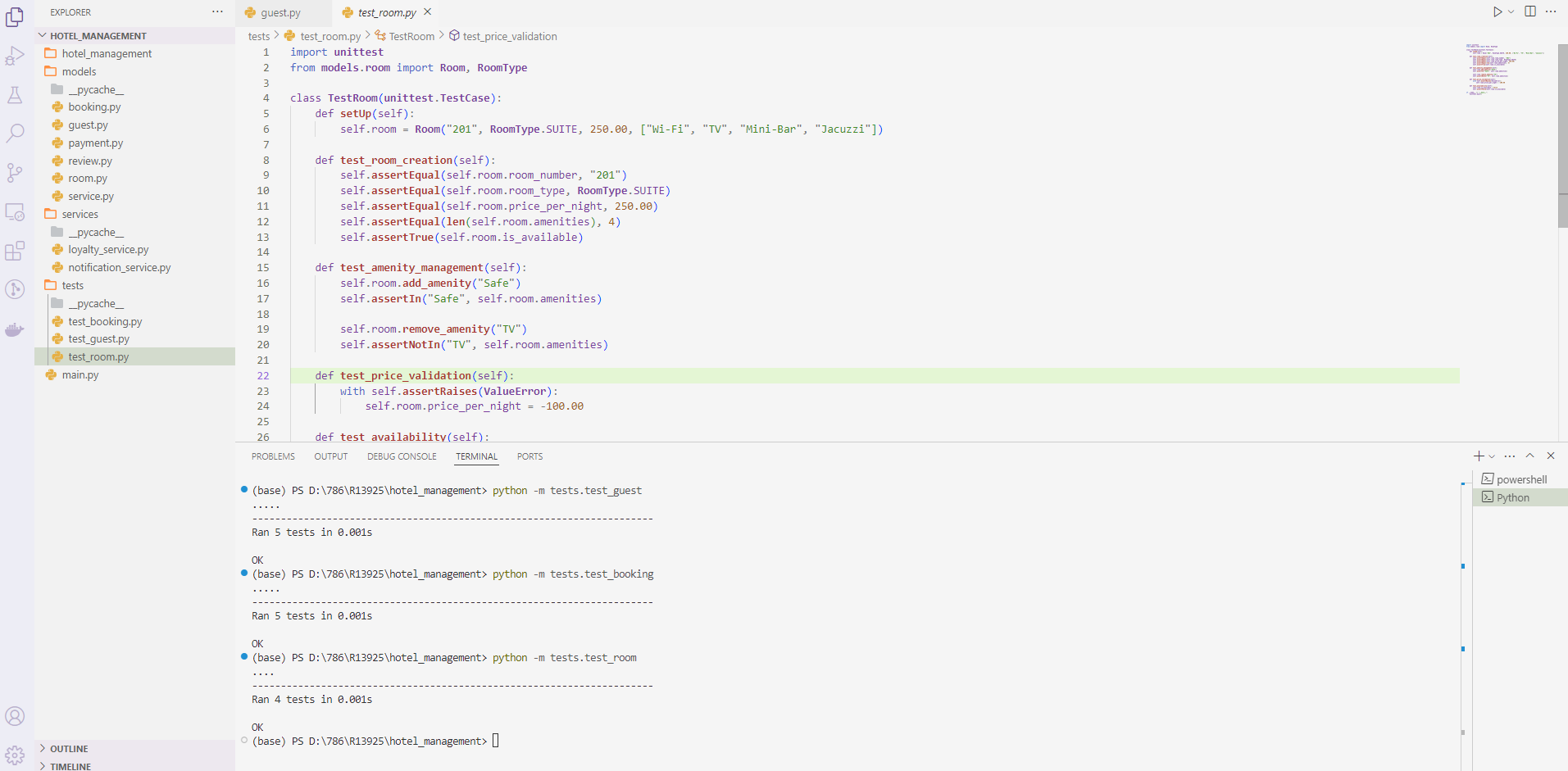
def **test\_availability**(self):

self.room.is\_available = False

self.**assertFalse**(self.room.is\_available)

if \_\_name\_\_ == "\_\_main\_\_":

**unittest**.**main**()



# **Github Repository**

# **Summary of Learnings**

Through the development of the Royal Stay Hotel Management System, I gained valuable insights into object-oriented programming (OOP) principles, including class design, encapsulation, inheritance, and polymorphism. The project reinforced the importance of modularity by separating concerns into distinct classes (e.g., `Guest`, `Room`, `Booking`) and utility services (e.g., `NotificationService`, `LoyaltyService`). I learned how to establish relationships between classes (one-to-one, one-to-many) and enforce data integrity through validation (e.g., ensuring check-out dates follow check-in dates). Additionally, implementing unit tests helped verify functionality and catch edge cases, improving code reliability. The project also highlighted the benefits of enumeration types (e.g., `RoomType`, `PaymentMethod`) for fixed-value attributes and dependency injection for flexible service integration. Overall, this experience strengthened my ability to design scalable, maintainable software systems using Python.