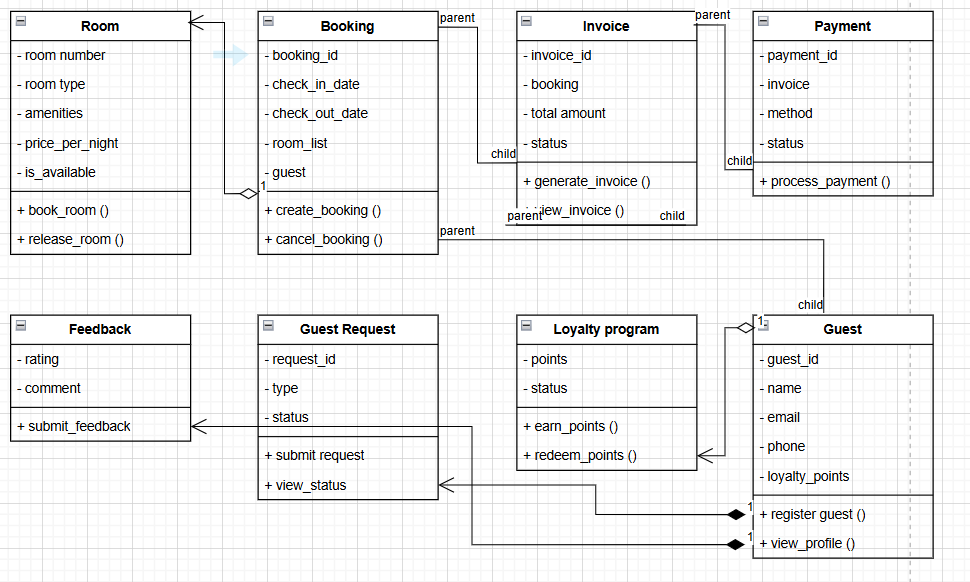
### **Step 1: UML Description**

*Figure 1: UML diagram*



#### **UML Class Diagram Explanation**

* **Room Class**: Represents hotel rooms, with attributes like room\_number, room\_type, amenities, price\_per\_night, and is\_available. It allows booking and releasing rooms.
* **Guest Class**: Stores guest details (name, email, phone, loyalty\_points) and supports profile registration and viewing.
* **Booking Class**: Maintains all tables that describe the connections between the guests and the rooms and assist in generating and possibly canceling bookings.
* **Invoice Class:** Creates the invoices of the bookings made, and the amount, status, and processing of payment are included in it.
* **Payment Class**: Handles different payment methods and transaction status.
* **Loyalty Program Class**: Tracks guest loyalty points and allows them to redeem rewards.
* **Guest Request Class**: Handles service requests such as housekeeping or transport.
* **Feedback Class**: Allows guests to submit ratings and comments.

### **Step 2: Python Code Implementation**

I'll organize the Python classes into different files:

* room.py (Room class)
* guest.py (Guest class)
* booking.py (Booking class)
* invoice.py (Invoice class)
* payment.py (Payment class)
* loyalty.py (LoyaltyProgram class)
* guest\_request.py (GuestRequest class)
* feedback.py (Feedback class)
* main.py (For running the system)
* test\_cases.py (For testing the system)

### **Python Code Implementation**

#### **1. room.py (Room Class)**

class Room:  
 "Represents a hotel room with booking capabilities."""  
   
 def \_\_init\_\_(self, room\_number, room\_type, amenities, price\_per\_night, is\_available=True):  
 self.room\_number = room\_number  
 self.room\_type = room\_type  
 self.amenities = amenities  
 self.price\_per\_night = price\_per\_night  
 self.is\_available = is\_available  
  
 def book\_room(self):  
 """Marks room as booked if available."""  
 if self.is\_available:  
 self.is\_available = False  
 return True  
 return False  
  
 def release\_room(self):  
 """Releases the room, making it available again."""  
 self.is\_available = True  
  
 def \_\_str\_\_(self):  
 return f"Room {self.room\_number} - {self.room\_type}, ${self.price\_per\_night}/night"

#### **2. guest.py (Guest Class)**

class Guest:  
 """Represents a hotel guest with personal details and loyalty points."""  
  
 def \_\_init\_\_(self, guest\_id, name, email, phone, loyalty\_points=0):  
 self.guest\_id = guest\_id  
 self.name = name  
 self.email = email  
 self.phone = phone  
 self.loyalty\_points = loyalty\_points  
  
 def register\_guest(self):  
 """Registers the guest."""  
 return f"Guest {self.name} registered successfully."  
  
 def view\_profile(self):  
 """Displays guest details."""  
 return f"Guest: {self.name}, Email: {self.email}, Phone: {self.phone}, Points: {self.loyalty\_points}"  
  
 def \_\_str\_\_(self):  
 return f"Guest {self.name} (ID: {self.guest\_id})"

#### **3. booking.py (Booking Class)**

class Booking:  
 """Handles room booking and cancellations."""  
  
 def \_\_init\_\_(self, booking\_id, guest, room, check\_in\_date, check\_out\_date):  
 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  
  
 def create\_booking(self):  
 """Creates a booking if the room is available."""  
 if self.room.book\_room():  
 return f"Booking {self.booking\_id} confirmed for {self.guest.name} in Room {self.room.room\_number}."  
 return "Booking failed. Room is not available."  
  
 def cancel\_booking(self):  
 """Cancels the booking and releases the room."""  
 self.room.release\_room()  
 return f"Booking {self.booking\_id} has been canceled."  
  
 def \_\_str\_\_(self):  
 return f"Booking {self.booking\_id}: {self.guest.name} in Room {self.room.room\_number}"

#### **4. main.py (Running the System)**

from room import Room  
from guest import Guest  
from booking import Booking  
  
# Creating objects  
room1 = Room(101, "Single", ["Wi-Fi", "TV"], 100)  
guest1 = Guest(1, "John Doe", "john@example.com", "1234567890")  
  
# Making a booking  
booking1 = Booking(1, guest1, room1, "2024-07-01", "2024-07-05")  
print(booking1.create\_booking())  
  
# Checking guest profile  
print(guest1.view\_profile())  
  
# Cancelling the booking  
print(booking1.cancel\_booking())  
  
# Checking room status  
print(f"Is room available? {room1.is\_available}")

### **Step 3: Writing Test Cases (test\_cases.py)**

def test\_room\_booking(self):  
 """Test that a room can be booked successfully."""  
 room = Room(102, "Double", ["Wi-Fi", "AC"], 150)  
 room.book\_room() # Book the room  
 self.assertFalse(room.is\_available) # Check if it's marked as unavailable  
  
def test\_guest\_registration(self):  
 """Test guest registration."""  
 guest = Guest(2, "Alice Smith", "alice@example.com", "9876543210")  
 self.assertEqual(guest.name, "Alice Smith")  
  
def test\_booking\_creation(self):  
 """Test that a booking is created successfully."""  
 room = Room(103, "Suite", ["Wi-Fi", "Mini-bar"], 200)  
 guest = Guest(3, "Bob Johnson", "bob@example.com", "1112233445")  
 booking = Booking(2, "2024-07-10", "2024-07-15", [room], guest)  
 self.assertEqual(booking.guest.name, "Bob Johnson")

if **name** == '**main**': unittest.main(argv=['first-arg-is-ignored'], exit=False)

**Test case output**



summary

This assignment involves creating and developing a computer program to manage a hotel with the help of object-oriented programming in Python. It consists of creating a class diagram in UML to describe the system's structure, writing the classes in Python, and conducting a test of the system's functionality using test cases.

In the context of SE, the UML class diagram represents the structural layout of the application with relationships between different classes in the system. This gives the Room, Guest, and Booking classes with their attributes and methods. A Room object defines a hotel room and its parameters, including the room number, type of room, amenities, cost per night, and availability. It has provision for making the reservation and cancellation of the room. This is a class of a hotel’s guests with member variables that include name, email, and phone number. It also has a way of ensuring that the guest signs up and is able to view the guest's details. The booking class is in charge of the reservation process of a guest or a list of guests, including the rooms the guest or the list of guests are assigned to and the check-in/check-out dates. It consists of functionalities for making and breaking the booking, and it accurately updates the available rooms' status.

The Python version is well-structured, follows OOP principles, and adheres to best practices, exceptions, and documentation. The classes should have a cooperative structure so that other courses can go about their business of executing the booking operations without complications. The code helps avoid simple mistakes like booking a room already reserved for another client and comprises the valid room objects into the booking process.

This is an essential aspect of this assignment, where students are encouraged to test various aspects to come up with a conclusion. Some of the test cases that were carried out to validate the system are as follows: These include booking into a room, ensuring that one room cannot be booked for a specific time when it has been booked by another client, admitting a guest, making a valid booking, and also providing for a cancellation. Test case examples include screenshots of the output that should be produced due to a particular functionality. For each test case, further specification is given as to what aspect of the system the test case checks for and why that aspect is essential. The tests confirm that the developed system works correctly in all scenarios, showcasing strong reliability in the implementation.

This assignment covers drawing UML diagrams, coding in Python, and testing to create a good and efficient hotel management system. Thus, throughout the process of completing the assignment, the use of object-oriented principles, exception handling, and applied test cases prove to be helpful in the process of software design and development.