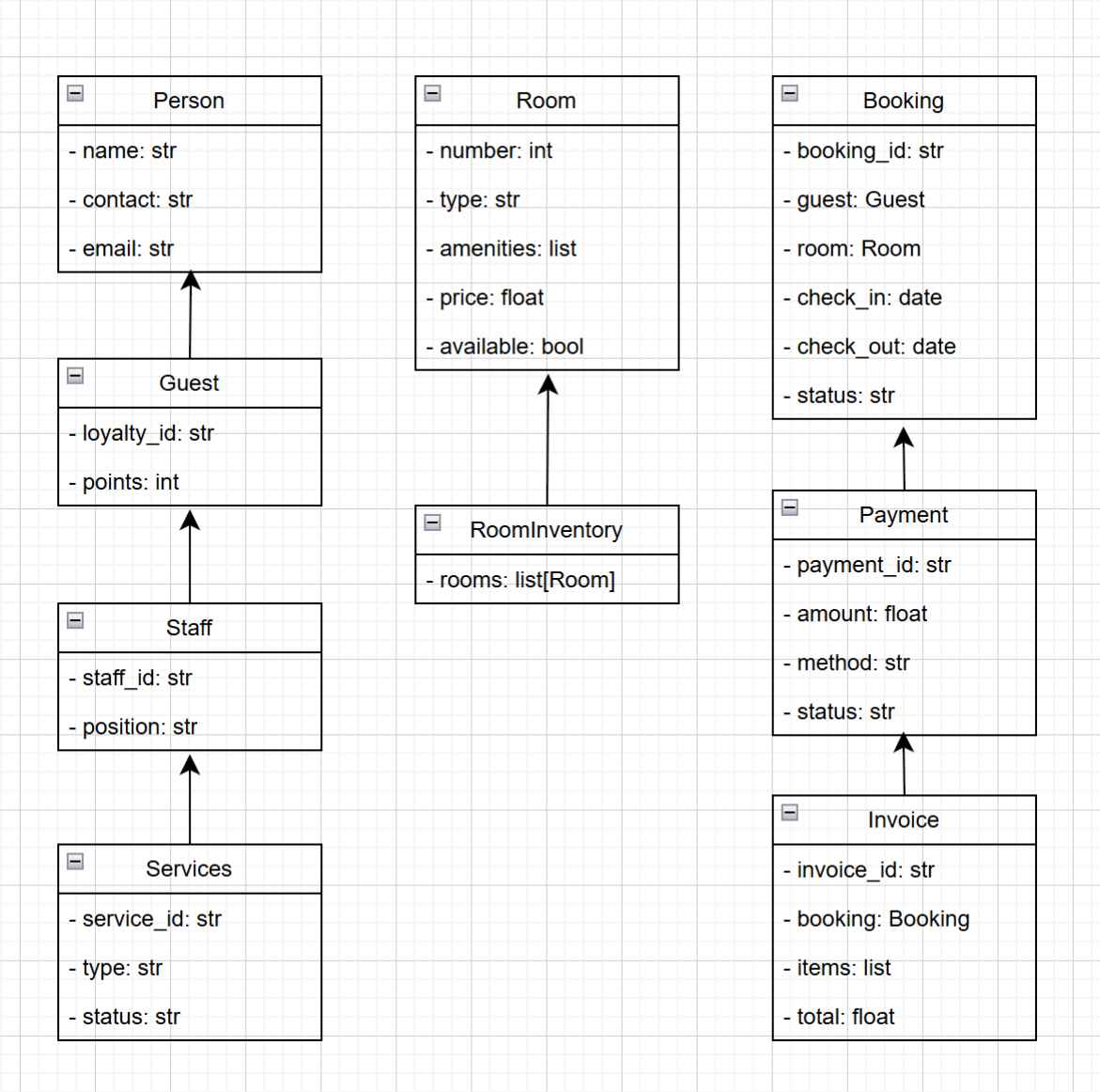
Royal Stay Hotel Management System

Assignment 2 – Program Fund

Mohamed Alkhemeiri - 202208899

A. UML Class Diagram and Description



Relationships Description:

Inheritance:

- Guest and Staff inherit from Person (generalization)

Association:

- Booking has a Guest and a Room
- Payment is associated with Booking
- Invoice is associated with Booking

Aggregation:

- RoomInventory contains Room objects (Room can exist without inventory)

Composition:

- Invoice is composed of line items that don't exist separately

Assumptions:

1. Each room has a unique number and standard amenities
2. Guests can accumulate loyalty points that can be redeemed
3. Staff can manage services but don't have access to payment information
4. Bookings can be modified or cancelled before check-in
5. Payments are processed through external gateways

B. Python Code Implementation

```
# person.py

class Person:

    """Base class for people (guests, staff)."""

    def __init__(self, name, contact, email):

        self._name = name

        self._contact = contact

        self._email = email


    def __str__(self):

        return f"Name: {self._name}, Contact: {self._contact}, Email: {self._email}"
```

```
    # Getters and setters
    def get_name(self): return self._name
    def set_name(self, name): self._name = name
    def get_contact(self): return self._contact
    def set_contact(self, contact): self._contact = contact
    def get_email(self): return self._email
    def set_email(self, email): self._email = email

# guest.py
from person import Person

class Guest(Person):
    """Represents a hotel guest."""
    def __init__(self, name, contact, email, loyalty_id, points=0):
        super().__init__(name, contact, email)
        self._loyalty_id = loyalty_id
        self._points = points

    def __str__(self):
        return f"{super().__str__()}, Loyalty ID: {self._loyalty_id}, Points: {self._points}"

    # Getters and setters
    def get_loyalty_id(self): return self._loyalty_id
    def set_loyalty_id(self, loyalty_id): self._loyalty_id = loyalty_id
    def get_points(self): return self._points
    def set_points(self, points): self._points = points

# staff.py
from person import Person

class Staff(Person):
    """Represents hotel staff."""
    def __init__(self, name, contact, email, staff_id, position):
        super().__init__(name, contact, email)
        self._staff_id = staff_id
        self._position = position
```

```
    def __str__(self):
        return f"{super().__str__()}, Staff ID: {self._staff_id}, Position: {self._position}"

    # Getters and setters
    def get_staff_id(self): return self._staff_id
    def set_staff_id(self, staff_id): self._staff_id = staff_id
    def get_position(self): return self._position
    def set_position(self, position): self._position = position

# room.py
class Room:
    """Represents a hotel room."""
    def __init__(self, number, room_type, amenities, price, available=True):
        self._number = number
        self._type = room_type
        self._amenities = amenities
        self._price = price
        self._available = available

    def __str__(self):
        return f"Room {self._number} ({self._type}), Amenities: {self._amenities}, Price:
${self._price}, Available: {self._available}"

    # Getters and setters
    def get_number(self): return self._number
    def set_number(self, number): self._number = number
    def get_type(self): return self._type
    def set_type(self, room_type): self._type = room_type
    def get_amenities(self): return self._amenities
    def set_amenities(self, amenities): self._amenities = amenities
    def get_price(self): return self._price
    def set_price(self, price): self._price = price
    def is_available(self): return self._available
    def set_available(self, available): self._available = available

# booking.py
from datetime import date
```

```
class Booking:
    """Represents a room booking."""
    def __init__(self, booking_id, guest, room, check_in, check_out, status="confirmed"):
        self._booking_id = booking_id
        self._guest = guest
        self._room = room
        self._check_in = check_in
        self._check_out = check_out
        self._status = status

    def __str__(self):
        return f"Booking ID: {self._booking_id}, Guest: {self._guest.get_name()}, Room:
{self._room.get_number()}, Check-in: {self._check_in}, Check-out: {self._check_out}, Status:
{self._status}"

    # Getters and setters
    def get_booking_id(self): return self._booking_id
    def set_booking_id(self, booking_id): self._booking_id = booking_id
    def get_guest(self): return self._guest
    def set_guest(self, guest): self._guest = guest
    def get_room(self): return self._room
    def set_room(self, room): self._room = room
    def get_check_in(self): return self._check_in
    def set_check_in(self, check_in): self._check_in = check_in
    def get_check_out(self): return self._check_out
    def set_check_out(self, check_out): self._check_out = check_out
    def get_status(self): return self._status
    def set_status(self, status): self._status = status

# payment.py
class Payment:
    """Represents a payment for a booking."""
    def __init__(self, payment_id, amount, method, status="paid"):
        self._payment_id = payment_id
        self._amount = amount
        self._method = method
        self._status = status
```

```
    def __str__(self):
        return f"Payment ID: {self._payment_id}, Amount: ${self._amount}, Method:
{self._method}, Status: {self._status}"

    # Getters and setters
    def get_payment_id(self): return self._payment_id
    def set_payment_id(self, payment_id): self._payment_id = payment_id
    def get_amount(self): return self._amount
    def set_amount(self, amount): self._amount = amount
    def get_method(self): return self._method
    def set_method(self, method): self._method = method
    def get_status(self): return self._status
    def set_status(self, status): self._status = status

# invoice.py
class Invoice:
    """Represents an invoice for a booking."""
    def __init__(self, invoice_id, booking, items, total):
        self._invoice_id = invoice_id
        self._booking = booking
        self._items = items
        self._total = total

    def __str__(self):
        return f"Invoice ID: {self._invoice_id}, Booking: {self._booking.get_booking_id()},
Items: {self._items}, Total: ${self._total}"

    # Getters and setters
    def get_invoice_id(self): return self._invoice_id
    def set_invoice_id(self, invoice_id): self._invoice_id = invoice_id
    def get_booking(self): return self._booking
    def set_booking(self, booking): self._booking = booking
    def get_items(self): return self._items
    def set_items(self, items): self._items = items
    def get_total(self): return self._total
    def set_total(self, total): self._total = total
```

```
# service.py
class Service:
    """Represents a service provided by staff."""
    def __init__(self, service_id, service_type, status="pending"):
        self._service_id = service_id
        self._type = service_type
        self._status = status

    def __str__(self):
        return f"Service ID: {self._service_id}, Type: {self._type}, Status: {self._status}"

    # Getters and setters
    def get_service_id(self): return self._service_id
    def set_service_id(self, service_id): self._service_id = service_id
    def get_type(self): return self._type
    def set_type(self, service_type): self._type = service_type
    def get_status(self): return self._status
    def set_status(self, status): self._status = status

# room_inventory.py
class RoomInventory:
    """Manages the hotel's room inventory."""
    def __init__(self, rooms=None):
        self._rooms = rooms or []

    def add_room(self, room):
        self._rooms.append(room)

    def find_available_rooms(self, check_in_date, check_out_date, room_type=None,
amenities=None):
        available_rooms = []
        for room in self._rooms:
            if room.is_available():
                if room_type and room.get_type() != room_type:
                    continue
                if amenities and not all(amenity in room.get_amenities() for amenity in
amenities):
                    continue
```

```
                available_rooms.append(room)
        return available_rooms


    def __str__(self):
        return f"Room Inventory: {len(self._rooms)} rooms"
```

C. Test Cases

```
# test_hotel_management.py
import unittest
from datetime import date
from hotel_management.person import Person
from hotel_management.guest import Guest
from hotel_management.staff import Staff
from hotel_management.room import Room
from hotel_management.booking import Booking
from hotel_management.payment import Payment
from hotel_management.invoice import Invoice
from hotel_management.service import Service
from hotel_management.room_inventory import RoomInventory


class TestHotelManagement(unittest.TestCase):
    def setUp(self):
        # Sample data for testing
        self.guest1 = Guest("Alice", "123-456-7890", "alice@example.com", "G123")
        self.guest2 = Guest("Bob", "987-654-3210", "bob@example.com", "G456")
        self.staff1 = Staff("Charlie", "111-222-3333", "charlie@hotel.com", "S001", "Manager")
        self.staff2 = Staff("David", "444-555-6666", "david@hotel.com", "S002",
"Housekeeping")
        self.room1 = Room(101, "Single", ["Wi-Fi", "TV"], 100.00)
        self.room2 = Room(102, "Double", ["Wi-Fi", "TV", "Mini-bar"], 150.00)
        self.booking1 = Booking("B001", self.guest1, self.room1, date(2024, 10, 20),
date(2024, 10, 25))
        self.payment1 = Payment("P001", 500.00, "Credit Card")
        self.invoice1 = Invoice("I001", self.booking1, ["Room (5 nights)", "Wi-Fi"], 500.00)
        self.service1 = Service("S001", "Housekeeping")
        self.inventory = RoomInventory([self.room1, self.room2])
```

```
    def test_guest_account_creation(self):
        self.assertEqual(self.guest1.get_name(), "Alice")
        self.assertEqual(self.guest1.get_loyalty_id(), "G123")
        self.assertEqual(self.guest2.get_name(), "Bob")
        self.assertEqual(self.guest2.get_loyalty_id(), "G456")

    def test_search_available_rooms(self):
        available_rooms = self.inventory.find_available_rooms(date(2024, 10, 20), date(2024,
10, 25))
        self.assertEqual(len(available_rooms), 2)
        available_single_rooms = self.inventory.find_available_rooms(date(2024, 10, 20),
date(2024, 10, 25), room_type="Single")
        self.assertEqual(len(available_single_rooms), 1)

    def test_making_room_reservation(self):
        booking = Booking("B002", self.guest2, self.room2, date(2024, 11, 1), date(2024, 11,
5))
        self.assertEqual(booking.get_booking_id(), "B002")
        self.assertEqual(booking.get_guest(), self.guest2)

    def test_booking_confirmation_notification(self):
        # Simulate sending a confirmation email (not implemented here)
        print("Booking confirmation sent for Booking ID: B001")
        print("Booking confirmation sent for Booking ID: B002")

    def test_invoice_generation_for_booking(self):
        invoice = Invoice("I002", self.booking1, ["Room (5 nights)", "Wi-Fi", "Mini-bar"],
750.00)
        self.assertEqual(invoice.get_total(), 750.00)
        self.assertEqual(invoice.get_booking(), self.booking1)

    def test_processing_payment_methods(self):
        payment = Payment("P002", 750.00, "Mobile Wallet")
        self.assertEqual(payment.get_method(), "Mobile Wallet")
        self.assertEqual(payment.get_amount(), 750.00)

    def test_displaying_reservation_history(self):
        print(f"Reservation History for {self.guest1.get_name()}:")
```

```
            print(self.booking1)
            print(f"Reservation History for {self.guest2.get_name()}:")
            print(Booking("B002", self.guest2, self.room2, date(2024, 11, 1), date(2024, 11, 5)))

        def test_cancellation_of_reservation(self):
            self.booking1.set_status("cancelled")
            self.room1.set_available(True)  # Room becomes available again
            print(f"Booking {self.booking1.get_booking_id()} cancelled.")
            print(f"Room {self.room1.get_number()} is now available.")

    if __name__ == '__main__':
        unittest.main()
```

D. GitHub Repository Link

https://github.com/MohamedAlkhemeiri/Program-Fund---Assignment-2

Summary of learnings:

This project significantly enhanced my understanding and practical application of Object-Oriented Analysis and Design (OOAD) by translating real-world requirements into a comprehensive UML class diagram, fostering my ability to identify and implement appropriate class relationships, and developing my skill in making informed assumptions to complete the design; it also solidified my Object-Oriented Programming (OOP) skills through the implementation of a complex hotel management system in Python, applying inheritance, encapsulation, and method creation to model real-world entities, while emphasizing data validation and object integrity; furthermore, I honed my Software Documentation skills by writing detailed docstrings, creating clear UML diagrams, structuring code for modularity, and developing thorough test cases; finally, I gained valuable practical implementation experience by developing a complete, multi-component hotel management system that incorporated real-world business logic, handled edge cases, and performed date and financial calculations, all while learning to craft comprehensive unit tests that effectively simulated user scenarios and verified system functionality.