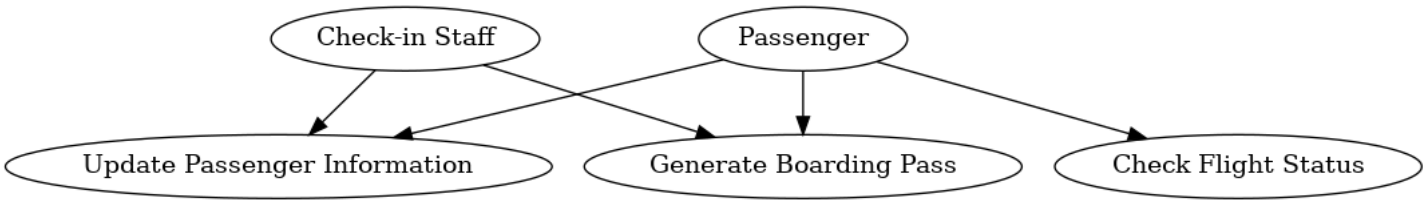
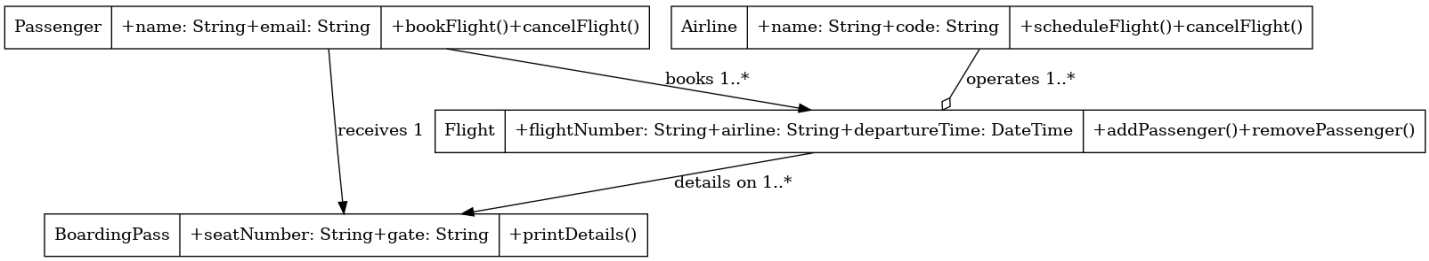
Question 1: Identify the use-cases for the software. Draw the UML use-case diagram and include supporting use-case descriptions. At least 3 scenarios must be identified. Use Case 1: Update Passenger Information Primary Actor: Check-in Staff Goal: To ensure that the passenger's personal and travel information is up-to-date before flight check-in. Scenario: The check-in staff accesses the airline's check-in system. The staff selects the option to update passenger information. The staff searches for the passenger's booking using their name or booking reference. Upon finding the correct booking, the staff updates the necessary information, such as contact details, passport information, or special needs requests. The system confirms the updates and saves the changes to the passenger's booking. Use Case 2: Generate Boarding Pass Primary Actor: Passenger Goal: To receive a boarding pass after check-in that allows the passenger to board the aircraft. Scenario: The passenger approaches the check-in counter or uses an online check-in service. The check-in staff or the online system verifies the passenger's booking and travel documents. Once verified, the system processes the check-in and assigns a seat to the passenger based on availability and preference. The system generates a boarding pass with the passenger's flight details, seat number, boarding group, and boarding time. The passenger receives the boarding pass either printed by the check-in staff or electronically through the online check-in service. Use Case 3: Check Flight Status Primary Actor: Passenger Goal: To obtain real-time information about the status of the flight. Scenario: The passenger wishes to confirm the departure time and gate information of their flight. The passenger uses the airline's mobile app, website, or airport information displays to check the status of their flight. The system provides the current status of the flight, including any delays, gate changes, or cancellations. The passenger uses this information to plan their arrival at the airport or make necessary adjustments to their travel plans.



Question 2: Identify the objects and their respective classes. Draw the UML class diagrams and include supporting descriptions to explain the relationships. At least 4 classes and respective relationships must be identified.Classes and Attributes: Passenger Attributes: name: String, email: String Operations: bookFlight(), cancelFlight() Description: Represents an individual who uses the airline's services. BoardingPass Attributes: seatNumber: String, gate: String Operations: printDetails() Description: Represents the boarding pass issued to a passenger, containing the seat number and gate information. Flight Attributes: flightNumber: String, airline: String, departureTime: DateTime Operations: addPassenger(), removePassenger() Description: Represents a specific flight with a unique number, associated airline, and scheduled departure time. Airline Attributes: name: String, code: String Operations: scheduleFlight(), cancelFlight() Description: Represents the airline company operating flights. Relationships: Passenger to BoardingPass Relationship: Receives (1 to 1) Description: Each passenger receives exactly one boarding pass. Passenger to Flight Relationship: Books (1 to many) Description: A passenger can book multiple flights. Flight to Airline Relationship: Operates (1 to many) Description: Each flight is operated by one airline, and an airline can operate multiple flights. Flight to BoardingPass Relationship: Details on (1 to many) Description: Each flight has many boarding passes issued for it, one for each passenger. Airline to Flight Relationship: Books (1 to many) Description: An airline can book passengers on many flights. This relationship is typically represented differently (e.g., an airline offers many flights), but it seems to be depicted as "books" in this diagram.



Question 3: For all the identified classes, create Python classes with the constructor, attributes (at least 5), and required setter/getter methods. Identify and include other required function headers in the classes where the function's body is just a pass statement and include a comment to indicate what the function should achieve. Question 4: Create objects of all the identified classes and use the object's functions to populate and display all the boarding pass details shown in the figure. Class: Passenger Attributes: name (String), email (String) Methods: bookFlight(), cancelFlight() Relationships: receives: This is a relationship between the Passenger and the BoardingPass, indicating that a Passenger receives exactly 1 BoardingPass. Class: BoardingPass Attributes:

seatNumber (String), gate (String) Methods: None listed Relationships: receives: As mentioned, this is the inverse relationship where a BoardingPass is received by a Passenger. Class: Flight Attributes: flightNumber (String), airline (String), departureTime (DateTime) Methods: printDetails(), addPassenger(), removePassenger() Relationships: details on: A Flight has a one-to-many relationship with BoardingPass, indicated by "details on 1..*", which suggests that one Flight instance can have details on many BoardingPass instances. This implies that each boarding pass is linked to a specific flight. books: A one-to-many relationship with the Airline, suggesting that one Flight can be booked under many instances of Airline (though this is typically not the case in real-world scenarios, it may represent code-sharing or partnerships between airlines). Class: Airline Attributes: name (String), code (String) Methods: scheduleFlight(), cancelFlight() Relationships: books: The inverse of the relationship with Flight, indicating that an Airline can book many flights. operates: A one-to-many relationship with Flight, suggesting that an Airline operates many flights.

In [9]:

```
class Passenger:
    def __init__(self, name, passport_number, frequent_flyer_number, seat_preference, spe
cial_needs):
        self.name = name
        self.passport_number = passport_number
        self.frequent_flyer_number = frequent_flyer_number
        self.seat_preference = seat_preference
        self.special_needs = special_needs

    # Setters and getters for each attribute
    def set_name(self, name):
        self.name = name

    def get_name(self):
        return self.name

    # Similar setter and getter methods will be created for the other attributes
    # ...

    # Example of a required function header
    def update_information(self):
        """Update passenger's personal information"""
        pass

class Flight:
    def __init__(self, flight_number, destination, departure_time, gate, aircraft_type, t
erminal):
        self.flight_number = flight_number
        self.destination = destination
        self.departure_time = departure_time
        self.gate = gate
        self.aircraft_type = aircraft_type
        self.terminal = terminal

    # Setters and getters for each attribute
    # ...

    # Example of a required function header
    def delay_flight(self, delay_time):
        """Delay the flight by a certain amount of time"""
        pass

class BoardingPass:
    def __init__(self, passenger, flight, seat_number, boarding_group, boarding_time, cla
ss_of_service):
        self.passenger = passenger
        self.flight = flight
        self.seat_number = seat_number
        self.boarding_group = boarding_group
        self.boarding_time = boarding_time
        self.class_of_service = class_of_service

    # Setters and getters for each attribute
    # ...

    # Example of a required function header
```

```
    def update_seat(self, new_seat_number):
        """Update the seat number on the boarding pass"""
        pass

class AirlineStaff:
    def __init__(self, employee_id, name, position, years_of_experience, base_airport):
        self.employee_id = employee_id
        self.name = name
        self.position = position
        self.years_of_experience = years_of_experience
        self.base_airport = base_airport

    # Setters and getters for each attribute
    # ...

    # Example of a required function header
    def check_in_passenger(self, passenger):
        """Check in a passenger and generate a boarding pass"""
        pass

# Create a Passenger object
passenger = Passenger(
    name="JAMES SMITH",
    passport_number=None,  # Not visible on the boarding pass
    frequent_flyer_number=None,  # Not visible on the boarding pass
    seat_preference=None,  # Not visible on the boarding pass
    special_needs=None  # Not visible on the boarding pass
)

# Create a Flight object
flight = Flight(
    flight_number="NA4321",
    destination="NEW YORK JFK",
    departure_time="2020-12-06 11:40",
    gate="03",
    aircraft_type=None,  # This can be set to a string if the aircraft type is known
    terminal="2"
)

# Create an AirlineStaff object
boarding_pass = BoardingPass(
    passenger=passenger,
    flight=flight,
    seat_number="09A",
    boarding_group=None,  # Boarding group is not specified in the provided boarding pass
    boarding_time="2020-12-06 11:20",
    class_of_service="First Class"  # Class of service as shown on the boarding pass
)

print("Boarding Pass Details:")
print("Passenger Name:", boarding_pass.passenger.get_name())
print("Flight Number:", boarding_pass.flight.flight_number)
print("Class of Service:", boarding_pass.class_of_service)  # Now including class of serv
ice
print("Terminal:", flight.terminal)
print("From:", "CHICAGO ORD")
print("To:", boarding_pass.flight.destination)
print("Date:", boarding_pass.flight.departure_time.split(" ")[0])
print("Departure Time:", boarding_pass.flight.departure_time.split(" ")[1])
print("Gate:", boarding_pass.flight.gate)
print("Seat Number:", boarding_pass.seat_number)
print("Boarding Time:", boarding_pass.boarding_time.split(" ")[1])
```

```
Boarding Pass Details:
Passenger Name: JAMES SMITH
Flight Number: NA4321
Class of Service: First Class
Terminal: 2
From: CHICAGO ORD
To: NEW YORK JFK
Date: 2020-12-06
Departure Time: 11:40
```

```
Gate: 03
Seat Number: 09A
Boarding Time: 11:20
```

- Summary of Learnings: #LO1_OOAD: When designing a boarding pass system I used object oriented analysis and design (OOAD) principles to represent real world elements and connections in a software model using Unified Modeling Language (UML) symbols. Initially I pinpointed the elements, in the system like Passenger, Flight, BoardingPass and AirlineStaff. These elements were then transformed into classes each with their characteristics and functions that define their roles. To show how these classes are related I created a UML class diagram showing attributes, operations and connections such as inheritance and aggregations. The application of UML provided a view of the systems layout and how its parts interacted with each other setting a base, for the next development stage. #LO2_OOProgramming: After laying the groundwork, with OOAD principles I moved on to the object oriented programming phase to develop a prototype for the boarding pass system. I crafted Python classes corresponding to each entity outlined in the UML diagram organizing their attributes and behaviors within these structures. Utilizing constructors to set up objects and implementing getter and setter methods ensured that class attributes were properly encapsulated. Furthermore I incorporated function headers as placeholders for implementation using pass statements along with comments outlining their intended functions. This approach led to a organized and coherent codebase that could be easily expanded upon. Through creating instances of these classes and calling their methods I showcased the programs ability to manage and present boarding pass details flawlessly highlighting its capacity for error operation, in addressing the designated task. GitHub Repository Link: https://github.com/7ammm/Assignment-1-Software-Modelling---UML-Use-Case-and-UML-Class-diagrams-MS7.git