**Boarding Pass Generation System**

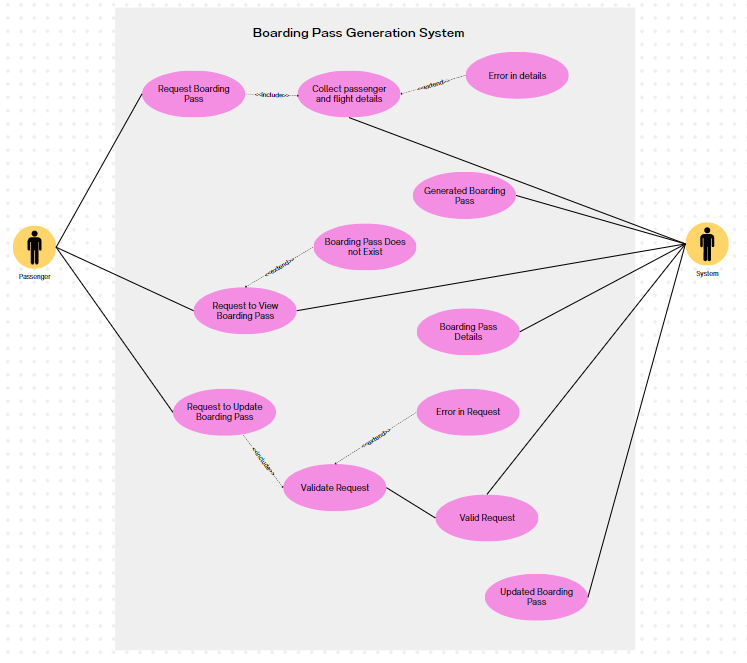
**Student Name:**

**Teacher Name:**

**Student ID:**

**Dated:**

1. **UML Use Case Diagram**

****

**Actors**

There are 2 actors within the Boarding Pass Generation System. These are:

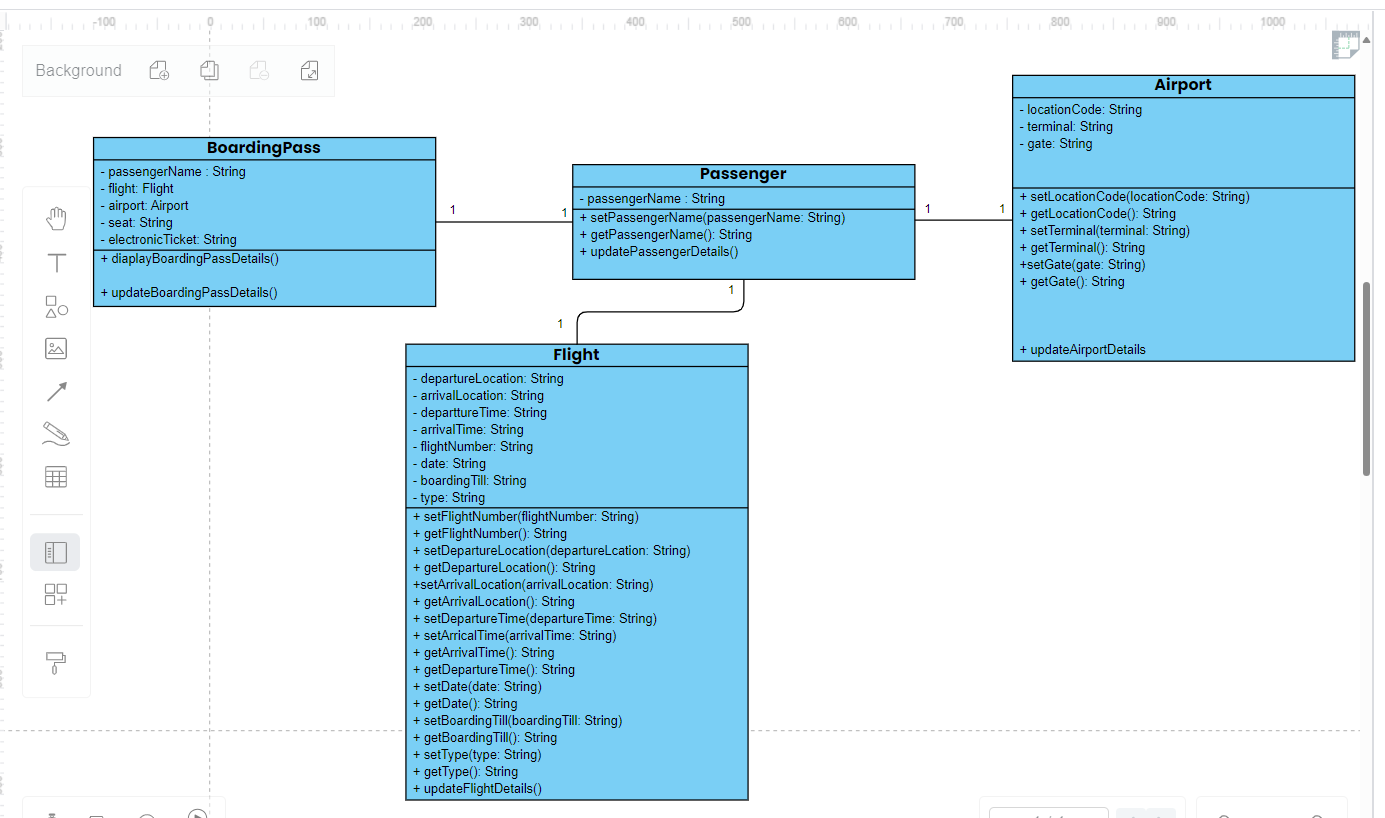
1. **Passenger:** Represents the person who interacts with the boarding pass generation system.
2. **System:** Represents the boarding pass generation system itself.

**Use Cases**

There are 3 use cases in the system. These are:

1. **Request Boarding Pass:** The Passenger initiates this process by requesting a boarding pass. Then the user moves to enter relevant details such as passenger information, flight details, and boarding time. In case of an error in response e.g. incorrect details an error is generated. After the user details are verified by the system the boarding pass is generated.
2. **Request to View Boarding Pass:** The passenger initiates by sending a request to view boarding pass details. In case such a user makes a request whose boarding pass does not exist an error is displayed. If the boarding pass exists, the details are shared with the user by the system.
3. **Request to Update Boarding Pass:** In this scenario user begins by generating a request to modify the boarding pass maybe due to any change in flight/airport. After the validation of the request, the system updates the requested changes.

**2. UML Class Diagram**

****

There are 4 different classes in the above UML class diagram. Following is a description of the classes:  
The provided code defines several classes for a boarding pass generation system. Let's go through each class:

**Passenger Class**

The `Passenger` class represents an individual traveler using the boarding pass generation system. It has a single attribute, `passengerName`, which holds the name of the passenger. The class provides setter and getter methods for updating and retrieving the passenger's name, respectively. Additionally, there is a placeholder method, `updatePassengerDetails`, designed for future implementation, allowing the passenger's details to be updated.

**Flight Class**

The `Flight` class encapsulates information about a specific flight within the boarding pass generation system. It contains attributes such as `flightNumber`, `departureTime`, `arrivalTime`, `departureLocation`, `arrivalLocation`, `date`, `boardingTill`, and `type`. Each attribute is associated with a corresponding setter and getter method, facilitating the modification and retrieval of flight details. The class includes a placeholder method, `updateFlightDetails`, intended for future implementation to update the flight schedule.

**Airport Class**

The `Airport` class represents an airport and holds information about its location. It includes attributes like `locationCode`, `terminal`, and `gate`. The class provides setter and getter methods for updating and retrieving the airport details. Similar to other classes, it features a placeholder method, `updateAirportDetails`, which can be implemented later to update information related to the airport.

**BoardingPass Class**

The `BoardingPass` class serves as a composite class, bringing together passenger, flight, and airport information into a single entity representing a boarding pass. It contains attributes such as `passenger`, `flight`, `airport`, `seat`, and `electronicTicket`. The class provides a method, `updateBoardingPassDetails`, intended for future implementation to update specific details on the boarding pass. The `displayBoardingPassDetails` method prints detailed information about the boarding pass, including passenger, flight, airport, seat, and electronic ticket details.

**Relationship between classes**

As evident from diagram, the classes have association relationship. BoardingPass has associations with Passenger, Flight, and Airport. This indicates that a boarding pass is composed of a passenger, a flight, and airport information.

**3. Python Code**

Following is the Python code implemented for the 4 classes:

#passenger class

class Passenger:

def \_\_init\_\_(self, passengerName):

self.passengerName = passengerName

def setPassengerName(self, passengerName):

self.passengerName = passengerName

def getPassengerName(self):

return self.passengerName

def updatePassengerDetails(self):

#update passenger's name

pass

#flight class

class Flight:

def \_\_init\_\_(self, flightNumber, departureTime, arrivalTime, departureLocation, arrivalLocation, date, boardingTill, type):

self.flightNumber = flightNumber

self.departureTime = departureTime

self.arrivalTime = arrivalTime

self.departureLocation = departureLocation

self.arrivalLocation = arrivalLocation

self.date = date

self.boardingTill = boardingTill

self. type = type

def setType(self, type):

self.type = type

def getType(self):

return self.type

def setFlightNumber(self, flightNumber):

self.flightNumber = flightNumber

def getFlightNumber(self):

return self.flightNumber

def setDepartureTime(self, departureTime):

self.departureTime = departureTime

def getDepartureTime(self):

return self.departureTime

def setDate(self, date):

self.date = date

def getDate(self):

return self.date

def setBoardingTill(self, boardingTill):

self.boardingTill = boardingTill

def getBoardingTill(self):

return self.boardingTill

def setArrivalTime(self, arrivalTime):

self.arrivalTime = arrivalTime

def getArrivalTime(self):

return self.arrivalTime

def setDepartureLoctaion(self, departureLocation):

self.departureLocation = departureLocation

def getDepartureLocation(self):

return self.departureLocation

def setArrivalLoctaion(self, arrivalLocation):

self.arrivalLocation = arrivalLocation

def getArrivalLocation(self):

return self.arrivalLocation

def updateFlightDetails(self):

#update flight schedule

pass

#Airport class

class Airport:

def \_\_init\_\_(self, locationCode, terminal, gate):

self.locationCode = locationCode

self.terminal = terminal

self.gate = gate

def setLocationCode(self, locationCode):

self.locationCode = locationCode

def getLocationCode(self):

return self.locationCode

def setTerminal(self, terminal):

self.terminal = terminal

def getTerminal(self):

return self.terminal

def setGate(self, gate):

self.gate = gate

def getGate(self):

return self.gate

def updateAirportDetails(self):

#update airport details

pass

#BoardingPass Class

class BoardingPass:

def \_\_init\_\_(self, passenger, flight, airport, seat, electronicTicket):

self.passenger = passenger

self.flight = flight

self.airport = airport

self.seat = seat

self.electronicTicket = electronicTicket

def updateBoardingPassDetails(self):

#update Boarding Pass Details

pass

def displayBoardingPassDetails(self):

print('-----NATIONAL AIRLINES-----')

print('--Passenger Class Data--')

print('Passenger Name: ', self.passenger.getPassengerName())

print('--Flight Class Data--')

print('Type: ', self.flight.getType())

print('Number: ', self.flight.getFlightNumber())

print('Departure Date: ', self.flight.getDate())

print('Boarding Till: ', self.flight.getBoardingTill())

print('Departure Time: ', self.flight.getDepartureTime())

print('Departure Location: ', self.flight.getDepartureLocation())

print('Arrival Time: ', self.flight.getArrivalTime())

print('Arrival Location: ', self.flight.getArrivalLocation())

print('--Airport Class Data--')

print('Loctaion Code: ', self.airport.getLocationCode())

print('Terminal: ' ,self.airport.getTerminal())

print('Gate: ', self.airport.getGate())

print('--BoardingPass Class Data--')

print('Seat: ', self.seat)

print('Electronic Ticket: ', self.electronicTicket)

This here is where objects of the 4 different classes are created and a display function is called to print the boarding pass details:  
#main

p1 = Passenger('JAMES SMITH')

f1 = Flight('NA4321', '11:40', '13:30','CHICAGO ORD', 'NEW YORK JFK', '06 DEC 20', '11:20', 'First Class')

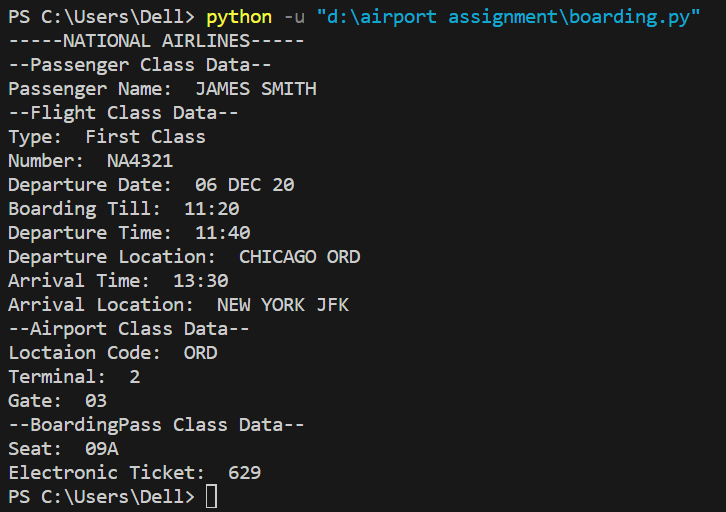
a1 = Airport('ORD', '2', '03') # 'ORD' from 'CHICAGO ORD'

b1 = BoardingPass(p1, f1, a1, '09A', '629')

b1.displayBoardingPassDetails()

**Output**

Following is the output generated by the above code:



1. **Github Repository Link**
2. **Summary of Learnings**

I learned a couple of things of important things from this assignment. To begin with I got a hands-on experience of UML diagrams (use case and class diagram). I came to know about the usefulness of these 2 diagrams. Specifically, how they ease the understanding of a complex problem. Also, the concepts of relationships in the UML class diagram aided in understanding OOP concepts. I learned about Python and OOP. The basics of OOP and Python syntax were explored during this assignment. Lastly, I realized the importance of GitHub and its significance in exploring complex problems and coming up with different solutions.