DOKUZ EYLUL UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

CME 2210

Object Oriented Analysis and Design

AIRPORT FLIGHT MANAGEMENT AND TRACING SYSTEM

by

İdrishan Parlayan

Rıdvan Özdemir

Alp Ercan Gültekin

Lecturers

Assoc. Prof. Semih Utku

PhD. Okan Öztürkmenoğlu

PhD. Yunus Dogan

IZMIR

10.05.2020

CHAPTER ONE

INTRODUCTION

AFMTS (Airport Flight Management and Tracing System) is a software program that helps tboth of the airport personnel and passengers by increasing the quality of IT service delivery. The main objective of AFMTS is tincrease the satisfaction of airport's personnel and passengers. AFMTS software achieves it's main objective by providing opportunity for the airport personnel tmanage flights conveniently and alsby providing opportunity for the passengers tsee the existing flights and tsearch suitable tickets for themselves easily. Under favour of AFMTS software, the flight planning personnel can easily trace the suitable pilots and cabin crew ta specific flight and appoint them for that flight. And alsAFMTS can calculate the the traffic on the pist for a given date/time and it solves delay problems during flight planning phase, before the delay even occurs.

CHAPTER TWO

REQUIREMENTS

Address class with:

- Four attributes: country, city, town
- One constructor method that receives all parameters
- Get / Set methods
- Display method

Phone class with:

- Four attributes: countrycode, code (city or mobile operator), number (7 digits), type (home, office or mobile)
- Twconstructor methods
- One that receives all parameters
- One that assigns countrycode +90 as default and receives other parameters
- Get / Set methods
- Display method

Date class with:

- Five attributes: day, month, year, hour, minute
- Twconstructor methods
- One that receives all parameters(24-hour format)
- One that day, month and year
- Get / Set methods
- Display method

Person class with:

- Eight attributes: name, surname, ID number, phone, birthday, gender, weight, height, Flight history(Araylist)
- One constructor method (name, surname, ID number, phone, birthday, gender, weight, height)
- addFlightToHistory(Flight flight)
- Get / Set methods
- Display method

Passenger class extends Person with:

- bag ID, bag weight
- One constructor method that receives all parameters
- Get / Set methods
- Display method

Pilot class extends Person with:

- rank(pilot or co-pilot),FligtCompany flightCompany, work experience, salary ,Flight nextFlights(Queue)

- One constructor method (name, surname, ID number, phone, birthday, gender, weight, height, rank(pilot or co-pilot),FligtCompany flightCompany, work experience, salary)
- addFlightToNextFlights(Flight flight)
- isAvaliable(Flight flight)
- addFlight(Flight flight)
- Get / Set methods
- Display method

Host class extends Person with:

- work experience, salary,FligtCompany flightCompany, Flight nextFlights(Queue)
- One constructor method (name, surname, ID number, phone, birthday, gender, weight, height,work experience, salary,FligtCompany flightCompany)
- addFlightToNextFlights(Flight flight)
- isAvaliable(Flight flight)
- addFlight(Flight flight)
- Get / Set methods
- Display method

Seat class with:

- int seatID, boolean status, Passenger, Date booking date, int price
- first constructor method (int seatID)
- Get / Set methods
- Display method

Plane class with:

- String planeCode, String name, int baggageCapacity,int seatNumber, int age, Flight history(ArayList),Flight nextFlights(Queue), int fuelCapacity, Seat[]seats, FlightCompany flightCompany
- One constructor method (String planeCode, String name, int baggageCapacity,int seatNumber, int age, Flight history(ArayList), int fuelCapacity, FlightCompany flightCompany)
- Get / Set methods
- isAvaliable(Flight flight)
- addSeatWithoutPrice(seatID)
- addSeatWithPrice(seatID,price)
- addFlightToNextFlights(Flight flight)
- Display method

Flight company class with:

- int id, String name, Address address, Phone phone ,Plane[] planes, Pilot[]pilots, Host[]hosts, int budget
- One constructor method (name, address, phone, budget)
- Get / Set methods
- Display method
- addPlane(Plane plane)
- addpilot(Pilot pilot)

- addHosts(Host host)
- deletePlane(Plane plane)
- deletepilot(Pilot pilot)
- deleteHosts(Host host)
- displayAllPilots() method
- displayAllHosts() method
- displayAllEmployees() method
- totalIncome() method
- totalOutcome() method
- totalGain() method

Pist class with:

- int id, String type, int capacity, Queue planeQueue
- One constructor method (type, capacity)
- addPlaneToQueue(Plane plane)
- Get / Set methods
- Display Method

Airport class with:

- int id, String name, Address adddress, int pistCount ,Pist[]pists
- One constructor method (name, address, pistCount)
- addPist(String type, String capacity)
- setPermission(int pistNumber, String type) method
- totalGain() method
- Get / Set methods
- Display Method

Flight class with:

- int id, Airport source, Airport destination, Date departureTime, Date arrivalTime, Plane plane, Pilot[]pilots, Host[]hosts, Passenger[] passengers,Seat[] seats, int flightCost
- One constructor method (Airport source, Airport destination, Date departureTime, Date arrivalTime, Plane plane,int flightCost)
- bookSeatmethod (Passenger pasenger, int seatID,int price)
- cancleReservation(int seatID
- changePilot(Pilot newPilot,Pilot oldPilot)
- changeHost(Host newHost, Host old Host)
- changeSeatOwner(int seatID, Passenger newPassenger)
- displayAllPassengers() method
- displayAllPilots() method
- displayAllHosts() method
- Get / Set methods
- Display Method
- calculateIncome() method

Manager class with:

- FlightCompany flightCompany(ArrayList)
- Airport airport(ArrayList)
- Flight flight(ArrayList)
- One constructor method without parameter
- start Method
- read functions from txt files(Examples : FlightCompany.txt, Pilot.txt, Host.txt, Plane.txt, Airport.txt, Pist.txt, Flight.txt, Passenger.txt,Reservation.txt)
- searchAvaliableSeats(String sourceAirportName , String destinationAirportName ,Date departureDate (due tdate not hour),int minPrice, int maxPrice)
- searchAvaliablePilots(String sourceAirportName, ,Date departureDate, Date arrivalDate)
- searchAvaliableHosts(String sourceAirportName, ,Date departureDate, Date arrivalDate)
- searchAvaliablePlane(String sourceAirportName, ,Date departureDate, Date arrivalDate)
- searchPassenger (name, surname)
- addFlightCompany (name, address, phone, budget)
- addPilot (name, surname, ID number, phone, birthday, gender, weight, height, rank(pilot or co-pilot),FligtCompany flightCompany, work experience, salary)
- addHost (name, surname, ID number, phone, birthday, gender, weight, height,work experience, salary,FligtCompany flightCompany)
- addPlane (String planeCode, String name, int baggageCapacity,int seatNumber, int age, Flight history(ArayList), int fuelCapacity, FlightCompany flightCompany)
- addAirport (name, address, pistCount)
- addPist (type, capacity)
- addFlight (Airport source, Airport destination, Date departureTime, Date arrivalTime, Plane plane,int flightCost)
- add Passenger (name, surname, ID number, phone, birthday, gender, weight, height , bagID, bagWeight)
- addReservation (flightId, seatId, Passenger passenger)
- addFlightCompany(String name, Address address, Phone phone, int budget
- delete methods(Examples : FlightCompany, Pilot, Host, Plane, Airport, Pist, Flight, Passenger,Reservation)
- menu Method

Main class with:

- Main method (it will create a manager object and just call starter method.)

CHAPTER THREE

UML DIAGRAMS

Use Case Diagram:

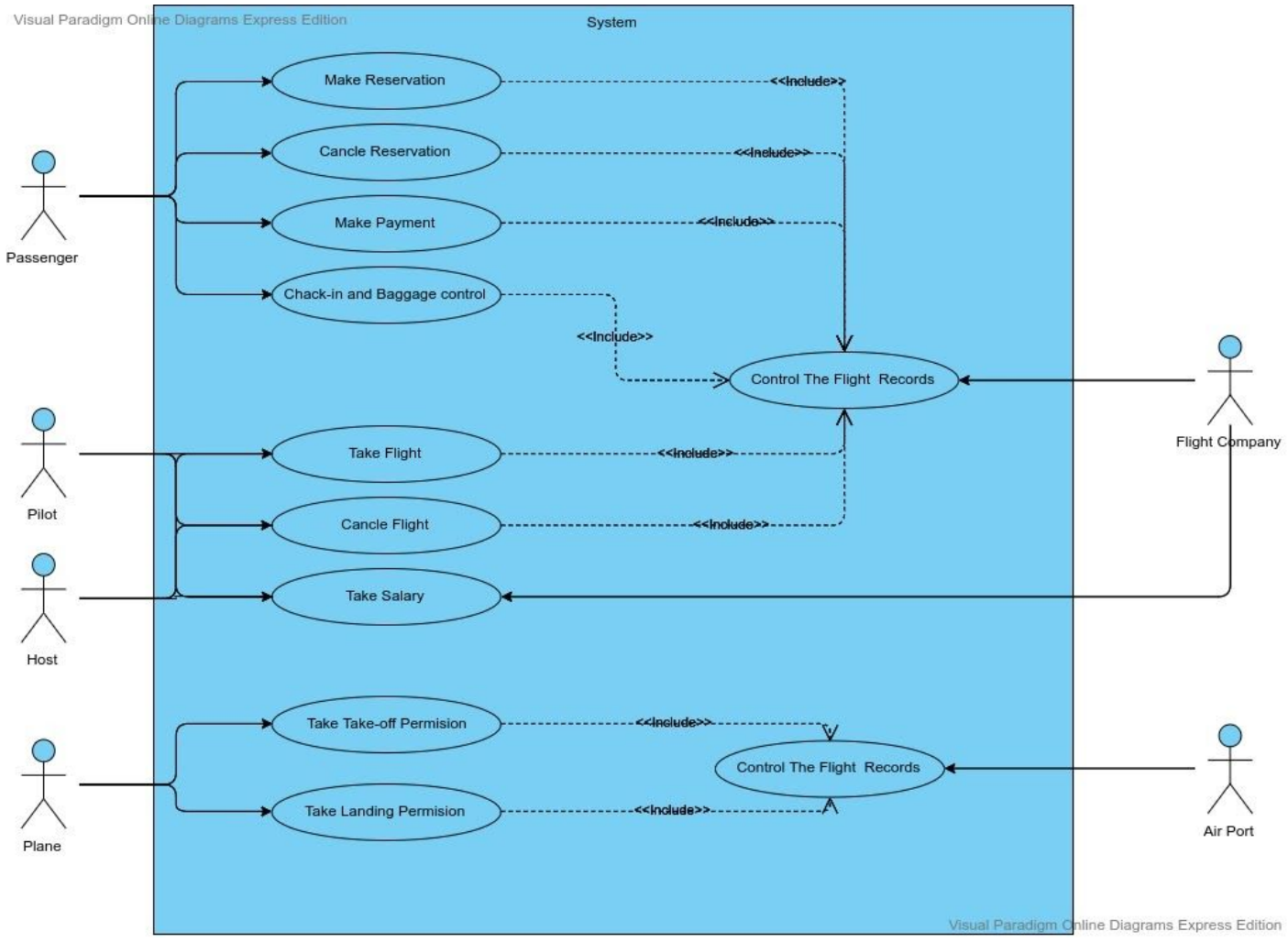

Figure 3.1: Use Case Diagram

There are six actors in our project. First of these actors is passenger. The passenger has 4 unique actions such as making reservation, reservation cancelation, making payment and check-in. Other two actors are pilot and host. They have same actions as taking flight, flight cancelation and taking salary. Actions of these three actors, except taking salary, depends on controlling the flight records action and this action is controlled by flight company. The remaining actors are plane and airport and their actions are about the landing and the take off.

Class Diagram:

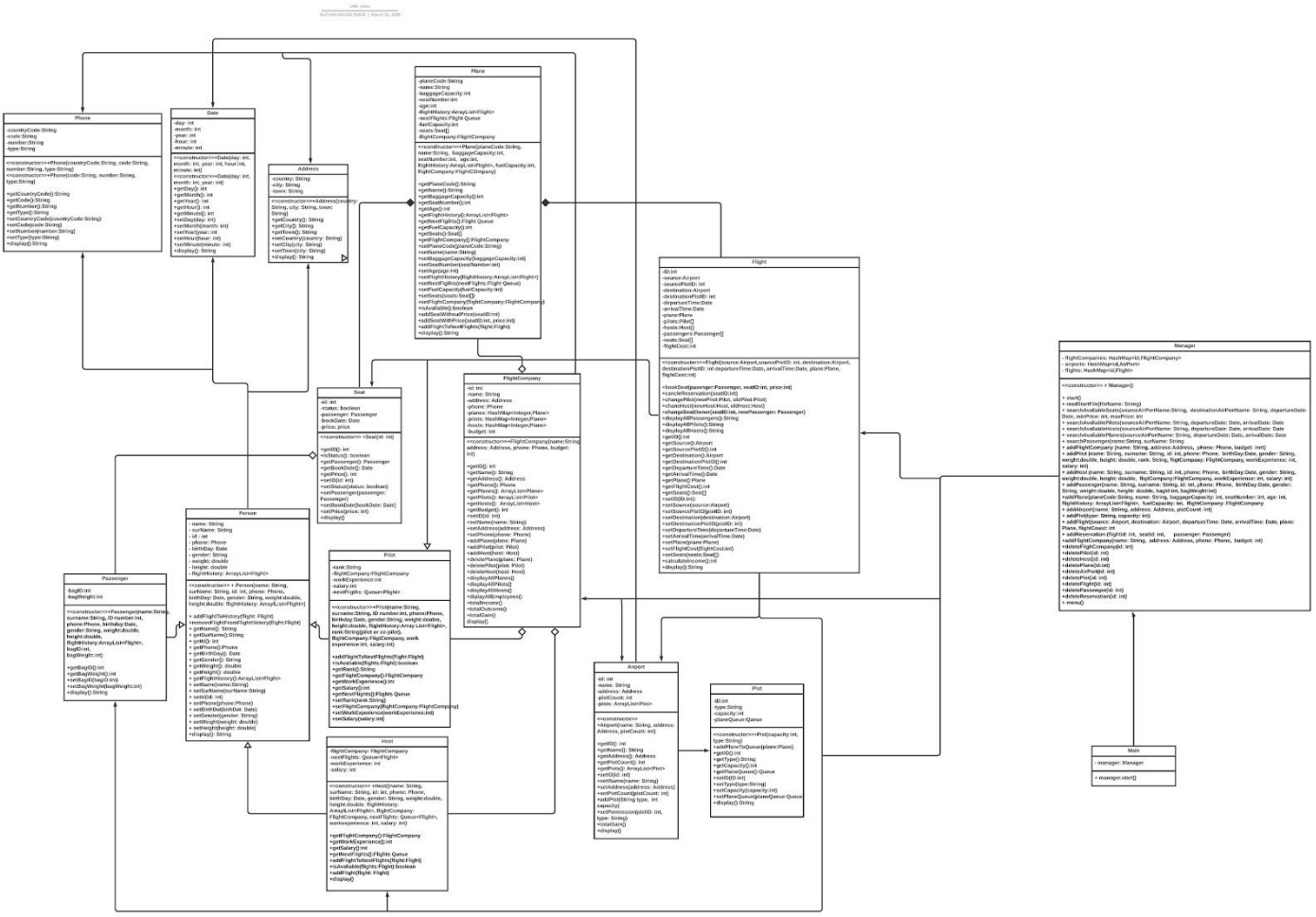

Figure 3.2: Class Diagram

There are fifteen classes in our project. There is one super class as person and other classes about the person extends from this super class. Management class is our administration class. All fundamental methods in our system are executed by management class

Activity Diagram:

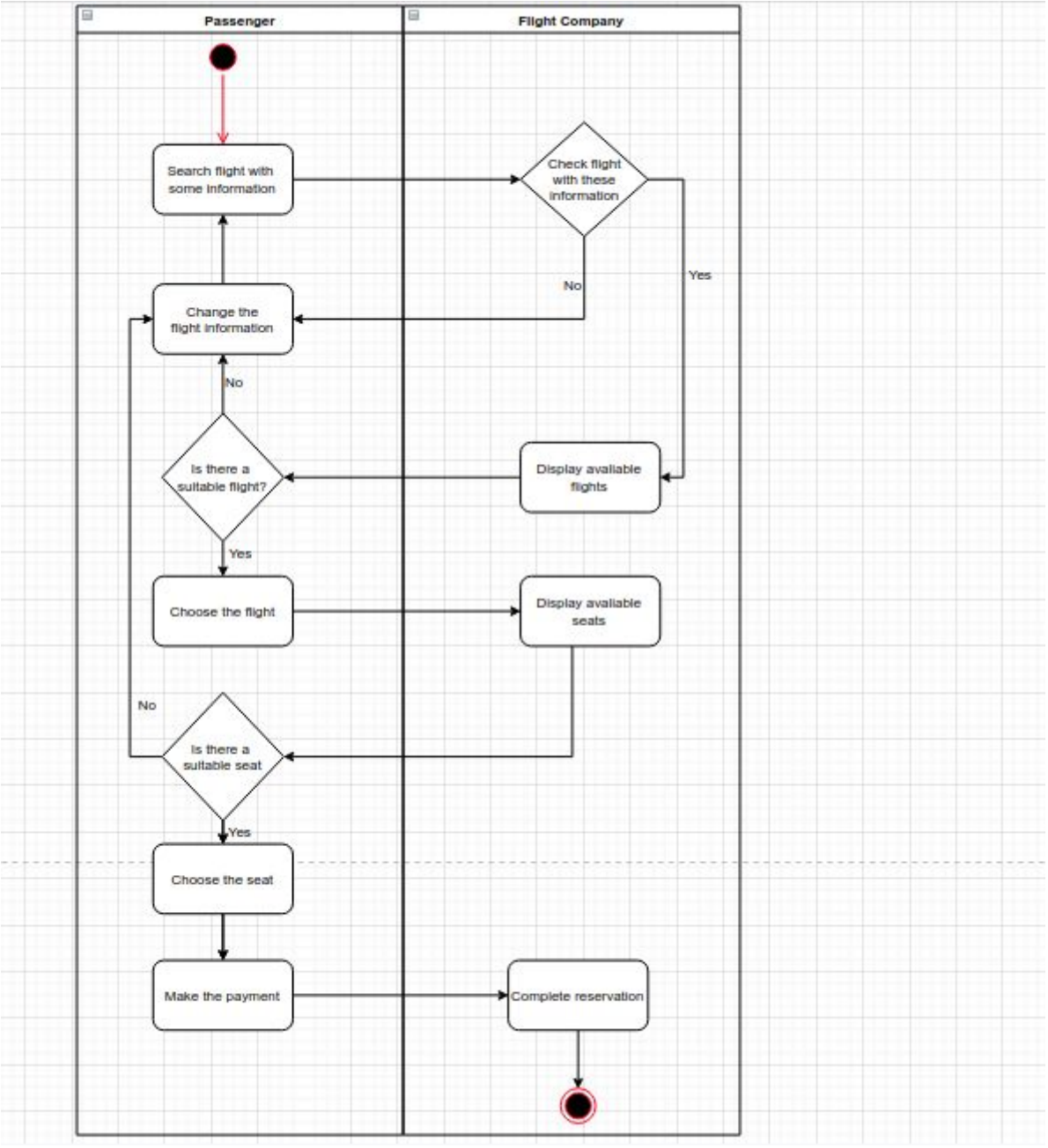

Figure 3.3: Activity Diagram

We show the making reservation steps between passenger and Flight Company in Activity Diagram. Firstly, passenger searches a flight with some information, such as “departure date, departure airport, destination airport”, and flight company returns available flights. If one of these flights are suitable, passenger chooses one of them and Flight Company returns list of available seats on this flight. If the flight isn’t suitable, passenger searches flight with different information. After that, passenger controls the

available seats. If one of them is suitable, passenger choose a seat. If it isn't suitable, passenger returns the searching flights. Finally passenger makes the payment and Flight Company completes the reservation.

State Diagram:

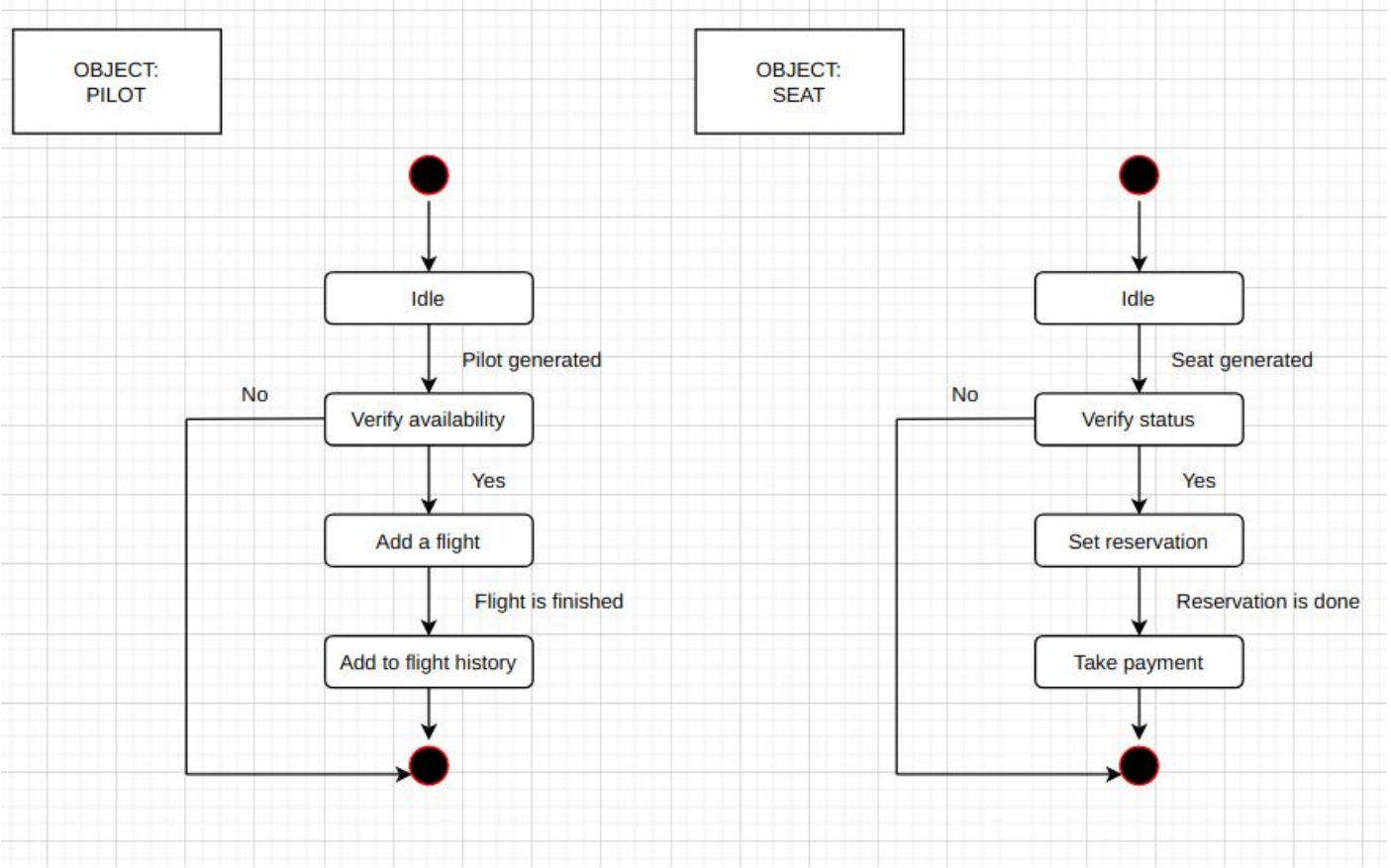

Figure 3.4: State Diagram

The state diagrams in Figure 3.4 show how the objects "pilot" and "seat" processes during their life cycles. The object "pilot" is generated when it is added to flight company. The pilot's availability is checked before assigning to a flight. If the pilot is not busy during the flight, the flight company can assign that pilot. And after the flight is finished the information about the flight is added to pilot's flight history.

The object "seat" is generated after a new plane is created. Before a passenger makes a reservation, the status of the seat is checked. If the seat is available, passenger can choose that seat and finalize reservation by making the payment.

Sequence Diagram:

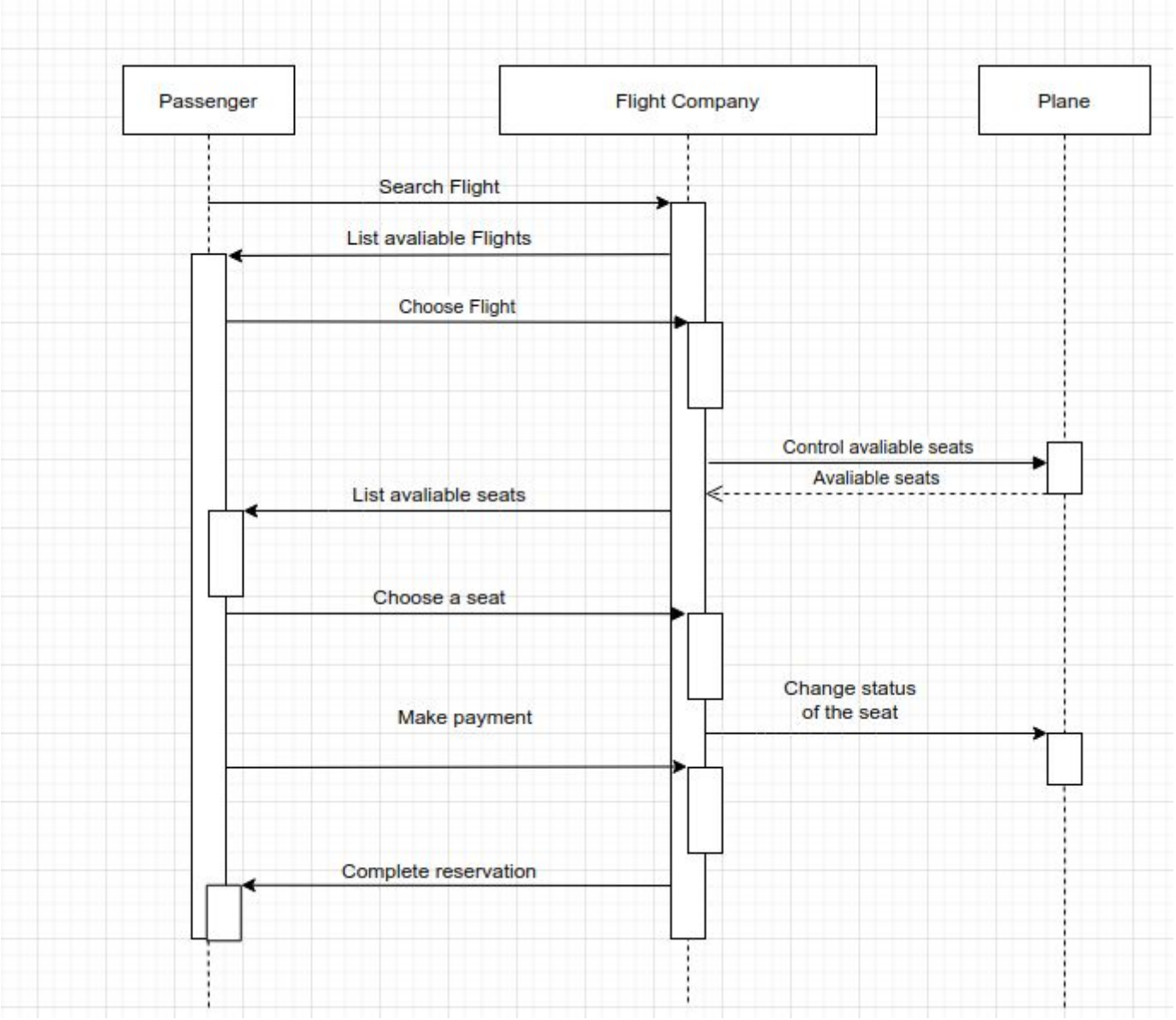

Figure 3.5: Sequence Diagram

We show the making reservation steps between passenger, Flight Company and Plane Class in Sequence Diagram. Firstly, passenger searches a flight with some information and flight company returns list of available flights. Passenger chooses one of them and Flight Company controls the seats of Plane. Plane returns list of available seats on this flight for Flight Company.Then, Flight Company returns them for Passenger. After that, passenger controls the available seats. Passenger chooses a seat and Flight Company changes the status of the seat on Plane. Finally passenger makes the payment and Flight Company completes the reservation.

CHAPTER FOUR

IMPLEMENTATION

All classes are created and all attributes are assigned. Constructors, getter/setter methods and toString methods are written for all classes except Manager class. Person class is created as a abstract class and it extended to passenger, pilot and host classes. An interface is added as a new arrangement in addition to previous design. This interface's name is 'Employee' and it implemented to pilot and host classes. There are lots of methods in the classes:

AIRPORT CLASS:

o addPist(Pist p) : This method add a pist into pist list on the airport.

o setPermission(int pistNumber, Plane p): This method give permission to plane for take-off or landing.

FLIGHT CLASS:

o bookSeat(Passenger p, int seatId, int price, Date date): This method find the seat according to seatId on available plane according to given date and update it with passenger's informations who book it.

o cancelReservation(int seatId): The reservation is canceled for seat that has given seat ıd by using for loop.

o changePilot(Pilot newPilot,Pilot oldPilot): The new pilot is placed instead of old pilot for flight.

o changeHost(Host newHost,Pilot oldHost): The new pilot is placed instead of old pilot for flight.

o changeSeatOwner(int seatId, Passenger newPassenger): The new passenger is placed instead of old passenger into seat for flight.

o calculateIncome(): This method calculates the incoming money of flight by using seat's price.

FLIGHTCOMPANY CLASS:

o add Methods: These methods add pilot, host or plane into the lists for a flight company.

o delete Methods: These methods deletes pilot, host or plane from the lists for a flight company.

o display Method: These methods displays the list elements for intendent lists.

o totalIncome(): This method calculates the incoming money of each flight by using calculateIncome method in the flight class and sum them to calculate total income.

o totalOutcome(): This method calculate the cost of a flight company by summing the employees salaries.

o totalGain(): This method calculate the earned money by using totalIncome and totalOutcome methods.

PILOT and HOST CLASSES:

o addFlightToNextFlight(Flight f): This method add a new flight into flight queue for pilots or hosts.

o addFlight(Flight f): This method add a completed flight into flight history list for pilots or hosts.

PIST CLASS:

o addPlaneToQueue(Plane p): This method add a plane that is take-off or landing into the pist queue.

MANAGER CLASS:

o searchAvaliablePlane(Date departureDate, Date arrivalDate): This method finds the available planes according to given parameters, and displays them by using loops.

o searchAvaliableSeats(Date departureDate,Date arrivalDate, int minPrice, int maxPrice): This method finds the available(empty) seats according to given parameters, and displays them. Firstly the available planes stores into a list between departureDate and arrivalDate. Then, each seats for the each airplane are controlled it is empty or not by using nested loop.

o searchAvaliablePilots(Date departureDate, Date arrivalDate): This method finds the available pilots according to given parameters, and displays them. Firstly the available planes stores into a list between departureDate and arrivalDate. Then, each pilot for the each airplane are controlled he or she is available or not by using nested loop.

o searchAvaliableHosts(Date departureDate, Date arrivalDate): This method finds the available hosts according to given parameters, and displays them. Firstly the available planes stores into a list between departureDate and arrivalDate. Then, each host for the each airplane are controlled he or she is available or not by using nested loop.

o searchPassenger (String name, String surname): This method finds the passenger according to given name and surname by using foreach.

o searchFlightCompany (String name): This method finds the flight company according to given name by using foreach.

o add Methods(): These methods put pilot, host, passenger, flight, airport, pist, flight company or plane into the hash maps.

o delete Methods(): These methods remove pilot, host, passenger, flight, airport, pist, flight company or plane from the hash maps.

CHAPTER FIVE

CONCLUSION AND FUTURE WORKS

We created all needed classes and interfaces of our project. Also all necessary methods and functions was implemented. class is implemented for We've learned the requirements of the OOP and how to implemented it in this project. Also we've learned the UML diagrams and how to design the project by using these diagrams. Usage of github is also efficient part of this project. We've learned how to use git in our IDE's and how to use push, pull, merge operations on github. We decided to continue as a web project for the continuation of project and use Angular for the GUI part of the project. All necessary files of Angular was pushed into github repository.