Project 1: Lounge Airlines

Profit Maximization

CmpE 160, Introduction to Object Oriented Programming, Spring 2022

Instructor: Tuna Tuğcu

Assistants : Yiğit Yıldırım, Bahadır Gezer

[bahadir.gezer@boun.edu.tr](mailto:bahadir.gezer@boun.edu.tr?subject=[CMPE160]%20%3csubject%3e)

Due: xx.xx.2022 Day, 23.55

1. **Introduction**

Modern aviation is great way for a safe, cost effective and fast way of transport for the public. From small propeller planes that operate whenever to vast fleets of modern aircraft with rigorous timetables; the aviation industry has grown significantly over the past century. This growth was partly fueled by advancements in the aircraft manufacturing process and partly form computer driven planning and pricing.

Price is the most important factor for acquiring success in the aviation industry. The modern traveler will prioritize prices over anything else. Not even brand equity is relevant when it comes to modern air travel. With websites where passengers can monitor prices for each route, for each air carrier, for any time slot and for any other variable; airlines need to be very aware of their pricing to stay competitive. Specialized software and algorithms are used to set aircraft schedules, analyze routes, price tickets, and manage resources.

In this project you will try to implement an acceptable version of these systems that work for Lounge Airlines.

You will be the airline which has some aircraft. Like any well-functioning company, you too have a profit incentive. So, your task is simple: Generate as much profit as possible.

1. **Class Hierarchy**

There will be 5 main containers. “airline”, “airport”, “passenger”, “interfaces” and “executable”. Inside “airline” there will be the “aircraft” container, and inside the “aircraft” container there will be the “concrete” container. You should not create any additional java files.

You must design the accessibility of your code in accordance with proper encapsulation techniques. Some methods and fields will be predetermined access modifiers, other than those you should determine access modifiers in a way which removes unrelated or potentially harmful access within your code.

src:

airline container:

Airline.java

aircraft container:

Aircraft.java

PassengerAircraft.java

concrete container:

JetPassengerAircraft.java

RapidPassengerAircraft.java

WidebodyPassengerAircraft.java

PropPassengerAircraft.java

airport container:

Airport.java

HubAirport.java

MajorAirport.java

RegionalAirport.java

passenger container:

Passenger.java

interfaces container:

AircraftInterface.java

PassengerInterface.java

executable container:

Main.java

1. **Project Details**

The implementation of this project has two parts. First, there is the functionality part, where you create all the classes and necessary methods. The second part is where you will try out different algorithms and interesting heuristics to maximize profits.

Since you are “the airline” your executable should not interact with any method outside of the Airline.java class (Except aircraft configuration functions) . This class will be the gateway to the functional part of the code.

1. **Airline:**
2. **Passenger:**
   * 1. **Passenger.java**

Necessary fields:

* private final int ID
* private final double weight
* private final int baggageCount
* private ArrayList<Airport> destinations

Necessary Methods:

* + - * boolean connection(Airport airport, int seatType)
      * boolean board(Airport airport, int seatType)
      * boolean disembark(Airport airport, double aircraftTypeMultiplier)

Each passenger will have an ID, weight and baggageCount. These will be provided for you in the inputs. There will also be a list of destinations which the passenger wants to visit.

Passenger.java will be an abstract class. Since the voluntary action of the consumer determines the final price in a free market, ticket prices will be calculated in the passenger class. Ticket revenue will be collected only when the passenger disembarks. Connections and boardings do not generate any revenue.

* + 1. **EconomyPassenger.java**
    2. **BusinessPassenger.java**
    3. **FirstClassPassenger.java**
    4. **LuxuryPassenger.java**

1. **Airport:**
2. **Aircraft:**
3. **Input & Output Format**
4. **Input**
5. **Output**
6. **Examples**
7. **Grading**
8. **Warnings**

Input Format

The first line of the input will consist of the max

Output Format

Some Remarks

Your task is to load, transfer and disembark passengers. Each passenger will have a list of destinations which is a list of airports and their starting airport. You will load a passenger to an airplane from its initial airport. From there the plane can fly to any airport it wants to, however, the passenger cannot disembark from the plane if the plane lands in an airport which is not a future destination of the passenger. If you don’t want to take the passenger to one of its future destinations by the aircraft it’s in, you can transfer the passenger to another airplane in the current airport.

If a passenger lands in an airport which has a higher index in the destinations airport list, then the currentAirport integer will be set to the index of that airport. Passengers cannot disembark in airports that have a smaller index than the currentAirport index.

In the input part, destinations list of airports will be given as a list of integers. These integers are airport ID’s. The airports which the passenger wants to visit will be in sorted order of airport types. If the index i is a hub airport, index i + 1 can be all three types of airports or the list could end. If the index i is a major airport, index i + 1 can be a major airport or a regional airport or the list could end. If the index i is a regional airport, the list must end.

Revenue (ticket price) will be collected only when the passenger disembarks. This revenue will be calculated from the current budget, the position of disembarkation, baggage count and from the seat type. The revenue will be subtracted from the budget of the passenger and added to the total revenue of the airline.

Every operation with an airplane must check if the airplane is in the

Funtionalities:

boolean addFuel(double fuel);

Adds the fuel amount to the current fuel of the plane. Must check the validity of the action. If it’s not valid, it should return false, if the operation is valid, it must take the necessary actions and then return true.

Fuel has some weight; plane has maximum fuel capacity and maximum weight limit.

Airport.java

Airport.java be an abstract class that will be the parent of the concrete Airport classes. These concrete classes are HubAirport, MajorAirport and RegionalAirport. All will extend the Airport class.

Aircrafts are refueled at the airport. The price of fuel will be determined by the airport the aircraft is currently in.

The airport has an operationCost. This is the fee paid of every operation done in the airport.