Project 1: Lounge Airlines

Profit Maximization

CmpE 160, Introduction to Object Oriented Programming, Spring 2022

Instructor: Tuna Tuğcu

TA : Yiğit Yıldırım

SA : 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

Abstract class, implements AircraftInterface

PassengerAircraft.java

Abstract class, extends Aircraft implements PassengerInterface

concrete container:

JetPassengerAircraft.java

Extends PassengerAircraft

RapidPassengerAircraft.java

Extends PassengerAircraft

WidebodyPassengerAircraft.java

Extends PassengerAircraft

PropPassengerAircraft.java

Extends PassengerAircraft

airport container:

Airport.java

Abstract class

HubAirport.java

Extends Airport

MajorAirport.java

Extends Airport

RegionalAirport.java

Extends Airport

passenger container:

Passenger.java

Abstract class

EconomyPassenger.java

Extends Passenger

BusinessPassenger.java

Extends Passenger

FirstClassPassenger.java

Extends Passenger

LuxuryPassenger.java

Extends Passenger

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:**

This class is the only class which should be imported to main. So, all of your functions should meet here. You will be given a few basic functionalities. You can build upon these to create methods that are suited to your needs.

You shouldn’t import aircraft classes into main the main class so there should not be any aircraft objects in the main class, all aircraft objects should be held in the airline class. Thus, when choosing which aircraft to use you should give an integer, which should be the index of that aircraft in your aircrafts ArrayList.

Revenue and expenses will also be calculated inside the airline class. You should hold both values separately and return the profit when requested.

(TODO: DEFINE OPERATIONAL COST) (MAYBE GIVE IT AS AN INPUT)

Primitive Operations:

* **boolean fly(Airport toAirport, int aircraftIndex)**

This method will let you fly the aircraft from its current airport to toAirport. aircraftIndex is the index at which the aircraft is stored in the aircrafts ArrayList. This method should return true if the flight operation is successful and return false if the flight operation could not be completed.

There is a running cost of every airline which increases as the airline gets larger. This must be considered. A running cost of an airline is the operationalCost times the aircrafts the airline has. Running cost should be added automatically to the expenses every time this method is called.

A flight cannot happen if the plane cannot reach the destination airport because of range limitations, or the other airport is full. There are some edge cases for invalid flights. You should think of them by yourself, all of them are very easy to figure out.

If the aircraft can fly, then there is a flight cost associated with every flight. This flight cost consists of the operational cost of the airplane, landing fee and departure fee. Calculation of this cost will be explained in the aircraft and airport classes.

(TODO : LOG)

* **boolean loadPassenger(Passenger passenger, Airport airport, int aircraftIndex)**

This method will let you load a passenger from a given airport to an aircraft of your choice. Passenger is the passenger to load, airport is the airport where the passenger is and aircraftIndex is the index of the aircraft which the passenger will be loaded to.

This method should return true if the load operation is successful and return false if the load operation could not be completed.

A load operation cannot happen when the aircraft and the passenger is not in the same airport, or the passenger cannot be loaded into the aircraft. A passenger cannot be loaded to an aircraft if the aircraft does not have seats for that passenger or if the aircraft exceeds the maximum weight limit with the addition of that passenger. Passenger weight calculation will be explained in the passenger class. (TODO) Passengers cannot sit in seats of higher class, however, if they have to, they can sit in seats of lower class. Do not forget that the priority is always the seat of higher class when a passenger is assigned a seat.

If a passenger can be loaded, then loading cost should be calculated and added to expenses. Every load operation has a loading cost associated with it. Calculation of this cost will be explained in the aircraft classes.

* **boolean unloadPassenger(Passenger passenger, int aircraftIndex)**

This method will let you unload a passenger from a given aircraft. Passenger is the passenger to unload, aircraftIndex is the index of the aircraft which the passenger will be unloaded from.

This method should return true if the unload operation is successful and return false if the unload operation could not be completed.

An unload operation cannot happen if the passenger cannot disembark in the airport which the aircraft is currently in. Disembarkation check will be done in the passenger classes. Passengers can disembark if the airport they want to disembark is a future destination which they want to visit. Passengers cannot go back to destinations that are lower in the destinations list.

Ticket revenue is earned when a passenger disembarks, transfers and loading operations does not generate any revenue. Details of the ticket price calculation will be given in the aircraft and passenger classes. Ticket price should be added to the revenue in this function.

(TODO : LOG)

* **Refueling functions:**

Refueling operations should be done through this class. You can implement any refueling function you desire so long as you write the amount of fuel bought, from which airport for which aircraft to the output log. For example you could have refuel(int aircraftIndex, double fuel) or depoyuFulle(int aircraftIndex) it is up to you. Do not forget to change the weight of the aircraft when adding fuel. Fuel weight for a given fuel amount is calculated by multiplying the fuelWeight constant with the fuel amount. You can think of the fuel amount as a volume.

Since fuel weight is a big part of the overall weight of the aircraft, you might want to dump fuel, in this case you do not get any refunds for dumped fuel. Any fuel dumping should also be written to the log.

* **Aircraft creation functions:**

As stated previously all aircraft should be added using the airline, not by importing aircrafts into main. Thus, you should have function for adding every type of aircraft. Every aircraft created should be appended to the aircrafts ArrayList. Your aircraft count should not exceed the maximum aircraft count which will be given in the inputs.

* **Seat setting functions:**

After creating the aircraft, you should specify the seat configuration want on that aircraft. Seat setting and resetting details will be given in aircraft classes.

(TODO : LOG)

1. **Passenger:**
   * 1. **Passenger.java**

* **public abstract double departAircraft(Aircraft aircraft)**

Does the necessary departure operations and returns the departure fee.

* **public abstract double landAircraft(Aircraft aircraft)**

Does the necessary landing operations and returns the landing fee.

Necessary fields:

* **private final int ID**
* **private final double weight**
* **private final int baggageCount**
* **private ArrayList<Airport> destinations**
  + - * **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. **Airport.java**

* **public abstract double departAircraft(Aircraft aircraft)**

Does the necessary departure operations and returns the departure fee.

* **public abstract double landAircraft(Aircraft aircraft)**

Does the necessary landing operations and returns the landing fee.

Necessary fields:

* **private final int ID**

Unique ID of the airport.

* **private final double x, y**

Coordinates of the airport. Used when calculating distance.

* **protected double fuelCost**

Price of fuel in this airport.

* **protected double operationFee**

Fee paid to this airport for ceratin operations.

* **protected int passengerAircraftCapacity**

Maximum number of passenger aircrafts this airport can hold.

You can add any helper functions of your liking to this class. Some helpful functions are: boolean equals(Airport other), boolean isFull(), double getDistance(Airport airport), void addPassenger(Passenger passenger), void removePassenger(Passenger passenger)…

Passenger objects should be held either in airport objects or in aircraft objects. You can use any collection you like to store passenger objects.

1. **HubAirport.java**

* **public double departAircraft(Aircraft aircraft)**

This method should do the necessary departure operations. Returns the departure fee.

To calculate departure fee, we need to define two values. First one is called the fullness coefficient. Fullness coefficient is the obtained by multiplying 0.6 with e -the mathematical constant- raised to the power of aircraft ratio. Aircraft ratio for an airport is the total number of aircrafts divided by the total aircraft capacity of that airport. The second value we need to define is aircraft weight ratio, it is simply the ratio of weight to maxWeight for the aircraft given in the parameters. To calculate departure fee, you must multiply operationFee, aircraft weight ratio, fullness coefficient and a constant value of 0.7.

* **public abstract double landAircraft(Aircraft aircraft)**

This method should do the necessary landing operations. Returns the landing fee.

To calculate landing fee, we need to define two values. First one is called the fullness coefficient. Which is the same as the one used in departAircraft(). The second value is also the same as departAircraft(). To calculate departure fee, you must multiply operationFee, aircraft weight ratio, fullness coefficient and a constant value of 0.8.

1. **MajorAirport.java**

* **public double departAircraft(Aircraft aircraft)**

This method should do the necessary departure operations. Returns the departure fee.

To calculate departure fee, we need to define two values. First one is called the fullness coefficient. Fullness coefficient is the obtained by multiplying 0.6 with e -the mathematical constant- raised to the power of aircraft ratio. Aircraft ratio for an airport is the total number of aircrafts divided by the total aircraft capacity of that airport. The second value we need to define is aircraft weight ratio, it is simply the ratio of weight to maxWeight for the aircraft given in the parameters. To calculate departure fee, you must multiply operationFee, aircraft weight ratio, fullness coefficient and a constant value of 0.9.

* **public abstract double landAircraft(Aircraft aircraft)**

This method should do the necessary landing operations. Returns the landing fee.

To calculate landing fee, we need to define two values. First one is called the fullness coefficient. Which is the same as the one used in departAircraft(). The second value is also the same as departAircraft(). To calculate departure fee, you must multiply operationFee, aircraft weight ratio and fullness coefficient.

1. **RegionalAirport.java**

* **public double departAircraft(Aircraft aircraft)**

This method should do the necessary departure operations. Returns the departure fee.

To calculate departure fee, we need to define two values. First one is called the fullness coefficient. Fullness coefficient is the obtained by multiplying 0.6 with e -the mathematical constant- raised to the power of aircraft ratio. Aircraft ratio for an airport is the total number of aircrafts divided by the total aircraft capacity of that airport. The second value we need to define is aircraft weight ratio, it is simply the ratio of weight to maxWeight for the aircraft given in the parameters. To calculate departure fee you must multiply operationFee, aircraft weight ratio, fullness coefficient and a constant value of 1.2.

* **public abstract double landAircraft(Aircraft aircraft)**

This method should do the necessary landing operations. Returns the landing fee.

To calculate landing fee, we need to define two values. First one is called the fullness coefficient. Which is the same as the one used in departAircraft(). The second value is also the same as departAircraft(). To calculate departure fee you must multiply operationFee, aircraft weight ratio, fullness coefficient and a constant value of 1,3.

1. **Aircraft:**

As the airline you have access to different kinds of aircraft each with its own stats. You must choose the correct aircraft with the correct seat configuration for the correct passenger base and for the correct routes.

1. **Aircraft.java**

You can implement other helper functions of your liking.

* protected double fly(Airport toAirport)

This function flies the aircraft to toAirport. It should do the necessary fuel calculations using getFuelConsumption() which will be explained later. It should also calculate the flight cost and return this value.

Flight cost consists of the operational cost of the aircraft, the departure fee and landing fee. Departure and landing fees will be explained in the airport classes. Operational cost is different for each type of aircraft, and it is based on fullness of the aircraft and distance of the flight. Full details of this calculation will be given in the concrete aircraft classes.

Since fly() always return the flight cost you should implement a boolean function for fly() which checks if a flight can happen or not. This boolean function and fly() should be used together to make sure that any invalid operations do not happen.

Necessary fields:

* **protected Airport currentAirport**

The airport which the plane is currently in.

* **protected double weight**

This is the present weight of the aircraft. Every aircraft has an empty weight, so this will be given different starting values depending on the aircraft. This value should not exceed maxWeight.

* **protected double maxWeight**

This is the maximum allowable weight of the aircraft. The aircraft should not exceed this weight.

* **protected double fuelWeight**

This field is equal to 0.7, it is the constant we use to convert fuel volume to fuel weight.

* **protected double fuel**

This field holds the amount of fuel the aircraft has at any moment. It should not exceed fuelCapacity. fuelCapacity and this field are both volumes, so you should multiply fuel amount with fuelWeight when you want to get the weight for some amount of fuel. This value is zero upon initialization.

* **protected double fuelCapacity**

Fuel capacity of the airplane. It is different and fixed for every type of aircraft.

**Refueling functions:**

These functions should add fuel to the aircraft. You are free to choose what kind of refueling function you want to implement. However, all refueling functions must add the appropriate fuel weight along with the fuel. Every refueling function must also return a double, which should be the cost of fuel. Fuel cost is calculated by multiplying fuel amount with the fuel cost of the airport which the aircraft is refueling at. These functions will be used by the matching caller functions of the airline.

1. **PassengerAircraft.java**

This class will hold functions related with passenger operations for aircrafts. Passenger objects must be held in this class if they are loaded in. Passenger objects should switch between airport objects and aircraft objects, they should not be stored anywhere else. You can use any collection you like to store passenger objects.

* **double loadPassenger(Passenger passenger)**

This method loads the passenger to the appropriate seat. It should return the loading fee. Loading fees are based on the operationFee, which is a fee specific to each type of aircraft. If the loading operation cannot be completed return value should only be the operationFee. If the loading operation is completed the complete loading fee is returned.

Loading fee is calculated by multiplying operationFee with aircraftTypeMultiplier, the detail of which will be given in the concrete aircraft classes, and a constant. This constant depends on the type of seat selected for that passenger. If the seat is a first class seat the constant is 2.5, if the seat is a business seat the constant is 1.5 and if the seat is an economy seat the constant is 1.2. Remember that passengers cannot sit in seats with a higher class then their class. LuxuryPassengers act the same way as FirstClassPassengers.

If you do not want to pay the operationFee every time you do an invalid loading operation, you can create a boolean method that will check if the operation is valid. If you use both methods together you can avoid unnecessary expenses.

* **double unloadPassenger(Passenger passenger)**

This method unloads the passenger. A passenger can be unloaded if it can disembark in the airport which the aircraft is in. This is where the ticket revenue will be calculated and collected. Ticket price is calculated by multiplying disembarkation revenue with the seat constant. Disembarkation revenue will be calculated in the passenger classes. (TODO : disembarkation revenue)

If the passenger was seated in economy the seat constant is 1.0, if the passenger was seated in business the seat constant is 2.8, if the passenger was seated in first class the seat constant is 7.5.

If the passenger cannot disembark you should return the operationFee. So, this method can return both revenue and expense, do not forget to take this into consideration.

* **double transferPassenger(Passenger passenger, PassengerAircraft toAircraft)**

This method transfers the passenger from the current aircraft to toAircraft. You can use this method if the passenger cannot disembark in an airport. It acts as a bypass to the unloadPassenger() operation when moving a passenger.

The implementation of this method is very similar to the loadPassenger operation, the only difference is this operation is between aircrafts rather than between an aircraft and an airport.

If the transfer operation is invalid, then you should return operationFee as an expense. If the transfer operation is valid, then you should return loading fee. This loading fee is the one you get when you use the loadPassenger() method above.

So, as a small summary, you can load passengers anytime you want. If you loaded them incorrectly you can disembark them in the same airport and the revenue function will not generate any revenue, but you will have to pay some fee for the loading and unloading operations. When you go to a new airport, if that airport is a future destination of the passenger, then the passenger can disembark and leave the aircraft. However, if the passenger cannot disembark in that airport, then you have two options. One, you can fly the aircraft to an airport which the passenger can disembark in, or you can transfer the passenger to another aircraft in the same airport.

Necessary fields:

* **protected double floorArea**

Total floor area of the aircraft, this field is different for each type of aircraft, so it should be initialized in the concrete aircraft classes.

* **private int economySeats, businessSeats, firstClassSeats**

Count of seats assigned for this aircraft for each seat type.

* **private int occupiedEconomySeats, occupiedBusinessSeats, occupiedFirstClassSeats**

Count of seats that are occupied for each seat type.

**Seat operations:**

Operations related to seat configuration will be done in this class. Each type of aircraft has a predetermined floor area and each type of seat also a predetermined area. You must make sure that the total area of seats assigned does not exceed the floor area of the plane. To calculate the occupied area, you must multiply each seat count with the respective seat area and add them all up.

Economy seats have an area of 1, business seats have an area of 3, and first-class seats have an area of 8.

You can set and reset seats as much as you want, however, you should make sure that the seat you are deleting is empty. If not, you should unload that passenger first.

1. **PropPassengerAircraft.java**

This class represents a small propeller aircraft which is designed for short routes and small destinations.

* **getFlightCost(Airport toAirport)**

This method calculates and returns the flight cost. It is used in the fly() function of Aircraft.java. flight cost is made up of three costs. Landing cost, departure cost and flight operation cost.

To calculate flight operation cost we need three values: fullness, distance and a constant. Distance is the distance between the current airport and toAirport. Fullness is the ratio of occupied seats to all seats. The constant for this type of aircraft is 0.1. Flight operation cost is the multiplication of these three values. Departure and landing cost is explained in Airport classes.

* **getFuelConsumption(double distance)**

This method calculates the fuel consumption of the aircraft for a given distance, it should return the total fuel needed for the given distance.

Real aircraft have a bathtub curve for fuel consumption. This project uses a similar curve.

To calculate fuel consumption, we must define two values. First one is distance ratio. Distance ratio for this type of aircraft is distance -given as a parameter- divided by 2000. The second value is called the bathtub coefficient. This value is calculated by putting the distance ratio in the curve (y = 25.9324 x4 - 50.5633 x3 + 35.0554 x2 - 9.90346 x + 1.97413). Fuel consumption has two parts: a takeoff part, and a cruise part. Fuel consumption is the addition of these two values. Takeoff fuel consumption is weight times a constant divided by fuelWeight. We must divide by fuelWeight because we are calculating the volume of fuel. The constant for this type of aircraft is 0.08. Cruise fuel consumption is fuelConsumption times bathtub coefficient times distance.

Necessary fields:

* **protected double weight**

The actual weight of the aircraft. Declared in Aircraft.java, initial value -which corresponds to the empty weight- is 14000. Weight should never drop below 14000 if the program runs correctly.

* **protected double maxWeight**

Maximum weight this type of aircraft can have, Declared in Aircraft.java, equal to 23000. Weight should never exceed maxWeight.

* **protected double floorArea**

Declared in PassengerAircraft.java, equal to 60. This field is used in the seat area calculations. The total seat area of your aircraft should never exceed floorArea.

* **protected double fuelCapacity**

Fuel capacity of the aircraft. Declared in Aircraft.java, equal to 6000. Fuel inside the aircraft should never exceed fuelCapacity. You can consider this value as fuel volume; fuel should be multiplied with fuelWeight to get the weight of a given volume of fuel.

* **protected double fuelConsumption**

Fuel consumption of the aircraft, which will be used in the getFuelConsumption() method. Declared in Aircraft.java, equal to 0.6.

* **protected double aircraftTypeMultiplier**

A constant to balance the usage of different aircrafts. This field was used when loading the passenger. Declared in Aircraft.java, equal to 0.9.

1. **WidebodyPassengerAircraft.java**

This class represents a large jet aircraft which is designed for very long routes and major destinations. It represents the Airbus A330.

* **getFlightCost(Airport toAirport)**

This method calculates and returns the flight cost. It is used in the fly() function of Aircraft.java. flight cost is made up of three costs. Landing cost, departure cost and flight operation cost.

To calculate flight operation cost we need three values: fullness, distance and a constant. Distance is the distance between the current airport and toAirport. Fullness is the ratio of occupied seats to all seats. The constant for this type of aircraft is 0.15. Flight operation cost is the multiplication of these three values. Departure and landing cost is explained in Airport classes.

* **getFuelConsumption(double distance)**

This method calculates the fuel consumption of the aircraft for a given distance, it should return the total fuel needed for the given distance.

Real aircraft have a bathtub curve for fuel consumption. This project uses a similar curve.

To calculate fuel consumption, we must define two values. First one is distance ratio. Distance ratio for this type of aircraft is distance -given as a parameter- divided by 14000. The second value is called the bathtub coefficient. This value is calculated by putting the distance ratio in the curve (y = 25.9324 x4 - 50.5633 x3 + 35.0554 x2 - 9.90346 x + 1.97413). Fuel consumption has two parts: a takeoff part, and a cruise part. Fuel consumption is the addition of these two values. Takeoff fuel consumption is weight times a constant divided by fuelWeight. We must divide by fuelWeight because we are calculating the volume of fuel. The constant for this type of aircraft is 0.1. Cruise fuel consumption is fuelConsumption times bathtub coefficient times distance.

Necessary fields:

* **protected double weight**

The actual weight of the aircraft. Declared in Aircraft.java, initial value -which corresponds to the empty weight- is 135000. Weight should never drop below 135000 if the program runs correctly.

* **protected double maxWeight**

Maximum weight this type of aircraft can have, Declared in Aircraft.java, equal to 250000. Weight should never exceed maxWeight.

* **protected double floorArea**

Declared in PassengerAircraft.java, equal to 450. This field is used in the seat area calculations. The total seat area of your aircraft should never exceed floorArea.

* **protected double fuelCapacity**

Fuel capacity of the aircraft. Declared in Aircraft.java, equal to 140000. Fuel inside the aircraft should never exceed fuelCapacity. You can consider this value as fuel volume; fuel should be multiplied with fuelWeight to get the weight of a given volume of fuel.

* **protected double fuelConsumption**

Fuel consumption of the aircraft, which will be used in the getFuelConsumption() method. Declared in Aircraft.java, equal to 3.0.

* **protected double aircraftTypeMultiplier**

A constant to balance the usage of different aircrafts. This field was used when loading the passenger. Declared in Aircraft.java, equal to 0.7.

1. **RapidPassengerAircraft.java**

This class represents a medium sized aircraft which is designed for medium range routes and major destinations. It is a representation of Concorde, which was a supersonic passenger aircraft.

* **getFlightCost(Airport toAirport)**

This method calculates and returns the flight cost. It is used in the fly() function of Aircraft.java. flight cost is made up of three costs. Landing cost, departure cost and flight operation cost.

To calculate flight operation cost we need three values: fullness, distance and a constant. Distance is the distance between the current airport and toAirport. Fullness is the ratio of occupied seats to all seats. The constant for this type of aircraft is 0.2. Flight operation cost is the multiplication of these three values. Departure and landing cost is explained in Airport classes.

* **getFuelConsumption(double distance)**

This method calculates the fuel consumption of the aircraft for a given distance, it should return the total fuel needed for the given distance.

Real aircraft have a bathtub curve for fuel consumption. This project uses a similar curve.

To calculate fuel consumption, we must define two values. First one is distance ratio. Distance ratio for this type of aircraft is distance -given as a parameter- divided by 7000. The second value is called the bathtub coefficient. This value is calculated by putting the distance ratio in the curve (y = 25.9324 x4 - 50.5633 x3 + 35.0554 x2 - 9.90346 x + 1.97413). Fuel consumption has two parts: a takeoff part, and a cruise part. Fuel consumption is the addition of these two values. Takeoff fuel consumption is weight times a constant divided by fuelWeight. We must divide by fuelWeight because we are calculating the volume of fuel. The constant for this type of aircraft is 0.1. Cruise fuel consumption is fuelConsumption times bathtub coefficient times distance.

Necessary fields:

* **protected double weight**

The actual weight of the aircraft. Declared in Aircraft.java, initial value -which corresponds to the empty weight- is 80000. Weight should never drop below 80000 if the program runs correctly.

* **protected double maxWeight**

Maximum weight this type of aircraft can have, Declared in Aircraft.java, equal to 185000. Weight should never exceed maxWeight.

* **protected double floorArea**

Declared in PassengerAircraft.java, equal to 120. This field is used in the seat area calculations. The total seat area of your aircraft should never exceed floorArea.

* **protected double fuelCapacity**

Fuel capacity of the aircraft. Declared in Aircraft.java, equal to 120000. Fuel inside the aircraft should never exceed fuelCapacity. You can consider this value as fuel volume; fuel should be multiplied with fuelWeight to get the weight of a given volume of fuel.

* **protected double fuelConsumption**

Fuel consumption of the aircraft, which will be used in the getFuelConsumption() method. Declared in Aircraft.java, equal to 5.3.

* **protected double aircraftTypeMultiplier**

A constant to balance the usage of different aircrafts. This field was used when loading the passenger. Declared in Aircraft.java, equal to 1.9.

1. **JetPassengerAircraft.java**

This class represents a private jet which is designed for short to medium range routes. It has a small floor area so it should be used for squeezing the maximum amount of profit from luxury passengers.

* **getFlightCost(Airport toAirport)**

This method calculates and returns the flight cost. It is used in the fly() function of Aircraft.java. flight cost is made up of three costs. Landing cost, departure cost and flight operation cost.

To calculate flight operation cost we need three values: fullness, distance and a constant. Distance is the distance between the current airport and toAirport. Fullness is the ratio of occupied seats to all seats. The constant for this type of aircraft is 0.08. Flight operation cost is the multiplication of these three values. Departure and landing cost is explained in Airport classes.

* **getFuelConsumption(double distance)**

This method calculates the fuel consumption of the aircraft for a given distance, it should return the total fuel needed for the given distance.

Real aircraft have a bathtub curve for fuel consumption. This project uses a similar curve.

To calculate fuel consumption, we must define two values. First one is distance ratio. Distance ratio for this type of aircraft is distance -given as a parameter- divided by 5000. The second value is called the bathtub coefficient. This value is calculated by putting the distance ratio in the curve (y = 25.9324 x4 - 50.5633 x3 + 35.0554 x2 - 9.90346 x + 1.97413). Fuel consumption has two parts: a takeoff part, and a cruise part. Fuel consumption is the addition of these two values. Takeoff fuel consumption is weight times a constant divided by fuelWeight. We must divide by fuelWeight because we are calculating the volume of fuel. The constant for this type of aircraft is 0.1. Cruise fuel consumption is fuelConsumption times bathtub coefficient times distance.

Necessary fields:

* **protected double weight**

The actual weight of the aircraft. Declared in Aircraft.java, initial value -which corresponds to the empty weight- is 10000. Weight should never drop below 10000 if the program runs correctly.

* **protected double maxWeight**

Maximum weight this type of aircraft can have, Declared in Aircraft.java, equal to 18000. Weight should never exceed maxWeight.

* **protected double floorArea**

Declared in PassengerAircraft.java, equal to 30. This field is used in the seat area calculations. The total seat area of your aircraft should never exceed floorArea.

* **protected double fuelCapacity**

Fuel capacity of the aircraft. Declared in Aircraft.java, equal to 10000. Fuel inside the aircraft should never exceed fuelCapacity. You can consider this value as fuel volume; fuel should be multiplied with fuelWeight to get the weight of a given volume of fuel.

* **protected double fuelConsumption**

Fuel consumption of the aircraft, which will be used in the getFuelConsumption() method. Declared in Aircraft.java, equal to 0.7.

* **protected double aircraftTypeMultiplier**

A constant to balance the usage of different aircrafts. This field was used when loading the passenger. Declared in Aircraft.java, equal to 5.

1. **Input & Output Format**
2. **Input**
3. **Output**
4. **Examples**
5. **Grading**

**(TODO: LATE SUBMISSION)**

1. **Warnings**

Any revenue generated or fee paid should be written to the output. You will not get any points if your output and operation log do not match.

Make sure that a child method implements the methods by using abstract keyword in parent classes. Every method must be generated in the appropriate class. Do not write every function in the last concrete class.

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.