Specifications:

* Overview: In this project a simulation of a private airlines is created which consists following elements:
* Requirements: Set of classes with given specifications

1. Passenger:
2. Attributes:
3. Id: String
4. Name: String
5. Age: int
6. Methods:
7. Conductor that take id, name and age as a parameter
8. getId():String and setId(id: String):void
9. getName():String and setName(name:String):void
10. getAge():int and setAge(age:int):void
11. MilesNSmilesMember- an abstract class child of passenger:

1. Attributes: None

b) Methods:

1. Conductor that take id, name, age as a parameter
2. getCardId():String
3. addCardMiles(Flight flight):void – an abstract method to add miles to card from the flight
4. getCardMiles():double – an abstract method which returns card miles
5. loungePrivelage():String- an abstract method to display if the member allowed to access the lounge
6. toString():String – an abstract method
7. ClassicMember – child of MilesNSmilesMember
8. Attributes:
9. cardId: String
10. milePercentage=0.1 :double
11. cardMiles: double
12. Methods:
13. Conductor that take id, name, age and cardId as a parameter
14. getCardId():String
15. addCardMiles(Flight flight): void
16. Method that take flight as a parameter and use getMile() method from flight
17. By multiplying miles from flight with milepercentage calculates how many miles will be added to card
18. getCardMiles():double – a method which returns card miles
19. loungePrivelage():String

1. Method checks if card miles is greater or equal to the 500 mile if so returns “Permitted ” and reduces 500 miles from the card

2. Else returns “Not Allowed”

1. toString():String - "Classic Member: {name} Card Id: {card id} Card Mile Total: {total card miles}
2. EliteMember – child of MilesNSmilesMember
3. Attributes:
4. cardId:Stirng
5. milePercentage=0.3 :double
6. cardMiles: double
7. Methods:
8. getCardId():String
9. Conductor that take id, name, age and cardId as a parameter
10. addCardMiles(Flight flight): void

1. Method that take flight as a parameter and use getMile() method from flight

2. By multiplying miles from flight with milepercentage calculates how many miles will be added to card

1. getCardMiles():double – a method which returns card miles
2. loungePrivelage():String – a method returns “permitted”
3. toString():String - "Elite Member: {name} Card Id: {card id} Card Mile Total: {total card miles}
4. Pilot:
5. Attributes:
6. Name: String
7. Id: String
8. License: String
9. hoursFlighted : double
10. Methods:
11. Conductor that take name, id and license as a parameter

1. Set hoursFlighted to zero.

1. getId():String
2. getName():String
3. getLicense():String
4. getHoursFlighted():double
5. addHours(double hours):void
6. If parameter hours greater than zero add the parameter to hoursFlighted
7. Else do nothing
8. initializeHours():void
9. Set the hoursFlighted zero
10. available():boolean
11. If hoursFlighted is smaller than 100 hours return true and print Pilot {name} Id : {id} is available for flight with weekly hours flighted {hours flighted}
12. Else return false and print Pilot {name} Id : {id} is not available for flight with weekly hours flighted {hours flighted}
13. toString:String
14. "Name : {name} Id : {id} License : {license}
15. <<interface>>IAircraft:

1. Attributes:
2. default\_availability=true : boolean
3. default\_safetyStat= true :boolean
4. defaultPilot\_requ=2: int
5. Methods:
6. getModel():String
7. getTailNumber():String
8. getSeatCapacity():int
9. getPilotReq():int
10. getAvailability():boolean and

setAvailability(boolean newStat):void

1. getSafetyStat():boolean and

setSafetyStat(boolean newStat):void

1. CommercialPlane – implements IAircraft
2. Attributes:
3. Model:String
4. TailNumber: String
5. SeatCapacity:int
6. PilotReq:int
7. Availability:boolean
8. SafetyStat: boolean
9. Methods:
10. Conductor that take model, tailNumber , seatCapacity and pilotReq as a parameter
11. getTailNumber():String
12. getSeatCapacity():int
13. getPilotReq():int
14. getAvailability():boolean and

setAvailability(boolean newStat):void

1. getSafetyStat():boolean and

setSafetyStat(boolean newStat):void

1. JetPlane- implements IAircraft
2. Attributes:
3. Model:String
4. TailNumber: String
5. SeatCapacity:int
6. PilotReq:int
7. Availability:boolean
8. SafetyStat: boolean
9. Methods:
10. Conductor that take model, tailNumber , seatCapacity as a parameter
11. Set availability to default\_availability and set pilotReq to default\_Pilotreq
12. getTailNumber():String
13. getSeatCapacity():int
14. getPilotReq():int
15. getAvailability():boolean and

setAvailability(boolean newStat):void

1. getSafetyStat():boolean and

setSafetyStat(boolean newStat):void

1. Flight
2. Attributes:
3. flightId:Sring
4. departureAirport:String
5. arrivalAirport:String
6. date:String
7. time:String
8. mile: double
9. reqLicense:String
10. plane:IAircraft
11. Array of Pilot
12. Methods:
13. Conductor that take flghtId, departureAirport, arrivalAirport, mile, date and time as a parameter
14. Creates new ArrayList of Passenger
15. getFlightId():String
16. getArrivalAirport:String and

setArrivalAirport(String airport):void

1. getDepartureAirport():String

and setDepartureAirport(String airport):void

1. getMile():double
2. getDate():String and setDate(String date):void
3. getTime():String and setTime(String time):void
4. assignPlane(IAircraft plane):String
5. Checks if plane is available by using getAvailability() method and safetyStat by using getSafetyStat() method
6. If plane is available and safe
7. creates array of Pilot with size of pilotReq
8. sets reqLicense to plane model
9. set capacity to plane seat capacity
10. creates String array of seats to assign passengers
11. assign this plane to the flight and set plane availability false
12. returns plane {model} with tail number {tail number} is assigned to the flight
13. If plane is not available or safe
14. Returns plane {model} with tail number {tail number} is not assigned to the flight
15. getPlane():IAircraft
16. Returns assigned aircraft
17. addPilot(Pilot pilot):void
18. Checks if pilot license is reqLicense and if pilot is available by hoursFlighted
19. Adds Pilot to the array of Pilot
20. If not throws PilotNotValidException
21. getPilots():void
22. Prints Pilots assigned to the flight
23. getCharterPrice():double
24. returns mile of flight multiply by 10.1
25. toString():String
26. returns Flight No: {flightId}

1. SalesCouter
2. Attributes:
3. flight: Flight
4. passenger: ArrayList of Passenger
5. initialPrice: double
6. capacity: int
7. seats: Array of String
8. line: int
9. totalRevenue: double
10. Methods:
11. Conductor that takes flight, initialPrice as parameter
12. Creates Array of seats using capacity of seats of plane that assigned flight
13. getPrice():double
14. If sold seats smaller than %30 of the seat capacity of plane returns initialPrice
15. If sold seats smaller than %50 of the seat capacity of plane returns initialPrice\*1.5
16. Else returns initialPrice\*2
17. buyTicket(Passenger passenger, Double payment):String
18. If there is available seat in the plane and passenger did not buy a ticket for that flight before add passenger to the arrayList of passenger, add price of the seat to totalRevenue and return

Date of your purchase is {Date that ticket bought} Cost of your purchase is { Price of seat} $

If payment is not enough return Not enough payment

1. Else throw NotAvailableSeatException
2. checkIn(Passenger passenger): String
3. If passenger buy a ticket before,

returns {passenger name} seat: {seat number} flight:{flightId} departure airport: {departure Airport } arrival airport: {arrival airport} date of departure: {date } time of departure:{time}

1. Else returns this passenger {passenger name } is not found
2. displayPassengers():String
3. Displays the passengers who buy ticket for that specific flight
4. assignPrivateFlight(Passenger passenger, double payment):String
5. If payment is enough, adds the charter of private plane to total Revenue and returns This Private Flight assigned to passenger {passenger name} CardId : {cardId}

Charter Price : "{price of the flight} $

Flight Number : {Flight id}

Departure Airport :{airport name}

Arrival Airport : {Airport Name}

Date of Departure : {date}

Time of Departure : {time}

Jet Plane : {model} Jet Plane Capacity for Passengers : {seat capacity}

1. Else return payment is not concluded
2. getTotalRevenue():double
3. Returns total revenue of tickets and private flights that sold by this sales counter
4. getFlight:Flight
5. Returns flight
6. NotAvailableSeatException
7. Attributes:
8. Passenger:Passenger
9. Methods:
10. Conductor that take passenger as a parameter
11. toString():String-"NotAvailableSeatException: There is no seat available for "{passenger name}
12. PilotNotValidException
13. Attributes:
14. Pilot:Pilot
15. reqLicense:Stirng
16. Hours:double
17. Methods:
18. Conductor that take pilot and reqLicense as a parameter
19. toString():String- PilotNotValidException: "{pilot name} Required License is {Aircraft model}

Flighted hours must me less than 100 hours

Flighted hours : {pilot flighted hours}

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**import** java.time.LocalDate;

**import** java.util.ArrayList;

**public** **class** Midterm\_20195156019 {

}

**class** Passenger {

**private** String id;

**private** String name;

**private** **int** age;

**public** Passenger(String id, String name, **int** age) {

**this**.id = id;

**this**.name = name;

**this**.age = age;

}

**public** Passenger() {

}

**public** String getId() {

**return** **this**.id;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getName() {

**return** **this**.name;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

**public** **int** getAge() {

**return** **this**.age;

}

@Override

**public** String toString() {

**return** **this**.name;

}

}

**abstract** **class** MilesNSmilesMember **extends** Passenger {

**protected** MilesNSmilesMember(String id, String name, **int** age) {

**super**(id, name, age);

}

**public** **abstract** **void** addCardMiles(Flight flight);

**public** **abstract** **double** getCardMiles();

**public** **abstract** String getCardId();

**public** **abstract** String loungePrivelage();

**public** **abstract** String toString();

}

**class** ClassicMember **extends** MilesNSmilesMember {

**private** String cardId;

**private** **static** **final** **double** ***milePercentage*** = 0.1;

**private** **double** cardMiles;

**protected** ClassicMember(String id, String name, **int** age, String cardId) {

**super**(id, name, age);

**this**.cardId = cardId;

}

@Override

**public** **void** addCardMiles(Flight flight) {

cardMiles += ***milePercentage*** \* flight.getMile();

}

@Override

**public** **double** getCardMiles() {

**return** **this**.cardMiles;

}

@Override

**public** String getCardId() {

**return** **this**.cardId;

}

@Override

**public** String loungePrivelage() {

**if** (**this**.cardMiles >= 500) {

**this**.cardMiles = **this**.cardMiles - 500;

**return** "Loung access is permitted";

} **else**

**return** "Lounge access is not permitted for " + **this**.getName();

}

@Override

**public** String toString() {

**return** "Classic Member: " + **this**.getName() + " Card Id: " + **this**.cardId + " Card Mile Total: " + **this**.cardMiles;

}

}

**class** EliteMember **extends** MilesNSmilesMember {

**private** String cardId;

**private** **static** **final** **double** ***milePercentage*** = 0.3;

**private** **double** cardMiles;

**protected** EliteMember(String id, String name, **int** age, String cardId) {

**super**(id, name, age);

**this**.cardId = cardId;

}

@Override

**public** **void** addCardMiles(Flight flight) {

**this**.cardMiles += ***milePercentage*** \* flight.getMile();

}

@Override

**public** **double** getCardMiles() {

**return** **this**.cardMiles;

}

@Override

**public** String getCardId() {

**return** **this**.cardId;

}

@Override

**public** String toString() {

**return** "Elite Member: " + **this**.getName() + " Card Id: " + **this**.cardId + " Card Mile Total: " + **this**.cardMiles;

}

@Override

**public** String loungePrivelage() {

**return** "Lounge access is permitted for " + **this**.getName();

}

}

**interface** IAircraft {

**public** **static** **boolean** ***default\_availability*** = **true**;

**public** **static** **boolean** ***default\_safetyStat*** = **true**;

**public** **static** **final** **int** ***defaultPilot\_requ*** = 2;

**public** String getModel();

**public** String getTailNumber();

**public** **int** getSeatCapacity();

**public** **int** getPilotReq();

**public** **boolean** getAvailability();

**public** **boolean** getSafetyStat();

**public** **void** setAvailability(**boolean** newStat);

**public** **void** setSafetyStat(**boolean** newStat);

}

**class** CommercialPlane **implements** IAircraft {

**private** String model;

**private** String tailNumber;

**private** **int** seatCapacity;

**private** **int** pilot\_req;

**private** **boolean** availability;

**private** **boolean** safetyStat;

**public** CommercialPlane(String model, String tailNumber, **int** seatCapacity, **int** pilot\_req) {

**this**.model = model;

**this**.tailNumber = tailNumber;

**this**.seatCapacity = seatCapacity;

**this**.availability = ***default\_availability***;

**this**.safetyStat = ***default\_safetyStat***;

**this**.pilot\_req = pilot\_req;

}

@Override

**public** **int** getSeatCapacity() {

**return** **this**.seatCapacity;

}

@Override

**public** **int** getPilotReq() {

**return** **this**.pilot\_req;

}

@Override

**public** **boolean** getAvailability() {

**return** **this**.availability;

}

@Override

**public** String getModel() {

**return** **this**.model;

}

@Override

**public** String getTailNumber() {

**return** **this**.tailNumber;

}

@Override

**public** **boolean** getSafetyStat() {

**return** **this**.safetyStat;

}

@Override

**public** **void** setAvailability(**boolean** newStat) {

**this**.availability = newStat;

}

@Override

**public** **void** setSafetyStat(**boolean** newStat) {

**this**.safetyStat = newStat;

}

}

**class** JetPlane **implements** IAircraft {

**private** String model;

**private** String tailNumber;

**private** **int** seatCapacity;

**private** **boolean** availability;

**private** **boolean** safetyStat;

**public** JetPlane(String model, String tailNumber, **int** seatCapacity) {

**this**.model = model;

**this**.tailNumber = tailNumber;

**this**.seatCapacity = seatCapacity;

**this**.availability = ***default\_availability***;

**this**.safetyStat = ***default\_safetyStat***;

}

@Override

**public** **int** getSeatCapacity() {

**return** **this**.seatCapacity;

}

@Override

**public** **int** getPilotReq() {

**return** JetPlane.***defaultPilot\_requ***;

}

@Override

**public** **boolean** getAvailability() {

**return** **this**.availability;

}

@Override

**public** String getModel() {

**return** **this**.model;

}

@Override

**public** String getTailNumber() {

**return** **this**.tailNumber;

}

@Override

**public** **boolean** getSafetyStat() {

**return** safetyStat;

}

@Override

**public** **void** setAvailability(**boolean** newStat) {

**this**.availability = newStat;

}

@Override

**public** **void** setSafetyStat(**boolean** newStat) {

**this**.safetyStat = newStat;

}

}

**class** Pilot {

**private** String name;

**private** String id;

**private** **double** hours\_Flighted;

**private** String license;

**public** Pilot(String name, String id, String license) {

**this**.name = name;

**this**.id = id;

**this**.license = license;

**this**.hours\_Flighted = 0;

}

**public** **void** addHours(**double** hours) {

**if** (hours >= 0) {

**this**.hours\_Flighted += hours;

}

}

**public** **void** initializeHours() {

**this**.hours\_Flighted = 0;

}

**public** **boolean** available() {

**if** (**this**.hours\_Flighted <= 100.0) {

System.***out***.println("\nPilot " + **this**.name + " Id : " + **this**.id

+ " is available for fligth with weekly hours flighted " + **this**.hours\_Flighted + "\n ");

**return** **true**;

} **else**

**return** **false**;

}

**public** String getName() {

**return** name;

}

**public** String getId() {

**return** id;

}

**public** **double** getHoursFlighted() {

**return** **this**.hours\_Flighted;

}

@Override

**public** String toString() {

**return** "Name :" + **this**.name + " Id :" + **this**.id + " License : " + **this**.license;

}

**public** String getLicense() {

**return** license;

}

}

**class** Flight {

**private** String flightId;

**private** Pilot[] pilot;

**private** String departureAirport;

**private** String arrivalAirport;

**private** String date;

**private** String time;

**private** **double** mile;

**private** String reqLicense;

**private** IAircraft plane;

**public** Flight(String flightId, String departureAirport, String arrivalAirport, **double** mile, String date,

String time) {

**this**.flightId = flightId;

**this**.setDepartureAirport(departureAirport);

**this**.setArrivalAirport(arrivalAirport);

**this**.mile = mile;

**this**.date = date;

**this**.time = time;

}

**public** String getFlightId() {

**return** flightId;

}

**public** **void** addPilot(Pilot pilot1) **throws** PilotNotValidException {

// If pilot got the license for required flight then it can be assigned to the

// flight

**for** (**int** i = 0; i < **this**.pilot.length; i++) {

**if** (**this**.pilot[i] == **null**) {

**if** (reqLicense == pilot1.getLicense() && pilot1.available() == **true**) {

**this**.pilot[i] = pilot1;

**break**;

}

**else** {

**throw** **new** PilotNotValidException(pilot1, **this**.reqLicense, pilot1.getHoursFlighted());

}

}

}

}

**public** **void** getPilots() {

System.***out***.println("Pilots assigned to the flight " + **this**.flightId + " are :");

**for** (**int** i = 0; i < pilot.length; i++) {

System.***out***.println(pilot[i]);

}

}

**public** String getDepartureAirport() {

**return** departureAirport;

}

**public** **void** setDepartureAirport(String departureAirport) {

**this**.departureAirport = departureAirport;

}

**public** String getArrivalAirport() {

**return** arrivalAirport;

}

**public** **void** setArrivalAirport(String arrivalAirport) {

**this**.arrivalAirport = arrivalAirport;

}

**public** IAircraft getPlane() {

**return** **this**.plane;

}

**public** String assignPlane(IAircraft plane) {

**if** (plane.getAvailability() == **true** && plane.getSafetyStat() == **true**) {

**this**.pilot = **new** Pilot[plane.getPilotReq()];

reqLicense = plane.getModel();

**this**.plane = plane;

plane.setAvailability(**false**);

**return** "\nPlane " + plane.getModel() + " with tail number " + plane.getTailNumber() + " is assigned to "

+ **this**.flightId;

} **else**

**return** "\nPlane " + plane.getModel() + " with tail number " + plane.getTailNumber() + " is not assigned to "

+ **this**.flightId;

}

**public** **double** getMile() {

**return** mile;

}

**public** String getDate() {

**return** date;

}

**public** **void** setDate(String date) {

**this**.date = date;

}

**public** String getTime() {

**return** time;

}

**public** **void** setTime(String time) {

**this**.time = time;

}

**public** **double** getCharterPrice() {

**return** **this**.mile \* 10.1;

}

@Override

**public** String toString() {

**return** "Flight No: " + **this**.flightId;

}

}

**class** SalesCounter {

**private** Flight flight;

**private** ArrayList<Passenger> passenger;

**private** **double** initialprice;

**private** **int** capacity;

**private** String[] seats;

**private** **int** line;

**private** **double** totalRevenue;

**public** SalesCounter(Flight flight, **double** initialprice) {

**this**.flight = flight;

**this**.initialprice = initialprice;

**this**.passenger = **new** ArrayList<Passenger>();

**this**.capacity = flight.getPlane().getSeatCapacity();

**if** (capacity % 6 != 0) {

**int** a = capacity % 6;

capacity = capacity - a;

line = capacity / 6 + 1;

capacity = flight.getPlane().getSeatCapacity();

}

**this**.seats = **new** String[capacity];

String[] harf = { "A", "B", "C", "D", "E", "F" };

**int** m = 0;

**for** (**int** i = 1; i < line; i++) {

**for** (**int** j = 0; j < 6; j++) {

seats[m] = i + harf[j % 6];

m++;

}

}

}

**public** **double** getPrice() {

**if** (**this**.passenger.size() < **this**.flight.getPlane().getSeatCapacity() \* 30 / 100)

**return** initialprice;

**else** **if** (**this**.passenger.size() < **this**.flight.getPlane().getSeatCapacity() \* 50 / 100)

**return** initialprice \* 1.50;

**else**

**return** initialprice \* 2;

}

**public** String buyTicket(Passenger passenger, **double** payment) **throws** NotAvailableSeatException {

**if** (capacity - **this**.passenger.size() > 0 && !**this**.passenger.contains(passenger)) {

**if** (payment >= **this**.getPrice()) {

**this**.passenger.add(passenger);

totalRevenue += **this**.getPrice();

String m = "Date of your purchase is " + LocalDate.*now*() + " Cost of your purchase is "

+ **this**.getPrice() + "$";

**return** m;

} **else**

**return** "Not enought payment ";

} **else**

**throw** **new** NotAvailableSeatException(passenger);

}

**public** String CheckIn(Passenger passenger) {

**if** (**this**.passenger.contains(passenger)) {

**return** "\n" + passenger.getName() + " Seat : " + seats[**this**.passenger.size() - 1] + "\n"

+ "Flight Number : " + flight.getFlightId() + "\nDeparture Airport : "

+ flight.getDepartureAirport() + "\nArrival Airport : " + flight.getArrivalAirport()

+ "\nDate of Departure : " + flight.getDate() + "\nTime of Departure : " + flight.getTime();

} **else**

**return** "this passenger" + passenger.getName() + " is not found";

}

**public** String displayPassenger() {

String m = "";

**for** (**int** i = 0; i < **this**.passenger.size(); i++) {

m += passenger.get(i).getName() + "\n";

}

**return** m;

}

**public** **double** getTotalRevenue() {

**return** **this**.totalRevenue;

}

**public** Flight getFlight() {

**return** **this**.flight;

}

**public** String assignPrivateFlight(Passenger passenger, Flight flight, **double** payment) {

**if** (payment > flight.getCharterPrice()) {

**this**.flight.getPlane().setAvailability(**false**);

totalRevenue += flight.getCharterPrice();

**return** "This Private Fligt assigned to passenger " + passenger.getName() + " CardId : " + passenger.getId()

+ " \nCharter Price : " + flight.getCharterPrice() + "$" + "\nFlight Number : "

+ flight.getFlightId() + "\nDeparture Airport : " + flight.getDepartureAirport()

+ "\nArrival Airport : " + flight.getArrivalAirport() + "\nDate of Departure : " + flight.getDate()

+ "\nTime of Departure : " + flight.getTime() + "\nJet Plane : " + flight.getPlane().getModel()

+ "\nJet Plane Capacity for Passengers : " + flight.getPlane().getSeatCapacity();

} **else**

**return** "Payment is not concluded";

}

}

**class** NotAvailableSeatException **extends** RuntimeException {

**private** Passenger passenger;

**public** NotAvailableSeatException(Passenger passenger) {

**this**.passenger = passenger;

}

**public** NotAvailableSeatException() {

}

@Override

**public** String toString() {

**return** "NotAvailableSeatException: There is no seat available for " + **this**.passenger;

}

}

**class** PilotNotValidException **extends** RuntimeException {

**private** Pilot pilot;

**private** String reqLicense;

**double** hours;

**public** PilotNotValidException(Pilot pilot, String reqLicense, **double** hours) {

**this**.pilot = pilot;

**this**.reqLicense = reqLicense;

**this**.hours = hours;

}

**public** PilotNotValidException() {

}

@Override

**public** String toString() {

**return** "PilotNotValidException: " + **this**.pilot + " Required License is " + **this**.reqLicense

+ " \nRequired Hours\_Flighted must me less than 100 hours\nThe Hours Flighted :" + **this**.hours;

}

}