**AIRLINE RESERVATION SYSTEM**

**ABSTRACT:**

A Customer opens the application and gives his departure and arrival details . If the flight between those destinations is available he will enter the date of travel , no of seats and whether he want economy or business class. The price will be displayed and if the customer wants cancellation policy he needs to pay a certain amount for 100% refund and if he doesn’t take the cancellation policy he will only get certain amount as refund.

Every time a flight is booked the amount is added to the airline bank and when a customer opts for cancellation it checks if bank has the required amount else the refund will be granted in two weeks.

If the Customer is premium card holder he will refunded the total ticket price without paying the required amount.

**OBJECT MODELING:**

1. Class: Customer

- Attributes:

- customerId: String

- Behaviors:

- Constructor: Initializes the customerId of the customer.

- CancellationPolicy(int cost) : int

- Input : cost (int) – the cost of the ticket is taken.

- Output : t\_price (int) – the total price including prior cancellation charges are added and given.

2. Class: PremiumCardHolder (Inherits from Customer)

- Attributes:

- connection : Connection

- cust\_name: String

- Behaviors:

- Constructor: Initializes the PremiumCardHolder with a database connection and initializes customerId of the premium cardholder.

- isPremium(String customerId): boolean

- Input: customerId (String) - The customer ID to check if the customer is a premium cardholder.

- Output: boolean - Returns true if the customer is a premium cardholder, false otherwise.

3. Class: FlightAvailable

- Attributes:

- departure: String

- arrival: String

- Behaviors:

- Constructor: Initializes the departure and arrival cities.

4. Class: FlightDate (Inherits from FlightAvailable)

- Attributes:

- dateOfTravel: String

- Behaviors:

- Constructor: Initializes the departure, arrival, and date of travel.

5. Class: FlightBook (Inherits from FlightDate)

- Attributes:

- type: char ('e' for Economy, 'b' for Business)

- seats: int

- Behaviors:

- Constructor: Initializes the departure, arrival, date of travel, type (economy/business), and number of seats.

6. Class: FlightManager

- Attributes:

- connection: Connection

- cost: int

- fid: int

- Behaviors:

- Constructor: Initializes the FlightManager with a database connection.

- checkFlight(FlightAvailable flight): boolean

- Input: flight (FlightAvailable) - The flight to check for availability.

- Output: boolean - Returns true if the flight is available, false otherwise.

- getFlightByDate(FlightDate flightdate): boolean

- Input: flightdate (FlightDate) - The flight on a specific date to check for availability.

- Output: boolean - Returns true if the flight is available on the specified date, false otherwise.

- getFlightBySeats(FlightBook flight1): boolean

- Input: flight1 (FlightBook) - The flight to book based on the number of seats and type.

- Output: boolean - Returns true if the flight is available with the specified number of seats, false otherwise.

7. Class: Airline

- Attributes:

- connection: Connection

- Behaviors:

- Constructor: Initializes the Airline with a database connection.

- updateAirlineBank(int amount): void

- Input: amount (int) - The amount to update the airline's bank balance.

- Behavior: Updates the airline's bank balance with the given amount.

- getAirlineBank(): int

- Output: int - Returns the current airline's bank balance.

- hasEnoughFunds(int amount): boolean

- Input: amount (int) - The amount to check if the airline has enough funds.

- Output: boolean - Returns true if the airline has enough funds, false otherwise.

8. Class: AirlineApplication (Main class)

* Behaviors:
* main(String[] args): The main method that contains the user interface for the airline application. It allows users to book and cancel flights, handles premium customers, and updates the database accordingly.

The above object model provides a detailed representation of the Airline Application, including classes, their attributes, behaviors, and state.

**CLASS DIAGRAM:**

**+-----------------------------------------------+**

**| Customer |**

**+-----------------------------------------------+**

**| customerId: String |**

**+-----------------------------------------------+**

**| Customer(customerId: String) |**

**| CancellationPolicy(cost: int): int |**

**+-----------------------------------------------+**

**^**

**|**

**+------------------------------------------------------------------+**

**| FlightAvailable |**

**+------------------------------------------------------------------+**

**| departure: String |**

**| arrival: String |**

**+------------------------------------------------------------------+**

**| FlightAvailable(departure: String, arrival: String) |**

**+------------------------------------------------------------------+**

**^**

**|**

**+-----------------------------------------------------------------------------------------+**

**| FlightDate |**

**+-----------------------------------------------------------------------------------------+**

**| dateOfTravel: String |**

**+-----------------------------------------------------------------------------------------+**

**| FlightDate(departure: String, arrival: String, dateOfTravel: String) |**

**+-----------------------------------------------------------------------------------------+**

**^**

**|**

**+----------------------------------------------------------------------------------------------------------------------+**

**| FlightBook |**

**+----------------------------------------------------------------------------------------------------------------------+**

**| type: char |**

**| seats: int |**

**+----------------------------------------------------------------------------------------------------------------------+**

**|FlightBook(departure: String, arrival: String, dateOfTravel: String, type: char, seats: int) |**

**+----------------------------------------------------------------------------------------------------------------------+**

**^**

**|**

**+-----------------------------------------------------------------------------------------+**

**| PremiumCardHolder |**

**+-----------------------------------------------------------------------------------------+**

**| connection: Connection |**

**| cust\_name: String |**

**+------------------------------------------------------------------------------------------+**

**| PremiumCardHolder(connection: Connection, customerId: String) |**

**| isPremium(customerId: String): boolean |**

**+------------------------------------------------------------------------------------------+**

**^**

**|**

**+-------------------------------------------------------------------+**

**| FlightManager |**

**+--------------------------------------------------------------------+**

**| connection: Connection |**

**| cost: int |**

**| fid: int |**

**+--------------------------------------------------------------------+**

**| FlightManager(connection: Connection) |**

**| checkFlight(flight: FlightAvailable): boolean |**

**| getFlightByDate(flightdate: FlightDate): boolean |**

**| getFlightBySeats(flight1: FlightBook): boolean |**

**+--------------------------------------------------------------------+**

**^**

**|**

**+---------------------------------------------------------------+**

**| Airline |**

**+---------------------------------------------------------------+**

**| connection: Connection |**

**+---------------------------------------------------------------+**

**| Airline(connection: Connection) |**

**| updateAirlineBank(amount: int): void |**

**| getAirlineBank(): int |**

**| hasEnoughFunds(amount: int): boolean |**

**+---------------------------------------------------------------+**

**^**

**|**

**+------------------------------------------+**

**| AirlineApplication |**

**+------------------------------------------+**

**| connection: Connection |**

**+------------------------------------------+**

**| main(args: String[]): void |**

**+------------------------------------------+**

**WORK FLOW:**

1. The application starts, and the main method in `AirlineApplication` is executed.

2. The user is presented with a menu to choose between booking a flight and canceling a booking.

3. If the user selects booking a flight:

a. The user is prompted to enter the departure and arrival cities.

b. The `FlightManager` checks if a flight is available between the specified departure and arrival cities by querying the database.

c. If a flight is available, the user is asked to enter the date of travel.

d. The `FlightManager` checks if the flight is available on the specified date by querying the database.

e. If the flight is available on the specified date, the user is asked to choose between Economy (e) or Business (b) class and the number of seats they want to book.

f. The `FlightManager` checks if there are enough available seats of the selected class for the specified date by querying the database.

g. If seats are available, the user is prompted to confirm the booking.

h. If the user confirms the booking, they are asked if they are a premium customer (holder of a premium card).

i. If the user is a premium customer, their premium status is checked, and if confirmed, the booking is processed without any cancellation policy fees.

j. If the user is not a premium customer, they are prompted to enter their name, and then asked if they want to add a cancellation policy.

k. If the user chooses to add a cancellation policy, the total price of the booking is adjusted with an additional fee of 3000, and the booking is processed.

l. If the user chooses not to add a cancellation policy, the booking is processed without any additional fees.

m. The ticket details are stored in the `tickets` table in the database, and the corresponding flight's available seats are updated.

n. The airline's bank balance is updated with the ticket price.

4. If the user selects canceling a booking:

a. The user is asked to enter the ticket ID of the booking they want to cancel.

b. The `Airline` class checks if there is sufficient balance in the airline's bank for the refund.

c. If there is sufficient balance, the refund is granted and the user is informed.

d. The corresponding flight's available seats are updated, and the ticket is removed from the `tickets` table.

e. If there is insufficient balance, the user is informed that the refund will be given shortly.

5. The user is asked if they want to continue using the application or exit.

6. If the user chooses to continue, they go back to the main menu; otherwise, the application terminates.

**CODE:**

import java.sql.\*;

import java.util.\*;

import java.util.Random;

class Customer {

String customerId;

public Customer(String customerId) {

this.customerId = customerId;

}

public int CancellationPolicy(int cost) {

int t\_price = cost + 3000;

return t\_price;

}

}

class FlightAvailable {

String departure;

String arrival;

public FlightAvailable(String departure, String arrival) {

this.departure = departure;

this.arrival = arrival;

}

}

class FlightDate extends FlightAvailable {

String dateOfTravel;

public FlightDate(String departure, String arrival, String dateOfTravel) {

super(departure, arrival);

this.dateOfTravel = dateOfTravel;

}

}

class FlightBook extends FlightDate {

char type;

int seats;

public FlightBook(String departure, String arrival, String dateOfTravel, char type, int seats) {

super(departure, arrival, dateOfTravel);

this.type = type;

this.seats = seats;

}

}

class FlightManager {

Connection connection;

int cost = 0;

int fid = 0;

public FlightManager(Connection connection) {

this.connection = connection;

}

public boolean checkFlight(FlightAvailable flight) {

boolean flag = false;

try {

PreparedStatement check\_Flight = connection.prepareStatement("Select depature, arrival from flights where depature=? and arrival=?");

check\_Flight.setString(1, flight.departure);

check\_Flight.setString(2, flight.arrival);

ResultSet flightResultSet1 = check\_Flight.executeQuery();

if (flightResultSet1.next()) {

System.out.println("Flight available");

flag = true;

} else {

System.out.println("Flight not available");

flag = false;

}

} catch (SQLException e) {

e.printStackTrace();

}

return flag;

}

public boolean getFlightByDate(FlightDate flightdate) {

boolean flag = false;

try {

PreparedStatement check\_Flight1 = connection.prepareStatement("Select date from flights where date=? and depature=? and arrival=?");

check\_Flight1.setString(1, flightdate.dateOfTravel);

check\_Flight1.setString(2, flightdate.departure);

check\_Flight1.setString(3, flightdate.arrival);

ResultSet flightResultSet2 = check\_Flight1.executeQuery();

if (flightResultSet2.next()) {

System.out.println("Flight available on this date");

flag = true;

} else {

System.out.println("Flight not available on this date");

flag = false;

}

} catch (SQLException e) {

e.printStackTrace();

}

return flag;

}

public boolean getFlightBySeats(FlightBook flight1) {

boolean flag = false;

try {

if (flight1.type == 'e') {

PreparedStatement selectFlight = connection.prepareStatement("SELECT e\_price, f\_id FROM flights WHERE date= ? AND e\_seats >= ? AND depature=? AND arrival=?");

selectFlight.setString(1, flight1.dateOfTravel);

selectFlight.setInt(2, flight1.seats);

selectFlight.setString(3, flight1.departure);

selectFlight.setString(4, flight1.arrival);

ResultSet flightResultSet = selectFlight.executeQuery();

if (flightResultSet.next()) {

flag = true;

int price1 = flightResultSet.getInt("e\_price");

fid = flightResultSet.getInt("f\_id");

System.out.println("Seats available");

System.out.println("The price of one ticket is " + price1);

cost = price1 \* flight1.seats;

System.out.println("The price for " + flight1.seats + " tickets is " + cost);

} else {

System.out.println("Seats not available");

}

} else if (flight1.type == 'b') {

PreparedStatement selectFlight = connection.prepareStatement("SELECT b\_price, f\_id FROM flights WHERE date= ? AND b\_seats >= ? AND depature=? AND arrival=?");

selectFlight.setString(1, flight1.dateOfTravel);

selectFlight.setInt(2, flight1.seats);

selectFlight.setString(3, flight1.departure);

selectFlight.setString(4, flight1.arrival);

ResultSet flightResultSet = selectFlight.executeQuery();

if (flightResultSet.next()) {

flag = true;

int price1 = flightResultSet.getInt("b\_price");

fid = flightResultSet.getInt("f\_id");

System.out.println("Seats available");

System.out.println("The price of one ticket is " + price1);

cost = price1 \* flight1.seats;

System.out.println("The price for " + flight1.seats + " tickets is " + cost);

} else {

System.out.println("Seats not available");

}

}

} catch (SQLException e) {

e.printStackTrace();

}

return flag;

}

}

class Airline {

Connection connection;

public Airline(Connection connection) {

this.connection = connection;

}

public void updateAirlineBank(int amount) {

try {

PreparedStatement updateAirlineBank = connection.prepareStatement("UPDATE airline\_bank set amt= amt+? where id=1");

updateAirlineBank.setInt(1, amount);

updateAirlineBank.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

}

}

public int getAirlineBank() {

try {

Statement stmt = connection.createStatement();

ResultSet rs = stmt.executeQuery("SELECT amt FROM airline\_bank WHERE id =1");

if (rs.next()) {

return rs.getInt("amt");

}

} catch (SQLException e) {

e.printStackTrace();

}

return 0;

}

public boolean hasEnoughFunds(int amount) {

return getAirlineBank() >= amount;

}

}

class PremiumCardHolder extends Customer {

Connection connection;

String cust\_name;

public PremiumCardHolder(Connection connection, String customerId) {

super(customerId);

this.connection = connection;

}

public boolean isPremium(String customerId) {

boolean flag = false;

try {

PreparedStatement checkPremium = connection.prepareStatement("SELECT name FROM p\_card WHERE c\_id=?");

checkPremium.setString(1, customerId);

ResultSet FlightResultSet2 = checkPremium.executeQuery();

if (FlightResultSet2.next()) {

flag = true;

cust\_name = FlightResultSet2.getString("name");

System.out.println("Welcome " + cust\_name);

} else {

System.out.println("Not a premium holder");

}

} catch (SQLException e) {

e.printStackTrace();

}

return flag;

}

}

public class AirlineApplication {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String c\_name;

String dp;

String ar;

String date;

char t;

int s;

int t\_price;

String cancel = "no";

char ch4;

try {

Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/java","root","Dee@2003");

FlightManager flightManager = new FlightManager(connection);

Airline airline = new Airline(connection);

do {

System.out.println("Hello!!");

System.out.println("1.Booking 2.cancellation");

System.out.println("Enter your choice");

int ch5 = sc.nextInt();

switch (ch5) {

case 1:

System.out.println("Enter Departure and Arrival city:");

dp = sc.next();

ar = sc.next();

FlightAvailable flight1 = new FlightAvailable(dp, ar);

if (flightManager.checkFlight(flight1) == true) {

System.out.println("Enter date of departure");

date = sc.next();

FlightDate fd = new FlightDate(dp, ar, date);

if (flightManager.getFlightByDate(fd) == true) {

System.out.println("Enter Economy(e) or Business(b) class");

t = sc.next().charAt(0);

System.out.println("Enter no of seats");

s = sc.nextInt();

FlightBook flight2 = new FlightBook(dp, ar, date, t, s);

if (flightManager.getFlightBySeats(flight2) == true) {

System.out.println("Do you want to book a flight Yes(y) or No(n)?");

char ch = sc.next().charAt(0);

if (ch == 'y') {

System.out.println("Are you a premium customer(y or n)?");

char ch1 = sc.next().charAt(0);

if (ch1 == 'y') {

System.out.println("Enter your Customer ID");

String id = sc.next();

PremiumCardHolder customer = new PremiumCardHolder(connection, id);

if (customer.isPremium(id) == true) {

c\_name = customer.cust\_name;

t\_price = flightManager.cost;

cancel = "premium";

} else {

System.out.println("Enter your name");

c\_name = sc.next();

System.out.println(c\_name);

System.out.println("Do you want to add cancellation policy(y or n)? An extra amount of 3000 will be added to your total price");

char ch2 = sc.next().charAt(0);

if (ch2 == 'y') {

t\_price = customer.CancellationPolicy(flightManager.cost);

cancel = "yes";

} else {

t\_price = flightManager.cost;

cancel = "no";

}

}

} else {

System.out.println("Enter your name");

c\_name = sc.next();

System.out.println("You want cancellation policy(y or n)");

char ch2 = sc.next().charAt(0);

if (ch2 == 'y') {

t\_price = flightManager.cost + 3000;

cancel = "yes";

} else {

t\_price = flightManager.cost;

cancel = "no";

}

}

Statement createTicketTable = connection.createStatement();

int c = 1;

createTicketTable.execute(

"CREATE table IF NOT EXISTS tickets(ticket\_id int UNIQUE,name varchar(30),departure varchar(20),arrival varchar(20),date varchar(10),seats int,type varchar(30),cancellation varchar(20),t\_price int,f\_id int)");

PreparedStatement insertTicket = connection.prepareStatement(

"INSERT INTO tickets VALUES(?,?,?,?,?,?,?,?,?,?)");

Random random = new Random();

int randomNumber = random.nextInt(100);

insertTicket.setInt(1, randomNumber);

insertTicket.setString(2, c\_name);

insertTicket.setString(3, flight2.departure);

insertTicket.setString(4, flight2.arrival);

insertTicket.setString(5, flight2.dateOfTravel);

insertTicket.setInt(6, flight2.seats);

char flightType = flight2.type;

insertTicket.setString(7, String.valueOf(flightType));

insertTicket.setString(8, cancel);

insertTicket.setInt(9, t\_price);

insertTicket.setInt(10, flightManager.fid);

insertTicket.executeUpdate();

if (flight2.type == 'e') {

PreparedStatement updateFlightSeats = connection.prepareStatement(

"UPDATE flights SET e\_seats = e\_seats - ? WHERE f\_id=? ");

updateFlightSeats.setInt(1, flight2.seats);

updateFlightSeats.setInt(2, flightManager.fid);

updateFlightSeats.executeUpdate();

} else if (flight2.type == 'b') {

PreparedStatement updateFlightSeats = connection.prepareStatement(

"UPDATE flights SET b\_seats = b\_seats - ? WHERE f\_id=? ");

updateFlightSeats.setInt(1, flight2.seats);

updateFlightSeats.setInt(2, flightManager.fid);

updateFlightSeats.executeUpdate();

}

airline.updateAirlineBank(t\_price);

System.out.println("Your ticket id is " + randomNumber);

System.out.println("Total ticket price:" + t\_price);

System.out.println("========================================================");

System.out.println("| AIRLINE TICKET |");

System.out.println("========================================================");

System.out.println("| Ticket ID: " + randomNumber + " |");

System.out.println("| Customer Name: " + c\_name + " |");

System.out.println("| Departure: " + flight2.departure + " |");

System.out.println("| Arrival: " + flight2.arrival + " |");

System.out.println("| Date of Travel: " + flight2.dateOfTravel + " |");

System.out.println("| Seats Booked: " + flight2.seats + " |");

System.out.println("| Flight Type: " + flightType + " |");

System.out.println("| Cancellation Policy: " + cancel + " |");

System.out.println("| Total Price: " + t\_price + " |");

System.out.println("========================================================");

}

}

}

}

break;

case 2:

System.out.println("Enter the ticketid:");

int t\_id = sc.nextInt();

PreparedStatement check\_Ticket = connection.prepareStatement(

"Select cancellation,t\_price from tickets where ticket\_id=?");

check\_Ticket.setInt(1, t\_id);

ResultSet finalset = check\_Ticket.executeQuery();

if (finalset.next()) {

String cancellation = finalset.getString("cancellation").trim();

int pricecancel = finalset.getInt("t\_price");

int totalpricecancel;

if ("yes".equalsIgnoreCase(cancellation)) {

totalpricecancel = pricecancel - 3000;

} else if ("premium".equalsIgnoreCase(cancellation)) {

totalpricecancel = pricecancel;

} else {

totalpricecancel = pricecancel \* 30 / 100;

}

if (airline.hasEnoughFunds(totalpricecancel)) {

airline.updateAirlineBank(-totalpricecancel);

System.out.println("Refund Granted");

PreparedStatement getDetails = connection.prepareStatement(

"Select seats,type,f\_id from tickets where ticket\_id = ?");

getDetails.setInt(1, t\_id);

ResultSet rs1 = getDetails.executeQuery();

if (rs1.next()) {

int s1 = rs1.getInt("seats");

char type1 = rs1.getString("type").charAt(0);

int id1 = rs1.getInt("f\_id");

if (type1 == 'e') {

PreparedStatement updateFlight = connection.prepareStatement(

"UPDATE flights SET e\_seats = e\_seats + ? WHERE f\_id=?");

updateFlight.setInt(1, s1);

updateFlight.setInt(2, id1);

updateFlight.executeUpdate();

} else if (type1 == 'b') {

PreparedStatement updateFlight = connection.prepareStatement(

"UPDATE flights SET b\_seats = b\_seats + ? WHERE f\_id=?");

updateFlight.setInt(1, s1);

updateFlight.setInt(2, id1);

updateFlight.executeUpdate();

}

}

PreparedStatement deleteTicket = connection.prepareStatement(

"DELETE FROM tickets WHERE ticket\_id=?");

deleteTicket.setInt(1, t\_id);

deleteTicket.executeUpdate();

} else {

System.out.println("Refund will be given shortly");

}

}

break;

default:

System.out.println("Enter a correct choice");

break;

}

System.out.println("Do you want to continue(y or n)");

ch4 = sc.next().charAt(0);

} while (ch4 == 'y');

System.out.println("Thank you");

} catch (SQLException e) {

e.printStackTrace();

}

}

}