CSCB07: Android Flight App

Special Instructions:
JUnit:

> Go to Project > Properties > Java Build Path > Libraries > Add Library > and select JUnit.

Serialization:
How to use:
A class needs to implement java.io.Serialization;
Create your object
To serialize (encode)

```
try {
  FileOutputStream fileOut = new FileOutputStream("/Users/Ciel/test.ser");
  ObjectOutputStream out = new ObjectOutputStream(fileOut);
  out.writeObject(YOUROBJECTHERE);
  out.close();
  fileOut.close();
} catch (IOException i) {
  i.printStackTrace();
}
```

Pass in your object **in out.writeObject** with the correct directory **in FileOutputStream**
To deserialize (decode)

```
OBJECTYPE object = null;
try {
   FileInputStream fileIn = new FileInputStream("/Users/Ciel/test.ser");
   ObjectInputStream in = new ObjectInputStream(fileIn);
   object = (Flight) in.readObject();
   in.close();
   fileIn.close();
}catch(IOException i) {
   i.printStackTrace();
   return;
}catch(ClassNotFoundException c) {
   System.out.println("Flight class not found");
   c.printStackTrace();
   return;
}
```

object would now be the deserialized object that you serialized, so if you serialized Flight flight1, if you deserialize it, you can now call object.getFlightNumber()

Feature List (Backend):

• A User can search available flights by entering a departure date, and travel origin and destination. A flight info should include: (1) flight number (alphanumeric), (2) departure date and time, (3) arrival date and time, (4) airline, (5) origin, (6) destination, (7) cost, and (8) travel time. The flight numbers should be unique. We will not deal with time zones, leap years, or any calendar or timing differences, to simplify your task.

• A User can search available itineraries by entering a departure date, and travel origin and destination. An itinerary should include, per flight: (1) flight number (alphanumeric), (2) departure date and time, (3) arrival date and time, (4) airline, (5) origin, and (6) destination, plus an overall itinerary cost and travel time. A valid itinerary contains no cycles, i.e. it does not visit the same place more than once.

• A User can view search results sorted by total travel time or by total cost.

• A User can view all stored client information, given the client's email address.

• A User can upload (see format below) personal and billing client information to be stored in the system.

• A User can upload (see format below) flights information into the system

Upload File Formats:

Clients:

LastName;FirstNames;Email;Address;CreditCardNumber;ExpiryDate

*where the expiry date is stored in the format yyyy-MM-dd.

Flights:

Number;DepartureDateTime;ArrivalDateTime;Airline;Origin;Destination;Price where the date and time is stored in the format yyyy-MM-dd HH:mm.

Layover:

• The time difference between one flight's arrival and the connecting flight's departure is at least Driver.MIN LAYOVER

• The time difference between one flight's arrival and the connecting flight's departure is at most Driver.MAX LAYOVER.

Class Design V 1.0:

Classes

+Account

Variables

- -email: String
- -firstName: String
- -lastName: String
- -password: String

Constructor

- +Account(e-mail: String, firstName: String, lastName: String, password: String)
- +searchFlight(departureDate: Date, arrivalDate: Date, isRoundTrip: boolean, startPoint: String, endPoint: String): resItinerary<E>
- +reserveFlight(Itinerary: Itinerary, Client client): void
- +displayReservations(): listOfItinerary
- +cancelReservation(Itinerary: Itinerary): void

+Client

Variables

- -age: int
- -address: String
- -e-mail: String
- -reservations: listOfItinerary
- -purchasedFlights: listOfPurschased

Constructor

- +Client(e-mail: String, firstName: String, lastName: String, password: String,
- address: String, age: int)

Methods

- +purchaseFlight(Itinerary: Itinerary): void
- +makePayment(): void

+Admin

Variables

Constructor

- +Admin(String: email, String: firstName, String: lastName, password: String)

Methods

- +uploadFlightInfo(flightFile: String): void
- +viewData(Client):
- +setClientFirstName(Client): void
- +setClientLastName(Client): void
- +setClientPassword(Client): void
- +setClientAge(Client): void
- +setClientEmail(Client): void
- +setClientBookings(Client): void

+Flight

Variables:

- -flightNumber: String
- -departureTime: Date
- -arrivalTime: Date
- -airline: String
- -origin: String
- -destination: String
- -travelCost: double
- -travelTime: Date

Constructor:

- +Flight(String flightNumber, Date departureTime, Date arrivalTime, Airline airline, String origin, String destination, double travelCost, Date travelTime)

Methods:

+Itinerary

Variables

- -flightList: ArrayList<Flight>

Constructors

- Itinerary(flightList: ArrayList<Flight>)

Methods

- buildCost():
- buildFlightTime():

+resItinerary

Variables

- -results: ArrayList<Itinerary>

Constructors

- resultItinerary(itineraryList: ArrayList<Itinerary>)

Methods

- sortByCost():
- sortByTime():

Helaina

+Airline

Variables

- -list of stops in order of flight time

Constructor

- +Airline()

Methods

- +getfirstDestination(): String
- +getlastDestination(): String
- +get stopovers(): List<Stops> locations
- +stopInAirline(Stop): boolean

+Stops

Variables

- -list of times of flights in order with their ids
- -String:location

Constructor

- +Stops(sheduleOfFlights)

Methods

- +getSoonestFlight(Date time):Flight