Transportation Company System

# Group participants:

* Mohamed Alaa eldin Abd Elnaby ,ID: 23101467
* Seif Eldean Ahmed Bassem , ID:23101399
* Ammar Yasser Mohamed , ID: 23101991

**Introduction:**

Our transportation company system project aims to develop a efficient software solution using Java to manage various aspects of a transportation company. The transportation system will facilitate the management of trips, vehicles, drivers, passengers, and managers within the company. It will provide functionalities for booking and managing trips, assigning drivers to trips, managing vehicles, handling passenger bookings, and generating reports for managerial insights.

Let’s start with explaining our UML diagram , we used “Lucid Chart” for the illustration.

**User Class**

The User class represents a user of the transportation system. It holds information about the user such as name, email, password, phone number, and username. Each user is assigned a unique identifier (uniqueID) generated using UUID.

Attributes:

* name: String - The full name of the user.
* email: String - The email address of the user.
* password: String - The password of the user.
* phoneNumber: String - The phone number of the user.
* userName: String - The username chosen by the user.
* uniqueID: UUID - The unique identifier of the user.
* userID: String - The protected user identifier.

Methods:

* User(userName: String, phoneNumber: String, name: String, email: String, password: String) - Constructor to create a new User object with the provided details.
* checkUserName(Database: String): boolean - Checks if the username is available in the specified database. Returns true if the username is available, false otherwise.
* saveData(): String - Saves user data to the database. Returns a confirmation message.
* saveData(d: Driver): String - Overloaded method to save driver data to the database. Returns a confirmation message.
* Register(path: String): void - Registers the user with the provided path to the database.
* Register(path: String, d: Driver): void - Overloaded method to register a driver with the provided path to the database.
* Login(Username: String, password: String, dataBase: String): boolean - Authenticates the user by comparing the provided username and password with the data in the specified database. Returns true if the authentication is successful, false otherwise.

Note:

* The userID attribute is protected to control access to it within the class hierarchy.
* The methods saveData() and Register() are overloaded to handle different scenarios such as saving user data and driver data, and registering users and drivers respectively.
* The Login() method authenticates the user by checking the provided username and password against the database.

**Passenger Class**

The Passenger class represents a passenger who uses the transportation system. It contains details about the passenger such as name, phone number, username, email, and password. Passengers can book trips, view trip details, cancel trips, review tickets, and manage their profiles.

Attributes:

* passengerTrips: ArrayList<Trip> - The list of trips booked by the passenger.
* allPassengers: ArrayList<Passenger> - The list of all passengers.
* deletedTrips: boolean - Indicates whether a trip has been deleted or not. Initialized to false by default.

Constructor:

Passenger(name: String, phoneNumber: String, userName: String, email: String, password: String) - Creates a new Passenger object with the provided details.

Methods:

* bookTicket(trip: Trip): void - Books a ticket for the passenger on the provided trip.
* isBooked(ID: String): boolean - Checks if the passenger has booked a trip with the specified ID. Returns true if booked, false otherwise.
* refund(): String - Initiates a refund process for the passenger. Returns a message confirming the refund.
* cancelTrip(cancelTrip: Trip): String - Cancels the provided trip for the passenger. Returns a message confirming the cancellation.
* tripDetails(trip: Trip): String - Retrieves and displays details of the provided trip for the passenger.
* reviewTickets(): String - Displays a summary of all tickets booked by the passenger.
* displayProfile(): void - Displays the profile details of the passenger.
* overWritePassengerDataBase(): void - Overwrites the passenger database with updated information.
* loadPassengersData(): void - Loads passenger data from the database.
* isTripInDataBase(tripID: String): boolean - Checks if the provided trip ID exists in the database. Returns true if found, false otherwise.
* searchTrip(tripID: String): Trip - Searches for a trip with the provided ID and returns it.
* LoginPassengerData(username: String): Passenger - Retrieves passenger data based on the provided username and returns the Passenger object.

Note:

* The passengerTrips attribute holds the list of trips booked by the passenger.
* Methods like refund(), cancelTrip(), tripDetails(), reviewTickets() provide functionality related to managing booked trips.
* displayProfile() allows passengers to view their profile information.
* Methods like overWritePassengerDataBase() and loadPassengersData() are for managing the passenger database.
* isTripInDataBase() and searchTrip() assist in searching for and managing trip data.
* LoginPassengerData() is used to retrieve passenger data for login purposes based on the username.

**Employee Class**

The Employee class represents an employee of the transportation company. Employees can be managers or drivers. It contains details about the employee such as username, phone number, name, email, and password.

Attributes:

* allManagers: ArrayList<Manager> - The list of all managers in the company.
* allDrivers: ArrayList<Driver> - The list of all drivers in the company.

Constructor:

Employee(userName: String, phoneNumber: String, name: String, email: String, password: String) - Creates a new Employee object with the provided details.

Methods:

None specific to the Employee class. Employees may have specific methods based on their role (e.g., Manager or Driver).

Final notes:

* The Employee class serves as a superclass for managers and drivers.
* The attributes allManagers and allDrivers are likely to be used for managing employee data within the company.
* Further functionality and methods specific to managers and drivers will be implemented in their respective subclasses.

**Manager Class**

The Manager class represents a manager within the transportation company. Managers have various responsibilities including managing trips, vehicles, and drivers. They can also generate reports and perform administrative tasks.

Constructor:

Manager(userName: String, phoneNumber: String, name: String, email: String, password: String) - Creates a new Manager object with the provided details.

Methods:

* managerAddTrip(journeyType: String, source: String, destination: String, year: int, month: int, day: int, hour: int, min: int, noOfStops: int, licensePlate: String, tripType: String): String - Adds a new trip to the system with the specified details.
* addVehicle(vehicleType: String, capacity: int, licensePlate: String): String - Adds a new vehicle to the system with the specified details.
* displayVehicles(): String - Displays the list of vehicles available in the system.
* displayFreeDrivers(): String - Displays the list of drivers who are currently available for assignments.
* assignDriverToTrip(d: Driver, t: Trip): String - Assigns a driver to a trip.
* isTripAssigned(tripID: String): boolean - Checks if a trip is already assigned.
* isDriverAssigned(d: Driver , tripID: String): boolean - Checks if a driver is already assigned to a trip.
* cancelTrip(trip: Trip): String - Cancels the specified trip.
* GenerateReport(): String - Generates a report on system activities or performance.
* displayDrivers(): String - Displays the list of drivers in the system.
* displayManagerInfo(): String - Displays information about the manager.
* overWriteTripsDataBase(): void - Overwrites the trip database with updated information.
* overWriteManagerDataBase(): void - Overwrites the manager database with updated information.
* loadManagersData(): ArrayList<Manager> - Loads manager data from the database and returns a list of managers.
* LoginManagerData(username: String): Manager - Retrieves manager data for login purposes based on the username and returns the Manager object.

**Driver Class**

The Driver class represents a driver within the transportation company. Drivers are responsible for completing assigned trips and maintaining information about their assigned trips and vehicles.

Attributes:

* assignedTrips: ArrayList<Trip> - The list of trips assigned to the driver.
* MAX\_TRIPS: int - The maximum number of trips a driver can be assigned, initialized to 2 by default.
* driverVehicle: String - The vehicle assigned to the driver.

Constructor:

Driver(userName: String, phoneNumber: String, name: String, email: String, password: String) - Creates a new Driver object with the provided details.

Methods:

* reviewAssignedTrips(): String - Displays information about the trips assigned to the driver.
* displayDriverInfo(): String - Displays information about the driver.
* loadDriversData(): ArrayList<Driver> - Loads driver data from the database and returns a list of drivers.
* overWriteDriverDataBase(): void - Overwrites the driver database with updated information.
* LoginDriverData(username: String): Driver - Retrieves driver data for login purposes based on the username and returns the Driver object.

Note:

The Driver class is responsible for managing the assigned trips and driver information.

**Class Vehicle**

The Vehicle class represents a vehicle within the transportation company. Vehicles have attributes such as vehicle type, capacity, and license plate.

Attributes:

* vehicleType: String - The type of the vehicle.
* capacity: int - The maximum capacity of the vehicle.
* licensePlate: String - The license plate number of the vehicle.

Constructor:

Vehicle(vehicleType: String, capacity: int, licensePlate: String) - Creates a new Vehicle object with the provided details.

Methods:

* displayInfo(): String - Displays information about the vehicle.
* preprocessingData(): String - Performs any necessary preprocessing tasks related to the vehicle.
* overWriteVehicleDataBase(): void - Overwrites the vehicle database with updated information.
* loadVehicleData(): ArrayList<Vehicle> - Loads vehicle data from the database and returns a list of vehicles.

Note:

The Vehicle class holds information about vehicles used in the transportation system.

**Trip Class**

The Trip class represents a trip within the transportation system. It contains information about the trip such as journey type, trip type, price, source, destination, number of stops, vehicle, date, and available seats.

Attributes:

* journeyType: String - The type of journey (e.g., one-way, round trip).
* tripType: String - The type of trip (e.g.,internal, external).
* price: double - The price of the trip.
* source: String - The starting point of the trip.
* destination: String - The destination of the trip.
* noOfStops: int - The number of stops during the trip.
* vehicle: Vehicle - The vehicle assigned to the trip.
* availableSeats: int - The number of available seats on the trip.
* date: Date - The date of the trip.
* MAX\_STOPS: int - The maximum number of stops allowed, initialized to 3 by default.
* uniqueID: UUID - The unique identifier of the trip.
* ID: String - The identifier of the trip.
* NoOfTickets: int - The number of tickets booked for the trip.
* totalPrice: double - The total price of all tickets booked for the trip.
* allTrips: ArrayList<Trip> - The list of all trips.

Constructor:

Trip(journeyType: String, tripType: String, price: double, source: String, destination: String, noOfStops: int, vehicle: Vehicle, date: Date) - Creates a new Trip object with the provided details.

Methods:

* bookSeat(): String - Books a seat for the trip. Returns a confirmation message.
* cancelSeat(): void - Cancels a booked seat for the trip.
* preprocessingData(): String - Performs any necessary preprocessing tasks related to the trip.
* preprocessingData\_overWrite(): String - Performs preprocessing tasks and overwrites data related to the trip.
* overWriteTripDataBase(): void - Overwrites the trip database with updated information.
* addToDataBase(): void - Adds the trip to the database.
* loadTripsData(): ArrayList<Trip> - Loads trip data from the database and returns a list of trips.

Note:

* The Trip class holds information about trips within the transportation system.
* The attributes uniqueID, ID, NoOfTickets, totalPrice, and allTrips are likely used for managing trip data.

**Members Contributions**

The project was divided among three team members. Each member took responsibility for specific classes within the system.

**Mohamed's Contributions:**

I primarily worked on the following classes:

* Employee: This class encapsulates the properties and methods related to company employees within the system.
* Vehicle: I developed the Vehicle class, which represents the vehicles initialized by the transportation company, including attributes and behaviors associated with them.
* I worked on the following classes in JavaFX: ManagerReviewTripController , ManagerReportVehicleController , ManagerReportTripsController.

**Seif's Contributions:**

Seif focused on implementing the following classes:

* Trip: Seif developed the Trip class, responsible for managing information related to trips, such as destination, departure time, and route details.
* Passenger and Driver: Seif handled the Passenger and Driver classes, which manage passenger and driver information respectively, including their roles and responsibilities within the system.
* Seif worked on the following JavaFX classes: DriverTripsController , ManagerReportDriversController , VehicleCardController , PassengerExploreController.

## **Ammar's Contributions:**

Ammar implemented the final classes within the system:

* Manager: The Manager class, which incorporates functionalities related to managerial roles within the transportation company system.
* User: Ammar implemented the User class, responsible for managing user interactions and authentication processes within the system.
* Ammar also handled the css file sheets using javaFX library.
* Ammar worked on the following classes in JavaFX: ManagerController , MainMenu , PassengerController , DriverController , ManagerSelectedTripController , PassengerOwnTripsController , ManagerFreeDriversController .

Additionally, The three of us created the UML diagram illustrating the class relationships within the system.

**Relationships:**

*Inheritance:*

* Driver inherits from Employee.
* Passenger inherits from User
* Employee inherits from User
* Manager Inherits from Employee

*Association:*

* Driver has an composition with Trip.
* Passenger has an aggregation in Trip. “passengerTrips”
* Employee has an aggregation with Manager and Driver. “allManagers” , “allDrivers”
* Manager has an aggregation with Trip , Vehicle and Driver.
* Trip has an aggregation with Vehicle in “vehicle” attribute.