

JAVA PROGRAMMING COURSE PROJECT

CAB BOOKING SYSTEM CABBER



S.NO	CONTENT
1.	PROBLEM DEFINITION
2.	OBJECTIVES
3.	LIST OF CLASSES
4.	DESIGN PATTERN
5.	CLASS DIAGRAM
6.	DESCRIPTION OF EACH CLASS DIAGRAM
7.	LIST OF DATA STRUCTURES
8.	USER INTERFACE SCREENSHOTS

1. PROBLEM DEFINITION

Most of the Big Players in the Cab Booking Industry have raised a bar by extremely User-Friendly Applications which have been very popular in grabbing interest among customers as they consider it to be Hassle-Free and something which happens extremely Handy on the go. The main aim of our application Cabber is to raise voice and go vocal for local, where cab-agencies across the city put the best services and rates forward to make the customer journey more comfortable and price -Friendly and compete with dominant startups which have dried down the business.

2. OBJETIVES

- Customers can sign up to the application and book rides
- The customer can book a ride types of car available according to their connivence and get the estimated fare for the ride based on the distance and type of car chosen.
- Drivers can sign up and transport Customers based on the ride booked.
- Customer can access their previous ride.
- Customer will be provided with details of driver and cab for the security reasons.
- If a driver is already assigned to a ride, then the subsequent rides will be assigned to other drivers

3. LIST OF CLASSES

1.Cabber

2.RideBookerApp

3. RideLog

4. Driver

5. Customer

6. Vehicle

6.1 SUV

6.2 mini

6.3 Sedan

4. DESIGN PATTERN

We have identified two design patterns for our project which could be used

1.Singleton

2. Factory

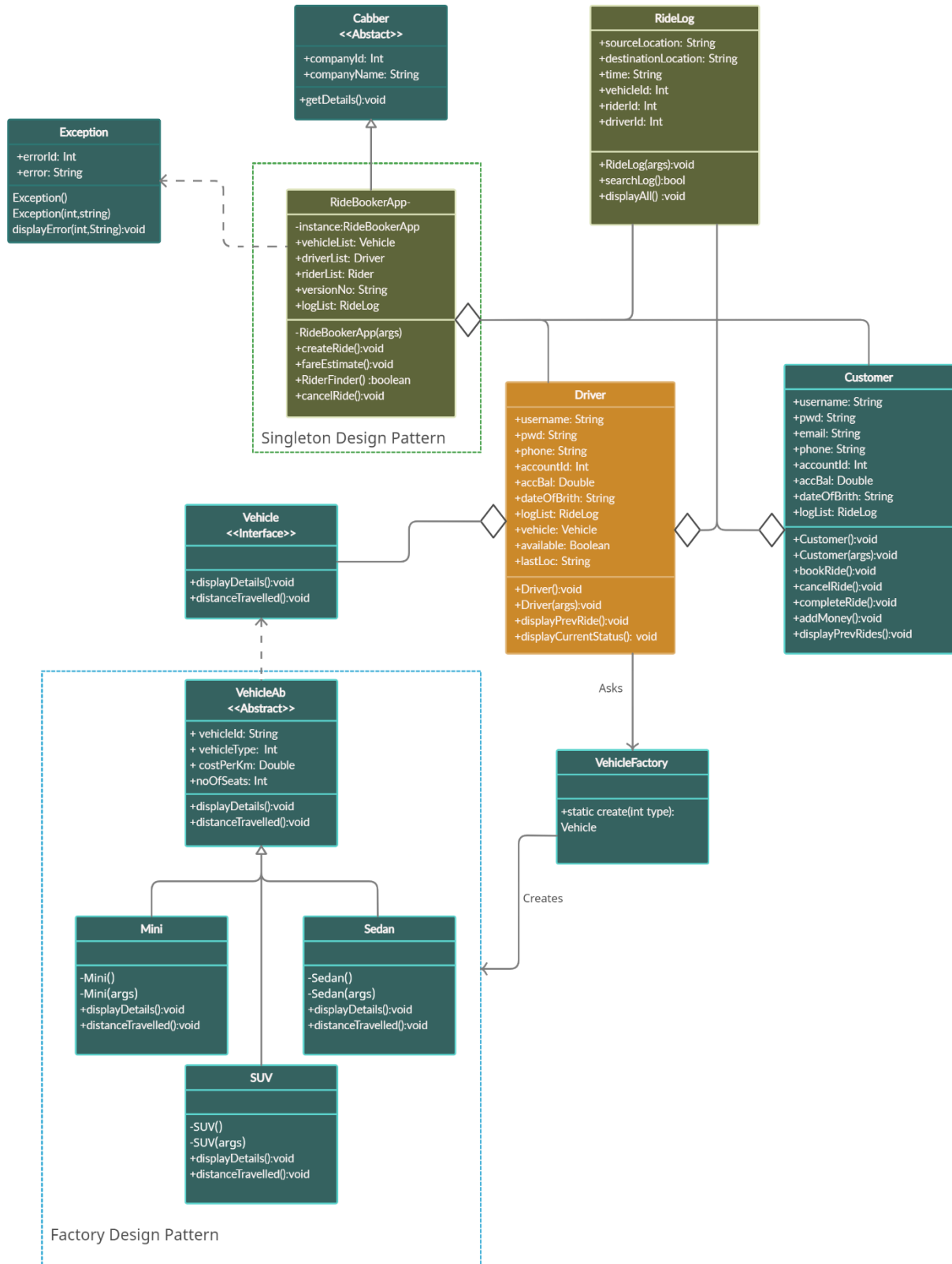
1.Singleton: Single class which is responsible to create an object while making sure that only a single object gets created. This class provides a way to access its only object which can be accessed directly without the need to instantiate the object of the class.

Here our single class will be the abstract class Cabber because one object i.e. agency is created so as to use the application to the fullest multiple agencies cannot

2. Factory Pattern: The factory design pattern is used when we have a superclass with multiple sub-subclass and based on input, we need to return one of the sub-class. This pattern takes out the responsibility of the instantiation of a class from program to the factory class

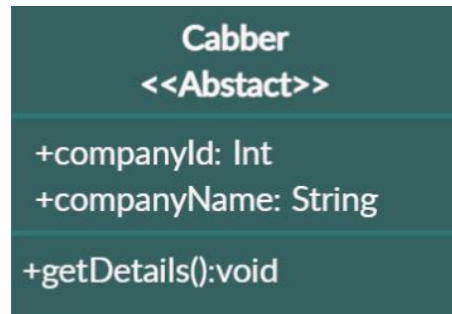
Superclass will be the class Vehicle which has subclass as SUV, mini, sedan as its subclass can be instantiated based on the user's input.

5. CLASS DIAGRAM



6. DESCRIPTION OF EACH CLASS DIAGRAM

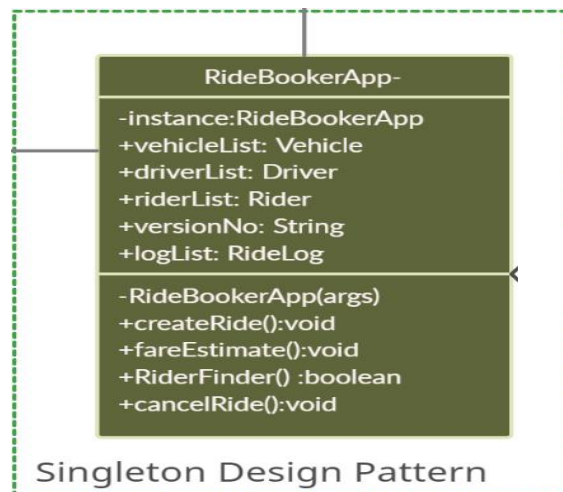
1. Cabber



This class will be the abstract class where the admin i.e., the company/agency will log in their details so as to reach out to the customers to provide their services.

- **+companyId: Int** company will be provided with the company id which is the unique id of integer type
- **+comapanyName: String** Name of the company
- **+getDetails (): void**

2. RideBookerApp

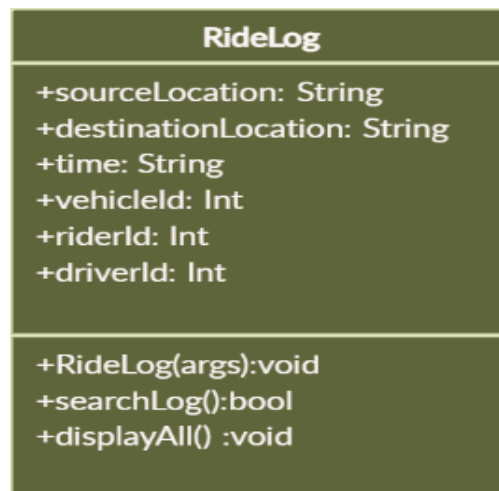


Class RideBookerApp is class Cabber

The class have mains 5 attributes and 4 function

- **+VehicleList** will consist of a list of vehicles in the particular company
- **+DriverList:** will consist of a list of drivers driving the cars of the company
- **+RiderList** will consist of the rider/customer List who took the ride on the cars of the company /agency
- **+versionNo: String**
- **+LogList: RideLog**
- **RideBookerApp (): void**
- **+createRide ():** this function will be called when customers book a ride on the application
- **+fareEstimation ()** function will calculate the fare of the ride based on the riders preferred car and the distance the rider is traveling
- **+RiderFinder ()** function will find the rider available for the particular ride chosen by the customer to the respective Pickup to destination Point
- **+CancelRide ()** Function will let the rider cancel his/her ride at any given instance of time

3. RideLog



RideBookerApp has Class RideLog

This class will have all the logs /record of the source and Destination Location of the Ride along the Time
vehicleId which took the Ride RiderId of the Rider also the DriverId of the Driver of vehicle

- **+sourceLocation: String** This is the location from where is Rider has been picked up for the Ride
- **+destinationLocation: String** This is the location from where is Rider has been Dropped
- **+vehicleId: Int:** Id of the vehicle which took the ride from the source to destination
- **+riderId: Int** id of the rider who booked the cab. The rider i.e, the customer who has registered in the application to use the provided service will have a unique rider id
- **driverId: Int** to fetch the driver details
- **+RideLog ()**
- **+searchLog ():** this function will help to search the rides taken by the Customer and the driver
- **+displayAll ():** will display the details of the Log

4. Driver

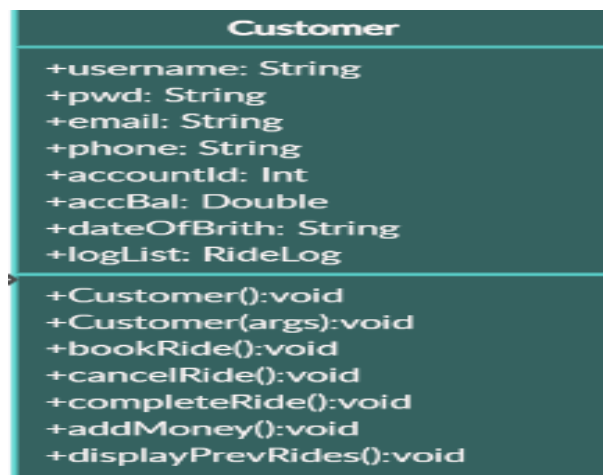
Driver
+username: String +pwd: String +phone: String +accountId: Int +accBal: Double +dateOfBrith: String +logList: RideLog +vehicle: Vehicle +available: Boolean +lastLoc: String
+Driver():void +Driver(args):void +displayPrevRide():void +displayCurrentStatus(): void

Class RideLog has Class Driver and Class RideBookingApp has Class Driver
 Class Driver has Class vehicle. This has all the details of the driver (here we are considering every Driver has a vehicle allotted by the agency)

- **+username: String:** this the Username by which the Driver/employee can log in to the application

- **+pwd: String:** password for the security purpose of their respective accounts
- **+dateofBirth:** this is to store date of birth
- **+phone: String** Phone Number of the Driver
- **+accountId: Double** Account Id associated to the driver
- **+accBal: Double** to store acc balance
- **+vehicle:** Vehicle fetch the details of the vehicle the driver is assigned with.
- **+available: Boolean** to check the status if he free or on ride
- **+lastLoc: String** to find the last location of the driver
- **+logList: RideLog:** To store the rides the particular Driver has taken.
- **+Driver (): void**
- **+Driver(args): void**
- **+displayPrevRide (): void:** this function will display all the previous rides the particular driver has taken from the Ridelog
- **+displayCurrentStatus (): void** Function used to display Current status of the driver to see if he is available or not

5. Customer

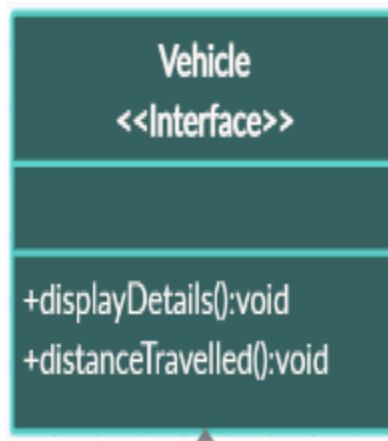


Class RideLog has Class Customer and Class RideBookingApp has Class Customer

- **+ userName: String** this the Username by which the customer can log in to the application

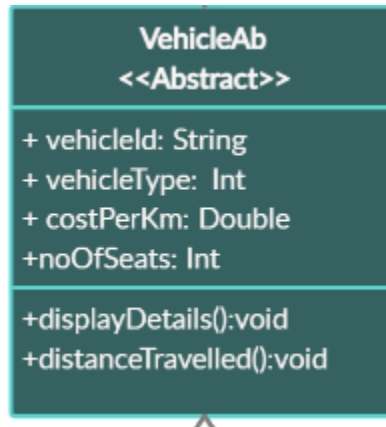
- **+Pwd: String** to store the password so that the customer can login using his/her password and username and password for the security purpose of their respective accounts.
- **+Email: String** to store the email id of the customer
- **+phone: String** Phone Number of the Customers
- **+accountId: Double** Account Id associated to the customer
- **+accBal: Double** to store acc balance
- **+LogList: RideLog**
- **+Customer (): Void**
- **+BookRide ():** function to Book the Ride
- **+cancelRide () Customer/rider** have the option of canceling the rides under valid circumstances.
- **+CompleteRide ():** Since we can't actually move the marker, we simulate the end of ride by this function
- **+AddMoney ():** To add money in the app wallet.
- **+DsplayPreviousRide (): void** This will display all the Previous Ride from the LogRide the customer has taken so far through this application (History)

6. Vehicle



- Interface that helps achieve total abstraction to vehicle class.
- **displayDetails (): Void** Function to display the vehicle details
- **distanceTravelled (): Void** Function to display distance travelled till date.

7. VehicleAb



Class VehicleAb implements interface Vehicle

- **+vehicleId: String:** every vehicle in the agency will have the unique Id
- **+VehicleType:int:** Type of vehicle (three categories)
- **+CostPerKm: Double:** Cost per Km it varies to different vehicle depending of the Type of the vehicle
- **+noOfSeats: Int:** Denotes maximum number of people that can be accommodated
- **+displayDetails (): void** display all of details of the vehicle
- **+distanceTravelled (): void** to have count of the distance travelled by the particular vehicle

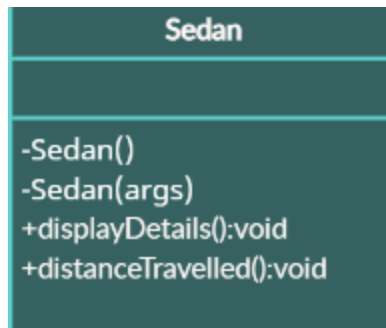
8. Mini:



Class Mini is VehicleAb

- **displayDetails (): void** Function to display the vehicle details
- **distanceTravelled (): void** Function to display distance travelled till date

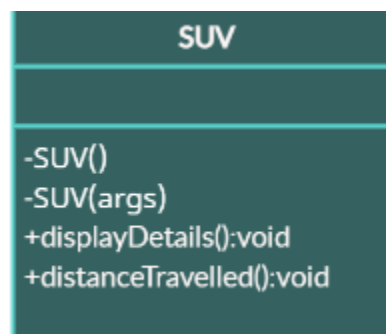
9. Sedan



Class Sedan is Vehicle

- **displayDetails (): void** Function to display the vehicle details
- **distanceTravelled (): void** Function to display distance travelled till date

10. SUV



Class SUV is Vehicle

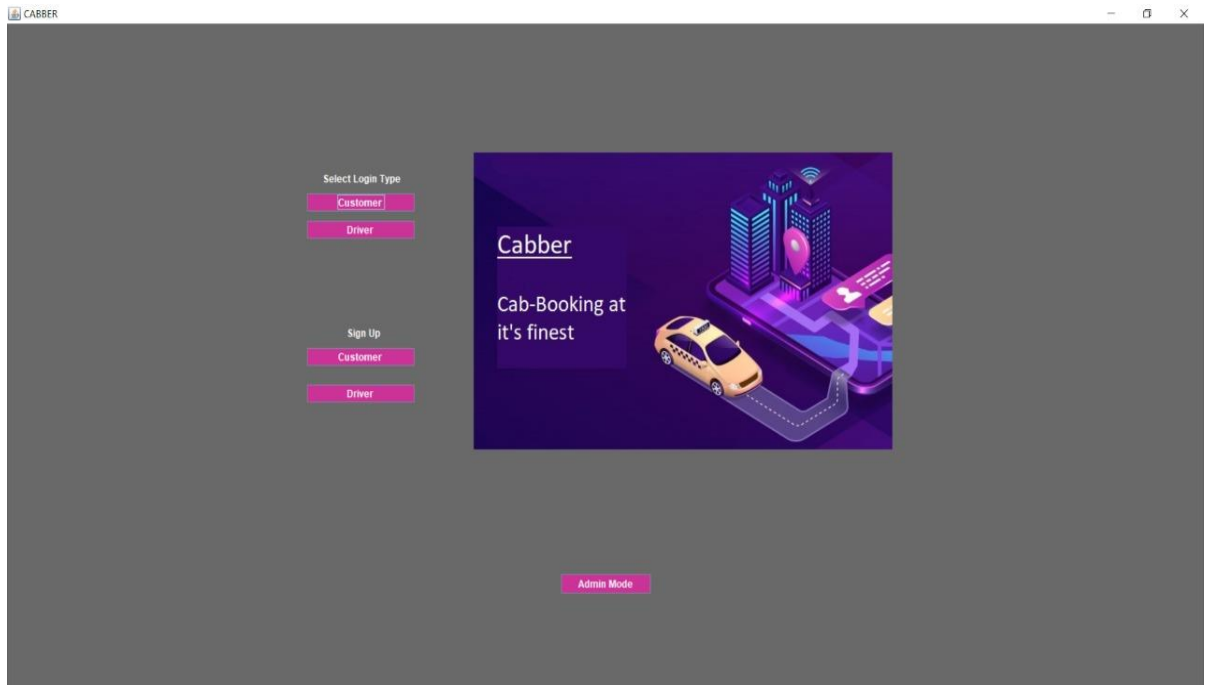
- **displayDetails (): void** Function to display the vehicle details
- **distanceTravelled (): void** Function to display distance travelled till date

7. DATA STRUCTURES:

1. ArrayList: To store multiple objects of Drivers and Customers in RideBookerApp we use Array Lists.
2. HashSet: To ensure no username is repeated while creation of users be that Driver or Customer, we create a static variable in RideBookerApp that keeps all the available usernames and makes sure only if username entered is not present then user will be created

8. USER INTERFACE SCREENSHOTS

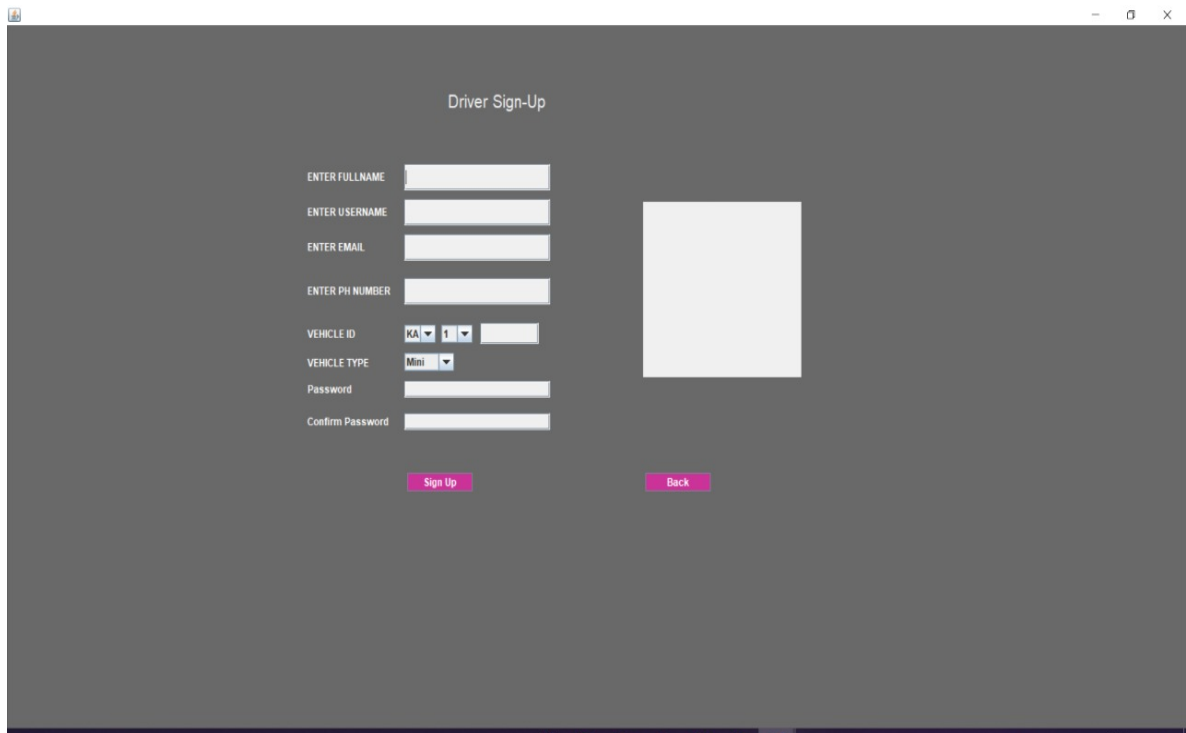
1) Main page Login/Signup



2) Customer Sign-up

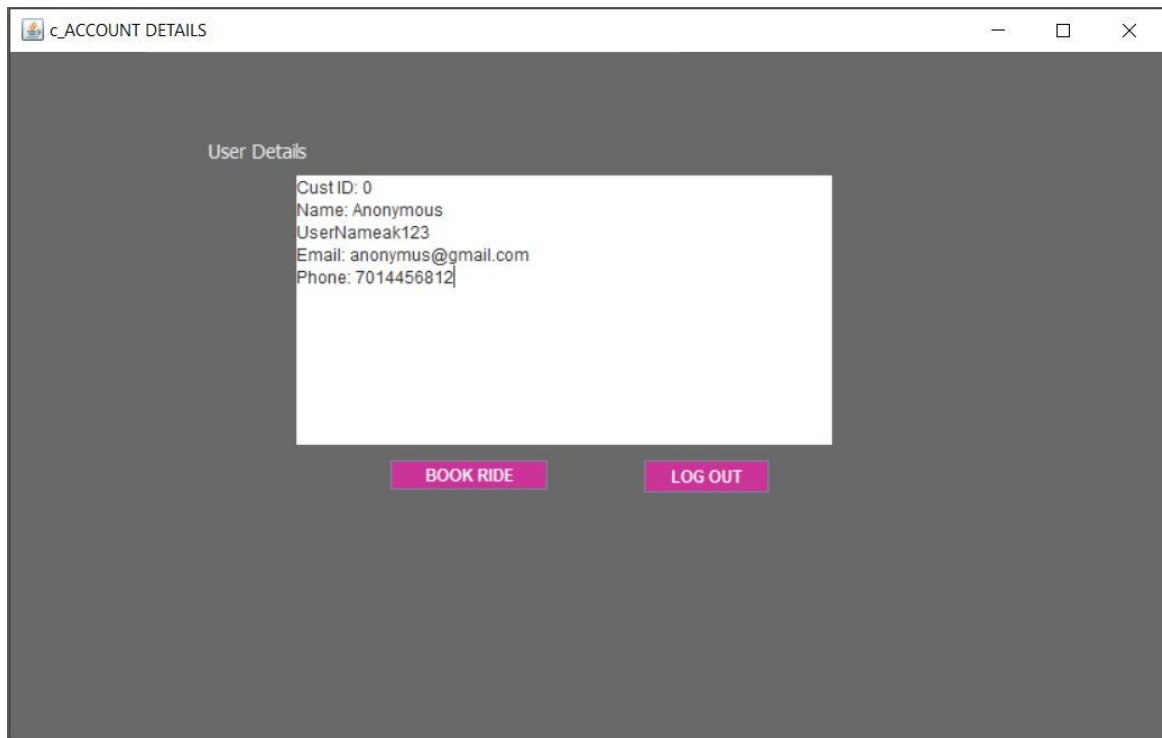
The screenshot shows the 'CUSTOMER SignUp' window. The title bar reads 'CUSTOMER SignUp'. The form is titled 'Customer Login'. It contains the following input fields: 'Enter FullName', 'Enter UserName', 'Enter Email', 'Enter Ph Number', 'Enter Password', and 'Confirm Password'. Each field is represented by a white text box. To the right of these fields is a large, empty white rectangular area. At the bottom of the form, there are two buttons: 'Back' and 'Sign-Up'.

3) Driver Signup



A screenshot of a web application window titled "Driver Sign-Up". The form is set against a dark gray background. It contains several input fields for registration: "ENTER FULLNAME", "ENTER USERNAME", "ENTER EMAIL", "ENTER PH NUMBER", "VEHICLE ID" (with a dropdown menu showing "KA" and "1"), "VEHICLE TYPE" (with a dropdown menu showing "Mini"), "Password", and "Confirm Password". To the right of these fields is a large, empty square box, likely for a profile picture. At the bottom of the form are two pink buttons: "Sign Up" and "Back".

4) Customer Account Details



A screenshot of a web application window titled "c_ACCOUNT DETAILS". The page has a dark gray background. In the center, there is a white rectangular box containing the following text: "Cust ID: 0", "Name: Anonymous", "UserName: ak123", "Email: anonymus@gmail.com", and "Phone: 7014456812". Below this box are two pink buttons: "BOOK RIDE" and "LOG OUT".

5) Customer Book Ride

The screenshot shows a web application window titled "BOOK RIDE". At the top right, there is a "Back" button. Below it, a navigation bar contains three tabs: "BOOK RIDE", "PREVIOUS RIDE", and "CANCEL RIDE". The main content area is a form titled "Ride Details". The form contains the following elements:

- Two dropdown menus for "SOUR..." and "DESTINATI...", both currently set to "HUBLI".
- A "Select" label followed by a dropdown menu set to "Mini".
- A pink "ESTIMATE" button.
- Four input fields labeled "DISTANCE", "DRIVER ALLOTTED", "VEHICLE ID", and "ESTIMATED FA...".
- A purple "CONFIRM BOOKING" button.

At the bottom of the form, there are two pink buttons: "Add Money" and "Account Details".

6) Rider Log

The screenshot shows a web application window titled "Rider Log". At the top right, there is a "Return" button. Below it, a navigation bar contains three tabs: "Customers", "Drivers", and "RideLog". The main content area is a form titled "Enter Custome..." with a text input field. Below the input field, there are two pink buttons: "Display Details" and "Display All Customers". The form is currently empty, showing a large white rectangular area.