CAR CONNECT

Create following tables in SQL Schema with appropriate class and write the unit test case for the application.

Creating a database

```
create database carconnect2;
use carconnect2;
```

SQL Tables:

1. Customer Table:

• CustomerID (Primary Key): Unique identifier for each customer.
• FirstName: First name of the custome.
• LastName: Last name of the customer.
• Email: Email address of the customer for communication.
• PhoneNumber: Contact number of the customer.
• Address: Customer's residential address.
• Username: Unique username for customer login.
• Password: Securely hashed password for customer authentication.
• RegistrationDate: Date when the customer registered.

```
CREATE TABLE Customer (
 CustomerID INT AUTO_INCREMENT PRIMARY KEY,
 FirstName VARCHAR(50),
 LastName VARCHAR(50),
 Email VARCHAR(100),
 PhoneNumber VARCHAR(15),
 Address VARCHAR(255),
 Username VARCHAR(50) UNIQUE,
 Password VARCHAR(255),
 RegistrationDate DATE
);
```

2. Vehicle Table:

• VehicleID (Primary Key): Unique identifier for each vehicle.
• Model: Model of the vehicle.
• Make: Manufacturer or brand of the vehicle.
• Year: Manufacturing year of the vehicle.
• Color: Color of the vehicle.

• RegistrationNumber: Unique registration number for each vehicle.
• Availability: Boolean indicating whether the vehicle is available for rent.
• DailyRate: Daily rental rate for the vehicle.

```
CREATE TABLE Vehicle (
 VehicleID INT AUTO_INCREMENT PRIMARY KEY,
 Model VARCHAR(50),
 Make VARCHAR(50),
 Year INT,
 Color VARCHAR(50),
 RegistrationNumber VARCHAR(20) UNIQUE,
 Availability BOOLEAN,
 DailyRate DECIMAL(10, 2)
);
```

3. Reservation Table:
• ReservationID (Primary Key): Unique identifier for each reservation.
• CustomerID (Foreign Key): Foreign key referencing the Customer table.
• VehicleID (Foreign Key): Foreign key referencing the Vehicle table.
• StartDate: Date and time of the reservation start.
• EndDate: Date and time of the reservation end.
• TotalCost: Total cost of the reservation.
• Status: Current status of the reservation (e.g., pending, confirmed, completed).

```
CREATE TABLE Reservation (
 ReservationID INT AUTO_INCREMENT PRIMARY KEY,
 CustomerID INT,
 VehicleID INT,
 StartDate DATETIME,
 EndDate DATETIME,
 TotalCost DECIMAL(10, 2),
 Status ENUM('pending', 'confirmed', 'completed'),
 FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),
 FOREIGN KEY (VehicleID) REFERENCES Vehicle(VehicleID)
);
```

4. Admin Table:
• AdminID (Primary Key): Unique identifier for each admin.
• FirstName: First name of the admin.

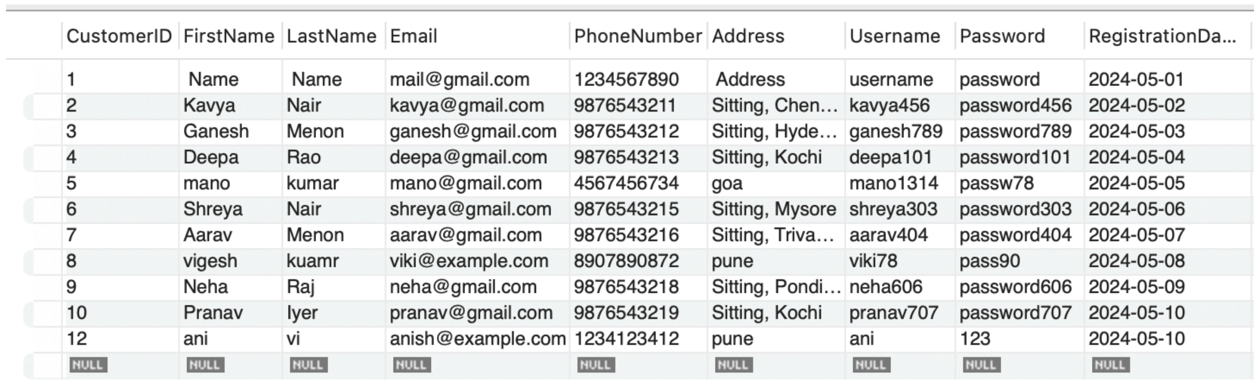
• LastName: Last name of the admin.
• Email: Email address of the admin for communication.
• PhoneNumber: Contact number of the admin.
• Username: Unique username for admin login.
• Password: Securely hashed password for admin authentication.
• Role: Role of the admin within the system (e.g., super admin, fleet manager).
• JoinDate: Date when the admin joined the system.

```
CREATE TABLE Admin (
 AdminID INT AUTO_INCREMENT PRIMARY KEY,
 FirstName VARCHAR(50),
 LastName VARCHAR(50),
 Email VARCHAR(100),
 PhoneNumber VARCHAR(15),
 Username VARCHAR(50) UNIQUE,
 Password VARCHAR(255),
 Role VARCHAR(20),
 JoinDate DATE
);
```

Inserting Data into tables:

Customer Table:

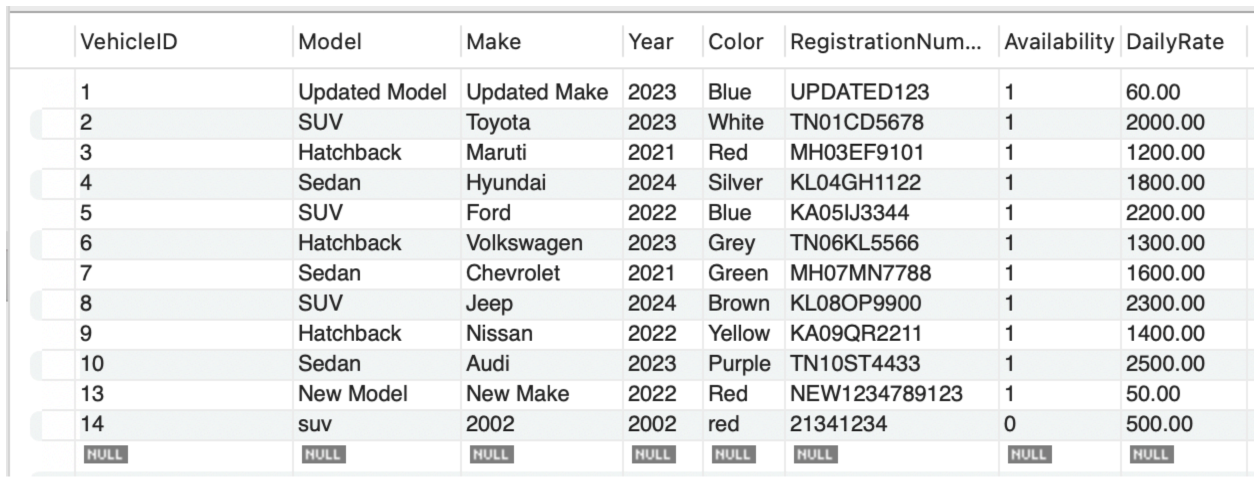
```
INSERT INTO Customer (FirstName, LastName, Email, PhoneNumber, Address, Username,
Password, RegistrationDate)
VALUES
('Arjun', 'Kumar', 'arjun@gmail.com', '9876543210', 'Sitting, Bangalore', 'arjun123',
'password123', '2024-05-01'),
('Kavya', 'Nair', 'kavya@gmail.com', '9876543211', 'Sitting, Chennai', 'kavya456', 'password456',
'2024-05-02'),
('Ganesh', 'Menon', 'ganesh@gmail.com', '9876543212', 'Sitting, Hyderabad', 'ganesh789',
'password789', '2024-05-03'),
('Deepa', 'Rao', 'deepa@gmail.com', '9876543213', 'Sitting, Kochi', 'deepa101', 'password101',
'2024-05-04'),
('Vishnu', 'Iyer', 'vishnu@gmail.com', '9876543214', 'Sitting, Coimbatore', 'vishnu202',
'password202', '2024-05-05'),
('Shreya', 'Nair', 'shreya@gmail.com', '9876543215', 'Sitting, Mysore', 'shreya303',
'password303', '2024-05-06'),
('Aarav', 'Menon', 'aarav@gmail.com', '9876543216', 'Sitting, Trivandrum', 'aarav404',
'password404', '2024-05-07'),
('Ananya', 'Sharma', 'ananya@gmail.com', '9876543217', 'Sitting, Mangalore', 'ananya505',
'password505', '2024-05-08'),
('Neha', 'Raj', 'neha@gmail.com', '9876543218', 'Sitting, Pondicherry', 'neha606', 'password606',
'2024-05-09'),
('Pranav', 'Iyer', 'pranav@gmail.com', '9876543219', 'Sitting, Kochi', 'pranav707', 'password707',
'2024-05-10');
```



CustomerID	FirstName	LastName	Email	PhoneNumber	Address	Username	Password	RegistrationDa…
1	Name	Name	mail@gmail.com	1234567890	Address	username	password	2024-05-01
2	Kavya	Nair	kavya@gmail.com	9876543211	Sitting, Chen…	kavya456	password456	2024-05-02
3	Ganesh	Menon	ganesh@gmail.com	9876543212	Sitting, Hyde…	ganesh789	password789	2024-05-03
4	Deepa	Rao	deepa@gmail.com	9876543213	Sitting, Kochi	deepa101	password101	2024-05-04
5	mano	kumar	mano@gmail.com	4567456734	goa	mano1314	passw78	2024-05-05
6	Shreya	Nair	shreya@gmail.com	9876543215	Sitting, Mysore	shreya303	password303	2024-05-06
7	Aarav	Menon	aarav@gmail.com	9876543216	Sitting, Triva…	aarav404	password404	2024-05-07
8	vigesh	kuamr	viki@example.com	8907890872	pune	viki78	pass90	2024-05-08
9	Neha	Raj	neha@gmail.com	9876543218	Sitting, Pondi…	neha606	password606	2024-05-09
10	Pranav	Iyer	pranav@gmail.com	9876543219	Sitting, Kochi	pranav707	password707	2024-05-10
12	ani	vi	anish@example.com	1234123412	pune	ani	123	2024-05-10
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Vehicle Table:

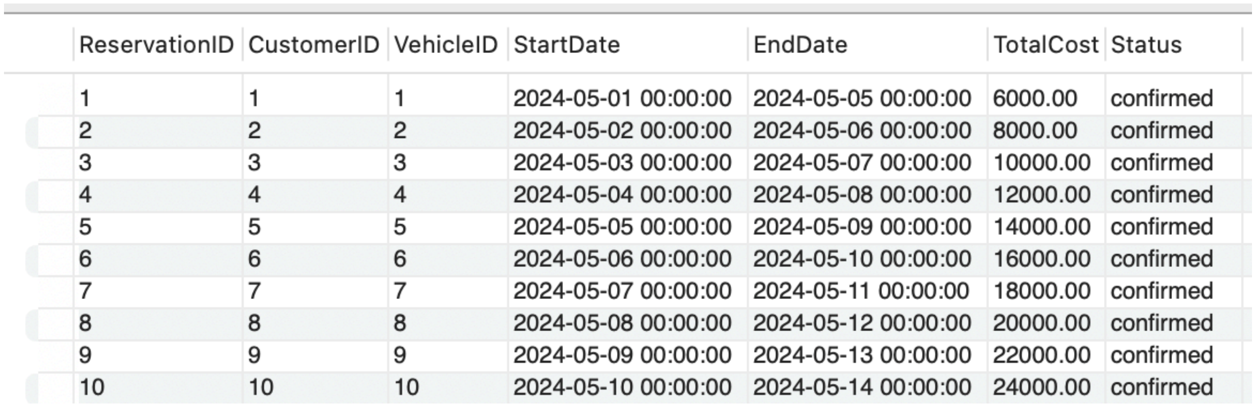
```
INSERT INTO Vehicle (Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate)
VALUES
('Sedan', 'Honda', 2022, 'Black', 'KA02AB1234', TRUE, 1500.00),
('SUV', 'Toyota', 2023, 'White', 'TN01CD5678', TRUE, 2000.00),
('Hatchback', 'Maruti', 2021, 'Red', 'MH03EF9101', TRUE, 1200.00),
('Sedan', 'Hyundai', 2024, 'Silver', 'KL04GH1122', TRUE, 1800.00),
('SUV', 'Ford', 2022, 'Blue', 'KA05IJ3344', TRUE, 2200.00),
('Hatchback', 'Volkswagen', 2023, 'Grey', 'TN06KL5566', TRUE, 1300.00),
('Sedan', 'Chevrolet', 2021, 'Green', 'MH07MN7788', TRUE, 1600.00),
('SUV', 'Jeep', 2024, 'Brown', 'KL08OP9900', TRUE, 2300.00),
('Hatchback', 'Nissan', 2022, 'Yellow', 'KA09QR2211', TRUE, 1400.00),
('Sedan', 'Audi', 2023, 'Purple', 'TN10ST4433', TRUE, 2500.00);
```



VehicleID	Model	Make	Year	Color	RegistrationNum...	Availability	DailyRate
1	Updated Model	Updated Make	2023	Blue	UPDATED123	1	60.00
2	SUV	Toyota	2023	White	TN01CD5678	1	2000.00
3	Hatchback	Maruti	2021	Red	MH03EF9101	1	1200.00
4	Sedan	Hyundai	2024	Silver	KL04GH1122	1	1800.00
5	SUV	Ford	2022	Blue	KA05IJ3344	1	2200.00
6	Hatchback	Volkswagen	2023	Grey	TN06KL5566	1	1300.00
7	Sedan	Chevrolet	2021	Green	MH07MN7788	1	1600.00
8	SUV	Jeep	2024	Brown	KL08OP9900	1	2300.00
9	Hatchback	Nissan	2022	Yellow	KA09QR2211	1	1400.00
10	Sedan	Audi	2023	Purple	TN10ST4433	1	2500.00
13	New Model	New Make	2022	Red	NEW1234789123	1	50.00
14	suv	2002	2002	red	21341234	0	500.00
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

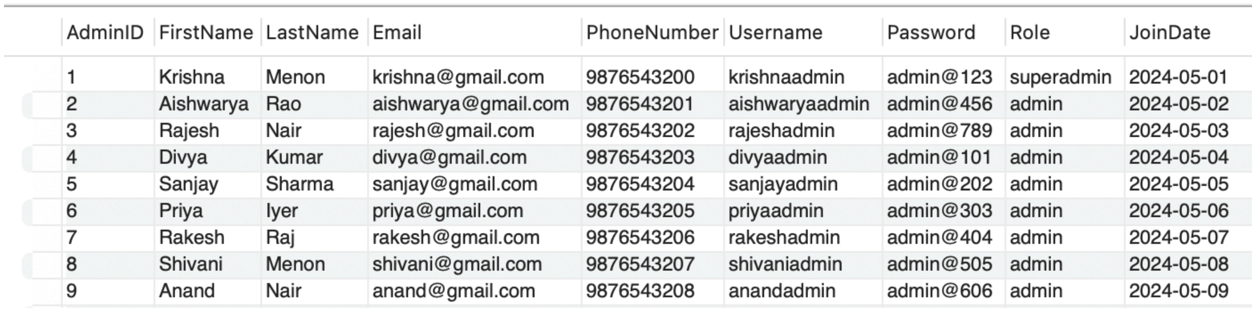
Reservation:

```
INSERT INTO Reservation (CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status)
VALUES
(1, 1, '2024-05-01', '2024-05-05', 6000.00, 'confirmed'),
(2, 2, '2024-05-02', '2024-05-06', 8000.00, 'confirmed'),
(3, 3, '2024-05-03', '2024-05-07', 10000.00, 'confirmed'),
(4, 4, '2024-05-04', '2024-05-08', 12000.00, 'confirmed'),
(5, 5, '2024-05-05', '2024-05-09', 14000.00, 'confirmed'),
(6, 6, '2024-05-06', '2024-05-10', 16000.00, 'confirmed'),
(7, 7, '2024-05-07', '2024-05-11', 18000.00, 'confirmed'),
(8, 8, '2024-05-08', '2024-05-12', 20000.00, 'confirmed'),
(9, 9, '2024-05-09', '2024-05-13', 22000.00, 'confirmed'),
(10, 10, '2024-05-10', '2024-05-14', 24000.00, 'confirmed');
```



ReservationID	CustomerID	VehicleID	StartDate	EndDate	TotalCost	Status
1	1	1	2024-05-01 00:00:00	2024-05-05 00:00:00	6000.00	confirmed
2	2	2	2024-05-02 00:00:00	2024-05-06 00:00:00	8000.00	confirmed
3	3	3	2024-05-03 00:00:00	2024-05-07 00:00:00	10000.00	confirmed
4	4	4	2024-05-04 00:00:00	2024-05-08 00:00:00	12000.00	confirmed
5	5	5	2024-05-05 00:00:00	2024-05-09 00:00:00	14000.00	confirmed
6	6	6	2024-05-06 00:00:00	2024-05-10 00:00:00	16000.00	confirmed
7	7	7	2024-05-07 00:00:00	2024-05-11 00:00:00	18000.00	confirmed
8	8	8	2024-05-08 00:00:00	2024-05-12 00:00:00	20000.00	confirmed
9	9	9	2024-05-09 00:00:00	2024-05-13 00:00:00	22000.00	confirmed
10	10	10	2024-05-10 00:00:00	2024-05-14 00:00:00	24000.00	confirmed

```
INSERT INTO Admin (FirstName, LastName, Email, PhoneNumber, Username, Password,
Role, JoinDate)
VALUES
('Krishna', 'Menon', 'krishna@gmail.com', '9876543200', 'krishnaadmin', 'admin@123',
'superadmin', '2024-05-01'),
('Aishwarya', 'Rao', 'aishwarya@gmail.com', '9876543201', 'aishwaryaadmin', 'admin@456',
'admin', '2024-05-02'),
('Rajesh', 'Nair', 'rajesh@gmail.com', '9876543202', 'rajeshadmin', 'admin@789', 'admin',
'2024-05-03'),
('Divya', 'Kumar', 'divya@gmail.com', '9876543203', 'divyaadmin', 'admin@101', 'admin',
'2024-05-04'),
('Sanjay', 'Sharma', 'sanjay@gmail.com', '9876543204', 'sanjayadmin', 'admin@202', 'admin',
'2024-05-05'),
('Priya', 'Iyer', 'priya@gmail.com', '9876543205', 'priyaadmin', 'admin@303', 'admin',
'2024-05-06'),
('Rakesh', 'Raj', 'rakesh@gmail.com', '9876543206', 'rakeshadmin', 'admin@404', 'admin',
'2024-05-07'),
('Shivani', 'Menon', 'shivani@gmail.com', '9876543207', 'shivaniadmin', 'admin@505', 'admin',
'2024-05-08'),
('Anand', 'Nair', 'anand@gmail.com', '9876543208', 'anandadmin', 'admin@606', 'admin',
'2024-05-09'),
('Manoj', 'Kumar', 'manoj@gmail.com', '9876543209', 'manojadmin', 'admin@707', 'admin',
'2024-05-10');
```



AdminID	FirstName	LastName	Email	PhoneNumber	Username	Password	Role	JoinDate
1	Krishna	Menon	krishna@gmail.com	9876543200	krishnaadmin	admin@123	superadmin	2024-05-01
2	Aishwarya	Rao	aishwarya@gmail.com	9876543201	aishwaryaadmin	admin@456	admin	2024-05-02
3	Rajesh	Nair	rajesh@gmail.com	9876543202	rajeshadmin	admin@789	admin	2024-05-03
4	Divya	Kumar	divya@gmail.com	9876543203	divyaadmin	admin@101	admin	2024-05-04
5	Sanjay	Sharma	sanjay@gmail.com	9876543204	sanjayadmin	admin@202	admin	2024-05-05
6	Priya	Iyer	priya@gmail.com	9876543205	priyaadmin	admin@303	admin	2024-05-06
7	Rakesh	Raj	rakesh@gmail.com	9876543206	rakeshadmin	admin@404	admin	2024-05-07
8	Shivani	Menon	shivani@gmail.com	9876543207	shivaniadmin	admin@505	admin	2024-05-08
9	Anand	Nair	anand@gmail.com	9876543208	anandadmin	admin@606	admin	2024-05-09

Create the model/entity classes corresponding to the schema within package entity with variables declared private, constructors (default and parametrized) and getters,setters)

Classes: **Customer**:
Properties: CustomerID, FirstName, LastName, Email, PhoneNumber, Address, Username, Password,
RegistrationDate Methods: Authenticate(password)

```
import re
from exception.exceptions import InvalidInputException
from datetime import datetime

class Customer:
   def __init__(self, customer_id=None, first_name="", last_name="", email="",
phone_number="", address="", username="", password="", registration_date=None):
       self.__customer_id = customer_id
       self.__first_name = first_name
       self.__last_name = last_name
       self.__email = self.__validate_email(email)
       self.__phone_number = phone_number
       self.__address = address
       self.__username = username
       self.__password = password
       self.__registration_date = datetime.now()

   def __validate_email(self, email):
       if re.match(r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,7}\b', email):
           return email
       else:
           raise InvalidInputException("Invalid email address provided.")
```

```python
@property
def customer_id(self):
    return self.__customer_id

@customer_id.setter
def customer_id(self, value):
    self.__customer_id = value

@property
def first_name(self):
    return self.__first_name

@first_name.setter
def first_name(self, first_name):
    if isinstance(first_name, str):
        self.__first_name = first_name
    else:
        raise InvalidInputException("First name must be a string.")

@property
def last_name(self):
    return self.__last_name

@last_name.setter
def last_name(self, last_name):
    if isinstance(last_name, str):
        self.__last_name = last_name
    else:
        raise InvalidInputException("Last name must be a string.")

@property
def email(self):
    return self.__email

@email.setter
def email(self, email):
    self.__email = self.__validate_email(email)

@property
def phone_number(self):
    return self.__phone_number
    @phone_number.setter
    def phone_number(self, phone):
        if isinstance(phone, str) and phone.isdigit():
            self.__phone_number = phone
        else:
            raise InvalidInputException("Phone number must be a string containing only
digits.")

    @property
    def address(self):
        return self.__address

    @address.setter
    def address(self, address):
        if isinstance(address, str):
            self.__address = address
        else:
            raise InvalidInputException("Address must be a string.")

    @property
    def username(self):
        return self.__username

    @username.setter
    def username(self, value):
        self.__username = value

    @property
    def password(self):
        return self.__password

    @password.setter
    def password(self, value):
        self.__password = value

    @property
    def registration_date(self):
        return self.__registration_date

    @registration_date.setter
    def registration_date(self, value):
        self.__registration_date = value
```

```
   def authenticate(self, password):
       return self.__password == password
```

Reservation: Properties: ReservationID, CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status Methods: CalculateTotalCost()

```
class Reservation:
   def __init__(self, reservation_id=None, customer_id=None, vehicle_id=None,
start_date=None, end_date=None, total_cost=None, status=None):
       self.__reservation_id = reservation_id
       self.__customer_id = customer_id
       self.__vehicle_id = vehicle_id
       self.__start_date = start_date
       self.__end_date = end_date
       self.__total_cost = total_cost
       self.__status = status

   @property
   def reservation_id(self):
       return self.__reservation_id

   @reservation_id.setter
   def reservation_id(self, value):
       self.__reservation_id = value

   @property
   def customer_id(self):
       return self.__customer_id

   @customer_id.setter
   def customer_id(self, value):
       self.__customer_id = value

   @property
   def vehicle_id(self):
       return self.__vehicle_id

   @vehicle_id.setter
   def vehicle_id(self, value):
       self.__vehicle_id = value

   @property
   def start_date(self):
       return self.__start_date

   @start_date.setter
   def start_date(self, value):
       self.__start_date = value

   @property
   def end_date(self):
       return self.__end_date

   @end_date.setter
   def end_date(self, value):
       self.__end_date = value

   @property
   def total_cost(self):
       return self.__total_cost

   @total_cost.setter
   def total_cost(self, value):
       self.__total_cost = value

   @property
   def status(self):
       return self.__status

   @status.setter
   def status(self, value):
       self.__status = value

   def calculate_total_cost(self):
       pass
```

Admin: Properties: AdminID, FirstName, LastName, Email, PhoneNumber, Username, Password, Role, JoinDate Methods: Authenticate(password)

```python
import re
from exception.exceptions import InvalidInputException

class Admin:
    def __init__(self, admin_id=None, first_name=None, last_name=None, email=None, phone_number=None, username=None, password=None, role=None, join_date=None):
        self.__admin_id = admin_id
        self.__first_name = first_name
        self.__last_name = last_name
        self.__email = self.__validate_email(email)
        self.__phone_number = phone_number
        self.__username = username
        self.__password = password
        self.__role = role
        self.__join_date = join_date

    def __validate_email(self, email):
        if re.match(r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,7}\b', email):
            return email
        else:
            print("Invalid email address.")

    @property
    def admin_id(self):
        return self.__admin_id

    @admin_id.setter
    def admin_id(self, value):
        self.__admin_id = value

    @property
    def first_name(self):
        return self.__first_name

    @first_name.setter
    def first_name(self, first_name):
        if isinstance(first_name, str):
            self.__first_name = first_name
        else:
            raise InvalidInputException("First name must be a string.")

    @property
    def last_name(self):
        return self.__last_name

    @last_name.setter
    def last_name(self, last_name):
        if isinstance(last_name, str):
            self.__last_name = last_name
        else:
            raise InvalidInputException("Last name must be a string.")

    @property
    def email(self):
        return self.__email

    @email.setter
    def email(self, email):
        self.__email = self.__validate_email(email)

    @property
    def phone_number(self):
        return self.__phone_number

    @phone_number.setter
    def phone_number(self, phone):
        if isinstance(phone, str) and phone.isdigit():
            self.__phone_number = phone
        else:
            raise InvalidInputException("Phone number must be a string containing only
digits.")

    @property
    def username(self):
        return self.__username

    @username.setter
    def username(self, value):
        self.__username = value

    @property
    def password(self):
        return self.__password
    @password.setter
    def password(self, value):
        self.__password = value

    @property
    def role(self):
        return self.__role

    @role.setter
    def role(self, value):
        self.__role = value

    @property
    def join_date(self):
        return self.__join_date

    @join_date.setter
    def join_date(self, value):
        self.__join_date = value

    def authenticate(self, password):
        return self.__password == password
```

CustomerService (implements ICustomerService): Methods: GetCustomerById, GetCustomerByUsername, RegisterCustomer, UpdateCustomer, DeleteCustomer

```
import mysql.connector

from dao.interface.ICustomerService import ICustomerService
from exception.exceptions import CustomerNotFoundException, InvalidInputException
from util.db_connection import DBConnection

class CustomerService(ICustomerService):

    def helper(self):
        connection = DBConnection.getConnection()
        cursor = connection.cursor()
        return cursor
```

```
   def raise_exception(self, e):
       print(e)
       return None

   def get_customer_by_id(self, customer_id):
       try:
           cursor = self.helper()
           cursor.execute("SELECT * FROM customer WHERE customerID = %s",
(customer_id,))
           customer = cursor.fetchone()
           cursor.close()
           if not customer:
               raise CustomerNotFoundException(f"Customer not found with ID
{customer_id}")
           return customer
       except CustomerNotFoundException as e:
           self.raise_exception(e)

   def get_customer_by_username(self, username):
       try:
           cursor = self.helper()
           cursor.execute("SELECT * FROM customer WHERE username = %s", (username,))
           customer = cursor.fetchone()
           cursor.close()
           if not customer:
               raise CustomerNotFoundException(f"Customer not found with username
{username}")
           return customer
       except CustomerNotFoundException as e:
           self.raise_exception(e)

   def register_customer(self, customer):
       try:
           cursor = self.helper()
           cursor.execute("SELECT COUNT(*) FROM customer WHERE username = %s",
(customer.username,))
           count = cursor.fetchone()[0]

           if count > 0:
               raise InvalidInputException("A customer with that username already
exists")
```

```
            cursor.execute("INSERT INTO customer (firstname, lastname, email,
phoneNumber, address, username, password, registrationDate) VALUES (%s, %s, %s, %s,
%s, %s, %s, %s)",
                            (customer.first_name, customer.last_name, customer.email,
customer.phone_number, customer.address, customer.username, customer.password,
customer.registration_date))
            self.connection.commit()
            cursor.close()
            return True
        except InvalidInputException as e:
            self.raise_exception(e)

    def update_customer(self, customer):
        try:
            cust = self.get_customer_by_id(customer.customer_id)
            if not cust:
                raise CustomerNotFoundException(message=f"customer with
{customer.customer_id} not found.")
            cursor = self.helper()
            cursor.execute("SELECT customerID FROM customer WHERE username = %s",
(customer.username,))
            existing_cust = cursor.fetchone()

            if existing_cust and existing_cust[0] != customer.customer_id:
                raise InvalidInputException("A customer with that username already
exists")
            cursor.execute("UPDATE customer SET firstname = %s, lastname = %s, email =
%s, phoneNumber = %s, address = %s, username = %s, password = %s WHERE customerID =
%s",
                            (customer.first_name, customer.last_name, customer.email,
customer.phone_number, customer.address, customer.username, customer.password,
customer.customer_id))
            self.connection.commit()
            cursor.close()
            return True
        except CustomerNotFoundException as e:
            self.raise_exception(e)

    def delete_customer(self, customer_id):
        try:
            cust = self.get_customer_by_id(customer_id)
            if not cust:
                raise CustomerNotFoundException(message=f"customer with {customer_id}
not found.")
            cursor = self.helper()
            cursor.execute("DELETE FROM customer WHERE customerID = %s",
(customer_id,))
            self.connection.commit()
            cursor.close()
            return True
        except CustomerNotFoundException as e:
            self.raise_exception(e)
```

VehicleService (implements IVehicleService): Methods: GetVehicleById, GetAvailableVehicles, AddVehicle, UpdateVehicle, RemoveVehicle

```
import mysql.connector

from dao.interface.IVehicleService import IVehicleService
from exception.exceptions import InvalidInputException, VehicleNotFoundException
from util.db_connection import DBConnection

class VehicleService(IVehicleService):

   def get_vehicle_by_id(self, vehicle_id):
       try:
           connection = DBConnection.getConnection()
           cursor = connection.cursor()
           cursor.execute("SELECT * FROM vehicle WHERE vehicleID = %s", (vehicle_id,))
           vehicle = cursor.fetchone()
           cursor.close()
           if not vehicle:
               raise VehicleNotFoundException("Vehicle not found.")
           return vehicle
       except mysql.connector.Error as e:
           print("Error getting vehicle by id:", e)
           return None
       except VehicleNotFoundException as e:
           print(e)
           return None
       finally:
           if 'connection' in locals() and connection.is_connected():
               connection.close()

   def get_all_vehicles(self):
       try:
           connection = DBConnection.getConnection()
           cursor = connection.cursor()
           cursor.execute("SELECT * FROM vehicle")
           vehicles = cursor.fetchall()
           cursor.close()
           for veh in vehicles:
               print(veh)
           return vehicles
       except mysql.connector.Error as e:
           print("Error getting vehicle by id:", e)
           return None
       finally:
           if 'connection' in locals() and connection.is_connected():
               connection.close()

   def get_available_vehicles(self):
       try:
           connection = DBConnection.getConnection()
           cursor = connection.cursor()
           cursor.execute("SELECT * FROM vehicle WHERE availability = %s", (True,))
           vehicles = cursor.fetchall()
           cursor.close()
           return vehicles
       except mysql.connector.Error as e:
           print("Error getting available vehicles:", e)
           return []
       finally:
           if 'connection' in locals() and connection.is_connected():
               connection.close()

   def add_vehicle(self, vehicle):
       try:
           connection = DBConnection.getConnection()
           cursor = connection.cursor()

           cursor.execute("SELECT COUNT(*) FROM vehicle WHERE registrationNumber =
%s", (vehicle.registration_number,))
           count = cursor.fetchone()[0]
           if count > 0:
               raise InvalidInputException("A vehicle with the same registration
number already exists")

           cursor.execute("INSERT INTO vehicle (model, make, year, color,
registrationNumber, availability, dailyRate) VALUES (%s, %s, %s, %s, %s, %s, %s)",
                       (vehicle.model, vehicle.make, vehicle.year, vehicle.color,
vehicle.registration_number, vehicle.availability, vehicle.daily_rate))
           connection.commit()
           cursor.close()
           return True
       except mysql.connector.Error as e:
           print("Error adding vehicle:", e)
           return False
       except InvalidInputException as e:
           print(e)
           return False
       finally:
           if 'connection' in locals() and connection.is_connected():
               connection.close()


   def update_vehicle(self, vehicle):
       try:
           veh = self.get_vehicle_by_id(vehicle.vehicle_id)
           if not veh:
               raise VehicleNotFoundException(message=f"vehicle with
{vehicle.vehicle_id} not found.")
           connection = DBConnection.getConnection()
           cursor = connection.cursor()
           cursor.execute("SELECT vehicleID FROM vehicle WHERE registrationNumber =
%s", (vehicle.registration_number,))
           existing_vehicle = cursor.fetchone()

           if existing_vehicle and existing_vehicle[0] != vehicle.vehicle_id:
               raise InvalidInputException("A vehicle with the same registration
number already exists")

           cursor.execute("UPDATE vehicle SET model = %s, make = %s, year = %s, color
= %s, registrationNumber = %s, availability = %s, dailyRate = %s WHERE vehicleID =
%s",
                          (vehicle.model, vehicle.make, vehicle.year, vehicle.color,
vehicle.registration_number, vehicle.availability, vehicle.daily_rate,
vehicle.vehicle_id))
           connection.commit()
           cursor.close()
           return True
       except mysql.connector.Error as e:
           print("Error updating vehicle:", e)
           return False
       except InvalidInputException as e:
           print(e)
           return False
       except VehicleNotFoundException as e:
           print(e)
           return False
       finally:
           if 'connection' in locals() and connection.is_connected():
               connection.close()

   def remove_vehicle(self, vehicle_id):
       try:
           veh = self.get_vehicle_by_id(vehicle_id)
           if not veh:
               raise VehicleNotFoundException(message=f"vehicle with {vehicle_id} not
found.")
           connection = DBConnection.getConnection()
           cursor = connection.cursor()
           cursor.execute("DELETE FROM vehicle WHERE vehicleID = %s", (vehicle_id,))
           connection.commit()
           cursor.close()
           return True
       except mysql.connector.Error as e:
           if e.errno == 1451:
               print("Cannot delete vehicle. There are reservations associated with
this vehicle.")
           else:
               print("Error removing vehicle:", e)
           return False
       except VehicleNotFoundException as e:
           print(e)
           return False
       finally:
           if 'connection' in locals() and connection.is_connected():
               connection.close()
```

ReservationService (implements IReservationService): Methods: GetReservationById, GetReservationsByCustomerId, CreateReservation, UpdateReservation, CancelReservation

```
from datetime import datetime
import mysql.connector

from dao.imp.customerService import CustomerService
from dao.imp.vehicleService import VehicleService
from dao.interface.IReservationService import IReservationService
from exception.exceptions import ReservationException
from util.db_connection import DBConnection

class ReservationService(IReservationService):
   def get_reservation_by_id(self, reservation_id):
       try:
           connection = DBConnection.getConnection()
           cursor = connection.cursor()
           cursor.execute("SELECT * FROM reservation WHERE reservationID = %s",
(reservation_id,))
           reservation = cursor.fetchone()
           cursor.close()
           return reservation
       except mysql.connector.Error as e:
           print("Error getting reservation by id:", e)
           return None
       finally:
           if 'connection' in locals() and connection.is_connected():
               connection.close()

   def get_reservations_by_customer_id(self, customer_id):
       try:
           connection = DBConnection.getConnection()
           cursor = connection.cursor()
           cursor.execute("SELECT * FROM reservation WHERE customerID = %s",
(customer_id,))
           reservations = cursor.fetchall()
           cursor.close()
```

```
            return reservations
        except mysql.connector.Error as e:
            print("Error getting reservations by customer id:", e)
            return []
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()

    def create_reservation(self, reservation):
        try:
            if not self.does_customer_exist(reservation.customer_id):
                raise ReservationException(f"Customer with ID {reservation.customer_id}
does not exist.")
            if not self.does_vehicle_exist(reservation.vehicle_id):
                raise ReservationException(f"Vehicle with ID {reservation.vehicle_id}
does not exist.")
            if self.is_vehicle_booked(reservation.vehicle_id, reservation.start_date,
reservation.end_date, reservation_id=None):
                raise ReservationException("Vehicle is already booked for the specified
dates.")

            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            cursor.execute("INSERT INTO reservation (customerID, vehicleID, startDate,
endDate, totalCost, status) VALUES (%s, %s, %s, %s, %s, %s)",
                          (reservation.customer_id, reservation.vehicle_id,
reservation.start_date, reservation.end_date, reservation.total_cost,
reservation.status))
            connection.commit()
            cursor.close()
            return True
        except mysql.connector.Error as e:
            print("Error creating reservation:", e)
            return False
        except ReservationException as e:
            print("Reservation error:", e)
            return False
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()

    def is_vehicle_booked(self, vehicle_id, start_date, end_date, reservation_id):
        try:
            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            start_date = datetime.strptime(start_date, '%Y-%m-%d')
            end_date = datetime.strptime(end_date, '%Y-%m-%d')
            if(reservation_id == None):
                cursor.execute("SELECT COUNT(*) FROM reservation WHERE vehicleID = %s
AND ((%s BETWEEN startDate AND endDate) OR (%s BETWEEN startDate AND endDate))",
                        (vehicle_id, start_date, end_date))
            else:
                cursor.execute("SELECT COUNT(*) FROM reservation WHERE vehicleID = %s
AND ((%s BETWEEN startDate AND endDate) OR (%s BETWEEN startDate AND endDate)) AND
reservationID != %s",
                        (vehicle_id, start_date, end_date, reservation_id))
            count = cursor.fetchone()[0]
            return count>0
        except mysql.connector.Error as e:
            print("Error checking existing reservation:", e)
            return False
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()

    def does_customer_exist(self, customer_id):
        customer_service = CustomerService()
        customer = customer_service.get_customer_by_id(customer_id)
        return customer is not None

    def does_vehicle_exist(self, vehicle_id):
        vehicle_service = VehicleService()
        vehicle = vehicle_service.get_vehicle_by_id(vehicle_id)
        return vehicle is not None


    def update_reservation(self, reservation):
        try:
            reser = self.get_reservation_by_id(reservation.reservation_id)
            if not reser:
                raise ReservationException(message=f"reservation with ID
{reservation.reservation_id} not found.")
            if self.is_vehicle_booked(reser[2], reservation.start_date,
reservation.end_date, reservation_id=reservation.reservation_id):
```

```
               raise ReservationException("Vehicle is already booked for the specified
dates.")
           connection = DBConnection.getConnection()
           cursor = connection.cursor()
           cursor.execute("UPDATE reservation SET customerID = %s, vehicleID = %s,
startDate = %s, endDate = %s, totalCost = %s, status = %s WHERE reservationID = %s",
                          (reser[1], reser[2], reservation.start_date,
reservation.end_date, reservation.total_cost, reservation.status,
reservation.reservation_id))
           connection.commit()
           cursor.close()
           return True
       except mysql.connector.Error as e:
           print("Error updating reservation:", e)
           return False
       except ReservationException as e:
           print(e)
           return False
       finally:
           if 'connection' in locals() and connection.is_connected():
               connection.close()

   def cancel_reservation(self, reservation_id):
       try:
           reser = self.get_reservation_by_id(reservation_id)
           if not reser:
               raise ReservationException(message=f"reservation with ID
{reservation_id} not found.")
           connection = DBConnection.getConnection()
           cursor = connection.cursor()
           cursor.execute("DELETE FROM reservation WHERE reservationID = %s",
(reservation_id,))
           connection.commit()
           cursor.close()
           return True
       except mysql.connector.Error as e:
           print("Error canceling reservation:", e)
           return False
       except ReservationException as e:
           print(e)
           return False
       finally:
           if 'connection' in locals() and connection.is_connected():
               connection.close()
```

DatabaseContext: A class responsible for handling database connections and interactions.
AuthenticationService: A class responsible for handling user authentication.

```
import mysql.connector
from exception.exceptions import DatabaseConnectionException

class DBConnection:
   __connection = None

   @staticmethod
   def getConnection():
       try:
           DBConnection.__connection = mysql.connector.connect(
               host="localhost",
               user="root",
               password="ani28790'",
               database="carconnect2",
               port=3306
           )
           print("Connected to MySQL database successfully.")
           return DBConnection.__connection
       except mysql.connector.Error as err:
           print("Error connecting to MySQL:", err)
           raise DatabaseConnectionException(err)
```

Interfaces:
 ICustomerService: • GetCustomerById(customerId) • GetCustomerByUsername(username) • RegisterCustomer(customerData) • UpdateCustomer(customerData) • DeleteCustomer(customerId)

```
from abc import ABC, abstractmethod
```

```python
class IAdminService(ABC):
   @abstractmethod
   def get_admin_by_id(self, admin_id):
       pass

   @abstractmethod
   def get_admin_by_username(self, username):
       pass

   @abstractmethod
   def register_admin(self, admin_data):
       pass

   @abstractmethod
   def update_admin(self, admin_data):
       pass

   @abstractmethod
   def delete_admin(self, admin_id):
       pass
```

IVehicleService: • GetVehicleById(vehicleId) • GetAvailableVehicles() • AddVehicle(vehicleData) • UpdateVehicle(vehicleData) • RemoveVehicle(vehicleId)

```python
from abc import ABC, abstractmethod

class IVehicleService(ABC):
   @abstractmethod
   def get_vehicle_by_id(self, vehicle_id):
       pass

   @abstractmethod
   def get_available_vehicles(self):
       pass

   @abstractmethod
   def add_vehicle(self, vehicle_data):
       pass
```

```
    @abstractmethod
    def update_vehicle(self, vehicle_data):
        pass

    @abstractmethod
    def remove_vehicle(self, vehicle_id):
        pass
```

IReservationService: • GetReservationById(reservationId) • GetReservationsByCustomerId(customerId) • CreateReservation(reservationData) • UpdateReservation(reservationData) • CancelReservation(reservationId)

```
from abc import ABC, abstractmethod

class IReservationService(ABC):
    @abstractmethod
    def get_reservation_by_id(self, reservation_id):
        pass

    @abstractmethod
    def get_reservations_by_customer_id(self, customer_id):
        pass

    @abstractmethod
    def create_reservation(self, reservation_data):
        pass

    @abstractmethod
    def update_reservation(self, reservation_data):
        pass

    @abstractmethod
    def cancel_reservation(self, reservation_id):
        pass
```

IAdminService: • GetAdminById(adminId) • GetAdminByUsername(username) • RegisterAdmin(adminData) • UpdateAdmin(adminData) • DeleteAdmin(adminId)

```
from abc import ABC, abstractmethod
```

```
class IAdminService(ABC):
   @abstractmethod
   def get_admin_by_id(self, admin_id):
       pass

   @abstractmethod
   def get_admin_by_username(self, username):
       pass

   @abstractmethod
   def register_admin(self, admin_data):
       pass

   @abstractmethod
   def update_admin(self, admin_data):
       pass

   @abstractmethod
   def delete_admin(self, admin_id):
       pass
```

Custom Exceptions: AuthenticationException: • Thrown when there is an issue with user authentication. • Example Usage: Incorrect username or password during customer or admin login. ReservationException: • Thrown when there is an issue with reservations. • Example Usage: Attempting to make a reservation for a vehicle that is already reserved. VehicleNotFoundException: • Thrown when a requested vehicle is not found. • Example Usage: Trying to get details of a vehicle that does not exist. AdminNotFoundException: • Thrown when an admin user is not found. • Example Usage: Attempting to access details of an admin that does not exist. InvalidInputException: • Thrown when there is invalid input data. • Example Usage: When a required field is missing or has an incorrect format. DatabaseConnectionException: • Thrown when there is an issue with the database connection. • Example Usage: Unable to establish a connection to the database.

```
class AuthenticationException(Exception):
   def __init__(self, message="Authentication error"):
       self.message = message
       super().__init__(self.message)
```

```python
class ReservationException(Exception):
   def __init__(self, message="Reservation error"):
       self.message = message
       super().__init__(self.message)
```

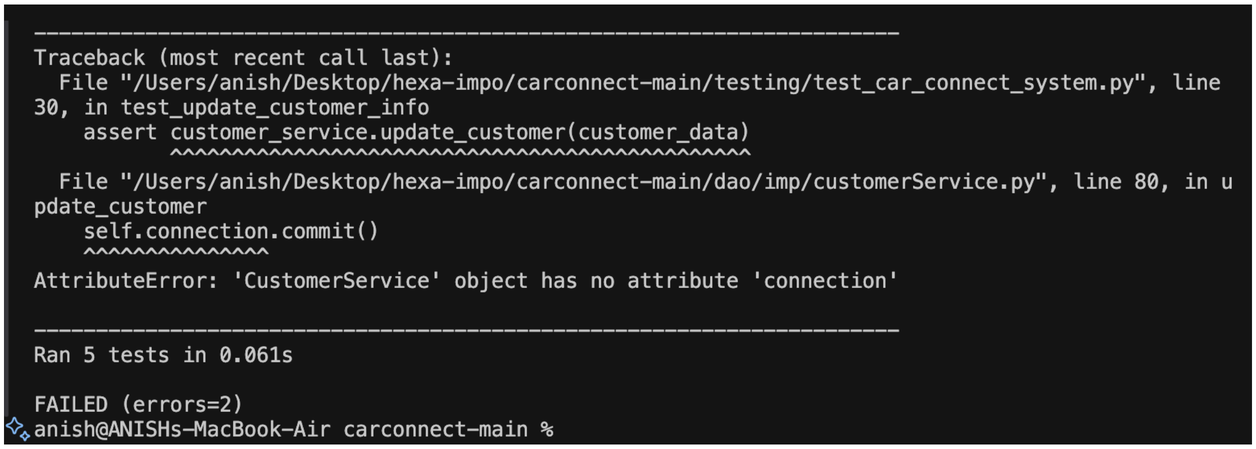
```python
class AdminNotFoundException(Exception):
   def __init__(self, message="Admin not found"):
       self.message = message
       super().__init__(self.message)
```

```python
class VehicleNotFoundException(Exception):
   def __init__(self, message="Vehicle not found"):
       self.message = message
       super().__init__(self.message)
```

```python
class DatabaseConnectionException(Exception):
   def __init__(self, message="Database connection error"):
       self.message = message
       super().__init__(self.message)
```

Unit Testing:

Create NUnit test cases for car rental System are essential to ensure the correctness and reliability of your system. Below are some example questions to guide the creation of NUnit test cases for various components of the system:



```
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/anish/Desktop/hexa-impo/carconnect-main/testing/test_car_connect_system.py", line
30, in test_update_customer_info
    assert customer_service.update_customer(customer_data)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/Users/anish/Desktop/hexa-impo/carconnect-main/dao/imp/customerService.py", line 80, in u
pdate_customer
    self.connection.commit()
    ^^^^^^^^^^^^^^^
AttributeError: 'CustomerService' object has no attribute 'connection'

----------------------------------------------------------------------
Ran 5 tests in 0.061s

FAILED (errors=2)
anish@ANISHs-MacBook-Air carconnect-main %
```