CaseStudy – CarConnect
By-Kunal Sudhir Mishra

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

1. Customer Table:

```
mysql> use carconnect;
Database changed
mysql> -- Customer Table
mysql> CREATE TABLE Customer (
    ->     CustomerID INT PRIMARY KEY,
    ->     FirstName VARCHAR(255),
    ->     LastName VARCHAR(255),
    ->     Email VARCHAR(255),
    ->     PhoneNumber VARCHAR(20),
    ->     Address TEXT,
    ->     Username VARCHAR(50) UNIQUE,
    ->     Password VARCHAR(255),
    ->     RegistrationDate DATETIME
    -> );
Query OK, 0 rows affected (0.06 sec)
```

Vehicle Table

```
mysql> -- Vehicle Table
mysql> CREATE TABLE Vehicle (
    ->     VehicleID INT PRIMARY KEY,
    ->     Model VARCHAR(255),
    ->     Make VARCHAR(255),
    ->     Year INT,
    ->     Color VARCHAR(50),
    ->     RegistrationNumber VARCHAR(20) UNIQUE,
    ->     Availability BOOLEAN,
    ->     DailyRate DECIMAL(10, 2)
    -> );
Query OK, 0 rows affected (0.03 sec)
```

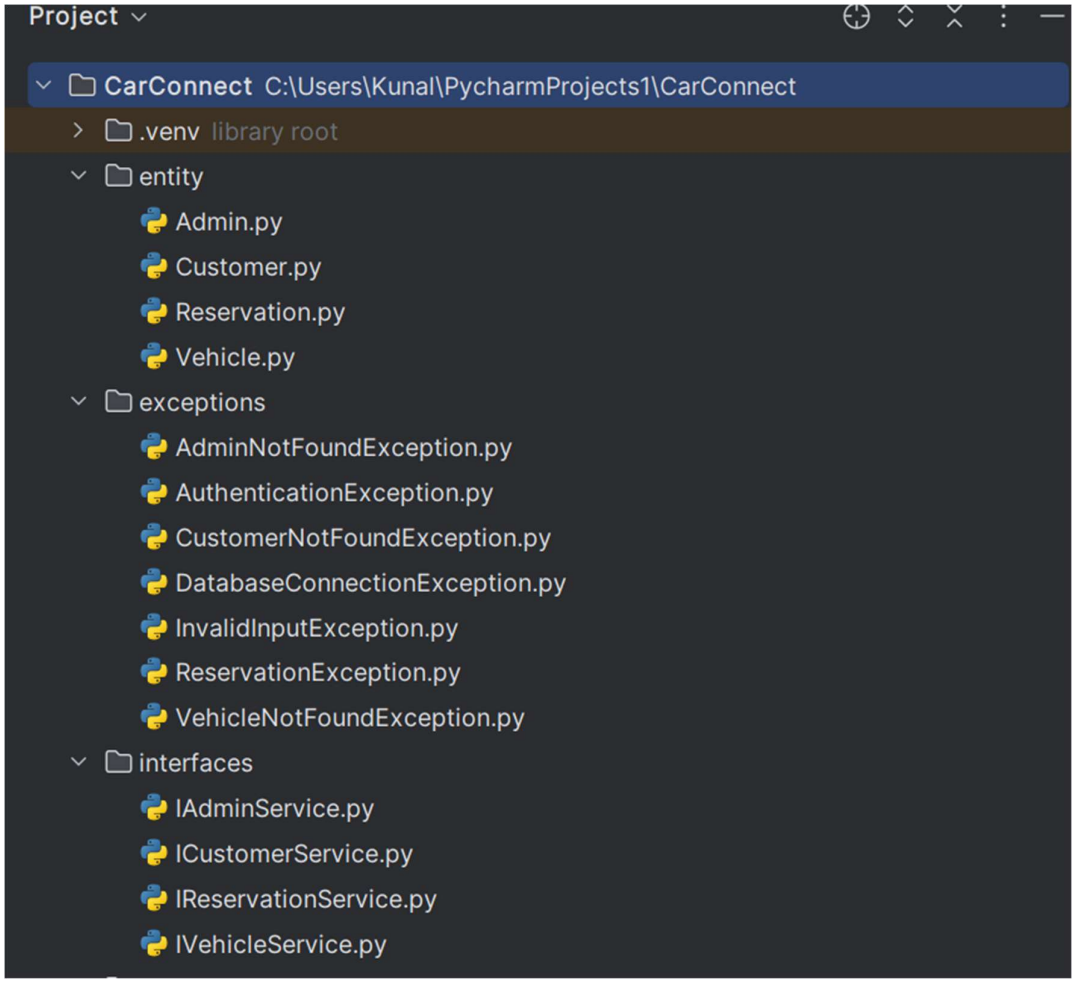
Reservation Table

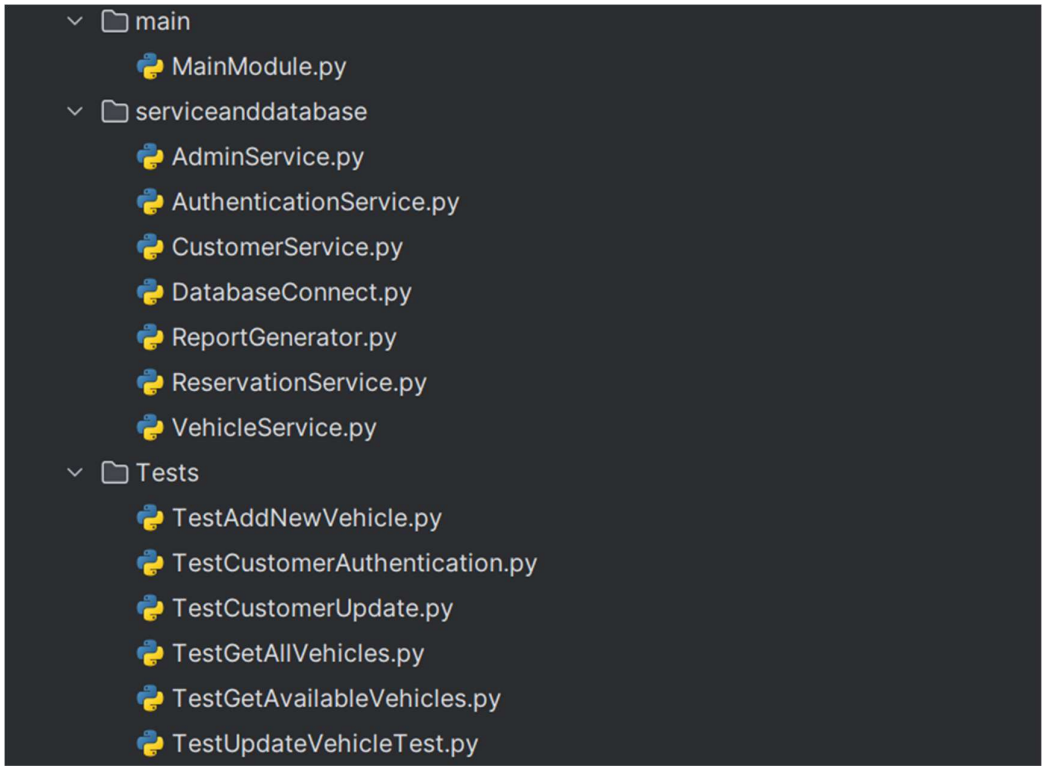
```
mysql> -- Reservation Table
mysql> CREATE TABLE Reservation (
    ->     ReservationID INT PRIMARY KEY,
    ->     CustomerID INT,
    ->     VehicleID INT,
    ->     StartDate DATETIME,
    ->     EndDate DATETIME,
    ->     TotalCost DECIMAL(10, 2),
    ->     Status VARCHAR(20),
    ->     FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),
    ->     FOREIGN KEY (VehicleID) REFERENCES Vehicle(VehicleID)
    -> );
Query OK, 0 rows affected (0.05 sec)
```

Admin Table

```
mysql> -- Admin Table
mysql> CREATE TABLE Admin (
    ->     AdminID INT PRIMARY KEY,
    ->     FirstName VARCHAR(255),
    ->     LastName VARCHAR(255),
    ->     Email VARCHAR(255),
    ->     PhoneNumber VARCHAR(20),
    ->     Username VARCHAR(50) UNIQUE,
    ->     Password VARCHAR(255),
    ->     Role VARCHAR(50),
    ->     JoinDate DATETIME
    -> );
Query OK, 0 rows affected (0.03 sec)
```

Project Structure




main
MainModule.py
serviceanddatabase
AdminService.py
AuthenticationService.py
CustomerService.py
DatabaseConnect.py
ReportGenerator.py
ReservationService.py
VehicleService.py
Tests
TestAddNewVehicle.py
TestCustomerAuthentication.py
TestCustomerUpdate.py
TestGetAllVehicles.py
TestGetAvailableVehicles.py
TestUpdateVehicleTest.py

Classes

Customer

```
    def __init__(self, customer_id, first_name, last_name, email, phone_number, address  2
        self.__CustomerID = customer_id
        self.__FirstName = first_name
        self.__LastName = last_name
        self.__Email = email
        self.__PhoneNumber = phone_number
        self.__Address = address
        self.__Username = username
        self.__Password = password
        self.__RegistrationDate = registration_date

    # Getter methods
    1 usage
    @property
    def customer_id(self):
        return self.__CustomerID

    2 usages (1 dynamic)
    @property
    def first_name(self):
        return self.__FirstName

    @customer_id.setter
    def customer_id(self, customer_id):
        self.__CustomerID = customer_id

    1 usage (1 dynamic)
    @first_name.setter
    def first_name(self, first_name):
        self.__FirstName = first_name

    1 usage (1 dynamic)
    @last_name.setter
    def last_name(self, last_name):
        self.__LastName = last_name

    1 usage (1 dynamic)
    @email.setter
    def email(self, email):
        self.__Email = email
```

```
1 usage (1 dynamic)
@address.setter
def address(self, address):
    self.__Address = address

@username.setter
def username(self, username):
    self.__Username = username

@password.setter
def password(self, password):
    self.__Password = password

@registration_date.setter
def registration_date(self, registration_date):
    self.__RegistrationDate = registration_date

3 usages (3 dynamic)
def authenticate(self, password):
    return self.__Password == password
```

Vehicle

```
class Vehicle:
    def __init__(self, vehicle_id, model, make, year, color, registration_number,
                 availability, daily_rate):
        self.__VehicleID = vehicle_id
        self.__Model = model
        self.__Make = make
        self.__Year = year
        self.__Color = color
        self.__RegistrationNumber = registration_number
        self.__Availability = availability
        self.__DailyRate = daily_rate

    # Getter methods with @property decorator
    1 usage
    @property
    def vehicle_id(self):
        return self.__VehicleID

    3 usages
    @property
    def model(self):
        return self.__Model
```

```
    @property
    def year(self):
        return self.__Year

    3 usages
    @property
    def color(self):
        return self.__Color

    3 usages
    @property
    def registration_number(self):
        return self.__RegistrationNumber

    4 usages
    @property
    def availability(self):
        return self.__Availability

    3 usages
    @property
    def daily_rate(self):
        return self.__DailyRate
```

```python
    def make(self, make):
        self.__Make = make

    @year.setter
    def year(self, year):
        self.__Year = year

    @color.setter
    def color(self, color):
        self.__Color = color

    @registration_number.setter
    def registration_number(self, registration_number):
        self.__RegistrationNumber = registration_number

    @availability.setter
    def availability(self, availability):
        self.__Availability = availability

    @daily_rate.setter
    def daily_rate(self, daily_rate):
        self.__DailyRate = daily_rate
```

Reservation

```
class Reservation:
    def __init__(self, reservation_id, customer_id, vehicle_id, start_date, end_date, total_cost
        self.__ReservationID = reservation_id
        self.__CustomerID = customer_id
        self.__VehicleID = vehicle_id
        self.__StartDate = start_date
        self.__EndDate = end_date
        self.__TotalCost = total_cost
        self.__Status = status

    # Getter methods with @property decorator
    1 usage
    @property
    def reservation_id(self):
        return self.__ReservationID

    1 usage
    @property
    def customer_id(self):
        return self.__CustomerID
```

```
    @reservation_id.setter
    def reservation_id(self, reservation_id):
        self.__ReservationID = reservation_id

    @customer_id.setter
    def customer_id(self, customer_id):
        self.__CustomerID = customer_id

    @vehicle_id.setter
    def vehicle_id(self, vehicle_id):
        self.__VehicleID = vehicle_id

    @start_date.setter
    def start_date(self, start_date):
        self.__StartDate = start_date

    @end_date.setter
    def end_date(self, end_date):
        self.__EndDate = end_date
```

```python
    @total_cost.setter
    def total_cost(self, total_cost):
        self.__TotalCost = total_cost

    @status.setter
    def status(self, status):
        self.__Status = status

    def calculate_total_cost(self):
        pass
```

Admin

```
3 usages
class Admin:
    def __init__(self, admin_id, first_name, last_name, email, phone_number, username, password
        self.__AdminID = admin_id
        self.__FirstName = first_name
        self.__LastName = last_name
        self.__Email = email
        self.__PhoneNumber = phone_number
        self.__Username = username
        self.__Password = password
        self.__Role = role
        self.__JoinDate = join_date

    1 usage
    @property
    def admin_id(self):
        return self.__AdminID

    2 usages (1 dynamic)
    @property
    def first_name(self):
        return self.__FirstName
```

```
    def admin_id(self, admin_id):
        self.__AdminID = admin_id

    1 usage (1 dynamic)
    @first_name.setter
    def first_name(self, first_name):
        self.__FirstName = first_name

    1 usage (1 dynamic)
    @last_name.setter
    def last_name(self, last_name):
        self.__LastName = last_name

    1 usage (1 dynamic)
    @email.setter
    def email(self, email):
        self.__Email = email

    1 usage (1 dynamic)
    @phone_number.setter
    def phone_number(self, phone_number):
        self.__PhoneNumber = phone_number
```

```python
    def username(self, username):
        self.__Username = username

    @password.setter
    def password(self, password):
        self.__Password = password

    @role.setter
    def role(self, role):
        self.__Role = role

    @join_date.setter
    def join_date(self, join_date):
        self.__JoinDate = join_date

    3 usages (3 dynamic)
    def authenticate(self, password):
        return self.__Password == password
```

Services

CustomerService (implements ICustomerService)

```python
from datetime import datetime
import mysql.connector

from exceptions.CustomerNotFoundException import CustomerNotFoundException
from exceptions.InvalidInputException import InvalidInputException
from interfaces.ICustomerService import ICustomerService
from entity.Customer import Customer
from Utils.Validator import InputValidator


6 usages
class CustomerService(ICustomerService):
    def __init__(self, db_context):
        self.db_context = db_context

    3 usages
    def get_customer_by_id(self, customer_id):
        query = "SELECT * FROM Customer WHERE CustomerID = %s"
        params = (customer_id,)
        result = self.db_context.execute_query(query, params)
        if result:
            return Customer(*result[0])
        else:
```

```python
        1 usage
        def update_customer(self, customer_data):
            self.get_customer_by_id(customer_data['CustomerID'])  # Validate if customer exists
            query = ("UPDATE Customer SET FirstName = %s,"
                     " LastName = %s, Email = %s, PhoneNumber = %s, Address = %s WHERE "
                     "CustomerID = %s")
            params = (
                customer_data['FirstName'],
                customer_data['LastName'],
                customer_data['Email'],
                customer_data['PhoneNumber'],
                customer_data['Address'],
                customer_data['CustomerID']
            )
            self.db_context.execute_query(query, params)

        def delete_customer(self, customer_id):
            self.get_customer_by_id(customer_id)  # Validate if customer exists
            query = "DELETE FROM Customer WHERE CustomerID = %s"
            params = (customer_id,)
            self.db_context.execute_query(query, params)
```

VehicleService (implements IVehicleService)

```python
from exceptions.VehicleNotFoundException import VehicleNotFoundException
from interfaces.IVehicleService import IVehicleService



10 usages
class VehicleService(IVehicleService):
    def __init__(self, db_context):
        self.db_context = db_context

    4 usages (1 dynamic)
    def get_vehicle_by_id(self, vehicle_id):
        query = "SELECT * FROM Vehicle WHERE VehicleID = %s"
        params = (vehicle_id,)
        result = self.db_context.execute_query(query, params)
        if result:
            return result
        else:
            raise VehicleNotFoundException(f"Vehicle with ID {vehicle_id} not found.")

    3 usages
    def get_available_vehicles(self):
        query = "SELECT * FROM Vehicle WHERE Availability = True"
```

ReservationService (implements IReservationService)

```python
from exceptions.ReservationException import ReservationException
from interfaces.IReservationService import IReservationService
from entity.Reservation import Reservation


2 usages
class ReservationService(IReservationService):
    def __init__(self, db_context):
        self.db_context = db_context

    1 usage (1 dynamic)
    def get_reservation_by_id(self, reservation_id):
        query = "SELECT * FROM Reservation WHERE ReservationID = %s"
        params = (reservation_id,)
        result = self.db_context.execute_query(query, params)
        if result:
            return Reservation(**result[0])

    def get_reservations_by_customer_id(self, customer_id):
        query = "SELECT * FROM Reservation WHERE CustomerID = %s"
        params = (customer_id,)
        results = self.db_context.execute_query(query, params)
        return [Reservation(**res) for res in results]
```

AdminService (implements IAdminService)

```python
from exceptions.AdminNotFoundException import AdminNotFoundException
from interfaces.IAdminService import IAdminService
from entity.Admin import Admin
from datetime import datetime


2 usages
class AdminService(IAdminService):
    def __init__(self, db_context):
        self.db_context = db_context

    2 usages
    def get_admin_by_id(self, admin_id):
        query = "SELECT * FROM Admin WHERE AdminID = %s"
        params = (admin_id,)
        result = self.db_context.execute_query(query, params)
        if result:
            return Admin(**result[0])
        else:
            raise AdminNotFoundException()

    1 usage (1 dynamic)
    def get_admin_by_username(self, username):
```

```python
    def update_admin(self, admin_data):
        self.get_admin_by_id(admin_data['AdminID'])  # Validate if admin exists
        query = ("UPDATE Admin SET FirstName = %s, LastName = %s, Email = %s, PhoneNumber = %s "
                 "WHERE AdminID = %s")
        params = (
            admin_data['FirstName'],
            admin_data['LastName'],
            admin_data['Email'],
            admin_data['PhoneNumber'],
            admin_data['AdminID']
        )
        self.db_context.execute_query(query, params)

    def delete_admin(self, admin_id):
        self.get_admin_by_id(admin_id)   # Validate if admin exists
        query = "DELETE FROM Admin WHERE AdminID = %s"
        params = (admin_id,)
        self.db_context.execute_query(query, params)
```

DatabaseConnect:

```python
import mysql.connector

from exceptions.DatabaseConnectionException import DatabaseConnectionException


14 usages
class DatabaseConnect:
    def __init__(self, host='localhost', user='root', password='root', database='mydatabase'):
        self.host = host
        self.user = user
        self.password = password
        self.database = database
        self.connection = None
        self.cursor = None

    7 usages
    def connect(self):
        try:
            self.connection = mysql.connector.connect(
                host=self.host,
                user=self.user,
                password=self.password,
                database=self.database
```

ReportGenerator

```python
class ReportGenerator:
    def __init__(self, db_context, reservation_service=None, vehicle_service=None):
        self.reservation_service = reservation_service
        self.vehicle_service = vehicle_service
        self.db_context = db_context

    def generate_reservation_report(self, reservation_id):
        reservation = self.reservation_service.get_reservation_by_id(reservation_id)
        if reservation:
            report = f"Reservation Report\nReservation ID: {reservation.get_reservation_id()}\nC
            return report
        return "Reservation not found."

    def generate_vehicle_report(self, vehicle_id):
        vehicle = self.vehicle_service.get_vehicle_by_id(vehicle_id)
        if vehicle:
            report = f"Vehicle Report\nVehicle ID: {vehicle.get_vehicle_id()}\nModel: {vehicle.g
            return report
        return "Vehicle not found."

    1 usage
    def view_overall_revenue(self):
        query = "SELECT SUM(TotalCost) AS OverallRevenue FROM Reservation"
```

Authentication

```python
from exceptions.AuthenticationException import AuthenticationException

4 usages
class AuthenticationService:
    def __init__(self, customer_service=None, admin_service=None):
        self.customer_service = customer_service
        self.admin_service = admin_service

    3 usages
    def authenticate_customer(self, username, password):
        customer = self.customer_service.get_customer_by_username(username)
        if not customer.authenticate(password):
            raise AuthenticationException("Incorrect Username or Password")
        if customer:
            return customer
        raise AuthenticationException()

    1 usage
    def authenticate_admin(self, username, password):
        admin = self.admin_service.get_admin_by_username(username)
        if not admin.authenticate(password):
            raise AuthenticationException("Incorrect Username or Password")
```

Interfaces

ICustomerService

```python
from abc import ABC, abstractmethod


2 usages
class ICustomerService(ABC):
    @abstractmethod
    def get_customer_by_id(self, customer_id):
        pass

    1 usage (1 dynamic)
    @abstractmethod
    def get_customer_by_username(self, username):
        pass

    @abstractmethod
    def register_customer(self, customer_data):
        pass

    @abstractmethod
    def update_customer(self, customer_data):
        pass

    @abstractmethod
```

IVehicleService

```python
from abc import ABC, abstractmethod


2 usages
class IVehicleService(ABC):
    1 usage (1 dynamic)
    @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
```

IReservationService

```python
from abc import ABC, abstractmethod


2 usages
class IReservationService(ABC):
    1 usage (1 dynamic)
    @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
```

IAdminService

```python
from abc import ABC, abstractmethod


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

    1 usage (1 dynamic)
    @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
```

Connect your application to the SQL database

Database connection is done through mysql-python-connector. DatabaseConnect.py

```python
import mysql.connector

from exceptions.DatabaseConnectionException import DatabaseConnectionException


14 usages
class DatabaseConnect:
    def __init__(self, host='localhost', user='root', password='root', database='mydatabase'):
        self.host = host
        self.user = user
        self.password = password
        self.database = database
        self.connection = None
        self.cursor = None

    7 usages
    def connect(self):
        try:
            self.connection = mysql.connector.connect(
                host=self.host,
                user=self.user,
                password=self.password,
                database=self.database
```

Initialized the connection before the menu display
MainModule.py

```python

if __name__ == "__main__":
    db_context = DatabaseConnect(database="carconnect")
    db_context.connect()
    interface = MainModule(db_context)
    interface.main_menu()
```

MainModule › show_vehicle_menu() › try › else

MainModule

```
C:\Users\Kunal\PycharmProjects1\CarConnect\.venv\Scripts\python.exe C:\Users\Kunal\PycharmProjects1\CarConnect
Connected to the database: carconnect

Process finished with exit code 0
```

Exceptions

AdminNotFoundException.py

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

AdminNotFoundException.py | AuthenticationException.py

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

AdminNotFoundException.py | AuthenticationException.py | CustomerNotFoundException.py

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

Exception.py | AuthenticationException.py | CustomerNotFoundException.py | DatabaseConnectionException.py

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

NotFoundException.py | DatabaseConnectionException.py | ReservationException.py | InvalidInputException.py

```python
class InvalidInputException(Exception):
    def __init__(self, message="Invalid input"):
        self.message = message
        super().__init__(self.message)
```

NotFoundException.py | DatabaseConnectionException.py | ReservationException.py | InvalidInputException.py

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

oundException.py | DatabaseConnectionException.py | ReservationException.py | VehicleNotFoundException.py

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

Unit Testing

1. **Test customer authentication with invalid credentials.**

```
import unittest
from exceptions.AuthenticationException import AuthenticationException
from serviceanddatabase.AuthenticationService import AuthenticationService
from serviceanddatabase.CustomerService import CustomerService
from serviceanddatabase.DatabaseConnect import DatabaseConnect


class TestCustomerAuthentication(unittest.TestCase):
    def setUp(self):
        db_context = DatabaseConnect(database="carconnect")
        db_context.connect()
        self.customer_service = CustomerService(db_context)
        self.auth_service = AuthenticationService(self.customer_service)

    def test_invalid_credentials(self):
        invalid_username = "notvatsal"
```

2. **Test updating customer information.**

```
import unittest
from exceptions.InvalidInputException import InvalidInputException
from serviceanddatabase.CustomerService import CustomerService
from serviceanddatabase.DatabaseConnect import DatabaseConnect


class TestCustomerUpdate(unittest.TestCase):
    def setUp(self):
        db_context = DatabaseConnect(database="carconnect")
        db_context.connect()
        self.customer_service = CustomerService(db_context)

        try:
            self.customer_service.update_customer(updated_info)

            updated_customer = self.customer_service.get_customer_by_id(existing_customer_id)
            self.assertEqual(updated_info['FirstName'], updated_customer.first_name)
            self.assertEqual(updated_info['LastName'], updated_customer.last_name)
            self.assertEqual(updated_info['Email'], updated_customer.email)
            self.assertEqual(updated_info['PhoneNumber'], updated_customer.phone_number)
            self.assertEqual(updated_info['Address'], updated_customer.address)

        except InvalidInputException as e:
            self.fail(f"Unexpected exception raised: {e}")


if __name__ == '__main__':
    unittest.main()
```

3. **Test adding a new vehicle.**

```python
import unittest

from entity.Vehicle import Vehicle
from serviceanddatabase.VehicleService import VehicleService
from serviceanddatabase.DatabaseConnect import DatabaseConnect


class TestAddNewVehicle(unittest.TestCase):
    def setUp(self):
        self.db_context = DatabaseConnect(database="carconnect")
        self.db_context.connect()
        self.vehicle_service = VehicleService(self.db_context)

    def test_add_new_vehicle(self):...


if __name__ == '__main__':
    unittest.main()
```

4. **Test updating vehicle details.**

```python
import unittest

from entity.Vehicle import Vehicle
from serviceanddatabase.DatabaseConnect import DatabaseConnect
from serviceanddatabase.VehicleService import VehicleService


class TestUpdateVehicleDetails(unittest.TestCase):
    def setUp(self):
        self.db_context = DatabaseConnect(database="carconnect")
        self.db_context.connect()
        self.vehicle_service = VehicleService(self.db_context)

    def test_update_vehicle_details(self):
        updated_vehicle_data = {...}
        self.vehicle_service.update_vehicle(updated_vehicle_data)

        updated_vehicle_result = self.vehicle_service.get_vehicle_by_id(updated_vehicle_data['VehicleID'])
        updated_vehicle = Vehicle(*updated_vehicle_result[0])
```

```python
        self.vehicle_service.update_vehicle(updated_vehicle_data)

        updated_vehicle_result = self.vehicle_service.get_vehicle_by_id(updated_vehicle_data['VehicleID'])
        updated_vehicle = Vehicle(*updated_vehicle_result[0])

        # Check if the details have been updated correctly
        self.assertEqual(updated_vehicle.model, second: 'Updated Model')
        self.assertEqual(updated_vehicle.make, second: 'Updated Make')
        self.assertEqual(updated_vehicle.year, second: 2023)
        self.assertEqual(updated_vehicle.color, second: 'Updated Color')
        self.assertEqual(updated_vehicle.registration_number, second: 'Updated123')
        self.assertFalse(updated_vehicle.availability)
        self.assertEqual(updated_vehicle.daily_rate, second: 60.0)


if __name__ == '__main__':
    unittest.main()
```

5. **Test getting a list of available vehicles.**

```python
import unittest

from entity.Vehicle import Vehicle
from serviceanddatabase.VehicleService import VehicleService
from serviceanddatabase.DatabaseConnect import DatabaseConnect


class TestGetAvailableVehicles(unittest.TestCase):
    def setUp(self):
        db_context = DatabaseConnect(database="carconnect")
        db_context.connect()
        self.vehicle_service = VehicleService(db_context)

    def test_get_available_vehicles(self):
        test_vehicles = [...]

        for vehicle_data in test_vehicles:
            self.vehicle_service.add_vehicle(vehicle_data)

        available_vehicles_result = self.vehicle_service.get_available_vehicles()
```

6. **Test getting a list of all vehicles.**

```
class TestGetAllVehicles(unittest.TestCase):
    def setUp(self):
        db_context = DatabaseConnect(database="carconnect")
        db_context.connect()
        self.vehicle_service = VehicleService(db_context)

    def test_get_all_vehicles(self):
        test_vehicles = [
            {...},
            {...},
            {...},
        ]

        for vehicle_data in test_vehicles:
            self.vehicle_service.add_vehicle(vehicle_data)

        all_vehicles = self.vehicle_service.get_all_vehicles()
        self.assertGreaterEqual(len(all_vehicles), len(test_vehicles))
```

****