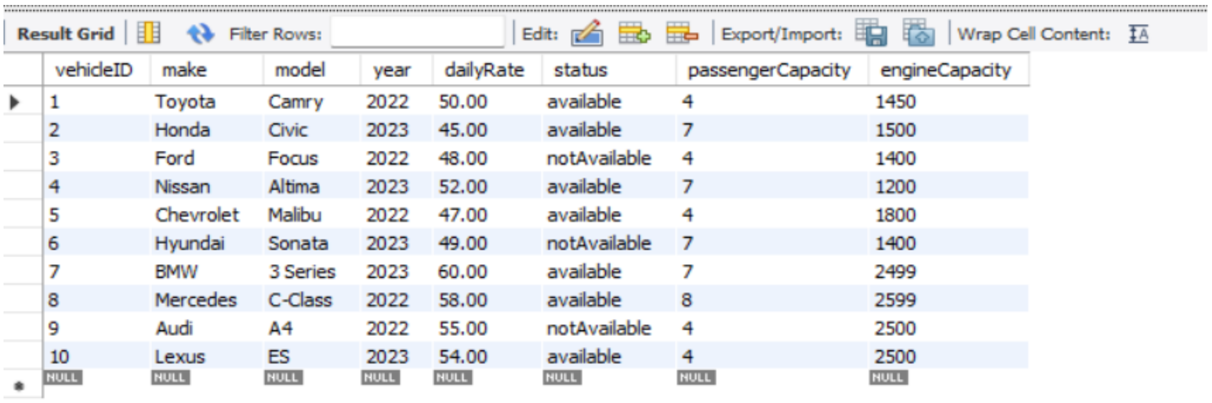
CASE STUDY – CAR RENTAL SYSTEM

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

1. **Vehicle Table:**

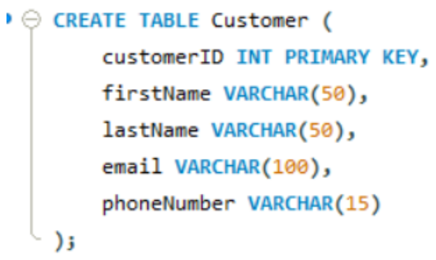
```sql
CREATE TABLE Vehicle (
    vehicleID INT PRIMARY KEY,
    make VARCHAR(50),
    model VARCHAR(50),
    year INT,
    dailyRate DECIMAL(10, 2),
    status ENUM('available', 'notAvailable'),
    passengerCapacity INT,
    engineCapacity INT
);
```

```sql
INSERT INTO Vehicle (vehicleID, make, model, year, dailyRate, status, passengerCapacity, engineCapacity)
VALUES
(1, 'Toyota', 'Camry', 2022, 50.00, 'available', 4, 1450),
(2, 'Honda', 'Civic', 2023, 45.00, 'available', 7, 1500),
(3, 'Ford', 'Focus', 2022, 48.00, 'notAvailable', 4, 1400),
(4, 'Nissan', 'Altima', 2023, 52.00, 'available', 7, 1200),
(5, 'Chevrolet', 'Malibu', 2022, 47.00, 'available', 4, 1800),
(6, 'Hyundai', 'Sonata', 2023, 49.00, 'notAvailable', 7, 1400),
(7, 'BMW', '3 Series', 2023, 60.00, 'available', 7, 2499),
(8, 'Mercedes', 'C-Class', 2022, 58.00, 'available', 8, 2599),
(9, 'Audi', 'A4', 2022, 55.00, 'notAvailable', 4, 2500),
(10, 'Lexus', 'ES', 2023, 54.00, 'available', 4, 2500);
```

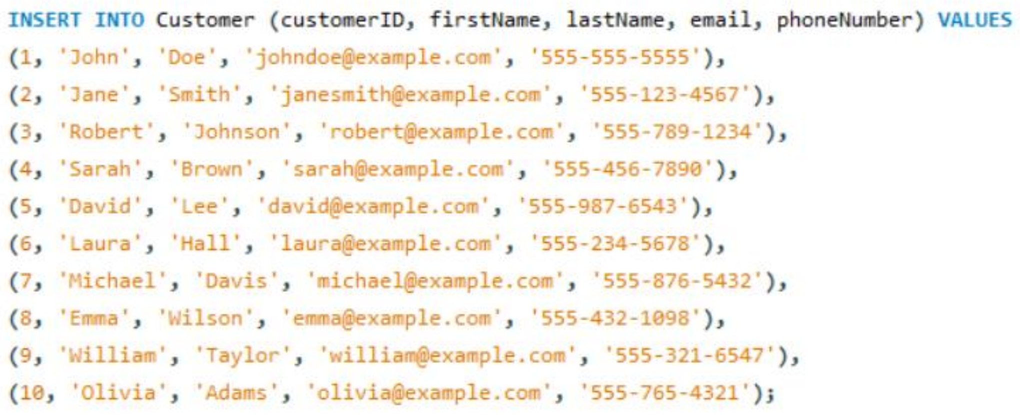


vehicleID	make	model	year	dailyRate	status	passengerCapacity	engineCapacity
1	Toyota	Camry	2022	50.00	available	4	1450
2	Honda	Civic	2023	45.00	available	7	1500
3	Ford	Focus	2022	48.00	notAvailable	4	1400
4	Nissan	Altima	2023	52.00	available	7	1200
5	Chevrolet	Malibu	2022	47.00	available	4	1800
6	Hyundai	Sonata	2023	49.00	notAvailable	7	1400
7	BMW	3 Series	2023	60.00	available	7	2499
8	Mercedes	C-Class	2022	58.00	available	8	2599
9	Audi	A4	2022	55.00	notAvailable	4	2500
10	Lexus	ES	2023	54.00	available	4	2500
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

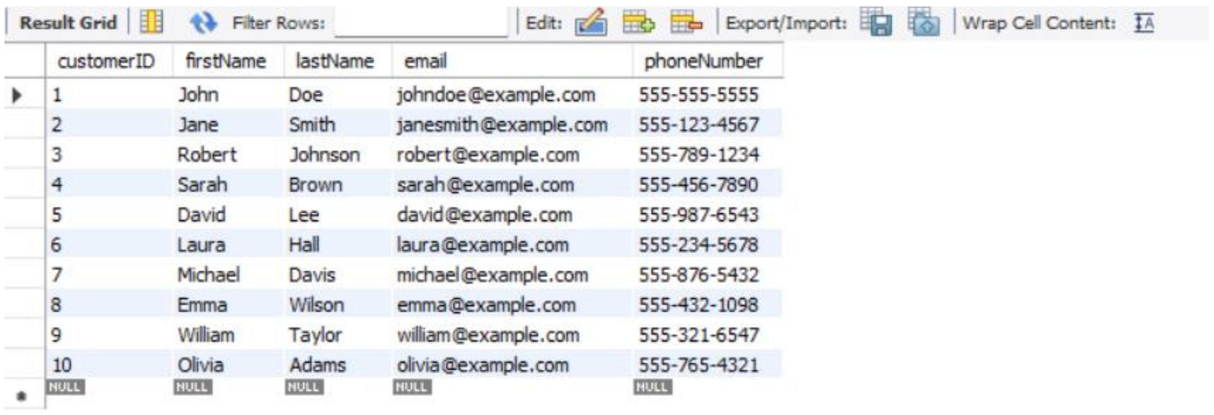
2. Customer Table:



```sql
CREATE TABLE Customer (
    customerID INT PRIMARY KEY,
    firstName VARCHAR(50),
    lastName VARCHAR(50),
    email VARCHAR(100),
    phoneNumber VARCHAR(15)
);
```



```sql
INSERT INTO Customer (customerID, firstName, lastName, email, phoneNumber) VALUES
(1, 'John', 'Doe', 'johndoe@example.com', '555-555-5555'),
(2, 'Jane', 'Smith', 'janesmith@example.com', '555-123-4567'),
(3, 'Robert', 'Johnson', 'robert@example.com', '555-789-1234'),
(4, 'Sarah', 'Brown', 'sarah@example.com', '555-456-7890'),
(5, 'David', 'Lee', 'david@example.com', '555-987-6543'),
(6, 'Laura', 'Hall', 'laura@example.com', '555-234-5678'),
(7, 'Michael', 'Davis', 'michael@example.com', '555-876-5432'),
(8, 'Emma', 'Wilson', 'emma@example.com', '555-432-1098'),
(9, 'William', 'Taylor', 'william@example.com', '555-321-6547'),
(10, 'Olivia', 'Adams', 'olivia@example.com', '555-765-4321');
```

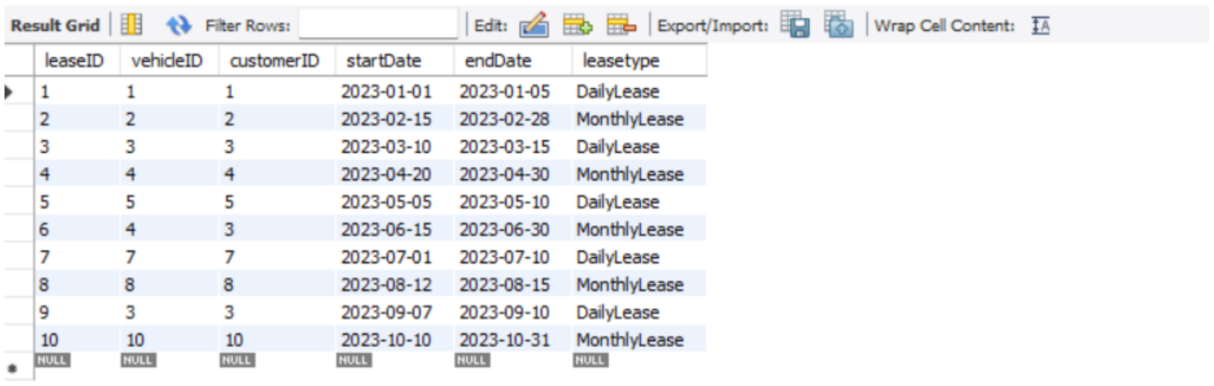


customerID	firstName	lastName	email	phoneNumber
1	John	Doe	johndoe@example.com	555-555-5555
2	Jane	Smith	janesmith@example.com	555-123-4567
3	Robert	Johnson	robert@example.com	555-789-1234
4	Sarah	Brown	sarah@example.com	555-456-7890
5	David	Lee	david@example.com	555-987-6543
6	Laura	Hall	laura@example.com	555-234-5678
7	Michael	Davis	michael@example.com	555-876-5432
8	Emma	Wilson	emma@example.com	555-432-1098
9	William	Taylor	william@example.com	555-321-6547
10	Olivia	Adams	olivia@example.com	555-765-4321
NULL	NULL	NULL	NULL	NULL

3. **Lease Table:**

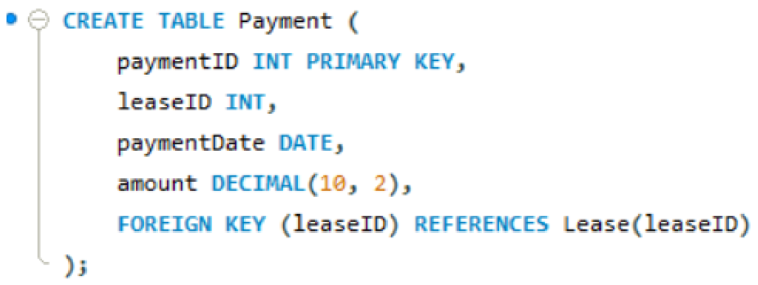
```sql
CREATE TABLE Lease (
    leaseID INT PRIMARY KEY,
    vehicleID INT,
    customerID INT,
    startDate DATE,
    endDate DATE,
    leasetype ENUM('DailyLease', 'MonthlyLease'),
    FOREIGN KEY (vehicleID) REFERENCES Vehicle(vehicleID),
    FOREIGN KEY (customerID) REFERENCES Customer(customerID)
);
```

```sql
INSERT INTO lease (leaseID, vehicleId, customerID, startDate, endDate, leaseType)
VALUES
(1, 1, 1, '2023-01-01', '2023-01-05', 'DailyLease'),
(2, 2, 2, '2023-02-15', '2023-02-28', 'MonthlyLease'),
(3, 3, 3, '2023-03-10', '2023-03-15', 'DailyLease'),
(4, 4, 4, '2023-04-20', '2023-04-30', 'MonthlyLease'),
(5, 5, 5, '2023-05-05', '2023-05-10', 'DailyLease'),
(6, 4, 3, '2023-06-15', '2023-06-30', 'MonthlyLease'),
(7, 7, 7, '2023-07-01', '2023-07-10', 'DailyLease'),
(8, 8, 8, '2023-08-12', '2023-08-15', 'MonthlyLease'),
(9, 3, 3, '2023-09-07', '2023-09-10', 'DailyLease'),
(10, 10, 10, '2023-10-10', '2023-10-31', 'MonthlyLease');
```

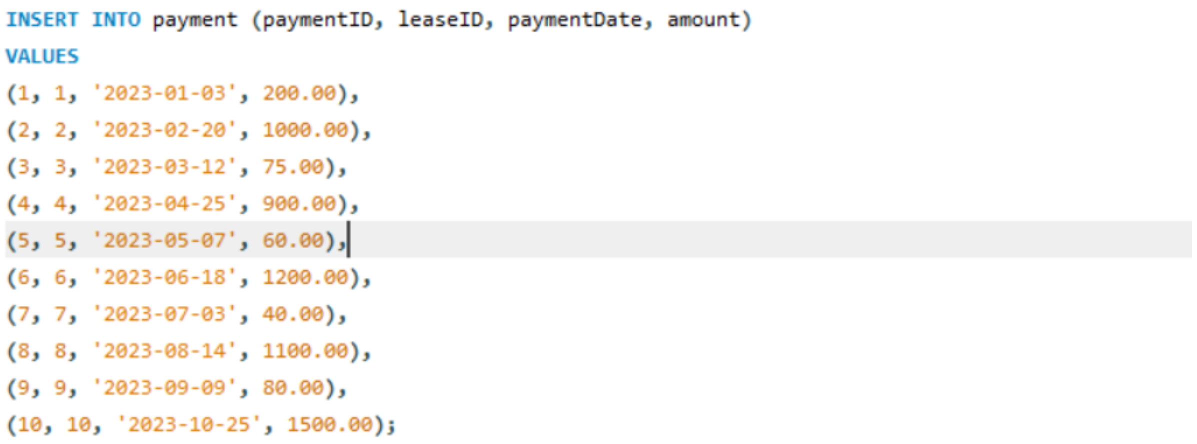


leaseID	vehicleID	customerID	startDate	endDate	leasetype
1	1	1	2023-01-01	2023-01-05	DailyLease
2	2	2	2023-02-15	2023-02-28	MonthlyLease
3	3	3	2023-03-10	2023-03-15	DailyLease
4	4	4	2023-04-20	2023-04-30	MonthlyLease
5	5	5	2023-05-05	2023-05-10	DailyLease
6	4	3	2023-06-15	2023-06-30	MonthlyLease
7	7	7	2023-07-01	2023-07-10	DailyLease
8	8	8	2023-08-12	2023-08-15	MonthlyLease
9	3	3	2023-09-07	2023-09-10	DailyLease
10	10	10	2023-10-10	2023-10-31	MonthlyLease
NULL	NULL	NULL	NULL	NULL	NULL

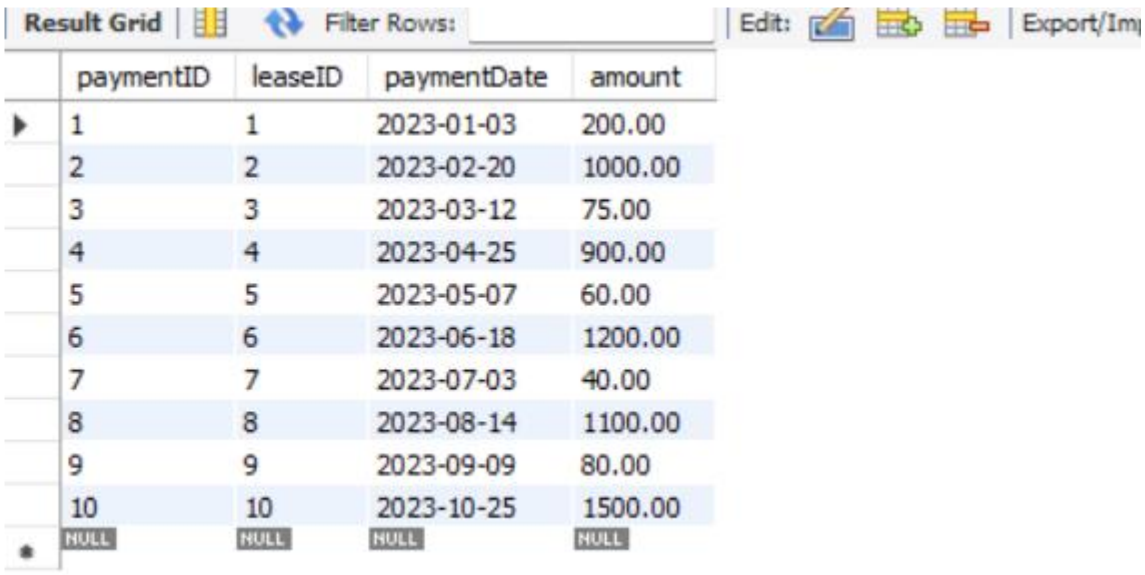
4. **Payment Table:**



```
CREATE TABLE Payment (
    paymentID INT PRIMARY KEY,
    leaseID INT,
    paymentDate DATE,
    amount DECIMAL(10, 2),
    FOREIGN KEY (leaseID) REFERENCES Lease(leaseID)
);
```

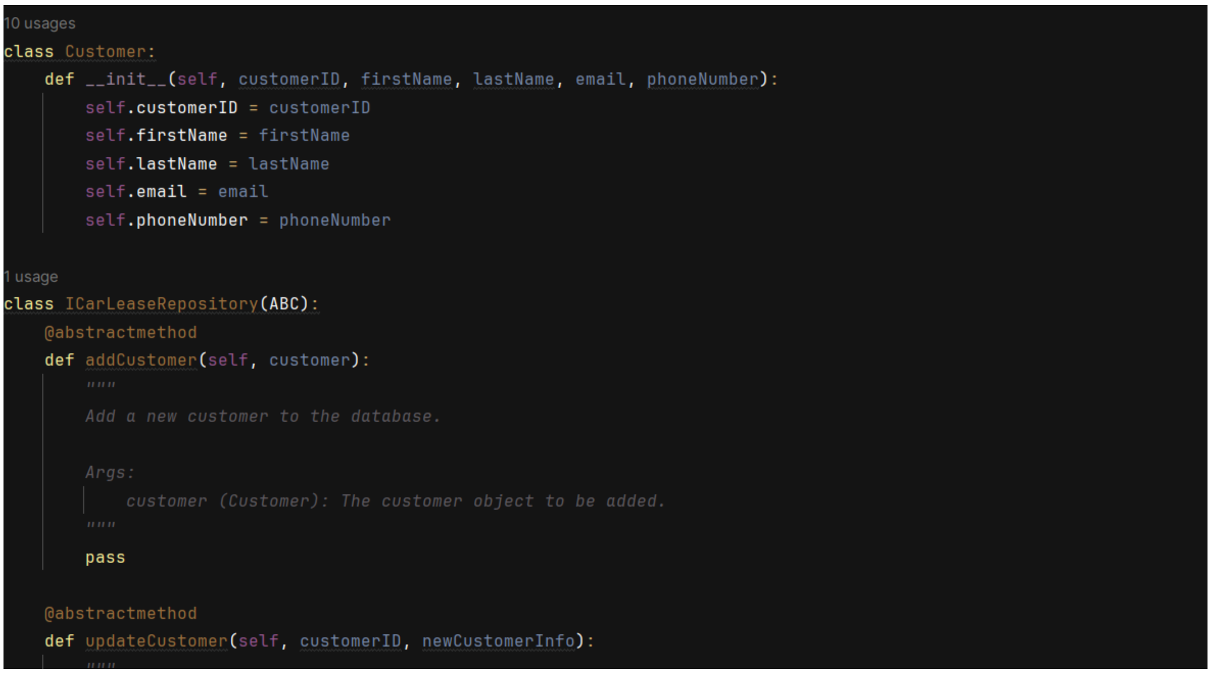


```
INSERT INTO payment (paymentID, leaseID, paymentDate, amount)
VALUES
(1, 1, '2023-01-03', 200.00),
(2, 2, '2023-02-20', 1000.00),
(3, 3, '2023-03-12', 75.00),
(4, 4, '2023-04-25', 900.00),
(5, 5, '2023-05-07', 60.00),
(6, 6, '2023-06-18', 1200.00),
(7, 7, '2023-07-03', 40.00),
(8, 8, '2023-08-14', 1100.00),
(9, 9, '2023-09-09', 80.00),
(10, 10, '2023-10-25', 1500.00);
```

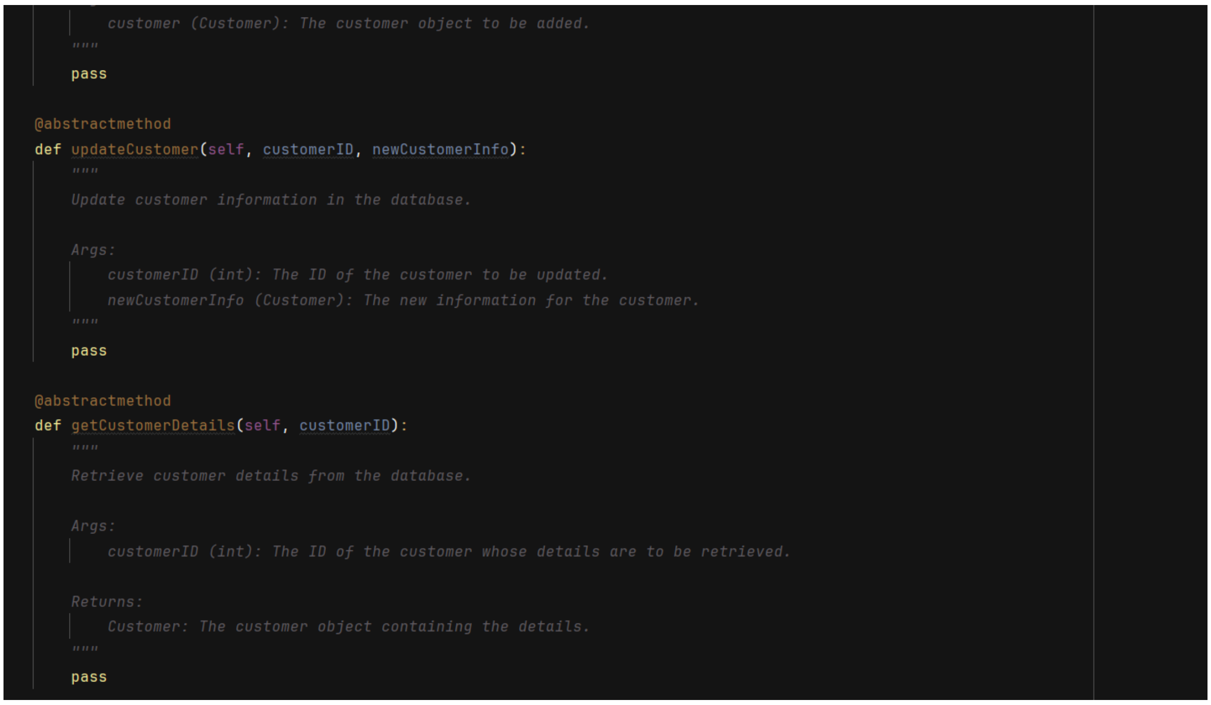


paymentID	leaseID	paymentDate	amount
1	1	2023-01-03	200.00
2	2	2023-02-20	1000.00
3	3	2023-03-12	75.00
4	4	2023-04-25	900.00
5	5	2023-05-07	60.00
6	6	2023-06-18	1200.00
7	7	2023-07-03	40.00
8	8	2023-08-14	1100.00
9	9	2023-09-09	80.00
10	10	2023-10-25	1500.00
NULL	NULL	NULL	NULL

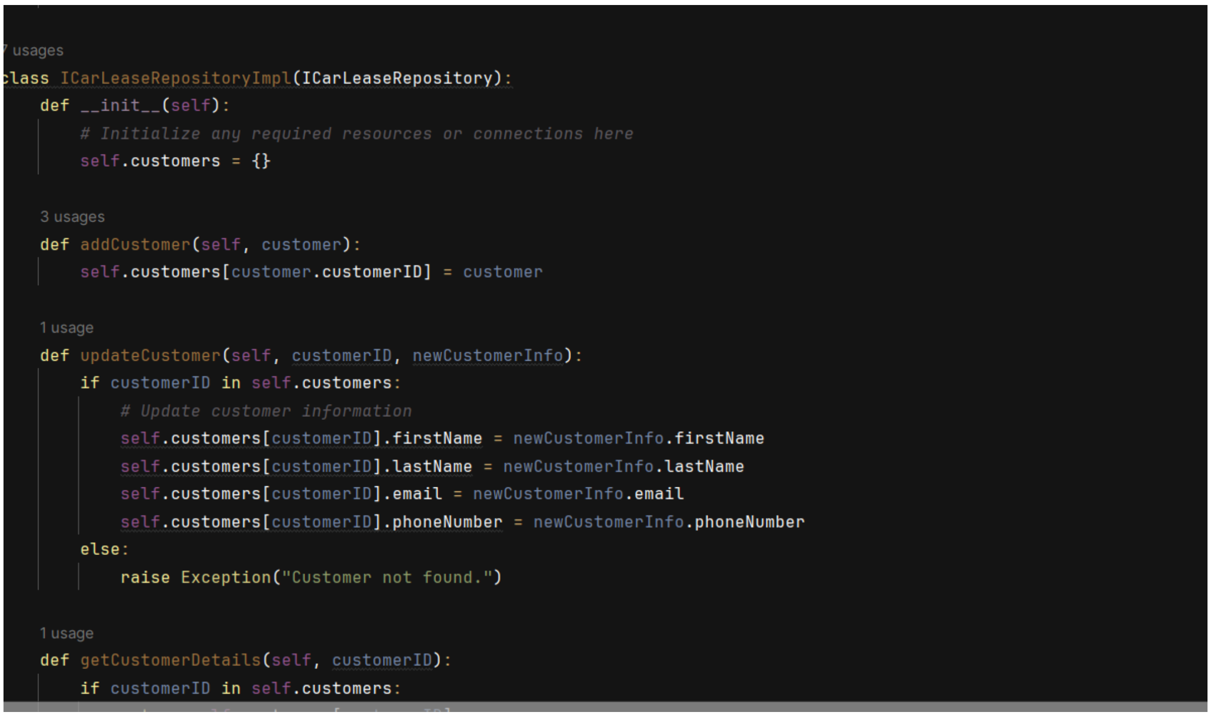
1. **Customer Management**
    - **Add new customers, Update customer information, Retrieve customer details.**



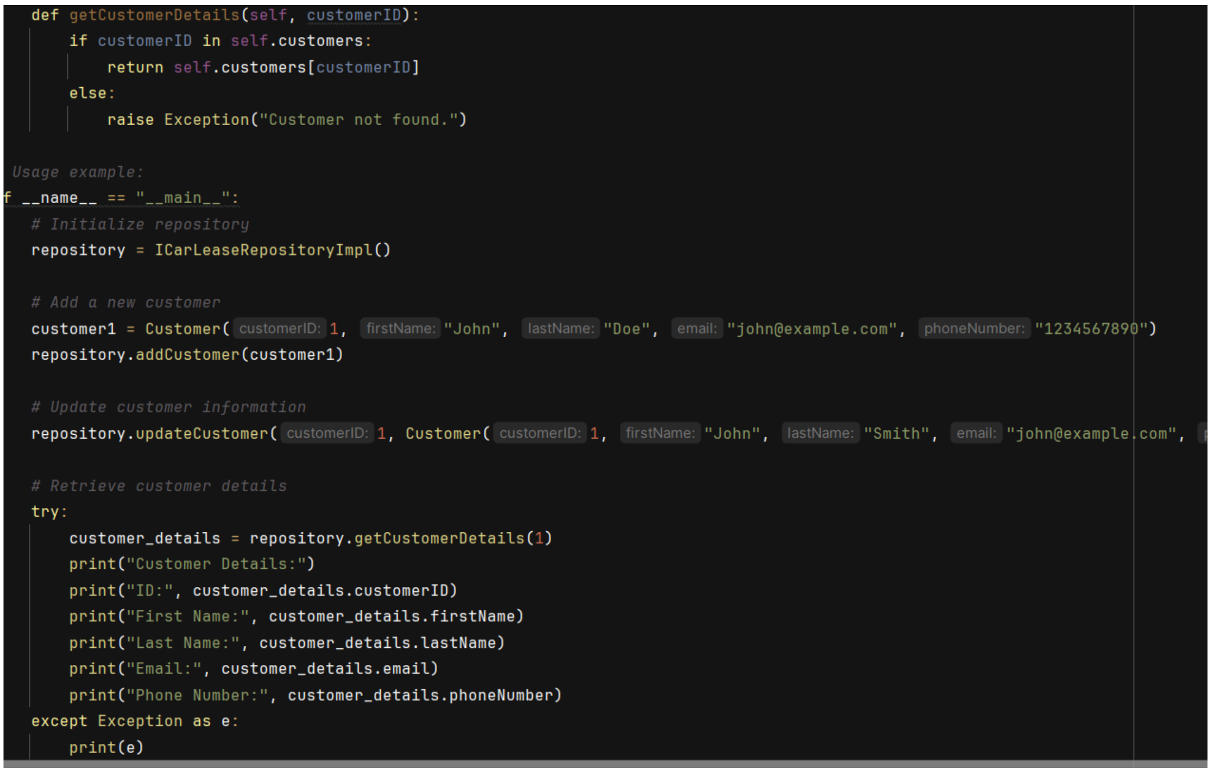
```
10 usages
class Customer:
    def __init__(self, customerID, firstName, lastName, email, phoneNumber):
        self.customerID = customerID
        self.firstName = firstName
        self.lastName = lastName
        self.email = email
        self.phoneNumber = phoneNumber

1 usage
class ICarLeaseRepository(ABC):
    @abstractmethod
    def addCustomer(self, customer):
        """
        Add a new customer to the database.

        Args:
            customer (Customer): The customer object to be added.
        """
        pass

    @abstractmethod
    def updateCustomer(self, customerID, newCustomerInfo):
        """
```



```
            customer (Customer): The customer object to be added.
        """
        pass

    @abstractmethod
    def updateCustomer(self, customerID, newCustomerInfo):
        """
        Update customer information in the database.

        Args:
            customerID (int): The ID of the customer to be updated.
            newCustomerInfo (Customer): The new information for the customer.
        """
        pass

    @abstractmethod
    def getCustomerDetails(self, customerID):
        """
        Retrieve customer details from the database.

        Args:
            customerID (int): The ID of the customer whose details are to be retrieved.

        Returns:
            Customer: The customer object containing the details.
        """
        pass
```

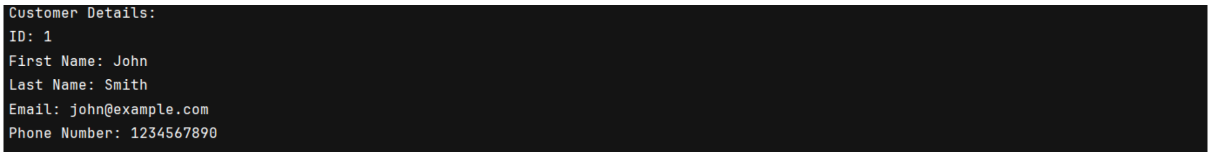


```
7 usages
class ICarLeaseRepositoryImpl(ICarLeaseRepository):
    def __init__(self):
        # Initialize any required resources or connections here
        self.customers = {}

    3 usages
    def addCustomer(self, customer):
        self.customers[customer.customerID] = customer

    1 usage
    def updateCustomer(self, customerID, newCustomerInfo):
        if customerID in self.customers:
            # Update customer information
            self.customers[customerID].firstName = newCustomerInfo.firstName
            self.customers[customerID].lastName = newCustomerInfo.lastName
            self.customers[customerID].email = newCustomerInfo.email
            self.customers[customerID].phoneNumber = newCustomerInfo.phoneNumber
        else:
            raise Exception("Customer not found.")

    1 usage
    def getCustomerDetails(self, customerID):
        if customerID in self.customers:
```



```
    def getCustomerDetails(self, customerID):
        if customerID in self.customers:
            return self.customers[customerID]
        else:
            raise Exception("Customer not found.")

 Usage example:
f __name__ == "__main__":
    # Initialize repository
    repository = ICarLeaseRepositoryImpl()

    # Add a new customer
    customer1 = Customer( customerID: 1,  firstName: "John",  lastName: "Doe",  email: "john@example.com",  phoneNumber: "1234567890")
    repository.addCustomer(customer1)

    # Update customer information
    repository.updateCustomer( customerID: 1, Customer( customerID: 1,  firstName: "John",  lastName: "Smith",  email: "john@example.com",

    # Retrieve customer details
    try:
        customer_details = repository.getCustomerDetails(1)
        print("Customer Details:")
        print("ID:", customer_details.customerID)
        print("First Name:", customer_details.firstName)
        print("Last Name:", customer_details.lastName)
        print("Email:", customer_details.email)
        print("Phone Number:", customer_details.phoneNumber)
    except Exception as e:
        print(e)
```

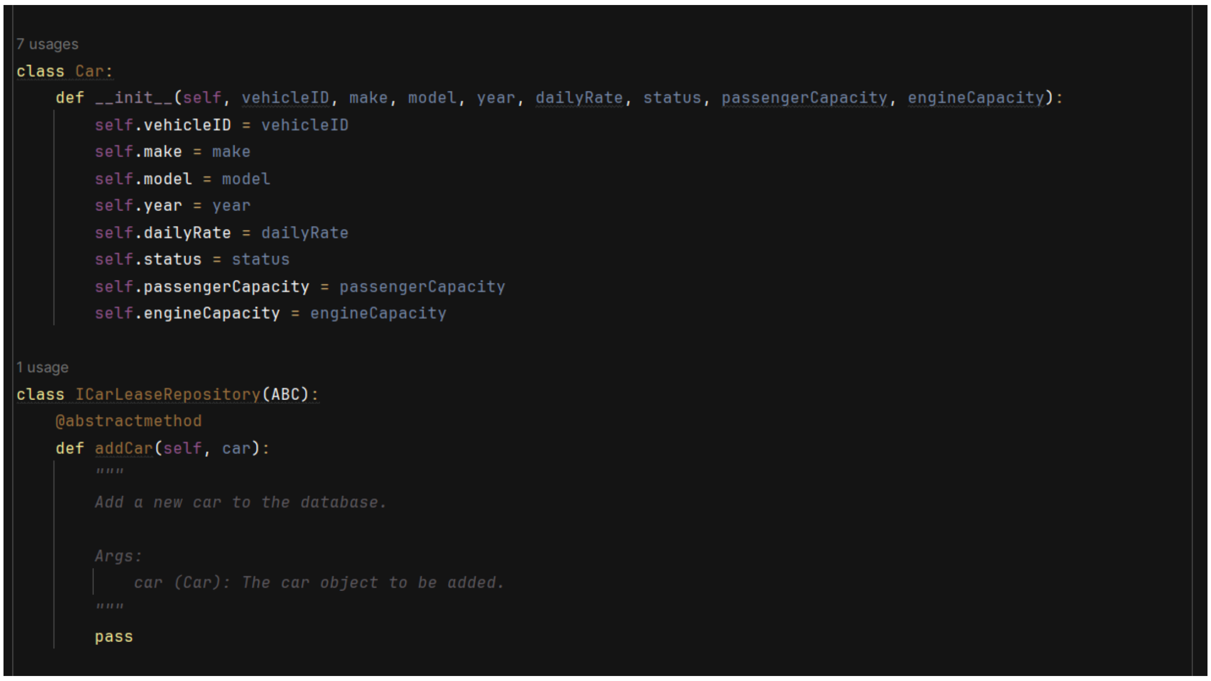
Output:



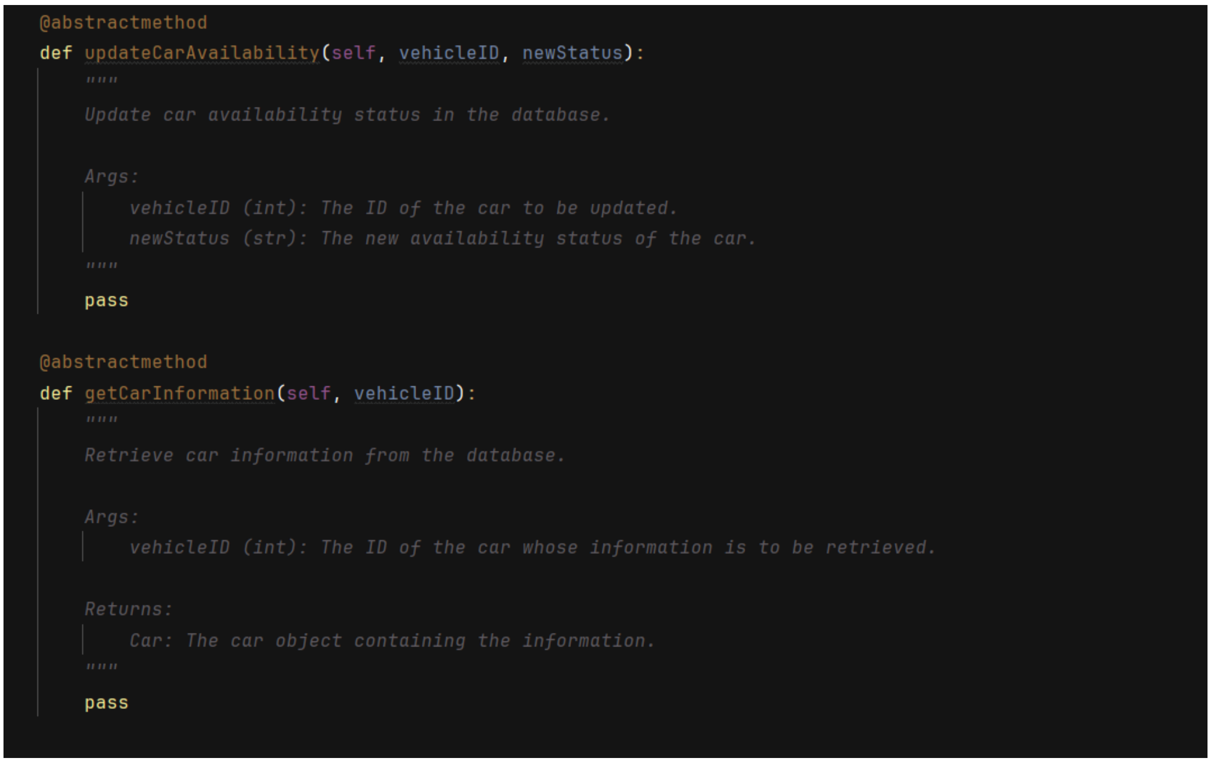
```
Customer Details:
ID: 1
First Name: John
Last Name: Smith
Email: john@example.com
Phone Number: 1234567890
```

2. **Car Management:**

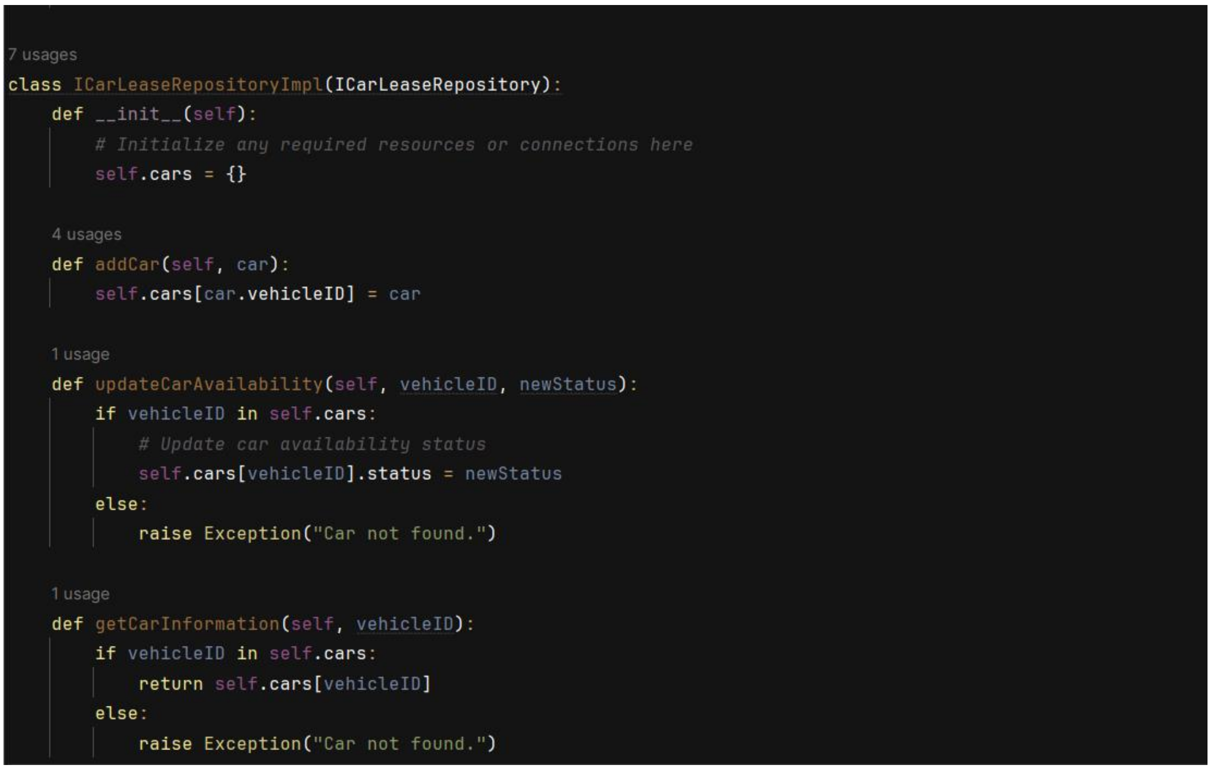
- **Add new cars to the system, Update car availability, Retrieve car information.**



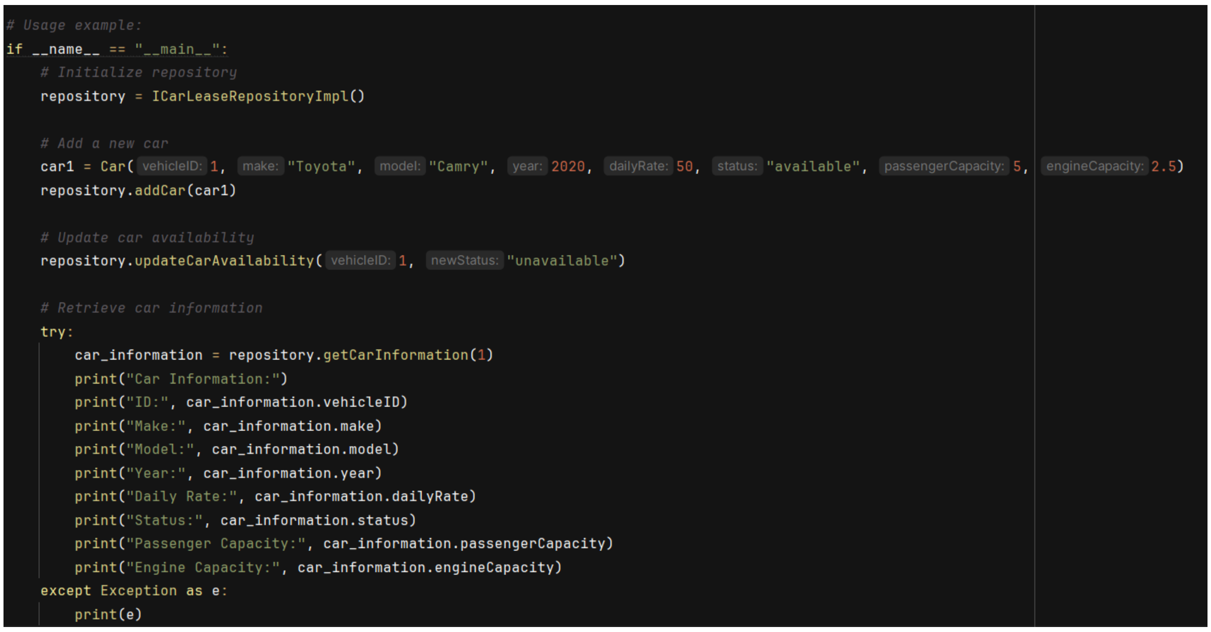
```
7 usages
class Car:
    def __init__(self, vehicleID, make, model, year, dailyRate, status, passengerCapacity, engineCapacity):
        self.vehicleID = vehicleID
        self.make = make
        self.model = model
        self.year = year
        self.dailyRate = dailyRate
        self.status = status
        self.passengerCapacity = passengerCapacity
        self.engineCapacity = engineCapacity

1 usage
class ICarLeaseRepository(ABC):
    @abstractmethod
    def addCar(self, car):
        """
        Add a new car to the database.

        Args:
            car (Car): The car object to be added.
        """
        pass
```



```python
    @abstractmethod
    def updateCarAvailability(self, vehicleID, newStatus):
        """
        Update car availability status in the database.

        Args:
            vehicleID (int): The ID of the car to be updated.
            newStatus (str): The new availability status of the car.
        """
        pass

    @abstractmethod
    def getCarInformation(self, vehicleID):
        """
        Retrieve car information from the database.

        Args:
            vehicleID (int): The ID of the car whose information is to be retrieved.

        Returns:
            Car: The car object containing the information.
        """
        pass
```

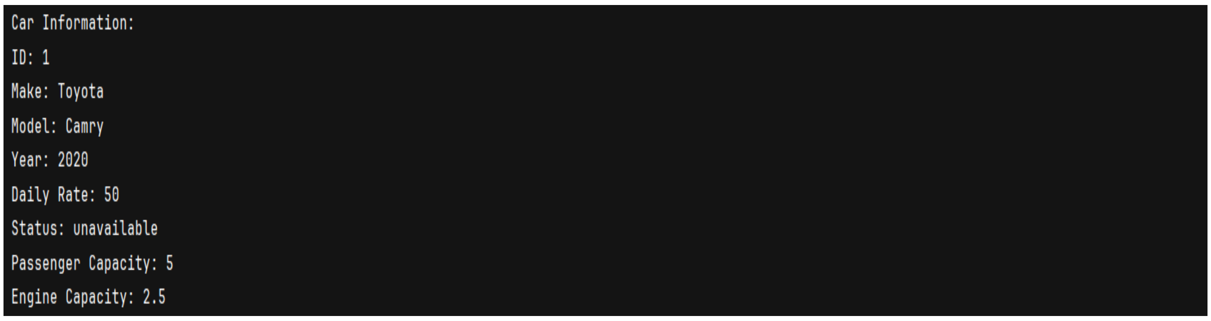


```python
7 usages
class ICarLeaseRepositoryImpl(ICarLeaseRepository):
    def __init__(self):
        # Initialize any required resources or connections here
        self.cars = {}

    4 usages
    def addCar(self, car):
        self.cars[car.vehicleID] = car

    1 usage
    def updateCarAvailability(self, vehicleID, newStatus):
        if vehicleID in self.cars:
            # Update car availability status
            self.cars[vehicleID].status = newStatus
        else:
            raise Exception("Car not found.")

    1 usage
    def getCarInformation(self, vehicleID):
        if vehicleID in self.cars:
            return self.cars[vehicleID]
        else:
            raise Exception("Car not found.")
```



```python
# Usage example:
if __name__ == "__main__":
    # Initialize repository
    repository = ICarLeaseRepositoryImpl()

    # Add a new car
    car1 = Car( vehicleID: 1,  make: "Toyota",  model: "Camry",  year: 2020,  dailyRate: 50,  status: "available",  passengerCapacity: 5,  engineCapacity: 2.5)
    repository.addCar(car1)

    # Update car availability
    repository.updateCarAvailability( vehicleID: 1,  newStatus: "unavailable")

    # Retrieve car information
    try:
        car_information = repository.getCarInformation(1)
        print("Car Information:")
        print("ID:", car_information.vehicleID)
        print("Make:", car_information.make)
        print("Model:", car_information.model)
        print("Year:", car_information.year)
        print("Daily Rate:", car_information.dailyRate)
        print("Status:", car_information.status)
        print("Passenger Capacity:", car_information.passengerCapacity)
        print("Engine Capacity:", car_information.engineCapacity)
    except Exception as e:
        print(e)
```

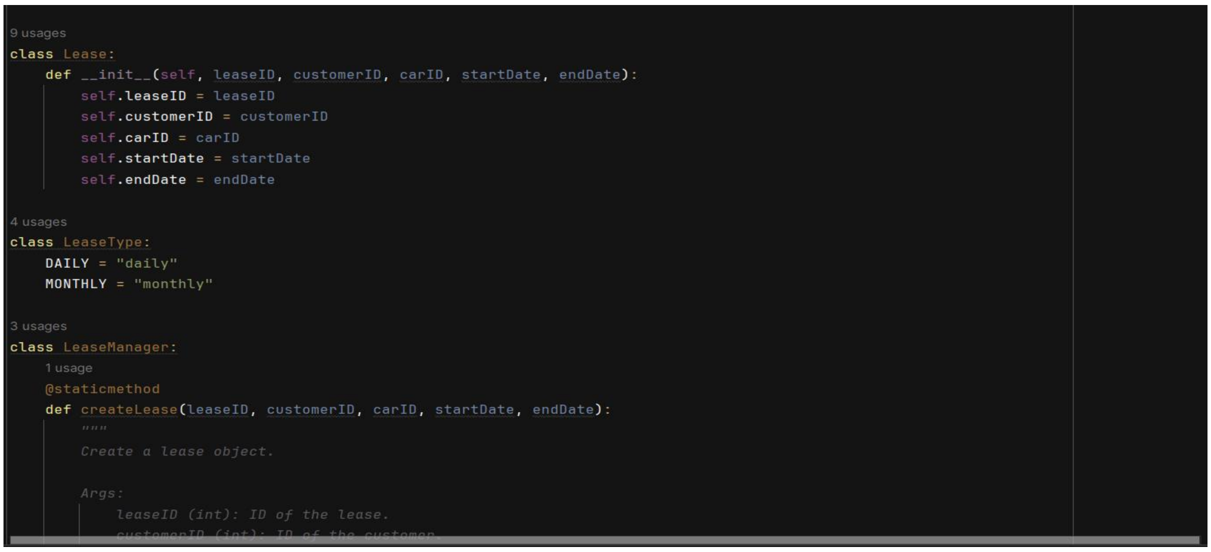
Output:



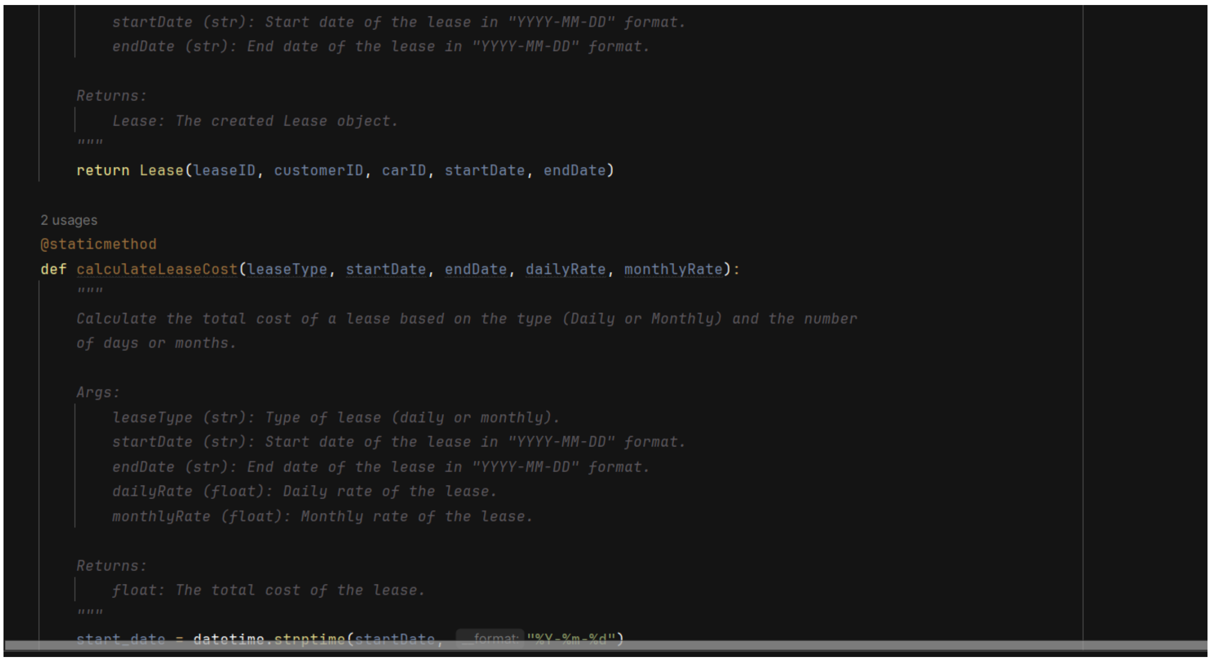
```
Car Information:
ID: 1
Make: Toyota
Model: Camry
Year: 2020
Daily Rate: 50
Status: unavailable
Passenger Capacity: 5
Engine Capacity: 2.5
```

3. Lease Management

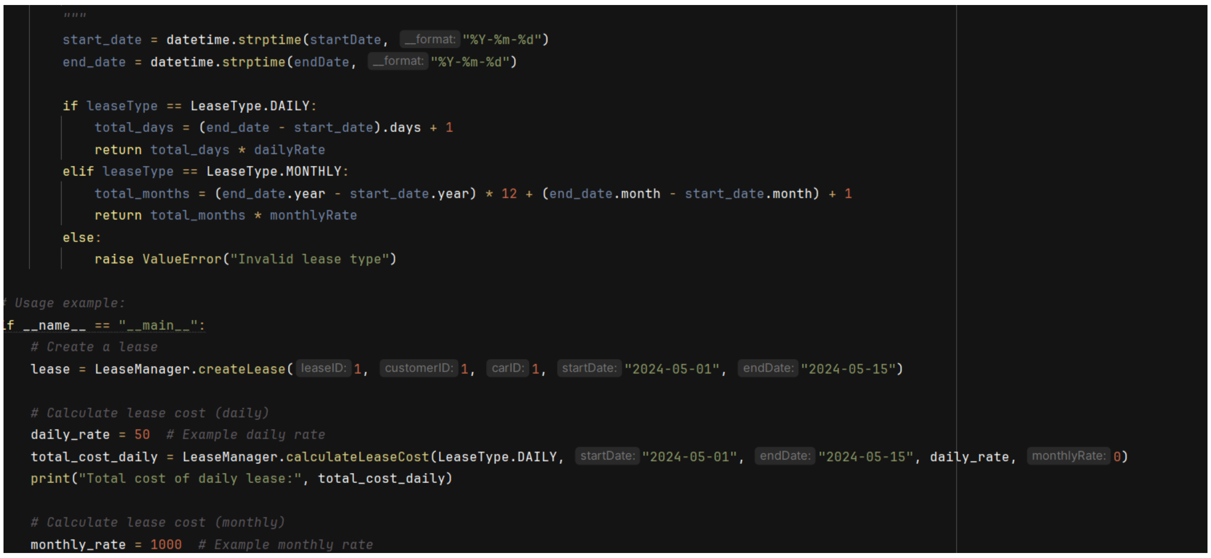
• Create daily or monthly leases for customers. • Calculate the total cost of a lease based on the type (Daily or Monthly) and the number of days or months.



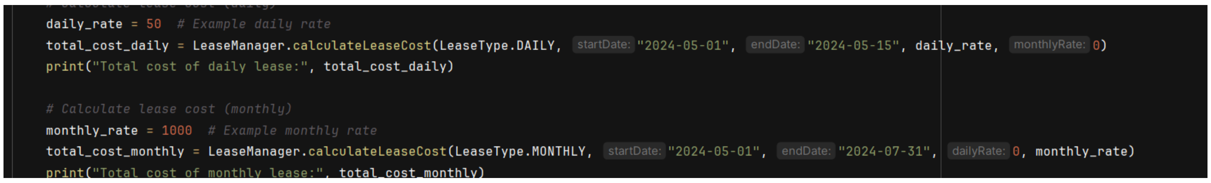
```
9 usages
class Lease:
    def __init__(self, leaseID, customerID, carID, startDate, endDate):
        self.leaseID = leaseID
        self.customerID = customerID
        self.carID = carID
        self.startDate = startDate
        self.endDate = endDate

4 usages
class LeaseType:
    DAILY = "daily"
    MONTHLY = "monthly"

3 usages
class LeaseManager:
    1 usage
    @staticmethod
    def createLease(leaseID, customerID, carID, startDate, endDate):
        """
        Create a lease object.

        Args:
            leaseID (int): ID of the lease.
            customerID (int): ID of the customer.
```



```
            startDate (str): Start date of the lease in "YYYY-MM-DD" format.
            endDate (str): End date of the lease in "YYYY-MM-DD" format.

        Returns:
            Lease: The created Lease object.
        """
        return Lease(leaseID, customerID, carID, startDate, endDate)

    2 usages
    @staticmethod
    def calculateLeaseCost(leaseType, startDate, endDate, dailyRate, monthlyRate):
        """
        Calculate the total cost of a lease based on the type (Daily or Monthly) and the number
        of days or months.

        Args:
            leaseType (str): Type of lease (daily or monthly).
            startDate (str): Start date of the lease in "YYYY-MM-DD" format.
            endDate (str): End date of the lease in "YYYY-MM-DD" format.
            dailyRate (float): Daily rate of the lease.
            monthlyRate (float): Monthly rate of the lease.

        Returns:
            float: The total cost of the lease.
        """
        start_date = datetime.strptime(startDate,  format: "%Y-%m-%d")
```

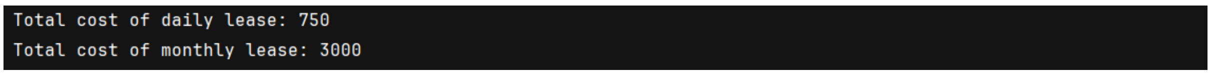


```
        """
        start_date = datetime.strptime(startDate, __format: "%Y-%m-%d")
        end_date = datetime.strptime(endDate, __format: "%Y-%m-%d")

        if leaseType == LeaseType.DAILY:
            total_days = (end_date - start_date).days + 1
            return total_days * dailyRate
        elif leaseType == LeaseType.MONTHLY:
            total_months = (end_date.year - start_date.year) * 12 + (end_date.month - start_date.month) + 1
            return total_months * monthlyRate
        else:
            raise ValueError("Invalid lease type")

# Usage example:
f __name__ == "__main__":
    # Create a lease
    lease = LeaseManager.createLease( leaseID: 1, customerID: 1, carID: 1, startDate: "2024-05-01", endDate: "2024-05-15")

    # Calculate lease cost (daily)
    daily_rate = 50  # Example daily rate
    total_cost_daily = LeaseManager.calculateLeaseCost(LeaseType.DAILY, startDate: "2024-05-01", endDate: "2024-05-15", daily_rate, monthlyRate: 0)
    print("Total cost of daily lease:", total_cost_daily)

    # Calculate lease cost (monthly)
    monthly_rate = 1000  # Example monthly rate
```



```
    daily_rate = 50  # Example daily rate
    total_cost_daily = LeaseManager.calculateLeaseCost(LeaseType.DAILY, startDate: "2024-05-01", endDate: "2024-05-15", daily_rate, monthlyRate: 0)
    print("Total cost of daily lease:", total_cost_daily)

    # Calculate lease cost (monthly)
    monthly_rate = 1000  # Example monthly rate
    total_cost_monthly = LeaseManager.calculateLeaseCost(LeaseType.MONTHLY, startDate: "2024-05-01", endDate: "2024-07-31", dailyRate: 0, monthly_rate)
    print("Total cost of monthly lease:", total_cost_monthly)
```

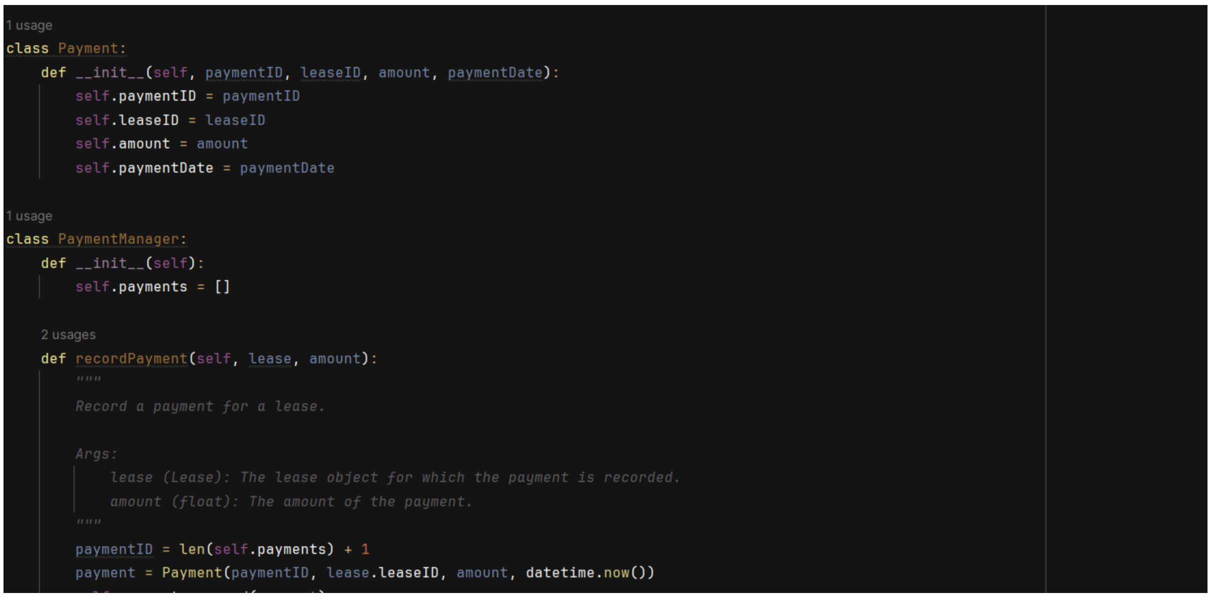
Output:



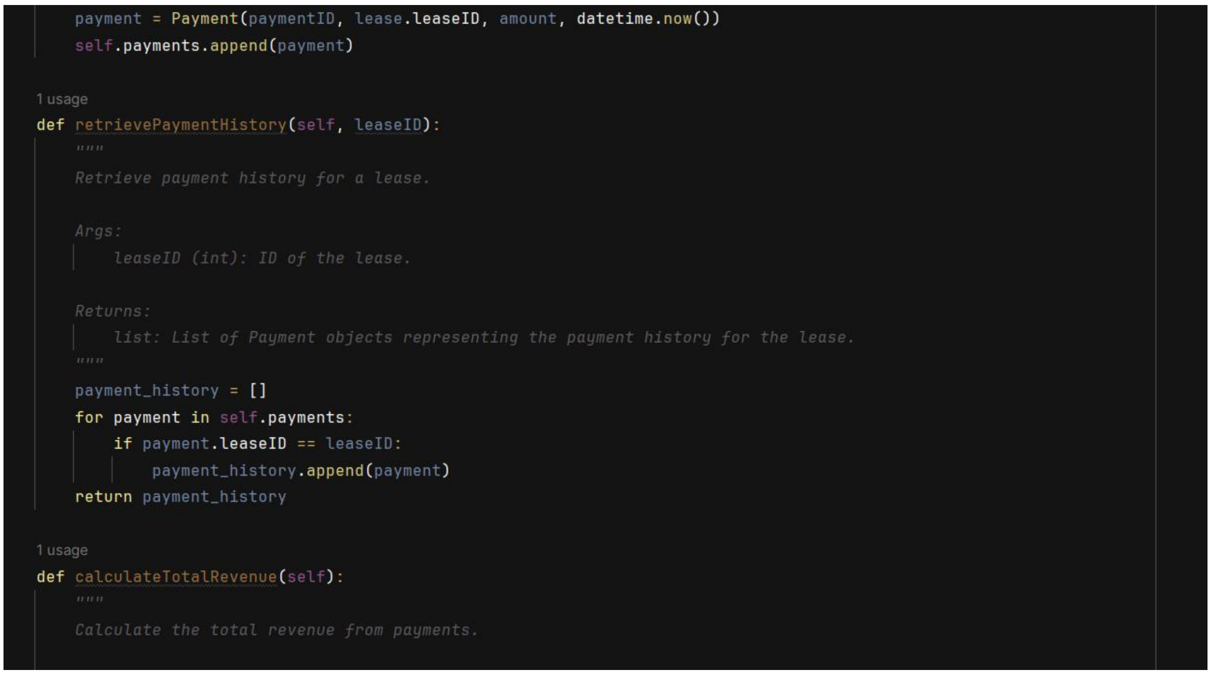
```
Total cost of daily lease: 750
Total cost of monthly lease: 3000
```

4. Payment Handling:

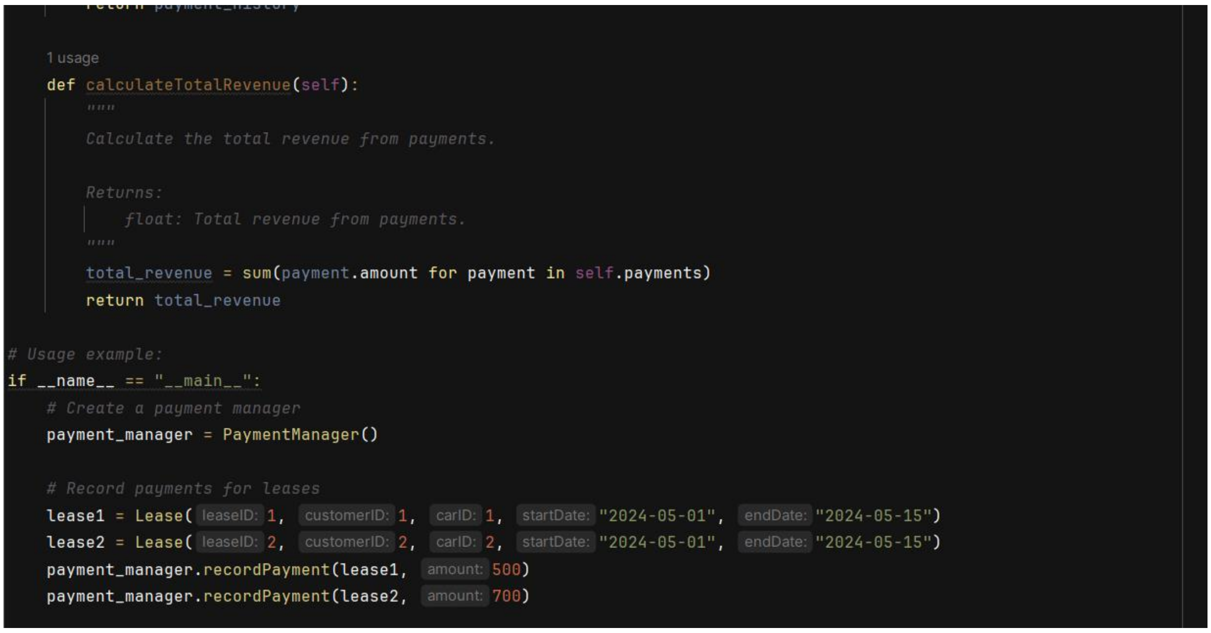
• Record payments for leases. • Retrieve payment history for a customer. • Calculate the total revenue from payments.



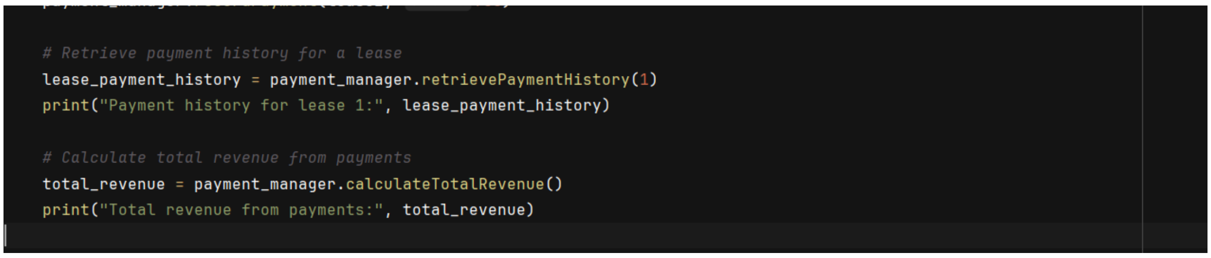
```
1 usage
class Payment:
    def __init__(self, paymentID, leaseID, amount, paymentDate):
        self.paymentID = paymentID
        self.leaseID = leaseID
        self.amount = amount
        self.paymentDate = paymentDate

1 usage
class PaymentManager:
    def __init__(self):
        self.payments = []

    2 usages
    def recordPayment(self, lease, amount):
        """
        Record a payment for a lease.

        Args:
            lease (Lease): The lease object for which the payment is recorded.
            amount (float): The amount of the payment.
        """
        paymentID = len(self.payments) + 1
        payment = Payment(paymentID, lease.leaseID, amount, datetime.now())
```



```
        payment = Payment(paymentID, lease.leaseID, amount, datetime.now())
        self.payments.append(payment)

    1 usage
    def retrievePaymentHistory(self, leaseID):
        """
        Retrieve payment history for a lease.

        Args:
            leaseID (int): ID of the lease.

        Returns:
            list: List of Payment objects representing the payment history for the lease.
        """
        payment_history = []
        for payment in self.payments:
            if payment.leaseID == leaseID:
                payment_history.append(payment)
        return payment_history

    1 usage
    def calculateTotalRevenue(self):
        """
        Calculate the total revenue from payments.
```

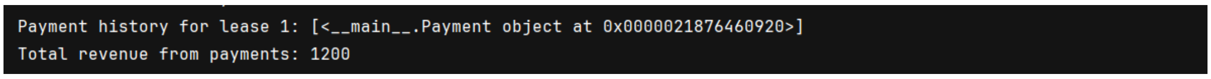


```
    1 usage
    def calculateTotalRevenue(self):
        """
        Calculate the total revenue from payments.

        Returns:
            float: Total revenue from payments.
        """
        total_revenue = sum(payment.amount for payment in self.payments)
        return total_revenue

# Usage example:
if __name__ == "__main__":
    # Create a payment manager
    payment_manager = PaymentManager()

    # Record payments for leases
    lease1 = Lease( leaseID: 1,  customerID: 1,  carID: 1,  startDate: "2024-05-01",  endDate: "2024-05-15")
    lease2 = Lease( leaseID: 2,  customerID: 2,  carID: 2,  startDate: "2024-05-01",  endDate: "2024-05-15")
    payment_manager.recordPayment(lease1,  amount: 500)
    payment_manager.recordPayment(lease2,  amount: 700)
```



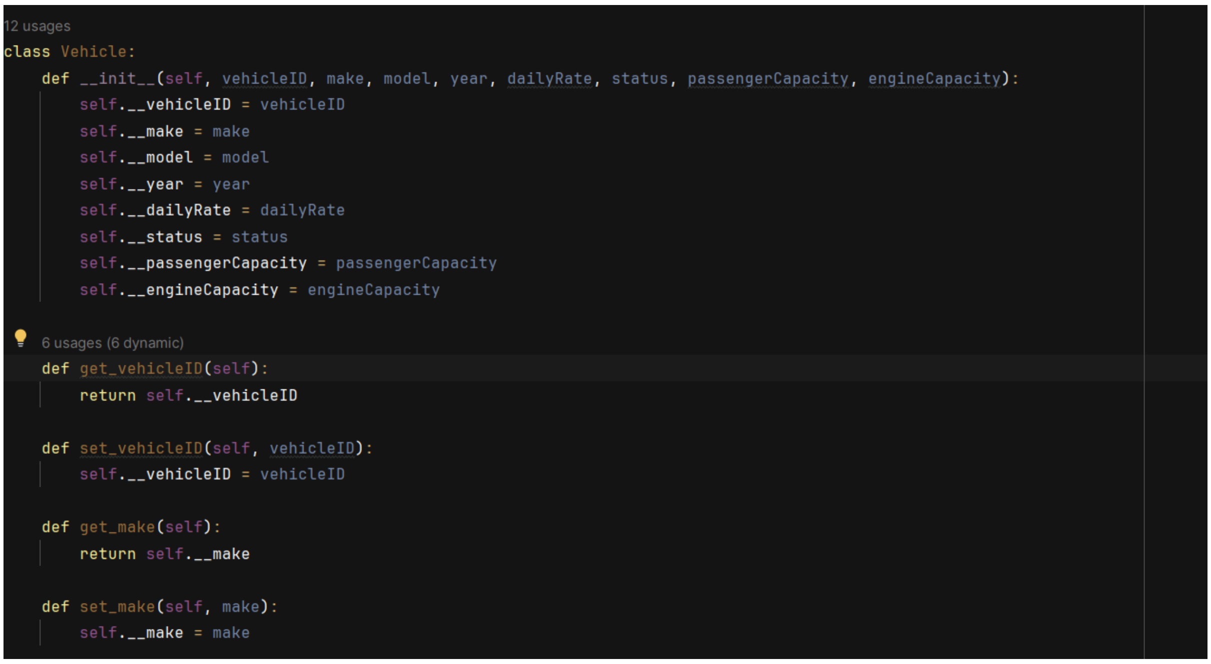
```
    # Retrieve payment history for a lease
    lease_payment_history = payment_manager.retrievePaymentHistory(1)
    print("Payment history for lease 1:", lease_payment_history)

    # Calculate total revenue from payments
    total_revenue = payment_manager.calculateTotalRevenue()
    print("Total revenue from payments:", total_revenue)
```

Output:

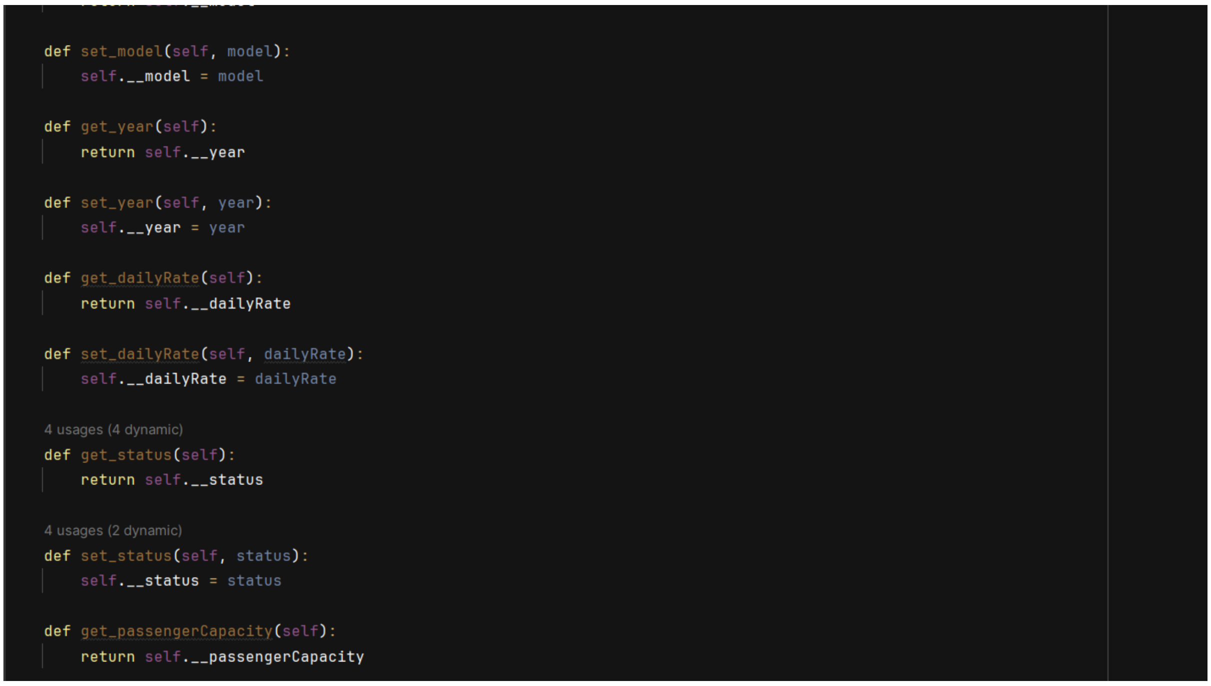


```
Payment history for lease 1: [<__main__.Payment object at 0x0000021876460920>]
Total revenue from payments: 1200
```

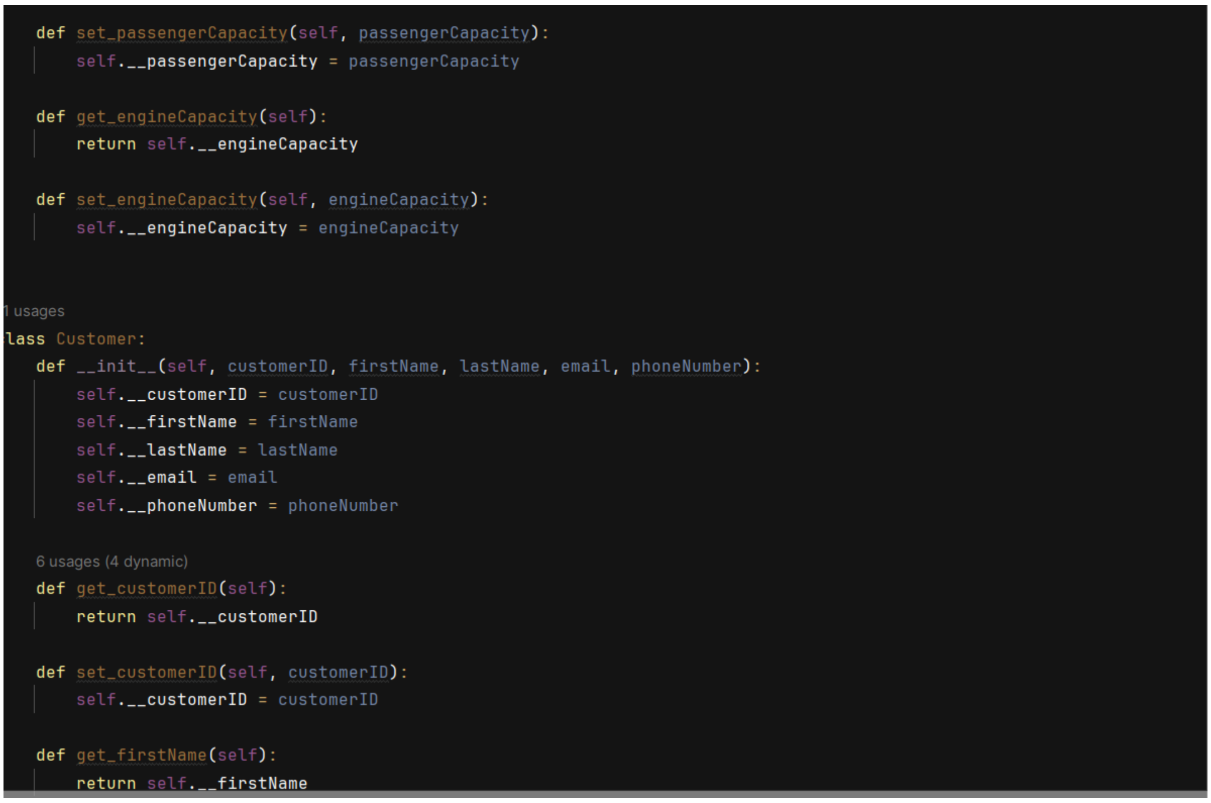
5.Create the model/entity classes corresponding to the schema within package entity with variables declared private, constructors(default and parametrized) and getters,setters)



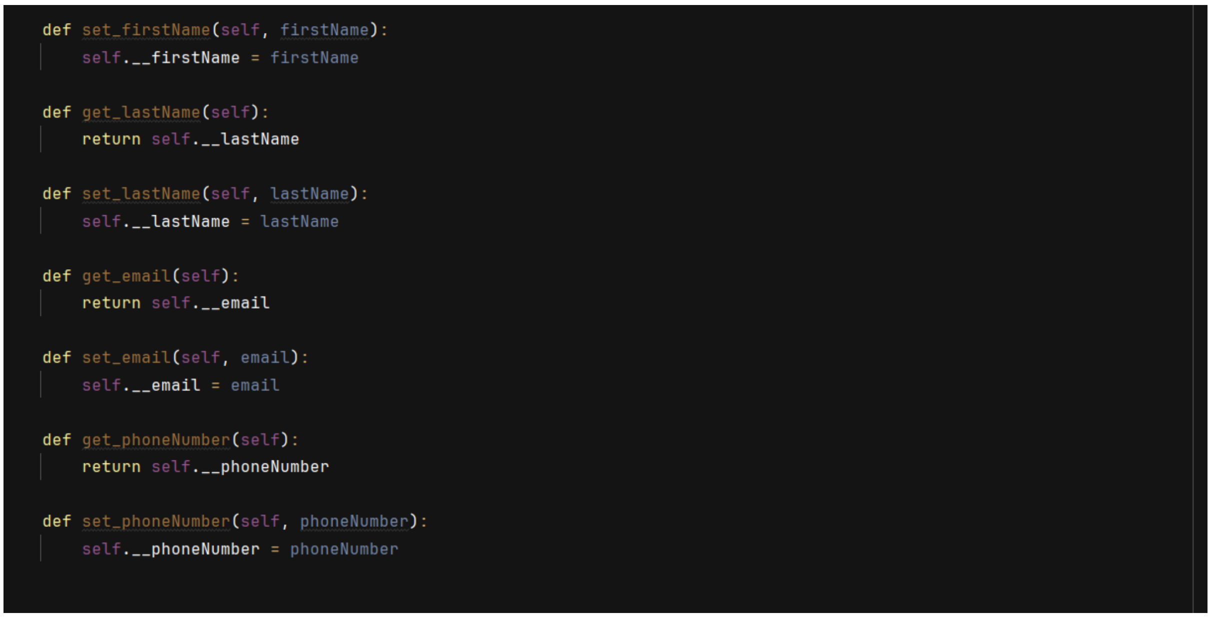
```
12 usages
class Vehicle:
    def __init__(self, vehicleID, make, model, year, dailyRate, status, passengerCapacity, engineCapacity):
        self.__vehicleID = vehicleID
        self.__make = make
        self.__model = model
        self.__year = year
        self.__dailyRate = dailyRate
        self.__status = status
        self.__passengerCapacity = passengerCapacity
        self.__engineCapacity = engineCapacity

    6 usages (6 dynamic)
    def get_vehicleID(self):
        return self.__vehicleID

    def set_vehicleID(self, vehicleID):
        self.__vehicleID = vehicleID

    def get_make(self):
        return self.__make

    def set_make(self, make):
        self.__make = make
```



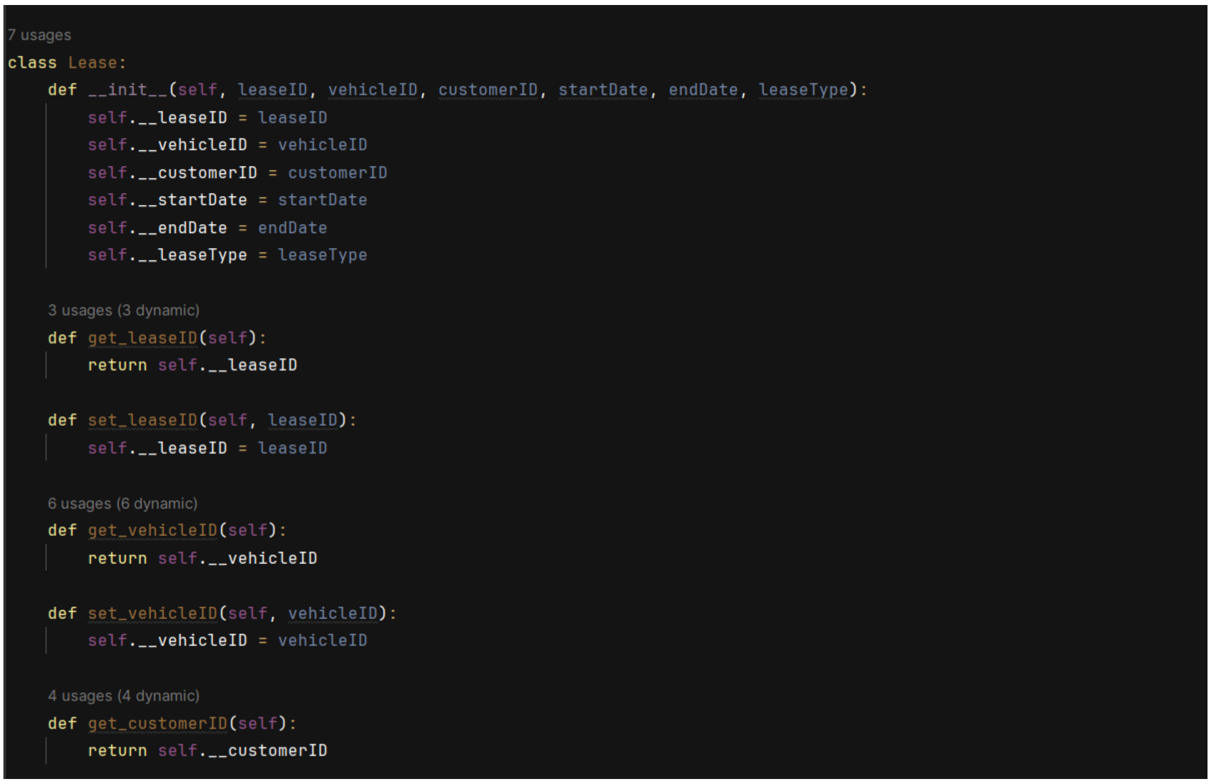
```
    def set_model(self, model):
        self.__model = model

    def get_year(self):
        return self.__year

    def set_year(self, year):
        self.__year = year

    def get_dailyRate(self):
        return self.__dailyRate

    def set_dailyRate(self, dailyRate):
        self.__dailyRate = dailyRate

    4 usages (4 dynamic)
    def get_status(self):
        return self.__status

    4 usages (2 dynamic)
    def set_status(self, status):
        self.__status = status

    def get_passengerCapacity(self):
        return self.__passengerCapacity
```



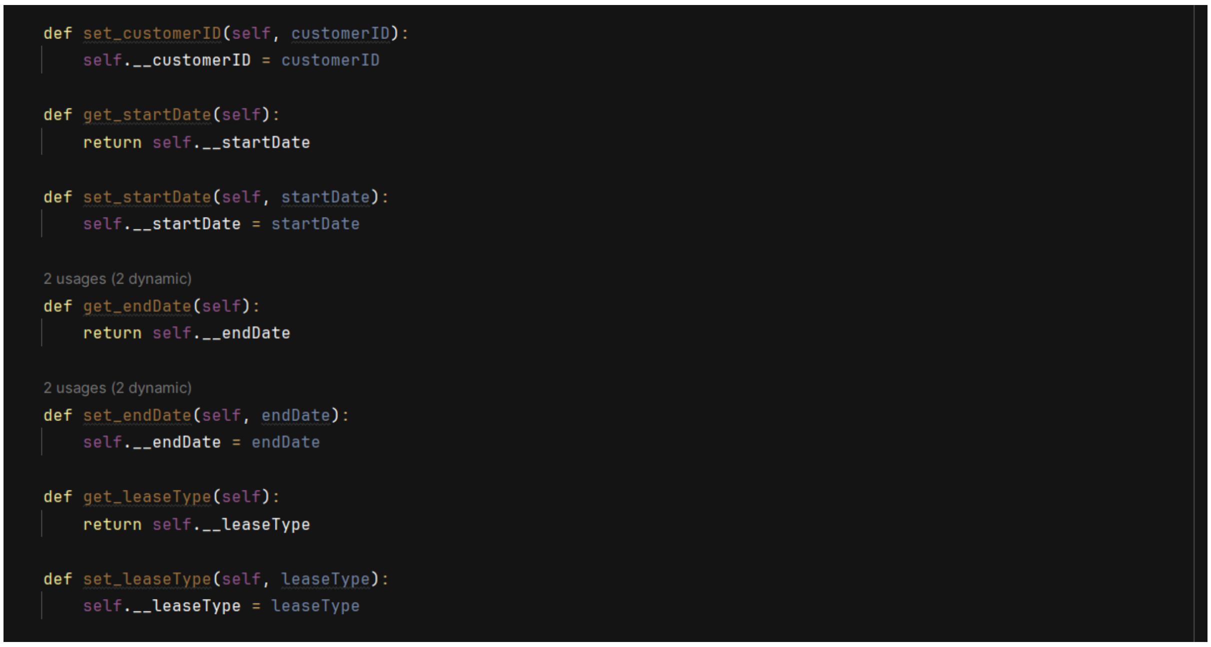
```
    def set_passengerCapacity(self, passengerCapacity):
        self.__passengerCapacity = passengerCapacity

    def get_engineCapacity(self):
        return self.__engineCapacity

    def set_engineCapacity(self, engineCapacity):
        self.__engineCapacity = engineCapacity


1 usages
lass Customer:
    def __init__(self, customerID, firstName, lastName, email, phoneNumber):
        self.__customerID = customerID
        self.__firstName = firstName
        self.__lastName = lastName
        self.__email = email
        self.__phoneNumber = phoneNumber

    6 usages (4 dynamic)
    def get_customerID(self):
        return self.__customerID

    def set_customerID(self, customerID):
        self.__customerID = customerID

    def get_firstName(self):
        return self.__firstName
```



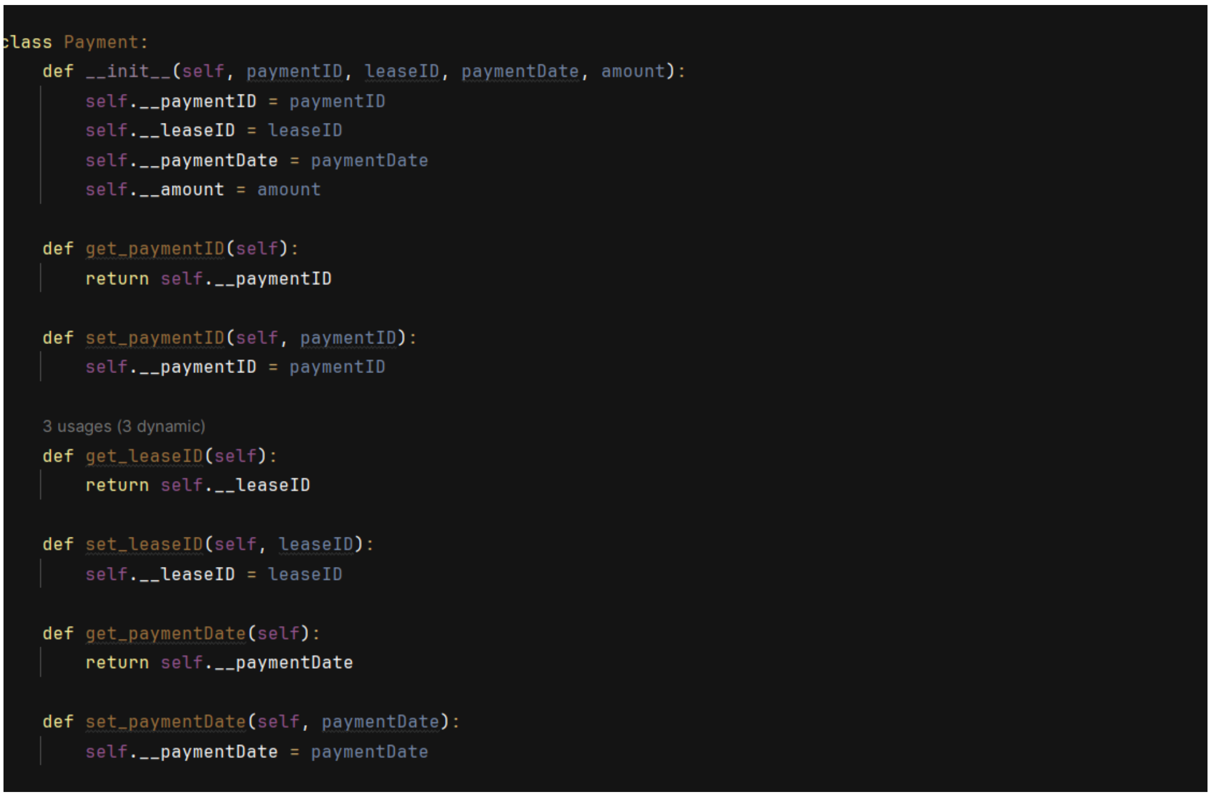
```
    def set_firstName(self, firstName):
        self.__firstName = firstName

    def get_lastName(self):
        return self.__lastName

    def set_lastName(self, lastName):
        self.__lastName = lastName

    def get_email(self):
        return self.__email

    def set_email(self, email):
        self.__email = email

    def get_phoneNumber(self):
        return self.__phoneNumber

    def set_phoneNumber(self, phoneNumber):
        self.__phoneNumber = phoneNumber
```



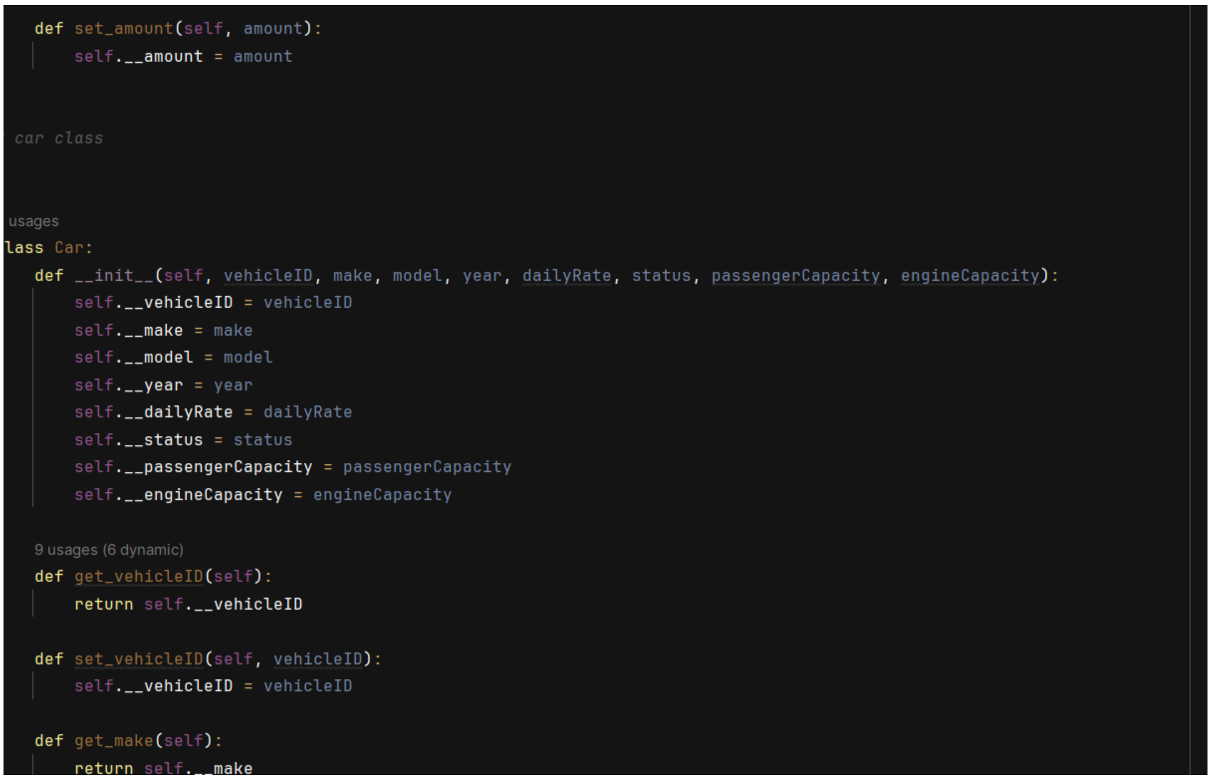
```
7 usages
class Lease:
    def __init__(self, leaseID, vehicleID, customerID, startDate, endDate, leaseType):
        self.__leaseID = leaseID
        self.__vehicleID = vehicleID
        self.__customerID = customerID
        self.__startDate = startDate
        self.__endDate = endDate
        self.__leaseType = leaseType

    3 usages (3 dynamic)
    def get_leaseID(self):
        return self.__leaseID

    def set_leaseID(self, leaseID):
        self.__leaseID = leaseID

    6 usages (6 dynamic)
    def get_vehicleID(self):
        return self.__vehicleID

    def set_vehicleID(self, vehicleID):
        self.__vehicleID = vehicleID

    4 usages (4 dynamic)
    def get_customerID(self):
        return self.__customerID
```



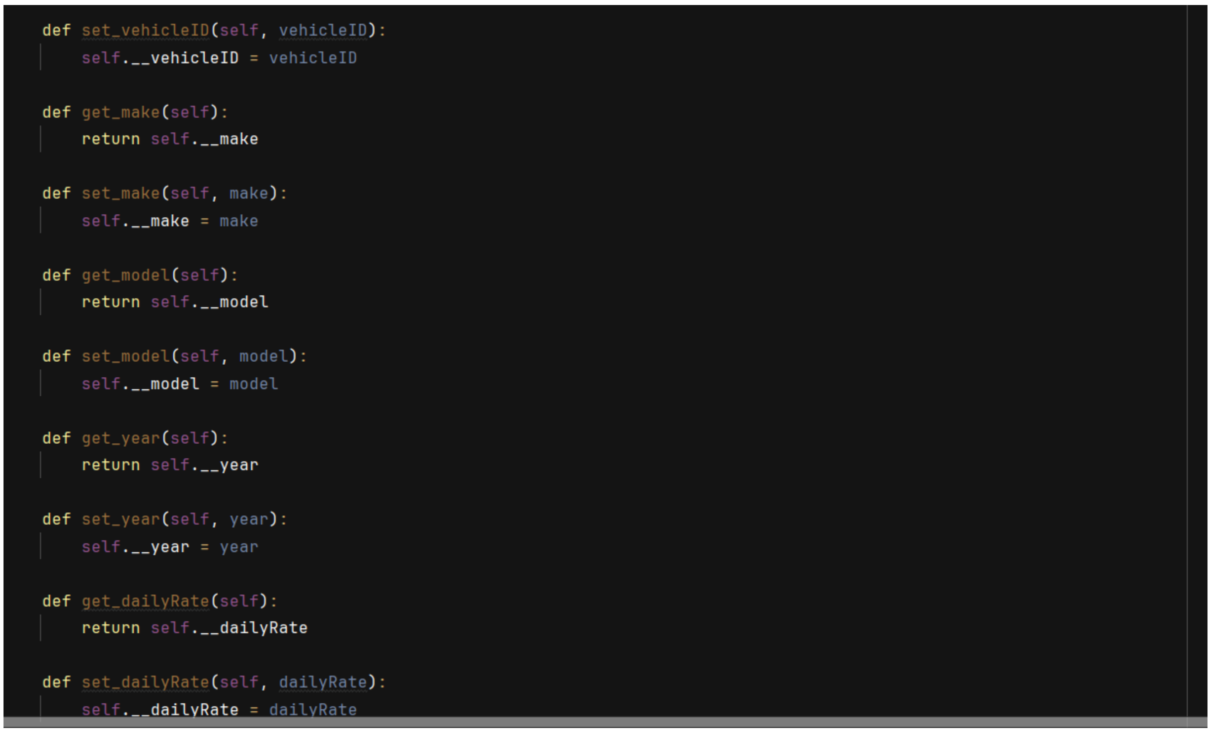
```
    def set_customerID(self, customerID):
        self.__customerID = customerID

    def get_startDate(self):
        return self.__startDate

    def set_startDate(self, startDate):
        self.__startDate = startDate

    2 usages (2 dynamic)
    def get_endDate(self):
        return self.__endDate

    2 usages (2 dynamic)
    def set_endDate(self, endDate):
        self.__endDate = endDate

    def get_leaseType(self):
        return self.__leaseType

    def set_leaseType(self, leaseType):
        self.__leaseType = leaseType
```



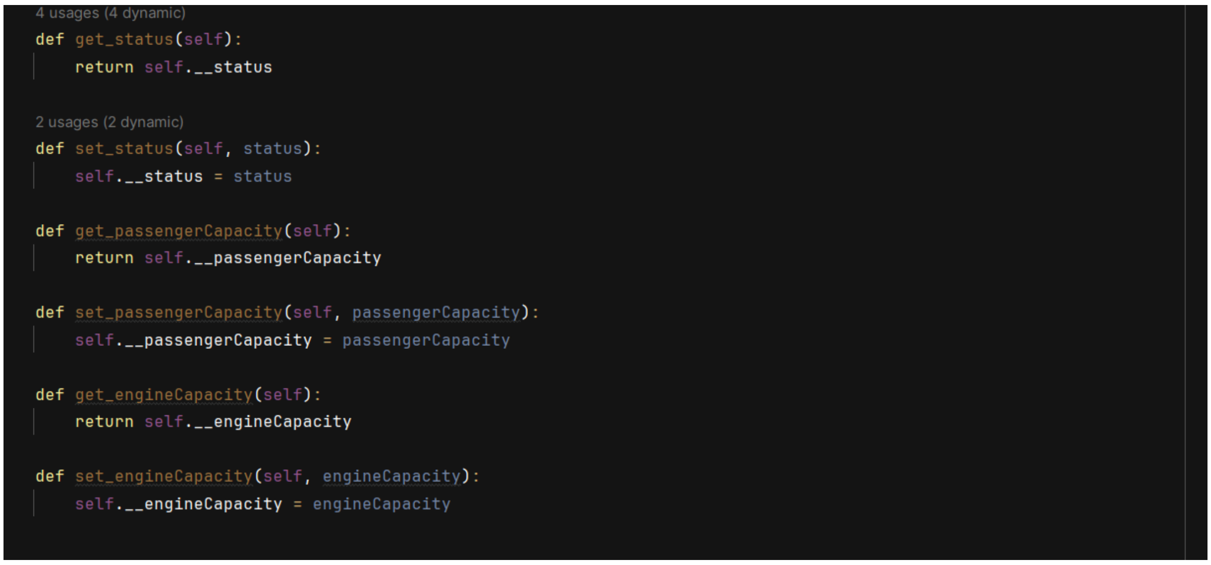
```python
class Payment:
    def __init__(self, paymentID, leaseID, paymentDate, amount):
        self.__paymentID = paymentID
        self.__leaseID = leaseID
        self.__paymentDate = paymentDate
        self.__amount = amount

    def get_paymentID(self):
        return self.__paymentID

    def set_paymentID(self, paymentID):
        self.__paymentID = paymentID

    3 usages (3 dynamic)
    def get_leaseID(self):
        return self.__leaseID

    def set_leaseID(self, leaseID):
        self.__leaseID = leaseID

    def get_paymentDate(self):
        return self.__paymentDate

    def set_paymentDate(self, paymentDate):
        self.__paymentDate = paymentDate
```



```python
    def set_amount(self, amount):
        self.__amount = amount


 car class


usages
lass Car:
    def __init__(self, vehicleID, make, model, year, dailyRate, status, passengerCapacity, engineCapacity):
        self.__vehicleID = vehicleID
        self.__make = make
        self.__model = model
        self.__year = year
        self.__dailyRate = dailyRate
        self.__status = status
        self.__passengerCapacity = passengerCapacity
        self.__engineCapacity = engineCapacity

    9 usages (6 dynamic)
    def get_vehicleID(self):
        return self.__vehicleID

    def set_vehicleID(self, vehicleID):
        self.__vehicleID = vehicleID

    def get_make(self):
        return self.__make
```



```
def set_vehicleID(self, vehicleID):
    self.__vehicleID = vehicleID

def get_make(self):
    return self.__make

def set_make(self, make):
    self.__make = make

def get_model(self):
    return self.__model

def set_model(self, model):
    self.__model = model

def get_year(self):
    return self.__year

def set_year(self, year):
    self.__year = year

def get_dailyRate(self):
    return self.__dailyRate

def set_dailyRate(self, dailyRate):
    self.__dailyRate = dailyRate
```

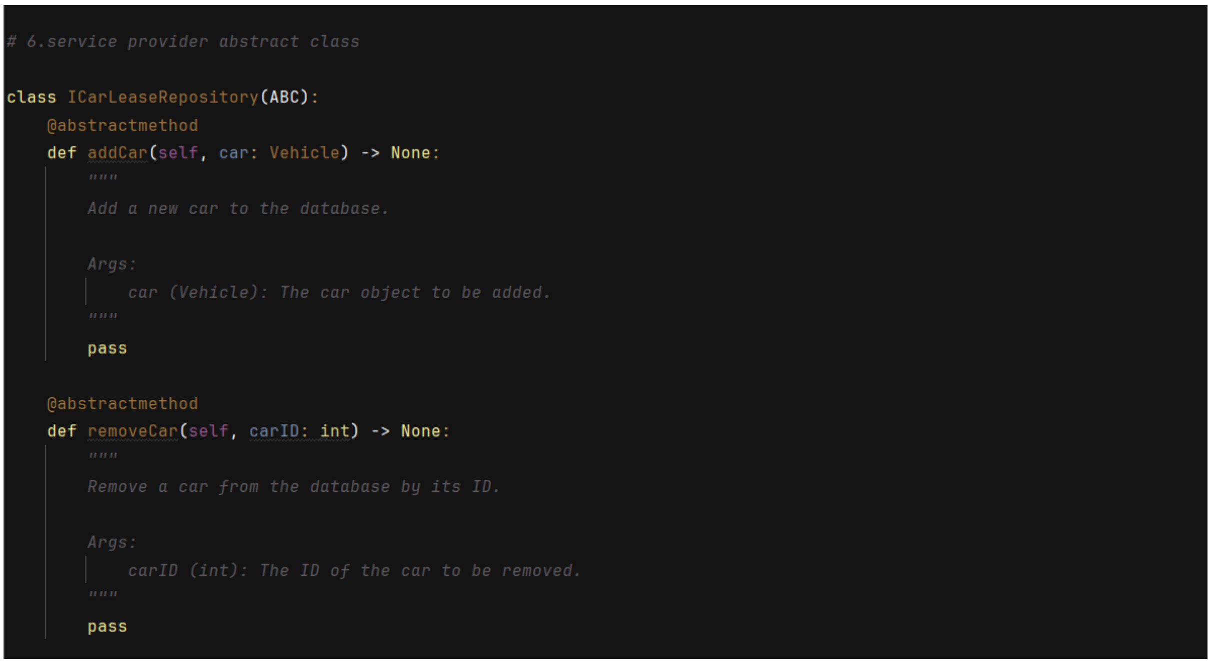


```
4 usages (4 dynamic)
def get_status(self):
    return self.__status

2 usages (2 dynamic)
def set_status(self, status):
    self.__status = status

def get_passengerCapacity(self):
    return self.__passengerCapacity

def set_passengerCapacity(self, passengerCapacity):
    self.__passengerCapacity = passengerCapacity

def get_engineCapacity(self):
    return self.__engineCapacity

def set_engineCapacity(self, engineCapacity):
    self.__engineCapacity = engineCapacity
```

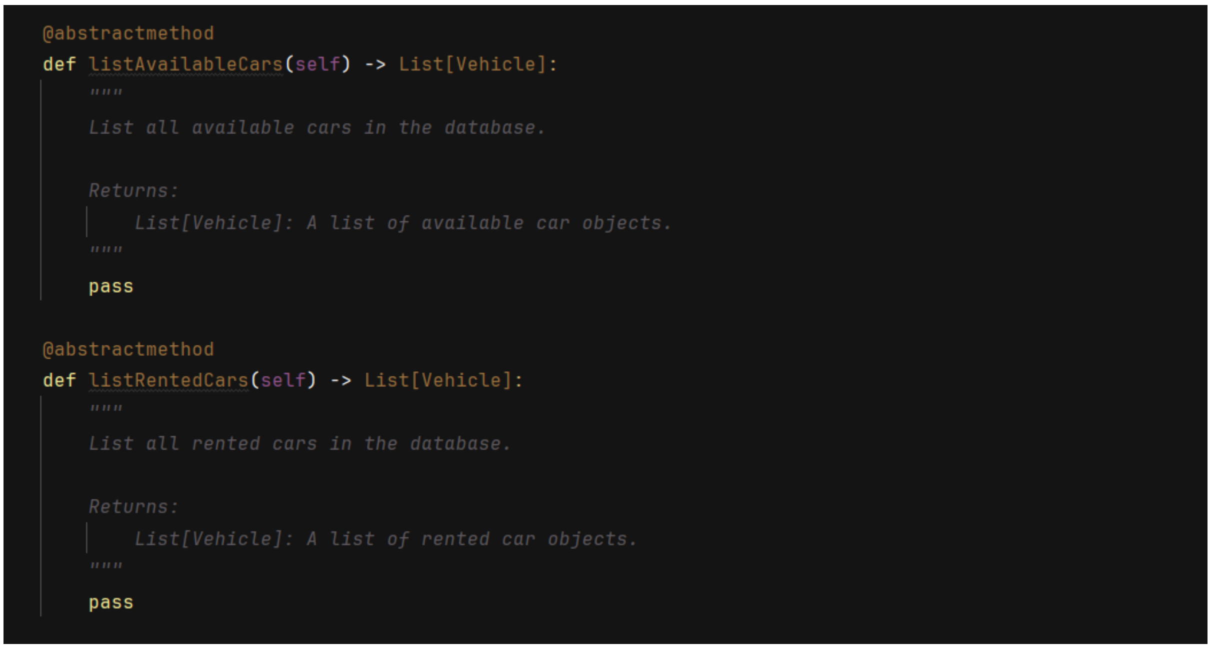
6.Service Provider Interface/Abstract class: Keep the interfaces and implementation classes in package dao

- **Create Interface for ICarLeaseRepository and add following methods which interact with database.**

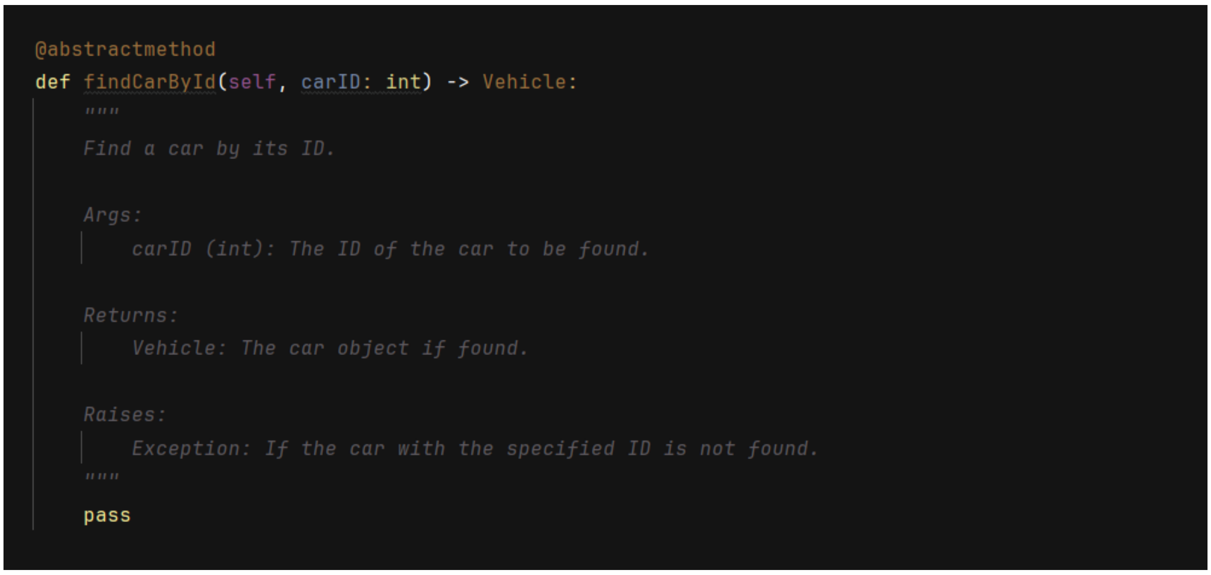
- **Car Management**



```
# 6.service provider abstract class

class ICarLeaseRepository(ABC):
    @abstractmethod
    def addCar(self, car: Vehicle) -> None:
        """
        Add a new car to the database.

        Args:
            car (Vehicle): The car object to be added.
        """
        pass

    @abstractmethod
    def removeCar(self, carID: int) -> None:
        """
        Remove a car from the database by its ID.

        Args:
            carID (int): The ID of the car to be removed.
        """
        pass
```

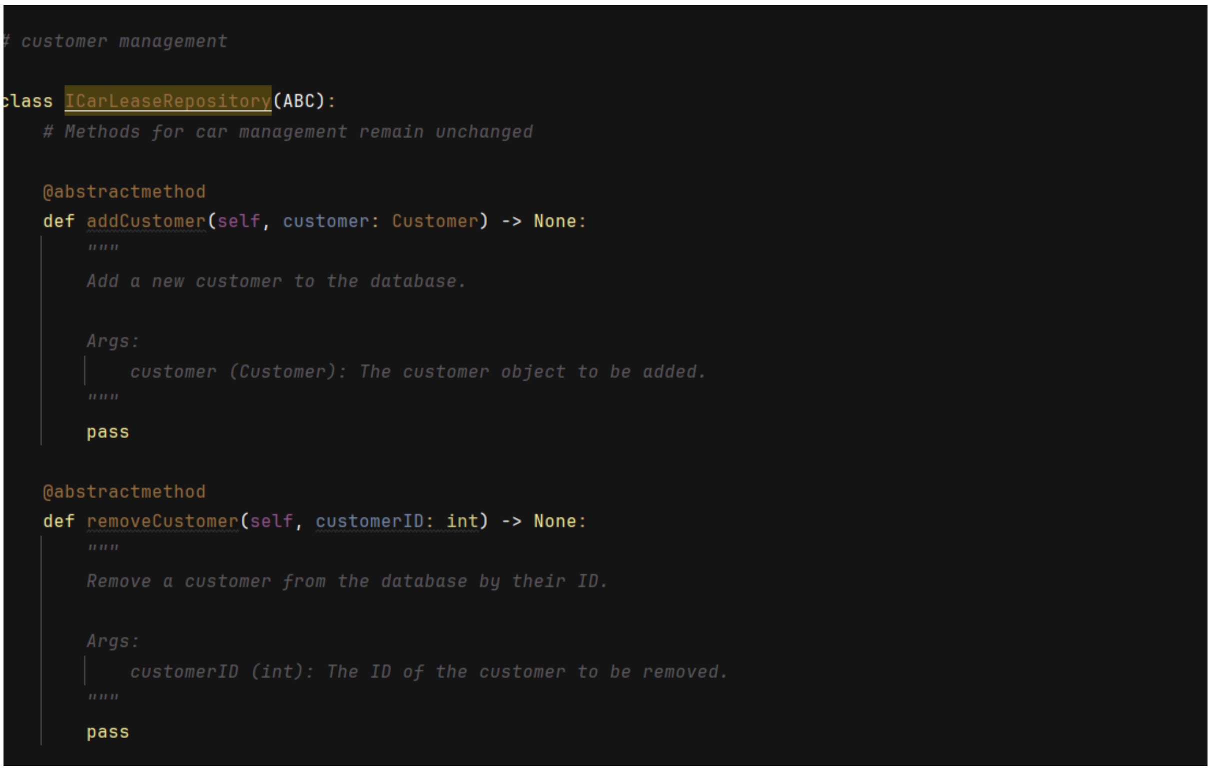


```
    @abstractmethod
    def listAvailableCars(self) -> List[Vehicle]:
        """
        List all available cars in the database.

        Returns:
            List[Vehicle]: A list of available car objects.
        """
        pass

    @abstractmethod
    def listRentedCars(self) -> List[Vehicle]:
        """
        List all rented cars in the database.

        Returns:
            List[Vehicle]: A list of rented car objects.
        """
        pass
```

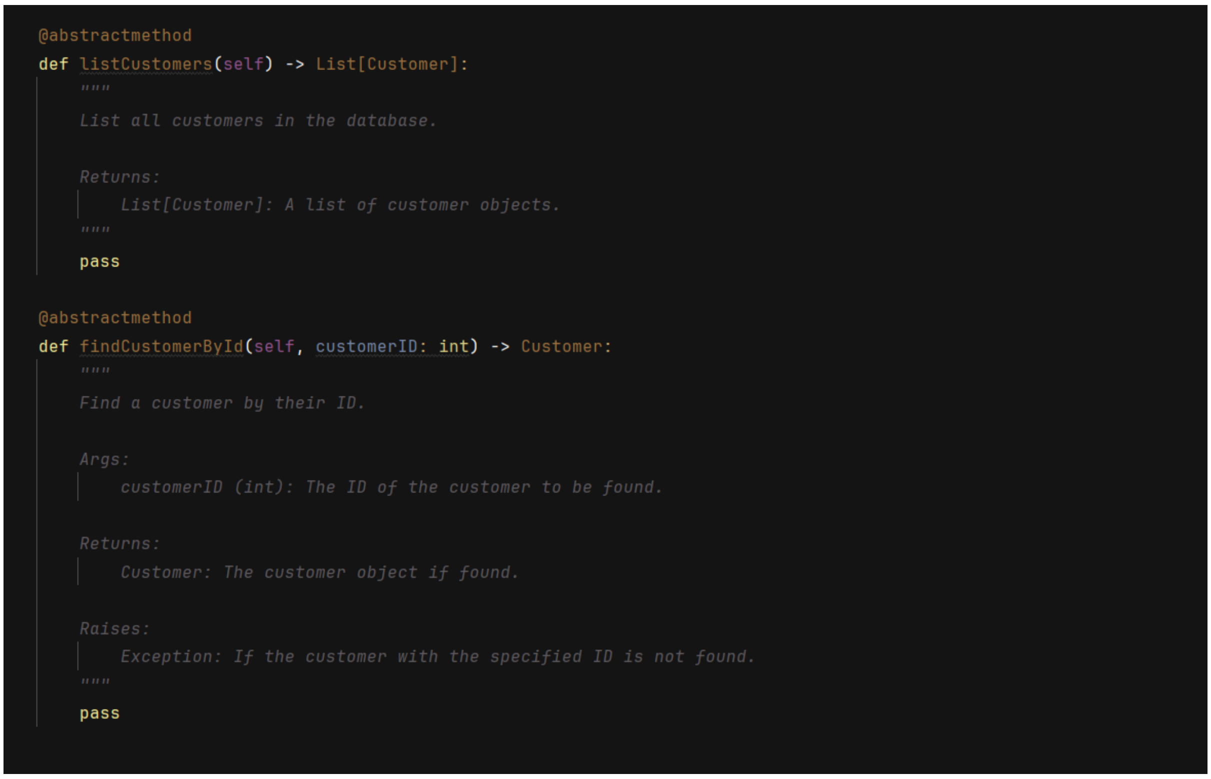


```python
@abstractmethod
def findCarById(self, carID: int) -> Vehicle:
    """
    Find a car by its ID.

    Args:
        carID (int): The ID of the car to be found.

    Returns:
        Vehicle: The car object if found.

    Raises:
        Exception: If the car with the specified ID is not found.
    """
    pass
```

• **Customer Management**

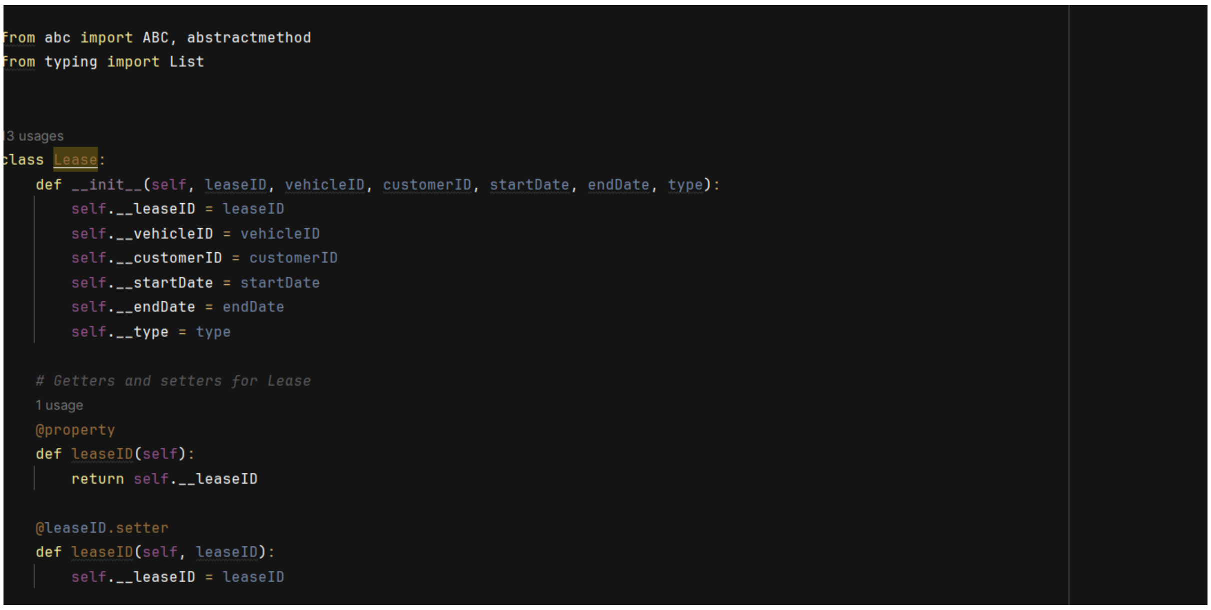


```python
# customer management

class ICarLeaseRepository(ABC):
    # Methods for car management remain unchanged

    @abstractmethod
    def addCustomer(self, customer: Customer) -> None:
        """
        Add a new customer to the database.

        Args:
            customer (Customer): The customer object to be added.
        """
        pass

    @abstractmethod
    def removeCustomer(self, customerID: int) -> None:
        """
        Remove a customer from the database by their ID.

        Args:
            customerID (int): The ID of the customer to be removed.
        """
        pass
```

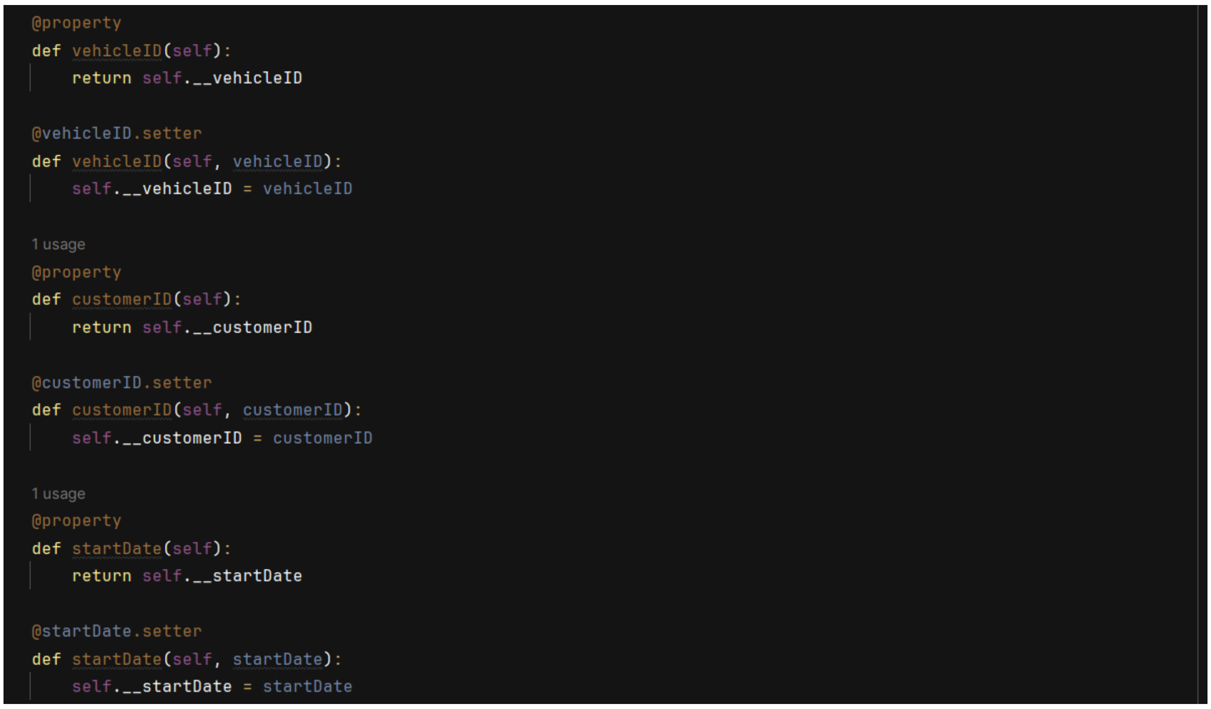


```
    @abstractmethod
    def listCustomers(self) -> List[Customer]:
        """
        List all customers in the database.

        Returns:
            List[Customer]: A list of customer objects.
        """
        pass

    @abstractmethod
    def findCustomerById(self, customerID: int) -> Customer:
        """
        Find a customer by their ID.

        Args:
            customerID (int): The ID of the customer to be found.

        Returns:
            Customer: The customer object if found.

        Raises:
            Exception: If the customer with the specified ID is not found.
        """
        pass
```

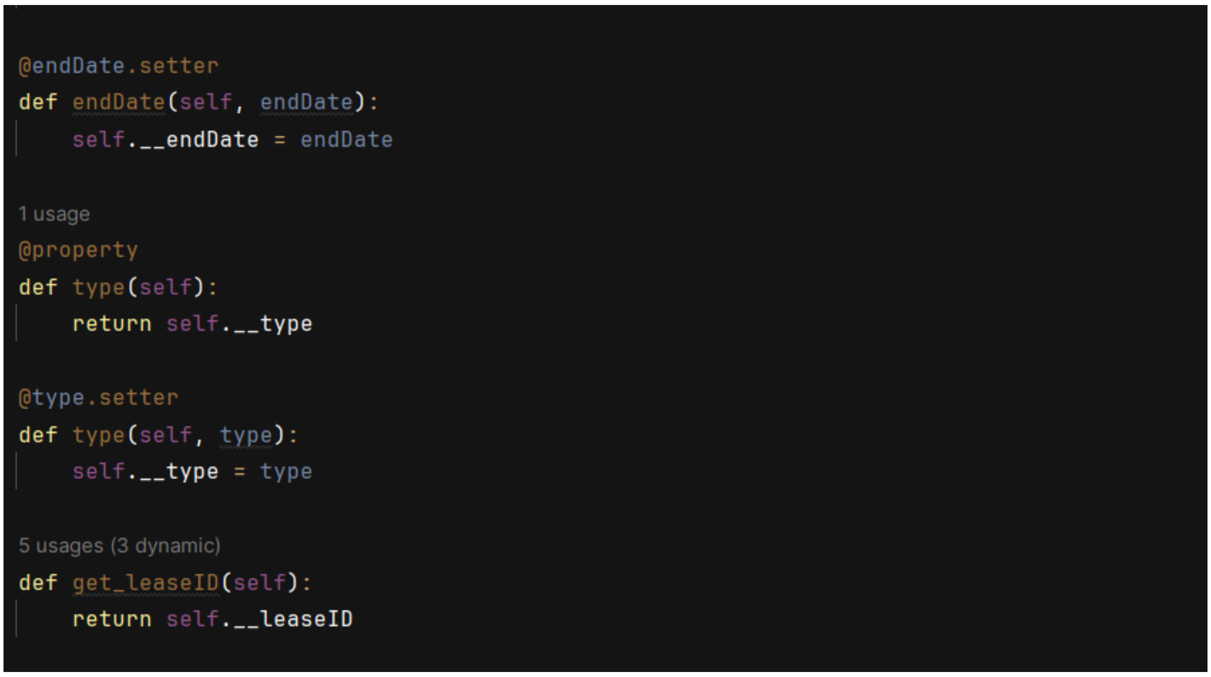
• **Lease Management**



```
from abc import ABC, abstractmethod
from typing import List


3 usages
class Lease:
    def __init__(self, leaseID, vehicleID, customerID, startDate, endDate, type):
        self.__leaseID = leaseID
        self.__vehicleID = vehicleID
        self.__customerID = customerID
        self.__startDate = startDate
        self.__endDate = endDate
        self.__type = type

    # Getters and setters for Lease
    1 usage
    @property
    def leaseID(self):
        return self.__leaseID

    @leaseID.setter
    def leaseID(self, leaseID):
        self.__leaseID = leaseID
```

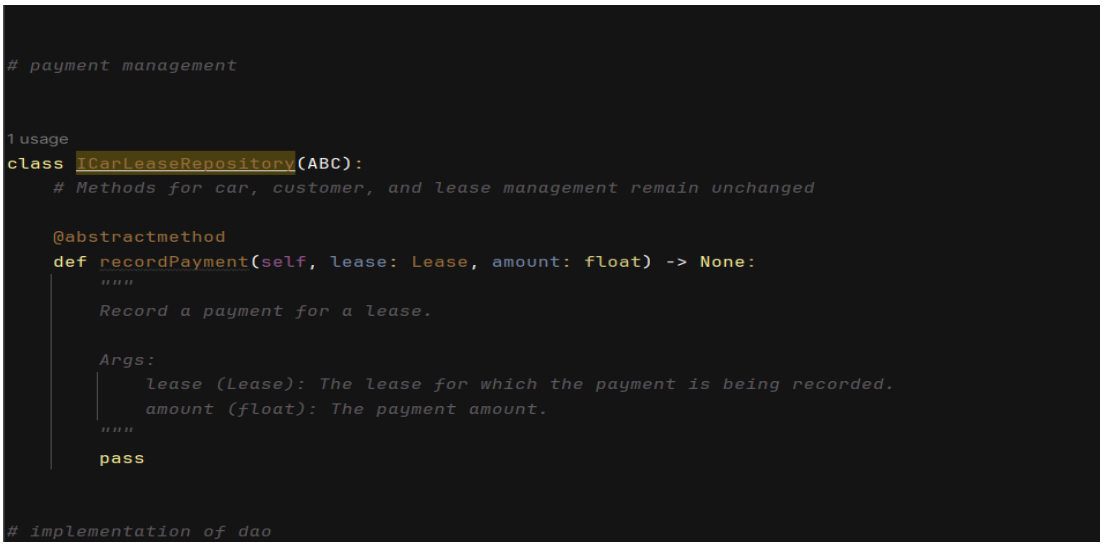


```
@property
def vehicleID(self):
    return self.__vehicleID

@vehicleID.setter
def vehicleID(self, vehicleID):
    self.__vehicleID = vehicleID

1 usage
@property
def customerID(self):
    return self.__customerID

@customerID.setter
def customerID(self, customerID):
    self.__customerID = customerID

1 usage
@property
def startDate(self):
    return self.__startDate

@startDate.setter
def startDate(self, startDate):
    self.__startDate = startDate
```



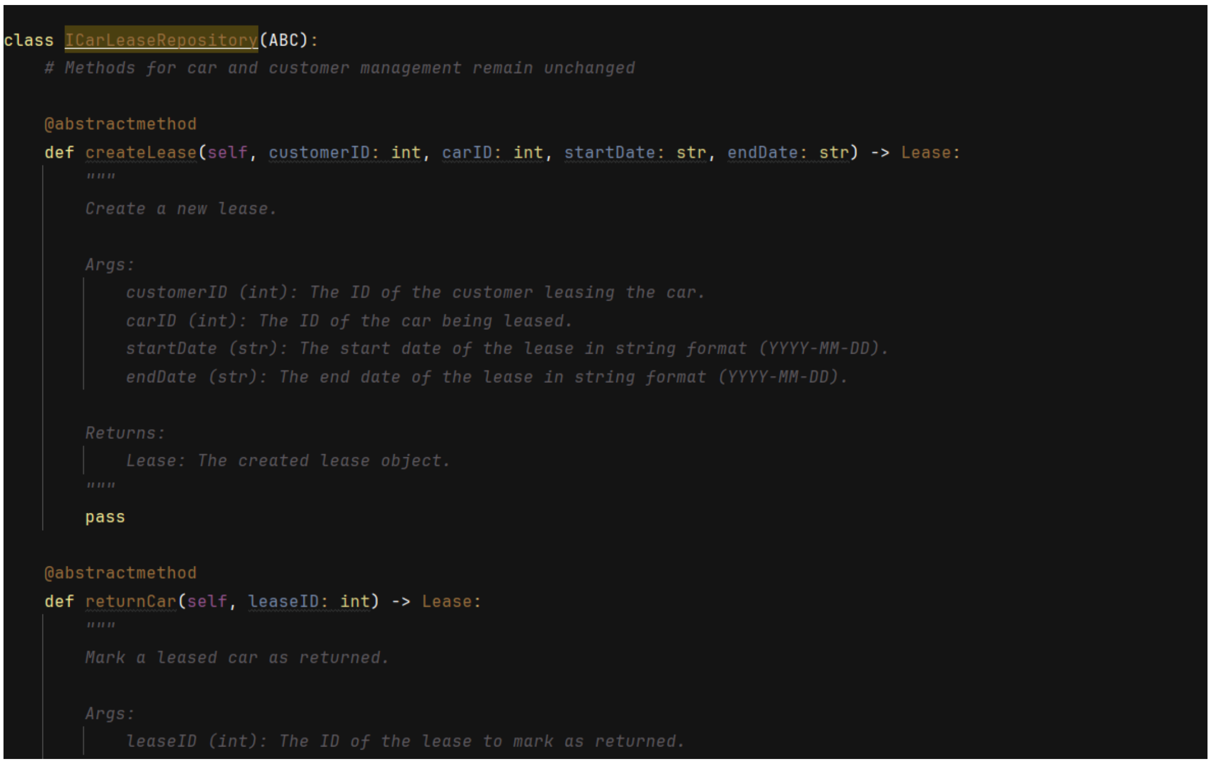
```
@endDate.setter
def endDate(self, endDate):
    self.__endDate = endDate

1 usage
@property
def type(self):
    return self.__type

@type.setter
def type(self, type):
    self.__type = type

5 usages (3 dynamic)
def get_leaseID(self):
    return self.__leaseID
```

• **Payment Handling**

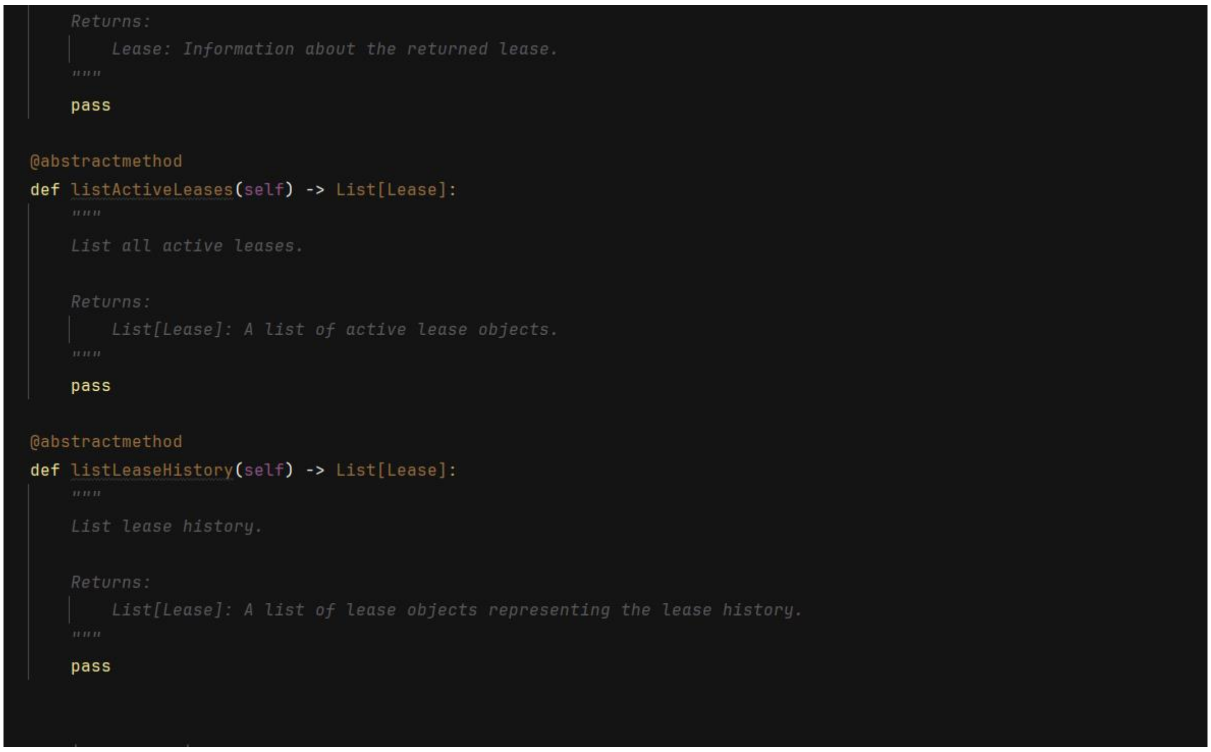


```
# payment management


1 usage
class ICarLeaseRepository(ABC):
    # Methods for car, customer, and lease management remain unchanged

    @abstractmethod
    def recordPayment(self, lease: Lease, amount: float) -> None:
        """
        Record a payment for a lease.

        Args:
            lease (Lease): The lease for which the payment is being recorded.
            amount (float): The payment amount.
        """
        pass


# implementation of dao
```

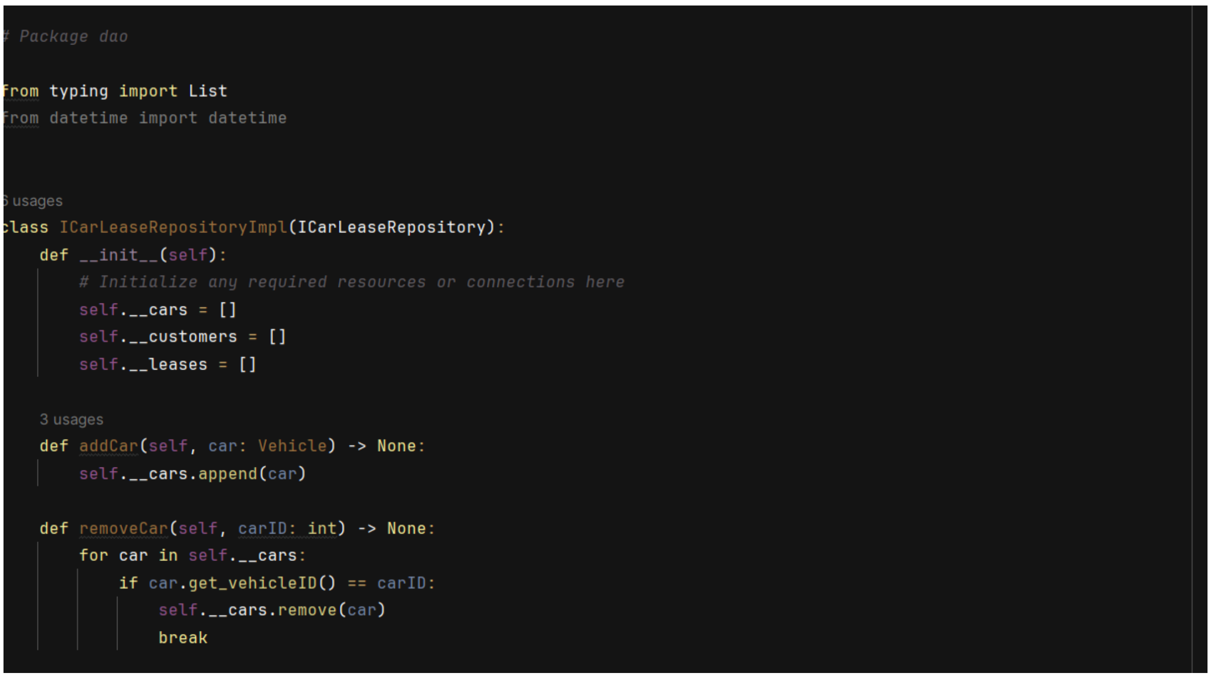
5. **Implement the above interface in a class called ICarLeaseRepositoryImpl in package dao.**



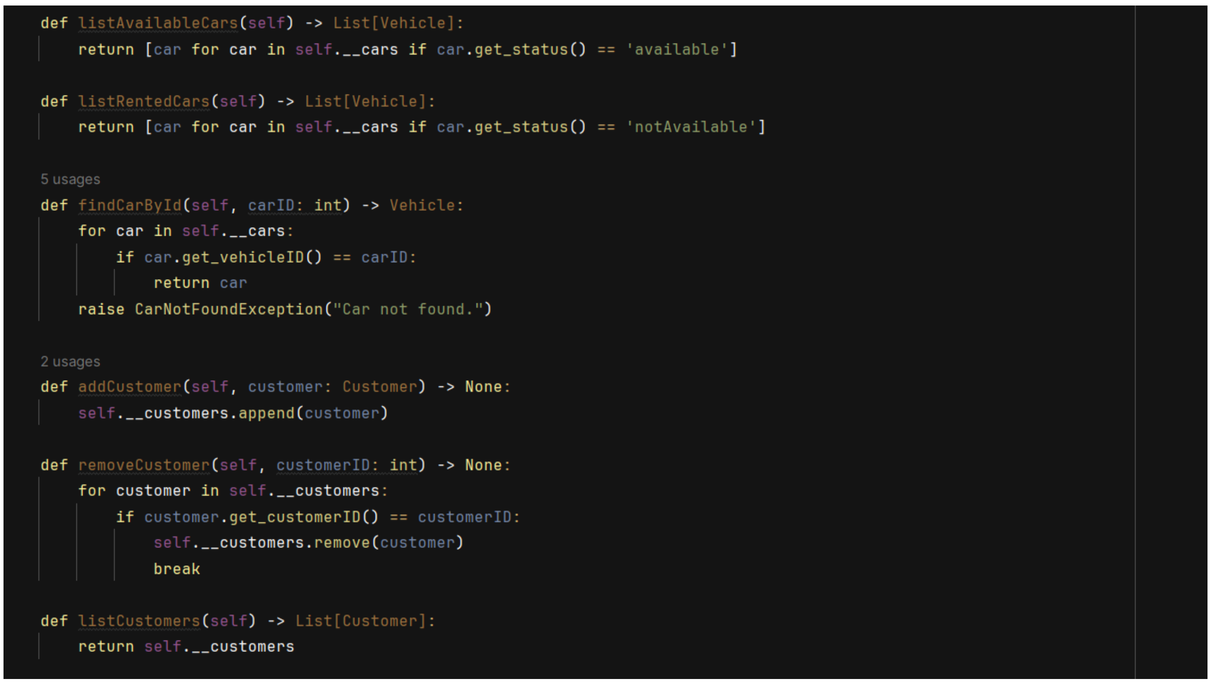
```python
class ICarLeaseRepository(ABC):
    # Methods for car and customer management remain unchanged

    @abstractmethod
    def createLease(self, customerID: int, carID: int, startDate: str, endDate: str) -> Lease:
        """
        Create a new lease.

        Args:
            customerID (int): The ID of the customer leasing the car.
            carID (int): The ID of the car being leased.
            startDate (str): The start date of the lease in string format (YYYY-MM-DD).
            endDate (str): The end date of the lease in string format (YYYY-MM-DD).

        Returns:
            Lease: The created lease object.
        """
        pass

    @abstractmethod
    def returnCar(self, leaseID: int) -> Lease:
        """
        Mark a leased car as returned.

        Args:
            leaseID (int): The ID of the lease to mark as returned.
```



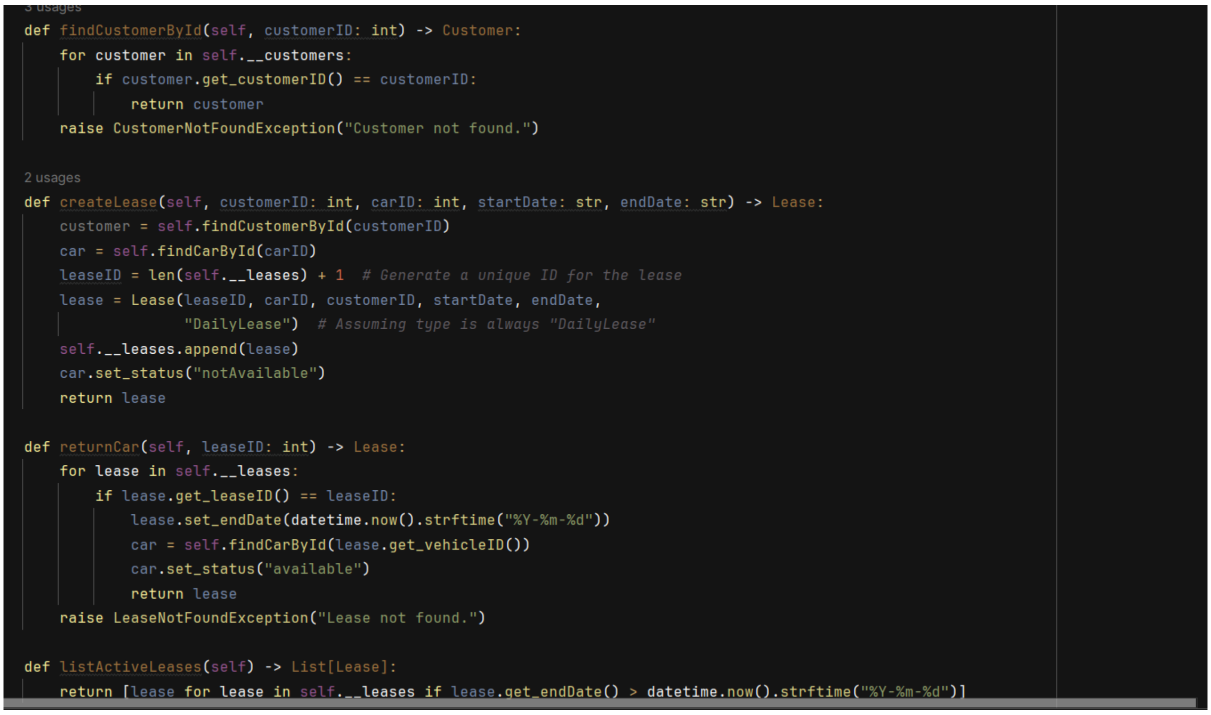
```python
        Returns:
            Lease: Information about the returned lease.
        """
        pass

    @abstractmethod
    def listActiveLeases(self) -> List[Lease]:
        """
        List all active leases.

        Returns:
            List[Lease]: A list of active lease objects.
        """
        pass

    @abstractmethod
    def listLeaseHistory(self) -> List[Lease]:
        """
        List lease history.

        Returns:
            List[Lease]: A list of lease objects representing the lease history.
        """
        pass
```



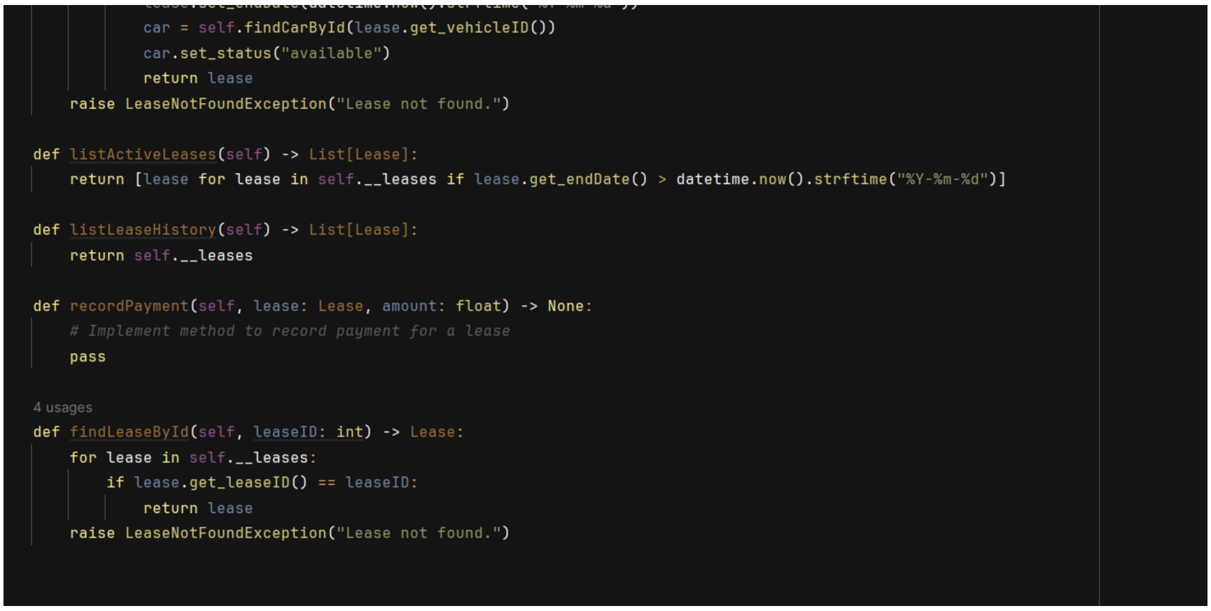
```
# Package dao

from typing import List
from datetime import datetime


6 usages
class ICarLeaseRepositoryImpl(ICarLeaseRepository):
    def __init__(self):
        # Initialize any required resources or connections here
        self.__cars = []
        self.__customers = []
        self.__leases = []

    3 usages
    def addCar(self, car: Vehicle) -> None:
        self.__cars.append(car)

    def removeCar(self, carID: int) -> None:
        for car in self.__cars:
            if car.get_vehicleID() == carID:
                self.__cars.remove(car)
                break
```



```
    def listAvailableCars(self) -> List[Vehicle]:
        return [car for car in self.__cars if car.get_status() == 'available']

    def listRentedCars(self) -> List[Vehicle]:
        return [car for car in self.__cars if car.get_status() == 'notAvailable']

    5 usages
    def findCarById(self, carID: int) -> Vehicle:
        for car in self.__cars:
            if car.get_vehicleID() == carID:
                return car
        raise CarNotFoundException("Car not found.")

    2 usages
    def addCustomer(self, customer: Customer) -> None:
        self.__customers.append(customer)

    def removeCustomer(self, customerID: int) -> None:
        for customer in self.__customers:
            if customer.get_customerID() == customerID:
                self.__customers.remove(customer)
                break

    def listCustomers(self) -> List[Customer]:
        return self.__customers
```

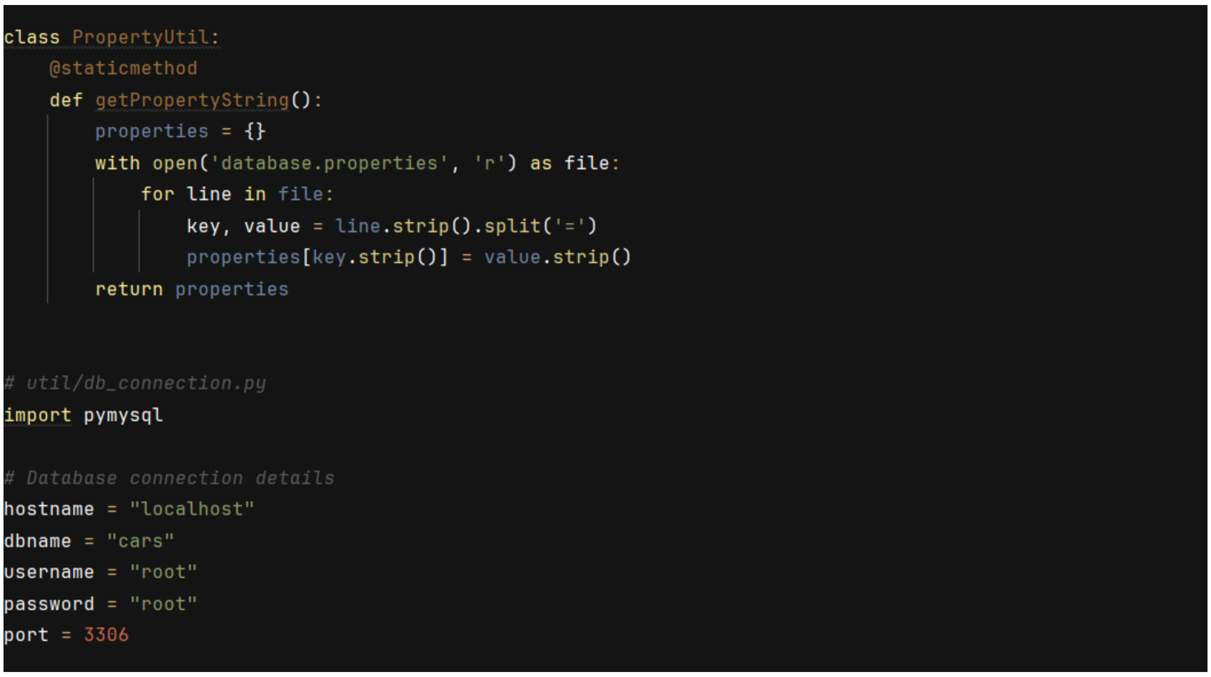


```
3 usages
def findCustomerById(self, customerID: int) -> Customer:
    for customer in self.__customers:
        if customer.get_customerID() == customerID:
            return customer
    raise CustomerNotFoundException("Customer not found.")

2 usages
def createLease(self, customerID: int, carID: int, startDate: str, endDate: str) -> Lease:
    customer = self.findCustomerById(customerID)
    car = self.findCarById(carID)
    leaseID = len(self.__leases) + 1  # Generate a unique ID for the lease
    lease = Lease(leaseID, carID, customerID, startDate, endDate,
                  "DailyLease")  # Assuming type is always "DailyLease"
    self.__leases.append(lease)
    car.set_status("notAvailable")
    return lease

def returnCar(self, leaseID: int) -> Lease:
    for lease in self.__leases:
        if lease.get_leaseID() == leaseID:
            lease.set_endDate(datetime.now().strftime("%Y-%m-%d"))
            car = self.findCarById(lease.get_vehicleID())
            car.set_status("available")
            return lease
    raise LeaseNotFoundException("Lease not found.")

def listActiveLeases(self) -> List[Lease]:
    return [lease for lease in self.__leases if lease.get_endDate() > datetime.now().strftime("%Y-%m-%d")]
```

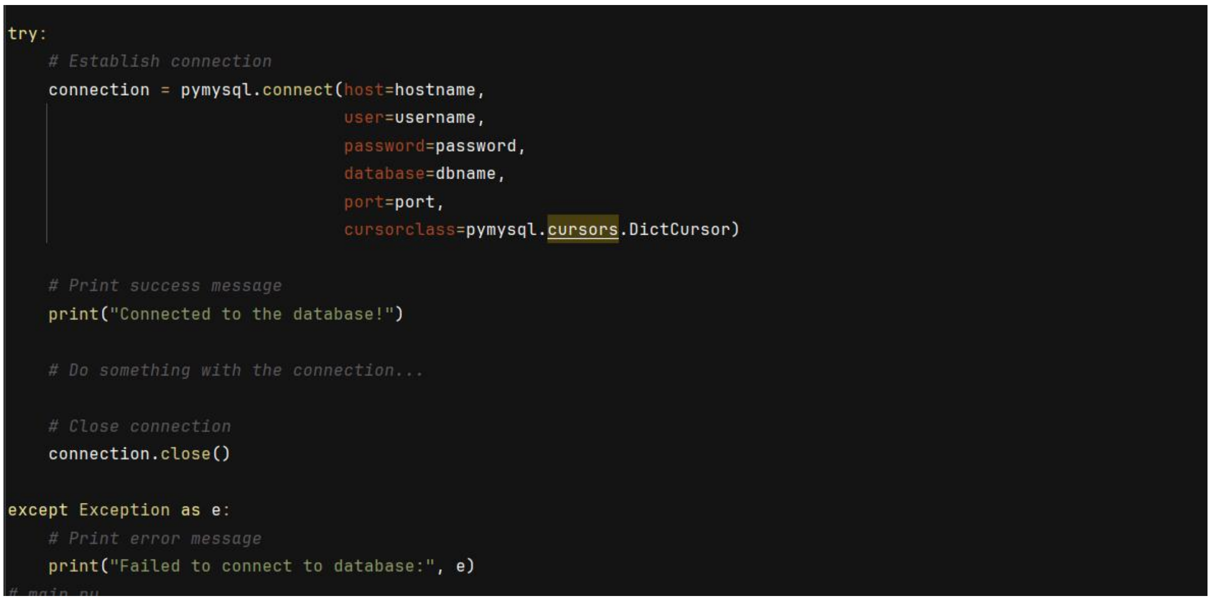


```
            car = self.findCarById(lease.get_vehicleID())
            car.set_status("available")
            return lease
    raise LeaseNotFoundException("Lease not found.")

def listActiveLeases(self) -> List[Lease]:
    return [lease for lease in self.__leases if lease.get_endDate() > datetime.now().strftime("%Y-%m-%d")]

def listLeaseHistory(self) -> List[Lease]:
    return self.__leases

def recordPayment(self, lease: Lease, amount: float) -> None:
    # Implement method to record payment for a lease
    pass

4 usages
def findLeaseById(self, leaseID: int) -> Lease:
    for lease in self.__leases:
        if lease.get_leaseID() == leaseID:
            return lease
    raise LeaseNotFoundException("Lease not found.")
```

8. Connect your application to the SQL database and write code to establish a connection to your SQL database

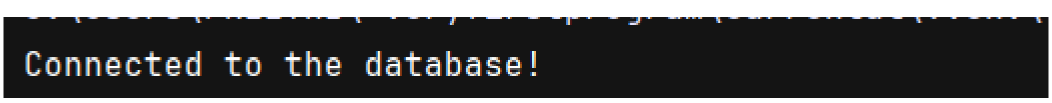


```python
class PropertyUtil:
    @staticmethod
    def getPropertyString():
        properties = {}
        with open('database.properties', 'r') as file:
            for line in file:
                key, value = line.strip().split('=')
                properties[key.strip()] = value.strip()
        return properties


# util/db_connection.py
import pymysql

# Database connection details
hostname = "localhost"
dbname = "cars"
username = "root"
password = "root"
port = 3306
```



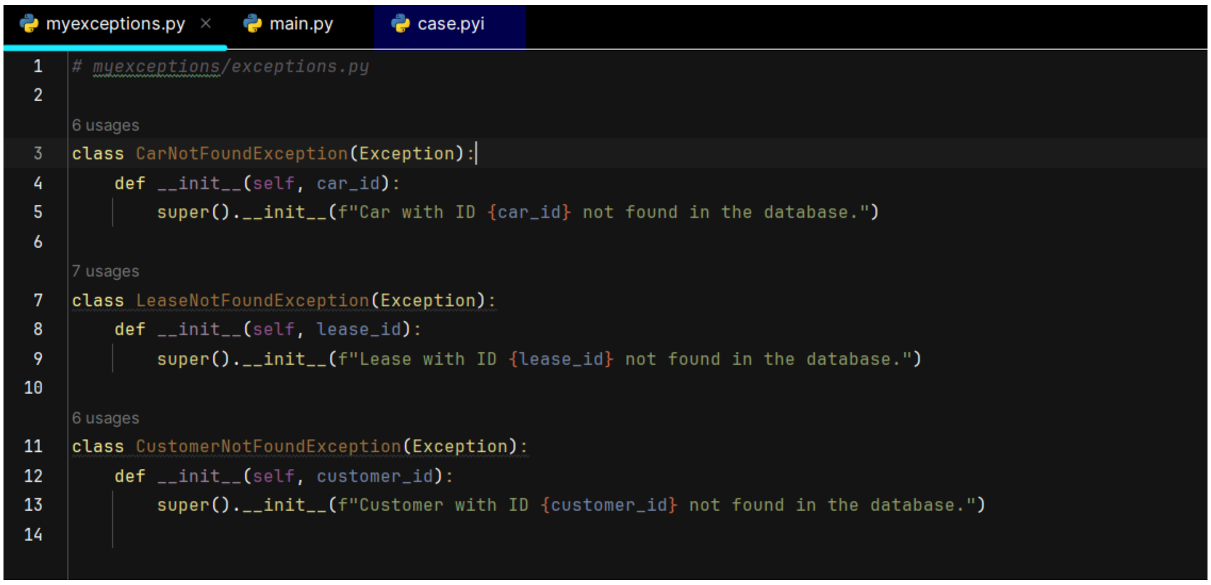
```python
try:
    # Establish connection
    connection = pymysql.connect(host=hostname,
                                 user=username,
                                 password=password,
                                 database=dbname,
                                 port=port,
                                 cursorclass=pymysql.cursors.DictCursor)

    # Print success message
    print("Connected to the database!")

    # Do something with the connection...

    # Close connection
    connection.close()

except Exception as e:
    # Print error message
    print("Failed to connect to database:", e)
```

Output:

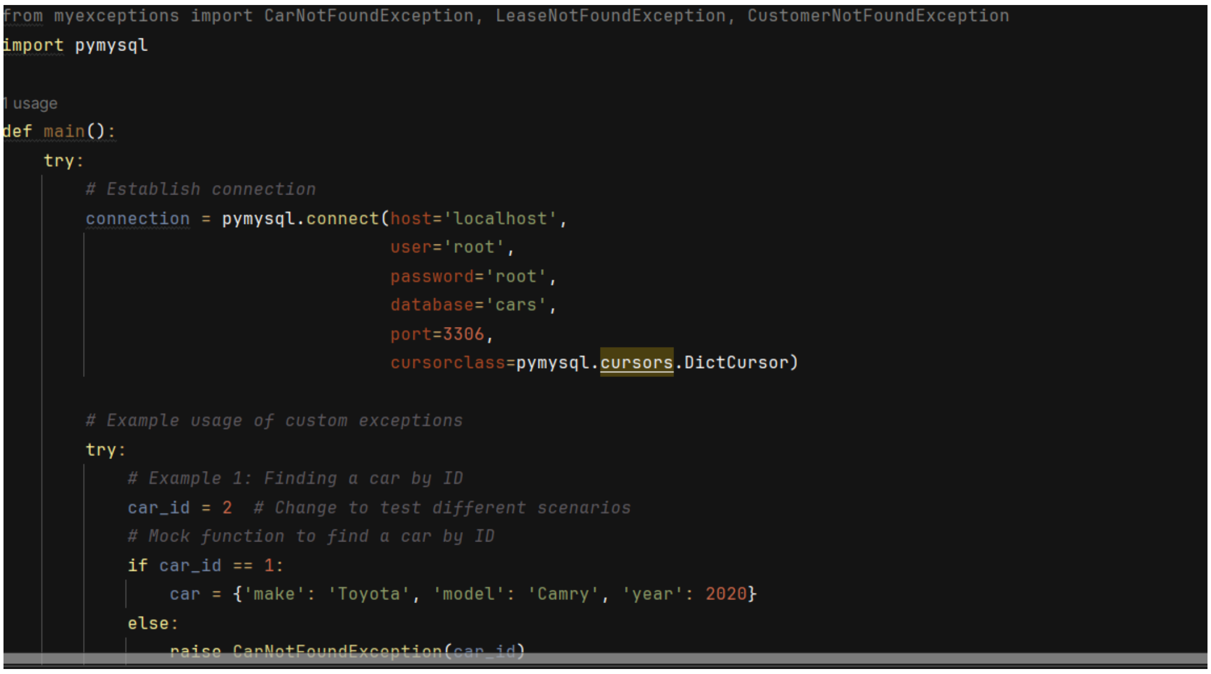


```
Connected to the database!
```

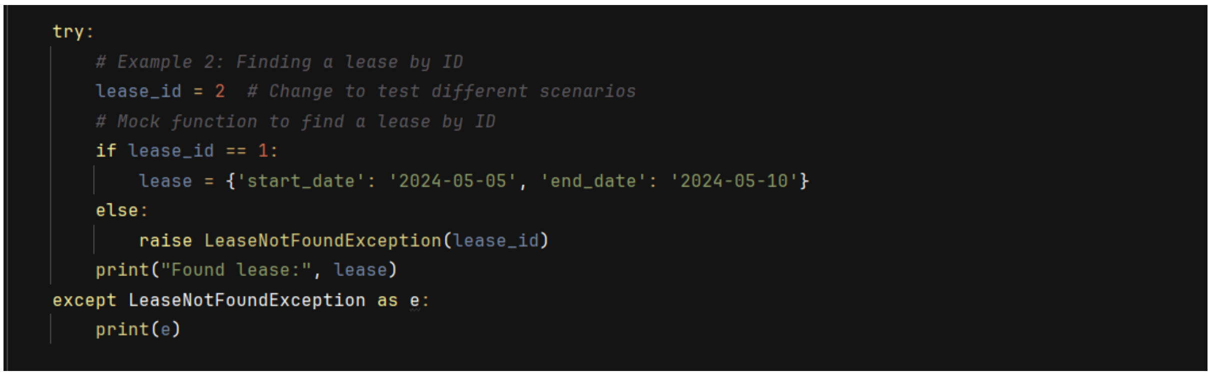
9. Create the exceptions in package myexceptions and create the following custom exceptions and throw them in methods whenever needed. Handle all the exceptions in main method



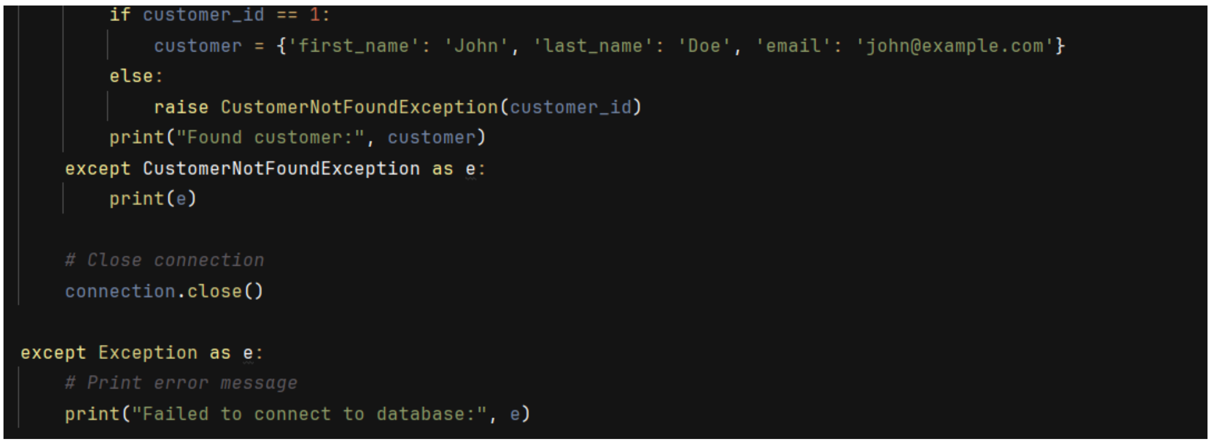
```python
# myexceptions/exceptions.py

6 usages
class CarNotFoundException(Exception):
    def __init__(self, car_id):
        super().__init__(f"Car with ID {car_id} not found in the database.")

7 usages
class LeaseNotFoundException(Exception):
    def __init__(self, lease_id):
        super().__init__(f"Lease with ID {lease_id} not found in the database.")

6 usages
class CustomerNotFoundException(Exception):
    def __init__(self, customer_id):
        super().__init__(f"Customer with ID {customer_id} not found in the database.")

```



```python
from myexceptions import CarNotFoundException, LeaseNotFoundException, CustomerNotFoundException
import pymysql

1 usage
def main():
    try:
        # Establish connection
        connection = pymysql.connect(host='localhost',
                                     user='root',
                                     password='root',
                                     database='cars',
                                     port=3306,
                                     cursorclass=pymysql.cursors.DictCursor)

        # Example usage of custom exceptions
        try:
            # Example 1: Finding a car by ID
            car_id = 2  # Change to test different scenarios
            # Mock function to find a car by ID
            if car_id == 1:
                car = {'make': 'Toyota', 'model': 'Camry', 'year': 2020}
            else:
                raise CarNotFoundException(car_id)
```

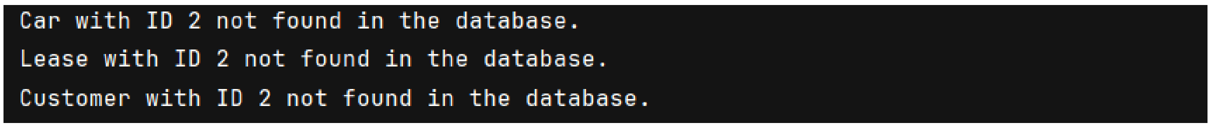


```
    try:
        # Example 2: Finding a lease by ID
        lease_id = 2  # Change to test different scenarios
        # Mock function to find a lease by ID
        if lease_id == 1:
            lease = {'start_date': '2024-05-05', 'end_date': '2024-05-10'}
        else:
            raise LeaseNotFoundException(lease_id)
        print("Found lease:", lease)
    except LeaseNotFoundException as e:
        print(e)
```



```
        if customer_id == 1:
            customer = {'first_name': 'John', 'last_name': 'Doe', 'email': 'john@example.com'}
        else:
            raise CustomerNotFoundException(customer_id)
        print("Found customer:", customer)
    except CustomerNotFoundException as e:
        print(e)

    # Close connection
    connection.close()

except Exception as e:
    # Print error message
    print("Failed to connect to database:", e)
```

Output:

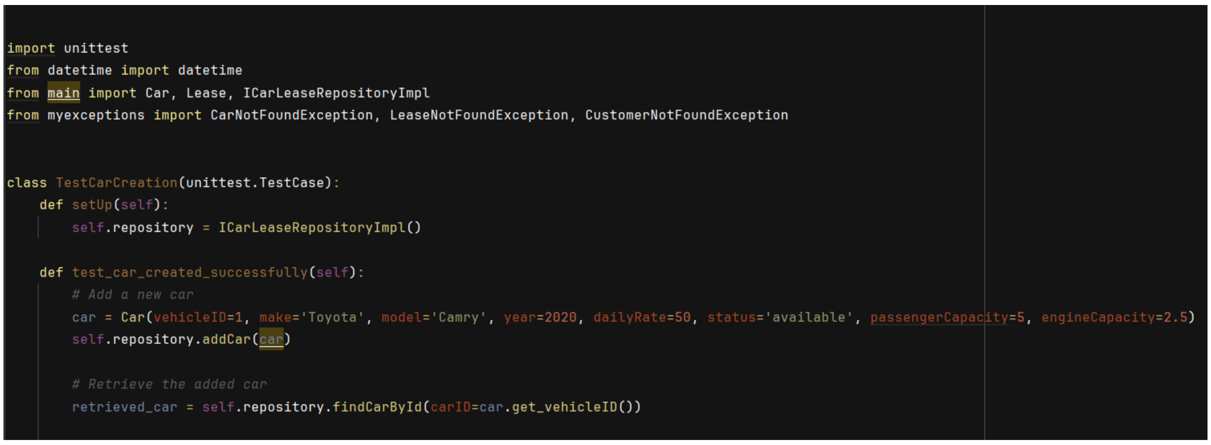


```
Car with ID 2 not found in the database.
Lease with ID 2 not found in the database.
Customer with ID 2 not found in the database.
```

Unit Testing:

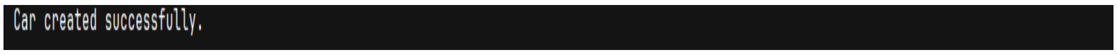
10. Create Unit test cases for Ecommerce System are essential to ensure the correctness and reliability of your system. Following questions to guide the creation of Unit test cases:

• Write test case to test car created successfully or not.



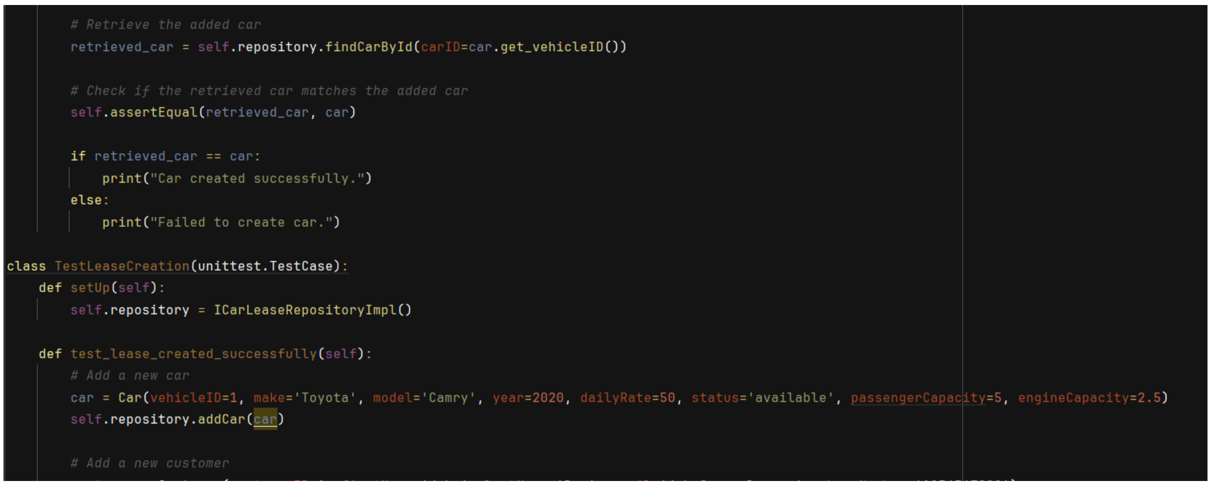
```python
import unittest
from datetime import datetime
from main import Car, Lease, ICarLeaseRepositoryImpl
from myexceptions import CarNotFoundException, LeaseNotFoundException, CustomerNotFoundException


class TestCarCreation(unittest.TestCase):
    def setUp(self):
        self.repository = ICarLeaseRepositoryImpl()

    def test_car_created_successfully(self):
        # Add a new car
        car = Car(vehicleID=1, make='Toyota', model='Camry', year=2020, dailyRate=50, status='available', passengerCapacity=5, engineCapacity=2.5)
        self.repository.addCar(car)

        # Retrieve the added car
        retrieved_car = self.repository.findCarById(carID=car.get_vehicleID())
```

Output:



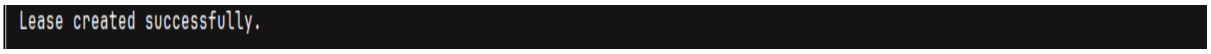
```
Car created successfully.
```

• Write test case to test lease is created successfully or not.



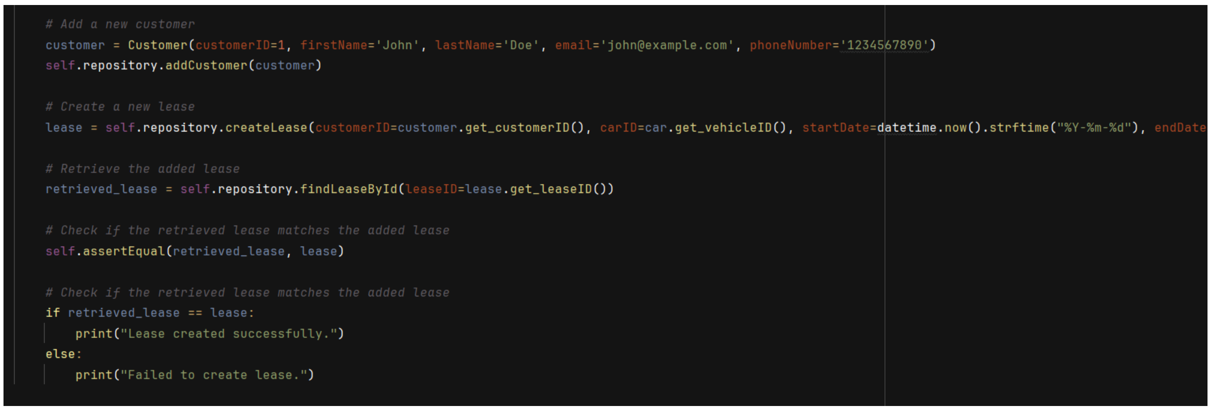
```python
        # Retrieve the added car
        retrieved_car = self.repository.findCarById(carID=car.get_vehicleID())

        # Check if the retrieved car matches the added car
        self.assertEqual(retrieved_car, car)

        if retrieved_car == car:
            print("Car created successfully.")
        else:
            print("Failed to create car.")

class TestLeaseCreation(unittest.TestCase):
    def setUp(self):
        self.repository = ICarLeaseRepositoryImpl()

    def test_lease_created_successfully(self):
        # Add a new car
        car = Car(vehicleID=1, make='Toyota', model='Camry', year=2020, dailyRate=50, status='available', passengerCapacity=5, engineCapacity=2.5)
        self.repository.addCar(car)

        # Add a new customer
```

Output:



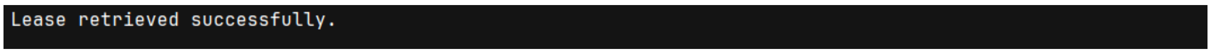
```
Lease created successfully.
```

• Write test case to test lease is retrieved successfully or not



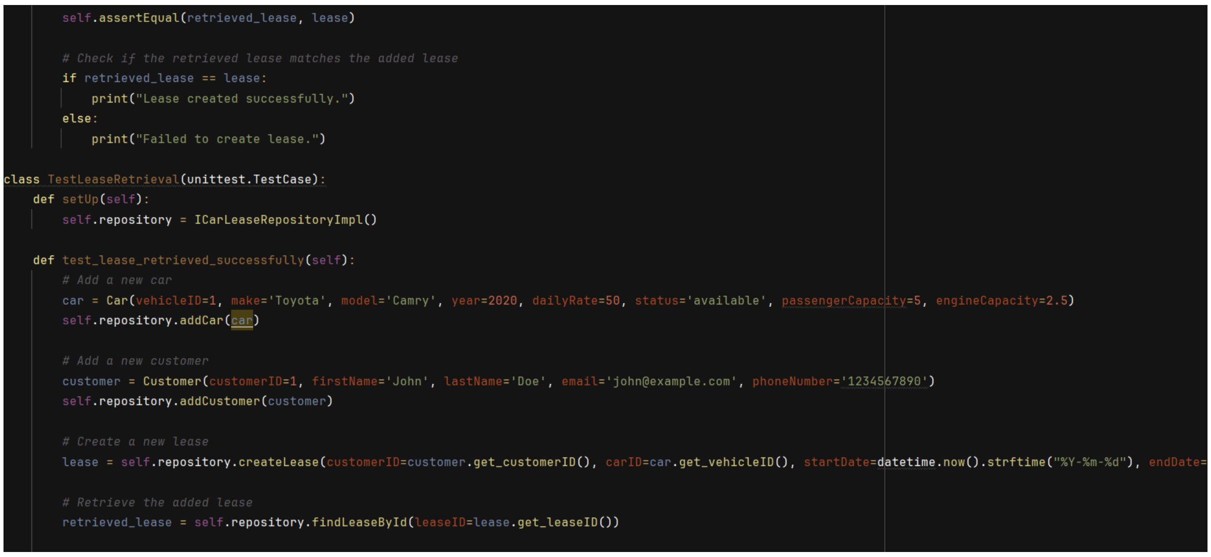
```
# Add a new customer
customer = Customer(customerID=1, firstName='John', lastName='Doe', email='john@example.com', phoneNumber='1234567890')
self.repository.addCustomer(customer)

# Create a new lease
lease = self.repository.createLease(customerID=customer.get_customerID(), carID=car.get_vehicleID(), startDate=datetime.now().strftime("%Y-%m-%d"), endDate

# Retrieve the added lease
retrieved_lease = self.repository.findLeaseById(leaseID=lease.get_leaseID())

# Check if the retrieved lease matches the added lease
self.assertEqual(retrieved_lease, lease)

# Check if the retrieved lease matches the added lease
if retrieved_lease == lease:
    print("Lease created successfully.")
else:
    print("Failed to create lease.")
```

Ouput:

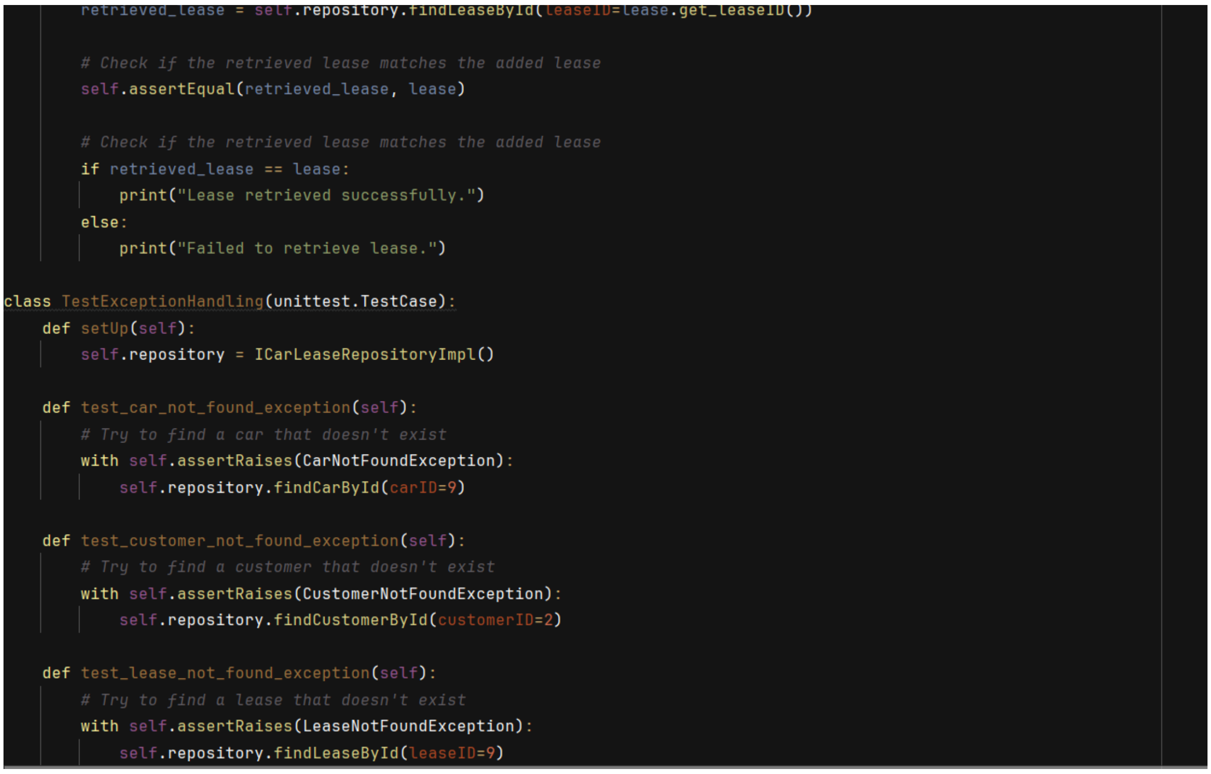


```
Lease retrieved successfully.
```

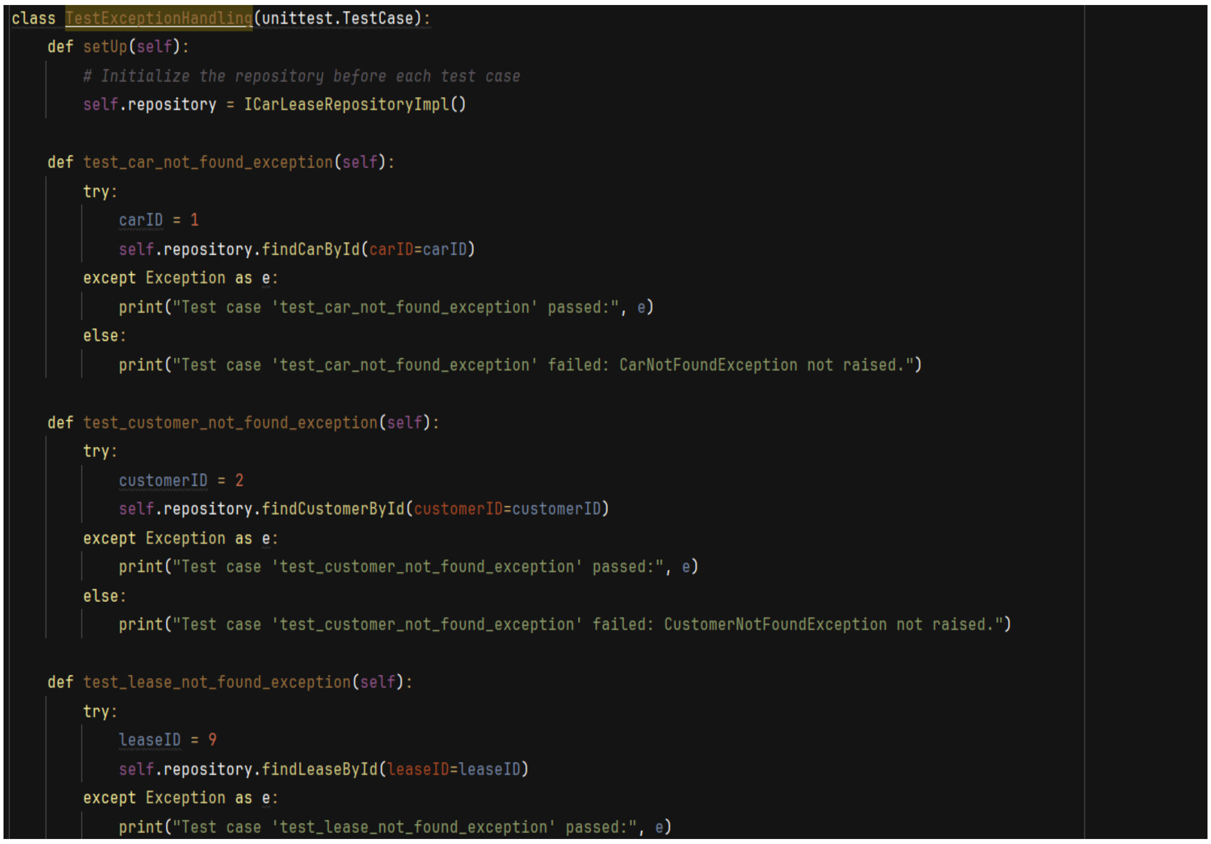
• write test case to test exception is thrown correctly or not when customer id or car id or lease id not found in database



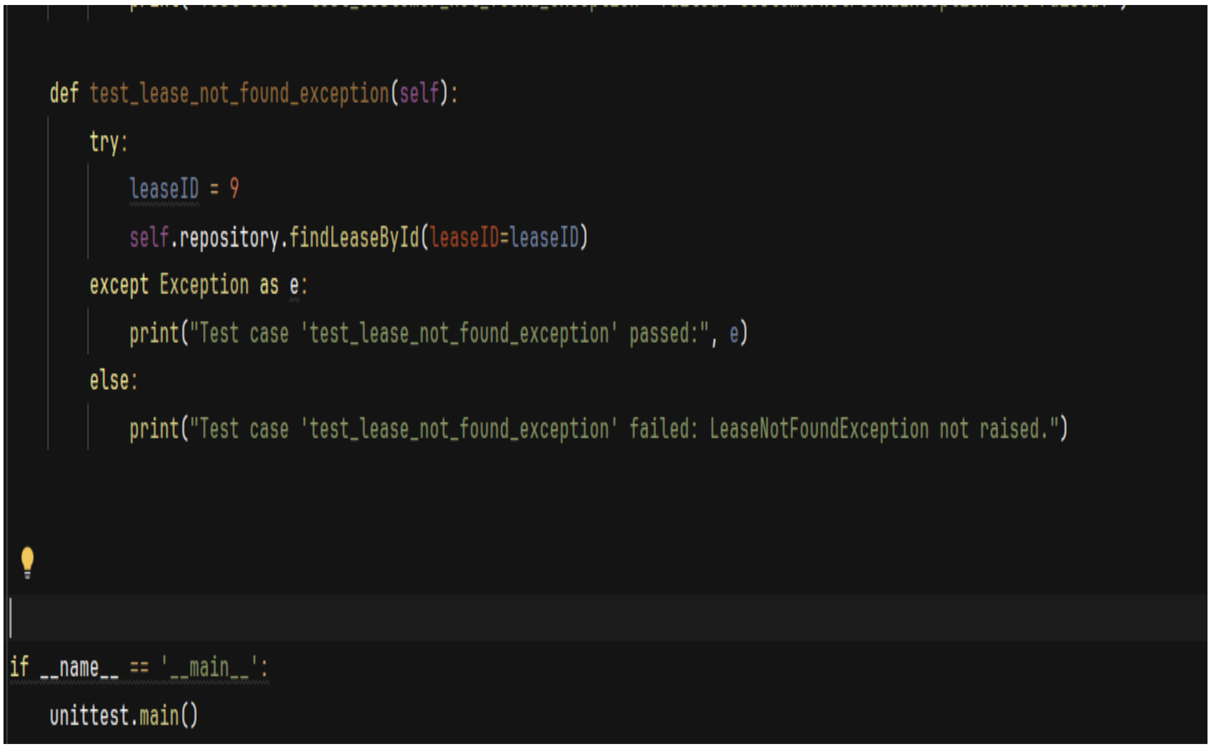
```python
        self.assertEqual(retrieved_lease, lease)

        # Check if the retrieved lease matches the added lease
        if retrieved_lease == lease:
            print("Lease created successfully.")
        else:
            print("Failed to create lease.")

class TestLeaseRetrieval(unittest.TestCase):
    def setUp(self):
        self.repository = ICarLeaseRepositoryImpl()

    def test_lease_retrieved_successfully(self):
        # Add a new car
        car = Car(vehicleID=1, make='Toyota', model='Camry', year=2020, dailyRate=50, status='available', passengerCapacity=5, engineCapacity=2.5)
        self.repository.addCar(car)

        # Add a new customer
        customer = Customer(customerID=1, firstName='John', lastName='Doe', email='john@example.com', phoneNumber='1234567890')
        self.repository.addCustomer(customer)

        # Create a new lease
        lease = self.repository.createLease(customerID=customer.get_customerID(), carID=car.get_vehicleID(), startDate=datetime.now().strftime("%Y-%m-%d"), endDate=

        # Retrieve the added lease
        retrieved_lease = self.repository.findLeaseById(leaseID=lease.get_leaseID())
```



```python
        retrieved_lease = self.repository.findLeaseById(leaseID=lease.get_leaseID())

        # Check if the retrieved lease matches the added lease
        self.assertEqual(retrieved_lease, lease)

        # Check if the retrieved lease matches the added lease
        if retrieved_lease == lease:
            print("Lease retrieved successfully.")
        else:
            print("Failed to retrieve lease.")

class TestExceptionHandling(unittest.TestCase):
    def setUp(self):
        self.repository = ICarLeaseRepositoryImpl()

    def test_car_not_found_exception(self):
        # Try to find a car that doesn't exist
        with self.assertRaises(CarNotFoundException):
            self.repository.findCarById(carID=9)

    def test_customer_not_found_exception(self):
        # Try to find a customer that doesn't exist
        with self.assertRaises(CustomerNotFoundException):
            self.repository.findCustomerById(customerID=2)

    def test_lease_not_found_exception(self):
        # Try to find a lease that doesn't exist
        with self.assertRaises(LeaseNotFoundException):
            self.repository.findLeaseById(leaseID=9)
```

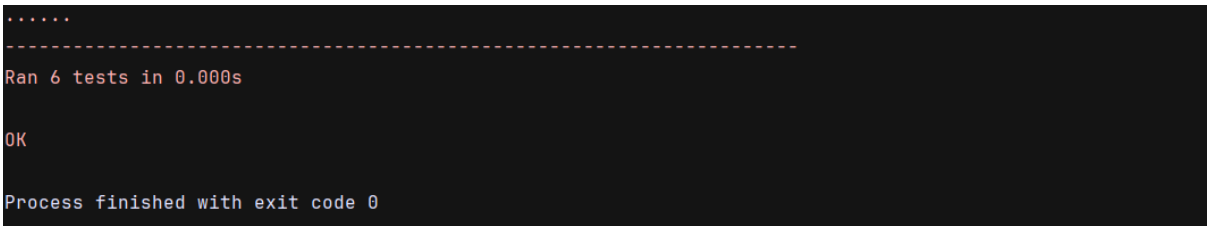


```python
class TestExceptionHandling(unittest.TestCase):
    def setUp(self):
        # Initialize the repository before each test case
        self.repository = ICarLeaseRepositoryImpl()

    def test_car_not_found_exception(self):
        try:
            carID = 1
            self.repository.findCarById(carID=carID)
        except Exception as e:
            print("Test case 'test_car_not_found_exception' passed:", e)
        else:
            print("Test case 'test_car_not_found_exception' failed: CarNotFoundException not raised.")

    def test_customer_not_found_exception(self):
        try:
            customerID = 2
            self.repository.findCustomerById(customerID=customerID)
        except Exception as e:
            print("Test case 'test_customer_not_found_exception' passed:", e)
        else:
            print("Test case 'test_customer_not_found_exception' failed: CustomerNotFoundException not raised.")

    def test_lease_not_found_exception(self):
        try:
            leaseID = 9
            self.repository.findLeaseById(leaseID=leaseID)
        except Exception as e:
            print("Test case 'test_lease_not_found_exception' passed:", e)
```

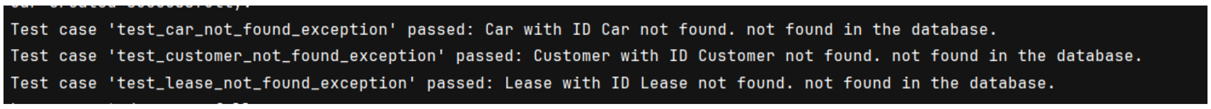


```python
    def test_lease_not_found_exception(self):
        try:
            leaseID = 9
            self.repository.findLeaseById(leaseID=leaseID)
        except Exception as e:
            print("Test case 'test_lease_not_found_exception' passed:", e)
        else:
            print("Test case 'test_lease_not_found_exception' failed: LeaseNotFoundException not raised.")


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

Output:



```
......
----------------------------------------------------------------------
Ran 6 tests in 0.000s

OK

Process finished with exit code 0
```



```
Test case 'test_car_not_found_exception' passed: Car with ID Car not found. not found in the database.
Test case 'test_customer_not_found_exception' passed: Customer with ID Customer not found. not found in the database.
Test case 'test_lease_not_found_exception' passed: Lease with ID Lease not found. not found in the database.
```