Parking System Classes for

Master of Science in Information Technology

Software Design and Programming

Estell Moore

University of Denver College of Professional Studies

10/4/2025

Faculty: Nirav Shah, M.S.

Director: Cathie Wilson, M.S.

Dean: Michael J. McGuire, MLS

Abstract

This project implements a university parking system, by translating a UML class diagram into fully functional classes and objects. This paper reflects on the difficulties, as well as outlines the thought process behind the logic implementations made on the code. A test-drive approach guided the new logic implementations, and certain design choices were executed to keep data consistent with what was provided in the initial diagram. The final code is a faithful realization of the modeled diagram, while also implementing new logic that was required to keep in mind for the system.

CONTENTS

Introduction

This week's assignment required a transposition of a class diagram into functional classes regarding the designed parking system. Converting classes, attributes, and relationships from the diagram into code was straightforward; the real challenge was shaping the runtime behavior to satisfy the “things to keep in mind” without overwriting the original structure of the provided diagram.

Experience Translating from Class Diagram

The class diagram was straightforward, and fairly easy to translate to functional classes and methods. There were a few areas of confusion where it felt that the class diagram was lacking in areas compared to the code expectations of what needed to be kept in mind, but the overall implementation was simple and straightforward. Before taking into consideration the “Things to Keep In Mind”, the five classes were small, simple, and implemented easily.

Difficulties Following the Updates

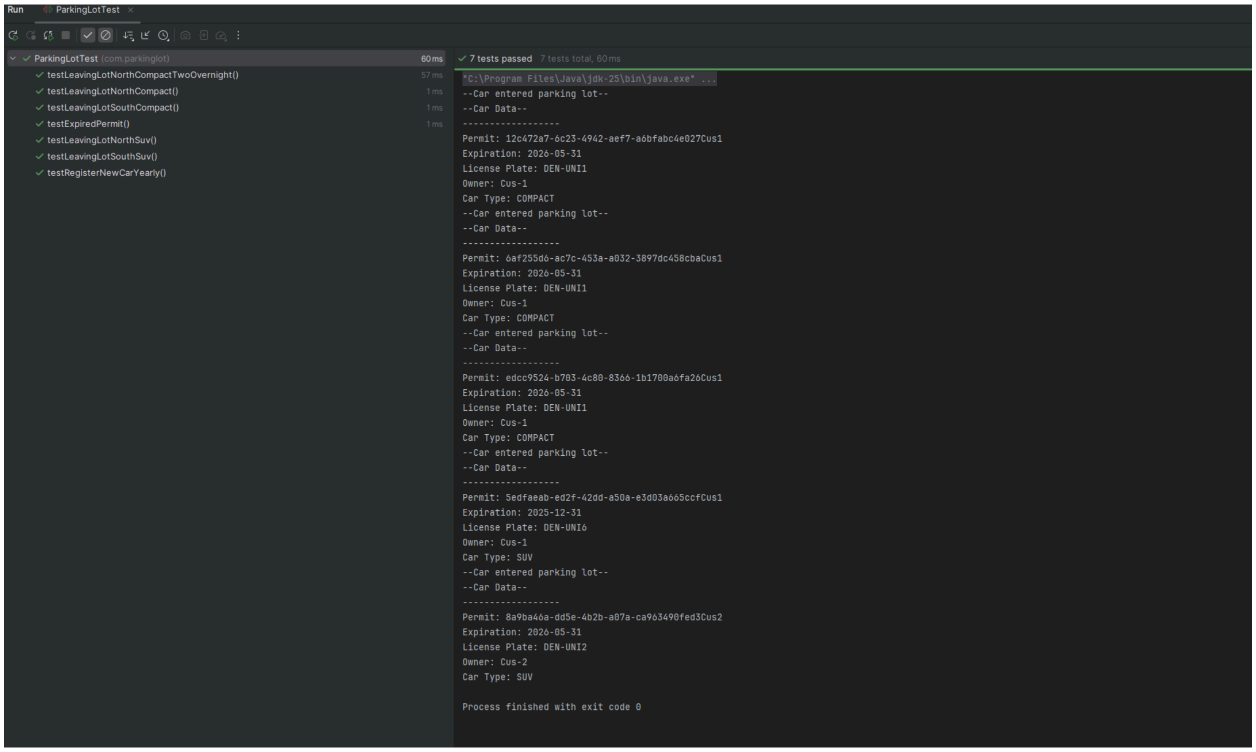
Upon reading the “Things to Keep in Mind” section of the assignment, I took that as an opportunity to try and iron out the classes to satisfy each bullet point provided. Because the class diagram was simple, I was able to model the newly implemented logic to also reflect the structure that the provided diagram portrayed. For example, instead of changing the return type of the entry() function in the ParkingLot.java, I created a getter that would instead be accessed to grab cost data, to keep return types consistent.

The biggest difficulty was finding out where each logical implementation would reside, as well as keeping the original UML model structure intact. I introduced new classes made to simplify accessors, but juggling multiple logic paths got complex without a complete picture. Something that helped me iron out the logic was the implementation of test-driven development approach (TDD) where I began to write a unit test that would outline what I wanted to test, then iterated on the code wherever the tests exposed gaps.

Conclusion

The minimal class diagram left room for richer behavior, but it also made it tricky to add features while preserving the original types and accessors. Adopting a test-driven approach enabled me to organize logic to satisfy the recommended implementation of parking lot system behaviors, while still maintaining the original framework of the expected classes.

Appendix A: Test Execution and Compilation



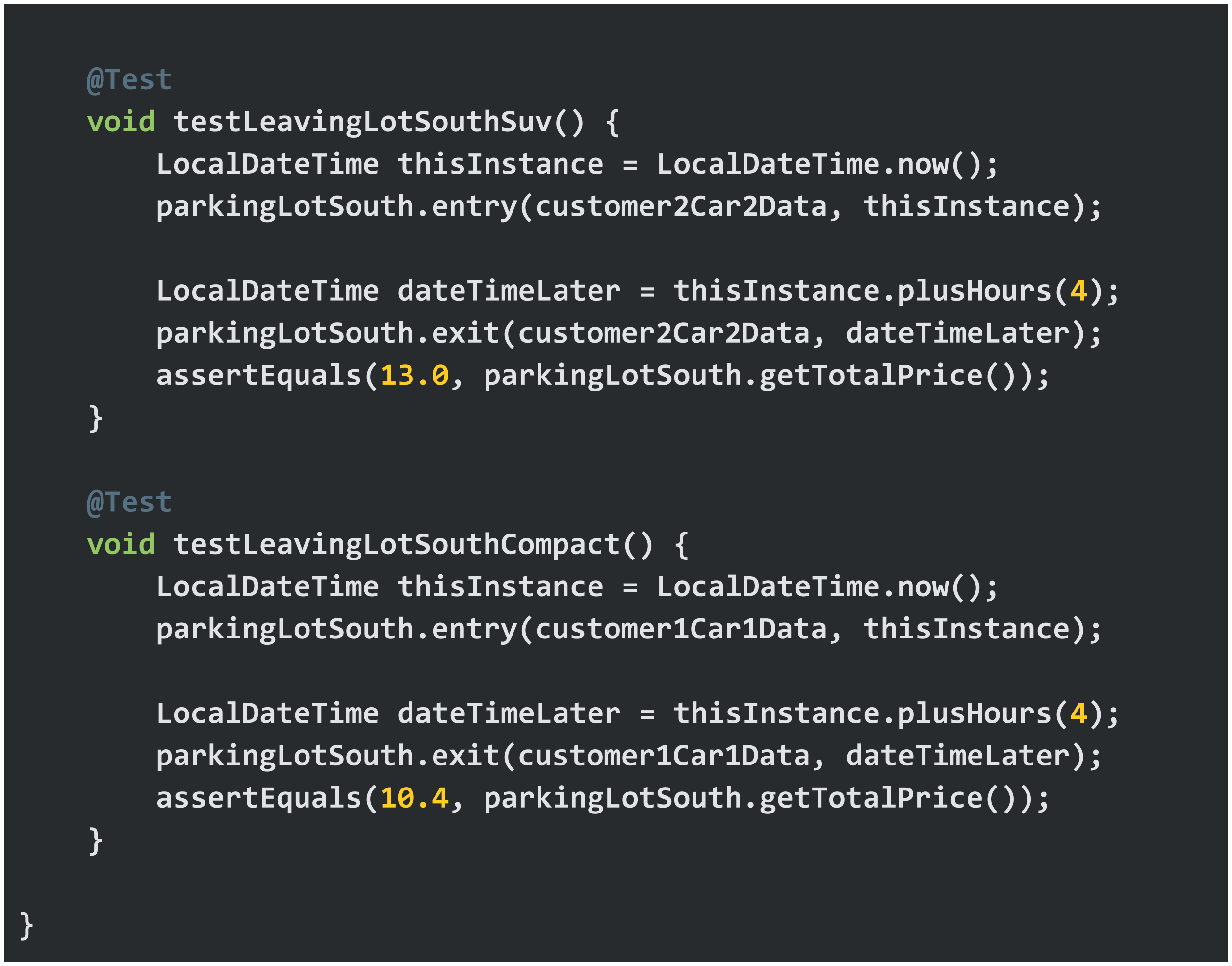
```
ParkingLotTest (com.parkinglot)                  60 ms
    testLeavingLotNorthCompactTwoOvernight()     57 ms
    testLeavingLotNorthCompact()                  1 ms
    testLeavingLotSouthCompact()                  1 ms
    testExpiredPermit()                           1 ms
    testLeavingLotNorthSuv()
    testLeavingLotSouthSuv()
    testRegisterNewCarYearly()
```

```
7 tests passed  7 tests total, 60 ms
"C:\Program Files\Java\jdk-25\bin\java.exe" ...
--Car entered parking lot--
--Car Data--
------------------
Permit: 12c472a7-6c23-4942-aef7-a6bfabc4e027Cus1
Expiration: 2026-05-31
License Plate: DEN-UNI1
Owner: Cus-1
Car Type: COMPACT
--Car entered parking lot--
--Car Data--
------------------
Permit: 6af255d6-ac7c-453a-a032-3897dc458cbaCus1
Expiration: 2026-05-31
License Plate: DEN-UNI1
Owner: Cus-1
Car Type: COMPACT
--Car entered parking lot--
--Car Data--
------------------
Permit: edcc9524-b703-4c80-8366-1b1700a6fa26Cus1
Expiration: 2026-05-31
License Plate: DEN-UNI1
Owner: Cus-1
Car Type: COMPACT
--Car entered parking lot--
--Car Data--
------------------
Permit: 5edfaeab-ed2f-42dd-a50a-e3d03a665ccfCus1
Expiration: 2025-12-31
License Plate: DEN-UNI6
Owner: Cus-1
Car Type: SUV
--Car entered parking lot--
--Car Data--
------------------
Permit: 8a9ba46a-dd5e-4b2b-a07a-ca963490fed3Cus2
Expiration: 2026-05-31
License Plate: DEN-UNI2
Owner: Cus-2
Car Type: SUV

Process finished with exit code 0
```

Appendix B: JUnit Test for Parking System

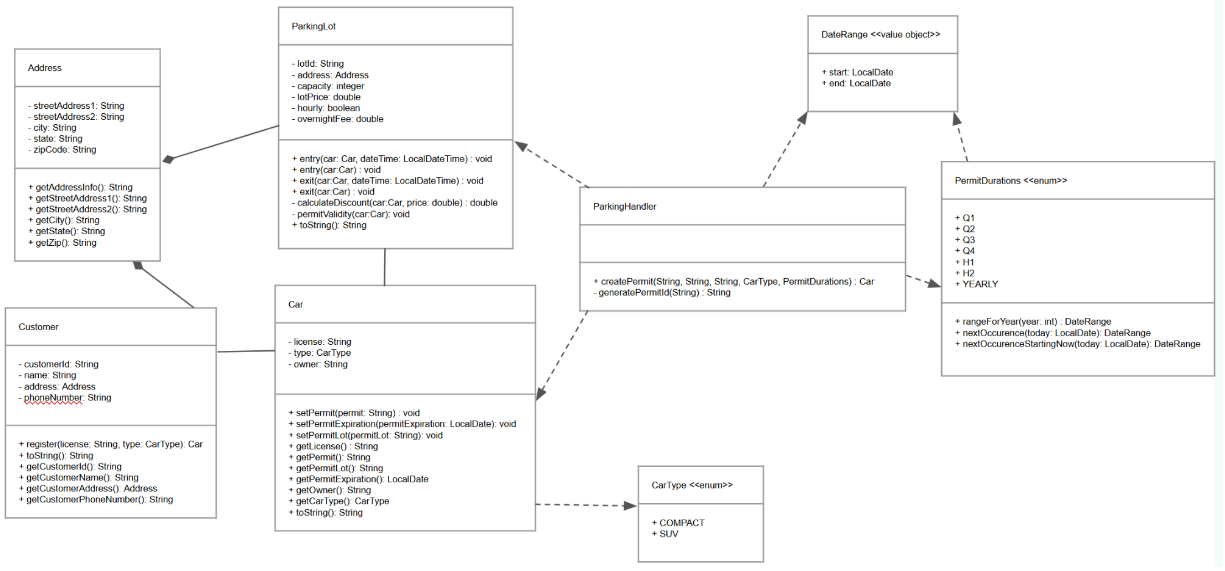
```java
package com.parkinglot;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import java.time.LocalDate;
import java.time.LocalDateTime;

import static org.junit.jupiter.api.Assertions.*;

public class ParkingLotTest {
    private ParkingLot parkingLotNorth;
    private ParkingLot parkingLotSouth;
    private Car customer1Car1Data;
    private Car customer1Car2Data;
    private Car customer2Car2Data;
    private Car customer2Car1;

    @BeforeEach
    void setUp() throws Exception{
        ParkingHandler handler = new ParkingHandler();

        // -- Set up parking lot rates --
        // North lot is not hourly but instantiates an overnight fee
        parkingLotNorth = new ParkingLot("PLN-1",
                new Address("123 Fake Street",
                        "", "Denver", "Colorado", "80210"),
                150, 15.50, false, 10.0);
        // South lot is hourly but has no overnight fee
        parkingLotSouth = new ParkingLot("PLN-2",
                new Address("456 Null Street",
                        "", "Denver", "Colorado", "80210"),
                250, 3.25, true);

        // Setup Customer 1's first car
        Customer customer1 = new Customer("Cus-1", "John Doe",
                new Address("123 West Drive",
                        "", "Denver", "Colorado", "80210"),
                "123-456-7890");
        Car customer1Car1 = new Car("DEN-UNI1", CarType.COMPACT,
customer1.getCustomerId());
        customer1Car1Data =
handler.createPermit(customer1.getCustomerId(), "PLN-1",
customer1Car1.getLicense(),
                customer1Car1.getCarType(), PermitDurations.YEARLY);
        // Setup the customer 1's second car
        Car customer1Car2 = new Car("DEN-UNI6", CarType.SUV,
customer1.getCustomerId());
        customer1Car2Data =
handler.createPermit(customer1.getCustomerId(), "PLN-1",
customer1Car2.getLicense(),
                customer1Car2.getCarType(), PermitDurations.Q2);

        // Setup Customer 2's first and only car
        Customer customer2 = new Customer("Cus-2", "Jane Smith",
                new Address("456 West Drive",
                        "", "Denver", "Colorado", "80210"),
                "098-765-4321");
        customer2Car1 = new Car("DEN-UNI2", CarType.SUV,
customer2.getCustomerId());
        customer2Car2Data =
handler.createPermit(customer2.getCustomerId(), "PLN-2",
customer2Car1.getLicense(),
                customer2Car1.getCarType(), PermitDurations.YEARLY);
    }

    @Test
    void testRegisterNewCarYearly() {
        // Test permit data
        assertTrue(customer1Car1Data.getPermit().contains("Cus1"));
        assertEquals("PLN-1", customer1Car1Data.getPermitLot());
        assertEquals(5,
customer1Car1Data.getPermitExpiration().getMonthValue());
        assertEquals(31,
customer1Car1Data.getPermitExpiration().getDayOfMonth());
```

```
        // Test Car data
        assertEquals("DEN-UNI1", customer1Car1Data.getLicense());
        assertEquals("Cus-1", customer1Car1Data.getOwner());
    }

    @Test
    void testExpiredPermit() {

customer2Car1.setPermitExpiration(LocalDate.now().minusDays(1));
        assertThrows(IllegalArgumentException.class, () ->
parkingLotNorth.entry(customer2Car1));
    }

    @Test
    void testLeavingLotNorthCompact() {
        LocalDateTime thisInstance = LocalDateTime.now();
        parkingLotNorth.entry(customer1Car1Data);
        parkingLotNorth.exit(customer1Car1Data, thisInstance);
        assertEquals(12.4, parkingLotNorth.getTotalPrice());
    }

    @Test
    void testLeavingLotNorthCompactTwoOvernight() {
        LocalDateTime thisInstance = LocalDateTime.now();
        LocalDateTime twoDaysLater = thisInstance.plusDays(2);
        parkingLotNorth.entry(customer1Car1Data, thisInstance);
        parkingLotNorth.exit(customer1Car1Data, twoDaysLater);

        assertEquals(28.4, parkingLotNorth.getTotalPrice());
    }

    @Test
    void testLeavingLotNorthSuv() {
        LocalDateTime thisInstance = LocalDateTime.now();
        parkingLotNorth.entry(customer1Car2Data);
        parkingLotNorth.exit(customer1Car2Data, thisInstance);
        assertEquals(15.5, parkingLotNorth.getTotalPrice());
    }
```



```
    @Test
    void testLeavingLotSouthSuv() {
        LocalDateTime thisInstance = LocalDateTime.now();
        parkingLotSouth.entry(customer2Car2Data, thisInstance);

        LocalDateTime dateTimeLater = thisInstance.plusHours(4);
        parkingLotSouth.exit(customer2Car2Data, dateTimeLater);
        assertEquals(13.0, parkingLotSouth.getTotalPrice());
    }

    @Test
    void testLeavingLotSouthCompact() {
        LocalDateTime thisInstance = LocalDateTime.now();
        parkingLotSouth.entry(customer1Car1Data, thisInstance);

        LocalDateTime dateTimeLater = thisInstance.plusHours(4);
        parkingLotSouth.exit(customer1Car1Data, dateTimeLater);
        assertEquals(10.4, parkingLotSouth.getTotalPrice());
    }

}
```

Appendix C: Updated Class Diagram


Address
- streetAddress1: String
- streetAddress2: String
- city: String
- state: String
- zipCode: String
+ getAddressInfo(): String
+ getStreetAddress1(): String
+ getStreetAddress2(): String
+ getCity(): String
+ getState(): String
+ getZip(): String
ParkingLot
- lotId: String
- address: Address
- capacity: integer
- lotPrice: double
- hourly: boolean
- overnightFee: double
+ entry(car: Car, dateTime: LocalDateTime) : void
+ entry(car:Car) : void
+ exit(car:Car, dateTime: LocalDateTime) : void
+ exit(car:Car) : void
- calculateDiscount(car:Car, price: double) : double
- permitValidity(car:Car): void
+ toString(): String
DateRange <<value object>>
+ start: LocalDate
+ end: LocalDate
ParkingHandler
+ createPermit(String, String, String, CarType, PermitDurations) : Car
- generatePermitId(String) : String
PermitDurations <<enum>>
+ Q1
+ Q2
+ Q3
+ Q4
+ H1
+ H2
+ YEARLY
+ rangeForYear(year: int) : DateRange
+ nextOccurence(today: LocalDate): DateRange
+ nextOccurenceStartingNow(today: LocalDate): DateRange
Customer
- customerId: String
- name: String
- address: Address
- phoneNumber: String
+ register(license: String, type: CarType): Car
+ toString(): String
+ getCustomerId(): String
+ getCustomerName(): String
+ getCustomerAddress(): Address
+ getCustomerPhoneNumber(): String
Car
- license: String
- type: CarType
- owner: String
+ setPermit(permit: String) : void
+ setPermitExpiration(permitExpiration: LocalDate): void
+ setPermitLot(permitLot: String): void
+ getLicense() : String
+ getPermit(): String
+ getPermitLot(): String
+ getPermitExpiration(): LocalDate
+ getOwner(): String
+ getCarType(): CarType
+ toString(): String
CarType <<enum>>
+ COMPACT
+ SUV