

**More Parking System Classes for**  
Master of Science in Information Technology  
Software Design and Programming

Estell Moore

University of Denver College of Professional Studies

10/19/2025

Faculty: Nirav Shah, M.S.

Director: Cathie Wilson, M.S.

Dean: Michael J. McGuire, MLS

### **Abstract**

This paper addresses the difficulties and design implementation decisions that accompanied factoring in new classes required for this week's assignment. There were many difficulties, which are further outlined, explaining that implementing old code with new code requires refactoring package structures, and migrating logic to simplify the repository to match expected class diagrams. Upon new unit tests being written, and new packages being created, the new classes were able to seamlessly be implemented into the existing repository, and ensures it utilizes every expected variable and function.

## CONTENTS

1. Introduction	3
2. Difficulties and Design Decisions	3
3. Conclusion	4
APPENDICES	5
Appendix A: Test Execution and Compilation	
Appendix B: Updated Class Diagram	
Appendix C: Sequence Diagram - Customer registering a Car and receiving a Parking Permit	
Appendix D: Sequence Diagram - Car entering a Parking Lot, resulting in the Customer incurring a charge	

## **Introduction**

Integrating new classes into an existing codebase is often challenging. Aligning new logic with preexisting structures can become complex when data types differ, static methods are involved, or responsibilities between components are unclear.

## **Difficulties and Design Decisions**

From the previous parking system assignment, I had followed the “Things to Keep In Mind” section. Though this was useful to have, it also posed difficulties implementing the new expected classes into the nomenclature and classes I currently have. There were a few big changes I needed to make, as well as reorganization of logic to fit the class diagrams provided for this assignment.

Originally, I had a class called ParkingHandler that handled the permit creation. Now, with the new classes, the ParkingOffice class took on the registration of a new permit, as well as a new customer. This was a fairly linear migration, luckily. Though, difficulties primarily lied in the integration of new expected variables to my existing logic. The ParkingLot class I had created needed some updating with the entry and exit functions I already implemented. I needed to utilize the new ParkingCharge class to handle the calculations of entry and exit costs, using the Instance time variable instead of LocalDate. This required me to manipulate ParkingLot to primarily be a class that instantiates parking lot data, as well as instantiates the calls to the calculations to set.

Since there is a significant amount of complex classes, a reorganization of packages needed to occur. The packages I had split the new classes into are as follows:

- Core - Main classes to instantiate objects
- Enums - Enum classes that hold data for the project
- Service - The parking office is the primary service for the parking logic
- Utils - Utilities the classes utilize, such as calculations of charges

By splitting up the classes into respective packages, it became more clear as to how each class is expected to function.

### **Conclusion**

Even with the preexisting code, the new required classes were able to be implemented with the appropriate logic and updated classes. Effective approaches of writing unit tests, reorganizing code into explicit packages and refactoring existing classes into the required new classes proved effective for linking together old versus new code.

## Appendix A: Test Execution and Compilation

### ParkingCharge Integration Test Results

✓ ParkingChargeIT (com.parkinglot.utils)	37 ms	✓
✓ testExitCharge()	37 ms	
✓ testEntryCharge()		

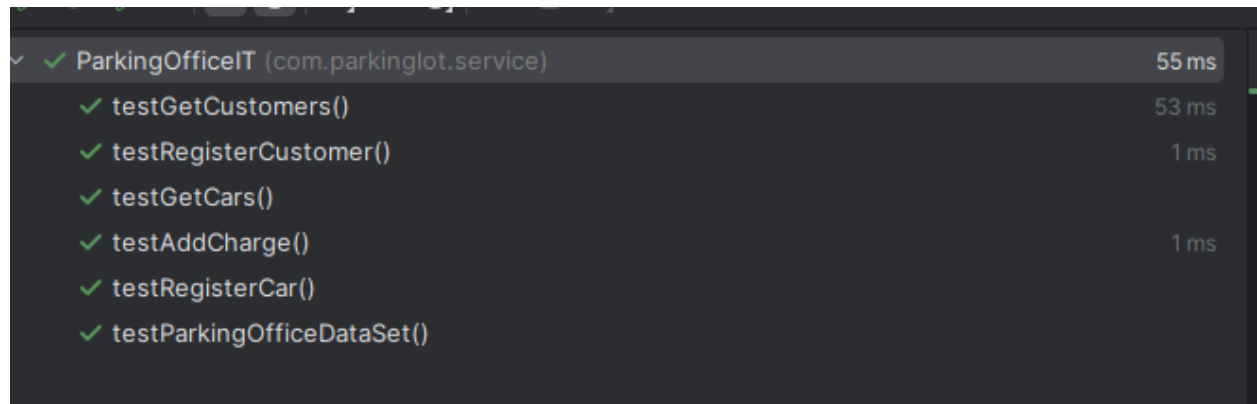
### ParkingLot Unit Test Results

✓ ParkingLotTest (com.parkinglot.core)	431 ms	✓
✓ testEntry()	429 ms	
✓ testExit()	2 ms	

### ParkingLot Integration Test Results

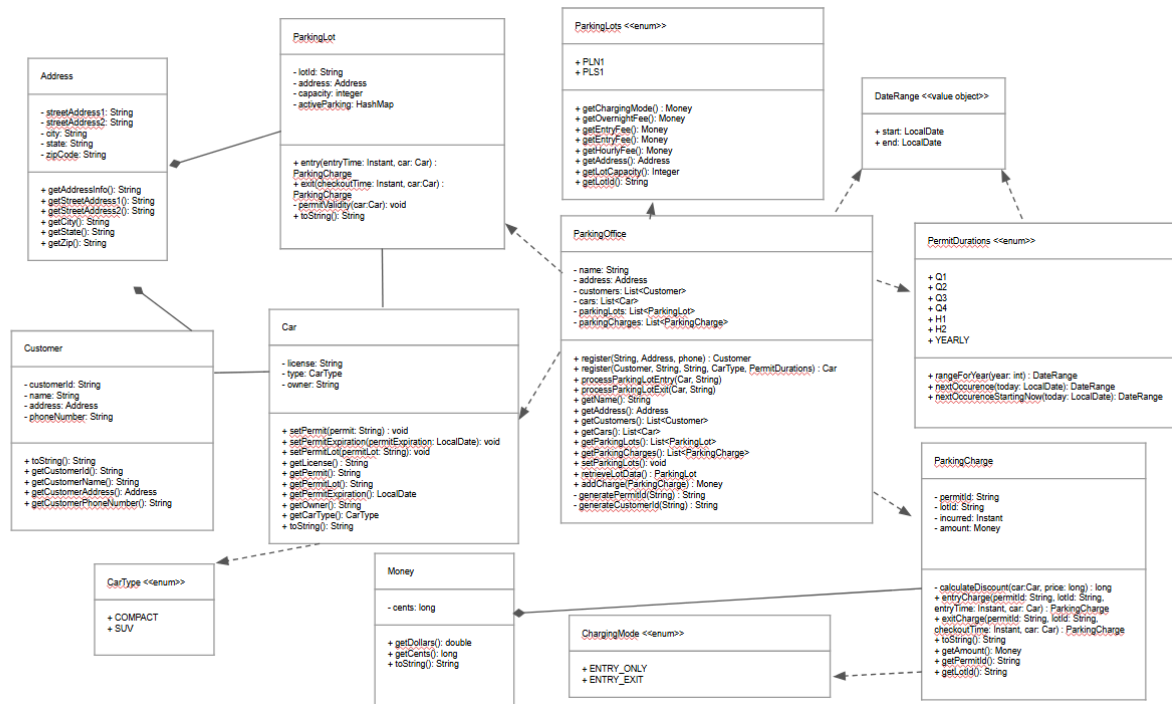
✓ ParkingLotIT (com.parkinglot.core)	60 ms	✓
✓ testLeavingLotNorthCompactTwoOvernight()	55 ms	
✓ testLeavingLotNorthCompact()	1 ms	
✓ testLeavingLotSouthCompact()	1 ms	
✓ testExpiredPermit()	1 ms	
✓ testLeavingLotNorthSuv()	1 ms	
✓ testLeavingLotSouthSuv()	1 ms	

## ParkingOffice Integration Test Results

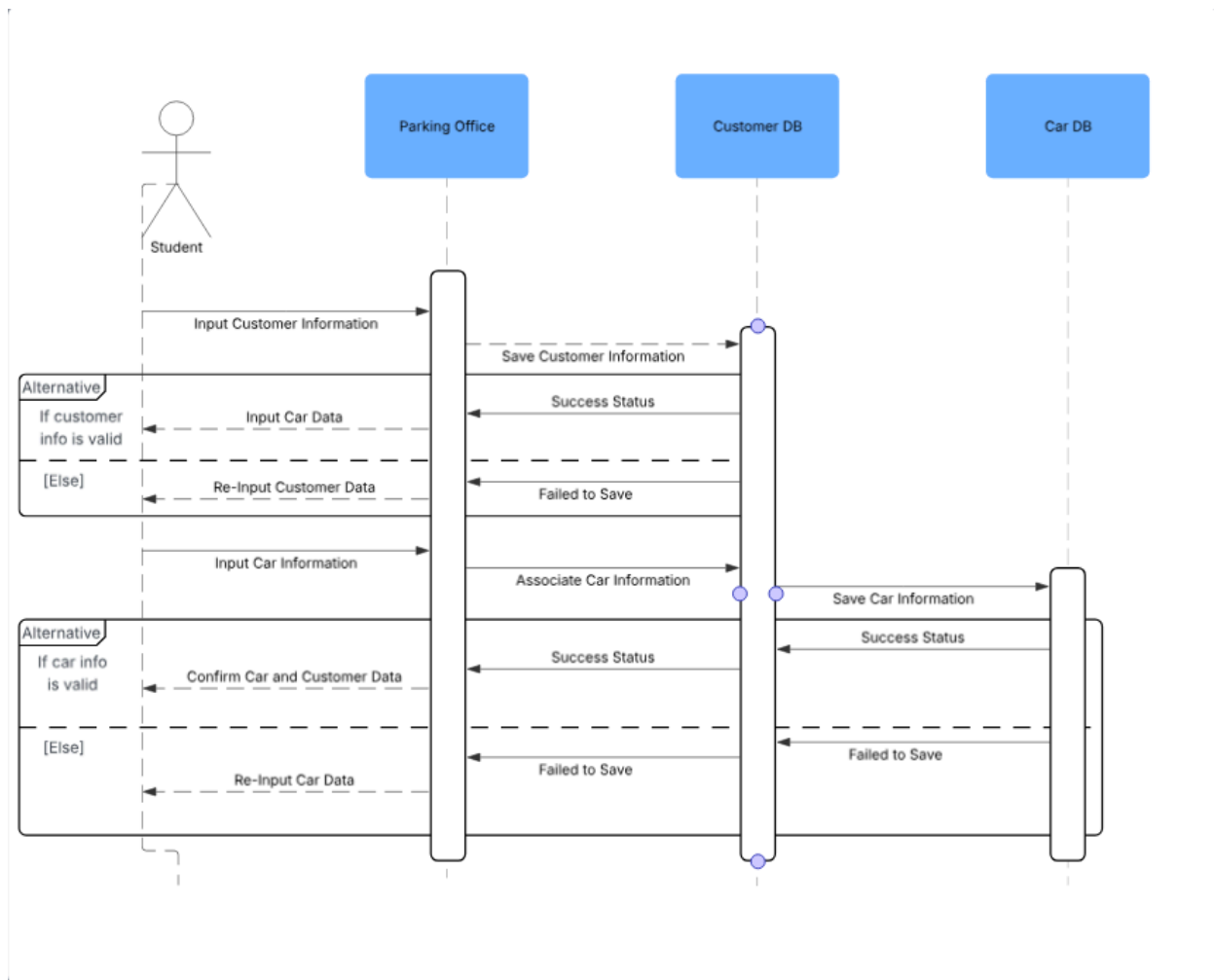
A screenshot of a JUnit test runner interface showing the results of integration tests for the ParkingOfficeIT class. The interface has a dark theme. At the top, a summary row shows a green checkmark, the class name 'ParkingOfficeIT (com.parkinglot.service)', and a total duration of '55 ms'. Below this, a list of individual test methods is shown, each with a green checkmark, the method name, and its duration. The methods are: testGetCustomers() (53 ms), testRegisterCustomer() (1 ms), testGetCars() (1 ms), testAddCharge() (1 ms), testRegisterCar() (1 ms), and testParkingOfficeDataSet() (1 ms).

✓	ParkingOfficeIT (com.parkinglot.service)	55 ms
✓	testGetCustomers()	53 ms
✓	testRegisterCustomer()	1 ms
✓	testGetCars()	1 ms
✓	testAddCharge()	1 ms
✓	testRegisterCar()	1 ms
✓	testParkingOfficeDataSet()	1 ms

## Appendix B: Updated Class Diagram





**Appendix C: Sequence Diagram - Customer registering a Car and receiving a Parking Permit**

**Appendix D: Sequence Diagram - Car entering a Parking Lot, resulting in the Customer incurring a charge**

