Portfolio Assignment

Software Design & Programming

Object-Oriented Mthd &Pgm I

Joseph DiBiasi

University of Denver College of Professional Studies

11/16/2025

Faculty: Nirav Shah, M.S.

Director: Cathie Wilson, M.S.

Dean: Bobbie Kite, MLS

Introduction

Universities with a large campus will have an equally large parking system to accommodate students and faculty. A parking application has been designed to help make managing this parking system a little bit easier. The parking application has been designed around the central parking office to allow it to interact with the surrounding classes. Features have been designed from the initial use cases developed for the parking system. The university needs to be able to register customers and assign them one or more car permits before customers are able to use the university parking lots. These parking lots vary between hourly and daily rates for charging. Daily lots scan car permits only on entry to determine charge rate, though cars left overnight in daily lots incur additional charges; hourly lots also scan car permits on exit. Scanning the car permits allows the university to keep track of customer charges, customers are billed monthly for all expenses; compact cars receive a discounted price of 20%.

Class Design

The parking office is the central class of the parking system. This class contains metadata about the parking office, the list of all customers registered with the university, and all owned parking lots. The functionality for managing all permit and parking transactions has been decoupled from the parking office class. Instead, the actions can be initiated here but the processing logic will be handled by the permit manager and transaction manager classes respectively. The permit manager is responsible for registering all customers and their car permits. The permit manager class does not store car permits as the current structure has these stored with the customer and retrieved based on the unique car permit id. The transaction manager is responsible for managing all logic pertaining to customers entering and exiting the university parking lots. The parking charges are also stored with the transaction manager. This allows the transaction manager to keep track of all parking charges by car permit. The parking office class is then able to use the transaction manager to calculate all monthly bills for customers. Since these charges are initially calculated per permit and not by customer, the bill will be discounted by 20% if the permit is for a compact car. All bills per customer will then be aggregated in the event that a customer has more than one permit registered to their name.

Lessons Learned

The main benefit of working on this project was developing a solid design plan for the class structures and functions. While I always appreciate a good design document to base my work off of, it was a good experience getting to repeatedly practice developing one myself. This was initially very challenging and time consuming, but it became easier with practice. Another big benefit to this process was the finding of specific tools such as LucidChart that helped with both the speed and style of designing planning diagrams.

The only problem with constantly designing and redesigning the application based on changing functional requirements was that the overall use case goals were presented early on in the process. The assumption was that these needed to be incorporated into the design of the initial classes. While this allowed me to gain more experience designing classes in a unique way, this also resulted in a lot of code refactoring that could have been avoided if I simply did not try to achieve all those goals from the start.

Future Enhancements

Previously alluded to as a lesson learned; the changing environment of the parking system started as a tightly coupled application centered around a few classes before evolving into one more resembling a typical MVC framework. Future enhancements to the parking system would be centered around that continuing theme. The parking office is a central hub for actions but not the logic, this class could be converted into a parking office controller that receives the input from the university systems, such as when a user scans a permit. The permit and transaction managers would continue their evolution into the true service layer handling all processing needs. This would allow a solid separation of concerns for organizational purposes.

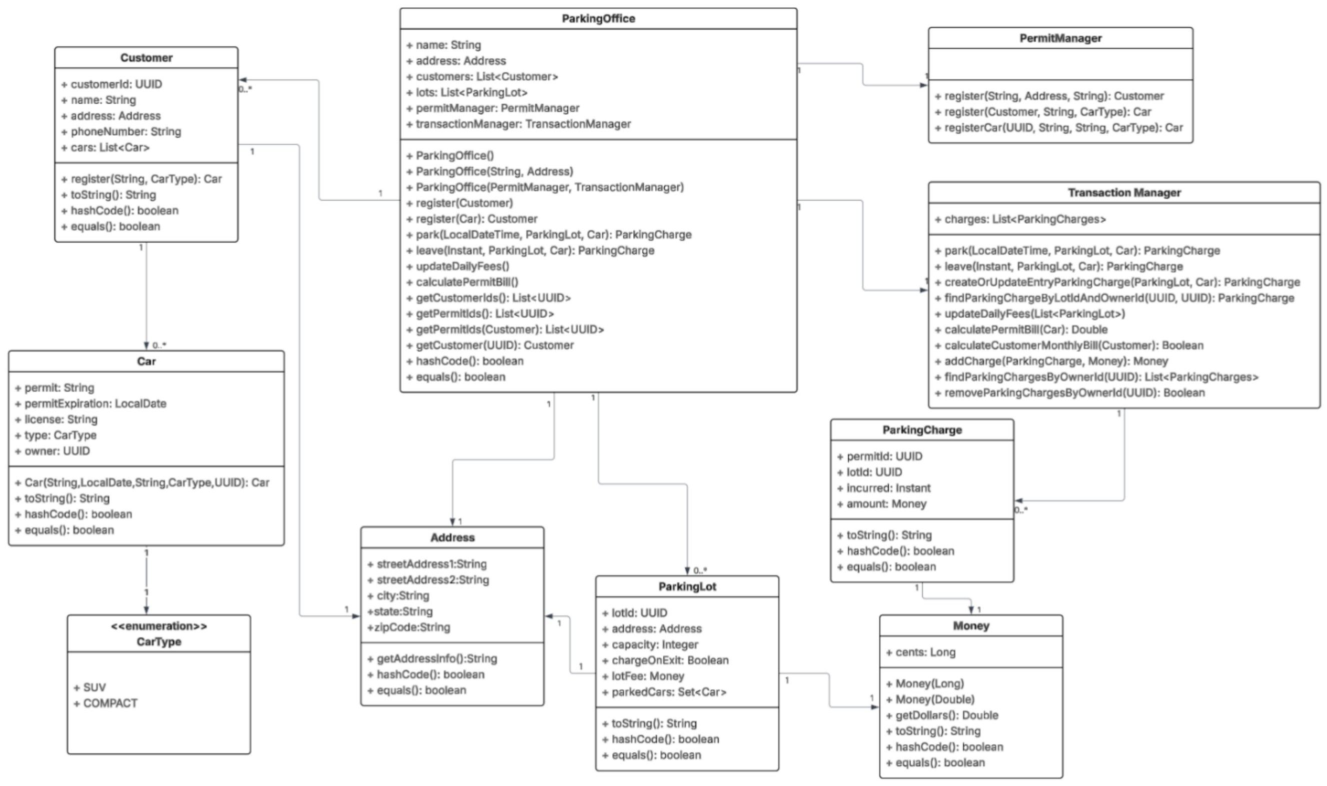
Summary

The design and development of the university parking system provided a solid foundational overview of the software development process. Use cases were developed early that provided a solid direction for the eventual code implementation. This meant that even when the class design structure needed more evolution, we were still able to write initial logic to meet the goals of the use cases. As the parking system evolved, so to could our existing logic be modified to fit into the new structure. This resulted in a living development process where the existing methods would be continually refined without ever losing their relationships which grounded them to the initial design. It was enjoyable to build an application from its bare bones into one which more resembled a modern system. I look forward to continuing development of the parking system in the subsequent object-oriented programming class.

References

Microsoft Copilot. 2025. “Microsoft Copilot.” Microsoft Copilot. Microsoft. 2025. https://copilot.microsoft.com/.

Appendix A: Class Diagram


Customer
+ customerId: UUID
+ name: String
+ address: Address
+ phoneNumber: String
+ cars: List<Car>
+ register(String, CarType): Car
+ toString(): String
+ hashCode(): boolean
+ equals(): boolean
ParkingOffice
+ name: String
+ address: Address
+ customers: List<Customer>
+ lots: List<ParkingLot>
+ permitManager: PermitManager
+ transactionManager: TransactionManager
+ ParkingOffice()
+ ParkingOffice(String, Address)
+ ParkingOffice(PermitManager, TransactionManager)
+ register(Customer)
+ register(Car): Customer
+ park(LocalDateTime, ParkingLot, Car): ParkingCharge
+ leave(Instant, ParkingLot, Car): ParkingCharge
+ updateDailyFees()
+ calculatePermitBill()
+ getCustomerIds(): List<UUID>
+ getPermitIds(): List<UUID>
+ getPermitIds(Customer): List<UUID>
+ getCustomer(UUID): Customer
+ hashCode(): boolean
+ equals(): boolean
PermitManager
+ register(String, Address, String): Customer
+ register(Customer, String, CarType): Car
+ registerCar(UUID, String, String, CarType): Car
Transaction Manager
+ charges: List<ParkingCharges>
+ park(LocalDateTime, ParkingLot, Car): ParkingCharge
+ leave(Instant, ParkingLot, Car): ParkingCharge
+ createOrUpdateEntryParkingCharge(ParkingLot, Car): ParkingCharge
+ findParkingChargeByLotIdAndOwnerId(UUID, UUID): ParkingCharge
+ updateDailyFees(List<ParkingLot>)
+ calculatePermitBill(Car): Double
+ calculateCustomerMonthlyBill(Customer): Boolean
+ addCharge(ParkingCharge, Money): Money
+ findParkingChargesByOwnerId(UUID): List<ParkingCharges>
+ removeParkingChargesByOwnerId(UUID): Boolean
Car
+ permit: String
+ permitExpiration: LocalDate
+ license: String
+ type: CarType
+ owner: UUID
+ Car(String,LocalDate,String,CarType,UUID): Car
+ toString(): String
+ hashCode(): boolean
+ equals(): boolean
ParkingCharge
+ permitId: UUID
+ lotId: UUID
+ incurred: Instant
+ amount: Money
+ toString(): String
+ hashCode(): boolean
+ equals(): boolean
Address
+ streetAddress1:String
+ streetAddress2:String
+ city:String
+state:String
+zipCode:String
+ getAddressInfo():String
+ hashCode(): boolean
+ equals(): boolean
ParkingLot
+ lotId: UUID
+ address: Address
+ capacity: Integer
+ chargeOnExit: Boolean
+ lotFee: Money
+ parkedCars: Set<Car>
+ toString(): String
+ hashCode(): boolean
+ equals(): boolean
Money
+ cents: Long
+ Money(Long)
+ Money(Double)
+ getDollars(): Double
+ toString(): String
+ hashCode(): boolean
+ equals(): boolean
<<enumeration>>
CarType
+ SUV
+ COMPACT
0..*
1
0..*
1
1
1
1
1
1
1
1
0..*
1
0..*
1
1
1
1
1
1
1
1