Week 3 Assignment: Implement the Parking Charge Calculator for

ICT 4315-1 Object-Oriented Methods & Paradigms II

Jack Hermanson

University of Denver University College

April 20, 2025

Faculty: Nathan Braun, MS

Director: Cathie Wilson, MS

Dean: Michael J. McGuire, MLS

Table of Contents

Introduction

This paper discusses the third assignment and my attempt to satisfy the requirements and learn about the strategy pattern, which is new to me. I chose 2 strategies:

1) Entry and Exit Charge Strategy
2) Entry Only Charge Strategy

The point of the strategy pattern is to change the behavior of some kind of algorithm at a program's runtime (Baeldung 2024). In this case, my thinking was that the lots that scan on entry and exit would have one type of behavior while lots that only scan on entry (and have patrols check them between 12am and 6am) would have another type of behavior. In the code, both would implement the same interface, so the calling method doesn't need to worry about it if the parking lot instance has the correct implementation set before the method is called.

I created an interface named Parking Charge Strategy, which defines a method named calculate charge, which takes a daily rate after discounts, an entry time, and an exit time.

Entry and Exit Charge Strategy

The entry and exit charge strategy makes the most intuitive sense to me. The parking lot knows when a permit was scanned on entry, and it hangs onto that information until the vehicle leaves. When the vehicle leaves, it looks at the entry and exit time, counts how many days the vehicle was parked there, and multiplies the discounted daily rate by the number of days. The exit method returns a transaction, since the assignment specifies that the transaction is created upon exiting the lot. The daily rate, entry time, and exit time are all passed to the calculate charge method, and they're all used. I didn't really have any problems with this.

Entry Only Charge Strategy

The entry only charge strategy makes sense conceptually, as I understand the idea of a lot only scanning on entry and relying on patrols to check the lots to see if anyone stayed parked overnight. However, implementing the strategy pattern here was not intuitive for me. It felt very odd to try to shoehorn this functionality into using the same interface as the entry and exit strategy. I also had to modify my Parking Lot code in a way that felt sloppy.

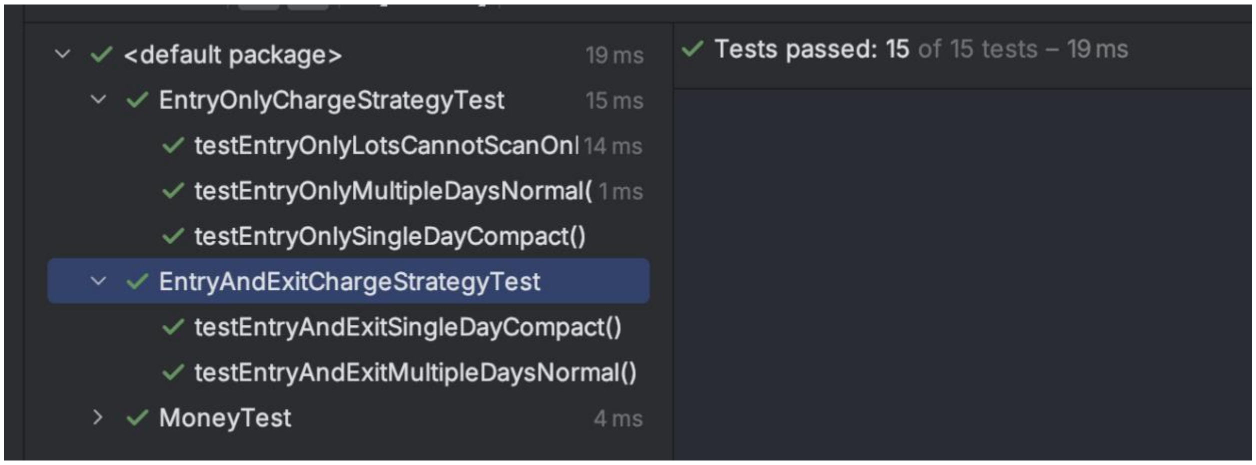
In the Parking Lot class, the scan on entry method has to return a Parking Transaction to make this logic work, since there is no scanning on exit, and the transaction is created upon entry (or patrol scan overnight). However, the entry and exit strategy does not want the transaction to be created upon entry—only upon exit. As a result, I had to make the code for this method return null if it's an entry and exit lot. Things get even sloppier in the implementation of the calculate charge method. Since the logic for this type of lot is that a transaction is created immediately upon entry, and another one is created when the patrols scan a vehicle overnight, you don't need the entry and exit times to calculate the amount to charge. You really only need the daily rate after discounts, which the parking lot already knows, so calling this method feels entirely pointless. The lot could simply create a transaction with its daily rate after discounts without the need for this strategy interface at all. I also felt the need to add a check that ensures the exit time is null in the implementation of this strategy, as these lots do not even scan on exit. This feels sloppy. If I need to enforce that something is null, then it shouldn't be a parameter at all. This feels too tightly coupled in a way that doesn't make sense. Either I misunderstood what this was asking me to do (likely) or it is poorly designed or both. After this assignment, I'd appreciate seeing an example of what *was* expected.

Conclusion

I would much rather have this logic within the Parking Lot class, allowing the entry and exit methods to figure out what they're supposed to do. You could even have each of the two have a method signature that says it will return a Parking Transaction, but have entry and exit lots return null on entry, and have the caller handle that properly. I kept going back and forth, starting over, rereading the assignment, rereading the discussion posts, and feeling frustrated when writing this code. A lot of my confusion comes from having an incomplete picture of how the different classes relate to each other, and which classes are responsible for doing what. I don't yet see how the transaction manager fits into this. I would prefer to be given a general set of requirements for this assignment and complete them in ways that make sense to me, rather than incomplete, piecewise, confusing instructions that make me feel like I am shooting in the dark as I write my code. The structure of the code is also confusing. It seems we are expected to put this code in a week 3 module, but a lot of the code for the parking lot was already written in week 1, so I copied it to week 3. Then my IDE warned me about repeated code, so I had to mark week 1 as excluded. This feels sloppy, and it would make more sense to iterate on one project and have "releases" rather than continuously add repeated code into the same repository.

I am very likely misunderstanding things and missing things. Other people seem to be figuring it out just fine. At this point, I just want to get through this class and be done with this program as soon as possible. I have no faith left in this program or its leaders, and that is my problem, not anyone else's. If others can get through this program using Chat GPT, I think my genuine best efforts (using my brain) will work, and I hope I got close enough with what I did in this assignment to receive an A and move onto the next one, hopefully with more clarity then.

My program compiles and my tests pass, as shown in the screenshot below.



```
✓ <default package>                          19 ms
  ✓ EntryOnlyChargeStrategyTest              15 ms
      ✓ testEntryOnlyLotsCannotScanOnl       14 ms
      ✓ testEntryOnlyMultipleDaysNormal(      1 ms
      ✓ testEntryOnlySingleDayCompact()
  ✓ EntryAndExitChargeStrategyTest
      ✓ testEntryAndExitSingleDayCompact()
      ✓ testEntryAndExitMultipleDaysNormal()
  ✓ MoneyTest                                 4 ms

✓ Tests passed: 15 of 15 tests – 19 ms
```

References

Baeldung. 2024. “Strategy Design Pattern in Java.” *Baeldung*. January 9, 2024. https://www.baeldung.com/java-strategy-pattern. Accessed April 20, 2025.