**Week 4 Assignment - CUSTOMER, CAR, ADDRESS, AND PARKING LOT CLASSES**

For

ICT-4305 Object-Oriented Method & Program I

Luther Chikumba

University of Denver University College

October 7, 2022

Instructor: Dr. Sherri Maciosek

Table of Contents

[Write up. 3](#_Toc116028293)

[Class Diagrams 5](#_Toc116028294)

[Screenshots: 8](#_Toc116028295)

[Reference: 12](#_Toc116028296)

# Write up.

For this week's assignment, I considered I had both; there were some things I found challenging and some I found easy. By now, writing classes and creating methods is something; I no longer think of as tricky, given that I had a bit of java background in my pocket before. Let us start with what was difficult; I found it challenging to extract the information from the class diagram and then put it into code. It did not click on me immediately until I had all the classes defined. Now the real challenge came into play when I had to figure out how the other classes have a relationship with each other. For instance, Car belongs to Customer or Address belongs to Customer, also noticing a relation between the Car and the Address. All those little details did not come to me instantly. So for me, connecting those dots when I looked at the class diagram, it just did not click right away, but after hours of iterating on the system, it started coming together piece by piece until I could write the rest of the system.

On the other hand, what I found easy was writing the tests. I think what makes testing easy for me is the way I process the objective of tests in general. Usually, when I look at a method, I think of each method as two-sided. You have the value you expect and the actual value. Once you have a value you expect to get from your method, you have another side of that coin: an actual value you will get when you call your methods. So, thinking it that way makes it easy because the first question I ask myself when I want to assert a method is what am I testing, and what is the return value? If I know those two things, it does give me a head start.

What helped was the videos from the first week of the course. Although I had a little experience with Java from my undergrad, those videos put things into perspective, and it just became even more convenient.

I wish I had known composition before. From this week's material, when I was reading about inheritance vs. composition, it made me wonder why I was not taught inheritance vs. composition. When I was writing these classes, I was getting ideas on where to apply composition because I could see the code duplication all over these classes in the places, I could have used inheritance.

Since it was not explicitly mentioned that we needed the scanner class to input the value for this week's assignment, I decided to hard-code my input values. The reasoning behind this implementation was that I did not want to add any more complexity and lines of code by adding the operation of asking the user to enter input. Also, I like the philosophy of failing fast and getting feedback quickly. Having a user type in the input slows me down while iterating the code, the way I did it is a bit quicker.

# Class Diagrams

|  |
| --- |
| ParkingLot |
| lotId: String address: String capacity: integer |
| entry(Car): void |

|  |
| --- |
| CarType <<enum>> |
| COMPACT SUV |

|  |
| --- |
| Car |
| permit: String permit expiration: LocalDate\*\* license: String type: CarType owner: String (customer id) |
|  |

|  |
| --- |
| Customer |
| customerId: String name: String address: Address phoneNumber: String |
| register(license: String, type: CarType): Car |

|  |
| --- |
| Address |
| streetAddress1: String streetAddress2: String city: String state: String zipCode: String |
| getAddressInfo(): String |

# Screenshots:

Main Class

Text

Description automatically generated

AddressTest Class

Text

Description automatically generated

CarTest Class

Graphical user interface, text

Description automatically generated

CustomerTest Class

Text

Description automatically generated

ParkingLotTest Class

Text

Description automatically generated

# Reference: