Week 1 Assignment - ASSIGNMENT: TRANSLATING UML INTO CODE

For

ICT- 4315 Object-Oriented Method & Program II

Luther Chikumba

University of Denver University College

January 8, 2023

Instructor: Dr. Mike Prasad

Table of Contents

Implementation

This week's assignment was constructing another layer for the parking system project. From last quarter, most of the implementation was driven by the parking office class. One would state that ParkingOffice was the back-born class for the university parking system. This week I had to add another layer to the university parking system, the ParkingServices. With the ParkingService, it will almost become the deriver class because it also introduces new functionality: accepting different commands from other classes to perform the functionality. In a nutshell, the ParkingService layer will drive and perform the registration of new customers and cars.

Design

Given the UML design and the class diagram provided in this week's assignment, I also had to assume how I would implement the design. I used the reverse engineering technique, which was first to establish the assignment's end goal of the project and then work backward to get to that goal. For me, it allows me to understand how different components work together and how they can go about achieving that goal. ParkingOffice always had the functionality to register a new customer and car. With that functionality, we needed to add RegisterCarCommand and RegisterCustomerCommand, which allows choosing which action to perform, whether adding a new customer or car. These commands would then let the system know what to do. These commands will not work alone; they need other classes that invoke them to maintain the loosely coupled infrastructure we have. This is where the ParkingServices class comes in handy. ParkingServices class will control the flow of these actions, which will

then be the one that performs these two commands, RegisterCustomerCommand and the RegisterCarCommand. That will be its only responsibility and nothing else. To keep our classes loosely coupled, I briefly mentioned that the system would interact with the user. Therefore, that means we needed a Command interface. The interface will have methods allowing the user to get the command they want to perform, display the name, and execute it.

Improvements

My university parking system has quite a lot of improvements that are needed. For most of this week, I spent a significant amount of time trying to understand the flow of the UML diagram provided to us. With that time all spent on understanding the design and the requirements, I had very little time to implement all the required functionality and testing to ensure my university parking system was working correctly. With that, these are the most critical improvements needed in my implementation: in RegisterCarCommand class, I am missing all the implementation of the Command interface class.

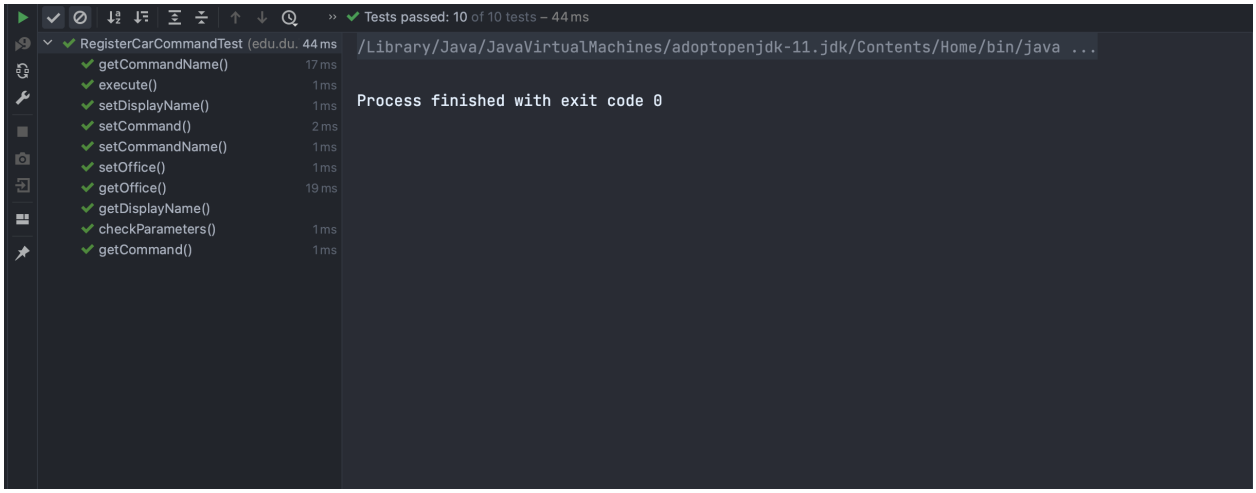
Difficulties

I had many challenges starting this week's assignment. Most of my confusion came from trying to use the knowledge I gained from the last course in this course in the sense that the requirements had more specific instructions that were less open to interpretation of what was needed. It could be that I was overthinking the assignment, or maybe I did not understand it. With that, I had to think of something to remedy that. I asked questions that I thought would help me understand the assignment better. To the best of my knowledge, I went back to the drawing board to see how best I could translate what I saw from the UMLs and class diagram. Another challenge I faced was: given the class diagrams, I still kept wondering if that was all needed to complete the assignment or if we were also required to add more implementation and design. There was not any mention of adding necessary class methods to help with what we had already been given, which made it very difficult to try and work with what we had been given.

Testing

My testing was incomplete. I only tested the class methods I had implemented: the RegisterCarCommand and RegisterCustomerCommnd.

RegisterCarCommandTest



RegisterCustomerCommandTest

Conclusion

In conclusion, getting a brief description of each class's implementation would have been valuable. Though the UML and class diagrams should be enough of what was required to complete the assignment, I do not think I understood or know how to write the code implementation without some of that description. I kept thinking that translating the UML and Class diagrams would be far much easier only if I knew and understood the holistic view of how these classes were supposed to interact with each other. Yet I did not understand how these individual classes were supposed to work together to achieve one goal: registering a customer or a car.