University Parking System: Translating UML into Code

for

Master of Science

Information Technology

Kassandra Vega Lucero

University of Denver College of Professional Studies

April 6, 2025

Faculty: Nathan Braun, MBA

Dean: Michael J. McGuire, MLS

Table of Contents

Background

The University is developing a new parking system that allows students and community members to park in their parking lot as long as they register with the University's Parking Office. After using the parking lots, customers will receive a bill that includes their parking lot transactions; customers will pay their monthly bill outside the parking lots. The transactions factor in several components, such as which parking lot the customer entered and the time spent in the parking lot. The transactions also account for the type of vehicle the customer drives and the discount associated with using a compact car. The development of this parking system has been completed in stages, and this development stage focuses on implementing and testing the ParkingService, RegisterCarCommand, and RegisterCustomerCommand classes. With those classes, the stage also implements and tests the Command interface. The development of the new components uses a UML diagram as its foundation.

UML Diagrams

The addition of the new classes to the continuation of the University Parking System was initiated with a UML diagram. The diagram presents a layer on top of the ParkingOffice class involving the ParkingService, which is connected to RegisterCarCommand and RegisterCustomerCommand, both of which have a relationship to the Command Interface as shown in Figure 1.

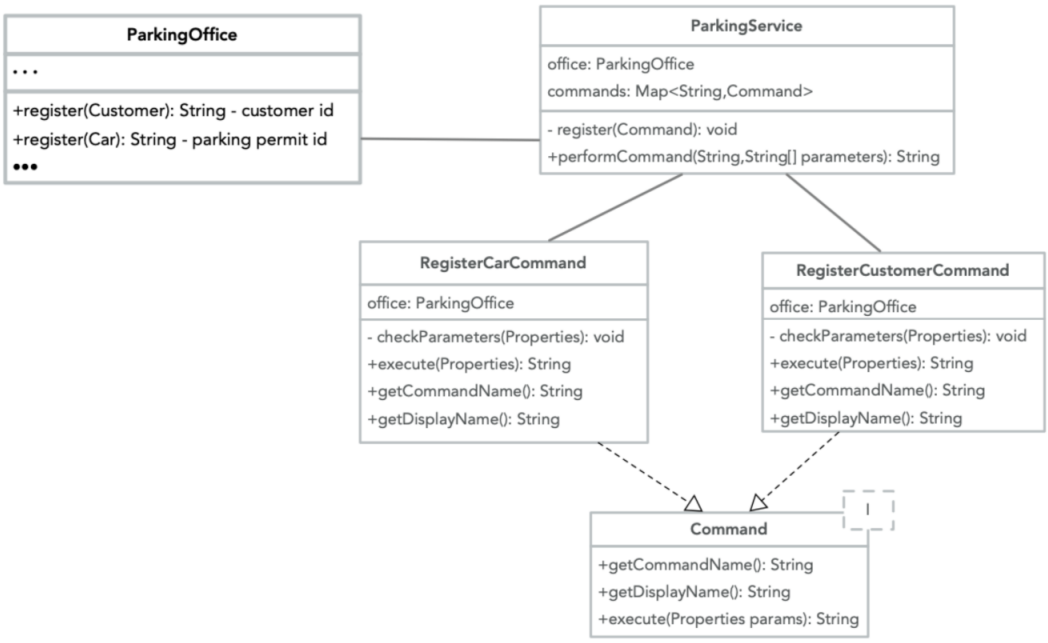

Figure 1. UML Diagram of the classes to be implemented.

The diagram above served as a foundation for designing and developing the parking system when I created the code in Java. I used Java 17.0.13 and Junit 4.13.2 to create the classes and their respective tests and included them to the pre-existing system as shown in the UML class diagram in Figure 2.

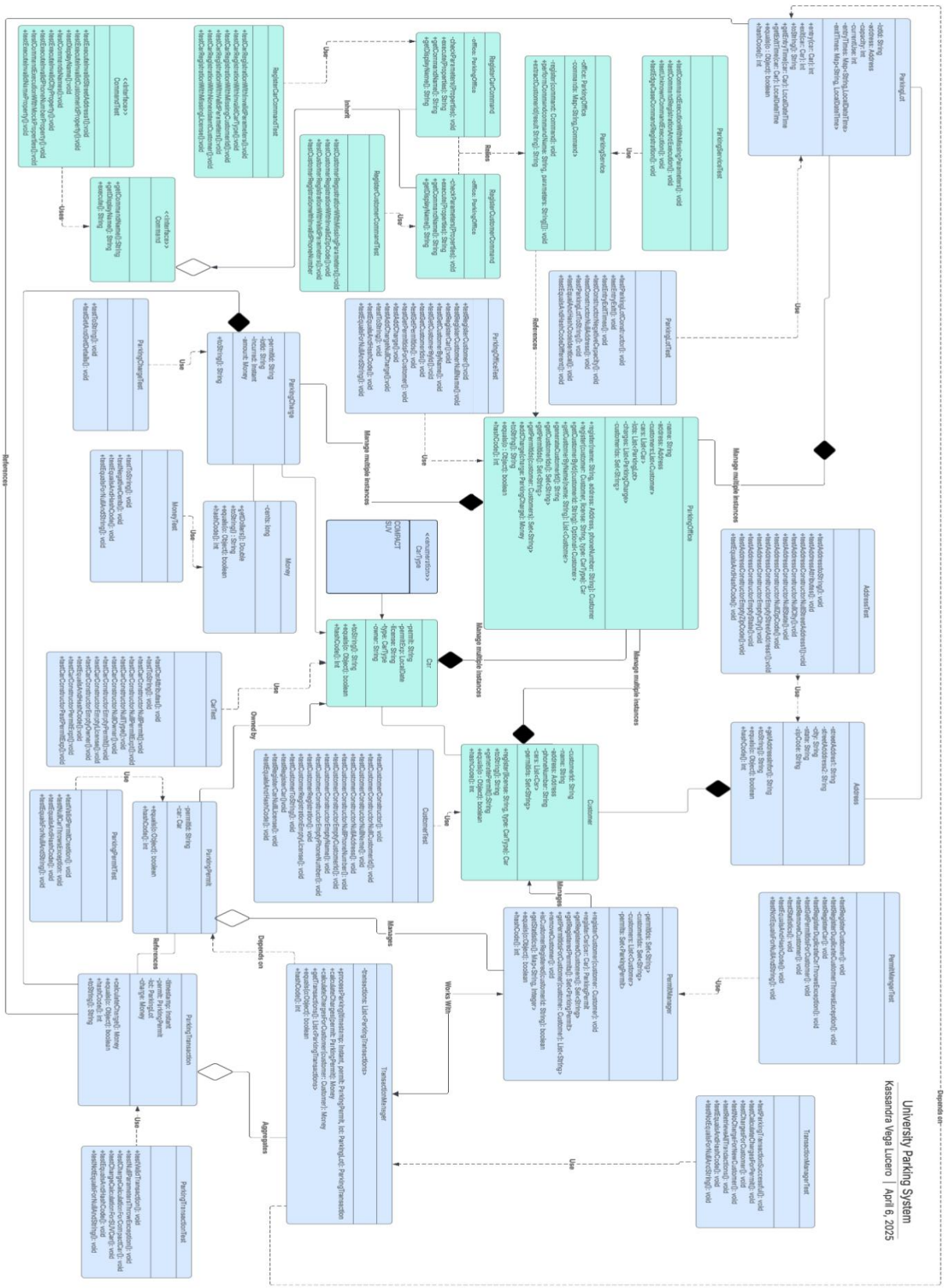

Figure 2. UML diagram with the focus classes in green.

Class Implementations

As I began the implementation process, I noticed all the classes had a rather familiar appearance—that is all but the Command class which had an “I” in the corner to signal the class is an interface. Due to this difference, I created that class first with the page on “Java Interface” (w3schools n.d.) which provide example on how to create the interface. I had originally used a Map for input handling but was running into compilation errors and ended up using the Properties class instead. When switching to properties, I had greater flexibility when it came to handling the inputs. Despite the lack of variety in the inputs and having used Maps before, I was having issues with the Map and its organization and could not uncover what the issue was. I then returned to the UML diagram and realized that Properties was a class I could use and did not stand for attributes. I used a combination of trial and error and “Getting Started with Java Properties” (2024) to understand and use properties and key values.

Unit Tests

There was also a bit of trial and error during the Unit testing. I usually began the unit testing by validating that a best-case scenario was a successful test. This was generally in the form of a constructor and valid parameters. I then went on to try various scenarios where the parameters were invalid entries. Invalid entries consisted of nonexistent entries, missing values, using invalid characters in the parameter, or having the incorrect length within the parameters. Doing so allowed me to realize my prior versions were not preventing phone numbers from having letters, as well as from my phone numbers and zip codes, to be longer than they should be. With this discovery, I am aiming to continue refining my error handling for those values such that phone numbers will be acceptable if they have paratheses around the area code or if they

use dashes to divide the sections. I also aim to increase the edge cases tested in the unit tests to be more inclusive of inputs that will impact the system's ability to successfully complete transactions for customers.

Unit Test Visuals

With the relationships among the classes, the implementation of ParkingService, Command, RegisterCarCommand, and RegisterCustomerCommand resulted in making changes to the Customer, Car, and ParkingOffice classes. Below are screenshots of the Unit Test results for each of those classes to summarize their testing.

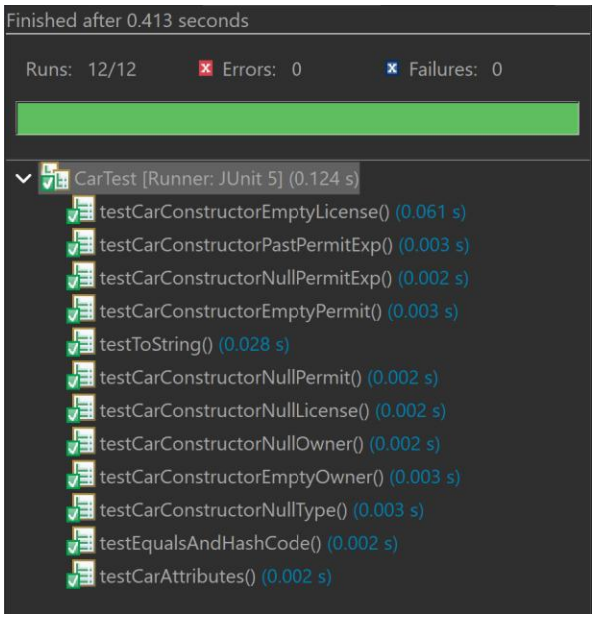

Figure 3. JUnit results for CarTest.java

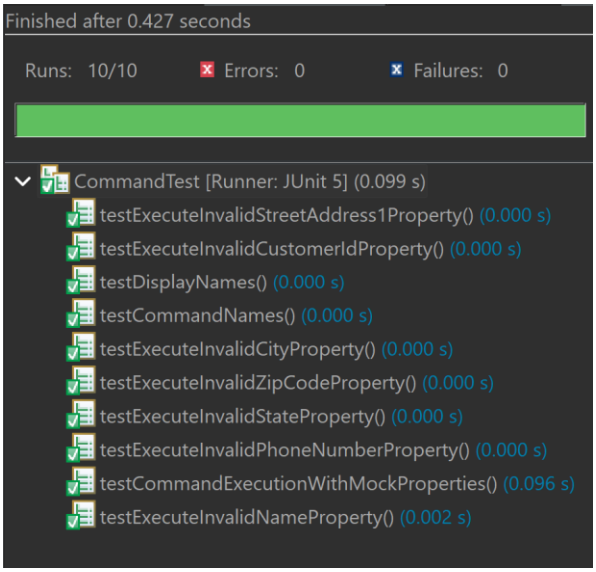

Figure 4. JUnit results for CommandTest.java

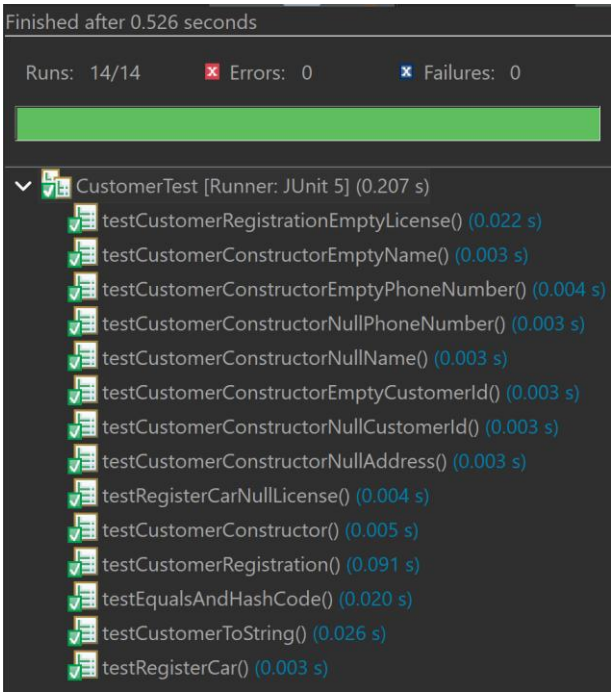

Figure 5. JUnit results for CustomerTest.java

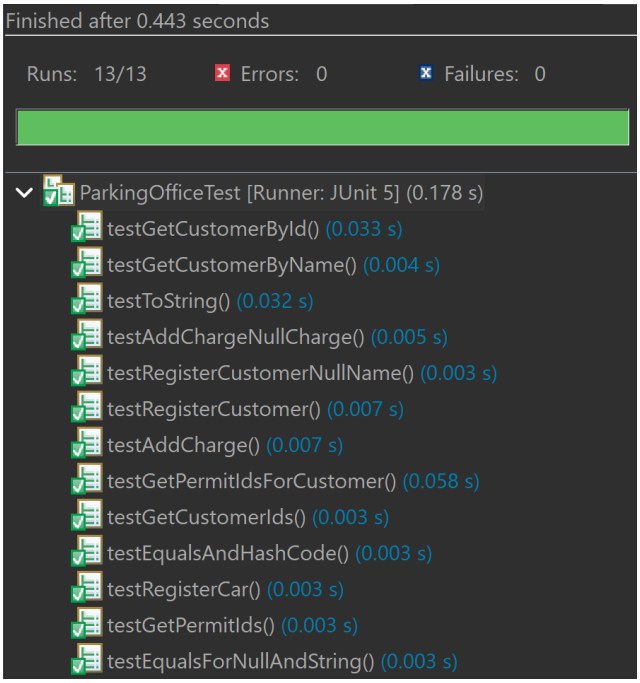

Figure 6. JUnit results for ParkingOfficeTest.java

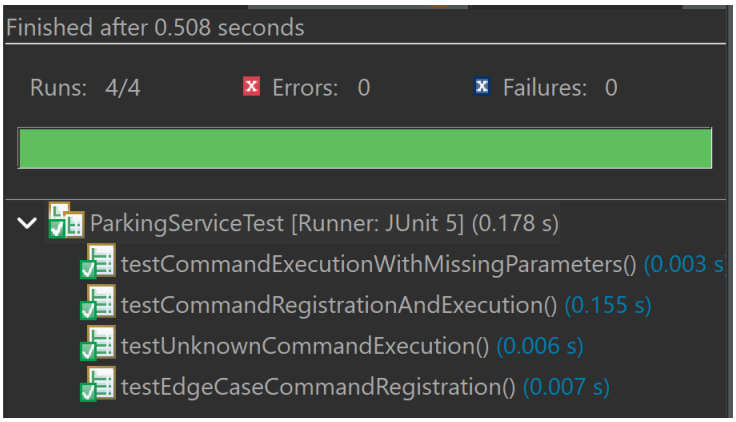

Figure 7. JUnit results for ParkingServiceTest.java

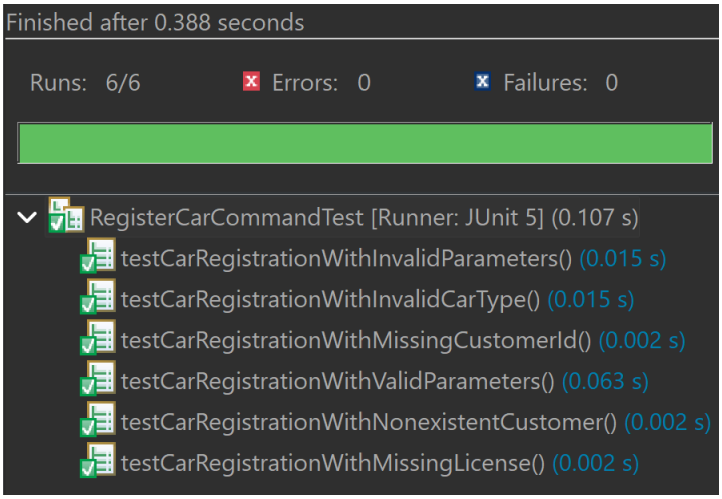

Figure 8. JUnit results for RegisterCarCommandTest.java

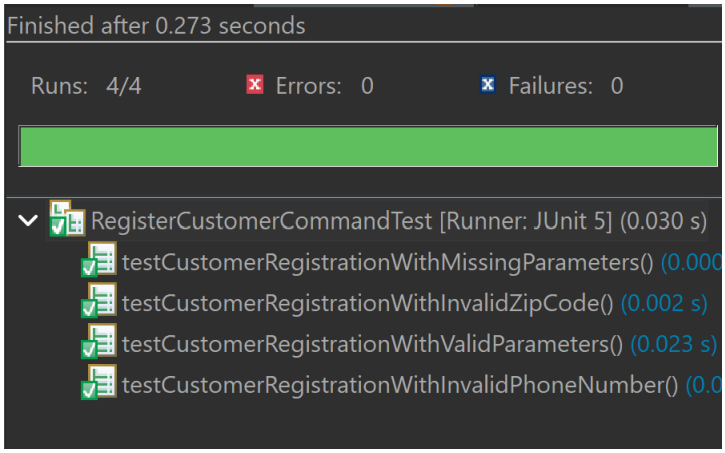

Figure 9. JUnit results for RegisterCustomerCommandTest.java

Development Challenges and Reflection

As mentioned earlier, I had issues using Maps for inputs. In hindsight, this may be due to the array of key values used for commands. Other issues I ran into dealt with the discovery of existing flaws in my system, which came to light during the additions to the system. The most significant problem was my randomly generated customer and permit. I realized I could create and use the randomly generated ID and permit but did not realize how obscure the values were or how to extract them from the registration to use for other purposes. This became the most

significant challenge as my Unit tests consistently failed for the RegisterCarCommand and RegisterCustomerCommand classes because I could not pinpoint the values assigned due to the random generation. I also came to realize that some portions of the code were using a manual creation for the customerId and permit while others were using random generators. I was able to address most of these conflicts, but due to time restrictions for this week, the remaining shall be mended in future submissions.

Overall, the development of the new classes has increased my awareness of how easy it is to overlook edge cases. While I have implemented a handful of them, I do know there are more edge cases I must consider and implement in future submissions.

References

Baeldung. 2024. “Getting Started with Java Properties.” Baeldung. Accessed April 05, 2025. https://www.baeldung.com/java-properties.

w3schools. n.d. “Java Interface.” w3schools. Accessed April 2, 2025. https://www.w3schools.com/java/java_interface.asp.