University Parking System: Decorating Our Parking Charges

for

Master of Science

Information Technology

Kassandra Vega Lucero

University of Denver College of Professional Studies

May 11, 2025

Faculty: Nathan Braun, MBA

Dean: Michael J. McGuire, MLS

Table of Contents

Background

The University developed a parking system which allows for students and community members to park in the University's parking lots. Previously the system was using parking strategies to assign charges for parking in the lots. This week the system transitioned to using a decorator approach. The approach allows for greater flexibility and scalability.

Class Implementations

By sectioning the parking charge into the decorators, a variety of charges and charge approaches can be implemented while reducing any duplication of code. Implementing this pattern structure was a bit more challenging than expected. My implementation process began by mirroring the examples I found online. I saw I would need to have the individual charge factors split into decorators including a decorator for the basic charge, based on the hour used, the day of the week and the type of car that is used. Along with the decorators there would need to be a calculator that add all of the dynamic components together. The HourlyStrategy is still used to calculate the hourly charge while the decorators add extra charges and discounts without needing to modify the base calculator.

Implementing the decorators involved a lot of trial and error. I began confusing the decorators with the functionality already included in the parking charge strategies. I noticed the instructions for the weekly assignment hinted at the decorators taking the place of the parking charge strategy. Although ideally, I would not make too many changes to my hourlyRateStrategy, I ended up starting brand new code for the strategy to remove the added factors like the car type, the day of the week, and the time of the day that the car was using the parking lot. The decorators still have a 20% discount for compact cars. Still, other charges were changed to

simplify other the implementation such as using fixed amounts for certain scenarios. After splitting the HourlyRateStrategy from the factors that were implemented in the decorators, I thought I was making decent progress when I realized the calculators were not connected the way I thought they would be. A recap of the new implementations can be observed in the UML diagrams in Figure 1, Figure 2 and Figure 3.

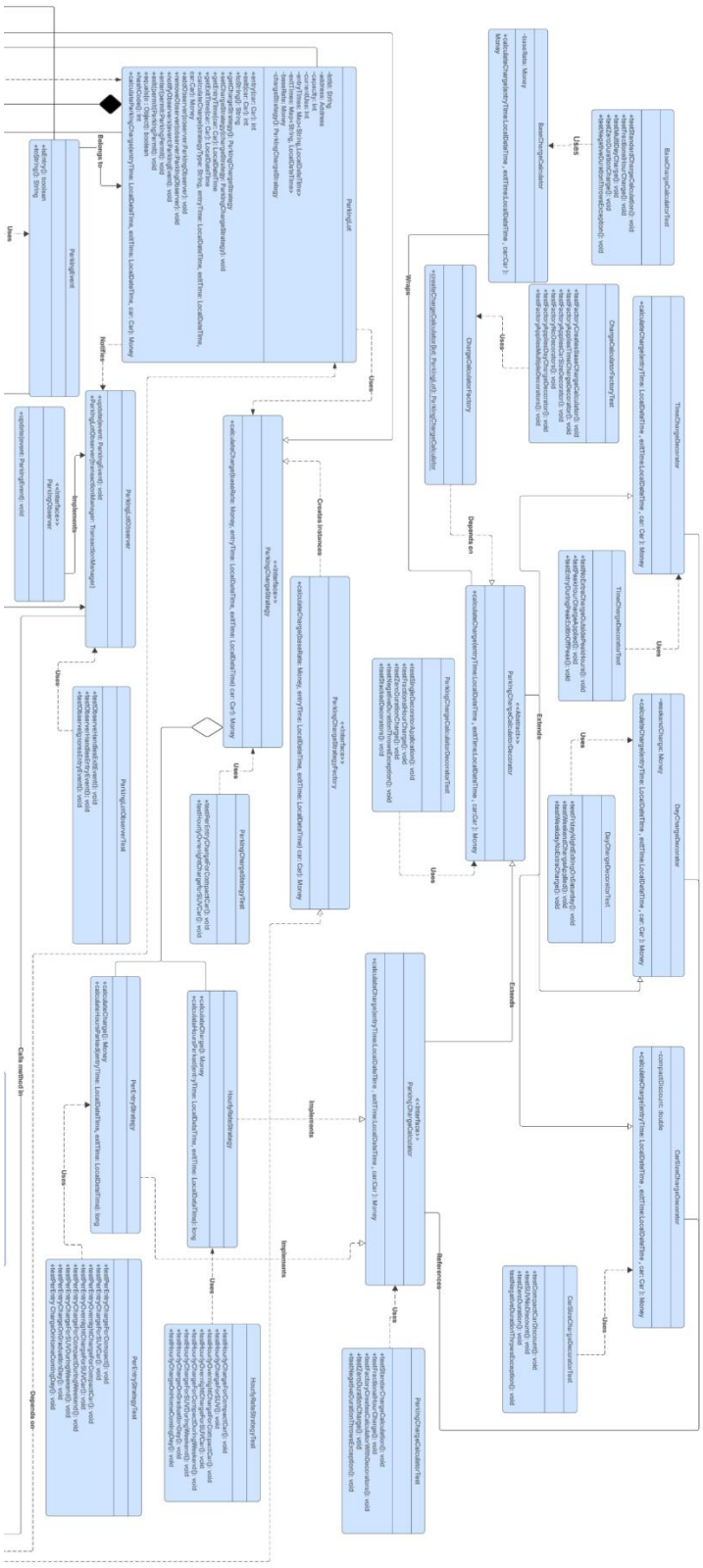

Figure 1. UML Diagram of the updated University Parking System (top).

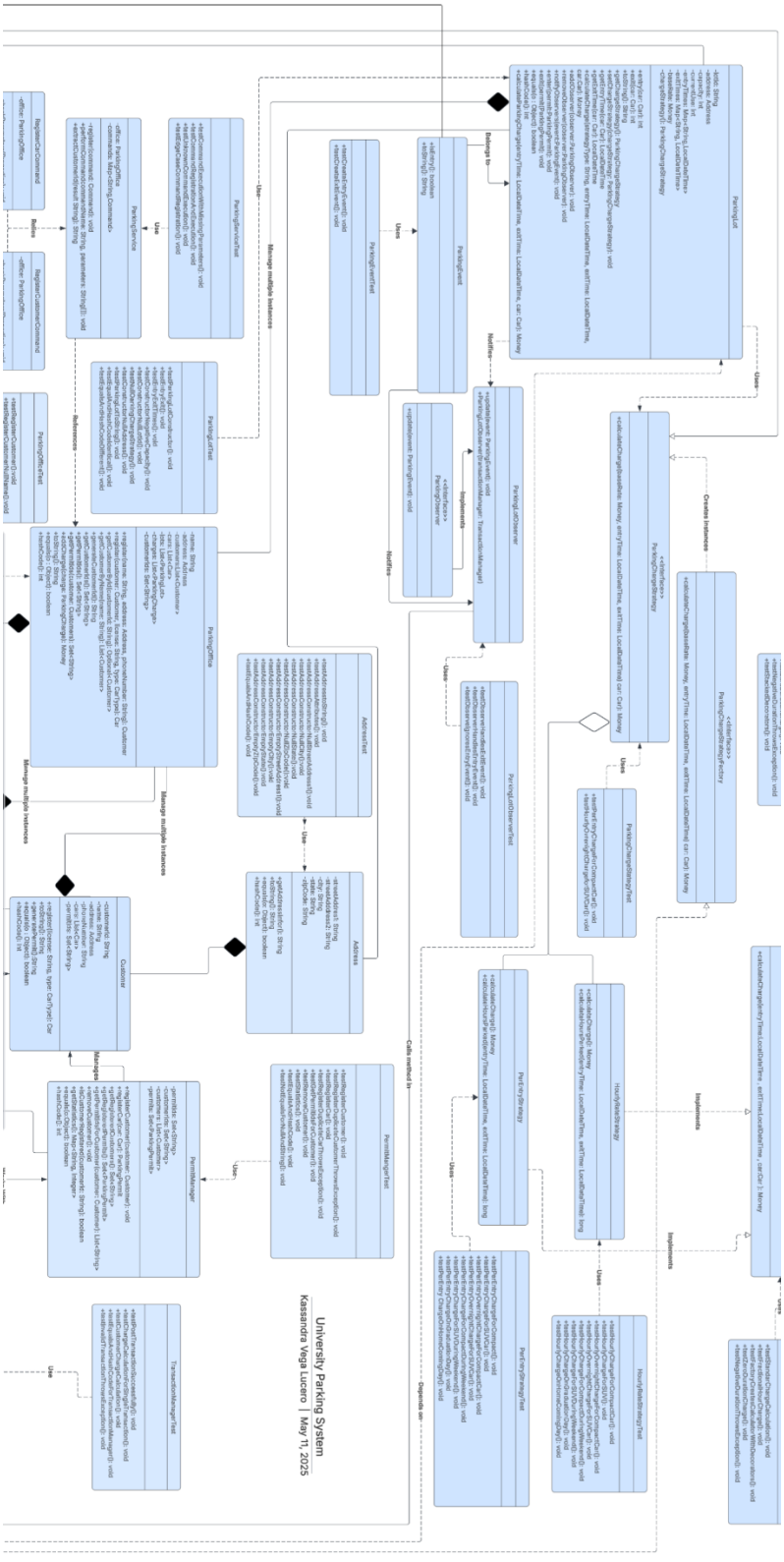

Figure 2. UML Diagram of the updated University Parking System (middle).

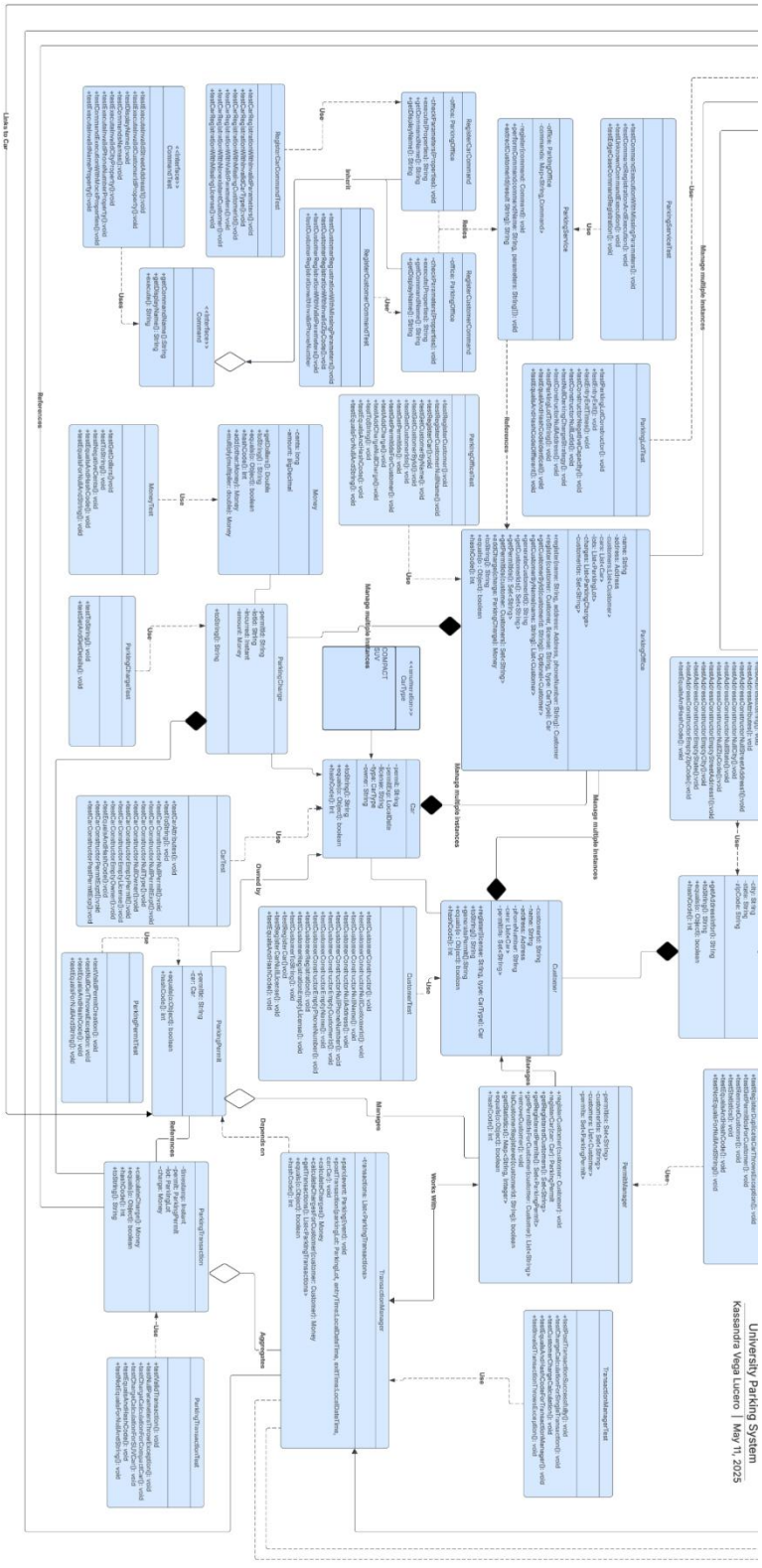

Figure 3. UML Diagram of the updated University Parking System (bottom).

Implementation Challenges

I originally intended to use the decorators and the calculators to link the charge without to much dependency on other contractors. After troubleshooting for a couple of hours, I discovered that the baseRate was linked to the parking lot instead of being linked to the decorators or the calculators. To be able to get the calculator to work, I needed to include the base rate in the parking lot constructor as well as in the calculator. This was a challenge I was unable to solve in its entirety this week and will be something that needs to be addressed in future modifications and implementations.

Other challenges I met was finding were the bugs I would find in the unit tests. I was getting the base charge, and the hourly charge confuse quite often so I needed to add print statements to each of the decorators as well at the parking charge strategy to see which charge was contributing what amount and when. I also needed to double check the values that were being used for each run. Luckily, there were several instances where the math was being done correctly by the system and was more of a user error for the comparison values. I also overlooked one of the implementations which had nested decorators. The order of the nested decorators changed the value by a dollar but was difficult to find because I was focused on just having the correct decorators while neglecting the order.

Through unit testing, I was also able to uncover gaps in my exceptions. I created a couple of edge test that failed the first few times. I was able to add code for the edge cases which included but was not limited to not charging customers who entered and left right away as well as rounding up to the next hour. Doing so was essential in providing a better client experience because customers would not want to be charged a full hour if they went in and left right away.

Unit Test and Visuals

The images below show a couple of tests that were ran to ensure new implementations for the decorator functioned properly. While making changes to the parking lot to implement the decorator pattern, it caused a ripple effect that made the tests for many of the other classes fail. This is due to the change in the requirements within constructors. Future implementation will need to address these changes to ensure the system is fully functional and has tests to support its functionality.

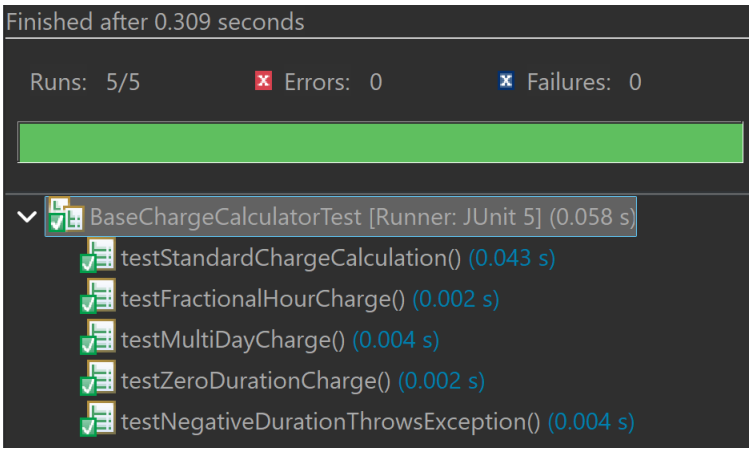

Figure 4. JUnit results for BaseChargeCalculatorTest.java.

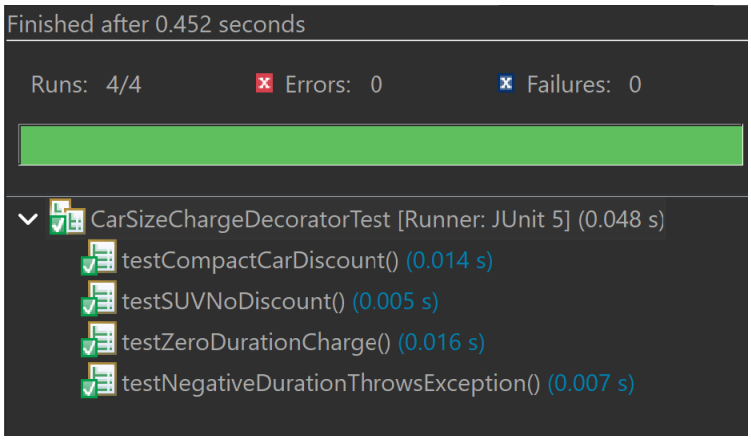

Figure 5. JUnit results for CarSizeDecoratorTest.java.

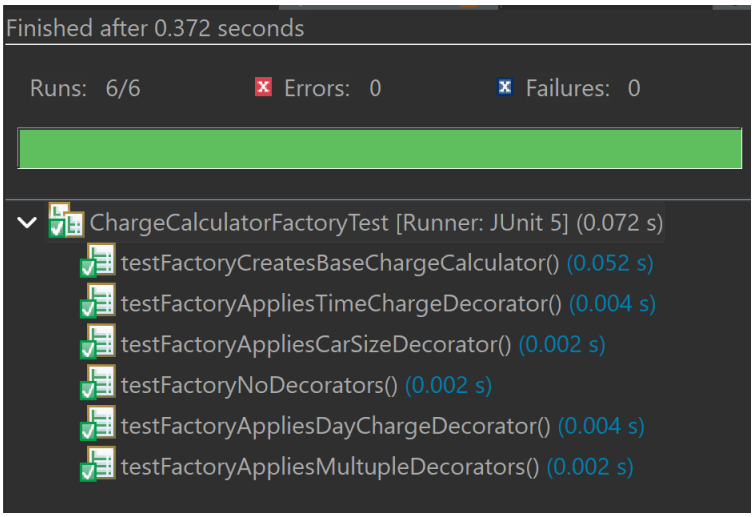

Figure 6. JUnit results for ChargeCalculatorFactoryTest.java.

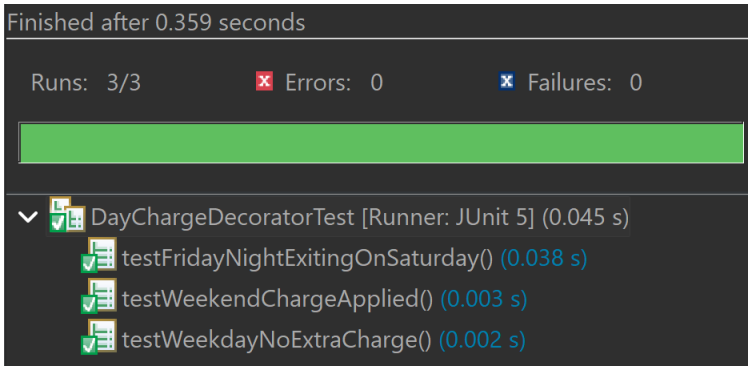

Figure 7. JUnit results for DayChargeDecoratorTest.java

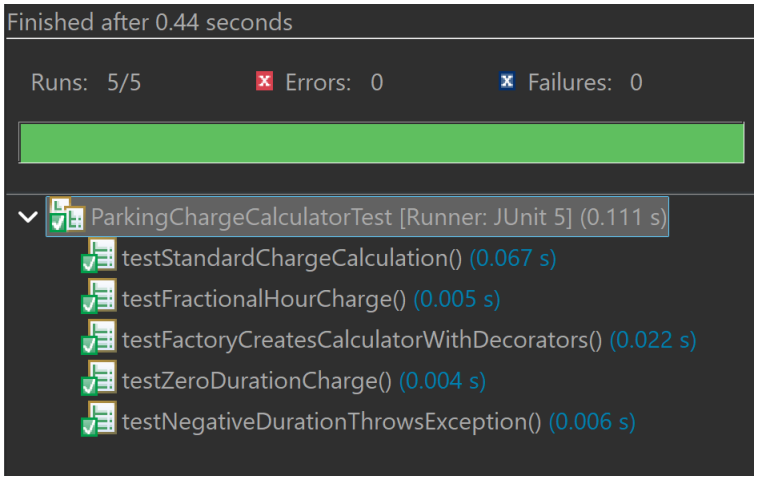

Figure 8. JUnit results for ParkingChargeCalculatorTest.java

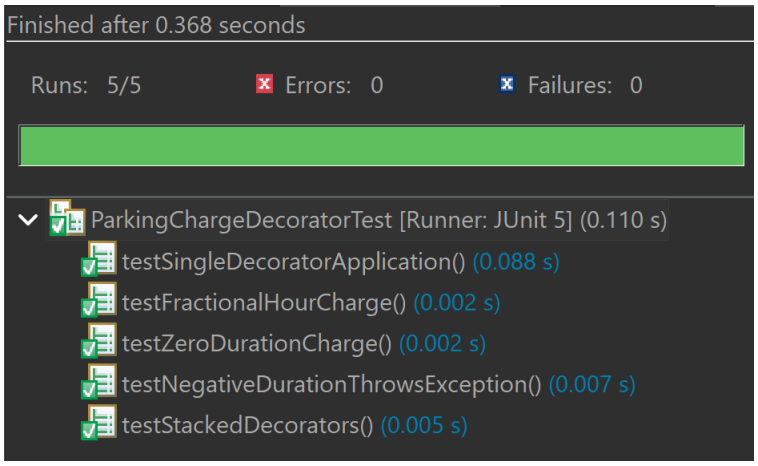

Figure 9. JUnit results for ParkingChargeDecoratorTest.java

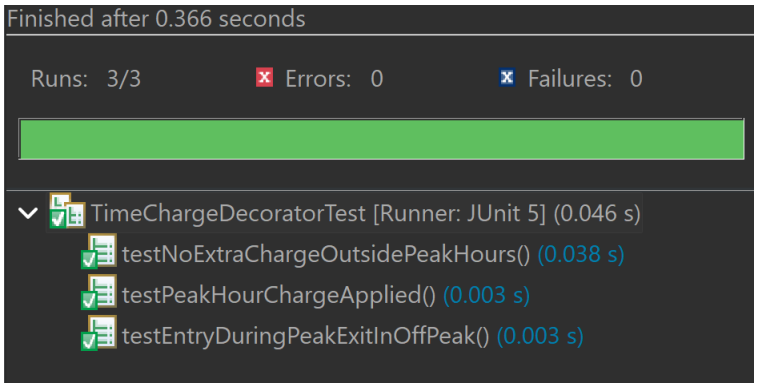

Figure 10. JUnit results for TimeChargeDecoratorTest.java

Reflection

Implementing the decorator structure this week allowed me to explore a different perspective. While I still have plenty of room for growth, I was able to learn and begin the implementation of the pattern. Moving forward, I would have to clean up the code to make it easier to follow as well as updating the tests that were affected by the changes in constructors.

References

Baeldung. 2024. “The Decorator Pattern in Java.” Reviewed by Eric Martin. Updated May 11, 2024. https://www.baeldung.com/java-decorator-pattern.

GeeksforGeeks. 2025. “Decorator Method Design Pattern in Java.” GeeksforGeeks. Updated January 3, 2025. https://www.geeksforgeeks.org/decorator-design-pattern-in-java-with-example/.

Refactoring Guru n.d. “Decorator in Java.” Refactoring Guru. Accessed May 06, 2025. https://refactoring.guru/design-patterns/decorator/java/example.