University Parking System: Implementing the Parking Charge Calculator

for

Master of Science

Information Technology

Kassandra Vega Lucero

University of Denver College of Professional Studies

April 20, 2025

Faculty: Nathan Braun, MBA

Dean: Michael J. McGuire, MLS

Table of Contents

Background

The University is developing a parking system that allows students and community members to park in their parking lot so long as they register with the University's Parking Office. The development of this parking system has been completed in stages, with this stage creating and implementing the parking charge calculator. The calculator will consider several components to assign a corresponding discount or surcharge to the final rate assignment.

Class Implementations

To begin the implementation of the parking charge calculator, I have implemented two different strategies. One strategy charges the customer per hour, considering several components such as the customer's car type, the day of the week, what time it is used, any special days such as graduation and homecoming, and any overnight stays. The second strategy charges customers per entry and considers the customer's car type, the day of the week, special days, and overnight stays. Each one of the strategies was created as its own class and using a separate package. This approach makes it so that future changes to parking charge strategies are easy to make or include. Since each parking lot can have different methods for how the charges are accrued, this implementation also makes it so that any parking lots with the exact implementation can reuse the strategy.

The HourlyRateStrategy calculates the hours spent and multiplies them by the baseRate, then goes through the remaining considerations of discounts and surcharges. The previous also applies to the PerEntryStrategy, only there is no hourly calculation, merely a fixed entry charge that is adjusted based on the components considered. The Parking charge interface has a calculateCharge method, which will be used if HourlyRateStrategy or PerEntryStrategy is not used. Strategies are also tied to the parking lots, which then allows each parking lot to have its own parking strategy. Figure 1 depicts the diagram of the classes that were updated or newly implemented in conjunction with the previously existing classes in the parking system.

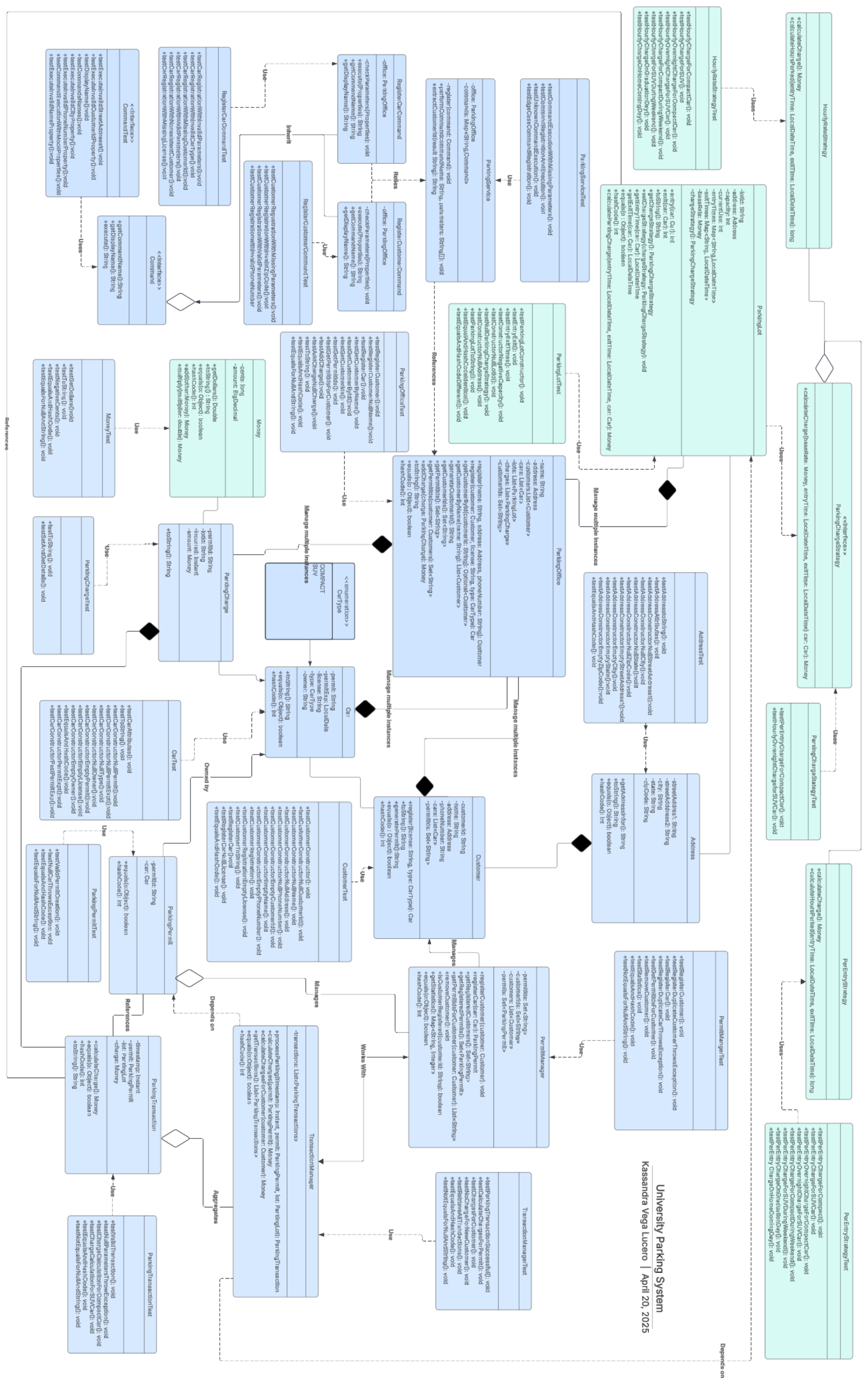


Figure 1. UML Diagram of updated University Parking System.

Implementation Challenges

While implementing the parking calculator, I encountered a handful of challenges. The first challenge was learning how to create a separate package and then getting the classes located in the original package to communicate with the new classes in the new package. I was able to find an article that walked me through some examples of packages in Java (GeeksforGeeks 2023). While the implementation was simple, the challenge was ensuring the proper classes were imported into the proper classes in order for the code to compile properly.

The other issue I ran into was testing the parking charge strategies. I had accidentally used doubles instead of using Money for variables in the classes, so when I was comparing values, the values were not matching after getting them into a comparable format. Another issue was handling the small rounding differences which I was able to learn more using an article on RoundingMode with BigDecimals (GeeksforGeeks 2019). A small challenge with the testing of the charges was manually calculating the charges to include in the tests. I often skipped consideration or did not have the correct conditions for the if statements, resulting in mismatched values. I overcame this challenge by including detailed descriptions in the comments to know which components I should be considering, and including print statements to visually see where the strategy was not matching the expected value in the calculation process.

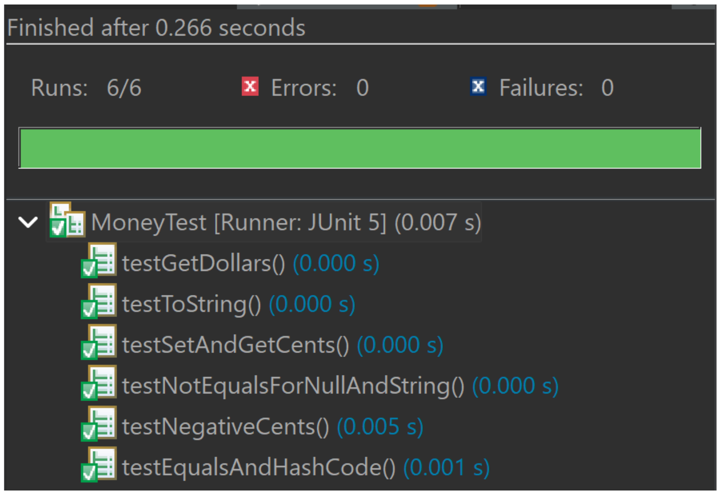
The unexpected challenge I have been troubleshooting, with no luck as of now, is that a couple of my classes from my prior submission have errors, resulting in compilation errors. I did not make any changes to the code in question, but when I was going through to take screenshots of the successful test, I began getting failed tests time after time. After some research, it seems that it may be an issue with my directory path or not using or having a JDK. I have attempted to install what I believe is the JDK I need, but it seems to vanish after downloading. This challenge has accounted for over 12 hours of troubleshooting with no resolution.

Unit Tests

The unit tests created a combination of scenarios that factor in different instances of the considered components. Examples include compact cars as well as SUVs. There are tests for different hours and durations, different base rates, overnights, weekend parking, and special days. Having a null parking charge strategy was also tested in the parking lot test. Hourly and per-entry tests were run separately as well as together in the ParkingChargeStrategy Test. A challenge I overcame in testing was that I was testing for a null exception and getting an error, only to realize the error came from not having an exception included in the tested code.

Unit Test Visuals

The images below show a combination of successful unit tests for the classes updated in or implemented for the parking charge calculator.

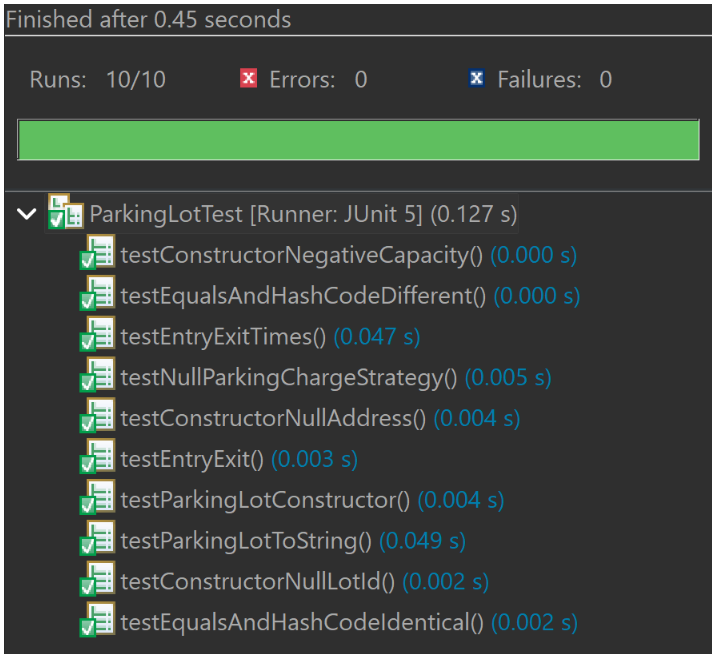


```
Finished after 0.266 seconds

Runs:  6/6        Errors:  0        Failures:  0

MoneyTest [Runner: JUnit 5] (0.007 s)
    testGetDollars() (0.000 s)
    testToString() (0.000 s)
    testSetAndGetCents() (0.000 s)
    testNotEqualsForNullAndString() (0.000 s)
    testNegativeCents() (0.005 s)
    testEqualsAndHashCode() (0.001 s)
```

Figure 2. JUnit results for Money.java

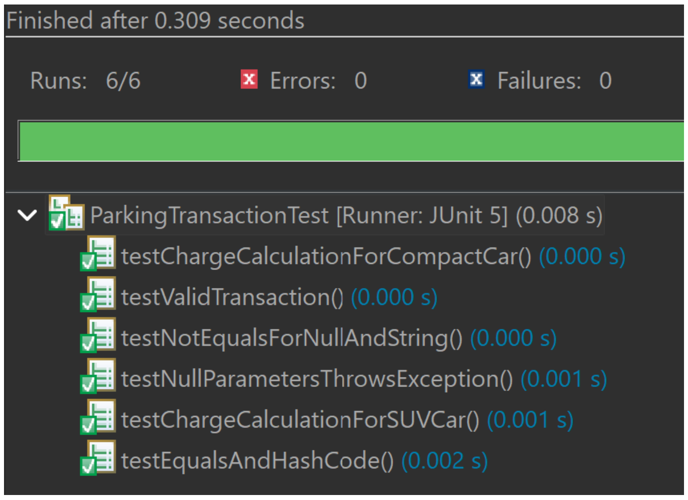


```
Finished after 0.45 seconds

Runs:  10/10        Errors:  0        Failures:  0

ParkingLotTest [Runner: JUnit 5] (0.127 s)
    testConstructorNegativeCapacity() (0.000 s)
    testEqualsAndHashCodeDifferent() (0.000 s)
    testEntryExitTimes() (0.047 s)
    testNullParkingChargeStrategy() (0.005 s)
    testConstructorNullAddress() (0.004 s)
    testEntryExit() (0.003 s)
    testParkingLotConstructor() (0.004 s)
    testParkingLotToString() (0.049 s)
    testConstructorNullLotId() (0.002 s)
    testEqualsAndHashCodeIdentical() (0.002 s)
```

Figure 3. JUnit results for ParkingLotTest.java

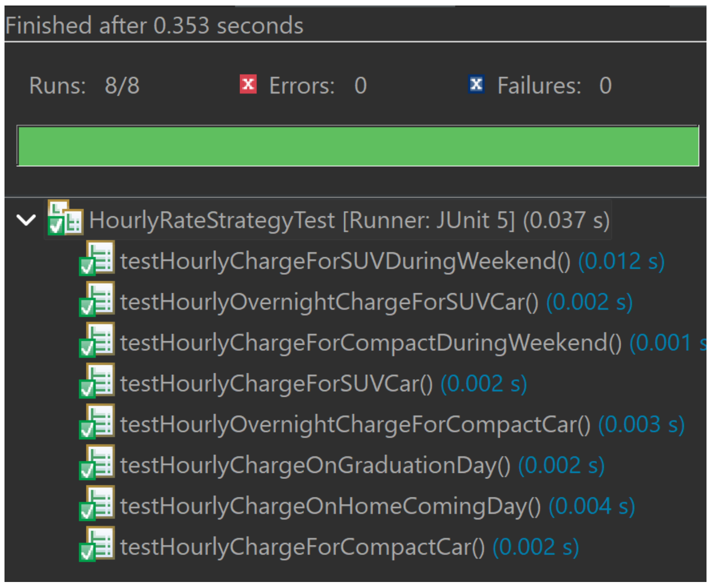


```
Finished after 0.309 seconds

Runs:  6/6        Errors:  0        Failures:  0

ParkingTransactionTest [Runner: JUnit 5] (0.008 s)
    testChargeCalculationForCompactCar() (0.000 s)
    testValidTransaction() (0.000 s)
    testNotEqualsForNullAndString() (0.000 s)
    testNullParametersThrowsException() (0.001 s)
    testChargeCalculationForSUVCar() (0.001 s)
    testEqualsAndHashCode() (0.002 s)
```

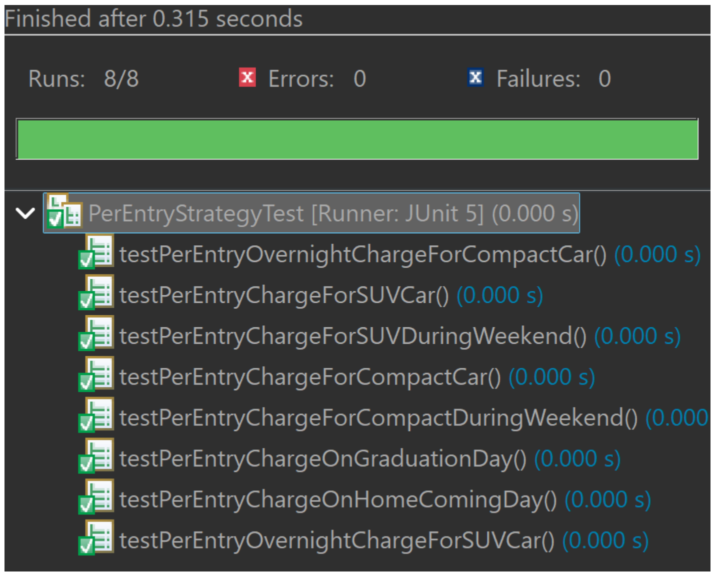
Figure 4. JUnit results for ParkingTransactionTest.java



```
Finished after 0.353 seconds
  Runs:  8/8        Errors:  0        Failures:  0

v HourlyRateStrategyTest [Runner: JUnit 5] (0.037 s)
     testHourlyChargeForSUVDuringWeekend() (0.012 s)
     testHourlyOvernightChargeForSUVCar() (0.002 s)
     testHourlyChargeForCompactDuringWeekend() (0.001 s
     testHourlyChargeForSUVCar() (0.002 s)
     testHourlyOvernightChargeForCompactCar() (0.003 s)
     testHourlyChargeOnGraduationDay() (0.002 s)
     testHourlyChargeOnHomeComingDay() (0.004 s)
     testHourlyChargeForCompactCar() (0.002 s)
```

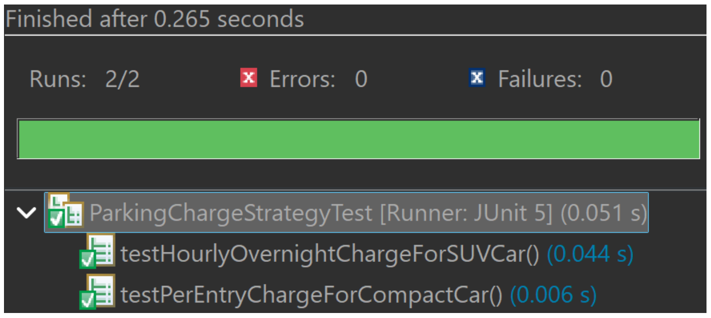
Figure 5. JUnit results for HourlyRateStrategyTest.java



```
Finished after 0.315 seconds
  Runs:  8/8        Errors:  0        Failures:  0

v PerEntryStrategyTest [Runner: JUnit 5] (0.000 s)
     testPerEntryOvernightChargeForCompactCar() (0.000 s)
     testPerEntryChargeForSUVCar() (0.000 s)
     testPerEntryChargeForSUVDuringWeekend() (0.000 s)
     testPerEntryChargeForCompactCar() (0.000 s)
     testPerEntryChargeForCompactDuringWeekend() (0.000
     testPerEntryChargeOnGraduationDay() (0.000 s)
     testPerEntryChargeOnHomeComingDay() (0.000 s)
     testPerEntryOvernightChargeForSUVCar() (0.000 s)
```

Figure 6. JUnit results for PerEntryStrategyTest.java



```
Finished after 0.265 seconds

Runs:  2/2          Errors:  0          Failures:  0

ParkingChargeStrategyTest [Runner: JUnit 5] (0.051 s)
    testHourlyOvernightChargeForSUVCar() (0.044 s)
    testPerEntryChargeForCompactCar() (0.006 s)
```

Figure 8. JUnit results for ParkingChargeStrategyTest.java

Development Challenges and Reflection

After navigating through the challenges, I am intrigued to have learned about creating an additional package as well as troubleshooting practices for tests that are calculation-heavy. Something I would like to know how to solve more quickly and efficiently is the compilation errors encountered in a combination of my Customer, ParkingOffice, and TransactionManager classes.

References

GeeksforGeeks. 2023. “How to Create Different Packages For Different Classes in Java?” GeeksforGeeks. Updated January 30, 2025. https://www.geeksforgeeks.org/how-to-create-different-packages-for-different-classes-in-java/.

GeeksforGeeks. 2019. “BigDecimal setScale() method in Java with Examples.” GeeksforGeeks. Updated June 17, 2019. https://www.geeksforgeeks.org/bigdecimal-setscale-method-in-java-with-examples/.