**Parking system classes - Reflection**

ICT 4305: Object Oriented Methods and Programming

Hamdi Ali

University of Denver College of Professional Studies

October 4, 2025

Faculty: Nirav Shah, M.S , MBA

Director: Cathie Wilson, M.S.

Dean: Michael J. McGuire, MLS

**Reflection**

The purpose of this assignment was to design and implement a small-scale parking management system using object-oriented programming (OOP). As part of the task, interrelated classes, Customer, Car, ParkingLot, and Address, and an enum CarType, were created. My favorite part of the process was implementing attributes and constructors. As the UML diagram provided a clear blueprint, I was able to translate these into Java fields and methods easily. Class relationships made coding easier, especially for basic data handling methods like getters and toString().

One of the most challenging aspects of the project was designing relationships between classes. Modeling the one-to-many relationship between customers and cars was an important decision. To accommodate multiple registered vehicles, a ListCar was needed inside the Customer class. As with ParkingLot, I had to ensure that capacity restrictions were respected when tracking parked cars. By translating multiplicity from a diagram to executable code, I gained a deeper understanding of OOP concepts like composition and association.

JUnit testing was another challenge. Initially, I was tempted to test functionality by using print statements in the main() method, but I realized that this is not industry best practice. I was able to systematically verify that car registration, permit generation, expiration dates, and parking lot capacity behaved as expected by writing unit tests with assertions. By becoming familiar with annotations such as @Test, I found that testing became not only more manageable, but also more rewarding. The unit test emphasized the importance of separating test logic from application logic in software design.

The implementation was balanced between realism and simplicity. By concatenating the license plate with a timestamp, I generated a unique parking permit identifier that ensures uniqueness without requiring a centralized database. By default, permits expire after one year, which is consistent with university parking policies. As well, while the problem statement mentioned different parking fee models, I kept fee calculations outside the scope of the project to focus on core class design. I structured the system so that future extensions, such as billing policies, could be added through new classes without modifying existing ones.

Looking back, I wish I had studied JUnit earlier in the course. In the end, understanding assertion methods, expected exceptions, and parameterized tests enhanced my confidence in writing reliable code. I would have also benefited from reviewing examples of UML-to-Java translation. Modeling associations would have been easier if I had seen more real-world class diagrams and how they map into code.

Overall, this assignment reinforced several key lessons: UML planning before coding, composition in modeling real-world relationships, and systematic unit testing. In addition, it demonstrated how object-oriented design principles such as encapsulation, modularity, and loose coupling make software more maintainable and scalable. These skills will serve as a foundation for larger projects later in the course.

**References:**

UML Diagrams. "UML Class Diagrams Overview." UML Diagrams. Accessed October 2, 2025. <https://www.uml-diagrams.org/class-diagrams-overview.html>

Oracle. "Object-Oriented Programming Concepts." Java Tutorials. Accessed October 2, 2025. <https://docs.oracle.com/javase/tutorial/java/concepts/>

Baeldung. "Java Exceptions." Baeldung. Accessed October 2, 2025. <https://www.baeldung.com/java-exceptions>

GeeksforGeeks. "Object-Oriented Programming (OOPs) Concept in Java." GeeksforGeeks. Accessed October 2, 2025. <https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/>

Visual Paradigm. "UML Class Diagram Tutorial." Visual Paradigm. Accessed October 2, 2025. <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/>

Creately. "Class Diagram Examples." Creately. Accessed October 2, 2025. <https://creately.com/diagram/example/in/class-diagram>

TutorialsPoint. "Java - Exceptions." TutorialsPoint. Accessed October 2, 2025. <https://www.tutorialspoint.com/java/java_exceptions.htm>