Reflection Log: Vehicles Program

Project Description

In this project, I developed a Java program to simulate various types of vehicles, including:

1. **Base Class**: `Vehicle` (common attributes like `make`, `model`, and `year`).
2. **Subclasses**:
    - `Car` (specific attribute: `numDoors`).
    - `Truck` (specific attribute: `payloadCapacity`).
    - `Minivan` (specific attribute: `seatingCapacity`).

The program demonstrated **inheritance**, **method overriding**, and **validation** while maintaining a modular design.

What I Learned

1. **Class Hierarchies**:
    - I learned to design a class hierarchy that separates shared functionality (in the `Vehicle` class) from specific behaviors (in the subclasses).
    - This approach ensured modularity and reduced redundancy in the code.
2. **Using `super`**:
    - I gained experience with the `super` keyword to call parent class constructors and methods.
    - This ensured that common attributes (`make`, `model`, `year`) were initialized consistently across all vehicle types.
3. **Overriding and Extending Behavior**:
    - I practiced overriding methods, such as the `getInfo` method in each subclass, to include details specific to the vehicle type.
4. **Validation**:
    - I incorporated validation in constructors to ensure that values like `year` and `payloadCapacity` were reasonable (e.g., non-negative).
    - This highlighted the importance of anticipating edge cases during development.

Challenges

1. **Initial Design Flaws**:
    - Initially, I nested the `Main` class within the program, making the code harder to manage and execute.
    - Refactoring required restructuring the program to separate the `main` method into its own class, which improved organization and maintainability.
2. **Standardizing Output**:
    - Generating consistent and readable output for all vehicle types was challenging.
    - I resolved this by centralizing common functionality in the `Vehicle` class, simplifying the process.
3. **Validation Complexity**:
    - Designing meaningful validations, like ensuring `year` and `payloadCapacity` were within realistic ranges, required careful thought and research on real-world constraints.

Improvements

1. **Dynamic Input**:
    - In future iterations, I could allow users to create vehicle objects dynamically and input their attributes.
    - This would make the program more interactive and user-friendly.
2. **Enhanced Validation**:
    - Adding more comprehensive checks, such as ensuring the `year` falls within a realistic range (e.g., 1886 to the current year), would make the program more robust.
3. **Extending the Hierarchy**:
    - I could introduce additional subclasses (e.g., `SUV`, `Motorcycle`) to expand the hierarchy and model a wider range of vehicles.
    - Each subclass could include unique attributes and behaviors.

Reflection

This project strengthened my understanding of key object-oriented programming concepts, particularly **inheritance**, **method overriding**, and **validation**. Key takeaways include:

- The importance of **class hierarchies** in designing modular and maintainable code.

- How using `super` ensures consistent initialization across subclasses.
- The value of **refactoring** to simplify code and improve readability.

By focusing on **planning**, **validation**, and **extensibility**, I was able to create a program that not only met current requirements but could also be adapted for future use cases. This experience has given me confidence in applying these principles to more complex projects.