Reflection Log

Project Description

In this project, I developed three Java programs to demonstrate core object-oriented programming (OOP) principles, focusing on **inheritance** and **polymorphism**:

1. **Employee Program**: Implemented classes `UEmployee`, `Faculty`, and `Staff`.
2. **Vehicle Program**: Created classes `Vehicle`, `Car`, `Truck`, and `Minivan`.
3. **Account Program**: Designed classes `Account`, `PersonalAcct`, and `BusinessAcct`.

Each program emphasized **inheritance**, **encapsulation**, and **method overriding**, illustrating how OOP principles can be applied to real-world scenarios.

What I Learned

1. **Inheritance and Polymorphism**:
   - I improved my ability to design class hierarchies using inheritance.
   - Polymorphism enabled me to work with subclasses (`Faculty`, `Staff`) through references to their parent class (`UEmployee`).
2. **Encapsulation**:
   - I utilized private fields and provided controlled access through getters and setters, ensuring proper data hiding and protection.
3. **Method Overriding**:
   - I practiced overriding base class methods in subclasses to customize behavior.
   - For example, in the Vehicle program, the `getInfo()` method displayed subclass-specific details for `Car`, `Truck`, and `Minivan`.
4. **Error Validation**:
   - I learned how to add validation checks to ensure data integrity, such as verifying non-negative values for salaries, vehicle years, and account balances.
5. **Reusability**:
   - I created reusable methods like `toString()` to standardize data presentation across different classes, reducing redundancy and making the code more maintainable.

Challenges

1. **Nested Classes**:
   - Initially, I used nested classes in the employee and vehicle programs, which led to cluttered and hard-to-read code.

    - Refactoring these into separate files improved the code organization and maintainability but required significant effort.
2. **Validation**:
    - Designing meaningful validation checks for edge cases, such as invalid salaries and extreme payload capacities, was time-consuming. However, it greatly enhanced the robustness of the programs.
3. **Redundant Code**:
    - Early versions of the programs contained repetitive logic in subclasses, such as string concatenations for output.
    - Refactoring and centralizing common logic in parent classes taught me to write more efficient and maintainable code.

Improvements

1. **Enhanced Testing**:
    - Including more comprehensive test cases to handle edge cases like invalid input and extreme values would improve the reliability of the programs.
2. **Custom Exceptions**:
    - Replacing general exceptions like `IllegalArgumentException` with custom exceptions, such as `InvalidSalaryException` or `InsufficientFundsException`, would provide clearer and more specific error handling.
3. **Serialization**:
    - Implementing file I/O to save and load data (e.g., employee records, vehicle details, account balances) would enable persistent data storage for real-world applications.

Reflection

This project greatly enhanced my understanding of OOP principles in Java, particularly:

- **Inheritance**: Designing class hierarchies to model relationships.
- **Polymorphism**: Simplifying code through parent-child relationships.
- **Encapsulation**: Protecting data and controlling access.

I also learned the importance of:

- **Validation**: Handling edge cases to ensure robust error handling.
- **Code Organization**: Refactoring and structuring classes for better maintainability.

This experience has provided me with a solid foundation for tackling more advanced Java projects, and I feel more confident applying OOP concepts in future development work.