Resident HealthCare System – Report

Student: Seokjun Jeong (s4118933)
Course: COSC1295 Advanced Programming

---

1. Overview of Design and Refactoring

This project was developed incrementally to apply advanced object-oriented programming principles within a real-world healthcare management context.
During refactoring, the primary focus was on enhancing maintainability, cohesion, and scalability by aligning the architecture with the SOLID design principles.
Key improvements included minimizing class coupling, refining inheritance hierarchies, and ensuring a clear separation between UI and business logic.
Core services such as AuthService and LogService were restructured to follow the Singleton and Facade design patterns, while the repository layer was generalized through Generics to support extensibility and alternative persistence mechanisms.

---

2. Key Learning Outcomes

Through this project, I strengthened my understanding of encapsulation, inheritance, and polymorphism, applying these concepts to design flexible domain models (Resident, Staff, and Ward).
I gained practical experience in using JavaFX to build an interactive graphical interface that integrates seamlessly with the underlying data model.
In addition, I implemented role-based authorization and a centralized logging mechanism to improve accountability and traceability.
Automated testing using JUnit validated system integrity by enforcing business rules such as roster compliance, serialization accuracy, and data persistence.

---

3. Refactoring Highlights

- SOLID Implementation: Refactored core logic into smaller, cohesive components to improve modularity and reduce dependencies between layers.
- Design Patterns Applied:
    - Singleton – Used in LogService to ensure a unified logging process across the system.
    - Facade – Implemented in AuthService to simplify access control and validation.
    - Strategy/Repository Pattern – Applied through Repository<T, ID> to allow flexible persistence strategies.
- Exception Management: Centralized validation and error handling for unauthorized access and invalid operations to enhance robustness.

- **UI Refactoring:** Simplified JavaFX UI refresh logic by re-rendering scenes instead of manual component updates, reducing threading issues.

---

4. Challenges and Solutions

One of the main challenges involved serialization failures, which were resolved by explicitly implementing the Serializable interface and avoiding lambda expressions that captured non-serializable contexts.
To address JavaFX concurrency issues, the system was refactored to refresh the interface through full scene re-rendering, ensuring smoother updates.
Furthermore, inconsistencies in role-based permissions were mitigated by centralizing all access validation in the AuthService class.

---

5. Personal Reflection

This project significantly enhanced my understanding of modular design and maintainable software architecture.
It demonstrated how clear separation of concerns between the core, repository, and UI layers enables scalability and long-term system evolution.
Looking ahead, future development could include integrating Hibernate/JPA for database persistence, implementing an Observer pattern for real-time GUI updates, and replacing file-based storage with a RESTful API for interoperability and external auditing.
Overall, this experience improved my ability to combine object-oriented principles with architectural best practices to design robust and extensible applications.