Project Overview

The Entity Mapping Project is designed to implement and test JPA (Java Persistence API) entities and their relationships in a Spring Boot application. The project focuses on creating a data model that includes `Employee`, `Doctor`, and `Department` entities to represent a healthcare management system. The goal is to demonstrate the use of inheritance, mapping strategies, and entity relationships in a Java-based application.

Entities and Inheritance Strategy

1. **Employee Entity:**
    - Serves as a base entity with fields such as `emNo`, `surname`, `firstName`, `address`, and `telephone`.
    - Uses `InheritanceType.JOINED` strategy to allow other entities like `Doctor` to inherit its properties, ensuring data normalization and separation of concerns in the database schema.
2. **Doctor Entity:**
    - Extends the `Employee` entity and adds a specific field, `specialty`, to denote the medical specialty of the doctor.
    - Inherits all properties from the `Employee` class, leveraging the JPA inheritance strategy to avoid duplication and maintain a clean schema design.
3. **Department Entity:**
    - Represents a hospital department with fields such as `depCode`, `depName`, and `building`.
    - Originally included a bidirectional relationship with `Employee` for the `director` field, and a `OneToMany` relationship with `Ward`. However, these relationships were removed to simplify testing and avoid circular references during JSON serialization.

Challenges and Solutions

- **Simplifying Entity Structure for Testing:** To facilitate easier testing and data manipulation, relationships between entities were minimized. This approach allowed for independent creation and testing of each entity, reducing complexity and enhancing maintainability.

Testing and Validation

- **Postman API Testing:** CRUD operations for `Employee`, `Doctor`, and `Department` entities were tested using Postman. The REST endpoints were verified for proper

creation, retrieval, updating, and deletion of records without triggering circular references or serialization issues.

- **Create Employee:** Successfully tested the creation of an `Employee` entity.
- **Create Doctor:** Tested creation and validation of a `Doctor` entity extending `Employee`.
- **Create Department:** Verified the creation of a `Department` entity without associated `Ward` or `Employee` references to prevent complex dependencies.

Lessons Learned

- **Effective Use of JPA Inheritance and Annotations:** Proper use of JPA inheritance strategies (`JOINED`) helped maintain a normalized database schema while allowing entities to share common fields. Understanding and applying serialization annotations such as `@JsonManagedReference` and `@JsonBackReference` is crucial in managing complex entity relationships.
- **Importance of Simplified Design:** A simplified design approach, especially during the development and testing phase, reduces complications and facilitates easier debugging and validation. It is often beneficial to start with a straightforward design and iteratively add complexity as needed.

Conclusion and Future Work

The Entity Mapping Project successfully demonstrated the use of JPA and Spring Boot to manage entity relationships and inheritance. The application structure now supports clean, maintainable code and allows for easy extension and modification. Future work may involve reintroducing more complex relationships with better management techniques, such as DTOs or additional serialization control, to fully utilize JPA's capabilities in a more advanced healthcare management system. Additionally, integrating unit and integration testing frameworks like JUnit and Mockito can enhance reliability and ensure robust application behavior.

This project lays a solid foundation for further exploration and refinement of entity mapping strategies and advanced data handling techniques in Java-based enterprise applications.