1. **Project Structure**

- **Core Classes**: Define the fundamental concepts (e.g., Employee, Department).

- **Specialized Classes**: Extend or implement core classes for specific purposes (e.g., FullTimeEmployee, PartTimeEmployee).

- **Factories and Builders**: Handle object creation in a systematic way (e.g., EmployeeFactory, EmployeeBuilder).

- **Proxies and Managers**: Manage access or add functionality to existing systems (e.g., PayrollProxy, DatabaseConnectionManager)

- **User Interface (UI):** Classes like GUI manage the application's graphical interface.

## 2. **Common Design Patterns in the Project**

Here's how the patterns might work based on typical implementations:

- **Factory Pattern (EmployeeFactory):**
    - Creates different types of employees (FullTimeEmployee, PartTimeEmployee) without specifying their exact classes.
    - Why Use It: Simplifies object creation and maintains flexibility when adding new types of employees.

- **Prototype Pattern (EmployeePrototype):**
    - Allows cloning of existing employee objects to create new ones.
    - Why Use It: Reduces overhead when creating complex objects.

- **Builder Pattern (EmployeeBuilder):**
    - Constructs complex employee objects step-by-step.
    - Why Use It: Improves readability and allows flexible construction of objects.

. **Proxy Pattern (PayrollProxy):**
    - Controls access to the PayrollSystem or adds additional functionality (e.g., security checks).
    - Why Use It: Protects the system from unauthorized access and enhances security.

- **Singleton Pattern (DatabaseConnectionManager):**
    - Ensures a single instance of the database connection manager exists throughout the application.
    - Why Use It: Prevents multiple database connections, saving resources.