Lab # 1 – 24F

Scenario

You are tasked with designing an **Employee Management System** to manage different types of employees in an organization. The employees have various attributes, such as name, id, department, and role, and the system must efficiently manage different types of employees (e.g., full-time, part-time etc.). The objective of this labs is to understand and implement the **Singleton**, **Builder**, and **Factory** design patterns. You need to apply these design patterns to create, manage, and organize the employees in this system.

Solution

Start your solution by defining an abstract class **Employee** with properties **Id**, **Name**, **Department**, **Role**, **Working Hours Per Week** and **salary** and abstract methods **clockIn(), clockOut(),** and **trackWorkHours().**

1. Singleton Pattern: Ensure that there is only one instance of an **EmployeeManager** that manages all employees in the system.

a. Create a class **EmployeeManager** following the Singleton pattern.
b. Implement methods in **EmployeeManager** to add, remove, and retrieve employees.

2. Builder Pattern: Implement a flexible way to construct different types of employees using the Builder pattern.

a. Define an interface **EmployeeBuilder** with methods for building different parts of an employee (e.g., setName(), setDepartment(), setRole() etc.).
b. Implement concrete builders (**FullTimeEmployeeBuilder**, **PartTimeEmployeeBuilder**, etc.) that implement **EmployeeBuilder**.
c. Create a **EmployeeDirector** class that takes an **EmployeeBuilder** and constructs an employee.

3. Factory Pattern: Implement a Factory to create different types of employees based on input.

a. Create a **EmployeeFactory** class with a method that takes an employee type and returns an instance of the corresponding class.

b. Implement concrete classes **FullTimeEmployee** and **PartTimeEmployee** based on **Employee**.
c. Use the **Factory** to create employee instances based on input.

4. Implement a separate class named **EMS** to simulate the entire solution. In this class create several instance of employees (using Builder and Factory patterns) and add them to **EmployeeManager**.

5. JUnit: Implement unit tests for the following classes

a. **EmployeeManager** – To ensure singleton behavior and employee management
b. **EmployeeDirector** – To test that it correctly constructs employees
c. **EmployeeFactory** – To test that it correctly creates different employee types

Deliverables

a. This Lab constitutes 7.5% of your overall grade.
b. Deliver a comprehensive coding solution and make it available in **GitHub**. Submit the repo link on the BrightSpace. Your code should include proper commenting according to Java coding standards
c. You must demo your solution during the lab session
d. Violating academic integrity or missing the deadline will result in a grade of 0 for your submission.