A diagram of a company

Description automatically generated

**Purpose of Each Class**

Employee: Represents basic employee attributes and provides getters/setters for these attributes.

EmployeeService: Interface defining methods for calculating compensation and renewal dates for employees.

PermanentEmployeeImpl: Implements EmployeeService for permanent employees, providing methods to calculate total compensation, pension contribution, and bonus.

ContractEmployeeImpl: Implements EmployeeService for contract employees, providing methods to calculate total compensation and renewal date.

EmployeeValidator: Validates employee attributes like name, email, salary, etc., ensuring they meet specified criteria.

EmployeeController: Processes, validates, and saves employee data using formatters and persistence services.

Rate: Holds constants for bonus and pension percentages.

Formatter: Interface for formatting employee data (JSON and Text).

JSONFormatter: Implements Formatter for formatting employee data in JSON format.

TextFormatter: Implements Formatter for formatting employee data in text format.

PersistenceService: Saves formatted employee data to files.

Main: Entry point to instantiate employees, process them using EmployeeController, and persist data.

**Application of SOLID Principles**

**Single Responsibility Principle (SRP): Each class has a single responsibility:**

**Employee:** Manages employee data.

**EmployeeService:** Calculates employee compensations and renewal dates.

**EmployeeValidator:** Validates employee attributes.

**EmployeeController:** Coordinates processing, validation, and persistence of employee data.

**Formatters and PersistenceService:** Handle formatting and saving of employee data.

**Open/Closed Principle (OCP):** Classes are open for extension but closed for modification:

New formatters or persistence strategies can be added (e.g., XMLFormatter) without modifying existing classes.

**Liskov Substitution Principle (LSP):** Subtypes like PermanentEmployeeImpl and ContractEmployeeImpl can be used interchangeably with Employee.

**Interface Segregation Principle (ISP):** Interfaces like EmployeeService and Formatter are specific to their clients, ensuring they only depend on what they need.

**Dependency Inversion Principle (DIP):** High-level modules (e.g., EmployeeController) depend on abstractions (e.g., EmployeeService, Formatter), not on concrete implementations.

.