

**Purpose of Each Newly Introduced Class**

1. PermanentEmployeeImplTest: This class is designed to test the functionalities of the PermanentEmployeeImpl class. It ensures that the methods for calculating pension contributions, bonuses, total compensation, and retrieving salary work as expected.
2. ContractEmployeeImplTest: This class tests the functionalities of the ContractEmployeeImpl class. It verifies the methods for calculating total compensation, retrieving salary, and determining the renewal date for contract employees.
3. PersistenceServiceTest: This class tests the PersistenceService class, ensuring that employee data is correctly saved to a file using the specified formatter.

**Application of SOLID Principles**

1. Single Responsibility Principle (SRP):  
   Each class has a single responsibility. For example, PermanentEmployeeImpl handles calculations specific to permanent employees, while ContractEmployeeImpl handles calculations specific to contract employees.  
   The PersistenceService class is responsible for saving employee data, and the JSONFormatter class is responsible for formatting employee data.
2. Open/Closed Principle (OCP):  
   The Employee class can be extended to add more types of employees without modifying the existing code.  
   The PersistenceService class can use different formatters (e.g., JSONFormatter) without changing its implementation.
3. Liskov Substitution Principle (LSP):  
   Subtypes (e.g., PermanentEmployeeImpl, ContractEmployeeImpl) can be used interchangeably with their base type (Employee) without affecting the correctness of the program.
4. Interface Segregation Principle (ISP):  
   The design avoids large, monolithic interfaces. For example, the Formatter interface is specific to formatting employee data, and different implementations can be provided as needed.
5. Dependency Inversion Principle (DIP):  
   High-level modules (PersistenceService) do not depend on low-level modules (JSONFormatter). Instead, they depend on abstractions (Formatter), allowing for flexible and interchangeable implementations.  
   By adhering to these principles, the solution is modular, maintainable, and scalable, ensuring that each class has a clear and focused responsibility.