**SRP** - Single Responsibility Principle means a class should have only one reason to change. For example, in a banking system, a CreditCardProcessor class should only handle processing credit card transactions. Validation, fraud checks, or sending notifications should be handled by separate classes like CreditCardValidator or CreditCardNotificationService. This way, changes in validation rules or notification logic don’t affect the transaction processing, making the system easier to maintain and less error-prone.

**OCP** - Open/Closed Principle states that software entities should be open for extension but closed for modification. In a banking system, for example, a PaymentProcessor class should depend on an IPayment interface. Existing code doesn’t need to change when adding new payment types like CreditCard, UPI, or Crypto. Each new payment type implements the IPayment interface, extending functionality without modifying tested classes, which makes the system flexible and maintainable. => Interface

**LISKOV** - Liskov Substitution Principle ensures that subclasses can replace their base class without breaking the client code. In banking, for example, not all accounts can withdraw money. So instead of forcing all Account types to implement a Withdraw () method, I define an IWithdrawableAccount interface for accounts like Savings or Current. Fixed Deposit accounts don’t implement this interface, preventing unexpected errors. This way, any withdrawable account can be substituted safely, maintaining correct behaviour and adhering to LSP. 🡺 Interface

ISP –

DI -