**SOLID Principles**

**Definition:**  
SOLID is a set of **five design principles** that help developers create more understandable, flexible, and maintainable software systems. These principles were introduced by Robert C. Martin (Uncle Bob).

**Principles:**

1. **S – Single Responsibility Principle (SRP):**  
   A class should have only one reason to change, meaning it should have only one responsibility.  
    *Example:*
   * Class InvoicePrinter should handle only printing invoices, not saving them to a database.
2. **O – Open/Closed Principle (OCP):**  
   Software entities should be **open for extension** but **closed for modification**.  
    *Example:*
   * Use interfaces or abstract classes so new functionality can be added without changing existing code.
3. **L – Liskov Substitution Principle (LSP):**  
   Subtypes must be substitutable for their base types without altering the correctness of the program.  
    *Example:*
   * If class Bird has a method fly(), then subclass Penguin shouldn’t override it inappropriately (since penguins can’t fly).
4. **I – Interface Segregation Principle (ISP):**  
   Clients should not be forced to depend on interfaces they don’t use.  
    *Example:*
   * Instead of one large interface Animal with fly(), swim(), and walk(), create smaller ones like Flyable, Swimmable, Walkable.

Interface Animale

{

A

B

C

}

Class Demo implements Animal

{

A

B

C

}

1. **D – Dependency Inversion Principle (DIP):**  
   High-level modules should not depend on low-level modules; both should depend on abstractions.  
    *Example:*
   * Instead of directly using MySQLDatabase, depend on an interface Database, and inject the implementation.

dB-connection

addproduct()

{

Conn =---0nject that injection instead having directly

url,username,password

**Separation of Concerns (SoC)**

**Definition:**  
This principle states that a program should be divided into distinct sections, each addressing a separate concern or responsibility.

5 services ---one service not depending on another service ----even though one service got down ….it will not crash the entire software….

Ecommerce ---add ----

Search product ---not product ---down

Purchase ----in cart and I can do the payment ----

**Purpose:**

* Improves readability and maintainability.
* Reduces coupling between modules.
* Increases reusability and testability.

**Example in a Web Application:**

* **Model:** Business logic and data.
* **View:** User interface.
* **Controller:** Handles input and connects Model and View.  
  (That’s why MVC — Model-View-Controller — follows SoC.)

*Example:*  
In a Java web app:

* UserService handles business logic.
* UserController(Main) handles HTTP requests.
* UserRepository handles database interactions.