DAY 05

1. What are SOLID Design Principles?

SOLID principles are a set of rules introduced by Robert C. Martin (2000) to help developers write code that is clean, maintainable, and scalable.

Analogy (The Messy House):

Imagine a house where all electrical wires, internet cables, and water pipes are tangled together in one big mess. If one wire breaks, finding and fixing it is a nightmare.

Similarly, if your code is "tightly coupled" (juda hua), fixing a bug or adding a feature becomes a disaster.

Why do we need them?

- **Maintainability:** Easy to add new features without breaking old ones.
- **Readability:** New engineers can understand the code quickly.
- **Bug Reduction:** Reduces the "Disaster" of introducing bugs when changing code.

2. The Acronym "SOLID"

The following table explains the 5 pillars (Note: Video covers S, O, L):

Letter	Principle	Meaning
S	**Single Responsibility Principle (SRP)**	A class should do only one thing.
O	**Open/Closed Principle (OCP)**	Open for Extension, Closed for Modification.
L	**Liskov Substitution Principle (LSP)**	Child classes should replace Parent classes without breaking code.
I	**Interface Segregation Principle (ISP)**	(Covered in next video)

D	**Dependency Inversion Principle (DIP)**	(Covered in next video)

3. S - Single Responsibility Principle (SRP)

Definition: "A class should have only one reason to change."

Hinglish Analogy:

Jaise TV remote ka kaam sirf TV control karna hai. Agar usi remote se Fridge aur AC bhi control karne lage, toh remote bohot complex ho jayega aur kharab hone par theek karna mushkil hoga.

- **Bad Code:** A ShoppingCart class that calculates total price, prints invoices, AND saves to the database.
- **Good Code:** Break it down!
    - ShoppingCart: Only calculates total.
    - InvoicePrinter: Handles printing.
    - DBStorage: Handles database saving.

4. O - Open/Closed Principle (OCP)

Definition: "A class should be OPEN for Extension but CLOSED for Modification."

The "Modification" Rule:

You should not touch or change existing, tested code (purani classes) just to add a new feature.

The "Extension" Rule:

You should be able to add new behavior by creating new classes that extend the original one.

Example Scenario: Saving Data

- **Problem:** You have a DBStorage class that saves to SQL. Now you need to save to MongoDB and Files.
- **Violation (Galat Tareeka):** You go into the DBStorage class and add if-else logic or new methods like saveToMongo inside it. You modified the tested class!
- **Solution (Sahi Tareeka):** Use **Interfaces/Abstraction**.
    1. Create an interface Persistence with a save() method.
    2. Make SqlPersistence, MongoPersistence, and FilePersistence classes that implement this interface.
    3. The main code just calls persistence.save().

4. **Result:** To add a new database, you just create a NEW class. You never touch the old code.

5. L - Liskov Substitution Principle (LSP)

Definition: "Subtypes must be substitutable for their base types."

Meaning: If Class B is a child of Class A, you should be able to use B anywhere you use A without the code crashing.

The Banking Example (Where it fails)

- **Scenario:** You have a parent class Account with methods deposit() and withdraw().
- **Child Classes:** SavingsAccount and CurrentAccount work fine.
- **The Problem Child:** You add a FixedDepositAccount (FD).
    - FDs allow deposit(), but **do not** allow withdraw().
    - If you call withdraw() on an FD object, code breaks (or throws an exception).
    - This violates LSP because FixedDepositAccount cannot fully replace Account.

The Fix:

Don't force a class to inherit methods it can't use.

- **Refactor Hierarchy:**
    1. Create interface NonWithdrawableAccount (has only deposit).
    2. Create interface WithdrawableAccount (extends above, adds withdraw).
    3. FixedDeposit implements NonWithdrawableAccount.
    4. Savings/Current implement WithdrawableAccount.

6. Summary & Key Points

Principle	Core Concept	Key Takeaway
SRP	One Class = One Job	**Separation of Concerns:** Don't mix logic (e.g., Calculation vs. Printing).
OCP	Extend, Don't Modify	Use **Interfaces** and **Polymorphism** to add features without touching old code.

LSP	Parent-Child Trust	If a child class can't do everything the parent does, it shouldn't be a child.

History:

These principles were introduced by **Robert C. Martin** (Uncle Bob) to solve issues of spaghetti code in large software projects.

Note:

The video covers the first 3 principles (S, O, L). The remaining two (I, D) are covered in the next part of the series.

Interview Questions

1. Why is the Single Responsibility Principle (SRP) important?

Answer: It makes code easier to maintain. If a class does too many things (Calculation + DB + Printing), changing one part (e.g., DB logic) might accidentally break another part (e.g., Calculation).

2. How does Open/Closed Principle help in real projects?

Answer: It prevents "Regression Bugs". Since you don't modify existing, working code to add new features, you don't risk breaking what is already working.

3. Give a real-world example of Liskov Substitution Principle violation.

Answer: A Square class inheriting from a Rectangle class. A rectangle's width and height can be different, but a square's must be same. If you set width on a Square object treated as a Rectangle, it might behave unexpectedly, violating LSP. (Or the Banking Example: FixedDeposit inheriting Withdraw capability).

4. What is the difference between Abstraction and Interface in context of OCP?

Answer: Interfaces allow us to define a contract (e.g., save()). New classes implement this contract differently (SQL, Mongo). The main code relies on the Interface, not the concrete classes, making it "Open for extension".

5. Can a class have multiple methods and still follow SRP?

Answer: Yes! SRP doesn't mean a class has only *one method*. It means all methods in the class should relate to *one single responsibility* or actor (e.g., all methods related to Invoice Printing).