🚀 Lecture 05 : SOLID Design Principles | Complete Guide with Code Examples🎯

📚 Introduction – Kya Problem Solve Karte Hain SOLID?

🔥 Common Software Problems:

Problem	Explanation
🔧 **Maintainability Issues**	Naye features add karte time purana code toot jaata hai
📖 **Readability Problems**	Naye developers ko code samajhne mein time lagta hai
🐛 **Debugging Challenges**	Bugs fix karne mein extra effort
🔗 **Tight Coupling**	Ek class badli to dusri bhi affect ho gayi

🧑‍💼 **Creator:** Robert C. Martin (2000)
SOLID = 5 Design Principles ka acronym

> 💡 *Goal:* Code ko maintainable, scalable aur reusable banana.

1️⃣ S - Single Responsibility Principle (SRP) 🎯

📖 Definition (Hinglish):

> "Ek class ka ek hi kaam hona chahiye"
> "Ek class ko change karne ka ek hi reason hona chahiye"

💡 Real-Life Example:

📺 TV Remote → Sirf TV control kare.
❌ Agar ek remote TV + Fridge + AC sab control kare to maintenance nightmare!

🛒 Shopping Cart Example

❌ Galat Tarika (SRP Todna) – *Figure 1*

```
┌────────────────────────────────┐
│       🛒 ShoppingCart      │
├────────────────────────────────┤
│ - products                 │
├────────────────────────────────┤
│ + addProduct()             │
│ + getProducts()            │
│ + calculateTotalPrice()    │
│ + printInvoice()       ←❌ Printing ka kaam
│ + saveToDatabase()    ← ❌ Database ka kaam
└────────────────────────────────┘
```

⚠️ Problem:

Ek hi class me 3 alag responsibilities —
1️⃣ **Business Logic** (add, calculate)
2️⃣ **Printing** (invoice)
3️⃣ **Database Operations** (save data)

🧠 *Result:*

- Code tightly coupled ho gaya
- Maintain karna mushkil
- SRP (Single Responsibility Principle) ka violation 🚫

✅ Sahi Tarika (SRP Follow) – Figure 2

```
┌────────────────────────────────┐
┌────────────────────────────┐
│     🛒 ShoppingCart     │       │ 🧾 ShoppingCartPrinter  │
├────────────────────────────────┤
├────────────────────────────┤
│ - products               │       │ - cart                 │
│ + addProduct()           │       │ + printInvoice()       │
│ + totalPrice()           │     └────────────────────────────┘
└────────────────────────────────┘
```

```
┌──────────────────────────┐
│   💾 ShoppingCartDB       │
├──────────────────────────┤
│   - cart                 │
│   + saveToDB()           │
└──────────────────────────┘
```

💡 Explanation:

Har class ka ek hi purpose hai 👇
🛒 `ShoppingCart` → Product add aur total calculate karega
🧾 `ShoppingCartPrinter` → Invoice print karega
💾 `ShoppingCartDB` → Cart data ko database me store karega

🎯 **Followed Principle:**
SRP (Single Responsibility Principle) → Har class ek hi responsibility ke liye bana hai, isse code reusable, maintainable aur scalable banta hai 🔥

🎯 SRP Benefits

✅ Maintain karna easy
✅ Code readable
✅ Side effects kam
✅ Testing simple

💡 *Tip:* Jab bhi class me "and" ya "or" aata ho, SRP break ho raha hai!

2️⃣ O - Open/Closed Principle (OCP) 🔄

📖 Definition:

"Class extension ke liye open honi chahiye, modification ke liye closed."

💬 Simple Words:

- **Open for Extension:** Naye features add kar sakte ho
- **Closed for Modification:** Purane code ko touch nahi karna

🗄️ Database Storage Example

❌ Galat Tarika (OCP Todna) – Figure 3

```
┌──────────────────────────────┐
│         DBStorage            │
├──────────────────────────────┤
│ + saveToSQL()                │ ✅ Original
│ + saveToMongo()              │ ❌ Modified / New
│ + saveToFile()               │ ❌ Modified / New
└──────────────────────────────┘
```

⚠️ Problem:

Har naye database type ke liye **class ko modify karna padta hai**

🧠 Result:

- **OCP (Open/Closed Principle) violate hua**
- Code tightly coupled ho gaya
- Maintainability aur scalability problem 😔

✅ Sahi Tarika (OCP Follow) – Figure 4

```
┌──────────────────────┐
│    Persistence       │  ← Interface
├──────────────────────┤
│ + save()             │
└──────────────────────┘
          △
          │
 ┌────────┴────────────┐
 │                     │
┌────────────┐   ┌────────────┐
```

```
|  SQLSave |    | FileSave |
├──────────┤    ├──────────┤
| + save() |    | + save() |
└──────────┘    └──────────┘
```

💡 Explanation:

- **Persistence (Interface)** → define karta hai `save()` method
- **SQLSave / FileSave** → implement karte hai `save()`
- **New Feature (e.g., MongoSave)** → add kar sakte ho **bina purane code ko modify kiye** ✅

🧠 Result:

- **OCP Followed** → Code open for extension, closed for modification
- Maintainable aur scalable design

🧰 Implementation Tips

- Abstraction + Inheritance + Polymorphism use karo
- Client sirf interface se baat kare

🎯 OCP Benefits

✅ Regression risk kam
✅ Easy scalability
✅ Loose coupling
✅ Future-proof design

💡 *Tip:* Agar tumhe “if/else” chain lag rahi ho naye types ke liye → OCP break ho raha hai!

3️⃣ L - Liskov Substitution Principle (LSP) 🔁

📖 Definition:

> “Child class parent class ki jagah use ho sakti ho.”

💬 Simple Words:

Client ko farq nahi padna chahiye ki kaunsa subclass use ho raha hai.
Behavior consistent rehna chahiye.

🏢 Bank Account Example

❌ Galat Tarika (LSP Todna) – Figure 5

```
┌──────────────┐
│   Account    │
├──────────────┤
│ + deposit()  │
│ + withdraw() │
└──────────────┘
      △
      │
┌────────────┬────────────┬──────────────┐
│ Savings    │ Current    │ FixedDeposit │
│ Account    │ Account    │ Account      │
└────────────┴────────────┴──────────────┘
                    ↓
          ❌ withdraw() throws Exception!
```

⚠️ Problem:

`FixedDepositAccount` **parent class ka contract tod raha hai**

- Parent `Account` ke `withdraw()` ko override kar raha hai, lekin functionality inconsistent hai

🧠 Result:

- **LSP (Liskov Substitution Principle) violate hua**
- Polymorphism aur code reuse break hota hai

- Maintainability aur reliability problem 😔

✅ Sahi Tarika (LSP Follow) – Figure 6

```
┌──────────────────────┐
│   DepositOnly      │  ← Interface 1
├──────────────────────┤
│ + deposit()        │
└──────────────────────┘
          △
          │
┌──────────────────────┐
│  Withdrawable      │  ← Interface 2
├──────────────────────┤
│ + withdraw()       │
└──────────────────────┘
          △
          │
  ┌─────────┴──────────┐
  │                │
┌─────────────┐    ┌─────────────────┐
│ Savings   │    │ FixedDeposit │
│ Account   │    │ Account      │
└─────────────┘    └─────────────────┘
```

💡 Explanation:

- `DepositOnly` → sirf deposit kar sakta hai
- `Withdrawable` → withdraw bhi kar sakta hai
- `SavingsAccount` → dono deposit & withdraw implement karta hai

- `FixedDepositAccount` → sirf deposit implement karta hai

🧠 Result:

- Har class **apni capability ke hisaab se behave karti hai** ✅
- **LSP Followed** → Parent class contract violated nahi hota
- Polymorphism aur maintainability safe hai 🔥

🎯 LSP Rules

✅ Child class parent ke behavior ko restrict nahi kare
✅ "Unimplemented" method me exception mat throw karo
✅ Client code me `type checking` avoid karo

💡 *Tip:* Agar subclass ka behavior parent se alag lagta hai — LSP break!

🧩 Quick Summary Table

Principle	Acronym	Core Idea	Galat Example	Sahi Solution
Single Responsibility	SRP	Ek class = ek kaam	Cart me printing + DB	Separate classes
Open-Closed	OCP	Extend, modify mat karo	DBStorage modify karna	Interface & Inheritance
Liskov Substitution	LSP	Child replace kar sake	FDAccount withdraw nahi kar sakta	Logical hierarchy

---

🧠 Golden Rule

SRP + OCP + LSP = Clean, scalable aur maintainable codebase 💪

□ Tips & Tricks

💡 *Memory Trick:*
S → **O** → **L** = “Simple, Open, Logical” flow
→ Code bhi waise hi hona chahiye!

⚙️ *Real-World Tip:*
Agar code me multiple “reasons to change” mil rahe ho → SRP break.
Agar “if-else” add kar rahe ho naye features ke liye → OCP break.
Agar subclass “nahi chal raha parent ke jagah” → LSP break.

🚀 Aage Kya Hai

4️⃣ **I - Interface Segregation Principle (ISP)**
5️⃣ **D - Dependency Inversion Principle (DIP)**

💡 *Final Mantra:*

> “SOLID follow karo → Code likho jo kal bhi chale!” ✨