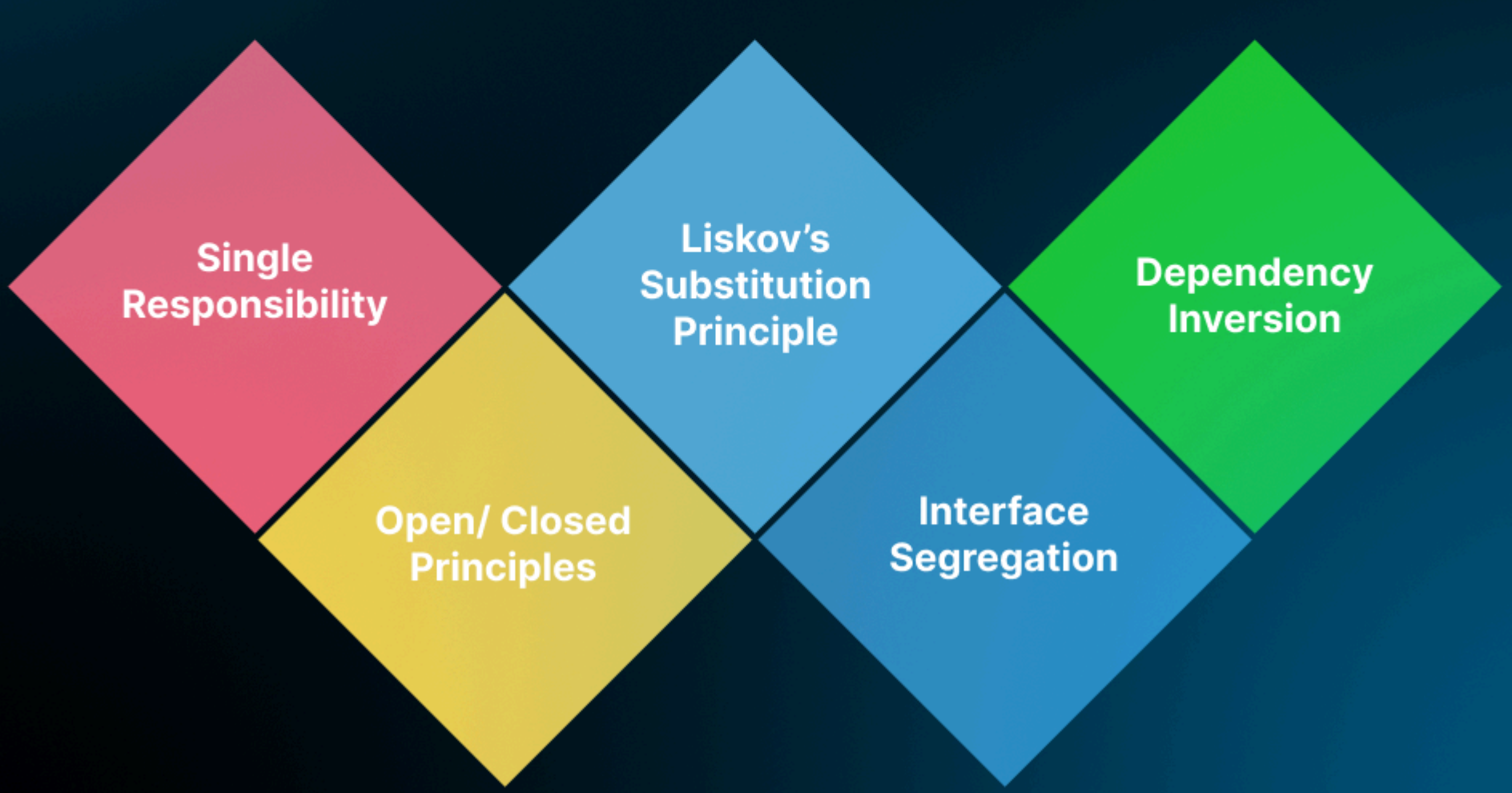
Meritshot | In Collaboration with Microsoft

# S.O.L.I.D. Principles




For Working Professionals

# SOLID Principles – Complete Learning Guide for Software Developers

## What Are SOLID Principles?

SOLID is a set of five object-oriented design principles that help developers write:

- Clean
- Maintainable
- Scalable
- Testable

**software systems.**

**Acronym**:

1. S — Single Responsibility Principle (SRP)
2. O — Open/Closed Principle (OCP)
3. L — Liskov Substitution Principle (LSP)
4. I — Interface Segregation Principle (ISP)
5. D — Dependency Inversion Principle (DIP)

Meritshot
EDUCATION

## Single Responsibility Principle (SRP)

### Definition

A class should have one and only one reason to change.
Meaning: A class should do one job only.

### Why SRP Matters

- Reduces code complexity
- Makes testing easier
- Avoids massive “God classes”
- Prevents bugs caused by unrelated changes

### Real-World Analogy

**A restaurant has:**

- Chef
- Waiter
- Cashier

If one person does everything → chaos.
Code works the same way.

**Bad Example (Violation)**

```
class Invoice {
    void calculateTotal() { }
    void saveToDb() { }
    void generatePDF() { }
    }
```

Here, the class is handling:

- Business logic
- Database logic
- File generation

Too many responsibilities.

**Correct (SRP Applied)**

```
class InvoiceCalculator { void calculateTotal() {} }
class InvoiceRepository { void saveToDb() {} }
class InvoicePDFGenerator { void generatePDF() {} }
```

Each class handles ONE reason to change.

Meritshot
EDUCATION

# Open / Closed Principle (OCP)

## Definition

Software entities should be open for extension but closed for modification.
Meaning: You should be able to add new functionality without changing existing code.

## Why OCP Matters

- Avoids breaking stable code
- Makes apps flexible for new requirements
- Perfect for plugin-style architectures

## Bad Example

```
def get_discount (user_type):

    if user_type == "regular": return 10
    if user_type == "prime": return 20
```

Every new user type = modify function → risk of breaking.

**Correct Example (OCP Applied)**

```
class Discount:
    def get(self): pass

class RegularDiscount(Discount):
   def get(self): return 10

class PrimeDiscount(Discount):
   def get(self): return 20
```

**To add “GoldDiscount”, you add a new class, not modify existing ones.**

Liskov Substitution Principle (LSP)

**Definition**

Child classes must be able to substitute their parent class without breaking the app.

**Why LSP Matters**

- Ensures inheritance hierarchy makes sense
- Prevents unexpected behavior
- Guarantees consistent API usage

Meritshot
EDUCATION

**Bad Example**

```
class Bird { void fly() {} }

class Penguin extends Bird {
void fly() { throw new UnsupportedOperationException(); }
}
```

Penguin cannot fly → violates LSP.

**Correct Fix**

```
interface Bird { }

interface FlyingBird extends Bird { void fly(); }

class Sparrow implements FlyingBird { public void fly() {} }
class Penguin implements Bird { }
```

Proper hierarchy = no contradictions.

Meritshot
EDUCATION

## Interface Segregation Principle (ISP)

Definition: Clients should not be forced to depend on interfaces they do not use.

**Why ISP Matters**

- Avoids “fat” interfaces
- Classes only implement what they need
- Makes code flexible for future changes

**Bad Example**

```
interface Worker {
    void work();
    void eat();
                        }

     class Robot : Worker {
      public void eat() { /* not applicable */ }
      }
```

**Robot doesn’t eat → unnecessary dependency.**

**Correct (ISP Applied)**

```
interface Workable { void work(); }
interface Eatable { void eat(); }

class Human: Workable, Eatable { }
class Robot: Workable { }
```

Interfaces are split into meaningful contracts.

## Dependency Inversion Principle (DIP)

**Definition**

- High-level modules should not depend on low-level modules.
- Both should depend on abstractions.
- Abstractions should not depend on details.
- Details should depend on abstractions.

Meritshot
EDUCATION

**Correct Example (DIP Applied)**

```
class Database:
    def connect(self): pass

class MySQLDatabase(Database):
    def connect(self): pass

class UserService:
    def __init__(self, db: Database):
        self.db = db
```

**Now you can inject MongoDB, PostgreSQL, etc.**

Meritshot
EDUCATION

## When to Apply SOLID?

**Use SOLID when:**

✓ Codebase is growing
✓ Multiple developers are working
✓ Frequent feature changes happen
✓ Testability & maintainability matter

**Avoid overusing SOLID when:**

❌ Project is very small
❌ Abstractions increase unnecessary complexity
❌ You don’t yet know future requirements

Meritshot
EDUCATION

## SOLID Principles in Real Companies

**Used in:**

- Microservices
- Modular monoliths
- Event-driven systems
- Enterprise backend apps
- Android/iOS apps
- Large frontend apps (React/Angular)

## Common Patterns influenced by SOLID:

- Strategy pattern
- Factory pattern
- Repository pattern
- Dependency Injection
- Interfaces & abstractions

Meritshot
EDUCATION

## Interview Questions (With Answers)

**Q1. What is SRP? Give an example.**
SRP states that a class should have one reason to change. Example: separate classes for invoice calculation, PDF generation, and database saving.

**Q2. How does OCP make code flexible?**
You extend behavior (via inheritance/interfaces) without modifying existing code → reduces risk.

**Q3. What is LSP? Why is it important?**
Child classes should behave like parent classes. Ensures reliable inheritance.

**Q4. Why is DIP important in modern development?**
Decouples high-level logic from low-level details → easier testing, cleaner architecture.

**Q5. Give real examples of ISP.**
Splitting large interfaces like "IMachine" into “IPrint”, “IScan”, “IFax”.

Meritshot
EDUCATION

**Your one-step destination for your Career Upskilling**

**Lets make a community of 20k+ learnerrs**

meritshoteducation