Software design principles

1️⃣ DRY – Don’t Repeat Yourself

💡 Every piece of knowledge must have a **single, unambiguous representation** in code. Duplication leads to bugs and maintenance pain.

```
// ❌ Bad
if(user.role.equals("ADMIN")) { ... }
if(account.owner.role.equals("ADMIN")) { ... }


// ✅ Good
boolean isAdmin(User user) { return "ADMIN".equals(user.role); }
```

👉 Write it once, use it everywhere.

2️⃣ KISS – Keep It Simple, Stupid

👨‍💻 Complexity is a liability.
The best code is often the **simplest code that works**.

```
// ❌ Over-engineered

new BigDecimal("2").pow(3).intValue();


// ✅ Simple

int result = 2 * 2 * 2;
```

👉 Don’t be clever, be clear.

3️⃣ YAGNI – You Aren't Gonna Need It

🔮 Don't build features just because you *think* you'll need them.
Chances are… you won't.

```
// ❌ Premature abstraction

class AbstractUserManagerFactory { ... }


// ✅ Just enough

class UserService { ... }
```

👉 Build for today, not for hypothetical futures.

4️⃣ SRP – Single Responsibility Principle (S in SOLID)

Each class should have **one reason to change**.

```
// ❌ Mixing concerns

class Report {
   void generate() {}

   void saveToFile() {}
}


// ✅ Separation

class ReportGenerator {}
class ReportSaver {}
```

👉 One class = one responsibility.

5️⃣ OCP – Open/Closed Principle (O in SOLID)

Open for extension ✅, closed for modification ❌.

```
// ❌ Modifying every time
if(shape instanceof Circle) { ... }
if(shape instanceof Rectangle) { ... }

// ✅ Extensible
interface Shape { double area(); }
class Circle implements Shape { ... }
class Rectangle implements Shape { ... }
```

👉 Add new behavior without breaking old code.

6️⃣ LSP – Liskov Substitution Principle (L in SOLID)

Subtypes must be usable **without breaking the base type's behavior**.

```
// ❌ Violates LSP
class Square extends Rectangle {
   void setWidth(int w) {
     super.setWidth(w); super.setHeight(w);
 }
}
```

👉 A subclass should be a proper replacement for its parent.

7️⃣ Interface Segregation Principle (I in SOLID)

Clients shouldn’t depend on methods they don’t use.

```
// ❌ Fat interface

interface Printer { void print(); void scan(); void fax(); }


// ✅ Split responsibilities

interface Printer { void print(); }
interface Scanner { void scan(); }
```

👉 Keep interfaces focused.

8️⃣ Dependency Inversion Principle (D in SOLID)

Depend on **abstractions**, not concretions.

```
// ❌ Direct dependency

class UserService {
   MySqlRepository repo = new MySqlRepository();
}



// ✅ Inversion
class UserService {
   Repository repo;
}
```

👉 Makes code testable and flexible.

9️⃣ Composition over Inheritance

Prefer **has-a** over **is-a** when modeling.

```
// ❌ Deep inheritance

class ElectricCar extends Car {
... }


// ✅ Composition

class ElectricCar {
   private Engine engine;
}
```

👉 More flexible, less fragile.

🔟 Boy Scout Rule

🏕️ *“Always leave the campground cleaner than you found it.”*
In code: whenever you touch a file, make it a little better.

```
// Before
if(a==true) { return false; }

// After
if(a) return false;
```

👉 Small improvements = big impact over time.