SOLID Principles

Introduction to SOLID Principles:

SOLID is an acronym that represents five principles of object-oriented design aimed at creating more maintainable and flexible software. They are meant to help developers design software that is easy to maintain and extend over time.

The five principles that make up SOLID are:

- **Single Responsibility Principle (SRP)**: Each class or module in your code should have a single responsibility, meaning it should do one thing and do it well. This makes it easier to understand, test, and modify your code.
- **Open/Closed Principle (OCP)**: Your code should be open for extension but closed for modification. This means that you should be able to add new functionality without changing the existing code.
- **Liskov Substitution Principle (LSP)**: Subtypes should be able to be used in place of their base types without affecting the correctness of your program. This means that you should be able to use a derived class wherever its parent class is expected.
- **Interface Segregation Principle (ISP)**: Clients should not be forced to depend on interfaces they do not use. This means that you should separate interfaces

into smaller, more specific ones so that clients only need to implement the methods they actually use.

- **Dependency Inversion Principle (DIP)**: High-level modules should not depend on low-level modules. Both should depend on abstractions. This means that you should use interfaces or abstract classes to define the dependencies between modules, instead of using concrete implementations.

By following these principles, you can create software that is more maintainable, flexible, and reusable.

Benefits of SOLID:

> 👉 SOLID principles improve software quality. They make software more maintainable, flexible, and reusable. Other advantages include:
>
> - Increased testability: When you have smaller, more focused classes, they are easier to test.
> - Easier to refactor: When code is well-organized and follows SOLID principles, it is easier to change without introducing new bugs.
> - Better code organization: SOLID principles provide a structure for code that makes it easier to understand and navigate.
> - Reduced complexity: Code that follows SOLID principles tends to be less complex and easier to reason about.

Single Responsibility Principle (SRP) - Definition:
The Single Responsibility Principle, or SRP, is the 'S' in SOLID. It states that a class should have only one reason to change. In other words, a class should have a single responsibility or job. This helps in keeping classes focused, making them easier to understand, maintain, and modify."

Understand SRP with an Example:
Let's understand SRP with a Java example. Imagine we're building a simple order processing system. We have a class called `Order` which represents an order.

Code Example - Violation of SRP:

```
class Order {
    public void createOrder() {
        // Logic to create an order
    }

    public void generateInvoice() {
        // Logic to generate an invoice
    }

    public void sendConfirmationEmail() {
        // Logic to send a confirmation email
    }
}
```

In this example, the `Order` class handles three distinct responsibilities: creating orders, generating invoices, and sending confirmation emails. This violates the SRP, as the class has more than one reason to change.

Applying SRP - Refactoring the Code:

To adhere to the SRP, we can refactor the code by creating separate classes for each responsibility. Let's start with the `Order` class.

Code Example - Applying SRP:

```
class Order {
    public void createOrder() {
        // Logic to create an order
    }
}

class InvoiceGenerator {
    public void generateInvoice(Order order) {
        // Logic to generate an invoice
    }
}

class EmailSender {
    public void sendConfirmationEmail(Order order) {
        // Logic to send a confirmation email
```

```
    }
}
```

Now, we have three separate classes, each with a single responsibility. The `Order` class handles only order creation, the `InvoiceGenerator` class handles invoice generation, and the `EmailSender` class handles sending confirmation emails.

Benefits of SRP:

👉 By following the Single Responsibility Principle, we gain several benefits:

- Improved code organization and readability.
- Easier maintenance and debugging.
- Reduced impact of changes to one responsibility on other parts of the system.

Conclusion:
That wraps up our explanation of the Single Responsibility Principle, the first principle in the SOLID series. Remember, designing classes with a single responsibility contributes to a more flexible and maintainable codebase. In the next video, we'll dive into the 'O' in SOLID: the Open-Closed Principle.