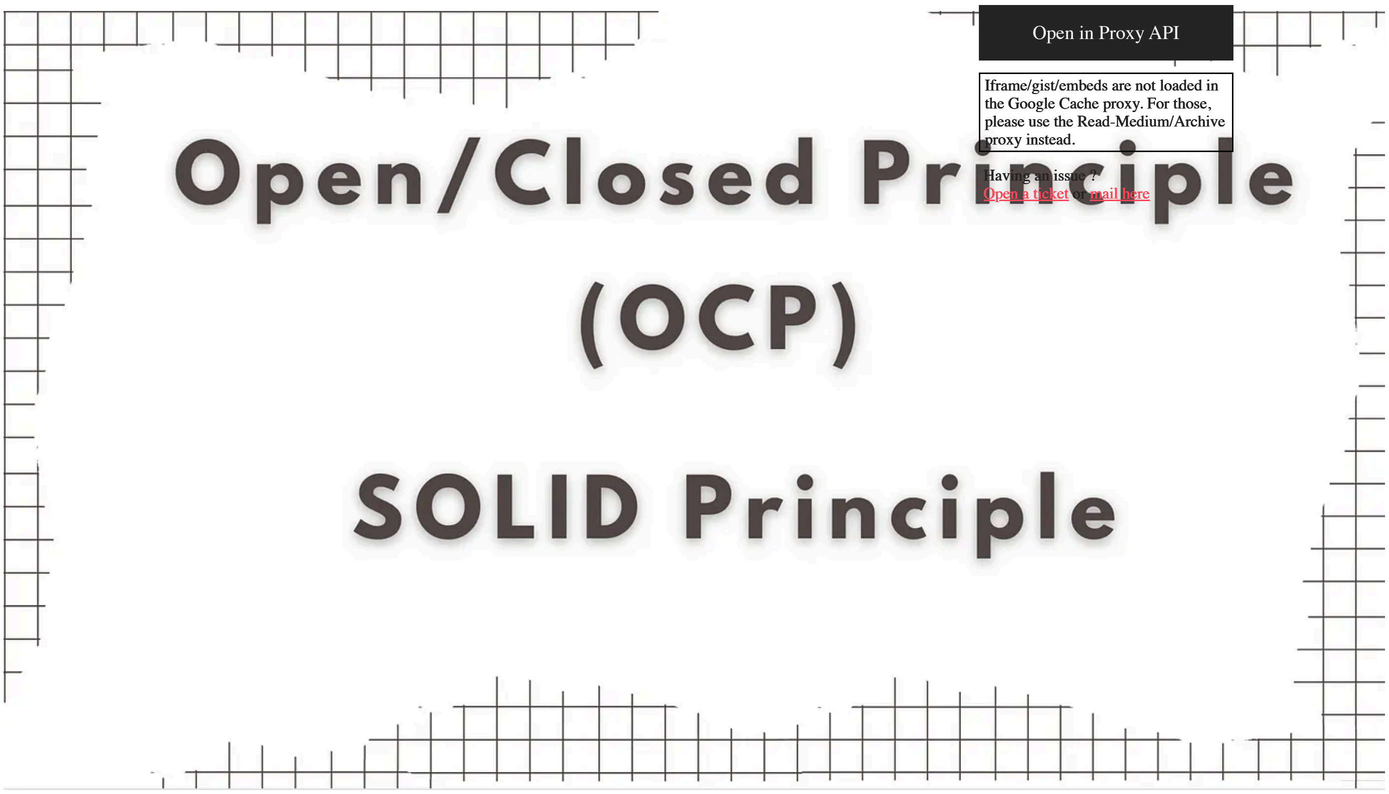
2. Open/Closed Principle(OCP) : SOLID Principle

Ramdhas · Follow

2 min read · Oct 22, 2023

Listen Share More



The Open/Closed Principle (OCP) is one of the SOLID principles of object-oriented design, which encourages the code to be open for extension but closed for modification.

- Open for extension
- Closed for modification
- Allows adding new features without changing existing code
- Enhances code extensibility and maintainability

Before OCP (Violation of OCP):

Imagine you have a class called `PaymentProcessor` that handles payments for your e-commerce application. Initially, it only supports credit card payments:

```
class PaymentProcessor {
    func processCreditCardPayment() {
        // Code to process credit card payment
    }
}
```

Later on, you decide to extend your application to support PayPal payments. To do this, you have to modify the existing `PaymentProcessor` class:

```
class PaymentProcessor {
    func processCreditCardPayment() {
        // Code to process credit card payment
    }

    func processPayPalPayment() {
        // Code to process PayPal payment
    }
}
```

In this “before” example, you violated the Open/Closed Principle because you had to modify the existing class to add support for a new payment method. This can introduce bugs and affect the stability of your existing codebase.

After OCP (Compliance with OCP):

To stick to the Open/Closed Principle, you can use an abstraction (e.g., a protocol) and create separate classes for each payment method without modifying the existing code:

```
protocol PaymentProcessing {
    func processPayment()
}

class CreditCardPaymentProcessor: PaymentProcessing {
    func processPayment() {
        // Code to process credit card payment
    }
}
class PayPalPaymentProcessor: PaymentProcessing {
    func processPayment() {
        // Code to process PayPal payment
    }
}
```

With this approach, you have introduced a protocol `PaymentProcessing`, and you've created specific classes for each payment method that conforms to this protocol. Now, when you need to add a new payment method, you can create a new class that implements the `PaymentProcessing` protocol, and you won't need to modify the existing code in the `PaymentProcessor` class.

By following the Open/Closed Principle, your code becomes more extensible and modular, making it easier to add new functionality without the risk of introducing errors into the existing code.

. . .

3. Liskov Substitution Principle (LSP):
https://medium.com/@ramdhasm5/3-liskov-substitution-principle-lsp-solid-principle-fc23a473939c

. . .

👏🏻👏🏻 👏🏻👏🏻 Applaud to express your encouragement. Join me for additional insights and let’s progress together.

IOS Swift Solid Ocp Open Closed Principle

Follow

Written by Ramdhas

225 Followers · 31 Following

Skills: iOS, Swift, SwiftUI, Html, Css, Javascript, React.js. Lives in Stockholm, Sweden.