Open in app Get started

KD Knowledge Diet Follow
Mar 30 · 4 min read · Listen

Save

Open Closed Principle: Make your code cost-free and flexible

Anecdote: Computer is closed for modification but it can be extended

Software is never dormant. It's constantly changing. The annoying fact about the software development is that every developer should build and re-build the same software as requirements from clients change. Frequent fixes are annoying, and often cause bugs. Bugs in the business means 'money'. Your client's money and your money too. So, how can you make your software bug-free? I will introduce you a great concept for that. It's one of SOLID Principles. **OPEN CLOSED PRINCIPLE.**

What is OPEN CLOSED PRINCIPLE?

> ***Software components should be closed for modification, but open for extension.***

Can you see the problem?

```
public class InsuranceCompany {

    public double discountRate(VehicleInsuranceCustomer customer) {
        if(customer.isLoyalCustomer()) {
            return 20;
        }
        return 10;
    }

}

class VehicleInsuranceCustomer {
    public boolean isLoyalCustomer() {
        return true;
    }
}
```

`InsuranceCompany` is holding a method `discountRate()`. But as you can see, the method parameters can only accept `VehichleInsuranceCustomer`. But what if I want to introduce another customers? You have to define another method again.

```
public class InsuranceCompany {

    public double discountRate(VehicleInsuranceCustomer customer) {
        if(customer.isLoyalCustomer()) {
            return 20;
        }
        return 10;
    }

    public double discountRate(HomeInsuranceCustomer customer) {
        if(customer.isLoyalCustomer()) {
            return 20;
        }
        return 10;
    }

    public double discountRate(LifeInsuranceCustomer customer) {
      if(customer.isLoyalCustomer()) {
}

class VehicleInsuranceCustomer {
    public boolean isLoyalCustomer() {
        return true;
    }
}

class HomeInsuranceCustomer {
    public boolean isLoyalCustomer() {
        return true;
    }
}

class LifeInsuranceCustomer {
    public boolean isLoyalCustomer() {
        return true;
    }
}
```

This is the problem. You are forced to introduce another method which executes the duplicate code block. This makes code hard to read. More important, you are fixing the codebase. This leads to a bug… You might understand my statement if you had ever developed a software. As your codebase grows, every time you directly modify your code, your chances of getting bugs increase dramatically.

How do I fix code according to OCP Principle?

```
public class InsuranceCompany {

    // Parameter is now abstracted
    public double discountRate(CustomerProfile customer) {
        if(customer.isLoyalCustomer()) {
            return 20;
        }
        return 10;
    }

}
```

```
class VehicleInsuranceCustomer implements CustomerProfile {

    public boolean isLoyalCustomer() {
        return true;
    }

}

class HomeInsuranceCustomer implements CustomerProfile {

    public boolean isLoyalCustomer() {
        return true;
    }

}

class LifeInsuranceCustomer implements CustomerProfile {

    public boolean isLoyalCustomer() {
        return true;
    }

}
```

Introduce interface. And make other **'additional classes'** conform to that interface. `discountRate()` method takes now 'interface' instead of a specific class. This is called **'abstraction'**. By doing this, *not only can you add other types of customers* ***(open for extension)****, you are not forced to modify directly your defined code* ***(closed for modification)***.

Benefits of this approach

- Ease of adding new features.
- Leads to minimal cost of developing and testing software.
- Open Closed Principle often requires decoupling, which, in turn, automatically follows the Single Responsibility Principle.

But this solution is not ultimate. Sometimes, code duplication works better. So, ***do not follow the open Closed Principle blindly***. Otherwise, you will end up with a huge number of classes. This can complicate your overall design.

Conclusion

Always be mindful of Open Closed Principle when you are writing code. When it is appropriate, apply it. Your code will look much better.

Always put the complexity in the first place. Would applying OCP make the code simpler? If not, don't follow it blindly. But if it does, apply it.

Other Articles for Solid Principles

Single Responsibility Principle: What? working as "a software engineer" you don't know it?!?!

Have you ever experienced that code refactoring doesn't make your code better? Adding spaghetti on top of spaghetti…

paigeshin1991.medium.com

Open Closed Principle: Make your code cost-free and flexible

Software is never dormant. It's constantly changing. The annoying fact about the software development is that every…

paigeshin1991.medium.com

Liskov Substitution Principle: Top Developer's technique which improves 2.5x

As you write code, doesn't anyone ever question whether your code

Interface Segregation Principle: Your only way to be a super competent developer

During development, the code tends to be long. At some point, the codebase becomes so bloated that you can't handle it…

paigeshin1991.medium.com

Dependency Inversion Principle: How Google Developers write code

To become a high-paid developer, you need to learn TDD. Basically, you have to develop software with TDD to get into a…

paigeshin1991.medium.com