



+Devs2Blu

Linguagem de programação JAVA

Profª. Heloisa Moura

O que vamos ver:

- Princípio Aberto-Fechado (O - Open/Closed Principle)
- Princípio de Substituição de Liskov (L - Liskov Substitution Principle)

Princípio Aberto-Fechado (OCP - Open/Closed Principle)

Afirma que uma classe deve ser aberta para extensão, mas fechada para modificação. Isso significa que você pode adicionar novas funcionalidades ao código sem precisar alterar o código existente.

- **Aberto para extensão:** O comportamento de uma classe ou método pode ser estendido sem precisar alterar o código original. Isso permite adicionar novas funcionalidades sem mexer no código que já foi testado e validado.
- **Fechado para modificação:** Uma vez que uma classe ou método foi implementado e validado, ele não deve ser alterado. Alterar o código pode introduzir novos bugs ou quebrar funcionalidades já existentes.

Princípio Aberto-Fechado (OCP - Open/Closed Principle)

- Pense em um caminhão: toda a sua implementação, como motor, bateria e cabine é fechada para modificação.
- Porém, podemos estender as tarefas que ele realiza dependendo da carroceria que anexamos.



Princípio Aberto-Fechado (OCP - Open/Closed Principle)

Na programação:

imagine que temos uma loja virtual e essa loja tem uma classe que calcula descontos, para a venda de diferentes tipos de produtos. Inicialmente, todos os produtos estão recebendo o mesmo desconto.

Temos o código

```
public class CalculadoraDesconto {  
  
    public double calcularDesconto(Produto produto) {  
        // 10% de desconto para eletrônico  
        return produto.getPreco() * 0.10;  
    }  
}
```

+Devs2Blu

Fundamentos avançados OOP

- Precisamos incluir uma nova funcionalidade ao sistema:

A loja vai começar a diferenciar os descontos para os diferentes tipos de produtos. Começando com eletrônico e alimento. Como incluir isso no nosso código?

Uma alternativa seria verificar qual o tipo de produto foi escolhido para aplicar o desconto.

Temos o código:

+Devs2Blu

Fundamentos avançados OOP

```
public class CalculadoraDesconto {  
  
    public double calcularDesconto(Produto produto) {  
        if (produto.getTipoProduto().equals("Eletronico")) {  
            return produto.getPreco() * 0.10;  
            // 10% de desconto para eletrônico  
        } else if (produto.getTipoProduto().equals("Alimento")) {  
            return produto.getPreco() * 0.05;  
            // 5% de desconto para alimentos  
        }  
        return 0;  
    }  
}
```


Problemática

- A princípio parece tudo certo. Nosso código executa normalmente e conseguimos adicionar a funcionalidade corretamente.
- Mas, se além de eletrônico e alimento a loja passasse a aplicar desconto também para roupas e eletrodomésticos?

Seguindo a lógica, iríamos adicionar mais um else if no código, seguido do desconto para cada tipo de produto que entrou.

Temos o código:

+Devs2Blu

Fundamentos avançados OOP

```
public class CalculadoraDesconto {  
  
    public double calcularDesconto(Produto produto) {  
        if (produto.getTipoProduto().equals("Eletronico")) {  
            return produto.getPreco() * 0.10;  
            // 10% de desconto para eletrônico  
        } else if (produto.getTipoProduto().equals("Alimento")) {  
            return produto.getPreco() * 0.05;  
            // 5% de desconto para alimentos  
        } else if (produto.getTipoProduto().equals("Roupa")) {  
            return produto.getPreco() * 0.15;  
            // 15% de desconto para alimentos  
        } else if (produto.getTipoProduto().equals("Eletrodomestico")) {  
            return produto.getPreco() * 0.20;  
            // 20% de desconto para alimentos  
        }  
  
        return 0;  
    }  
}
```

+Devs2Blu

Fundamentos avançados OOP

Princípio Aberto-Fechado (OCP - Open/Closed Principle)

- Essa definitivamente não é uma boa estratégia. Cada vez que incluir uma função, a classe vai ficando mais complexa, precisando toda vez ser modificada, violando o OCP.
- Por isso, é necessário uma estratégia para adicionar mais recursos ao projeto, sem modificar e bagunçar a classe original.

Princípio Aberto-Fechado (OCP - Open/Closed Principle)

Solução

Nesse cenário, o projeto compreende vários tipos de produto. Assim, podemos criar uma classe ou uma interface que representa desconto de forma genérica, seguindo o OCP.

- Classe Produto: Mantém as informações de preço e tipo do produto.
- Interface Desconto: define o método `calcularDesconto()`

A cada tipo de produto fornecido pela loja, é possível criar novos tipos de `calcularDesconto`, mais específicos, que irão implementar a interface. Assim, podemos ter o código refatorado:

+Devs2Blu

Fundamentos avançados OOP

Princípio Aberto-Fechado (OCP - Open/Closed Principle)

Exemplo prático

Usando classe concreta

Usando interface

+Devs2Blu

Fundamentos avançados OOP

Resumo

- O Princípio Aberto/Fechado (OCP) nos permite estender o comportamento do software sem modificá-lo diretamente.
- Aplicamos esse princípio em Java, utilizando interfaces e classes separadas para diferentes tipos de descontos.
- Com o OCP, nosso código se torna mais flexível e fácil de manter.

+Devs2Blu

Fundamentos avançados OOP

Princípio Aberto-Fechado (OCP - Open/Closed Principle)

Exercícios