SOLID — OCP — Open Closed Principle

THIAGO ARAGÃO JANUARY 13, 2019

um dos mais importantes da OOP (princípios da orientação a objetos) e certamente um dos mais complicados para se implementar, por isso dificilmente programadores o praticam.

O principio tem como objetivo trabalhar diretamente com a manutenção das classes e tem como premissa a seguinte afirmativa:

> "You should be able to extend a classes behavior, without modifying it."

or

> "Você deve ser capaz de estender um comportamento de uma classe sem a necessidade de modificá-lo."

Ou seja, as entidades de software como classes, módulos, funções, etc, devem estar abertas para extensão, porém fechadas para modificação.

Em outras palavras significa que esta classe pode ter seu comportamento alterado com facilidade quando necessário, sem a alteração do seu código fonte. Essa extensão pode ser feita através de herança, interface e composição.

Vamos ver um exemplo?

Acima temos o exemplo de uma classe que valida o tipo de e-mail para tratar a mensagem de acordo com o seu tipo, mas, quando uma nova forma de mensagem for criada, a classe deverá ser editada e um novo if deverá ser acrescentado a ela.

Bem simples a solução não? Mas, com isso não quebraríamos o princípio do OCP (Open closed Principle)?

O conceito de aberto/fechado (OCP) tem a premissa de criarmos novas classes para funcionalidades de tipos semelhantes, e caso tenhamos novas funcionalidades, novas classes sejam criadas.

Legal, e o que ganhamos aplicando o OCP?

Extensibilidade

Como citei, ao termos uma nova funcionalidade ou comportamento, não precisaremos alterar a classe já existente, e sim estendê-la. Com isso mantemos o código original confiável e intacto, e criamos um código com design duradouro, de qualidade e manutenibilidade altas.

Abstração

Com toda certeza já ouvimos sobre abstração, ela permite que toda a mágica aconteça. Se temos abstrações bem definidas, conseguimos de forma fácil estender os métodos da nossa aplicação.
O conceito OCP indica principalmente o uso da herança para praticarmos o extensão dos métodos.

Vamos ver como a classe deveria ficar?

Veja que na solução, criamos várias classes, cada uma com uma responsabilidade definida, suas próprias regras de negócios e sem a necessidade de alterarmos a funcionalidade padrão devido a criação de uma nova regra.

Este design é a base de muitos dos padrões de projetos, um exemplo é o Factory Method, onde as classes de negócio estão implementadas sem uma especificação da classe concreta.

Conforme comentei no post sobre SRP (Single Responsibility Principle), não deixe de acompanhar os próximos artigos e aprender um pouco mais sobre os princípios SOLID.

Comente sobre o artigo, e se gostar, compartilhe em suas redes sociais com seus amigos e parceiros de trabalho :)

Grande [] e até a próxima!!!