STUDIED

__/__/__

Gustavo de Oliveira Pontes

Resumo: The SOLID principle of Object-Oriented Programming

Códigos não reutilizáveis são definidos pelo Robert Martin (Uncle Bob), a partir de 3 termos: Rigidez, Fragilidade e Imobilidade.
Rigidez: Quando muda uma parte do programa, outra quebra
Fragilidade: Quando certas quebras em lugares não relacionados
Imobilidade: Quando o código não pode ser reutilizado fora do contexto original

Assim, foram introduzidos os princípios SOLID os: SRP (Single responsibility Principle), OCP (Open/Close Principle), LSP (Liskov Substitution Principle), ISP (Interface Segregation Principle), DIP (Dependency Inversion Principle).

1) SRP: Cada classe / método deve focar apenas em uma única função, e as classes devem possuir apenas um motivo para mudar. Por exemplo, uma classe que calcula a área de uma forma não deveria conter o output. Este deveria ser inserido em outra classe próprio para isso.

2) OCP: Você deveria ser capaz de estender o comportamento da classe sem precisar modificá-la. Por exemplo, um método consegue calcular a área de formas, porém apenas de retângulos, e para calcular a área de um círculo, seria necessário criar outro método ou modificar do. Cada classe deve conter sua própria implementação de método e a classe principal apenas pega a área da forma.

3) LSP: O design deve permitir que pode substituir qualquer ocorrência da classe parente com as classes filhas, ou seja, a classe-filho deve conseguir fazer o mesmo caso que o parente. Apesar de um quadrado ser um retângulo, o quadrado só recebe um parâmetro, enquanto o retângulo recebe dois, por isso o quadrado não pode ser filho de retângulo.

spiral

__/__/__

S T Q Q S S D

4) ISP: É melhor ter várias interfaces pequenas do que poucas grandes. Ao invés de criar uma interface com todos os comportamentos dos animais, temos várias interfaces com um único comportamento, tendo o design baseado em composição ao invés de forçar comportamentos não.

5) DIP: Código deve depender em abstrações, em preciso instanciar diversos objetos. Inversão de dependências e injeção de dependências é muitas vezes confundidos como a mesma coisa, porém, não são.

- Inversão de dependências: o princípio de inverter dependências
- Injeção de dependências: o ato de inverter dependências
- Injeção de construtores: forço a injeção de dependência a partir de um construtor
- Injeção de parâmetro (" " " " ") a partir de um parâmetro de um método, como um setter.

O objetivo disso é transformar algo concreto em algo abstrato, ou seja, escolher objeto em tempo de execução, e não de compilação. Ele segue 3 passos, no 1º, tiro o código inicial que instanciava uma classe junto de todos os seus métodos, no 2º, separamos os comportamentos em uma classe abstrata separada e criamos sub-classes para este objeto e invertemos este comportamento na classe principal (no caso, apagamos o comportamento do gato, criamos uma sub-classe para ele, e a invertemos como um comportamento no gato), porém ir ainda estamos instanciando o comportamento para ele criarmos. E no 3º, retiramos os acoplamentos e criamos um único objeto com o comportamento como parâmetro para criação do objeto, permitindo criar apenas uma classe para criar diferentes objetos com diferentes comportamentos.

spiral