

Resumo 5 - Pietro Branca Rossi

MEMO

O objetivo principal da P.O.O, segundo o autor, é gerenciar dependências para evita a criação de código que seja rígido, uma mudança quebra outras partes, frágil, modificações causam quebras em locais não relacionados, e não reutilizável, código que não pode ser reutilizado fora de seu contexto original. Os princípios SOLID foram introduzidos para criar designs de software mais compreensível, flexível e de fácil manutenção.

Os cinco princípios:

- SRP: Single Responsibility Principle.
- OCP: Open / Close principle.
- LSP: Liskov Substitution Principle.
- ISP: Interface Segregation Principle.
- DIP: Dependency Inversion Principle.

I. SRP.

Uma classe deve ter apenas uma única razão para mudar. Isso significa que cada classe ou módulo deve ter responsabilidade de sobre uma única parte da funcionalidade.

Resumo 5 - Pietro Branco Rossi

MEMO

de do Software.

- **Problema técnico:** Uma classe CalculateAreas é responsável tanto por áreas de várias formas quanto por imprimir o resultado no console. Se o formato da saída precisar mudar, a classe que contém a lógica de cálculo, que não mudou, precisará ser recompilada e redistribuída. Isso ~~que copia duas funções~~ responsabilidades distintas.
- **Solução técnica:** As responsabilidades são separadas em classes distintas. A classe CalculateAreas fica responsável apenas pelo cálculo da soma das áreas. Uma nova classe, OutputAreas, fica responsável apenas pelo cálculo da soma das por formatar e exibir o resultado, podendo conter métodos para diferentes formatos de console, como HTML. dessa forma, adicionar um novo formato de saída exige modificar apenas a classe OutputArea, sem impactar a lógica de cálculo.

2. OCP

As entidades de software (classe, módulo) devem ser abertas para extensões, mas fechadas para modificações.

Resumo 5 - Pietro Branco Rossi

MEMO

- Problema técnico: Uma classe CalculateAreas possui um método que calcula a área especificamente para um objeto Rectangle. Para o cálculo de áreas de um Circle, seria necessário modificar o código existente da classe CalculateAreas, o que viola o princípio.
- Solução técnica: Utiliza a abstração. Cria uma classe abstrata Shape com um método abstrato GetArea(). As classes concretas (Rectangle, Circle) herdam de Shape, indo cando o método getArea() sem precisar conhecer o tipo concreto da forma. Isso permite adicionar novas formas (nova subclasse de Shape) sem qualquer modificação na classe CalculateAreas, que performance "fechada" para alterações, mas "aberta" à extensão de novas funcionalidade.

3. L.S.P.

Sobtipos devem ser substituíveis por seus tipos base. Ou seja, uma instância de uma classe filha deve poder substituir uma instância de classe pai sem qualquer alteração alterar o comportamento esperado do programa.

Resumo 5 - Pietro Branco Rossi

MEMO

- Problema técnico:

Modelar uma class Square como sub-classe de Rectangle. O construtor da classe espera dois parâmetros (comprimento e largura), mas Square só precisa de um (lado). A classe "engana" o construtor pai passando o mesmo valor duas vezes (super(s)). Essa inconsistência viola o contrato da classe pai e pode gerar bugs, pois o comportamento de um Square é diferente de um Rectangle genérico.

- Solução: Um square não deve ser uma subclasse Rectangle. Ao invés disso, ambas devem ser classificadas separadamente em classes separadas que herdam de uma abstração comum, como Shape.

H.I.S.P.:

- Prob. Técnico: Uma ~~única~~ única interface monolítica IMammal contém múltiplos comportamentos, como eat() e makeNoise(). Uma classe que implementa essa interface é forçada a fornecer implementações p/ todos os métodos, mesmo que um comportamento específico não se aplique a

Resumo 5 - Pietro Branco Rossi

MEMO

ela.

Por exemplo: Um mamífero que absorve nutrientes pela pele não precisaria do método.

- Solução:

A interface grande é segregada em interfaces menores e mais coesas, cada uma definindo um único comportamento, como I Eat e I Make Noise. As classes podem então implementar apenas as interfaces correspondentes aos comportamentos que efetivamente possuem. Promovendo um design mais flexível e baseado em composições, desacoplando os comportamentos.

5. D.I.P.

- Problema técnico:

Uma classe