Relatório de Implementação SOLID

Matheus Batista Rodrigues TSI5

Este relatório explica como cada princípio SOLID foi aplicado, destacando os erros das implementações iniciais e como foram corrigidos.

1. SRP - Single Responsibility Principle (Princípio da Responsabilidade Única)

Problema na Classe Errada (FuncionarioIncorreto)

A classe FuncionarioIncorreto viola o SRP porque ela tem duas responsabilidades:

Gerenciar os dados do funcionário.

Calcular o salário líquido.

Isso significa que, se a regra de cálculo mudar, será necessário modificar essa classe, quebrando a separação de responsabilidades.

Solução na Classe Correta (FuncionarioCorreto + CalculadoraSalario)

FuncionarioCorreto apenas armazena os dados do funcionário.

CalculadoraSalario calcula o salário líquido, respeitando o SRP.

Agora, se a regra de cálculo mudar, apenas a classe CalculadoraSalario será alterada.

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

Problema na Classe Errada (DescontoIncorreto)

A classe DescontoIncorreto está fechada para extensão e aberta para modificação.

Sempre que um novo tipo de desconto precisar ser adicionado, será necessário modificar essa classe.

Solução na Classe Correta (Desconto e suas implementações)

Criamos a interface Desconto, e classes específicas (DescontoEletronico, DescontoRoupa) implementam diferentes tipos de desconto.

Agora, novas regras de desconto podem ser adicionadas sem modificar código existente.

3. LSP - Liskov Substitution Principle (Princípio da Substituição de Liskov)

Problema na Classe Errada (QuadradoIncorreto herdando de RetanguloIncorreto)

A herança não mantém um comportamento previsível:

Um quadrado deve ter largura = altura, mas a classe QuadradoIncorreto sobrescreve setLargura() e setAltura(), causando efeitos inesperados.

Solução na Classe Correta (RetanguloCorreto e QuadradoCorreto)

Criamos a classe abstrata Forma e implementamos RetanguloCorreto e QuadradoCorreto separadamente.

Isso evita que o quadrado herde métodos inconsistentes, respeitando LSP.

4. ISP - Interface Segregation Principle (Princípio da Segregação de Interfaces)

Problema na Interface Errada (TrabalhadorIncorreto)

A interface TrabalhadorIncorreto força todas as classes a implementarem os métodos trabalhar() e comer(), mas robôs não comem.

O método comer() no robô gera um erro (exceção).

Solução na Interface Correta (Trabalhador e SerVivo)

Criamos duas interfaces menores e específicas:

Trabalhador (para trabalhar).

SerVivo (para comer).

Humanos implementam ambas, enquanto robôs apenas Trabalhador, evitando código desnecessário.

5. DIP - Dependency Inversion Principle (Princípio da Inversão de Dependência)

Problema na Classe Errada (MotoristaIncorreto)

A classe MotoristaIncorreto depende diretamente da classe concreta Carro.

Isso torna impossível usar outro tipo de veículo sem modificar essa classe.

Solução na Classe Correta (Veiculo + Implementações)

Criamos a interface Veiculo.

CarroCorreto e Moto implementam Veiculo.

Agora, MotoristaCorreto depende da abstração Veiculo, podendo dirigir qualquer veículo sem precisar alterar a classe.