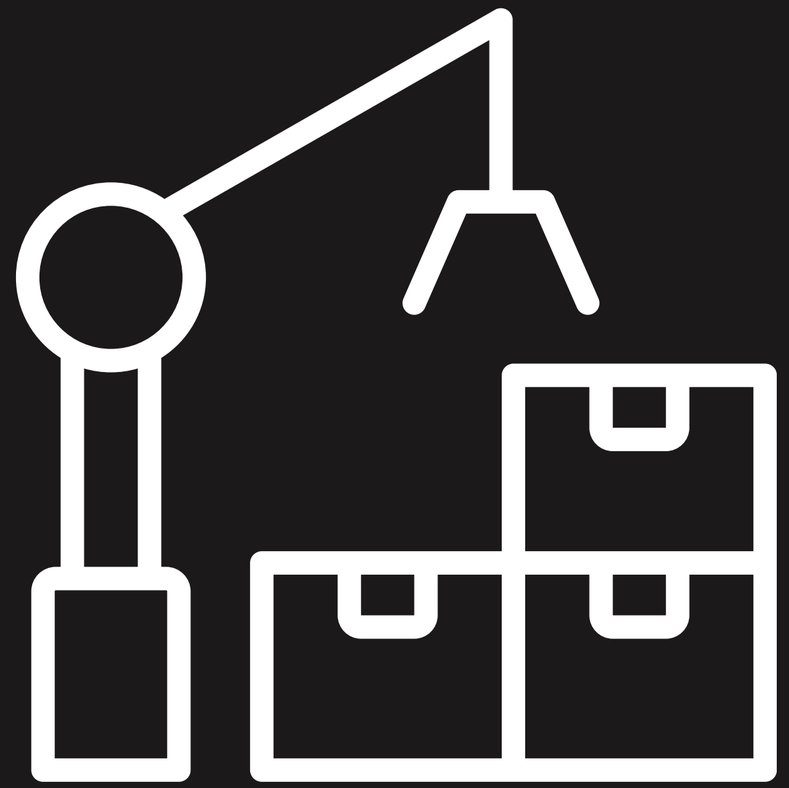
Faculdade Serra Dourada

SOLID

Prof.ª Juliana Conde

Os princípios SOLID fornecem diretrizes valiosas para escrever código limpo, modular e fácil de manter. Ao seguir esses princípios, podemos criar sistemas mais flexíveis, extensíveis e resilientes a mudanças.



S -Single Responsibility Principle - SRP (Princípio da Responsabilidade Única)

Este princípio afirma que uma classe deve ter apenas uma razão para mudar, ou seja, ela deve ter apenas uma responsabilidade. Classes com múltiplas responsabilidades tendem a ser difíceis de entender, testar e manter.

S -Single Responsibility Principle - SRP (Princípio da Responsabilidade Única)

Exemplo: Considere uma classe Employee que é responsável por calcular o salário de um funcionário e também por gerar relatórios. Seria melhor dividir essas responsabilidades em duas classes separadas: uma para calcular o salário e outra para gerar relatórios.

O - Open/Closed Principle - OPC
(Princípio Aberto/Fechado)

Este princípio afirma que as entidades de software (classes, módulos, funções, etc.) devem ser abertas para extensão, mas fechadas para modificação.

Isso significa que você deve ser capaz de estender o comportamento de uma classe sem modificar seu código-fonte.

O - Open/Closed Principle - OPC
(Princípio Aberto/Fechado)

Exemplo: Em vez de modificar diretamente uma classe existente para adicionar novos recursos, você pode criar subclasses ou utilizar interfaces para adicionar funcionalidades adicionais sem alterar o código existente.

L - Liskov Substitution Principle - LSP (Princípio da Substituição de Liskov)

Este princípio afirma que objetos de um tipo base devem ser substituíveis por objetos de um tipo derivado sem afetar a integridade do programa.

Em outras palavras, uma classe derivada deve ser capaz de substituir sua classe base sem alterar o comportamento do programa.

L - Liskov Substitution Principle - LSP (Princípio da Substituição de Liskov)

Exemplo: Se temos uma classe Rectangle com métodos setWidth e setHeight, e uma classe Square derivada de Rectangle, o método setWidth ou setHeight não deveria alterar as propriedades de largura e altura de forma diferente para a classe Square.

I - Interface Segregation Principle - ISP
(Princípio da Segregação de Interface)

Este princípio afirma que uma classe não deve ser forçada a depender de interfaces que ela não usa.

Em vez disso, as interfaces devem ser segregadas em interfaces menores e mais específicas, adequadas às necessidades das classes que as utilizam.

I - Interface Segregation Principle - ISP
(Princípio da Segregação de Interface)

Exemplo: Suponha que tenhamos uma interface Printable que define um método print. Se uma classe Scanner implementa essa interface, ela não deve ser obrigada a implementar um método que não é relevante para ela. Em vez disso, devemos dividir a interface em partes menores, como Printable e Scannable, para que as classes implementem apenas o que é necessário.

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

Este princípio afirma que módulos de alto nível não devem depender de módulos de baixo nível, ambos devem depender de abstrações.

Além disso, abstrações não devem depender de detalhes, mas sim detalhes devem depender de abstrações.
Isso promove o desacoplamento entre módulos e facilita a manutenção e a extensibilidade do código.

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

Exemplo: Em vez de uma classe de alto nível depender diretamente de uma classe de baixo nível, ambas devem depender de uma interface abstrata. Isso permite que as implementações concretas possam ser trocadas facilmente sem afetar a classe de alto nível.