Introdução ao SOLID

O que esperar do curso?

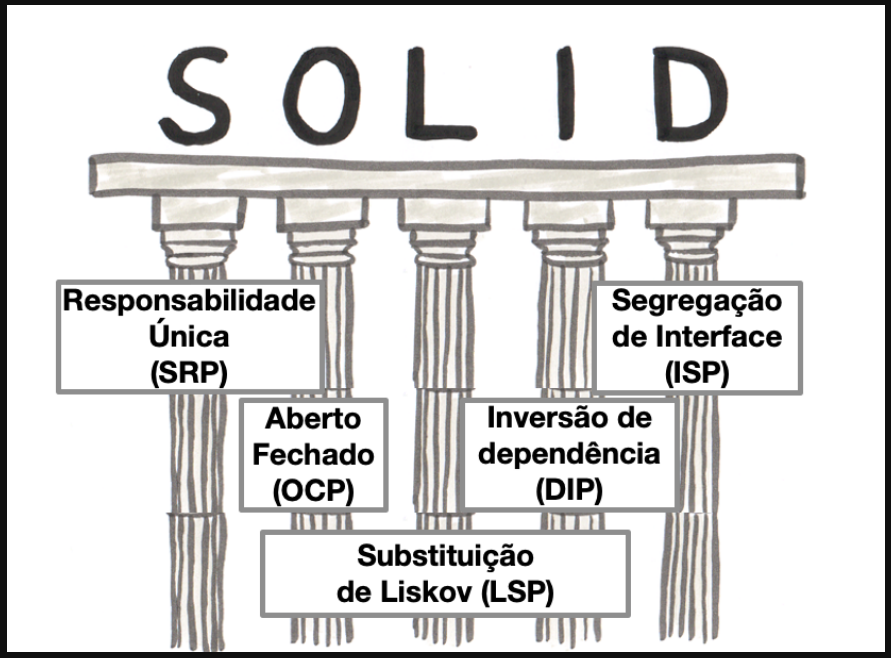
- **Visão geral:**
  - O que é SOLID?
  - Estudo de cada princípio com aplicações práticas
- **Método de ensino:**
  - Suporte téorico através dos slides
  - Repositório git com todos os exemplos de código

O que é SOLID?

- **Definição:** é um acrônimo que representa cinco princípios fundamentais de design de software, que buscam criar sistemas compreensíveis, flexíveis e manuteníveis (sustentáveis).
- Estes princípios foram organizados e publicado por Robert C. Martin (Uncle Bob), estes princípios são de extrema relevância em programação orientada à objetos (poo)

Por que preciso estudar isto?

- Código mais **manutenível**
- ~~Facilidade?~~ Flexibilidade para **estender** e **evoluir** o sistema
- Maior **reusabilidade** de código
- Código mais **testável**
- Redução de **acoplamento** aferente/eferente indevido entre componentes



SRP

Single Responsability Principle

Princípio da Responsabilidade Única

SRP

- **Conceito:** Uma classe (um componente), deve ter uma, e apenas uma, razão para mudar.

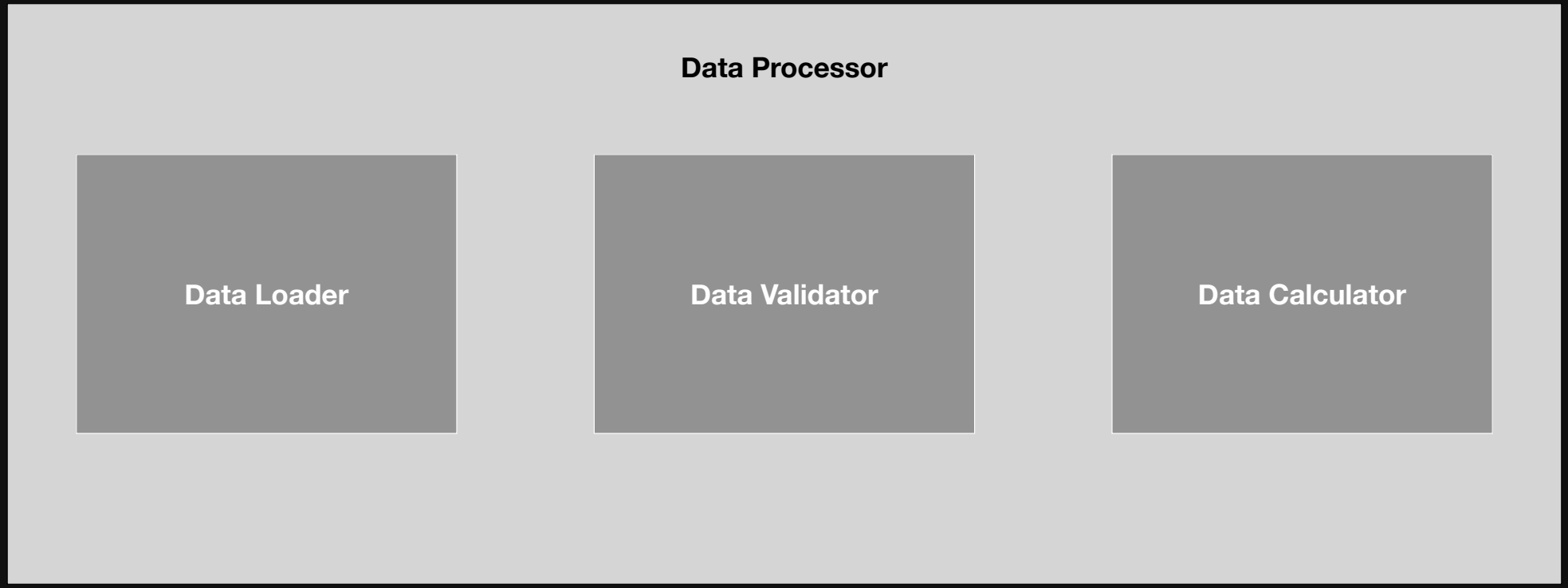
SRP - Exemplo 1

- **Exemplo teórico:** Imagine uma classe ***User*** que além da representação de um usuário também é responsável por salvar informações no banco de dados e enviar um e-mail.
- **Violação:** Explícita! Dificuldade de manutenção e alto grau de acomplamento

SRP – Exemplo 2

- **Exemplo teórico:** Imagine uma classe ***DataProcessor*** que possui 3 métodos: ***Load, Validate*** *e* ***CalculateStatistics.*** Todos estes métodos fazem parte do grupo de funcionalidades de “Processamento de dados”.
  - Load: Carrega as informações de um arquivo
  - Validate: Valida se os dados estão no formato correto e atendem as regras de negócio
  - CalculateStatistics: Calcula estatísticas com base nos dados importados
- **Violação:** Implícita! Dificuldade de manutenção e alto grau de acomplamento

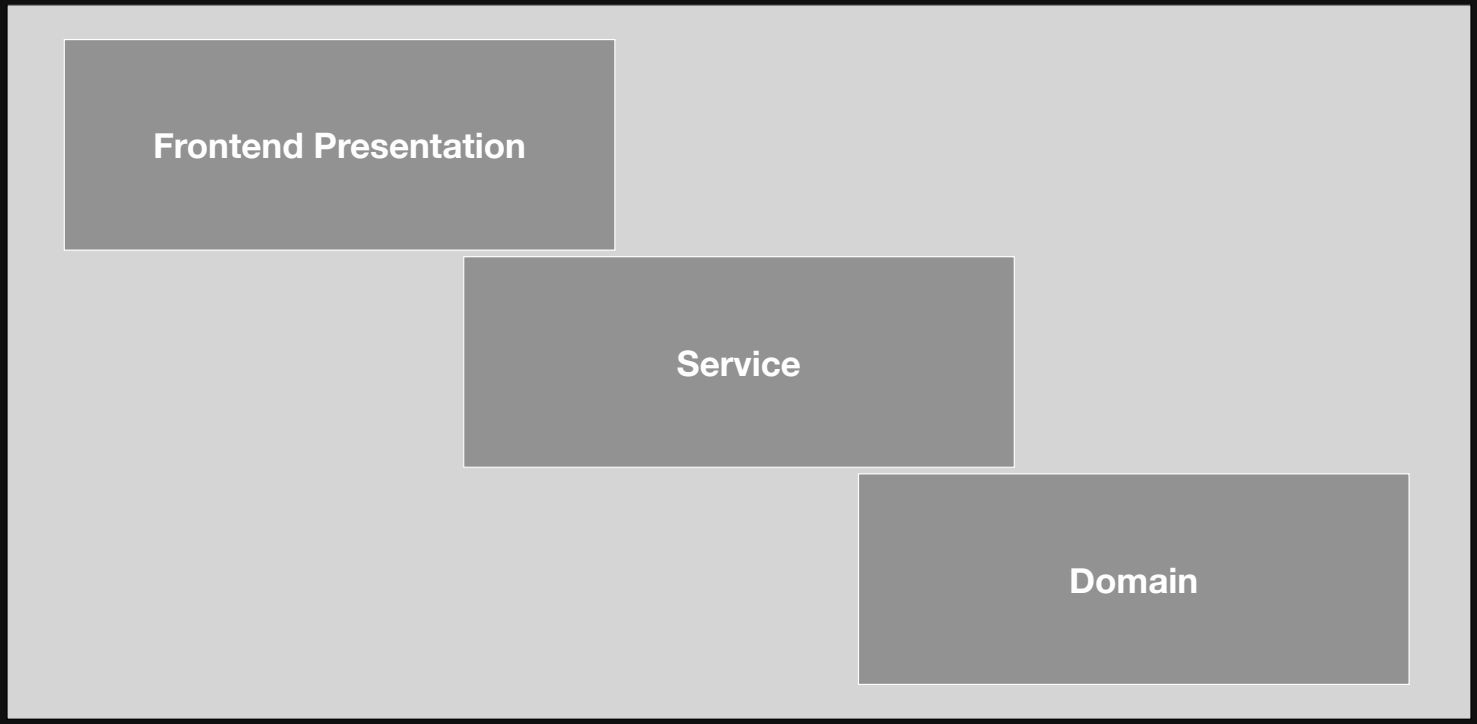
SRP – Exemplo 2



SRP - Exemplo 3

- **Exemplo teórico:** Imagine uma classe ***GetPurchaseOrderService*** que tem a função de buscar dados de um Pedido de compra.
  - Contexto: A aplicação já está funcionando! Recebemos a solicitação de acrescentarmos o nome do cliente para ser exibido no frontend.
- **Violação:** Implícita! Dificuldade de manutenção e alto grau de acomplamento

SRP – Exemplo 3



SRP - Benefícios

- **Manutenção:** Facilidade em atualizar ou modificar funcionalidades específicas.
- **Testabilidade:** Aumento na facilidade de testar cada responsabilidade de forma isolada.

OCP

Open/Closed Principle

Princípio Aberto-Fechado

OCP

- **Conceito:** As entidades de software (classes, módulos, funções) devem estar abertas para extensão, mas fechadas para modificação.

OCP - Exemplo 1

- **Exemplo teórico:** Imagine um sistema RH que calcula o salário de funcionários. Temos as seguintes classes:
  - **Employee**: Representa um funcionário
  - **EmployeeSalaryCalculator**: Faz o cálculo de salário baseado no tipo de funcionário
- **Violação:** Explícita! A cada novo tipo de remuneração, precisamos modificar o código da calculadora de salário

OCP - Exemplo 2

- **Exemplo teórico:** Imagine a geração de relatório de custos por departamento. O relatório pode ser impresso ou gerado em arquivo CSV (Comma Separated Values).
- **Violação:** Explícita! A cada novo tipo de saída, precisamos modificar o código da geração do relatório

OCP - Exemplo 3

- **Exemplo teórico:** Imagine um sistema que possui notificações e o usuário pode decidir em quais canais deseja receber a notificação.
- **Violação:** Explícita! A cada novo canal de notificação, precisamos modificar o código.

OCP - Benefícios

- **Extensibilidade:** Permite adicionar novas funcionalidades de forma rápida e segura.
- **Redução de Bugs:** Minimize o risco de introduzir erros ao não modificar o código existente.

LSP

Liskov (Barbara) Substitution Principle

Princípio de Substituição de Liskov

LSP

- **Conceito:** Se **S** é um subtipo de **T**, então objetos do tipo **T** podem ser substituídos por objetos do tipo **S** sem alterar as propriedades desejáveis do programa.

LSP

- **Conceito:** Objetos de uma superclasse devem poder ser substituídos por objetos de uma subclasse sem alterar o comportamento esperado do programa.

LSP - Exemplo 1

- **Exemplo clássico:** Imagine um software de desenho (Paint). Temos as seguintes classes:
  - **Rectangle**: Representa um retângulo
  - **Square**: Herda da classe Rectangle
- **Violação:** Explícita! A subclasse não respeita o comportamento definido pela superclasse

LSP - Exemplo 2

- **Exemplo:** Imagine um software que lide com diversos tipos de documentos. Temos as seguintes classes:
  - **MyDocument**: Representação abstrata de um documento
  - **Contract**: Herda da classe MyDocument
  - **Report**: Herda da classe MyDocument
- **Violação:** Explícita! A subclasse não respeita o comportamento definido pela superclasse

LSP - Exemplo 3

- **Exemplo:** Nossa aplicação precisa fazer requisições HTTP usando diferentes libs. Temos as seguintes classes:
  - **FetchHttpClient**: Client de lib open source
  - **CustomHttpClient**: Construção própria
- **Violação:** Explícita! A subclasse não respeita o comportamento definido pela interface

Identificar violação LSP

1. **Testes de substituição:**
   - Verifique se objetos da subclasse podem ser usados no lugar da superclasse sem alterar o comportamento esperado.
2. **Verifique sobrescritas de métodos:**
   - Se uma subclasse sobrescreve métodos da superclasse de forma que altera o comportamento esperado, isso pode indicar uma violação do LSP.
3. **Analise pré-condições e pós-condições:**
   - As pré-condições (requisitos para executar um método) não devem ser mais restritivas na subclasse.
   - As pós-condições (resultados garantidos após a execução) não devem ser mais fracas na subclasse.

Identificar violação LSP

4. **Verifique invariantes de classe:**
    - As propriedades que sempre devem ser verdadeiras para uma classe (invariantes), não devem ser quebradas pela subclasse.
5. **Use ferramentas de análise estática:**
    - Ferramentas como linters e analisadores de código podem ajudar a identificar possíveis violações do LSP.

LSP - Benefícios

- **Consistência:** Garante que a substituição de classes não altera o comportamento do sistema.
- **Reutilização:** Facilita o uso de componentes em diferentes contextos sem ajustes adicionais.