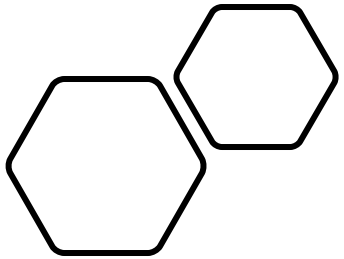
Arquitetura de Software

Prof. Bernardo Copstein

Os princípios SOLID



Leituras recomendadas:

- Martin, Robert C.. Clean Architecture (Robert C. Martin Series) . Pearson Education.

Princípios SOLID

ORIGENS

- Em programação orientada a objetos, SOLID é um acrônimo para 5 princípios de projeto de software que tem por objetivo tonar o software mais fácil de entender e manter. Correspondem a um subconjunto dos princípios promovidos por Robert C. Martin, respeitado autor na área de Engenharia de Software.
- Uma vez que se aplicam a qualquer projeto orientado a objetos, os princípios SOLID formam a base da filosofia de metodologias tais como Desenvolvimento ágil ou Desenvolvimento de software adaptativo.
- Os princípios SOLID foram introduzidos por Robert Martin no artigo “Design Principles and Design Patterns” publicado em 2000. O Acrônimo SOLID foi introduzido mais tarde por Michael Feathers.

SOLID: Princípios de Projeto

Objetivos:

Criar software que tolera modificações

Criar software fácil de entender (do ponto de vista de código)

Criar software cujos componentes possam ser reusados em outros sistemas

SOLID:

SRP: The Single Responsibility Principle

OCP: The Open-Closed Principle

LSP: The Liskov Substitution Principle

ISP: The Interface Segregation Principle

DIP: The Dependency Inversion Principle

SOLID x CAMADAS

- Os princípios SOLID nem sempre são compatíveis com os princípios de uma arquitetura em camadas clássica.
- O principal princípio da arquitetura em camadas que é quebrado é o princípio que diz que as camadas superiores só devem depender das camadas inferiores.
- Durante os exercícios de apresentação dos princípios não se preocupe em “quebrar” estas regras.
- A Arquitetura Limpa (CLEAN Architecture) proposta pelo próprio Robert Martin que será vista na sequencia busca resolver estes conflitos.

Inicialmente serão analisados 3 princípios

SRP – Single Responsibility Principle

ISP – Interface Segregation Principle

DIP – Dependency Inversion Principle

SRP: Single Responsibility Principle

Um subprograma deve fazer alguma coisa e apenas uma coisa

Um módulo deve ter uma razão para mudar, mas apenas uma razão para mudar

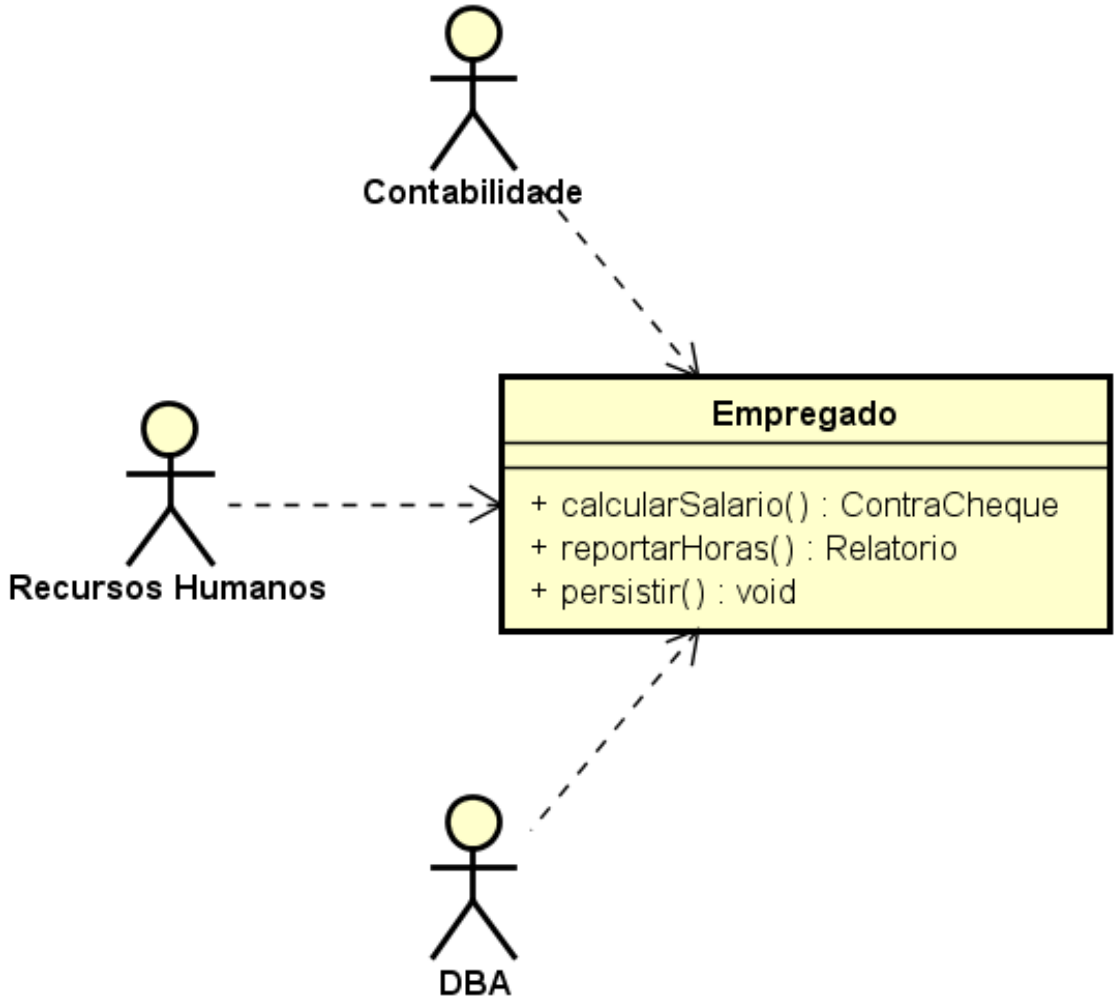
Softwares são alterados em função de usuários ou stakeholders

"Razão para mudar" = atores (usuários, stakeholders)

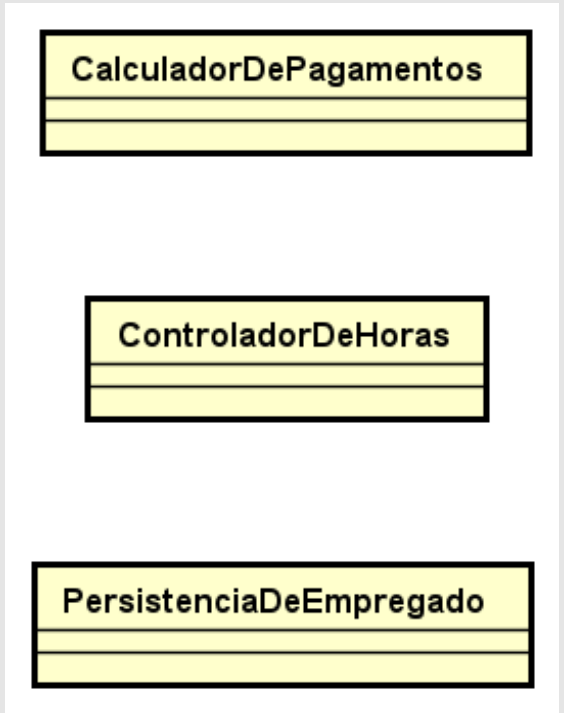
Reescrevendo: **"Um módulo deve ser responsável por um ator e por um ator apenas"**

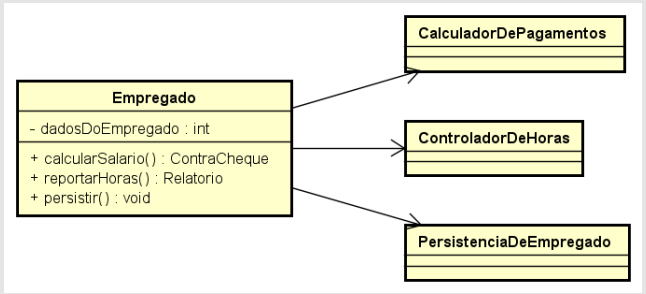
SRP: contra exemplo

A classe não respeita o SRP porque cada método atende um ator diferente



SRP: solução



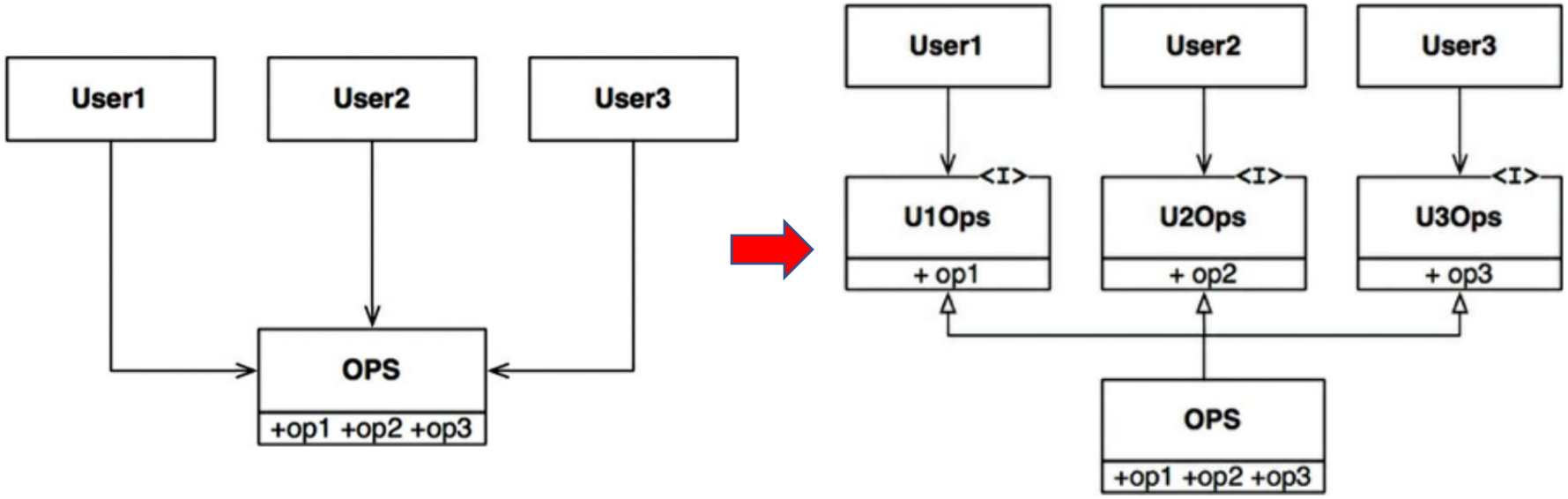


- **A solução é quebrar em 3 classes de maneira que cada uma atenda apenas um ator**
- **A classe “Empregado”, neste caso pode atuar como uma “fachada”**

ISP: Interface Segregation Principle

“Cada módulo deve depender apenas das operações de que necessita”

ISP: exemplo



Os usuários sofrem as consequências das alterações em suas operações e nas dos outros usuários também.

DIP: The Dependency Inversion Principle

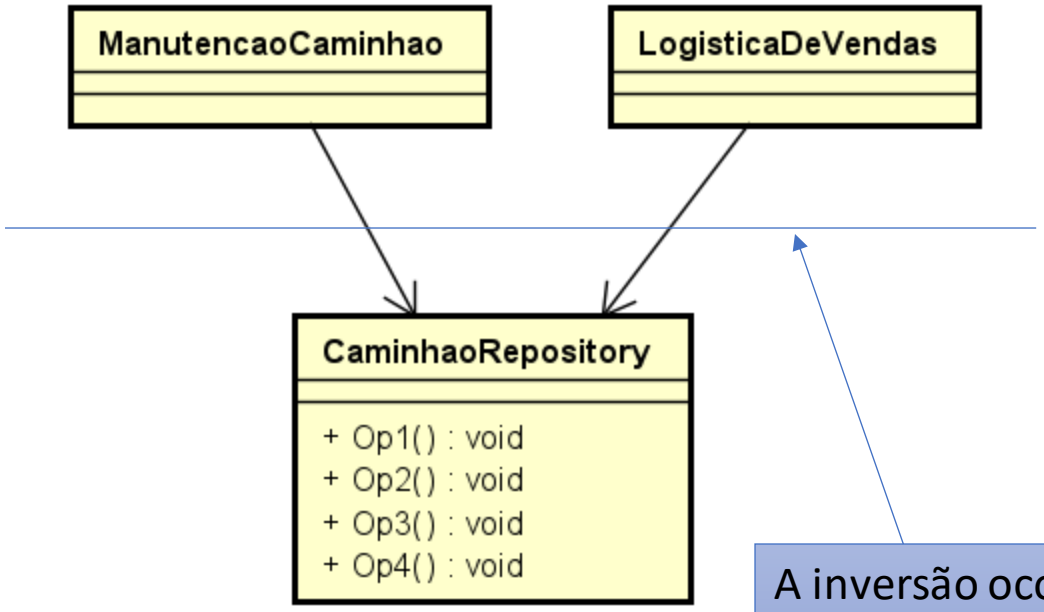
Os sistemas mais flexíveis são aqueles que dependem de abstrações e não de “concretudes”

Orientações:

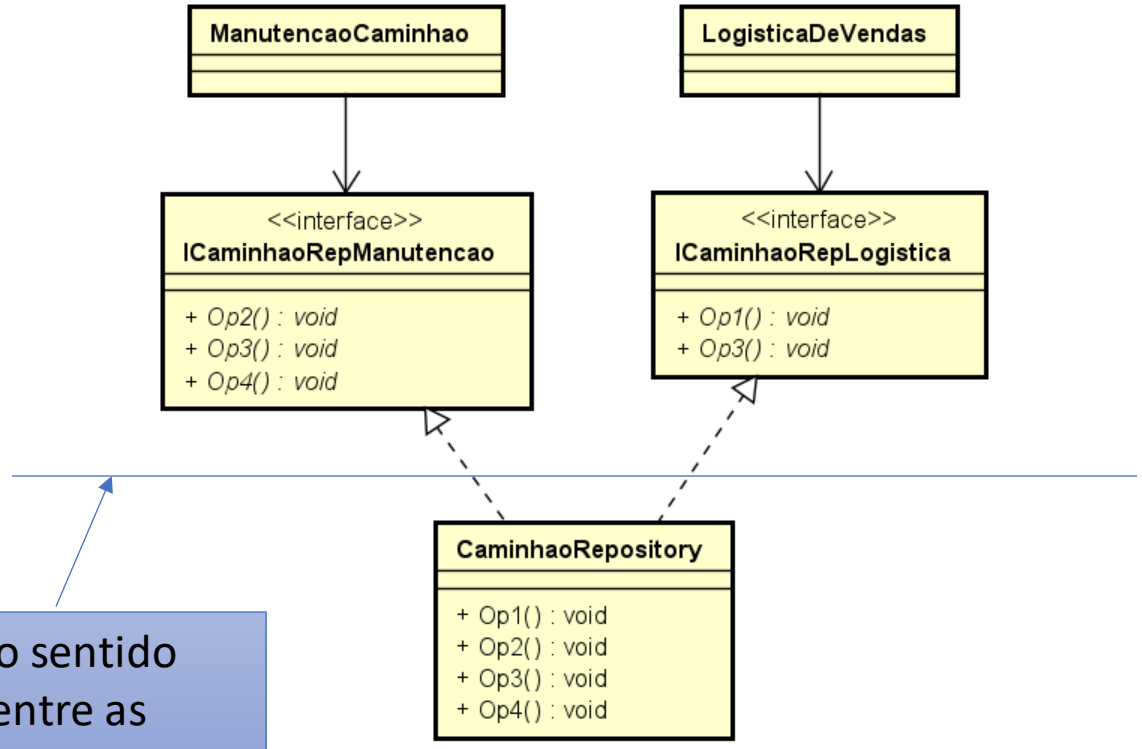
Nunca referenciar uma classe concreta não estável; use interfaces	Só referencie classes concretas realmente estáveis (no contexto)	Nunca derive de classes concretas não estáveis	Não sobrecarregue funções concretas

DIP: exemplo 1

"LogisticaDeVendas" e "ManutencaoCaminhao" dependem de "CaminhaoRepository". Esta, porém, não é uma classe estável porque a tecnologia pode variar.



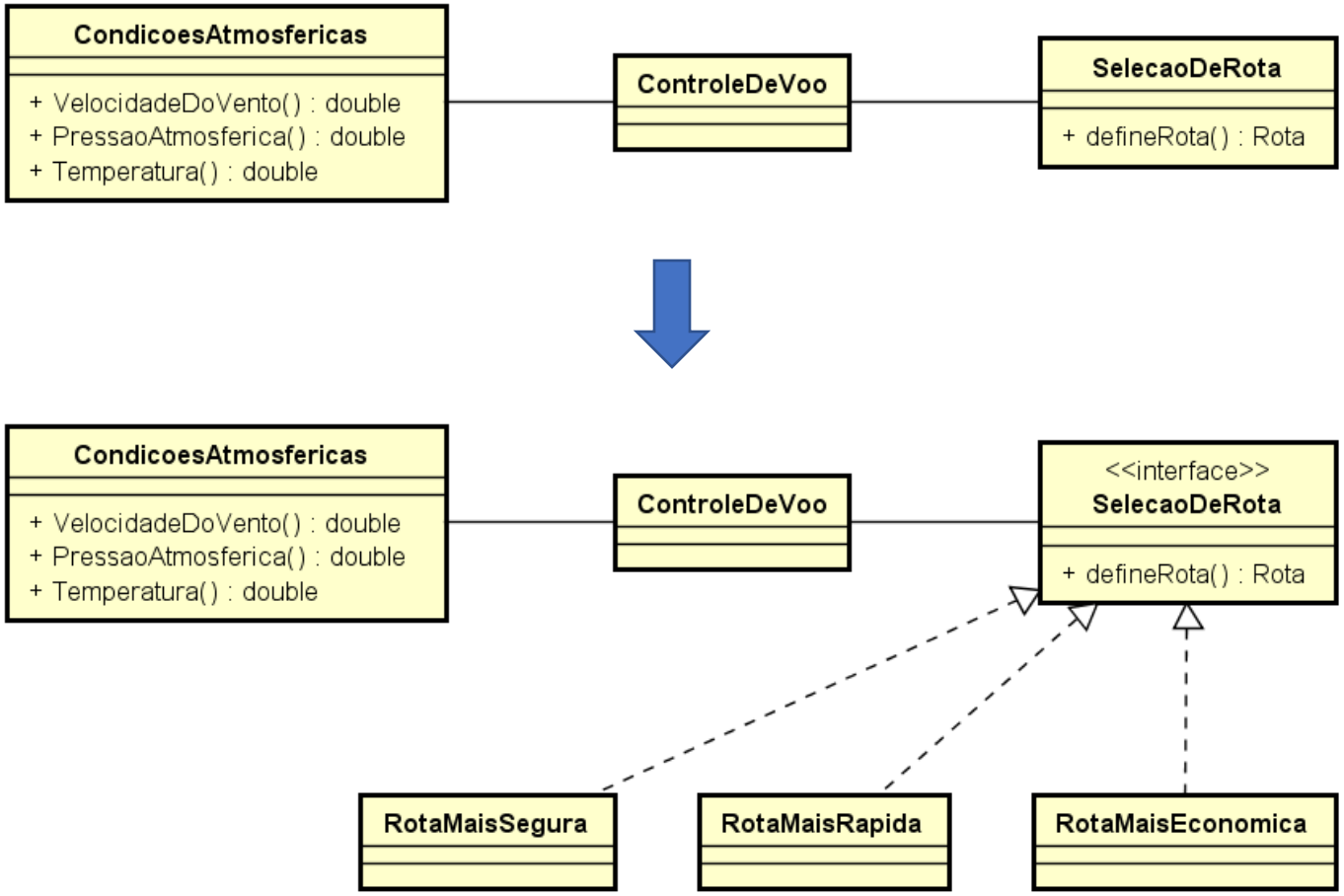
Note que agora "LogisticaDeVendas" e "manutençaoCaminhao" não dependem mais de "CaminhaoRepository" e sim de interfaces específicas que atendem as suas necessidades (ISP). "CaminhaoRepository" agora implementa as interfaces específicas de maneira que o sentido da dependência foi invertido (DIP)



A inversão ocorre no sentido das setas no limite entre as camadas

DIP: exemplo 2

- A probabilidade da forma de se coletar as condições atmosféricas mudar é pequena: classe estável
- A política de seleção da melhor rota pode variar conforme os objetivos da companhia ao longo do tempo (rapidez, economia etc): classe não estável



Veja a lista de exercícios: corrigindo o sistema de cálculo de frete

Na sequencia discuta suas conclusões contra a solução apresentada pelo professor