AULA 6 – PADRÃO *STRATEGY*

GSI020 – Programação Orientada a Objetos II

Prof. Dr. Murillo G. Carneiro
mgcarneiro@ufu.br

UFU

Material baseado nos slides cedidos pelo Prof. Rafael D. Araújo (FACOM/UFU)

Objetivo da aula

- Entender o funcionamento do padrão de projeto Strategy.

Strategy

- É um padrão de projeto de propósito **comportamental** para aprimorar a comunicação entre **objetos**

Intenção

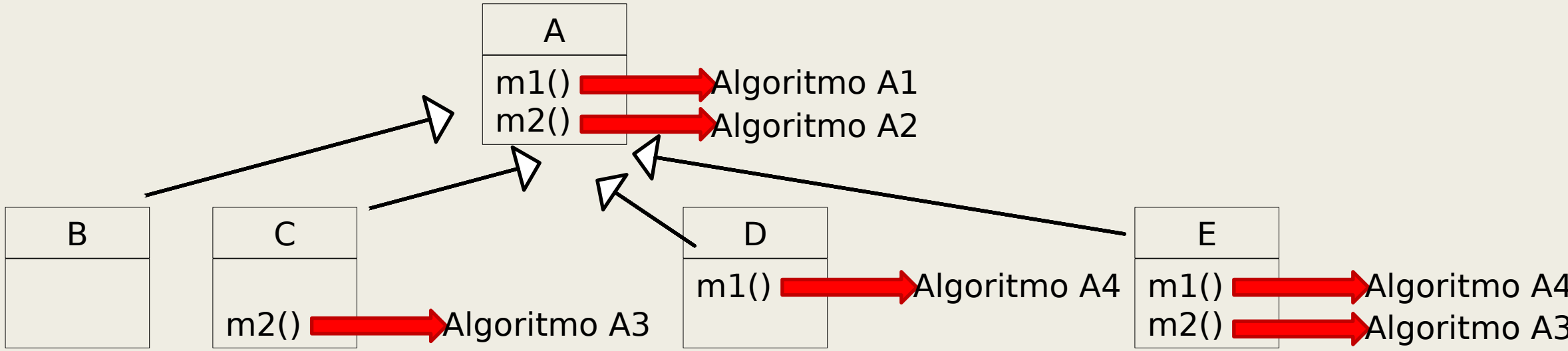
- Definir uma família de algoritmos (comportamentos), encapsular cada uma delas e torná-las intercambiáveis e reutilizáveis
- Permite que o algoritmo varie independentemente dos clientes que o utilizam

Quando usar

- Muitas classes relacionadas se diferem somente no seu comportamento. As estratégias fornecem uma maneiras de configurar uma classe com um dos possíveis comportamentos
- Um determinado método possui diferentes algoritmos
- É útil quando se tem **operações comuns a uma série de sub-classes**, mas não é possível o uso de herança de forma eficiente

Exemplo

- Considere a seguinte situação: *As classes B, C, D e E herdam a classe A*



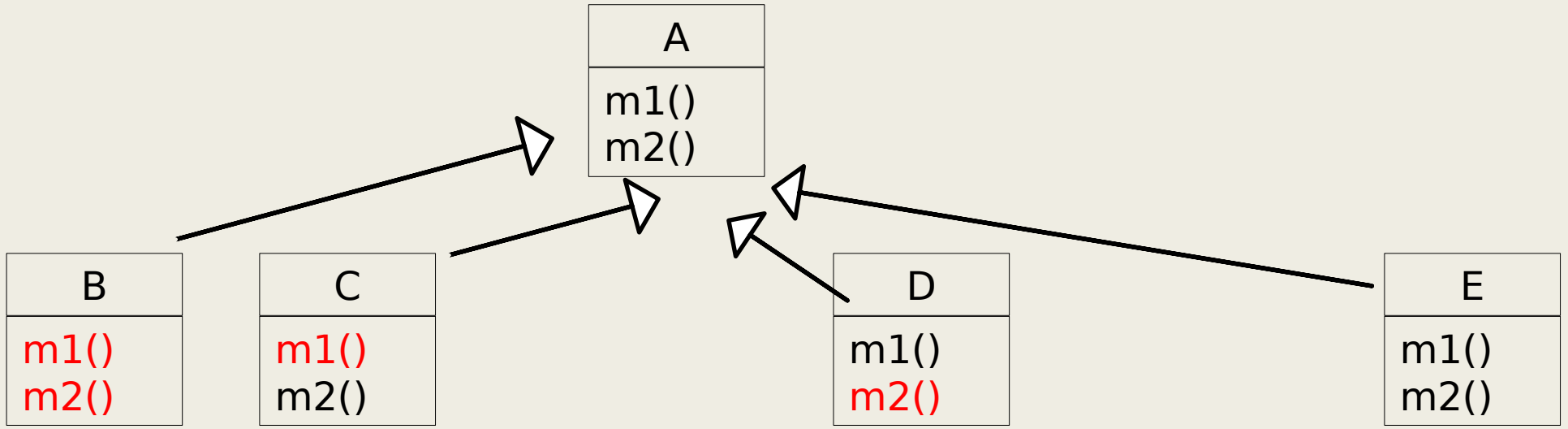
Neste caso temos uma clara **replicação** de implementação dos algoritmos A3 e A4 nas subclasses 'C', 'D' e 'E'.

E se tirarmos a implementação de A1 e A2 da classe A?

Exemplo

Neste caso, **somente herança não resolve**, e, então, devemos utilizar **composição.**

- Considere a seguinte situação: *A se tornou uma interface*



Além da redundância que já existia, passamos a ter **redundância** de A1 e A2 nas classes 'B', 'C' e ‘D’.

Vamos então refatorar este projeto aplicando as orientações do padrão *strategy*

Orientações:

1 - Programe sempre para interfaces;

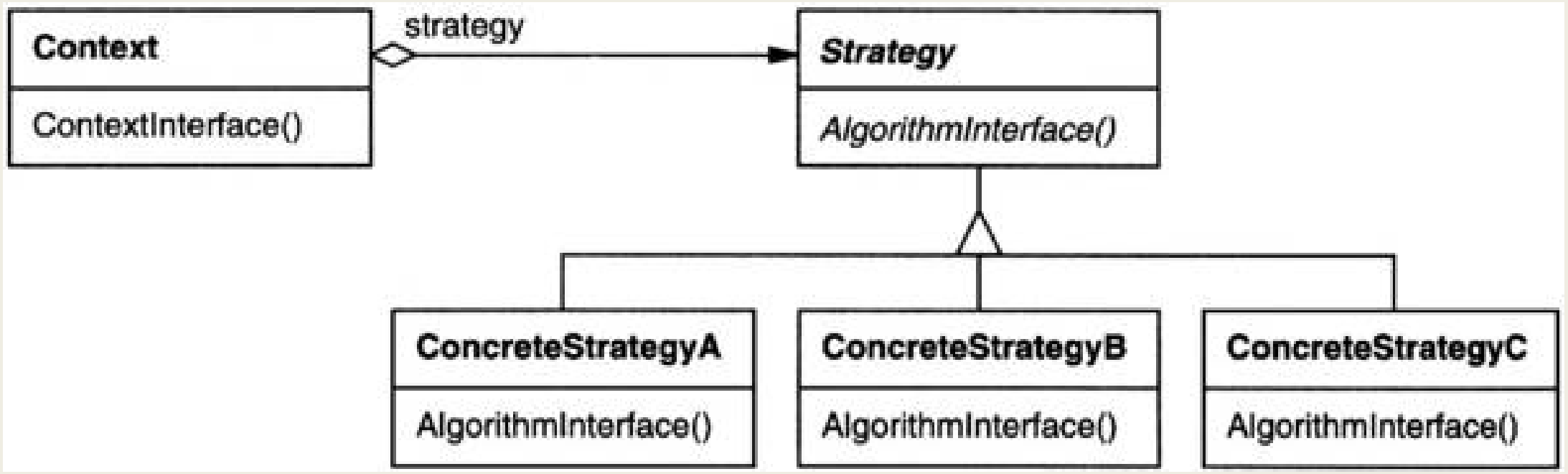
2- Dê preferência à composição ao invés de herança.

- Desta forma consegue-se o reuso e intercâmbio de comportamentos entre diversas classes, facilitando a expansão e manutenção.

Elementos participantes

- **Strategy**, que define a interface comum para todos os comportamentos suportados
- **ConcreteStrategy**, que implementa o comportamento definido pela interface *Strategy*
- **Context**, que tem seu "comportamento" ou parte dele definido pelo algoritmo implementado por uma *Strategy.* Mantém uma referência para um objeto *Strategy* específico

Estrutura


Context
ContextInterface()
strategy
Strategy
AlgorithmInterface()
ConcreteStrategyA
AlgorithmInterface()
ConcreteStrategyB
AlgorithmInterface()
ConcreteStrategyC
AlgorithmInterface()

Benefícios

- Alto nível de reuso
- Facilidade de expansão sem modificar o que está pronto (aberto para extensão, fechado para modificação)
- Alto nível de flexibilidade

Desvantagens

- Clientes devem conhecer as classes *Strategy*
  - *Se o cliente não compreender como essas classes funcionam, não poderá escolher o melhor comportamento*
- Parâmetros da interface Strategy não utilizados
  - *Cada ConcreteStrategy deve inicializar todas as informações da interface*
- Aumento do número de objetos

Problema prático

- Preciso implementar três tipos de cargos: Atendente, Vendedor e Gerente.
- Todos eles recebem comissão das vendas, porém o Atendente recebe 1%, o Vendedor recebe 2% e o Gerente recebe 3%.
- Implemente o cenário acima utilizando o padrão Strategy.
  - *O que seriam as classes Strategy, ConcreteStrategy e Context?*

Problema prático

1) Crie uma aplicação (main) que possui 3 funcionários, um de cada tipo.
2) Defina um valor de vendas igual para todos (500,00).
3) Imprima cada elemento da lista com o valor da venda e comissão.

Benefícios no cenário implementado

- Alto nível de reuso
  - *4) Crie um novo funcionário do tipo Gerente na aplicação.*
- Facilidade de expansão sem modificar o que está pronto
  - *5) Crie um novo tipo de funcionário (Diretor) cuja comissão é de 4%*
  - *6) Crie um objeto desse novo tipo na aplicação*

Benefícios no cenário implementado

- Alto nível de Flexibilidade
  - *7) Faça o objeto criado do tipo Vendedor receber comissão de Gerente na aplicação*

Referências

- GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1995. Capítulo 5.