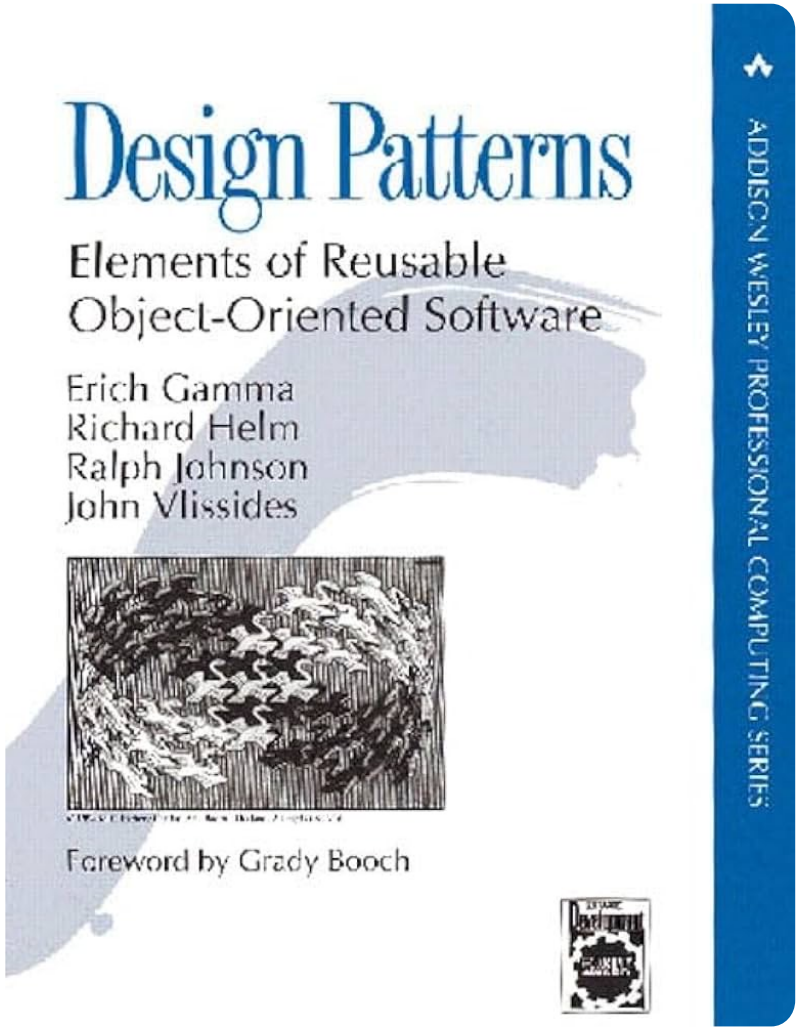
ARQUITETURA DE SOFTWARE

João Choma Neto
joao.choma@unicesumar.edu.br
https://github.com/JoaoChoma/arquitetura-software
Unicesumar – Maringá

PADRÃO DE PROJETO



CATÁLOGO

- Referência ao livro "Design Patterns: Elements of Reusable Object-Oriented Software", escrito por Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides

CATÁLOGO

- Este livro, muitas vezes abreviado como o livro "GoF", foi publicado em 1994 e é considerado uma referência fundamental no campo da engenharia de software.

CATÁLOGO

- Os padrões de projeto são comumente agrupados em três categorias principais:
  - Criacionais
  - Estruturais
  - Comportamentais

PADRÕES CRIACIONAIS

- Os padrões criacionais estão relacionados à criação de objetos de maneira adequada para a situação em que se encontram
- Mecanismos de criação de objetos, tentando criar objetos de uma maneira que seja adequada às circunstâncias

PADRÕES ESTRUTURAIS

- Os padrões estruturais se preocupam com a composição de classes e objetos
- Como as classes e objetos são compostos para formar estruturas maiores
- Soluções fornecem maneiras de simplificar as estruturas

PADRÕES COMPORTAMENTAIS

- Os padrões comportamentais estão preocupados com a comunicação entre objetos
- Como os objetos colaboram entre si para alcançar funcionalidades específicas

SINGLETON

- O padrão Singleton é usado quando você precisa garantir que uma classe tenha apenas uma instância e fornece um ponto global de acesso a essa instância

JÁ VIMOS


Estratégia... Strategy... Stratégie
NASCIMENTO

STRATEGY

- STRATEGY é um padrão comportamental que permite definir:
  - uma família de algoritmos,
  - encapsular cada um deles como um objeto e
  - torná-los intercambiáveis.

INTERCAMBIÁVEL

- Um objeto intercambiável, no contexto de padrões de projeto como o Strategy, refere-se à capacidade de substituir um objeto por outro do mesmo tipo sem que o código cliente (que usa o objeto) precise ser alterado
- Essa substituição pode ocorrer em tempo de execução

OBJETIVO DO PADRÃO STRATEGY

- Permitir a seleção de um algoritmo em tempo de execução, dependendo do contexto em que é necessário.
- Encapsular comportamento que pode variar e separá-lo do código que permanece inalterado.

FUNCIONAMENTO

- Imagine que você está construindo um aplicativo de mapas e precisa de diferentes algoritmos de roteamento: um para **pedestres**, outro para **automóveis** e um terceiro para **bicicletas**

FUNCIONAMENTO

- Cada algoritmo pode ser encapsulado dentro de sua própria classe que implementa uma interface RoteamentoStrategy
- O contexto, que neste caso poderia ser o próprio aplicativo de mapas, mantém uma referência a um objeto RoteamentoStrategy que é utilizado para calcular o roteamento

FUNCIONAMENTO

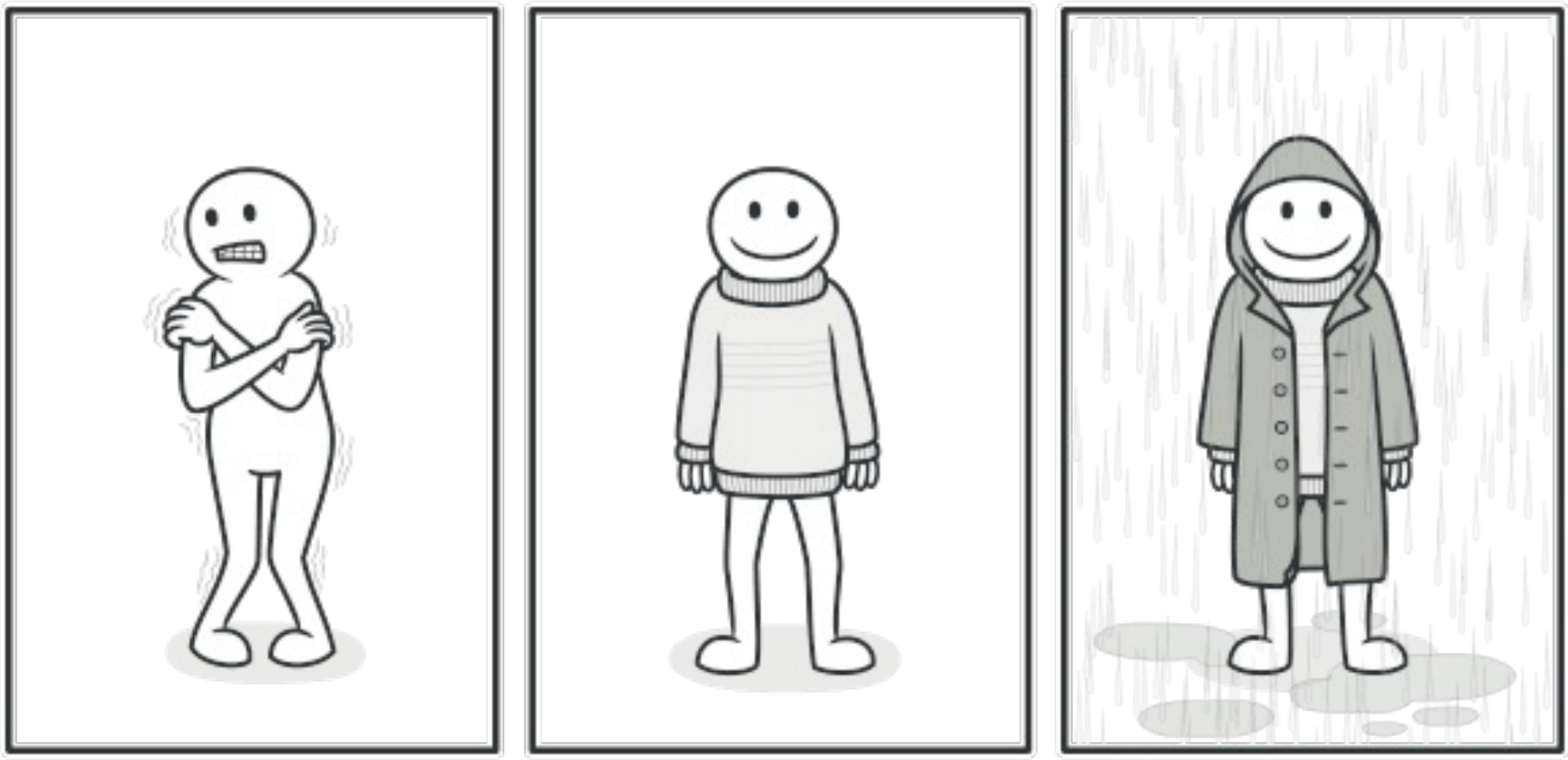
- O usuário decide qual estratégia de roteamento deseja (por exemplo, selecionando um modo de transporte)
- O aplicativo ajusta a estratégia de roteamento conforme necessário durante a execução

STRATEGY - VANTAGENS

- **Flexibilidade:** Facilmente intercambiável em tempo de execução
- **Encapsulamento:** Separa a implementação de um algoritmo das classes que o utilizam
- **Extensibilidade:** Novos comportamentos podem ser adicionados sem alterar o contexto

STRATEGY - DESVANTAGENS

- **Complexidade:** Pode aumentar a complexidade do código devido à criação de múltiplas classes.
- **Quantidade de Objetos:** Pode aumentar o número de objetos no sistema.



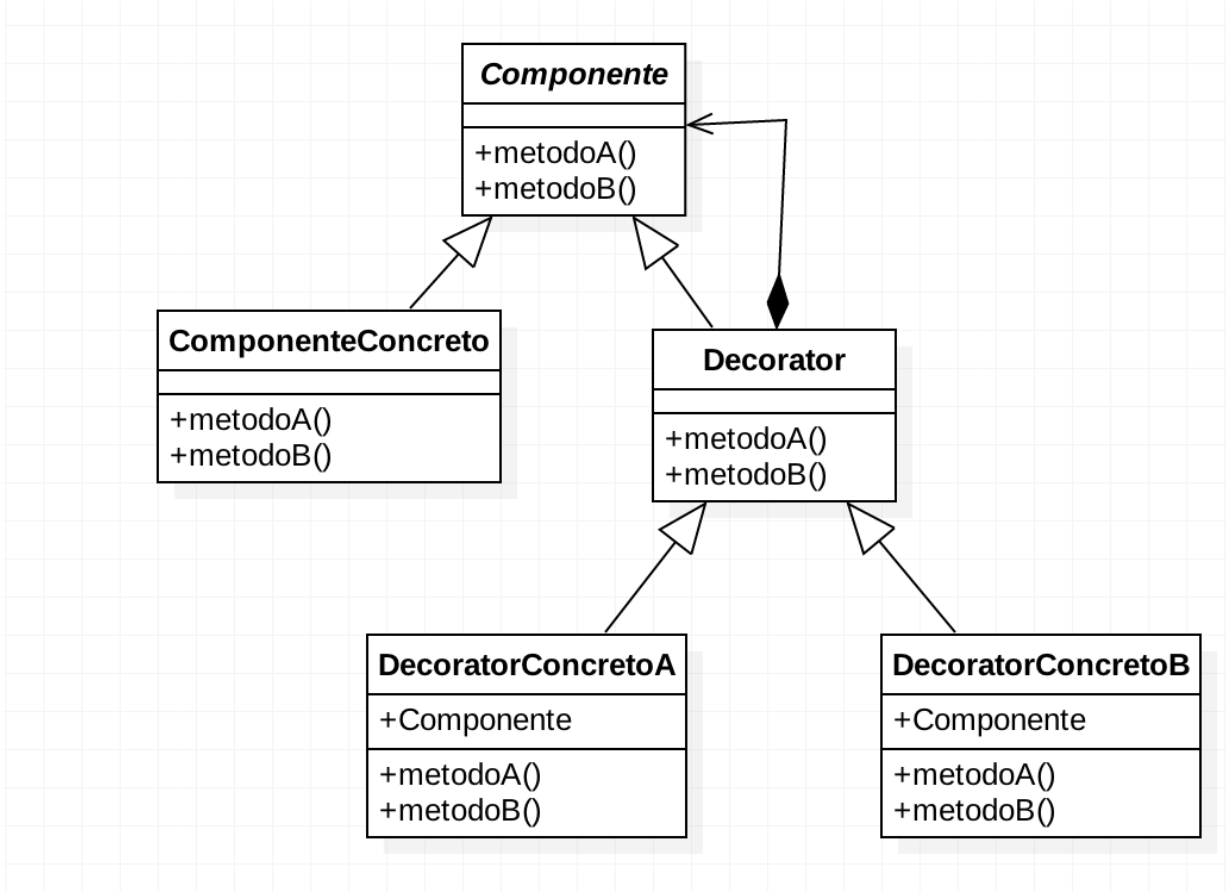
PADRÃO DECORATOR

- O padrão de projeto Decorator é um padrão de projeto estrutural
- Permite adicionar funcionalidades extras dinamicamente a objetos existentes sem modificar sua estrutura básica

PADRÃO DECORATOR

- Isso é alcançado através da criação de uma série de "decoradores" que envolvem o objeto original e adicionam novos comportamentos a ele
- Essencialmente, o padrão Decorator promove a composição sobre a herança

DECORATOR


Componente
+metodoA()
+metodoB()
ComponenteConcreto
+metodoA()
+metodoB()
Decorator
+metodoA()
+metodoB()
DecoratorConcretoA
+Componente
+metodoA()
+metodoB()
DecoratorConcretoB
+Componente
+metodoA()
+metodoB()

DECORATOR

- O padrão é útil quando você tem várias combinações possíveis de funcionalidades e deseja evitar a explosão de subclasses para cada combinação
- O padrão permite adicionar ou remover funcionalidades de forma flexível e em tempo de execução

DECORATOR

- O padrão Decorator é usado quando você precisa adicionar funcionalidades extras de forma dinâmica a objetos existentes sem modificar sua estrutura.

ESTRUTURA

1. **Componente Interface**: Define a interface comum para os objetos que serão decorados. Pode ser uma interface ou uma classe abstrata que representa o objeto básico
2. **Componente Concreto**: Implementa a interface do componente e define o objeto básico ao qual funcionalidades adicionais podem ser adicionadas

ESTRUTURA

3. **Decorator**: É uma classe abstrata ou interface que implementa a mesma interface do componente base e contém uma referência para o componente base. Este é o ponto central do padrão, pois permite adicionar novas funcionalidades ao componente base.
4. **Decoradores Concretos**: São classes que estendem o decorador abstrato e adicionam funcionalidades adicionais ao objeto base. Eles podem adicionar ou modificar o comportamento do objeto base conforme necessário.

ONDE USAR

- **Interfaces Gráficas de Usuário (GUIs)**: Em uma GUI, você pode usar o padrão Decorator para adicionar funcionalidades como bordas coloridas, sombras, ou efeitos especiais a elementos gráficos, como botões ou janelas, sem alterar sua implementação original
- **Processamento de Dados**: Em operações de processamento de dados, como leitura e escrita de arquivos, você pode usar Decorators para adicionar funcionalidades como criptografia, validação de dados

ONDE USAR

- **Geração de Relatórios**: Em sistemas de geração de relatórios, você pode usar Decorators para adicionar funcionalidades como formatação de texto, adição de cabeçalhos ou rodapés, ou inclusão de gráficos aos relatórios gerados.
- **Manipulação de Imagens ou Vídeos**: Em aplicativos de processamento de mídia, como editores de imagens ou reprodutores de vídeo, você pode usar Decorators para adicionar funcionalidades como filtros, efeitos especiais, ou ajustes de cor

ARQUITETURA DE SOFTWARE

João Choma Neto
joao.choma@unicesumar.edu.br
https://github.com/JoaoChoma/arquitetura-software
Unicesumar – Maringá