DIGITAL BUSINESS ENABLEMENT

#01
DESIGN PATTERNS 01

Felipe Cabrini

CONCEITO

"

Design Patterns ou padrões de projetos são soluções generalistas para problemas recorrentes durante o desenvolvimento de um software.

CONCEITO

"

Design Patterns ou padrões de projetos são soluções generalistas para problemas recorrentes durante o desenvolvimento de um software.

REFERÊNCIA

Erich Gamma
JUnit/Eclipse

Richard Helm

Ralph Johnson

John Vlissides

VANTAGENS AO USAR

1. Reutilização de código:
    1. Evita a duplicação de código e facilita a manutenção.
2. Flexibilidade:
    1. Permite mudanças no sistema com menor impacto em outras partes do código.
3. Padronização:
    1. Oferece uma linguagem comum para discutir e documentar soluções de design.
4. Eficiência:
    1. Reduz o tempo de desenvolvimento ao usar soluções testadas.

CRÍTICAS AOS PADRÕES

CUIDADO

- Design Patterns não são a bala de prata
- Design Patterns são soluções para os problemas, não servem para encontrar problemas
- Se usado corretamente deve melhorar seu código e não torná-lo mais confuso

Complexidade Adicional:

- O uso de design patterns pode adicionar complexidade ao código, especialmente para desenvolvedores menos experientes que podem achar difícil entender a estrutura complexa dos padrões.

CUIDADO

Overengineering:

- Existe o risco de aplicar design patterns quando eles não são realmente necessários, levando a um excesso de complexidade no sistema. É importante avaliar se o uso de um padrão é realmente justificado para o problema em questão.

CUIDADO

3. Rigidez no Design:

- Alguns design patterns podem tornar o código mais rígido e menos adaptável a mudanças futuras, pois eles podem introduzir acoplamento entre classes e componentes.

CUIDADO

Sobrecarga de Abstração:

- O uso excessivo de abstrações em design patterns pode tornar o código mais difícil de entender e depurar, tornando a lógica de negócios menos visível.

OS PADRÕES

PADRÕES
CRIACIONAIS

PADRÕES
ESTRUTURAIS

PADRÕES
COMPORTAMENTAIS

OS PADRÕES

PADRÕES
CRIACIONAIS

PADRÕES
ESTRUTURAIS

PADRÕES
COMPORTAMENTAIS

 **Factory Method**

 **Abstract Factory**

 **Singleton**

 **Builder**

 **Prototype**

OS PADRÕES

PADRÕES CRIACIONAIS

PADRÕES ESTRUTURAIS

PADRÕES COMPORTAMENTAIS

 Adapter

 Decorator

 Bridge

 Facade

 Proxy

 Composite

 Flyweight

OS PADRÕES

PADRÕES CRIACIONAIS

PADRÕES ESTRUTURAIS

PADRÕES COMPORTAMENTAIS


Chain of Responsibility


Observer


State


Template Method


Memento


Command


Iterator


Mediator


Strategy


Visitor

STRATEGY

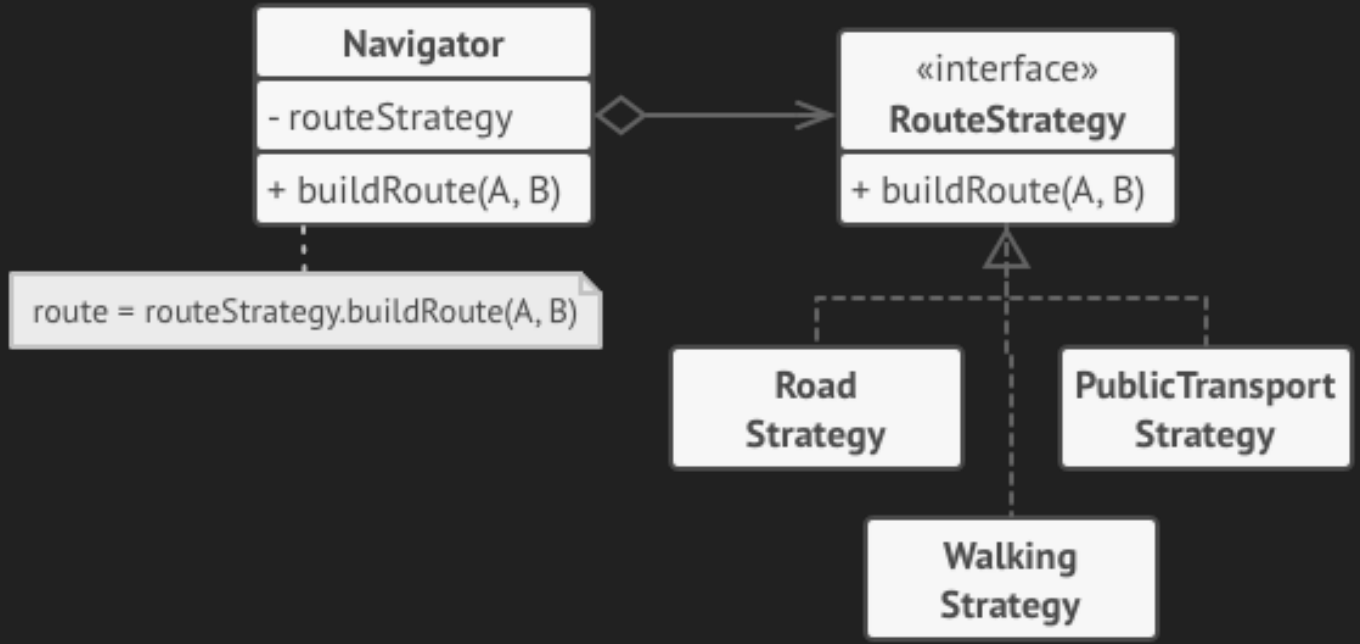
" Padrão de projeto comportamental que permite que você defina uma família de algoritmos, coloque-os em classes separadas, e faça os objetos deles intercambiáveis.

STRATEGY

O Problema

STRATEGY

A Solução



Hands ON!

- Problema:
- Você foi contratado para desenvolver um Sistema de processamento de pagamentos para um E-commerce Famoso. O seu cliente pediu para você inicialmente colocar dois métodos de pagamento Cartão de crédito e Paypal. Mas falou que em breve ele vai querer outros tipos de pagamento.


KEEP
CALM
IT'S
CODE
TIME!!!

Exercício!

- Agora que lançou o aplicativo de processamento de pagamentos, o cliente solicitou dois novos meios de pagamento:
  Pix
- Pagamento via boleto.

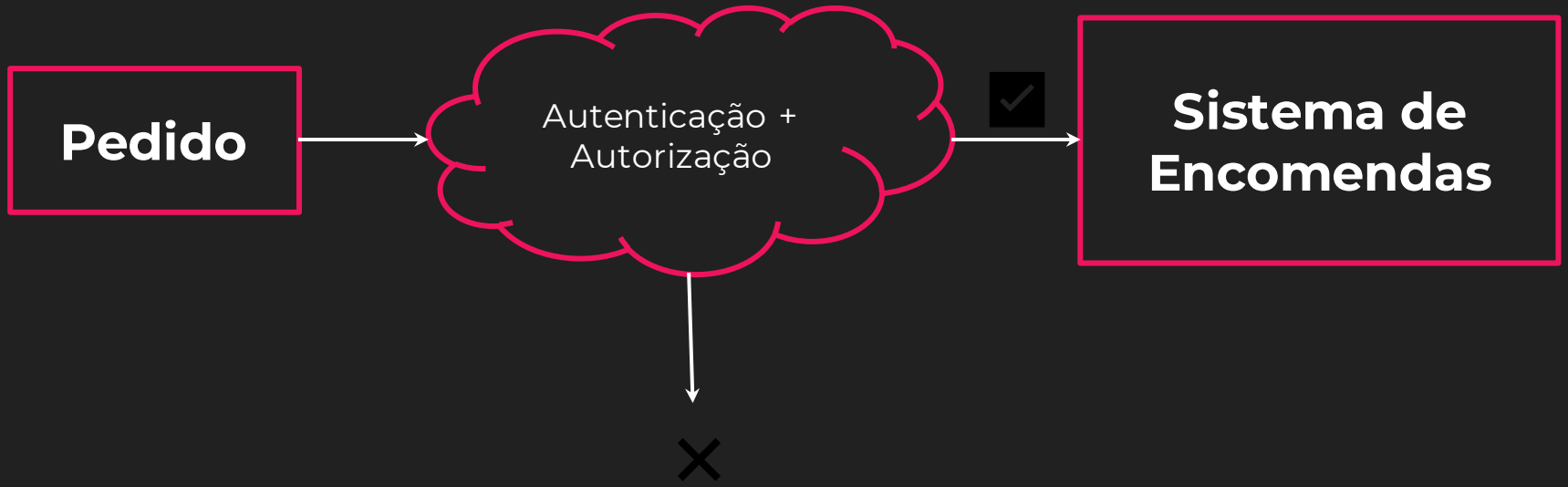
- Neste exemplo, o padrão Strategy é utilizado para permitir a troca dinâmica entre diferentes métodos de pagamento (estratégias) sem alterar o código do contexto (classe PaymentProcessor). Isso torna o sistema mais flexível e extensível, permitindo adicionar novos métodos de pagamento no futuro sem modificar o código existente.

CHAIN OF RESPONSIBILITY

" Padrão de projeto comportamental que permite que você passe pedidos por uma corrente de handlers. Ao receber um pedido, cada handler decide se processa o pedido ou o passa adiante para o próximo handler na corrente.

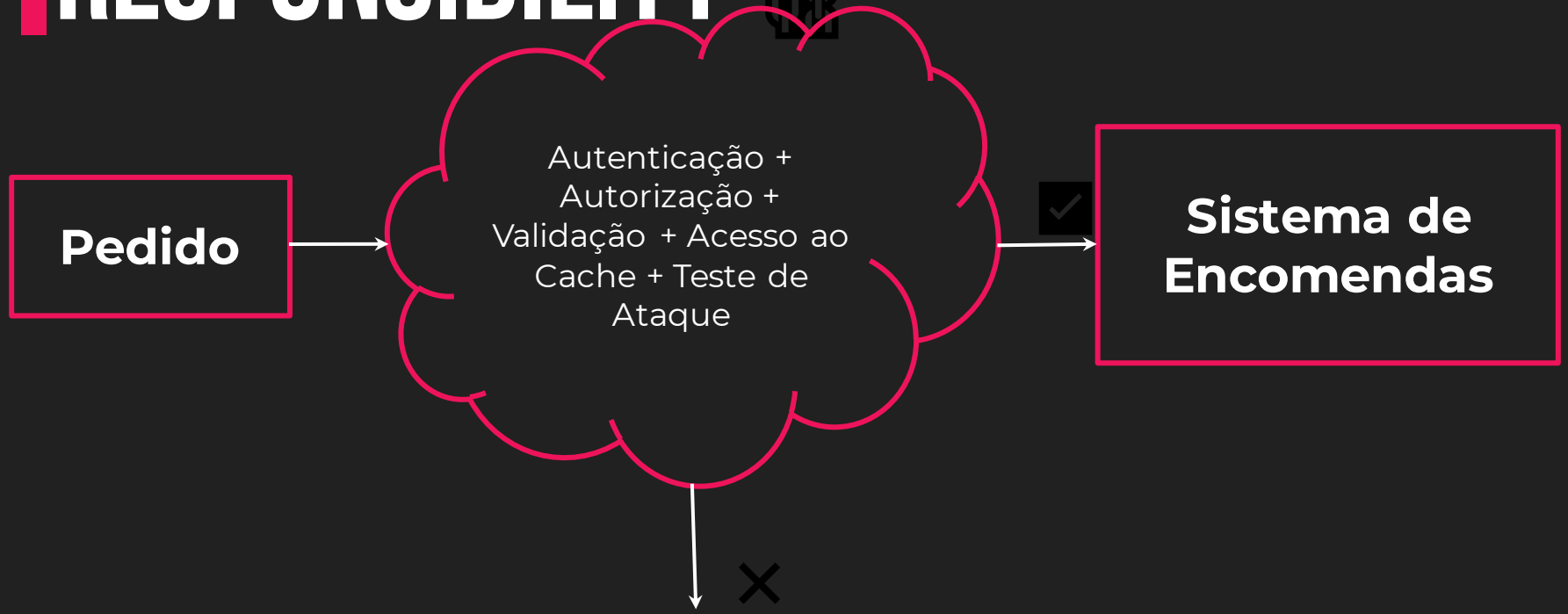
CHAIN OF RESPONSIBILITY

O Problema



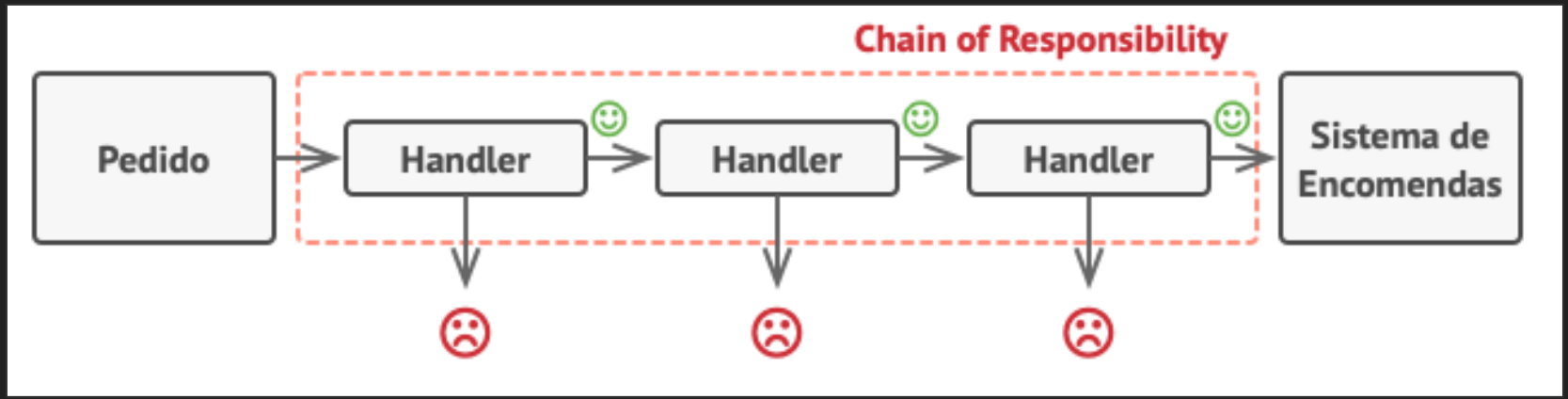
CHAIN OF RESPONSIBILITY

O Problema



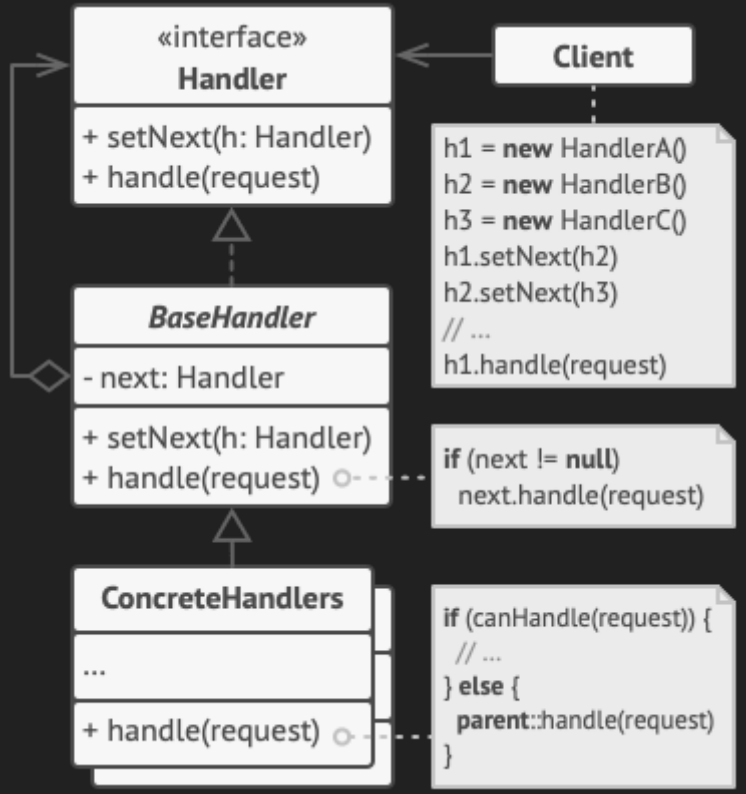
CHAIN OF RESPONSIBILITY

A Solução



CHAIN OF RESPONSIBILITY

A Solução



CHAIN OF RESPONSIBILITY

Problema

- Imagine um sistema de atendimento ao cliente de uma empresa, onde os clientes podem enviar diferentes tipos de solicitações de suporte, como "Reembolso", "Troca de Produto" e "Suporte Técnico". Cada tipo de solicitação requer um nível de atenção e resolução diferente, e queremos implementar um sistema flexível que possa lidar com essas solicitações de forma escalonada.

Exercício!

- O cliente solicitou uma feature nova. Agora ele que que o Sistema lide com cancelamento de serviços. Implemente esse cancelamento na chain of responsibility.

- Neste exemplo, o padrão Chain of Responsibility é usado para criar uma sequência de manipuladores que podem lidar com diferentes tipos de solicitações de suporte. Cada manipulador verifica o tipo da solicitação e decide se pode processá-la ou passá-la para o próximo manipulador na sequência. Isso permite que diferentes níveis de suporte sejam tratados de forma escalonada e flexível, sem a necessidade de acoplamento direto entre os manipuladores e as solicitações.
- O padrão Chain of Responsibility é particularmente útil quando você tem uma hierarquia de processamento ou tratamento de solicitações e deseja evitar acoplamento rígido entre os elementos da hierarquia. Isso torna o sistema mais expansível e fácil de manter.

- Design Patterns são soluções valiosas para problemas de design de software.
- Eles promovem a reutilização de código, a padronização e a flexibilidade.
- Existem três categorias principais de Design Patterns: Criacionais, Estruturais e Comportamentais.
- Isso é uma base para você começar a criar seus slides sobre Design Patterns. Lembre-se de adicionar imagens relevantes, exemplos visuais e qualquer outro conteúdo que torne sua apresentação mais envolvente.

Links:

- https://refactoring.guru/design-patterns/catalog