AULA 5 – Padrões de Projeto

Prof. Igor Justino Rodrigues
08/11/2024

INSTITUTO FEDERAL
Goiano
Campus Ceres

OBJETIVOS DA AULA

- Introduzir os padrões comportamentais Strategy e Observer.
- Demonstrar suas aplicações e como esses padrões ajudam a flexibilizar o código em sistemas baseados em eventos e escolha de algoritmos.

- 08/11/2024

Introdução aos Padrões Comportamentais

Definição: Padrões que lidam com a comunicação entre objetos.

Por que usar?

Redução de acoplamento entre objetos.

Facilitação de extensões e modificações.

Exemplos: Strategy, Observer, State, Template Method.

Padrão Strategy

Propósito: Permitir que algoritmos sejam substituídos em tempo de execução.

Exemplo de Aplicação: Calculadora de frete que usa métodos diferentes de transporte.

Diagrama Básico:

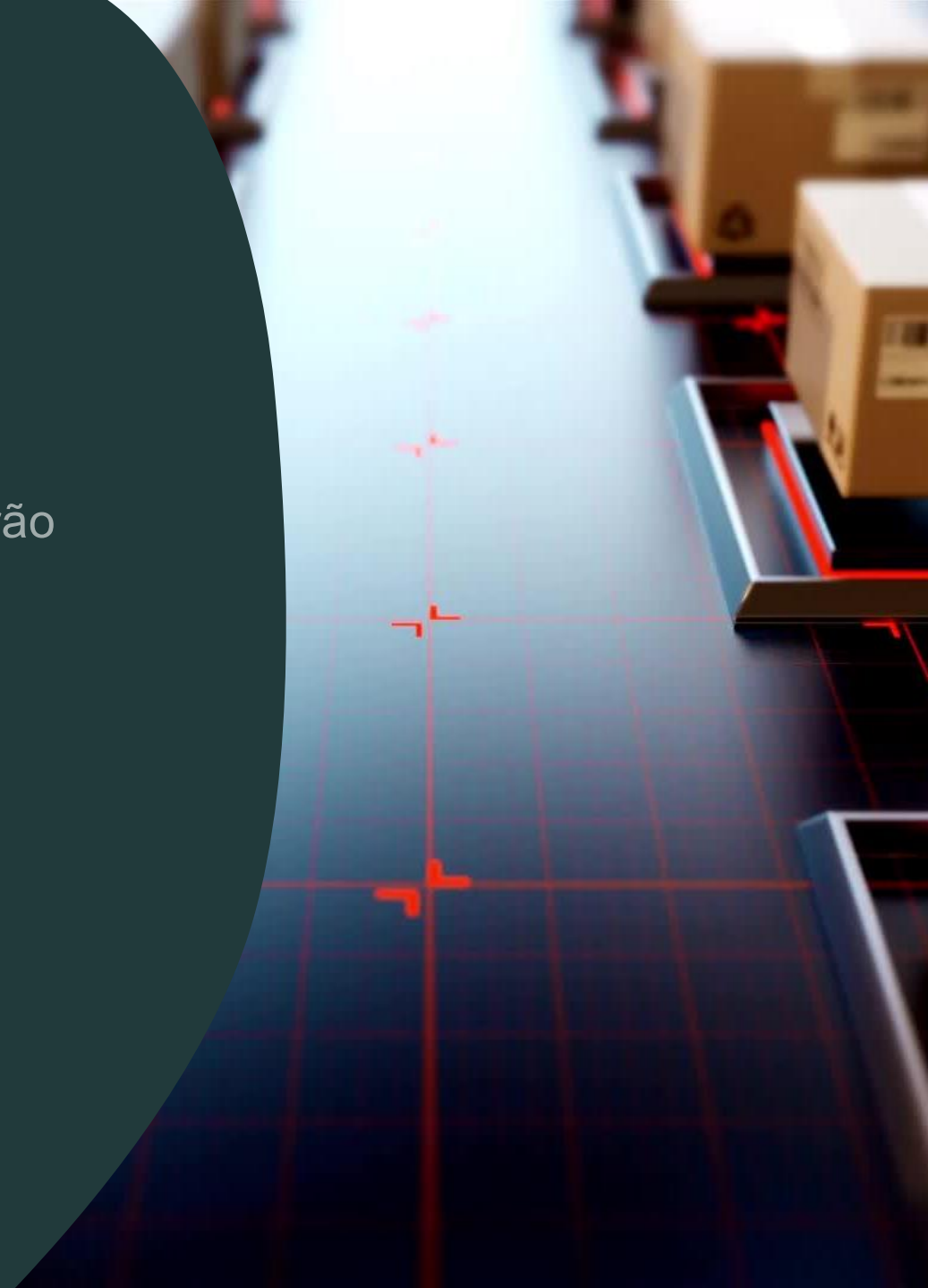
Contexto (Ex: CalculadoraFrete)
Estratégia:

(Ex: FreteTerrestre, FreteAereo).

Prática (Strategy)

Refatore um código de cálculo de frete aplicando o padrão Strategy.

Objetivo: Fortalecer a prática de encapsular comportamentos.



CÓDIGO DE EXEMPLO

```php
<?php

class CalculadoraFrete {
    public function calcularFrete($tipoFrete, $peso) {
        if ($tipoFrete === 'terrestre') {
            return $peso * 0.5;
        } elseif ($tipoFrete === 'aereo') {
            return $peso * 1.5;
        } else {
            throw new Exception("Tipo de frete desconhecido.");
        }
    }
}

// Uso do código
$calculadora = new CalculadoraFrete();
echo "Frete Terrestre: " . $calculadora->calcularFrete('terrestre', 10) . "\n";
echo "Frete Aéreo: " . $calculadora->calcularFrete('aereo', 10) . "\n";

?>
```

```php
<?php

// Interface para estratégia de frete
interface FreteStrategy {
    public function calcular($peso);
}

// Estratégia de frete terrestre
class FreteTerrestre implements FreteStrategy {
    public function calcular($peso) {
        return $peso * 0.5;
    }
}

// Estratégia de frete aéreo
class FreteAereo implements FreteStrategy {
    public function calcular($peso) {
        return $peso * 1.5;
    }
}
```

CÓDIGO COM STRATEGY

```php
// Classe que usa a estratégia de frete
class CalculadoraFrete {
    private $estrategia;

    // Define a estratégia no construtor
    public function __construct(FreteStrategy $estrategia) {
        $this->estrategia = $estrategia;
    }

    public function calcularFrete($peso) {
        return $this->estrategia->calcular($peso);
    }
}

// Uso do código refatorado

// Calculando frete terrestre
$calculadoraTerrestre = new CalculadoraFrete(new FreteTerrestre());
echo "Frete Terrestre: " . $calculadoraTerrestre->calcularFrete(10) . "\n";

// Calculando frete aéreo
$calculadoraAereo = new CalculadoraFrete(new FreteAereo());
echo "Frete Aéreo: " . $calculadoraAereo->calcularFrete(10) . "\n";
```

CÓDIGO COM STRATEGY

Padrão Observer

- **Propósito**: Permitir que objetos "observem" mudanças em outro objeto e reajam automaticamente.
- **Exemplo de Aplicação**: Sistema de notificações em tempo real.
- **Diagrama Básico**:
  - Sujeito (Ex: Evento) -> Observadores (Ex: EmailNotificacao, SMSNotificacao).

Prática (Observer)

01

Implemente um sistema de notificação de eventos usando Observer.

02

Desafio: Adicionar uma notificação nova (Ex: Notificação por Push).

03

Objetivo: Praticar a implementação de um sistema de notificações usando Observer.

Criação da Interface do Observador

```php
<?php


interface Observador {
    public function atualizar($mensagem);
}
```

Implementação dos Observadores Concretos

```php
class EmailNotificacao implements Observador {
    public function atualizar($mensagem) {
        echo "Notificação por Email: $mensagem\n";
    }
}

class SMSNotificacao implements Observador {
    public function atualizar($mensagem) {
        echo "Notificação por SMS: $mensagem\n";
    }
}
```

Criação da Classe Observada (Evento)

```php
class Evento {
    private $observadores = [];

    // Método para adicionar observadores
    public function adicionarObservador(Observador $observador) {
        $this->observadores[] = $observador;
    }

    // Método para notificar todos os observadores
    public function notificar($mensagem) {
        foreach ($this->observadores as $observador) {
            $observador->atualizar($mensagem);
        }
    }
}
```

Uso do Sistema de Notificação

```php
// Instância do evento
$evento = new Evento();

// Adicionando observadores de notificações
$evento->adicionarObservador(new EmailNotificacao());
$evento->adicionarObservador(new SMSNotificacao());

// Disparando um novo evento
$evento->novoEvento("Atualização de sistema disponível.");
```