DESENVOLVIMENTO WEB III

Design Patterns – Padrões Estruturais

Prof. Esp. Érick Henrique Pereira Nicolau

O que são Padrões Estruturais?

- Categoria de Design Patterns focada no relacionamento entre classes e objetos.
- Facilitam a composição de objetos para formar estruturas maiores e mais complexas.
- Permitem criar sistemas flexíveis, reutilizáveis e com baixo acoplamento.

Padrões Estruturais

- Adaptador (Adapter)
- Decorador (Decorator)
- Fachada (Facade)
- Ponte (Bridge)
- Composto (Composite)

Adaptador (Adapter)

- Objetivo: Permite que classes com interfaces incompatíveis trabalhem juntas.
- Como: Converte a interface de uma classe em outra interface esperada pelo cliente.
- Quando usar: Quando você precisa usar uma classe existente que não possui a interface que seu código espera.

```
// Classe Pato (interface esperada)
class Pato {
    grasnar() { /* ... */ }
}

// Classe Cachorro (interface incompatível)
class cachorro {
    latir() { /* ... */ }
}

// AdaptadorCachorro (adapta a interface de Cachorro para a de Pato)
class AdaptadorCachorro {
    constructor(cachorro) {
        this.cachorro = cachorro;
    }

    grasnar() {
        this.cachorro.latir();
    }
}
```

Decorador (Decorator)

- Objetivo: Adicionar responsabilidades a um objeto dinamicamente, sem alterar sua estrutura original.
- Como: Envolve o objeto original em um "decorador" que adiciona novos comportamentos.
- Quando usar: Quando você precisa adicionar funcionalidades a objetos individuais sem afetar outros objetos da mesma classe.

```
// Classe base (componente a ser decorado)
class Cafe {
    getPreco() { /* ... */ }
}

// Decoradores (adicionam funcionalidades)
class LeiteDecorator {
    constructor(cafe) { /* ... */ }
    getPreco() { /* ... */ }
}

class ChantillyDecorator {
    constructor(cafe) { /* ... */ }
    getPreco() { /* ... */ }
}
```

Fachada (Facade)

- Objetivo: Fornece uma interface simplificada para um subsistema complexo.
- Como: Cria uma classe que encapsula a interação com o subsistema, ocultando sua complexidade.
- Quando usar: Quando você precisa interagir com um sistema complexo, mas quer uma interface mais fácil de usar.

```
// Subsistemas complexos
class SistemaDeAudio { /* ... */ }
class SistemaDeVideo { /* ... */ }

// Fachada (interface simplificada)
class HomeTheaterFacade {
    constructor() { /* ... */ }
    assistirFilme(filme) { /* ... */ }
    finalizarFilme() { /* ... */ }
}
```

Ponte (Bridge)

- Objetivo: Desacopla uma abstração de sua implementação, permitindo que ambas variem independentemente.
- Como: Usa uma ponte (interface) para conectar a abstração à implementação.
- Quando usar: Quando você quer evitar uma ligação permanente entre uma abstração e sua implementação.

```
// Abstração
class Forma {
    constructor(desenhador) { /* ... */ }
    desenhar() { /* ... */ }
}

// Implementações
class DesenhoVetorial { /* ... */ }
class DesenhoRaster { /* ... */ }
```

Funcionamento

- A classe Forma representa a abstração de uma forma geométrica. É como um conceito geral de uma forma, sem se preocupar com os detalhes específicos de como ela será desenhada na tela.
- O construtor constructor(desenhador) recebe um objeto desenhador como argumento. Este objeto será responsável por implementar a lógica de desenho da forma.
- O método desenhar() delega a responsabilidade de desenhar a forma para o objeto desenhador, chamando seu método desenharForma().

Funcionamento

- As classes DesenhoVetorial e DesenhoRaster representam implementações concretas de como desenhar uma forma.
- Elas provavelmente conteriam a lógica específica para desenhar a forma usando gráficos vetoriais ou raster, respectivamente (embora o código detalhado não seja mostrado aqui).
- O importante é que ambas as classes implementam o método desenharForma(), que é o que a classe Forma espera.

Resumindo

- O padrão Ponte separa a abstração de uma forma (o que ela é) da sua implementação (como ela é desenhada). Isso permite que você tenha diferentes implementações de desenho (vetorial, raster, etc.) e as use de forma intercambiável com a mesma abstração de forma.

Composto (Composite)

- Objetivo: Permite que você trate objetos individuais e composições de objetos de forma uniforme.
- Como: Cria uma estrutura de árvore onde cada nó pode ser uma folha (objeto individual) ou um ramo (composição de objetos).
- Quando usar: Quando você precisa representar uma hierarquia de objetos que podem ser tratados da mesma maneira.

```
// Componentes da hierarquia
class Arquivo { /* ... */ }
class Pasta { /* ... */ }
```