Resumo dos Padrões de Projetos (Design Patterns)

Publicado por Walmir Silva em 2 de novembro de 2021

Atualizado pela última vez em 31 de dezembro de 2021

É sempre um pouco difícil memorizar todos os **Padrões de Projetos** que temos à nossa disposição extraídos do **Livro GOF**. Porém, é de suma importância para alavancarmos nossa carreira como programadores , arquitetos ou engenheiros de software. Por este motivo, fiz este pequeno resumo para te ajudar na identificação e uso dos padrões da forma mais resumida possível.

Padrões de projetos de criação:

Abstract Factory

👉 Padrão de Projeto Abstract Factory em PHP com exemplo

O **Abstract Factory** é usado para criarmos diferentes famílias de objetos sem precisar especificar a classe concreta. Assim, definimos uma classe abstrata que cria os recursos e as classes concretas criam as classes de cada família. Este padrão tem relação com os padrões **Factory Method**, **Prototype** e **Singleton**.

Factory Method

👉 Padrão de Projeto Factory Method em PHP com exemplo

O padrão **Factory Method** utiliza um método de uma classe que cria objetos de outro tipo. Em outras palavras, este padrão fornece uma interface para criar objetos em uma classe mãe e permite que seus filhos alterem o tipo de objeto que será criado. Os padrões **Factories** costumam ter vários **Factory Methods** dentro deles.

Prototype

👉 Padrão de Projeto Prototype em PHP com exemplo

Este padrão de criação é usado quando temos um "modelo" de um objeto que queremos criar e para criar novos objetos clonamos este objeto. Em uma **Factory** podemos ter vários **Prototypes** e a criação de objetos irá gerar cópias deste objeto.

Singleton

👉 Padrão de Projeto Singleton em PHP com exemplo

O **Singleton** é utilizado quando queremos que exista só uma única instância de uma classe e que ela possa ser facilmente acessível de qualquer ponto do programa. É uma boa alternativa às variáveis globais e as classes que só possuem métodos estáticos. Em um **Singleton**, a própria classe é responsável por gerenciar sua única instância.

Builder

👉Padrão de Projeto Builder em PHP com exemplo

A definição do padrão **Builder** é bem clara. Ele separa a construção de um objeto complexo de sua representação de modo que o mesmo processo de construção possa criar diferentes representações. Em outras palavras, você tem a estrutura da classe e consegue criar os objetos como se fossem um passo a passo.

Padrões de projetos estruturais:

Façade

Um uso comum do **Façade** é quando temos que fazer a comunicação com um subsistema complexo. Para isso, o padrão fornece uma interface única para um conjunto de interfaces de um subsistema. O objetivo é diminuir o acoplamento no sistema ao concentrar o uso do subsistema em uma fachada, ou seja, em uma interface de nível mais elevado que torna o subsistema mais fácil de usar.

Composite

👉 Padrão de Projeto Composite em PHP com exemplo

O objetivo do **Composite** é compor objetos em estruturas de árvore, de um modo que possamos tratar um objeto da mesma maneira que uma única instância do mesmo tipo de objeto. Existe também uma estrutura todo-parte em um **Composite**: um nó na hierarquia de um **Composite** tem como partes integrantes de si todos os seus filhos.

Um bom exemplo são as interfaces gráficas: cada componente da tela herda de Widget e pode ter vários filhos. Com isso podemos tratar tanto um simples botão como uma janela completa do mesmo modo.

Este padrão se relaciona com o **Visitor**, que pode ser utilizado para realizar uma operação em um **Composite** e também com o **Iterator**, que pode ser utilizado para percorrer o **Composite**.

Adapter

👉 Padrão de Projeto Adapter em PHP com exemplo

O padrão **Adapter** serve como um adaptador. Ele converte a interface de um objeto para a interface esperada por um cliente. Permite que objetos com interfaces incompatíveis comuniquem-se efetivamente.

Bridge

O padrão **Bridge** permite que possamos desacoplar uma abstração de sua implementação, gerando hierarquias gêmeas. É como se você dividisse uma classe grande ou um conjunto de classes intimamente ligadas em duas hierarquias separadas. Literalmente cria-se uma “ponte” de ligação.

Decorator

De forma jocosa, o padrão **Decorator** permite adicionar coisas a um objeto em tempo de execução. Ou seja, ele nos permite adicionarmos responsabilidades adicionais a um objeto dinamicamente. Com isso ganhamos mais flexibilidade em detrimento ao uso de herança.

Um exemplo poderia ser em uma biblioteca de streams: um stream que pode escrever em arquivos pode ser decorado por outro que compacta os dados antes, que pode ser decorado por outro que criptografa os dados, e assim por diante.

Proxy

👉 Padrão de Projeto Proxy em PHP com exemplo

Um **Proxy** controla o acesso a um objeto, de modo a prover somente as funções permitidas a um objeto cliente. Um exemplo pode ser um proxy de proteção, que nega acesso a algumas partes de um objeto a um grupo de clientes.

Flyweight

Um **Flyweight** pode ser utilizado quando temos muitos objetos que compartilham uma grande parte de seu estado. A solução é criar um **Flyweight** que guarda esse estado compartilhado e o estado que muda é guardado com cada objeto.

Padrões de projetos comportamentais:

Strategy

👉 Padrão de Projeto Strategy em PHP com exemplo

Usamos o **Strategy** quando queremos ter uma família de algoritmos, de modo que possamos encapsular cada um deles , e torná-los intercambiáveis, permitindo assim que os algoritmos variem independentemente dos clientes que os utilizam.

A ideia do **Strategy** é encapsular esses algoritmos em uma classe separada e assim podemos trocar entre eles apenas mudando uma referência.

É uma alternativa a herança (cada algoritmo herdaria da classe cliente mudando só o método que faz o algoritmo), mas com a diferença que podemos “trocar de classe” em tempo de execução.

Um exemplo interessante poderia ser um programa que aplica filtros à imagem: cada filtro é uma classe separada que pode ser “carregada” e aplicada sobre uma imagem. Esse padrão se relaciona muito bem com o **Template Method**.

Template Method

👉Padrão de Projeto Template Method em PHP com exemplo

Um **Template Method** é um método concreto em uma classe abstrata que define os passos de um algoritmo a ser executado, porém cada passo deve ser implementado pelas subclasses concretas. Este padrão se relaciona com um **Strategy.** Enquanto o **Template Method** cria variações entre as partes de um algoritmo usando herança, o **Strategy** varia o algoritmo todo usando delegação.

State

Usar um **State** é necessário quando queremos que objeto altere seu comportamento quando seu estado interno muda. É como se o objeto mudasse de classe. Neste caso o estado é encapsulado em uma classe separada.

Um exemplo poderia ser um inimigo inteligente em um jogo: dependendo da situação do ambiente, seu estado muda para agressivo ou defensivo (este estado pode ser uma classe separada que controla as ações do inimigo quando ele está naquele estado).

Mediator

👉 Padrão de Projeto Mediator em PHP com exemplo

Utilizamos um **Mediator** quando vários objetos relacionam-se bastante. Como estas relações geram muito acoplamento, concentramos este acoplamento em uma classe mediadora. Assim, cada classe fala somente com o mediador, que repassa as mensagens para as outras classes. O mediador também concentra as regras de negócio dos objetos que ele media.

Iterator

👉 Padrão de Projeto Iterator em PHP com exemplo

Um **Iterator** serve para dar acesso aos elementos de uma coleção ou agregado sem expor sua implementação interna. Exemplos desse padrão são encontrados em coleções que suportam a iteração usando o comando *foreach* presente em algumas linguagens de programação.

Memento

👉 Padrão de Projeto Memento em PHP com exemplo

O **Memento** é usado para externalizar o estado de um objeto sem violar o encapsulamento, de modo a poder restaurar este estado posteriormente.

Command

👉 Padrão de Projeto Command em PHP com exemplo

Com o uso do padrão **Command**, criamos uma maneira de encapsular requisições em classes, para que estas requisições possam ser ordenadas, enfileiradas, etc. Um uso muito comum é em logs e em distribuição de tarefas.

Chain of Responsability

👉 Padrão de Projeto Chain of Responsability em PHP com exemplo

Através da cadeia de responsabilidades, este padrão desacopla o receptor do emissor da mensagem, criando uma cadeia de objetos em que um objeto decide se vai processar a mensagem ou passar a responsabilidade para o próximo. Este padrão se relaciona com o Composite.

Interpreter

Ele define uma espécie de gramática. Um **Interpreter** define uma representação para esta gramática e um interpretador para interpretar sentenças nesta gramática. Utilizado principalmente em compiladores e interpretadores (de XML, de linguagens de programação).

Visitor

👉 **Padrão de Projeto Visitor em PHP com exemplo**

Um **Visitor** representa uma operação a ser executada em todos os elementos de uma estrutura de objetos. Este padrão se relaciona com o **Composite** (percorre os elementos do Composite) e **Interpreter** (neste caso o **Visitor** realiza a interpretação).

Diferença com o **Iterator**: enquanto o **Visitor** representa uma operação executada em uma estrutura de elementos, o **Iterator** simplesmente fornece um meio de acessar os elementos da estrutura.

Observer

👉 **Padrão de Projeto Observer em PHP com exemplo**

Um **Observer** define uma relação de um para muitos tal que quando algo do primeiro objeto mudar, todos os outros objetos são avisados . Isso é feito de forma a não criar acoplamento entre os objetos.

Este é um resumo do resumo. A ideia é termos um mapa de consulta rápida para quando não lembrarmos de um determinado padrão. Espero que tenha gostado e não se esqueça de compartilhar para que este site alcance o maior número de pessoas possíveis.

Confiança Sempre!!!

Fontes:

- https://vitruvius.com.br/revistas/read/arquitextos/18.212/6866
- https://sites.icmc.usp.br/rtvb/apostila.pdf
- https://repositorio.ufpe.br/bitstream/123456789/11981/1/DISSERTA%C3%87%C3%83O%20Roberto%20Tenorio%20Figueiredo.pdf

- https://speakerdeck.com/hhamon/learning-design-patterns-with-symfony?slide=29
- https://refactoring.guru/
- https://netgen.io/learn/php-workshops/php-design-patterns
- https://phptherightway.com/pages/Design-Patterns.html
- http://www.facom.ufu.br/~bacala/ESOF/05b-Padr%C3%B5es%20Gof.pdf
- https://www.opus-software.com.br/design-patterns/

Publicado em Design Pattern Padrões de Projetos PHP Programação

Abstract Factory Adapter Brifge Builder Chain of Responsability Command Composite Decorator design pattern Façade Flyweight Gang of Four GOF Interpreter Iterator Mediator Memento Method Factory Observer padrões de projetos programação Proxy Singeton State Strategy Template Method Visitor

Publicação Anterior
Como calcular Complexidade de Algoritmo na prática

Próxima Publicação
Descomplicando os Padrões de Projetos (Design Patterns)

Seja o primeiro a comentar

Deixe um comentário

O seu endereço de e-mail não será publicado. Campos obrigatórios são marcados com *

Comentário

Nome*

Email*

Website

Salvar meus dados neste navegador para a próxima vez que eu comentar.

Publicar comentário