Tema 4

Módulo 1

**Padrão Chain of Responsibility**

Encadeia objetos para que cada um tenha a chance de processar uma solicitação. Se um objeto não puder lidar com a solicitação, ele a passa para o próximo na cadeia.

**Vantagens:**

* Reduz o acoplamento entre objetos.
* Facilita a adição de novos manipuladores.
* Segue o princípio da responsabilidade única (SRP).

**Desvantagens:**

* Pode ser difícil depurar devido à passagem da solicitação por múltiplos manipuladores.
* Nem sempre garante que a solicitação será processada.

Diagrama

O conteúdo gerado por IA pode estar incorreto.

A cadeia de tratamento de erros é semelhante a esse modelo:

Interface gráfica do usuário, Texto, Aplicativo

O conteúdo gerado por IA pode estar incorreto.

**Consequências e padrões relacionados ao padrão Chain of Responsibility**

O padrão Chain of Responsibility reduz a complexidade de uma classe que tenha que lidar com várias possibilidades de tratamento de uma requisição, transformando as diversas operações e estruturas condicionais complexas originalmente existentes em um conjunto de objetos interconectados, que podem ser combinados de diferentes formas, gerando uma solução menos acoplada e mais flexível. Por outro lado, existe o risco de uma requisição não ser respondida de forma adequada, caso a configuração da cadeia não seja corretamente realizada.

**Atenção**

Esse padrão é frequentemente utilizado em conjunto com o padrão Composite. Nesse caso, não é necessário implementar um sucessor, dado que podemos utilizar o relacionamento entre o agregado e as suas partes para encadear as chamadas pelos elementos da estrutura de composição.

Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto.

**Padrão Command**

Encapsula uma ação como um objeto, permitindo desfazer, refazer e armazenar comandos para execução posterior.

Uma imagem contendo Diagrama

O conteúdo gerado por IA pode estar incorreto.

**Vantagens:**

* Permite desfazer/refazer operações facilmente.
* Desacopla o emissor do receptor da ação.
* Facilita a criação de macros (sequências de comandos).

**Desvantagens:**

* Pode aumentar a complexidade do código devido à criação de muitas classes de comando.
* Pode consumir mais memória se muitos comandos forem armazenados.

Diagrama

O conteúdo gerado por IA pode estar incorreto.

Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto.

**Iterator**

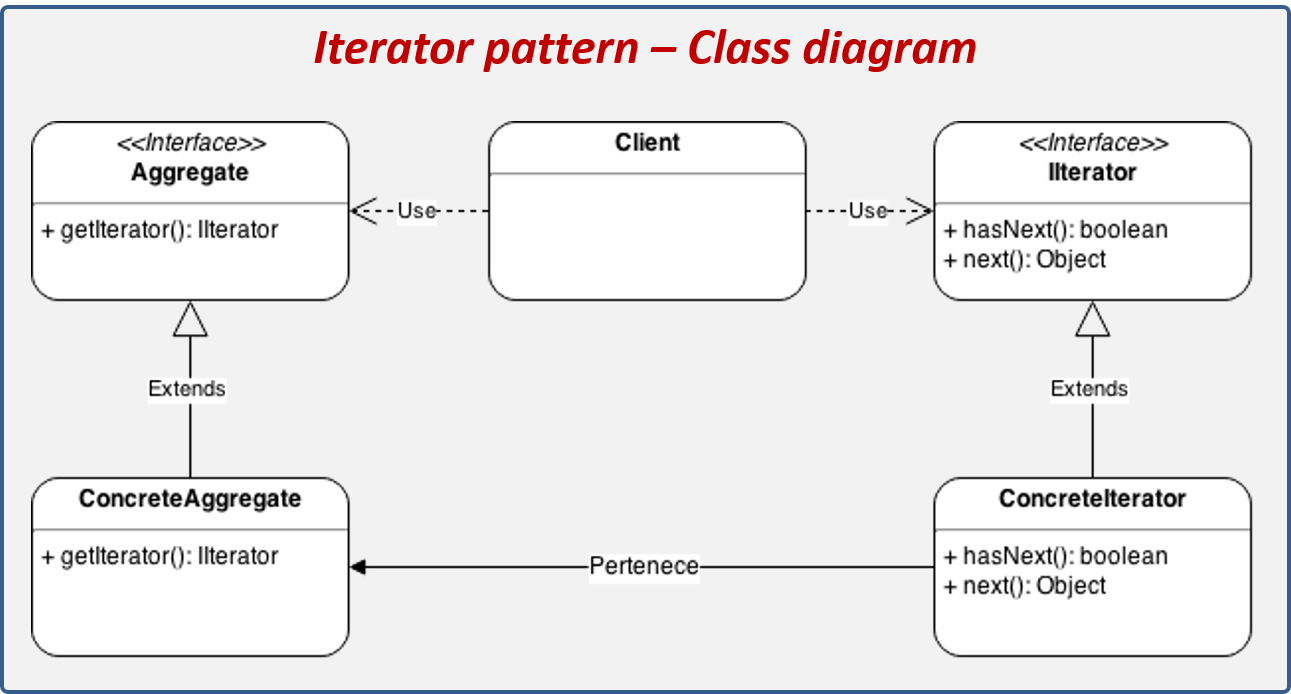
**Descrição:** Fornece um mecanismo para percorrer coleções sem expor sua estrutura interna.

**Vantagens:**

* Facilita a navegação em coleções sem depender de sua implementação interna.
* Suporta diferentes formas de iteração (ex: normal, reversa, filtrada).

**Desvantagens:**

* Pode ser menos eficiente se comparado a iterações diretas em certas estruturas de dados.
* Pode aumentar a complexidade ao lidar com coleções grandes e concorrentes.



O framework de estrutura de dados da linguagem Java implementa esse padrão. Coleções como ArrayList, LinkedList, HashSet e TreeSet são descendentes da classe genérica **Collection**. Nesse caso, as coleções específicas correspondem ao participante **ConcreteAggregate**, enquanto a classe Collection corresponde ao participante Aggregate. Em Java, a interface genérica Iterator define um conjunto de operações um pouco diferente daquele definido na estrutura do padrão:

Tela de celular com texto preto sobre fundo branco

O conteúdo gerado por IA pode estar incorreto.

Diagrama

O conteúdo gerado por IA pode estar incorreto.

**Atenção**

O padrão Iterator é frequentemente utilizado com o padrão Factory Method, uma vez que cada método Iterator do participante ConcreteAggregate é um Factory Method responsável por instanciar o respectivo ConcreteIterator.

Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto.

**Mediator**

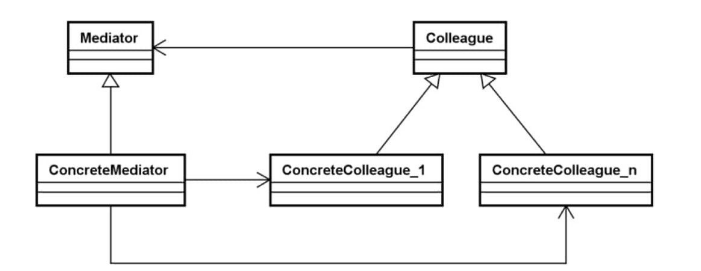
**Descrição:** Centraliza a comunicação entre objetos, reduzindo dependências diretas entre eles.

**Vantagens:**

* Reduz o acoplamento entre objetos.
* Facilita a manutenção e escalabilidade do sistema.

**Desvantagens:**

* O Mediator pode se tornar um ponto único de falha.
* Pode crescer excessivamente e se tornar difícil de gerenciar.



\*\*O mediador deve ser apenas um concentrador de eventos e um coordenador de execução, ficando a lógica do processamento distribuída pelos elementos a ele conectados.

**Atenção**

Como a comunicação entre os participantes e o mediador se dá por meio de notificações de eventos, é comum o Mediator ser aplicado em conjunto com o padrão Observer. Nesse caso, o mediador corresponde ao participante Observer, enquanto os objetos notificadores correspondem ao participante Subject do padrão Observer.

Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto.

**Memento**

**Descrição:** Permite salvar e restaurar o estado de um objeto sem violar seu encapsulamento.

**Vantagens:**

* Permite implementar funcionalidades como "desfazer".
* Mantém a integridade do encapsulamento dos objetos.

**Desvantagens:**

* Pode consumir muita memória se muitos estados forem armazenados.
* Pode ser difícil gerenciar estados complexos.

**Strategy**

**Descrição:** Define um conjunto de algoritmos intercambiáveis, permitindo escolher o mais adequado em tempo de execução.

**Vantagens:**

* Reduz a complexidade do código, separando os algoritmos da lógica principal.
* Facilita a manutenção e a adição de novas estratégias.

**Desvantagens:**

* Pode aumentar o número de classes no sistema.
* Pode ser mais difícil escolher a estratégia correta dinamicamente.

**Observer**

**Descrição:** Permite que múltiplos objetos sejam notificados automaticamente sobre mudanças de estado em outro objeto.

**Vantagens:**

* Reduz o acoplamento entre os objetos observados e os observadores.
* Facilita a criação de sistemas reativos.

**Desvantagens:**

* Pode ser difícil de depurar devido à natureza assíncrona das notificações.
* Se não for gerenciado corretamente, pode causar problemas de desempenho devido a notificações em cascata.

**Visitor**

**Descrição:** Permite adicionar novos comportamentos a uma estrutura de classes sem modificar essas classes diretamente.

**Vantagens:**

* Facilita a adição de operações sem modificar a estrutura de classes existente.
* Segue o princípio aberto-fechado (OCP).

**Desvantagens:**

* Pode violar o encapsulamento ao expor detalhes internos das classes.
* Pode tornar o código mais difícil de entender se houver muitas visitas diferentes.

**State**

**Descrição:** Permite que um objeto mude seu comportamento com base em seu estado interno, encapsulando estados em classes separadas.

**Vantagens:**

* Reduz estruturas de decisão complexas (ex: muitos if e switch).
* Facilita a manutenção e evolução dos estados.

**Desvantagens:**

* Pode aumentar a complexidade devido à criação de muitas classes de estado.
* Se mal implementado, pode resultar em mudanças de estado inconsistentes.

**Interpreter**

**Descrição:** Define uma gramática e um interpretador para processar sentenças nessa gramática.

**Vantagens:**

* Útil para interpretar linguagens específicas e processar comandos dinamicamente.
* Segue o princípio aberto-fechado (permite extensão sem modificar o código existente).

**Desvantagens:**

* Pode ser ineficiente para gramáticas complexas.
* Pode ser difícil de escalar para linguagens grandes.

**Template Method**

**Descrição:** Define a estrutura de um algoritmo na superclasse, permitindo que subclasses implementem etapas específicas.

**Vantagens:**

* Evita duplicação de código, promovendo reuso.
* Facilita a manutenção, pois a lógica principal fica centralizada.

**Desvantagens:**

* Pode dificultar a modificação de certas partes do algoritmo sem afetar o restante.
* Pode levar a uma hierarquia de classes rígida e difícil de modificar.