
Padrões de Projeto

Padrões Comportamentais

— Dra. Alana Morais —

Objetivo Aula

Apresentar e discutir sobre os padrões comportamentais:

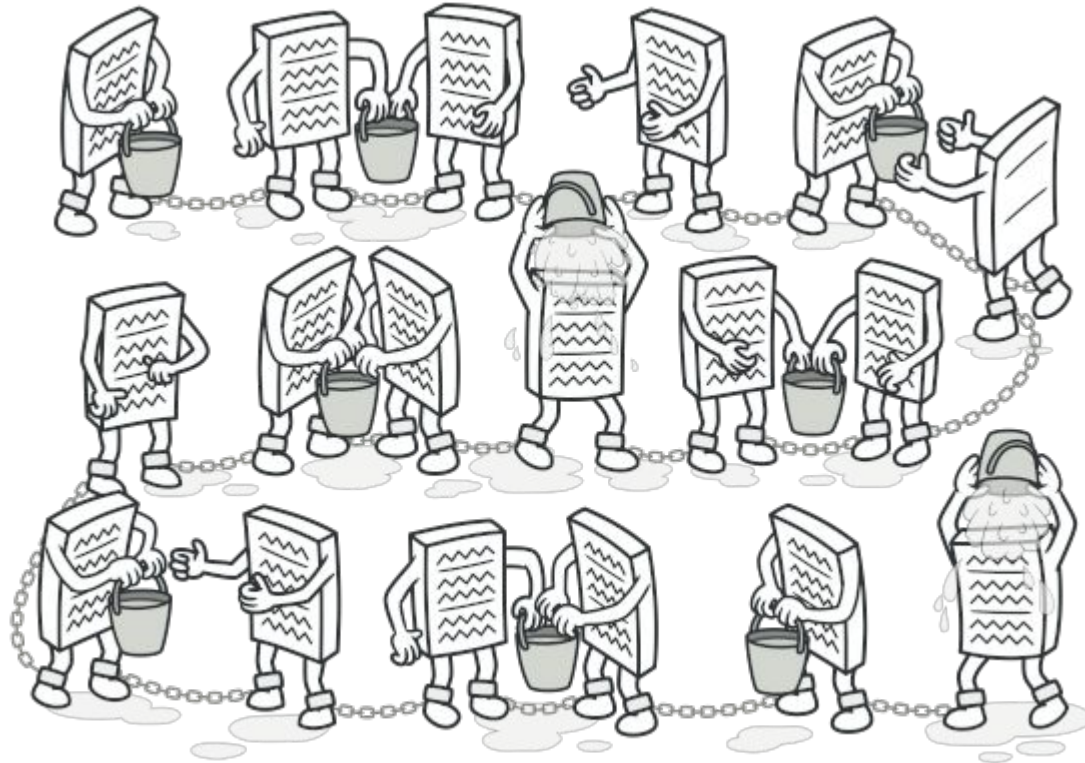
- Interpreter

- Chain of Responsibility

Chain of Responsibility

“EVITA ACOPLAR O REMETENTE DE UMA REQUISIÇÃO AO SEU DESTINATÁRIO AO DAR A MAIS DE UM OBJETO A CHANCE DE SERVIR A REQUISIÇÃO. COMPÕE OS OBJETOS EM CASCATA E PASSA A REQUISIÇÃO PELA CORRENTE ATÉ QUE UM OBJETO A SIRVA”.

Chain of Responsibility



Chain of Responsibility

- A intenção deste padrão é evitar o acoplamento do remetente de uma solicitação ao seu receptor, ao dar a mais de um objeto a oportunidade de tratar essa solicitação.
- Cria uma cadeia de objetos e vai passando a responsabilidade entre eles até que alguém possa responder pela chamada.
- Chain of Responsibility (Cadeia de Responsabilidade) é um padrão de design comportamental que permite passar solicitações ao longo de uma cadeia de manipuladores.

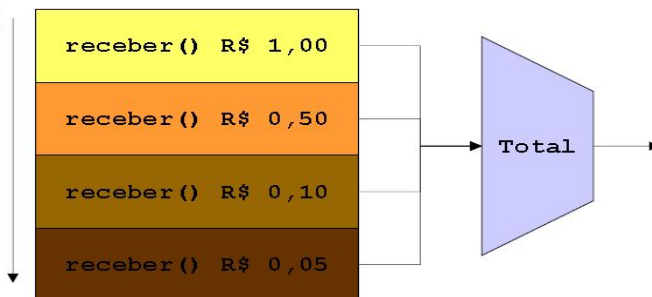
Chain of Responsibility - Problema



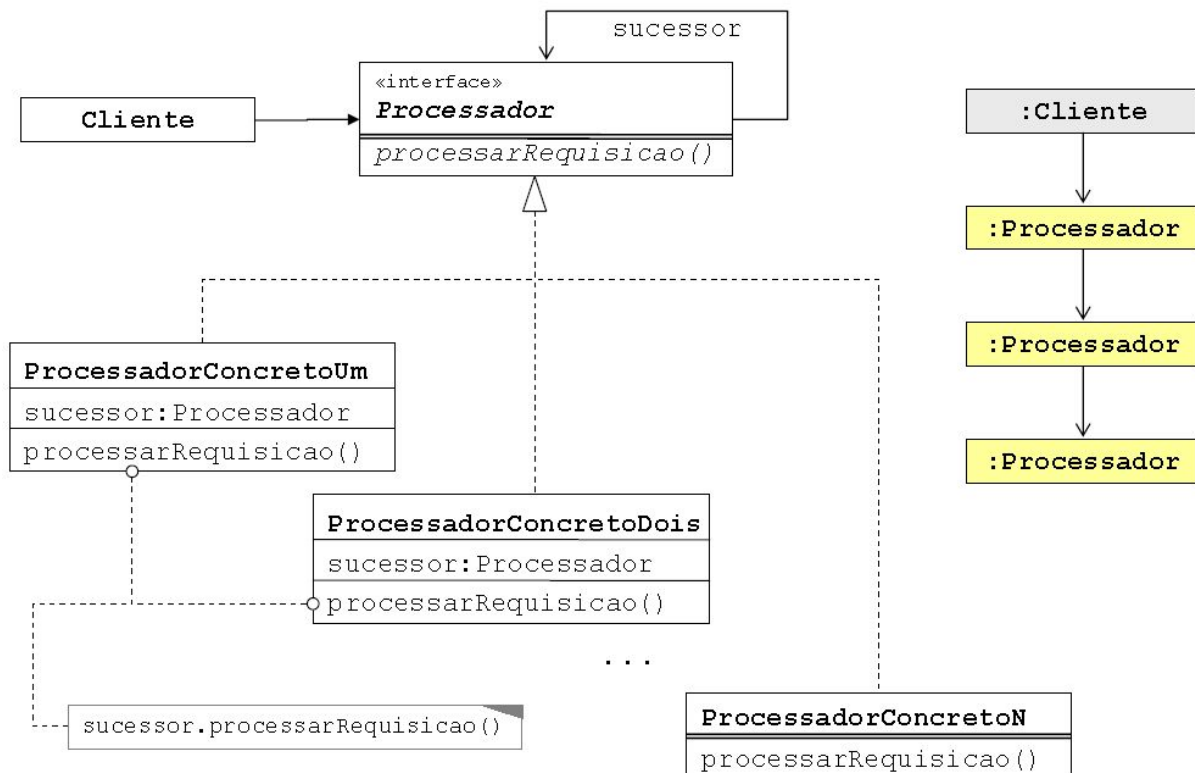
Permitir que vários objetos possam servir a uma requisição ou repassá-la

Permitir divisão de responsabilidades de forma transparente

*Um objeto pode ser uma **folha** ou uma **composição** de outros objetos*



Chain of Responsibility - Solução



Chain of Responsibility - Solução

```
public class cliente {  
    ...  
    Processador p1 = ...  
    Object resultado = p1.processarRequisicao();  
    ...  
}
```

```
public class ProcessadorUm implements Processador {  
    private Processador sucessor; ←  
    public Object processarRequisicao() {  
        ... // código um  
        return sucessor.processarRequisicao();  
    }  
}
```

Nesta **estratégia**
cada participante
conhece seu
sucessor

...

```
public class ProcessadorFinal implements Processador {  
    public Object processarRequisicao() {  
        return objeto;  
    }  
}
```

```
public interface Processador {  
    public Object processarRequisicao();  
}
```


Chain of Responsibility - Solução

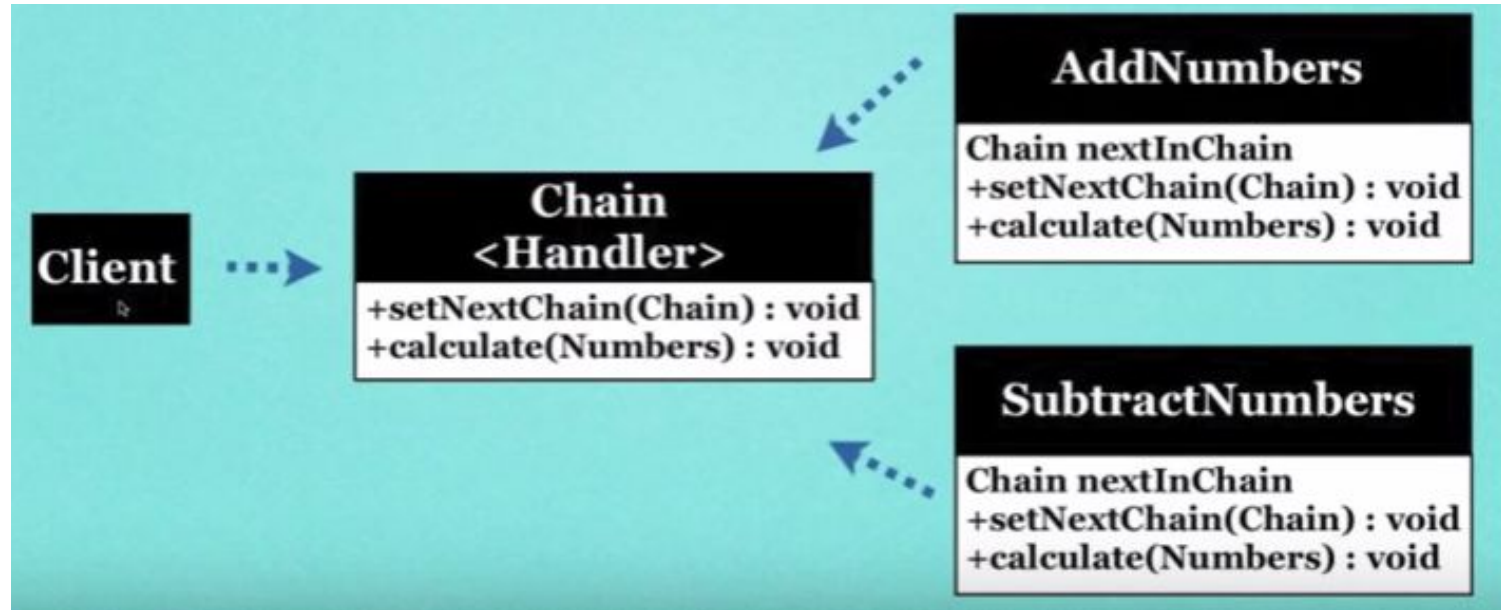
Pode-se implementar um padrão de várias formas diferentes.

- Cada forma é chamada de estratégia (ou idioma)

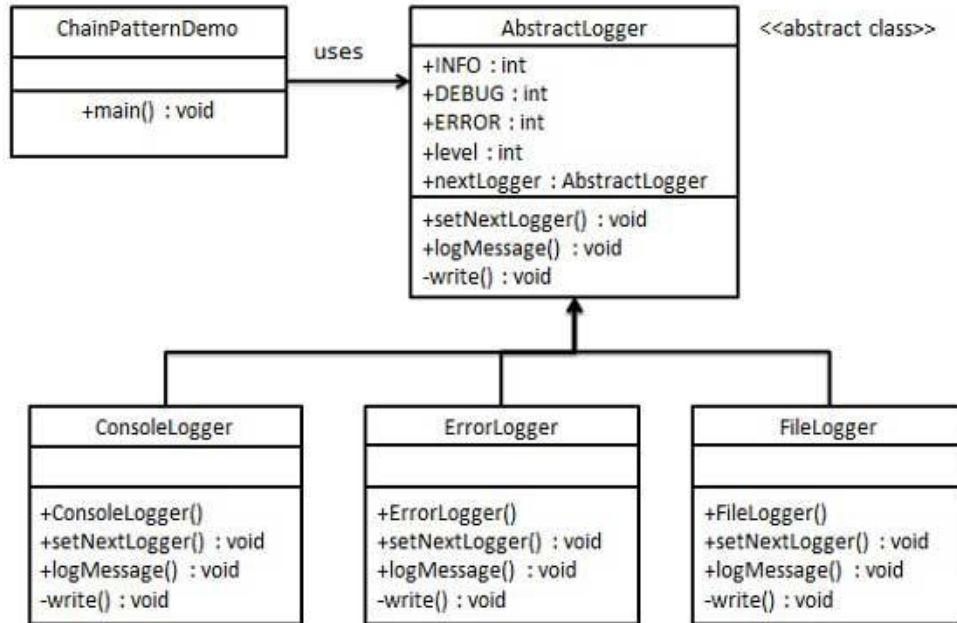
Chain of Responsibility pode ser implementada com estratégias que permitem maior ou menor acoplamento entre os participantes

- Usando um mediador: só o mediador sabe quem é o próximo participante da cadeia
- Usando delegação: cada participante conhece o seu sucessor

Chain of Responsibility - Exemplo



Chain of Responsibility - Exemplo 2



Chain of Responsibility

Vantagens

- Você pode controlar a ordem de tratamento de solicitações.
- Princípio da responsabilidade única .
Você pode separar classes que invocam operações de classes que executam operações.
- Princípio Aberto / Fechado. Você pode introduzir novos manipuladores no aplicativo sem quebrar o código do cliente existente.

Desvantagens

- Algumas solicitações podem acabar sem tratamento.

Interpreter

"DADA UMA LINGUAGEM, DEFINIR UMA REPRESENTAÇÃO PARA SUA GRAMÁTICA JUNTAMENTE COM UM INTERPRETADOR QUE USA A REPRESENTAÇÃO PARA INTERPRETAR SENTENÇAS DESSA LINGUAGEM."

Interpreter



Interpreter

- Definir a gramática de uma linguagem e criar um interpretador que leia instruções nesta linguagem e interprete-as para realizar tarefas.
 - Muito utilizado para interpretar DSL's ou criar compiladores.
- Pode mapear um domínio para uma língua, a língua para uma gramática e a gramática para um projeto de design hierárquico orientado a objetos.

Quando Usar?

- Existe uma linguagem a ser interpretada que pode ser descrita como uma árvore sintática;
- Funciona melhor quando:
 - A linguagem é simples;
 - Desempenho não é uma questão crítica.

Interpreter - Problema

- Se comandos estão representados como objetos, eles poderão fazer parte de algoritmos maiores
 - Vários padrões repetitivos podem surgir nesses algoritmos
 - Operações como iteração ou condicionais podem ser frequentes: representá-las como objetos Command
- Solução em OO: elaborar uma gramática para calcular expressões compostas por objetos
 - Interpreter é uma extensão do padrão Command (ou um tipo de Command; ou uma micro-arquitetura construída com base em Commands) em que toda uma lógica de código pode ser implementada com objetos

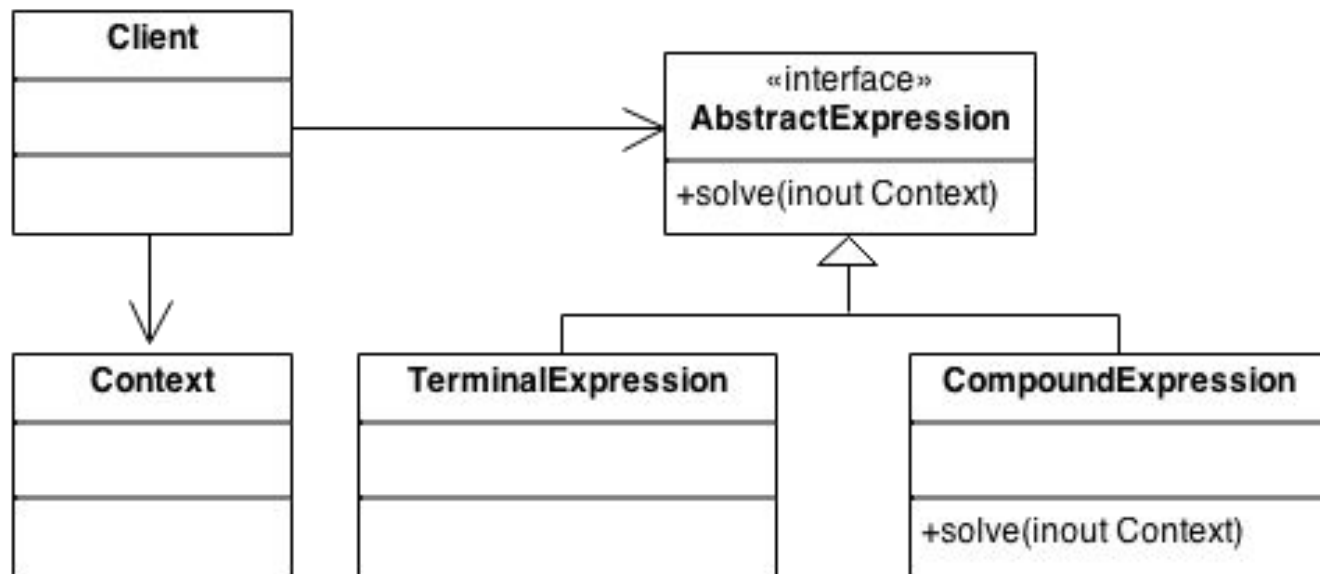
Interpreter - Solução

- O padrão usa uma classe para representar cada regra gramatical.
 - E como as gramáticas geralmente são hierárquicas na estrutura, uma hierarquia de herança de classes de regras é bem mapeada.
- Uma classe base abstrata especifica o método `interpret()`.
 - Cada subclasse concreta implementa `interpret()` aceitando (como um argumento) o estado atual do fluxo de idioma e adicionando sua contribuição ao processo de solução de problemas.

Interpreter - Solução

- O padrão sugere modelar o domínio com uma gramática recursiva.
- Cada regra na gramática é um 'composto' (uma regra que faz referência a outras regras) ou um terminal (um nó de folha em uma estrutura de árvore).
- O interpreter baseia-se na travessia recursiva do padrão Composite para interpretar as "sentenças" que ele pede para processar.

Interpreter - Solução



Interpreter

- Vantagens
 - É fácil mudar e estender a gramática: Pode alterar expressões existentes, criar novas expressões, etc.
 - Implementação é simples, pois as estruturas são parecidas.
- Desvantagens
 - Gramáticas complicadas dificultam: Se a gramática tiver muitas regras complica a manutenção.

Interpreter e Command

- Interpreter pode ser usado como um refinamento de Command:
 - Comandos são escritos numa gramática criada para tal;
 - Interpreter lê esta linguagem e cria objetos Command para execução.

Dúvidas?

alanamm.prof@gmail.com