

**IFNMG — CAMPUS MONTES CLAROS**  
**CIÊNCIA DA COMPUTAÇÃO**

**Giordani Andre Versiani Silva**

**Seminário (POO)**

**Command e Proxy**

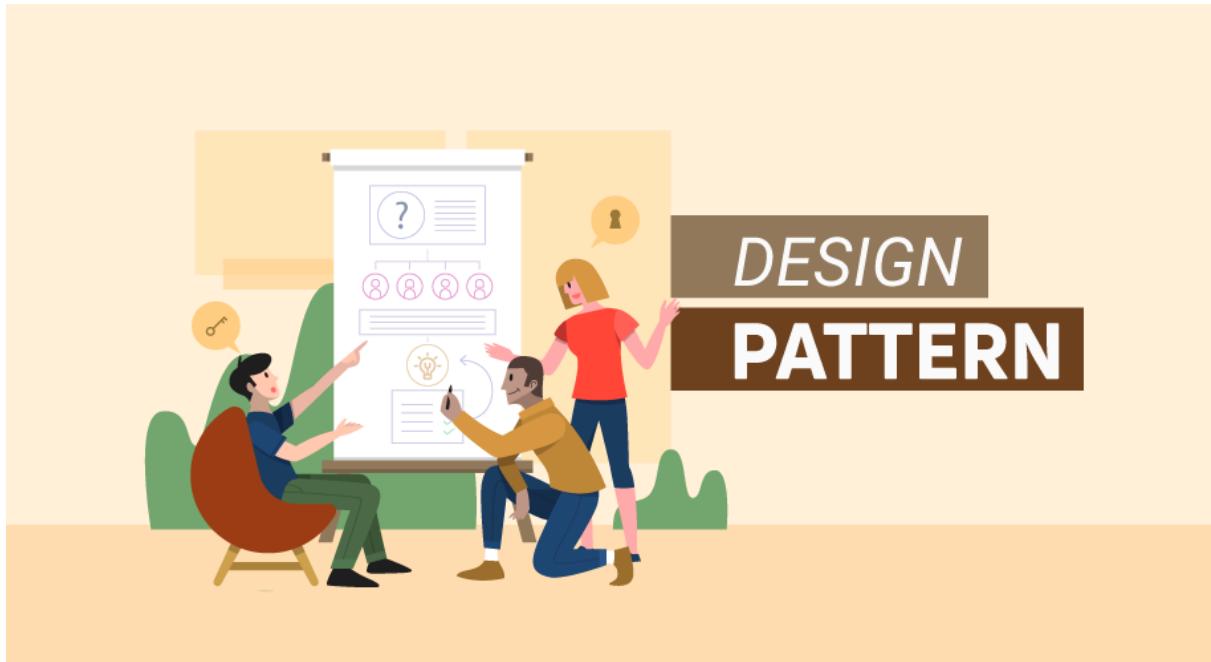
**Montes Claros - Minas Gerais**

**16 de setembro de 2024**

<b>1. Design Patterns.....</b>	
<b>1.1.Visão Geral e Motivações.....</b>	
<b>1.2. História.....</b>	
<b>1.3.Vantagens de sua aplicação.....</b>	
<b>1.4.Tipos de Padrões.....</b>	
<b>2. Proxy.....</b>	
<b>2.1.Objetivo.....</b>	
<b>2.2.Problema.....</b>	
<b>2.3.Conceitos Envolvidos .....</b>	
<b>2.4.Funcionamento e Aplicação.....</b>	
<b>2.5.Código do exemplo.....</b>	
<b>3. Command.....</b>	
<b>3.1.Objetivo.....</b>	
<b>3.2.Problema.....</b>	
<b>3.3.Conceitos Envolvidos .....</b>	
<b>3.4.Funcionamento e Aplicação.....</b>	
<b>3.5.Código do exemplo.....</b>	
<b>4.Referências.....</b>	

# 1. Design Patterns

## 1.1. Visão Geral e Motivações



Os design patterns são as soluções utilizadas para problemas de programação frequentemente enfrentados por empresas desenvolvedoras na hora de produzir um software. Eles funcionam como um padrão que pode ser implementado em diferentes linguagens de programação, otimizando o trabalho e ajudando a entregar bons resultados com menores esforços.

Esses padrões ajudam programadores a reutilizar soluções que já possuem resultados comprovados, além de facilitar a comunicação entre os profissionais envolvidos no projeto.

## 1.2. História

O conceito de padrões foi primeiramente descrito por Christopher Alexander em **Uma Linguagem de Padrões**. O livro descreve uma “linguagem” para o projeto de um ambiente urbano. As unidades dessa linguagem são os padrões.

A ideia foi seguida por quatro autores: Erich Gamma, John Vlissides, Ralph Johnson, e Richard Helm. Em 1994, eles publicaram **Padrões de Projeto — Soluções Reutilizáveis de Software Orientado a Objetos**, no qual eles aplicaram o conceito de padrões de projeto para programação. O livro mostrava 23 padrões que resolviam vários problemas de projeto orientado a objetos e se tornou um best-seller

rapidamente.

### **1.3.Vantagens de sua aplicação**

1. São reutilizáveis em vários projetos.
2. Fornecem soluções que ajudam a definir a arquitetura do sistema.
3. Melhoram as experiências de engenharia de software.
4. As alterações ou modificações tornam-se mais fáceis.
5. Fornecem transparência ao design de um aplicativo.
6. Torna o código reutilizável, livre de bugs e limpo.
7. Acelera o processo de desenvolvimento.
8. Reduz problemas comuns e recorrentes durante o processo de desenvolvimento.
9. São soluções comprovadas e testadas, pois foram construídas com base no conhecimento e na experiência de desenvolvedores de software especializados.

### **1.4.Tipos de Padrões**

1) Padrões criacionais: estes padrões oferecem diversas alternativas de criação de objetos, o que aumenta a flexibilidade e a reutilização de código. Alguns dos padrões desse tipo são:

- Factory Method
- Abstract Factory
- Builder
- Prototype
- Singleton

2) Padrões estruturais: Nos mostram como montar objetos e classes em estruturas maiores, sem perder a eficiência e flexibilidade. Alguns dos padrões desse tipo são:

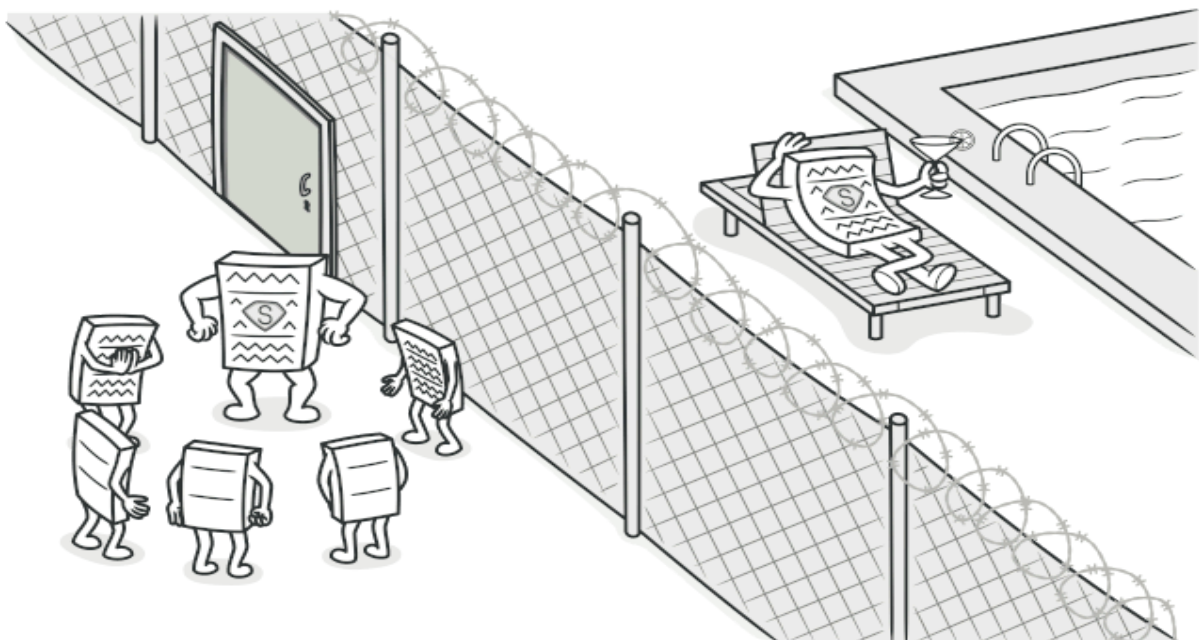
- Adapter

- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

3) Padrões comportamentais: Nos ajudam a trabalhar melhor com os algoritmos e com a delegação de responsabilidades entre os objetos. Os padrões que se destacam nesse tipo são:

- Chain of Responsibility
- Command
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

## 2. Proxy



### 2.1. Objetivo

Em termos simples, o padrão Proxy é um padrão estrutural que permite a criação de um objeto intermediário que atua como substituto para outro objeto. Esse substituto, conhecido como “proxy”, controla o acesso

ao objeto real, podendo adicionar funcionalidades extras, como controle de acesso, logging ou lazy loading.

## 2.2.Problema

O padrão Proxy é útil em situações onde se deseja controlar o acesso a um objeto ou recurso de maneira mais granular, permitindo ou negando o acesso conforme necessário. Alguns casos em que o padrão Proxy pode ser útil incluem:

1. **Redução do uso de recursos:** se um objeto é caro em termos de recursos, como memória ou processamento, o uso de um proxy pode ajudar a reduzir o número de vezes que o objeto é criado ou acessado, melhorando a eficiência do sistema.
2. **Controle de acesso:** o proxy pode ser usado para controlar o acesso a objetos sensíveis ou críticos, permitindo apenas que usuários autorizados tenham acesso a eles. Isso ajuda a garantir a segurança do sistema.
3. **Gerenciamento de objetos remotos:** o proxy remoto é usado para acessar objetos remotos em um sistema distribuído, permitindo que os clientes acessem objetos remotos como se estivessem acessando objetos locais.
4. **Criação sob demanda:** o proxy virtual é usado para criar objetos sob demanda, evitando a criação desnecessária de objetos caros. Isso ajuda a melhorar a eficiência do sistema. Em resumo, o padrão Proxy é útil sempre que há a necessidade de controlar o acesso a um objeto ou recurso de forma mais granular, garantindo a eficiência e a segurança do sistema.

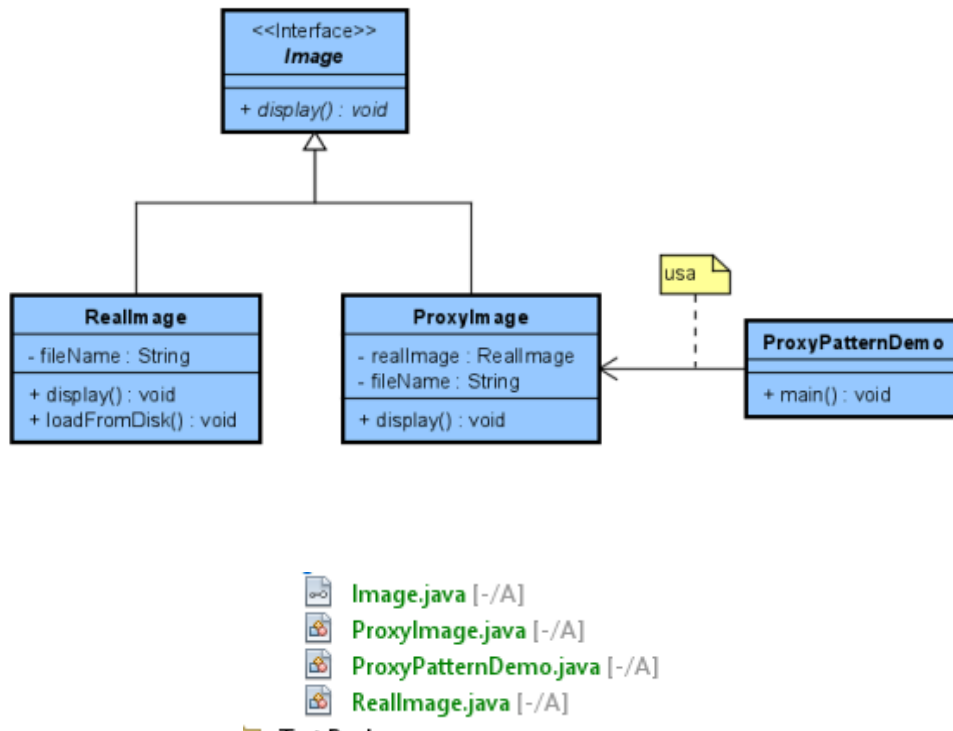
## 2.3.Conceitos Envolvidos

- **Interface do Serviço:** Define o contrato que o serviço e o proxy devem seguir.
- **Serviço:** Implementa a lógica de negócio real.
- **Proxy:** Encapsula o serviço, controlando acesso e gerenciando seu ciclo de vida.
- **Cliente:** Usa tanto o serviço quanto o proxy através da mesma interface.

## 2.4.Funcionamento e Aplicação

A aplicação do proxy é útil em situações onde se deseja controlar o acesso a um objeto ou recurso de maneira mais granular, permitindo ou negando o acesso conforme necessário.

Exemplo de aplicação do padrão Proxy (Diagrama de Classe):



### Image:

Interface que define o método `display()`, obrigando as classes que a implementam a fornecer uma implementação para exibir uma imagem.

### ProxyImage:

Implementa a interface **Image** e controla o acesso ao objeto **ReallImage**, carregando-o da memória apenas quando necessário (inicialização preguiçosa).

### ReallImage:

Implementa a interface **Image**, contém a lógica para carregar uma imagem do disco e exibi-la, representando a operação real do serviço.

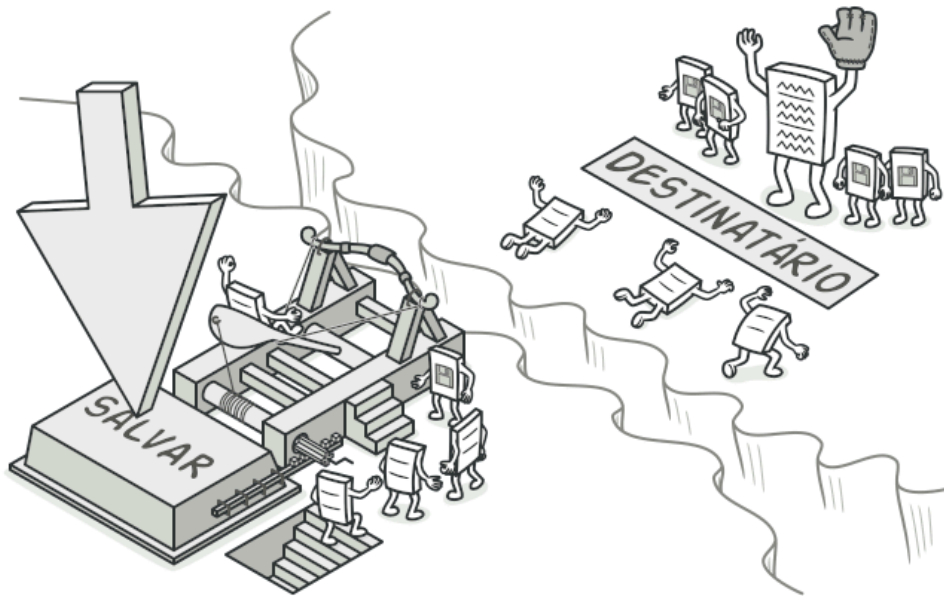
## ProxyPatternDemo:

Demonstra o uso do padrão Proxy, onde uma imagem é carregada do disco apenas uma vez e subsequentemente é reutilizada sem recarregamento.

## 2.5.Código do exemplo :

<https://github.com/GiordaniVersiani/Seminario-Poo>

## 3. Command



### 3.1.Objetivo

É um padrão de design comportamental que transforma uma solicitação em um objeto independente que contém todas as informações sobre a solicitação. Essa transformação permite parametrizar métodos com diferentes solicitações, atrasar ou enfileirar a execução de uma solicitação e oferecer suporte a operações que podem ser desfeitas.

### 3.2.Problema

O Command resolve o problema de encapsular uma solicitação como um objeto, permitindo que você:



Desacople o solicitante (quem faz a solicitação) do executante (quem executa a solicitação), facilitando a modificação, adiamento ou registro das operações.

Permitir operações reversíveis (como desfazer/redo) com facilidade.

Empilhar, enfileirar ou registrar comandos para execução futura ou processamento em lote, garantindo flexibilidade e extensibilidade.

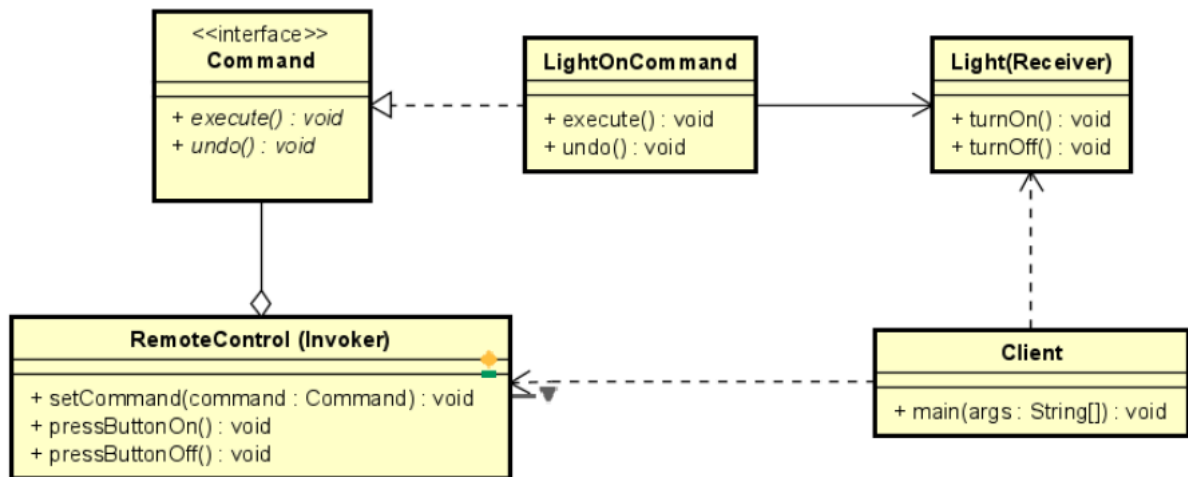
### 3.3. Conceitos Envolvidos






- **Remetente (Invocadora):** Responsável por iniciar os pedidos, armazenando e acionando um comando pré-criado, sem interagir diretamente com o destinatário.
- **Interface Comando:** Declara um único método para executar o comando.
- **Comandos Concretos:** Implementam diferentes pedidos e delegam a execução para um objeto destinatário, podendo armazenar parâmetros necessários.
- **Destinatária:** Contém a lógica de negócio real, executando o trabalho quando acionado por um comando.
- **Cliente:** Cria e configura comandos concretos, passando parâmetros e o destinatário para o construtor do comando.

### 3.4. Funcionamento e Aplicação

O Command encapsula uma solicitação como um objeto, permitindo desacoplar quem faz o pedido de quem o executa. O remetente (invocador) armazena e dispara comandos sem conhecer a lógica de execução, que fica com o destinatário. Esse padrão facilita a implementação de operações reversíveis (desfazer/refazer), enfileiramento ou agendamento de tarefas. É aplicado em sistemas que precisam de histórico de ações, automação de tarefas e processamento de comandos em lote.

Exemplo de aplicação do padrão Command (Diagrama de Classe):



 **Client.java** [-/A]  
 **Command.java** [-/A]  
 **Light.java** [-/A]  
 **LightOnCommand.java** [-/A]  
 **RemoteControl.java** [-/A]

**Client:** Cria os objetos principais (Light, LightOnCommand, RemoteControl), define o comando no controle remoto e aciona os métodos para ligar e desligar a luz.

**Command:** Interface que define os métodos `execute()` e `undo()` para comandos, representando ações como ligar e desligar a luz.

**Light:** Classe que contém os métodos `turnOn()` e `turnOff()`, representando a lógica real de ligar e desligar a luz.

**LightOnCommand:** Implementa a interface Command, encapsula o objeto Light e executa o comando para ligar ou desligar a luz.

**RemoteControl:** Dispara os comandos definidos, usando `pressButtonOn()` para executar o comando e `pressButtonOff()` para desfazê-lo.

### 3.5.Código do exemplo:

<https://github.com/GiordaniVersiani/Seminario-Poo>

## 4.Referências:

- 1-<https://refactoring.guru/pt-br>
- 2-<https://medium.com/@jonesroberto/design-patterns-parte-16-command-9c73af726c9c>
- 3-<https://medium.com/@jonesroberto/design-patterns-parte-14-proxy-9f72c15a2ee1>
- 4-<https://tallesvaliatti.com/design-pattern-command-1fc099a2a89e>
- 5-<https://www.opservices.com.br/design-patterns/>
- 6-<https://pt.linkedin.com/pulse/padr%C3%A3o-proxy-higor-dieg-o>
- 7-<https://www.devmedia.com.br/conheca-o-pattern-proxy-gof-gang-of-four/4066>
- 8-<https://uds.com.br/blog/a-importancia-dos-design-patterns-no-dev-de-software/>
- 9-[https://www.alura.com.br/artigos/design-patterns-introducao-padroes-projeto?srsId=AfmBOooamo5KrV4VMFICwmBHOwRbxqKRgHly2aEwRegpYF\\_m1OOOXXWR](https://www.alura.com.br/artigos/design-patterns-introducao-padroes-projeto?srsId=AfmBOooamo5KrV4VMFICwmBHOwRbxqKRgHly2aEwRegpYF_m1OOOXXWR)
- 10-<https://pt.linkedin.com/pulse/padr%C3%A3o-comportamental-command-s%C3%A9rie-design-patterns-r-woelke->