Padrão de Projetor Creator e Proxy

Alunos: Leonardo e Vinicius

INSTITUTO FEDERAL
Rio Grande do Sul
Campus Farroupilha

Padrão de projeto creator

Características

- Faz parte dos princípios do GRASP;
- E utilizado para quando se deseja eleger a responsabilidade de criar objetos para outras classes;
- Ajuda a facilitar a manutenção e organização do código;
- Menos dependente de classes externas;

Onde e utilizado

- Melhora o design e a organização do código;
- Utilizado principalmente onde a criação de objetos precisa ser controlada para garantir consistência ou evitar a duplicação do código;
- Utilizado quando uma classe dependente de várias instâncias de outra classe, alguns exemplos são:
    - Um sistema de um jogo, a classe Jogo cria a instância do personagem;
    - Um sistema de pedidos, onde a classe pedido cria a instancia de item.

Vantagens do criador

- Reduz o acoplamento entre classes;
- Lógica de criação de objetos está centralizada;
- Torna o código mais organizado e fácil de entender;
- Facilidade em realizar testes unitários;

Desvantagens do criador

- Introduz muitas responsabilidades para a classe implementar;
- Para iniciantes o padrão tem uma certa complexibilidade;
- Pode levar ao uso excessivo de objetos intermediários ou a criação de classes que não seriam necessárias sem eles;

Exemplo de código

```
class Produto {  4 usages
    private String nome;  2 usages
    private double preco;  2 usages

    public Produto(String nome, double preco) {  1 usage
        this.nome = nome;
        this.preco = preco;
    }

    @Override
    public String toString() { return "Produto{" + "nome='" + nome + '\'' + ", preco=" + preco + '}'; }
}
```

```java
class Pedido {  2 usages
    private List<Produto> produtos;  3 usages

    public Pedido() { this.produtos = new ArrayList<>(); }

    // Método Creator que cria e adiciona um novo Produto ao pedido
    public void adicionarProduto(String nome, double preco) {  2 usages
        Produto produto = new Produto(nome, preco);
        produtos.add(produto);
    }

    public List<Produto> getProdutos() { return produtos; }
}
```

```java
public class Main {
    public static void main(String[] args) {
        Pedido pedido = new Pedido();

        // Adiciona produtos ao pedido usando o método Creator
        pedido.adicionarProduto( nome: "Notebook",  preco: 3500.00);
        pedido.adicionarProduto( nome: "Mouse",  preco: 150.00);

        System.out.println(pedido.getProdutos());
    }
}
```

Proxy

- O Proxy é um padrão estrutural que atua como um substituto ou intermediário para outro objeto;
- Ele permite controlar o acesso ao objeto original, possibilitando a execução de ações antes ou depois do acesso ao objeto real.

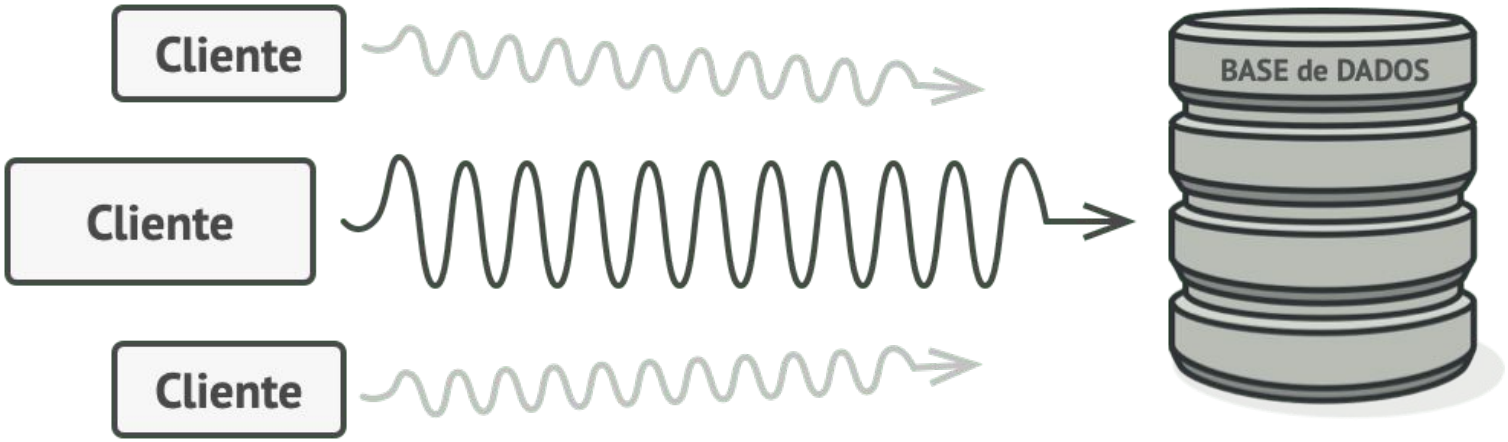
Proxy

Problema

- Objetos grandes podem consumir muitos recursos e são necessários apenas ocasionalmente;
- Consultas a bases de dados são lentas e consomem recursos;
- Inicialização preguiçosa poderia resolver, mas cada cliente teria que duplicar esse código;
- Em bibliotecas de terceiros, o código do objeto não pode ser modificado diretamente.

Proxy

Problema



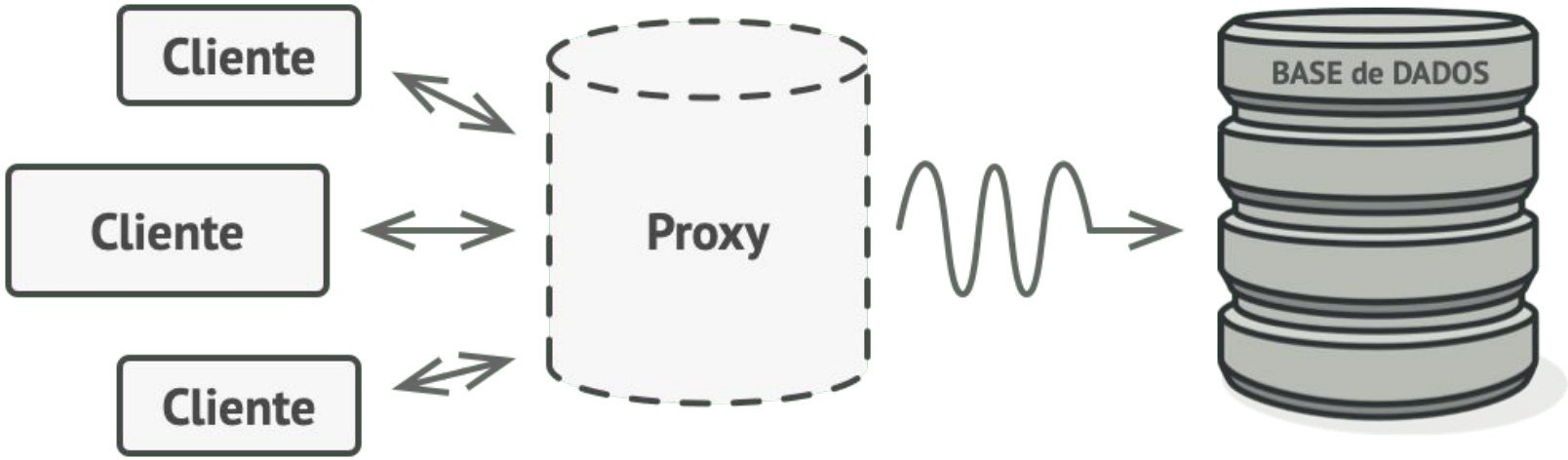
Proxy

Solução

- Crie uma classe Proxy com a mesma interface do objeto original;
- A aplicação usa o Proxy, que cria o objeto real apenas quando necessário;
- O Proxy cuida de inicialização preguiçosa e cache, sem que o cliente perceba;
- O Proxy permite executar ações antes ou depois do acesso ao objeto real, sem alterar o objeto original;
- Como implementa a mesma interface, o Proxy substitui o objeto real para qualquer cliente.

Proxy

Solução



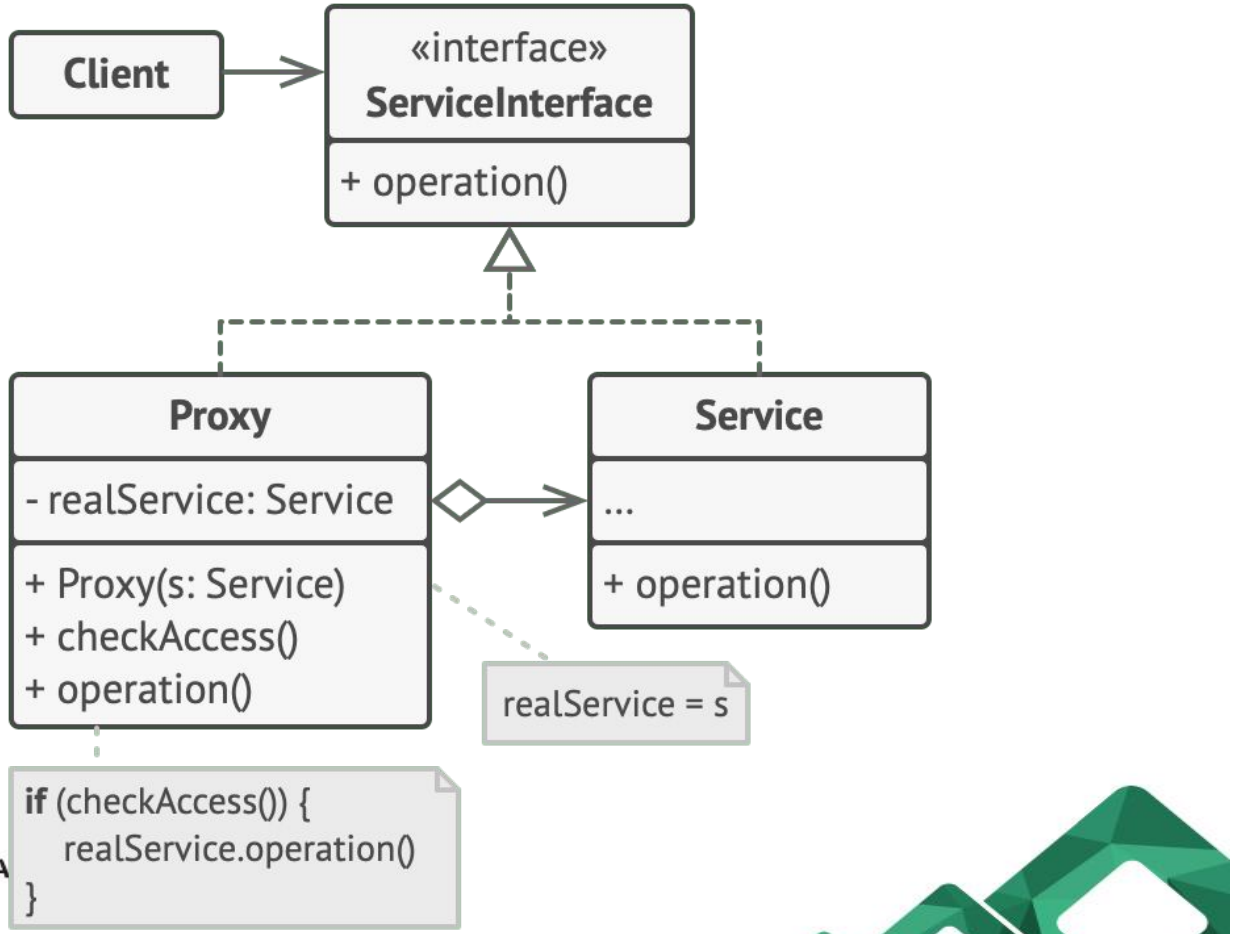
Proxy

Estrutura

- Interface do Serviço: Define a interface comum para o serviço e o proxy, permitindo que o proxy se disfarce do objeto real;
- Serviço: Classe que realiza a lógica de negócio principal;
- Classe Proxy: Contém uma referência ao objeto do serviço e gerencia o acesso a ele. Pode aplicar inicialização preguiçosa, controle de acesso ou cache antes de delegar a solicitação ao serviço;
- Cliente: Interage com o serviço ou com o proxy por meio da mesma interface. Isso permite que o cliente utilize o proxy no lugar do objeto real sem precisar de modificações.

Proxy

Estrutura



Proxy

Exemplo de código

```
interface Subject {
    2 usages   2 implementations
    void request();
}

class RealSubject implements Subject {
    2 usages
    @Override
    public void request() {
        System.out.println("Request to RealSubject.");
    }
}
```

Proxy

Exemplo de código

```
class Proxy implements Subject {
    3 usages
    private RealSubject realSubject;

    2 usages
    @Override
    public void request() {
        // Inicializa RealSubject apenas quando necessário
        if (realSubject == null) {
            realSubject = new RealSubject();
        }
        // Chama o método do RealSubject
        realSubject.request();
    }
}
```

Proxy

Exemplo de código

```
class ProxyPatternExample {
    public static void main(String[] args) {
        Subject proxy = new Proxy(); // Cria um Proxy
        proxy.request(); // Faz o pedido através do Proxy
    }
}
```

Proxy

Tipos

- Proxy Virtual: Inicialização preguiçosa, usado para objetos pesados;
- Proxy de Segurança: Controla o acesso, garantindo que apenas usuários autorizados acessem o objeto;
- Proxy Remoto: Representa objetos localizados em servidores remotos;
- Proxy de Cache: Armazena resultados para reutilização.

Proxy

Vantagens

- Redução do uso de recursos com inicialização sob demanda;
- Melhor controle de acesso e gerenciamento de ciclo de vida;
- Separação de responsabilidades (código de controle fica no Proxy);

Desvantagens

- Pode introduzir mais complexidade no código;
- Possível latência, já que o Proxy adiciona uma camada intermediária.