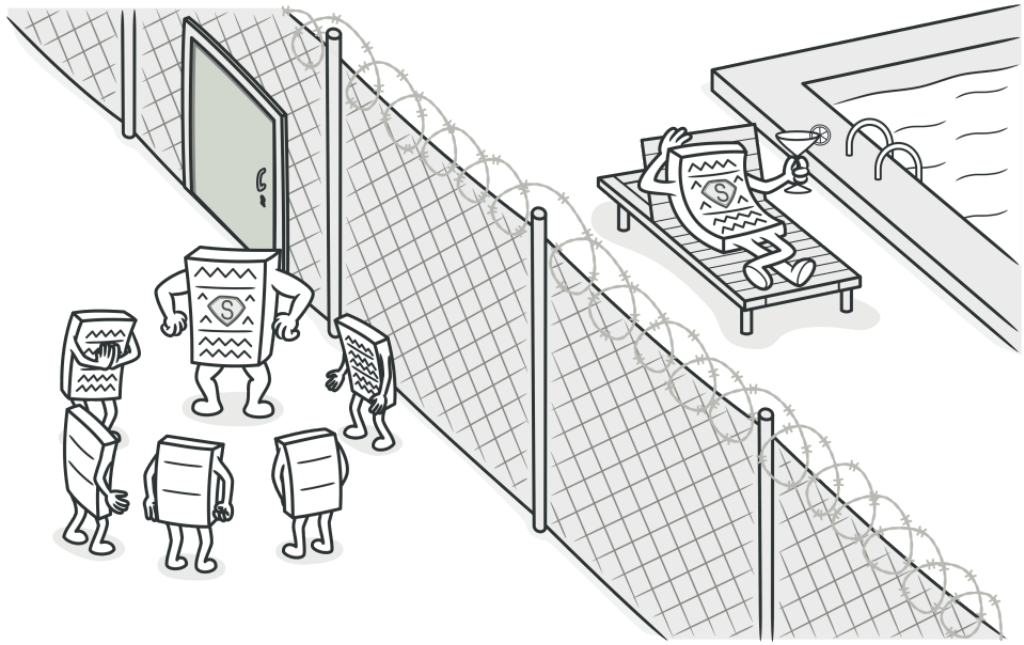
TÓPICO 14 - PROXY

Design Patterns - Professor Ramon Venson - SATC 2026.1

Motivação

Quando um objeto é caro, remoto ou sensível, o cliente nem sempre deveria acessá-lo diretamente.

O padrão Proxy resolve isso oferecendo um substituto com a **mesma interface** do objeto real.



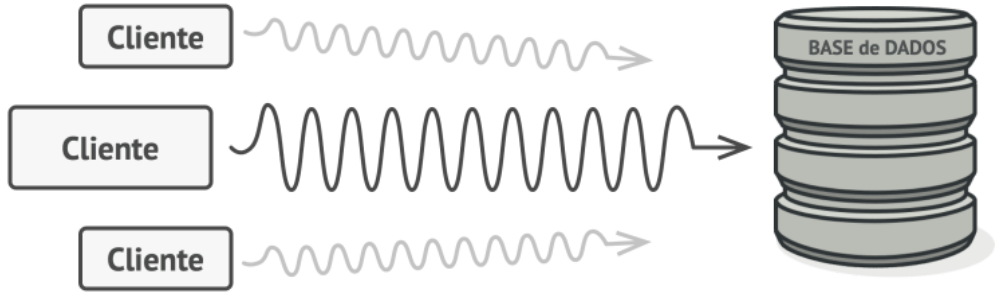
Problema

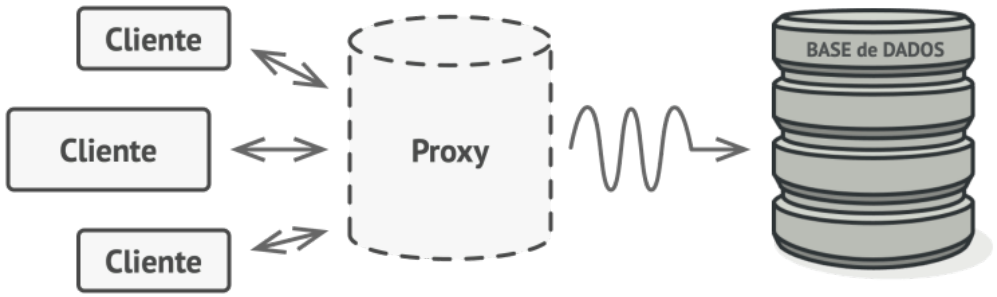
Imagine um sistema acadêmico que consulta um serviço remoto para gerar relatórios detalhados de alunos.

Sem intermediação, o cliente pode repetir chamadas caras, instanciar serviços pesados cedo demais e espalhar verificações de acesso.

Conceito principal

- **Proxy** é um padrão estrutural
- Ele preserva a mesma interface do serviço real
- O cliente pode usar proxy ou objeto real do mesmo jeito
- O proxy decide quando delegar a chamada
- Controle de acesso fica concentrado em um intermediário



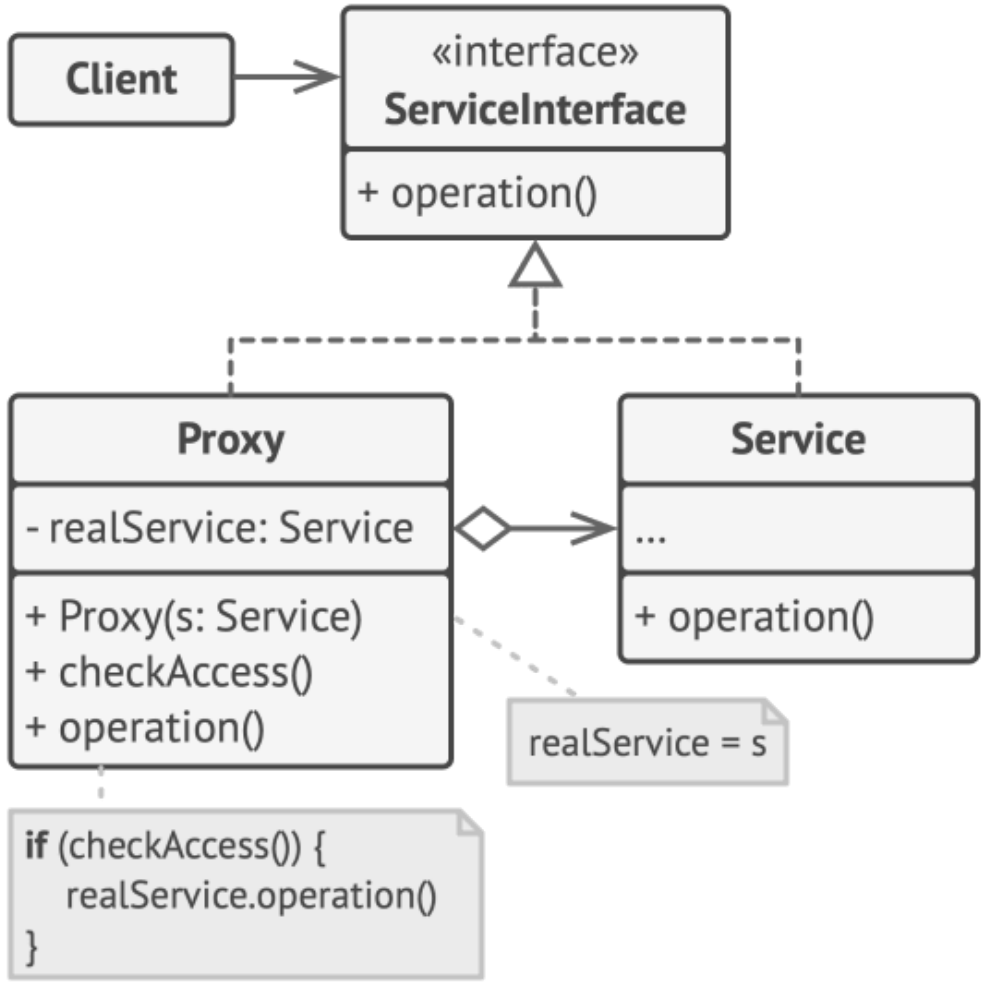


Como funciona

- O cliente chama a interface do serviço
- O proxy recebe a solicitação primeiro
- Ele pode validar, registrar, armazenar em cache ou criar o objeto real
- Depois delega ao serviço concreto quando necessário
- O cliente não precisa conhecer essas regras extras

Estrutura / Componentes

- **Service Interface** define o contrato comum
- **Real Service** executa a lógica principal
- **Proxy** implementa o mesmo contrato
- **Proxy** guarda referência para o serviço real
- **Cliente** depende apenas da interface



Tipos comuns

- **Proxy virtual** para inicialização preguiçosa
- **Proxy de proteção** para autorização
- **Proxy remoto** para objetos em outro servidor
- **Proxy de cache** para reaproveitar resultados
- **Proxy de log** para auditoria e monitoramento

Exemplo conceitual

- Sistema universitário consulta histórico completo do aluno
- O serviço real está em outro servidor
- Somente coordenadores podem acessar
- Consultas repetidas podem usar cache
- O cliente continua chamando a mesma interface

Código: contrato e serviço real

```
interface RelatorioService {
    String gerarRelatorio(int alunoId);
}

class RelatorioRemotoService implements RelatorioService {
    public String gerarRelatorio(int alunoId) {
        System.out.println("Consultando servidor remoto...");
        return "Relatorio completo do aluno " + alunoId;
    }
}
```

Código: proxy

```
class RelatorioProxy implements RelatorioService {
    private RelatorioService service;
    private String cache;
    private Integer ultimoAlunoId;

    public String gerarRelatorio(int alunoId) {
        if (cache != null && ultimoAlunoId != null && ultimoAlunoId.equals(alunoId)) {
            return cache;
        }

        if (service == null) {
            service = new RelatorioRemotoService();
        }

        cache = service.gerarRelatorio(alunoId);
        ultimoAlunoId = alunoId;
        return cache;
    }
}
```

Quando usar

- O objeto real é pesado ou caro
- O acesso precisa de autorização
- Existe comunicação remota a encapsular
- Resultados repetidos merecem cache
- Logs ou métricas devem ser adicionados sem mudar o serviço

Quando não usar

- O objeto real já é simples e barato
- A camada extra só adicionaria indireção
- O problema real é simplificação de subsistema inteiro
- O comportamento desejado é extensão funcional, não controle de acesso
- Um serviço direto resolveria melhor

Vantagens

- Controla o acesso sem alterar o cliente
- Permite lazy loading de objetos pesados
- Facilita cache, log e proteção
- Mantém contrato estável para o cliente
- Pode esconder detalhes de localização remota

Desvantagens

- Introduz mais uma classe e mais indireção
- Pode dificultar depuração
- Pode esconder latência real do serviço
- Se crescer demais, vira concentrador de responsabilidades
- Uso sem motivo claro adiciona complexidade acidental