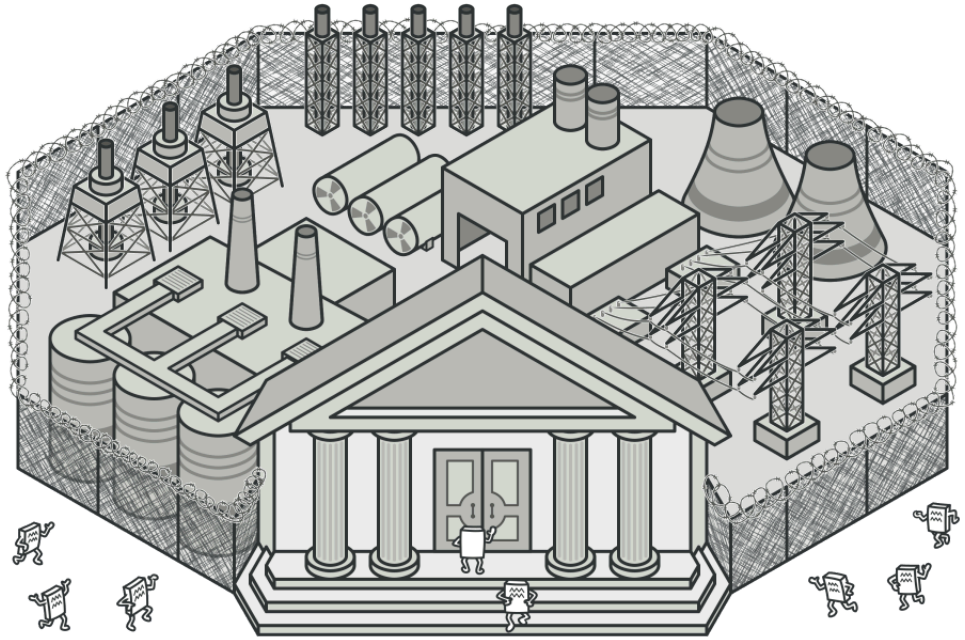
TÓPICO 13 - FACADE

Design Patterns - Professor Ramon Venson - SATC 2026.1

Motivação

Quando um cliente precisa conversar com muitas classes, APIs ou serviços internos, o código tende a ficar acoplado e difícil de manter.

O padrão Facade resolve isso oferecendo uma interface simples para um fluxo complexo.



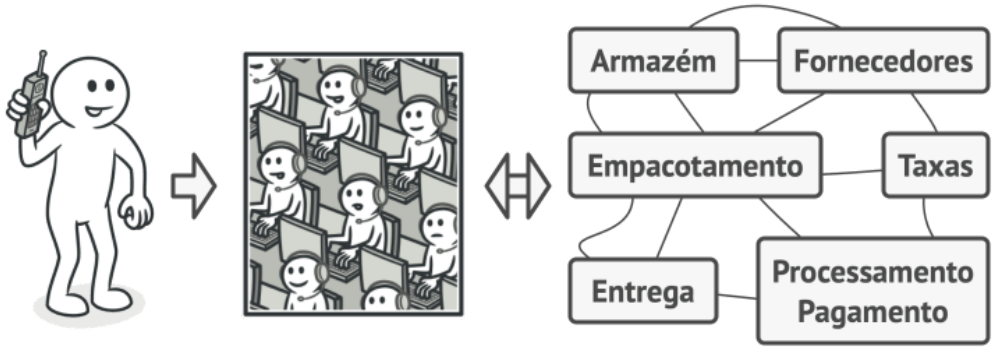
Problema

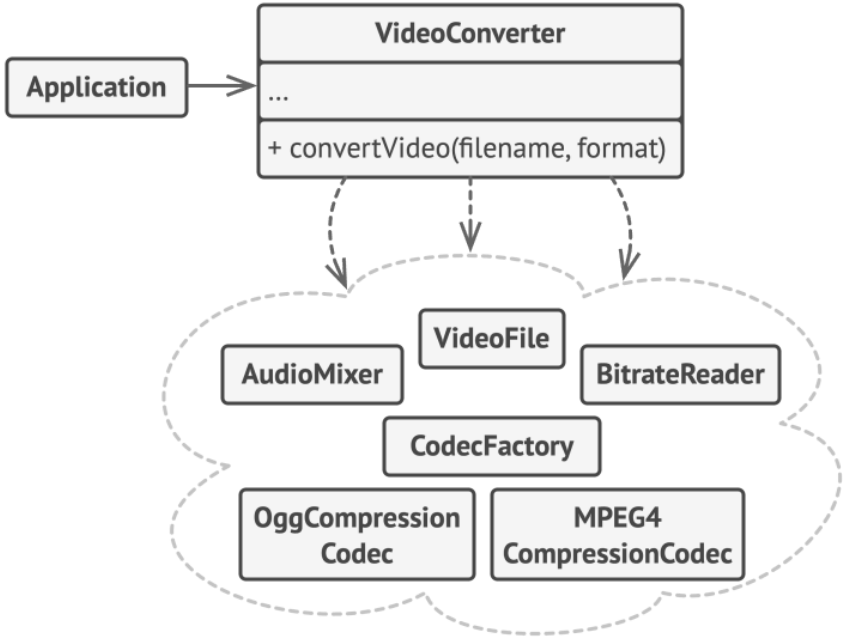
Imagine um sistema que precisa matricular um aluno, validar pendências, verificar vagas e enviar confirmação.

Sem uma fachada, o controlador pode acabar conhecendo detalhes demais do subsistema.

Conceito principal

- **Facade** é um padrão estrutural
- Ele fornece um ponto de entrada simples
- O cliente chama operações de alto nível
- A fachada coordena várias classes internas
- A complexidade fica escondida do lado do subsistema



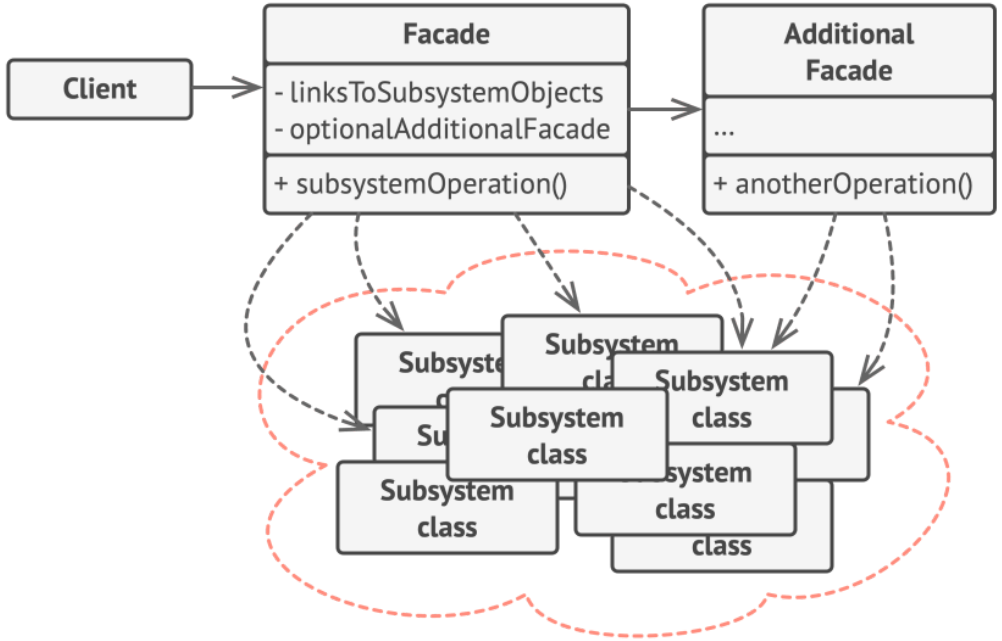


Como funciona

- O cliente chama a fachada
- A fachada conhece os serviços necessários
- Ela organiza a ordem das chamadas
- Cada serviço continua com sua responsabilidade
- O cliente depende menos de detalhes internos

Estrutura / Componentes

- **Cliente** deseja uma operação simples
- **Facade** expõe métodos mais diretos
- **Subsistema** contém classes técnicas e especializadas
- A fachada delega para os componentes corretos
- O subsistema pode continuar evoluindo internamente



Exemplo conceitual

- Sistema acadêmico de emissão de certificados
- Cliente quer apenas `emitirCertificado()`
- Fachada consulta aluno e histórico
- Depois gera PDF e envia por email
- O fluxo completo fica centralizado em uma classe

Código: serviços internos

```
class AlunoService {
    public String buscarNomeAluno(int alunoId) {
        return "Marina Souza";
    }
}

class HistoricoService {
    public boolean concluiuCurso(int alunoId) {
        return true;
    }
}
```

Código: fachada

```
class CertificadoFacade {
    private AlunoService alunoService = new AlunoService();
    private HistoricoService historicoService = new HistoricoService();
    private PdfService pdfService = new PdfService();

    public void emitirCertificado(int alunoId) {
        if (!historicoService.concluiuCurso(alunoId)) {
            return;
        }

        String nome = alunoService.buscarNomeAluno(alunoId);
        pdfService.gerarPdf("Certificado de conclusao para " + nome);
    }
}
```

Quando usar

- Há muitas classes para uma tarefa comum
- O cliente precisa de uma interface mais simples
- Fluxos de integração se repetem
- O acoplamento com o subsistema está alto
- Você quer estabilizar o ponto de entrada

Quando não usar

- O subsistema já é simples o suficiente
- A fachada esconderia detalhes importantes demais
- A classe central começaria a acumular tudo
- O problema real é de regra de negócio, não de integração
- Uma abstração menor resolveria melhor

Vantagens

- Simplifica o uso de bibliotecas e subsistemas
- Reduz acoplamento entre cliente e implementação
- Melhora leitura de fluxos de alto nível
- Centraliza orquestrações repetidas
- Facilita evolução interna dos componentes

Desvantagens

- Pode crescer demais se mal controlado
- Pode virar ponto de acoplamento excessivo
- Nem toda simplificação é desejável
- Não elimina a complexidade interna
- Pode confundir com service layer genérica