refactoring.guru

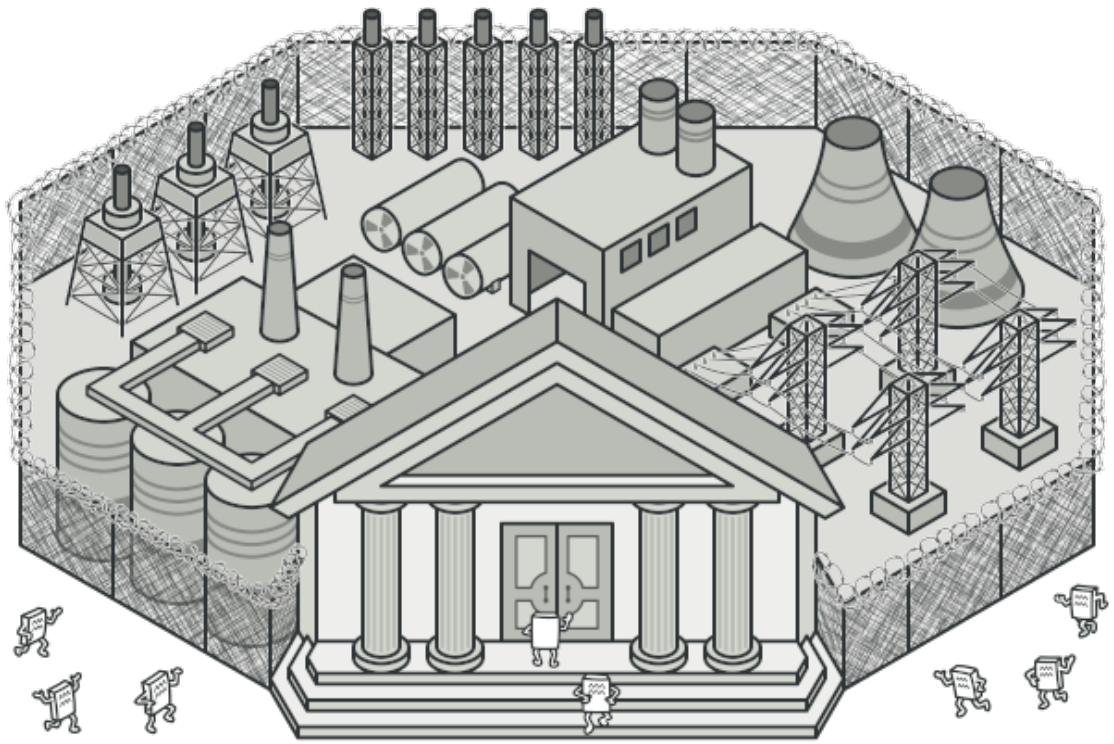
Facade

9–12 minutes

Também conhecido como: Fachada

Propósito

O **Facade** é um padrão de projeto estrutural que fornece uma interface simplificada para uma biblioteca, um framework, ou qualquer conjunto complexo de classes.



Problema

Imagine que você precisa fazer seu código funcionar com um

amplo conjunto de objetos que pertencem a uma sofisticada biblioteca ou framework. Normalmente, você precisaria inicializar todos aqueles objetos, rastrear as dependências, executar métodos na ordem correta, e assim por diante.

Como resultado, a lógica de negócio de suas classes vai ficar firmemente acoplada aos detalhes de implementação das classes de terceiros, tornando difícil compreendê-lo e mantê-lo.

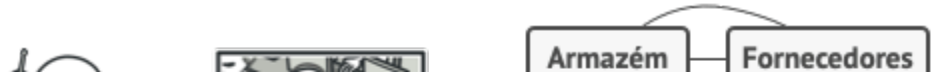
Solução

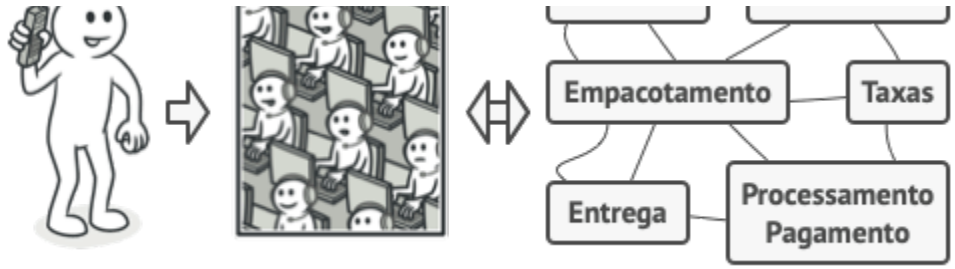
Uma fachada é uma classe que fornece uma interface simples para um subsistema complexo que contém muitas partes que se movem. Uma fachada pode fornecer funcionalidades limitadas em comparação com trabalhar com os subsistemas diretamente. Contudo, ela inclui apenas aquelas funcionalidades que o cliente se importa.

Ter uma fachada é útil quando você precisa integrar sua aplicação com uma biblioteca sofisticada que tem dúzias de funcionalidades, mas você precisa de apenas um pouquinho delas.

Por exemplo, uma aplicação que carrega vídeos curtos engraçados com gatos para redes sociais poderia potencialmente usar uma biblioteca de conversão de vídeo profissional. Contudo, tudo que ela realmente precisa é uma classe com um único método `codificar(nomeDoArquivo, formato)`. Após criar tal classe e conectá-la com a biblioteca de conversão de vídeo, você terá sua primeira fachada.

Analogia com o mundo real





Fazer pedidos por telefone.

Quando você liga para uma loja para fazer um pedido, um operador é sua fachada para todos os serviços e departamentos da loja. O operador fornece a você uma simples interface de voz para o sistema de pedido, pagamentos, e vários sistemas de entrega.

Estrutura

1. A **Fachada** fornece um acesso conveniente para uma parte particular da funcionalidade do subsistema. Ela sabe onde direcionar o pedido do cliente e como operar todas as partes móveis.
2. Uma classe **Fachada Adicional** pode ser criada para prevenir a poluição de uma única fachada com funcionalidades não relevantes que podem torná-lo mais uma estrutura complexa. Fachadas adicionais podem ser usadas tanto por clientes como por outras fachadas.
3. O **Subsistema Complexo** consiste em dúzias de objetos variados. Para tornar todos eles em algo que signifique alguma coisa, você tem que mergulhar fundo nos detalhes de implementação do subsistema, tais como objetos de inicialização na ordem correta e suprí-los com dados no formato correto.

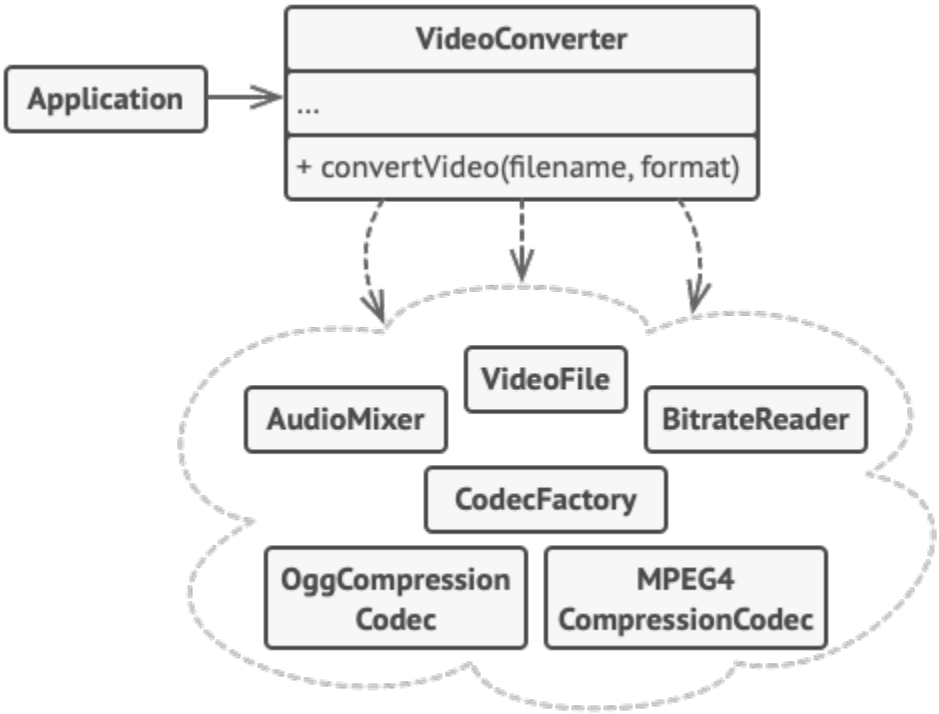
   As classes do subsistema não estão cientes da existência da

fachada. Elas operam dentro do sistema e trabalham entre si diretamente.

4. O **Cliente** usa a fachada ao invés de chamar os objetos do subsistema diretamente.

Pseudocódigo

Neste exemplo, o padrão **Facade** simplifica a interação com um framework complexo de conversão de vídeo.



Um exemplo de isolamento de múltiplas dependências dentro de uma única classe fachada.

Ao invés de fazer seu código funcionar com dúzias de classes framework diretamente, você cria a classe fachada que encapsula aquela funcionalidade e a esconde do resto do código. Essa estrutura também ajuda você a minimizar o esforço usando para atualizar para futuras versões do framework ou substituí-lo por outro. A única coisa que você precisaria mudar em sua aplicação

seria a implementação dos métodos da fachada.

```
// Essas são algumas das classes de um framework complexo de
um
// conversor de vídeo de terceiros. Nós não controlamos aquele
// código, portanto não podemos simplificá-lo.

class VideoFile


class OggCompressionCodec


class MPEG4CompressionCodec


class CodecFactory


class BitrateReader


class AudioMixer



// Nós criamos uma classe fachada para esconder a complexidade
// do framework atrás de uma interface simples. É uma troca
// entre funcionalidade e simplicidade.
class VideoConverter is
```

```
    method convert(filename, format):File is
        file = new VideoFile(filename)
        sourceCodec = (new CodecFactory).extract(file)
        if (format == "mp4")
            destinationCodec = new MPEG4CompressionCodec()
        else
            destinationCodec = new OggCompressionCodec()
        buffer = BitrateReader.read(filename, sourceCodec)
        result = BitrateReader.convert(buffer, destinationCodec)
        result = (new AudioMixer()).fix(result)
        return new File(result)

// As classes da aplicação não dependem de um bilhão de classes
// fornecidas por um framework complexo. Também, se você
decidir
// trocar de frameworks, você só precisa reescrever a classe
// fachada.
class Application is
    method main() is
        convertor = new VideoConverter()
        mp4 = convertor.convert("funny-cats-video.ogg", "mp4")
        mp4.save()
```

Aplicabilidade

Utilize o padrão Facade quando você precisa ter uma interface limitada mas simples para um subsistema complexo.

Com o passar do tempo, subsistemas ficam mais complexos. Até mesmo aplicar padrões de projeto tipicamente leva a criação de mais classes. Um subsistema pode tornar-se mais flexível e mais

fácil de se reutilizar em vários contextos, mas a quantidade de códigos padrão e de configuração que ele necessita de um cliente cresce cada vez mais. O Facade tenta consertar esse problema fornecendo um atalho para as funcionalidades mais usadas do subsistema que corresponde aos requerimentos do cliente.

Utilize o Facade quando você quer estruturar um subsistema em camadas.

Crie fachadas para definir pontos de entrada para cada nível de um subsistema. Você pode reduzir o acoplamento entre múltiplos subsistemas fazendo com que eles se comuniquem apenas através de fachadas.

Por exemplo, vamos retornar ao nosso framework de conversão de vídeo. Ele pode ser quebrado em duas camadas: relacionados a vídeo e áudio. Para cada camada, você cria uma fachada e então faz as classes de cada camada se comunicarem entre si através daquelas fachadas. Essa abordagem se parece muito com o padrão Mediator.

Como implementar

1. Verifique se é possível providenciar uma interface mais simples que a que o subsistema já fornece. Você está no caminho certo se essa interface faz o código cliente independente de muitas classes do subsistema.
2. Declare e implemente essa interface em uma nova classe fachada. A fachada deve redirecionar as chamadas do código cliente para os objetos apropriados do subsistema. A fachada deve ser responsável por inicializar o subsistema e gerenciar seu ciclo de vida a menos que o código cliente já faça isso.

3. Para obter o benefício pleno do padrão, faça todo o código cliente se comunicar com o subsistema apenas através da fachada. Agora o código cliente fica protegido de qualquer mudança no código do subsistema. Por exemplo, quando um subsistema recebe um upgrade para uma nova versão, você só precisa modificar o código na fachada.
4. Se a fachada ficar grande demais, considere extrair parte de seu comportamento para uma nova e refinada classe fachada.

Prós e contras

- Você pode isolar seu código da complexidade de um subsistema.
- Uma fachada pode se tornar um objeto deus acoplado a todas as classes de uma aplicação.

Relações com outros padrões

- O Facade define uma nova interface para objetos existentes, enquanto que o Adapter tenta fazer uma interface existente ser utilizável. O *Adapter* geralmente envolve apenas um objeto, enquanto que o *Facade* trabalha com um inteiro subsistema de objetos.
- O Abstract Factory pode servir como uma alternativa para o Facade quando você precisa apenas esconder do código cliente a forma com que são criados os objetos do subsistema.
- O Flyweight mostra como fazer vários pequenos objetos, enquanto o Facade mostra como fazer um único objeto que represente um subsistema inteiro.
- O Facade e o Mediator têm trabalhos parecidos: eles tentam

organizar uma colaboração entre classes firmemente acopladas.

- O *Facade* define uma interface simplificada para um subsistema de objetos, mas ele não introduz qualquer nova funcionalidade. O próprio subsistema não está ciente da fachada. Objetos dentro do subsistema podem se comunicar diretamente.
- O *Mediator* centraliza a comunicação entre componentes do sistema. Os componentes só sabem do objeto mediador e não se comunicam diretamente.
- Uma classe fachada pode frequentemente ser transformada em uma singleton já que um único objeto fachada é suficiente na maioria dos casos.
- O Facade é parecido como o Proxy no quesito que ambos colocam em buffer uma entidade complexa e inicializam ela sozinhos. Ao contrário do *Facade*, o *Proxy* tem a mesma interface que seu objeto de serviço, o que os torna intermutáveis.