

Padrões de Projeto: Façade e Proxy

TED-6

José Carlos Ribeiro Soares Junior

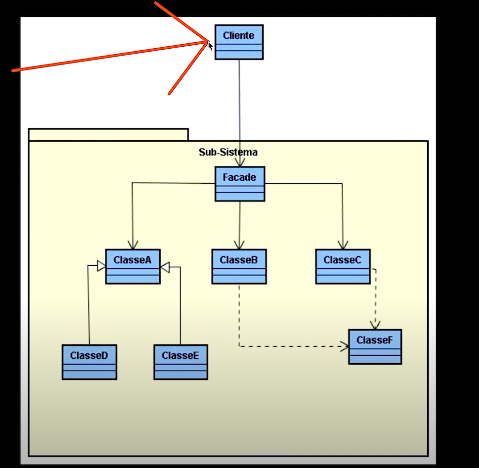
**Padrão de projeto FAÇADE.**

Este padrão é certamente um dos mais conhecidos do **Padrões GoF** ('Gang of Four') onde é utilizado para criar novas estruturas através dele, e organiza de forma estruturada classes e objetos.

**1\* O nome desse padrão é bem a cara dele, uma fachada, é um objeto que faz a comunicação com o corpo de código todo complexo.**

O Façade organiza em sua **interface de forma sintetizada** todo o modulo que está sendo desenvolvido e não só a classe de negócio como muitos fazem, além de que acabam por não usufruir de todo o poder de organização do padrão. Dentro deste padrão de projetos é possível construir a orquestração para as chamadas das camadas de negócios, tratamento de erros, onde permite que seja centralizado ou mais limpo possível seu código, facilitando demais a organização do projeto.

**2\* A intenção deste é apresenta uma camada de ligação para deixar as coisas mais simples além de se responsabilizar com a comunicação entre o back e o front.**

 O interessante dessa classe é que ela realmente é a fachada, como diz a tradução do seu nome, onde toda a **organização do seu código está por trás**, bem organizada, toda a complexidade é encapsulada. Usar o façade é ideal para a manutenção do front-and e do back-and, trabalhando com essa camada simples, sintetizada, ajuda aos desenvolvedores que forem fazer a manutenção do seu código futuramente, deixa fácil o entendimento de como funciona a regra de negócio, toda a sequência de chamadas, ao olhar para o código já é possível entender do que se trata.

**3\* É interessante usar este padrão sempre que puder, para a organização da comunicação, bem melhor usar um intermediador que fará todo o trabalho para uma classe especifica, do que ficar delegando funções para uma única classe.**

Exemplo de como usar o padrão, na imagem a classe cliente poderia ser qualquer classe que chamasse o back-and, ao invés da comunicação ser direta com as ClasseA, ClasseB ou ClassC, a organização passa a ter um ponto de referencia que vai trabalhar pra o programa de forma organizada e gerenciando as chamadas.

Um exemplo do dia a dia que achei nas pesquisas sobre este padrão de projeto façade é o DDD onde ele organiza de forma bem sintética na camada de aplicação.

**4\* A organização é simples, basta direcionar ao ClasseFaçade o que ela deve fazer e ela irá se responsabilizara de fazer todo o trabalho, isso permite que o usuário de forma simples tenha uma interação com a fachada e esta faça toda a comunicação com a classe complexa.**

**5\* O façade é altamente acoplado ao subsistema. Ou seja, seu sistema tem que ser bem montado, se não é break.**

Segundo o site do refactoring.guru em resumo: O **Facade** é um padrão de projeto estrutural que fornece uma interface simplificada para uma biblioteca, um framework, ou qualquer conjunto complexo de classes.

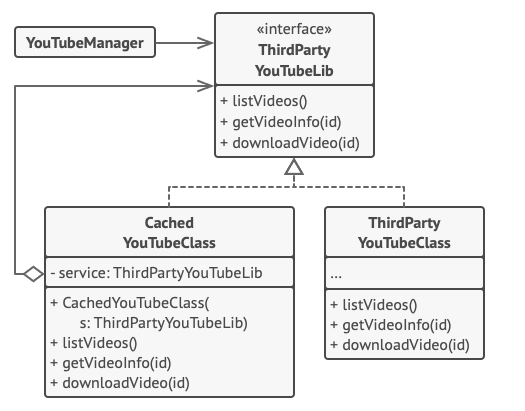
**Padrão de projeto PROXY**

Ao meu entendimento o uso desse padrão é interessante, ele se passa pelo objeto que deveria ser referido diretamente, um verdadeiro impostor, mas com boas intenções.

**1\* O termo PROXY é utilizado para definir quem é intermediário, entre o usuário e o servidor. A palavra proxy significa procuração.**

Quando existe a necessidade de instanciar um objeto grande ou complexo demais, usa-se esse padrão, ele permite usar um espaço reservado para outro objeto e controla o acesso a ele, neste caso o objeto só irá tratar da regra de negócio.

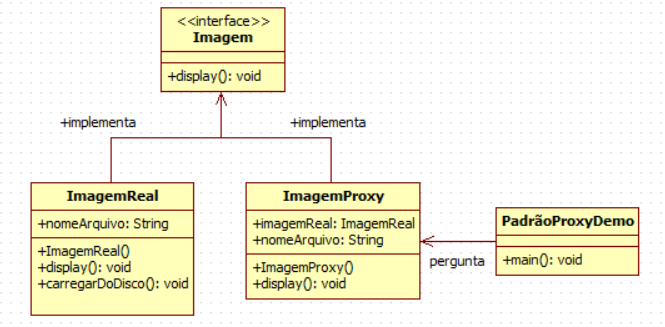
**2\* A intenção do padrão é se passar pelo objeto que deveria ser instanciado, ele fornece um substituto.**

**** Na imagem abaixo retirada do site refactoring.guro tem o exemplo de um pseudocódigo, onde mostra a atuação do proxy junto ao objeto e em contato com a interface.

Nesse exemplo é visto que ao solicitar um vídeo do youtube, as informações ficam salva em cache onde é reutilizada varias vezes caso o usuário necessite.

A classe proxy faz todo o trabalho que deveria ser da classe original, ou seja o impostor (proxy) faz um serviço rápido devido já possuir as informações solicitadas anteriormente.

**3\* Esse padrão deve ser usado sempre que houver a necessidade de criar um objeto grande ou complexo.**

****

Nesta imagem é possível ver a organização da estrutura da implementação do uso do proxy, o proxy tem implementado em si a classe ImagemReal que é passada para a <<interface>> Imagem.

Exemplo do código extraído da wikipedia.

**public** **interface** **Image**{

void display();

}

**public** **class** **RealImage** **implements** Image {

**private** String fileName;

**public** RealImage(String fileName){

**this**.fileName = fileName;

loadFromDisk(fileName);

}

@Override

**public** void display() {

System.out.println("Displaying " + fileName);

}

**private** void loadFromDisk(String fileName){

System.out.println("Loading " + fileName);

}

}

**public** **class** **ProxyImage** **implements** Image{

**private** RealImage realImage;

**private** String fileName;

**public** ProxyImage(String fileName){

**this**.fileName = fileName;

}

@Override

**public** void display() {

**if**(realImage == **null**){

realImage = **new** RealImage(fileName);

}

realImage.display();

}

}

**public** **class** **ProxyPatternDemo** {

**public** **static** void main(String args) {

Image image = **new** ProxyImage("test\_10mb.jpg");

*//image will be loaded from disk*

image.display();

System.out.println("");

*//image will not be loaded from disk*

image.display();

}

}

**Saidas**

Loading test\_10mb.jpg

Displaying test\_10mb.jpg

Displaying test\_10mb.jpg

**As Desvantagens do uso desse padrão de projeto** é que o código pode ficar mais complicado tendo em vista que mais classes serão criadas além de respostas de um serviço poder demorar mais.