

**Software
WebQuest
Padrão Fachada/Façade/Facade
Refatoração/LoD/GRASP/Design Patterns**

Nomes da Dupla: Ítalo Tabatinga e Dicksiano Melo
[CONHECIMENTO]

O Padrão Façade

Em padrões de projeto de software, uma fachada – fachada em francês/inglês, e é uma das poucas palavras com “ç” em inglês – é um objeto que disponibiliza uma interface para uma grande quantidade de funcionalidades de uma API, por exemplo. Um façade permite o seguinte:

- Tornar uma biblioteca de software mais fácil de entender e usar.
- Tornar o código que utiliza esta biblioteca mais fácil de entender.
- Reduzir as dependências em relação às características internas de uma biblioteca, trazendo flexibilidade no desenvolvimento do sistema.
- Envolver uma interface mal desenhada com uma interface melhor definida.

Uma façade é um padrão de projeto (design pattern) do tipo estrutural. Os façades são muito comuns em projetos orientados a objeto. Por exemplo, a biblioteca padrão da linguagem Java contém dúzias de classes para processamento do arquivo fonte de um caractere, geração do seu desenho geométrico e dos pixels que formam este caractere. Entretanto, a maioria dos programadores Java não se preocupam com esses detalhes, pois a biblioteca contém as classes do tipo façade (Font e Graphics) que oferecem métodos simples para as operações relacionadas com fontes.

Quando é usado?

- Criação de interfaces mais simples para um ou mais subsistemas complexos.
- Redução de dependência entre o cliente e as classes existentes nos subsistemas, ocasionando a redução da coesão do sistema.

- Criação de sistemas em camadas. Este padrão provê o ponto de entrada para cada camada (nível) do subsistema.

Estrutura do Padrão Façade: antes!

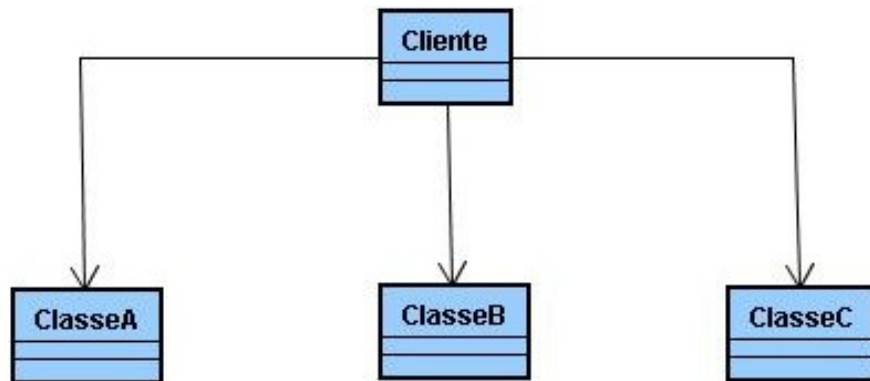


Fig. 1 - Exemplo relacionando a complexidade do acesso a várias classes

Estrutura do Padrão Façade: depois!

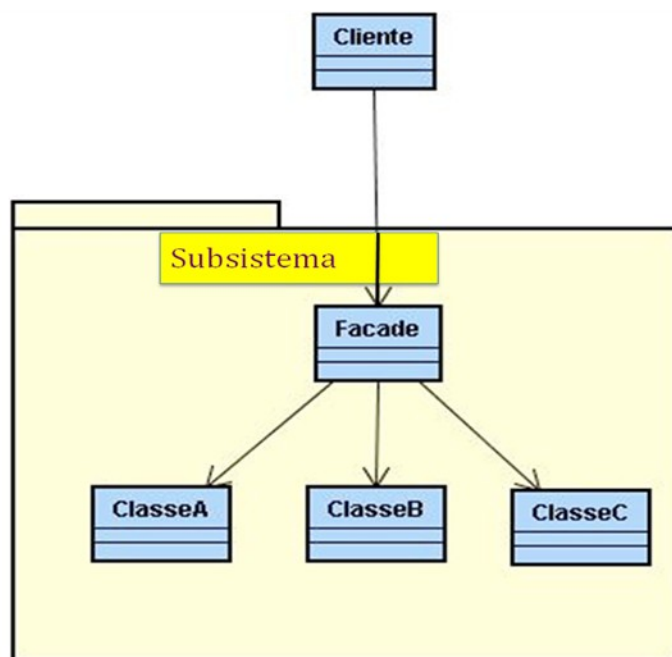
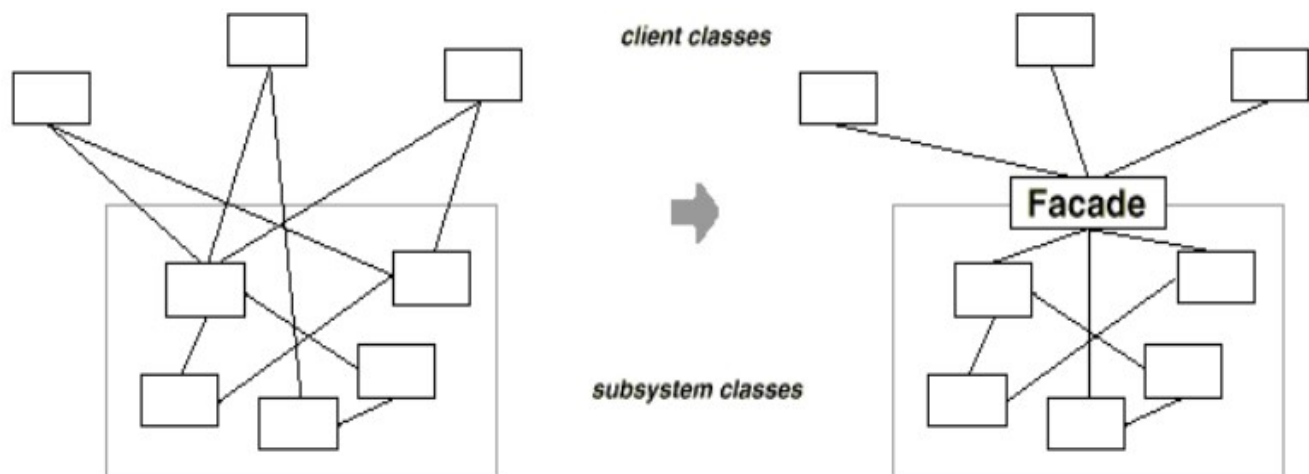


Figura 2: Diagrama UML apresentando uma possível estrutura do Facade.



Participantes

As classes e objetos que participam no padrão são os seguintes:

- Cliente: aguarda respostas da interação do Façade e as Classes do subsistema.
- Façade: conhece quais classes do subsistema seriam responsáveis pelo atendimento de uma solicitação e delega solicitações de clientes a objetos apropriados dos subsistemas.
- Classes de subsistema: (ClasseA, ClasseB, ClasseC ...)
- o Implementam as funcionalidades do subsistema.
- o Respondem a solicitações de serviços da Façade.
- o Não têm conhecimento da Façade.

Consequências

- Protege os clientes da complexidade dos componentes do subsistema.
- Promove acoplamento fraco entre o subsistema e seus clientes.
- Reduz dependências de compilação, possivelmente complexas ou circulares.
- Facilita a portabilidade do sistema.
- Reduz a união entre subsistemas desde que cada subsistema utilize seu próprio padrão Façade e outras partes do sistema utilizem o padrão Façade para comunicar-se com o subsistema.
- Não evita que aplicações possam acessar diretamente as subclasses do sistema, se assim o desejarem.

Responder: Quais dos princípios SOLID são promovidos por este DP? Forneça uma explicação sucinta para cada princípio promovido. [pode responder depois da implementação se sentir que a implementação ajudará a entender melhor]

1. Single Responsibility Principle - Uma vez que o padrão Façade separa a interação do cliente com os subsistemas, faz com que os subsistemas sejam responsáveis por suas atividades específicas sem interagir com o cliente.

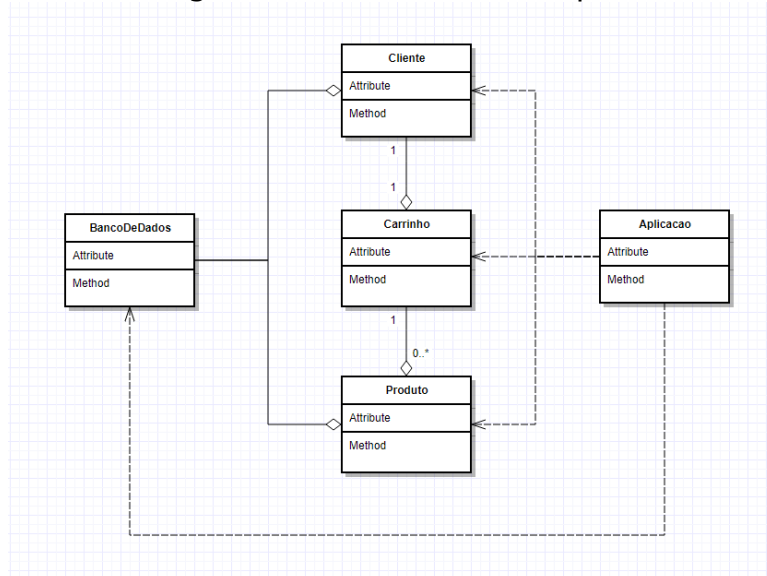
2. Open/Closed Principle - Uma vez que a interação com o usuário está definida, o projeto está fechado para modificações (iteração com o usuário definida) e aberto para extensões (modificações internas no subsistemas sem alterar a fachada).

[APLICAÇÃO] Duplas

Sejam as classes do subsistema Mercado Virtual, que está no package mercadoVirtual, e a classe Aplicacao, do package application, que faz uso dos serviços providos por objetos das classes do subsistema Mercado Virtual.

Questão 1 - Verificar como funciona e implementar o código dos dois pacotes que compõem o sistema exemplo. Fazer funcionar se for o caso!

Questão 2 - Desenhar o diagrama de classes correspondente da situação Antes!



Questão 3 - Qual é o mau cheiro que sai desse código em termos de acoplamento entre a classe do pacote application e as classes do pacote mercadoVirtual? Desconsiderar neste ponto a não aderência da classe application ao LoD e eventuais outros maus cheiros nas demais classes!

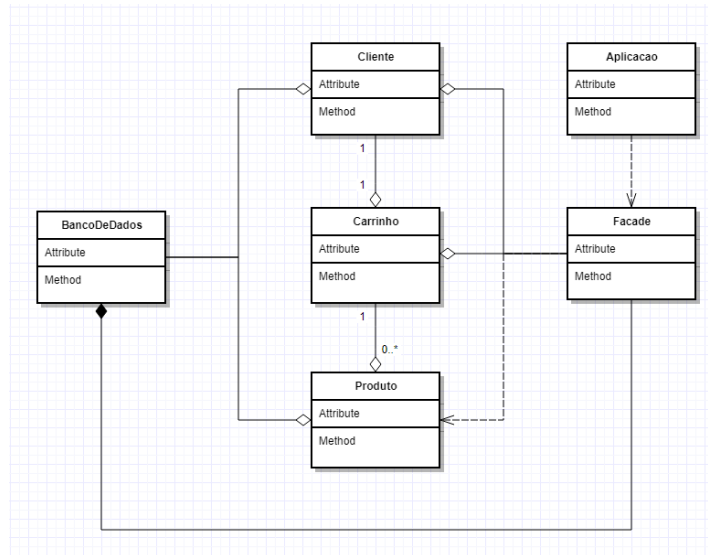
A Classe application depende de todas as classes de mercadoVirtual.

Questão 4 - Como fazer para eliminar esse mau cheiro?

Podemos implementar o design pattern Façade para que a Classe Application dependa apenas de uma classe API que tem todas as funções de mercadoVirtual, funcionando como a porta de um subsistema.

Questão 5 - Apresentar uma solução que faça uso da solução apontada na questão anterior!

Questão 6 - Desenhar o diagrama de classes correspondente da situação Depois!



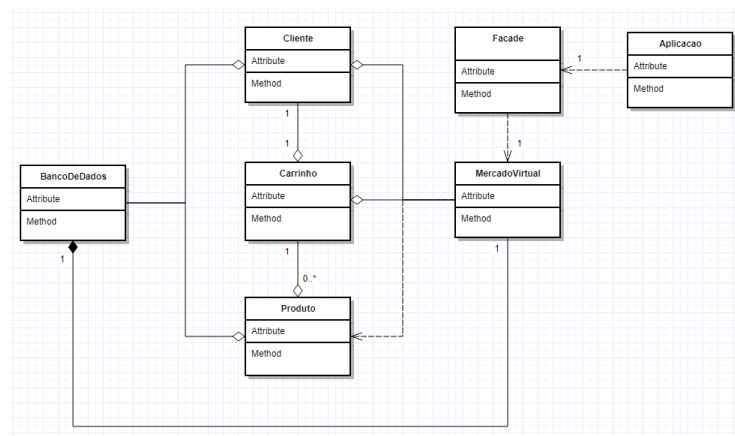
Questão 7 - Eliminar os métodos de fábrica estáticos usados e usar padrão de projeto Singleton Preguiçoso para a solução da Questão 5. Crie classes novas apenas se e quando for estritamente necessário!

O padrão de projeto Singleton foi utilizada para que exista apenas um objeto de Facade, dessa forma, como Facade possui o Banco De Dados, ele é o mesmo para qualquer instância ou acesso e só é instanciado uma vez.

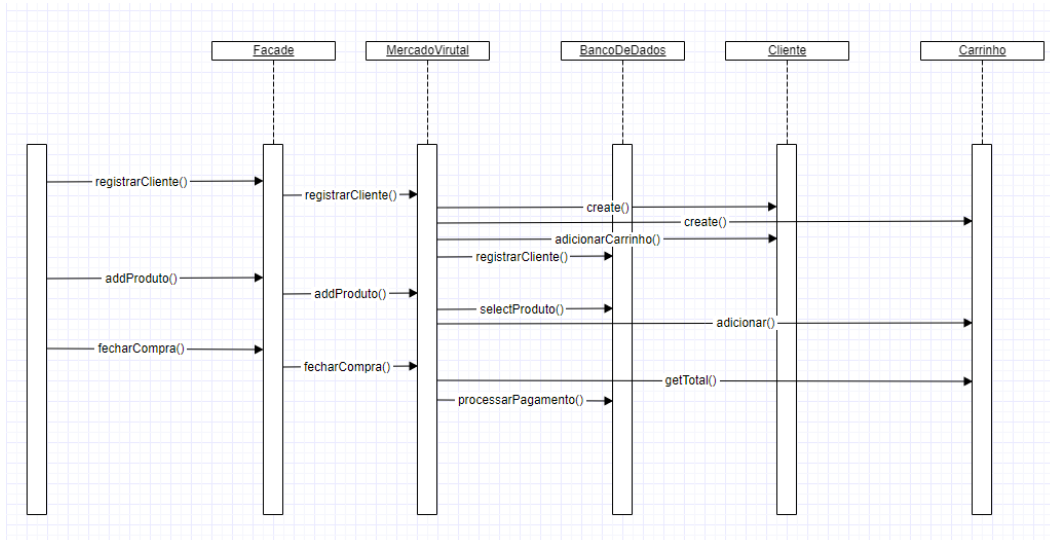
Questão 8 - Fazer com que cada um dos métodos da solução da Questão 5 sejam apenas delegações, caso ainda não sejam! Crie classes novas apenas se e quando for estritamente necessário!

Questão 9 - Examinar se ainda persiste algum mau cheiro que viole a LoD, princípios GRASP ou ficariam melhor usando algum padrão de projeto! Retire o mau cheiro e crie classes novas apenas se e quando for estritamente necessário!

Questão 10 - Desenhar o diagrama de classes correspondente da situação Novo Depois!



Questão 11 - Desenhar um diagrama de sequência para uma compra sendo efetuada a partir de objeto da classe Aplicacao.



Obs. 1: Colocar todas as respostas neste documento, inclusive os diagramas, que podem ser desenhados a mão, mas neste caso devem ser fotografados e ter suas imagens correspondentes inseridas na questão correta do texto! O melhor seria usar o site gliffy.com ou outro semelhante, para quem não tem editor de diagramas UML no seu notebook ou computador!

Obs. 2: Zipar documento (com códigos inseridos no texto) + códigos gerados em Q5, Q7, Q8 e Q9 no formato de texto simples; cada conjunto de pacotes/classes deve conter em cada resposta apenas as classes novas e antigas que mudaram em relação ao ponto anterior!

Obs. 3: Cada dupla deve então colocar o documento + códigos zipado na atividade correspondente do TIDIA, não esquecendo de colocar nomes dos componentes da dupla.

Package mercadoVirtual **Subsistema Mercado Virtual**

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
```

```
public class Carrinho {
    List<Produto> listaDeCompas;
    private Carrinho() {
        listaDeCompas = new ArrayList<>();
    }
    public static Carrinho create() {
        return new Carrinho();
    }
    public void adicionar(Produto p) {
        listaDeCompas.add(p);
    }
    public double getTotal() {
        double total = 0;
        for (Iterator<Produto> p = listaDeCompas.iterator(); p.hasNext();) {
            Produto produto = (Produto) p.next();
            System.out.println("Valor do item: " + produto.getPreco());
            total += produto.getPreco();
        }
        return total;
    }
}
```

```

    }
}
public class Produto {
    private String nome;
    private int id;
    private double preco;
    private Produto(String nome, int id, double preco) {
        this.nome = nome;
        this.id = id;
        this.preco = preco;
    }
    public static Produto create(String nome, int id, double preco) {
        return new Produto(nome, id, preco);
    }
    public double getPreco() {
        return preco;
    }
}
public class Cliente {
    private int id;
    private String nome;
    private Carrinho carrinho;
    private Cliente(String nome, int id) {
        this.id = id;
        this.nome = nome;
    }
    public static Cliente create(String nome, int id) {
        return new Cliente(nome, id);
    }
    public void adicionarCarrinho(Carrinho c) {
        this.carrinho = c;
    }
    public Carrinho getCarrinho() {
        return carrinho;
    }
    public int getId() {
        return id;
    }
}
import java.util.HashMap;
import java.util.Map;
public class BancoDeDados {
    Map<Integer, Produto> produtosById;
    Map<Integer, Cliente> clientesById;
    public BancoDeDados() {
        produtosById = new HashMap<>();
        clientesById = new HashMap<>();
        for (int i = 0; i < 500; i++) {
            produtosById.put(new Integer(i), Produto.create("Produto
#" + i, i, (double)(i*3 + 1)));
        }
    }
    public Cliente selectCliente(int id) {
        return clientesById.get(new Integer(id));
    }
    public Produto selectProduto(int id) {
        return produtosById.get(new Integer(id));
    }
    public void processarPagamento(Cliente cliente, double valor) {
        System.out.println("(Pagamento processado) Cliente: " +
cliente.getId() + ", Valor: " + valor);
    }
}

```

```
        public void registrarCliente(Cliente cliente) {  
            clientesById.put(cliente.getId(), cliente);  
        }  
    }  
}
```

Package Application

```
import q1.mercadovirtual.BancoDeDados;  
import q1.mercadovirtual.Carrinho;  
import q1.mercadovirtual.Cliente;  
import q1.mercadovirtual.Produto;  
public class Aplicacao {  
    public static void main(String[] args) {  
        BancoDeDados banco = new BancoDeDados();  
        // registrar comprador  
        Cliente cliente = Cliente.create("ZÈ", 123);  
        Carrinho car = Carrinho.create();  
        cliente.adicionarCarrinho(car);  
        banco.registrarCliente(cliente);  
        // realizar uma compra  
        Produto produto = banco.selectProduto(223);  
        cliente.getCarrinho().adicionar(produto);  
        // realizar outra compra  
        Produto produto2 = banco.selectProduto(342);  
        cliente.getCarrinho().adicionar(produto2);  
        // fechar compra  
        double valor = cliente.getCarrinho().getTotal();  
        banco.processarPagamento(cliente, valor);  
    }  
}
```