Relatório Técnico – Aplicação do Padrão Facade no Sistema de Pedidos (TiaLuDelivery)

Alex Oliveira[1], Alice Aragão[2], Ana Clara Ribeiro[3], Monique Prado[4]

Sistemas da Informação – Centro Universitário de Excelência UNEX - Vitória da Conquista – Bahia – Brasil

Abstract. *This paper presents the refactoring of a food ordering system using the Facade design pattern. The approach aims to simplify access to complex subsystems by centralizing operations related to customers, menu items, and orders. The results show improvements in code readability, maintainability, and system scalability.*

Resumo. *Este trabalho apresenta a refatoração de um sistema de pedidos utilizando o padrão de projeto Facade. A abordagem visa simplificar o acesso a subsistemas complexos, centralizando as operações relacionadas a clientes, itens de menu e pedidos. Os resultados evidenciam melhorias na legibilidade, manutenibilidade e escalabilidade do código.*

1. Introdução

Este trabalho tem como objetivo apresentar a refatoração de um sistema de pedidos utilizando o padrão de projeto Facade, no contexto da disciplina de Técnicas e Práticas Ágeis de Desenvolvimento de Software.

A refatoração de código com o uso de padrões de projeto busca organizar (Fowler, 2018), simplificar e tornar mais compreensível a manutenção do sistema, além de resolver problemas recorrentes no desenvolvimento de software, como duplicidade de lógica, dificuldade de evolução e acoplamento excessivo entre classes.

O padrão Facade é amplamente utilizado (Gamma et al., 1995) para fornecer uma interface unificada que simplifica o acesso a subsistemas complexos, oferecendo uma maneira mais clara e direta de interagir com o sistema. No caso do TiaLuDelivery, a refatoração foi aplicada para centralizar as operações de clientes, itens de menu e pedidos em uma única classe responsável por intermediar todas as interações.

2. Fundamentação Teórica

2.1 O problema que o padrão Facade resolve

Em sistemas sem o uso de Facade (Freeman et al., 2004), o desenvolvedor precisa conhecer os detalhes internos e interagir diretamente com diversas classes (como Customer, Order, MenuItem, OrderItem), aumentando o acoplamento e reduzindo a manutenibilidade do código.

O padrão Facade resolve esse problema fornecendo uma camada de abstração que encapsula as interações mais comuns, facilitando o uso do sistema e reduzindo a complexidade percebida.

2.2 Diagrama de Classe (conceitual do padrão)

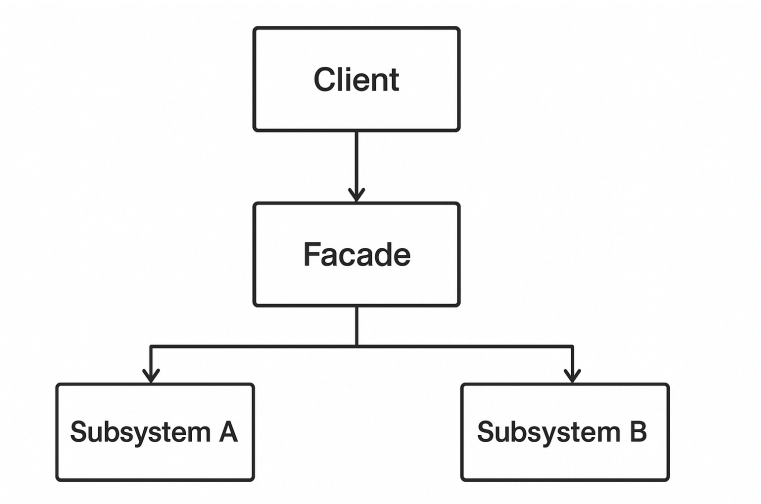

Figura 1. Um diagrama conceitual

2.3 Exemplo Facade

```
1  public class SistemaFacade {
2
3      private SubsistemaA a;
4      private SubsistemaB b;
5
6      public SistemaFacade() {
7          new SubsistemaA(;
8          new SubsistemaB(;
9      }
10     public void executarProcesso() {
11         a.acao();
12         b.acao();
13     }
```

Figura 2. Exemplo genérico de Facade em JAVA

2.4 Vantagens do padrão Facade

- Reduz a complexidade do sistema ao fornecer uma interface única para operações comuns.
- Diminui o acoplamento entre clientes e subsistemas internos.
- Facilita a manutenção e evolução do código.
- Centraliza operações recorrentes, promovendo consistência no uso do sistema.

2.5 Desvantagens do padrão Facade

- Pode gerar um ponto único de falha caso a interface não seja bem projetada.
- Pode esconder detalhes importantes do funcionamento interno, dificultando personalizações.
- Em projetos muito grandes, pode se tornar um "Deus Objeto" se concentrar responsabilidades demais.

3. Metodologia

O desenvolvimento seguiu as seguintes etapas:

- Análise do código existente e identificação da necessidade de simplificação do acesso às operações de clientes, itens de menu e pedidos.
- Implementação do padrão Facade por meio da classe **SistemaPedidosFacade**, responsável por centralizar as operações em uma única interface.
- Organização em pacotes, respeitando a estrutura **br.edu.unex.tiaLuDelivery.facade** e **br.edu.unex.tiaLuDelivery.model**.
- Testes locais: foram simulados cadastros de clientes, adição de itens de menu, criação de pedidos e mudança de status.
- Comparativo: em relação ao código sem Facade, a solução ficou mais simples, pois o desenvolvedor interage apenas com a classe **SistemaPedidosFacade**, em vez de manipular diretamente múltiplas classes.

4. Resultados

4.1 Diagrama de Classe

O diagrama mostra a classe SistemaPedidosFacade centralizando a gestão de clientes, itens de menu e pedidos. As classes Customer, MenuItem, Order e OrderItem representam os subsistemas internos, enquanto OrderStatus define os estados possíveis de um pedido. A Facade simplifica a interação com o sistema, reduzindo o acoplamento e melhorando a legibilidade.

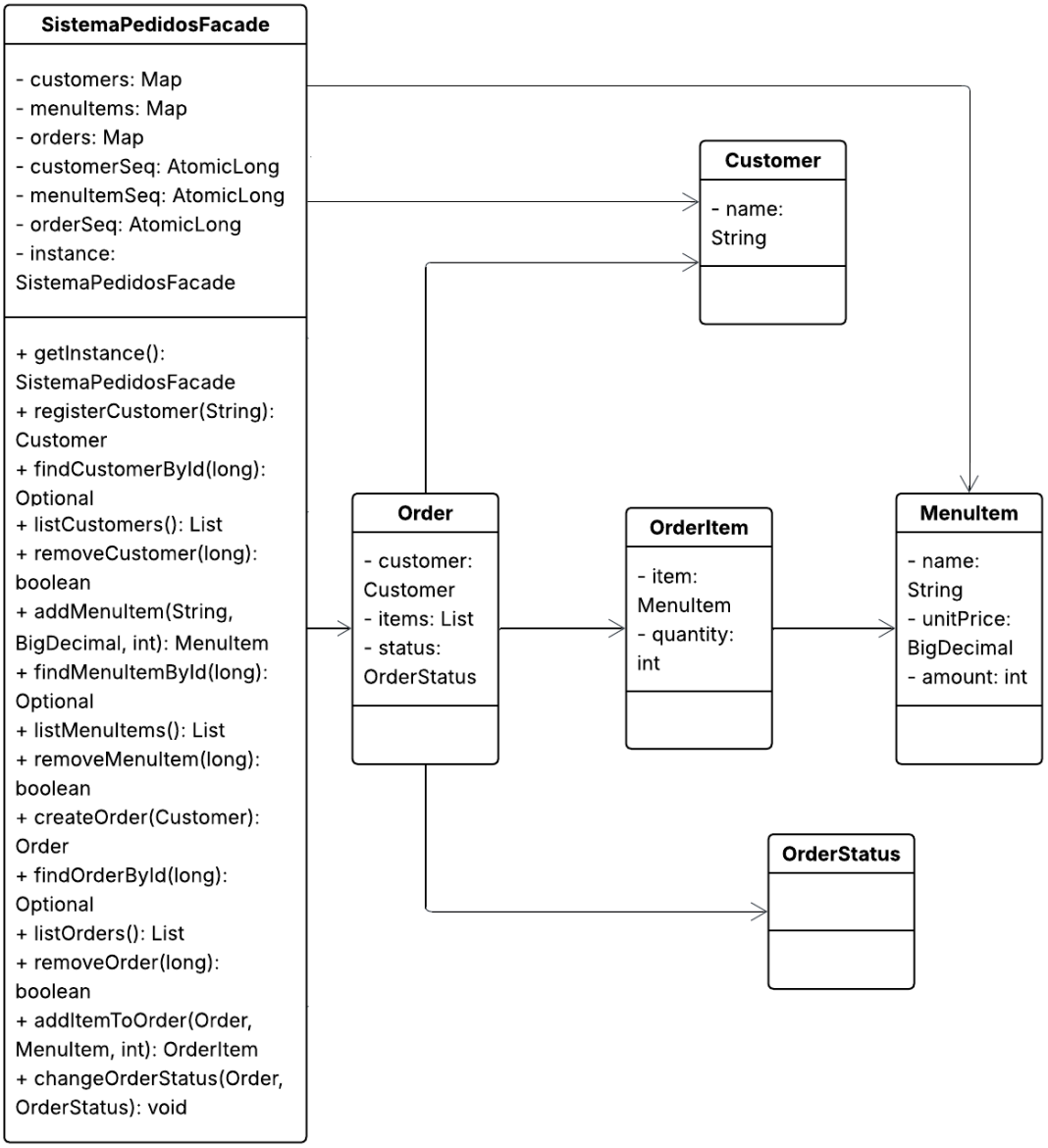

Figura 3. Diagrama de Classe UML

4.2 Código Desenvolvido (Simplificado)

A figura abaixo mostra parte do código que aplica o padrão Facade no sistema TiaLuDelivery. A Facade centraliza as operações de cadastro de clientes, adição de itens de menu e gerenciamento de pedidos, permitindo criar pedidos, adicionar itens e alterar o status de forma simples, sem que seja necessário acessar diretamente as classes internas.

```java
public class SistemaPedidosFacade {
    // Cadastro de clientes, itens e pedidos
    private List<Customer> customers = new ArrayList<>();
    private List<MenuItem> menuItems = new ArrayList<>();
    private List<Order> orders = new ArrayList<>();

    // Clientes
    public Customer registerCustomer(String name) {
        Customer c = new Customer(name);
        customers.add(c);
        return c;
    }
    // Itens de menu
    public MenuItem addMenuItem(String name, double price) {
        MenuItem mi = new MenuItem(name, price);
        menuItems.add(mi);
        return mi;
    }
    // Pedidos
    public Order createOrder(Customer customer) {
        Order order = new Order(customer);
        orders.add(order);
        return order;
    }
    public void addItemToOrder(Order order, MenuItem item, int quantity) {
        order.addItem(new OrderItem(order, item, quantity));
    }
    public void changeOrderStatus(Order order, OrderStatus status) {
        order.setStatus(status);
    }
}
```

Figura 4. Código simplificado explicativo do padrão implementado

4.3 Vantagens da Solução

- Centraliza a lógica do sistema em uma única classe.
- Facilita a manutenção e o entendimento do código.
- Reduz o acoplamento entre classes.
- Permite evoluir o sistema sem alterar a forma de interação com os usuários da API.

4.4 Desvantagens da Solução

- A classe Facade pode acumular muitas responsabilidades se o sistema crescer muito.
- Alterações nas classes internas podem exigir mudanças na Facade.
- Pode esconder detalhes importantes do funcionamento interno, dificultando customizações avançadas.

5. Considerações Finais

O uso do padrão Facade mostrou-se eficaz na simplificação do sistema de pedidos, proporcionando uma interface única de acesso às operações mais importantes e facilitando a manutenção do código.

Como melhorias futuras, o sistema poderá:

- Implementar tratamento de exceções mais robusto.
- Adotar testes automatizados para validar o comportamento do Facade.
- Integrar um mecanismo de persistência com banco de dados, utilizando padrões como DAO (Data Access Object).

A experiência reforçou a importância dos padrões de projeto como ferramentas de refatoração que garantem **qualidade, simplicidade e escalabilidade** em sistemas orientados a objetos.

7. Referências

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

Freeman, E., Freeman, E., Bates, B., & Sierra, K. (2004). Head First Design Patterns. O'Reilly.

Fowler, M. (2018). Refactoring: Improving the Design of Existing Code. Addison-Wesley.