

fora do padrão!!

Sistema de Gestão de Pedidos com Padrões de Projeto: Uma Abordagem Orientada a Objetos

~~Autores:~~ Joaqson Rodrigues, Juan Pablo, Lucas Santos, Matheus Santos, Rafael Pires

~~Instituição:~~ Centro Universitário de Excelência – UNEX

R. Ubaldino Figueira, 200 - Recreio, Vitória da Conquista - BA, 45020-510

Resumo. Este relatório detalha o processo de desenvolvimento de um sistema de gerenciamento de pedidos via linha de comando, utilizando a linguagem Java. O projeto foi estruturado com base nos princípios da Programação Orientada a Objetos, com ênfase na separação de responsabilidades e na aplicação de padrões de projeto como Singleton, Strategy e Observer para resolver desafios de arquitetura. O resultado é um sistema coeso e extensível que atende a todos os requisitos funcionais e de design propostos, centralizando a gestão de dados, permitindo a geração flexível de relatórios e notificando partes interessadas sobre eventos relevantes.

1. Introdução

A Programação Orientada a Objetos (POO) é um paradigma fundamental no desenvolvimento de software moderno, pois promove a criação de sistemas mais eficientes, escaláveis e de fácil manutenção. Sua principal força reside na capacidade de estruturar o código em classes e objetos, que modelam entidades e conceitos do mundo real. Pilares como o encapsulamento, que protege os dados internos de um objeto e expõe apenas o necessário, aumentam a segurança e a robustez do sistema, evitando manipulações indevidas.

A modularidade inherente à POO, onde cada classe funciona como um componente independente, simplifica a manutenção e a evolução do software, pois alterações em uma parte do sistema têm menor probabilidade de impactar outras. Este paradigma é a base de linguagens amplamente utilizadas como Java, C++ e Python, sendo aplicado em uma vasta gama de aplicações, desde sistemas corporativos complexos até jogos digitais, demonstrando sua versatilidade e poder para impulsionar a inovação tecnológica.

2. Fundamentação Teórica

O sistema foi construído com base nos conceitos essenciais da Programação Orientada a Objetos: **Classes, atributos, construtores e métodos.**

- **Classes:** São os moldes, ou modelos, que definem a estrutura (atributos) e o comportamento (métodos) dos objetos do sistema. No projeto, **Cliente**, **Pedido** e **ItemCardapio** são exemplos de classes de domínio.
- **Atributos:** São as variáveis declaradas dentro de uma classe que representam as características ou o estado de um objeto. Por exemplo, **nome** e **preco** na classe **ItemCardapio**.
- **Construtores:** Métodos especiais responsáveis por inicializar um objeto no momento de sua criação, garantindo que ele nasça em um estado consistente e válido.
- **Métodos:** Representam as ações ou comportamentos que um objeto pode executar. O método **avancarStatus()** na classe **Pedido** é um exemplo de comportamento.
- **Diagrama de Classes:** Uma representação visual da estrutura estática de um sistema, que exibe as classes, seus atributos, métodos e os relacionamentos entre elas, servindo como um blueprint da arquitetura.

3. Metodologia

O desenvolvimento do sistema seguiu uma abordagem orientada a objetos, com foco na clara separação de responsabilidades. A aplicação foi construída em Java e opera via interface de linha de comando (CLI).

A arquitetura foi estruturada da seguinte forma:

1. **Camada de Domínio:** Composta pelas classes que modelam o negócio: **Cliente**, **ItemCardapio**, **Pedido** e **StatusPedido**. A relação de muitos-para-muitos entre **Pedido** e **ItemCardapio** foi solucionada pela classe de associação **ItemPedido**. → *o que é uma classe de associação?*
2. **Padrão Singleton para Gestão de Dados:** Para atender ao requisito de um ponto de acesso único e centralizado para os dados, foi criada a classe **CentralDeDados** e implementado o padrão de projeto **Singleton**. Isso garante uma única instância da classe em toda a aplicação, funcionando como um "banco de dados" em memória e evitando inconsistências.
3. **Padrão Strategy para Relatórios:** Para atender ao requisito de "relatórios flexíveis", foi implementado o padrão de projeto **Strategy**. Foi criada a interface **RelatorioStrategy** e duas implementações concretas (**RelatorioVendasSimplificado** e **RelatorioVendasDetalhado**), desacoplando a lógica de geração de relatórios da classe de interface.

*Não deveria
estar na
fundamenta-
ção.*

*OPA que conceitos como
herança e polimorfismo não devria
ser usado.*

*ura
arquite-
tura
Santu
image
expli-
cando*

*deveria
ter na
funda-
menta-
ção.*

*→ interfa-
ce não dev-
eria ser
usada.*

- Não era para sermos interlocutores, heranças e polymorfismo.*
- Padrão Observer para Notificações:** Para atender ao requisito de "notificações futuras", foi implementado o padrão de projeto **Observer**. A classe **Pedido** foi transformada em um "Subject" (Observado), e a interface **Observer** foi criada para ser implementada por classes "Listeners", como **GerenteNotificador** e **LogNotificador**. Isso permite que a classe **Pedido** notifique outras partes do sistema sobre mudanças de estado sem conhecê-las diretamente, garantindo alta flexibilidade.
 - Camada de Interface com o Usuário:** A classe **AplicacaoCLI** atua como o ponto de entrada e o controlador da interface, orquestrando as chamadas para a **CentralDeDados** e registrando os **Observers** nos novos **Pedidos**.

4. Resultados e Discussões

queis foram esses requisitos?

A arquitetura final resultou em um sistema coeso, funcional e que atende a todos os requisitos propostos. O uso de padrões de projeto não apenas solucionou os desafios de design, mas também tornou o código mais limpo, organizado e preparado para futuras extensões. O diagrama de classes abaixo ilustra a estrutura final da solução:

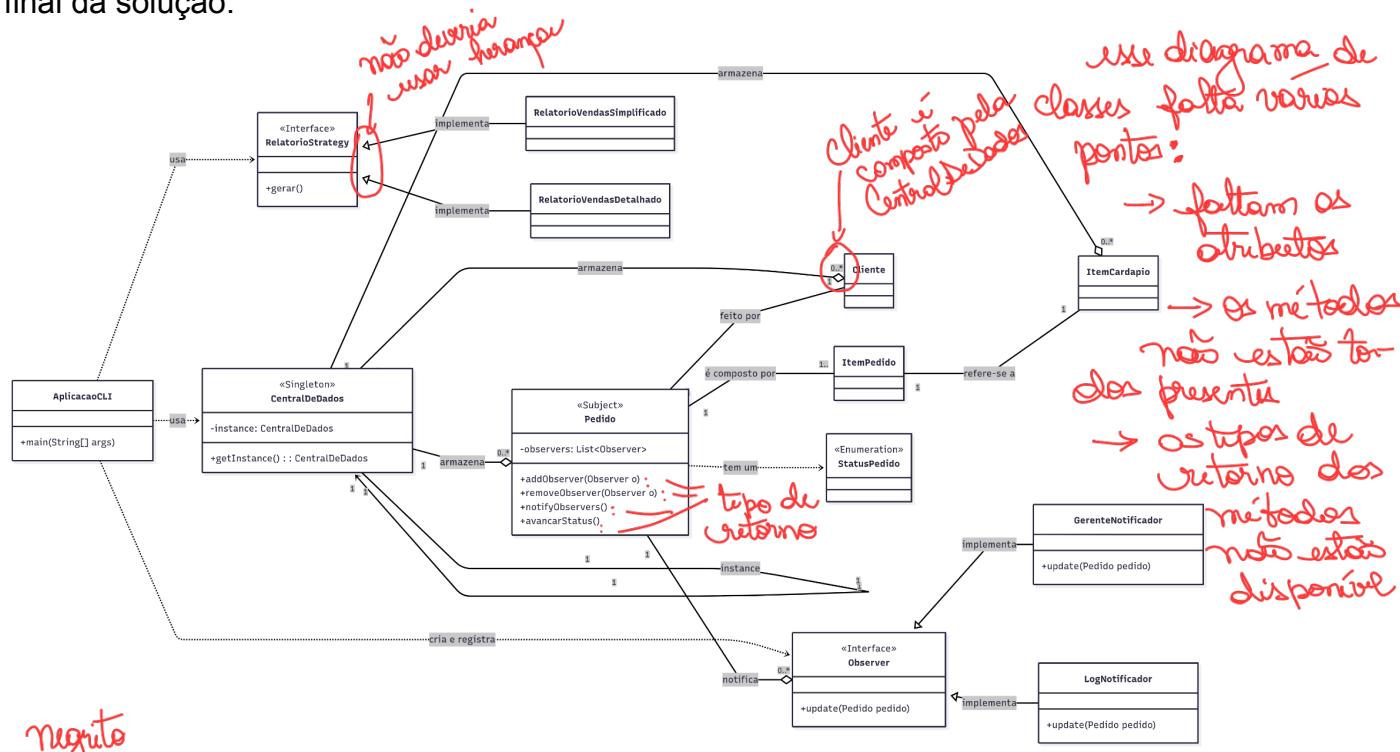


Figura 1. Diagrama de classes final do sistema, incluindo os padrões Singleton, Strategy e Observer. Fonte:

O diagrama evidencia a completa separação de responsabilidades. A **AplicacaoCLI** orquestra o fluxo, a **CentralDeDados** garante a consistência dos dados, o padrão Strategy permite relatórios intercambiáveis, e o padrão Observer

possibilita um sistema de notificações desacoplado e extensível. Essa abordagem se mostrou altamente eficaz e alinhada com as melhores práticas da POO.

5. Considerações Finais

O desenvolvimento deste projeto foi uma experiência prática valiosa sobre modelagem de sistemas com POO. O maior desafio foi traduzir os requisitos em uma estrutura de classes coesa, decidindo onde cada responsabilidade deveria residir.

A aplicação dos padrões de projeto foi um ponto de destaque. O Singleton resolveu de forma elegante o gerenciamento do estado global. O Strategy atendeu diretamente ao requisito de flexibilidade para os relatórios. Finalmente, a implementação do padrão Observer para o sistema de notificações cumpriu o requisito de desacoplamento, permitindo que a classe Pedido notifique partes interessadas sem conhecê-las, o que representa uma solução robusta e de fácil manutenção.

O projeto, em sua versão final, não apenas cumpre os requisitos funcionais, mas também demonstra uma arquitetura sólida e bem planejada, pronta para futuras evoluções.

6. Referências

[1] GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. *Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos*. Porto Alegre: Bookman, 2000.

[2] PRESSMAN, R. S. *Engenharia de Software: Uma Abordagem Profissional*. 8. ed. Porto Alegre: AMGH, 2016.

[3] FOWLER, M. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3. ed. Addison-Wesley Professional, 2003.

Observações Gerais:

Primeiro, eu gostaria de parabenizá-los pelo esforço de escrita, apresentação e desenvolvimento. Sei que foi desafiador, mas espero que tenha colaborado para o desenvolvimento técnico de vocês.

Fazendo algumas observações gerais:

1. O texto apresenta alguns problemas de formatação. Eles podem ser: fonte, tamanho, alinhamento de parágrafos, dentre outros.

2. A escrita resou pouca, se nenhuma, citando as referências. Referências devem ser adicionadas quando citadas no texto. Elas colaboram com o seu desenvolvimento, pois reforçam que nenhum conhecimento surge do nada, mas é apurado no trabalho colívo. Procurem exemplos de citações direta e indireta. No próximo trabalho vai ser fundamental.

3. A seção de introdução deve contextualizar o problema, falar o que foi proposto, levar o leitor a entender brevemente o que o autor espera no decorrer do texto. Sempre tente fazer a introdução seguindo esse esquema:

→ Qual o contexto/problema?

→ O que era esperado que fosse feito?

→ Dá uma visão geral de como você pensou e fez.

→ Comentar o que terá nos próximos meses.

4. A seção de fundamentação pouco expõe os conceitos. Nessa seção vocês devem explicar tudo que o leitor precisa saber de conceitos técnicos novos para compreender a sua solução. Você们 deveriam ter:

- apresentando o diagrama de classes
- exemplo de código java diferente desses diagramas.
- falando dos tipos de relacionamentos e os mapeamentos de um relacionamento em atributos no código.

5. A seção de metodologia deve focar no como foi feita a solução, o passo a passo. Nessa seção, é interessante colocar as justificativas de escolhas de projetos, mas o resultado delas deve ficar na seção de resultados.

6. Aplicação ou explicação de conceitos que foram expressamente defendidos para não serem utilizados nesse primeiro problema/oft.