IMPLEMENTAÇAO DAS CLASSES, OBJETOS E RELACIONAMENTOS EM JAVA

Elisson Bastos O. M. Junior[1], Rafael H. Bordini[2], Flávio Rech Wagner[1], Jomi F. Hübner[3]

Centro Universitário de Excelência

Sistemas de Informação

Abstract. *This paper documents the development of a Java-based application, serving as a practical exploration of fundamental Object-Oriented Programming (OOP) concepts. The work begins by contextualizing the relevance of OOP as a paradigm for structuring modern, complex, and maintainable software. The theoretical foundation meticulously details the core constructs of OOP—such as classes, attributes, constructors, methods—and introduces UML Class Diagrams as an modeling tool. The methodology section describes the hands-on implementation process of the project, outlining the translation of a conceptual model into functional Java code. The results present the finalized Class Diagram of the solution and highlight key implementation snippets that exemplify the applied concepts. Finally, the concluding reflections discuss the encountered challenges, the most insightful aspects of the development journey, and propose potential future improvements and alternative approaches, providing a comprehensive and honest account of the learning experience.*

Resumo. *Este artigo relata o desenvolvimento de um sistema em Java para explorar os princípios da Programação Orientada a Objetos (POO). Apresenta uma fundamentação teórica sobre classes, atributos, métodos e diagramas seguida pela metodologia de implementação prática. Os resultados incluem o diagrama de classes da solução e trechos de código relevantes. As considerações finais refletem sobre os desafios enfrentados, insights adquiridos e melhorias potenciais, destacando a importância do design orientado a objetos para criar software maintainable e escalável. O projeto demonstra a aplicação eficaz dos conceitos teóricos de POO em um contexto prático.*

1. Introdução

A Programação Orientada a Objetos (POO) consolida-se como um paradigma fundamental no desenvolvimento de software moderno, oferecendo um modelo estruturado e intuitivo para abstrair entidades do mundo real em representações computacionais, esses objetos em questão refere-se a uma classe que passa a existir de instância com atributos e métodos definidos, e através dessas capacidade de organizar sistemas complexos através de conceitos como encapsulamento, herança e polimorfismo se cria um código reutilizável de fácil manutenção.

2. Fundamentação teórica

A estruturação fundamental da programação orientada a objetos se solidifica em torno do conceito central de classe, uma abstração que funciona como um modelo para a criação de objetos. Cada objeto é uma instância concreta de uma classe, possuindo um estado individual definido por seus atributos e um comportamento determinado por seus métodos, os quais operam sobre esses atributos. Para garantir que um objeto seja criado em um estado inicial válido, invoca-se um construtor, método especial responsável pela sua inicialização no momento da instanciação. Essa estrutura é governada por princípios fundamentais como o encapsulamento, que agrupa dados e comportamento em uma unidade coesa; a herança, que promove o reuso de código através de hierarquias; e o polimorfismo, que permite a objetos distintos responderem à mesma mensagem de formas diferentes. Para planejar e documentar visualmente essa arquitetura, utiliza-se o Unified Modeling Language (UML), que serve como uma planta ao representar graficamente as classes, seus componentes e os relacionamentos entre elas.

3. Metodologia.

A arquitetura do sistema foi concebida seguindo os princípios de orientação a objetos e separação de preocupações. Inicialmente, desenvolveu-se um diagrama de classes UML para modelar o domínio do problema, resultando nas entidades principais: Cliente (contendo dados cadastrais), ItemCardapio (representando os produtos do cardápio) e Pedido (gerenciando as transações). Para resolver o relacionamento muitos-para-muitos entre pedidos e itens, introduziu-se a classe associativa ItemPedido, responsável por armazenar a quantidade específica de cada item em um pedido.

Para persistência em memória, implementou-se a classe BancoDeDados seguindo o Padrão Singleton, garantindo uma única instância global para armazenamento das coleções de objetos. A lógica de negócio foi centralizada na camada de serviço através da classe FoodDeliveryFacade, que implementa o Padrão Facade para fornecer uma interface unificada e simplificada para operações de cadastro, gestão de pedidos e relatórios.

A interface com o usuário foi desenvolvida na classe AplicacaoCLI, responsável pela apresentação e captura de dados. O fluxo da aplicação inicia na classe Main, que coordena a inicialização do sistema. Esta abordagem em camadas (domínio, persistência, serviço e apresentação) promoveu um baixo acoplamento e alta coesão, facilitando a manutenção e possibilitando futuras evoluções do sistema.

4. Resultados

O sistema FoodDelivery foi implementado com sucesso, atendendo integralmente aos requisitos funcionais estabelecidos. A implementação demonstrou a eficácia da arquitetura em camadas adotada, com destaque para o funcionamento robusto do cadastro de clientes, que valida a obrigatoriedade do nome e telefone enquanto gera identificadores únicos automaticamente. O gerenciamento do cardápio mostrou-se eficiente, garantindo a integridade dos dados através de regras de validação que impedem preços inválidos ou nomes vazios, utilizando geração sequencial de códigos. O módulo de registro de pedidos operou com precisão, criando associações consistentes entre clientes e itens. Estes resultados validam as escolhas de design e implementação, comprovando que a solução atende aos objetivos de confiabilidade, usabilidade e consistência dos dados.

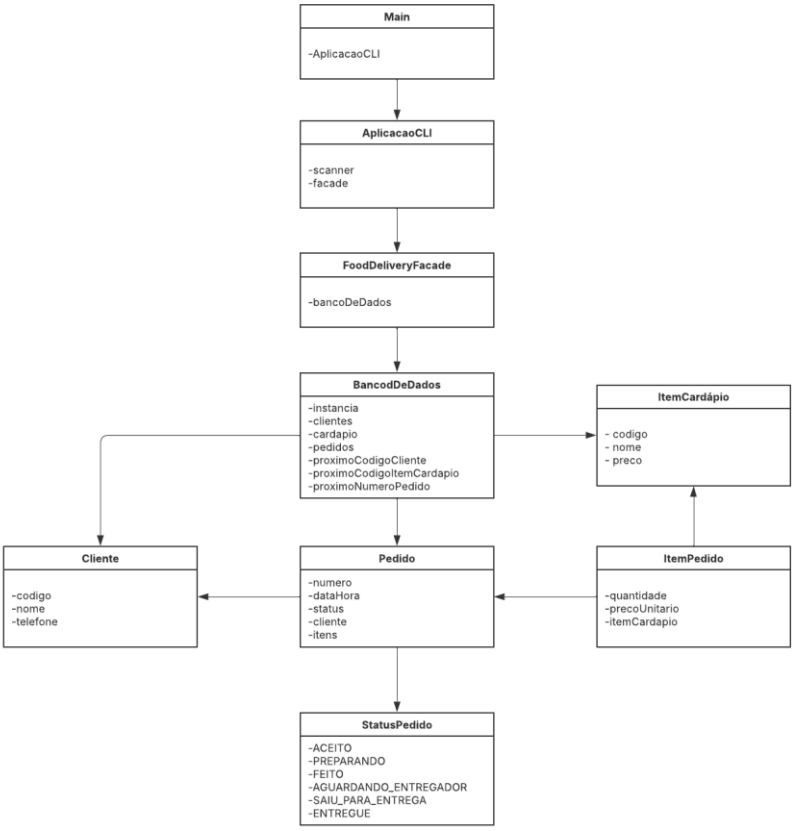

Figura 1. Diagrama de Classes

5. Considerações finais

O desenvolvimento do sistema FoodDelivery representou um valioso exercício prático na aplicação dos princípios da orientação a objetos e de padrões de projeto. Entre os aspectos mais desafiadores, destacam-se a modelagem do relacionamento muitos-para-muitos entre pedidos e itens, que exigiu a criação da classe associativa ItemPedido para manter a consistência dos dados, e a implementação rigorosa do controle de transições de estado dos pedidos, garantindo que nenhum pedido siga um fluxo inválido.

Do ponto de vista arquitetural, a adoção dos padrões de projeto Singleton e Facade foi uma decisão acertada, resultando em um sistema com baixo acoplamento e uma clara separação de responsabilidades, o que simplifica sua manutenção e futuras evoluções.

Embora o protótipo atual seja funcional, a prioridade seria a substituição do repositório em memória por um sistema de persistência de dados robusto, utilizando um banco de dados relacional como PostgreSQL ou MySQL. Em paralelo, seria implementado um tratamento de exceções mais granular para aumentar a resiliência do sistema a falhas e por fim, a experiência do usuário seria enriquecida com a criação de uma interface gráfica moderna.

References

DEITEL, P. J.; DEITEL, H. M. Java: Como Programar. 10ª. ed. São Paulo: Pearson, 2017.

SOUZA, M. et al. Use a Cabeça! Java. 3ª. ed. Rio de Janeiro: Alta Books, 2022. LEWIS, J.; LOFTUS, W. Java Software Solutions: Foundations of Programming Design. 6. ed. [S.l.]: Pearson Education Inc., 2008. ISBN 978-0-321-53205-3. Seção 1.6, "Object-Oriented Programming".

WIKIPÉDIA. Linguagem de programação. Disponível em: https://pt.wikipedia.org/wiki/Linguagem_de_programa%C3%A7%C3%A3o. Acesso em: 28 agosto. 2025.