Desenvolvimento WEB em POO – FoodDelivery

Erica A. de Jesus, Nalbert S. Santana, Mickael C. Alencar, Monyc L. A. Cerqueira, Pedro César. P. Jesus

Curso de Sistemas de Informação - Centro Universitário de Excelência (UNEX)
Av. Artêmia Pires Freitas, s/n - Sim - 44085-370 - Feira de Santana - BA - Brasil

erica.jesus2@aluno.unex.edu.br, nalbert.santana@aluno.unex.edu.br, mickael.alencar
monyc.cerqueira@aluno.unex.edu.br, cesar.jesus2@aluno.unex.edu.br

Introdução

A evolução dos sistemas de informação e o aumento da complexidade das aplicações digitais exigiram metodologias mais produtivas e sustentáveis para o desenvolvimento de software. Nesse contexto, a Programação Orientada a Objetos (POO) consolidou-se como um dos paradigmas mais importantes, pois possibilita maior produtividade por meio do reuso de código, além de reduzir custos e facilitar a manutenção de aplicações. A POO representa de maneira mais natural processos do mundo real dentro de sistemas computacionais, promovendo modularidade, confiabilidade, extensibilidade e reutilização de componentes, características que tornam o software mais robusto e compreensível. Por isso, ao longo das últimas décadas, diversas linguagens foram criadas com foco nesse paradigma, como Java, C++ e Python, popularizando ainda mais sua aplicação tanto em ambientes acadêmicos quanto empresariais .

No âmbito da FoodDelivery, a utilização da POO é essencial para estruturar o backend do projeto Lu Delivery. A aplicação desse paradigma permite organizar o sistema em classes e objetos que refletem os elementos centrais do negócio, como usuários, pedidos e restaurantes, garantindo escalabilidade e manutenibilidade. Assim, a POO torna-se a base para que o sistema cresça de forma organizada e segura, acompanhando as necessidades do mercado e oferecendo maior confiabilidade aos clientes e parceiros.

Fundamentação Teórica

- **Classes**
  Uma classe é a estrutura fundamental da programação orientada a objetos. Ela funciona como um molde para a criação de objetos, descrevendo quais atributos (características) e métodos (ações) estarão presentes nas instâncias geradas. Dessa forma, uma classe não é o objeto em si, mas sim a definição que orienta como ele será construído e utilizado pelo programa.

- **Atributos**

Atributos são variáveis declaradas dentro de uma classe e que representam as propriedades ou características de um objeto. São responsáveis por armazenar valores específicos de cada instância, como nome, idade ou preço. Cada objeto criado a partir de uma classe pode possuir valores diferentes para seus atributos, garantindo a individualidade.

- **Construtores**
  Construtores são métodos especiais utilizados para inicializar objetos. Eles possuem o mesmo nome da classe e não possuem tipo de retorno. Sua função principal é atribuir valores iniciais aos atributos de uma instância no momento da sua criação, permitindo maior controle sobre como os objetos são instanciados.

- **Métodos**
  Métodos representam os comportamentos ou ações de um objeto. São funções definidas dentro da classe que podem manipular atributos, executar cálculos ou interagir com outras classes. Em essência, os métodos definem "o que o objeto pode fazer" e "como ele faz".

- **Diagrama de Classes**
  O diagrama de classes é uma representação gráfica utilizada na UML (*Unified Modeling Language*) para descrever a estrutura estática de um sistema. Ele mostra as classes, seus atributos, métodos e os relacionamentos entre elas, como herança, associação e dependência. Esse tipo de diagrama é amplamente utilizado na análise e projeto de sistemas, pois facilita a compreensão da arquitetura antes da implementação do código.

Metodologia

O projeto foi implementado a partir do diagrama conceitual inicial fornecido pelo professor, que apresentava as classes Cliente, Pedido, ItemPedido e ItemCardapio. Essas classes foram criadas para representar o domínio do sistema. A classe Cliente contém os atributos de identificação, como id, nome e telefone. A classe Pedido registra as informações de cada pedido realizado, incluindo id, data e status, além da associação com o cliente correspondente. A classe ItemCardapio representa os produtos disponíveis, armazenando id, nome e preço. Já a classe ItemPedido foi utilizada como intermediária entre Pedido e ItemCardapio, permitindo registrar a quantidade e o preço unitário de cada item, resolvendo a relação muitos-para-muitos entre pedidos e itens do cardápio.

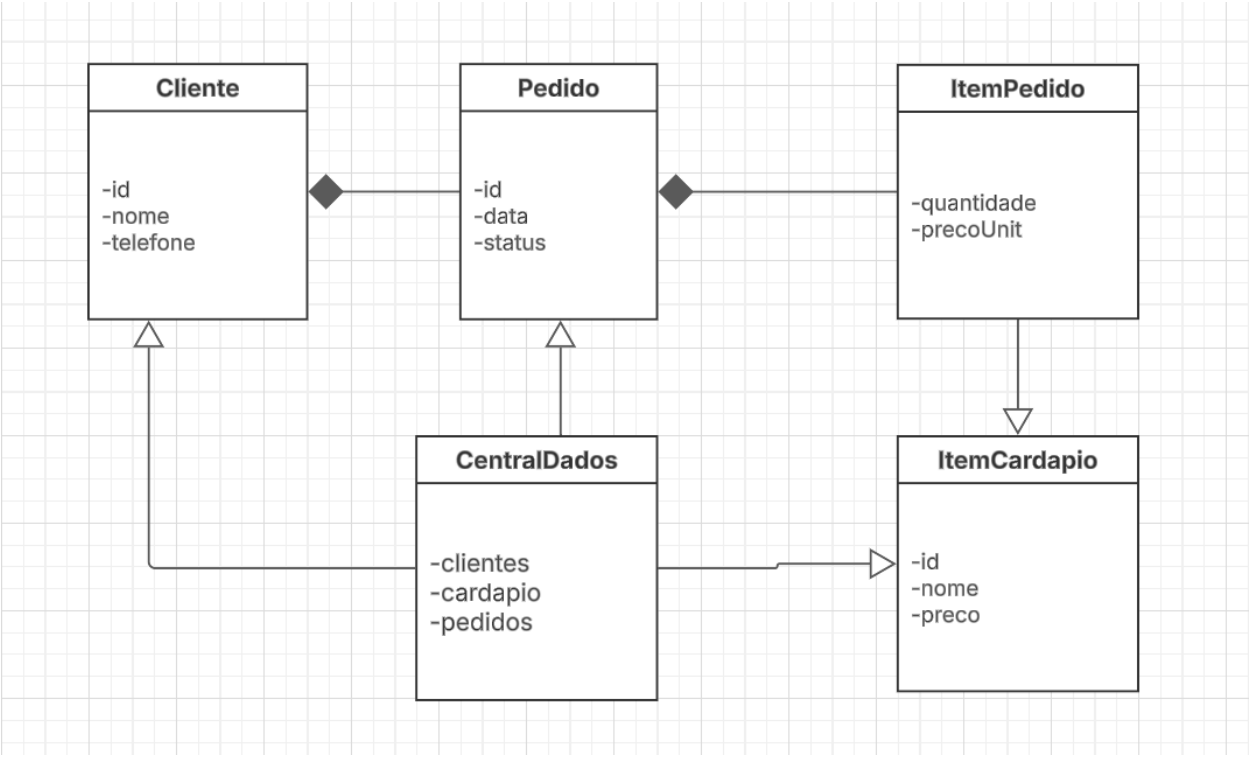
Além das classes de domínio, foram implementadas classes de apoio para organizar melhor a lógica do sistema. A classe CentralDados funciona como um repositório em memória, responsável por armazenar listas de clientes, pedidos e itens do cardápio. Para separar as regras de negócio, foram criadas as classes de serviço. A ServicoCadastro gerencia o cadastro de

clientes e itens do cardápio, inserindo-os na CentralDados. A ServicoPedido é responsável pela criação e gerenciamento dos pedidos, permitindo associar pedidos a clientes já cadastrados, adicionar itens aos pedidos e atualizar seu status.

A execução do sistema foi centralizada na classe Main, que atua como ponto de entrada da aplicação. Nela, foram simuladas operações de cadastro de clientes, cadastro de itens do cardápio, criação de pedidos e inclusão de itens nos pedidos. Dessa forma, é possível reproduzir o funcionamento básico de um sistema de gerenciamento de pedidos em um restaurante.

Essa estrutura foi escolhida para garantir clareza e organização. As classes de domínio refletem o modelo conceitual original, enquanto CentralDados e as classes de serviço permitem separar responsabilidades, facilitando a manutenção e possibilitando futuras expansões do sistema.

Resultados e Discussões



Criamos o diagrama de classes do sistema com base no modelo fornecido pelo professor, adaptando-o para, a partir dele, projetar e criar outras classes necessárias para atender a todos os requisitos. O sistema tem quatro classes principais: Cliente, Pedido, ItemPedido e ItemCardapio.

Na classe Cliente, colocamos os atributos id, nome e telefone. Cada cliente pode ter um ou mais pedidos, criando uma relação de um para muitos com a classe Pedido, que tem os atributos id, data e status. Cada pedido está ligado a um cliente e pode ter um ou mais itens de pedido. Para

registrar corretamente a quantidade de cada item em um pedido, criamos a classe ItemPedido, que guarda a quantidade e o precoUnit de cada item e se conecta a um único ItemCardapio. A classe ItemCardapio tem id, nome e preco, podendo aparecer em vários pedidos através da ligação com ItemPedido.

Também criamos a classe CentralDados, que funciona como um lugar para guardar os dados em memória, armazenando clientes, pedidos e itens do cardápio de forma centralizada e organizada. As classes de serviço, como ServicoCadastro e ServicoPedido, usam o CentralDados para gerenciar o cadastro de clientes, itens e pedidos, mantendo a lógica do sistema separada da estrutura dos dados.

Durante o desenvolvimento, procuramos manter a separação de responsabilidades, aplicar verificações para garantir que os dados fiquem corretos e usar o padrão singleton no CentralDados, garantindo um único ponto de acesso aos objetos. Dessa forma, o diagrama de classes mostra a estrutura do sistema de forma clara, organizada e fácil de manter ou ampliar no futuro.

Considerações Finais

Durante o desenvolvimento do projeto, alguns pontos se mostraram desafiadores, enquanto outros foram bastante interessantes. Um deles foi a organização de diversas classes e seus métodos de forma que o sistema não ficasse confuso, garantindo que cada parte tivesse uma função clara e que a comunicação entre elas fosse correta.

Entre os aspectos interessantes, destacamos a implementação de novas soluções, como o uso do padrão singleton para o armazenamento dos dados no CentralDados, que facilitou a centralização e consistência das informações em todo o sistema.

Se tivéssemos mais tempo, procuraríamos deixar o diagrama e a implementação mais compactos e organizados, além de explorar melhorias na validação de dados e na criação de funcionalidades adicionais para tornar o sistema mais robusto e fácil de expandir.

Referências Bibliográficas

MORAES, Caique Vinicius de. Programação Orientada a Objetos: domine os fundamentos da POO. São Paulo: Novatec, 2021.

CARVALHO, Thiago Leite e. Orientação a Objetos: aprenda seus conceitos e suas aplicabilidades de forma efetiva. São Paulo: Casa do Código, 2016.

REFACTORING GURU. Singleton. Refactoring Guru, s.d. Disponível em: https://refactoring.guru/pt-br/design-patterns/singleton. Acesso em: 28 ago. 2025.

ORACLE. Java Platform, Standard Edition Documentation. Disponível em: https://docs.oracle.com/en/java/. Acesso em: 28 ago. 2025.