

Lu Delivery – Peru: Sistema de Pedidos para Restaurantes com Programação Orientada a Objetos

Franklin Ferreira dos Santos, Giovanna Salomão Rodrigues, Kaila de Souza Costa, Lucas de Jesus Barreto, Lucas Silva Oliveira

~~Curso de Sistemas de Informação – 6º semestre~~ Centro Universitário de Excelência –
~~(UNEX)~~ Feira de Santana, BA - Brasil

franklinferreira280@gmail.com, irodrigues708@gmail.com,
kaillacsouza@gmail.com, ljbarreto04@gmail.com, lucasolv.info@gmail.com

Abstract. This work presents the development of an order system called Lu Delivery - Peru, developed in Java following Object-Oriented Programming (OOP) concepts. The system was built divided into five parts (P1–P5), one for each team member, covering from infrastructure to report generation. The application allows customer registration, menu management, order placement, status tracking, and report generation. The project aimed to consolidate practices of encapsulation, composition, and service usage in OOP, focusing on the clarity and robustness of the solution.

Resumo. Este trabalho apresenta o desenvolvimento de um sistema de pedidos denominado FoodDelivery - Peru, desenvolvido em Java seguindo conceitos de Programação Orientada a Objetos (POO). O sistema foi construído sendo dividido em cinco partes (P1–P5) uma para cada membro da equipe, abrangendo desde a infraestrutura até a geração de relatórios. A aplicação permite o cadastro de clientes, gerenciamento de cardápio, realização de pedidos, acompanhamento de status e geração de relatórios. O projeto teve como objetivo consolidar práticas de encapsulamento, composição e uso de serviços em POO, focando na clareza e robustez da solução.



1. Introdução

Este trabalho apresenta a implementação de um sistema denominado FoodDelivery - Peru, cujo objetivo é simular um sistema de pedidos, utilizando conceitos fundamentais de Programação Orientada a Objetos (POO). O desenvolvimento foi dividido entre integrantes do grupo, em cinco partes (P1 a P5), cada uma responsável por um módulo específico. O trabalho buscou reforçar práticas de encapsulamento, coesão, composição e lógica de programação.



Na introdução, vocês devem dar as boas vindas ao leitor:

- O contexto do problema apresentado
- Quais os objetivos que eram esperados
- Um breve indicativo dos seus resultados e de metodologia

→ ve o que esperar mais para frente no trabalho.

1º O que é POO?
2º Por que a usamos?

2. Fundamentação Teórica

No projeto FoodDelivery - Peru, aplicamos os seguintes conceitos de Programação Orientada a Objetos (POO):

Modelo de escrita similar a IA Generativa, baseada em Tópicos e como resume. O relatório é para dar preferência a texto descritivo.

Encapsulamento

Ocultação de detalhes internos das classes, expondo apenas métodos controlados.
Exemplo: Classes Cliente e Pedido com atributos privados e métodos públicos.

Classes ??

Classes: Modelos como Cliente, Pedido, ItemCardapio.

Arquitetura em Camadas

Modelo: Cliente, Pedido, ItemCardapio, PedidoItem.

Interface: MenuCLI (com funções para clientes, cardápio, pedidos e relatórios). Por que restringir camadas?

Relações entre Objetos

Associação: Relação simples.

Exemplo: Pedido associado a um Cliente.

Agregação: Relação "todo-parte" fraca.

Exemplo: CentralDeDados agrupa Cliente.

Composição: Relação "todo-parte" forte.

Exemplo: Pedido compõe PedidoItem.

Singleton

Esse padrão permite criar objetos únicos para os quais há apenas uma instância. Este padrão oferece um ponto de acesso global.

Exemplo: CentralDeDados, gerencia todos os dados globalmente.

Enum

Constantes

Tipo com valores fixos.

enumeração de valores constantes

Exemplo: PedidoStatus define estados do pedido (ACEITO, PREPARANDO, ...).

Como esse padrão é definido?
Como ele consegue justificar isso?

CLI (Command-Line Interface)

Interface textual para interação com o usuário.

Exemplo: MenuCLI com opções para cada módulo.

→ Cadê a explicação sobre Diagrama de classe?

3. Metodologia e Desenvolvimento

O projeto foi desenvolvido em Java, adotando uma estratégia de divisão modular de tarefas, onde cada membro foi responsável por uma parte específica e sequencial do sistema (P1 a P5). O desenvolvimento seguiu um ciclo incremental, em que cada nova parte integra-se e dependia das funcionalidades implementadas na anterior, assegurando a coesão e a evolução orgânica do projeto.

mas precisava ser apenas incremental?

A estrutura do código foi organizada em camadas, separando a lógica de negócios (modelos e serviços) da interface com o usuário (CLI), conforme detalhado na divisão de tarefas abaixo.

3.1 Divisão de Tarefas

essa es-
tutura de
texto nôo
era o espe-
godo, pois
isso configura-
sem status
Report nôo
um relatório
técnico
científico.

P1 – Infraestrutura e Núcleo Inicial, gerido por Lucas Oliveira:

Objetivo: Criar a base estável para o sistema.

Implementação: Desenvolveu o ponto de entrada (Main) que inicia o menu principal. Criou o enum PedidoStatus (ACEITO, PREPARANDO, FEITO, etc.) com a lógica de validação que impede transições de estado inválidas (ex.: pular de ACEITO para ENTREGUE). Garantiu que o projeto compilasse e apresentasse um menu inicial funcional com todas as opções listadas.

→ não precisavam
incrementar e digerir o
que cada um fez.
Não é um status report

P2 – Módulo de Clientes, gerido por Kaila Costa:

Objetivo: Permitir o cadastro e listagem de clientes.

Implementação: Modelou a classe Cliente com ID automático e campos obrigatórios (nome, telefone). Implementou as funções no MenuCLI para criar novos clientes e listar os existentes, incluindo validações para evitar dados vazios. A funcionalidade foi integrada ao CentralDeDados (Singleton).

P3 – Módulo de Cardápio, gerido por Giovanna Rodrigues:

Objetivo: Gerenciar os itens disponíveis para venda.

Implementação: Construiu a classe ItemCardapio com código sequencial, nome e preço. Implementou as operações de CRUD no MenuCLI, aplicando regras de negócio como nome não-vazio e preço maior que zero. Os itens cadastrados foram armazenados no CentralDeDados.

P4 – Módulo de Pedidos, gerido por Lucas Barreto:

Objetivo: Criar pedidos, gerenciar seus itens e controlar seu ciclo de vida através dos status.

Implementação: Desenvolveu as classes Pedido (com número sequencial, data de criação e status inicial) e PedidoItem (para composição com o cardápio). Implementou a lógica complexa de adicionar/remover itens a um pedido, calcular o total e, crucialmente, avançar o status do pedido respeitando o fluxo rígido definido em P1. Todas as operações foram disponibilizadas no MenuCLI.

P5 – Central de Dados e Relatórios, gerido por Franklin Ferreira:

Objetivo: Consolidar todos os dados em uma base única e gerar relatórios.

Implementação: Finalizou a implementação do padrão Singleton na classe CentralDeDados, garantindo que fosse o único repositório global para listas de clientes, itens e pedidos. Desenvolveu o serviço de relatórios, que percorre os dados centralizados para gerar duas visões: um relatório simplificado (totais gerais) e um detalhado (com lista de todos os pedidos, seus itens e valores).

→ faltou uma
explicação
mais
detalhada

3.2 Diagrama de Classe

A Classe PedidoItem

→ O que é um diagrama de classes?
Qual o objetivo dele?

essa seção do Diagrama de Classe do Jeto que foi escrita devria estar em resultados.

Um desafio central na modelagem do sistema foi representar adequadamente a quantidade de cada item dentro de um pedido. Uma associação direta entre as classes Pedido e ItemCardapio se mostrou insuficiente, pois apenas indicaria que um item estava no pedido, mas não quantas unidades desse item foram solicitadas.

Para resolver essa limitação, foi introduzida a classe PedidoItem, que atua como uma classe de associação. Esta decisão de projeto quebra a relação muitos-para-muitos entre pedidos e itens, permitindo que a quantidade seja armazenada como um atributo específico dessa relação. Dessa forma, o sistema pode representar com precisão que um Pedido contém **n** unidades de um determinado ItemCardapio, resolvendo eficazmente a questão da cardinalidade e dos atributos de relacionamento.

N

Pedido → Cliente

Você turham que explicar melhor: associação, agregação e Composição

Optou-se pela seta simples de associação (\rightarrow) em vez do diamante preto de composição ($\blacklozenge \rightarrow$) entre Pedido e Cliente (no exemplo mostrado pelo professor), porque os objetos possuem ciclos de vida independentes. Enquanto um pedido depende da existência de um cliente para ser criado, a exclusão de um pedido não implica na exclusão do cliente associado. O cliente é uma entidade independente que pode existir e realizar múltiplos pedidos no sistema, caracterizando uma relação de associação e não de composição.

Isso também é usado em banco de dados.

e está certo pois é o Círculo de vida do pedido que depende do cliente.

3.2 Estratégia de Integração e Validação

↳ a composição só dá justamente porque um obj pedido não deve existir se um objeto cliente não existir

A integração entre os módulos foi contínua. A CentralDeDados (Singleton), iniciado em P1 e finalizado em P5, atuou como o núcleo de integração, garantindo que todos os módulos acessassem os mesmos dados. Para validar o funcionamento de cada parte, foram utilizados critérios claros de Definition of Done (DoD), como "clientes registrados devem aparecer na listagem" ou "pedidos devem avançar de status sem pular etapas". A validação foi feita através de testes manuais sistemáticos via interface CLI após a conclusão de cada módulo.

→ atuou como um repositório centralizado.

*DoD → é legal → gente!!!
Isso pode ficar na metodologia em subsequentes impressões.*

4. Resultados e discussões

→ o que é seu RP? Você deveria colocar na fundamento final.

O sistema FoodDelivery foi implementado com sucesso, atendendo a todos os requisitos funcionais especificados. Abaixo destacam-se os principais resultados:

4.1 Funcionalidades Implementadas

Cadastros da experiência? **Cadastro de Clientes:** O sistema permite registrar clientes com nome e telefone (obrigatórios). IDs são gerados automaticamente, *a partir de ...*

Os códigos identificadores

Gerenciamento de Cardápio: Itens podem ser cadastrados com nomes (não vazios) e preço (> 0). Códigos são sequenciais e gerados automaticamente, *a partir de ...*
maior que zero (preço > 0.0)

Registro de Pedidos: Pedidos são criados com número sequencial, data/hora de criação, status inicial ACEITO e associados a um cliente. Itens do cardápio são adicionados com

↓ Júnio

quantidades *ao pedido*

bodrío em fluxo
Controle de Status: Transições de status *de pedido* seguem fluxo rígido (ACEITO → PREPARANDO → ... → ENTREGUE), sem que seja possível fazer saltos.

Relatórios:

Simplificado: Total de pedidos e valor arrecadado.

Detalhado: Lista completa de pedidos com clientes, itens e valores.

*→ Codi o juntinho
→ Onde estou os prints apresentando os relatórios*

4.2 Diagrama de Classes Final

O diagrama abaixo ilustra a estrutura completa do sistema, incluindo:

Classes de domínio: Cliente, Pedido, ItemCardapio, PedidoItem.

Singleton: CentralDeDados.

Relações: Associação, agregação e composição.

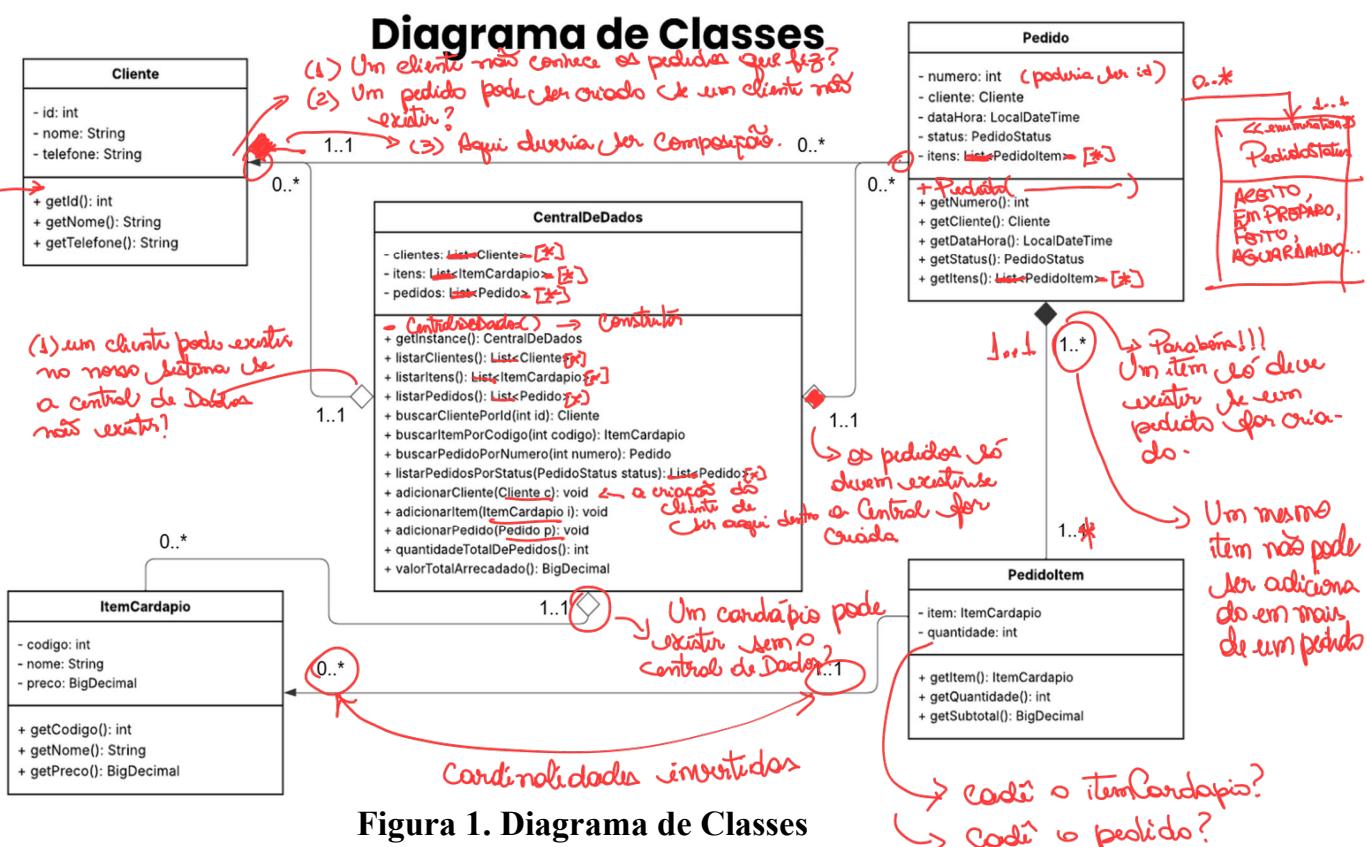


Figura 1. Diagrama de Classes

5. Conclusão

O projeto *FoodDelivery - Peru* proporcionou a consolidação de conceitos fundamentais de POO em Java, aplicados em um caso prático. A divisão modular (P1–P5) permitiu o trabalho colaborativo e a especialização de responsabilidades. A ausência de herança e polimorfismo forçou a equipe a reforçar o uso de composição e serviços, resultando em

ache que faltam uns poucos mais de discussões

uma arquitetura clara e facilmente extensível. Como trabalhos futuros, o sistema pode evoluir para uma integração com banco de dados.

Referências

Guanabara, G. (2020). Curso de Java POO. [s.l.]: Curso em Vídeo. Disponível em: https://www.youtube.com/watch?v=KJIL63MeyMY&list=PLHz_AreHm4dkqe2aR0tQK74m8SFe-aGsY. Acesso em: 19 ago. 2025.

DEVMEDIA. (2010). Padrão de Projeto Singleton em Java. [s.l.]: DevMedia. Disponível em: <https://www.devmedia.com.br/padrao-de-projeto-singleton-em-java/26392>. Acesso em: 22 ago. 2025.

DEVMEDIA. Encapsulamento em Java. São Paulo: DevMedia, 2018. Disponível em: <https://www.devmedia.com.br/encapsulamento-em-java/29106>. Acesso em: 23 ago. 2025.

Observações Gerais:

Primeiro, eu gostaria de parabenizá-los pelo esforço de escrita, apresentação e desenvolvimento. Seu gfee 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 usou pouca, se nenhuma, citação 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 gerado no trabalho coletivo. Procurem exemplos de citações direta e indireta. No próximo trabalho vai ser fundamental.
3. A leçon 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 Sétā de fundamentação pouco explora os conceitos. Nessa etapa vocês devem explicar tudo que o leitor precisa saber de conceitos técnicos novos para compreender a sua solução. Você deve ter:

- apresentado o diagrama de classes
- exemplo de código java referente esses diagramas.
- falando dos tipos de relacionamentos e os mapeamentos de um relacionamento entre atributos no código.

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

6. Aplicações ou explicações de conceitos que foram expressamente definidos para não serem utilizados nesse primeiro problema/oft.