

FoodDelivery: Implementação de Sistema de Gerenciamento de Pedidos para Restaurante

Clara Gabryellen P. Aderno, Igor dos S. Vieira, Jhon Luiz S. Santos, Natan C. Da Silva, Rayan S. Silva

Sistemas de Informação – Faculdade de Excelência (UNEX)

I

gabryellenpaixao@gmail.com, igorvieira1445@gmail.com,
devjhon01010@gmail.com, natancorreiatr@gmail.com,
rayansantos725@gmail.com

Abstract. This technical-scientific report details the implementation of the first challenge within the FoodDelivery trainee program, focusing on object-oriented backend development. The objective was to create a prototype restaurant order management system, operating via a Command-Line Interface (CLI) in Java. The solution utilizes fundamental Object-Oriented Programming concepts such as classes, objects, attributes, methods, and ~~implements~~ ~~key~~ Association, Aggregation, and Composition ~~L~~ relationships. Functionalities ~~the features~~ include customer and menu item registration, order management with a defined lifecycle, and sales report generation, with ~~All data~~ maintained solely in memory.

Resumo. Este relatório técnico-científico detalha a implementação do primeiro desafio do programa de trainee ~~do~~ FoodDelivery, focado no desenvolvimento back-end orientado a objetos. O objetivo foi criar um protótipo de sistema de gerenciamento de pedidos para um restaurante, operando via interface de linha de comando (CLI) em Java. A solução emprega conceitos fundamentais de Programação Orientada a Objetos, como classes, objetos, atributos, métodos e relacionamentos de Associação, Agregação e Composição. Foram desenvolvidas funcionalidades para cadastro de clientes e itens de cardápio, gerenciamento de pedidos com ciclo de vida definido e geração de relatórios de vendas, mantendo os dados exclusivamente em memória.

1. Introdução

Esse relatório técnico-científico documenta a implementação do primeiro entregável do programa trainee da FoodDelivery, uma empresa de tecnologia focada em conectar estabelecimentos gastronômicos a consumidores, ~~que~~ o programa visa desenvolver talentos em diversas áreas, incluindo o Desenvolvimento Web Orientado a Objetos, no qual os trainees enfrentam desafios crescentes. Neste contexto, o presente documento descreve a solução desenvolvida para o ~~o desafio maior~~ a criação de um protótipo de sistema de gerenciamento de pedidos para um restaurante, operando através de uma interface de linha de comando (CLI) e ~~utilizando~~ na linguagem de programação Java.

A Programação Orientada a Objetos (POO) desempenha um papel fundamental no desenvolvimento de aplicações robustas e de fácil manutenção, sendo essencial para

desenvolver
uma
aplicação
interagindo
entre os
Conceitos de

traduzir requisitos em um modelo de objetos coeso e funcional. Para este desafio, o foco principal foi a aplicação de Classes, Objetos, Atributos, Métodos e Relacionamentos de Associação, Agregação e Composição. O sistema desenvolvido permite o cadastro e listagem de clientes e itens de cardápio, o gerenciamento completo de pedidos – desde a criação e atualização de status (ACEITO, PREPARANDO, FEITO, AGUARDANDO ENTREGADOR, SAIU PARA ENTREGA e ENTREGUE) – e a geração de relatórios de vendas simplificados e detalhados, com todos os dados mantidos exclusivamente em memória.

Neste projeto

nossa objetivo

do
programa
POO Orientado
a
Objetos

2. Fundamentação Teórica

A Programação Orientada a Objetos (POO) é um paradigma de programação que modela o mundo real através de objetos que interagem entre si, sendo crucial para o desenvolvimento de aplicações robustas e de fácil manutenção. Este projeto de desenvolvimento do sistema de gerenciamento de pedidos da FoodDelivery baseia-se nos pilares fundamentais da POO, aplicados na linguagem Java, sendo eles:

1. Classes: Uma classe é um modelo ou “planta” para criar objetos. Ela define um tipo de entidade, especificando suas características (atributos) e comportamentos (métodos). No contexto do sistema FoodDelivery, classes como *Cliente*, *ItemCardapio* e *Pedido* servem para representar os conceitos do domínio do problema. Por exemplo, a classe *Cliente* define o que um cliente é no sistema, enquanto *ItemCardapio* define as propriedades de um item vendável.
2. Atributos: Atributos são as características, ou propriedades, que definem o estado de um objeto, sendo declarados dentro de uma classe. Eles armazenam os dados que descrevem um objeto individual. Para o sistema FoodDelivery, exemplos de atributos incluem nome e telefone para um Cliente, ou nome e preço para um ItemCardapio. O sistema também exige que códigos numéricos únicos para clientes e itens, e um número sequencial para pedidos, sejam gerados automaticamente e gerenciados como atributos.
3. Construtores: Um construtor é um método especial em uma classe, responsável por inicializar um novo objeto dessa classe. Ele é chamado automaticamente no momento da criação de um objeto e garante que o objeto seja criado em um estado válido, com seus atributos devidamente configurados. No gerenciamento de pedidos, um construtor de *Pedido* pode, por exemplo, garantir que o pedido seja criado com um cliente associado, um status inicial “ACEITO” e a data/hora atual.
4. Métodos: Métodos são funções ou rotinas que definem os comportamentos, ou ações, que um objeto pode realizar. Eles encapsulam a lógica de negócios e permitem a interação entre objetos.
5. Diagrama de Classes: O diagrama de classes é uma ferramenta gráfica de Linguagem de Modelagem Unificada, utilizada para visualizar a estrutura estática de um sistema orientado a objetos. Ele mostra as classes do sistema, seus atributos e métodos, e as relações entre elas, como Associação, Agregação e Composição. Para este desafio, um diagrama parcial foi fornecido, indicando que “Um pedido é feito por exatamente

preendendo

esse não é
um conceito da
Programação
Orientada a
Objetos

Nesse caso
é um
número
de classe
e não de
objeto

→ esse ponto
deveria estar
nos resultados

agora devria ter
uma explicação
numa subseção sobre
Diagrama de Classe.
Não tem uma imagem
para descrever um
diagrama.

um cliente" e que "Um cliente pode ter um ou mais pedidos". A relação entre Pedido e ItemCardapio é de muitos-para-muitos, sugerindo a necessidade de uma classe intermediária para gerenciar a quantidade de cada item dentro de um pedido. A criação de um diagrama de classes final é uma tarefa essencial para documentar a solução completa.]

3. Metodologia

A metodologia empregada foi definida nas seguintes etapas:

1. Análise e modelagem inicial: A primeira fase consistiu na análise detalhada dos requisitos funcionais e não funcionais. Isso incluiu a compreensão das funcionalidades de gerenciamento de cardápio, clientes e pedidos, bem como a necessidade de relatórios de vendas. As regras de negócio, como a geração automática, e única de códigos para clientes e itens, números sequenciais para pedidos, o ciclo de vida rígido dos pedidos com transições de status específicas (ACEITO, PREPARANDO, FEITO, AGUARDANDO ENTREGADOR, SAIU PARA ENTREGA, ENTREGUE), e a criação de pedidos em múltiplas etapas, foram cuidadosamente estudadas para serem incorporadas ao design do sistema.
2. Implementação das classes de domínio e gerenciamento: Com base no diagrama de classes estendido e na análise dos requisitos, procedeu-se à implementação das classes em Java. As classes Cliente, ItemCardapio e Pedido foram codificadas conforme suas responsabilidades, encapsulando atributos (como nome, preço, telefone, status) e métodos (para cadastrar, listar, atualizar). Foi dada especial atenção à implementação dos construtores para garantir a inicialização correta dos objetos.
3. Desenvolvimento da Interface de Linha de Comando: A interação com o usuário foi desenvolvida através de uma interface de linha de comando (CLI), que é o ponto de entrada para todas as operações do sistema. A lógica da CLI foi projetada para interagir com os subsistemas de forma simplificada, utilizando métodos unificados.
4. Versionamento e colaboração: O projeto foi versionado utilizando Git, com um modelo de desenvolvimento centralizado e o repositório foi compartilhado com a equipe e o tutor. A colaboração entre os membros da equipe foi fundamental para a divisão de tarefas e integração do código.

4. Resultados

A implementação do primeiro entregável do programa de trainee da FoodDelivery resultou em um protótipo funcional de sistema de gerenciamento de pedidos para um restaurante, operando através de uma interface de linha de comando (CLI) e desenvolvido na linguagem Java. Este sistema foi projetado para gerenciar clientes, itens de cardápio e pedidos, com todos os dados mantidos exclusivamente em memória sem persistência de banco de dados. Este sistema foca na aplicação estrita dos pilares da Programação Orientada a Objetos, excluindo o uso de herança e polimorfismo conforme as diretrizes do desafio. As funcionalidades principais incluem o gerenciamento de cardápio (cadastro e listagem de itens), gerenciamento de clientes (registro e listagem) e gerenciamento de pedidos (registro, atualização de status e consulta por status). Duas modalidades de relatórios de vendas foram implementadas: um simplificado e outro

Vocês deviam
ter colocado
na fundamento
é o que é
RF e RNF

mas vocês não
fizeram que
expanderam o
diagrama de
classe. Como
foi essa expansão?

Quais classes e
métodos foram
criados?

1. Análise e modelagem inicial: A primeira fase consistiu na análise detalhada dos requisitos funcionais e não funcionais. Isso incluiu a compreensão das funcionalidades de gerenciamento de cardápio, clientes e pedidos, bem como a necessidade de relatórios de vendas. As regras de negócio, como a geração automática, e única de códigos para clientes e itens, números sequenciais para pedidos, o ciclo de vida rígido dos pedidos com transições de status específicas (ACEITO, PREPARANDO, FEITO, AGUARDANDO ENTREGADOR, SAIU PARA ENTREGA, ENTREGUE), e a criação de pedidos em múltiplas etapas, foram cuidadosamente estudadas para serem incorporadas ao design do sistema.
2. Implementação das classes de domínio e gerenciamento: Com base no diagrama de classes estendido e na análise dos requisitos, procedeu-se à implementação das classes em Java. As classes Cliente, ItemCardapio e Pedido foram codificadas conforme suas responsabilidades, encapsulando atributos (como nome, preço, telefone, status) e métodos (para cadastrar, listar, atualizar). Foi dada especial atenção à implementação dos construtores para garantir a inicialização correta dos objetos.
3. Desenvolvimento da Interface de Linha de Comando: A interação com o usuário foi desenvolvida através de uma interface de linha de comando (CLI), que é o ponto de entrada para todas as operações do sistema. A lógica da CLI foi projetada para interagir com os subsistemas de forma simplificada, utilizando métodos unificados.
4. Versionamento e colaboração: O projeto foi versionado utilizando Git, com um modelo de desenvolvimento centralizado e o repositório foi compartilhado com a equipe e o tutor. A colaboração entre os membros da equipe foi fundamental para a divisão de tarefas e integração do código.

Como foi
fazida essa
análise de
requisitos?

Command
line
Interface,
você tem
que colocar
o nome que
dá origem
a isso,
nesto seu
título obo-
cado ou
traduzido?

Importante
destacar
isso!

Como foi isso?
Como essa flexibilidade se daria?

Os resultados detalhado, projetados para serem flexíveis na escolha do algoritmo de geração. O sistema incorpora regras de negócio cruciais, como a geração automática e única de códigos para clientes e itens, e um número sequencial para pedidos. A criação de pedidos é um processo multi-etapas com confirmação final, e o ciclo de vida dos pedidos possui transações de status rígidas e bem definidas (*ACEITO*, *PREPARANDO*, *FEITO*, *AGUARDANDO ENTREGADOR*, *SAIU PARA ENTREGA*, *ENTREGUE*), controlando o comportamento dos objetos conforme seu estado. Uma “central de dados” foi estabelecida como um único ponto de acesso para gerenciar todas as listas de dados, garantindo consistência. Além disso, o projeto foi desenvolvido em grupo e versionado usando Git, empregando o modelo de desenvolvimento centralizado. Um diagrama de classes final foi produzido, detalhando a estrutura estática completa da solução, incluindo a resolução da relação muitos-para-muitos.

6. Considerações finais

O desenvolvimento do programa proporcionou uma experiência de aprendizado intensa e altamente formativa, focando na aplicação dos pilares fundamentais da Programação Orientada a Objetos em um contexto prático de gerenciamento de pedidos de restaurante via CLI. As funcionalidades essenciais incluíram gerenciamento de cardápio, clientes e pedidos, além de relatórios de vendas. O sistema implementou regras de negócio como geração automática de códigos, processo multi-etapas para criação de pedidos, e um ciclo de vida rígido para os status dos pedidos, tudo coordenado por uma “central de dados” única. Os principais desafios foram as restrições impostas aos conceitos de POO sem herança e polimorfismo, a manipulação de dados exclusivamente em memória, e a complexidade do ciclo de vida dos pedidos. No entanto, a aplicação prática da POO, a modelagem de regras de negócio e a consolidação de habilidades em Java e Git foram aspectos interessantes. Com mais tempo, a equipe focaria na persistência de dados utilizando um banco de dados como MySQL, na incorporação de herança e polimorfismo em um tratamento de exceções mais robusto, em testes automatizados e na implementação de “notificações futuras”.

↑ no tratamento de
erro? → não foi
pedido nesse
etapa.

Referências??

↓ como seria
isso

Não tem referência em nenhum lugar.

Procurar!!!

Observações Gerais:

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

Fizemos 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 colíbro. 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 Série de Fundamentos para explorar os conceitos. Nessa fase vocês devem explicar tudo que o leitor precisa saber de conceitos técnicos novos para compreender a sua solução. Você deve ir mais fundo.

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

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

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