

Fora do padrão!!!

Relatório Técnico-Científico: OttawaDelivery

Alex Oliveira, Alice Aragão, Ana Clara Ribeiro, Monique Prado

Centro Universitário de Excelência – UNEX, R. Ubaldino Figueira, 200 - Recreio, Vitória da Conquista - BA, 45020-510.

~~Sistemas de Informação,~~

~~- Disciplina: Desenvolvimento Web Orientada a Objetos.~~

Abstract. This report presents the development of OttawaDelivery, a prototype of a food delivery system implemented in Java with an object-oriented approach. The project simulates the entire flow of a delivery service, from customer registration to order completion. The solution was modeled using classes, attributes, methods, and design patterns, such as Singleton, to ensure data centralization and consistency. The implementation relied on a modular architecture and command-line interaction, allowing for clarity in testing and usability. The results show a functional system that reflects core object-oriented programming principles and offers potential for future expansion.

Você não precisa dizer que foi usando do Classes, atributos e métodos, pois já disse que o trabalho foi implementado usando do uma abordagem em Ocupação de Objetos.

Resumo. Este relatório apresenta o desenvolvimento do OttawaDelivery, um protótipo de sistema de delivery implementado em Java, utilizando a abordagem orientada a objetos. O projeto simula o fluxo completo de um serviço de entrega, desde o cadastro de clientes até a conclusão dos pedidos. A solução foi modelada com classes, atributos, métodos e padrões de projeto, como o Singleton, garantindo centralização e consistência dos dados. A implementação seguiu uma arquitetura modular com interface em linha de comando, facilitando a interação e os testes. Os resultados demonstram um sistema funcional, alinhado aos princípios da POO, com possibilidades de futuras expansões.

1. Introdução

A programação orientada a objetos (POO) é um paradigma de programação que se concentra na organização do código em torno de "objetos" em vez de ações e dados separados. Essa abordagem visa representar o mundo real em termos de objetos de software, que combinam dados (atributos) e comportamentos (métodos). A utilização da POO permite a criação de sistemas modulares e escaláveis, que são mais fáceis de manter, expandir e reutilizar, o que foi um dos objetivos principais do projeto. Este relatório detalha a implementação do sistema, um protótipo de serviço de delivery desenvolvido em Java para simular o fluxo de pedidos de um restaurante, desde o cadastro do cliente até a entrega final. O sistema foi projetado com foco nos princípios de POO e utiliza uma interface de linha de comando (CLI) para interação.

Aqui nesse tópico tem uma mistura de que deveria estar na fundamentação. Como que está na introdução. Eu acredito que vocês também devem ter contextualizado o problema que foi apresentado. Sugestões:

- (1) Contexto da Food Delivery
- (2) Contexto do modelagem utilizando a orientação a Objetos (OO)
- (3) Dar uma visão do que esperar no restante do relatório.

2. Fundamentação Teórica

A escrita baseada em
tópicos é uma das características mais forte das
mais fortes das gerações.

O sistema foi idealizado com base em conceitos fundamentais da programação orientada a objetos:

- **Classes:** são estruturas que definem as características e os comportamentos que os objetos criados a partir delas terão como.
- **Atributos:** São as características, ou dados, que um objeto possui. Por exemplo, a classe Cliente, tem atributos como nome e telefone.
- **Construtores:** São métodos especiais utilizados para inicializar um objeto quando ele é criado. Na classe Cliente, o construtor é usado para definir o código, nome e telefone de um novo cliente.
- **Métodos:** São as ações, ou comportamentos, que um objeto pode realizar. A classe Pedido, por exemplo, possui um método avancarStatus() para gerenciar o ciclo de vida de um pedido.
- **Diagrama de Classes:** É uma representação visual que mostra a estrutura estática do sistema, exibindo as classes, seus atributos, métodos e os relacionamentos entre elas. O diagrama de classes final do projeto demonstra como as diferentes partes do sistema se conectam para funcionar de forma coesa.

Vocês deviam tirar trazido com exemplo

ilustrar o que
é visual



3. Metodologia

O projeto foi implementado ~~na linguagem de programação~~ em Java, utilizando o paradigma de Programação Orientada a Objetos (POO) como base para a modelagem do sistema. O desenvolvimento utilizou uma estrutura modular com classes dedicadas a diferentes funcionalidades, o que permitiu uma organização clara do código. O controle de versão foi realizado com ~~o repositório~~ GitHub, permitindo o trabalho em equipe e o acompanhamento das modificações.

que os módulos

O que
é essa estrutura
modular a qual
vocês se referem.

O sistema foi estruturado para operar através de uma interface de console (CLI), o que facilitou a interação e os testes. A aplicação segue um fluxo lógico que simula o processo de um serviço de delivery:

Linha de comando
CLI -
Command Line
Interface

- **Cadastro:** Clientes e itens do cardápio são registrados no sistema.
- **Criação de Pedidos:** Um pedido é criado, associando um cliente a um ou mais itens do cardápio.
- **Controle de Entrega:** O status do pedido é atualizado em um ciclo de vida predefinido.
- **Relatórios:** Relatórios de vendas são gerados para fornecer uma visão geral do desempenho do negócio.

O desenvolvimento do **OttawaDelivery** seguiu ~~o paradigma de POO~~ em Java, adotando boas práticas de modularização e controle de dados:

1. Modelagem Orientada a Objetos:

O sistema foi dividido em classes especializadas:

- **App:** Responsável pela interface em linha de comando (CLI) e menus do sistema.
- **SistemaPedidos:** Reúne a lógica principal do negócio (cadastro, criação de pedidos, atualização de status, relatórios).
- **CentralDados:** Implementada com o padrão Singleton para centralizar o armazenamento em memória e evitar inconsistências.
- **Pedido, ItemPedido, Cliente, ItemCardapio:** Modelam as entidades do domínio de delivery.

Veja de
veriam
ter falado
na fundo
muitas

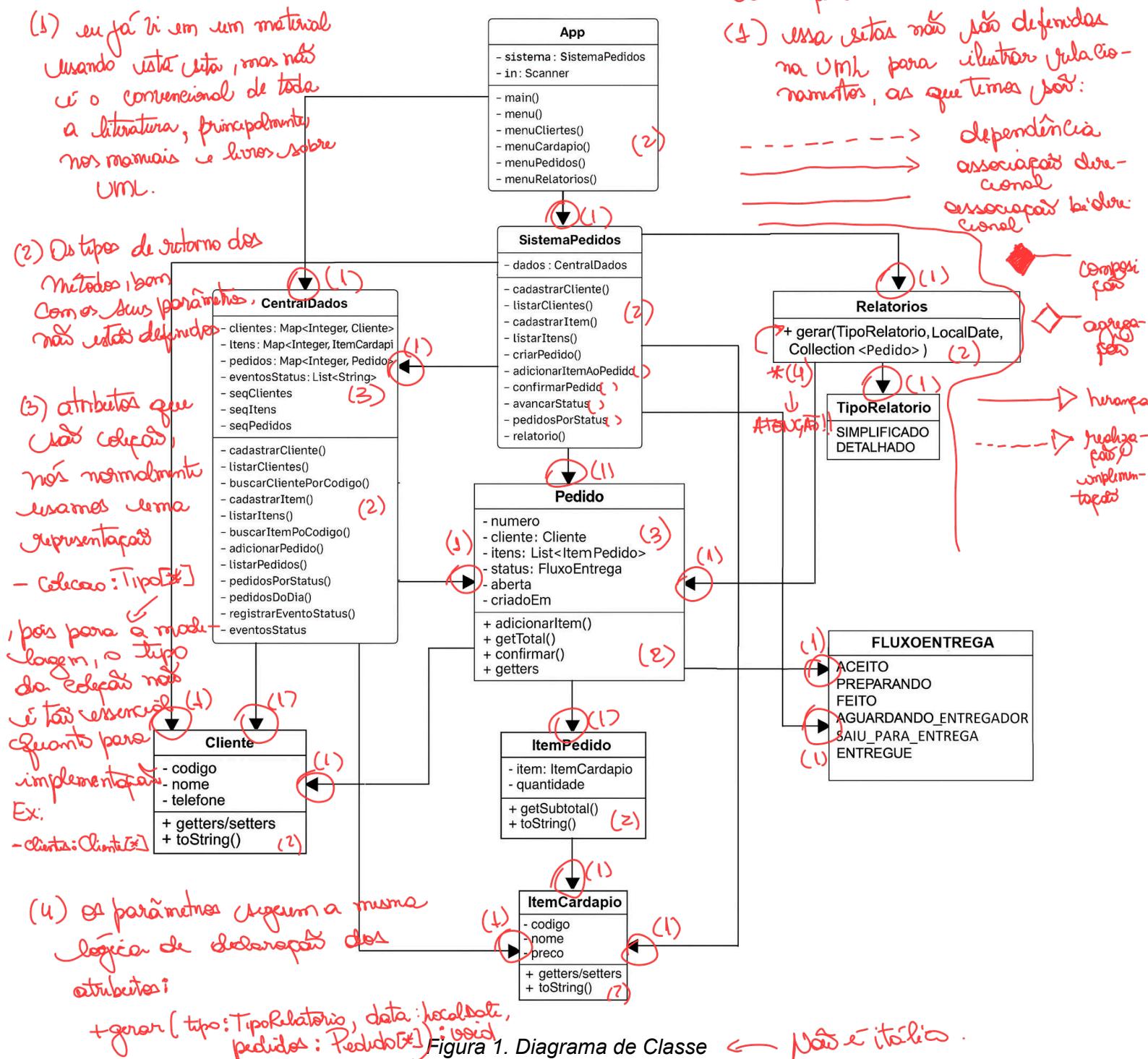
2. Fluxo de Operações:

- **Cadastro:** Clientes e itens do cardápio são registrados.
 - **Criação de Pedidos:** Os pedidos associam clientes a itens, criando uma estrutura com **ItemPedido**.
 - **Ciclo de Entrega:** O status do pedido é atualizado conforme o ciclo definido em **FluxoEntrega**.
 - **Relatórios:** A classe **Relatorios** gera relatórios simplificados ou detalhados a partir dos pedidos.
- (textos juntados em julgados ao início da leitura de método)*
- Lógica*
- 3. **Interface CLI:** A escolha pela linha de comando simplificou testes e validou a estrutura de backend sem dependência de interface gráfica.
 - 4. **Versionamento com GitHub:** O uso do GitHub permitiu controle de versão, colaboração e histórico de alterações.
- Já falou anteriormente*

4. Resultados e Discussões

4.1 Diagrama de caso de Uso

O OttawaDelivery resultou em um sistema modular, escalável e funcional, simulando o fluxo de um aplicativo de delivery. O Diagrama de Classes (Figura 1) reflete a estrutura completa do sistema:



- **App** usa **SistemaPedidos** para centralizar interações;
- **SistemaPedidos** depende de **CentralDados** para gerenciar dados;

- **Pedido** compõem Cliente e ItemPedido, além de usar **FluxoEntrega**;
- **CentralDados** mantém coleções de clientes, itens e pedidos, sendo o núcleo de persistência;
- **Relatorios** utilizam **TipoRelatorio** (SIMPLIFICADO, DETALHADO) para gerar análises.

4.2 Pontos importantes de implementação:

- **Centralização de dados:** A classe **CentralDados** utiliza o padrão Singleton para manter consistência e evitar múltiplas instâncias de armazenamento.
 - **Modularização:** O sistema foi dividido em classes especializadas (**Pedido**, **Cliente**, **ItemCardapio**, **Relatorios**), seguindo o princípio da responsabilidade única.
 - **Fluxo de pedidos estruturado:** O ciclo de entrega é controlado pela enumeração **FluxoEntrega**, garantindo clareza no avanço dos status
 - **Interface simplificada:** A escolha da CLI reduziu a complexidade e facilitou testes.
 - **Relatórios versáteis:** A classe **Relatorios** permite análises rápidas (simplificadas) ou detalhadas.
- Início

5. Considerações Finais

O **OttawaDelivery** é um sistema funcional, modular e bem estruturado, que aplica de forma prática os princípios da ~~programação orientada a objetos~~ em Java. A modelagem orientada a classes, aliada ao uso de padrões de projeto como o Singleton, conferiu consistência, organização e robustez à arquitetura, evidenciando boas práticas de desenvolvimento de software.

Entre os principais pontos fortes, destacam-se a centralização de dados, a separação clara de responsabilidades e o ciclo de pedidos bem definido. Embora a aplicação utilize uma interface em linha de comando, sua lógica de negócio apresenta possibilidades de expansão, incluindo integração com bancos de dados, interfaces gráficas ou sistemas web, tornando o sistema escalável e apto a evoluir para um produto real.

~~6.~~ Referências

- Nos fomos os padres! (/)*
- DevMedia - Principais conceitos da Programação Orientada a Objetos
<https://www.devmedia.com.br/principais-conceitos-da-programacao-orientada-a-objeto-s/32285>.
 - Alura - Programação Orientada a Objetos (POO)
https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos?srsltid=AfmBOorUixodtqmF044417xR1-FIYUBbpwP5QPt4Hp_pr74_5qJcTBV4.
 - DevMedia - Diagrama de Classes (UML) -
<https://www.devmedia.com.br/orientacoes-basicas-na-elaboracao-de-um-diagrama-de-classes/37224>.
 - DevMedia - Padrão de Projeto Singleton em Java -
<https://www.devmedia.com.br/padrao-de-projeto-singleton-em-java/26392>.

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.

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, citações 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 colívo. 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.
 - Conclua o que terá nos próximos meses.

4. A seção de fundamentação deve explorar os conceitos. Nessa seção vocês devem explicar tudo que o leitor precisa saber de conceitos técnicos novos para compreender a tese. Isso só pode ser feito com base em teoria e prática.

- 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 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 projeto, mas o resultado todo delas deve ficar na seção de resultados.

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