Relatório Técnico-Científico OAT1.1 - Desenvolvendo Aplicações

Tiago F. de Novais, Renato G. Souza, Haniel P. A. da Silva, Carlos E. S. Santos, Ênio F. T. Soares

Curso de Sistemas da Informação – Faculdade UNEX
Vitória da Conquista – Bahia – Brasil

Abstract. *This technical report presents an overview of Object-Oriented Programming (OOP), its fundamental concepts, and a practical implementation example. The document covers the importance of OOP, explains key concepts such as classes, attributes, constructors, methods, and class diagrams, describes the implementation methodology, presents the developed program, and reflects on challenges and lessons learned.*

Resumo. *Este relatório técnico apresenta uma visão geral da Programação Orientada a Objetos (POO), seus conceitos fundamentais e um exemplo de implementação prática. O documento aborda a importância da POO, explica conceitos-chave como classes, atributos, construtores, métodos e diagramas de classes, descreve a metodologia de implementação, apresenta o programa desenvolvido e reflete sobre os desafios e lições aprendidas.*

1. Fundamentação Teórica

A Programação Orientada a Objetos (POO) é um paradigma de programação fundamental que revolucionou o desenvolvimento de software ao introduzir conceitos que permitem modelar sistemas complexos de forma mais intuitiva, modular e próxima do mundo real. Seu princípio é a organização do código em torno de "objetos" que encapsulam dados (atributos) e comportamentos (métodos), promovendo assim maior reutilização, manutenibilidade e escalabilidade do software. Em um contexto empresarial moderno, onde a agilidade e a robustez são cruciais, dominar a POO é indispensável para construir aplicações funcionais, sustentáveis e adaptáveis a mudanças.

Este relatório documenta o desenvolvimento do programa da “Lu Delivery”, focado na implementação de um sistema de gestão de pedidos para restaurantes.

2. Fundamentação Teórica

2.1 Conceitos Básicos da Programação Orientada a Objetos

Classes representam objetos, funcionando como modelos ou templates para a criação de objetos. Uma classe define os atributos (características) e métodos (comportamentos) que seus objetos terão. No sistema implementado, as classes “Cliente”, “ItemCardapio”, “Pedido” e "ItemPedido” são exemplos deste conceito.

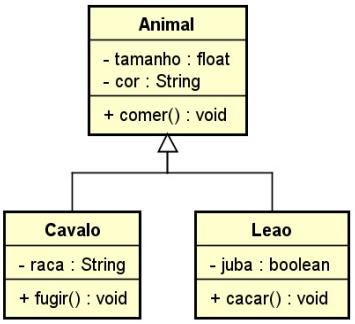
Atributos são variáveis que definem as características de um objeto. Por exemplo, a classe “Cliente” possui os atributos “codigo”, “nome” e “telefone”, enquanto a classe “ItemCardapio” possui “codigo”, “nome” e “preco”.

Construtores são responsáveis por inicializar os atributos do objeto em um estado válido. No projeto, cada classe possui construtores apropriados: a classe “Cliente” recebe “nome” e “telefone”, gerando automaticamente um código único; a classe “ItemCardapio” recebe “nome” e “preço”, também gerando um código automaticamente; e a classe “Pedido” recebe um objeto “Cliente” e inicializa sua lista de itens e status.

Métodos definem o comportamento dos objetos, representando as operações que podem ser executadas. Eles podem modificar o estado do objeto (métodos de instância) ou fornecer informações sobre seu estado (métodos de acesso). No sistema, exemplos incluem “adicionarItem()” na classe “Pedido” ou que permite incluir itens ao pedido; “avancarStatus()”, que gerencia as transições de estado do pedido.

O Diagrama de Classes representa visualmente a estrutura de um sistema orientado a objetos. Este diagrama mostra as classes do sistema, seus atributos, métodos e os relacionamentos entre elas, servindo como diretrizes para o código.

Classes são representadas por retângulos divididos em três compartimentos (nome, atributos e métodos). Atributos são listados com sua visibilidade (+, -, #), nome e tipo. Métodos são listados com sua visibilidade, nome, parâmetros e tipo de retorno. Relacionamentos são mostrados através de linhas conectando as classes.



Exemplo de Diagrama de Classes

3. Metodologia

3.1 Análise e Modelagem

Partindo dos requisitos funcionais e do diagrama de classes fornecido, realizou-se a identificação das entidades principais do sistema:

Cliente: Entidade responsável por armazenar informações dos clientes do restaurante.

ItemCardápio: Entidade que representa os itens disponíveis para venda.

Pedido: Entidade central que gerencia o ciclo de vida completo dos pedidos.

ItemPedido: Classe associativa que resolve o relacionamento muitos-para-muitos entre Pedido e ItemCardapio.

CentralDeDados: Classe singleton para gestão centralizada de dados.

3.2 Implementação das Classes

Cada classe foi implementada em Java, seguindo o princípio de encapsulamento:

Cliente e ItemCardapio:

Atributos declarados como private para garantir encapsulamento.

Construtores que recebem parâmetros essenciais (nome, telefone, preço).

Geração automática de códigos únicos através de variáveis estáticas (proximoCodigo).

Métodos de acesso (getters) para leitura controlada dos atributos.

Pedido:

Implementação de uma máquina de estados através do enum StatusPedido.

Todo pedido começa com o estado EM_ANALISE.

Método adicionarItem() para composição de itens no pedido.

Método avancarStatus() que controla as transições de estado conforme regras de negócio.

Método getValorTotal() para cálculo automático do valor total do pedido.

ItemPedido:

Classe associativa que armazena a referência ao ItemCardapio e a quantidade solicitada.

Método getSubtotal() para cálculo do valor parcial de cada item.

3.3 Interface de Linha de Comando (CLI)

A classe SistemaPedidosCLI implementa a interface com o usuário através de um sistema de menus interativos:

Menu principal com as opções Gerenciamento de Cardápio, Clientes, Pedidos e Relatórios.

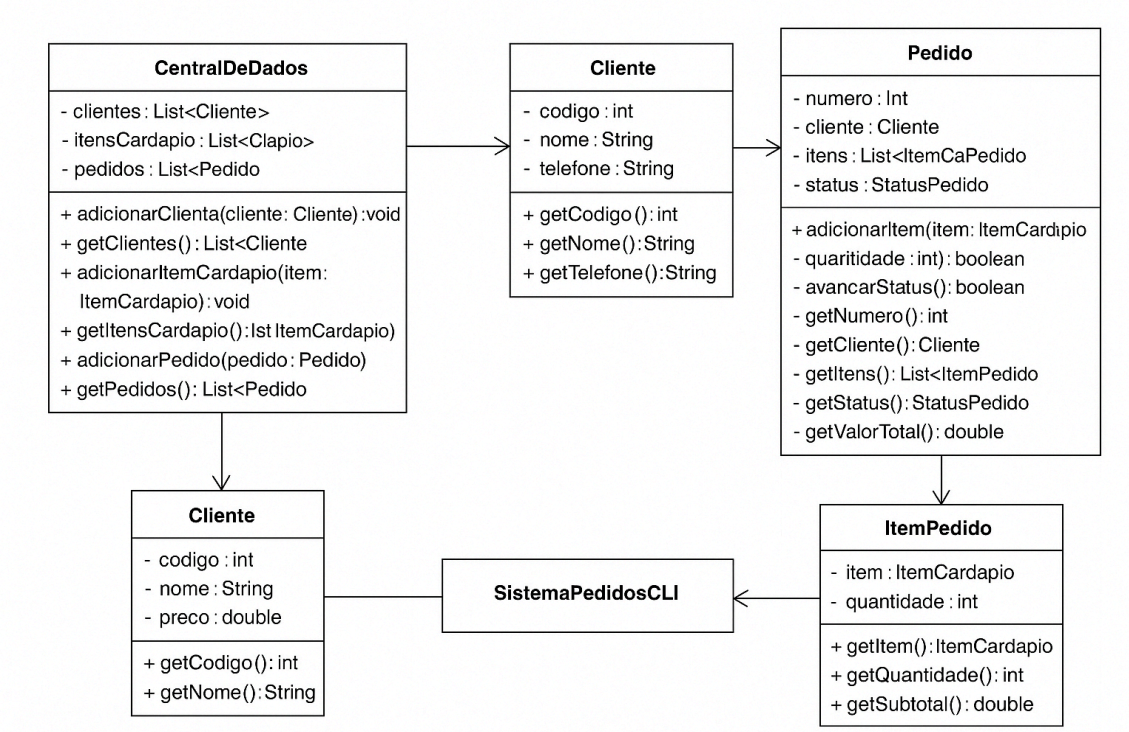
Submenus para operações específicas (cadastrar, listar, atualizar status).

Integração com a CentralDeDados para todas as operações de CRUD.

Validação básica de entradas do usuário para garantir consistência dos dados.

4. Resultados e Discussões

4.1 Diagrama de classes da nossa solução:



5. Considerações Finais

Pontos desafiadores: O aprendizado e uso de java para a produção de um projeto em um todo foi desafiador para a equipe pois é uma linguagem nova para os integrantes e robusta.

Pontos interessantes: O fato de java ser uma linguagem usável em vários cenários diferentes com uma ótima performance.

Melhorias futuras: Possível rastreamento do pedido em GPS (em caso de delivery), implementação de UI/UX para melhoria estética e simplificação do uso do sistema.

6. Referências

Booch, G., Rumbaugh, J., e Jacobson, I. (2005). Guia do Usuário da Linguagem de Modelagem Unificada (UML). Addison-Wesley Professional.

Gamma, E., Helm, R., Johnson, R., e Vlissides, J. (1994). Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos. Addison-Wesley.

Martin, R. C. (2002). Desenvolvimento Ágil de Software: Princípios, Padrões e Práticas. Prentice Hall.