PROGRAMAÇÃO 2

CIÊNCIA DA COMPUTAÇÃO
MÁSIO CÉSAR DE CARVALHO MORAES

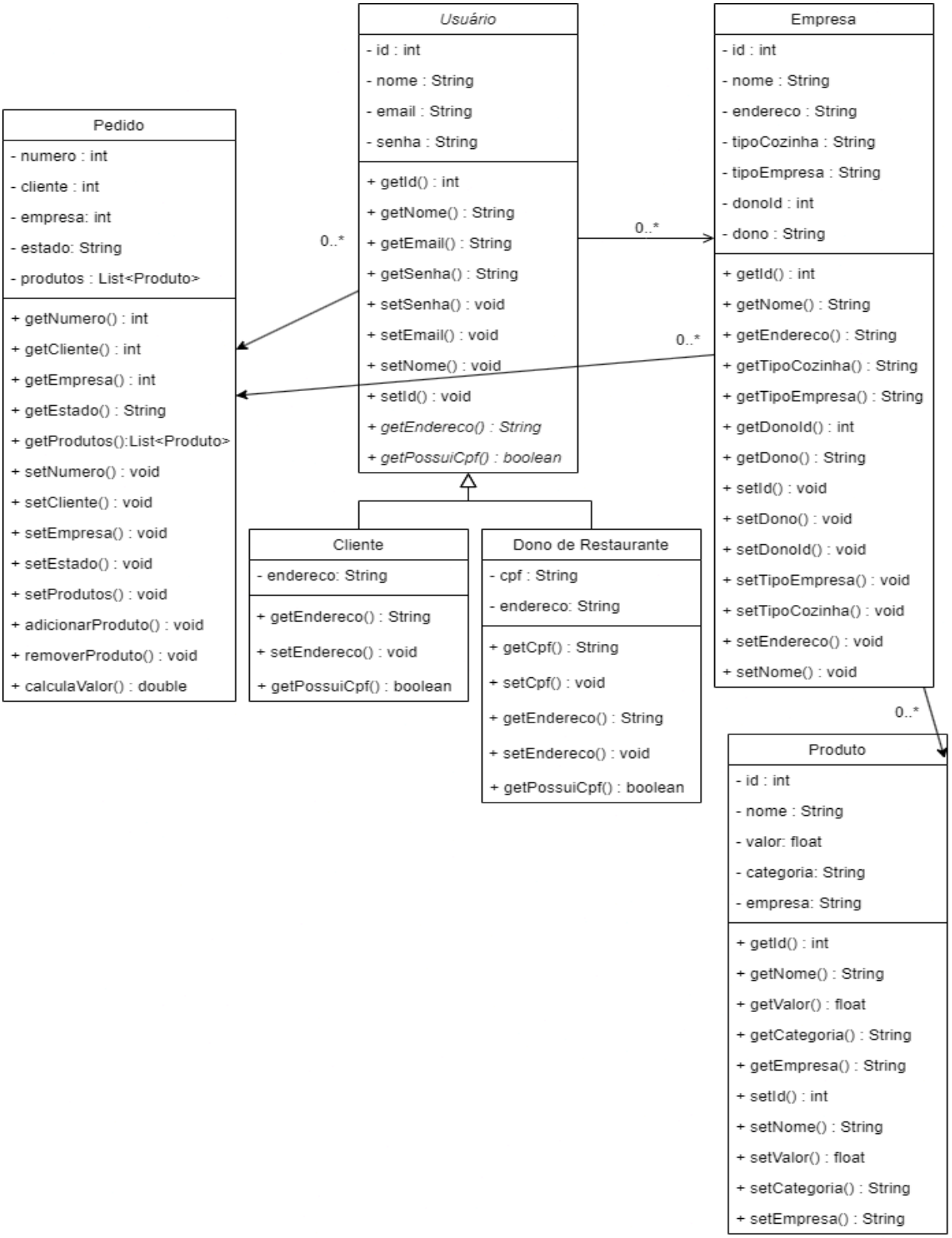
RELATÓRIO DE PROGRAMAÇÃO 2 - MYFOOD

Maceió/AL - 2024

SUMÁRIO

1. INTRODUÇÃO

Este relatório apresenta o projeto do sistema MyFood, uma aplicação desenvolvida para gerenciar empresas, pedidos, produtos e usuários em um sistema simples de delivery. O sistema inclui a gestão de diferentes tipos de usuários, produtos oferecidos, pedidos realizados e informações sobre as empresas.



2. DESIGN ARQUITETURAL DO SISTEMA

O design arquitetural do sistema foi estruturado para garantir uma organização clara e eficiente através da implementação do padrão Facade. Esse padrão possibilitou a centralização das operações, simplificando e unificando a interação entre os diversos subsistemas do projeto. A arquitetura foi planejada para assegurar manutenção e reutilização do código, organizando-o com base em suas funções e responsabilidades. As principais camadas que interagem com a Facade incluem: Exceptions, que gerencia o tratamento de erros; Models, que define as estruturas dos principais objetos; e Services, que coordena as operações do sistema e a persistência de dados.

- ## 2.1. Exceptions

  Para centralizar o tratamento de erros e simplificar a manutenção, a classe de Exceções foi criada. Ela é responsável por capturar e gerenciar todas as exceções geradas durante a execução do sistema, garantindo que problemas inesperados sejam tratados de forma consistente e eficiente.

- ## 2.2. Models

  Os modelos representam as principais entidades do sistema, estruturando dados e comportamentos relacionados aos usuários, empresas, produtos e pedidos. Eles foram desenvolvidos com base no diagrama de classes inicial e ajustados conforme as necessidades do projeto evoluíram, refletindo as User Stories e as funcionalidades do sistema.

- ## 2.3. Services

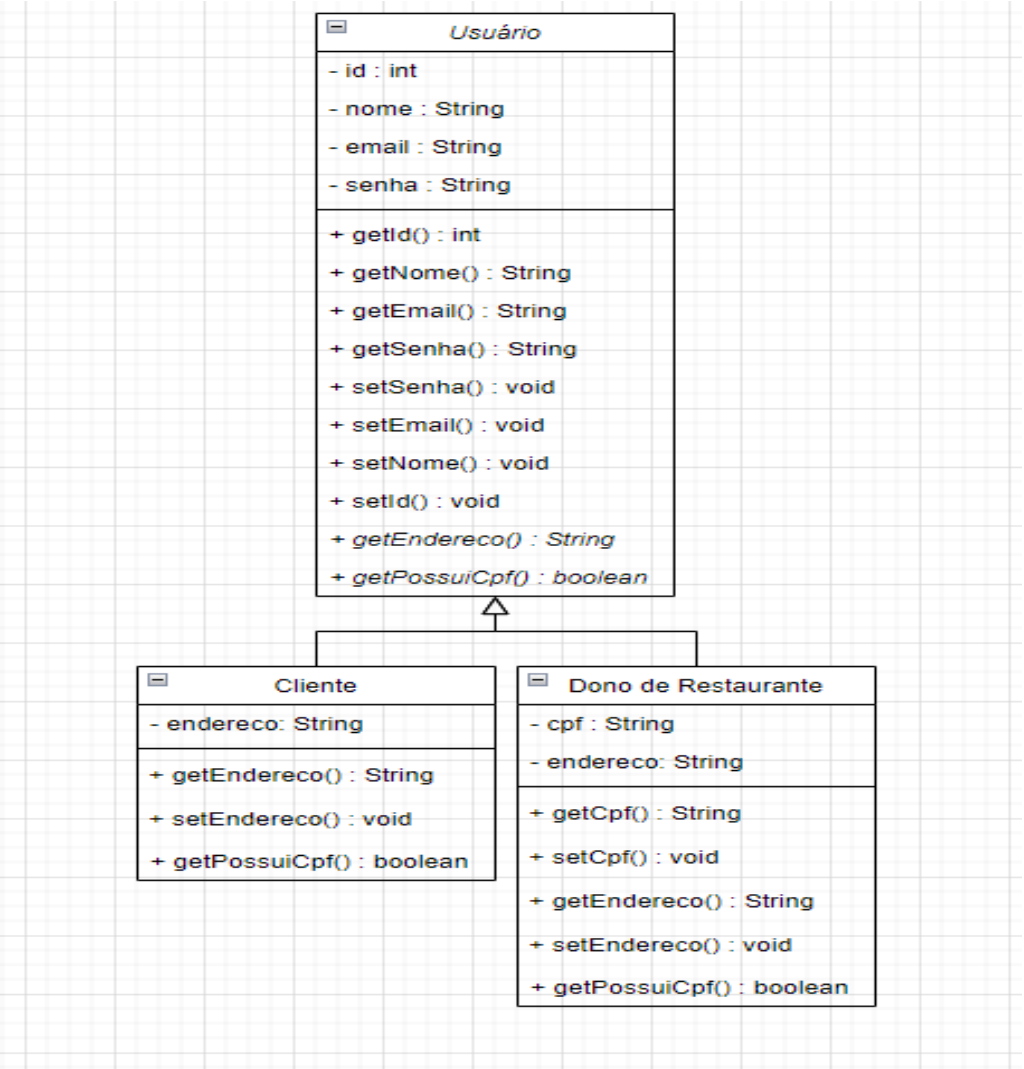
  Os serviços gerenciam as operações principais do sistema, incluindo configurações e histórico de estados. Eles são responsáveis pela criação e manipulação dos dados, controle da persistência através de arquivos XML, e execução de funcionalidades essenciais, como verificação e processamento de dados. Os serviços garantem a integração e o funcionamento adequado entre os diferentes componentes do sistema.

3. PRINCIPAIS COMPONENTES

A seguir, detalhamos os principais componentes do sistema.

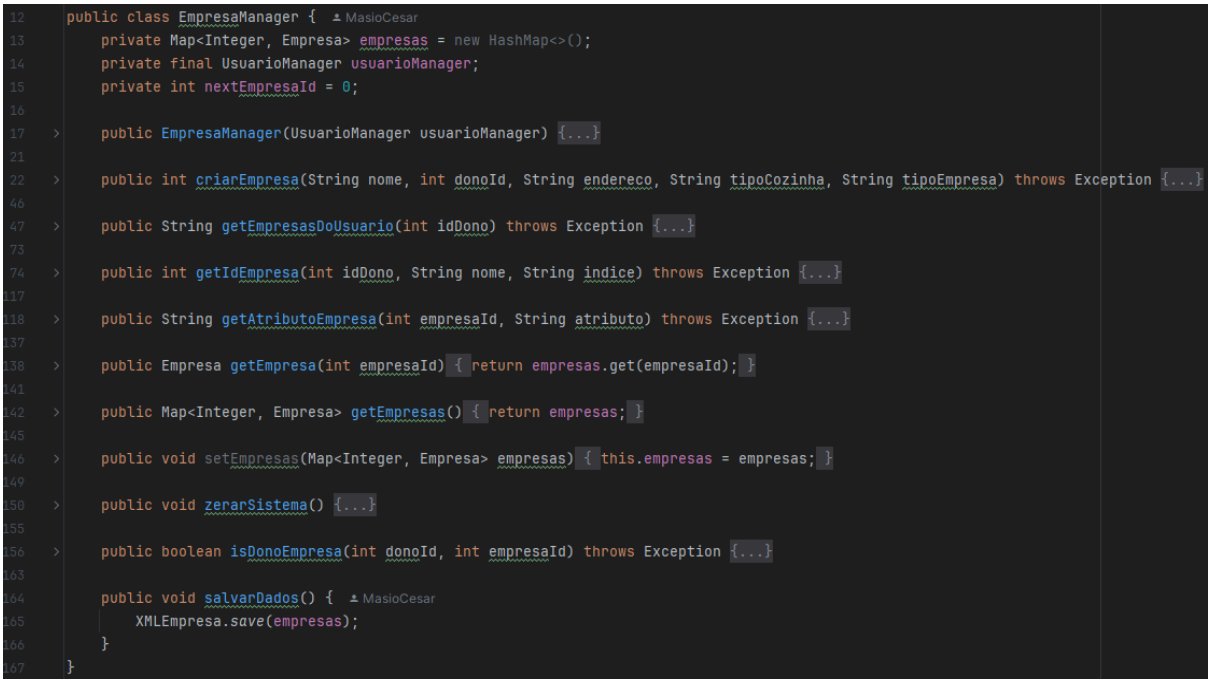
3.1. Usuario

A classe **Usuario** é a base para diferentes tipos de usuários no sistema. Inclui atributos como ID, nome, email, senha e endereço. Sendo esta classe herdada por outras 2 classes: **Cliente** e **DonoRestaurante.** Além do mais, também é uma classe abstrata, então há somente 2 tipos de usuário, cliente ou dono de restaurante.



3.2. EmpresaManager

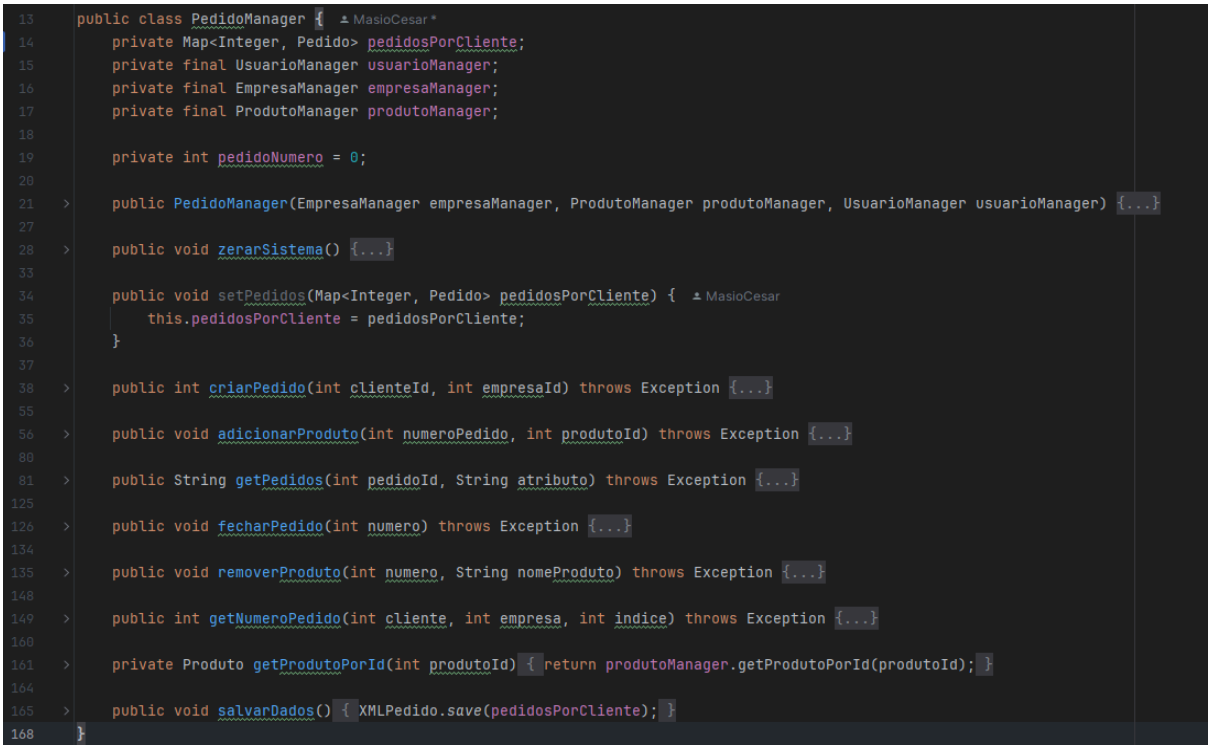
EmpresaManager é responsável por administrar todas as operações relacionadas às empresas no sistema. Suas responsabilidades incluem a criação de novas empresas, a verificação das empresas associadas a um usuário específico e a validação de dados de entrada para garantir que não ocorram duplicidades. Ele assegura que cada empresa seja única e corretamente associada ao seu respectivo dono.



```java
public class EmpresaManager {  MasioCesar
    private Map<Integer, Empresa> empresas = new HashMap<>();
    private final UsuarioManager usuarioManager;
    private int nextEmpresaId = 0;

    public EmpresaManager(UsuarioManager usuarioManager) {...}

    public int criarEmpresa(String nome, int donoId, String endereco, String tipoCozinha, String tipoEmpresa) throws Exception {...}

    public String getEmpresasDoUsuario(int idDono) throws Exception {...}

    public int getIdEmpresa(int idDono, String nome, String indice) throws Exception {...}

    public String getAtributoEmpresa(int empresaId, String atributo) throws Exception {...}

    public Empresa getEmpresa(int empresaId) { return empresas.get(empresaId); }

    public Map<Integer, Empresa> getEmpresas() { return empresas; }

    public void setEmpresas(Map<Integer, Empresa> empresas) { this.empresas = empresas; }

    public void zerarSistema() {...}

    public boolean isDonoEmpresa(int donoId, int empresaId) throws Exception {...}

    public void salvarDados() {  MasioCesar
        XMLEmpresa.save(empresas);
    }
}
```

3.3. PedidoManager

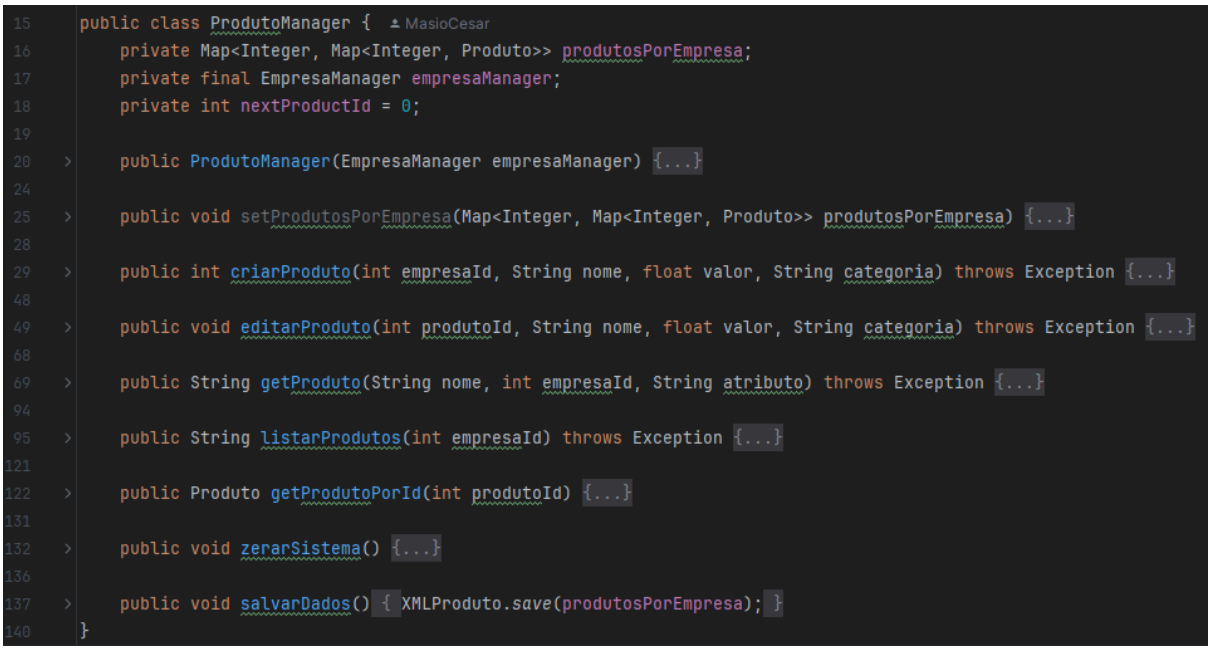
PedidoManager gerencia todos os aspectos relacionados aos pedidos dentro do sistema. Suas funções abrangem a criação de novos pedidos e a atualização do status dos pedidos existentes. Ele garante que todos os pedidos sejam corretamente registrados e atualizados.



```java
public class PedidoManager {  MasioCesar *
    private Map<Integer, Pedido> pedidosPorCliente;
    private final UsuarioManager usuarioManager;
    private final EmpresaManager empresaManager;
    private final ProdutoManager produtoManager;

    private int pedidoNumero = 0;

    public PedidoManager(EmpresaManager empresaManager, ProdutoManager produtoManager, UsuarioManager usuarioManager) {...}

    public void zerarSistema() {...}

    public void setPedidos(Map<Integer, Pedido> pedidosPorCliente) {  MasioCesar
        this.pedidosPorCliente = pedidosPorCliente;
    }

    public int criarPedido(int clienteId, int empresaId) throws Exception {...}

    public void adicionarProduto(int numeroPedido, int produtoId) throws Exception {...}

    public String getPedidos(int pedidoId, String atributo) throws Exception {...}

    public void fecharPedido(int numero) throws Exception {...}

    public void removerProduto(int numero, String nomeProduto) throws Exception {...}

    public int getNumeroPedido(int cliente, int empresa, int indice) throws Exception {...}

    private Produto getProdutoPorId(int produtoId) { return produtoManager.getProdutoPorId(produtoId); }

    public void salvarDados() { XMLPedido.save(pedidosPorCliente); }
}
```

3.4. ProdutoManager

ProdutoManager é responsável pela administração dos produtos oferecidos pelas empresas no sistema. Ele permite a criação, edição e listagem de produtos, além de possibilitar a verificação e atualização das informações dos produtos existentes. Ele assegura que todos os produtos estejam corretamente catalogados e atualizados.



3.5. UsuarioManager

UsuarioManager cuida da gestão dos usuários no sistema. Suas responsabilidades incluem a criação de novos usuários, a autenticação de login e a recuperação de informações dos usuários. Além disso, ele realiza a validação dos dados de entrada para garantir a integridade e a segurança das informações dos usuários.

```java
public class UsuarioManager {  MasioCesar
    private Map<String, Usuario> users;
    private final Map<Integer, Usuario> usersById;
    private int nextUserId = 0;

    public UsuarioManager() {...}

    // DONO RESTAURANTE
    public void criarUsuario(String nome, String email, String senha, String endereco, String cpf) throws Exception {...}

    // CLIENTE
    public void criarUsuario(String nome, String email, String senha, String endereco) throws Exception {...}

    public int login(String email, String senha) throws Exception {...}

    public String getAtributoUsuario(int id, String atributo) throws Exception {...}

    public Usuario getUser(int userId) throws Exception {  MasioCesar
        Usuario usuario = usersById.get(userId); // Use o mapa de IDs
        if (usuario == null) {...}
        return usuario;
    }

    private Usuario findUserById(int id) { return usersById.get(id); }

    public void zerarSistema() {...}

    // Método para salvar os dados no XML ao encerrar o sistema
    public void salvarDados() { XMLUsuario.save(users); }
}
```

3.6. DataPersistenceManager

A classe **DataPersistenceManager** gerencia a persistência de dados em formato XML. Utiliza XMLEncoder para salvar um mapa de dados em um arquivo e XMLDecoder para carregar esses dados quando necessário. O método save grava os dados no arquivo especificado, enquanto o método load recupera os dados do arquivo, retornando um novo HashMap se o arquivo não existir. Esta classe é essencial para a persistência de usuários, empresas, produtos e pedidos no sistema, garantindo que essas informações sejam armazenadas e recuperadas corretamente entre sessões.

```java
public class DataPersistenceManager {  MasioCesar

    private final String fileName;

    public DataPersistenceManager(String fileName) { this.fileName = fileName; }

    public void save(Map<?, ?> data) {  MasioCesar
        try (XMLEncoder encoder = new XMLEncoder(new FileOutputStream(fileName))) {
            encoder.writeObject(data);
        } catch (Exception ignored) {
        }
    }

    public Map<?, ?> load() {  MasioCesar
        File file = new File(fileName);
        if (!file.exists()) {
            return new HashMap<>();
        }

        try (XMLDecoder decoder = new XMLDecoder(new FileInputStream(fileName))) {
            return (Map<?, ?>) decoder.readObject();
        } catch (Exception ignored) {
        }

        return new HashMap<>();
    }
}
```

4. PADRÕES DE PROJETO

4.1. FACADE

Descrição Geral

O padrão Facade é um padrão de design estrutural que fornece uma interface unificada e simplificada para um conjunto de interfaces em um subsistema. O objetivo principal é abstrair e esconder a complexidade do sistema subjacente, oferecendo uma interface mais acessível e compreensível para os clientes. A ideia é desacoplar o cliente da complexidade do sistema, facilitando o uso e a manutenção do sistema.

Problema Resolvido

O padrão Facade é projetado para resolver o problema de sistemas complexos com múltiplas interfaces e subsistemas. Em um cenário onde o sistema possui diversas operações e interações complexas, o padrão Facade ajuda a reduzir o acoplamento e a dependência direta entre o cliente e o subsistema. Ele simplifica a comunicação com o sistema, tornando a interface de uso mais intuitiva e menos propensa a erros.

Identificação da Oportunidade

No desenvolvimento do sistema, a oportunidade de aplicar o padrão Facade foi identificada devido à complexidade crescente das operações e interações entre diferentes componentes, como gerenciamento de usuários, empresas, produtos e pedidos. Com diversos serviços e modelos envolvidos, tornou-se necessário um mecanismo para simplificar e centralizar a interação com o sistema, garantindo uma interface mais clara e fácil de usar.

Aplicação no Projeto

O escopo do projeto é extenso e envolve a administração de diversos modelos, como usuários, empresas, produtos e pedidos, além de múltiplos processos de validação e manipulação de dados. Esse ambiente complexo é ideal para a aplicação do padrão Facade, que visa simplificar a interação com o sistema e tornar o acesso às funcionalidades mais direto e intuitivo.

A Facade foi implementada para fornecer uma interface centralizada e simplificada, facilitando a comunicação com o sistema e ocultando a complexidade dos subsistemas subjacentes. Ela oferece métodos que atendem aos principais requisitos funcionais do sistema, como criação, alteração, leitura e remoção de entidades, além de operações relacionadas a pedidos e produtos.

Alguns métodos da Facade incluem:

1. `public void criarUsuario(String nome, String email, String senha, String endereco)` - Cria um novo usuário no sistema.
2. `public String getAtributoUsuario(int id, String atributo)` - Recupera um atributo específico de um usuário.
3. `public int criarEmpresa(String tipoEmpresa, int donoId, String nome, String endereco, String tipoCozinha)` - Cria uma nova empresa associada a um dono.
4. `public void editarProduto(int produtoId, String nome, float valor, String categoria)` - Edita os detalhes de um produto existente.
5. `public int criarPedido(int cliente, int empresa)` - Cria um novo pedido associando um cliente a uma empresa.

Esses métodos facilitam o gerenciamento das entidades e processos do sistema, simplificando a interface com o usuário e proporcionando uma experiência mais coesa e organizada. A Facade atua como um ponto de acesso único para essas operações, reduzindo o acoplamento e a complexidade na comunicação com os subsistemas.

.