Professor: Gleiph Ghiotto Lima de Menezes.
Data de entrega: 06/08/2025
Integrantes do Grupo:
- João Victor Granatto de Carvalho / 202465045A
- Adrian Maciel Rocha / 202465001A

Trabalho Final - DCC025

Introdução

Esse documento visa apresentar o projeto final da disciplina DCC025 - turma A, ao longo dele apresentaremos toda a dinâmica de desenvolvimento do trabalho, as ferramentas utilizadas, a modelagem, as divisões, tomadas de decisões e como executar o programa em sua própria máquina.

Tecnologias Utilizadas

O trabalho foi desenvolvido completamente na linguagem de programação Java, utilizando o Maven para construir o projeto. A versão do Java utilizada foi a *21.0.7.* e o Maven na versão *3.9.10*.

A IDE utilizada foi o Intellij Ultimate, onde essa versão permitiu a utilização do GUI Designer, uma ferramenta que auxiliou na criação das interfaces gráficas do projeto. Além disso, também utilizamos um repositório público no GitHub para ter o controle de versão do projeto, ajudando a se organizar melhor. O repositório pode ser acessado pelo link ao lado: Repositório do Projeto . E utilizamos a biblioteca GSon para fazer a persistência dos dados em um arquivo de texto.

Diagrama de Classes:

Nos baseamos no seguinte diagrama de classes para desenvolver o projeto:

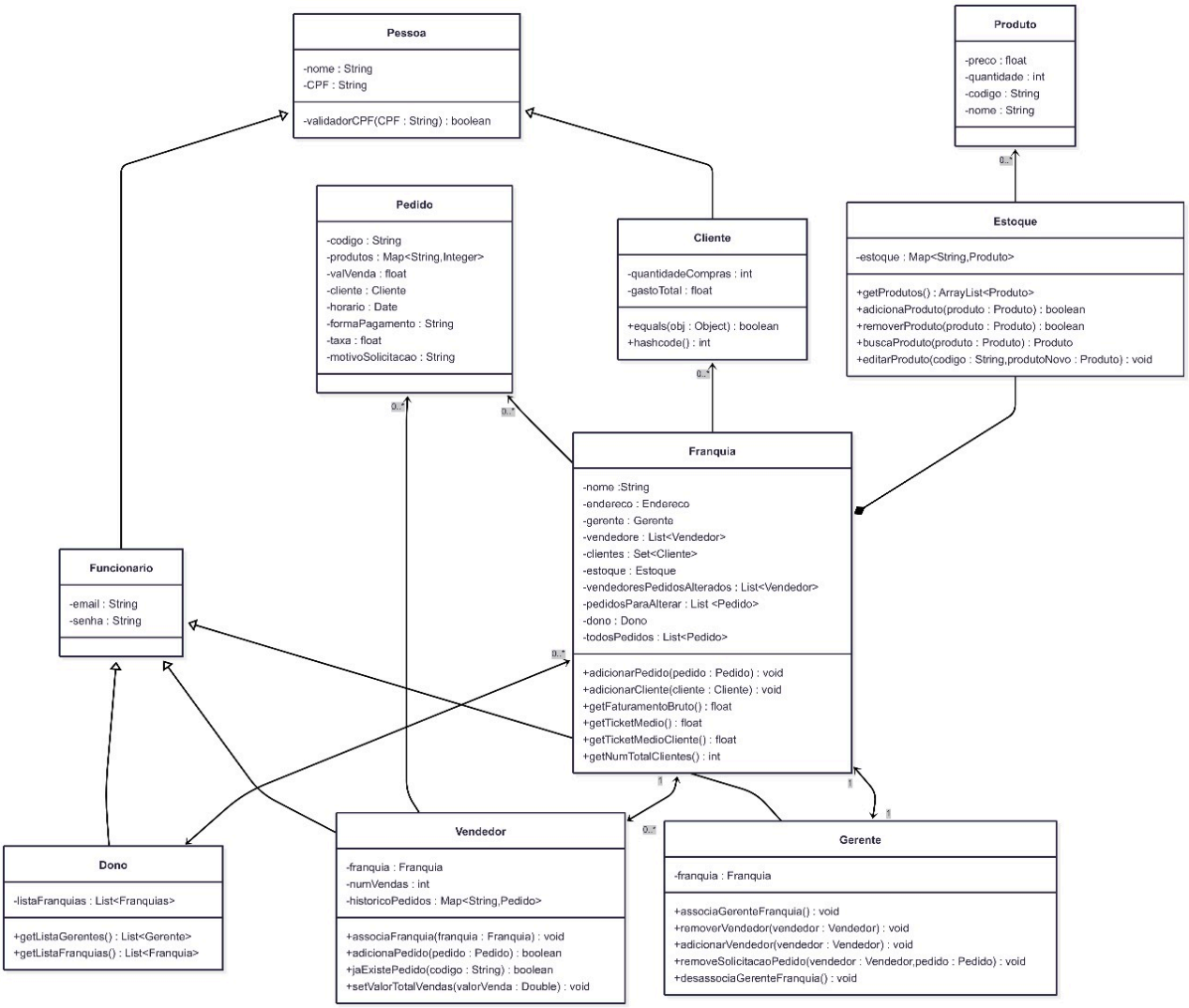

Figura 1: Diagrama de classes base para o projeto.

Diagrama com melhor qualidade e mais completo, está no repositório do GitHub.

Desenvolvimento do Projeto

Arquitetura:

Utilizamos a arquitetura Model-View–Controller(MVC) para desenvolver o sistema. Nosso código fonte tem os seguintes pacotes principais, o model, view e controller, e adiante vamos exemplificar o que cada um faz. Dentro do pacote model, estão as classes principais que modelam as principais entidades presentes no sistema, o pacote view é responsável pela interação com o usuário e o controller é responsável por operar com as entidades via tela e trazer dados para a tela, ou seja, faz a “ponte” entre o model e o view.

Persistência de Dados:

A parte da persistência de dados foi realizada utilizando a biblioteca Gson, que ajuda a serializar e desserilizar arquivos JSon. A entidade Dono, no nosso código, possui uma lista de franquias que administra, que por sinal, possuem uma lista de todos os funcionários que nela trabalham, então assim, quando o programa inicializa só precisamos desserializar o Json que contém um objeto da classe dono, e na hora de fechar o programa, fazemos o processo inverso, transformamos esse dono em um arquivo texto Json, e ele contém todas as informações necessárias para o funcionamento do sistema.

Herança:

O principal exemplo de herança no nosso código se refere a classe Pessoa. A classe Pessoa possui os seguintes atributos:Nome e CPF. Ela possui 2 subclasses, Cliente e Funcionário. Cliente herda todos os atributos de pessoa, já o Funcionário herda os 2 atributos além de adicionar os campos email e senha. A classe Funcionário possui 3 subclasses, vendedor, gerente e dono. Essas subclasses herdam os atributos de funcionários e implementam os métodos relativos à cada posição dentro da empresa.

Polimorfismo:

Nós usamos polimorfismo em uma parte crucial do nosso código, para a validação do login e para entrar no sistema. Na classe loginController, ela recebe um dono no Construtor, que como já dito na parte da persistência, possui todos os usuários cadastrados. A partir desses funcionários, utilizamos o método abrirTela() da classe funcionarioController. E as classes, donoController, vendedorController e

gerenteController herdam de funcionarioController e tem o seu próprio abrirTela(). Com isso, a partir de um funcionário, o programa consegue saber qual dela deverá abrir.

Interface:

Utilizamos interface no pacote Validadores, onde a interface ValidadorEntrada tem o método validar(String entrada). E a partir dela temos 3 validadores que implementam o método, de acordo com a necessidade. Tem o validador de campo vazio, validador de código e validador de email. E com isso conseguimos verificar a validade da entrada a partir do retorno do método, que é booleano.

Estruturas Usadas:

Utilizamos algumas estruturas em nosso projeto, principalmente a List e o Map. O Map é muito útil para fazer as buscas procurando pela chave, usamos ele para a verificação do usuário no sistema, para guardar quantidades de produtos, entre outras utilidades. Quando tínhamos casos em que precisávamos apenas guardar certas informações relativas a um determinado atributo, usamos a List por ser uma estrutura dinâmica, sem precisarmos definir o tamanho, por exemplo.

APIs Utilizadas:

No projeto, utilizamos 2 APIs externas para nos auxiliar. Utilizamos a **Caelum Stella** para fazer a validação do CPF dos usuários, não permitindo que algum CPF inválido seja cadastrado no sistema, e lançando uma exceção se isso acontecer. A segunda foi o **ViaCep Service**, que o utilizamos na Tela de Cadastro de uma nova franquia, para assim, quando o CEP for digitado e um botão for clicado, a partir do CEP informado a API consegue puxar todos os dados e preencher os campos automaticamente, como o estado,cidade,rua e etc.

Testes:

Os testes do projeto foram feitos utilizando o JUnit 5, na versão *5.10.0*. Fizemos testes unitários em todos os métodos que lançavam exceções, verificando com o *assertThrows* e *assertDoesNotThrow.* Além disso, fizemos testes em todos os validadores usando o *assertTrue* ou *assertFalse* para verificar o retorno dos métodos de validação. Para executar o testes basta entrar na pasta raiz do projeto e executar o comando “***mvn test”***.

Como compilar e executar o código?

Instale o arquivo .zip do projeto na sua máquina e extraia-o. Certifique-se de que o Maven está instalado e na versão 3.9.10 (que foi a utilizada no projeto), e o java na versão 21.0.7, para evitar qualquer problema na execução ou compilação do código. Para rodar o código via terminal, você deve executar o seguinte comando partindo da pasta raiz "***cd out\artifacts\Sistema_Franquia_jar***", e quando chegar nesse repositório, execute o comando "***java -jar Sistema_Franquia.jar***"

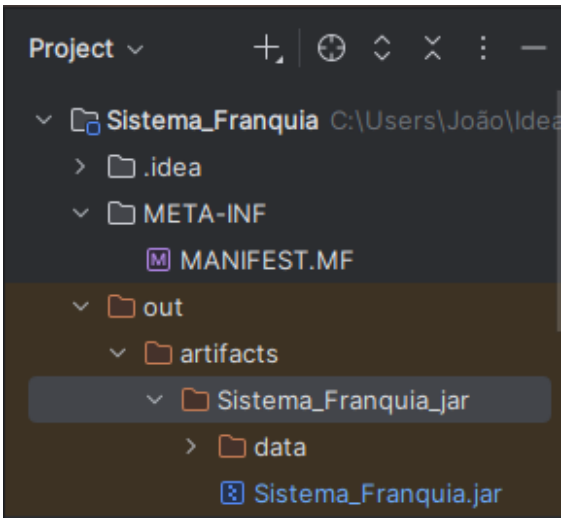

Figura 2: Caminho para chegar no executável do projeto.