Licenciatura em Engenharia Informática
Ano letivo 2024/2025

RELATÓRIO PROJETO "POOFS"

no âmbito da Unidade Curricular de
Programação Orientada a Objetos

Trabalho realizado por
Henrique Diz (2023213681)
Tomás Gonçalves (2019232712)

ÍNDICE

INTRODUÇÃO

O presente relatório pretende descrever todas as características do programa denominado "**POOFS - POO Finacial Services**", desenvolvido de acordo com parâmetros de programação orientada a objetos.

O programa tem como objetivo principal facilitar a gestão financeira de empresas, ao permitir a emissão de Faturas que contêm Produtos (Alimentares e/ou Farmacêuticos) para cada Cliente.

Cada **Cliente** tem nome, número de contribuinte e localização (Portugal Continental, Madeira ou Açores) associados.

Cada **Fatura** tem um número, o cliente ao qual está associado, a sua data de criação e a lista dos produtos associados.

Cada **Produto** pode ser Alimentar ou Farmacêutico:

- **Produtos Alimentares** poderão ser de Taxa de IVA Reduzida, Intermédia ou Normal.
- **Produtos Farmacêuticos** poderão ter Prescrição ou não.

A aplicação já é disponibilizada com um ficheiro texto que contém informação sobre 4 clientes, 7 faturas, e pelo menos dois produtos de cada tipo.
Vamos analisar as parcelas principais deste programa, analisar a sua composição, e os métodos que desta forma foram implementados para garantir a maior eficácia na sua execução.

Nota: A estrutura principal desta aplicação pode ser consultada através da documentação gerada pelo Javadoc. Para esse efeito, abrimos o arquivo "index.html" no zip "javadoc", onde está detalhada a descrição das classes e métodos do projeto.Caso queira compilar novamente o javadoc deve correr – "javadoc -private *.java" – e para correr – "start index.html"

Aplicação de Metodologias Orientadas a Objetos

Neste projeto, foram implementados conceitos de programação orientada a objetos, para estruturar a aplicação de forma eficiente e modular. Tais conceitos são fundamentais para promover a reutilização de código, facilitar a manutenção, e garantir uma melhor organização da aplicação.

Polimorfismo

O Polimorfismo permite que métodos com o mesmo nome tenham comportamentos diferentes dependendo do objeto.

Por exemplo, na class "Produto" implementamos o método abstrato "calcularTaxaIVA" que por sua vez é implementado em cada subclasse consoante a localização do cliente e consoante atributos específicos (se o produto alimentar for biológico é retirado 10% à sua taxa de IVA).

Herança

A Herança é um mecanismo que permite que uma classe (ou subclasse) herde atributos e métodos de outra classe (ou superclasse).

Por exemplo, a classe "Produto" serve de Superclasse para as classes dos Produtos Alimentares e de Farmacêuticos.

Isto permite que as subclasses reutilizem o código da Superclasse, permitindo melhor gestão e evitando redundâncias no código.

Todas as classes dos Produtos herdam propriedades como o "nome", "codigo", "descricao", "quantidade" e "valorUnitario", enquanto cada tipo de produto pode adicionar as suas características específicas (o produto farmacêutico com prescrição terá atributos como o nome do médico e qual a prescrição, enquanto um produto alimentar de taxa reduzida terá um atributo com as suas certificações).

Classe “Produto.java” e Subclasses

Classe que representa um produto genérico. Contém os atributos comuns a todos os produtos, como “codigo”, “nome”, “descricao”, “quantidade” e “valorUnitario”. Inclui ainda métodos para calcular o valor com, sem e do IVA do produto, que são depois sobrepostos pelas subclasses.

Subclasse: ProdutoAlimentar.java

Classe Abstrata dos produtos Alimentares.

Inclui um atributo comum a todas os produtos Alimentares que é se é biológico ou não.

Subclasse: ProdutoAlimentarTaxaIntermédia.java

Representa produtos alimentares com taxa intermédia de IVA.

Inclui a categoria destes produtos (congelados, enlatados ou vinho) representada por uma enumeração – “categoriaAlimentar”.

Subclasse: ProdutoAlimentarTaxaNormal.java

Representa produtos alimentares com taxa normal de IVA.

Não inclui atributos adicionais.

Subclasse: ProdutoAlimentarTaxaReduzida.java

Representa produtos alimentares com taxa reduzida de IVA.

Inclui um Set para armazenar as certificações do produto, que podem ser ISO22000, FSSC22000, HACCP, GMP, representadas por uma enumeração – “categoriaFarmacia”.

Subclasse: ProdutoFarmaciaComPrescricao.java

Representa os produtos de farmácia que requerem prescrição médica.

Inclui atributos para armazenar informações sobre a prescrição e o médico responsável.

Subclasse: ProdutoFarmaciaSemPrescricao.java

Representa os produtos de farmácia que não requerem prescrição.

Inclui a categoria produto que pode ser beleza, bem-estar, bebés, animais ou outros, que foram representadas por meio de uma enumeração - “categoriaFarmacia”.

Classe “Cliente.java”

Cada cliente é caracterizado pelo seu nome, número de contribuinte e localização (Portugal Continental, Madeira ou Açores).
A classe também mantém uma lista de faturas associadas a cada cliente.

Atributos

- “nome” » Cadeia para armazenar o nome do cliente
- “contribuinte” » Inteiro que representa o número de contribuinte do cliente
- “localização » Enumeração com as possíveis localizações do cliente
- “id” » Inteiro que identifica cada cliente
- “faturas” » Lista que contém as faturas do cliente

Métodos Principais

- “getLocalizacaoFormatada()” – Método para obter a localização formatada
- “adicionarFatura(Fatura fatura)” – Método para adicionar uma fatura
- “getNumeroFaturas” – Método para saber número de faturas de um cliente
- “searchFaturaNumero” – Método para procurar uma fatura pelo seu número

Nota: Esta classe implementa a interface “Serializable”, que permite que os objetos do cliente sejam serializados, ou seja, convertidos em formatos de byte stream, permitindo guardar de forma correta os dados num ficheiro binário. Esta interface foi aplicada a todos os objetos que iremos ver depois.

Classe “Fatura.java”

A classe Fatura representa uma fatura emitida para um cliente. Cada fatura contém um número, uma data, um cliente associado e uma lista de produtos.
A classe também fornece métodos para calcular o total com, sem e do IVA da fatura, imprimir a fatura e os seus produtos, bem como tratar de editá-los e de os adicionar.

Atributos

- “numero” » Inteiro que identifica a fatura
- “cliente” » Objeto “Cliente” que recebeu a fatura
- “data” » Objeto da classe “Data” que representa a data da fatura
- “produtos” » Lista de produtos da fatura

Métodos Principais

- “adicionarProduto(Produto produto)” » Método para adicionar um produto à lista de produtos da fatura
- “inputProduto(Scanner scanner)” » Método para ler input de um produto
- “editarProduto(String codigo, Scanner scanner)” » Método para editar um produto
- “removerProduto(String codigo)” » Método para remover um produto
- “imprimirFatura(boolean detalhada)” » Método para imprimir uma fatura (se o argumento “detalhada” for verdadeiro imprimimos a fatura conforme a opção 7, caso contrário é relativo à opção 6)
- “listarProdutos()” » Método para listar os produtos de uma fatura
- “calcularTotalSemIVA()” » Método que devolve o total da fatura sem IVA
- “calcularTotalComIVA()” » Método que devolve o total da fatura com IVA
- “calcularTotalDoIVA()” » Método que devolve o total de IVA da fatura

Classe “Data.java”

A classe Data representa uma data específica, com o dia, mês e ano.
É utilizada pararegistar a data das faturas e pode incluir métodos para manipulação e formatação de datas.

Atributos

- “dia” » Inteiro que representa o dia
- “mes” » Inteiro que representa o mês
- “ano” » Inteiro que representa o ano

Métodos

- “getDia()” » Devolve o dia da data
- “getMes()” » Devolve o mês da data
- “getAno()” » Devolve o ano da data
- “toString()” » Devolve a string da data no formato "dia/mês/ano"

Esta classe implementa igualmente a interface “Serializable”, como explicado anteriormente.

Classe “Auxiliar.java”

A classe Auxiliar contém métodos de input de dados do utilizador, e outras operações auxiliares.
Esta classe é projetada para ser uma classe estática, onde os métodos podem ser chamados sem precisar de instanciar um objeto.

Métodos

- “lerString(String msg, String scanner, boolean editar)” » Método que devolve uma cadeia. Permite retornar uma cadeia vazia caso “editar” seja verdadeiro
- “lerInteiro(String msg, Scanner scanner, boolean contribuinte)” » Método que devolve um inteiro. Caso contribuinte seja verdadeiro, obriga o utilizador a introduzir um número de 9 dígitos
- “lerDouble(String msg, Scanner scanner)” » Método que devolve um double
- “lerBooleano(String msg, Scanner scanner)” » Método que devolve “true” se o input for “S”, false se o input for “N”
- “lerCertificacoes(String msg, Scanner scanner)” » Método que devolve as certificações do produto (retorna sempre pelo menos uma certificação)
- “lerLocalizacao(String msg, boolean editar, Scanner scanner)” » Método que devolve a localização do cliente. Permite retornar um “null” caso “editar” seja verdadeiro
- “lerData(String msg, boolean editar, Scanner scanner)” » Método que devolve a data da fatura. Permite retornar “null” caso “editar” seja verdadeiro
- “lerCategoriaAlimentar(String msg, Scanner scanner)” » Método que devolve a categoria alimentar
- “lerCategoriaFarmacia(String msg, Scanner scanner)” » Método que devolve a categoria farmacêutica
- “binFileExists(String filePath)” » verifica se o arquivo existe no path especificado

Classe “POOFS.java”

A classe POOFS coordena a lógica do programa, incluindo a criação e edição de clientes e faturas. Através desta classe, mantemos uma lista de clientes, que têm uma lista de faturas, que por sua vez têm uma lista de produto.
A classe tem métodos para realizar operações sobre eles, como por exemplo criar e editar faturas e clientes, listar os mesmos, bem como apresentar estatísticas sobre os dados da aplicação.
Além disso, a classe é responsável por carregar e exportar dados de e para ficheiros, garantindo assim o ciclo de informações ativo. Da primeira vez que o programa corre, a aplicação irá ler os dados no ficheiro de texto. A partir da segunda de execução, a aplicação lerá e escreverá todos os dados da aplicação para um ficheiro binário. O método responsável por escrever os dados para a aplicação binária é chamado de cada vez que editamos ou criamos um cliente ou uma fatura na aplicação garantindo a segurança da informação.

Atributos

- “clientes” » Lista com todos os clientes registados na aplicação POOFS

Métodos

- “criarCliente(Scanner scanner)” » Método que cria um cliente a partir do input do utilizador, e adiciona esse cliente à lista de clientes
- “editarCliente(Scanner scanner)” » Método que permite editar as informações de um cliente existente
- “listarClientes()” » Método que lista todos os clientes registados na aplicação
- “criarFatura(Scanner scanner)” » Método que cria uma fatura para um cliente selecionado. O utilizador tem de introduzir uma lista de produtos (pelo menos 1)
- “editarFatura(Scanner scanner)” » Método que permite editar uma fatura existente, incluindo adicionar, remover ou editar produtos (nome, descrição e quantidade)
- “listarFaturas()” » Método que lista todas as faturas registadas na aplicação
- “visualizarFatura(Scanner scanner)” » Método que permite visualizar os detalhes de uma fatura específica
- “estatisticas()” » Método que apresenta as estatísticas sobre o número total de faturas, o número total de produtos, e o valor total com, sem e do IVA de todas as faturas registadas

- “loadTxt”(String filePath)” » Método que carrega os dados iniciais dos clientes e das faturas a partir de um ficheiro de texto
- “loadBin(String filePath)” » Método que carrega os dados dos clientes e das faturas a partir de um ficheiro binário
- “exportBin()” »Método que exporta os dados dos clientes para um ficheiro binário
- “searchClientePorId(int id)” » Método Auxiliar que verifica a existência de um cliente na aplicação. Caso encontre retorna o cliente, caso contrário, retorna “null”
- “getNumeroFaturas()” » Método que retorna o número de faturas totais registadas na aplicação

Classe “Main.java”

A classe Main é o ponto de entrada da aplicação, e é responsável pela UI do programa.
Contém o método “main”, que inicia a execução do programa, permitindo que o utilizador consiga interagir com o sistema através de um menu de opções no terminal.
Esta classe é responsável por chamar os métodos da aplicação POOFS.

Métodos

- “main(String[] args)” » Método principal que inicializa a aplicação, apresentando um menu ao utilizador, que processa as opções introduzidas, chamando os métodos do objeto POOFS para executar as mesmas operações.

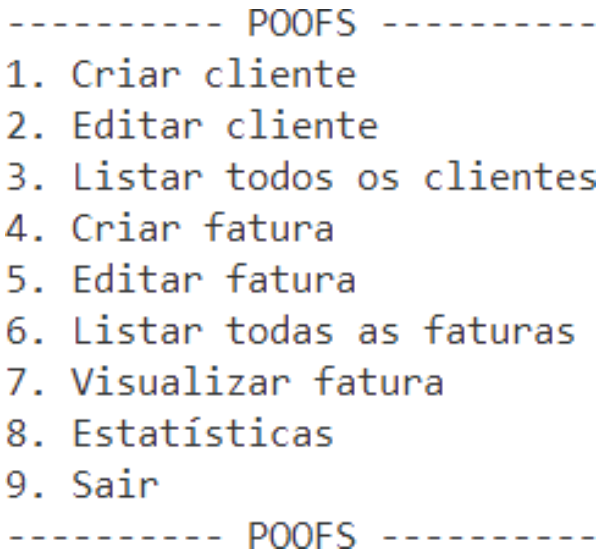

Fig. 1 – Menu principal da aplicação POOFS

Considerações Finais

Este projeto demonstra a eficácia da metodologia orientada a objetos na construção de sistemas de gestão financeira.

A estrutura hierárquica das classes permitiu-nos reutilizar de forma eficiente ocódigo, e manipular o mesmo para diferentes tipos de produtos. Isto permitiu a adição de novos produtos sem reescrever a lógica já existente.

A classe "Auxiliar" garante inputs seguros e válidos pelo utilizador, essencial para a precisão dos dados numa aplicação financeira. Além disso, a capacidade de carregare exportar dados assegura a persistência das informações.

Por fim, a Interface do User (UI), embora simples e diretamente implementada no Terminal, é intuitiva e permite que os utilizadores realizem todas as operações de forma mais eficiente.

A consulta dos parâmetros, classes, hierarquias e outras características do programa pode ser feita através do ficheiro index.html, que se encontra no zip "javadoc".

Relativamente ao UML inicial e ao Final da nossa aplicação, encontramos algumas diferenças como:

- No início tínhamos apenas uma classe Alimentar com um método para calcular a taxa de IVA. No entanto, concluímos que seria mais eficaz se existisse uma classe para cada tipo de taxa otimizar a aplicação;
- Antes o POOFS, em vez de ter uma lista de clientes com as suas faturas, tinha uma lista de faturas com os seus clientes la dentro. Desta maneira não permitíamos a criação de clientes sem existir uma fatura antes;
- Foi adicionada também a classe "Auxiliar" que permite que leiamos dados do utilizador de forma segura evitando erros;

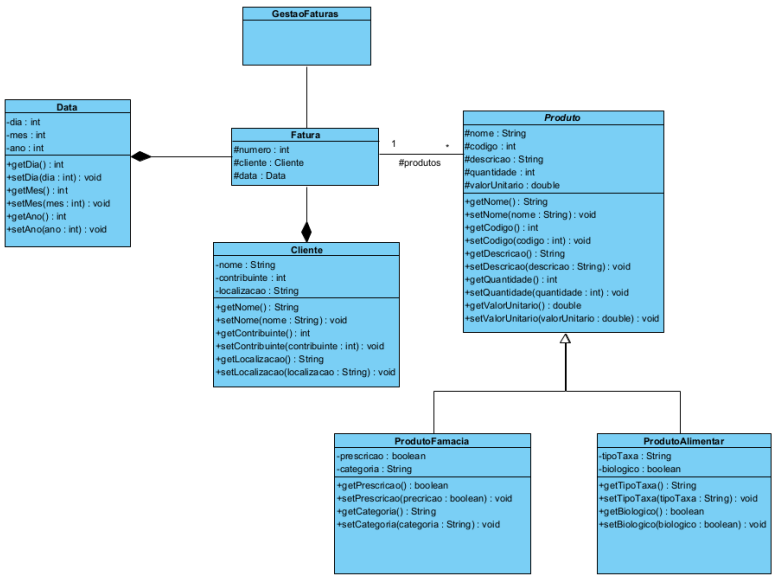

Fig. 2 - UML inicial

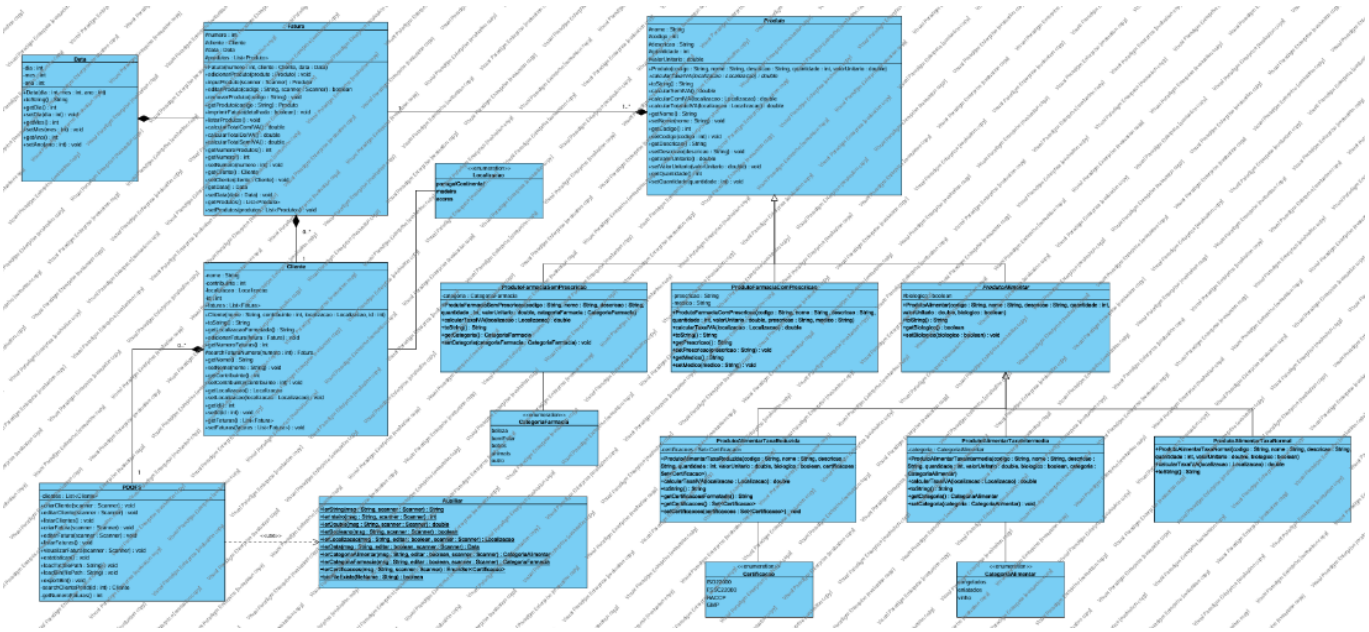

Fig. 2 - UML final