

1 2



9 0

FACULDADE DE  
CIÊNCIAS E TECNOLOGIA  
UNIVERSIDADE DE  
COIMBRA

Licenciatura em Engenharia Informática  
Ano letivo 2024/2025

# **RELATÓRIO PROJETO “POOFS”**

no âmbito da cadeira de  
**Programação Orientada aos Objetos**

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

# ÍNDICE

---

<b>Introdução</b>	2
<b>Aplicação de Met. Orientadas a Objetos</b>	3
<b>Classe “Produto.java”</b>	4
<b>Classe “Cliente.java”</b>	6
<b>Classe “Fatura.java”</b>	7
<b>Classe “Data.java”</b>	8
<b>Classe “Auxiliar.java”</b>	9
<b>Classe “POOFS.java”</b>	10
<b>Classe “Main.java”</b>	11
<b>Considerações Finais</b>	12

---

# **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 n.º de contribuinte, e localização (Portugal Continental, Madeira ou Açores) associados.*

*Cada **Fatura** tem um número, o cliente em si, 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.*

*Na Database do projeto foram já introduzidos 3 Clientes, com as respectivas Faturas e Produtos.*

*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.*

## ***Aplicação de Metodologias Orientadas a Objetos***

*No nosso projeto, foram implementados conceitos de Orientação a Objetos, para estruturar a aplicação de forma eficiente e modular.*

### ***Polimorfismo***

*O Polimorfismo permite que métodos com o mesmo nome tenham comportamentos diferentes, dependendo do objeto, e neste caso esse conceito era extremamente importante.*

*Por exemplo, a partir da Classe “Produto” podemos ver que subclasses como “ProdutoAlimentarTaxaIntermedia” ou “ProdutoFarmaciaSemDescricao” implementam as suas próprias versões dos métodos utilizados, permitindo assim que o programa trate os produtos diferentes de uma forma uniforme, mas com comportamentos específicos a cada classe.*

### ***Herança***

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

*Por exemplo, a classe “Produto” serve de Superclasse para as classes dos Produtos Alimentares e de Farmácia.*

*Isto permite que as subclasses reutilizem o código da Superclasse, permitindo a melhor gestão e organização dos métodos.*

*Todas as classes dos Produtos herdam propriedades como o “nome”, “codigo”, “descricao”, “quantidade” e “valorUnitario”, enquanto que cada tipo de produto pode adicionar as suas características específicas.*

## **1. Classe “Produto.java”**

*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 preço com e sem IVA dos produtos, que serão depois sobrepostos pelas subclasses*

### *Subclasse: ProdutoAlimentar.java*

*Representa os produtos alimentares. Esta classe pode ser ainda subdividida em diferentes categorias de produtos alimentares.*

*Esta classe está dividida em 3 subclasses.*

### *Subclasse: ProdutoAlimentarTaxaIntermédia.java*

*Representa produtos alimentares que estão sujeitos a uma taxa intermédia de IVA.*

*Inclui ainda atributos específicos como a categoria alimentar (congelados, enlatados, vinhos).*

### *Subclasse: ProdutoAlimentarTaxaNormal.java*

*Representa produtos alimentares que estão sujeitos à taxa normal do IVA.*

*Esta classe pode incluir atributos adicionais relacionados a produtos normais.*

### *Subclasse: ProdutoAlimentarTaxaReduzida.java*

*Representa produtos alimentares com taxa reduzida de IVA.*

*Inclui atributos para certificações, que podem ser ISO22000, FSSC22000, HACCP, GMP*

### *Subclasse: ProdutoFarmacia.java*

*Extensão da classe “Produto”, que representa produtos de farmácia.  
Esta classe é dividida em duas subclasses.*

Subclasse: ProdutoFarmaciaComPrescricao.java

*Representa produtos de farmácia que requerem prescrição médica.*

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

Subclasse: ProdutoFarmaciaSemPrescricao.java

*Representa produtos de farmácia que não requerem prescrição. Inclui categorias como beleza, bem-estar, bebês, animais ou outros.*

## 2. Classe “*Cliente.java*”

*A classe Cliente representa um cliente que pode receber faturas. Cada cliente é caracterizado por um nome, um número de contribuinte e uma localização (Portugal Continental, Madeira ou Açores). A classe também mantém uma lista de faturas associadas a esse cliente.*

### **Atributos**

- “nome” » string para armazenar o nome do cliente
- “contribuinte” » inteiro que representa o n° de contribuinte do cliente
- “localização” » Enum que aponta a localização do cliente
- “id” » inteiro que identifica cada cliente
- “fatura” » lista que contém as faturas do cliente

### **Métodos**

- “adicionarFatura(Fatura fatura)” » adiciona uma fatura à lista de faturas do cliente
- “getNumeroFaturas()” » devolve o n° total de faturas associadas ao cliente
- “getFaturas()” » devolve a lista de faturas do cliente
- “getId()” » devolve o ID do cliente
- “toString()” » devolve a string do cliente, com o nome, contribuinte e localização

### **3. 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 que compõem a fatura. A classe também fornece métodos para calcular o total da fatura, incluindo e excluindo o IVA.*

#### **Atributos**

- “numero” » inteiro que identifica a fatura
- “cliente” » referenciamos o objeto “Cliente” que recebeu a fatura
- “data” » objeto da classe “Data” que representa a data da fatura
- “produtos” » lista de produtos da fatura

#### **Métodos**

- “adicionarProduto(Produto produto)” » adiciona um produto à lista de produtos da fatura
- “calcularTotalSemIVA()” » calcula e devolve o total da fatura sem IVA
- “calcularTotalComIVA()” » calcula e devolve o total da fatura com IVA
- “calcularTotalDoIVA()” » calcula e devolve o IVA aplicado
- “toString()” » devolve a string do cliente, com o número, a data, o cliente e os produtos



## **4. Classe “Data.java”**

*A classe Data representa uma data específica, com o dia, mês e ano. É utilizada para registrar 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”

## **5. Classe “Auxiliar.java”**

*A classe Auxiliar contém métodos de input de dados do user, 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()” » lê a string do user*
- *“lerInteiro()” » lê o inteiro do input do user*
- *“lerBooleano()” » lê um booleano de input do user*
- *“lerLocalizacao()” » lê e devolve a localização do cliente a partir do input*
- *“lerData()” » lê e devolve a data a partir do input do user*
- *“verificarExistenciaArquivo(String filePath)” » verifica se o arquivo existe no path especificado*

## 6. 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 e fornecemos métodos para realizar operações sobre eles, como por exemplo criar faturas, editar clientes e listar informações. Além disso, a classe é responsável por carregar e exportar dados de e para arquivos, garantindo assim o ciclo de informações ativo.*

### **Atributos**

- “clientes” » lista com todos os clientes registados no programa

### **Métodos**

- “criarCliente(Scanner scanner)” » cria um novo cliente a partir do input do user, e adiciona esse cliente à lista de clientes
- “editarCliente(Scanner scanner)” » permite editar as informações de um cliente existente
- “listarClientes()” » exhibe todos os clientes registados na aplicação
- “criarFatura(Scanner scanner)” » cria uma nova fatura para um cliente selecionado, permitindo também adicionar produtos
- “editarFatura(Scanner scanner)” » permite editar uma fatura existente, incluindo adicionar, remover ou editar produtos
- “listarFaturas()” » exhibe todas as faturas registadas na aplicação
- “visualizarFatura(Scanner scanner)” » permite visualizar os detalhes de uma fatura específica
- “estatisticas()” » apresenta as estatísticas sobre nº de faturas, produtos e valores totais
- “loadTxt”(String filePath)” » carrega os dados dos clientes e faturas a partir de um ficheiro de texto
- “loadBin”(String filePath)” » carrega os dados dos clientes e faturas a partir de um ficheiro binário
- “exportBin()” » exporta os dados dos clientes para um arquivo binário

## **7. 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 user consiga interagir com o sistema através de um menu de opções no terminal. Esta classe é responsável por chamar os métodos como criar clientes, editar faturas, listar faturas, etc...*

### **Métodos**

- “main(String[] args)” » método principal que inicia a aplicação, apresenta um menu ao user, e processa as opções introduzidas, chamando os métodos da classe POOFS para executar as mesmas operações.

## ***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 o código, 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 user, essencial para a precisão dos dados numa aplicação financeira. Além disso, a capacidade de carregar e exportar dados assegura a persistência das informações.*

*Por fim, a interface de usuário (UI), embora simples e diretamente implementada no Terminal, é intuitiva e permite que os users realizem todas as operações de forma mais eficiente.*