



Universidade do Minho
Escola de Engenharia

TRABALHO PRÁTICO

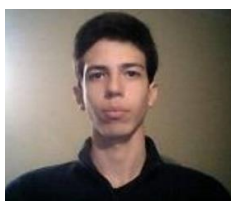
Mestrado Integrado em Engenharia Informática

Programação Orientada aos Objetos

(2º semestre, 2º ano)

Grupo 7

A78322 André Filipe Ferreira de Mira Vieira **A77048** Eduardo Gil Ribeiro Rocha



A78764 Ricardo André Araújo Neves



Data:

27 de maio de 2018

Índice

Índice	2
Resumo	3
Introdução	4
Entidades do Sistema	5
Estruturas de Dados Usados	7
Funcionalidades Implementadas	8
Instruções	9
Discussão	10
Conclusão	11

Resumo

Este relatório serve como complemento ao trabalho prático da Unidade Curricular de Programação Orientada aos Objetos.

Aqui iremos expressar a nossa linha de pensamento e como realizamos o trabalho proposto, seguida de uma breve conclusão crítica sobre o projeto realizado pelo grupo.

A linguagem de programação que foi utilizada para a criação do Sistema de Software foi JAVA, abordada nas aulas durante todo o semestre. A IDE utilizada pelo grupo para o desenvolvimento do projeto foi IntelliJ.

Introdução

Este projeto final teve em vista a criação de uma aplicação “JavaFactura” que disponibiliza, aos contribuintes, informações relevantes relativas às faturas que são emitidas associadas ao seu NIF (número de identificação fiscal). O sistema deve também ser capaz de permitir a Empresas que lancem novas faturas.

Como se pode ver, é uma aplicação que tem funcionalidades semelhantes ao do sistema e-Fatura, um sistema oficial das Finanças portuguesas.

Esta aplicação inclui uma série de requisitos funcionais, que serão abordados mais à frente neste documento.

Entidades do Sistema

A aplicação foi implementada com atenção aos dois principais tipos de utilizadores, ou seja: os Contribuintes Individuais e as Empresas. Estas entidades partilham certas informações: NIF, email, nome, morada e password de acesso ao sistema, que são contidas dentro da Classe que ambos estendem, Contribuinte. Tanto os Contribuintes Individuais como as Empresas possuem características adicionais que os distinguem entre si.

Para além dessas entidades, criamos também uma classe Despesa, que define o formato assumido pelas despesas que estão envolvidas no sistema.

```
private String NIF;  
private String email;  
private String nome;  
private String morada;  
private String password;
```

Figura 1 – Estrutura da Classe Contribuinte

```
private HashMap<String,String> ativRealiza;  
private float factor;
```

Figura 2 – Estrutura da Classe Empresa

```
private int numDependentes;  
private float coeficiente;  
private HashMap<String,String> NIFs;  
private HashMap<String,String> ativDeduzir;
```

Figura 3 – Estrutura da Classe Individual

As classes Main e Processamento são classes utilizadas para suportar as funcionalidades do sistema. A class Main é a responsável por iniciar o sistema e apresentar os menus da interface, enquanto que Processamento contem todos os algoritmos que realizam os diversos requisitos do trabalho, assim como os dados que são utilizados por esses mesmos métodos.

No desenvolvimento do projeto deste ano letivo, tivemos em atenção tentar não repetir os mesmo erros que fizeram-nos obter nota negativa no ano letivo anterior.

Um dos erros que tentamos evitar foi a má estrutura dos dados do sistema. Para corrigir isto, tentamos encapsular as diversas classes de uma forma que faz sentido, tentando evitar uso excessivo de recursos e tentando também aproveitar ao máximo as capacidades únicas do paradigma da Programação Orientada a Objetos. Logo, a classe Processamento é a que contem os dados da aplicação, que carrega a partir de ficheiros quando é criada uma instância dessa classe pela classe Main. A class Processamento é a que realiza as várias tarefas de processamento de dados dentro da aplicação, logo consideramos que seria o melhor lugar para manter os dados. Porém, temos atenção em usar clones caso seja necessário realizar alterações a dados que estamos a tentar preservar.

Estruturas de Dados Usados

Outro erro cometido por nós no ano anterior, para além dos discutidos no capítulo superior, foi o uso incorreto de Estruturas de Dados. Este ano decidimos tirar melhor partido das funcionalidades do JAVA, para poder criar um projeto com melhor desempenho.

Para conter os dados do sistema, definimos três HashMap. Uma contendo as várias Empresas, outra contendo os Contribuintes Individuais e a última contendo as várias Despesas.

```
private HashMap<String, Empresa> empresas;  
private HashMap<String, Individual> individuais;  
private HashMap<String, HashMap<Integer, Despesa>> despesas;
```

Figura 4 – HashMaps principais usadas

Como podemos verificar na figura acima, o terceiro HashMap usada contem dentro de si vários HashMaps. Esta decisão foi tomada pois tínhamos uma necessidade de catalogar conjuntos de Despesas com uma chave que fosse única a uma despesa ou um grupo de despesas. Visto que uma Empresa pode realizar faturas várias vezes ao mesmo cliente, decidimos agrupar despesas tal que cada HashMap exterior tenha como chave o NIF da Empresa emissora. As despesas do HashMap interior são catalogadas com chaves unicas.

Em certos processos, usamos também ArrayLists como estruturas auxiliares à resolução do problema. Usamos ArrayLists apenas em casos onde temos de iterar sobre o conjunto de dados inteiro.

```
ArrayList<Despesa> despesasAux = new ArrayList<>();
```

Figura 5 – Exemplo de uso de ArrayList

Funcionalidades Implementadas

Neste trabalho, implementamos várias funcionalidades requisitadas pelo enunciado.

O sistema permite aos utilizadores existentes fazer login por avenidas diferentes dependentemente de se são Empresas ou Contribuintes. Permite também a um Administrador entrar no sistema.

Para utilizadores que não possuem contas, o sistema permite também criar novas contas de Contribuintes Individuais ou de Empresas.

Para as empresas, é-lhes permitido:

- Criar novas faturas associadas a um contribuinte individual
- Associar novas atividades económicas a documentos de despesa existentes
- Rastrear histórico de mudança de classificação de documentos de despesa
- Obter listagem de facturas da empresa, ordenadas por ordem decrescente de valor
- Obter listagem de facturas da empresa por contribuinte num determinado intervalo de datas
- Obter listagem de facturas da empresa por contribuinte ordenadas por valor decrescente de despesa
- Indicar o total faturado pela empresa num determinado período

Os Contribuintes Individuais são capazes de verificar quais despesas foram emitidas em seu nome e verificar o montante de dedução fiscal acumulado, por si e pelo seu agregado familiar.


```

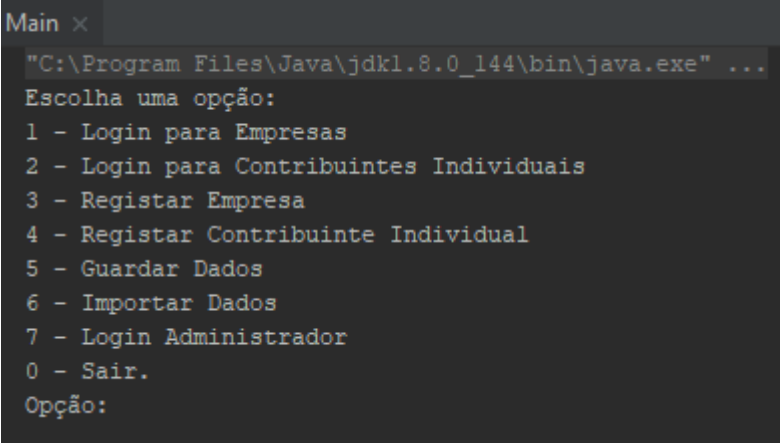
public void totalFaturado(String NIF) {
    HashMap<Integer, Despesa> despesas = this.despesas.get(NIF);
    Date[] data = scannerData();
    int total = 0;
    for(Despesa d : despesas.values()){
        if(!d.getData().before(data[0]) && !d.getData().after(data[1])) {
            total += d.getValor();
        }
    }
    System.out.print("Total faturado: "+total+"\n");
}

```

Figura 6 – Exemplo de funcionalidade implementada (calcular total faturado por empresa)

Instruções

O nosso trabalho foi realizado utilizando *IntelliJ*, logo, para executá-lo, basta importar o projeto, compilá-lo e corrê-lo. Após isto, o terminal interior da IDE irá mostrar a interface do programa.



```

Main x
"C:\Program Files\Java\jdk1.8.0_144\bin\java.exe" ...
Escolha uma opção:
1 - Login para Empresas
2 - Login para Contribuintes Individuais
3 - Registar Empresa
4 - Registar Contribuinte Individual
5 - Guardar Dados
6 - Importar Dados
7 - Login Administrador
0 - Sair.
Opção:

```

Figura 7 – Exemplo do programa a correr dentro do IntelliJ

Discussão

Durante um possível desenvolvimento futuro da aplicação, poderia ser necessário a inclusão de novos tipos de Despesa ou novos algoritmos de cálculo de deduções fiscais.

Para a inclusão de novos tipos de Despesa, poderíamos recorrer a uma estratégia semelhante à que já usamos neste trabalho para a distinção entre Contribuintes Individuais e Empresas, ou seja, criar uma classe abstrata denominada “Despesa”, com várias outras sub-classes a estender essa classe inicial. Porém, as características desse problema hipotético poderão não necessitar essa solução.

A implementação de um novo algoritmo poderia passar pelo cálculo de novos valores de despesa, baseados no valor original de cada despesa processada e baseado também no catálogo de atividades realizadas pelo contribuinte em questão.

Conclusão

Em conclusão, achamos importante referir que neste ano letivo desenvolvemos o projeto de Trabalho Prático desta Unidade Curricular com muito maior confiança nas nossas capacidades de produzir código competente de Java. Conseguimo-nos livrar de várias dúvidas que tínhamos em relação ao uso de Estruturas de Dados de Java, assim como em relação à estrutura e organização de um Sistema de Software realizado numa linguagem pertencente ao paradigma da Programação Orientada em Objetos.

Apesar disto, e apesar do trabalho realizado, lamentamos não ter-nos sido possível completar os requisitos adicionais referidos no enunciado do Trabalho Prático.