

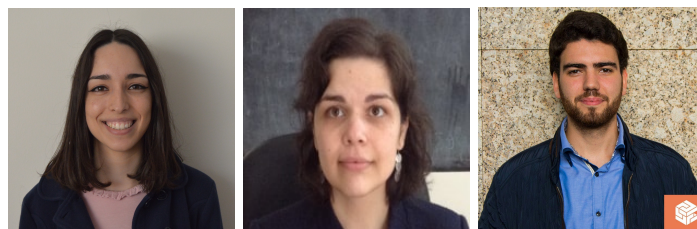
UNIVERSIDADE DO MINHO

DEPARTAMENTO DE INFORMÁTICA

Trabalho Prático
Programação Orientada a Objetos
Grupo 18

Catarina Machado (a81047) Cecília Soares (a34900)
João Vilça (a82339)

27 de Maio de 2018



(a) Catarina.

(b) Cecília.

(c) João.

Resumo

O presente relatório descreve o trabalho prático realizado no âmbito da disciplina de *Programação Orientada a Objetos* (POO), ao longo do segundo semestre, do segundo ano, do Mestrado Integrado em Engenharia Informática da Universidade do Minho.

O objetivo do projeto foi efetuar uma plataforma informática, semelhante à do e-factura, que consiste em disponibilizar aos contribuintes a informação referente às faturas emitidas em seu nome.

Neste documento descrevemos sucintamente a aplicação desenvolvida, as classes e interfaces utilizadas, bem como discutimos as decisões tomadas durante o projeto.

Conteúdo

1	Introdução	3
1.1	Descrição do Problema	3
1.2	Conceção da Solução	4
2	Classes	4
2.1	Contribuinte	4
2.1.1	Contribuinte Individual	5
2.1.2	FamiliaNumerosa	5
2.1.3	Contribuinte Coletivo	5
2.2	Factura	6
2.3	Concelho	7
2.4	Atividades Económicas	7
2.4.1	Subclasses Económicas	7
2.5	EmpresasTop	9
2.6	JavaFactura	9
3	Interfaces	9
3.1	Incentivo Fiscal	9
4	Classes de Exceções	10
4.1	CINullException	10
4.2	CCNullException	10
4.3	FacturaNaoExisteException	10
4.4	CSVformatIncorretException	10
5	Implementação e Manual de Utilizador	10
5.1	Registar Contribuintes	11
5.2	Login	12
5.3	Criar Faturas	15
5.4	Verificar Despesas e Montantes Dedução Fiscal	17
5.5	Associar CAE	17
5.6	Mudar CAE	19
5.7	Listagem das Faturas Empresas (Data e Valor)	21
5.8	Listagem das Faturas Empresas por Contribuinte (e Data)	24
5.9	Listagem das Faturas Empresas por Contribuinte (e Valor)	24
5.10	Total faturado por uma empresa num determinado período	25
5.11	Coleção dos 10 Contribuintes que mais gastam	26
5.12	Coleção X Empresas	27
5.13	Gravar Estado da Aplicação em Ficheiro	28
5.14	Famílias Numerosas e Empresas do Interior	29
5.15	Criação de Agregados	30
5.16	Download e Impressão de Facturas	32
5.17	Exportar e Importar informação em formato CSV	33
6	Conclusões	34

1 Introdução

O presente relatório foi elaborado no âmbito da unidade curricular *Programação Orientada a Objetos* (POO), ao longo do segundo semestre, do segundo ano, do Mestrado Integrado em Engenharia Informática da Universidade do Minho, e tem como objetivo a criação de uma plataforma onde os contribuintes possam acompanhar as suas faturas e os montantes de todas as deduções a que têm direito, entre muitas outras operações que serão explicitadas em seguida e que são intrínsecas a esta principal funcionalidade.

1.1 Descrição do Problema

O projeto proposto implica desenvolver uma plataforma que permita a criação de contribuintes e empresas, a emissão de faturas, a associação das empresas a determinada(s) atividade(s) económica(s) e o cálculo dos montantes de deduções fiscais que cabe a cada contribuinte. Acresce que, a empresa deve ser capaz de emitir uma fatura a um determinado contribuinte, em virtude de um serviço prestado ou venda de bens ao mesmo. Por seu turno, o contribuinte poderá ter de associar as faturas ainda por classificar, caso em que a empresa tem mais do que um setor de atividade definido. A aplicação deve ainda guardar registo de todas as operações efetuadas e deve também ter mecanismos para as disponibilizar, por exemplo, as faturas emitidas por uma empresa, extrato de faturação de uma empresa num determinado período, valor total de despesas de um contribuinte, valor total de despesas de um contribuinte por atividade económica, entre outras. Finalmente, pretende-se que seja criada a definição de Família Numerosa e de Empresa do Interior, as quais concedem um benefício fiscal ao contribuinte que tem mais de quatro filhos ou que compra ou bem ou obtém um serviço de uma empresa sediada num concelho do interior, respetivamente.

De forma mais clara e sucinta os requisitos básicos do programa são os seguintes:

- Registrar um contribuinte, quer seja individual ou empresa;
- Validar o acesso à aplicação utilizando as credenciais (nif e password), por parte de diferentes atores;
- Criar faturas associadas a despesas feitas por um contribuinte individual. São as empresas que alimentam esta informação no sistema;
- Verificar, por parte do contribuinte individual, as despesas que foram emitidas em seu nome e verificar o montante de dedução fiscal acumulado, por si e pelo agregado familiar;
- Associar classificação de actividade económica a um documento de despesa;
- Corrigir a classificação de actividade económica de um documento de despesa. Esta alteração deve deixar registo para ser depois rastreada;
- Obter a listagem das faturas de uma determinada empresa, ordenada por data de emissão ou por valor;
- Obter por parte das empresas, as listagens das faturas por contribuinte num determinado intervalo de datas;
- Obter por parte das empresas, as listagens das faturas por contribuinte ordenadas por valor decrescente de despesa;
- Indicar o total faturado por uma empresa num determinado período;
- Determinar a relação dos 10 contribuintes que mais gastam em todo o sistema (esta operação deve ser só disponibilizada para o administrador da aplicação);

- Determinar a relação das X empresas que mais faturas em todo o sistema e o montante de deduções fiscais que as despesas registadas (dessas empresas) representam (esta operação deve ser só disponibilizada para o administrador da aplicação);
- Gravar o estado da aplicação em ficheiro, para que seja possível retomar mais tarde a execução.
- O sistema deverá ser capaz de gerir a definição de número de filhos para qualificação de uma família como sendo numerosa (mais de quatro filhos).
- O sistema deverá ser capaz de tratar de forma privilegiada uma Empresa do Interior, mantendo uma lista que associa os concelhos com incentivo ao valor (percentual) definido.

1.2 Conceção da Solução

Para resolvermos este problema foram cruciais três momentos. Numa primeira fase, analisamos o problema e implementamos as classes e interfaces que iríamos precisar para a realização do trabalho. Posteriormente, averiguamos todos os requisitos do trabalho prático e fizemos as funções que dão respostas aos objetivos do mesmo. Em último lugar, construímos uma interface gráfica de utilizador, com o auxílio da biblioteca *javafx*, que permite o acesso às funcionalidades implementadas.

O relatório está organizado da seguinte forma: a Secção 2 descreve as **classes** adoptadas, a Secção 3 descreve as **interfaces** usadas e a Secção 4 refere-se às **classes de exceções** utilizadas. A Secção 5 refere-se à **apresentação dos requisitos** do trabalho prático e respetiva **implementação e manual de utilizador**.

2 Classes

Para desenvolvermos este trabalho adotamos as seguintes classes, com o intuito de detalhar as características das principais entidades do sistema:

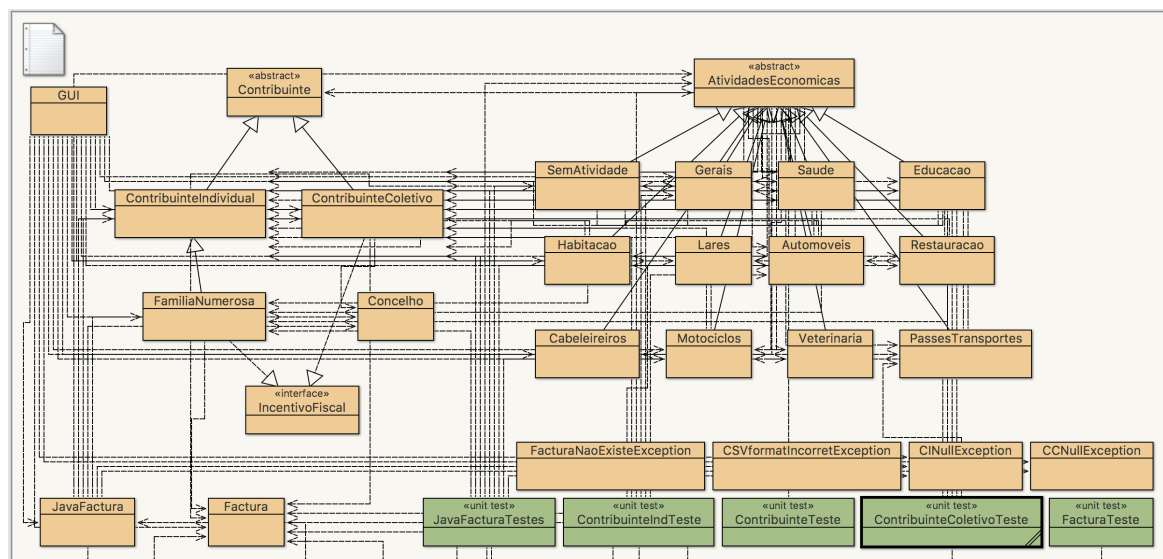


Figura 2: Imagem das classes BlueJ.

2.1 Contribuinte

A classe *Contribuinte* é uma classe abstrata que tem as seguintes variáveis de instância:

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

O NIF, email, nome, morada e password são atributos essenciais (informação comum) a todos os contribuintes (tanto coletivos como individuais) pelo que consideramos que seria mais útil e intuitivo elaborar esta classe abstrata.

2.1.1 Contribuinte Individual

A classe *ContribuinteIndividual*, extensão da classe *Contribuinte*, apresenta outras variáveis de instância que complementam a informação necessária para identificar um contribuinte individual:

```
private int n_direcao;
private int n_dependentes;
private List<Integer> nifsDependentesAgregado;
private List<Integer> nifsDirecaoAgregado;
private double coeficiente_fiscal;
private Map<Integer, ArrayList<Factura>> grupos_facturas;
```

A variável *n_direcao* traduz o número de pessoas que constituem a direção do agregado familiar (no máximo 2 contribuintes) e a variável *n_dependentes* traduz o número de dependentes do agregado (não há limite estipulado).

As listas de inteiros *nifsDependentesAgregado* e *nifsDirecaoAgregado* armazenam os NIFs dos dependentes e dos contribuintes a quem cabe a direção do agregado familiar, respetivamente.

A variável *coeficiente_fiscal* guarda o coeficiente fiscal para efeitos de dedução (fator multiplicativo que é associado a cada despesa elegível). O valor do coeficiente fiscal é de uma unidade, à qual acresce uma décima (0,1) por cada pessoa que integre o agregado familiar. Esta variável será tida em conta no cálculo das deduções do indivíduo e do respetivo agregado.

Por último, temos um “map” *grupos_facturas* onde a chave é o número que identifica o setor de atividade económica (no nosso programa cada setor corresponde a um número pré-definido, de 0 (sem atividade) até 11), e o valor corresponde a um *ArrayList* de faturas. Deste modo, através do código que identifica a atividade económica temos acesso a todas as faturas emitidas a um determinado contribuinte individual por setor económico.

2.1.2 FamiliaNumerosa

A classe *FamiliaNumerosa*, extensão da classe *ContribuinteIndividual*, tem uma única variável, a qual é uma variável de classe. Optamos por criar uma única variável porque esta classe difere da de *ContribuinteIndividual* somente no número de dependentes que integram um agregado familiar para efeitos fiscais. Nessa medida, uma instância de *FamiliaNumerosa* tem todos os atributos de um contribuinte individual e a característica de ter mais de 4 dependentes. Assim, a variável criada define o limite inferior do número de dependentes desta classe e optamos por classificá-la como sendo de classe e não de instância porque todos os objetos desta classe partilham obrigatoriamente o referido limite inferior. Ademais, esta classe foi criada para implementar o interface *IncentivoFiscal*, o qual possui um único método, *reducaoImposto*. Este método é definido nesta classe e concede uma bonificação de 5% por cada dependente que integra o agregado.

2.1.3 Contribuinte Coletivo

A classe *ContribuinteColetivo*, extensão da classe *Contribuinte*, apresenta outras variáveis de instância que complementam a informação necessária para identificar um contribuinte coletivo:

```
private List<Integer> atividades_economicas;
private double fator_deducao_fiscal;
private String concelho;
private List<Factura> facturas;
```

Para além das variáveis de instância definidas em *Contribuinte*, o contribuinte coletivo possui ainda uma lista de inteiros (*atividades_economicas*) onde estão armazenados os códigos das atividades económicas onde a empresa se insere (1 no mínimo e 11 no máximo). A variável *fator_deducao_fiscal* que representa o fator que a empresa tem no cálculo da dedução fiscal. O *concelho* que guarda o nome do concelho onde se localiza a sede da empresa. Esta informação é necessária uma vez que as empresas do interior terão um benefício extra associado. Finalmente, também possui ainda uma lista de *facturas* com todas as faturas emitidas pelo contribuinte coletivo.

2.2 Factura

A classe *Factura* apresenta as seguintes variáveis de instância, que são necessárias para a representação de uma fatura:

```
private int id;
private int nif_emitente;
private String designacao;
private LocalDate data;
private int nif_cliente;
private String descricao;
private int natureza;
private BigDecimal valor;
List<Integer> historico_caes;
```

Cada fatura é identificada através do seu id, o qual é único e fixado de forma sequencial, aquando da emissão da fatura e armazenado na variável id. O NIF do contribuinte coletivo que emitiu a fatura está identificado na variável *nif_emitente*. A *designacao* guarda a firma do emitente; a *data* indica a data em que a fatura foi emitida, no formato *LocalDate*; o *nif_cliente*, refere-se ao nif do contribuinte a quem a fatura foi emitida; a *descricao* da fatura é o campo onde se identifica o tipo de bem ou serviço transacionado e o *valor*, expresso no formato *BigDecimal*, é o valor da despesa efetuada pelo contribuinte.

Para além disso, cada fatura contém a sua *natureza*, a qual corresponde ao setor de atividade da empresa ou, no caso da empresa emitente ter mais do que um setor de atividade associado, a natureza assumirá o valor de zero até que o contribuinte associe o setor de atividade correspondente, classificando a despesa de acordo com a sua natureza correta. No caso do contribuinte não associar a despesa ao correto setor de atividade da empresa, a fatura não será considerada para efeitos fiscais, não obstante a fatura que identifica a despesa continua a estar associada ao contribuinte individual.

Por último, a fatura tem ainda uma lista de inteiros com o histórico dos setores de atividade (*historico_caes*) da fatura. Note-se que se a empresa só tiver um setor de atividade, a classificação da atividade económica (CAE) constante da fatura nunca pode ser alterada, pelo que a referida lista sempre é vazia. No entanto, se a empresa tiver mais do que um setor de atividade económica, o CAE de todas as faturas emitidas por si começa por ser 0 (sem atividade definida), informação que também é conservada no histórico da fatura. Aquando da alteração do CAE, por parte do contribuinte individual, o novo código do setor é adicionado à lista *historico_caes*.

É de salientar que as **faturas são partilhadas** entre os contribuintes individuais e os contribuintes coletivos, ou seja, por exemplo, se o contribuinte individual alterar o CAE da fatura o contribuinte coletivo terá conhecimento.

2.3 Concelho

```
private String nome;  
private double beneficio;  
  
static private Map<String, Concelho> concelhos;
```

Esta classe foi criada para que determinados concelhos possam conceder um incentivo fiscal. De facto, o que se pretendeu foi criar uma lista de concelhos de Portugal Continental e associar a determinados concelhos, os do interior do país, um benefício fiscal que será concedido ao contribuinte que compre um bem ou adquira um serviço a uma empresa sediada num concelho do interior. Esta classe possui duas variáveis de instância, **nome**, **beneficio**. A primeira indica o nome do concelho em questão e a segunda o benefício fiscal que lhe está associado. Além destas duas variáveis, esta classe contém uma variável de classe **concelhos**, que representa uma hashmap de todos os concelhos existentes na nossa aplicação, sendo a chave dessa mesma hashmap o nome do concelho.

2.4 Atividades Económicas

A necessidade da criação desta classe abstrata surgiu para que fosse mais fácil incluir novos tipos de despesas passíveis de dedução, sendo, por isso, mais eficaz criar uma classe superior que encerra os atributos comuns a todas as atividades, mas que não pode ser instanciada, e criar as diversas subclasses que descrevem o estado e o comportamento de cada uma das atividades económicas existentes que dão direito a dedução.

A classe *AtividadesEconomicas* é uma classe abstrata que tem as seguintes variáveis de instância:

```
private int codigo;  
private String setor;  
private double percentagem_deducacao;  
private BigDecimal limite_deducacao;  
static private Map<Integer, AtividadesEconomicas> atividades;
```

O código da atividade económica é um número pré-definido por nós para cada um dos setores, de modo a simplificar a sua utilização no programa. O **setor** armazena o nome da atividade económica.

A **percentagem_deducacao** e o **limite_deducacao** estabelecem, respetivamente, a percentagem e o limite de dedução de cada atividade económica. Convém referir que o limite das deduções é aplicável às despesas suportadas por cada um dos indivíduos que compõem o agregado familiar. Por outras palavras, cada membro do agregado familiar pode beneficiar de deduções até ao máximo definido para cada atividade económica.

Finalmente, implementamos uma variável de classe **atividades**, isto é, variável comum a todas as instâncias da classe, que representa uma hashmap de todas as atividades económicas, sendo que o setor económico de uma empresa terá de pertencer, obrigatoriamente, a uma ou várias destas atividades pré-definidas. Esta variável permite-nos guardar as informações que dizem respeito à globalidade das instâncias criadas de atividades.

2.4.1 Subclasses Económicas

A classe *SemAtividade* será utilizada nas faturas que não têm CAE definido logo à partida (faturas que foram emitidas por empresas que têm mais do que uma atividade económica). Esta "atividade" terá o código 0 no nosso programa.

```
public static final SemAtividade objeto = new SemAtividade();
```


Possui uma única variável que é de classe porque queremos uma única instância daquela classe no programa. Entretanto, essa variável encontra-se desde logo inicializada com os valores do código, do setor, da percentagem de dedução e do limite de dedução, todos eles atributos herdados da classe superior *AtividadesEconomicas* por razões de pragmaticidade, já que são imutáveis. De ressaltar que este procedimento verifica-se em todas as classes referente às atividades económicas criadas, mudando somente os valores dos referidos atributos.

Na classe *SemAtividade*, o código é o zero, o setor designa-se por Sem Atividade, a percentagem de dedução e o limite da dedução são ambos zero.

Na classe *Despesas Gerais Familiares*, o código é o um, o setor tem a designação de Gerais, a percentagem de dedução é de 35% sobre a despesa e o limite é de 100 euros.

Na classe *Saúde*, o código é o dois, o setor tem a mesma designação, a percentagem de dedução é de 15% sobre a despesa e o limite é de 400 euros.

Na classe *Educação*, o código é o três, o setor tem a mesma designação, a percentagem de dedução é de 30% sobre a despesa e o limite é de 400 euros.

Na classe *Habitação*, o código é o quatro, o setor tem a mesma designação, a percentagem de dedução é de 15% sobre a despesa e o limite é de 400 euros.

Na classe *Lares*, o código é o cinco, o setor tem a mesma designação, a percentagem de dedução para este setor é de 25% sobre a despesa e o limite da dedução é de 400 euros.

Na classe *Reparação de Automóveis*, o código é o seis, a designação é Automoveis, a percentagem de dedução é de 5% sobre a despesa e o limite é de 100 euros.

Na classe *Restauração e Alojamento*, o código é o sete, a designação é Restauracao, a percentagem de dedução é de 5% sobre a despesa e o limite é de 100 euros.

Na classe *Cabeleireiros*, o código é o oito, o setor tem a mesma designação, este setor não dá direito a qualquer tipo de dedução e, por isso, também não tem limite dedutível.

Na classe *Motociclos*, o código é o nove, o setor tem a mesma designação, a percentagem de dedução é de 5% sobre a despesa e o limite é de 100 euros.

Na classe *Atividades Veterinárias*, o código é o dez, o setor tem a designação de Veterinarias, a percentagem de dedução é de 5% sobre a despesa e o limite é de 100 euros.

Finalmente, a classe *PassesTransportes* é a atividade com o código onze, o setor tem a designação de PassesTransportes, o limite da dedução para este setor é 100 euros e a percentagem de dedução é de 5% sobre o valor da despesa.

Os valores de percentagem de dedução, assim como o limite acima referidos são essenciais para o cálculo do montante das deduções do contribuinte, bem como do seu agregado familiar. Outros fatores determinantes no cálculo destes montantes são o coeficiente fiscal do contribuinte, o número de elementos do agregado familiar e a sede da empresa que presta o serviço ou vende o bem. Ora, entendemos que o cálculo desse valor deveria variar em função da atividade económica, já que as despesas feitas pelo contribuinte devem ter tratamento diferente consoante a natureza da mesma.

A classe de Cabeleireiros não confere nenhum tipo de benefício fiscal.

No caso das atividades das classes Gerais, Lares e Veterinaria o cálculo das deduções de cada elemento do agregado familiar é feito multiplicando o valor da despesa pelo coeficiente fiscal e pela percentagem da dedução prevista para cada uma dessas atividades até ao limite de dedução previsto.

Por seu turno, no caso dos setores de atividade como a Saúde, a Educação e a Habitação, o cálculo do montante dos valores a deduzir tem em consideração o número de dependentes que integram o agregado familiar. De facto, se o contribuinte contrair uma despesa relativa a um desses setores de atividade e pertencer a uma família numerosa, o cálculo da dedução far-se-á nos termos anteriormente descritos, mas o limite da dedução inicialmente previsto aumenta. Esse aumento

varia de acordo com o número de dependentes, o limite é majorado em 5% por cada dependente. Por exemplo, um elemento de uma família com 5 dependentes que tenha contraído uma despesa de saúde verá o seu limite de dedução do setor aumentar da seguinte forma:

$$5\% * 5 * 400 = 100 \quad (1)$$

$$400 + 100 = 500 \quad (2)$$

Assim, as despesas de saúde passam a ter um novo limite de dedução que ascende a 500 euros para cada elemento do agregado familiar.

No caso das restantes atividades económicas, automóveis, motociclos, restauração e passes de transportes, o cálculo das deduções está relacionado com a sede da empresa que presta o serviço. Efetivamente, neste caso o valor deduzido em cada transação varia de acordo com a localização da sede da empresa, já que se a empresa que emitiu a fatura estiver sediada num concelho do interior, a percentagem de dedução aumenta, de acordo com o benefício estipulado na classe Concelho.

2.5 EmpresasTop

Esta classe foi criada dentro do interface gráfico, visto que somente é necessária para apresentar a listagem das empresas com mais faturas do sistema e respetivos valores de despesas e de deduções.

2.6 JavaFactura

Esta classe é provavelmente a classe crucial para a realização do trabalho uma vez que uma grande maioria das funções utilizadas para a realização dos requisitos propostos se encontram explicitadas nesta classe. Todas essas implementações serão devidamente apresentadas na Secção 5.

Assim, para guardar os dois principais atores do sistema recorreremos ao uso de dois “maps”, um para armazenar os contribuintes individuais e outro para armazenar os contribuintes coletivos:

```
private Map<Integer, ContribuinteIndividual> individuais;  
private Map<Integer, ContribuinteColetivo> coletivos;
```

A chave para ambos os “maps” é o NIF, dos individuais para o caso dos individuais, e dos coletivos no caso dos coletivos.

3 Interfaces

A ponto de colmatar um dos últimos requisitos surgiu a necessidade de criar uma interface, explicada em seguida:

3.1 Incentivo Fiscal

Este interface possui apenas um método, *reducaoImposto*, comum às classes *FamiliaNumerosa* e *ContribuinteColetivo*, porquanto tenta relacionar essas duas classes, criando um novo tipo de dados que serão os objetos de instância dessa classe que têm um único comportamento, a redução do imposto. Por um lado, a família numerosa estabelece uma redução de imposto, prevendo uma bonificação de cinco por cento por cada dependente que integre o agregado. Por outro lado, o contribuinte coletivo implementa esse mesmo método, *reducaoImposto*, fazendo depender o cálculo da dedução da localização da sua sede. Assim, e conforme já explicamos anteriormente, este método tem um papel determinante no cálculo das deduções do contribuinte, bem como das do seu agregado familiar.

4 Classes de Exceções

A ponto de exceções que eventualmente poderão acontecer no programa desenvolvemos duas classes de *Exception* de modo a conseguirmos tratar de forma adequada estes “bugs” que poderão surgir e de separar código de tratamento de erros de código regular.

4.1 *CINullException*

A classe *CINullException* diz respeito a quando um contribuinte individual a que tentamos aceder não existe (é null).

4.2 *CCNullException*

A classe *CCNullException* diz respeito a quando um contribuinte coletivo a que tentamos aceder não existe (é null).

4.3 *FacturaNaoExisteException*

A classe *FacturaNaoExisteException* diz respeito a quando uma factura a que se tenta aceder não existe (é null).

4.4 *CSVformatIncorretException*

A classe *CSVformatIncorretException* é usada quando ao importar um ficheiro CSV a linha a ser analisada não contém informação de um objeto que pertença à aplicação.

5 Implementação e Manual de Utilizador

Para a concretização do trabalho prático, e com a extrema ajuda das classes já mencionadas, respondemos a todos os requisitos do trabalho prático.

Como já foi referido, recorreremos à biblioteca *javaafx* para representar a interface gráfica do programa e todos os métodos efetuados para tal encontram-se na classe *GUI*.

Nas imagens que iremos apresentar o programa já se encontra pré-populado com um conjunto de dados significativos, que permite testar toda a aplicação.

É de salientar, mais uma vez, que a classe *JavaFactura* é a que contém os “maps” com as informações dos contribuintes individuais e dos contribuintes coletivos, que por sua vez armazenam as faturas.

O ponto de partida do programa é a Figura 3, onde os contribuintes podem fazer login ou registarem-se, caso sejam individuais ou coletivos, carregando no botão que mais lhes convém.



Figura 3: Demonstração da página inicial do programa.

Em seguida, explicamos a forma como resolvemos cada uma das questões e a respetiva demonstração na aplicação:

5.1 Registar Contribuintes

- **Registar um contribuinte, quer seja individual ou empresa.**

De forma a responder ao ponto acima mencionado, depois do utilizador se deparar com a Figura 3, se a intenção do mesmo for **registar um novo contribuinte individual** aparecerá a Figura 4.

Figura 4: Registar um Contribuinte Individual.

Deste modo, o utilizador terá que preencher obrigatoriamente todos os campos e, dentro do nosso programa, é chamado o construtor parametrizado do contribuinte individual. É adicionado ao respetivo “map”, da classe `JavaFactura`, a informação deste contribuinte individual criado, como mostra a Figura 5.

```

ContribuinteIndividual ci = new ContribuinteIndividual(
    nifCI,
    emailCI,
    nomeCI,
    moradaCI,
    pwCI,
    0, //n_direcao
    0, //n_dependentes
    new ArrayList<>(), //lista nifs dependentes
    new ArrayList<>(), //lista nifs direcao
    1, //coeficiente_fiscal
    new HashMap<>() //grupo faturas
);
this.struct.add(ci);

```

Figura 5: Registrar um Contribuinte Individual (métodos).

Existem no nosso sistema de input output (classe GUI) “trys and catches” para perceber se o nif introduzido se trata de um número e se o contribuinte que está a ser adicionado já não pertence ao sistema (quer como contribuinte individual, quer como contribuinte coletivo).

Em caso de sucesso é enviada uma mensagem para o ecrã e em caso de insucesso também.

No caso da intenção do utilizador ser **registar um contribuinte coletivo** o raciocínio utilizado é exatamente o mesmo. Aparece no ecrã do utilizador a seguinte janela (Figura 6), e é igualmente obrigatório o utilizador preencher todos os campos, inclusivé o campo das atividades económicas, onde o contribuinte terá que escolher pelo menos um dos setores de atividade, e no máximo todos eles (são 11 no total).

Figura 6: Registrar um Contribuinte Coletivo.

No programa, é chamado o construtor parametrizado do contribuinte coletivo e as suas informações são adicionadas ao “map” da classe `JavaFactura`.

5.2 Login

- **Validar o acesso à aplicação utilizando as credenciais (nif e password), por parte de diferentes atores.**

Como já apresentado na Figura 3, se a intenção do utilizador for efetuar login na aplicação aparecerá a figura apresentada em seguida, onde o utilizador deverá preencher ambos os campos:

Início de Sessão

NIF:

Password:

Figura 7: Login.

O nosso sistema possui três tipos de atores: Contribuintes Individuais, Contribuintes Coletivos e o Administrador.

O Administrador possui um **NIF = 0** e uma **password = admin**.

Assim, no nosso sistema de input output é averiguado se foram introduzidas as credenciais do administrador, e, caso afirmativo, o utilizador é enviado para a respetiva dashboard (Figura 12).

Caso não tenha sido o administrador a iniciar sessão, é necessário averiguar se foi um contribuinte individual ou um contribuinte coletivo. Para isso, recorreremos às seguintes funções, sendo em primeiro lugar averiguado se foi um individual (Figura 8), e em segundo lugar se foi um coletivo (Figura 9). Se não existir nenhuma entidade com as credenciais inseridas é enviada uma mensagem de aviso ao utilizador.

```
/**
 * Método que verifica se existe um contribuinte individual no sistema com o NIF e password passados como parâmetro.
 *
 * @param NIF Identificador do Contribuinte Individual
 * @param password Password
 * @return true se existe um contribuinte individual com o NIF e password passados como parâmetro.
 * @throws CInullException
 */
public Boolean containsIndividual(int NIF, String password) throws CInullException {
    ContribuinteIndividual ci = this.individuais.get(NIF);
    boolean flag = false;

    if(ci == null)
        throw new CInullException("CI com nif " + NIF + " não existe");
    else
        flag = ci.getPassword().equals(password);

    return flag;
}
```

Figura 8: Login Contribuinte Individual.

```
/**
 * Método que verifica se existe um contribuinte coletivo no sistema com o NIF e password passados como parâmetro.
 *
 * @param NIF Identificador do Contribuinte Coletivo
 * @param password Password
 * @return true se existe um contribuinte coletivo com o NIF e password passados como parâmetro
 * @throws CCNullException
 */
public Boolean containsColetivo(int NIF, String password) throws CCNullException {
    ContribuinteColetivo cc = this.coletivos.get(NIF);
    boolean flag = false;

    if(cc == null)
        throw new CCNullException("CC com nif " + NIF + " não existe");
    else
        flag = cc.getPassword().equals(password);

    return flag;
}
```

Figura 9: Login Contribuinte Coletivo.

Caso tenha sido um Contribuinte Individual é apresentada a seguinte dashboard:

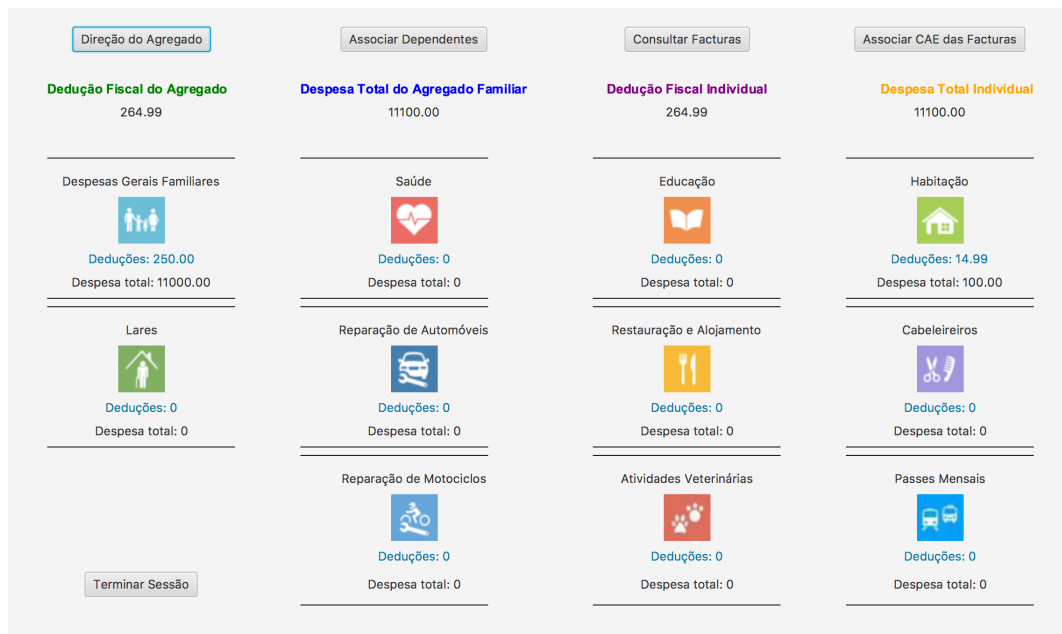


Figura 10: Dashboard Contribuinte Individual.

E caso tenha sido um Contribuinte Coletivo aparece a seguinte dashboard:



Figura 11: Dashboard do Contribuinte Coletivo.



Figura 12: Dashboard do Administrador.

É a partir destas dashboards que os atores do sistema poderão realizar as operações que mais lhes convêm, uma vez que é aqui que as funcionalidades do nosso programa estão incutidas.

5.3 Criar Faturas

- **Criar faturas associadas a despesas feitas por um contribuinte individual. São as empresas que alimentam esta informação no sistema.**

Os contribuintes coletivos podem criar faturas em nome de contribuintes individuais. Para tal, deverão seleccionar o respetivo botão de “Registar Facturas” na sua dashboard, onde são reencaminhados para o seguinte formulário:

Figura 13: Registar uma Fatura.

Deverão preencher todos os campos disponíveis. Existem outros campos que são automaticamente preenchidos tendo em conta alguns aspetos do contribuinte coletivo, como é o caso do nome da empresa emitente, o número da fatura, a natureza económica da fatura (que é já definida se o contribuinte coletivo só tiver inserido numa atividade económica, caso contrário, fica a 0 (Sem Atividade)) e o histórico dos caes da fatura (que caso a empresa esteja emergente em mais do um setor de atividade começa por ser 0, caso contrário, o histórico é e sempre será vazio).

Assim, é criada a fatura através do seu construtor parametrizado e respetivamente adicionada a cada um dos dois contribuintes relacionados com a mesma:

```
Factura f = new Factura(
    Integer.parseInt(numFacturaField.getText()),
    logged_user_nif,
    contribuinte_coletivo.getNome(),
    datePicker.getValue(),
    nif_cliente,
    descricaoField.getText(),
    contribuinte_coletivo.inicializarAtividadeEconomica(),
    new BigDecimal(valorField.getText().replaceAll(",", "")),
    contribuinte_coletivo.inicializarHistoricoCAES()
);

struct.addFactura(f);
```

Figura 14: Registrar uma Fatura (métodos).

O método `addFactura`, "chama" outros dois métodos, que adicionam as faturas a cada um dos objetos (Contribuinte Individual e Contribuinte Coletivo):

```
/**
 * Método que adiciona uma nova Factura ao sistema
 *
 * @param f Factura
 */
public void addFactura(Factura f) {
    individuais.get(f.getNif_cliente()).addFacturaRegistada(f);
    coletivos.get(f.getNif_emitente()).addFacturaEmitida(f);
}
```

Figura 15: Adicionar uma Fatura (método geral).

O método que adiciona a fatura ao Contribuinte Individual é a apresentada em seguida, e o raciocínio utilizado consiste em inicialmente determinar qual é o código do setor de atividade da fatura (0 se não tiver atividade definida) para posteriormente adiciona-la ao "map" das faturas do Contribuinte Individual, onde a chave é o código do setor.

```
/**
 * Adiciona nova factura emitida ao contribuinte individual.
 *
 * @param factura factura registada
 */
public void addFacturaRegistada(Factura factura) {
    Integer key = factura.getNatureza();

    ArrayList<Factura> setor = grupos_facturas.get(key);

    if (setor == null) {
        setor = new ArrayList<>();
        grupos_facturas.put(key, setor);
    }
}
```

Figura 16: Adicionar uma Fatura ao Contribuinte Individual.

E para adicionar a fatura ao Contribuinte Coletivo basta simplesmente adicionar a fatura à lista das faturas emitidas:

```
/**
 * Adiciona nova factura emitida pelo contribuinte coletivo.
 *
 * @param factura factura emitida
 */
public void addFacturaEmitida(Factura factura) {
    facturas.add(factura);
}
```

Figura 17: Adicionar uma Fatura ao Contribuinte Coletivo.

5.4 Verificar Despesas e Montantes Dedução Fiscal

- **Verificar, por parte do contribuinte individual, as despesas que foram emitidas em seu nome e verificar o montante de dedução fiscal acumulado, por si e pelo agregado familiar.**

Tendo em conta que uma das principais funções do programa é calcular as deduções a que os contribuintes têm direito, implementamos métodos que nos permitem não só calcular as deduções do contribuinte individual e despesas por si suportadas, mas também verificar o montante da despesa suportada pelo seu agregado, bem como verificar o valor total das deduções obtidas por todos os elementos do agregado familiar. De facto, conforme podemos observar na parte superior do painel informativo representado na Figura 10, estão contabilizadas as deduções fiscais e a despesa total dos elementos do seu agregado, bem como as deduções e as despesas efetuadas pelo contribuinte individual. Ademais, definimos ainda métodos que nos permitem também discriminar o montante das deduções e das despesas efetuadas por um indivíduo em cada um dos setores de atividade, o que é também possível verificar através do referido painel. Assim, sempre que é emitida uma fatura a um determinado contribuinte os valores inscritos no seu painel informativo são atualizados.

5.5 Associar CAE

- **Associar classificação de atividade económica a um documento de despesa.**

Outros dos objetivos do trabalho prático é associar uma classificação de atividade económica a um documento de despesa. Assim como no e-fatura, quando a empresa que emite uma fatura está inserida em mais do que um setor de atividade o CAE da fatura fica indefinido (no nosso caso fica com código 0, que corresponde a Sem Atividade).

Como forma de colmatar este problema, na sua dashboard, o contribuinte individual tem a opção **Associar CAE das Faturas**, como se pode ver na Figura 10, onde se encontram impressas todas as faturas do contribuinte individual que têm natureza 0 (corresponde ao ArrayList com chave 0 do “map”).

Deste modo, após selecionar essa opção o utilizador encontra uma tabela com todas as faturas sem CAE definido, onde o mesmo poderá selecionar o CAE que se adequa ao serviço que usufruiu. No nosso caso, optamos por só imprimir os CAEs disponíveis pelo contribuinte coletivo que emitiu a fatura, mas caso imprimissemos todos o programa apenas aceitaria o CAE se a empresa emitente o tivesse.

Número da Fatura	Data de Emissão	Valor	Natureza
6	2018-01-12	34.12	Desconhecido
			Reparação de Automóveis
			Reparação de Motociclos

Voltar

Figura 18: Associar CAE das Faturas (Tabela).

Após selecionar o CAE e carregar em **Submeter**, no nosso programa é chamado o método **setCAEFatura** (Figura 19), e passados como parâmetros o apontador da fatura, o NIF do contribuinte individual que está loggado e o número do código correspondente ao setor de atividade escolhido pelo contribuinte individual.

Em seguida, se o contribuinte coletivo possuir esse determinado CAE (o que no nosso caso é garantido devido à disposição da interface gráfica) é chamado o método **setCAE** (Figura 20), um método do Contribuinte Individual.

Este método tem a responsabilidade de eliminar a fatura do ArrayList de faturas com o código 0 do “map” e de a adicionar ao ArrayList com o código a que a fatura agora pertence. Para além disso, como é pedido no requisito da Secção 5.6, a fatura tem associada a si um histórico dos CAEs que já possuiu, e, assim sendo, este método adiciona a esse histórico o novo CAE da fatura (este método e em particular a funcionalidade do histórico vai ser novamente explicada na Secção 5.6).

```
/**
 * Atualiza o CAE de uma fatura, caso o contribuinte coletivo pertença a esse setor de atividade económica.
 *
 * @param fatura fatura a ser alterada
 * @param NIF identificador do contribuinte
 * @param newNatureza nova natureza da fatura
 * @throws CInullException, CCNullException
 */
public void setCAEFatura(Fatura fatura, int NIF, int newNatureza) throws CInullException, CCNullException {
    int nif_emitente = fatura.getNif_emitente();
    int id = fatura.getId();
    ContribuinteIndividual umContribuinteI = individuais.get(NIF);
    ContribuinteColetivo umContribuinteC = coletivos.get(nif_emitente);

    if(umContribuinteI == null)
        throw new CInullException("CI com nif " + NIF + " não existe");
    else if(umContribuinteC == null)
        throw new CCNullException("CC com nif " + nif_emitente + " não existe");
    else if(umContribuinteC.temAtividade(newNatureza)){
        umContribuinteI.setCAE(fatura, newNatureza);
    }
}
```

Figura 19: Associar CAE das Faturas- Método da classe JavaFatura.

```

**
* Atualiza o CAE de uma factura.
*
* @param factura factura a ser alterada
* @param newNatureza nova natureza da factura
*/
public void setCAE(Factura factura, int newNatureza) {
    int nif_emitente = factura.getNif_emitente();
    int id = factura.getId();

    ArrayList<Factura> facturasSemAtividade = this.getGrupos_facturas().get(0).stream().
        filter(f -> f.getId() != id ||
            f.getNif_emitente() != nif_emitente).
        collect(Collectors.toCollection(ArrayList::new));

    this.setFacturasSetor(facturasSemAtividade, 0);

    factura.setNatureza(newNatureza);

    //significa que se trata do 0 temporário
    if (factura.historico_caes.size() != 1){
        factura.removeSAHistoricoCae();
    }

    factura.addHistoricoCae(newNatureza);

    this.addFacturaRegistada(factura);
}

```

Figura 20: Associar CAE das Faturas- Método da classe ContribuinteIndividual.

5.6 Mudar CAE

- Corrigir a classificação de atividade económica de um documento de despesa. Esta alteração deve deixar registo para ser depois rastreada.

Na aba **Consultar Facturas** (visível na dashboard do Contribuinte Individual, Figura 10) o contribuinte individual tem a opção de visualizar todas as faturas emitidas com o seu NIF.

Nesse separador, ilustrado na Figura 21, o contribuinte tem a hipótese de alterar o CAE de uma fatura com um CAE já definido. É de referir que as faturas emitidas por uma empresa que exerce em apenas um setor económico não oferecem essa hipótese de mudança de CAE. Na nossa interface gráfica, para essas faturas com CAE "obrigatório" não é sequer impressa a opção de **Alterar CAE**, mas caso fosse, se o utilizador tentasse alterar o CAE o programa não deixaria.

Número da Fa...	Data de Emis...	Nif do Emite...	Natureza	Valor (€)	Alterar CAE?
3	2018-05-09	12	1	21.99	
4	2018-05-11	12	1	33.21	
1	2018-05-12	20	2	15.30	<input type="checkbox"/>
2	2018-05-24	20	7	9	<input type="checkbox"/>

Nome Ficheiro
 Id da Factura
 Imprime Factura

[Voltar](#)

Figura 21: Consulta Faturas (tabela).

Quando pressionado o botão da coluna **Alterar CAE** a natureza da fatura passa a 0 uma

vez que é invocado o método `eliminarCaeFatura` (classe `JavaFatura`, Figura 22), que por sua vez chama o método `eliminaCAE` (classe `Contribuinte Individual`, Figura 23).

Estes métodos são as responsáveis por averiguar se o contribuinte coletivo emissor da fatura está inserido em mais do que um setor (é chamado o método `podemAlterarCAEfaturas`, classe `Contribuinte Coletivo`) e por alterar efetivamente o CAE da fatura, respetivamente.

O método que altera efetivamente o CAE da fatura (Figura 23) é a responsável por eliminar a fatura do `ArrayList` correspondente ao código da natureza antiga do “map” e inseri-la no `ArrayList` correspondente ao código 0, com a fatura agora também com natureza 0.

```
/**
 * Atualiza o CAE de uma fatura para sem atividade, caso o contribuinte coletivo esteja inserido
 * em mais do que 1 setor de atividade económica.
 *
 * @param fatura fatura a ser alterada
 * @param NIF identificador do contribuinte
 * @throws CInullException, CNullException
 */
public void eliminarCaeFatura(Fatura fatura, int NIF) throws CInullException, CNullException {
    int nif_emitente = fatura.getNif_emitente();
    ContribuinteIndividual umContribuinteI = individuais.get(NIF);
    ContribuinteColetivo umContribuinteC = coletivos.get(nif_emitente);

    if(umContribuinteI == null)
        throw new CInullException("CI com nif " + NIF + " não existe");

    else if(umContribuinteC == null)
        throw new CNullException("CC com nif " + nif_emitente + " não existe");

    else if(umContribuinteC.podemAlterarCAEfaturas()){
        umContribuinteI.eliminaCAE(fatura);
    }
}
```

Figura 22: Eliminar o CAE da Fatura (método `JavaFatura`).

```
/**
 * Atualiza o CAE de uma fatura para sem atividade.
 *
 * @param fatura fatura a ser alterada
 */
public void eliminaCAE(Fatura fatura) {
    int id = fatura.getId();
    int nif_emitente = fatura.getNif_emitente();
    ArrayList<Integer> historico = fatura.getHistorico_caes();
    Integer key = fatura.getNatureza();

    ArrayList<Fatura> facturasExSetor = this.getGrupos_facturas().get(key).stream().
        filter(f -> f.getId() != id || f.getNif_emitente() != nif_emitente).
        collect(Collectors.toCollection(ArrayList::new));

    this.setFacturasSetor(facturasExSetor, key);

    fatura.setNatureza(0);

    //poderá ser um 0 temporário uma vez que a intenção do utilizador
    //poderá ser apenas alterar o cae da fatura e não removê-lo
    if(historico.get(historico.size() - 1) != 0)
        fatura.addHistoricoCae(SemAtividade.objeto.getCodigo());

    this.addFaturaRegistada(fatura);
}
```

Figura 23: Eliminar o CAE da Fatura (método `ContribuinteIndividual`).

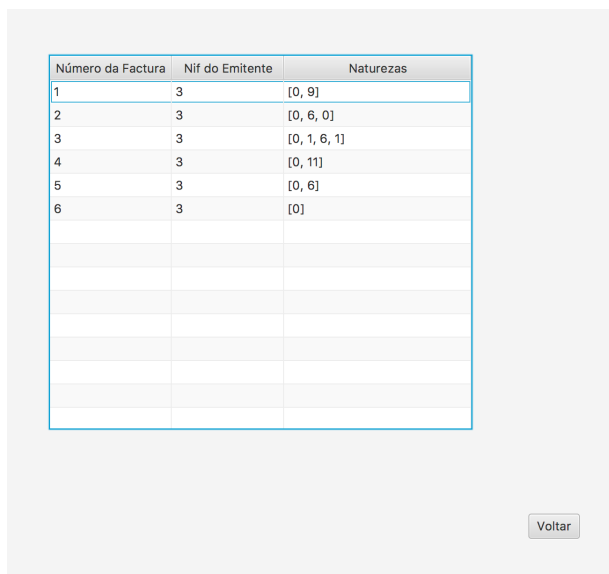
Para além disso, este método é também responsável por adicionar um 0 ao histórico dos CAEs, caso não haja já lá um 0 (quando a fatura é criada e não possui à partida CAE definido o seu histórico começa com um 0), ou seja, a última posição do `ArrayList` histórico será necessariamente um 0. Este 0 é importante para que a empresa possa saber que, de momento, aquela fatura não tem um CAE escolhido pelo contribuinte individual.

Assim, uma vez que a fatura pertence agora ao `ArrayList` de faturas sem CAE definido, quando o utilizador pressionar novamente o botão de **Associar CAE das Faturas**, presente na sua dashboard individual, essa fatura estará impressa nessa tabela e o utilizador poderá escolher de novo qual o CAE que melhor se adequa àquela fatura.

Deste modo, depois de eliminar o CAE atual da fatura, é adicionado ao `ArrayList` histórico o código 0. Se a intenção do utilizador for mesmo alterar o CAE da fatura para um outro de imediato esse 0 foi apenas um código temporário, pelo que o método `setCAE`, (Figura 20,

mentionado na Secção 5.5), tem em atenção esse 0 e elimina-o do histórico. O único 0 do histórico que se manterá é o 0 inicial do ArrayList, que representa que a empresa que emitiu a fatura atua em mais do que um setor de atividade económica, o que dá para concluir que o CAE da fatura é o CAE entendido pelo utilizador.

O contribuinte coletivo tem a oportunidade de consultar esse histórico clicando no botão **Consultar Histórico CAES**, presente na sua dashboard (Figura 11). Nesse separador é apresentada a tabela presente na Figura 24, que basicamente imprime para cada fatura emitida pela empresa a sua variável de instância `historico_caes`.



Número da Fatura	Nif do Emitente	Naturezas
1	3	[0, 9]
2	3	[0, 6, 0]
3	3	[0, 1, 6, 1]
4	3	[0, 11]
5	3	[0, 6]
6	3	[0]

Voltar

Figura 24: Histórico dos CAEs das Faturas (Contribuinte Coletivo).

5.7 Listagem das Faturas Empresas (Data e Valor)

- Obter a listagem das faturas de uma determinada empresa, ordenada por data de emissão ou por valor.

O contribuinte coletivo tem a opção de obter a listagem das suas faturas ordenadas por data de emissão. Para tal, na sua dashboard (Figura 11), após pressionar o botão **Consultar Facturas**, aparecerá a dashboard representada na Figura 25, onde estão visíveis todas as faturas emitidas por si. Para as ordenar por data, o contribuinte terá que carregar no botão **Ordenar todas por Data**.

Número da Fa...	Data de Emis...	Nif do Consum...	Natureza	Valor (€)
1	2018-05-12	1	2	15.30
2	2018-05-24	1	7	9
5	2018-05-21	1	0	12
6	2018-05-04	2	0	321.11
7	2018-05-04	2	0	132.99

Ordenar todas por Valor

Ordenar todas por Data

Data inicial

Data final

NIF

Filtrar por data

NIF

Filtrar por valor

Limpar

Nome Ficheiro

Id da Factura

Imprime Factura

Voltar

Figura 25: Aba Consultar Faturas do Contribuinte Coletivo.

Esse botão faz com que o método `ordenaFaturaData`, apresentado na Figura 26, seja acionado. Este método devolve a lista das faturas da empresa com o NIF passado como parâmetro ordenadas por data. Esta lista é então impressa na tabela, podendo deste modo o contribuinte coletivo visualizar as suas faturas segundo esta ordem, como podemos ver na Figura 27, que é o estado do programa depois do botão ser pressionado.

```

/**
 * Ordena as faturas de um contribuinte coletivo por data das mesmas.
 *
 * @param NIF NIF de um contribuinte coletivo
 * @return uma Lista de facturas ordenadas por data
 */
public List<Factura> ordenaFaturasData(int NIF) {
    if(!coletivos.containsKey(NIF)) {
        return new ArrayList<Factura>();
    }

    return coletivos.get(NIF).getFacturas().stream().
        sorted(Comparator.comparing(Factura :: getData)).
        collect(Collectors.toList());
}

```

Figura 26: Método que ordenada as faturas de uma dada empresa por data.

Número da Fa...	Data de Emis...	Nif do Consum...	Natureza	Valor (€)
8	2018-04-18	1	0	12.49
6	2018-05-04	2	0	321.11
7	2018-05-04	2	0	132.99
1	2018-05-12	1	2	15.30
5	2018-05-21	1	0	12
2	2018-05-24	1	7	9

Ordenar todas por Valor

Ordenar todas por Data

Data inicial

Data final

NIF

Filtrar por data

NIF

Filtrar por valor

Limpar

Nome Ficheiro

Id da Factura

Imprime Factura

Voltar

Figura 27: Consultar Faturas por Data (Resultado Tabela).

Por outro lado, poderá também carregar no botão **Ordenar todas por Valor**, que fará que com que as faturas fiquem ordenadas por valor, segundo o método `ordenaFaturaValor`, ilustrado na Figura 28. O resultado após pressionado o botão é o apresentado na Figura 29.

```
/**
 * Ordena as faturas de um contribuinte coletivo por valor das mesmas.
 *
 * @param umCC um contribuinte coletivo
 * @return uma Lista de facturas ordenadas por valor
 */
public List<Factura> ordenaFaturasValor(int NIF) {
    if(!coletivos.containsKey(NIF)) {
        return new ArrayList<Factura>();
    }

    return coletivos.get(NIF).getFacturas().stream().
        sorted(Comparator.comparing(Factura :: getValor)).
        collect(Collectors.toList());
}
```

Figura 28: Método que ordenada as faturas de uma dada empresa por valor.

Número da Fa...	Data de Emis...	Nif do Consum...	Natureza	Valor (€)
2	2018-05-24	1	7	9
5	2018-05-21	1	0	12
1	2018-05-12	1	2	15.30
7	2018-05-04	2	0	132.99
6	2018-05-04	2	0	321.11

Ordenar todas por Valor

Ordenar todas por Data

Data inicial

Data final

NIF

Filtrar por data

NIF

Filtrar por valor

Limpar

Nome Ficheiro

Id da Factura

Imprime Factura

Voltar

Figura 29: Consultar Faturas por Valor (Resultado Tabela).

5.8 Listagem das Faturas Empresas por Contribuinte (e Data)

- Obter por parte das empresas, as listagens das faturas por contribuinte num determinado intervalo de datas.

Para usufruir desta funcionalidade, o contribuinte coletivo apenas tem que preencher os campos Data Inicial, Data Final e NIF (do contribuinte individual) na sua dashboard de **Consultar Facturas** (Figura 25) e pressionar o botão **Filtrar por Data**.

É accionado o método da Figura 30, e o procedimento é exatamente igual ao mencionado na secção anterior (Secção 5.7), irá aparecer na tabela as faturas do determinado contribuinte individual realizadas no intervalo de tempo escolhido.

```
/**
 * Devolve a lista das facturas emitidas a um contribuinte num determinado periodo.
 *
 * @param NIFcc identificador do contribuinte coletivo
 * @param NIFci identificador do contribuinte individual
 * @param init data de inicio
 * @param end data de fim
 * @return Lista da facturas emitidas a determinado contribuinte num determinado periodo
 */
public List<Factura> facturasPeriodoContribuinte(int NIFcc, int NIFci, LocalDate init, LocalDate end) {
    if(!coletivos.containsKey(NIFcc)) {
        return new ArrayList<Factura>();
    }

    return coletivos.get(NIFcc).getFacturas().stream().
        filter(f-> f.getNif_cliente() == NIFci || NIFci == 0).
        filter(f -> f.getData().isAfter(init) && f.getData().isBefore(end)).
        collect(Collectors.toList());
}
```

Figura 30: Método que retorna as faturas de uma dada empresa passadas para um determinado contribuinte, no intervalo de tempo mencionados.

5.9 Listagem das Faturas Empresas por Contribuinte (e Valor)

- Obter por parte das empresas, as listagens das faturas por contribuinte ordenadas por valor decrescente de despesa.

Para esta funcionalidade o raciocínio é igual ao anterior mencionado. O contribuinte coletivo preenche o campo NIF do contribuinte individual (na segunda linha que pede o NIF), na dashboard **Consultar Facturas** (Figura 25), e pressionar **Filtrar por Valor**.

As faturas do contribuinte individual escolhido aparecerão na tabela, ordenadas por valor decrescente de despesa (o método que é chamado é o método apresentado na Figura 31).

```
/**
 * Devolve a lista das facturas emitidas a um contribuinte ordenadas por valor decrescente de despesa.
 *
 *
 * @param NIFcc identificador do contribuinte coletivo
 * @param NIFci identificador do contribuinte individual
 * @return Lista da facturas emitidas a determinado contribuinte ordenadas por valor decrescente
 */
public List<Factura> facturasValorContribuinte(int NIFcc, int NIFci) {
    if(!coletivos.containsKey(NIFcc)) {
        return new ArrayList<Factura>();
    }

    return coletivos.get(NIFcc).getFacturas().stream().
        filter(f-> f.getNif_cliente() == NIFci || NIFci == 0).
        sorted(Comparator.comparing(Factura :: getValor).reversed()).
        collect(Collectors.toList());
}
```

Figura 31: Método que retorna as faturas de uma dada empresa passadas para um determinado contribuinte, ordenadas por valor decrescente de despesa.

5.10 Total faturado por uma empresa num determinado período

- Indicar o total faturado por uma empresa num determinado período.

O contribuinte coletivo tem ainda a oportunidade de consultar o valor total faturado num determinado período. Para isso, basta clicar, através da sua dashboard (Figura 11), no botão **Consultar Valores de Facturas**. Abrirá o seguinte separador:

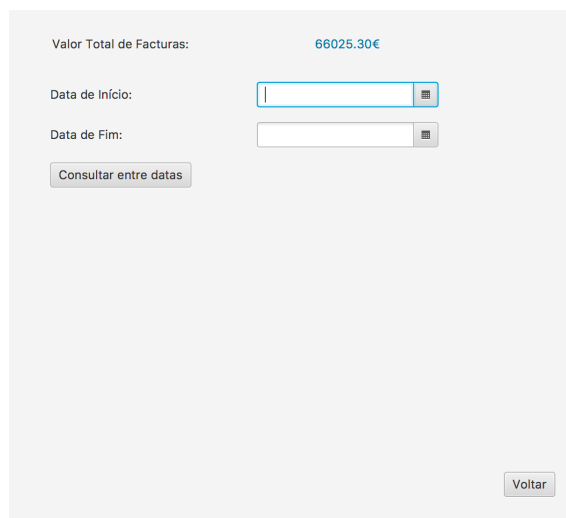


Figura 32: Consultar Valores de Facturas Ecrã.

E, aqui, o contribuinte coletivo pode seleccionar a Data de Início e a Data de Fim e **Consultar entre datas** o valor faturado. Depois de escolhidas as datas e seleccionado o botão, irá aparecer numa Label o valor correspondente (tal como apresentado na Figura 33).

Valor Total de Facturas: 66025.30€

Data de Início:

Data de Fim:

Valor entre o período selecionado: 15.30€

Figura 33: Consultar Valores de Facturas Exemplo de Intervalo Ecrã.

Os métodos chamados para tal são os métodos apresentados na Figura 34 e Figura 35, respetivamente.

```
/**
 * Devolve o valor facturado entre duas datas por um contribuinte coletivo do sistema.
 *
 * @param NIF identificador do contribuinte coletivo
 * @param init data de início
 * @param end data de fim
 * @return total facturado
 */
public BigDecimal totalFacturado(int NIF, LocalDate init, LocalDate end) {
    return coletivos.get(NIF).totalFacturado(init, end);
}
```

Figura 34: Método da classe JavaFactura que calcula o total fatura por uma empresa num determinado período.

```
/**
 * Devolver valor total facturado entre duas datas.
 *
 * @param init data inicial
 * @param end data final
 * @return valor total facturado entre as duas datas de argumento
 */
public BigDecimal totalFacturado(LocalDate init, LocalDate end) {
    BigDecimal total = BigDecimal.ZERO;
    int size = facturas.size();

    for (int i = 0; i < size; i++) {
        Factura f = facturas.get(i);
        if (f.getData().isAfter(init) && f.getData().isBefore(end))
            total = total.add(f.getValor());
    }

    return total;
}
```

Figura 35: Método da classe ContribuinteColetivo que efetivamente calcula o valor pedido.

5.11 Coleção dos 10 Contribuintes que mais gastam

- **Determinar a relação dos 10 contribuintes que mais gastam em todo o sistema (esta operação deve ser só disponibilizada para o administrador da aplicação).**

O administrador do sistema pode consultar a coleção dos 10 contribuintes que mais gastam. Para tal, carregam no botão **10 contribuintes que mais gastam**, presente na sua dashboard de administrador (Figura 12).

Será apresentado o separador na imagem seguinte, onde estão impressos os NIFs dos 10 contribuintes que mais gastaram e respetivo valor de gasto.

NIF: 11	NIF: 10	NIF: 9
11100€	10034.12€	9000€
NIF: 8	NIF: 7	NIF: 6
8000€	7000€	6000€
NIF: 5	NIF: 4	NIF: 3
5000€	4000€	3000€
NIF: 2		
2015.30€		

Voltar

Figura 36: Aba 10 contribuintes que mais gastam.

Os métodos que são utilizados para tal são os métodos presentes na seguinte figura:

```
/**
 * Adiciona ao ArrayList users um dos dez contribuintes que mais gastaram
 *
 * @param users ArrayList com os 10 contribuintes que mais gastaram
 * @param v um contribuinte
 */
private void addTopContribuinte(ContribuinteIndividual v, ArrayList<ContribuinteIndividual> users) {
    for (int i = 0; i < 10; i++)
        if (v.getValorTotal().compareTo(users.get(i).getValorTotal()) > 0 ) {
            users.add(i, v);
            return;
        }
}

/**
 * Atualiza o ArrayList argumento com os 10 contribuintes que mais gastaram
 *
 * @param users ArrayList com os 10 contribuintes que mais gastaram
 */
public void getTop10Contribuintes(ArrayList<ContribuinteIndividual> users) {
    for (int i = 0; i < 10; i++)
        users.add(new ContribuinteIndividual());
    individuais.forEach((k,v) -> addTopContribuinte(v,users));
}
```

Figura 37: 10 contribuintes que mais gastam (Métodos).

5.12 Coleção X Empresas

- Determinar a relação das X empresas que têm mais faturas em todo o sistema e o montante de deduções fiscais que as despesas registadas (dessas empresas) representam (esta operação deve ser só disponibilizada para o administrador da aplicação).

Para determinar a coleção das X empresas que têm mais faturas e respetivo o montante de deduções fiscais foi utilizado o método da Figura 38.

```

/**
 * Devolve as X as empresas que mais emitiram facturas.
 *
 * @return int qtasEmpresas- numero de empresas
 */
public List<ContribuinteColetivo> EmpresasComMaisFacturas(int qtasEmpresas) {
    return coletivos.values().stream().
        sorted(Comparator.comparing(ContribuinteColetivo:: numFacturas).reversed()).
        limit(qtasEmpresas).collect(Collectors.toList());
}

```

Figura 38: Coleção X Empresas Métodos.

Para consultar esta funcionalidade o administrador da aplicação deverá seleccionar o botão **Empresas com mais facturas emitidas**, presente na sua dashboard (Figura 12).

Aparecerá o seguinte separador:

Figura 39: Coleção X Empresas Ecrã.

Preenchendo o campo **Numero de Empresas**, e carregando em **Submeter**, aparecerá na tabela as informações pretendidas.

Para facilitar o preenchimento da tabela foi criada uma classe auxiliar chamada **EmpresasTop** dentro da classe **GUI**.

5.13 Gravar Estado da Aplicação em Ficheiro

- Gravar o estado da aplicação em ficheiro, para que seja possível retomar mais tarde a execução.

Para gravarmos e retomarmos o estado da nossa aplicação usamos a API **java.io** e implementamos os métodos **guardaEstado** e **carregaEstado**, sendo que o primeiro guarda o estado da aplicação em binário e o segundo recupera esse mesmo estado. Estes métodos utilizam as classes **FileOutputStream** e **FileInputStream** os quais permitem criar uma stream de escrita e de leitura, respetivamente.

Para tanto, todas as classes cujas instâncias devem escrever e ler os seus atributos em formato binário para e de um ficheiro implementam a interface `Serializable`. Este processo, conhecido como serialização, é inicializado pela escrita do `JavaFactura` através do método `writeObject` da classe `ObjectOutputStream` e todas as referências a outros objetos a partir do `JavaFactura` são recursivamente escritas.

Para realizar a “desserialização” dos bytes escritos para o ficheiro, ou a leitura desses bytes utilizamos o método `readObject` da classe `ObjectInputStream`.

Em simultâneo, usando `FileOutputStream` e `FileInputStream`, é também guardado em ficheiro, em modo de texto, o valor da variável de classe de `Factura`, `nextID`, que representa o Id único da próxima factura a ser emitida.

5.14 Famílias Numerosas e Empresas do Interior

- Gestão da definição do número de filhos para qualificação de uma família como sendo numerosa

Uma família numerosa é aquela que tem mais do que 4 dependentes. Para que o nosso programa considere uma família numerosa o agregado familiar terá de ser constituído por, pelo menos, um contribuinte que assuma a sua direcção e cindo ou mais dependentes. O programa faz essa gestão através do método `addNovoDependenteAgregado`.

```
*/
public void addNovoDependenteAgregado(int nif_pai, int nif_dependente) throws CInullException {
    ContribuinteIndividual ci_pai = individuais.get(nif_pai);
    ContribuinteIndividual ci_dependente = individuais.get(nif_dependente);

    if (ci_pai == null)
        throw new CInullException("CI com nif " + nif_pai + " não existe");
    else if (ci_dependente == null)
        throw new CInullException("CI com nif " + nif_dependente + " não existe");
    else {
        //se o contribuinte dependente é elegível para ser um dependente do agregado
        if (ci_dependente.possuSerDependente()) {
            //se o contribuinte pai é elegível para ser associado um dependente (faz parte da direcção)
            if (ci_pai.pertenceDirecaoAgregado(nif_pai)) {
                for (int nifs_dependentes: ci_pai.getNifsDependentesAgregado()) {
                    ContribuinteIndividual contr = individuais.get(nifs_dependentes);
                    contr.addDependente(nif_dependente);
                }
                for (int nifs_direcao: ci_pai.getNifsDirecaoAgregado()) {
                    ContribuinteIndividual contr = individuais.get(nifs_direcao);
                    contr.addDependente(nif_dependente);
                }
                ci_dependente.atualizaInfoNovoMembro(ci_pai);
            }
        }
    }
}
```

```

//verifica se o contribuinte pertence a uma familia numerosa
if(ci_pai.getN_dependentes() >= FamiliaNumerosa.getFamiliaNumerosa()){
    //se o pai faz parte de uma familia numerosa e ainda não foi instanciado como tal
    if(!(ci_pai instanceof FamiliaNumerosa)) {
        for(int nifsAgregado : ci_pai.getNifsDependentesAgregado()) {
            ContribuinteIndividual umContribuinte = individuais.get(nifsAgregado);
            FamiliaNumerosa umCiFamiliaNumerosa = new FamiliaNumerosa(umContribuinte);
            individuais.put(nifsAgregado, umCiFamiliaNumerosa);
        }

        for(int nifsDirecao : ci_pai.getNifsDirecaoAgregado()) {
            ContribuinteIndividual umContribuinte = individuais.get(nifsDirecao);
            FamiliaNumerosa umCiFamiliaNumerosa = new FamiliaNumerosa(umContribuinte);
            individuais.put(nifsDirecao, umCiFamiliaNumerosa);
        }
    }
    //se o pai já foi instanciado como FamiliaNumerosa mas um dos seus dependentes não,
    //por exemplo, por ter sido gerado depois da familia numerosa estar constituída
    else if (!(ci_dependente instanceof FamiliaNumerosa)){
        FamiliaNumerosa umCiFamiliaNumerosa = new FamiliaNumerosa(ci_dependente);
        individuais.put(nif_dependente, umCiFamiliaNumerosa);
    }
}
}

```

Figura 40: Método *addNovoDependenteAgregado*.

Sempre que um dependente é adicionado a um agregado, verifica-se o número total de dependentes que o integram e no caso de atingir os 5, todos os contribuintes individuais que compõem esse agregado passam a ser instâncias de *FamiliaNumerosa*.

A constituição do agregado é explicada detalhadamente na Secção 5.15.

- **Tratamento privilegiado da Empresa do Interior.**

Conforme referimos anteriormente, em determinadas situações o cálculo das deduções tem em consideração a localização da sede da empresa. Por isso, aquando do registo do contribuinte coletivo, Figura 6, é necessário introduzir o distrito da sede da empresa, visto que se a empresa estiver sediada num concelho do interior do país e se efetuar uma transação no setor automóvel, restauração, motociclos ou transportes o adquirente do serviço ou do bem beneficiará de uma dedução fiscal superior. Essa funcionalidade é implementada pelo código infra, o qual cria os 18 concelhos, associa-lhes o valor percentual que irá acrescer à percentagem de dedução estipulada nos setores de atividade acima mencionados e insere-os na variável de classe atividades.

5.15 Criação de Agregados

No nosso programa implementamos também uma forma simples e realista de criar agregados familiares.

Assim, existem 2 entidades dentro de um agregado: os diretores do agregado e os dependentes do agregado (como já foi mencionado na Secção 2.1.1).

Como tal, toda a gente deve inserir-se (caso de diretor) ou ser inserida (caso de dependente) em um desses setores.

Para se inserir como diretor do agregado o contribuinte tem que, dentro da sua dashboard (Figura 10), aceder ao separador **Direção do Agregado**. Dentro deste separador, demonstrado na Figura 41, o utilizador tem a hipótese de **Fazer parte da direção deste agregado**. Se o fizer, o mesmo passa imediatamente a ser o diretor do seu agregado.

Figura 41: Ecrã do Agregado.

Um diretor de um agregado pode adicionar dependentes acedendo, a partir da sua dashboard (Figura 10), ao separador **Associar Dependentes**. Dentro desta aba (apresentada na Figura em seguida) o contribuinte tem que digitar o NIF do dependente e carregar em Registrar.

Figura 42: Ecrã Associar Dependente.

No entanto algumas regras têm que ser cumpridas em ambos os casos, passo a enumerar:

- **Ser diretor de um agregado:**
 1. Não pode ser dependente de nenhum agregado;
 2. Direção do agregado ainda não pode estar cheia (número de diretores é no máximo 2).
- **Ser dependente de um agregado:**
 1. Não fazer parte de nenhum agregado, quer como dependente, quer como diretor;
 2. O contribuinte que o está a tentar adicionar tem que ser diretor de um agregado.

Assim, quando algum contribuinte individual é adicionado por um diretor a um agregado as informações do agregado (valores das deduções, despesas, NIFs dos outros membros do agregado e respetivo número (de dependentes e de diretores)) são atualizadas em ambos os contribuintes.

Os contribuintes individuais têm ainda a oportunidade de se associarem a outro agregado, isto é, por exemplo, se existem dois agregados sendo um deles constituído por um pai e dois filhos e o outro constituído por uma mãe e um filho tanto o pai como a mãe dos diferentes agregados podem optar por se juntarem e com isto juntarem também os seus dependentes, passando a serem um só agregado com dois pais e três filhos. As informações do novo agregado são respetivamente atualizadas em todos os membros.

Esta opção está disponível quando o utilizador acede ao botão **Fazer parte da direção de outro agregado**, presente na aba **Direção do Agregado** (Figura 41). Esta será a dashboard apresentada:

Inserir NIF do contribuinte da outra direção:

Submeter Voltar

Figura 43: Ecrã intenção de se associar a outro agregado.

É de salientar que terá que ser um dos dois diretores do agregado a adicionar o outro diretor, e que ambos os agregados só podem ter um membro diretor para deste modo o agregado se juntar e ficarem os dois como diretores (como já foi mencionado o número máximo de diretores é 2) .

Esta implementação facilitou o cálculo das deduções e despesas e a implementação do conceito de Família Numerosa.

5.16 Download e Impressão de Facturas

Tanto um contribuinte individual como um contribuinte coletivo tem disponível uma ferramenta, a partir do ecrã de consulta de faturas disponível a cada um, para fazer download de um fatura, em formato HTML, que poderá depois ser impressa através de uma navegador de Internet.

[illegible]

Figura 44: Download de Fatura por parte do Contribuinte Individual.

Número da Fa...	Data de Emis...	Nif do Consum...	Natureza	Valor (€)
1	2018-05-12	1	2	15.30
2	2018-05-24	1	7	9
5	2018-05-21	1	0	12
6	2018-05-04	2	0	321.11
7	2018-05-04	2	0	132.99

Ordenar todas por Valor

Ordenar todas por Data

Data inicial

Data final

NIF

Filtrar por data

NIF

Filtrar por valor

Limpar

Nome Ficheiro

Id da Factura

Imprime Factura

Voltar

Figura 45: Download de Factura por parte do Contribuinte Coletivo.

Esta é uma funcionalidade relativamente simples, existe na aplicação um layout de HTML/CSS na qual são introduzidos os dados da factura e das entidades intervenientes e, de seguida, o resultado é, através do recurso à API `FileOutputStream`, impresso para um ficheiro cujo nome pode ou não ser definido pelo utilizador, em codificação UTF-8.

F A C T U R A

Adelbert Anetts
NIF:176673568

Abata
NIF:895012920
2 Kinsman Parkway
ldangel0@devhub.com

Bem 6

Factura	6
Data	2018-03-20
Total	€32.99

T E R M O S

Processado por JavaFactura 1.0
Não indicado para fins comerciais.

Figura 46: Factura HTML.

5.17 Exportar e Importar informação em formato CSV

Está disponível ao administrador da aplicação uma ferramenta de importação e exportação de dados da aplicação. Este poderá fazer um *dump* de todo o conteúdo através de "Exportar CSV" para o ficheiro à sua escolha. Esta funcionalidade percorre toda a estrutura de dados da aplicação e imprime todos os objetos que ela contém para o ficheiro.

[illegible]

Figura 47: Ficheiro CSV.

Poderá também, a qualquer momento carregar dados totais de um aplicação ou pequena parcelas de dados, como por exemplo, apenas faturas de uma determinada empresa, para a aplicação. O ficheiro importado é percorrido e para cada objeto introduzido é feito o *parse* do seus dados e depois é introduzido na aplicação com recurso aos mesmo métodos usados no registo direto na aplicação.

Empresas com mais facturas emitidas

Gravar Estado

Nome Ficheiro

Exportar para CSV

10 contribuintes que mais gastam

Recuperar Estado

Nome Ficheiro

Importar CSV

Voltar

Figura 48: Exportar/Importar CSV.

6 Conclusões

Face ao problema apresentado e analisando criticamente a solução proposta concluímos que cumprimos as tarefas, conseguindo atingir os objetivos definidos. De facto, conseguimos implementar a aplicação que foi pedida. Em particular pensamos as nossas estruturas de dados de forma a serem facilmente adaptáveis a situações futuras, nomeadamente quando criamos a classe abstrata `AtividadesEconomicas`, fizemo-lo com o intuito de facilitar a inserção de atividades económicas que conferissem direito a dedução fiscal. Com efeito, para adicionar uma nova atividade, basta criar uma nova subclasse, extensão da classe `AtividadesEconomicas`, que contenha os dados necessários (código, designação, percentagem de dedução e limite) para a sua implementação. Acresce que o mesmo se passa com o cálculo das deduções, visto que este está associado ao setor de atividade

económica. Com efeito, apesar da superclasse definir o método `calculaDeducacao`, que é herdado pelas subclasses que não o definem, a maioria das subclasses de atividades económicas implementa o seu próprio método `calculaDeducacao`, fazendo-o da forma que considera mais adequada tendo em conta diversos fatores, tais como a localização da empresa que presta o serviço ou o número de elementos do agregado familiar.

Do mesmo modo, pensando em alterações futuras, implementamos a classe `FamiliaNumerosa` com uma variável de classe que define o limite mínimo de dependentes, porquanto desta forma podemos facilmente alterar a quantidade de dependentes que constituem uma família numerosa para efeitos fiscais. É ainda necessário aumentar a robustez da importação de CSV, que neste momento tem muitas limitações, para se tornar mais útil a todos os utilizadores, de modo a, que no futuro pudesse ser possível aos contribuintes fazer uso desta ferramenta para maior conforto na introdução de dados.

Em suma, não obstante as potenciais melhorias que poderiam ser feitas no programa, nomeadamente a introdução de diferentes taxas de IVA; a discriminação dos bens ou serviços prestados, sobre os quais poderia incidir diferentes taxas de IVA; a possibilidade do contribuinte individual ser capaz de registar uma fatura emitida por uma empresa; a possibilidade de associar receitas médicas a faturas de saúde com taxas de IVA superiores a 6% e, finalmente, a implementação do sorteio da *Fatura da Sorte*, as quais não implementamos apenas por falta de tempo, consideramos que o nosso projeto cumpre todos os requisitos pedidos.