Universidade do Minho
Escola de Engenharia

**Mestrado Integrado em Engenharia Informática**
**Unidade Curricular:** Programação Orientada a Objetos
**Grupo:** 32
**Docente:** António Manuel Nestor Ribeiro

# Relatório do projeto prático (JavaFatura)

**A79987 -** Ana Rita de Sousa e Castro Guimarães
**Mail:** a79987@alunos.uminho.pt
**Áreas de interesse:** tecnologia, jogos

**A82145 -** Filipa Correia Parente
**Mail:** a82145@alunos.uminho.pt
**Áreas de interesse:** tecnologia, desporto e música

# 1 Índice

# 2 Introdução

No âmbito da unidade curricular de Programação Orientada a Objetos, foi proposto a criação de uma plataforma de validação de faturas(JavaFatura), utilizando a linguagem **Java** para o efeito.

Para a realização deste projeto teve de se ter em conta os seguintes princípios:

- Existem 2 tipos de contribuintes. O individual, que corresponde a um contribuinte individual, e o coletivo, onde se inserem as empresas;

- Cada tipo de contribuinte tinha permissões diferentes (ex: um contribuinte individual pode verificar as faturas mas não pode emitir, enquanto que um contribuinte coletivo pode verificar as faturas que emitiu bem como emitir novas);

- Para além dos contribuintes(individuais e coletivos) a aplicação disponibiliza um acesso priviligiado ao sistema (apenas acessivel aos administradores) onde podem verificar os contribuintes (individuais) que mais contribuiram para IRS bem como as empresas que emitiram mais faturas, entre outras informações;

Neste relatório serão abordadas as estratégias usadas para a implementação desta plataforma.

# 3 Estruturas de dados utilizadas

Para a elaboração da plataforma, utilizaram-se as seguintes estruturas:

- **HashMap**
  Esta estrutura foi utilizada para guardar todos os contribuintes. Como key, foi utilizado o nif de cada um, pois cada contribuinte tem um nif diferente. Também foi utilizado o mesmo para a elaboração dos métodos do administrador;

- **ArrayList**
  Esta estrutura foi utilizada muitas vezes no projeto:

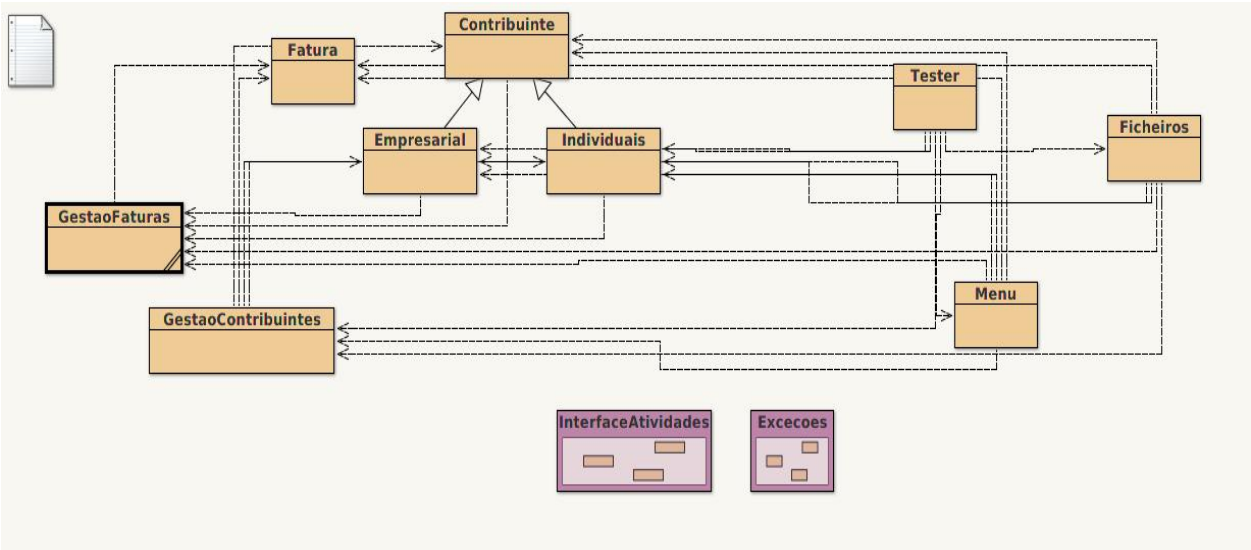
  - Para guardar todas as faturas de cada contribuinte;

  - Para guardar as atividades de dedução de cada empresa, bem como a atividade(s) para o qual o cliente deduziu, e os códigos de cada atividade. Caso fosse mais que uma, cabia ao contribuinte escolher uma dessas atividades escrevendo o código da atividade correspondente. Caso não houvesse nenhuma atividade não se deduzia nada para acrescentar ao montante fiscal;

  - Para armazenar a lista de contribuintes que possuem um maior montante de dedução fiscal.

# 4 Arquitetura de classes utilizada

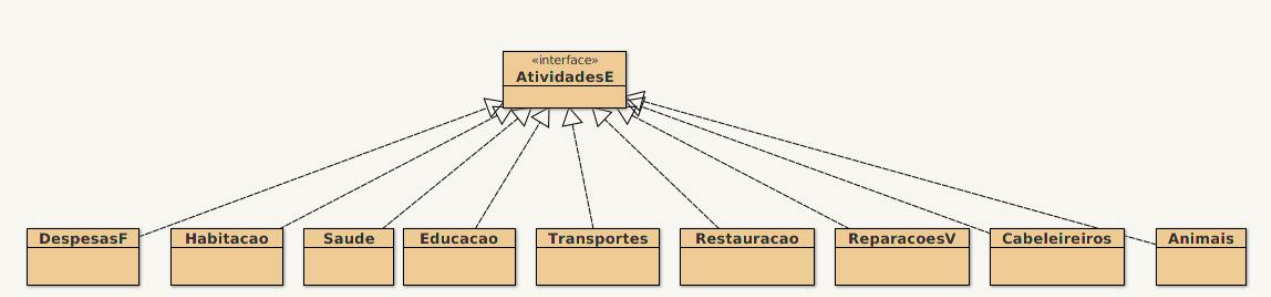
Utilizou-se a arquitetura presente na seguinte imagem:




Eis uma pequena descrição de cada classe/interface:

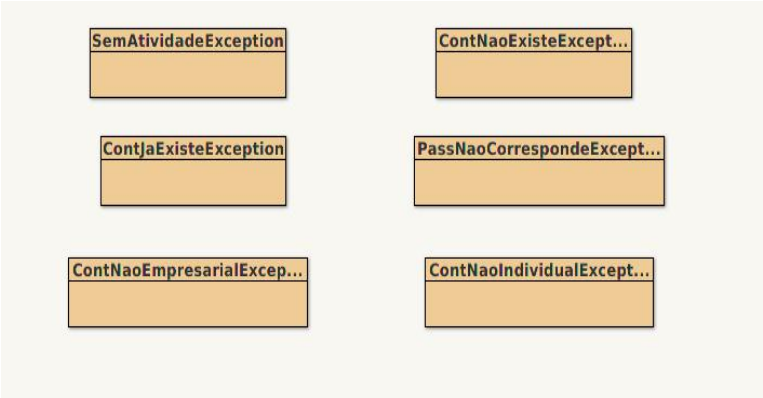
- **GestaoContribuintes.java:** classe onde se gerem todos os contribuintes;
  - **Contribuinte.java:** Informação genérica as empresas e aos contribuintes individuais;
  - **Empresarial.java:** Contém a informaçao especifica para o cliente coletivo;
  - **Individuais.java:** Contém a informaçao especifica para o cliente individual;
- **GestaoFaturas.java:** classe onde se gerem todas as faturas de um determinado contribuinte;
  - **Fatura.java:** Contém a informação para ser inserida nas faturas;
- **Tester.java:** Classe onde se encontra o menu (a ser executado no dia da apresentação);
  - **Menu.java**: Classe onde se encontram alguns métodos auxiliares ao Tester.java;
  - **Ficheiros.java:** Classe onde se encontra um exemplo de um pequeno sistema (a ser executado no dia da apresentação).

Dentro da package **InterfaceAtividades**:




- **AtividadesE.java:** interface onde estão os métodos comuns a todas as atividades económicas. Como classes relativas às atividades económicas temos:
  - **Saude.java:** setor da saúde;
  - **Educacao.java:** setor da educação;
  - **Habitacao.java:** setor imóvel;
  - **ReparacoesV.java:** setor da reparação de veículos;
  - **DespesasF.java:** setor das despesas familiares;
  - **Transportes.java:** setor dos transportes(os passes contam para efeitos de dedução fiscal);
  - **Animais.java:** setor veterinário;
  - **Cabeleireiros.java:** setor onde se localizam os salões de beleza.

Dentro da package **Excecoes** (onde se encontram as classes de exceção):




- **SemAtividadeException.java:** Para as Empresas que não deduzem, bem como para as faturas que não apresentam nenhuma atividade;
- **ContNaoExisteException.java:** Utilizado no caso de remoção de um determinado contribuinte e ele não existir;
- **ContJaExisteException.java:** Utilizado no caso de inserção de um determinado contribuinte e ele já existir;
- **ContNaoEmpresarialException.java:** Utilizado no caso de o contribuinte não ser do tipo Empresarial;
- **ContNaoIndividualException.java:** Utilizado no caso de o contribuinte não ser do tipo Individual;
- **PassNaoCorrespondeException.java:** Utilizado no caso de no processo de login a palavra passe digitada não corresponder à do contribuinte em questão.

# 5 Decisões tomadas para a resolução do problema

Tendo em conta a estrutura de dados utilizada, para a resolução dos requisitos básicos foi apenas necessário percorrer a estrutura que guardava esses dados:

- **Registar um contribuinte, quer seja individual ou empresa:** Para a resolução deste problema primeiro, foi preciso fazer o scan desse novo contribuinte. Caso ele fosse do tipo empresarial, adicionava-se à informação genérica a lista das atividades na qual essa empresa deduzia. Caso fosse do tipo individual adicionava-se à informação genérica, as informações relativamente ao agregado familiar, inserindo as informações num objeto da classe correspondente. Esse objeto era guardado no Hashmap. No caso de o nif que o novo contribuinte digitou existisse na estrutura, libertava uma exceção avisando, o utilizador que esse contribuinte já existe no sistema.
- **Validar o acesso a aplicação utilizando as credenciais (nif e password), por parte de diferentes atores:** Para a resolução deste problema apenas foi necessário fazer o scan do nif e da password correspondente. Caso o nif não existisse no sistema libertava uma exceção, avisando o utilizador que esse contribuinte não existe no sistema. Na hipótese de a password estar errada libertava uma exceção a avisar que a password não corresponde á do contribuinte em questão;
- **Criar faturas associadas a despesas feitas por um contribuinte individual:** Esta ação apenas é possível para os contribuintes coletivos. Então, para emitir uma fatura, é feita à empresa um scan das informações relativas à fatura que deseja emitir. Posteriormente é/são adicionada(s) a a(s) atividade(s) para o qual a empresa deduz, e as informações inseridas num objeto da classe Fatura. Finalmente, é atualizada a sua lista de faturas e também a lista de faturas do contribuinte ao qual vai ser emitida a mesma.
- **Verificar por parte do contribuinte individual as faturas emitidas, bem como o montante de dedução fiscal acumulado, por si e pelo agregado familiar:** Num primeiro momento, após o contribuinte ter feito o login, é preciso imprimir para o ecrã as faturas emitidas para o seu nif. Para isso bastou aceder à sua lista de faturas (presente na classe Contribuinte e invocar o método toString da classe GestaoFaturas. Para imprimir o montante de dedução fiscal acumulado pelo próprio, apenas se precisou de imprimir o valor inerente. Para imprimir o montante fiscal acumulado pelo agregado familiar apenas foi necessário aceder ao objeto com o nif de cada elemento do agregado e fazer o somatório do montante fiscal de cada um.

- **Associar classificação da atividade económica a um documento de despesa:** É o contribuinte coletivo que associa a(s) sua(s) atividades de dedução à fatura (explicado no ponto 3). No entanto, como exixtem empresas que deduzem para mais do que uma atividade, pelo que é tarefa do contribuinte individual corrigir a fatura com a atividade na qual ele realmente deduziu (este método será explicado no próximo ponto).

- **Corrigir a classificação de uma atividade económica de um documento de despesa:** Como foi explicado no parágrafo anterior, é tarefa do contribuinte individual corrigir a fatura com a atividade na qual ele realmente deduziu. Para isso, é feito um rastreamento das faturas pendentes (isto é, cuja lista de atividades é maior que 1). Este escolhe a fatura na qual deseja definir a atividade, escolhe o código da atividade na qual deseja deduzir, e se o código pertencer a alguma atividade, atualiza o montante de dedução fiscal. Se não pertencer, não é atualizado valor do montante.

- **Obter a listagem das faturas por contribuinte, ordenada por data e por valor decrescente de despesa:** Para este requisito foi necessário apenas aceder às faturas da empresa e filtrar aquelas que correspodiam ao nif escolhido pelo utilizador. Depois foi só ordenar as faturas, ja filtradas por intervalos de datas, por datas e por valor.

- **Indicar o total faturado por uma empresa num determinado intervalo de tempo:** Para responder a este requisito foi só percorrer a lista de faturas da empresa, fazendo o somatório dos valores de despesa de cada fatura.

- **Determinar a relação dos 10 contribuintes que mais gastam em todo o sistema:** Neste ponto primeiro, foi necessário filtrar no HashMap dos contribuintes apenas os do tipo individual, e inseri-los num HashMap intermédio, cuja chave era o próprio Contribuinte e o valor era o somatório do valor de despesa da lista de faturas de cada um. Depois foi só ordenar os contribuintes por valor decrescente de despesa e inseri-los num ArrayList.

- **Determinar a relação das X empresas que mais faturas emitem em todo o sistema:** Para este requisito foi necessário filtrar no HashMap dos contribuintes apenas os do tipo empresarial, e inseri-los num HashMap intermédio, cuja chave era o próprio Contribuinte e o valor era o somatório do número de faturas de cada um. Depois foi só ordenar os contribuintes por valor decrescente do número de faturas e inseri-los num ArrayList. Repetiu-se o mesmo processo

- **Determinar a relação das X empresas com maior montante fiscal deduzido:** Repetiu-se o mesmo processo do ponto anterior. Apenas se mudou o valor do HashMap intermédio para o montante total deduzido.

- **Gravar o estado da aplicação em ficheiro:** Para gravar a plataforma em ficheiro foi preciso criar 3 métodos, um que guardasse o estado atual bem como o HashMap de

contribuintes em ficheiro binário, outro que guardasse em ficheiro .txt o HashMap, e outro que carregasse o conteúdo do ficheiro binário. À medida que se iam acrescentando contribuintes e faturas, o estado do ficheiros ia sendo atualizado.

Para a **segunda parte** do projeto, os métodos para a resolução do problema mantiveram-se os mesmos. De um modo mais conciso, apenas se adicionou uma variável de instância adicional nos contribuintes coletivos(“*private boolean local*”), que distinguiam aqueles que se localizam no interior dos do litoral, e nos contribuintes individuais, que distinguiam aqueles que pertencem a uma família numerosa daqueles que fazem parte de uma familia normal (“*private boolean categoria*”). Conforme se tratasse de uma empresa do interior, ou de uma familia numerosa, o coeficiente fiscal de cada fatura para IRS subia 5%.

# 6 Instruções para executar a aplicação

**1 -** Extrair o projeto Grupo32_POO2018.zip;
**2 -** Abrir o projeto chamado “trabalho” no BlueJ dentro da pasta Grupo32_POO2018;
**3 -** Compilar o ficheiro **Ficheiros.java**;
**4 -** Executar a main do **Ficheiros.java**;
**5 -** Compilar o ficheiro **Tester.java**;
**6 -** Executar a main do **Tester.java**;
**7 -** Interagir com a aplicação;

# 7 Conclusão

Através da elaboração deste projeto prático foi possível o aprofundamento do conhecimento da linguagem Java no contexto da programação orientada a objetos, bem como de conceitos como o de encapsulamento e modularidade do código, e a sua aplicação na vida real.

Como pontos positivos, concluiu-se que o paradigma de programação orientada a objetos é ótimo para reutilização e manutenção do código, através dos mecanismos de herança e polimorfismo.

Como pontos a melhorar destaca-se, a possibilidade de se ter posto mais opções no menu, nomeadamente a possibilidade de um contribuinte coletivo deduzir para o fisco, e uma melhor distinção entre empresas (em vez de se distinguir entre uma empresa do litoral ou do interior distinguir-se por concelhos, por exemplo).