**Universidade do Minho**
Escola de Engenharia
Mestrado Integrado em Engenharia Informática

# JavaFactura

Programação Orientada aos Objetos

2º Semestre
2017-2018

Grupo 62


A78997 Bruno José Infante De Sousa


A79034 - Rafaela Maria Soares da Silva


A77045 - Ricardo Barros Pereira

27 de maio de 2018
Braga

# Conteúdo

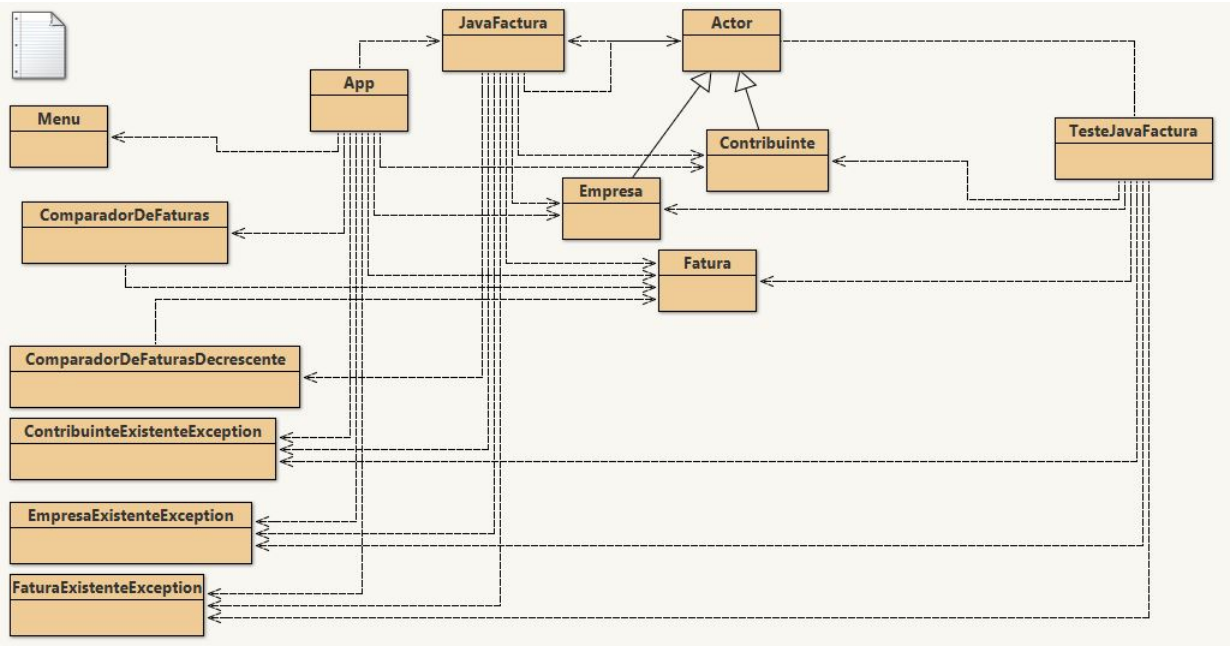
# 1 Introdução

No âmbito da Unidade Curricular de Programação Orientada aos Objetos, foi proposto ao grupo de trabalho a realização de um projeto, JavaFactura, implementado em linguagem Java.

Este consiste na elaboração de uma "plataforma" capaz de disponibilizar informação útil acerca das faturas de contribuintes individuais e de empresas, sendo a associação feita através do número de identificação fiscal. Estas entidades devem ter acesso às faturas que lhe estão associadas e ao montante de deduções fiscais.

Assim, os requisitos principais que este projeto desafia a realizar são a possibilidade de os contribuintes individuais poderem aceder às faturas associadas ao seu número fiscal, associar faturas (ainda por classificar) ao sector de atividade económica respetivo (como saúde, educação, transportes, entre outros não menos relevantes) e visualizar detalhadamente o valor acumulado de deduções fiscais. Para além disso, também desafia permitir que as empresas associem faturas de um serviço a um determinado cliente.

# 2 Classes




Todas as classes têm os métodos de instância habituais.

## 2.1 Classe JavaFactura

Nesta classe, temos um Ator que representa o utilizador que está a interagir com a plataforma, temos três *HashMap's*, um com os contribuintes registados no sistema, outro com as empresas e, por fim, um com as faturas. Temos ainda uma variável do tipo *int* que nos indica quantas faturas temos registadas e que vai ser usada para atribuir o *ID* às faturas que são registadas.

Além dos métodos para registar Contribuintes, Empresas ou Faturas no sistema, temos ainda um método **iniciaSessao( *int nif*, *String pass*, *int* o)**, porventura o método mais importante desta classe, pois é ele que dá acesso às restantes funcionalidades, que, dependendo do valor do parâmetro **o** , verifica se o Contribuinte(**o=1**) ou a Empresa(**o=2**) estão registados no sistema, e em caso positivo, inicia sessão, permitindo ao ator em questão, explorar as funcionalidades da aplicação.

Nesta classe temos ainda os restantes métodos para devolver listagens de faturas, totais faturados pelas empresas, entre outros dos quais, os métodos que permitem guardar/ler o estado da aplicação para/de um ficheiro.

## 2.2 Classe Actor

Na classe Actor, escolhemos como variáveis de instância a informação que os contribuintes individuais e as empresas têm em comum – *nif* (número de identificação fiscal), *email*, nome, morada e *pass(password)*, de forma a simplificar a elaboração da classe Contribuinte (que se refere ao contribuinte individual) e da classe Empresa.

Assim, a classe Actor é a *"superclass"* e as classes Contribuinte e Empresa são as *"subclasses"* da classe Actor.

### 2.2.1 Classe Empresa

Na classe Empresa, as variáveis de instância escolhidas foram o fator que a empresa tem no cálculo de dedução fiscal e as atividades em que ela atua.

### 2.2.2 Classe Contribuinte

Na classe Contribuinte, as variáveis de instância escolhidas foram o número de agregado familiar, os *nif's* dos membros do agregado familiar, o coeficiente fiscal para efeitos de dedução e os códigos das atividades economicas para as quais um determinado contribuinte tem possibilidade de deduzir despesas.

## 2.3 Classe Fatura

Na classe Fatura, guardamos a informação de cada fatura, como o *nif* e a designação da empresa que emite a fatura, o *nif* do cliente, a data de emissão da despesa, a atividade a que se refere a mesma, o valor, a descrição e o histórico(que permite verificar se a atividade económica foi alterada). Cada fatura tem ainda associado um *ID*. Este identificador é sempre inicializado a 0 no construtor paramentrizado e posteriormente é alterado com base no número de faturas já registadas no *HashMap* da classe JavaFatura.

## 2.4 Classe Menu

Nesta classe temos os métodos para criação de menus e para a sua execução. É aqui que manipulamos os mesmos, mostramos ao utilizador as suas opções e lemos a escolha do utilizador.

## 2.5 Classe App

Esta é a classe principal do nosso projeto. É aqui que invocamos os métodos da classe JavaFatura para registar Contribuintes, Empresas e Faturas. É também nesta classe que criamos os menus e é a partir daqui que ocorre a interação entre os dados da JavaFatura e os dados inseridos pelos utilizadores da plataforma.

## 2.6 Classe de exceções

No intuito de um eficiente tratamento de erros, decidiu-se criar um conjunto de classes de exceções.

- **ContribuinteExistenteException** - Exceção que indica que o contribuinte já está registado.
- **EmpresaExistenteException** - Exceção que indica que a empresa já está registada.
- **FaturaExistenteException** - Exceção que indica que a fatura não existe.

## 2.7 Classe de *Comparators*

Para comparar faturas da forma pretendida, decidiu-se criar um conjunto de classes de *Comparators*.

- **ComparadorDeFaturas** - *Comparator* que compara duas faturas considerando o valor da despesa..
- **ComparadorDeFaturasDecrescente** - *Comparator* que compara duas faturas considerando o valor da despesa com cariz decrescente.

## 2.8 Classe Teste

Esta classe foi usada ao longo do desenvolvimento do projeto para irmos testando as nossas soluções e melhorando as nossas opções relativamente a alguns pontos do código. No final, usamos também esta classe para desenvolver uma espécie de base de dados com os dados que vamos utilizar durante a apresentação do projeto para mostrar a aplicação a funcionar. Para tudo isto, desenvolvemos apenas o método *main* onde criamos uma entidade do tipo *JavaFactura* e à qual adicionamos vários contribuintes, empresas e faturas, sendo que no final invocamos o método *guardarEstado("Teste.obj")* para guardar tudo o que foi criado num ficheiro que será a tal base de dados da nossa apresentação.

# 3 Requisitos básicos

## 3.1 Registar um contribuinte

O método que permite registar um contribuinte, quer seja individual ou empresarial é o **registo** que se situa na classe App. Este pede ao utilizador que escolha entre a opção 1(registar-se como contribuinte individual) e a opção 2(registar-se como empresa).

De seguida, são pedidas as informações necessárias para o registo de cada entidade.

Caso escolha a opção 1, é pedido o *nif* , o *email*, o nome, a morada, a *password*, o número de membros do agregado familiar, os *nif's* dos membros do agregado familiar e o coeficiente fiscal para efeitos de dedução. Verifica-se se o *nif* do contribuinte individual em questão existe. Se existir, não pode registar-se.

Caso escolha a opção 2, é pedido o *nif* , o *email*, o nome, a morada, a *password*, o fator para o cálculo de dedução fiscal, o número de atividades económicas e as atividades económicas que a empresa presta serviços. Verifica-se se o *nif* da empresa em questão existe. Se existir, não pode registar-se.

## 3.2 Login

O método que permite que o contribuinte individual ou a empresa faça o login é o método **login** que se situa na classe App.

Este método recebe um inteiro que permite identificar qual é a entidade que quer fazer o login(1, caso seja um contribuinte individual e 2, se for uma empresa).

São pedidas as informações necessárias – o *nif* e a *password*. De seguida, recorre-se ao método iniciaSessao(situado na classe JavaFatura) que verifica se o nif existe. Se existir, verifica se a *password* inserida corresponde à *password* da entidade em questão.

## 3.3 Criar faturas

O método que permite criar uma fatura é o método **criarFatura** que se situa na classe App.

São pedidas as informações necessárias para criar uma fatura – o *nif* do cliente, a data, o valor da despesa e a descrição da despesa.

Caso a empresa em questão que quer emitir a fatura preste serviços apenas em uma atividade económica, a fatura é automaticamente catalogada. Caso contrário, o cliente tem de a catalogar.

## 3.4 Verificar despesas e o montante de dedução fiscal

O método que permite ao contribuinte individual verificar as despesas que foram emitidas em seu nome e verificar o montante de dedução fiscal acumulado, por si e pelo agregado familiar é o método **montanteContribuinteAgregado** que se situa na classe App.

Para isso, recorre-se a duas listas em que uma contém as faturas do contribuinte em questão e outra que contém as faturas do seu agregado.

De seguida, é calculado o montante de dedução fiscal através do método montanteDeducaoPorListagemFatura que se situa na classe JavaFactura.

Decidiu-se calcular o montante de dedução fiscal multiplicando o valor da despesa pelo valor de dedução associado à atividade da despesa e pelo valor do coeficiente fiscal do contribuinte.

## 3.5 Associar classificação de atividade económica

O método que permite ao contribuinte individual associar uma atividade económica é o método **associarAtividade** que se situa na classe App.

O contribuinte só pode associar a atividade, caso a fatura não tenha sido catalogada.

É pedido ao utilizador que insira o *ID* da fatura. Verifica-se se a fatura com esse *ID* existe, se essa fatura pertence ao cliente e se a empresa que a emitiu tem mais de uma atividade económica.

Caso a empresa tenha mais que uma atividade económica, verifica-se se a fatura já foi catalogada. Se não foi, é permitido ao utilizador associar.

Caso a empresa tenha apenas uma atividade económica, já foi catalogada automaticamente e, portanto, o utilizador não tem a possibilidade de associar.

Se a fatura não existir, é dito ao utilizador que não existe.

## 3.6 Corrigir a classificação de atividade económica

O método que permite ao contribuinte individual corrigir uma atividade económica é o método **corrigirAtividade** que se situa na classe App.

É pedido ao utilizador que insira o *ID* da fatura. Verifica-se se a fatura com esse *ID* existe, se essa fatura pertence ao cliente, se a empresa que a emitiu tem mais de uma atividade económica e se a fatura não tem de ser catalogada.

Se se verificar estas condições, é disponibilizado as atividades que a empresa presta serviços e pede-se ao utilizador que escolha a que diz respeito ao serviço que foi prestado. Assim, a atividade económica é corrigida e é guardado no histórico uma *string* que diz a atividade económica anterior e a atual.

Se a fatura não existir, é dito ao utilizador que não existe.

Caso a fatura tenha de ser catalogada, é dito ao utilizador.

Caso a fatura exista e não tenha de ser catalogada nem a possa catalogar, é porque a fatura foi catalogada automaticamente.

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

O método que permite uma empresa obter a sua listagem de faturas ordenada por valor crescente é o método **listagemFaturasDaEmpresa** situado na classe App. Recorre-se a uma lista que contém as faturas da empresa, ordenada pelo valor(com cariz ascendente) através do **ComparadorDeFaturas**.

## 3.8 Obter por parte das empresas, as listagens das faturas por contribuinte num determinado intervalo de datas

O método que permite que uma empresa obtenha as listagens das faturas por contribuinte num determinado intervalo de datas é o método **listagemFaturasPorContribuinte** situado na classe App.

É pedido ao utilizador que insira a data do início e do fim do perído que quer verificar.

Recorre-se a um *map* que contém os *nifs* dos contribuintes associados às listas com as faturas respetivas que estão associadas à empresa num determinado intervalo de datas através do método **listFaturasPorContribuinte** que se situa na classe JavaFactura.

## 3.9 Obter por parte das empresas, as listagens das faturas por contribuinte ordenadas por valor decrescente de despesa

O método que permite que uma empresa obtenha as listagens das faturas por contribuinte num determinado intervalo de datas é o método **listagemFaturasPorContribuinteOrdenado** situado na classe App.

Este método é muito semelhante ao **listagemFaturasPorContribuinte**. Difere apenas no facto de que não considera um intervalo específico de tempo e no facto de que a listagem é ordenada por valor decrescente de despesa(através doComparadorDeFaturasDecrescente).

Recorre ao método **listFaturasPorContribuinteOrdenado** situado na classe JavaFactura.

## 3.10 Indicar o total faturado por uma empresa num determinado período

O método que permite que uma empresa verifique o total faturado num determinado período é o método **totalFaturadoEmpresa** situado na classe App.

É pedido a data do início e do fim do período que quer ser verificado. A soma é feita através do método **totalEmpresa** na classe JavaFactura que acumula o valor das despesas que estão associadas à empresa.

### 3.11 Determinar a relação dos dez contribuintes que mais gastam em todo o sistema

O método que permite que o administrador verifique os dez contribuintes que mais gastam em todo o sistema é o método **dezQueMaisGastam** situado na classe JavaFactura.

Recorre-se à lista pretendida através de uma *stream* de contribuintes individuais ordenada pelo valor gasto, delimitada a dez contribuintes:

**contribuintes.values().stream().sorted(new ContribuinteQuantoGasta()).limit(10).**
**collect(Collectors.toList())**.

### 3.12 Determinar a relação das X empresas que mais faturas têm em todo o sistema e o montante de deduções fiscais que as despesas registadas (dessas empresas) representam

O método que permite que o administrador verifique a a relação das X empresas que mais faturas têm em todo o sistema e o montante de deduções fiscais que as despesas registadas (dessas empresas) representam é o método **relacaoXempresas** situado na classe App.

É pedido o número de empresas que se quer verificar.

Recorre-se ao método xEmpresas situado na classe JavaFactura. Este obtém uma lista através de uma *stream* de empresas individuais ordenada pelo valor de faturas, demilitada a X empresas:

**empresas.values().stream().sorted(new EmpresaMaisFaturas()).limit(x).**
**collect(Collectors.toList())**.

### 3.13 Gravar o estado da aplicação em ficheiro

Para guardar o estado da aplicação em ficheiro, desenvolvemos dois métodos:
**guardarFicheiroTxt(nomeFicheiro)** e **guardarEstado(nomeFicheiro)**, sendo que um é para guardar em ficheiro *.txt* e outro para gravar em ficheiro de objetos(*.obj*), respetivamente.

Estes métodos são invocados na classe App sempre que fechamos a aplicação, para assim podermos retomar o estado atual quando iniciarmos a aplicação no futuro. Para isso, desenvolvemos ainda o método **iniciaApp(nomeFicheiro)** para podermos ler um ficheiro de objetos e retomar o estado anterior quando iniciarmos a aplicação. No código que enviamos, iniciamos a aplicação com um ficheiro gerado pela classe de teste(*Teste.obj*, para podermos ter os dados na aplicação aquando da apresentação do projeto e temos ainda uma linha de código comentada, onde em vez do ficheiro de teste, invocamos o método *iniciaApp* com o ficheiro *Estado.obj*. É isto que permite que depois de testarmos com o ficheiro de teste, se invertermos o código e comentarmos a linha de teste em vez da linha do ficheiro de estado, possamos continuar a aplicação no mesmo ponto, mesmo depois de terminar sessão e encerrar a aplicação.

```
app.javaFactura = app.javaFactura.iniciaApp("Teste.obj");
//app.javaFactura = app.javaFactura.iniciaApp("Estado.obj");
```

# 4 Interface do utilizador

Na Interface desta aplicação, disponibolizamos ao utilizador num menu inicial as opções de se registar e/ou de iniciar sessão se já tiver o ser registo efetuado. Criamos também uma conta para o admnistrador que é acessivel a partir do menu Inicial.

Para os contribuintes individuais, disponiblizamos a informação sobre as despesas emitidas em seu nome, assim como montante fiscal acomulado por si e também pelo seu agregado familiar. O contribuinte pode também, corrigir ou associar uma atividade económica a uma fatura que nao esteja catalogada.

Para as empresas, disponobilizamos as opções de criação de uma fatura, de visualização das listagens das mesmas, tanto todas as que foram emitidas a qualquer contribuinte, como apenas a um determinado contribuinte. Tem também acesso ao valor total faturado num determinado período de tempo.

Ao administrador disponibilizamos a lista dos dez contribuintes que mais gastam, e também a relação de um determinado número de empresas que tem mais faturas no sistema e o montante de deduções fiscais que as despesas registadas representam.

Por último, para além das funcionalidades que disponibilizamos individualmente atrás referidas, o nosso interface permite o retrocesso a qualquer utilizador dos menus ou opções disponibilizadas, o término das suas sessões e também da aplicação.

# 5 Discussão

Através de uma pesquisa bem realizada acerca das deduções fiscais, dos limites de dedução e de conhecimentos na área da Economia, seria possível obter algoritmos semelhantes ao e-Fatura.

Como, infelizmente, não nos focamos em valores semelhantes, criamos "o nosso próprio algoritmo de cálculo de deduções fiscais".

Para o montante de dedução fiscal do contribuinte:

montante = valor da despesa * valor da dedução associada à despesa * coefiente do contribuinte individual em questão.

Para o montante de dedução fiscal da empresa:

montante = valor da despesa * valor da dedução associada à despesa * fator de dedução fiscal associada à empresa.

Caso a atividade da fatura seja:

- Educacao: É associado 0.30.
- Saude: É associado 0.15.
- Habitacao: É associado 0.15.
- Geral: É associado 0.35.
- Outros: Não é associado qualquer valor de dedução fiscal.

```java
public double montanteDeducaoPorFatura (Fatura f)
    double montante = 0;
     switch(f.getAtividade()) {
        case "Educacao" :
            montante = f.getValor() * 0.30;
            break;

        case "Saude" :
            montante = f.getValor() * 0.15;
            break;

        case "Geral" :
            montante = f.getValor() * 0.35;
            break;

        case "Habitacao" :
            montante = f.getValor() * 0.15;
    }
    return montante;
}
```

(Esté é o montante base. Em cada método específico que calcula o montante do contribuinte individual e da empresa é adicionado o coeficiente de dedução fiscal e o fator de dedução fiscal, respetivamente.)

# 6 Conclusão

Apesar de, infelizmente, termos realizado apenas o primeiro patamar proposto pelo projeto, consideramos que tenha sido consideravelmente satisfatório.

Tivemos alguns conflitos de organização entre a classe App e a classe JavaFactura, visto que surgia alguns problemas de *"static context"*.

Esperamos ter a possibilidade de concluir o projeto num futuro próximo.