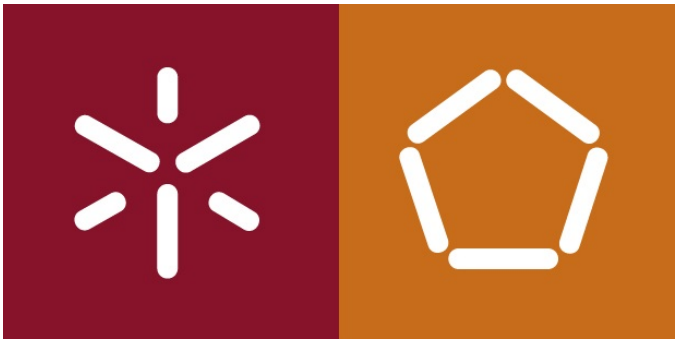
Universidade do Minho

Mestrado Integrado em Engenharia Informática



JavaFactura

Programação Orientada aos Objetos

Grupo 12

Adriana Meireles A82582 Eduardo Jorge A83344 Filipe Monteiro A80229

Figure 1

May 27, 2018

Contents

Introdução

No âmbito da unidade curricular de Programação Orientada aos Objetos, foi pedido o desenvolvimento de uma aplicação em Java. Este trabalho prático incidiu na criação de uma solução baseada no já existente "e-Fatura", onde um utilizador - Contribuinte/Individual - pode consultar as suas faturas e alterar a situação destas, montantes de deduções ou caso seja uma empresa, gerá-las. Além disso existe uma entidade que gere a plataforma, o administrador. Sendo um tema tão fortemente ligado à indústria de software é um exemplo perfeito de como linguagens orientadas aos objetos são trabalhadas e moldam problemas bastante pragmáticos e do dia a dia.

Enunciado

O projeto consiste na criação de uma plataforma com a informação referente às Faturas sobre os contribuintes. É pedida a implementação de várias entidades, faturas e setores/despesas económicas.

Entidades

A aplicação funciona alterando o estado de dois tipos de entidades: Contribuintes individuais e Empresas. Para gerir a aplicação existem administradores.

Faturas

As faturas representam despesas no sistema. Tipos de despesas podem ser feitas pelos contribuintes sob a forma de Faturas de vários setores económicos.

Interface

Para poder fazer-se uso do programa, é pedido também a implementação de uma interface que permita acesso às funcionalidades implementadas. Dado o contexto da cadeira, foi implementada uma interface de texto *à la DOS* tentado seguir a aquitetura *MVC*.

Classes

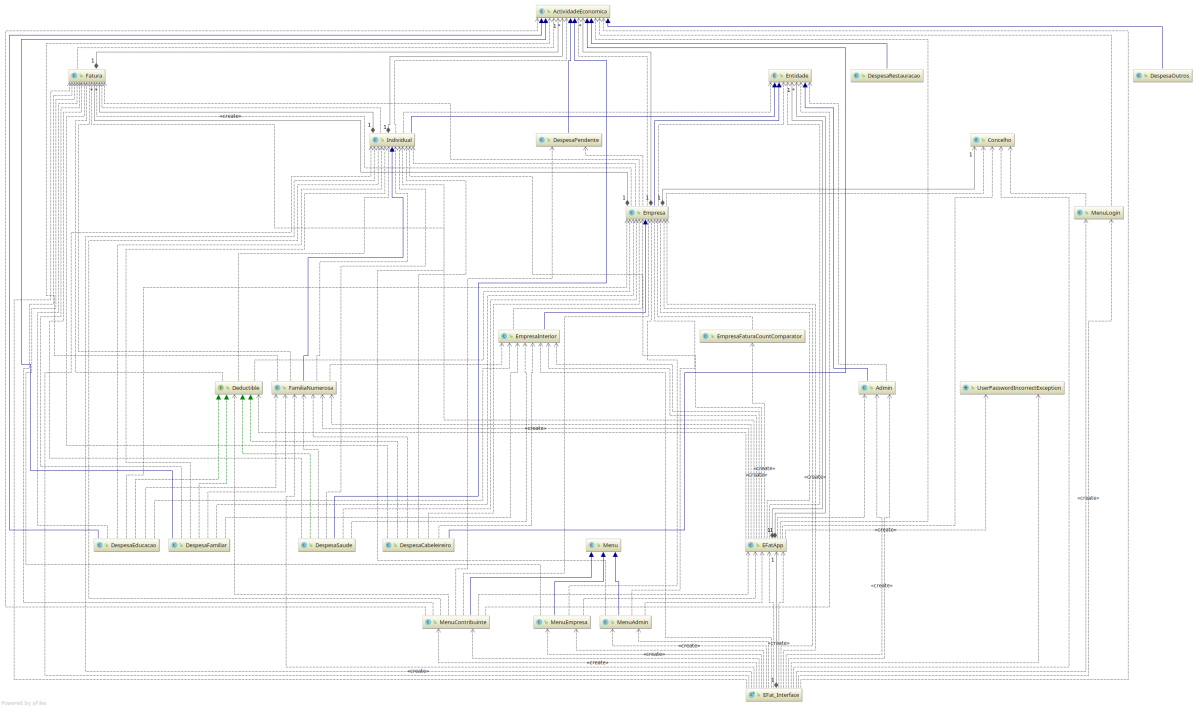


Figure 2

A figura anterior explicita um diagrama de dependências de todo o projecto.
A hierarquia dos utilizadores é apresentada no seguinte diagrama.

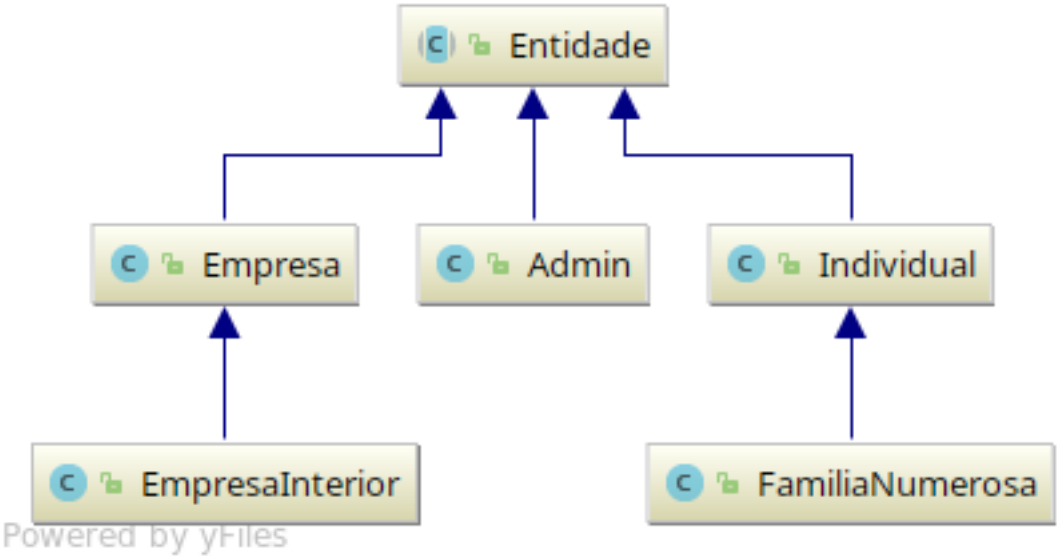


Figure 3

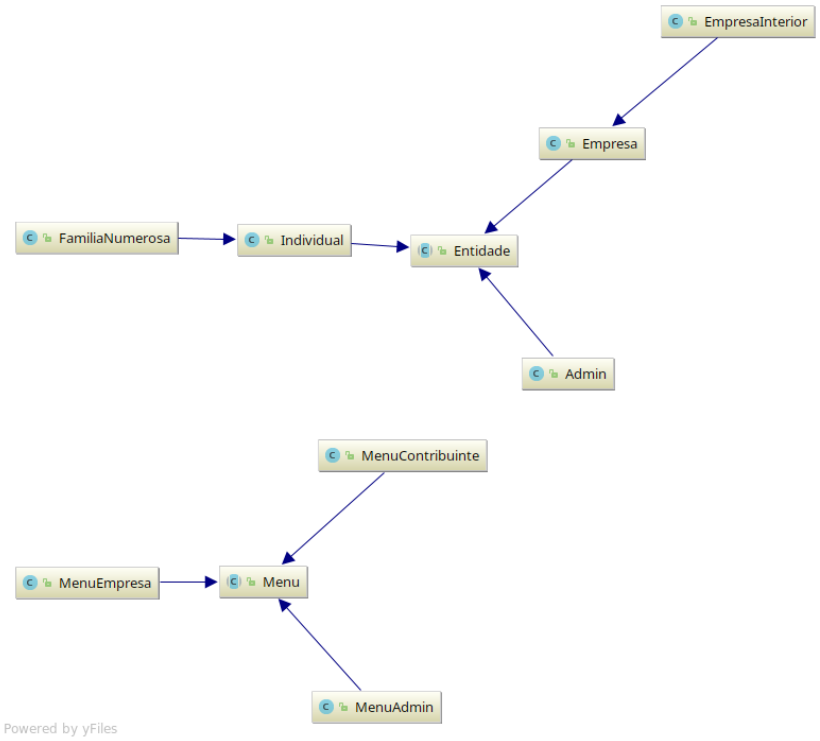


Figure 4

Entidades e menus.

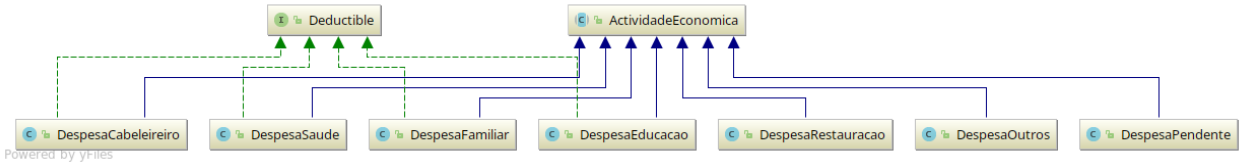


Figure 5

Setores/Actividades.

Entidade

Estas classe não terá métodos muito específicos, sendo estes deixados para Empresas, Individual e Admin. É responsável pela implementação base das 3 entidades. Tanto Empresas, Individual como Admin terão que guardar alguma informação que têm em comum tal como o nif, email, nome, morada e password.

Tanto os contribuintes individuais como as empresas partilham os apontadores das faturas. Esta escolha é totalmente propositada e tem em vista manter a integridade da informação. Caso fosse feito o clone das faturas, a atualização de uma dada fatura não se iria refletir na empresa emitente, por exemplo. Visto que apenas os utilizadores que são contribuintes individuais podem editar as faturas é plenamente viável esta solução. Neste trabalho empresas não podem ser clientes de outras empresas.

```
public abstract class Entidade implements Serializable {

    private static final long serialVersionUID = -2858812485414995458L;
    private int nif;
    private String email;
    private String nome;
    private String morada;
    private String password;
```

Figure 6

Empresas

Classe que especifica uma entidade em empresa. Cada empresa tem uma lista com as Atividades Económicas, o seu código fiscal, uma lista ligada com as faturas correspondentes e o concelho onde é sediada.

```
public class Empresa extends Entidade {

    private static final long serialVersionUID = 5207834366333453518L;
    private List<ActividadeEconomica> codActivades;
    private float cofFiscal;
    private LinkedList<Fatura> listaFact;
    private Concelho concelho;
```

Figure 7

Individual

Classe que especifica de uma entidade num Indivíduo. Cada indivíduo terá número de filhos (dependentes), lista com os números fiscais da família, coeficiente fiscal, lista com as Atividades Económicas e uma lista ligada com as faturas correspondentes.

```
public class Individual extends Entidade {

    private static final long serialVersionUID = -2952930655382522953L;
    private int nDependentes; // nDependentes != nElementos da Familia
    private List<Integer> numFiscais; // Familia
    private float cofFiscal;
    private List<ActividadeEconomica> codigos; // so pode deduzir para estas
    private LinkedList<Fatura> listaFact;
```

Figure 8

Admin

Classe que especifica uma entidade em Administrador. Foi criada para a resolução de algumas queries.

EmpresaInterior

Classe que classifica uma Empresa como Empresa do Interior.

FamiliaNumerosa

Classe que especifica uma classe Individual, afetando a formula de dedução fiscal.

ActividadeEconomica

As atividades económicas foram implementadas como singletons e guardadas num HashSet.

Deductible e Actividades

Visto que apenas algumas actividades económicas podem deduzir, foi escolhido implementar uma interface com métodos de dedução. Desta forma é possóivel filtrar as actividades económicas que podem deduzir pela sua instância e fica a cargo de cada actividade decidir como implementar os métodos. As várias actividades são singletons e guardadas num HashSet.

```java
public abstract class ActividadeEconomica implements Serializable {

    private static final long serialVersionUID = -5009892463782091683L;


    @Override
    public boolean equals(Object obj) { return obj != null && this.getClass() == obj.getClass(); }

    @Override
    public String toString() { return this.getClass().getSimpleName(); }


    private static final class SectorWrapper {

        private static final Set<ActividadeEconomica> actividades = new HashSet<>();

        static{
            actividades.add(DespesaSaude.getInstance());
            actividades.add(DespesaRestauracao.getInstance());
            actividades.add(DespesaCabeleireiro.getInstance());
            actividades.add(DespesaEducacao.getInstance());
            actividades.add(DespesaFamiliar.getInstance());
            actividades.add(DespesaOutros.getInstance());
        }

        private static Set<ActividadeEconomica> get() { return new HashSet<>(actividades); }
    }


    /**
     * @return Todos os setores possiveis de Actividades economicas
     */
    public static Set<ActividadeEconomica> getAllSectors() { return SectorWrapper.get(); }
}
```

Figure 9

```java
public class DespesaEducacao extends ActividadeEconomica implements Deductible {

    private static final DespesaEducacao instance = new DespesaEducacao();
    private static final long serialVersionUID = 1598674907172052678L;
    private static final float limite = 1500;
    private static final float coef = 0.3977f;

    public static DespesaEducacao getInstance() { return instance; }

    private DespesaEducacao() { super(); }


    @Override
    public float deduct_fatura(Fatura f, Individual user, Empresa empresa){
        float tmp = f.getValorPagar() * (coef + (user.getnDependentes() * 0.012f))
        if(empresa instanceof EmpresaInterior){
            tmp *= ((EmpresaInterior)empresa).reducaoImposto();
        }

        return tmp;
    }

    @Override
    public double deduct_total(Double total, Individual user){
        if(user instanceof FamiliaNumerosa){
            total *= ((FamiliaNumerosa) user).reducaoImposto();
        }

        if(total >= limite){
            return limite;
        }else{
            return total;
        }
    }

    protected Object readResolve() { return getInstance(); }

}
```

Figure 10

Concelhos

As empresas têm um concelho onde estão sediadas. Pensou-se em implementar os concelhos da mesma forma que as actividades económicas mas visto que apenas entram em jogo na fórmula de dedução e para explorar a lingaguem Java, implementou-se usando enums. Cada concelho corresponde a um valor que mais tarde entra na fórmula.

Estruturas de Dados

Os dados da EFatApp são guardados em HashMaps visto que permitem o acesso em tempo constante, sendo por isso mais eficiente que outras alternativas dado o seu propósito. Também são utilizados ArrayLists, TreeSets, LinkedLists e HashSets para outros dados.

Interface

De modo a interagir como o mundo exterior, foi criado uma interface de texto. Os vários menus atuam como *views* e a EFatApp como *controller*. Por fim a Interface agrega tudo. As várias entidades são vistas como modelos que contêm apenas lógica interna. Os vários menus especifico agregam as queries relativas a sua entidade.

MenuLogin

Este menu agrega os métodos relativos ao controlo de sessão

MenuEmpresa

O MenuEmpresa é a especificação do Menu para o caso das Empresas.

MenuContribuinte

O MenuContribuinte fará a especificação do Menu para o caso dos Contribuintes Individuais.

MenuAdmin

Este menu foi criado para se aceder à informação global de JavaFatura.

Conclusão

Após a realização deste trabalho, conseguimos não só aprender como fazer um projeto (com início, meio e fim) através da programação por objetos, mas também melhoramos as nossas capacidades gerais como programadores. Os conceitos de hierarquia e de reutilização de código poupam muito trabalho em projetos deste tipo, onde, por exemplo, temos apenas que fazer uma classe de Entidades, sendo preciso adicionar o pouco necessário para criar diferentes tipos de Entidades. Em suma, achamos que cumprimos os objetivos necessários, ficando assim com uma ideia positiva do trabalho final.