

UNIVERSIDADE DO MINHO

Trabalho Prático (JAVA) - JavaFatura

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA

PROGRAMAÇÃO ORIENTADA AOS OBJETOS
(2º SEMESTRE - 2017/2018)

a70565 Bruno Manuel Borlido Arieira 

a73864 João Miguel Freitas Palmeira 

a74264 Rafael Machado da Silva 

Resumo

É do conhecimento geral a existência uma plataforma que disponibiliza aos contribuintes a informação referente às facturas que foram emitidas em seu nome (na realidade a associação é feita através do número de identificação fiscal de cada contribuinte). Desta feita, irá ser elaborado uma plataforma semelhante a referida acima utilizando a linguagem computacional JAVA.

Conteúdo

1	Introdução	4
2	Arquitetura da aplicação	5
2.1	JavaFaturaApp	5
2.2	JavaFatura	6
2.3	Menu	9
2.4	Despesa	9
2.5	Utilizador	10
2.6	Contribuinte	11
2.7	Empresa	12
2.8	AtividadeEconomica	13
2.8.1	Classe Restauracao	14
2.8.2	Classe Educacao	14
2.8.3	Classe Servicos	14
2.8.4	Classe Saude	15
2.8.5	Classe ReparacaoVeiculos	15
2.8.6	Classe Transportes	15
2.8.7	Classe Habitacao	15
2.9	Exceptions	16
2.9.1	UtilizadorInexistenteException	16
2.9.2	UtilizadorRepetidoException	16
2.9.3	ActividadeEconomicaInexistenteException	16
2.9.4	FaturaInexistenteException	16
2.9.5	ActividadeEconomicaRepetidaException	16
2.9.6	UtilizadorSemFaturaException	16
2.10	Comparators	16
2.10.1	ComparatorData	16
2.10.2	ComparatorValor	17
2.10.3	ValueComparator	17
2.11	Manual de Utilização	18
3	Interface	23
3.1	Requisitos	23
3.1.1	Registar um contribuinte, quer seja individual ou empresa	23
3.1.2	Validar o acesso à aplicação utilizando as credenciais (nif e password), por parte diferentes actores	24
3.1.3	Criar facturas associadas a despesas feitas por um contribuinte individual	26
3.1.4	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	27
3.1.5	Corrigir a classificação de actividade económica de um documento de despesa	28
3.1.6	Obter a listagem das facturas de uma determinada empresa, ordenada por data de emissão ou por valor	29
3.1.7	Obter por parte das empresas, as listagens das facturas por contribuinte num determinado intervalo de datas	30
3.1.8	Obter por parte das empresas, as listagens das facturas por contribuinte ordenadas por valor decrescente de despesa	31
3.1.9	Indicar o total facturado por uma empresa num determinado período	32
3.1.10	Determinar a relação dos 10 contribuintes que mais gastam em todo o sistema	33
3.1.11	Determinar a relação das X empresas que mais facturas tem em todo o sistema e o montante de deduções fiscais que as despesas registadas representam	34
4	Conclusão	35

Lista de Figuras

1	Arquitetura de Classes	5
2	Classe JavaFaturaApp	6
3	Classe JavaFatura	8
4	Classe Menu	9
5	Classe Despesa	10
6	Classe Utilizador	11
7	Classe Contribuinte	12
8	Classe Empresa	13
9	Classe AtividadeEconomica	14
10	Classe Restauracao	15
11	Classe UtilizadorInexistenteException	16
12	Primeiro passo intermédio para início da aplicação	18
13	Segundo passo intermédio para início da aplicação	19
14	Menu exibido quando a aplicação é iniciada	19
15	Menu exibido após um contribuinte individual iniciar sessão	20
16	Menu exibido após uma empresa iniciar sessão	21
17	Menu exibido após um administrador iniciar sessão	22
18	Registo dos dois tipos de utilizadores	23
19	Regista contribuinte individual	24
20	Início da Sessão Contribuinte Individual	25
21	Início da Sessão Empresa	25
22	Início da Sessão Administrador	26
23	Cria faturas associada a um contribuinte individual	26
24	Verifica Faturas	27
25	Corrigir uma despesa	28
26	Lista de faturas por data de emissão	29
27	Lista de faturas por valor	30
28	Lista de faturas de um intervalo de datas	31
29	Lista de faturas por valor	32
30	Valor total faturado por uma empresa num período de tempo	33
31	10 contribuintes que mais gastam da aplicação	33
32	Relação de X empresas com mais faturas	34

1 Introdução

Foi-nos proposto pela equipa docente da Unidade Curricular de Programação Orientada aos Objetos a realização de um projeto em JAVA que envolve-se a criação de uma plataforma semelhante *E-Fatura*, isto é, uma plataforma onde cada contribuinte pudesse introduzir todas as suas faturas e realizar deduções fiscais.

Todos os anos todos os contribuintes devem aceder a plataforma em causa e validar certas informações, tais como a associar algum tipo de despesa ao documento da fatura, em situações em que o software não o tenha realizado automaticamente, verificar se a fatura corresponde realmente a um serviço que o contribuinte realizou ou ainda verificar qual é o montante de deduções fiscais que as facturas associadas permitem ter.

Partindo do pressuposto do que foi mencionado anteriormente, a aplicação que iremos criar deverá, entre outros casos, permitir:

- aos contribuintes individuais, aceder às facturas emitidas e associadas ao seu número fiscal;
- aos contribuintes individuais, associar as facturas, ainda por classificar, ao sector de actividade económica respectivo (saúde, educação, restauração, transportes, etc.);
- aos contribuintes individuais, visualizar, e ter o detalhe do cálculo associado, o valor acumulado de deduções fiscais;
- às empresas, associar uma factura de um serviço a um determinado contribuinte.

Para a realização desta plataforma irá também ser necessário, a nosso ver, guardar toda a informação relevante tanto dos contribuintes individuais como das empresas, das atividades económicas bem como associação de empresas a actividades económicas, emissão de facturas, cálculo dos montantes de dedução fiscal associado entre outros requisitos fundamentais para este projeto.

2 Arquitetura da aplicação

Para dar resposta ao problema proposto, optamos pela seguinte arquitetura de classes:

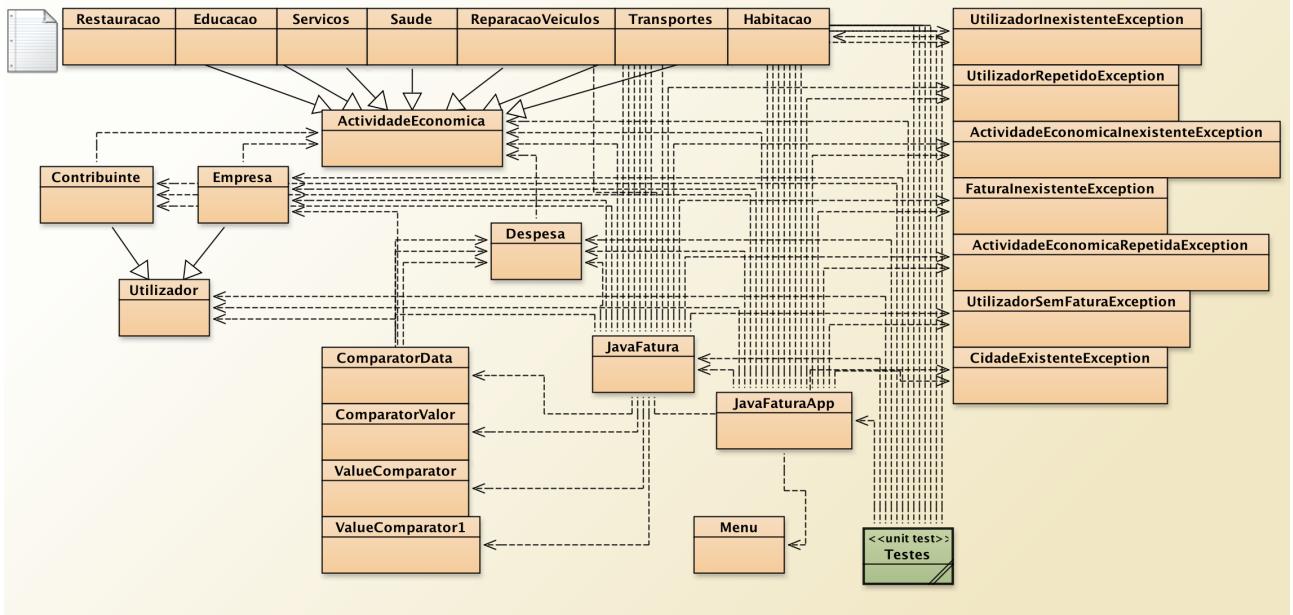


Figura 1: Arquitetura de Classes

2.1 JavaFaturaApp

- É responsável pela interação com o utilizador da app;
- Input/Output existe apenas nesta classe, excluindo as exceções, também presentes em métodos de outras classes;
- Estabelece ligação à classe Menu, gerindo a disposição dos diferentes menus;
- Aqui são chamados os requisitos apresentados no enunciado que fornecem respostas às várias faculdades da aplicação, selecionáveis pelo utilizador;
- As únicas variáveis presentes nesta classe são uma instância da classe JavaFatura, *funcao*, *contribuinte-Sessao* e quatro da classe Menu (menuPrincipal, menuContribuinte, menuEmpresa, menuAdmin);

```

public class JavaFaturaApp {
    private JavaFatura javaFatura;
    private Menu menuPrincipal, menuContribuinte, menuEmpresa, menuAdmin;
    private int funcao; // 1-Contribuinte, 2-Empresa, 3-Administrador
    private int contribuinteSessao; // contribuinte com sessao iniciada

    /**
     * O método main cria a aplicação e invoca o método run()
     */
    public static void main(String[] args) {
        new JavaFaturaApp().run();
    }

    /**
     * Construtor.
     *
     * Cria os menus e a camada de negócio.
     */
    private JavaFaturaApp() {
        // Criar o menu
        String[] opcoes = {"Registrar Utilizador",
                           "Iniciar Sessão",
                           "Listagem de faturas de uma determinada empresa",
                           "Total faturado por uma empresa num período",
                           };
        this.menuPrincipal = new Menu(opcoes);

        String[] opcoesContribuinte = {"Verificar faturas",
                                      "Corrigir classificação de ActividadeEconómica",
                                      "Associar Actividades Económicas"
                                      };
        this.menuContribuinte = new Menu(opcoesContribuinte);

        String[] opcoesEmpresa = {"Criar Faturas associadas a um contribuinte",
                                  "Faturas por contribuinte de um intervalo de datas",
                                  "Faturas por contribuinte ordenadas por valor"
                                  };
    }
}

```

Figura 2: Classe JavaFaturaApp

2.2 JavaFatura

- Esta é a classe principal apesar de a *main* estar na classe *JavaFaturaApp*;
- Aqui foram introduzidos os métodos requisitados no enunciado;
- Começamos por definir algumas variáveis de instância:

```

private Map<Integer,Utilizador> utilizadores;
private Map<Integer, List<Despesa>> faturas;
private Map<Integer, List<Integer>> despesasAlterada;
private Map<String,Double> concelhos;

```

- Estas variáveis que definimos são os *maps* criados para guardar todos os utilizadores, todas as faturas e todas as despesas. Foram também definidos todos os métodos necessários bem como os seus *gets* e *sets* e ainda os métodos *equals*, *toString* e *clone*.
- O método *iniciaSessao(int contribuinte, String pass)* verifica inicialmente se o utilizador se encontra registado, ou seja, se o contribuinte já existe na plataforma. Caso isso aconteça e se é verificado que o contribuinte é do administrador, é, posteriormente, avaliada a palavra passe e caso seja válida entra na app caso não seja é lançada uma exceção. Se o contribuinte não for do administrador, mas existir na app, é verificada também a palavra passe do mesmo, seguindo a mesma ideologia do primeiro caso;
- O *adicionaU(Utilizador u)* verifica através do nif de u, ou seja, pesquisa no *containsKey* do *Map* se este já existe na lista de utilizadores. Caso não exista insere-o, caso contrário lança uma exceção.
- No método *associarActEmp(ActividadeEconomica act, int contribuinte)* é percorrida a lista dos utilizadores comparando o nif passado como argumento com todos os existentes e caso exista um igual, é verificada a empresa do mesmo contribuinte. Caso a empresa já contenha a atividade económica em questão, é lançada uma exceção, caso contrário é-lhe adicionada a atividade através do método disponibilizado na classe *Empresa*.
- No *associarActCont(ActividadeEconomica act, int contribuinte)* o processo é igual embora agora o contribuinte seja individual e não uma empresa.

- Através do método *criaFaturaAss*(*Despesa d*) é possível criar uma fatura associada a um contribuinte individual. Recebe uma despesa criada pela Empresa imitante e antes de a emitir vai confirmar se a empresa imitante possui o código de atividade económica dessa despesa. Após a confirmação, verifica-se se o contribuinte existe na aplicação e, caso não exista, lança uma excessão. Caso exista, a fatura é lançada para a lista das faturas.
- Já no método *validaFact*(*int cF*) é validado automaticamente todas as faturas de um contribuinte. É acedida a estrutura das faturas e verifica se o contribuinte em questão tem ou não faturas e se este tiver faturas, realiza a validação, caso contrário é lançada uma exceção.
- O *adicionaId*(*int nif, int id*) é um método que adiciona o id de uma despesa que um contribuinte modificou.
- No método *consultaFat*(*int nifC*) é criada uma lista para serem adicionadas de um dado contribuinte. Após isso verifica-se se o nif do contribuinte passado como argumento está contido na lista das faturas através do *containsKey* do *Map* e, caso esteja contido, as faturas são guardadas na nova lista e essa mesma lista é retornada no fim. Caso contrário é retornada uma excessão.
- Já no método *consultaDeducoes*(*int nifC*) é dado o contribuinte que tem função iniciada. É verificado se este tem ou não faturas em seu nome. Caso o contribuinte tenha faturas em seu nome, a aplicação vai guardar numa lista os nif dos agregados a ele. De seguida vai percorrer a lista dos agregados de modo a verificar se estes também têm faturas associadas a si mesmos. Caso tenham faturas, guardas as válidas para depois calcular a dedução de cada uma das despesas.
- *calcula(List<Despesa> despesas)* é uma função auxiliar do método anterior que irá receber uma lista de despesas de um contribuinte e vai calcular para cada uma das despesas que esse contribuinte possui a dedução de cada despesa. Por último irá calcular a soma de todas as deduções realizadas.
- O *reducaoImposto*(*double total, int nife, int nifC, Despesa d*) é um requisito avançado que previliga as famílias numerosas ou as empresas pertencentes a uma cidade com incentivo fiscal.
- O método *adicionaConcelho*(*String cidade, double perc*) irá adicionar um concelho à estrutura de dados que possui incentivo fiscal associando a ele um valor percentual.
- Através do *factValiDedu*(*List<Integer> nFis*) é possível verificar as faturas validas de um determinado contribuinte e do seu agregado. Inicialmente cria-se dois *maps* (um para o resultado final e outra para guardar as faturas válidas do contribuinte) e duas listas (uma auxiliar e uma para as deduções).
- O método que fornece as faturas deduzíveis de um determinado contribuinte e do seu agregado *factDedu*(*Map<Integer,List<Despesa> factVali*) recebe uma lista com as faturas válidas como argumento.
- No *faturasInvalidas*(*int nifC*) que recebe um nif de um contribuinte como argumento é realizada uma procura no *map* das faturas e caso existam faturas registadas naquele nif são guardadas numa lista auxiliar. De seguida são procuradas nessa lista as despesas que têm a variável validade como falso e ao encontrar essas despesas guarda-as noutra lista que irá retornar no final.
- Já no *codActiEmp*(*int id, List<Despesa> despInv*) é recebido o id da despesa que o contribuinte pertende alterar e a lista de despesas por validar. Verifica se o id é correto, caso seja correto indica as atividades económicas que essa empresa possui.
- Através do método *codActiCont*(*int nif*) um contribuinte individual consegue saber quais as atividades económicas que pode deduzir.
- Com o método *corrigefact*(*int id, ActividadeEconomica ae, int nif*) é disponibilizada ao utilizador a opção de poder corrigir uma fatura, isto é, é-lhe permitido alterar a atividade económica de determinada fatura e validar a mesma no final.
- Com a *ordenaDataFact*(*int nif*) é possível consultar as faturas de uma empresa por ordem de data pesquisando no *containsKey* do *map* o nif do utilizador e depois captar na lista de despesas desse mesmo utilizador, todas as faturas desse contribuinte.
- No método *ordenaValorFact*(*int nif*) é aplicada a mesma ideia da função anterior embora a consulta neste caso seja realizada através do valor da fatura.

- Graças à função `consuFactData(int nifE, int nifC, LocalDate d1, LocalDate d2)` é possível consultar as faturas de um dado contribuinte que esteja entre duas datas. Numa primeira fase são recolhidas todas as faturas do contribuinte pesquisando no `containsKey` do map dos contribuintes e também no das faturas. Numa segunda fase é captado da lista que foi gerada, pelo meio de um ciclo `for`, todas as faturas que estão nessa lista entre as duas datas fornecidas inicialmente.
- Na função `consuFactValor(int nifE, int nifC)` é realizada uma pesquisa através dos `containsKey` dos maps dos contribuintes e das faturas para se realizar um `clone` de todas as faturas desse contribuinte. Já na parte final desta função é aplicado um `sort` e, posteriormente, um `reverse` de modo a ordenar de modo decrescente a lista.
- Devido à ferramenta `totalFact(int nifE, LocalDate d1, LocalDate d2)` é possível recolher a lista de todas as faturas de uma dada empresa.
- A função `Map<Contribuinte,Double> topCliente()` é o método que realiza a *query* do top 10 contribuintes que mais gastam na aplicação. Neste caso irá-se percorrer todos os utilizadores e realizar a soma de todas as faturas dos mesmos e guardar essa soma num map. Posteriormente irá-se realizar um `sort` nesse `map` de modo a que fique ordenado de modo decrescente e retorna essa mesma lista.
- Com o predicado `Map<Empresa,Double> topEmpresas()` é possível verificar as n empresas com mais faturas na aplicação bem como as suas deduções das despesas que essas empresas emitiram.
- Por último, definiu-se os métodos `equals`, `toString` e `clone`.

```

public class JavaFatura implements Serializable{
    // variáveis de instância

    private Map<Integer,Utilizador> utilizadores;
    private Map<Integer, List<Despesa>> faturas;
    private Map<Integer, List<Integer>> despesasAlterada;

    public JavaFatura(){
        this.utilizadores = new HashMap<>();
        this.faturas = new HashMap<>();
        this.despesasAlterada = new HashMap<>();
    }

    public JavaFatura(Map<Integer,Utilizador> utilizadores, Map<Integer,List<Despesa>> faturasS, Map<Integer, List<Integer>> desp){
        this.utilizadores = new HashMap<>();
        for(Utilizador u : utilizadores.values()){
            this.utilizadores.put(u.getNif(),u.clone());
        }
        this.faturas = new HashMap<>();
        for(List<Despesa> despesas : faturasS.values()){
            for(Despesa d : despesas){
                this.faturas.put(d.getNif(),despesas);
            }
        }
        this.despesasAlterada = new HashMap<>();
        for(List<Integer> ids : desp.values()){
            for(Integer i : ids){
                this.despesasAlterada.put(i,ids);
            }
        }
    }

    public JavaFatura(JavaFatura jf){
        this.utilizadores = jf.getUtilizadores();
        this.faturas = jf.getFaturas();
    }
}

```

Figura 3: Classe JavaFatura

2.3 Menu

- Permite a formação de opções que dão opção de escolha ao utilizador das funcionalidades. É a classe responsável por permitir a representação dos menus no ecrã.
- São definidas duas variáveis de instância, designadas *opcoes* e *op*, respetivamente. A primeira é uma lista de Strings onde representa as opções que aparecem no menu e a segunda é um inteiro que guarda a opção escolhida pelo utilizador.

```
public class Menu
{
    // variáveis de instância

    private List<String> opcoes;
    private int op;

    /**
     * Constructor for objects of class Menu
     */
    public Menu(String[] opcoes) {
        this.opcoes = Arrays.asList(opcoes);
        this.op = 0;
    }

    /**
     * Método para apresentar o menu e ler uma opção.
     */
    public void executa() {
        do {
            showMenu();
            this.op = lerOpcao();
        } while (this.op == -1);
    }

    /** Apresentar o menu */

    private void showMenu() {
        System.out.println("\n ***** Menu ***** ");
        int n = this.opcoes.size();
        for (int i=0; i<n; i++) {
            System.out.print(i+1);
            System.out.print(" - ");
            System.out.println(this.opcoes.get(i));
        }
    }
}
```

Figura 4: Classe Menu

2.4 Despesa

- Classe que implementa todas as despesas obtidas pelo utilizador.
- Uma despesa tem como atributos, requisitados pelo enunciado, nif do emitente, designação do emitente, data da despesa, nif do cliente, descrição da despesa, a atividade económica da respetiva despesa e ainda o seu montante.
- Ainda criamos outras duas variáveis de instância designadas *id*, que presenta o id da fatura da despesa, e *validade*, que confirma se a despesa foi ou não validada pelo contribuinte em causa.

```

public class Despesa implements Serializable
{
    // variaveis de instancia criadas pelo grupo
    private int id;                                // id da fatura
    private boolean validade;                      // se a despesa foi validada pelo contribuinte

    // variaveis de instancia pedidas no enunciado
    private int nife;                             // numero fiscal do emitente
    private String designacaoE;                  // designação do emitente
    private LocalDate data_despesa;               // data da despesa
    private int nifc;                            // numero fiscal do cliente
    private String designacaoD;                  // Descrição da despesa
    private ActividadeEconomica naturezaDespesa; // Atividade economica da despesa (ex:Saudade)
    private double valor_despesa;                // montante da despesa

    public Despesa(){
        this.id = 0;
        this.validade = false;
        this.nife = 0;
        this.designacaoE = " ";
        this.data_despesa = null;
        this.nifc = 0;
        this.designacaoD = " ";
        this.naturezaDespesa = new ActividadeEconomica();
        this.valor_despesa = 0.0;
    }

    public Despesa(Despesa d){
        this.id = d.getId();
        this.validade = d.getValidade();
        this.nife = d.getNife();
        this.designacaoE = d.getDesignacaoEmpresa();
        this.data_despesa = d.getDataDespesa();
        this.nifc = d.getNifC();
        this.designacaoD = d.getDesignacaoD();
        this.naturezaDespesa = d.getNaturezaDespesa();
        this.valor_despesa = d.getValorDespesa();
    }
}

```

Figura 5: Classe Despesa

2.5 Utilizador

- Classe que contém as variáveis de instância que são semelhantes tanto como para o utilizador individual, como para uma empresa.
- Assim sendo os atributos definidos foram um inteiro *nif* (que corresponde ao número de identificação fiscal) e quatro Strings que são relativas ao email, nome, morada e palavra-passe.

```

public class Utilizador implements Serializable
{
    // variáveis de instância

    private int nif;
    private String email;
    private String nome;
    private String morada;
    private String pass;

    public Utilizador(){
        this.nif = 000000000;
        this.email = " ";
        this.nome = " ";
        this.morada = " ";
        this.pass = " ";
    }

    /**
     * construtor por parâmetros
     */

    public Utilizador(int nif, String email, String nome, String morada, String pass){
        this.nif = nif;
        this.email = email;
        this.nome = nome;
        this.morada = morada;
        this.pass = pass;
    }

    /**
     * construtor por cópia
     */

    public Utilizador(Utilizador u){
        this.nif = u.getNif();
        this.email = u.getEmail();
        this.nome = u.getNome();
    }
}

```

Figura 6: Classe Utilizador

2.6 Contribuinte

- Estende a classe *Utilizador* e contém os atributos adicionais que são representativos dos utilizadores individuais.
- As respetivas variáveis de instância são dois inteiros *nAgrFam* (número de dependentes do agregado familiar) e *coef* (coeficiente para as deduções fiscais), uma lista *nFa* (contém os números de identificação fiscal de cada agregado familiar) e um Map *codAct* (lista com as atividades económicas para as deduções de cada contribuinte).

```

public class Contribuinte extends Utilizador implements Serializable
{
    // variáveis de instância
    private int nAgrFam;                                // numero de dependentes do agregado familiar
    private int coef;                                    // coeficiente fiscal para efeitos de dedução fiscal
    private List<Integer> nFa;                          // numeros fiscais do agregado familiar(nifs)
    private Map<Integer,ActividadeEconomica> codAct;   // lista das actividades económicas para as quais os contribuintes tem dedução

    public Contribuinte(){
        super();
        this.nAgrFam = 0;
        this.coef = 0;
        this.nFa = new ArrayList<>();
        this.codAct = new HashMap<>();
    }

    public Contribuinte(int nif, String email, String nome, String morada, String pass,
                        int nAgrFam, int coef, List<Integer> numeroFiscais, Map<Integer,ActividadeEconomica> codActividades) {
        super(nif,email,nome,morada,pass);
        this.nAgrFam = nAgrFam;
        this.nFa = new ArrayList<>();
        for(Integer i : numeroFiscais){
            this.nFa.add(i);
        }
        this.codAct = new HashMap<>();
        for(ActividadeEconomica ae : codActividades.values()){
            this.codAct.put(ae.getCodigo(),ae.clone());
        }
    }

    public Contribuinte(Contribuinte c){
        super(c);
        this.nAgrFam = c.getNagrFam();
        this.nFa = c.getNfa();
        this.codAct = c.getCodAct();
    }
}

```

Figura 7: Classe Contribuinte

2.7 Empresa

- Estende a classe utilizador, e constitui um dos dois tipos de "clientes".
- A classe empresa integra como variáveis de instância a String *designacao* (indica a função da respetiva empresa), String *localizacao* (que caso esteja localizada no interior é privilegiada, muda o valor de dedução fiscal), o inteiro *factDeducao* (fator de dedução) e por fim, criamos um Map designado *activEcono*, que representa todas as atividades económicas relacionadas com a empresa.

```

public class Empresa extends Utilizador implements Serializable
{
    // variáveis de instância
    private String designacao;                                // designacão da função da empresa em questão
    private String localizacao;                             // interior / litoral (prevé ligar interior)
    private int factDeducao;                               // fator de dedução de uma empresa
    private Map<Integer,ActividadeEconomica> activEcono; // lista de actividade económicas para que a empresa deduz

    public Empresa(){
        super();
        this.designacao = " ";
        this.localizacao = " ";
        this.factDeducao = 0;
        this.activEcono = new HashMap();
    }

    public Empresa(int nif, String email, String nome, String morada, String pass,
                  String designacao, String localizacao, int factDeducao, Map<Integer,ActividadeEconomica> actEcono){
        super(nif,email,nome,morada,pass);
        this.designacao = designacao;
        this.localizacao = localizacao;
        this.factDeducao = factDeducao;
        this.activEcono = new HashMap<>();
        for(ActividadeEconomica ae : actEcono.values()){
            this.activEcono.put(ae.getCodigo(),ae.clone());
        }
    }

    public Empresa(Empresa e){
        super(e);
        this.designacao = e.getDesignacao();
        this.localizacao = e.getLocalizacao();
        this.factDeducao = e.getFactDeducao();
        this.activEcono = e.getActivEcono();
    }
}

```

Figura 8: Classe Empresa

2.8 AtividadeEconomica

```

public class AtividadeEconomica implements Serializable
{
    private boolean deduzivel; // true-> é deduzivel false -> não é
    private int codigo; // código da Atividade Económica // 1-Restauração , 2-Educação , etc

    public AtividadeEconomica(){
        this.deduivel = false;
        this.codigo = 0;
    }

    public AtividadeEconomica(boolean deduzivel, int codigo){
        this.deduivel = deduzivel;
        this.codigo = codigo;
    }

    public AtividadeEconomica(AtividadeEconomica ae){
        this.deduivel = ae.getDeduivel();
        this.codigo = ae.getCodigo();
    }

    // método getters

    public boolean getDeduivel(){
        return this.deduivel;
    }

    public int getCodigo(){
        return this.codigo;
    }

    // método setters

    public void setDeduivel(boolean deduzivel){
        this.deduivel = deduzivel;
    }

    public void setCodigo(int codigo){
        this.codigo = codigo;
    }
}

```

Figura 9: Classe AtividadeEconomica

Aqui estão as diferentes atividades económicas:

1. Restauração
2. Educação
3. Serviços
4. Saúde
5. Reparação de Veículos
6. Transportes
7. Habitação

Para cada uma delas foi criada uma classe onde se declarou duas variáveis, uma para o valor da dedução fiscal (*valorDeducao*) e outra para a descrição da atividade (*descricao*) e implementou os respetivos métodos. De seguida apresentamos, as respetivas classes, um pouco mais detalhadas e um exemplo de uma dessas classes:

2.8.1 Classe Restauracao

Subclasse que contém os dados relativos à atividade económica restauração. É constituída pelo *valorDeducao* e por uma String *descricao*. Tem como código, na classe AtividadeEconomica, o inteiro 1.

2.8.2 Classe Educacao

Tal como na subclasse anterior tem como variáveis *valorDeducao* e *descricao*. Tem como código, na classe AtividadeEconomica, o inteiro 2.

2.8.3 Classe Servicos

Subclasse que tem as duas variáveis definidas nas classes anteriores, mais a variável *tipoServico* que pode ser agua, gás, eletricidade e saneamento. Tem como código, na classe AtividadeEconomica, o inteiro 3.

2.8.4 Classe Saude

Esta subclasse contém as duas variáveis definidas nas últimas subclasses. Tem como código, na classe AtividadeEconomica, o inteiro 4.

2.8.5 Classe ReparacaoVeiculos

Tem como nova variável *tipoVeiculo* que pode ser uma moto, carro, camiao, etc, e tem como código, na classe AtividadeEconomica, o inteiro 5.

2.8.6 Classe Transportes

Subclasse que faz referência á atividade económica ao nível dos transportes públicos, tendo as duas variáveis usuais referidas anteriormente mais a variável *tipoTransporte*. Tem como código, na classe AtividadeEconomica, o inteiro 6.

2.8.7 Classe Habitacao

Subclasse que contem os dados de valor de dedução, descrição e tipo de habitação(apartamento,casa,etc). Tem como código, na classe AtividadeEconomica, o inteiro 7.

```
public class Restauracao extends AtividadeEconomica implements Serializable
{
    private double valorDeducao;
    private String descricao;

    public Restauracao(){
        super();
        this.valorDeducao = 0.0;
        this.descricao = " ";
    }

    public Restauracao(boolean deducao,int codigo,double valorDeducao, String descricao){
        super(deducao,codigo);
        this.valorDeducao = valorDeducao;
        this.descricao = descricao;
    }

    public Restauracao(Restauracao r){
        super(r);
        this.valorDeducao = r.getValorDeducao();
        this.descricao = r.getDescricao();
    }

    // metodos getters

    public double getValorDeducao(){
        return this.valorDeducao;
    }

    public String getDescricao(){
        return this.descricao;
    }

    // metodos setters

    public void setValorDeducao(double valor){
        this.valorDeducao = valor;
    }
}
```

Figura 10: Classe Restauracao

2.9 Exceptions

Este tipo de classes são importantes pois são responsáveis por evitar erros durante a execução do programa. Temos como exemplos de erros, a sobreposição de dados ou a inexistência destes, a inserção de dados incorretos e problemas de acesso.

2.9.1 UtilizadorInexistenteException

Subclasse responsável por verificar se determinado utilizador existe, por exemplo, ao tentar iniciar sessão.

```
/**  
 * Escreva a descrição da classe UtilizadorInexistenteException aqui.  
 *  
 * @author (seu nome)  
 * @version (número de versão ou data)  
 */  
public class UtilizadorInexistenteException extends Exception  
{  
    public UtilizadorInexistenteException() {  
        super();  
    }  
    public UtilizadorInexistenteException(String message) {  
        super(message);  
    }  
}
```

Figura 11: Classe UtilizadorInexistenteException

2.9.2 UtilizadorRepetidoException

Subclasse responsável por verificar se dado utilizador já existe na aplicação. Este caso acontece quando se efetua o registo.

2.9.3 ActividadeEconomicaInexistenteException

Subclasse responsável por verificar se a atividade económica existe, por exemplo, quando se associa uma atividade económica a despesas.

2.9.4 FaturaInexistenteException

Esta subclasse é responsável por verificar, por exemplo, as faturas de um contribuinte.

2.9.5 ActividadeEconomicaRepetidaException

Subclasse que verifica se determinada atividade económica existe, por exemplo, ao associar uma atividade económica a um contribuinte individual.

2.9.6 UtilizadorSemFaturaException

Subclasse que verifica se um determinado utilizador da aplicação tem fatura, por exemplo, ao consultar as faturas de um contribuinte.

2.10 Comparators

Este tipo de classes têm como finalidade principal a comparação (como o próprio nome sugere) de dados pretendidos para se obter uma melhor inserção ou consultas dos mesmos na aplicação.

2.10.1 ComparatorData

Esta subclasse tem como função comparar duas datas relativas a despesas.

2.10.2 ComparatorValor

Subclasse que tem como funcionalidade a comparação do montante entre duas despesas.

2.10.3 ValueComparator

Subclasse **Value**

Subclasse que tem como funcionalidade comparar dois values num map, de modo a ordená-lo.

2.11 Manual de Utilização

Nesta secção, é disponibilizado um pequeno manual para facilitar a utilização desta aplicação. Para iniciar a sua execução, é necessário primeiramente aceder à classe JavaFaturaAPP com o botão direito do rato e selecionar a função main como se vê na figura abaixo.

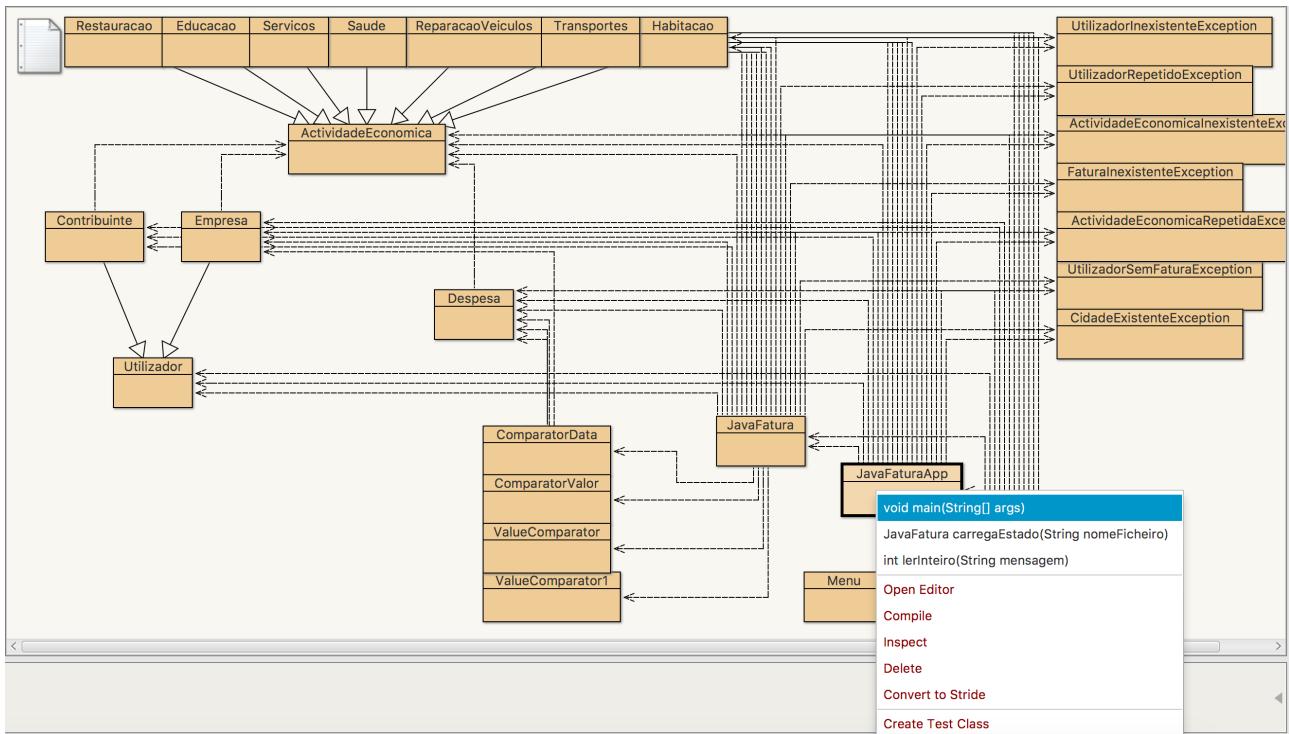


Figura 12: Primeiro passo intermédio para início da aplicação

De seguida, o segundo passo é selecionar apenas a opção OK na janela disponibilizada.

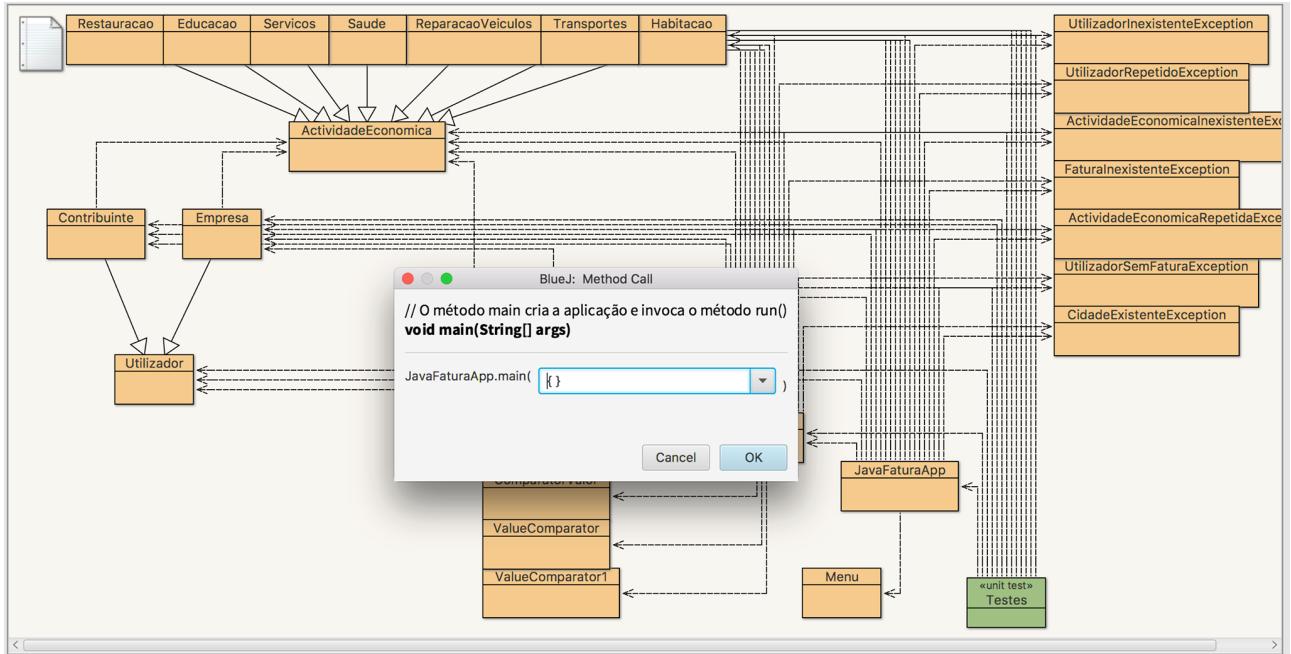


Figura 13: Segundo passo intermédio para início da aplicação

Após os passos anteriores, surge uma janela de terminal do BlueJ com um menu principal que possui várias funcionalidades. Para aceder a estas funcionalidades, escolhe-se o número correspondente a essa opção.

```
***** Menu *****
1 - Registar Utilizador
2 - Iniciar Sessão
3 - Listagem de faturas de uma determinada empresa
4 - Total faturado por uma empresa num periodo
0 - Sair
Opção :
```

Figura 14: Menu exibido quando a aplicação é iniciada

Dessas opções, podemos destacar a de registar utilizador em que fornecemos instruções claras durante a execução para melhor guiar alguém menos experiente e ainda a de iniciar sessão, que conforme o utilizador que inicia sessão seja contribuinte individual ou empresa ou administrador, vai apresentar menus diferentes, pois estes têm acesso a funcionalidades diferentes na aplicação. Como se pode visualizar, para iniciar sessão basta fornecer o nif e a password e mais funcionalidades serão apresentadas, podendo qualquer uma delas ser também selecionada.

```
Bem-Vindo ao JavaFatura !!!  
  
***** Menu *****  
1 - Registar Utilizador  
2 - Iniciar Sessão  
3 - Listagem de faturas de uma determinada empresa  
4 - Total faturado por uma empresa num período  
0 - Sair  
Opção: 2  
Digite os seus dados  
Contribuinte: 1111111111  
Password: 11  
  
Validou a sua entrada como Contribuinte individual!  
  
***** Menu *****  
1 - Verificar faturas  
2 - Corrigir classificação de ActividadeEconómica  
3 - Associar Actividades Económicas  
0 - Sair  
Opção:
```

Figura 15: Menu exibido após um contribuinte individual iniciar sessão

```
***** Menu *****
1 - Registar Utilizador
2 - Iniciar Sessão
3 - Listagem de faturas de uma determinada empresa
4 - Total faturado por uma empresa num periodo
0 - Sair
Opção: 2
Digite os seus dados
Contribuinte: 123456789
Password: 12

Validou a sua entrada como Empresa!

***** Menu *****
1 - Criar Faturas associadas a um contribuinte
2 - Faturas por contribuinte de um intervalo de datas
3 - Faturas por contribuinte ordenadas por valor
4 - Associar Actividades Economicas e respectivas deduções
0 - Sair
Opção:
```

Figura 16: Menu exibido após uma empresa iniciar sessão

```
***** Menu *****  
1 - Registrar Utilizador  
2 - Iniciar Sessão  
3 - Listagem de faturas de uma determinada empresa  
4 - Total faturado por uma empresa num periodo  
0 - Sair  
Opção: 2  
Digite os seus dados  
Contribuinte: 1234  
Password: admin  
  
Validou a sua entrada como Administrador!  
  
***** Menu *****  
1 - Os dez contribuintes com mais gastos  
2 - Relação de X empresas mais fatura  
3 - Adiciona cidade com incentivo fiscal  
0 - Sair  
Opção:
```

Figura 17: Menu exibido apóis um administrador iniciar sessão

3 Interface

3.1 Requisitos

3.1.1 Registar um contribuinte, quer seja individual ou empresa

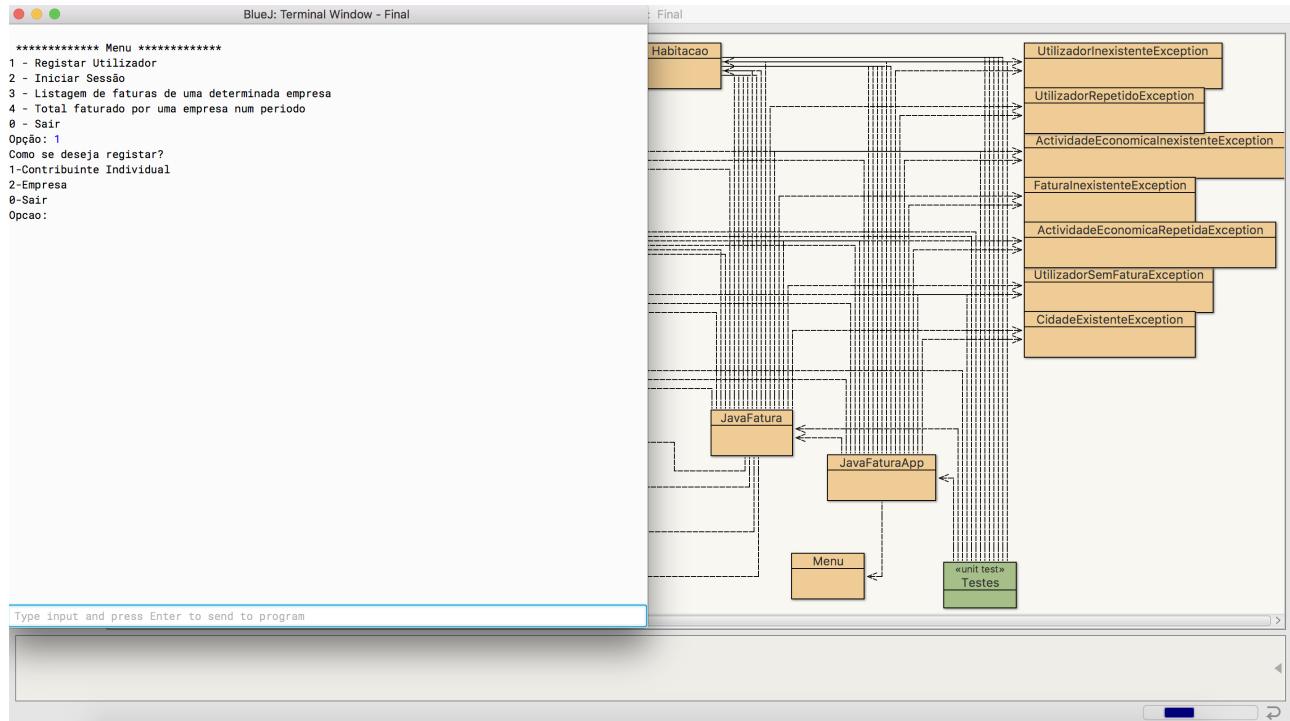


Figura 18: Registo dos dois tipos de utilizadores

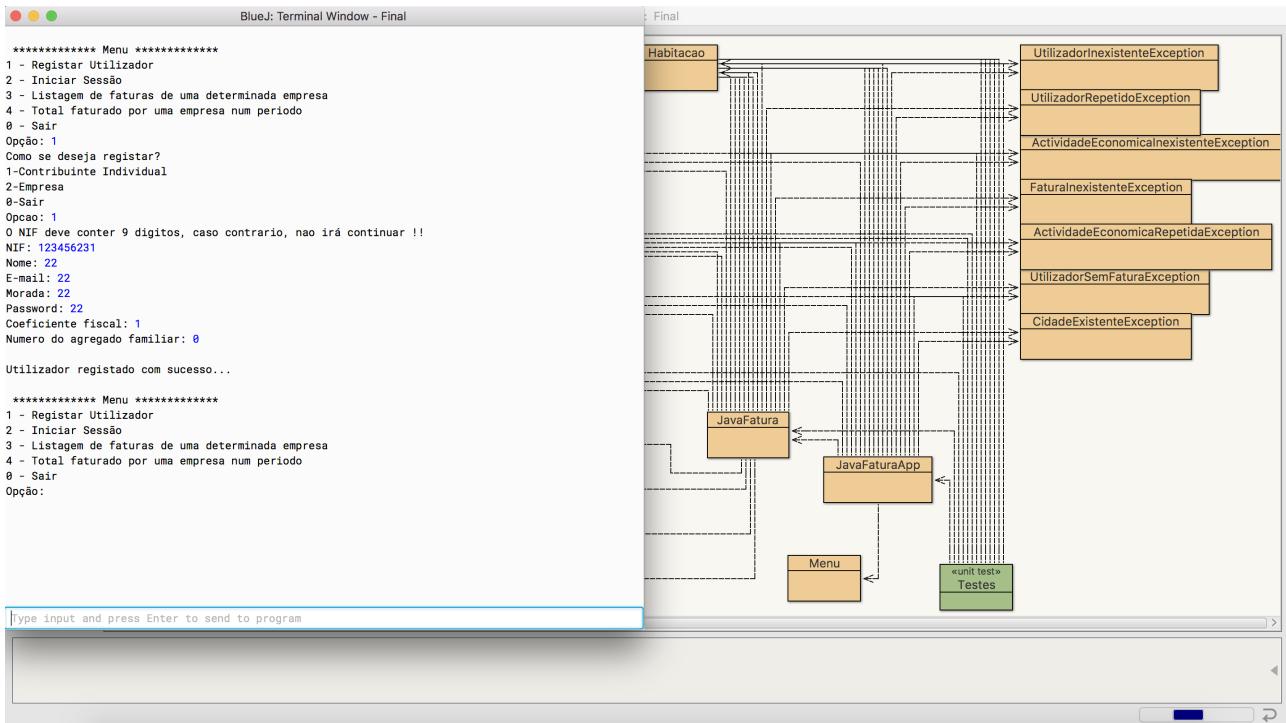


Figura 19: Regista contribuinte individual

3.1.2 Validar o acesso à aplicação utilizando as credenciais (nif e password), por parte diferentes actores

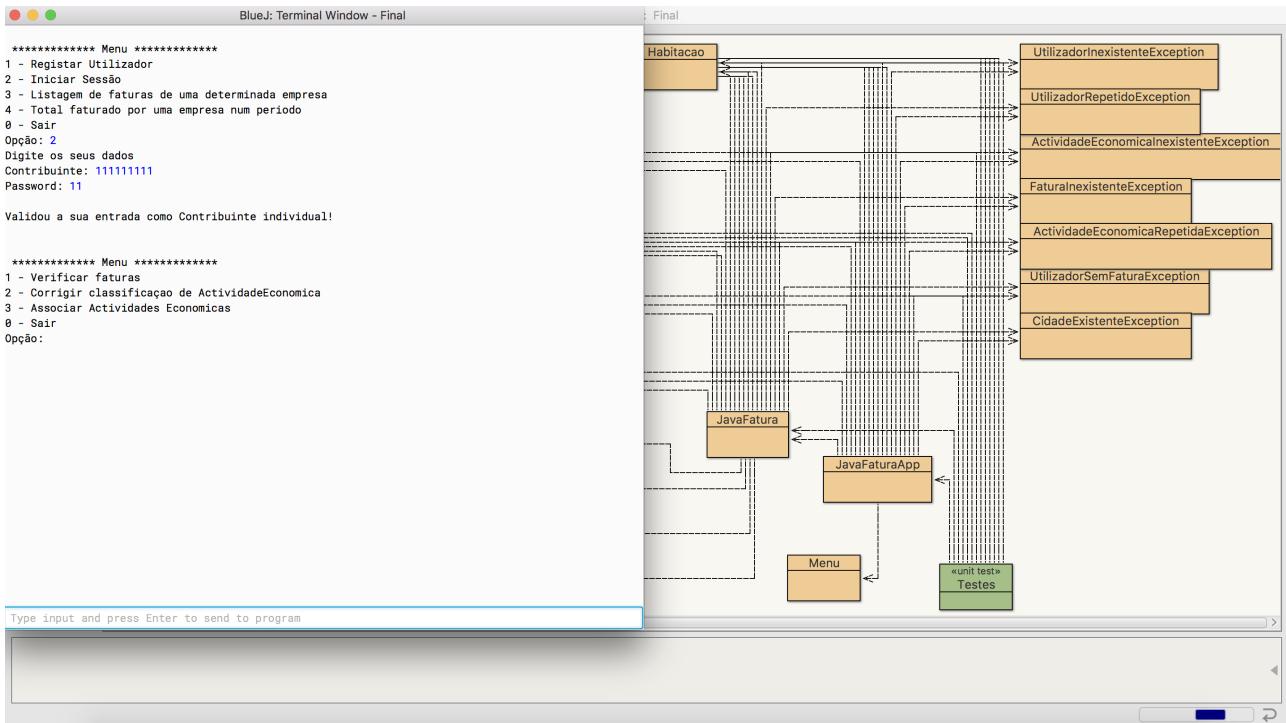


Figura 20: Início da Sessão Contribuinte Individual

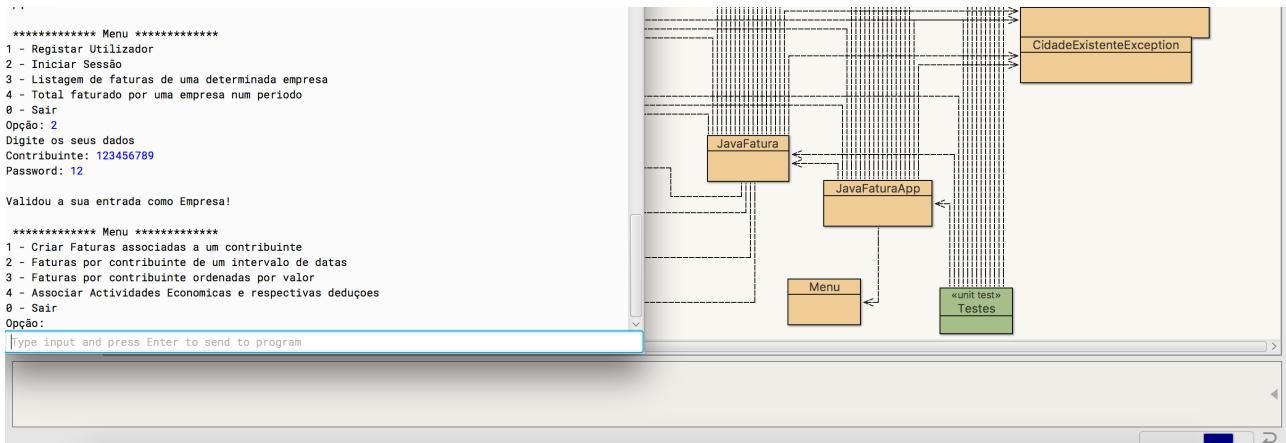


Figura 21: Início da Sessão Empresa

```
***** Menu *****
1 - Registar Utilizador
2 - Iniciar Sessão
3 - Listagem de faturas de uma determinada empresa
4 - Total faturado por uma empresa num periodo
0 - Sair
Opção: 2
Digite os seus dados
Contribuinte: 1234
Password: admin

Validou a sua entrada como Administrador!
```

```
***** Menu *****
1 - Os dez contribuintes com mais gastos
2 - Relação de X empresas mais fatura
3 - Adiciona cidade com incentivo fiscal
0 - Sair
Opção:
```

Type input and press Enter to send to program

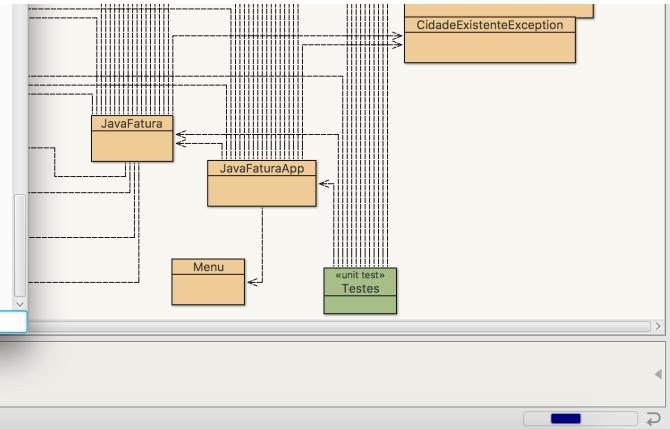


Figura 22: Início da Sessão Administrador

3.1.3 Criar facturas associadas a despesas feitas por um contribuinte individual

```
Validou a sua entrada como Empresa!
***** Menu *****
1 - Criar Faturas associadas a um contribuinte
2 - Faturas por contribuinte de um intervalo de datas
3 - Faturas por contribuinte ordenadas por valor
4 - Associar Actividades Economicas e respectivas deduções
0 - Sair
Opção: 1
Introduza o NIF do cliente. (9 -> numeros) !!
NIF: 111111111
Ano: 2018
De 1 a 12, para relembrar
Mês: 2
Há no máximo 31 dias por mês
Dia: 2
Introduza uma breve descrição da despesa: www
Qual é a natureza da despesa:
1-Restauracao
2-Educacao
3-Servicos
4-Saude
5-Reparacao de Veiculos
6-Transportes
7-Habitacao
0-Para sair
Opção:2
Qual o valor da despesa: 20
Fatura criada com sucesso...

***** Menu *****
1 - Criar Faturas associadas a um contribuinte
2 - Faturas por contribuinte de um intervalo de datas
3 - Faturas por contribuinte ordenadas por valor
4 - Associar Actividades Economicas e respectivas deduções
0 - Sair
Opção:
```

Type input and press Enter to send to program

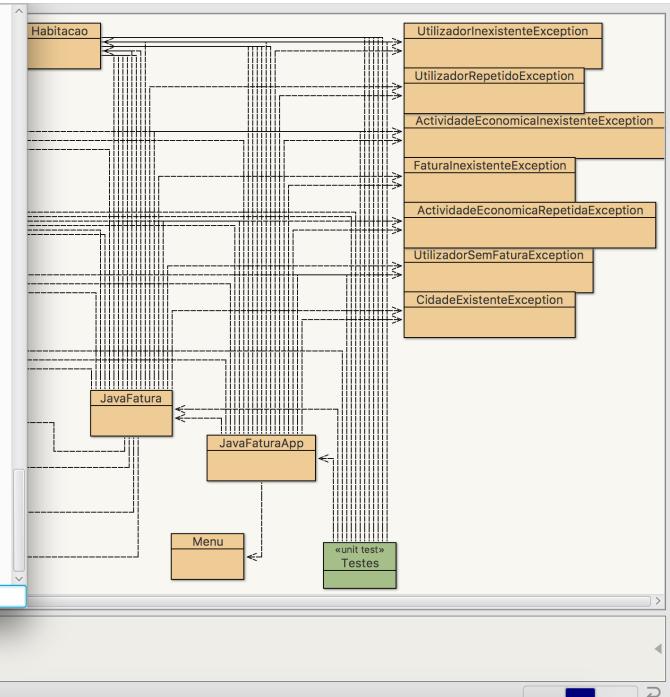


Figura 23: Cria faturas associada a um contribuinte individual

3.1.4 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

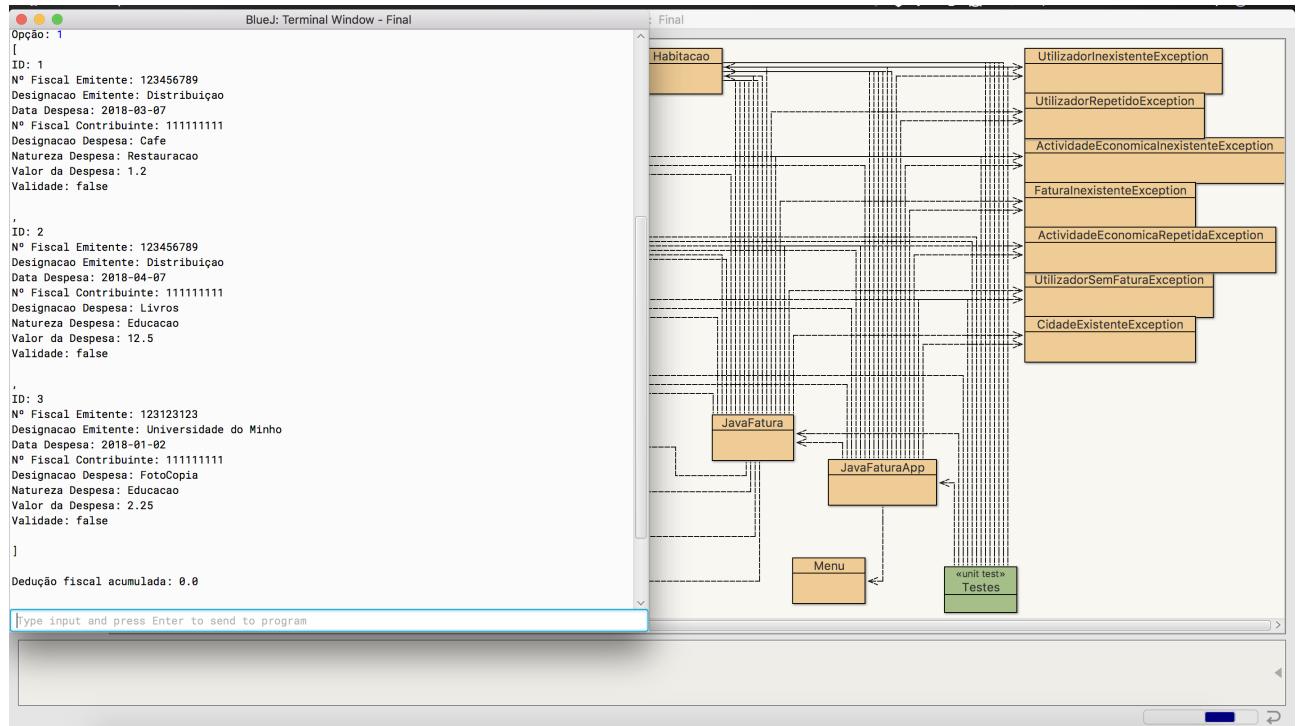


Figura 24: Verifica Faturas

3.1.5 Corrigir a classificação de actividade económica de um documento de despesa

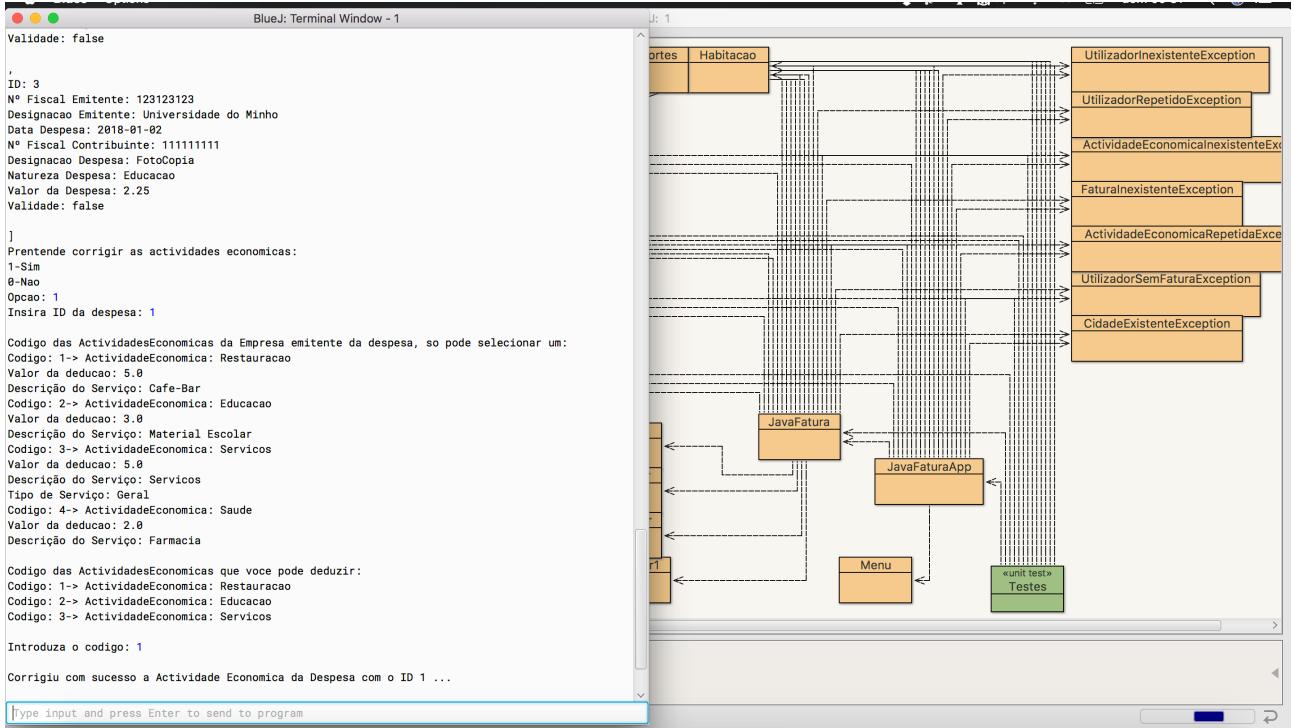


Figura 25: Corrige uma despesa

3.1.6 Obter a listagem das facturas de uma determinada empresa, ordenada por data de emissão ou por valor

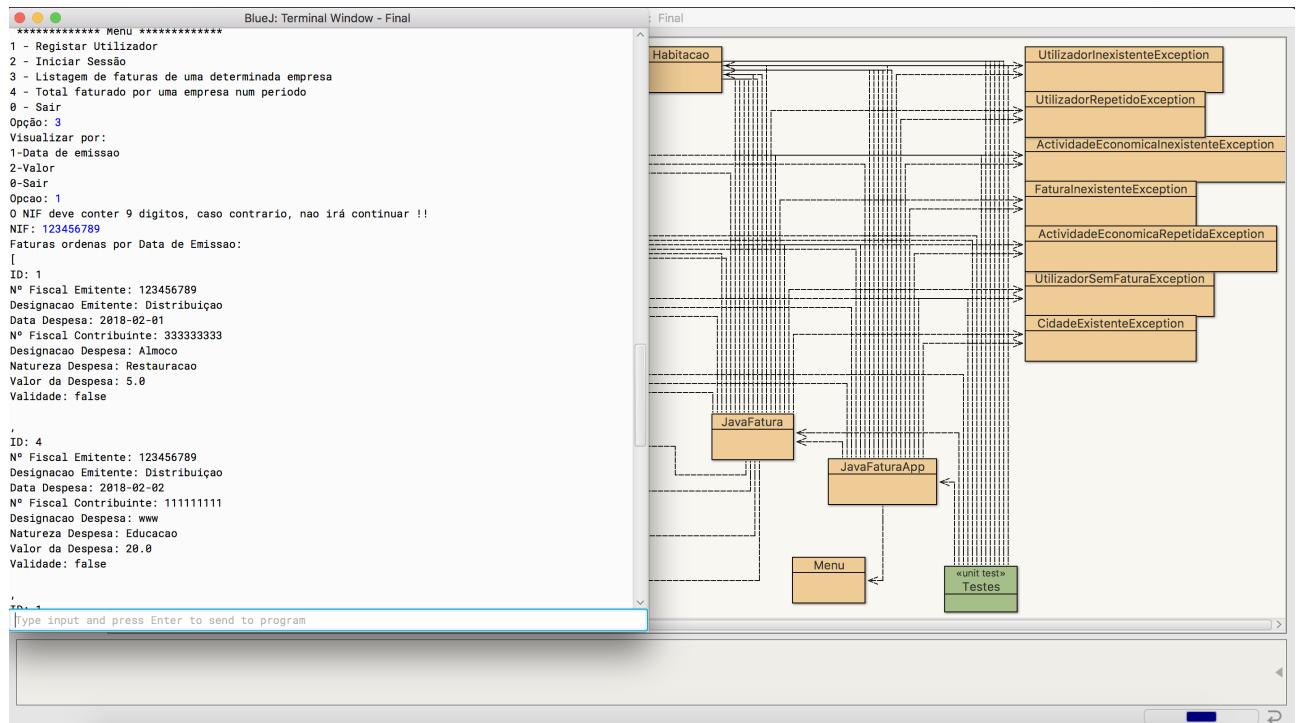


Figura 26: Lista de faturas por data de emissão

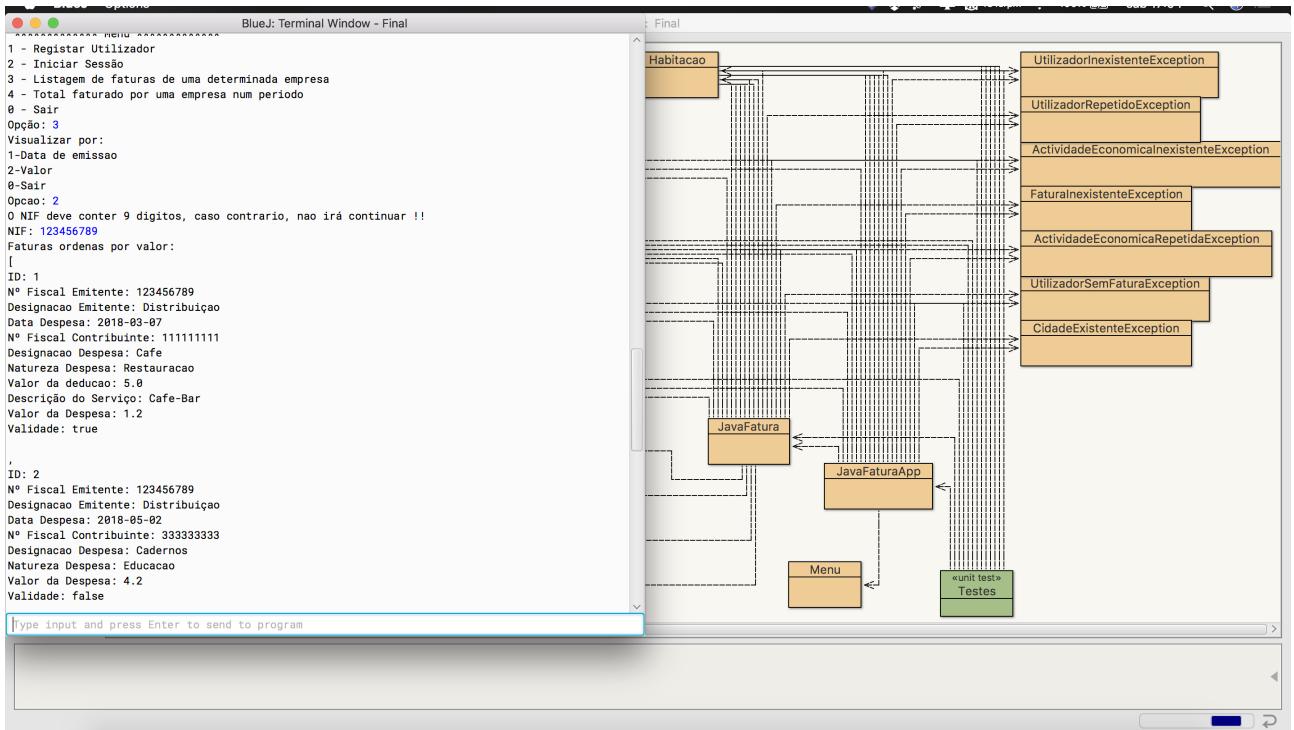


Figura 27: Lista de faturas por valor

3.1.7 Obter por parte das empresas, as listagens das facturas por contribuinte num determinado intervalo de datas

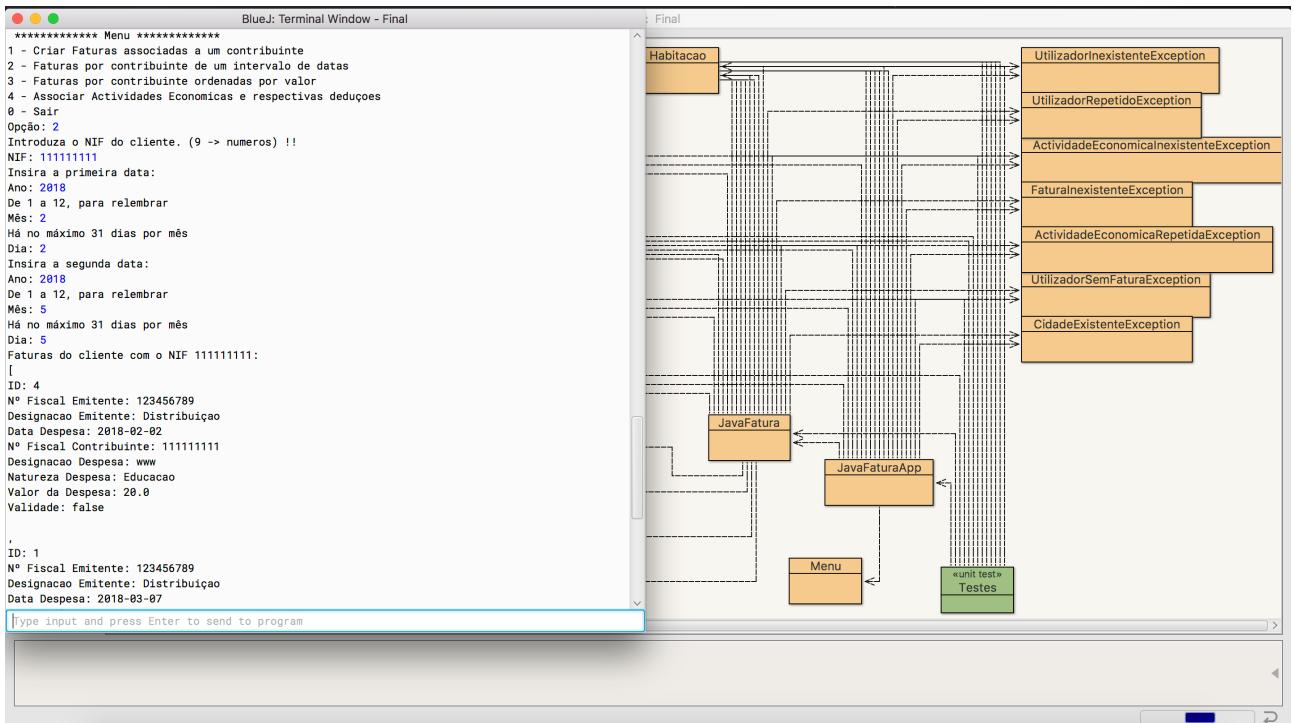


Figura 28: Lista de faturas de um intervalo de datas

3.1.8 Obter por parte das empresas, as listagens das facturas por contribuinte ordenadas por valor decrescente de despesa

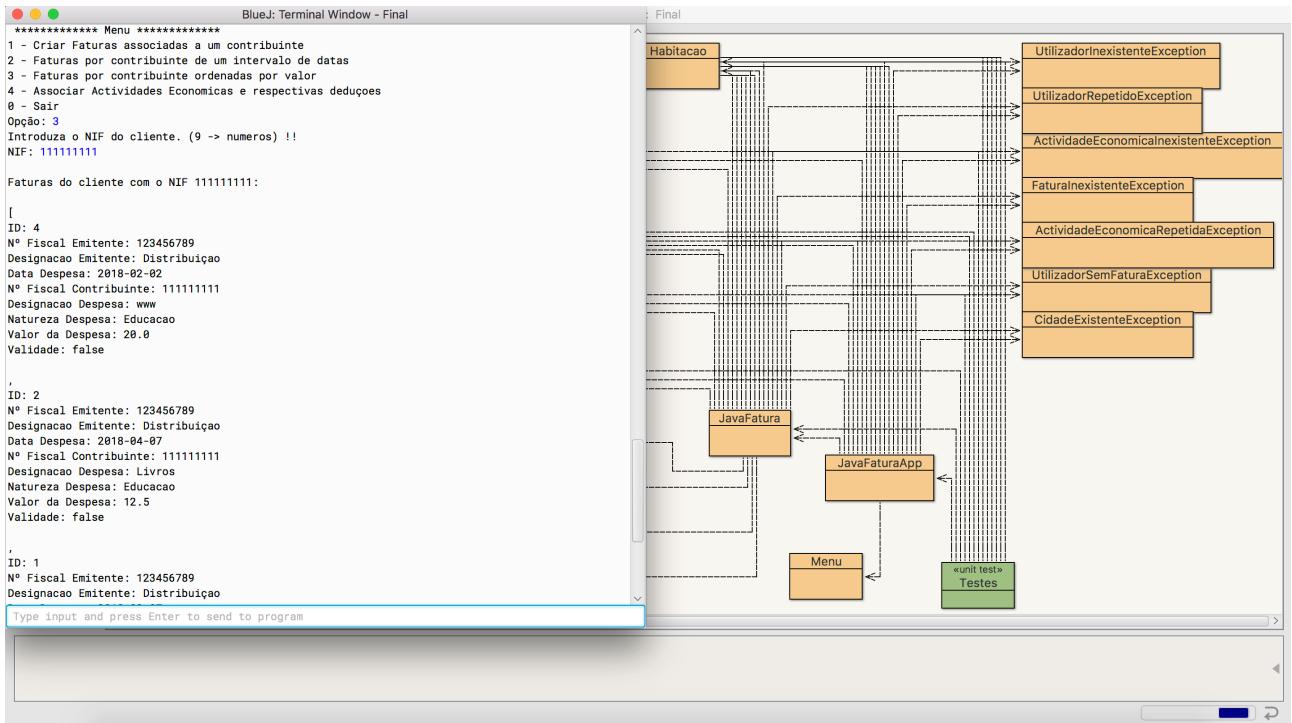


Figura 29: Lista de faturas por valor

3.1.9 Indicar o total facturado por uma empresa num determinado período

```

Designacao Despesa: Comprimidos
Natureza Despesa: Saude
Valor da Despesa: 22.0
Validade: false

]

***** Menu *****
1 - Registar Utilizador
2 - Iniciar Sessão
3 - Listagem de faturas de uma determinada empresa
4 - Total faturado por uma empresa num período
0 - Sair
Opção: 4
Introduza o NIF da Empresa. (9 -> numeros) !!
NIF: 123456789
Insira a primeira data:
Ano: 2018
De 1 a 12, para relembrar
Mês: 1
Há no máximo 31 dias por mês
Dia: 1
Insira a segunda data:
Ano: 2018
De 1 a 12, para relembrar
Mês: 5
Há no máximo 31 dias por mês
Dia: 5

Total faturado da empresa -> 123456789 <- entre as datas 2018-01-01-->2018-05-05: 64.9€

***** Menu *****
1 - Registar Utilizador
2 - Iniciar Sessão
3 - Listagem de faturas de uma determinada empresa
4 - Total faturado por uma empresa num período
0 - Sair
Opção:

```

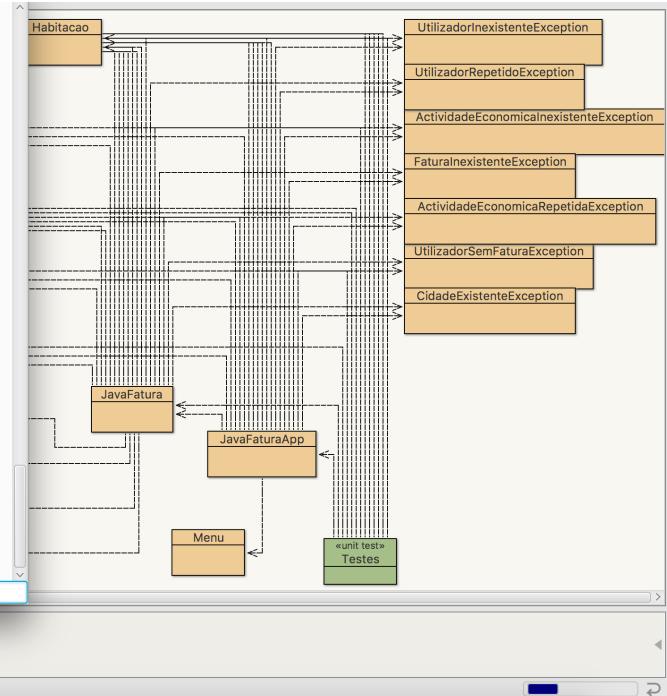


Figura 30: Valor total faturado por uma empresa num período de tempo

3.1.10 Determinar a relação dos 10 contribuintes que mais gastam em todo o sistema

```

***** Menu *****
1 - Os dez contribuintes com mais gastos
2 - Relação de X empresas mais fatura
3 - Adiciona cidade com incentivo fiscal
4 - Sair
Opção: 1
10 contribuinte que mais gastam na aplicacao:

Contribuinte: 1111111111
Valor Gasto: 35.95

Contribuinte: 4444444444
Valor Gasto: 1.0

Contribuinte: 5555555555
Valor Gasto: 125.0

Contribuinte: 2222222222
Valor Gasto: 43.0

Contribuinte: 3333333333
Valor Gasto: 29.43

***** Menu *****
1 - Os dez contribuintes com mais gastos
2 - Relação de X empresas mais fatura
3 - Adiciona cidade com incentivo fiscal
4 - Sair
Opção:

```

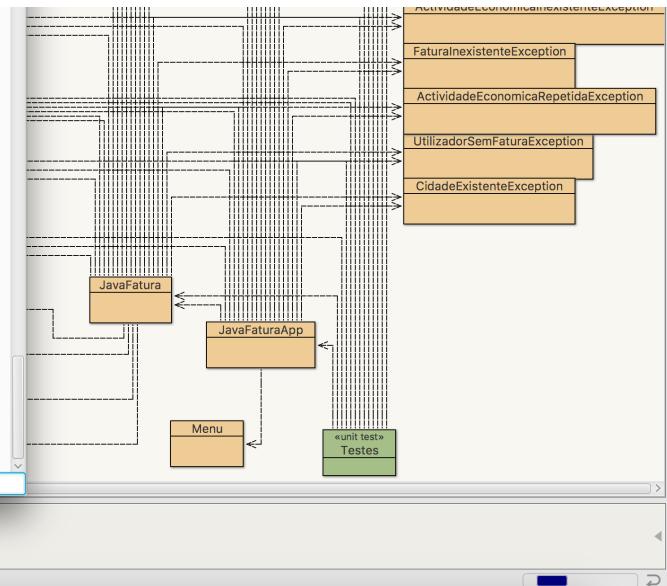


Figura 31: 10 contribuintes que mais gastam da aplicação

3.1.11 Determinar a relação das X empresas que mais facturas tem em todo o sistema e o montante de deduções fiscais que as despesas registadas representam

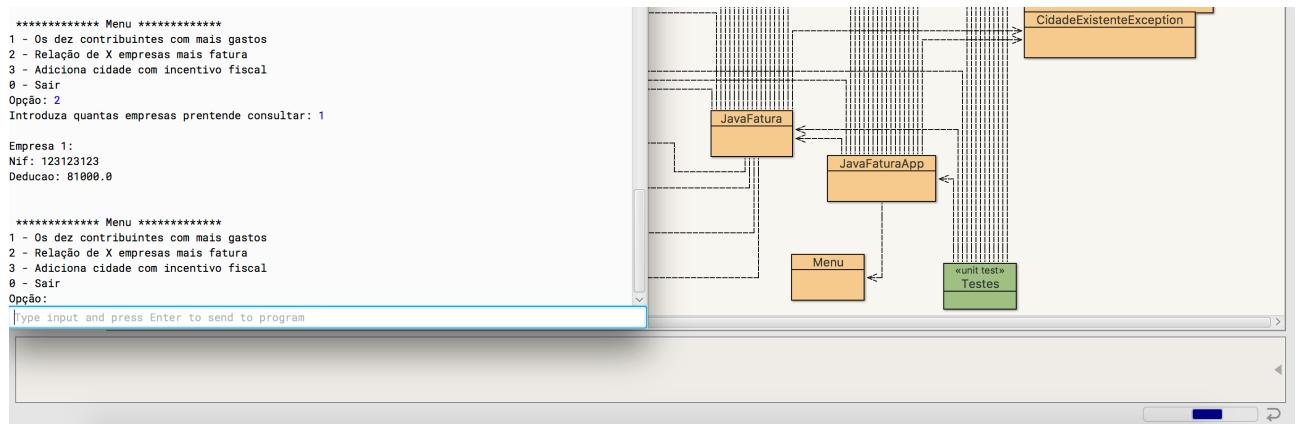


Figura 32: Relação de X empresas com mais faturas

4 Conclusão

Fazendo uma avaliação global da realização deste projeto, deparamo-nos com algumas dificuldades, mas criamos um projeto, na nossa opinião, fluído e bem pensado, passivo de uma boa compreensão por parte de elementos externos à realização do mesmo.

A principal dificuldade com que nos deparamos foi a necessidade de popular a base de dados da aplicação e ainda a implementação das funcionalidades extra relacionadas embora, no nosso ponto de vista, tenhamos cumprido com sucesso os objetivos deste trabalho prático elaborando os requisitos básicos solicitados bem como os avançados.