

Relatório de Projeto

Introdução à Programação em Java

Discentes:

Anna Helena Drumond – 2021151911

Joana Pais Valente – 2021152666

Docentes:

Vítor Graveto

Maria José Marcelino

ÍNDICE

CAPÍTULO 1	5
COMPARAÇÃO ENTRE O PRIMEIRO DIAGRAMA DE CLASSES E O DIAGRAMA DE CLASSES FINAL	5
1.1 DIAGRAMA DE CLASSES INICIAL	5
1.2 DAS ALTERAÇÕES REALIZADAS NO DIAGRAMA DE CLASSES INICIAL.	6
1.3 DIAGRAMA DE CLASSES FINAL E ESTRUTURA DO PROJETO.....	8
CAPÍTULO 2	9
ESCLARECIMENTOS SOBRE AS REGRAS DE NEGÓCIO DO PROJETO.....	9
2.1 FUNCIONALIDADES DISPONÍVEIS AO INTERMEDIÁRIO.....	9
2.2 FUNCIONALIDADES DISPONÍVEIS AO INTERMEDIÁRIO FUNDADOR:.....	9
2.3 FUNCIONALIDADES DISPONÍVEIS AOS PUBLICITÁRIOS:.....	10
2.3 FUNCIONALIDADES DISPONÍVEIS AOS PROGRAMADORES:	10
2.4 REGRAS DE NEGÓCIO PARA CONTRATAÇÃO DE ANÚNCIOS:.....	11
CAPÍTULO 3	11
SOBRE A ESTRUTURA DO PROGRAMA.	11
3.1 CLASSE DETENTORA DO MÉTODO MAIN.	11
3.2 CLASSE GESTORA EMPRESAAORWORDS:.....	12
3.3 CLASSE UTILIZADOR:	18
3.4 CLASSE PUBLICITÁRIO:	19
3.5 CLASSE PROGRAMADOR:.....	19
3.6 CLASSE INTERMEDIÁRIO:	20
3.7 CLASSE FRASE:	20
3.9 CLASSE APLICAÇÃO:	21
3.10 CLASSE ANÚNCIO:.....	21
3.11 CLASSE CONTA:	22
3.12 CLASSE MOVIMENTOFINANCEIRO:.....	23
3.13 ENUMERADOR TIPODEMOVIMENTO:	24
CAPÍTULO 4	24
ALGUMAS FUNÇÕES BÁSICAS DO SISTEMA.....	24
4.1 SOBRE A LEITURA E ATUALIZAÇÃO DO FICHEIRO DE CONFIGURAÇÃO:	24
4.2 CLASSE QUE IMPLEMENTA OS COMANDOS DE LEITURA E ESCRITA DOS FICHEIROS DE TEXTO:	26
4.3 SOBRE A LEITURA E ESCRITA NO FICHEIRO DE OBJETOS:	27
4.4 CLASSE QUE IMPLEMENTA OS COMANDOS DE LEITURA E ESCRITA DO FICHEIRO DE OBJETOS:.....	28
4.5 CONTROLE DO UTILIZADOR ATIVO:.....	29
4.6 SOBRE A LÓGICA IMPLEMENTADA PARA A TROCA DE IDIOMAS DO SISTEMA:	29
4.6.1 Classe TrocaDeIdioma:.....	34
CAPÍTULO 5.	34
ABORDAGEM PADRÃO USADA PARA A CONSTRUÇÃO DE TODAS AS CLASSES DA INTERFACE GRÁFICA:	34
5.1 JFRAME ÚNICA UTILIZADA EM TODO O PROGRAMA	35
5.2 ESTRUTURA COMUM A TODAS AS CLASSES DA INTERFACE GRÁFICA	35
5.3 DINÂMICA NECESSÁRIA NA INTERFACE GRÁFICA PARA A TROCA ENTRE IDIOMAS EM TEMPO DE EXECUÇÃO	37
5.4 MIGRAÇÃO ENTRE ECRÃS	38

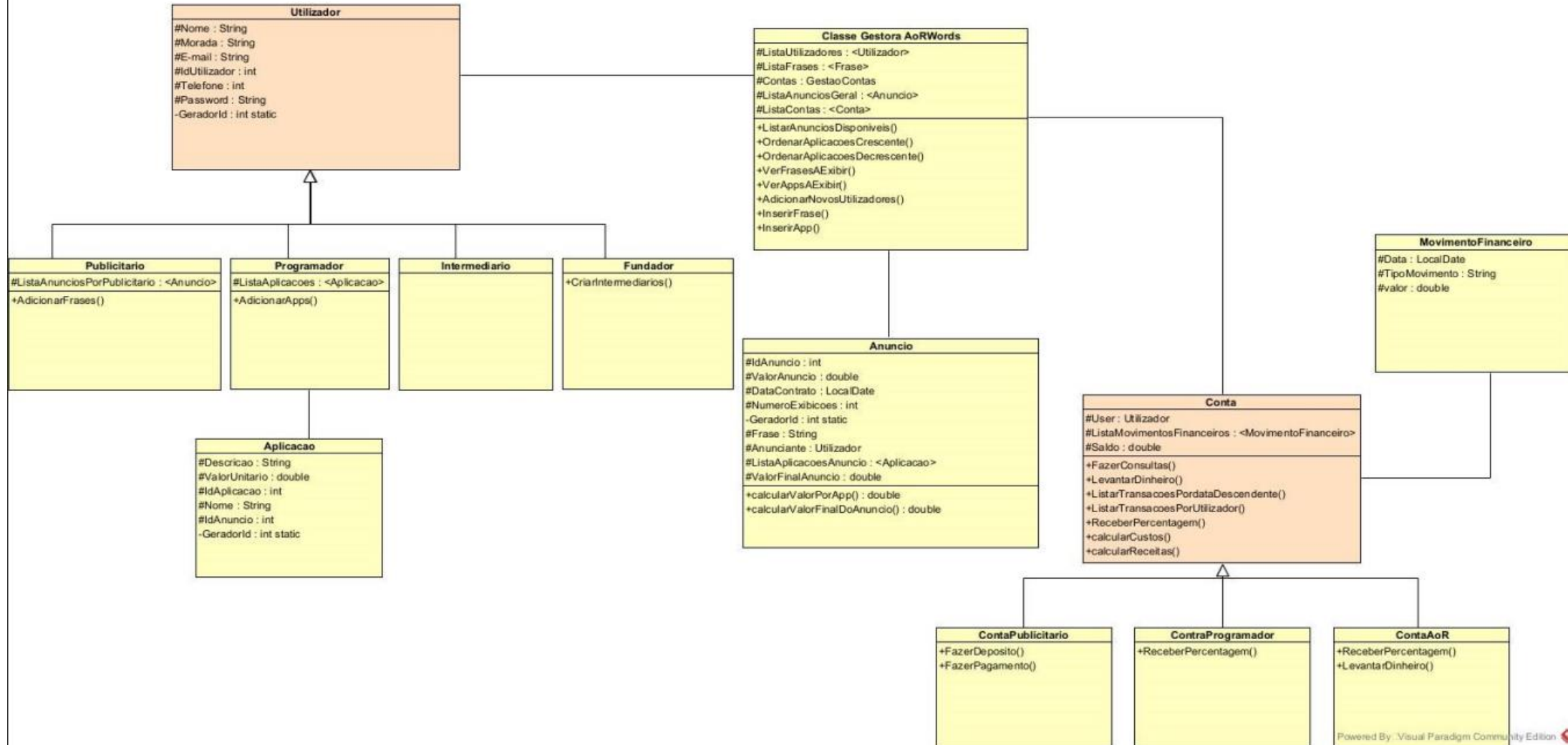
5.5 CLASSE FRAMEUNICA:	38
5.6 CLASSE AVISOSPOPUP:	40
5.7 CLASSE VALIDADORGERAL:	40
CAPÍTULO 6.	41
DA IMPLEMENTAÇÃO DE CADA ECRÃ DE INTERFACE COM O UTILIZADOR:	41
6.1 CLASSE ECRAINICIAL:	41
6.2 CLASSE ECRASignUp:	41
6.3 CLASSE ECRAADDUTILIZADOR:	42
6.4 CLASSE ECRACONSULTARUTILIZADORES:	43
6.4.1 Sobre a construção padrão do componente JTable:	44
6.5 CLASSE ECRALOGIN:	47
6.6 CLASSES QUE CORRESPONDEM AO MENU DE OPÇÕES DISPONÍVEL PARA CADA TIPO DE UTILIZADOR:	47
6.7 CLASSE ECRACONFIGURACOES	48
6.8 CLASSE ECRAADDFRASE	50
6.9 CLASSE ECRAADDAPLICACAO	51
6.10 CLASSE ECRAEDITARFRASES	52
6.10.1 Do método <i>capturarAppSelecionadaNaJtable()</i> :	52
6.11 CLASSE ECRAEDITARAPLICACOES	54
6.12 CLASSE ECRAADDANUNCIO	56
6.13 CLASSE ECRAESCOLHERAPPCONSULTA	62
6.14 CLASSE ECRAESCOLHERFRASECONSULTA	64
6.15 CLASSE ECRAVERFRASESPOREXIBIR	64
6.16 CLASSE ECRAVERAPPSONDEEXIBIDAS	65
6.17 CLASSE ECRACONTRATOSEMPRESA	65
6.18 CLASSE ECRACONTRATOSPROGRAMADOR	68
6.19 CLASSE ECRACONTRATOSPUBLICITARIO	68
6.20 CLASSE ECRACONTA	68
6.20.1 Dos componentes inseridos a JPanel para o ecrã conta do intermediário fundador	70
6.20.2 Dos componentes inseridos a JPanel para o ecrã conta dos demais intermediários	71
6.20.3 Dos componentes inseridos a JPanel para o ecrã conta dos programadores	71
6.20.4 Dos componentes inseridos a JPanel para o ecrã conta dos publicitários.....	71
6.21 CLASSE ECRAMOVIMENTOSFINANCEIROS	72
6.22 CLASSE ECRAMOVIMENTOSUSERS	75
CAPÍTULO 7	79
BIBLIOGRAFIA UTILIZADA	79
CAPÍTULO 8	79
ESQUEMAS DE ECRÃ	79
8.1 ESQUEMA DE ECRÃ DO INTERMEDIÁRIO	80
8.2 ESQUEMA DE ECRÃ DO PUBLICITÁRIO	81
8.3 ESQUEMA DE ECRÃ DO PROGRAMADOR	82
CAPÍTULO 9	83
MANUAL DE INSTRUÇÕES DO PROGRAMA	83
1. ECRÃS COMUNS A TODOS OS UTILIZADORES	83
1.1 Ecrã Inicial	83
1.2 Ecrã de Sign-up	84
1.3 Ecrã Login	84
2. AORWORDS PARA PUBLICITÁRIOS	85

2.1	<i>Ecrã adicionar nova frase</i>	86
2.2	<i>Ecrã Conta Corrente</i>	86
2.3	<i>Ecrã Editar as suas Frases</i>	90
2.4	<i>Ecrã adicionar Anúncio</i>	91
2.5	<i>Ecrã dos contratos estabelecidos</i>	92
2.6	<i>Ecrã configurações</i>	93
2.7	<i>Ecrã ver as aplicações em que as frases vão ser exibidas</i>	95
3.	AORWORDS PARA PROGRAMADORES	96
3.1	<i>Ecrã adicionar uma nova aplicação</i>	96
3.2	<i>Ecrã Conta Corrente</i>	97
3.3	<i>Ecrã editar aplicação</i>	99
3.4	<i>Ecrã das frases a exibir</i>	100
3.5	<i>Ecrã dos contratos estabelecidos</i>	101
3.6	<i>Ecrã configurações</i>	102
4.	AORWORDS PARA INTERMEDIÁRIOS	104
4.1	<i>Ecrã adicionar nova frase</i>	104
4.2	<i>Ecrã Conta Corrente</i>	105
4.3	<i>Ecrã adicionar um novo utilizador</i>	109
4.4	<i>Ecrã adicionar uma nova aplicação</i>	110
4.5	<i>Ecrã dos contratos estabelecidos</i>	111
4.6	<i>Ecrã configurações</i>	112
4.7	<i>Ecrã adicionar novo anúncio</i>	113
4.8	<i>Ecrã visualizar utilizadores registados</i>	115
CAPÍTULO 10		116
LISTAGEM DO PROGRAMA		116
10.1	CLASSE ANUNCIO.....	116
10.2	CLASSE APLICACAO	118
10.3	CLASSE CONTA	120
10.4	CLASSE EMPRESAAORWORDS	122
10.5	CLASSE FRASE.....	130
10.6	CLASSE INTERMEDIÁRIO	130
10.7	CLASSE MOVIMENTOFINANCEIRO	131
10.8	CLASSE PROGRAM (MAIN)	132
10.9	CLASSE PROGRAMADOR.....	132
10.10	CLASSE PUBLICITARIO	133
10.11	ENUMERADOR TIPODEMOVIMENTO.....	134
10.12	CLASSE ABSTRATA UTILIZADOR	135

CAPÍTULO 1

Comparação entre o primeiro Diagrama de Classes e o Diagrama de Classes Final

1.1 Diagrama de Classes Inicial



1.2 Das alterações realizadas no Diagrama de Classes inicial.

Nas primeiras análises do projeto, foi criado um primeiro diagrama de classes ainda rudimentar e sem considerar todos os pormenores necessários para perfeita implementação de todas as funcionalidades.

Após brainstorm e esclarecimentos de dúvidas, foram efetuadas algumas mudanças que consideramos necessárias relativamente à estrutura do diagrama:

1. Foram excluídas as subclasses da classe Conta, uma vez que estas subclasses somente iriam guardar cálculos referentes à criação de anúncios. Assim, estes métodos ficaram distribuídos entre a classe Conta e a classe EmpresaAorWords de acordo com o que seria mais conveniente com as regras de negócio do projeto.
2. A classe Publicitário deixou de ter um ArrayList de anúncios e passou a ter um ArrayList das frases que cada publicitário poderia registrar/editar/publicitar em sua conta, pois foi verificado que não havia necessidade de duplicar as informações dos anúncios, bastando a classe EmpresaAorWords ter uma lista com todos os anúncios criados.
3. Havia a necessidade de uma classe Frase, a fim de permitir a ArrayList citada acima.
4. Foram excluídos todos os atributos definidos como ID, sendo mantido apenas o atributo idUtilizador que constituía um dos requisitos do projeto, por se ter verificado que os restantes eram desnecessários.
5. A classe Gestora EmpresaAorWords recebeu diversos novos atributos para:
 - Armazenar os dados lidos do ficheiro de configuração.
 - Armazenar os dados lidos do ficheiro com os idiomas em língua portuguesa e inglesa.
 - Armazenar os dados lidos do ficheiro de objetos.
 - Armazenar a instância do utilizador ativo no programa.
 - Instanciar um objeto que permite ler e escrever ficheiros de objetos.
 - Instanciar um objeto que permite ler e escrever ficheiros de texto.
 - Instanciar e criar a conta da empresa administrada pelo fundador.
 - Armazenar a instância do ecrã/classe ativo e a instância do JPanel ativo, cruciais na lógica implementada quanto a troca entre idiomas em tempo de execução.
6. No decorrer do projeto, acabaram por ser inseridos novos métodos que permitiam uma maior usabilidade e satisfação do programa por parte dos utilizadores, e outros que permitiam uma maior segurança quanto ao controlo na inserção de dados pelos utilizadores. Além disso, alguns dos métodos que constavam no diagrama inicial não foram constam na versão final do mesmo uma vez que se afiguravam desnecessários.
7. Foi ainda constatado que o projeto poderia beneficiar de um Enumerador, denominado TipoDeMovimento, que armazenaria os tipos de movimentos financeiros, uma vez que são uma lista de valores pré-definidos e imutáveis.¹
8. A classe Anuncio recebeu alguns atributos para além dos acima representados de forma a armazenar outras informações relativas ao anúncio, necessárias em outros locais do programa.

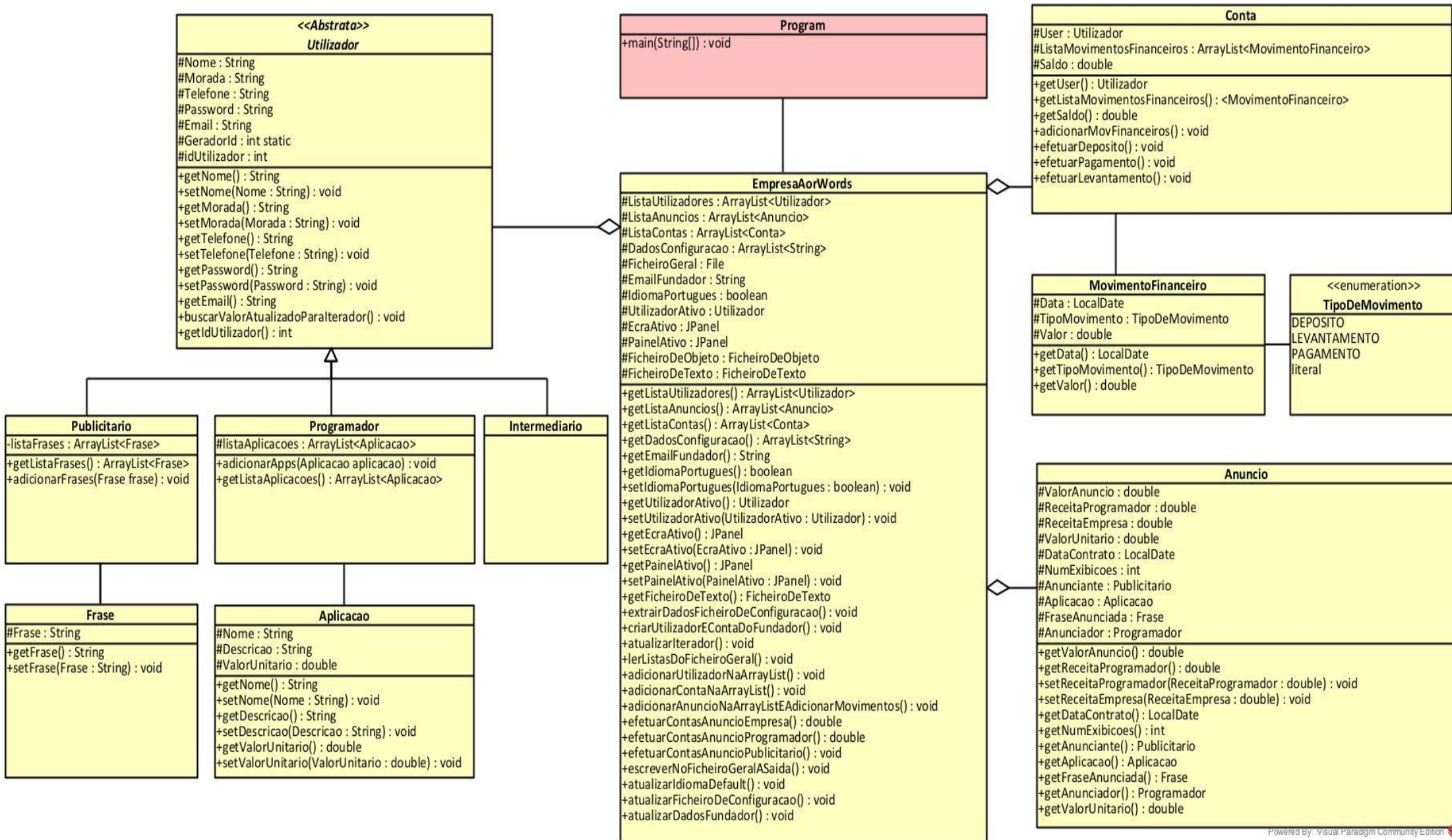
¹ <https://www.devmedia.com.br/tipos-enum-no-java/25729> (acedido em 20/12/2021).

9. Foi constatado que a classe Utilizador poderia ser definida como uma classe abstrata, uma vez que esta classe não seria instanciada em momento algum, sendo instanciadas somente as suas subclasses.²
10. Foi ainda observado que alguns objetos usados como atributos, referentes aos componentes da GUI, como JPanel, JButtons etc, deveriam ter os modificadores de acesso do tipo:
- *Private*: pois somente havia a necessidade de serem acedidos dentro da própria classe, e assim, fechamos o seu acesso em relação às restantes classes, garantindo a sua proteção.³
 - *Final*: garante segurança e uma implementação única do componente na interface gráfica. Como estes componentes que recebem o modificador final são componentes da GUI (sendo, por isso, objetos), permite alterar o conteúdo atribuído a este objeto, mas simultaneamente impede que estes objetos sejam novamente instanciados. Esta capacidade de modificar o conteúdo do objeto é fulcral para poder implementar a transição entre idiomas durante a execução do programa. A limitação do número de instâncias que se podem criar daquele objeto aumenta a sua segurança pois previne que o mesmo seja instanciado com conteúdo não desejado.

² <https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html> (acedido em 22/12/2021)

³ <https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html> (acedido em 26/12/2021)

1.3 Diagrama de classes final e estrutura do projeto



CAPÍTULO 2

Esclarecimentos sobre as Regras de Negócio do Projeto

O sistema possui três tipos de utilizadores:

1. Intermediários,
 - a) Intermediário comum,
 - b) Intermediário fundador - este atua como o presidente da empresa.
2. Publicitários,
3. Programadores.

Os dados de acesso do Intermediário fundador são:

E-mail de acesso: vitor@pt.com

Password: 12345

2.1 Funcionalidades disponíveis ao Intermediário

Todos os intermediários (comuns e o fundador) podem usufruir das seguintes funcionalidades:

1. Criar novos utilizadores dos tipos 1, 2 e 3 a) definidos acima.
2. Visualizar e ordenar todas as informações de todos os utilizadores, com exceção da palavra-passe.
3. Consultar e/ou editar seus dados pessoais.
4. Alterar sua palavra-passe.
5. Visualizar todas as informações de todos os contratos de anúncios de todos os utilizadores, nomeadamente o nome do programador dono da aplicação e o nome do publicitário que contratou o anúncio, o que permite controlar os anúncios criados, bem como controlar as frases a serem exibidas e a aplicação onde serão exibidas. A tabela com todos estes dados vai permitir ainda ordenar os valores de qualquer coluna e filtrar os contratos de anúncio por data de contratação.
6. Adicionar novas frases e criar novos anúncios em nome de um publicitário.
7. Adicionar novas aplicações em nome de um programador.
8. Aceder à conta da empresa e visualizar seu saldo.
9. Visualizar todos os movimentos financeiros da empresa e filtrar estes movimentos por data e/ou ordenar qualquer das informações.
10. Visualizar todos os movimentos financeiros de todos os utilizadores e filtrar estas informações por data e/ou utilizador, assim como ordenar qualquer uma das informações.

2.2 Funcionalidades disponíveis ao intermediário fundador:

O intermediário fundador, para além das funcionalidades supramencionadas, pode ainda efetuar levantamento de valores da conta da empresa.

2.3 Funcionalidades disponíveis aos publicitários:

Primeiramente, vale a pena lembrar que cada publicitário possui um ArrayList com as suas frases.

1. Adicionar anúncios, desde que tenha saldo suficiente para o fazer.
2. Fazer um orçamento do anúncio, visualizando o valor do mesmo, antes de fazer a efetiva contratação do serviço.
3. Criar um novo anúncio caso disponha de saldo suficiente e concorde com o preço apresentado no orçamento referido no número anterior.
4. Registrar novas frases.
5. Editar frases suas que já foram anteriormente registadas.
6. Consultar e/ou editar seus dados pessoais.
7. Modificar sua palavra-chave, tal que é fundamental para garantir a segurança da sua conta, isto porque a sua conta é criada por um intermediário, e, esta alteração impede que estes tenham posteriormente acesso à sua conta.
8. Eliminar permanentemente sua conta dos serviços da empresa.
9. Verificar as aplicações onde uma determinada frase será exibida e o custo atual dos anúncios para esta frase.
10. Visualizar as informações de todos os anúncios contratados por si, bem como ordenar qualquer uma destas informações.
11. Aceder à sua conta e visualizar o seu saldo.
12. Efetuar levantamentos e depósitos.
13. Visualizar todos os seus movimentos financeiros, filtrar estes movimentos por data e/ou ordenar as informações dos movimentos.

2.3 Funcionalidades disponíveis aos programadores:

Primeiramente, cabe informar que cada programador possui um ArrayList de suas aplicações.

1. Registrar novas aplicações.
2. Editar aplicações suas, já registadas anteriormente.
3. Consultar e/ou editar seus dados pessoais.
4. Modificar sua palavra-chave, tal que é fundamental para garantir a segurança da sua conta, isto porque a sua conta é criada por um intermediário, e, esta alteração impede que estes tenham posteriormente acesso à sua conta.
5. Excluir permanentemente sua conta dos serviços da empresa.
6. Verificar as frases que devem ser exibidas numa das suas aplicações e a receita atual dos anúncios contratados para esta aplicação.
7. Visualizar as informações de todos os anúncios contratados para suas aplicações, bem como ordenar qualquer uma destas informações.
8. Aceder à sua conta e visualizar o seu saldo.
9. Efetuar levantamentos.
10. Visualizar todos os seus movimentos financeiros, filtrar estes movimentos por data e/ou ordenar as informações dos movimentos.

2.4 Regras de negócio para contratação de anúncios:

1. Uma frase pode ser publicitada em diversas aplicações, ou seja, a mesma frase pode aparecer em várias aplicações.
2. Uma aplicação pode anunciar várias frases.
3. Cada contrato de anúncio contém uma aplicação escolhida e uma frase a ser anunciada. O publicitário, aquando da formulação do contrato, pode escolher quantas vezes pretende publicitar esta frase na aplicação escolhida.
4. Caso o publicitário deseje anunciar a mesma frase em outras aplicações, deverá criar um novo contrato de anúncio para a mesma frase.
5. O anúncio só pode ser criado se o publicitário cuja frase vai ser anunciada possuir saldo suficiente na sua conta corrente para poder cobrir os custos do anúncio.
6. Por cada anúncio que é criado, o publicitário cuja aplicação foi contratada fica com 70% do valor do anúncio, e a empresa AoRWords fica com os restantes 30%.

CAPÍTULO 3

Sobre a estrutura do Programa.

3.1 Classe detentora do método main.

A classe que contém o método main, ficou denominada como Program.

```
public class Program {  
    public static void main(String[] args) {  
  
        EmpresaAorWords empresaObjeto = new EmpresaAorWords();  
  
        FrameUnica frameUnica = new FrameUnica(empresaObjeto, empresaObjeto.getFicheiroDeTexto());  
    }  
}
```

Possui apenas duas linhas de código:

1. Uma que cria e instancia um objeto do tipo EmpresaAorWords, que é a única instância da classe gestora a ser usada em todo o programa. Permite que o programa, ao arrancar, realize suas instruções iniciais, tais como buscar os dados no ficheiro de configurações e, caso exista, ler os dados do ficheiro de objetos.

2. Uma segunda linha que cria e instancia um objeto do tipo FrameUnica, que faz o arranque da interface gráfica e permite ao programa realizar as instruções iniciais da GUI, tais como carregar o primeiro ecrã de interface com o utilizador e buscar os dados ao ficheiro de Idioma

3.2 Classe gestora EmpresaAorWords:

A classe EmpresaAorWords é uma das classes de maior importância do programa, uma vez que nela estão implementados:

1. A lógica que cria e instancia as listas de utilizadores, contas e anúncios, que cria e instancia a lista dos dados de configuração para ser preenchida através do método extrairDadosFicheiroDeConfiguracao(), e, caso o ficheiro de objetos exista, preencher as restantes listas enumeradas com o método lerListasDoFicheiroGeral(), conforme demonstra o construtor:

```
public EmpresaAorWords() {  
    listaContas = new ArrayList<>{10};  
    listaUtilizadores = new ArrayList<>{10};  
    listaAnuncios = new ArrayList<>{10};  
    dadosConfiguracao = new ArrayList<>{10};  
    extrairDadosFicheiroDeConfiguracao();  
    if (ficheiroGeral.exists()) {  
        lerListasDoFicheiroGeral();  
    }  
}
```

2. Os métodos que recebem os dados dos utilizadores, contas e anúncios como parâmetro e adicionam na ArrayList respetiva.

2.1 Vale a pena referir que o método adicionarAnuncioNaArrayListEAdicionarMovimentos(), para além de adicionar os anúncios na respetiva ArrayList, também chama os métodos efetuarContasAnuncioProgramador() e efetuarContasAnuncioPublicitario(), ambos presentes nesta mesma classe e detalhados nos itens 14.1 a 14.3 desta unidade 3.2 do capítulo 3. Isto garante que, sempre que é criado um contrato de anúncio, são simultaneamente efetuados e registados todos os movimentos financeiros na conta corrente de todos os envolvidos neste contrato.

Quanto à efetiva criação dos contratos de anúncios, cabe informar que esta parte está descrita na unidade 6.12 do capítulo 6 deste relatório, que trata sobre a classe de interface com o usuário responsável pela criação dos anúncios.

```
public void adicionarAnuncioNaArrayListEAdicionarMovimentos(Anuncio anuncio, Programador programador, Publicitario publicitario, double custoTotal) {  
  
    if (anuncio != null) {  
  
        double receitaEmpresa = efetuarContasAnuncioEmpresa(custoTotal);  
  
        double receitaProgramador = 0.0;  
  
        for (Conta conta : listaContas) {  
            if (conta.getUser().equals(programador)) {  
  
                receitaProgramador = efetuarContasAnuncioProgramador(custoTotal, conta);  

```

```

    } else if (conta.getUser().equals(publicitario)) {

        efetuarContasAnuncioPublicitario(custoTotal, conta);
    }
}
anuncio.setReceitaEmpresa(receitaEmpresa);
anuncio.setReceitaProgramador(receitaProgramador);
listaAnuncios.add(anuncio);
}
}

```

3. O método que realiza leitura do ficheiro de configuração.

```

public void extrairDadosFicheiroDeConfiguracao() {

    try {
        ficheiroDeTexto.abrirLeitura("FicheiroDeConfiguracao.dat");
        String auxiliarDeLinha;

        while ((auxiliarDeLinha = ficheiroDeTexto.lerLinha()) != null) {
            dadosConfiguracao.add(auxiliarDeLinha);
        }
        ficheiroDeTexto.fecharLeitura();
    } catch (IOException erro) {
        erro.printStackTrace();
    }
    emailFundador = dadosConfiguracao.get(3);
    ficheiroGeral = new File(dadosConfiguracao.get(5));
    idiomaPortugues = Boolean.parseBoolean(dadosConfiguracao.get(8));

    criarUtilizadorEContaDoFundador();
}

```

4. O método que realiza a escrita e atualização de dados do ficheiro de configuração.

```

public void atualizarFicheiroDeConfiguracao() {
    atualizarIdiomaDefault();
    try {
        ficheiroDeTexto.abrirEscrita(dadosConfiguracao.get(7));
        for (String elemento : dadosConfiguracao) {
            ficheiroDeTexto.escreverLinha(elemento);
        }
        ficheiroDeTexto.fecharEscrita();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

Para melhor entendimento de como decorre a lógica para a correta leitura e escrita dos dados constantes no ficheiro de configuração, ver unidade 4.1 capítulo 4 deste relatório.

5. O método que realiza a leitura do ficheiro de objetos denominado por FicheiroDeObjetosGeral.dat.

```

        public void lerListasDoFicheiroGeral() {

            try {
                ficheiroDeObjeto.abreLeitura(ficheiroGeral);

                listaUtilizadores = (ArrayList<Utilizador>) ficheiroDeObjeto.leObjeto();
                listaContas = (ArrayList<Conta>) ficheiroDeObjeto.leObjeto();
                listaAnuncios = (ArrayList<Anuncio>) ficheiroDeObjeto.leObjeto();

                ficheiroDeObjeto.fechaLeitura();

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

```

6. O método que realiza a escrita do ficheiro de objetos.

```

        public void escreverNoFicheiroGeralASaida() {

            try {

                ficheiroDeObjeto.abreEscrita(ficheiroGeral);
                ficheiroDeObjeto.escreveObjeto(listaUtilizadores);
                ficheiroDeObjeto.escreveObjeto(listaContas);
                ficheiroDeObjeto.escreveObjeto(listaAnuncios);
                ficheiroDeObjeto.fechaEscrita();

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

```

Para melhor entendimento de como decorre a lógica para a correta escrita dos dados binários no ficheiro de objetos, ver unidade 4.3 capítulo 4 deste relatório.

7. A ArrayList que recebe os dados do ficheiro de configuração.

```

protected ArrayList<String> dadosConfiguracao;

```

8. Alguns atributos que irão receber os valores extraídos do ficheiro de configuração. É de referir que somente foram criados atributos nos casos em que exista a necessidade de existir um método *getter* de forma a aceder ao valor do atributo em várias classes de forma mais organizada e legível. Os demais valores extraídos do ficheiro de configuração, são armazenados em memória apenas na ArrayList citada no item anterior e acedidos através de funcionalidades da própria ArrayList.
9. O método que verifica o último idioma escolhido pelo utilizador e atualiza o idioma constante no ArrayList que contém os dados do ficheiro de configuração, de forma a que registe, aquando o fim da execução do programa, o idioma em que o programa se encontrava quando foi fechado no ficheiro de configurações, conforme demonstrado no item 3 acima.

```

        public void atualizarIdiomaDefault(){
            if (isIdiomaPortugues()) {
                dadosConfiguracao.set(8, String.valueOf(true));
            } else {

```

```

        dadosConfiguracao.set(8, String.valueOf(false));
    }
}

```

10. Os atributos que irão armazenar a instância do ecrã ativo e a instância do JPanel ativo, cruciais na lógica adotada pela equipa do projeto, com relação a troca entre idiomas.

```

protected JPanel ecrãAtivo;
protected JPanel painelAtivo;

```

11. O atributo que guarda a referência ao utilizador que fez login, ou seja, a referência ao utilizador ativo durante toda a execução do programa, que é o que permite correr o programa com informações e ações respetivas ao utilizador correto. Bem como os métodos getter e setter respetivos. É relevante referir que quando o utilizador ativo faz log-out, o utilizador ativo passa para a qualidade de null.

```

protected Utilizador utilizadorAtivo;

```

12. A lógica que cria o intermediário fundador e a conta da empresa administrada pelo fundador. A condição criada no fim serve para circunstâncias em que não exista o ficheiro de objetos, ou seja, caso não exista ficheiro de objetos, a lista de contas encontra-se vazia, pelo que é necessário adicionar a conta da empresa ao ArrayList listaContas.

```

public void criarUtilizadorEContaDoFundador() {
    Utilizador fundador = new Intermediario(dadosConfiguracao.get(0), dadosConfiguracao.get(1),
    dadosConfiguracao.get(2), emailFundador, dadosConfiguracao.get(4));

    Conta contaEmpresa = new Conta(fundador);

    listaUtilizadores.add(fundador);

    if (listaContas.size() == 0) {
        listaContas.add(contaEmpresa);
    }
}

```

13. O método que atualiza o iterador estático que irá constituir o ID de cada utilizador. Este método é chamado sempre que o intermediário cria um novo utilizador (ver unidade 6.3 do Capítulo 6).

```

public void atualizarIterador() {
    int novoValor=listaUtilizadores.get(0).getIdUtilizador();

    for (Utilizador utilizador:listaUtilizadores){
        if (novoValor<=utilizador.getIdUtilizador()){
            novoValor=utilizador.getIdUtilizador()+1;
        }
    }
    Utilizador.buscarValorAtualizadoParaIterador(novoValor);
}

```


14. Ademais, tendo em vista que na regra de negócio do programa, considerou-se que a empresa deveria ser a entidade responsável pelo cálculo das receitas/custos dos anúncios gerados na sua plataforma. Desta forma, a classe EmpresaAorWords contém:

14.1 O método `efetuarContasAnuncioEmpresa()`, que recebe como parâmetro o custo total do anúncio, verifica e localiza a conta da empresa dentro da lista de contas, calcula o valor a ser recebido pela empresa mediante a percentagem que esta recebe por anúncio criado (30%), e chama o método `efetuarDeposito()` declarado na Classe Conta (unidade 3.11 do capítulo 3) que fará o depósito da receita na conta da empresa.

```
public double efetuarContasAnuncioEmpresa(double custoTotal) {

    double percentagemEmpresa = Double.parseDouble(dadosConfiguracao.get(10)); //referência à percentagem da empresa

    double valor = Double.parseDouble(String.valueOf(new BigDecimal(custoTotal * percentagemEmpresa).setScale(2, RoundingMode.UP)));

    for (Conta elemento : listaContas) {

        if (elemento.getUser().getEmail().equals(emailFundador)) // a conta da Empresa é associada ao fundador, que é o seu administrador
            elemento.efetuarDeposito(valor);
        }
    }
    return valor;
}
```

14.2 O método `efetuarContasAnuncioProgramador()`, que recebe como parâmetro o custo total do anúncio e a referência para o programador dono da aplicação que foi contratada pelo anúncio, calcula o valor a ser recebido pelo programador mediante a percentagem (70%) que este recebe por cada anúncio criado com a sua aplicação, e chama o método `efetuarDeposito()` declarado na Classe Conta (unidade 3.11 do capítulo 3) que fará o depósito da receita conta do programador envolvido no anúncio.

```
public double efetuarContasAnuncioProgramador(double custoTotal, Conta contaProgramador) {

    double percentagemProgramador = Double.parseDouble(dadosConfiguracao.get(9)); ///% programador

    double valor = Double.parseDouble(String.valueOf(new BigDecimal(custoTotal * percentagemProgramador).setScale(2, RoundingMode.UP)));

    contaProgramador.efetuarDeposito(valor);
    return valor;
}
```

14.3 O método `efetuarContasAnuncioPublicitario()`, que recebe como parâmetro o custo total do anúncio e a referência para o publicitário que contratou o anúncio, e chama o método `efetuarPagamento()` declarado na Classe Conta (unidade 3.11 do capítulo 3) que irá retirar o valor do anúncio do saldo do publicitário.

```
public void efetuarContasAnuncioPublicitario(double custoTotal, Conta contaPublicitario) {
    contaPublicitario.efetuarPagamento(custoTotal);
}
```

15. O método que realiza atualizações nos dados do fundador, caso este opte em alterar seus dados pessoais.

```
public void atualizarDadosFundador() {  
  
    for (int i = 0; i < dadosConfiguracao.size(); i++) {  
  
        if (i == 0 && !dadosConfiguracao.get(0).equals(listaUtilizadores.get(0).getNome())) {  
            dadosConfiguracao.set(0, listaUtilizadores.get(0).getNome());  
        }  
  
        if (i == 1 && !dadosConfiguracao.get(1).equals(listaUtilizadores.get(0).getMorada())) {  
            dadosConfiguracao.set(1, listaUtilizadores.get(0).getMorada());  
        }  
  
        if (i == 2 && !dadosConfiguracao.get(2).equals(listaUtilizadores.get(0).getTelefone())) {  
            dadosConfiguracao.set(2, listaUtilizadores.get(0).getTelefone());  
        }  
  
        if (i == 4 && !dadosConfiguracao.get(4).equals(listaUtilizadores.get(0).getPassword())) {  
            dadosConfiguracao.set(4, listaUtilizadores.get(0).getPassword());  
        }  
    }  
}
```

16. O método que adiciona o utilizador criado à lista de todos os utilizadores.

```
public void adicionarUtilizadorNaArrayList(Utilizador utilizador) {  
    if (utilizador != null) {  
        listaUtilizadores.add(utilizador);  
    }  
}
```

17. O método que adiciona a conta corrente do novo utilizador criado (caso este seja um publicitário ou um programador) à lista de todas as contas corrente.

```
public void adicionarContaNaArrayList(Conta conta) {  
    if (conta != null) {  
        listaContas.add(conta);  
    }  
}
```

Por fim, cabe destacar que a presente classe implementa a interface *Serializable* a fim de identificar que esta classe é serializável, e que permite salvar o estado atual/dados dos objetos em ficheiros externos em formato binário, permitindo ainda que estes dados sejam recuperados posteriormente recriando o objeto em memória assim como ele estava no momento da sua serialização.

3.3 Classe Utilizador:

É onde são declarados os atributos comuns a todos os tipos de utilizadores que sejam intermediários, publicitários ou programadores.

Ficou definida como uma classe abstrata, pois nunca foi necessário instanciar a mesma em momento algum do programa, sendo certo que, somente são criadas instâncias de suas subclasses.

A classe tem como atributos:

1. Nome.
2. Morada.
3. Telefone.
4. Id Utilizador.
5. E-mail.
6. Password.
7. GeradorId: auxiliar que garante a geração do Id de forma automática pelo programa.

Este sétimo atributo foi declarado como *static* para auxiliar na atribuição de um número de identificação a cada utilizador e foi concebido tendo em conta os requisitos do projeto: este ID deve ser gerado automaticamente pelo sistema.

Ao declarar o atributo *geradorId* como um atributo *static*, ficou garantido que o mesmo se comporte como uma constante, e que todos os objetos ao acederem e modificarem esse atributo, acederão ao mesmo valor, ao mesmo espaço da memória, e que a mudança poderá ser vista em todos os objetos.

```
public static void buscarValorAtualizadoParaAlterador(int novoValor){  
    geradorId = novoValor;  
}
```

O método acima, em conjunto com o demonstrado no item 13 do item 3.2 capítulo 3, permitiu utilizar um iterador e gerar automaticamente os IDs dos utilizadores na ordem e numeração corretas, mesmo após terminar e recomeçar a execução do programa.

Esta classe possui os métodos:

- Getter de todos os seus atributos.
- Setter dos atributos nome, morada e telefone que são necessários à implementação da lógica que permite ao utilizador atualizar/alterar estes dados.
- Construtor, que para além de receber os parâmetros e os atribuir aos atributos da classe, também possui o iterador static do id do utilizador:

```
public Utilizador(String nome, String morada, String telefone, String email, String password) {  
    this.nome = nome;  
    this.morada = morada;  
    this.telefone = telefone;  
    this.email = email;
```

```
this.idUtilizador = geradorId++ ;  
this.password = password;  
}
```

Por fim, cabe destacar que a classe implementa a interface *Serializable* a fim de identificar que esta classe é serializável, e que permite salvar o estado atual/dados dos objetos em ficheiros externos em formato binário, permitindo ainda que estes dados sejam recuperados posteriormente recriando o objeto em memória assim como ele estava no momento da sua serialização.

3.4 Classe Publicitário:

É uma de três subclasses da classe Utilizador.

Possui somente um atributo, além dos atributos herdados de Utilizador:

- ArrayList <Frase> listaFrases: garante que cada publicitário tenha sua própria lista de frases associadas a si.

É importante destacar que os publicitários podem registar suas frases para serem acedidas e publicitadas em outros momentos, assim como podem atualizar/editar o conteúdo de suas frases, como será melhor detalhado na parte do relatório referente a interface gráfica.

A presente classe possui os métodos:

- adicionarFrases() que recebe como parâmetro a frase registada pelo publicitário e adiciona-a na ArrayList.
- Getter do atributo listaFrases.
- Construtor.

Por fim, cabe destacar que a classe implementa a interface *Serializable* a fim de identificar que esta classe é serializável, e que permite salvar o estado atual/dados dos objetos em ficheiros externos em formato binário, permitindo ainda que estes dados sejam recuperados posteriormente recriando o objeto em memória assim como ele estava no momento da sua serialização.

3.5 Classe Programador:

É outra de três subclasses da classe Utilizador.

Possui apenas um atributo, além dos atributos herdados de Utilizador:

- ArrayList<Aplicacao> listaAplicacoes: garante que cada programador tenha sua própria lista de aplicações associadas a si.

É relevante destacar que os programadores podem registrar suas aplicações, assim como podem atualizar/editar o conteúdo das mesmas, como será melhor detalhado na parte do relatório referente a interface gráfica.

Esta classe possui os seguintes métodos:

- adicionarApps () que recebe como parâmetro a aplicação registada pelo programador e a adiciona na ArrayList respetiva.
- Getter do atributo listaAplicacoes.
- Construtor.

Por fim, cabe destacar que a classe implementa a interface *Serializable* a fim de identificar que esta classe é serializável, e que permite salvar o estado atual/dados dos objetos em ficheiros externos em formato binário, permitindo ainda que estes dados sejam recuperados posteriormente recriando o objeto em memória assim como ele estava no momento da sua serialização.

3.6 Classe Intermediário:

É a última de três subclasses da classe Utilizador.

Não possui nenhum atributo ou método, para além dos atributos herdados de Utilizador, pelo que o seu construtor serve de referente à superclasse.

Esta classe foi criada com o objetivo de permitir que existam utilizadores com nível de acesso diferenciado, que atuam como empregados da empresa AorWords. Como empregados da empresa, acabam por ter acesso a funcionalidades únicas, podendo até, em alguns momentos do programa, atuar em nome dos publicitários ou dos programadores.

Por fim, cabe destacar que a classe implementa a interface *Serializable* a fim de identificar que esta classe é serializável, e que permite salvar o estado atual/dados dos objetos em ficheiros externos em formato binário, permitindo ainda que estes dados sejam recuperados posteriormente recriando o objeto em memória assim como ele estava no momento da sua serialização.

3.7 Classe Frase:

É composta somente pelo atributo “frase”.

A classe foi criada para permitir que cada publicitário tenha a sua própria lista de frases.

Possui os métodos getter e setter do atributo, este último criado a fim de permitir que o publicitário, possa proceder a alterações do conteúdo das frases.

Por fim, é de destacar que a classe implementa a interface *Serializable* a fim de identificar que esta classe é serializável, e que permite salvar o estado atual/dados dos objetos em ficheiros externos em formato binário, permitindo ainda que estes dados sejam recuperados posteriormente recriando o objeto em memória assim como ele estava no momento da sua serialização.

3.9 Classe Aplicação:

Possui os atributos:

1. nome: nome da aplicação;
2. descricao: descrição da aplicação;
3. valorUnitarioPorExibicao: valor cobrado por cada exibição de uma frase nesta aplicação.

A presente classe foi criada para permitir que cada programador tenha sua própria lista de aplicações.

Possui os métodos getter e setter do atributo, este último criado a fim de permitir que o programador, possa atualizar/alterar os conteúdos das suas aplicações.

Por fim, cabe referir que a classe implementa a interface *Serializable* a fim de identificar que esta classe é serializável, e que permite salvar o estado atual/dados dos objetos em ficheiros externos em formato binário, permitindo ainda que estes dados sejam recuperados posteriormente recriando o objeto em memória assim como ele estava no momento da sua serialização.

3.10 Classe Anúncio:

A classe encontra-se dotada dos seguintes atributos:

1. valorAnuncio: valor total do anúncio contratado.
2. valorUnitario: valor cobrado por cada exibição na aplicação escolhida para o anúncio.
3. dataContrato: data em que o contrato do anúncio foi efetuado. Esta data é automaticamente gerada pelo programa ao fazer uso da data atual do sistema e atribuí-la ao atributo.
4. numExibicoes: quantidade de exibições contratadas.
5. anunciante: referência ao publicitário que contratou o anúncio.
6. anunciador: referência ao programador a quem pertence a aplicação contratada para o anúncio.
7. fraseAnunciada: referência a frase que deve ser exibida.
8. aplicacao: referência à aplicação onde a frase deve ser exibida.
9. receitaProgramador: valor da receita do programador a quem pertence a aplicação.
10. receitaEmpresa: valor da receita da empresa com aquele anúncio.

Construtor da classe:

```
public Anuncio(int numExibicoes, double valorAnuncio, Aplicacao aplicacao, Frase fraseAnunciada,
Publicitario anunciante, Programador anunciador) {
```

```
    this.numExibicoes = numExibicoes;
    this.valorAnuncio = valorAnuncio;
    this.aplicacao = aplicacao;
```

```

this.anunciante = anunciante;
this.anunciador = anunciador;
this.fraseAnunciada = fraseAnunciada;
this.dataContrato = java.time.LocalDate.now();
this.valorUnitario = aplicacao.getValorUnitarioPorExibicao();

}

```

Possui os métodos *getter* de todos os atributos acima, com exceção do atributo 3.

Possui os métodos *setter* dos atributos 9 e 10.

Por fim, destaca-se que a classe implementa a interface *Serializable* a fim de identificar que esta classe é serializável, e que permite salvar o estado atual/dados dos objetos em ficheiros externos em formato binário, permitindo ainda que estes dados sejam recuperados posteriormente recriando o objeto em memória assim como ele estava no momento da sua serialização.

3.11 Classe Conta:

Esta classe possui os seguintes atributos:

1. user: que guarda a referência do utilizador a quem pertence a conta.
2. saldo: é o saldo atual daquela conta.
3. listaMovimentosFinanceiros: ArrayList do tipo MovimentosFinanceiros que armazena todos os movimentos financeiros daquela determinada conta.

Possui os métodos:

1. Construtor: recebe como parâmetro o utilizador a quem irá pertencer a conta criada.
2. *Getter* de todos os atributos.
3. efetuarDeposito() : recebe como parâmetro o respetivo valor a ser depositado na conta do publicitário/programador/empresa, faz o depósito e adiciona o movimento à listaMovimentosFinanceiros respetiva.

```

public void efetuarDeposito(double valorADepositar) {

    saldo = saldo + valorADepositar;

    MovimentoFinanceiro movimentoFinanceiro = new
    MovimentoFinanceiro(String.valueOf(TipoDeMovimento.DEPOSITO), valorADepositar);

    adicionarMovFinanceiros(movimentoFinanceiro);
}

```

4. efetuarPagamento() : recebe como parâmetro o custo do anúncio, e retira este valor da conta do respetivo publicitário e adiciona o movimento à listaMovimentosFinanceiros do mesmo:

```

public void efetuarPagamento(double valorAPagar) {

```



```

        saldo = saldo - valorAPagar;

        MovimentoFinanceiro movimentoFinanceiro = new
MovimentoFinanceiro(String.valueOf(TipoDeMovimento.PAGAMENTO), valorAPagar);

        adicionarMovFinanceiros(movimentoFinanceiro);
    }

```

5. `efetuarLevantamento()` : recebe como parâmetro o valor que o utilizador deseja levantar e retira este valor do respetivo saldo e, de seguida adiciona o movimento à lista `MovimentosFinanceiros` do utilizador a que concerne:

```

    public void efetuarLevantamento(double valorALevantar) {

        saldo = saldo - valorALevantar;

        MovimentoFinanceiro movimentoFinanceiro = new
MovimentoFinanceiro(String.valueOf(TipoDeMovimento.LEVANTAMENTO), valorALevantar);

        adicionarMovFinanceiros(movimentoFinanceiro);
    }

```

Por fim, cabe destacar que a classe implementa a interface *Serializable* a fim de identificar que esta classe é serializável, e que permite salvar o estado atual/dados dos objetos em ficheiros externos em formato binário, permitindo ainda que estes dados sejam recuperados posteriormente recriando o objeto em memória assim como ele estava no momento da sua serialização.

3.12 Classe MovimentoFinanceiro:

A classe detém como atributos:

1. `data`: atributo do tipo `LocalDate`, recebe a data atual do sistema como sendo a data do movimento financeiro.
2. `tipoMovimento`: recebe o tipo de movimento financeiro.
3. `valor`: recebe o valor daquele movimento financeiro.

a classe é possuidora dos métodos:

1. Construtor: recebe como parâmetro o tipo de movimento e o valor do movimento:

É de ressaltar que no construtor, o atributo `data` recebe o valor da data atual do próprio sistema, que será a data atribuída ao movimento.

```

    public MovimentoFinanceiro(String tipoMovimento, double valor) {

        this.data = java.time.LocalDate.now();
        this.tipoMovimento = tipoMovimento;
        this.valor = valor;
    }

```

2. *Getter* de todos os atributos.

Por fim, chama-se a atenção de que a presente classe implementa a interface *Serializable* a fim de identificar que esta classe é serializável, e que permite salvar o estado atual/dados dos objetos em ficheiros externos em formato binário, permitindo ainda que estes dados sejam recuperados posteriormente recriando o objeto em memória assim como ele estava no momento da sua serialização

3.13 Enumerador TipoDeMovimento:

Guarda uma lista de valores pré-definidos a serem usados como os tipos de movimentos financeiros existentes no sistema. Estes são os tipos de movimentos contidos no enumerador:

```
public enum TipoDeMovimento {  
    DEPOSITO,  
    LEVANTAMENTO,  
    PAGAMENTO  
}
```

A escolha de usar um enumerador deve-se ao facto de que os tipos de movimentos financeiros consistem num conjunto fixo de constantes pré-definidos, ou seja, que não devem/irão mudar em momento algum durante a execução do programa.

CAPÍTULO 4

Algumas funções básicas do Sistema

4.1 Sobre a leitura e atualização do Ficheiro de Configuração:

Antes de prosseguir para a lógica adotada na interface gráfica do programa, afigura-se fulcral esclarecer de que modos se procedeu à leitura e carregamento dos dados do ficheiro de configuração, uma vez que o programa somente poderá decorrer de forma satisfatória após obter os dados iniciais necessários que estão definidos neste ficheiro.

Para o ficheiro de configuração, foi utilizado um ficheiro de texto denominado como “FicheiroDeConfiguracao.dat”, onde cada linha do ficheiro se refere a um dado específico relevante ao funcionamento do programa, garantindo que não fossem inseridos dados em formato *hardcoded*.

No ficheiro de configuração temos as seguintes informações:

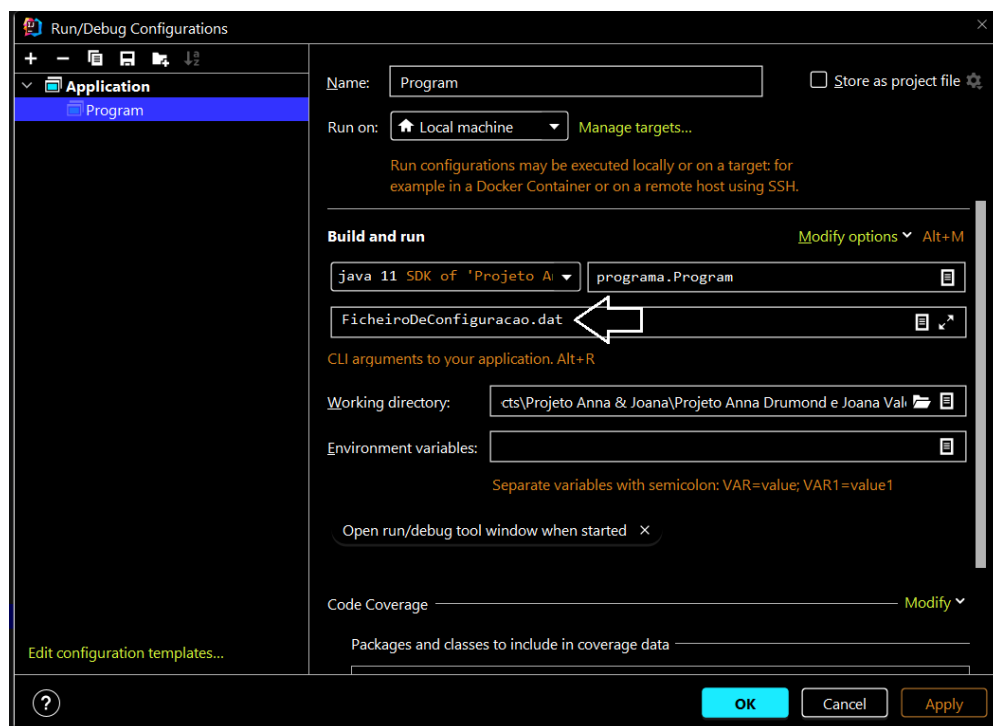
1. Os dados pessoais do intermediário fundador, o que inclui seu e-mail e palavra-chave de acesso, já informados ao início deste relatório;
2. O nome do ficheiro de objetos, que será usado para resgatar a informação do ficheiro de objetos (caso este exista) ou proceder à sua criação (caso este não exista) e ainda proceder à escrita no

mesmo aquando do fecho do programa, eliminando desta forma a necessidade de deixar o seu nome em formato *hardcoded*;

3. O nome do ficheiro de idiomas, essencial para se poder alterar o idioma aquando da execução do programa, o que permite retirar o *hardcode* referente ao nome deste ficheiro;
4. O nome do próprio ficheiro de configuração, para ser usado no método `atualizarFicheiroDeConfiguracao()` sem a necessidade de por o nome *hardcoded*;
5. O valor booleano que define o idioma em que o sistema será iniciado, em que o valor *true* é referente à língua portuguesa e o *false* à língua inglesa;
6. As percentagens referentes aos cálculos da receita da empresa e do programador;
7. O nome que aparece ao topo da JFrame;
8. O nome do ficheiro do logotipo;
9. O nome do ficheiro da imagem usada para o comando fechar/encerrar o programa;
10. O nome dos ficheiros das imagens usadas nos botões de troca de idiomas;
11. Os nomes usados para os botões de idiomas;

A leitura destes dados foi efetuada através do método `extrairDadosFicheiroDeConfiguracao()`, que se encontra implementado na classe gestora `EmpresaAorWords` e detalhado no item 3 da unidade 3.2 do capítulo 3 deste relatório.

Conforme orientação em sala de aula, para fins de aprendizagem, foi inserido no vetor “args” do tipo String do método `main`, o nome do ficheiro de configuração:



Após, para que pudesse ser utilizado este valor inserido na posição 0, o vetor “args” foi enviado como parâmetro para a classe `EmpresaAorWords`, que por sua vez o enviou como parâmetro para o método `extrairDadosFicheiroDeConfiguracao()`.

Para evitar possíveis erros, o método `extrairDadosFicheiroDeConfiguracao()`, recebeu ainda a estrutura condicional abaixo:

```

if (args.length > 0 && args[0].equals("FicheiroDeConfiguracao.dat")) {
    if (!ficheiroDeTexto.abrirLeitura(args[0])) {
        System.out.println("FicheiroDeConfiguracao.dat não encontrado.");
        System.exit(1);
    }
} else if (!ficheiroDeTexto.abrirLeitura("FicheiroDeConfiguracao.dat")) {
    System.out.println("FicheiroDeConfiguracao.dat não encontrado.");
    System.exit(1);
}

```

Cumpre destacar que o método acima citado, é chamado dentro do construtor da classe EmpresaAorWords, que por sua vez, é instanciada na classe Program (detentora do método main, detalhada na unidade 3.1 capítulo 3), logo quando do arranque do programa. Isso irá garantir que já ao início da execução, todos os dados do ficheiro de configuração já estejam carregados em memória.

Afim de garantir esta funcionalidade a classe gestora EmpresaAorWords, contém os seguintes componentes:

- Uma única instância da classe FicheiroDeTexto usada no programa.
- Um método denominado extrairDadosFicheiroDeConfiguracao() acima explorado, que lê e extrai todos os dados do ficheiro para um ArrayList do tipo String denominado dadosConfiguracao.
- Atributos que recebem cada uma das informações lidas.
- Métodos *getter* que permitem o acesso à informação, através de uma instância de EmpresaAorWords declarada na classe Program (que contém o método main), e passada como parâmetro para todas as classes da GUI.

Além disso, há ainda o método atualizarFicheiroDeConfiguracao() (que se encontra no item 4 da unidade 3.2 do capítulo 3), que realiza a escrita dos dados que constam no ArrayList dadosConfiguracao.

Isto garante que o ficheiro de configuração seja atualizado com possíveis alterações ocorridas em tempo de execução, nos seguintes casos:

- O idioma inicial do programa é o português, contudo, se o utilizador durante a execução do programa, alterar o idioma, este método irá realizar a alteração no ficheiro de configuração, garantindo que quando o programa seja executado novamente, a interface com o utilizador seja carregada no último idioma escolhido em tempo de execução.
- Os dados pessoais do intermediário fundador, se encontram gravados no ficheiro de configuração. O programa ao prever a hipótese do fundador querer alterar algum ou vários de seus dados pessoais, precisou igualmente de prever um método que garantisse que o ficheiro de configuração fosse atualizado com as novas informações do fundador.

4.2 Classe que implementa os comandos de leitura e escrita dos ficheiros de texto:

De forma a compreender como se efetivamente efetua a leitura e a escrita do ficheiro de texto, tanto o ficheiro de configuração como o ficheiro de idiomas (que se será mais adiante analisado), não se pode ignorar o papel que a classe denominada FicheiroDeTexto tem para o exercício desta atividade. É fundamental referir que nesta classe foi utilizado o modelo disponibilizado em sala de aula pela professora (mas com ausência de alguns dos métodos que não foram utilizados no presente projeto).

A classe `FicheiroDeTexto` é constituída por dois atributos:

- `bufferedReaderAtributo`: permite ler ficheiros, mas apresenta algumas vantagens relativamente à classe `FileReader`, pois em vez de ler carácter a carácter, a classe pública `BufferedReader` (que é uma subclasse da classe `FileReader`) lê conjuntos de caracteres de cada vez, armazenando estes conjuntos no buffer, pelo que, desta forma, se torna mais eficiente uma vez que, em vez do leitor, lerá deste buffer.⁴
- `bufferedWriterAtributo`: permite escrever no ficheiro de texto, no entanto, como faz uso de um buffer, não vai escrever diretamente no ficheiro, mas sim no buffer, e, quando o buffer estiver cheio ou se encerrar a escrita, procede à escrita no ficheiro de objeto.⁵ A classe pública `BufferedWriter` é, também, uma subclasse de `FileWriter`.

A atual classe encontra-se ainda dotada de 6 métodos:

- `abrirLeitura`: vai criar uma nova instância do atributo `bufferedReader` para poder ler aquele ficheiro. O construtor desta classe vai receber uma nova instância da classe `InputStreamReader`, e esta que, por sua vez, receberá uma nova instância de `FileInputStream` que tem a capacidade de ler bytes de informação do ficheiro indicado⁶ e descodificá-los em caracteres através de um charset específico (neste caso, o selecionado foi o UTF8⁷). Este método retorna um valor booleano, consoante o resultado quanto a tentativa de abrir o ficheiro para leitura.

```
public boolean abrirLeitura(String nomeDoFicheiro){  
  
    try{  
        bufferedReaderAtributo = new BufferedReader(new InputStreamReader(new  
FileInputStream(nomeDoFicheiro), StandardCharsets.UTF_8));  
        return true;  
    } catch (IOException erro){  
        return false;  
    }  
}
```

- `abrirEscrita`: vai criar uma nova instância do atributo `bufferedWriter` de forma a poder escrever no ficheiro. O seu construtor necessita que seja enviado para lá um escritor e o nome ficheiro a escrever.
- `lerLinha`: lê uma linha do ficheiro de texto.
- `escreverLinha`: escreve uma linha do ficheiro de texto.
- `fecharLeitura`: quando se terminar a leitura do ficheiro de texto, é importante chamar o método `close()` para libertar os recursos que foram alocados à leitura.⁸
- `fecharEscrita`: quando terminar a escrita no ficheiro de texto, é importante chamar o método `close()` para limpar o buffer e parar a escrita.

4.3 Sobre a leitura e escrita no Ficheiro de Objetos:

A leitura e escrita dos objetos no ficheiro de objetos ocorre na classe gestora `EmpresaAorWords` com o auxílio da classe `FrameUnica` na escrita (isto porque o que vai ativar a escrita no ficheiro de objetos é o clicar de um botão da `JFrame` - ver abaixo e na unidade 5.5 do capítulo 5).

⁴ <https://www.baeldung.com/java-buffered-reader> (acedido a 07/01/2022).

⁵ <https://www.programiz.com/java-programming/bufferedwriter> (acedido a 07/01/2022).

⁶ <https://docs.oracle.com/javase/7/docs/api/java/io/FileInputStream.html> (acedido a 11/01/2022).

⁷ <https://docs.oracle.com/javase/7/docs/api/java/io/InputStreamReader.html> (acedido 11/01/2022).

⁸ <https://docs.oracle.com/javase/7/docs/api/java/io/BufferedReader.html> (acedido a 08/01/2022).

Para implementar a lógica necessária, a classe EmpresaAorWords:

1. Contém a única instância da classe FicheiroDeObjeto usada no programa.
2. Seu construtor possui chamada para o método lerListasDoFicheiroGeral(): possui as chamadas dos métodos necessários para a leitura dos dados binários gravados no ficheiro. Insere os dados lidos nas suas ArrayLists respetivas de modo a permitir a sua utilização durante a execução do programa.
3. Contém o método escreverNoFicheiroASaida(): possui as chamadas dos métodos necessários para a escrita dos dados inseridos durante a execução do programa.

Para a gravação dos dados ocorrer de forma satisfatória, concebeu-se que o utilizador somente poderia encerrar sua interação com o programa através do botão sair, caracterizado por uma imagem de uma porta de saída. Neste seguimento, foi desabilitada a hipótese do utilizador encerrar o programa através do símbolo X que surge no canto superior direito do ecrã (ver unidade 5.2 do capítulo 5).



Tal fato permitiu que, quando o utilizador clique na imagem da porta de saída, seja acionado um ActionListener, que antes do efetivo encerramento do programa chama os métodos de escrita no ficheiro de objetos e de escrita no ficheiro de configuração, como demonstrado abaixo:

```
botaoSair.addActionListener(eventoSair -> {  
  
    empresaObjeto.escreverNoFicheiroGeralASaida();  
  
    empresaObjeto.atualizarFicheiroDeConfiguracao();  
  
    objetoJframe.dispose();  
  
    System.exit(0);  
  
});
```

A funcionalidade do método atualizarFicheiroDeConfiguracao(), será melhor explicada na parte deste relatório respetiva a troca de idiomas do programa (ver unidade 4.6 do mesmo capítulo).

4.4 Classe que implementa os comandos de leitura e escrita do ficheiro de objetos:

De forma a compreender como se efetivamente efetua a leitura e a escrita do ficheiro de objetos, é essencial olhar para a classe FicheiroDeObjeto. É relevante informar que na presente classe foi utilizado o modelo disponibilizado em sala de aula pela professora.

A classe FicheiroDeObjeto é constituída por dois atributos:

- `objectInputStream`: é um objeto da classe `ObjectInputStream` que, por sua vez, é uma subclasse da classe `InputStream` bem como implementa os métodos da interface `ObjectInput` e `ObjectStreamConstants`. Este atributo vai permitir a leitura de ficheiros de objetos ao desserializar os dados/objetos que foram previamente escritos pela classe `ObjectOutputStream`⁹.
- `objectOutputStream`: é um objeto da classe `ObjectOutputStream` que, por sua vez, é uma subclasse da classe `OutputStream` bem como implementa os métodos da interface `ObjectOutput` e `ObjectStreamConstants`. Este atributo vai permitir escrever ficheiros de objetos para que estes possam posteriormente ser lidos pela classe `ObjectInputStream`. É relevante lembrar que apenas os objetos pertencentes a classes que implementem a interface *Serializable* podem ser escritos no ficheiro de objetos¹⁰.

A presente classe possui os seguintes métodos:

- `abreLeitura`: vai criar uma nova instância do atributo `objectInputStream` para poder ler aquele ficheiro de objetos. O construtor desta classe necessita que seja enviado para lá uma instância da classe `FileInputStream` com ficheiro de objetos a ler.
- `abreEscrita`: vai criar uma nova instância do atributo `objectOutputStream` para poder escrever o ficheiro de objetos. O construtor desta classe necessita que seja enviado para lá uma instância da classe `FileOutputStream` com o ficheiro de objetos a escrever.
- `leObjeto`: lê um objeto.
- `escreveObjeto`: escreve um objeto.
- `fechaLeitura`: fecha a leitura. É importante chamar este método após terminar a leitura do ficheiro de objetos para libertar os recursos que foram alocados à leitura do mesmo (ver nota de rodapé 7).
- `fechaEscrita`: fecha a escrita. É importante chamar este método após terminar a escrita no ficheiro de objetos de forma a libertar os recursos que foram alocados à escrita no mesmo (ver nota de rodapé 8).

4.5 Controle do utilizador ativo:

Para poder controlar as informações a serem apresentadas e/ou modificadas, é necessário que durante toda a execução do programa ficasse armazenada a referência para o utilizador que fez login na aplicação.

No intuito de garantir que toda a interface gráfica tivesse acesso a esta referência, foram implementados os seguintes comandos:

- A classe `EmpresaAorWords` contém um atributo denominado como `utilizadorAtivo` que é um atributo do tipo `Utilizador`.
- A mesma classe possui métodos *getter* e *setter* deste atributo.

Resta destacar que a classe `EmpresaAorWords` é instanciada uma única vez na classe `Program` (detentora do método `main`), e que esta instância é passada como parâmetro para todas as classes que compõe a GUI, permitindo acesso aos métodos `getter` e `setter`, garantindo o perfeito acesso a informação relativa ao utilizador com login ativo durante toda a execução do programa.

4.6 Sobre a lógica implementada para a troca de idiomas do sistema:

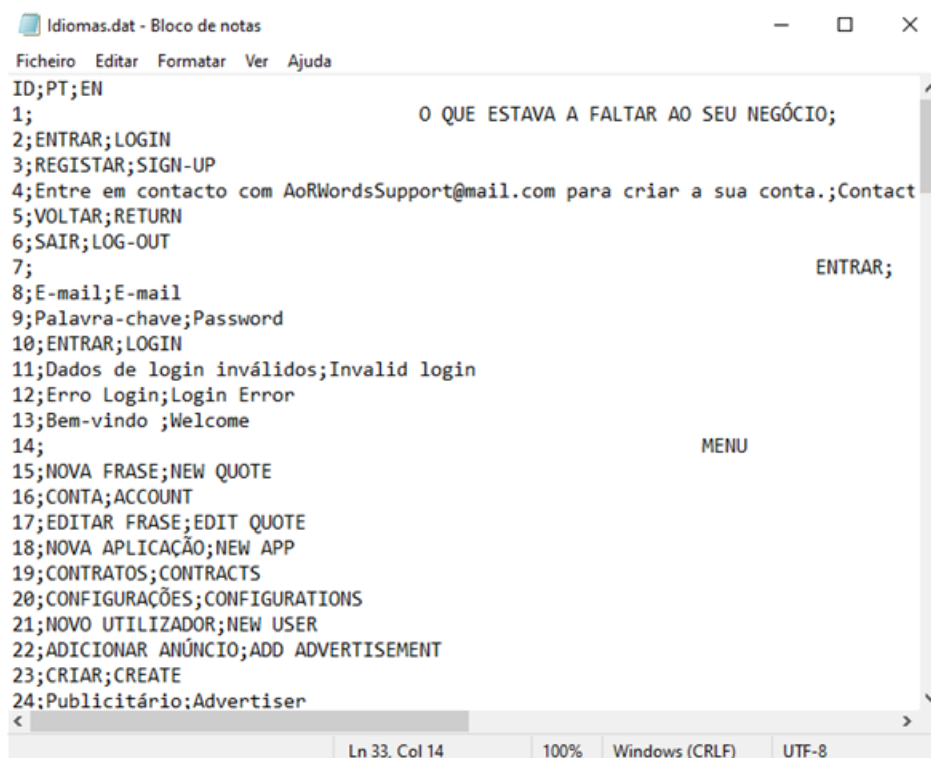
⁹ <https://docs.oracle.com/javase/7/docs/api/java/io/ObjectInputStream.html> (acedido a 08/01/2022).

¹⁰ <https://docs.oracle.com/javase/7/docs/api/java/io/ObjectOutputStream.html> (acedido a 08/01/2022).

A troca de idiomas do sistema foi construída com os seguintes critérios:

1. Existe previamente um ficheiro de texto, onde se encontram armazenados todos os textos usados no programa em ambos os idiomas, português e inglês.

Cada linha do ficheiro contém um ID (inteiro), o texto em português (PT) e o texto em inglês (EN), todos separados por “;”, nos termos definidos nos requisitos do projeto, como se pode verificar na imagem abaixo que mostra uma parte do mesmo:



```
Idiomas.dat - Bloco de notas
Ficheiro Editar Formatar Ver Ajuda
ID;PT;EN
1; O QUE ESTAVA A FALTAR AO SEU NEGÓCIO;
2;ENTRAR;LOGIN
3;REGISTAR;SIGN-UP
4;Entre em contacto com AoRWordsSupport@mail.com para criar a sua conta.;Contact
5;VOLTAR;RETURN
6;SAIR;LOG-OUT
7; ENTRAR;
8;E-mail;E-mail
9;Palavra-chave;Password
10;ENTRAR;LOGIN
11;Dados de login inválidos;Invalid login
12;Erro Login;Login Error
13;Bem-vindo ;Welcome
14; MENU
15;NOVA FRASE;NEW QUOTE
16;CONTA;ACCOUNT
17;EDITAR FRASE;EDIT QUOTE
18;NOVA APLICAÇÃO;NEW APP
19;CONTRATOS;CONTRACTS
20;CONFIGURAÇÕES;CONFIGURATIONS
21;NOVO UTILIZADOR;NEW USER
22;ADICIONAR ANÚNCIO;ADD ADVERTISEMENT
23;CRIAR;CREATE
24;Publicitário;Advertiser
```

Figura 2 - Demonstração do conteúdo do Ficheiro de Idiomas

2. O idioma inicial do sistema é o português.
3. Para a troca dos idiomas de facto ocorrer em fase de execução do programa, a classe *EmpresaAorWords* recebeu alguns atributos, quais sejam:
 - a. boolean *idiomaPortugues*: recebe o valor *true* quando escolhido o idioma em português e recebe o valor *false*, quando escolhido o idioma em inglês.
 - b. *JPanel* *ecraAtivo*: recebe uma referência à classe respetiva ao ecrã em que o utilizador está no momento em que for acionado um dos botões de troca de idioma.
 - c. *JPanel* *painelAtivo*: recebe uma referência ao *JPanel* que está adicionado ao *JFrame* no momento em que for acionado um dos botões de troca de idioma.

Cumpre destacar que, em relação aos atributos acima, estes receberam também um método *getter* e *setter* para cada um deles, para permitir acesso aos seus valores no resto do programa, através de uma instância

de EmpresaAorWords criada na classe Program (contém o método main) e passada como parâmetro para todas as classes relativas a GUI.

4. Na classe TrocaDeIdioma, foi definido um atributo denominado idiomasLista do tipo ArrayList<String[]>, e também um método extrairLinhasFicheiroIdiomas(), responsável por ler o ficheiro de idiomas, linha a linha, separar os valores e depois armazenar em memória no atributo idiomasLista, a fim de que os textos possam ser usados durante a execução do programa.

```
public void extrairLinhasFicheiroIdiomas(EmpresaAorWords empresaObjeto) {
    try {

        if (!ficheiroDeTexto.abrirLeitura(empresaObjeto.getDadosConfiguracao().get(6))) {
            System.out.println("Idiomas.dat não encontrado.");
            System.exit(1);
        }

        String auxiliarDeLinha;
        String[] auxiliarDeSplit;

        while ((auxiliarDeLinha = ficheiroDeTexto.lerLinha()) != null) {
            auxiliarDeSplit = auxiliarDeLinha.split(";");
            idiomasLista.add(auxiliarDeSplit);
        }
        ficheiroDeTexto.fecharLeitura();
    } catch (IOException erro) {
        erro.printStackTrace();
    }
}
```

5. A classe TrocaDeIdioma foi instanciada apenas uma vez (dentro da classe FrameUnica, que será detalhada mais adiante), e, esta instância foi passada como parâmetro para as demais classes da GUI.

6. A implementação dos botões que permitem a troca entre idiomas ficou na classe FrameUnica, assim como o método actionPerformed() destes botões.

7. O método actionPerformed() é responsável pela troca dos valores booleanos no atributo idiomaPortugues (definido na classe EmpresaAorWords), através do método setter deste atributo. É responsável também, por chamar o método recarregarEcraAtivoComNovoIdioma().

```
@Override
public void actionPerformed(ActionEvent eventoBotaoIdioma) {

    if (eventoBotaoIdioma.getSource() == botaoPortugues) {
        empresaObjeto.setIdiomaPortugues(true);
        botaoPortugues.setSelected(true);
        recarregarEcraAtivoComNovoIdioma();

    } else if (eventoBotaoIdioma.getSource() == botaoIngles) {
        empresaObjeto.setIdiomaPortugues(false);
        botaoIngles.setSelected(true);
        recarregarEcraAtivoComNovoIdioma();
    }
}
```

8. Para melhor entendimento quanto às funcionalidades é fulcral proceder a alguns breves esclarecimentos sobre os atributos destacados nas alíneas b. e c. do item 3 acima. Cada um dos ecrãs de interface com o utilizador está construído numa classe própria, que recebe o nome do ecrã a que se refere. Dentro do construtor de cada classe é possível evidenciar a existência:

- Da criação de uma instância de JPanel.
- Da adição dos componentes do ecrã ao JPanel criado.
- Da adição do JPanel ao JFrame.
- E, por fim, da adição ao atributo `ecraAtivo`, a referência para o presente ecrã, e também é adicionado ao atributo `painelAtivo`, a referência do JPanel que naquele momento específico, está adicionado ao JFrame:

```
this.empresaObjeto.setEcraAtivo(this);  
this.empresaObjeto.setPainelAtivo(panellInicial);
```

9. O método `recarregarEcraAtivoComNovoIdioma()` faz uso da referência armazenada no atributo `ecraAtivo` para verificar qual a instância do ecrã se encontra ativo por meio do comando *instanceOf*, para, de seguida, utilizar a referência armazenada no atributo `painelAtivo` para remover completamente o JPanel da JFrame. Feito isso, o método cria uma nova instância do mesmo ecrã armazenado no atributo `ecraAtivo`, que será novamente carregado, porém com o novo idioma.

```
public void recarregarEcraAtivoComNovoIdioma() {  
  
    if (empresaObjeto.getEcraAtivo() instanceof EcraInicial) {  
  
        objetoJframe.remove(empresaObjeto.getPainelAtivo());  
  
        EcraInicial ecraInicial = new EcraInicial(empresaObjeto, objetoJframe,  
trocaDeldiomaObjeto, validadorGeral, avisosPopUp);  
  
    } else if (empresaObjeto.getEcraAtivo() instanceof EcraLogin) {  
  
        objetoJframe.remove(empresaObjeto.getPainelAtivo());  
  
        EcraLogin ecraLogin = new EcraLogin(empresaObjeto, objetoJframe, trocaDeldiomaObjeto,  
validadorGeral, avisosPopUp);  
    }  
}
```

Desta forma, foi possível remover o JPanel atual antes de criar um novo, evitando assim, “empilhar” novos JPanels por cima dos mais antigos, o que resultaria no uso de novos recursos sem libertar os que já não estão a ser utilizados e que levaria a um gasto desnecessário de memória.

Nesta fase, foi realizada uma pesquisa de forma a procurar uma forma de “destruir” a instância anterior do ecrã antes de se criar uma nova, contudo, foi verificado que o Java não possui este tipo de funcionalidade, devido à existência do Garbage Collector, e sua funcionalidade em ser responsável pela alocação e liberação da memória.

No que refere ao uso dos textos na interface gráfica, foram utilizadas estruturas condicionais de forma a verificar se o valor booleano armazenado no atributo “idiomaPortugues” era um valor *true* (português) ou um valor *false* (inglês), e, somente após esta verificação, o sistema acedia ao texto da respetiva componente e fazia uso do mesmo, como se pode verificar no trecho de código abaixo retirado da classe `EcraInicial`:

```

public BotoesEcraInicial() {

    if (empresaObjeto.isIdiomaPortugues()) {

        login.setText(trocaDeIdiomaObjeto.getIdiomasLista().get(2)[1]);
        signUp.setText(trocaDeIdiomaObjeto.getIdiomasLista().get(3)[1]);

    } else {

        login.setText(trocaDeIdiomaObjeto.getIdiomasLista().get(2)[2]);
        signUp.setText(trocaDeIdiomaObjeto.getIdiomasLista().get(3)[2]);
    }

    login.setBounds(270, 600, 120, 50);
    login.addActionListener(this);
    signUp.setBounds(410, 600, 120, 50);
    signUp.addActionListener(this);
}

```

Para carregar em cada componente o texto necessário, basta aceder à respetiva posição no ArrayList e aceder à posição 1 do vetor para textos em português e à posição 2 do vetor para textos em inglês. Isto foi possível porque, como já foi referido no item 4 acima, os textos foram carregados em memória num ArrayList do tipo vetor de String (ArrayList<String[]>).

Por fim, faltava registar o idioma escolhido pelo utilizador aquando do encerramento do programa, ou seja, se o utilizador escolher o idioma inglês e encerrar o programa, quando este programa voltar a ser aberto, o idioma se encontrasse em inglês.

Para garantir isso, a classe EmpresaAorWords, como já descrito no item, possui o método atualizarIdiomaDefault() que irá atualizar o valor booleano armazenado na ArrayList dadosConfiguracao, para quando o ficheiro de configuração for reescrito com os dados atualizados, seja também atualizado idioma inicial do programa:

```

public void atualizarIdiomaDefault(){

    if (isIdiomaPortugues()) {

        dadosConfiguracao.set(8, String.valueOf(true));
    } else {
        dadosConfiguracao.set(8, String.valueOf(false));
    }
}

```

Foi implementada ainda, uma lógica que segue as seguintes diretrizes:

- O utilizador escolhe o idioma durante o decorrer do programa.
- O utilizador clica na imagem de uma porta de saída que encerra o programa.

```

botaoSair.addActionListener(eventoSair -> {
    empresaObjeto.escreverNoFicheiroGeralASaida();
    empresaObjeto.atualizarFicheiroDeConfiguracao();
    objetoJframe.dispose();
    System.exit(0);
});

```

Antes do encerramento efetivo, é chamado o método `atualizarFicheiroDeConfiguracao()` (que se encontra acima no item 4 da unidade 3.2 do capítulo 3), que acede o ficheiro de texto de configuração e escreve/grava o idioma que se encontrava em utilização antes do encerramento do programa, garantindo que, quando o mesmo for reaberto, seja acionado o último idioma escolhido pelo utilizador.

E, desta forma, foram concluídos todos os requisitos definidos pelo projeto, referentes a troca de idiomas durante a execução do código.

4.6.1 Classe TrocaDeIdioma:

A classe `TrocaDeIdioma` é uma classe de implementação simples, responsável por extrair os textos do ficheiro de idiomas denominado `Idiomas.dat`, armazenado no diretório do projeto.

Estes textos são extraídos do ficheiro de texto e armazenados numa `ArrayList<String[]>` para serem usados no decorrer do programa.

Possui ainda um atributo denominado *String bufferAuxiliar* que será usado em algumas classes do programa (nas unidades 6.13, 6.14, 6.15 e 6.16 do Capítulo 6) para armazenar informações em memória e garantir que estas informações não se percam aquando da mudança entre um idioma e outro, uma vez que, ao acionar um dos botões de troca do idioma, o programa recarrega completamente a classe respetiva ao ecrã e seus componentes, mas sem este *buffer*, a informação enviada para esta classe seria perdida.

Capítulo 5

Abordagem padrão usada para a construção de todas as classes da interface Gráfica:

Primeiramente, é de destacar que a interface gráfica do projeto foi totalmente construída sem recorrer a ferramentas de geração automática de código.

Para a construção da interface de interação com o utilizador (GUI), houve a preocupação de atender aos conceitos de eficiência, eficácia e satisfação, tendo em conta o contexto de uso da aplicação.

Cada uma destas classes segue o conceito de classes aninhadas (*Nested Classes*) sugerido durante as aulas, isto é, cada classe possui outras classes dentro de si. Estas classes aninhadas correspondem às componentes gráficas necessárias nesta classe¹¹.

A interface gráfica começou a ser construída usando uma instância de `JFrame` para cada ecrã, que, por sua vez, gerava a necessidade de criar uma instância para cada um destes componentes:

- `JFrame`;
- `JRadioButtons` de troca de idiomas;

¹¹ <https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html> (acedido em 18/12/2021).

- JLabel do logotipo.

Para a troca entre os ecrãs inicialmente foi usado um método que se utilizava de um *Singleton*, que capturava a instância correta de JFrame e permitia o uso do método `.dispose()` na JFrame correta, fechando assim o ecrã, antes da abertura de um novo ecrã. Porém, isso gerava a necessidade de que, cada classe tivesse um objeto JFrame declarado como *static*, bem como a necessidade de criação de uma instância de cada um dos componentes citados.

Posteriormente, a interface gráfica foi modificada a fim sanar esta dispendiosa situação.

5.1 JFrame Única utilizada em todo o programa

Toda a GUI foi construída usando apenas uma única instância de JFrame e uma única instância dos componentes JRadioButtons de troca de idiomas e JLabel do logotipo, sendo estes instanciados e adicionados à JFrame na classe FrameUnica.

Esta instância criada foi passada como parâmetro para todos os ecrãs do projeto, evitando, desta forma, o uso de membros estáticos.

5.2 Estrutura comum a todas as classes da interface gráfica

No que se refere a estrutura básica comum as classes da GUI, cumpre destacar que:

1. Cada classe possui uma instância de JPanel.
2. Foi definido o *layout null* para todas as instâncias de JPanel.
3. Cada classe corresponde a um ecrã de interface com o utilizador.
4. Obedece sempre aos seguintes critérios básicos:

4.1 Cada componente que será adicionada à JPanel possui a sua própria classe, que é uma classe interna aninhada dentro da classe principal, após a declaração dos atributos mas antes do construtor desta classe principal. Cada classe principal representa um ecrã de interação com o utilizador. Para melhor entendimento, passa-se a citar a classe EcrãInicial que representa o ecrã inicial a ser visualizado pelo utilizador, e que tem, dentro de si, as classes dos seus componentes:

```
public class EcrãInicial extends JPanel {

    class BotoesEcrãInicial implements ActionListener {
        ...
    }
    class LabelEcrãInicial {
        ...
    }
}
```

4.2. A criação e instanciação dos componentes que integram aquele determinado ecrã, tais como JButtons, JComboBox, JTable, JTextFields, entre outros, decorre no respetivo construtor da classe interna que contém aquele determinado componente.

4.3. Implementação das estruturas condicionais para definição do idioma a ser exibido nos componentes.

4.4. Implementação das ações respectivas a cada componente, conforme necessidade.

4.5. Definição da localização de cada componente no ecrã.

4.6. Criação e instanciação de um objeto JPanel.

4.7. Adição dos componentes criados, ao JPanel.

4.8. Adição do JPanel ao JFrame.

Para melhor entendimento do acima descrito, segue abaixo trecho da classe EcraInicial que demonstra claramente a estrutura base de toda a GUI:

```
public EcraInicial(EmpresaAorWords empresaObjeto, JFrame objetoJframe, TrocaDeldioma trocaDeldiomaObjeto, ValidadorGeral validadorGeral, AvisosPopUp avisosPopUp) {

    this.objetoJframe = objetoJframe;
    this.empresaObjeto = empresaObjeto;
    this.trocaDeldiomaObjeto = trocaDeldiomaObjeto;
    this.validadorGeral = validadorGeral;
    this.avisosPopUp = avisosPopUp;
    panellInicial = new JPanel();

    BotoesEcraInicial objetoBotoes = new BotoesEcraInicial();
    LabelEcraInicial objetoLabel = new LabelEcraInicial();

    panellInicial.setBounds(800, 225, 800, 580);
    panellInicial.setLayout(null);
    panellInicial.add(login);
    panellInicial.add(signUp);
    panellInicial.add(labelEcraInicial);
    this.objetoJframe.add(panellInicial);

    this.empresaObjeto.setEcraAtivo(this);
    this.empresaObjeto.setPainelAtivo(panellInicial);

    this.objetoJframe.setVisible(true);
    panellInicial.setVisible(true);
}
```

E, por fim, o construtor de cada classe também é responsável por receber como parâmetro a instância única dos objetos abaixo, usadas durante todo o programa:

1. Instância do objeto respetivo a única JFrame usada no projeto.
2. Instância do objeto respetivo a classe agregadora EmpresaAorWords.

3. Instância do objeto respetivo a classe TrocaDeIdiomas, que será fundamental a troca entre idiomas durante a execução.
4. Instância do objeto respetivo a classe ValidadorGeral, que permitirá acesso aos métodos de validação de dados inseridos pelo utilizador.
5. Instância do objeto respetivo a classe AvisosPopUp, que armazena todas as PopUps de interação com o utilizador a fim de manter o código mais organizado.

Numa fase já avançada do projeto, foi verificado que algumas destas instâncias passadas como parâmetro para todas as classes da GUI, poderiam ser definidas com o modificador de acesso *static*, uma vez que estas instâncias seriam as mesmas a serem usadas durante o decorrer de todo o programa.

Contudo, optou-se por manter a implementação atual do programa, uma vez que, a passagem de parâmetros entre as classes foi de suma importância para o aprendizado, pois permitiu desenvolver uma melhor perceção quanto ao conceito de instâncias e a importância do uso da instância correta para um bom resultado de certos comandos.

A JFrame única ficou definida num tamanho 800x800 pixels, e cada JPanel ficou definido num tamanho 800x580 pixels a fim de ficasse visível a parte da JFrame que contém o logotipo e os botões de troca de idioma.

É ainda de referir que se impossibilitou a capacidade do utilizador aumentar e/ou diminuir o tamanho da JFrame, tal que foi feito com o objetivo de evitar que o programa sofresse desconfigurações a nível estético. Tal foi possível através da seguinte linha de código:

```
objetoJframe.setResizable(false);
```

Para além disso, para que quando o programa fosse iniciado surgisse centrado no monitor do utilizador, foi implementada a seguinte instrução no código:

```
objetoJframe.setLocationRelativeTo(null);
```

Por fim, como já foi acima referido, de forma a se poder proceder à escrita dos ficheiros de configuração e ficheiro de objetos, eliminou-se a possibilidade do utilizador sair do programa ao clicar o X no canto superior direito do programa através da seguinte linha de código:

```
objetoJframe.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
```

5.3 Dinâmica necessária na interface gráfica para a troca entre idiomas em tempo de execução

Conforme descrito anteriormente, todas as classes e todos os componentes que as compõem, utilizam estruturas condicionais a fim de verificar se o valor booleano armazenado no atributo “idiomaPortugues” é um valor *true* (português) ou um valor *false* (inglês), e somente após esta verificação, o sistema acede o respetivo texto e o usa no devido componente, como se pode verificar no pequeno trecho de código abaixo retirado da classe EcraLogin:

```
public BotoesEcraLogin() {
```

```

if (empresaObjeto.isIdiomaPortugues()) {

    entrar.setText(trocaDeIdiomaObjeto.getIdiomasLista().get(2)[1]);
    esqueceuDados.setText(trocaDeIdiomaObjeto.getIdiomasLista().get(134)[1]);

} else {
    entrar.setText(trocaDeIdiomaObjeto.getIdiomasLista().get(2)[2]);
    esqueceuDados.setText(trocaDeIdiomaObjeto.getIdiomasLista().get(134)[2]);
}
esqueceuDados.setBounds(275, 455, 250, 35);
esqueceuDados.addActionListener(this);

entrar.setBounds(340, 600, 120, 50);
entrar.addActionListener(this);
}

```

Ademais, o método construtor de cada classe, além de ser responsável por implementar os itens 4.5 a 4.7 da unidade 5.2 deste capítulo, também garante que seja armazenado em memória a instância da classe atual e a instância do seu JPanel, tal que é essencial para levar a cabo a lógica criada para a troca entre idiomas, conforme exemplo abaixo extraído da classe EcraLogin:

```

this.empresaObjeto.setEcraAtivo(this);

this.empresaObjeto.setPainelAtivo(panelLogin);

```

5.4 Migração entre ecrãs

Quanto à troca entre um ecrã e outro, foi resolvida da seguinte forma:

- A migração entre um ecrã e outro é feita através dos JButtons e seus `actionListeners`.
- Uma vez acionado algum botão que leva o utilizador para outro ecrã, primeiramente, procede-se à remoção da componente atual da JPanel do JFrame através do método `.remove()` disponível para JFrame. Tal garante que, antes de um novo JPanel ser adicionado à JFrame, o anterior foi removido, evitando o empilhar de JPanel's que já não se encontram em utilização.
- Feito isso, de acordo com o botão acionado pelo utilizador, será carregada o respetivo ecrã.

5.5 Classe FrameUnica:

A classe `FrameUnica` é a classe mais importante de toda a GUI pois, tal como já foi supramencionado, é nela que está definida a instância da única JFrame usada em todo o projeto. Além disso, é no seu construtor que são instanciados a maioria dos objetos que serão cruciais no decorrer de toda a interface gráfica. Em suma, é nesta classe que são criados e instanciados:

- Um objeto JFrame.
- Um objeto do tipo `TrocaDeIdioma` que dará acesso ao método `getIdiomasLista()`, que permitirá aceder aos textos em português/inglês que deverão ser alocados nos componentes de classe/ecrã do programa.

- Um objeto do tipo ValidadorGeral que dará acesso aos métodos necessários para validar os dados inseridos pelo utilizador no decorrer do programa.

É também no construtor que são criados os JButtons da troca de idiomas, a JLabel que recebe o logotipo do projeto (ao centro) e o JButton que permite ao utilizador encerrar o programa (ícone à direita):



Figura 3 – JFrame comum a todos ecrãs de interface com o utilizador

Foi definido um ActionListener para o componente JButton denominado como “botaoSair”, que garante que quando o utilizador clique na imagem referente ao botão, seja chamado o método que garante a escrita dos dados gerais do programa no ficheiro de objetos e também, o método que garante a gravação do último idioma escolhido pelo utilizador, permitindo que aquando da reabertura do programa, este apresente o último idioma escolhido¹².

```
botaoSair.addActionListener(eventoSair -> {
    empresaObjeto.escreverNoFicheiroGeralASaida();
    empresaObjeto.atualizarFicheiroDeConfiguracao();
    objetoJframe.dispose();
    System.exit(0);
});
```

Esta classe também instancia a primeira classe/ecrã de interface com o utilizador, iniciando assim a real interação entre o utilizador e o programa.

```
EcranInicial ecranInicial = new EcranInicial(empresaObjeto, objetoJframe, trocaDeIdiomaObjeto, validadorGeral, avisosPopUp);
```

Já a própria classe FrameUnica é instanciada uma única vez na classe Program, que é a classe que contém o método main.

```
public class Program {
    public static void main(String[] args) {
        EmpresaAorWords empresaObjeto = new EmpresaAorWords();
        FrameUnica frameUnica = new FrameUnica(empresaObjeto, empresaObjeto.getFicheiroDeTexto());
    }
}
```

É também em FrameUnica que temos o método actionPerformed(ActionEvent eventoBotaoIdioma), que trata da lógica dos botões de troca entre idiomas e o método recarregarEcranAtivoComNovoIdioma(), que garante que a página de interação com o utilizador seja novamente carregada com o novo idioma, conforme já detalhado anteriormente na parte deste relatório referente a lógica implementada para a troca de idiomas do sistema.

¹² <https://www.guj.com.br/t/resolvido-programar-botao-sair/81702> (acedido em 21/12/2021)

5.6 Classe AvisosPopUp:

Criada para melhor organizar a estrutura do programa e evitar repetição desnecessária de código.

Contém os métodos que implementam o código padrão aos diversos PopUps de aviso e informações que interagem com o utilizador ao decorrer do programa.

Possui apenas dois atributos:

1. `empresaObjeto`: referência a instância de `EmpresaAorWords` criada na classe `Program` (detentora do método `main`), que vai permitir aceder ao método `getter` do atributo booleano que guarda o valor referente ao idioma.
2. `trocaDeIdiomaObjeto`: referência a instância de `TrocaDeIdioma` criada na classe `FrameUnica`, que vai permitir que a `PopUp` seja exibida de acordo com o idioma escolhido.

Esta classe possui 5 métodos, que permitem criar 5 tipos de Pop-ups diferentes:

- `mensagemDialogo`: cria um Pop-up de aviso. Recebe como atributos a mensagem a mostrar e o nome da janela de Pop-up.
- `mensagemInformacao`: cria um Pop-up de informação. Recebe como atributo a mensagem a mostrar.
- `popUpSimOuNao`: cria um Pop-up com as informações do contrato de anúncio para perceber se o Publicitário ou o Intermediário pretende efetuar o contrato. Recebe como atributo a frase selecionada, o número de exibições contratadas, o nome da aplicação contratada, o valor unitário, e o custo total do anúncio.
- `inputDialog`: cria um Pop-up para o utilizador colocar o valor que pretende levantar ou depositar na sua conta corrente. Recebe como atributo a mensagem a mostrar.
- `popUpSimOuNaoSimples`: cria um Pop-up para perguntar ao utilizador se tem a certeza que pretende eliminar a sua conta `AoRWords`. Recebe como atributo a mensagem e o nome da janela de Pop-up.

É instanciada uma única vez em `FrameUnica`, e esta instância é passada como parâmetro para as classes da GUI.

5.7 Classe ValidadorGeral:

Outra classe criada para organizar melhor a estrutura do programa e evitar repetição desnecessária de código.

Contém os cinco métodos que implementam as expressões regulares (Regex) necessárias para validar os dados inseridos pelo utilizador em diversos pontos do programa. Todos os métodos seguem a mesma estrutura, caso os dados que são enviados como parâmetros para o método passem no validador, o método retorna uma variável boolean com o valor *true*, caso contrário, retorna com o valor *false*.

- `validarEmail`: verifica se o e-mail introduzido pelo intermediário quando está a adicionar um novo utilizador possui letras e/ou números, o `@` e é seguido de letras e /ou números.

- `validarNome`: verifica se o nome introduzido pelo intermediário quando está a adicionar um novo utilizador é constituído por duas palavras apenas com letras.
- `validarTelefone`: verifica se o número de telefone introduzido pelo intermediário quando está a adicionar um novo utilizador é constituído por nove números de 0 a 9.
- `validarValorUnitario`: verifica se o valor introduzido corresponde a um valor válido em euros.
- `validarNumExibicoes`: verifica se o valor introduzido corresponde a um valor inteiro.

É instanciada uma única vez em `FrameUnica`, e esta instância é passada como parâmetro para as classes da GUI.

Capítulo 6.

Da implementação de cada ecrã de interface com o utilizador:

Para além, da implementação base destacada na unidade 5.1 deste relatório, cabe destacar as peculiaridades de cada classe que compõe a GUI do programa.

6.1 Classe `EcraInicial`:

Corresponde ao primeiro ecrã que surge quando o utilizador inicia o programa. É comum a todos os utilizadores, tal como é possível evidenciar pelos esquemas de ecrã do Intermediário (item 8.1), do Publicitário (8.2) e do Programador (8.3) do Capítulo 8.

Segue o conceito de classes aninhadas (*Nested Classes*), ou seja, cada classe possui outras classes dentro de si, classes estas que correspondem às componentes gráficas desta classe.

Segue a implementação padrão explicitada no capítulo 5 deste relatório, porém com as peculiaridades destacadas abaixo.

É uma das classes mais simples do programa. Possui apenas um componente `JLabel` e dois componentes `JButtons`, que permitem ao utilizador ou fazer login no programa ou visualizar as informações necessárias para se registar ao mesmo.

6.2 Classe `EcraSignUp`:

Corresponde ao ecrã no qual o utilizador pode obter as informações para registar-se na aplicação. É comum a todos os utilizadores, tal como é possível evidenciar pelos esquemas de ecrã do Intermediário (item 8.1), do Publicitário (8.2) e do Programador (8.3) do Capítulo 8.

A Classe foi construída tendo em conta o conceito de classes aninhadas (*Nested Classes*), ou seja, cada classe possui outras classes dentro de si, classes estas que correspondem às suas componentes gráficas.

Segue a implementação padrão explicitada no capítulo 5 deste relatório, porém com as peculiaridades destacadas abaixo, ou seja: implementa um ecrã simples composto apenas de dois JLabels:

1. O primeiro faz parte do visual do programa.
2. O segundo apresenta as informações de como se registar na aplicação.

Implementa ainda um JButton, que segue o descrito na unidade 5.4 do capítulo 5 deste relatório e recarrega o ecrã inicial.

6.3 Classe EcraAddUtilizador:

Corresponde ao ecrã que permite aos intermediários registarem novos utilizadores. Este ecrã existe apenas na versão do programa destinada a Intermediários, tal como é possível evidenciar pelo esquema de ecrã do Intermediário (item 8.1) do Capítulo 8.

Segue o conceito de classes aninhadas (*Nested Classes*), ou seja, cada classe possui outras classes dentro de si, classes estas que correspondem às componentes gráficas desta classe.

Segue a implementação padrão explicitada no capítulo 5 deste relatório, porém com algumas peculiaridades de forma a implementar a lógica de criação de novos utilizadores e novas contas corrente associadas ao utilizador criado (caso o utilizador criado seja do tipo publicitário ou programador).

Contém os seguintes componentes:

- Cinco JTextFields que irão receber os dados do utilizador a ser registado, tais como : nome, morada, telefone, e-mail e palavra-passe.
- Uma JComboBox que apresenta os três tipos possíveis de utilizador, para que o intermediário escolha o tipo de utilizador que pretende.
- Dois JButtons:
 1. O primeiro corresponde a ação de “criar” um novo utilizador: o seu ActionListener irá chamar os métodos de validação de dados constantes da classe ValidadorGeral para validar os dados inseridos e que foram acima explorados na unidade 5.7 do capítulo 5.

Nos casos em que qualquer um dos dados for inválido, ocorre a chamada a um dos métodos da classe AvisosPopUp (unidade 5.6) a fim de exibir a informação ao intermediário. A(s) JTextField(s) que contenha(m) informação inválida verão os dados nelas limpos através do método setText(“”).

Caso todos os dados introduzidos sejam válidos, é verificado o tipo de utilizador escolhido na JComboBox, e consoante esta escolha será instanciado/criado um novo utilizador e de imediato já é logo criada uma nova conta-corrente, que recebe como parâmetro este utilizador, caso este seja um publicitário ou um programador:

```

if (validador) {

    empresaObjeto.atualizarIterador();

    if (novoUtilizador.getSelectedItem() == vetorTipoUtilizadores[0]) { //publicitário

        utilizadorAtributo = new Publicitario(nome, morada, telefone, email, password);
        contaAtributo = new Conta(utilizadorAtributo);

        avisosPopUp.mensagemInformacao(77);

    } else if (novoUtilizador.getSelectedItem() == vetorTipoUtilizadores[1]) { //programador

        utilizadorAtributo = new Programador(nome, morada, telefone, email, password);
        contaAtributo = new Conta(utilizadorAtributo);

        avisosPopUp.mensagemInformacao(78);

    } else if (novoUtilizador.getSelectedItem() == vetorTipoUtilizadores[2]) { //intermediario

        utilizadorAtributo = new Intermediario (nome, morada, telefone, email, password);
        avisosPopUp.mensagemInformacao(79); //não possuem conta

    }

}

```

Por fim, estes objetos serão adicionados às respetivas ArrayLists através de uma chamada dos métodos adicionarUtilizadorNaArrayList() e adicionarContaNaArrayList() que irá levar como parâmetro o objeto a ser inserido na lista (ver item 16 e 17, respetivamente, da unidade 3.2 do capítulo 3).

2. O segundo corresponde a opção “voltar” que segue o descrito na unidade 5.4 do capítulo 5 deste relatório e recarrega o ecrã respetivo ao menu de opções consoante o tipo de utilizador.

6.4 Classe EcrãConsultarUtilizadores:

Corresponde ao ecrã que permite os intermediários visualizarem numa tabela e ordenarem a seu gosto as informações de todos os utilizadores registados. Este ecrã surge apenas na versão do programa que se destina a utilizadores Intermediários, tal como é possível evidenciar pelos esquemas de ecrã do Intermediário (item 8.1) do Capítulo 8.

Este ecrã permite aceder às seguintes informações dos utilizadores registados:

- Nome
- Morada
- Telefone
- E-mail
- Id de utilizador
- Tipo de utilizador: consoante seja programador, publicitário, intermediário ou o fundador.

Segue o conceito de classes aninhadas (*Nested Classes*), ou seja, cada classe possui outras classes dentro de si, classes estas que correspondem às componentes gráficas desta classe.

Segue a implementação padrão explicitada no capítulo 5 deste relatório, porém com as seguintes características:

- JLabel com texto a informar ao utilizador as orientações de como ordenar os dados da tabela, caso deseje.
- JButton "voltar" que segue o descrito no item 5.4 do capítulo 5 deste relatório e recarrega o ecrã respetivo ao menu de opções consoante o tipo de utilizador.
- JTable construído na forma abaixo descrita.

6.4.1 Sobre a construção padrão do componente JTable:

Este ecrã possui um componente JTable que segue o padrão de todas as tabelas utilizadas no programa¹³.

A classe aninhada respetiva à JTable, está construída da seguinte forma:

1. No método construtor está a chamada para o método construirCarregarDadosDaJtable() e também a funcionalidade que determina a localização da tabela no ecrã de interface com o utilizador:

```
public JTableConsultarUtilizadores() {
    construirCarregarDadosDaJtable();
    jTableConsultarUtilizadores.setBounds(50, 330, 700, 300);
}
```

2. A seguir ao construtor, encontra-se o método definirScroll() que vai instanciar um objeto JScrollPane, enviando a JTable como parâmetro ao construtor deste componente e define que a localização do componente JScrollPane no ecrã da GUI (que deverá ser a mesma localização definida para a tabela). Conforme o capítulo 5 unidade 5.1 deste relatório, o JPanel que receberá a tabela se utiliza de um *layout* null¹⁴

```
public JScrollPane definirScroll() {
    JScrollPane jScrollPane = new JScrollPane(jTableConsultarUtilizadores);
    jScrollPane.setBounds(50, 330, 700, 300);
    return jScrollPane;
}
```

3. De seguida, é implementado o método construirCarregarDadosDaJtable() que, por sua vez:

- É responsável por resgatar todos os valores que se encontram em memória que devem ser inseridos na tabela.
- Para a construção da tabela, fez uso do modelo padrão DefaultTableModel.
- A utilização do DefaultTableModel decorre da seguinte maneira:

¹³ [How to Use Tables \(The Java™ Tutorials > Creating a GUI With Swing > Using Swing Components\) \(oracle.com\)](https://www.oracle.com/pt/java/javatutorial/creating-a-gui-with-swing/using-swing-components/) (acedido em 12/12/2021)

¹⁴ <https://www.ti-enxame.com/pt/java/como-adicionar-jtable-no-jpanel-com-layout-nulo/971321042/> (acedido em 12/12/2021)

3.1. Primeiro através do método `setModel()` é inserido na respetiva `JTable` o construtor de `DefaultTableModel` a ser utilizado, neste caso o escolhido foi:

```
jTableConsultarUtilizadores.setModel(new DefaultTableModel(new Object[][]{{}}, vetorColunasTabela));
```

3.2. Depois, foi criado um atributo do tipo `DefaultTableModel` que recebe a referência da `JTable` e do modelo criado anteriormente.

```
DefaultTableModel model = (DefaultTableModel) jTableConsultarUtilizadores.getModel();
```

3.3 Para receber os nomes das colunas, foi criado um vetor do tipo `Object` denominado por: `Object[] vetorColunasTabela`. Para alimentar este vetor com os nomes de cada coluna, é utilizada a estrutura condicional que verifica o idioma em uso no sistema e insere os respetivos textos. O tamanho deste vetor é definido pela quantidade de textos inseridos que servirão de nomes às colunas da tabela.

```
Object[] vetorColunasTabela;
```

```
if (empresaObjeto.isIdiomaPortugues()) {
```

```
    vetorColunasTabela = new Object[]{trocaDeIdiomaObjeto.getIdiomasLista().get(30)[1],
    trocaDeIdiomaObjeto.getIdiomasLista().get(31)[1],
    trocaDeIdiomaObjeto.getIdiomasLista().get(32)[1], trocaDeIdiomaObjeto.getIdiomasLista().get(8)[1],
    trocaDeIdiomaObjeto.getIdiomasLista().get(140)[1], trocaDeIdiomaObjeto.getIdiomasLista().get(144)[1]};
```

```
} else {
```

```
    vetorColunasTabela = new Object[]{trocaDeIdiomaObjeto.getIdiomasLista().get(30)[2],
    trocaDeIdiomaObjeto.getIdiomasLista().get(31)[2],
    trocaDeIdiomaObjeto.getIdiomasLista().get(32)[2], trocaDeIdiomaObjeto.getIdiomasLista().get(8)[2],
    trocaDeIdiomaObjeto.getIdiomasLista().get(140)[2], trocaDeIdiomaObjeto.getIdiomasLista().get(144)[2]};
}
```

3.4. Depois, é instanciado um novo vetor do tipo `Object` denominado `Object[] dadosLinha`, que receberá os valores a serem exibidos em cada linha da tabela a ser visualizada pelo utilizador.

O tamanho deste vetor é definido por "`vetorColunasTabela.length`", uma vez que a quantidade de valores exibidos por linha é igual a quantidade de colunas existentes.

```
Object[] dadosLinha = new Object[vetorColunasTabela.length];
```

3.5. Para inserir os dados na tabela, são utilizados ciclos `forEach`, que percorrerão a(s) `ArrayList(s)` necessárias para buscar os dados que irão ser exibidos.

Cumpre destacar que o acesso a todas as `ArrayLists` do programa ocorre através de uma instância da classe `EmpresaAorWords` declarada na classe `Program` (detentora do método `main`), que é enviada como parâmetro a todas as classes da GUI.

3.6. Dentro do ciclo `forEach`, até se esgotarem os dados a inserir, cada posição do vetor `dadosLinha` recebe seu respetivo valor a ser exibido, e por fim, este vetor é inserido no modelo da tabela com o comando "`modeloTabela.addRow(dadosLinha)`".

```
for (Utilizador utilizador : empresaObjeto.getListaUtilizadores()) {
```

```
    dadosLinha[0] = utilizador.getNome();
```

```

dadosLinha[1] = utilizador.getMorada();
dadosLinha[2] = utilizador.getTelefone();
dadosLinha[3] = utilizador.getEmail();
dadosLinha[4] = utilizador.getIdUtilizador();

if (empresaObjeto.isIdiomaPortugues()) {

    if (utilizador instanceof Publicitario) {
        dadosLinha[5] = trocaDeIdiomaObjeto.getIdiomasLista().get(24)[1];

    } else if (utilizador instanceof Programador) {
        dadosLinha[5] = trocaDeIdiomaObjeto.getIdiomasLista().get(25)[1];

    } else if (utilizador.getEmail().equals(empresaObjeto.getEmailFundador()) && utilizador instanceof Intermediario) {

        dadosLinha[5] = trocaDeIdiomaObjeto.getIdiomasLista().get(145)[1];

    } else if (utilizador instanceof Intermediario) {
        dadosLinha[5] = trocaDeIdiomaObjeto.getIdiomasLista().get(26)[1];
    }
} else {

    if (utilizador instanceof Publicitario) {
        dadosLinha[5] = trocaDeIdiomaObjeto.getIdiomasLista().get(24)[2];

    } else if (utilizador instanceof Programador) {
        dadosLinha[5] = trocaDeIdiomaObjeto.getIdiomasLista().get(25)[2];

    } else if (utilizador.getEmail().equals(empresaObjeto.getEmailFundador()) && utilizador instanceof Intermediario) {

        dadosLinha[5] = trocaDeIdiomaObjeto.getIdiomasLista().get(145)[2];

    } else if (utilizador instanceof Intermediario) {
        dadosLinha[5] = trocaDeIdiomaObjeto.getIdiomasLista().get(26)[2];
    }
}

model.addRow(dadosLinha); //adiciona o vetor dadosLinha na tabela
}

```

3.7 Componente que permite ordenar em valor crescente/decrescente qualquer coluna da tabela:

3.7.1 Foi ainda instanciado no mesmo método, um objeto do tipo TableRowSorter<TableModel>, com o intuito de permitir ao utilizador ordenar à sua vontade os dados da tabela. Para isso, basta que o utilizador clique no cabeçalho de uma das quaisquer colunas para que toda a tabela se ordene de a-z ou de z-a, ou ainda por ordem crescente ou decrescente, mediante a coluna escolhida.

```

TableRowSorter<TableModel> sorter = new TableRowSorter<>(model);

```

3.7.2 Para de fato a ordenação funcionar com êxito, a classe TableRowSorter recebeu como parâmetro o modelo DefaultTableModel criado. E logo em seguida, na JTable, foi adicionado este RowSorter através do método setRowSorter().

`jTableConsultarUtilizadores.setRowSorter(sorter);`

3.8 Ademais, para garantir a segurança dos dados inseridos na tabela e impedir que o utilizador consiga editar a mesma, foi inserido na tabela o comando: `setDefaultEditor(Object.class, null)`.

`jTableConsultarUtilizadores.setDefaultEditor(Object.class, null);`

3.9 Por fim, é importante referir que o que é adicionado ao JPanel respetivo ao ecrã é uma chamada para o método `definirScroll()` destacado no item 2 acima, e com isso o JScrollPane criado leva consigo a tabela e seus dados a exibir e o RowSorter inserido na tabela.

6.5 Classe EcraLogin:

Corresponde ao ecrã que permite ao utilizador fazer login no sistema. É comum a todos os utilizadores, tal como é possível evidenciar pelos esquemas de ecrã do Intermediário (item 8.1), do Publicitário (8.2) e do Programador (8.3) do Capítulo 8.

Segue o conceito de classes aninhadas (*Nested Classes*), ou seja, cada classe possui outras classes dentro de si, classes estas que correspondem às componentes gráficas desta classe.

Segue a implementação padrão explicitada no capítulo 5 deste relatório, mas com as seguintes características que a distingue:

- Um JButton que permite ao utilizador aceder a informações sobre como proceder, caso tenha esquecido o seu e-mail e/ou password.
- Uma JTextField para ser introduzido o e-mail.
- Uma JPasswordField para ser introduzida a password.
- Método `validarDadosDeLogin()`: verifica a existência de um utilizador com os dados inseridos, e em caso de dados inválidos, chama um dos métodos da classe `AvisosPopUp` para mostrar ao utilizador uma PopUp de aviso.
- Método `limparTextFields()`: usado quando os dados estão incorretos, permite limpar as TextFields para que o utilizador insira novos dados.

Para além do acima descrito contém um JButton com a denominação “entrar” que segue o descrito na unidade 5.4 do capítulo 5 deste relatório e carrega o ecrã respetivo ao menu de opções consoante o tipo de utilizador.

6.6 Classes que correspondem ao Menu de opções disponível para cada tipo de utilizador:

Correspondem aos ecrãs de menu de opções, consoante o tipo de utilizador.

Segue o conceito de classes aninhadas (*Nested Classes*), ou seja, cada classe possui outras classes dentro de si, classes estas que correspondem às componentes gráficas desta classe.

Segue a implementação padrão explicitada no capítulo 5 deste relatório, porém, é relevante destacar que, de forma a implementar os menus dos utilizadores, o programa conta com três classes:

1. EcraIntermediario (ver item 8.1 do Capítulo 8),
2. EcraPublicitario (ver item 8.2 do Capítulo 8),
3. EcraProgramador (ver item 8.3 do Capítulo 8).

Todas as três classes seguem o mesmo design composto de JButtons que permitem ao utilizador ativo aceder às funcionalidades disponíveis, consoante seja programador, publicitário ou intermediário.

Cada JButton segue o descrito na unidade 5.4 do capítulo 5 deste relatório e irá carregar o ecrã correspondente ao botão clicado pelo utilizador.

Foram implementadas nestes ecrãs diversas chamadas a métodos da classe AvisosPopUp, no intuito de controlar algumas ações dos utilizadores, como por exemplo, informar ao publicitário com saldo zero de que não é possível aceder ao ecrã de adicionar anúncios, ou informar ao programador que ainda não criou aplicações que não pode editar aplicações (porque estas, de facto, ainda não existem).

6.7 Classe EcraConfiguracoes

Corresponde ao ecrã que permite aos utilizadores visualizarem e/ou editarem os seus dados pessoais. Este ecrã também permite que todos os utilizadores façam log-out da sua conta ao clicar no botão designado “sair”. É comum a todos os utilizadores, todavia, apresenta algumas diferenças que serão elencadas abaixo (número 5 deste item) e que são passíveis de verificação nos esquemas de ecrã do Intermediário (item 8.1), do Publicitário (8.2) e do Programador (8.3) do Capítulo 8.

Caso o utilizador ativo seja um publicitário ou um programador, haverá ainda a funcionalidade de excluir completamente sua conta, desde que não exista saldo a levantar na sua conta corrente. A presente classe segue o conceito das classes aninhadas acima explorado e que leva a presente classe possua classes dentro de si, estas que correspondem às componentes gráficas da mesma.

Esta classe tem em conta a implementação padrão explicitada no capítulo 5 deste relatório, porém com algumas particularidades no que toca a alguns elementos como:

1. Quatro JTextFields, previamente preenchidas com os dados de nome, morada, telefone e palavra-passe do utilizador ativo no sistema, no intuito de melhorar a usabilidade do programa para o utilizador, que somente precisa efetuar alterações aos dados preenchidos que se encontram errados.

Cabe destacar que a funcionalidade de trocar a palavra-passe foi inserida no programa com o objetivo de promover uma maior segurança da conta-corrente dos publicitários e programadores uma vez que, devido ao facto de somente utilizadores intermediários poderem criar novos utilizadores, estes ficam a conhecer a palavra-chave dos utilizadores que criarem.

Desta forma, os programadores e publicitários podem mudar sua palavra-chave de acesso logo após obterem os dados do registo da sua conta para poderem, assim, restringir o acesso à sua conta-corrente.

2. JButton “sair” que segue o descrito no item 5.4 do capítulo 5 deste relatório e recarrega o ecrã inicial de interface com o utilizador, ou seja, faz log-out a conta.
3. JButton “atualizar” que em seu ActionListener faz uso de estruturas condicionais e métodos da classe ValidadorGeral, para validar os novos dados inseridos pelo utilizador.

Caso todos os dados sejam validados corretamente, os novos dados são inseridos nos dados do utilizador ativo através dos respetivos métodos setter, declarados na classe Utilizador.

Para melhor entendimento, segue abaixo um breve excerto do código em questão:

```
else if (eventosEcrãConfiguracoes.getSource() == atualizar) {
    String nome = nomeTxt.getText(), morada = moradaTxt.getText(), telefone = telefoneTxt.getText(), password =
passwordTxt.getText();
    boolean validador = false;

    boolean validarNome = validadorGeral.validarNome(nome);
    if (!validarNome) {
        avisosPopUp.mensagemDialogo(55, 56);
        nomeTxt.setText(empresaObjeto.getUtilizadorAtivo().getNome());
    }
    boolean validarTelefone = validadorGeral.validarTelefone(telefone);

    if (!validarTelefone) {
        avisosPopUp.mensagemDialogo(27, 28);
        telefoneTxt.setText(empresaObjeto.getUtilizadorAtivo().getTelefone());
    }

    if (morada.equals("")){
        avisosPopUp.mensagemInformacao(142);
        moradaTxt.setText(empresaObjeto.getUtilizadorAtivo().getMorada());
    }

    if (password.equals("")){
        avisosPopUp.mensagemInformacao(142);
        passwordTxt.setText(empresaObjeto.getUtilizadorAtivo().getPassword());
    }

    if (validarNome && validarTelefone && !morada.equals("")&&!password.equals("")) {
        validador = true;
    }

    if (validador ) {

        empresaObjeto.getUtilizadorAtivo().setNome(nome);
        empresaObjeto.getUtilizadorAtivo().setMorada(morada);
        empresaObjeto.getUtilizadorAtivo().setTelefone(telefone);
        empresaObjeto.getUtilizadorAtivo().setPassword(password);

        if (empresaObjeto.getUtilizadorAtivo().getEmail().equals(empresaObjeto.getEmailFundador())) {
            empresaObjeto.atualizarDadosFundador();
        }

        avisosPopUp.mensagemInformacao(60);
    }
}
```

4. JButton “voltar” que segue o descrito na unidade 5.4 do capítulo 5 deste relatório e recarrega o ecrã respetivo ao menu de opções consoante o tipo de utilizador.
5. Para os utilizadores que são instâncias da classe Programador ou da classe Publicitario, há ainda um JButton “apagar conta”, que em seu ActionListener verifica a instância do utilizador ativo e vai também fazer uso de ciclos forEach para encontrar a conta corrente deste utilizador e verificar o saldo.

Desta forma, foi possível controlar que somente programadores ou publicitários sem saldo que se encontrassem com a sessão iniciada no programa AoRWords poderiam eliminar de forma permanente a sua conta da aplicação.

Caso o utilizador queira excluir completamente sua conta mas tenha saldo a levantar, é chamado um dos métodos da classe AvisosPopUp para informar o utilizador desse facto. Neste caso, o utilizador pode levantar todo o valor disponível e depois retornar ao ecrã de configurações e excluir a sua conta.

```
} else if (eventosEcraConfiguracoes.getSource() == apagarConta) {  
  
    if (empresaObjeto.getUtilizadorAtivo() instanceof  
Programador || empresaObjeto.getUtilizadorAtivo() instanceof Publicitario){  
        for (Conta conta: empresaObjeto.getListaContas()){  
            if (conta.getUser().equals(empresaObjeto.getUtilizadorAtivo())){  
                if (conta.getSaldo() > 0.0){  
                    avisosPopUp.mensagemInformacao(146);  
                    break;  
                } else {  
                    mostrarPopUpCancelarConta();  
                }  
            }  
        }  
    } else {  
        mostrarPopUpCancelarConta();  
    }  
}
```

6.8 Classe EcraAddFrase

Corresponde ao ecrã que permite aos publicitários ou aos intermediários em nome dos publicitários, adicionar frases à ArrayList de frases daquele determinado publicitário, permitindo que este utilizador possa aceder às suas frases registadas quando quiser, bem como criar anúncios de forma otimizada e simples. Em suma, é comum aos utilizadores do tipo Publicitários e Intermediários, no entanto com uma diferença relativamente ao número 3 do presente item; diferença esta que pode ser evidenciada pela análise do esquema de ecrã do Intermediário (item 8.1) e do Publicitário (8.2), que se encontram no Capítulo 8.

Esta classe contém classes aninhadas (*Nested Classes*), isto é, possui outras classes dentro de si, classes estas que correspondem às componentes gráficas desta classe principal. A presente classe vai ainda seguir a implementação padrão explicitada no capítulo 5 deste relatório, todavia, com algumas diferenças.

É importante referir que, de forma a não criar duas classes para adicionar frase (uma para o publicitário e outra para o intermediário), recorreu-se a uma estrutura condicional para que se verifique a instância do utilizador ativo e, adicionar a componente à JPanel, ou não.

A classe EcraAddFrase dispõe da seguinte estrutura:

1. JLabel com dados informativos, com o objetivo de auxiliar a interação do utilizador com o ecrã.
2. JTextField que irá receber a frase a adicionar à lista de frases do publicitário.
3. Caso o utilizador ativo seja um intermediário surgirá, ainda, uma segunda JTextField onde deverá ser inserido o e-mail do publicitário a quem irá pertencer a frase registada, conforme demonstra o trecho abaixo retirado do construtor da classe EcraAddFrase:

```
if (empresaObjeto.getUtilizadorAtivo() instanceof Intermediario) {  
    panelAddFrase.add(emailPublicitario);  
}
```

4. Dois JButtons :
 - a. O primeiro botão corresponde a ação de adicionar a frase e tem em seu actionListener:
 - O código para verificação da instância do utilizador ativo.
 - Os ciclos forEach que irão buscar o publicitário a quem deve pertencer a frase, no caso do utilizador ativo ser um intermediário.
 - Algumas chamadas aos métodos da classe AvisoPopUp, que implementam PopUps de aviso ao utilizador, caso este clique no botão de adicionar sem antes inserir um texto na JTextField ou um aviso informando que a frase foi inserida com sucesso.
 - A criação de uma instância para a frase inserida.
 - A chamada para o método adicionarFrases() declarado na classe Publicitario, este método recebe como parâmetro a frase criada e adiciona a mesma na ArrayList de frases daquele determinado publicitário.
 - b. O segundo botão a que corresponde a opção “voltar” que segue o descrito no item 5.4 do capítulo 5 deste relatório e recarrega o ecrã respetivo ao menu de opções consoante o tipo de utilizador.

É de relevo destacar que nesta classe é o único local do programa onde são criadas as instâncias correspondentes a cada uma das frases registadas na aplicação deste projeto.

6.9 Classe EcraAddAplicacao

Corresponde ao ecrã que permite aos programadores ou aos intermediários em nome dos programadores, adicionar aplicações à ArrayList de aplicações daquele determinado programador, permitindo que este utilizador possa aceder às suas aplicações registadas quando quiser e também que estas aplicações fiquem disponibilizadas para anúncios. Resumindo, é comum a todos os utilizadores do tipo Programador e Intermediário, possuindo a mesma diferença elencada no item anterior (6.8); tal que é possível verificar pelos esquemas de ecrã do Intermediário (item 8.1) e do Programador (8.3) do Capítulo 8.

Esta classe implementa o conceito de classes aninhadas (*Nested Classes*), ou seja, possui outras classes dentro de si, estas que correspondem às componentes gráficas desta classe. Segue a implementação padrão explicitada no capítulo 5 deste relatório, porém com as peculiaridades destacadas abaixo.

Tal como a classe referida na unidade anterior (6.8), a presente classe detém alguns componentes que, dependem de uma estrutura condicional que verifique a instância do utilizador ativo, de forma a perceber se a componente deve ser, ou não, adicionada à JPanel da classe. Graças à semelhança de implementação que a presente classe e a anterior partilham, releva referir que a atual classe é seguidora do mesmo conceito de implementação citado anteriormente nos itens 1 à 4 da unidade 6.8 deste relatório, cabendo, todavia, destacar:

1. No item supracitado, onde se lê “frase” deve perceber “aplicação”, para melhor percepção da implementação deste presente EcraAddAplicacao – tal que se aplica para a JLabel como para a JTextField mencionada no item 1 e 2 da unidade anterior.
2. A classe EcraAddAplicacao possui ainda os seguintes acréscimos:
 - a. Uma JTextField que irá receber a descrição da aplicação inserida.
 - b. Uma segunda JTextField que irá receber o valor cobrado por exibição naquela aplicação.

Com exceção destes componentes citados no item 2 acima, este ecrã/classe tem a mesma implementação citada anteriormente.

Graças às semelhanças partilhadas entre a presente e a anterior classe, é igualmente reiterar que também nesta classe é o único local do programa onde são criadas as instâncias correspondentes a cada uma das aplicações registadas na aplicação deste projeto.

6.10 Classe EcraEditarFrases

Corresponde ao ecrã que permite aos publicitários editarem as suas frases anteriormente registadas. Somente Publicitários terão acesso a este ecrã, tal que é possível evidenciar pelo esquema de ecrã do Publicitário (8.2) que se encontra no Capítulo 8.

Segue o conceito de classes aninhadas (*Nested Classes*), ou seja, cada classe possui outras classes dentro de si, classes estas que correspondem às componentes gráficas desta classe.

Segue a implementação padrão explicitada no capítulo 5 deste relatório, porém com as peculiaridades destacadas abaixo.

Possui os seguintes componentes:

1. JLabel com informação, de forma a auxiliar a interação do utilizador com o ecrã.
2. Uma JTable que irá exibir ao publicitário, todas as suas frases já registadas. Esta tabela segue o mesmo padrão de implementação detalhado neste capítulo no item 6.4.1.
3. A tabela desta classe possui ainda, o seguinte acréscimo quanto a implementação padrão:

6.10.1 Do método capturarAppSelecionadaNaJtable():

Para ser possível ao utilizador com apenas um clique sobre uma linha da tabela, seleccionar o valor/informação que deseja, foram implementados os seguintes comandos:

1. Através do modelo criado para a tabela, foi adicionado a esta um `ListSelectionListener`.
2. Dentro deste `ListSelectionListener`, há a chamada dos métodos `getValueAt()` e `getSelectedRow()` que servirão para “capturar” o valor exibido na linha seleccionada e na coluna determinada na implementação do método. Para este caso foi escolhido o valor da primeira coluna (0) que se refere ao nome do texto.
3. Este valor é então transformado numa `String` e armazenado numa variável deste mesmo tipo, a fim de ser utilizado posteriormente.

```
public void capturarAppSelecionadaNaJtable() {  
  
    jTableEditorFrases.getSelectionModel().addListSelectionListener(event -> {  
  
        nomeFraseSelecionada = jTableEditorFrases.getValueAt(jTableEditorFrases.getSelectedRow(), 0).toString();  
  
        adicionarFraseSelecionadaNaTextField();  
    });  
}
```

4. A chamada deste método encontra-se dentro do construtor da classe externa principal.

É importante não esquecer que todas as tabelas do programa que necessitam da funcionalidade de “capturar” o valor seleccionado pelo utilizador, seguem este mesmo modelo, com algumas exceções que serão sempre demonstradas.

Finda a exposição do funcionamento do método de captura, resume-se à exposição dos componentes específicos da classe `EcraEditarFrases`, esta é constituída ainda por:

- Uma `JTextField`: para garantir pragmatismo ao utilizador, a frase seleccionada é exibida dentro desta `JTextField`, através de chamada para o método `adicionarFraseSelecionadaNaTextField()` que se encontra declarado dentro do método `capturarAppSelecionadaNaJtable()`. Com isso basta ao utilizador modificar o que deseja na frase, e depois clicar em “atualizar”.

```
public void adicionarFraseSelecionadaNaTextField(){  
  
    for (Frase frase : arrayDeFrasesAuxiliar) {  
  
        if (nomeFraseSelecionada.equals(frase.getFrase())) {  
            fraseAuxiliar = frase;  
            fraseTxt.setText(fraseAuxiliar.getFrase());  
        }  
    }  
}
```

- Duas JButtons que permitem os comandos:

1. “voltar” que segue o descrito na unidade 5.4 do capítulo 5 deste relatório e recarrega o ecrã respetivo ao menu de opções consoante o tipo de utilizador.
2. “atualizar”: cujo ActionListener contém os seguintes comandos:
 - a. Estrutura condicional que verifica se o utilizador, ao tentar editar sua frase, apagou completamente o texto, e se a JTextField estiver vazia, realiza uma chamada ao método constante da classe AvisosPopUp no intuito de avisar o utilizador deste fato
 - b. Estrutura condicional que verifica se foi selecionada alguma linha/texto da tabela, e caso não tenha sido, realiza uma chamada ao método constante da classe AvisosPopUp no intuito de pedir ao utilizador que escolha a frase a ser editada.
 - c. Se após as verificações acima, estiver tudo correto, procede a modificação do texto da frase, através do método setFrase(), declarado na classe Frase do programa.

6.11 Classe EcraEditarAplicacoes

Corresponde ao ecrã que permite aos programadores editarem qualquer uma das informações das suas aplicações anteriormente registadas. Somente programadores têm acesso a este ecrã, tal como é possível evidenciar pelo esquema de ecrã do Programador (8.3), no Capítulo 8.

É detentora de classes aninhadas (*Nested Classes*), ou seja, vai possuir outras classes dentro de si, classes estas que correspondem às componentes gráficas desta classe. A classe vai ainda seguir a implementação padrão referida no capítulo 5, contudo, com algumas particularidades específicas à atual classe:

- JLabel com dados informativos, de forma a ajudar o utilizador a interagir com o ecrã.
- Uma JTable que irá exibir ao programador, todas as suas aplicações já registadas. Esta tabela segue o mesmo padrão de implementação detalhado neste capítulo na unidade 6.4.1.
- A tabela desta classe possui ainda um acréscimo quanto a implementação padrão descrita no item 6.4.1 que é similar ao descrito neste capítulo no item 6.10.1, referente aos pormenores do método capturarAppSelecionadaNaJtable() e suas funcionalidades.

Quanto aos componentes específicos deste ecrã temos:

- Três JTextFields: para garantir praticidade da aplicação ao utilizador. As três componentes apresentam os dados de nome, descrição e valor unitário por exibição da aplicação selecionada através da chamada para o método adicionarDadosAplicacaoSelecionadaNasTextFields(). Para proceder à modificação das características da aplicação, basta ao programador alterar o que deseja na aplicação, e depois clicar em “atualizar”.
- Duas JButtons que permitem os comandos:

1. “voltar” que segue o descrito na unidade 5.4 do capítulo 5 deste relatório e recarrega o ecrã respetivo ao menu de opções consoante o tipo de utilizador.

2. “atualizar”: o `actionListener` deste botão contém os seguintes comandos:

a. Estrutura condicional que verifica se o utilizador, ao tentar editar sua aplicação, apagou completamente algum dos textos e, caso alguma `TextField` esteja vazia, realiza uma chamada ao método constante da classe `AvisosPopUp` no intuito de avisar o utilizador deste facto.

b. Estrutura condicional que verifica se foi selecionada alguma linha/texto da tabela, e caso não tenha sido, realiza uma chamada ao método constante da classe `AvisosPopUp` no intuito de pedir ao utilizador que escolha a frase a ser editada.

`@Override`

```
public void actionPerformed(ActionEvent eventosEcraEditarAplicacoes) {

    objetoJframe.remove(panelEditarApps);

    if (eventosEcraEditarAplicacoes.getSource() == voltar) {

        EcraProgramador ecraProgramador = new EcraProgramador(empresaObjeto, objetoJframe,
        trocaDeldiomaObjeto, validadorGeral, avisosPopUp);

    } else if (eventosEcraEditarAplicacoes.getSource() == atualizar) {

        if (nomeAppSelecionada.equals("")) {

            avisosPopUp.mensagemDialogo(69, 88);

        } else {

            String nome = nomeTxt.getText(), descricao = descricaoTxt.getText(), valor = valorUnitarioTxt.getText();

            if (nome.equals("") || descricao.equals("") || valor.equals("")) {

                avisosPopUp.mensagemDialogo(69, 88);

                EcraEditarAplicacoes ecraEditarAplicacoes = new EcraEditarAplicacoes(empresaObjeto,
                objetoJframe, trocaDeldiomaObjeto, validadorGeral, avisosPopUp);

            } else {

                boolean validador = validadorGeral.validarValorUnitario(valor);

                if (validador) { //se passar no validador
                    appAuxiliar.setNome(nome);
                    appAuxiliar.setDescricao(descricao);
                    double valorUnitarioDouble = Double.parseDouble(valor);
                    appAuxiliar.setValorUnitarioPorExibicao(valorUnitarioDouble);

                    avisosPopUp.mensagemInformacao(70);

                } else { //se não passar no validador

                    avisosPopUp.mensagemDialogo(51, 52);
                    valorUnitarioTxt.setText("");

                }

            }

        }

    }

}
```

c. Se após as verificações acima, o valor unitário digitado pelo utilizador não passar no método validador `validarValorUnitario`, realiza uma chamada ao método constante da classe `AvisosPopUp` no intuito de avisar o utilizador deste facto.

d. Se após as verificações acima, estiver tudo correto, procede a modificação das respetivas informações da aplicação, através dos métodos `setNome()`, `setDescricao()` e `setValorUnitarioPorExibicao()`, declarados na classe `Aplicacao` do programa.

6.12 Classe `EcraAddAnuncio`

Corresponde ao ecrã que permite aos publicitários ou aos intermediários em nome dos publicitários, criarem anúncios. É um ecrã comum a utilizadores do tipo `Publicitário` e `Intermediário`, tal como é possível evidenciar pelos esquemas de ecrã do `Intermediário` (item 8.1) e do `Publicitário` (8.2), no Capítulo 8.

A presente classe vai seguir o conceito de classes aninhadas (*Nested Classes*), ou seja, vai possuir outras classes dentro de si, classes estas que correspondem às componentes gráficas desta classe. Além disso, vai fazer uso da implementação padrão explicitada no capítulo 5 deste relatório, porém com algumas características diferenciadoras destacadas abaixo.

Possui os seguintes componentes:

1. Uma `JComboBox<String>` que irá exibir uma lista com frases, seguindo o seguinte conceito:
 - a. Se o utilizador ativo for um intermediário, a `ComboBox` irá exibir uma lista única de todas as frases registadas no sistema, desde que o publicitário dono daquela frase possua saldo na conta, bastando ao intermediário escolher a frase que precisa, de acordo com o pedido do publicitário que requereu a criação do anúncio.
 - b. Se o utilizador ativo for um publicitário, a `ComboBox` irá exibir a sua respetiva lista de frases.

Em relação ao presente ecrã, é fulcral referir que este foi construído de forma a que quando um intermediário fosse criar um novo anúncio, não surgiram frases cujo publicitário a quem pertencem não possua saldo na sua conta corrente; e, caso o utilizador ativo no programa fosse um publicitário, não poderia aceder ao `EcraAddAnuncio` caso não possua saldo na sua conta corrente.

2. Método `carregarDadosComboBox()`:
 - a. Utiliza de uma estrutura condicional para verificar a instância do utilizador ativo.
 - b. Recorre a ciclos `forEach` para percorrer as `ArrayLists` necessárias a fim de pesquisar as informações que serão exibidas. O acesso a todas as `ArrayLists` do programa é possível através de uma instância da classe `EmpresaAorWords` declarada na classe `Program` (detentora do método `main`), que é enviada como parâmetro a todas as classes da GUI.
 - c. No caso do utilizador ativo ser um intermediário, usa uma estrutura condicional para que somente sejam exibidas frases de publicitários que possuam saldo na sua conta corrente.
 - d. Adiciona as frases à `JComboBox<String>` através do método `addItem()`.

```
public void carregarDadosComboBox() {  
  
    if (empresaObjeto.getUtilizadorAtivo() instanceof Intermediario) {  
  
        for (Utilizador utilizador : empresaObjeto.getListaUtilizadores()) {
```

```

if (utilizador instanceof Publicitario) {

    for (Conta conta : empresaObjeto.getListaContas()) {
        if (conta.getUser().equals(utilizador)) {

            if (conta.getSaldo() > 0.0) {

                for (Frase frase : ((Publicitario) utilizador).getListaFrases()) {
                    comboBoxFrases.addItem(frase.getFrase());
                }
            }
        }
    }
} else if (empresaObjeto.getUtilizadorAtivo() instanceof Publicitario) {

    for (Frase frase : ((Publicitario) empresaObjeto.getUtilizadorAtivo()).getListaFrases()) {
        comboBoxFrases.addItem(frase.getFrase());
    }
}
}

```

3. Método `capturarAcaoComboBox()`: contém o método `addItemListener()` inserido na `JComboBox<String>` que através dos métodos `getStateChange()` e `getItem()`, irá “capturar” e armazenar a frase selecionada pelo utilizador numa variável `String` denominada `fraseSelecionada` para que não se perca a referência para a frase a ser introduzida no anúncio:

```

public void capturarAcaoComboBox() {

    comboBoxFrases.addItemListener(eventoComboBox -> {

        if (eventoComboBox.getStateChange() == ItemEvent.SELECTED) {
            fraseSelecionada = (String) eventoComboBox.getItem();
        }
    });
}

```

4. Uma `JTable` que irá exibir ao utilizador, todas as aplicações disponíveis para anúncios, ou seja, todas as aplicações registadas até o momento.

Nesta tabela é exibido ao utilizador: o nome do programador, o nome da sua aplicação, a descrição e o valor unitário cobrado por cada exibição da frase nesta aplicação.

Esta tabela segue o mesmo padrão de implementação detalhado neste capítulo no item 6.4.1 da unidade 6.4. Para permitir ao utilizador ordenar as aplicações disponíveis para anúncios por valor crescente/decrescente do seu custo unitário por exibição da frase publicitária, esta tabela conta ainda com o componente `TableRowSorter` já detalhado neste capítulo na unidade 6.4 item 6.4.1, item 3.7, bastando ao utilizador clicar no cabeçalho de qualquer coluna da tabela para a ordenar por ordem crescente ou decrescente ou de a-z ou z-a.

A arquitetura utilizada para construir esta tabela permitiu cumprir o requisito do projeto que dita que os programadores sem aplicações não devem ser visíveis para a criação de anúncios. Tal não era possível sem o método `construirCarregarDadosDaJtable()` (item 6.4.1 deste capítulo), mas com algumas alterações:

4.1 São usados ciclos `forEach` para percorrer a `ArrayList` de `Utilizadores` à procura de todos os utilizadores que são instâncias da classe `Programador`.

4.2 Estes programadores são armazenados numa `ArrayList` auxiliar do tipo `Programador`, denominada “`programadorLista`”:

```
for (Utilizador utilizador : empresaObjeto.getListaUtilizadores()) {  
    if (utilizador instanceof Programador) {  
        programadorAux = (Programador) utilizador;  
        programadorLista.add(programadorAux);  
    }  
}
```

4.3 De seguida, é utilizado um método `removeIf()` para eliminar da lista de programadores auxiliar, todos os programadores que não possuem aplicações registadas.

```
programadorLista.removeIf(elemento -> elemento.getListaAplicacoes().size() == 0);
```

Esta lista final, após o `removeIf()` será lista que vai permitir exibir na tabela as aplicações registadas no sistema, sem correr o risco da tabela apresentar dados de um programador sem aplicações.

5. A tabela desta classe possui, ainda, o método `capturarAppSelecioneada()` que segue o mesmo modelo de implementação acima descrito no item 6.10.1 deste relatório, porém, com o acréscimo de ciclos `forEach` que irão percorrer a `ArrayList` de programadores e a `ArrayList` de aplicações de cada um destes programadores, de forma a alcançarem a referência da aplicação respetiva ao nome selecionado na tabela. Após localizada a aplicação escolhida pelo utilizador, a sua referência é armazenada numa variável denominada `appAuxiliar` do tipo `Aplicacao` para ser utilizada na criação do anúncio.
6. Uma `JTextField` que pede ao utilizador que introduza a quantidade de exibições pretendidas para este contrato de anúncio. Destaque-se que, conforme o descrito no capítulo 2 deste relatório, cada contrato de anúncio corresponde a uma frase que será exibida numa aplicação, na quantidade de vezes que o utilizador preferir.

Para o utilizador publicitar a mesma frase noutras aplicações, deverá ser criado um novo contrato de anúncio. Para mais detalhes sobre a regra de negócio adotada no programa, consultar o capítulo 2.

7. Método `verificarAcaoTextFieldQuantidadeExibicoes()`.

Antes de mais, a fim de facilitar o entendimento, cumpre demonstrar o código do método:

```
public void verificarAcaoTextFieldQuantidadeExibicoes() {  
    quantidadeExibicoesTxt.addKeyListener(new KeyAdapter() {  
  
        @Override  
        public void keyReleased(KeyEvent evt) {
```

```

if (appAuxiliar == null) {
    avisosPopUp.mensagemDialogo(87, 88);
} else {
    String numExib = quantidadeExibicoesTxt.getText();
    boolean validarNumExibicoes = validadorGeral.validarNumExibicoes(numExib);

    if (validarNumExibicoes) {
        valorPorExibicao = appAuxiliar.getValorUnitarioPorExibicao();

        if (!quantidadeExibicoesTxt.getText().isEmpty()) {
            custoDouble = Integer.parseInt(quantidadeExibicoesTxt.getText()) * valorPorExibicao;

            BigDecimal bigObjeto = new BigDecimal(custoDouble).setScale(2, RoundingMode.UP);
            String custo = String.valueOf(bigObjeto);
            custoLabel.setText(custo + " €");
            remove(panelAddAnuncio);
        } else {
            custoLabel.setText("");
        }
    } else {
        avisosPopUp.mensagemDialogo(51, 52);
    }
}
}
});
}
}

```

Uma vez demonstrado como o método está implementado no programa, cumpre então detalhar suas funcionalidades:

7.1 Em primeiro lugar, cria um *listener* de ação para a JTextField mencionada no item 6 acima, utilizando o método addKeyListener() que recebe como parâmetro uma instância da classe abstrata KeyAdapter.

7.2 A classe KeyAdapter, por sua vez, implementa a interface KeyListener, o que obriga à implementação do método keyReleased().

7.3 Dentro do método keyReleased() verifica-se a existência de:

7.3.1 Uma estrutura condicional que irá verificar se o utilizador já selecionou alguma aplicação daquelas exibidas na tabela. Em caso negativo, é chamado um dos métodos da classe AvisosPopUp a fim de o alertar de tal facto.

7.3.2 Caso tenha selecionado corretamente a aplicação pretendida, através de nova estrutura condicional, procede-se à validação do valor inserido para a quantidade de exibições na JTextField corresponde a um algarismo numérico inteiro, caso contrário, avisa o utilizador para o facto e limpa a JTextField.

7.3.3 Depois, é realizado o cálculo do custo daquele determinado anúncio, que é formatado com auxílio de um objeto do tipo BigDecimal¹⁵, para ser exibido ao utilizador.

É importante destacar, que o valor a ser exibido irá ser atualizado conforme o utilizador modifique a quantidade de exibições pretendidas. Para além disso, no caso de o utilizador modificar a aplicação

¹⁵ <https://www.devmedia.com.br/arredondando-numeros-em-java/28248> (acedido em 24/12/2021)

pretendida, basta que informe novamente quantas exibições pretende, e o valor é novamente atualizado.

8. Dois JButtons:

8.1 O primeiro botão denominado como “criar” possui no seu ActionListener os comandos para verificar se foram selecionados a frase e a aplicação pretendida, bem como se foi introduzida quantidade de exibições pretendidas. Caso falte alguma informação, chama um dos métodos existentes na classe AvisosPopUp para informar o utilizador deste fato, impedindo que sejam criados anúncios sem todas as informações necessárias.

8.1.1 Caso esteja tudo devidamente selecionado e preenchido, são utilizados ciclos forEach para chegar à conta corrente do publicitário que pretende criar o anúncio e verificar se existe saldo suficiente na sua conta. Para esta verificação, é feita uma chamada ao método verificarSaldoParaFazerAnuncio() e enviados como parâmetros a referência da conta corrente, a frase a ser anunciada e a referência do utilizador que requereu o anúncio.

```
@Override
public void actionPerformed(ActionEvent eventosEcraAddAnuncio) {

    objetoJframe.remove(panelAddAnuncio);

    if (eventosEcraAddAnuncio.getSource() == criar) {

        if (nomeAppSelecionada.equals("") || fraseSelecionada.equals("") ||
            quantidadeExibicoesTxt.getText().equals("")) {
            avisosPopUp.mensagemInformacao(143);
            EcraAddAnuncio ecraAddAnuncio = new EcraAddAnuncio(empresaObjeto, objetoJframe, trocaDeldiomaObjeto,
                validadorGeral, avisosPopUp);

        } else {
            if (empresaObjeto.getUtilizadorAtivo() instanceof Intermediario) {

                for (Utilizador utilizador : empresaObjeto.getListaUtilizadores()) {
                    if (utilizador instanceof Publicitario) {

                        for (Frase frase : ((Publicitario) utilizador).getListaFrases()) {
                            if (frase.getFrase().equals(fraseSelecionada)) {

                                for (Conta conta : empresaObjeto.getListaContas()) {
                                    if (conta.getUser().equals(utilizador)) {

                                        verificarSaldoParaFazerAnuncio(conta, frase, (Publicitario) utilizador);
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    } else if (empresaObjeto.getUtilizadorAtivo() instanceof Publicitario) {

        for (Frase frase : ((Publicitario) empresaObjeto.getUtilizadorAtivo()).getListaFrases()) {
            if (frase.getFrase().equals(fraseSelecionada)) {

                for (Conta conta : empresaObjeto.getListaContas()) {
                    if (conta.getUser().equals(empresaObjeto.getUtilizadorAtivo())) {

                        verificarSaldoParaFazerAnuncio(conta, frase, (Publicitario) empresaObjeto.getUtilizadorAtivo());
                    }
                }
            }
        }
    }
}
```



```

    }
    }
    }
    }
}

} else if (eventosEcraAddAnuncio.getSource() == voltar) {

    if (empresaObjeto.getUtilizadorAtivo() instanceof Publicitario) {
        EcraPublicitario ecraPublicitario = new EcraPublicitario(empresaObjeto, objetoJframe, trocaDeldiomaObjeto,
validadorGeral, avisosPopUp);

    } else if (empresaObjeto.getUtilizadorAtivo() instanceof Intermediario) {
        EcraIntermediario ecraIntermediario = new EcraIntermediario(empresaObjeto, objetoJframe,
trocaDeldiomaObjeto, validadorGeral, avisosPopUp);

    }
}
}
}

```

8.2 O segundo tem o comando “voltar” que segue o descrito na unidade 5.4 do capítulo 5 deste relatório e recarrega o ecrã respetivo ao menu de opções consoante o tipo de utilizador.

9. Método verificarSaldoParaFazerAnuncio(): tem como função verificar o saldo do publicitário que tem interesse em criar o anúncio de forma a perceber se possui saldo suficiente para o anúncio pretendido, e funciona da seguinte forma:

9.1 Faz uso de uma estrutura condicional, para a verificação do saldo do respetivo publicitário.

9.2 Caso não exista saldo suficiente, chama um dos métodos existentes na classe AvisosPopUp para informar o utilizador deste fato.

9.3 Caso exista saldo para cobrir os custos do contrato de anúncio pretendido, chama o método popUpSimOuNao() declarado na classe AvisosPopUp exibindo ao utilizador um resumo do contrato e pedindo a confirmação do mesmo, antes de efetivamente criar o contrato em questão.

9.4 Se o utilizador confirmar o contrato, é chamado o método adicionarAnuncio().

```

public void verificarSaldoParaFazerAnuncio(Conta conta, Frase frase, Publicitario publicitario) {

    if (custoDouble > conta.getSaldo()) {

        avisosPopUp.mensagemDialogo(108, 88);

        objetoJframe.remove(panelAddAnuncio);

        EcraAddAnuncio ecraAddAnuncio = new EcraAddAnuncio(empresaObjeto, objetoJframe,
trocaDeldiomaObjeto, validadorGeral, avisosPopUp);

    } else {

        String custo = String.valueOf(new BigDecimal(custoDouble).setScale(2, RoundingMode.UP)),
valorUnitarioPExib = String.valueOf(appAuxiliar.getValorUnitarioPorExibicao()),
quantExib = String.valueOf(quantidadeExibicoesTxt.getText());

        int resultado = avisosPopUp.popUpSimOuNao(fraseSelecionada, quantExib, nomeAppSelecionada,
valorUnitarioPExib, custo);
    }
}

```

```

        adicionarAnuncio(resultado, frase, publicitario);
    }
}

```

10. Método adicionarAnuncio(): Verifica a resposta do utilizador quanto à confirmação da contratação do anúncio.

```

public void adicionarAnuncio(int resultado, Frase frase, Publicitario publicitario) {

    if (resultado == JOptionPane.YES_OPTION) {

        Anuncio anuncio = new Anuncio(Integer.parseInt(quantidadeExibicoesTxt.getText()),
            custoDouble, appAuxiliar, frase, publicitario, programadorAux);

        empresaObjeto.adicionarAnuncioNaArrayListEAdicionarMovimentos(anuncio, programadorAux, publicitario,
            custoDouble);

        objetoJframe.remove(panelAddAnuncio);
        EcraAddAnuncio ecraAddAnuncio = new EcraAddAnuncio(empresaObjeto, objetoJframe, trocaDeldiomaObjeto,
            validadorGeral, avisosPopUp);

    } else {

        objetoJframe.remove(panelAddAnuncio);

        EcraAddAnuncio ecraAddAnuncio = new EcraAddAnuncio(empresaObjeto, objetoJframe, trocaDeldiomaObjeto,
            validadorGeral, avisosPopUp);
    }
}
}

```

10.1 Caso a resposta seja “sim”, de imediato instancia um novo objeto Anuncio e chama o método adicionarAnuncioNaArrayListEAdicionarMovimentos() declarado na classe EmpresaAorWords (item 2.1 da unidade 3.2 do capítulo 3), que irá adicionar o anúncio na respetiva ArrayList e irá dar início aos processos de pagamentos e depósitos necessários a cada anúncio, como será melhor detalhado mais adiante.

10.2 Por fim, recarrega a página para que o utilizador possa iniciar o processo de criação de um novo anúncio. Este processo também ocorre, caso a escolha na PopUp tenha sido “não”.

6.13 Classe EcraEscolherAppConsulta

Corresponde ao ecrã onde os programadores podem escolher uma das suas aplicações para consultar as frases que deverão ser exibidas nesta aplicação. Este ecrã somente se encontra disponível para utilizadores do tipo Programador, tal como é possível verificar no esquema de ecrãs 8.3 do capítulo 8.

Cabe destacar que, para os intermediários, está prevista a funcionalidade de verificar todas as informações de todos os contratos de todos os anúncios gerados, como já explicitado no capítulo 2 unidade 2.1 deste relatório e a explorar na unidade 6.17 deste capítulo que trata sobre a classe EcraContratosEmpresa.

Esta classe foi desenvolvida tendo em conta o requisito 5 b. do enunciado do projeto que dita que cada utilizador deve poder “verificar as frases a exibir/ou as aplicações nas quais as mesmas serão exibidas (conforme seja programador ou publicitário)”.

A Classe `EcrãEscolherAppConsulta` implementa o conceito de classes aninhadas (*Nested Classes*), ou seja, vai possuir outras classes dentro de si, classes estas que correspondem às componentes gráficas desta classe. A Classe vai ainda seguir a implementação padrão explicitada no capítulo 5 deste relatório, porém com as peculiaridades destacadas abaixo.

A presente classe possui os seguintes componentes:

1. `JLabel` com dados informativos, que tenciona ajudar o utilizador a interagir com o ecrã.
2. Uma `JTable` que irá exibir ao programador o nome de todas as suas aplicações já registadas. Esta tabela segue o mesmo padrão de implementação detalhado neste capítulo na unidade 6.4.1.
3. A tabela desta classe possui ainda, o método `capturarAppSelecionada()` que segue o mesmo modelo de implementação descrito no item 6.10.1 deste relatório, e que tem como intuito “capturar” o nome da aplicação selecionada pelo programador na tabela. Contudo, há uma pequena diferença na implementação do método acima citado relativamente à variável onde será armazenada a referência para a aplicação selecionada.

Para este ecrã concreto, existe a necessidade de enviar esta referência como parâmetro para uma outra classe que irá, de facto, apresentar as frases a serem exibidas (classe descrita no capítulo 6 unidade 6.15 deste relatório) nesta aplicação escolhida, porém, este parâmetro a mais gerava problemas relativamente à dinâmica da troca entre idiomas, uma vez que, não seria possível recarregar perfeitamente o ecrã de frases a exibir, caso um novo idioma fosse escolhido.

Para sanar esta situação, conforme já descrito no capítulo 4 item 4.6.1 deste relatório, a classe `TrocaDeIdioma` possui um atributo denominado “bufferAuxiliar” que armazena informações transitórias, neste caso, foi utilizado o método `setBufferAuxiliar()` para armazenar o nome da aplicação selecionada pelo utilizador na tabela.

```
public void capturarAppSelecionada() {  
  
    jTableAppConsulta.getSelectionModel().addListSelectionListener(event -> { // Captura o evento  
    de quando o utilizador clicar numa linha da tabela  
    trocaDeIdiomaObjeto.setBufferAuxiliar  
    (jTableAppConsulta.getValueAt(jTableAppConsulta.getSelectedRow(), 0).toString());  
    });  
}
```

4. Um `JButton` denominado “pesquisar” que em seu `actionListener` usa uma estrutura condicional para verificar se foi selecionada alguma aplicação pelo programador, e em caso negativo, chama um dos métodos da classe `AvisosPopUp` a fim de alertar este fato. Caso tenha selecionado o nome de alguma aplicação, segue o descrito na unidade 5.4 do capítulo 5 deste relatório e carrega o ecrã onde o programador irá visualizar as frases a serem exibidas na aplicação escolhida.

- Um segundo JButton com o comando “voltar” que segue o descrito na unidade 5.4 do capítulo 5 deste relatório e recarrega o ecrã respetivo ao menu de opções consoante o tipo de utilizador.

6.14 Classe EcraEscolherFraseConsulta

Corresponde ao ecrã onde os publicitários podem escolher uma de suas frases para consultar as aplicações onde esta frase deverá ser exibida. Apenas de encontra disponível para utilizadores do tipo Publicitário, tal como é possível verificar no item 8.2 do Capítulo 8, referente aos Esquemas de Ecrãs.

Segue exatamente o mesmo conceito explicitado no unidade 6.13 deste capítulo, cabendo apenas citar que onde se lê “aplicação”, deve ser percebido “frase”.

Cumprir também que esta classe possui o mesmo componente descrito no item 3 da unidade 6.13 deste capítulo, com a única diferença de que o nome do método, neste caso, é `capturarFraseSelecionada()`.

6.15 Classe EcraVerFrasesPorExibir

Corresponde ao ecrã que permite aos programadores, após escolher uma de suas aplicações (funcionalidade descrita na unidade 6.13 deste capítulo), visualizar uma tabela com todas as frases a serem exibidas na aplicação que tinha previamente selecionado. Permite ainda visualizar a receita atual gerada com os anúncios desta aplicação. Este ecrã somente se encontra disponível para utilizadores do tipo Programador, tal como é possível verificar no esquema de ecrãs 8.3 do capítulo 8.

A presente classe é detentora de classes aninhadas (*Nested Classes*), ou seja, vai possuir outras classes dentro de si, respetivas aos componentes gráficos desta classe. Esta vai ainda seguir a implementação padrão explicitada no capítulo 5 deste relatório, porém com as peculiaridades destacadas abaixo.

Esta classe detém os seguintes componentes:

- JLabel com dados informativos, que tenciona ajudar o utilizador a interagir com o ecrã.
- Uma JTable que irá apresentar ao programador, todas as frases a serem exibidas na aplicação escolhida. Esta tabela segue o mesmo padrão de implementação detalhado neste capítulo na unidade 6.4.1, porém com o seguinte acréscimo:

2.1 Dentro do ciclo `forEach` criado para alimentar o vetor `Object dadosLinha[]` existe ainda o comando abaixo, que irá calcular a receita total atual gerada pelos anúncios da aplicação escolhida e armazenar dentro de uma variável do tipo `double`:

```
Object dadosLinha[] = new Object[vetorColunasTabela.length];

for (Anuncio anuncio : anuncioListaAuxiliar) {
    dadosLinha[0] = anuncio.getFraseAnunciada().getFrase();
    dadosLinha[1] = anuncio.getNumExibicoes();
    dadosLinha[2] = BigDecimal.valueOf(anuncio.getReceitaProgramador()).setScale(2,
    RoundingMode.UP) + " €";
```

```

        receitas = receitas + anuncio.getReceitaProgramador();

        modeloTableFrasesExibir.addRow(dadosLinha);
    }

```

3. Uma JLabel que irá receber o valor calculado acima já formatado com o auxílio da classe BigDecimal, exibindo, assim, a receita daquela aplicação ao programador.

6.16 Classe EcraVerAppsOndeExibidas

Corresponde ao ecrã que permite aos publicitários, após escolher uma de suas frases (funcionalidade descrita no unidade 6.14 deste capítulo), visualizar uma tabela com todas as aplicações onde esta frase será exibida. Permite ainda visualizar o custo atual com os anúncios desta frase. Apenas de encontra disponível para utilizadores do tipo Publicitário, tal como é possível verificar no item 8.2 do Capítulo 8, referente aos Esquemas de Ecrãs.

Segue exatamente o mesmo conceito explicitado na unidade 6.15 do presente capítulo, cabendo apenas informar que onde se lê “frase”, deve ser percebido “aplicação”, e vice-versa.

Esta classe vai ainda possuir o mesmo componente descrito no item 2.1 da unidade 6.15 deste capítulo, com a única diferença de que o cálculo é referente ao custo dos anúncios:

```

for (Anuncio anuncio : anuncioListaAuxiliar) {
    dadosLinha[0] = anuncio.getAplicacao().getNome();
    dadosLinha[1] = BigDecimal.valueOf(anuncio.getValorUnitario()).setScale(2, RoundingMode.UP) + " €";
    dadosLinha[2] = anuncio.getNumExibicoes();
    dadosLinha[3] = BigDecimal.valueOf(anuncio.getValorAnuncio()).setScale(2, RoundingMode.UP) + " €";
    custo = custo + anuncio.getValorAnuncio();
    modeloTableAppsOndeExibir.addRow(dadosLinha);
}

```

6.17 Classe EcraContratosEmpresa

Corresponde ao ecrã que permite aos intermediários visualizarem numa tabela todos os anúncios de todos os utilizadores (ver item 8.1 do Capítulo 8), permite ainda que os contratos dos anúncios sejam ordenados e/ou filtrados por data do contrato. Este ecrã permite aos intermediários ter acesso às seguintes informações de cada anúncio:

1. Nome da aplicação contratada;
2. Nome do programador a quem pertence esta aplicação;
3. Frase a ser anunciada;
4. Nome do publicitário que requereu o anúncio;
5. Data em que foi feito o contrato do anúncio;
6. Valor cobrado por exibição;
7. Número de exibições contratadas;
8. Receitas que a empresa gerou com o anúncio;

Esta classe vai ter em conta o conceito de classes aninhadas (Nested Classes), isto é, a classe vai possuir outras classes dentro de si, estas que correspondem às componentes gráficas desta classe. Além disso, vai seguir a implementação padrão explicitada no capítulo 5 deste relatório, porém com as particularidades abaixo enumeradas.

Possui os seguintes componentes:

1. Uma JComboBox<String> que irá exibir uma lista com todas as datas em que foram estabelecidos contratos de anúncio.
2. Método carregarDadosComboBox():

2.1. Utiliza ciclos foreach para percorrer a ArrayList listaAnuncios (contida na classe EmpresaAorWords), o que é necessário para obter as datas a exibir. Reitera-se que o acesso a todas as ArrayLists do programa ocorre através de uma instância da classe EmpresaAorWords declarada na classe Program (detentora do método main), que é enviada como parâmetro a todas as classes da GUI.

2.2. Para evitar que a comboBox exiba uma lista de datas com valores repetidos, foi utilizado uma lista auxiliar do tipo HashSet para receber estas datas, e que, por si só, descarta os valores repetidos da lista:

```
public void carregarDadosComboBox() {  
  
    HashSet<String> datasLista = new HashSet<>();  
    for (Anuncio anuncio : empresaObjeto.getListaAnuncios()) {  
        datasLista.add(String.valueOf(anuncio.getDataContrato()));  
    }  
  
    ArrayList<String> datas = new ArrayList<>(datasLista);  
    Collections.sort(datas);  
  
    for (String elementoData : datas) {  
        comboBoxDatas.addItem(elementoData);  
    }  
}
```

2.3. De seguida, a lista foi convertida para ArrayList de forma a que pudesse ser ordenada com o auxílio do método sort() da classe Collections, o que permitiu que as datas fossem visíveis ao utilizador em ordem crescente. Finalmente, adicionou-se os itens desta lista ordenada à JComboBox<String>, através do método addItem().

3. Método capturarAcaoComboBox(): contém o método addItemListener() inserido na JComboBox<String> que através dos métodos getStateChange() e getItem(), irá “capturar” e armazenar a data selecionada pelo utilizador numa variável String denominada dataSelecionada, a fim de armazenar a referência para a data a ser usada no filtro:

```
public void capturarAcoesComboBox() {  
  
    comboBoxDatas.addItemListener(eventoComboBoxData -> {  
  
        if (eventoComboBoxData.getStateChange() == ItemEvent.SELECTED) {  
            dataSelecionada = (String) eventoComboBoxData.getItem();  
        }  
    })  
}
```

```

    });
}

```

4. Uma JTable que irá exibir ao utilizador, todos os anúncios contratados até o momento, com as informações mencionadas no início desta unidade 6.17.
Esta tabela segue o mesmo padrão de implementação detalhado neste capítulo na unidade 6.4.1.
5. Um JButton denominado “filtrar”.
6. Método que permite filtrar a tabela por data:

6.1 Um método denominado `inserirAcaoBotaoFiltrar()` que contém, dentro de si, o `actionListener` do botão “filtrar”, referido no número anterior, que vai utilizar o método `setRowFilter()`, para inserir no objeto do tipo `TableRowSorter` (já detalhado nas unidades 3.7 e 3.8 do item 6.4.1 capítulo 6) o comando que irá permitir que a tabela seja filtrada.

```

public void inserirAcaoBotaoFiltrar() {

    filtrar.addActionListener(eventoBotaoFiltrar -> {

        if (dataSelecionada.equals("")) {
            sorter.setRowFilter(null);
            avisosPopUp.mensagemInformacao(148);
        } else {
            sorter.setRowFilter(RowFilter.regexFilter(dataSelecionada));
        }
    });
}

```

6.3 Utiliza uma estrutura condicional para verificar se previamente foi selecionada, na `comboBox`, alguma data (detalhes no item 3 acima). Para evitar exceções `NullPointerException`, define o parâmetro a ser enviado como `null`. Ademais, caso o utilizador acione o botão “filtrar” sem antes ter selecionado uma data na `comboBox`, ocorre a chamada de um dos métodos da classe `AvisosPopUp` a fim de o alertar.

6.4 Caso já tenha sido selecionada alguma data como chave para a filtragem, é inserido no objeto `TableRowSorter` por meio do método `setRowFilter()`, o comando que irá usar a classe abstrata `RowFilter` para aceder o método estático `regexFilter()` enviando como parâmetro a data selecionada na `comboBox`¹⁶.

6.5 Cabe destacar que o método `regexFilter()` usa um comando de código que utiliza de uma expressão regular para determinar quais linhas devem ser incluídas na tabela a ser exibida, garantindo que somente as linhas que contenham pelo menos um valor igual ao parâmetro recebido, neste caso a data escolhida pelo utilizador, sejam inseridas na tabela a ser exibida no ecrã.¹⁷

¹⁶ <https://docs.oracle.com/javase/8/docs/api/javafx/swing/class-use/RowFilter.html> (acedido em 26/12/2021)

¹⁷ <https://docs.oracle.com/javase/7/docs/api/javafx/swing/RowFilter.html> (acedido em 26/12/2021)

7. E um segundo JButton com o comando “voltar” que segue o descrito na unidade 5.4 do capítulo 5 deste relatório e recarrega o ecrã respetivo ao menu de opções consoante o tipo de utilizador, neste caso concreto, carregando o Menu do intermediário.

6.18 Classe EcraContratosProgramador

Corresponde ao ecrã que permite aos programadores visualizarem as informações de todos os contratos de anúncios que possuem, inclusive a sua parcela de receita que aquele contrato de anúncio gerou. Este é um tipo de ecrã que apenas se encontra disponível para utilizadores do tipo programador, como é possível verificar no esquema de ecrã do item 8.3, do Capítulo 8.

A classe vai seguir o conceito de classes aninhadas (Nested Classes), ou seja, vai possuir outras classes dentro de si, classes estas que correspondem às componentes gráficas da classe. Vai ainda seguir a implementação padrão explicitada no capítulo 5 deste relatório, porém com as peculiaridades destacadas abaixo.

Possui os seguintes componentes:

1. JLabel com dados informativos, auxiliando o utilizador a interagir com o ecrã.
2. Uma JTable que irá exibir ao programador as informações dos contratos que envolvem suas aplicações, definida no método construirCarregarDadosDaJtable().

Esta tabela segue o mesmo padrão de implementação detalhado neste capítulo no item 6.4.1, inclusive com a funcionalidade que permite ao utilizador ordenar os dados de qualquer coluna que desejar.

3. JButton com o comando “voltar” que segue o descrito na unidade 5.4 do capítulo 5 deste relatório e recarrega o ecrã respetivo ao menu de opções consoante o tipo de utilizador. Neste caso, carregará o Menu do programador.

6.19 Classe EcraContratosPublicitario

Corresponde ao ecrã que permite aos publicitários visualizarem as informações de todos os seus anúncios contratados, inclusive o custo gerado por cada anúncio. Como tal, é um ecrã que apenas se encontra disponível para utilizadores do tipo publicitários, como é possível evidenciar no item 8.2 do Capítulo 3.

Segue exatamente a mesma implementação explicitada na unidade 6.18 acima, todavia, ressalva-se que onde se lê “aplicação”, deve ser percebido “frase” e que a tabela ao invés de mostrar a receita, mostra o custo de cada contrato de anúncio.

6.20 Classe EcraConta

É a classe que corresponde ao ecrã de visualização da conta-corrente de cada utilizador. É comum a todos os utilizadores (apesar de com algumas diferenças), tal como é possível evidenciar pelos esquemas de ecrã do Intermediário (item 8.1), do Publicitário (8.2) e do Programador (8.3) do Capítulo 8.

Segue segue o conceito de classes aninhadas (*Nested Classes*), mais concretamente, é uma classe que possui outras classes dentro de si, as últimas que correspondem às componentes gráficas da classe. E, também, segue a implementação padrão explicitada no capítulo 5 deste relatório, porém com as peculiaridades destacadas abaixo.

Contém, no seu construtor, código necessário para criar quatro visualizações distintas do mesmo ecrã, consoante a referência do utilizador ativo no sistema. Para melhor entendimento, encontram-se abaixo, um excerto do construtor da classe EcraConta, que permite controlar os componentes que serão inseridos no JPanel no momento da construção do ecrã.

Estes primeiros dois componentes são inseridos em todas as hipóteses:

```
panelConta.add(labelSaldo);
panelConta.add(historico);
```

Em seguida, temos as estruturas condicionais que controlam como deverão ser carregados os ecrãs, consoante o utilizador ativo:

```
if (empresaObjeto.getUtilizadorAtivo() instanceof Publicitario) {

    panelConta.add(levantar);
    panelConta.add(depositar);

} else if (empresaObjeto.getUtilizadorAtivo() instanceof Programador) {

    panelConta.add(levantar);

} else if (empresaObjeto.getUtilizadorAtivo().getEmail().equals(empresaObjeto.getEmailFundador())) { //
    seria o intermediário fundador

    panelConta.add(levantar);
    panelConta.add(depositar);
    panelConta.add(todosMov);

} else { // seriam os demais intermediários

    panelConta.add(todosMov);

}
```

Para que não surjam dúvidas, vale a pena recordar que os utilizadores da aplicação AoRWords podem ser: publicitários, programadores, intermediários ou o intermediário fundador (o último que tem como particularidade o facto de ser o único que pode levantar dinheiro da conta da empresa).

Antes de explicitar as componentes específicas que cada ecrã possui, todos os EcraConta vão possuir as seguintes componentes, independentemente do tipo de utilizador ativo:

1. Método carregarSaldoAtualdaConta() : utiliza ciclos foreach para procurar o saldo da conta corrente do utilizador ativo, quando este for um programador ou um publicitário, ou o saldo da empresa quando o utilizador ativo for um intermediário.

```

public String carregarSaldoAtualdaConta() {
    String saldo = "";

    if (empresaObjeto.getUtilizadorAtivo() instanceof Intermediario) {

        for (Conta elemento : empresaObjeto.getListaContas()) {
            if (elemento.getUser().getEmail().equals(empresaObjeto.getEmailFundador())) {
                saldo = String.valueOf(BigDecimal.valueOf(elemento.getSaldo()).setScale(2, RoundingMode.UP));
            }
        }

    } else {
        for (Conta elemento : empresaObjeto.getListaContas()) {
            if (elemento.getUser().equals(empresaObjeto.getUtilizadorAtivo())) {
                saldo = String.valueOf(BigDecimal.valueOf(elemento.getSaldo()).setScale(2, RoundingMode.UP));
            }
        }
    }
    return saldo;
}

```

2. JLabel “labelSaldo” irá exibir o valor do saldo da respetiva conta corrente. O valor exibido corresponde ao valor retornado pelo método analisado no item anterior. Este é inserido na JLabel através do método setText().
3. Possui ainda os comandos que permitem controlar quais serão os componentes adicionados à JPanel, de acordo com o tipo de utilizador ativo no momento.

Desta forma, alguns dos componentes adicionados ao JPanel mudam consoante o tipo de utilizador, conforme será detalhado abaixo:

6.20.1 Dos componentes inseridos a JPanel para o ecrã conta do intermediário fundador

Para criar o ecrã de interface da conta do intermediário fundador (ver item 8.1 do Capítulo 8), são adicionados os seguintes componentes a JPanel:

1. JButton “levantar” que contém em seu ActionListener:

1.1 Chamada para o método inputDialog() declarado na classe AvisoPopUp. Exibe uma PopUp com uma caixa de texto a solicitar que o utilizador informe o valor que deseja levantar da sua conta corrente. Esta PopUp permite ao utilizador inserir o valor desejado e escolher as opções “ok” ou “cancelar”.

1.2 Estrutura condicional a fim de validar se o que foi introduzido pelo utilizador, se trata de um valor numérico (inteiro ou real), e outra a verificar se foi inserido algum valor antes do utilizador clicar em “ok”, caso o valor introduzido não passe no validador ou o utilizador clicar em “ok” sem ter digitado nada na caixa de texto, é chamado um método da classe AvisoPopUp para o informar desse facto.

1.3 Um comando para modificar o separador de casas decimais para “.”, nos casos do utilizador introduzir “;”, com o intuito de evitar erros na execução do código e melhorar a usabilidade do programa

```

valor = valor.replaceAll(";", ".");

```

1.4 Ciclos forEach que irão localizar a referência exata para a conta da empresa, no caso do fundador ou do utilizador ativo, nos casos dos programadores ou publicitários.

1.5 Estrutura condicional para verificar se existe saldo suficiente para o valor que o utilizador introduziu ser levantado. Caso não tenha saldo suficiente, ocorre a chamada para outro método da classe AvisosPopUp que alerta o utilizador da insuficiência de saldo.

1.6 Caso ocorra tudo bem nas verificações acima, é chamado o método efetuarLevantamento() declarado na classe Conta descrito no item 5 da unidade 3.11 capítulo 3 deste relatório.

2. JButton “historico” que segue o descrito na unidade 5.4 do capítulo 5 deste relatório e carrega o ecrã respetivo ao histórico de todos os movimentos financeiros da empresa, no caso dos intermediários ou ao ecrã com todos os movimentos financeiros do utilizador ativo, no caso deste ser um publicitário ou programador (já detalhado na unidade 6.21 capítulo 6).
3. JButton “todosMov” que segue o descrito na unidade 5.4 do capítulo 5 deste relatório e carrega o ecrã respetivo ao histórico de todos os movimentos financeiros de todos os utilizadores registados (já detalhado na unidade 6.22 capítulo 6).
4. JButton com o comando “voltar” que segue o descrito na unidade 5.4 do capítulo 5 deste relatório e recarrega o ecrã respetivo ao menu de opções do intermediário.

6.20.2 Dos componentes inseridos a JPanel para o ecrã conta dos demais intermediários

Para o ecrã de conta dos restantes intermediários, os componentes adicionados a JPanel respetiva são os descritos nos itens 2, 3 e 4 do item 6.20.1 acima, ou seja, somente para o intermediário fundador, é adicionado o JButton “levantar” (ver item 8.1 do Capítulo 8). Esta limitação evita que qualquer outro intermediário levante dinheiro da conta da empresa.

6.20.3 Dos componentes inseridos a JPanel para o ecrã conta dos programadores

Para o ecrã de conta dos programadores (ver item 8.3 do Capítulo 8), os componentes adicionados a JPanel respetiva são os descritos nos itens 1, 2 e 4 do item 6.20.1 acima.

6.20.4 Dos componentes inseridos a JPanel para o ecrã conta dos publicitários

Para o ecrã de conta dos publicitários (ver item 8.2 do Capítulo 8), os componentes adicionados a JPanel respetivo são os descritos nos itens 1, 2 e 4 da unidade 6.20.1 acima, com o seguinte acréscimo:

1. JButton “depositar” que contém, no seu ActionListener, os mesmos comandos presentes no JButton “levantar” descrito no item 1 da unidade 6.20.1 acima, com uma diferença apenas quanto aos itens 1.1 e 1.6:

1.1 Chamada para o método inputDialog() declarado na classe AvisoPopUp. Exibe uma PopUp com uma caixa de texto a solicitar que o publicitário informe o valor que deseja depositar da sua conta

corrente. Esta PopUp permite ao utilizador inserir o valor desejado e escolher as opções “ok” ou “cancelar”.

1.6 Caso ocorra tudo bem em todas as verificações, é chamado o método `efetuarDeposito()` declarado na classe `Conta` descrito no item 3 da unidade 3.11 capítulo 3 deste relatório.

6.21 Classe `EcrãMovimentosFinanceiros`

Corresponde ao ecrã que permite visualizar numa tabela, o histórico de todos os movimentos financeiros. É comum a todos os utilizadores, tal como é possível evidenciar pelos esquemas de ecrã do Intermediário (item 8.1), do Publicitário (8.2) e do Programador (8.3) do Capítulo 8.

Vai implementar o conceito de classes aninhadas (*Nested Classes*), isto é, possui outras classes dentro de si, respetivas aos componentes gráficos desta classe. E, vai ainda seguir a implementação padrão explicitada no capítulo 5 deste relatório, porém com as peculiaridades destacadas abaixo.

Cumprе informar que, as informações a serem exibidas nesta tabela têm em conta o tipo de utilizador que se encontra ativo na aplicação, ou seja:

- Caso o utilizador ativo seja um intermediário, as informações exibidas serão sobre o histórico de movimentos da conta corrente da empresa (ver item 8.1 do Capítulo 8).
- Caso o utilizador ativo seja um programador ou publicitário, as informações exibidas serão sobre o histórico de movimentos da sua conta corrente pessoal (ver itens 8.3 e 8.2, respetivamente, do capítulo 8).

Este ecrã detém os seguintes componentes:

1. Uma `JComboBox<String>` que irá exibir uma lista com as datas de todos os movimentos financeiros existentes.

2. Método `carregarDadosComboBox()`:

2.1. Utiliza ciclos `forEach` para percorrer as `ArrayLists` necessárias para procurar as datas a exibir. Este método contempla estruturas condicionais que irão verificar a instância do utilizador ativo, garantindo, desta forma, que são carregadas as datas corretas dos movimentos financeiros.

2.2 Para evitar que a `comboBox` exiba uma lista de datas com valores repetidos, foi utilizado uma lista auxiliar do tipo `HashSet` para receber estas datas, e que por si só, descarta os valores repetidos da lista:

```
public void carregarDadosComboBox() {  
  
    HashSet<String> datasLista = new HashSet<>();  
  
    if (empresaObjeto.getUtilizadorAtivo() instanceof Publicitario || empresaObjeto.getUtilizadorAtivo() instanceof Programador) {  
  
        for (Conta elementoConta : empresaObjeto.getListaContas()) {  
  
            if (elementoConta.getUser().equals(empresaObjeto.getUtilizadorAtivo())) {  
  
                for (MovimentoFinanceiro elementoMov :  
                    elementoConta.getListaMovimentosFinanceiros()) {
```

```

        datasLista.add(String.valueOf(elementoMov.getData()));
    }
}
}

if (empresaObjeto.getUtilizadorAtivo() instanceof Intermediario) {

    for (Conta elementoConta : empresaObjeto.getListaContas()) {

        if (elementoConta.getUser().getEmail().equals(empresaObjeto.getEmailFundador())) {

            for (MovimentoFinanceiro elementoMov :
                elementoConta.getListaMovimentosFinanceiros()) {

                datasLista.add(String.valueOf(elementoMov.getData()));

            }
        }
    }
}

ArrayList<String> datas = new ArrayList<>(datasLista);
Collections.sort(datas);

for (String elemento : datas) {
    comboBoxDatas.addItem(elemento);
}
}

```

2.3. A seguir, a lista foi convertida em ArrayList para se pudesse fazer uso do método sort() da classe Collections, o que permitiu que as datas fossem visíveis ao utilizador em ordem crescente. Depois, os itens ordenados foram adicionados à JComboBox<String>, através do método addItem().

3. Método capturarAcaoComboBox(): contém o método addItemListener() inserido na JComboBox<String>, que, através dos métodos getStateChange() e getItem(), irá “capturar” e armazenar a data selecionada pelo utilizador numa variável String denominada dataSelecionada, armazenando, desta forma, a data a ser usada no filtro:

```

public void capturarAcoesComboBox() {

    comboBoxDatas.addItemListener(eventoComboBoxData -> {

        if (eventoComboBoxData.getStateChange() == ItemEvent.SELECTED) {
            dataSelecionada = (String) eventoComboBoxData.getItem();
        }
    });
}

```

4. Uma JTable que irá exibir ao utilizador, todos os movimentos financeiros ocorridos até o momento, e que possui o mesmo número de colunas a serem exibidas para todos os tipos de utilizadores, com os seguintes dados:

1. Data em que ocorreu o movimento financeiro: gerado automaticamente pelo construtor da classe MovimentoFinanceiro detalhada na unidade 3.12 capítulo 3 deste relatório.
2. O tipo de movimento: definido através do Enumerador descrito na unidade 3.13 capítulo 3.
3. Valor do movimento financeiro.

4.1 Esta tabela segue o mesmo padrão de implementação detalhado neste capítulo na unidade 6.4 item 6.4.1, com apenas uma peculiaridade adicional no método `construirCarregarDadosDaJtable()` relativamente à introdução das informações a serem exibidas na tabela:

Como esclarecido anteriormente, esta `JTable` irá ser carregada com informações consoante o tipo de utilizador ativo. É relevante referir que, de forma a que seja possível mudar o nome do movimento realizado, utilizou-se a seguinte linha de pensamento:

```
if (empresaObjeto.isIdiomaPortugues()) {
    dadosLinha[1] = listaDeMovimentosFinanceiro.getTipoMovimento();
} else {
    if (listaDeMovimentosFinanceiro.getTipoMovimento().equals(TipoDeMovimento.DEPOSITO)) {
        dadosLinha[1] = trocaDeIdiomaObjeto.getIdiomasLista().get(151)[2];
    } else if (listaDeMovimentosFinanceiro.getTipoMovimento().equals(TipoDeMovimento.LEVANTAMENTO)) {
        dadosLinha[1] = trocaDeIdiomaObjeto.getIdiomasLista().get(152)[2];
    } else {
        dadosLinha[1] = trocaDeIdiomaObjeto.getIdiomasLista().get(153)[2];
    }
}
```

Sempre que se adicionava a linha à tabela, verificava-se se, em relação à posição 1 do vetor, se o idioma do programa se encontrava em português, se sim, bastava adicionar a informação como consta da lista de movimentos, caso contrário, é necessário proceder à verificação do tipo de movimento através de uma estrutura condicional, conforme o tipo de movimento que se encontra armazenado, é colocado o tipo de movimento correspondente da língua estrangeira.

4.2 Para controlar esta entrada de informações, foi criado um `ArrayList` auxiliar denominado por “`listaDeMovimentosFinanceiros`” que irá receber os dados que serão, de facto, carregados na tabela.

4.3 Posteriormente foram usadas estruturas condicionais para verificar a instância do utilizador ativo, e, de seguida, foram usados ciclos `for Each` para trazer os dados dos movimentos financeiros da conta corrente respetiva.

4.4 A seguir, estes movimentos são adicionados na `ArrayList` auxiliar.

```
listaDeMovimentosFinanceiros = new ArrayList<>(10);

if (empresaObjeto.getUtilizadorAtivo() instanceof Intermediario) {
    for (Conta elementoConta : empresaObjeto.getListaContas()) {
        if (elementoConta.getUser().getEmail().equals(empresaObjeto.getEmailFundador())) {
            listaDeMovimentosFinanceiros.addAll(elementoConta.getListaMovimentosFinanceiros());
        }
    }
} else {
    for (Conta elementoConta : empresaObjeto.getListaContas()) {
        if (elementoConta.getUser().equals(empresaObjeto.getUtilizadorAtivo())) {
            listaDeMovimentosFinanceiros.addAll(elementoConta.getListaMovimentosFinanceiros());
        }
    }
}
```

}

5. Um JButton denominado “filtrar”.

6. Um método denominado `inserirAcaoBotaoFiltrar()` que tem a mesma implementação já descrita no item 6 da unidade 6.17 capítulo 6 que descreve a lógica implementada para permitir ao utilizador filtrar as informações por data.

Importante destacar que não foi implementado neste ecrã a funcionalidade de filtrar os movimentos por utilizador, pelo simples fato de que este ecrã exibe ao utilizador somente dados de uma única fonte, ou seja, ou exibe seus movimentos financeiros pessoais caso o utilizador seja programador/publicitário, ou exibe os movimentos da empresa caso o utilizador ativo seja um intermediário,

7. E, um segundo JButton com o comando “voltar” que segue o descrito na unidade 5.4 do capítulo 5 deste relatório e recarrega o ecrã respetivo ao menu de opções consoante o tipo de utilizador.

6.22 Classe `EcrãMovimentosUsers`

Corresponde ao ecrã que permite aos intermediários visualizar, numa tabela, o histórico de todos os movimentos financeiros de todos os utilizadores registados na aplicação, assim como permite também que os intermediários possam filtrar estes movimentos por data e/ou utilizador. Então, é um ecrã que apenas é possível ter acesso caso o utilizador seja intermediário, tal como é possível evidenciar pelos esquemas de ecrã do Intermediário (item 8.1) do Capítulo 8.

Este ecrã vai permitir cumprir o requisito que consta no número 6 do enunciado do projeto.

A presente classe vai fazer uso do conceito de classes aninhadas (Nested Classes), ou seja, vai possuir outras classes dentro de si, respetivas às componentes gráficas desta classe; e vai ainda seguir a implementação padrão explicitada no capítulo 5 deste relatório, porém com as peculiaridades destacadas abaixo.

Possui a mesma implementação e os mesmos componentes da classe `EcrãMovimentosFinanceiros` descrita na unidade 6.21 acima, com algumas pequenas diferenças e acréscimos que serão melhor esclarecidos abaixo:

1. Os dados a serem exibidos na JTable deste ecrã são:

1.1 Data em que ocorreu o movimento financeiro: gerado automaticamente pelo construtor da classe `MovimentoFinanceiro` detalhada na unidade 3.12 capítulo 3 deste relatório.

1.2 O tipo de movimento: definido através do Enumerador descrito na unidade 3.13 capítulo e tendo em conta o raciocínio necessário para a mudança de língua que consta do item 4 da unidade 6.21 acima.

1.3 Valor do movimento financeiro.

1.4 Nome do utilizador a quem corresponde cada movimento.

1.5 Id do utilizador a quem corresponde cada movimento.

2. O método `construirCarregarDadosDaJtable()` desta classe utiliza ciclos `forEach` para resgatar os dados de programadores e publicitários de duas `ArrayLists` distintas (`listaContas` e `listaMovimentosFinanceiros`) através de uma estrutura condicional.

Importante destacar que, para garantir a correta troca de idiomas quanto aos nomes dos movimentos financeiros, foi utilizada a mesma lógica já referida no item 4.1 da unidade 6.21 deste capítulo 6.

```
for (Conta conta : empresaObjeto.getListaContas()) {

    if (conta.getUser() instanceof Publicitario || conta.getUser() instanceof Programador) {

        for (MovimentoFinanceiro movimentoFinanceiro : conta.getListaMovimentosFinanceiros()) {

            dadosLinha[0] = movimentoFinanceiro.getData();

            if (empresaObjeto.isIdiomaPortugues()) {
                dadosLinha[1] = movimentoFinanceiro.getTipoMovimento();
            } else {
                if (movimentoFinanceiro.getTipoMovimento().equals(TipoDeMovimento.DEPOSITO)) {
                    dadosLinha[1] = trocaDeIdiomaObjeto.getIdiomasLista().get(151)[2];
                } else if
(movimentoFinanceiro.getTipoMovimento().equals(TipoDeMovimento.LEVANTAMENTO)) {
                    dadosLinha[1] = trocaDeIdiomaObjeto.getIdiomasLista().get(152)[2];
                } else {
                    dadosLinha[1] = trocaDeIdiomaObjeto.getIdiomasLista().get(153)[2];
                }
            }
            dadosLinha[2] = BigDecimal.valueOf(movimentoFinanceiro.getValor()).setScale(2,
RoundingMode.UP) + trocaDeIdiomaObjeto.getIdiomasLista().get(154)[1];
            dadosLinha[3] = conta.getUser().getNome();
            dadosLinha[4] = conta.getUser().getIdUtilizador();
            listaAuxDeMovimentos.add(dadosLinha);
            model.addRow(dadosLinha);
        }
    }
}

sorter = new TableRowSorter<>(model);
jTableTransacoes.setRowSorter(sorter);
}
```

3. Neste ecrã há um segundo JComboBox<String> que irá exibir os IDs dos utilizadores, para que o intermediário possa escolher qual o utilizador cujos movimentos financeiros deseja visualizar.

2.1 Este segundo comboBox, gerou a necessidade de uma segunda lista do tipo HashSet<String> que irá receber estes IDs, porém, o método segue a mesma implementação já descrita no item 2 da unidade 6.21 deste capítulo.

```
public void carregarDadosComboBox() {

    HashSet<String> idLista = new HashSet<>();
    HashSet<String> datasLista = new HashSet<>();

    for (Conta elementoConta : empresaObjeto.getListaContas()) {

        if (elementoConta.getUser() instanceof Publicitario || elementoConta.getUser() instanceof
Programador) {
            idLista.add(String.valueOf(elementoConta.getUser().getIdUtilizador()));
        }
    }
}
```



```

        for (MovimentoFinanceiro elementoMov : elementoConta.getListaMovimentosFinanceiros()) {
            datasLista.add(String.valueOf(elementoMov.getData()));
        }
    }
}

ArrayList<String> datas = new ArrayList<>(datasLista);
Collections.sort(datas);

for (String elementoData : datas) {
    comboBoxDatas.addItem(elementoData);
}
for (String elementoID : idLista) {
    comboBoxID.addItem(elementoID);
}
}

```

2.2 Houve ainda a necessidade de acrescentar no método `capturarAcoesComboBox()`, que permite implementar os comandos necessários para “capturar” o ID selecionado pelo intermediário, para além do comando de “capturar” a data selecionada.

```

public void capturarAcoesComboBox() {
    comboBoxDatas.addItemListener(eventoComboBoxData -> {
        if (eventoComboBoxData.getStateChange() == ItemEvent.SELECTED) {
            dataSelecionada = (String) eventoComboBoxData.getItem();
        }
    });

    comboBoxID.addItemListener(eventoComboBoxID -> {
        if (eventoComboBoxID.getStateChange() == ItemEvent.SELECTED) {
            idSelecionado = (String) eventoComboBoxID.getItem();
        }
    });
}

```

4. Como neste ecrã existe a possibilidade do intermediário fazer um filtro por data ou um filtro por id de utilizador ou um filtro por data e id de utilizador, o método `inserirAcaoBotaoFiltrar()` para além da implementação descrita no item 6 da unidade 6.17 capítulo 6, que se refere a funcionalidade de filtrar por data, teve ainda de contar com mais alguns comandos de código:

Para explicar melhor como funciona este método desta classe `EcrãMovimentosUsers`, considera-se fulcral recorrer à divisão do código em excertos e proceder à sua explicação “passo-a-passo”:

3.1 Como temos a hipótese do utilizador escolher um filtro ou outro, ou escolher os dois filtros, existiu a necessidade de criar um `ArrayList` do tipo `RowFilter` que receberia uma ou duas referências relativas aos filtros escolhidos pelo utilizador.

```

public void inserirAcaoBotaoFiltrar() {

    filtrar.addActionListener(eventoBotaoFiltrar -> {

        ArrayList<RowFilter<Object, Object>> listaFiltros = new ArrayList<>();
    });
}

```

3.2 Além disso, foi necessário utilizar estruturas condicionais que impedissem uma exceção do tipo `NullPointerException`, pois tal inibiria o bom funcionamento dos filtros.

3.3 A primeira condição irá verificar os casos em que o intermediário selecionou uma data para o filtro, porém não selecionou um ID. O código adotado permitiu evitar uma exceção `NullPointerException` e garantir que a tabela seja filtrada apenas por data:

```
if (idSelecionado.equals("")) {// se utilizador não selecionou nenhum id  
    listaFiltros.add(RowFilter.regexFilter(dataSelecionada));  
    sorter.setRowFilter(RowFilter.andFilter(listaFiltros));  
}
```

3.4 A segunda condição irá verificar os casos em que o intermediário selecionou um ID para o filtro, porém não selecionou uma data. Novamente, o código utilizado preveniu uma exceção `NullPointerException` e garantiu que a tabela surgisse, simultaneamente, filtrada apenas por ID:

```
if (dataSelecionada.equals("")) {// se utilizador não selecionou nenhuma data  
    listaFiltros.add(RowFilter.regexFilter(idSelecionado));  
    sorter.setRowFilter(RowFilter.andFilter(listaFiltros));  
}
```

3.5 A terceira condição irá verificar os casos em que o intermediário selecionou uma data e um ID para o filtro, porém garantindo que as informações não se percam, e que a tabela seja filtrada levando em consideração os dois parâmetros de filtragem. Para que isso ocorra com êxito, foi necessário inserir na `ArrayList` descrita acima no item 3.1 os parâmetros de filtragem e em seguida colocar estes dados da `ArrayList` no objeto `TableRowSorter`:

```
if (!dataSelecionada.equals("") && !idSelecionado.equals("")) {// se utilizador selecionou data e id  
    listaFiltros.add(RowFilter.regexFilter(dataSelecionada));  
    listaFiltros.add(RowFilter.regexFilter(idSelecionado));  
    sorter.setRowFilter(RowFilter.andFilter(listaFiltros));  
}
```

3.6 A última condição irá verificar os casos em que o intermediário não nenhuma dado de nenhuma das `comboBox` e de imediato, irá chamar um dos métodos da classe `AvisosPopUp` a fim de alertar o utilizador de tal facto:

```
if (dataSelecionada.equals("")&&idSelecionado.equals("")){  
    // se utilizador não selecionou nem data e nem id  
    sorter.setRowFilter(null);  
    avisosPopUp.mensagemInformacao(149);  
}  
});  
}
```

Para além do acima destacado e como informado anteriormente este ecrã segue a mesma implementação e os mesmos componentes da classe `EcrãMovimentosFinanceiros` descrita na unidade 6.21 acima.

CAPÍTULO 7

BIBLIOGRAFIA UTILIZADA

<https://pt.stackoverflow.com/questions/159708/como-fazer-com-que-os-valores-do-jtable-sejam-os-mesmos-do-arraylistpessoa>

<https://www.greelane.com/pt/ci%C3%A2ncia-tecnologia-matem%C3%A1tica/ci%C3%A2ncia-da-computa%C3%A7%C3%A3o/defaulttablemodel-overview-2033890/>

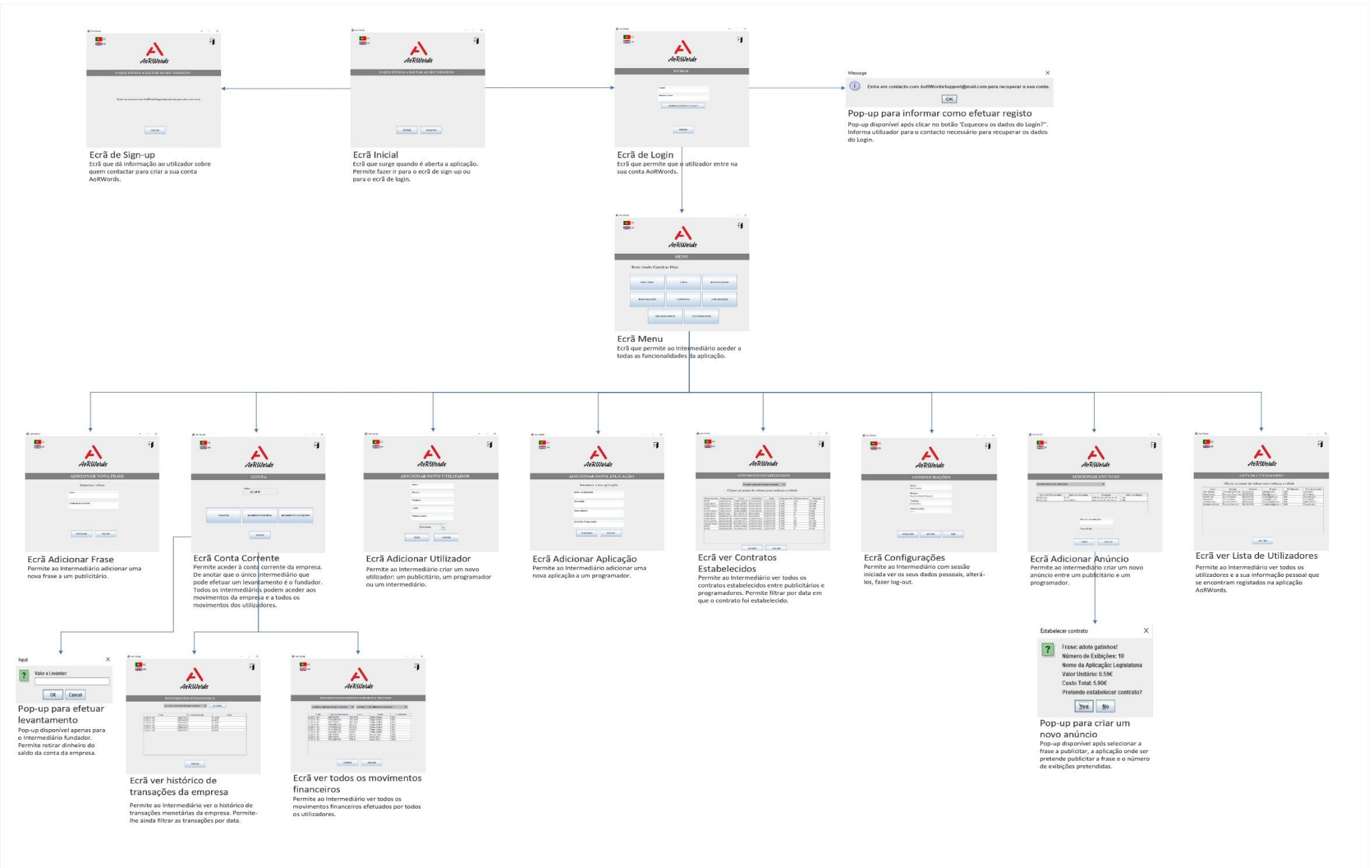
<https://www.devmedia.com.br/ordenando-as-linhas-do-jtable-baseado-nos-valores-da-coluna/2143>

<https://www.guj.com.br/t/ordenar-data-em-uma-jtable/820>

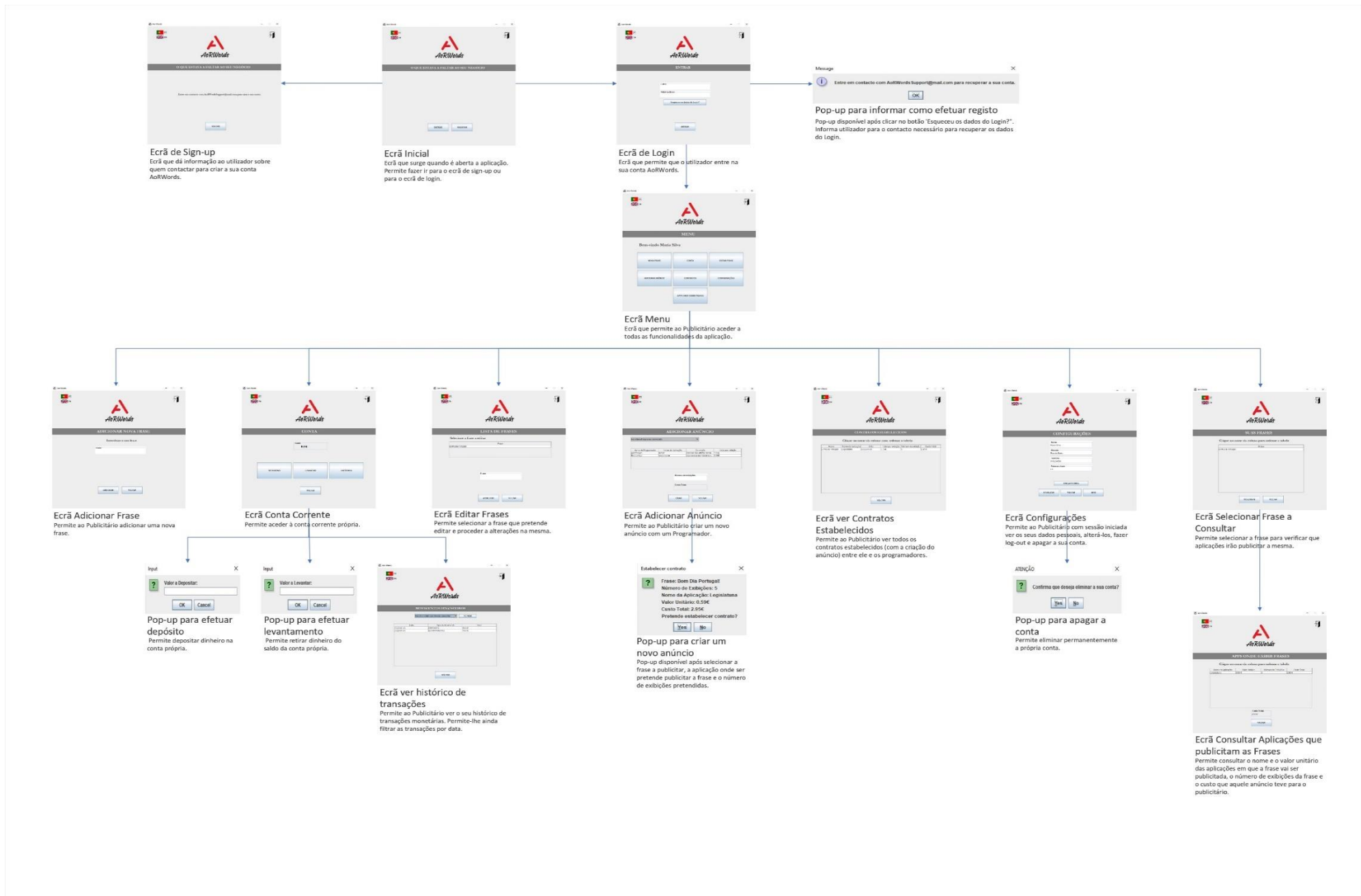
CAPÍTULO 8

ESQUEMAS DE ECRÃ

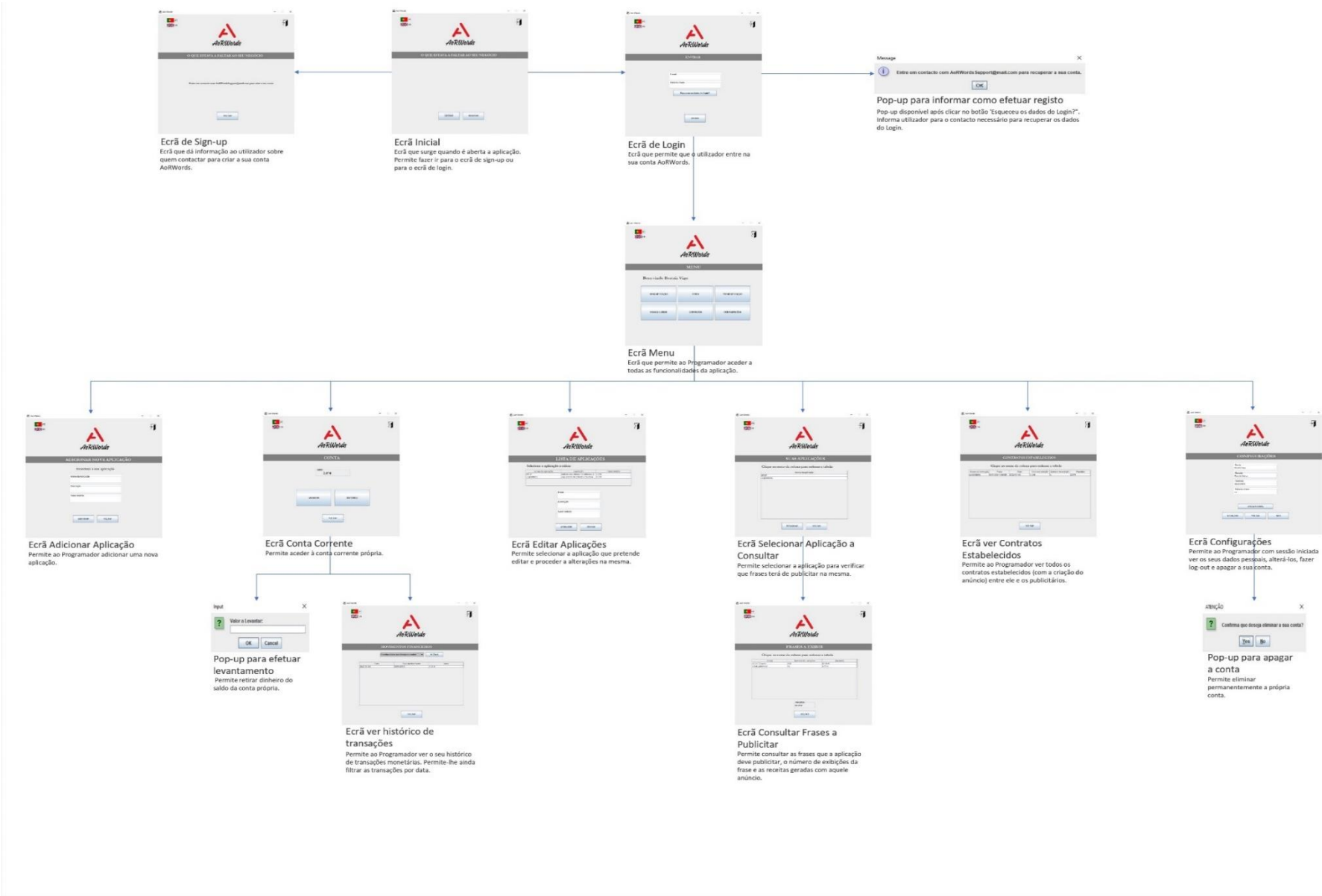
8.1 Esquema de Ecrã do Intermediário



8.2 Esquema de Ecrã do Publicitário



8.3 Esquema de Ecrã do Programador

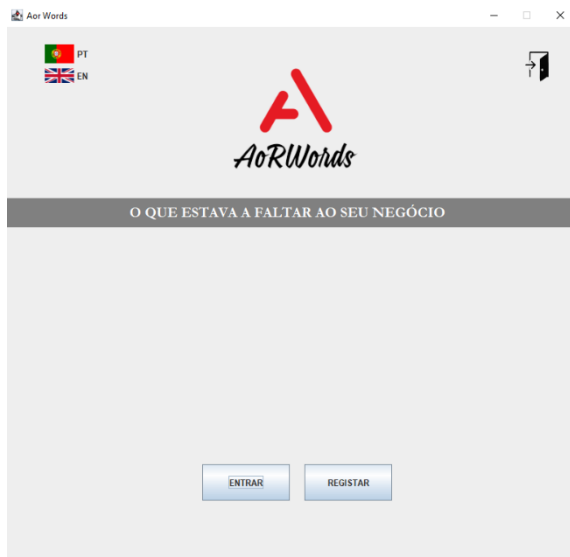


CAPÍTULO 9

Manual de Instruções do programa

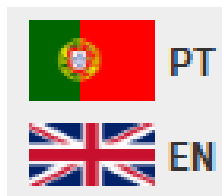
1. Ecrãs comuns a todos os utilizadores

1.1 Ecrã Inicial



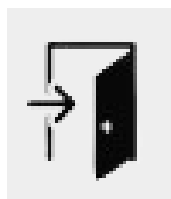
Quando entrar na aplicação, é com esta tela que se irá deparar, e é com algumas destas características que vai poder interagir por toda a aplicação.

1.1.1 Mudança de Idioma



Estes ícones são dois botões e vão permitir mudar o idioma da aplicação. Se clicar na bandeira de Portugal, muda o idioma para português, no entanto, se clicar na bandeira do Reino Unido, muda o idioma para inglês.

1.1.2 Ícone de Saída



Este ícone é um botão e permite-lhe sair e fechar a aplicação.

1.2 Ecrã de Sign-up



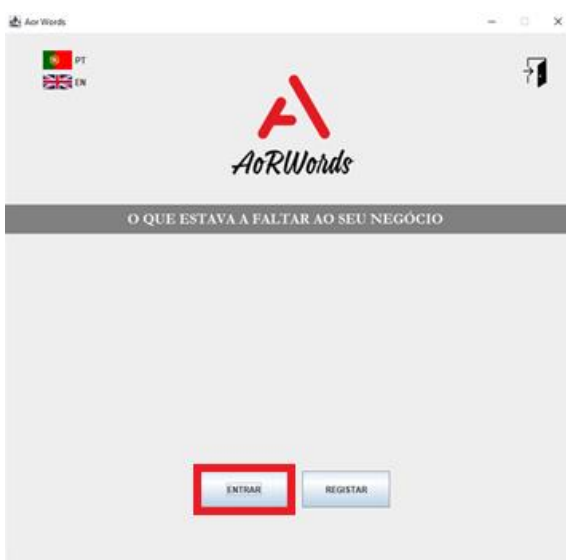
Se clicar no botão destacado a vermelho designado por ‘REGISTAR’, é levado para o ecrã abaixo:



Este ecrã fornece-lhe o contacto de que necessita para poder proceder à criação da sua conta na aplicação AoRWords.

Caso clique no botão designado ‘VOLTAR’, será levado de volta para o ecrã inicial.

1.3 Ecrã Login



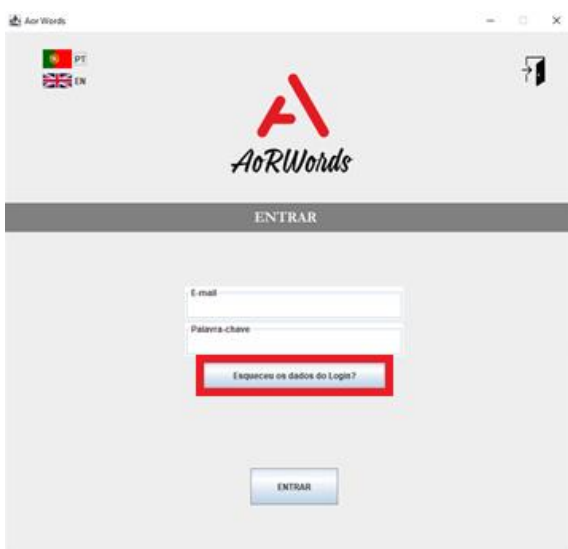
Se clicar no botão vermelho designado por ‘ENTRAR’, é levado para o ecrã abaixo:



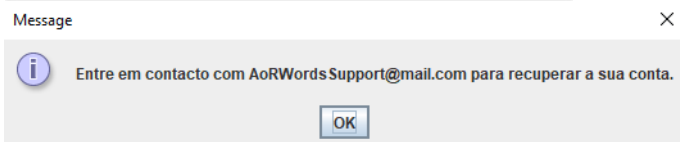
Este ecrã é constituído por duas caixas de texto, a primeira, para colocar o e-mail, e a segunda para colocar a palavra-chave associada ao mesmo.

Para entrar na sua conta, tem duas opções:

- a) Pode clicar no botão entrar, que se encontra na parte inferior da tela, destacado a vermelho, ou;
- b) Pode clicar na tecla 'enter' do seu teclado.

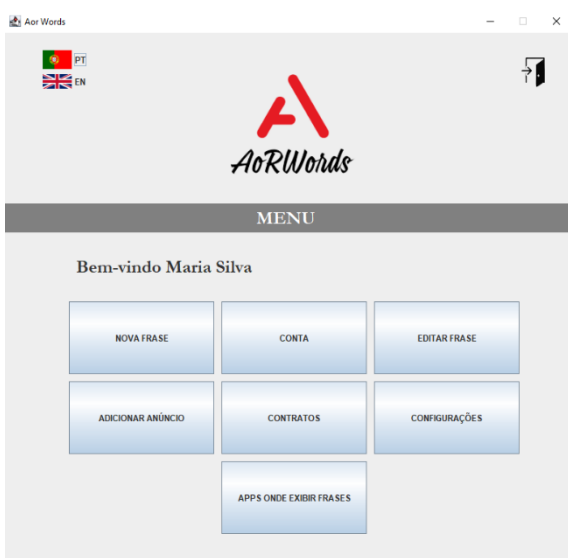


Caso se esqueça da sua informação de login, isto é, do e-mail registado, da palavra-chave, ou de ambos, pode clicar no botão das caixas de texto, designado por 'Esqueceu os dados do Login?'.



- × Ao clicar neste botão, surgirá uma nova janela com informação sobre o contacto de e-mail para o ajudar a recuperar o acesso à sua conta.

2. AoRWords para Publicitários



Após fazer o login com sucesso, deparar-se-á com o Menu.

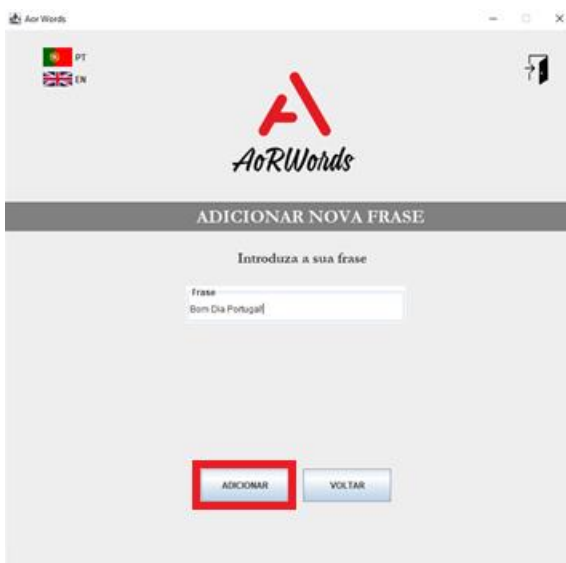
Neste, existem sete (7) botões, cada um com uma função diferente. Este menu permite-lhe aceder a todas as funcionalidades da aplicação.

2.1 Ecrã adicionar nova frase



Antes de poder começar a criar anúncios, deve começar por criar uma frase.

Se clicar no botão destacado a vermelho, surgirá o ecrã abaixo:



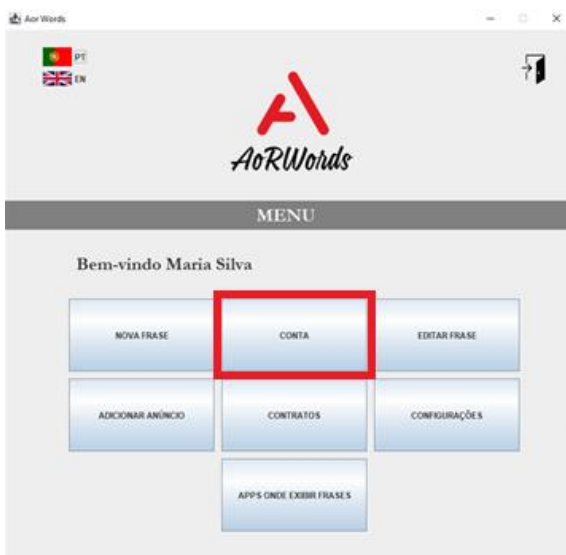
Este ecrã permite-lhe adicionar uma frase, de forma a que possa, mais tarde, criar um anúncio.

Para adicionar uma nova frase, após preencher a caixa de texto, clique no botão designado 'ADICIONAR', que se encontra destacado a vermelho.

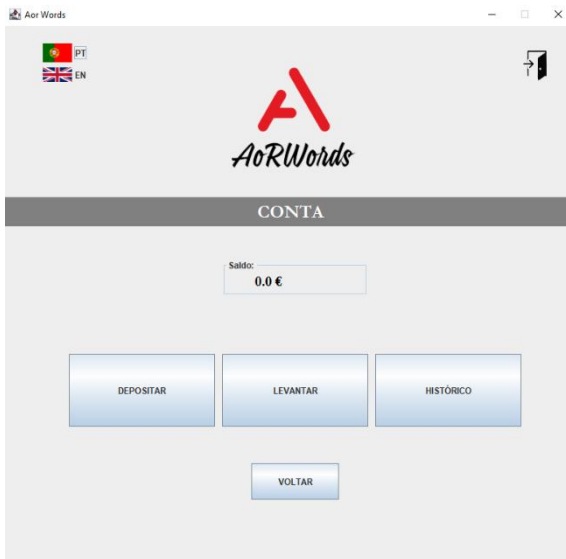


Se clicar no botão designado 'VOLTAR', que se encontra destacado a vermelho, regressa ao Menu.

2.2 Ecrã Conta Corrente



De forma a poder criar um anúncio, precisa de, em primeiro lugar, adicionar saldo à sua conta corrente AorWords. Para isso, no ecrã Menu, clique no segundo botão da primeira linha, que se encontra destacado a vermelho.



Encontra-se, agora, na sua conta corrente AoRWords.

Este ecrã permite-lhe aceder logo ao saldo de que dispõe na sua conta para poder criar anúncios.

Este ecrã dá-lhe três possibilidades de ação: depositar dinheiro da sua conta, levantar dinheiro da sua conta e verificar o histórico das transações monetárias que realizou.

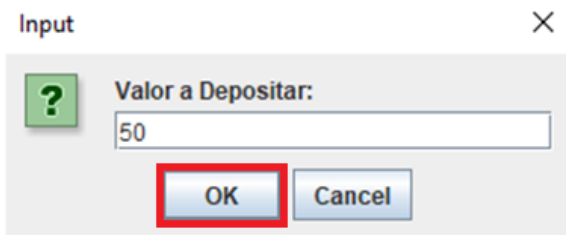
Se pretender voltar para o Menu, clique no botão designado 'VOLTAR' que se encontra à esquerda.

2.2.1 Depositar



Para efetuar um depósito, clique no botão designado 'DEPOSITAR', que se encontra destacado a vermelho.

Após clicar no botão 'DEPOSITAR', surgirá uma janela menor onde poderá indicar o valor que pretende depositar.



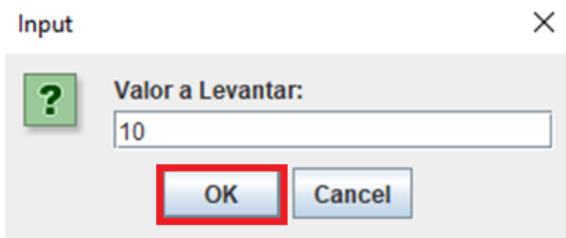
Após clicar no botão 'OK' da janela menor, caso o valor introduzido seja um valor válido, o valor introduzido na caixa de texto da janela menor será depositado na sua conta corrente AoRWords.

2.2.2 Levantar



Neste caso, a conta encontra-se com saldo, pelo que pode efetuar um levantamento. Para tal, clique no botão designado 'LEVANTAR' que se encontra destacado a vermelho.

Após clicar no botão designado 'LEVANTAR', surgirá uma janela menor onde poderá indicar o valor que pretende levantar.



Após clicar no botão 'OK' da janela menor, caso o valor introduzido seja um valor válido, o valor introduzido na caixa de texto da janela menor será retirado da sua conta corrente AoRWords.

2.2.3 Histórico de Transações



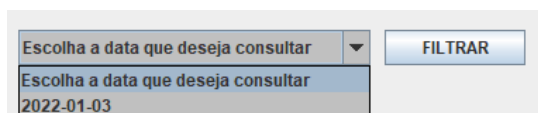
Para poder aceder ao histórico de transações que efetuou, clique no botão designado 'HISTÓRICO' que se encontra destacado a vermelho.

Após clicar no botão designado 'HISTÓRICO', surgirá um novo ecrã onde poderá ver todas as transações monetárias que efetuou.

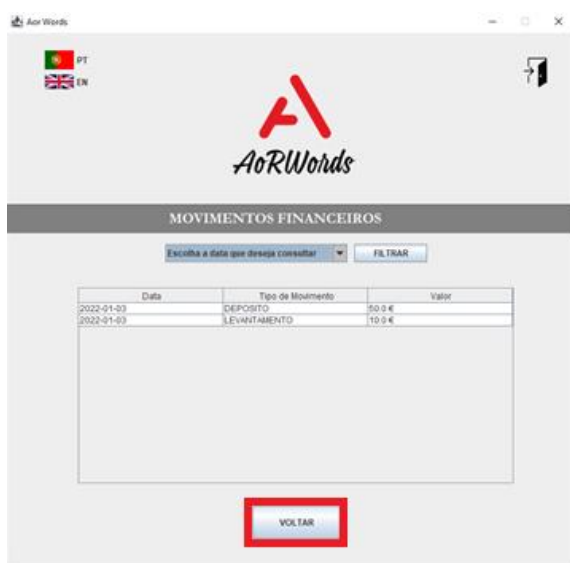


Neste ecrã pode visualizar as suas transações de forma ordenada ao clicar no título da coluna que pretende ver ordenada, podendo ordenar por ordem crescente ou decrescente, e de a-z ou z-a.

Pode também visualizar as suas transações monetárias filtradas por uma data à sua escolha. Para tal, basta clicar no retângulo cinzento acima da tabela e escolher a data que pretende. Neste retângulo cinzento surgirão apenas as datas em que se realizaram transações económicas.



Após seleccionar a data cujas transações económicas pretende visualizar, clique no botão designado 'FILTRAR' do lado direito do retângulo cinzento. Deparar-se-á com as transações monetárias que se realizaram no dia seleccionado. Pode também ordenar estes dados ao clicar no nome da coluna que pretende ordenar.



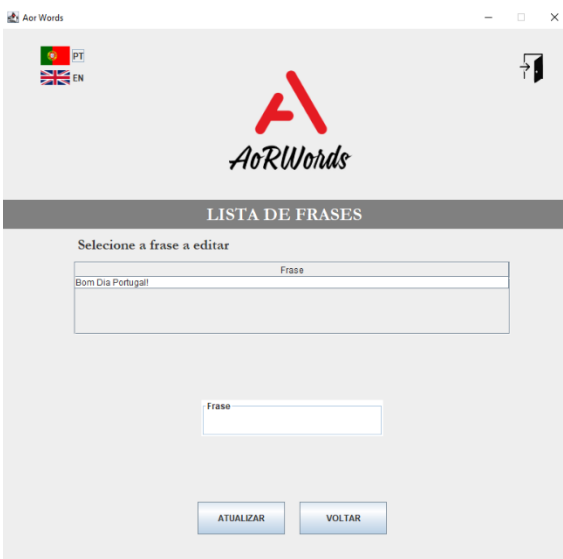
Quando terminar de visualizar as suas transações, clique no botão designado 'VOLTAR' que se encontra destacado a vermelho para voltar ao ecrã da sua conta corrente.

2.3 Ecrã Editar as suas Frases



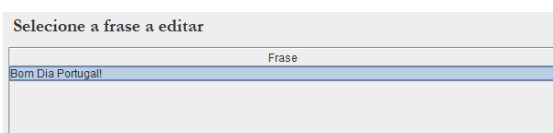
Para poder editar as frases que introduziu anteriormente, clique no botão mais à direita da primeira linha designado 'EDITAR FRASE' que se encontra destacado a vermelho.

Após clicar, deparar-se-á com o ecrã que lhe vai permitir editar as suas frases. Encontrará uma tabela com as suas frases.



Neste ecrã pode visualizar as suas frases de forma ordenada ao clicar no título da coluna que pretende ver ordenada, podendo ordenar de a-z e z-a.

Para editar as suas frases deve levar a cabo os seguintes passos:

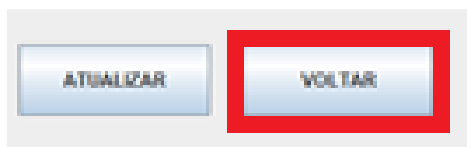


2.3.1 Deve selecionar a frase que pretende editar;



2.3.2 A frase que selecionou surgirá na caixa de texto da tabela, e pode editar a frase ao reescrever para o que pretende que a frase de torne;

2.3.3 Por fim, clique no botão designado 'ATUALIZAR' que se encontra destacado a vermelho para alterar a frase selecionada para o conteúdo que deixou escrito na caixa de texto.



Quando terminar de editar as suas frases, clique no botão designado ‘VOLTAR’ que se encontra destacado a vermelho para voltar ao ecrã do Menu.

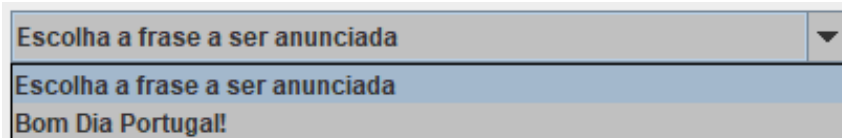
2.4 Ecrã adicionar Anúncio



Após criar a sua frase e efetuar um depósito e possuir saldo na sua conta corrente AoRWords, pode criar um anúncio. Para isso, clique no botão mais à esquerda da segunda linha de botões, que se designa ‘ADICIONAR ANÚNCIO’ e se encontra destacado a vermelho.



Após clicar, deparar-se-á com o ecrã que lhe vai permitir adicionar um novo anúncio. Encontrará uma caixa cinzenta, na qual se encontram as frases que criou; uma tabela com todas as aplicações disponíveis para efetuarem o anúncio; e uma caixa de texto na qual deve introduzir quantas vezes pretende que a sua frase seja exibida na aplicação que selecionar.



2.4.1 Escolher a frase a publicitar: em primeiro lugar, deve clicar no retângulo cinzento e selecionar qual das suas frases pretende publicitar;

Nome do Programador	Nome da Aplicação	Descrição	Valor por exibição
Clara Sousa	SIC Notícias	canal de notícias	1.1

2.4.2 De seguida, deve selecionar em que aplicação pretende anunciar a sua frase. Neste ecrã pode visualizar as aplicações de forma ordenada ao

clicar no título da coluna que pretende ver ordenada, podendo ordenar por ordem crescente ou decrescente, e de a-z ou z-a;

2.4.3 Deve ainda colocar na primeira caixa de texto, designada ‘Número de exibições’, quantas vezes pretende que a sua frase seja publicitada naquela aplicação;

5.4 A seguir, deve clicar no botão designado ‘CRIAR’, que se encontra destacado a vermelho.

2.4.5 Por fim, surgirá uma janela menor com as condições do contrato que está a estabelecer, para confirmar o contrato e criar o anúncio, clique no botão ‘Yes’ à esquerda, caso não concorde ou pretenda voltar atrás, clique no botão ‘No’ à direita.

Quando terminar de criar novos anúncios, clique no botão designado ‘VOLTAR’ que se encontra destacado a vermelho para voltar ao ecrã da sua conta corrente.

2.5 Ecrã dos contratos estabelecidos

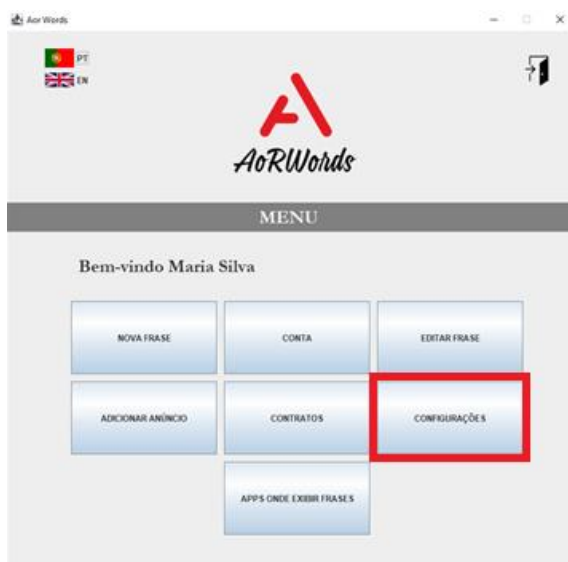
Após criar o seu primeiro anúncio, já pode ver os seus contratos estabelecidos. Para isso, clique no botão central da segunda linha de botões, que se designa ‘CONTRATOS’ e se encontra destacado a vermelho.



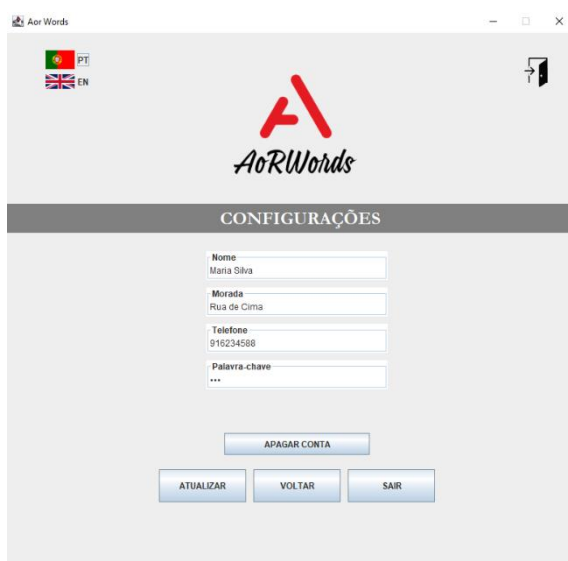
Após clicar no botão supramencionado, deparar-se-á com o ecrã que lhe vai permitir verificar os contratos estabelecidos. Encontrará uma tabela com a seguinte informação: a frase a ser anunciada, o nome da aplicação onde a frase vai ser publicitada, a data em que foi estabelecido o contrato, o valor de cada exibição naquela aplicação, o número de exibições contratas da frase naquela aplicação e o custo total que o anúncio tem para o publicitário. Neste ecrã pode visualizar os contratos que estabeleceu de forma ordenada ao clicar no título da coluna que pretende ver ordenada, podendo ordenar por ordem crescente ou decrescente, e de a-z ou z-a.

Quando terminar de visualizar os contratos estabelecidos, clique no botão designado ‘VOLTAR’ que se encontra destacado a vermelho para voltar ao ecrã do Menu.

2.6 Ecrã configurações



Se pretender editar as suas informações pessoais (nome, morada e número de telefone), pode fazê-lo ao clicar o botão mais à direita da segunda linha, que se designa ‘CONFIGURAÇÕES’ e se encontra destacado a vermelho.



Após clicar, deparar-se-á com o ecrã que lhe vai permitir verificar as suas informações pessoais. Encontrará quatro caixas de texto com o seu nome, morada, número de telefone e palavra-chave, um botão que lhe permite eliminar a sua conta e, mais abaixo, três botões, para atualizar os dados, voltar para o menu, e sair da conta, fazendo log-out.

Recomendamos que, logo após a receber os dados da sua nova conta, proceda à alteração da palavra-chave. Para alterar a palavra-chave ou qualquer outra informação exposta nas caixas de texto, basta clicar na caixa de texto e reescrever a informação para o que pretende alterar, como efetuado no exemplo:

Para guardar as alterações efetuadas, precisa de, após alterar a sua informação pessoal, clicar no botão designado ‘ATUALIZAR’ que se encontra destacado a vermelho.

Caso pretenda eliminar a sua conta, precisa de clicar no botão designado ‘APAGAR CONTA’ que se encontra destacado a vermelho.

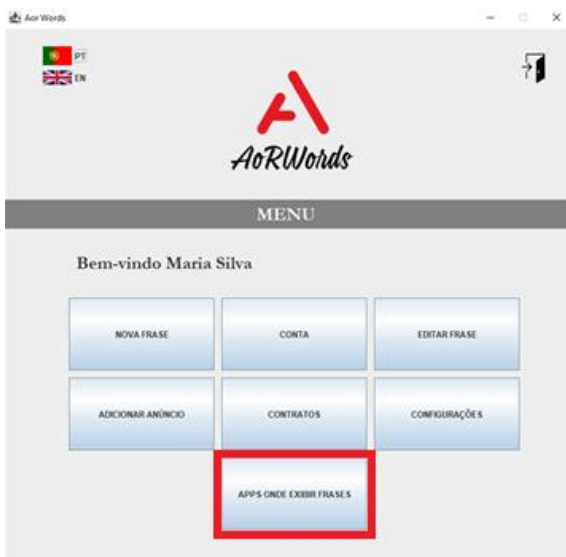
Após clicar, deparar-se-á com uma janela menor a pedir para confirmar que pretende, de facto, eliminar a sua conta.

Para eliminar a sua conta, clique no botão designado ‘Yes’ à direita para eliminar a conta, ou no botão designado ‘No’ para voltar atrás e não eliminar a conta.

Para regressar ao Menu, clique no botão designado ‘VOLTAR’.

Para sair da sua conta (log-out), clique no botão designado ‘SAIR’.

2.7 Ecrã ver as aplicações em que as frases vão ser exibidas



Após criar o seu primeiro anúncio, já pode ver em que aplicações é que a sua frase vai ser exibida. Para isso clique no botão que se encontra na terceira linha que se designa 'APPS ONDE EXIBIR FRASES' que se encontra destacado a vermelho.



Após clicar, deparar-se-á com o ecrã que lhe vai permitir selecionar qual das suas frases pretende verificar as aplicações em que esta frase vai ser anunciada. Após selecionar a frase, clique no botão designado 'PESQUISAR' que se encontra à direita.

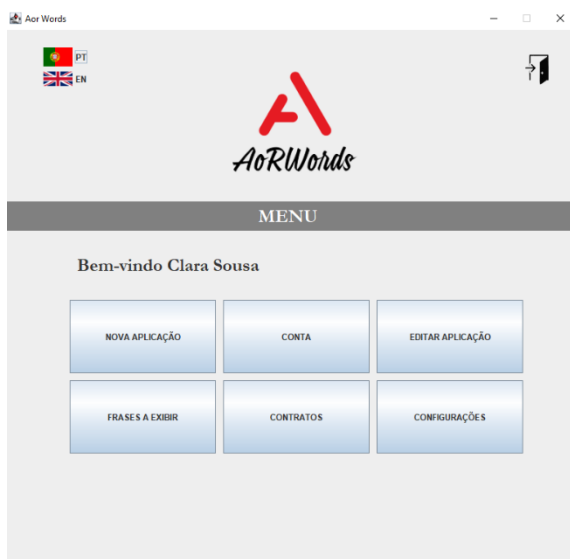
Se pretender voltar para o Menu, clique no botão designado 'VOLTAR' que se encontra à esquerda.



Após selecionar a frase e clicar no botão designado 'PESQUISAR', surgirá um novo ecrã onde poderá ver todas as aplicações em que a frase selecionada vai ser anunciada. Neste novo ecrã pode visualizar as aplicações e a sua informação de forma ordenada ao clicar no título da coluna que pretende ver ordenada, podendo ordenar por ordem crescente ou decrescente, e de a-z ou z-a.

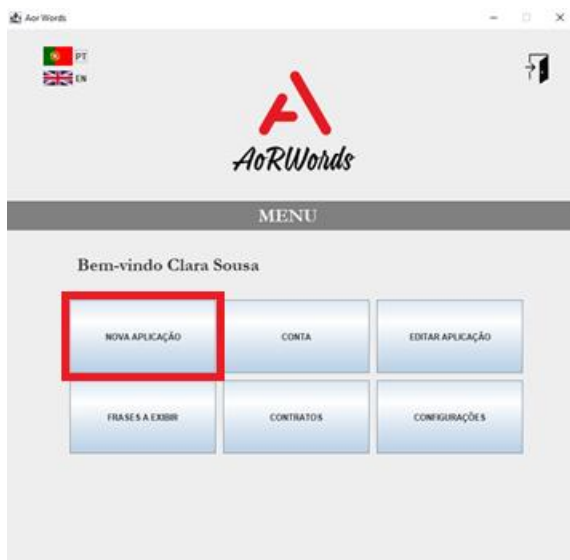
Para regressão ao ecrã para selecionar a frase a verificar, clique no botão designado 'VOLTAR' que se encontra destacado a vermelho.

3. AoRWords para Programadores

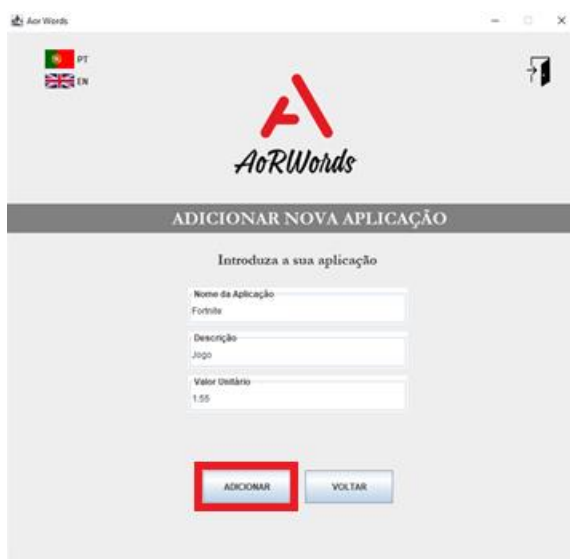


Após fazer o login com sucesso, deparar-se-á com o Menu. Neste, existem seis (6) botões, cada um com uma função diferente. Este menu permite-lhe aceder a todas as funcionalidades da aplicação.

3.1 Ecrã adicionar uma nova aplicação



O primeiro passo a fazer é criar uma aplicação de forma a que os publicitários possam criar anúncios com a sua aplicação.



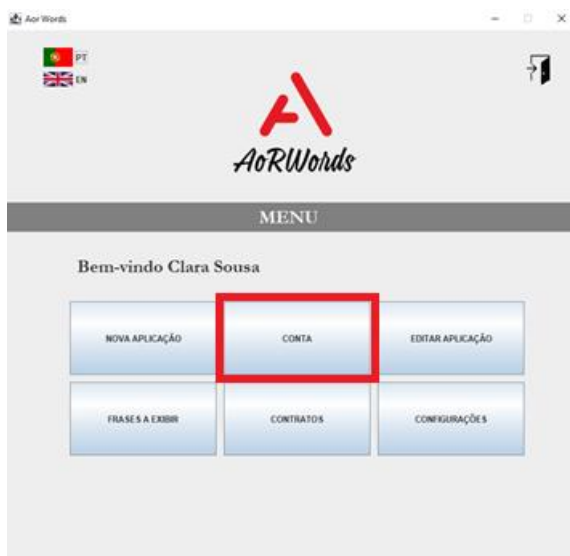
Este ecrã permite-lhe adicionar uma aplicação, de forma a que esta possa, mais tarde, ser contratada para anunciar uma frase.

Para adicionar uma nova aplicação, após preencher as caixas de texto, clique o botão designado 'ADICIONAR', que se encontra destacado a vermelho.

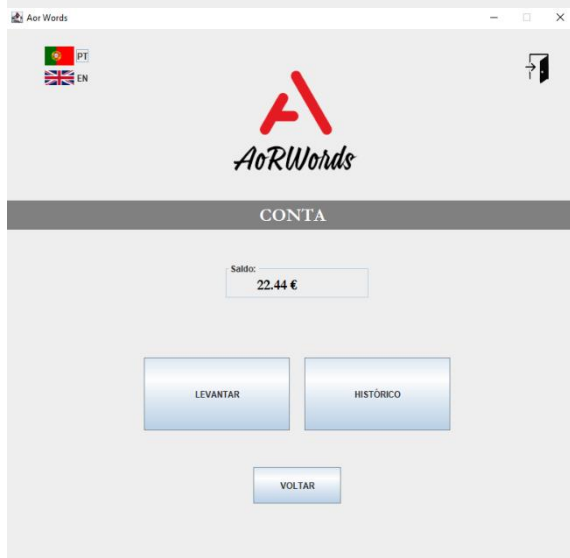


Se clicar no botão designado ‘VOLTAR’, que se encontra destacado a vermelho, regressa ao Menu.

3.2 Ecrã Conta Corrente



De forma a poder aceder à sua conta corrente AoRWords, no ecrã Menu, clique no segundo botão da primeira linha, que se encontra destacado a vermelho.



Encontra-se agora na sua conta AoRWords.

Este ecrã permite-lhe aceder logo ao saldo de que dispõe na sua conta.

Este ecrã dá-lhe duas possibilidades de ação: levantar dinheiro da sua conta e verificar o histórico de transações monetárias realizadas.

Se pretender voltar para o Menu, clique no botão designado ‘VOLTAR’ que se encontra à esquerda.

3.2.1 Levantar



Neste caso, a conta encontra-se com saldo, pelo que pode efetuar um levantamento. Para tal, clique no botão designado ‘LEVANTAR’ que se encontra destacado a vermelho.

Após clicar no botão designado ‘LEVANTAR’, surgirá uma janela menor onde poderá indicar o valor que pretende levantar.

Input

?

Valor a Levantar:

5

OK Cancel

Após clicar no botão ‘OK’ da janela menor, caso o valor introduzido seja um valor válido, o valor introduzido na caixa de texto da janela menor será retirado da sua conta corrente AoRWords.

3.2.2 Histórico de Transações

AoRWords

PT EN

AoRWords

CONTA

Saldo: 22.44 €

LEVANTAR HISTÓRICO VOLTAR

Para poder aceder ao histórico de transações que efetuou, clique no botão designado ‘HISTÓRICO’ que se encontra destacado a vermelho.

Após clicar no botão designado ‘HISTÓRICO’, surgirá um novo ecrã onde poderá ver todas as transações monetárias que efetuou.

AoRWords

PT EN

AoRWords

MOVIMENTOS FINANCEIROS

Escolha a data que deseja consultar FILTRAR

Data	Tipo de Movimento	Valor
2022-01-03	DEPOSITO	7.7 €
2022-01-03	DEPOSITO	3.85 €
2022-01-03	LEVANTAMENTO	5.0 €

VOLTAR

Neste ecrã pode visualizar as suas transações de forma ordenada ao clicar no título da coluna que pretende ver ordenada, podendo ordenar por ordem crescente ou decrescente, e de a-z ou z-a.

Pode também visualizar as suas transações monetárias filtradas por uma data à sua escolha. Para tal, basta clicar no retângulo cinzento acima da tabela e escolher a data que pretende. Neste retângulo cinzento surgirão apenas as datas em que se realizaram transações económicas.

Escolha a data que deseja consultar

FILTRAR

Escolha a data que deseja consultar

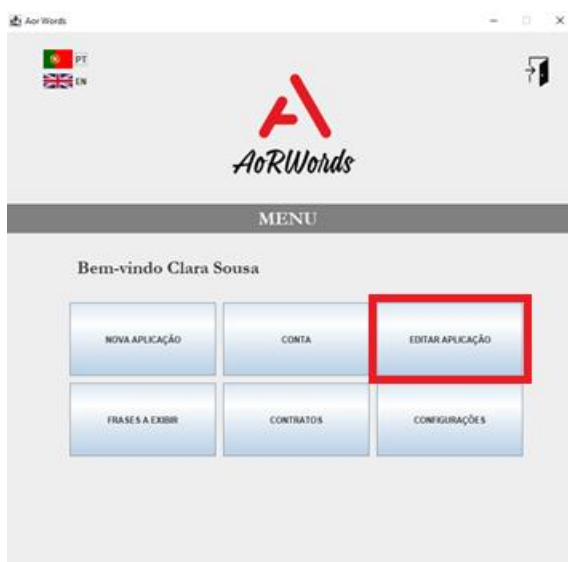
2022-01-03

Após selecionar a data cujas transações económicas pretende visualizar, clique no botão designado ‘FILTRAR’ do lado direito do retângulo cinzento. Deparar-se-á com as transações monetárias que se realizaram no dia selecionado. Pode também ordenar estes dados ao clicar no nome da coluna que pretende ordenar.



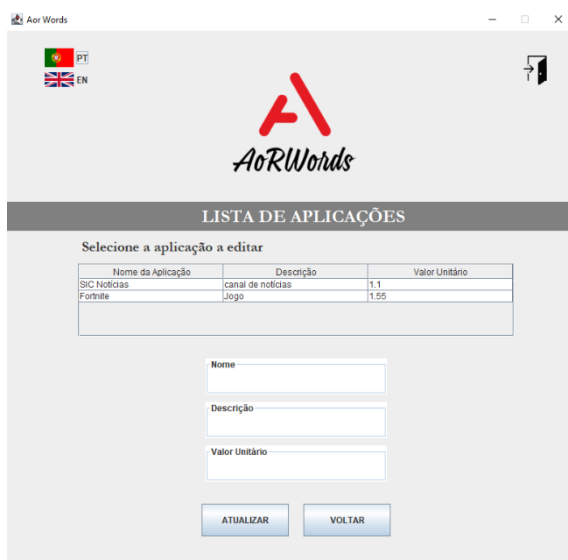
Quando terminar de visualizar as suas transações, clique no botão designado ‘VOLTAR’ que se encontra destacado a vermelho para voltar ao ecrã da sua conta corrente.

3.3 Ecrã editar aplicação



Para poder editar as aplicações que introduziu anteriormente, clique no botão mais à direita da primeira linha designado ‘EDITAR APLICAÇÃO’ que se encontra destacado a vermelho.

Após clicar, deparar-se-á com o ecrã que lhe vai permitir editar as suas aplicações. Encontrará uma tabela com as suas aplicações.



Neste ecrã pode visualizar as suas aplicações de forma ordenada ao clicar no título da coluna que pretende ver ordenada, podendo ordenar por ordem crescente ou decrescente e de a-z e z-a.

Para editar as suas aplicações deve levar a cabo os seguintes passos:

Nome da Aplicação	Descrição	Valor Unitário
SIC Notícias	canal de notícias	1.1
Fortnite	Jogo	1.55

3.3.1 Deve seleccionar a frase que pretende editar;

The screenshot shows the 'LISTA DE APLICAÇÕES' screen. At the top, there's a header with the AoRWords logo and language flags (PT, EN). Below the header, there's a section titled 'Selecione a aplicação a editar' which contains a table with the same data as the one in the previous block. Below the table, there's a form with fields for 'Nome', 'Descrição', and 'Valor Unitário'. The 'ATUALIZAR' button is highlighted with a red box.

3.3.2 A frase que seleccionou surgirá na caixa de texto abaixo da tabela, e pode editar a frase ao reescrever para o que pretende que a frase de torne;

3.3.3 Por fim, clique no botão designado 'ATUALIZAR' que se encontra destacado a vermelho para alterar a frase seleccionada para o conteúdo que deixou escrito na caixa de texto.

This image shows a close-up of two buttons: 'ATUALIZAR' and 'VOLTAR'. The 'VOLTAR' button is highlighted with a red box.

Quando terminar de editar as suas frases, clique no botão designado 'VOLTAR' que se encontra destacado a vermelho para voltar ao ecrã do Menu.

3.4 Ecrã das frases a exibir

The screenshot shows the 'MENU' screen. At the top, there's a header with the AoRWords logo and language flags (PT, EN). Below the header, there's a section titled 'MENU' with a welcome message 'Bem-vindo Clara Sousa'. Below the message, there's a grid of six buttons: 'NOVA APLICAÇÃO', 'CONTA', 'EDITAR APLICAÇÃO', 'FRASES A EXIBIR', 'CONTRATOS', and 'CONFIGURAÇÕES'. The 'FRASES A EXIBIR' button is highlighted with a red box.

Após a sua aplicação ter sido contratada, já pode ver que frases tem de publicitar na sua aplicação. Para isso clique no botão mais à esquerda que se encontra na segunda linha que se designa 'FRASES A EXIBIR' que se encontra destacado a vermelho.



Após clicar, deparar-se-á com o ecrã que lhe vai permitir selecionar qual das suas aplicações pretende verificar as frases que esta aplicação terá de anunciar. Após selecionar a aplicação, clique no botão designado ‘PESQUISAR’ que se encontra à direita.

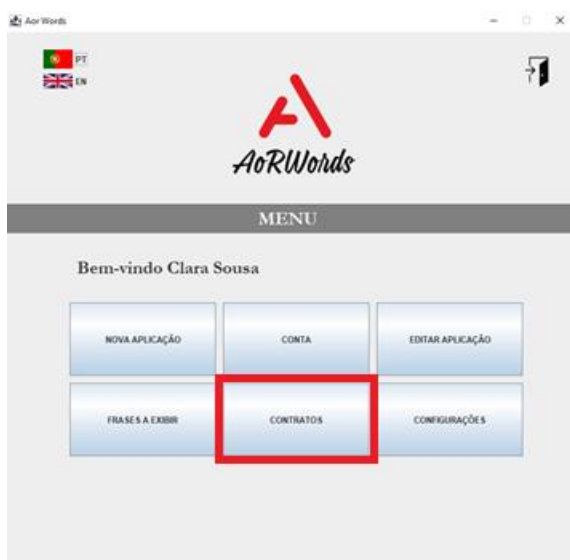
Se pretender voltar para o Menu, clique no botão designado ‘VOLTAR’ que se encontra à esquerda.



Após selecionar a aplicação e clicar no botão designado ‘PESQUISAR’, surgirá um novo ecrã onde poderá ver todas as frases que a aplicação vai anunciar. Neste novo ecrã pode visualizar as frases e a sua informação de forma ordenada ao clicar no título da coluna que pretende ver ordenada, podendo ordenar por ordem crescente ou decrescente, e de a-z ou z-a.

Para regressão ao ecrã para selecionar a aplicação a verificar, clique no botão designado ‘VOLTAR’ que se encontra destacado a vermelho.

3.5 Ecrã dos contratos estabelecidos



Após a sua aplicação ter sido contratada, já pode ver os seus contratos estabelecidos. Para isso, clique no botão central da segunda linha de botões, que se designa ‘CONTRATOS’ e se encontra destacado a vermelho.

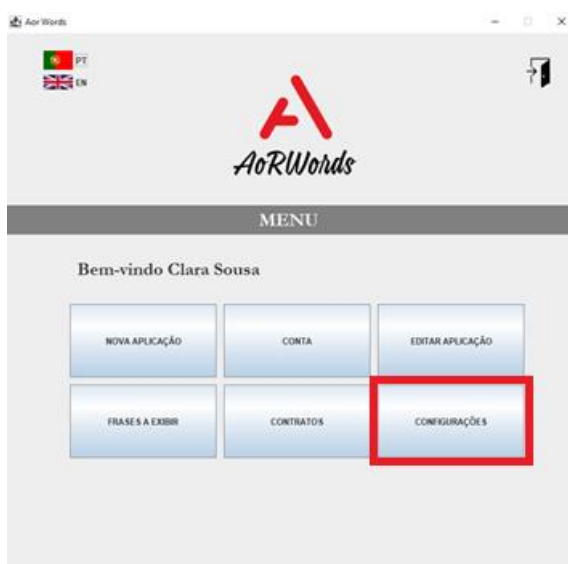


Após clicar, deparar-se-á com o ecrã que lhe vai permitir verificar os contratos estabelecidos. Encontrará uma tabela com a seguinte informação: o nome da aplicação, a frase a ser anunciada, a data em que foi estabelecido o contrato, o valor de cada exibição naquela aplicação, o número de exibições contratadas da frase naquela aplicação e as receitas que o anúncio gera ao programador dono daquela aplicação. Neste ecrã pode visualizar os contratos que estabeleceu de forma ordenada ao clicar no título da coluna que pretende ver ordenada, podendo ordenar por ordem crescente ou decrescente, e de a-z ou z-a.

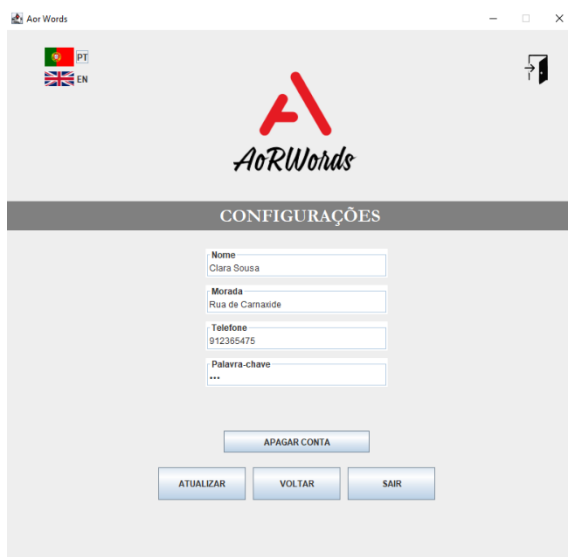
Quando terminar de visualizar os contratos estabelecidos, clique no botão designado ‘VOLTAR’ que se encontra

destacado a vermelho para voltar ao ecrã do Menu.

3.6 Ecrã configurações



Se pretender editar as suas informações pessoais (nome, morada e número de telefone), pode fazê-lo ao clicar o botão mais à direita da segunda linha, que se designa ‘CONFIGURAÇÕES’ e se encontra destacado a vermelho.



Após clicar, deparar-se-á com o ecrã que lhe vai permitir verificar as suas informações pessoais. Encontrará quatro caixas de texto com o seu nome, morada, número de telefone e palavra-chave, um botão que lhe permite eliminar a sua conta, e, mais abaixo, três botões, para atualizar os dados, voltar para o menu, e sair da conta, fazendo log-out.

Nome
Clara Sousa

Morada
Rua de Cascais

Telefone
912365475

Palavra-chave
...

Recomendamos que, logo após receber os dados da sua nova conta, proceda à alteração da palavra-chave. Para alterar a palavra-chave ou qualquer outra informação exposta nas caixas de texto, basta clicar na caixa de texto e reescrever a informação para o que pretende alterar, como efetuado no exemplo.

ATUALIZAR VOLTAR SAIR

Para guardar as alterações efetuadas, precisa de, após alterar a sua informação pessoal, clicar no botão designado ‘ATUALIZAR’ que se encontra destacado a vermelho.

AoRWords

CONFIGURAÇÕES

Nome
Clara Sousa

Morada
Rua de Cascais

Telefone
912365475

Palavra-chave
...

APAGAR CONTA

ATUALIZAR VOLTAR SAIR

Caso pretenda eliminar a sua conta, precisa de clicar no botão designado ‘APAGAR CONTA’ que se encontra destacado a vermelho.

Após clicar, deparar-se-á com uma janela menor a pedir para confirmar que pretende, de facto, eliminar a sua conta.

ATENÇÃO



?

Confirma que deseja eliminar a sua conta?

Yes No

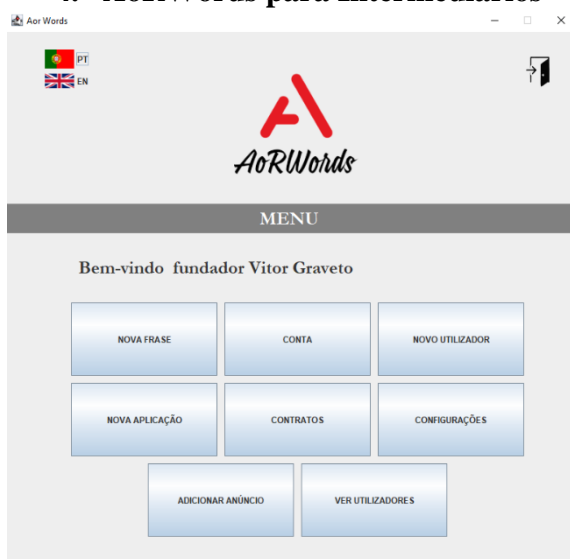
Para eliminar a sua conta, clique no botão designado ‘Yes’ à direita para eliminar a conta, ou no botão designado ‘No’ para voltar atrás e não eliminar a conta.



Para regressar ao Menu, clique no botão designado ‘VOLTAR’.

Para sair da sua conta (log-out), clique no botão designado ‘SAIR’.

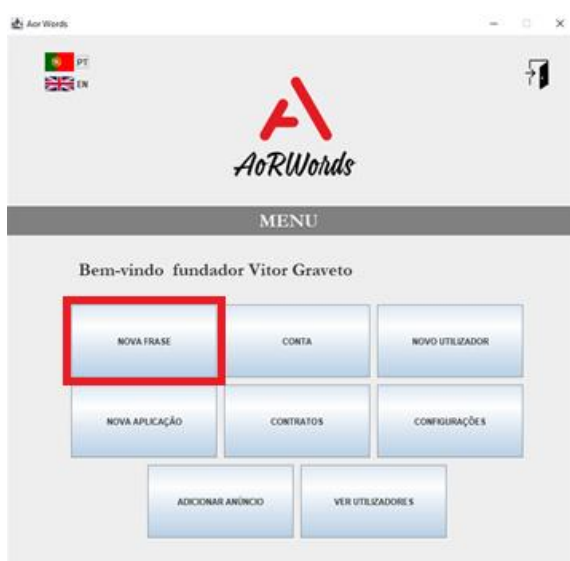
4. AoRWords para Intermediários



Após fazer o login om sucesso, deparar-se-á com o Menu.

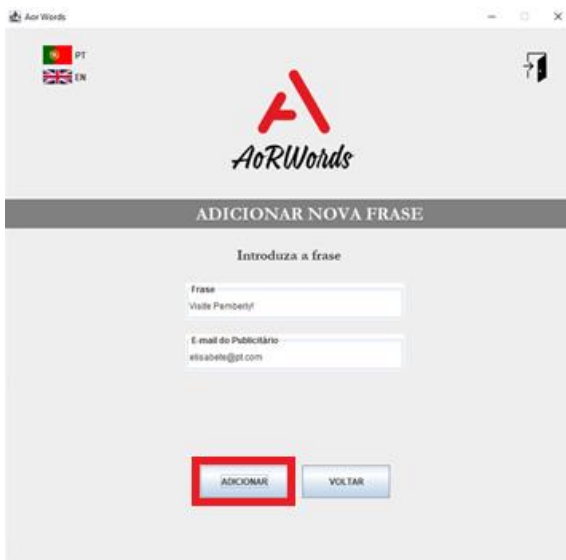
Neste, existem oito (8) botões, cada um com uma função diferente. este menu permite-lhe aceder a todas as funcionalidades da aplicação.

4.1 Ecrã adicionar nova frase



O primeiro botão que surge mais à esquerda da primeira linha e que se encontra destacado a vermelho vai permitir criar uma nova frase a um determinado publicitário.

Se clicar no botão destacado, surgirá o ecrã seguinte.



Este ecrã permite adicionar uma frase a um publicitário, de forma a que, mais tarde, este publicitário possa criar um anúncio.

Para adicionar uma nova frase, após preencher a caixa de texto e introduzir o e-mail do publicitário, clique no botão designado 'ADICIONAR', que se encontra destacado a vermelho.

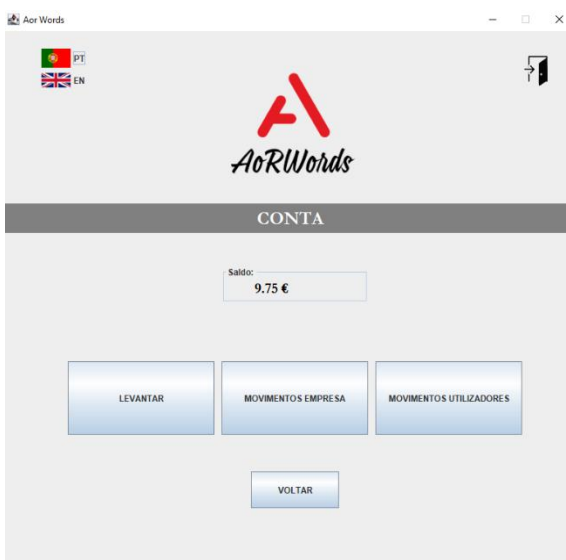


Se clicar no botão designado 'VOLTAR', que se encontra destacado a vermelho, regressa ao Menu.

4.2 Ecrã Conta Corrente

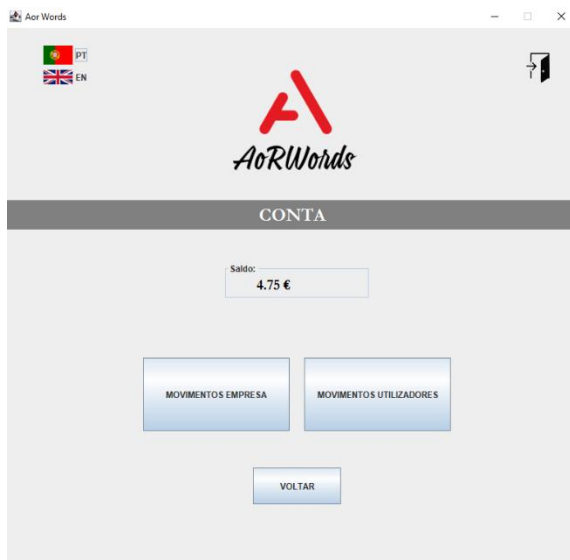


De forma a poder aceder à conta corrente AoRWords, no ecrã Menu, clique no segundo botão da primeira linha, que se encontra destacado a vermelho.



Encontra-se agora na conta da empresa AoRWords. Este ecrã permite-lhe aceder logo ao saldo de que dispõe na conta da empresa AoRWords.

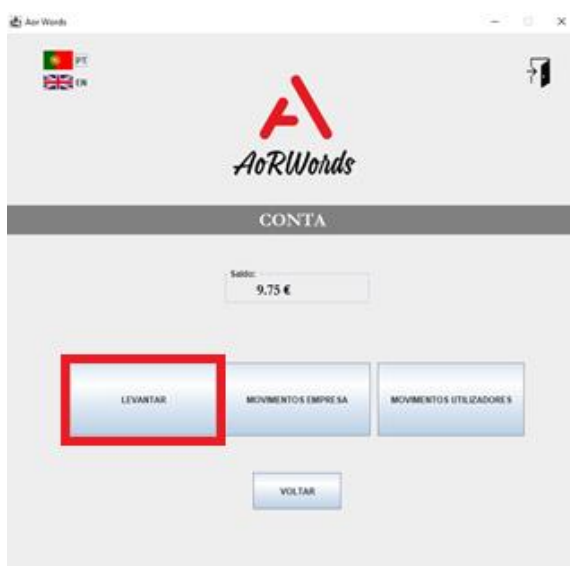
A imagem corresponde ao ecrã da conta corrente caso o utilizador com sessão iniciada seja o fundador, no entanto, se o utilizador com sessão iniciada for um intermediário, por essa razão, este ecrã dá-lhe três possibilidades de ação: levantar dinheiro da sua conta, verificar o histórico de transações monetárias que foram realizadas e verificar as transações realizadas por todos os utilizadores.



Aqui também está perante a conta da empresa AoRWords, todavia, a imagem corresponde ao ecrã da conta corrente caso o utilizador com sessão iniciada seja um intermediário comum, isto leva a que o ecrã passe a ter apenas duas funcionalidades: verificar o histórico de transações monetárias que foram realizadas e verificar as transações realizadas por todos os utilizadores.

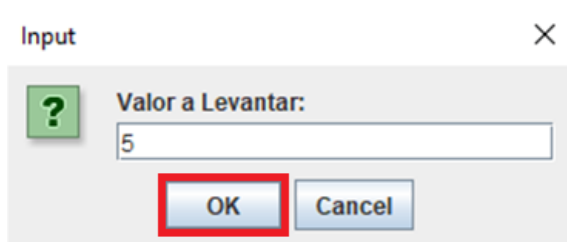
Se pretender voltar para o Menu, clique no botão designado 'VOLTAR' que se encontra à esquerda.

4.2.1 Levantar



Para efetuar um depósito, clique no botão designado 'LEVANTAR', que se encontra destacado a vermelho.

Após clicar no botão 'LEVANTAR', surgirá uma janela menor onde poderá indicar o valor que pretende depositar.



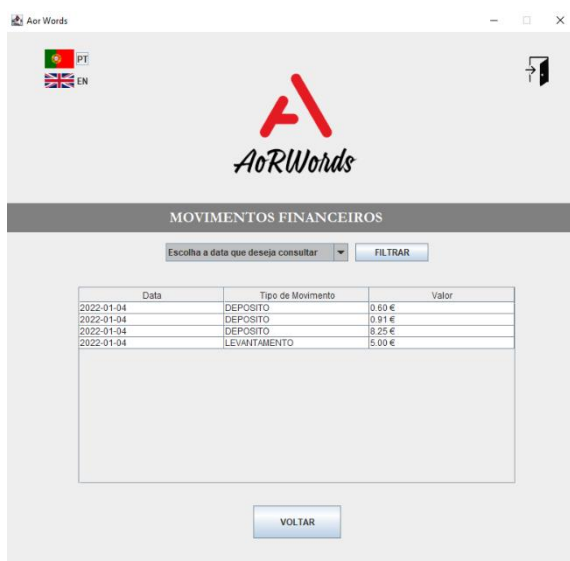
Após clicar no botão 'OK' da janela menor, caso o valor introduzido seja um valor válido, o valor introduzido na caixa de texto da janela menor será retirado da sua conta corrente AoRWords.

4.2.2 Histórico de movimentos da empresa



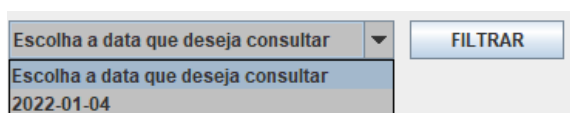
Para poder aceder ao histórico de transações que efetuadas pela empresa, clique no botão designado 'MOVIMENTOS EMPRESA' que se encontra destacado a vermelho.

Após clicar no botão designado 'MOVIMENTOS EMPRESA', surgirá um novo ecrã onde poderá ver todas as transações monetárias efetuadas pela empresa.



Neste ecrã pode visualizar as transações de forma ordenada ao clicar no título da coluna que pretende ver ordenada, podendo ordenar por ordem crescente ou decrescente, e de a-z ou z-a.

Pode também visualizar as transações monetárias filtradas por uma data à sua escolha. Para tal, basta clicar no retângulo cinzento acima da tabela e escolher a data que pretende. Neste retângulo cinzento surgirão apenas as datas em que se realizaram transações económicas.



Após selecionar a data cujas transações económicas pretende visualizar, clique no botão designado 'FILTRAR' do lado direito do retângulo cinzento. Deparar-se-á com as transações monetárias que se realizaram no dia selecionado. Pode também ordenar estes dados ao clicar no nome da coluna que pretende ordenar.



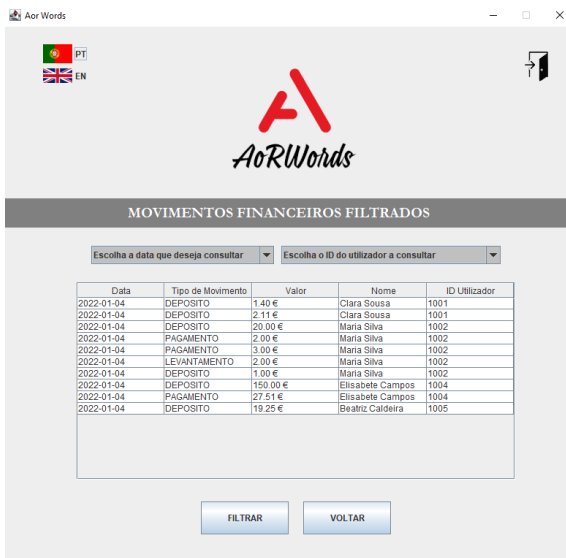
Quando terminar de visualizar as suas transações, clique no botão designado 'VOLTAR' que se encontra destacado a vermelho para voltar ao ecrã da sua conta corrente.

4.2.3 Histórico de movimentos dos utilizadores



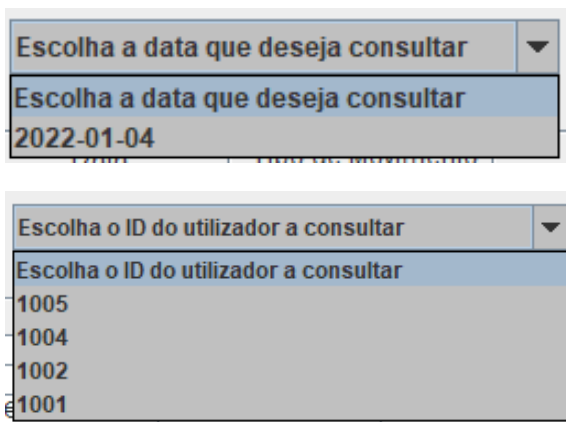
Para poder aceder ao histórico de transações que efetuadas por todos os utilizadores, clique no botão designado ‘MOVIMENTOS UTILIZADORES’ que se encontra destacado a vermelho.

Após clicar no botão designado ‘MOVIMENTOS UTILIZADORES’, surgirá um novo ecrã onde poderá ver todas as transações monetárias efetuadas Por todos os utilizadores.

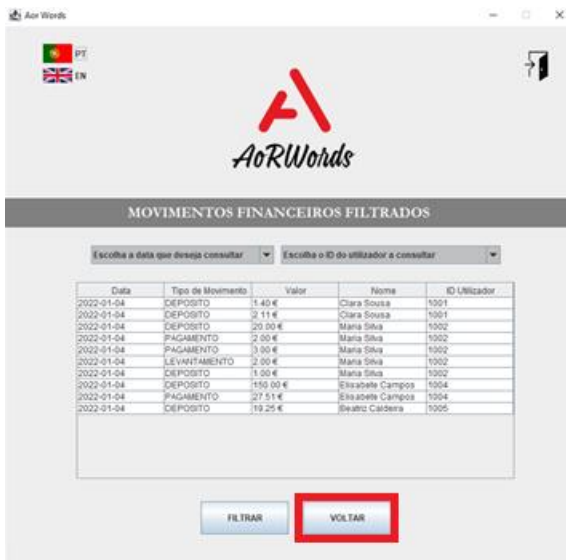


Neste ecrã pode visualizar as transações de forma ordenada ao clicar no título da coluna que pretende ver ordenada, podendo ordenar por ordem crescente ou decrescente, e de a-z ou z-a.

Pode também visualizar as transações monetárias filtradas por uma data e/ou por um utilizador à sua escolha. Para filtrar por data, basta clicar no retângulo cinzento mais à esquerda e escolher a data que pretende. Para filtrar por utilizador, clique no retângulo cinzento do lado direito do primeiro, e escolher o utilizador cujas transações pretende filtrar. No caso de pretender visualizar as transações de um utilizador numa data concreta, basta configurar ambas os retângulos.

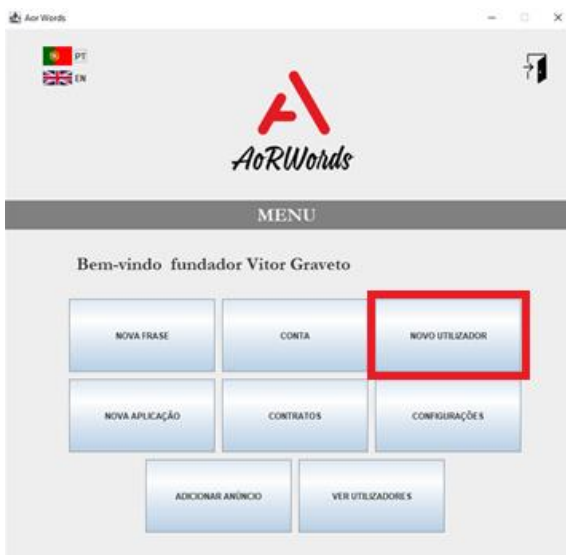


Após seleccionar a data e/ou utilizador cujas transações económicas pretende visualizar, clique no botão designado ‘FILTRAR’ do lado direito dos retângulos cinzentos. Deparar-se-á com a tabela das transações atualizada. Pode também ordenar estes dados ao clicar no nome da coluna que pretende ordenar.



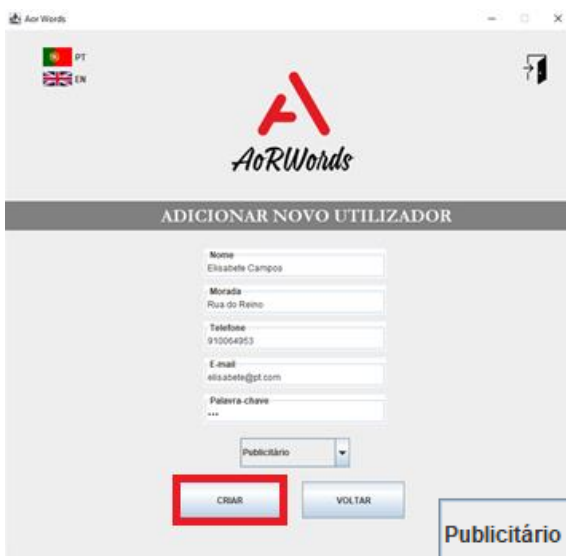
Quando terminar de visualizar as suas transações, clique no botão designado ‘VOLTAR’ que se encontra destacado a vermelho para voltar ao ecrã da sua conta corrente.

4.3 Ecrã adicionar um novo utilizador



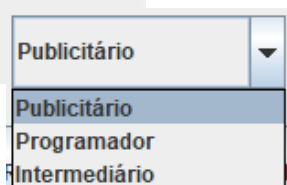
O botão que surge mais à esquerda da primeira linha e que se encontra destacado a vermelho vai permitir criar um novo utilizador.

Se clicar no botão destacado, surgirá o ecrã seguinte.



Este ecrã permite criar um novo utilizador: publicitários, para que possam criar frases e anúncios; programadores, para que possam criar aplicações que recebem os anúncios; ou intermediários, para que possam levar a cabo tarefas relacionadas com a sua jornada normal de trabalho.

Para adicionar um novo utilizador, após preencher as caixas de texto com as informações pessoais do utilizador a ser criado (como o nome, morada, número de telefone, e-mail e palavra-chave). A seguir, deve clicar no retângulo cinzento de forma a seleccionar o tipo de utilizador que está a ser criado. Por fim, clique no botão designado ‘CRIAR’, que se encontra destacado a vermelho.



Nome

Morada

Telefone

E-mail

Palavra-chave

Publicitário

criar voltar

Se clicar no botão designado ‘VOLTAR’, que se encontra destacado a vermelho, regressa ao Menu.

4.4 Ecrã adicionar uma nova aplicação

Bem-vindo fundador Vitor Graveto

NOVA FRASE CONTA NOVO UTILIZADOR

NOVA APLICAÇÃO CONTRATOS CONFIGURAÇÕES

ADICIONAR ANÚNCIO VER UTILIZADORES

O primeiro botão que surge mais à esquerda da segunda linha e que se encontra destacado a vermelho vai permitir criar uma nova frase a um determinado publicitário.

Introduza a sua aplicação

Nome da Aplicação

Descrição

Valor Unitário

E-mail do Programador

adicionar voltar

Este ecrã permite-lhe adicionar uma aplicação a um programador, de forma a que esta possa, mais tarde, esta aplicação possa ser contratada para anunciar frases.

Para adicionar uma nova aplicação, após preencher as caixas de texto e introduzir o e-mail do programador, clique no botão designado ‘ADICIONAR’, que se encontra destacado a vermelho.

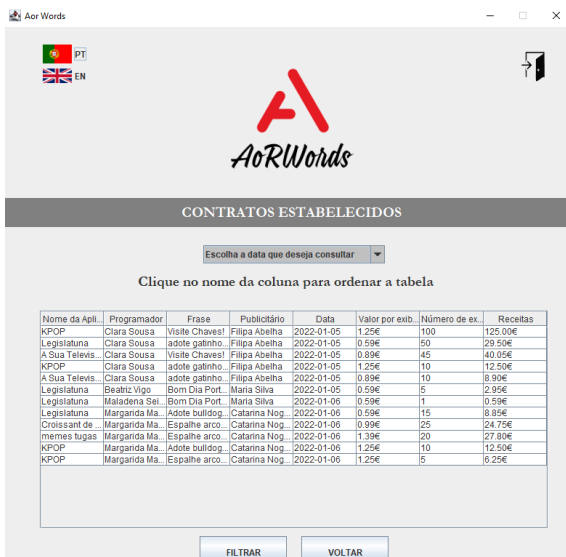
adicionar voltar

Se clicar no botão designado ‘VOLTAR’, que se encontra destacado a vermelho, regressa ao Menu.

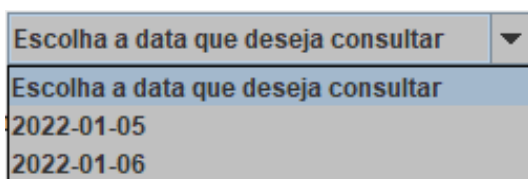
4.5 Ecrã dos contratos estabelecidos



Após já existirem anúncios criados, já pode ver os contratos estabelecidos. Para isso, clique no botão central da segunda linha de botões, que se designa 'CONTRATOS' e se encontra destacado a vermelho.



Após clicar, deparar-se-á com o ecrã que lhe vai permitir verificar os contratos estabelecidos. Encontrará uma tabela com a seguinte informação: o nome da aplicação, o nome do programador daquela aplicação, a frase a ser anunciada, o nome do publicitário daquela frase, a data em que foi estabelecido o contrato, o valor de cada exibição naquela aplicação, o número de exibições contratas da frase naquela aplicação e as receitas que o anúncio gerou para a empresa. Neste ecrã pode visualizar os contratos estabelecidos de forma ordenada ao clicar no título da coluna que pretende ver ordenada, podendo ordenar por ordem crescente ou decrescente, e de a-z ou z-a.



Pode também visualizar os contratos estabelecidos filtrados por data à sua escolha. Para tal, basta clicar no retângulo cinzento e escolher a data que pretende.

CONTRATOS ESTABELECIDOS

Escolha a data que deseja consultar

Clique no nome da coluna para ordenar a tabela

Nome da Apl.	Programador	Frase	Publisher	Data	Valor por exit	Número de ex.	Receitas
KPOP	Clara Sousa	Viajei Chaves!	Filipa Abelha	2022-01-05	1.254	100	125.004
Legislatura	Clara Sousa	adote gatinho	Filipa Abelha	2022-01-05	0.596	50	29.804
A Sua Teless	Clara Sousa	Viajei Chaves!	Filipa Abelha	2022-01-05	0.596	45	40.054
KPOP	Clara Sousa	adote gatinho	Filipa Abelha	2022-01-05	1.254	10	12.504
A Sua Teless	Clara Sousa	adote gatinho	Filipa Abelha	2022-01-05	0.596	10	5.904
Legislatura	Beatrix Vigo	Bom Dia Portugal	Maria Silva	2022-01-05	0.596	5	2.984
Legislatura	Maladrena Bel	Bom Dia Portugal	Maria Silva	2022-01-05	0.596	1	0.596
Legislatura	Marganda Ma.	Adote bulldog	Catrina Fog	2022-01-05	0.596	15	8.954
Crossant de	Marganda Ma.	Espalhe arco	Catrina Fog	2022-01-05	0.596	25	24.754
memes fofos	Marganda Ma.	Espalhe arco	Catrina Fog	2022-01-05	1.394	20	27.804
KPOP	Marganda Ma.	Adote bulldog	Catrina Fog	2022-01-05	1.254	10	12.504
KPOP	Marganda Ma.	Espalhe arco	Catrina Fog	2022-01-05	1.254	5	5.254

FILTRAR VOLTAR

Após selecionar a data cujos contratos estabelecidos pretende visualizar, clique no botão designado ‘FILTRAR’ em baixo, do lado esquerdo. Deparar-se-á com a tabela das transações atualizada. Pode também ordenar estes dados ao clicar no nome da coluna que pretende ordenar.

Quando terminar de visualizar os contratos estabelecidos, clique no botão designado ‘VOLTAR’ que se encontra em baixo, à direita, para voltar ao ecrã do Menu.

4.6 Ecrã configurações



Se pretender editar as suas informações pessoais (nome, morada e número de telefone), pode fazê-lo ao clicar o botão mais à direita da segunda linha, que se designa ‘CONFIGURAÇÕES’ e se encontra destacado a vermelho.

CONFIGURAÇÕES

Nome
Vitor Graveto

Morada
Rua Humberto Delgado

Telefone
911922933

Palavra-chave

ATUALIZAR VOLTAR SAIR

Após clicar no botão, deparar-se-á com o ecrã que lhe vai permitir verificar as suas informações pessoais. Encontrará quatro caixas de texto com o seu nome, morada, número de telefone e palavra-chave, e, mais abaixo, três botões, para atualizar os dados, voltar para o menu, e sair da conta, fazendo log-out.

Nome
Carolina Dias

Morada
Rua do Carmo

Telefone
910060045

Palavra-chave

Recomendamos que, logo após receber os dados da sua nova conta, proceda à alteração da palavra-chave. Para alterar a palavra-chave ou qualquer outra informação exposta nas caixas de texto, basta clicar na caixa de texto e reescrever a informação para o que pretende alterar, como efetuado no exemplo:

ATUALIZAR VOLTAR SAIR

Para guardar as alterações efetuadas, precisa de, após alterar a sua informação pessoal, clicar no botão designado 'ATUALIZAR' que se encontra destacado a vermelho.

AoRWords

CONFIGURAÇÕES

Nome
Vitor Graveto

Morada
Rua Humberto Delgado

Telefone
911922933

Palavra-chave

ATUALIZAR VOLTAR SAIR

Para regressar ao Menu, clique no botão designado 'VOLTAR'.

Para sair da sua conta (log-out), clique no botão designado 'SAIR'.

4.7 Ecrã adicionar novo anúncio

AoRWords

MENU

Bem-vindo fundador Vitor Graveto

NOVA FRASE CONTA NOVO UTILIZADOR

NOVA APLICAÇÃO CONTRATOS CONFIGURAÇÕES

ADICIONAR ANÚNCIO VER UTILIZADORES

De forma a criar um anúncio por um publicitário, clique no botão mais à esquerda na terceira linha de botões, que se designa 'ADICIONAR ANÚNCIO' e se encontra destacado a vermelho.

Após clicar, deparar-se-á com o ecrã que lhe vai permitir adicionar um novo anúncio. Encontrará uma caixa cinzenta, na qual se encontram todas as frases com que se podem criar novos anúncios (pois os publicitário a quem pertencem possuem saldo na sua conta corrente própria); uma tabela com todas as aplicações disponíveis para efetuarem o anúncio; e uma caixa de texto na qual deve introduzir quantas vezes pretende que a sua frase seja exibida na aplicação que selecionar.

4.7.1 1 Escolher a frase a publicitar: em primeiro lugar, deve clicar no retângulo cinzento e selecionar qual das suas frases pretende publicitar;

4.7.2 De seguida, deve selecionar em que aplicação pretende anunciar a sua frase. Neste ecrã pode visualizar as aplicações de forma ordenada ao clicar no título da coluna que pretende ver ordenada, podendo ordenar por ordem crescente ou decrescente, e de a-z ou z-a.;

Nome do Programador	Nome da Aplicação	Descrição	Valor por exibição
Clara Sousa	Organizador de estudos	Ferramenta que ajuda estu...	1.0
Beatriz Caldeira	KPOP	notícias das últimas tendê...	0.55

4.7.3 Deve ainda colocar na primeira caixa de texto, designada 'Número de exibições', quantas vezes pretende que a sua frase seja publicitada naquela aplicação;

4.7.4 A seguir, deve clicar no botão designado 'CRIAR', que se encontra destacado a vermelho.

4.7.5 Por fim, surgirá uma janela menor com as condições do contrato que está a estabelecer, para confirmar o contrato e criar o anúncio, clique no botão 'Yes' à esquerda, caso não concorde ou pretenda voltar atrás, clique no botão 'No' à direita.



Quando terminar de criar novos anúncios, clique no botão designado ‘VOLTAR’ que se encontra destacado a vermelho para voltar ao ecrã da sua conta corrente.

4.8 Ecrã visualizar utilizadores registados



De forma a visualizar os utilizadores que se encontram registados na aplicação AoRWords, bem como consultar as suas informações, clique no botão mais à direita na terceira linha de botões, que se designa ‘VER UTILIZADORES’ e se encontra destacado a vermelho.



Após clicar no botão, deparar-se-á com o ecrã que lhe vai permitir visualizar todos os utilizadores registados na aplicação. Encontrará uma tabela com a seguinte informação do utilizador: nome, morada, número de telefone, e-mail, número de identificação único do utilizador e o tipo de utilizador. Neste ecrã pode visualizar os utilizadores de forma ordenada ao clicar no título da coluna que pretende ver ordenada, podendo ordenar por ordem crescente ou decrescente, e de a-z ou z-a.

Quando terminar de visualizar os utilizadores registados, clique no botão designado ‘VOLTAR’ que se encontra destacado a vermelho para voltar ao ecrã do Menu.

CAPÍTULO 10

Listagem do Programa

10.1 Classe Anuncio

```
package programa;
```

```
import java.io.Serializable;  
import java.time.LocalDate;
```

```
//Anna Helena Drumond - 2021151911  
//Joana Valente - 2021152666
```

```
/**  
 * Classe que permite criar uma nova instância de Anuncio com as informações necessárias.  
 * É instanciada no ecrãAddAnuncio quando o utilizador confirma que pretende estabelecer contrato na popUp.  
 */
```

```
public class Anuncio implements Serializable {
```

```
    protected double valorAnuncio;  
    protected double receitaProgramador;  
    protected double receitaEmpresa;  
    protected double valorUnitario;  
    protected LocalDate dataContrato;  
    protected int numExibicoes;  
    protected Publicitario anunciante;  
    protected Aplicacao aplicacao;  
    protected Frase fraseAnunciada;  
    protected Programador anunciador;
```

```
    /**  
     * Construtor.  
     * Permite criar uma nova instância de Anuncio tendo em conta a informação.  
     * Obtém a data atual e coloca dentro do atributo dataContrato.  
     * Guarda o valor unitário naquele momento da aplicação que vai receber o anúncio em valorUnitario.  
     *  
     * @param numExibicoes número de exibições desejado para aquela frase, introduzido pelo utilizador na classe  
     EcraAddAnuncio.  
     * @param valorAnuncio custo total do anúncio a pagar pelo publicitário.  
     * @param aplicacao instância de Aplicacao, corresponde à aplicação que vai publicitar a frase.  
     * @param fraseAnunciada instância de Frase, corresponde à frase que vai ser publicitada.  
     * @param anunciante instância de Publicitario, corresponde ao utilizador que vai publicitar a frase.  
     * @param anunciador instância de Programador, corresponde ao programador que vai publicitar a frase na sua  
     aplicação.  
     */
```

```
    public Anuncio(int numExibicoes, double valorAnuncio, Aplicacao aplicacao, Frase fraseAnunciada, Publicitario anunciante,  
Programador anunciador) { //todo receber aplicacoes onde se vai anunciar  
        this.numExibicoes = numExibicoes;  
        this.valorAnuncio = valorAnuncio;  
        this.aplicacao = aplicacao;  
        this.anunciante = anunciante;  
        this.anunciador = anunciador;  
        this.fraseAnunciada = fraseAnunciada;  
        this.dataContrato = java.time.LocalDate.now();  
        this.valorUnitario = aplicacao.getValorUnitarioPorExibicao();  
    }
```



```

/**
 * Getter do atributo valorAnuncio.
 *
 * @return retorna o custo total do anúncio.
 */
public double getValorAnuncio() {
    return valorAnuncio;
}

/**
 * Getter do atributo receitaProgramador.
 *
 * @return retorna a receita que o programador obteve com o anúncio.
 */
public double getReceitaProgramador() {
    return receitaProgramador;
}

/**
 * Setter do atributo receitaProgramador.
 *
 * @param receitaProgramador modifica a receita que o programador obteve com o anúncio.
 */
public void setReceitaProgramador(double receitaProgramador) {
    this.receitaProgramador = receitaProgramador;
}

/**
 * Setter do atributo receitaEmpresa.
 *
 * @param receitaEmpresa modifica a receita que a empresa obteve com o anúncio.
 */
public void setReceitaEmpresa(double receitaEmpresa) {
    this.receitaEmpresa = receitaEmpresa;
}

/**
 * Getter do atributo dataContrato.
 *
 * @return retorna a data em que o contrato foi estabelecido.
 */
public LocalDate getDataContrato() {
    return dataContrato;
}

/**
 * Getter do atributo numExibicoes.
 *
 * @return retorna o número de vezes contratadas pelo publicitário para a frase ser publicitada.
 */
public int getNumExibicoes() {
    return numExibicoes;
}

/**
 * Getter do atributo anunciante.
 *

```

```

    * @return retorna a instância de Publicitario que corresponde ao utilizador que criou o anúncio.
    */
    public Publicitario getAnunciante() {
        return anunciante;
    }

    /**
     * Getter do atributo aplicacao.
     *
     * @return retorna a instância de Aplicacao que corresponde à aplicação em que a frase vai ser publicitada.
     */
    public Aplicacao getAplicacao() {
        return aplicacao;
    }

    /**
     * Getter do atributo fraseAnunciada.
     *
     * @return retorna a instância de Frase que corresponde à frase que vai ser publicitada.
     */
    public Frase getFraseAnunciada() {
        return fraseAnunciada;
    }

    /**
     * Getter do atributo anunciador.
     *
     * @return retorna a instância de Programador que corresponde ao utilizador que praticar o anúncio na sua aplicação.
     */
    public Programador getAnunciador() {
        return anunciador;
    }

    /**
     * Getter do atributo valorUnitario.
     *
     * @return retorna o valor unitário, aquando do estabelecimento do contrato, para anunciar a frase uma vez naquela aplicação.
     */
    public double getValorUnitario() {
        return valorUnitario;
    }
}

```

10.2 Classe Aplicacao

```
package programa;
```

```
import java.io.Serializable;
```

```
//Anna Helena Drumond - 2021151911
//Joana Valente - 2021152666
```

```

/**
 * Classe que permite criar uma nova instância de Aplicacao com as informações necessárias.
 * É instanciada no ecrãAddAplicacao quando o utilizador clica no botão 'adicionar' da mesma classe.
 */
public class Aplicacao implements Serializable {

```

```

protected String nome;
protected String descricao;
protected double valorUnitarioPorExibicao;

/**
 * Construtor.
 * Permite criar uma nova instância de Aplicacao tendo em conta a informação.
 *
 * @param nome           nome que a aplicação criada possui.
 * @param descricao      breve texto a apresentar e descrever de que se trata aquela aplicação.
 * @param valorUnitarioPorExibicao custo por cada exibição.
 */
public Aplicacao(String nome, String descricao, double valorUnitarioPorExibicao) {
    this.nome = nome;
    this.descricao = descricao;
    this.valorUnitarioPorExibicao = valorUnitarioPorExibicao;
}

/**
 * Getter do atributo nome.
 *
 * @return retorna o nome escolhido para aquela aplicação.
 */
public String getNome() {
    return nome;
}

/**
 * Setter do atributo nome.
 *
 * @param nome modifica o nome da aplicação.
 */
public void setNome(String nome) {
    this.nome = nome;
}

/**
 * Getter do atributo descricao.
 *
 * @return retorna o texto de apresentação e descrição daquela aplicação.
 */
public String getDescricao() {
    return descricao;
}

/**
 * Setter do atributo descricao.
 *
 * @param descricao modifica o texto de descrição daquela aplicação.
 */
public void setDescricao(String descricao) {
    this.descricao = descricao;
}

/**
 * Getter do atributo valorUnitarioPorExibicao.

```

```

*
* @return retorna o valor por cada publicitação efetuada naquela aplicação.
*/
public double getValorUnitarioPorExibicao() {
    return valorUnitarioPorExibicao;
}

/**
 * Setter do atributo valorUnitarioPorExibicao.
 *
 * @param valorUnitarioPorExibicao modifica o valor por para publicitação efetuada naquela aplicação.
 */
public void setValorUnitarioPorExibicao(double valorUnitarioPorExibicao) {
    this.valorUnitarioPorExibicao = valorUnitarioPorExibicao;
}
}

```

10.3 Classe Conta

package programa;

```

import java.io.Serializable;
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.ArrayList;

```

```

//Anna Helena Drumond - 2021151911
//Joana Valente - 2021152666

```

```

/**
 * Classe que permite criar uma nova instância de Conta com a informação do utilizador a quem esta pertence.
 * Classe que é instanciada sempre que um novo utilizador do tipo programador ou publicitario é adicionado.
 * Classe é instanciada aquando da criação do fundador, correspondendo à conta da empresa.
 */

```

```

public class Conta implements Serializable {

```

```

    protected Utilizador user;
    protected double saldo;
    protected ArrayList<MovimentoFinanceiro> listaMovimentosFinanceiros;

```

```

/**
 * Construtor.
 * Permite criar uma nova instância de Conta.
 * Aquando da criação da conta, esta não possui ainda saldo nem qualquer movimento.
 *
 * @param user instância de Utilizador que corresponde ao dono da conta.
 */
public Conta(Utilizador user) {
    this.user = user;
    this.saldo = 0.0;
    this.listaMovimentosFinanceiros = new ArrayList<>(10);
}

```

```

/**
 * Getter do atributo user.
 *
 * @return retorna o utilizador a quem pertence a conta.
 */

```

```

public Utilizador getUser() {
    return user;
}

/**
 * Getter do atributo saldo.
 *
 * @return retorna o dinheiro que se encontra, à ordem, naquela conta.
 */
public double getSaldo() {
    saldo = Double.parseDouble(new BigDecimal(saldo).setScale(2, RoundingMode.UP).toString());
    return saldo;
}

/**
 * Getter do atributo ArrayList listaMovimentosFinanceiros.
 *
 * @return retorna a lista de movimentos daquela conta.
 */
public ArrayList<MovimentoFinanceiro> getListaMovimentosFinanceiros() {
    return listaMovimentosFinanceiros;
}

/**
 * Adiciona um novo movimento financeiro à listaMovimentosFinanceiros daquele utilizador.
 * @param movimentoFinanceiro corresponde ao movimento financeiro criado sempre que existe uma nova transação monetária.
 */
public void adicionarMovFinanceiros(MovimentoFinanceiro movimentoFinanceiro) {
    listaMovimentosFinanceiros.add(movimentoFinanceiro);
}

/**
 * Adiciona ao saldo atual o valor introduzido pelo publicitário ou o valor das receitas obtidas com o anúncio (no caso dos programados e empresa).
 *
 * @param valorADepositar corresponde ao valor introduzido pelo publicitário ou o valor calculado nos métodos efetuarContasAnuncioEmpresa e efetuarContasAnuncioProgramador da classe EmpresaAorWords.
 */
public void efetuarDeposito(double valorADepositar) {
    saldo = saldo + valorADepositar;
    MovimentoFinanceiro movimentoFinanceiro = new MovimentoFinanceiro(TipoDeMovimento.DEPOSITO, valorADepositar);
    adicionarMovFinanceiros(movimentoFinanceiro);
}

/**
 * Retira ao saldo atual daquele utilizador o custo do anúncio.
 *
 * @param valorAPagar corresponde ao custo total do anúncio.
 */
public void efetuarPagamento(double valorAPagar) {
    saldo = saldo - valorAPagar;
    MovimentoFinanceiro movimentoFinanceiro = new MovimentoFinanceiro(TipoDeMovimento.PAGAMENTO, valorAPagar);
    adicionarMovFinanceiros(movimentoFinanceiro);
}

```

```

/**
 * Retira ao saldo atual o valor introduzido pelo utilizador programador, publicitário ou intermediário fundador.
 *
 * @param valorALevantar corresponde ao valor introduzido que vai ser retirado da conta.
 */
public void efetuarLevantamento(double valorALevantar) {
    saldo = saldo - valorALevantar;
    MovimentoFinanceiro movimentoFinanceiro = new MovimentoFinanceiro(TipoDeMovimento.LEVANTAMENTO,
valorALevantar);
    adicionarMovFinanceiros(movimentoFinanceiro);
}
}

```

10.4 Classe EmpresaAorWords

`package` programa;

`import` ficheiros.FicheiroDeObjeto;
`import` ficheiros.FicheiroDeTexto;

`import` javax.swing.*;
`import` java.io.File;
`import` java.io.IOException;
`import` java.io.Serializable;
`import` java.math.BigDecimal;
`import` java.math.RoundingMode;
`import` java.util.ArrayList;

//Anna Helena Drumond - 2021151911
//Joana Valente - 2021152666

```

/**
 * Classe gestora do programa.
 * Possui os atributos que armazenam em memória os dados do Ficheiro de Configuração.
 * Lê os dados do Ficheiro Geral, onde fica guardado o ficheiro de objetos do programa.
 * Possui o atributo que permite guardar o utilizador ativo, bem os atributos que permitem guardar o ecrã e o Jpanel ativo.
 * Possui o atributo booleano que armazena a informação do idioma. Instancia um objeto FicheiroDeTexto e um objeto
FicheiroDeObjeto.
 */

```

`public class` EmpresaAorWords `implements` Serializable {

```

    protected ArrayList<Utilizador> listaUtilizadores;
    protected ArrayList<Conta> listaContas;
    protected ArrayList<Anuncio> listaAnuncios;
    protected File ficheiroGeral;
    protected String emailFundador;
    protected boolean idiomaPortugues;
    protected Utilizador utilizadorAtivo;
    protected JPanel ecraAtivo;
    protected JPanel painelAtivo;
    protected ArrayList<String> dadosConfiguracao;
    protected FicheiroDeObjeto ficheiroDeObjeto = new FicheiroDeObjeto();
    protected FicheiroDeTexto ficheiroDeTexto = new FicheiroDeTexto();

```

```

/**
 * Construtor.
 * Instancia o ArrayList de contas, o de utilizadores e o de anúncios.
 * Chama o método extrairDadosFicheiroDeConfiguracao().

```

```

* Chama o método lerListasDoFicheiroGeral().
*/
public EmpresaAorWords() {
    listaContas = new ArrayList<>(10);
    listaUtilizadores = new ArrayList<>(10); //boa prática começar arraylist com 10 ou 20 para a memória estar a contar
    listaAnuncios = new ArrayList<>(10);
    dadosConfiguracao = new ArrayList<>(10);
    extrairDadosFicheiroDeConfiguracao();
    if (ficheiroGeral.exists()) {
        lerListasDoFicheiroGeral();
    }
}

/**
 * Getter do atributo listaUtilizadores.
 *
 * @return retorna o ArrayList da lista de todos os utilizadores registados.
 */
public ArrayList<Utilizador> getListaUtilizadores() {
    return listaUtilizadores;
}

/**
 * Getter do atributo utilizadorAtivo.
 *
 * @return retorna o utilizador que se encontra que está, naquele momento, com sessão iniciada.
 */
public Utilizador getUtilizadorAtivo() {
    return utilizadorAtivo;
}

/**
 * Setter do atributo utilizadorAtivo.
 *
 * @param utilizadorAtivo modifica o utilizador que se encontra a utilizar a aplicação naquele momento.
 */
public void setUtilizadorAtivo(Utilizador utilizadorAtivo) {
    this.utilizadorAtivo = utilizadorAtivo;
}

/**
 * Getter do atributo idiomaPortugues.
 *
 * @return retorna boolean com valor booleano true se o programa se encontra em português e false se se encontra em inglês.
 */
public boolean isIdiomaPortugues() {
    return idiomaPortugues;
}

/**
 * Setter do atributo idiomaPortugues.
 *
 * @param idiomaPortugues modifica a língua em que o programa se encontra.
 */
public void setIdiomaPortugues(boolean idiomaPortugues) {
    this.idiomaPortugues = idiomaPortugues;
}

```

```

/**
 * Getter do atributo ecraAtivo.
 *
 * @return retorna o Ecrã que está, neste momento, a ser mostrado ao utilizador.
 */
public JPanel getEcraAtivo() {
    return ecraAtivo;
}

/**
 * Setter do atributo EcraAtivo.
 *
 * @param ecraAtivo modifica o ecrã que naquele momento está a ser mostrado ao utilizador.
 */
public void setEcraAtivo(JPanel ecraAtivo) {
    this.ecraAtivo = ecraAtivo;
}

/**
 * Getter do atributo painelAtivo.
 *
 * @return retorna o Painel que está, neste momento, a ser mostrado ao utilizador.
 */
public JPanel getPainelAtivo() {
    return painelAtivo;
}

/**
 * Setter do atributo painelAtivo.
 *
 * @param painelAtivo modifica o painel que naquele momento está a ser mostrado ao utilizador.
 */
public void setPainelAtivo(JPanel painelAtivo) {
    this.painelAtivo = painelAtivo;
}

/**
 * Getter do atributo listaContas.
 *
 * @return retorna o ArrayList da lista de todas as contas criadas.
 */
public ArrayList<Conta> getListaContas() {
    return listaContas;
}

/**
 * Getter do atributo ficheiroDeTexto.
 *
 * @return retorna a instância da classe de FicheiroDeTexto.
 */
public FicheiroDeTexto getFicheiroDeTexto() {
    return ficheiroDeTexto;
}

/**
 * Getter do atributo listaAnuncios
 *

```



```

* @return retorna o ArrayList da lista de todos os anúncios que foram criados.
*/
public ArrayList<Anuncio> getListaAnuncios() {
    return listaAnuncios;
}

/**
* Getter do atributo emailFundador.
* @return retorna o e-mail do utilizador fundador da empresa AoRWords recolhido do ficheiro de configuração.
*/
public String getEmailFundador() {
    return emailFundador;
}

public ArrayList<String> getDadosConfiguracao() {
    return dadosConfiguracao;
}

/**
* Extrai os dados do Ficheiro de Configuração.
* Armazena estes dados em atributos da Classe EmpresaAorWords.
*/
public void extrairDadosFicheiroDeConfiguracao() {

    try {
        ficheiroDeTexto.abrirLeitura("FicheiroDeConfiguracao.dat");
        String auxiliarDeLinha;

        while ((auxiliarDeLinha = ficheiroDeTexto.lerLinha()) != null) {
            dadosConfiguracao.add(auxiliarDeLinha);
        }
        ficheiroDeTexto.fecharLeitura();
    } catch (IOException erro) {
        erro.printStackTrace();
    }

    // Deixei como atributo somente aquilo que acedemos em diversas outras classes ou escopos.
    // E assim, consegui tirar alguns métodos getter que se mostraram desnecessários.
    emailFundador = dadosConfiguracao.get(3); //email fundador
    ficheiroGeral = new File(dadosConfiguracao.get(5)); //nome ficheiro de objetos
    idiomaPortugues = Boolean.parseBoolean(dadosConfiguracao.get(8)); //idioma

    criarUtilizadorEContaDoFundador();

    System.out.println("Extraí dados ficheiro de configuração");
}

/**
* Cria uma instância de Intermediário com os dados do fundador.
* Cria uma instância de Conta, criando a conta da empresa administrada pelo fundador.
* Adiciona a instância de intermediário criada à lista de utilizadores
*/
public void criarUtilizadorEContaDoFundador() {
    Utilizador fundador = new Intermediario(dadosConfiguracao.get(0), dadosConfiguracao.get(1), dadosConfiguracao.get(2),
    emailFundador, dadosConfiguracao.get(4));
    Conta contaEmpresa = new Conta(fundador);
    listaUtilizadores.add(fundador);
}

```

```

    if (listaContas.size() == 0) {
        listaContas.add(contaEmpresa);
    }
}

/**
 * Atualiza a contagem inicial do iterador que gera os Ids dos utilizadores.
 * Chama o método static buscarValorAtualizadoParaAlterador passando este valor
 * atualizado como parâmetro.
 */
public void atualizarIterador() {
    int novoValor=listaUtilizadores.get(0).getIdUtilizador();

    for (Utilizador utilizador:listaUtilizadores){
        if (novoValor<=utilizador.getIdUtilizador()){
            novoValor=utilizador.getIdUtilizador()+1;
        }
    }
    Utilizador.buscarValorAtualizadoParaAlterador(novoValor);
}

/**
 * Lê os dados do ficheiro de objetos do programa.
 * Armazena estes dados em memória nas respetivas listas.
 */
public void lerListasDoFicheiroGeral() { // mandar paramteros

    try {
        ficheiroDeObjeto.abreLeitura(ficheiroGeral);

        listaUtilizadores = (ArrayList<Utilizador>) ficheiroDeObjeto.leObjeto();
        listaContas = (ArrayList<Conta>) ficheiroDeObjeto.leObjeto();
        listaAnuncios = (ArrayList<Anuncio>) ficheiroDeObjeto.leObjeto();

        ficheiroDeObjeto.fechaLeitura();

        System.out.println("leu ficheiro Geral");
        for (Utilizador utilizador : listaUtilizadores) {
            System.out.println(utilizador);
        }
        for (Conta conta : listaContas) {
            System.out.println(conta);
        }
        for (Anuncio anuncio : listaAnuncios) {
            System.out.println(anuncio);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

/**
 * Adiciona os utilizadores inseridos no programa dentro da ArrayList de utilizadores.
 *
 * @param utilizador instância de Programador/Publicitário/Intermediário criada em EcraAddUtilizador
 */
public void adicionarUtilizadorNaArrayList(Utilizador utilizador) {
    if (utilizador != null) {
        listaUtilizadores.add(utilizador);
    }
}

```

```

    }
}

/**
 * Adiciona a conta criada aquando da criação de um novo utilizador para a ArrayList de Contas
 *
 * @param conta instância de Conta criada em EcraAddUtilizador
 */

public void adicionarContaNaArrayList(Conta conta) {
    if (conta != null) {
        listaContas.add(conta);
    }
}

/**
 * Chama o método efetuarContasAnuncioEmpresa para calcular as receitas que a empresa gerou com o novo anúncio.
 * Chama o método efetuarContasAnuncioProgramador para calcular as receitas que o programador gerou com o novo
anúncio.
 * Chama o método efetuarContasAnuncioPublicitario para calcular o custo que o publicitário possui ao criar o novo anúncio.
 * Faz set das receitas geradas pelo programador e pela empresa com o anúncio.
 * Adiciona o anúncio à listaAnuncios.
 *
 * @param anuncio instância de Anuncio criada em EcraAddAnuncio.
 * @param programador instância de Programador criada em EcraAddAnuncio.
 * @param publicitario instância de Publicitario criada em EcraAddAnuncio.
 * @param custoTotal custo total do anúncio calculado no método verificarAcaoTextFieldQuantidadeExibicoes da classe
EcraAddAnuncio.
 */
public void adicionarAnuncioNaArrayListEAdicionarMovimentos(Anuncio anuncio, Programador programador, Publicitario
publicitario, double custoTotal) {

    if (anuncio != null) {

        double receitaEmpresa = efetuarContasAnuncioEmpresa(custoTotal);

        double receitaProgramador = 0.0;

        for (Conta conta : listaContas) {
            if (conta.getUser().equals(programador)) {

                receitaProgramador = efetuarContasAnuncioProgramador(custoTotal, conta);

            } else if (conta.getUser().equals(publicitario)) {

                efetuarContasAnuncioPublicitario(custoTotal, conta);
            }
        }
        anuncio.setReceitaEmpresa(receitaEmpresa);
        anuncio.setReceitaProgramador(receitaProgramador);
        listaAnuncios.add(anuncio);
    }
}

/**
 * Calcula o volume de receitas que a empresa gera com a criação daquele anúncio.
 * Chama o método efetuarDeposito da classe Conta.
 *
 * @param custoTotal custo total do anúncio calculado no método verificarAcaoTextFieldQuantidadeExibicoes da classe
EcraAddAnuncio.

```

```

* @return retorna as receitas que a empresa gerou com a criação daquele anúncio.
*/
public double efetuarContasAnuncioEmpresa(double custoTotal) {

    double percentagemEmpresa = Double.parseDouble(dadosConfiguracao.get(10)); ///% empresa
    double valor = Double.parseDouble(String.valueOf(new BigDecimal(custoTotal * percentagemEmpresa).setScale(2,
RoundingMode.UP)));

    for (Conta elemento : listaContas) {
        if (elemento.getUser().getEmail().equals(emailFundador)) { // porque somente o fundador tem a conta da empresa
            elemento.efetuarDeposito(valor);
        }
    } // porque é preciso converter primeiro o big decimal em 'string' e só depois converter essa 'string' em double
    return valor;
}

/**
* Calcula o volume de receitas que o programador gera com a criação daquele anúncio.
* Chama o método efetuarDeposito da classe Conta.
*
* @param custoTotal    custo total do anúncio calculado no método verificarAcaoTextFieldQuantidadeExibicoes da classe
EcraAddAnuncio.
* @param contaProgramador instância de Conta que corresponde à conta do programador.
* @return retorna as receitas que o programador gerou com a criação daquele anúncio.
*/
public double efetuarContasAnuncioProgramador(double custoTotal, Conta contaProgramador) {

    double percentagemProgramador = Double.parseDouble(dadosConfiguracao.get(9)); ///% programador

    double valor = Double.parseDouble(String.valueOf(new BigDecimal(custoTotal * percentagemProgramador).setScale(2,
RoundingMode.UP)));

    contaProgramador.efetuarDeposito(valor);
    return valor;
}

/**
* Chama o método efetuarPagamento da classe Conta.
*
* @param custoTotal    custo total do anúncio calculado no método verificarAcaoTextFieldQuantidadeExibicoes da classe
EcraAddAnuncio.
* @param contaPublicitario instância de Conta que corresponde à conta do publicitário.
*/
public void efetuarContasAnuncioPublicitario(double custoTotal, Conta contaPublicitario) {
    contaPublicitario.efetuarPagamento(custoTotal);
}

/**
* Escreve a informação que se encontra nas ArrayLists listaUtilizadores, listaContas e listaAnuncios no ficheiro
* de objetos quando se sai do programa.
*/

public void escreverNoFicheiroGeralASaida() {
    try {
        ficheiroDeObjeto.abreEscrita(ficheiroGeral);

        ficheiroDeObjeto.escreveObjeto(listaUtilizadores);
        ficheiroDeObjeto.escreveObjeto(listaContas);
        ficheiroDeObjeto.escreveObjeto(listaAnuncios);

        ficheiroDeObjeto.fechaEscrita();
    }
}

```

```

    } catch (Exception e) {
        e.printStackTrace();
    }
    System.out.println("escreveu ficheiro geral");
}

/**
 * Verifica o último idioma escolhido pelo utilizador.
 * Atualiza a lista dadosConfiguracao onde estão armazenados os dados do ficheiro de configuração.
 */
public void atualizarIdiomaDefault(){
    if (isIdiomaPortugues()) {
        dadosConfiguracao.set(8, String.valueOf(true));
    } else {
        dadosConfiguracao.set(8, String.valueOf(false));
    }
}

/**
 * Reescreve o ficheiro de configuração com todas as atualizações de dados realizadas em tempo de execução.
 */
public void atualizarFicheiroDeConfiguracao() {

    atualizarIdiomaDefault();

    try {
        ficheiroDeTexto.abrirEscrita(dadosConfiguracao.get(7));
        for (String elemento : dadosConfiguracao) {
            ficheiroDeTexto.escreverLinha(elemento);
        }
        ficheiroDeTexto.fecharEscrita();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

/**
 * Atualiza os dados do fundador na lista dadosConfiguracao.
 * É chamado na Classe Configurações, caso o fundador opte em atualizar/alterar algum dos seus dados pessoais.
 */
public void atualizarDadosFundador() {

    for (int i = 0; i < dadosConfiguracao.size(); i++) {
        if (i == 0 && !dadosConfiguracao.get(0).equals(listaUtilizadores.get(0).getNome())) {
            dadosConfiguracao.set(0, listaUtilizadores.get(0).getNome());
        }
        if (i == 1 && !dadosConfiguracao.get(1).equals(listaUtilizadores.get(0).getMorada())) {
            dadosConfiguracao.set(1, listaUtilizadores.get(0).getMorada());
        }
        if (i == 2 && !dadosConfiguracao.get(2).equals(listaUtilizadores.get(0).getTelefone())) {
            dadosConfiguracao.set(2, listaUtilizadores.get(0).getTelefone());
        }
        if (i == 4 && !dadosConfiguracao.get(4).equals(listaUtilizadores.get(0).getPassword())) {
            dadosConfiguracao.set(4, listaUtilizadores.get(0).getPassword());
        }
    }
}
}

```

10.5 Classe Frase

```
package programa;
```

```
import java.io.Serializable;
```

```
//Anna Helena Drumond - 2021151911
```

```
//Joana Valente - 2021152666
```

```
/**  
 * Classe permite criar uma nova instância de Frase com a informação introduzida pelo utilizador na classe EcraAddFrase.  
 * Classe é instanciada sempre que o utilizador (Publicitário ou Intermediário) clica no botão 'adicionar' na classe  
 EcraAddFrase.  
 */
```

```
public class Frase implements Serializable {
```

```
    protected String frase;
```

```
    /**  
     * Construtor.  
     * Permite criar uma nova instância de Frase.  
     *  
     * @param frase corresponde à frase que o utilizador introduziu na classe EcraAddFrase.  
     */
```

```
    public Frase(String frase) {  
        this.frase = frase;  
    }
```

```
    /**  
     * Getter do atributo frase.  
     *  
     * @return retorna a frase introduzida pelo utilizador.  
     */
```

```
    public String getFrase() {  
        return frase;  
    }
```

```
    /**  
     * Setter do atributo frase.  
     *  
     * @param frase modifica a frase, permitindo que esta seja atualizada.  
     */
```

```
    public void setFrase(String frase) {  
        this.frase = frase;  
    }
```

```
}
```

10.6 Classe Intermediário

```
package programa;
```

```
import java.io.Serializable;
```

```
//Anna Helena Drumond - 2021151911
```

```
//Joana Valente - 2021152666
```

```
/**  
 * Extends Utilizador.  
 * Permite criar um tipo de Utilizador, tendo o objeto a mesma informação que a classe Utilizador, todavia,
```

** permite diferenciar o tipo de interface a que aquele utilizador concreto é exposto.*
** É instanciada na classe EcraAddUtilizador sempre o intermediário seleciona a opção 'Intermediário' da JComboBox*
** e clica em criar, ou é instanciado o fundador.*
**/*

```
public class Intermediario extends Utilizador implements Serializable {

    /**
     * Construtor.
     * Permite criar uma nova instância de Utilizador do tipo Intermediário.
     *
     * @param nome corresponde ao nome do utilizador a ser criado que o intermediário introduziu no EcraAddUtilizador ou
     que foi recolhido através da leitura do ficheiro de configuração.
     * @param morada corresponde à morada do utilizador a ser criado que o intermediário introduziu no EcraAddUtilizador
     ou que foi recolhido através da leitura do ficheiro de configuração.
     * @param email corresponde ao e-mail do utilizador a ser criado que o intermediário introduziu no EcraAddUtilizador ou
     que foi recolhido através da leitura do ficheiro de configuração.
     * @param telefone corresponde ao número de telefone do utilizador a ser criado que o intermediário introduziu no
     EcraAddUtilizador ou que foi recolhido através da leitura do ficheiro de configuração.
     * @param password corresponde à password da conta do utilizador a ser criado que o intermediário introduziu no
     EcraAddUtilizador ou que foi recolhido através da leitura do ficheiro de configuração.
     */

    public Intermediario(String nome, String morada, String email, String telefone, String password) {
        super(nome, morada, email, telefone, password);
    }
}
```

10.7 Classe MovimentoFinanceiro

```
package programa;
```

```
import java.io.Serializable;
import java.time.LocalDate;
```

```
//Anna Helena Drumond - 2021151911
//Joana Valente - 2021152666
```

```
/**
 * Classe que permite criar uma nova instância de MovimentoFinanceiro com a informação do valor da transação e o tipo de
 movimento realizado.
 * Classe é instanciada sempre que existe alguma transação, permitindo registar os pormenores dos trocas monetárias
 * entre os utilizadores ou o levantamento e/ou depósito na conta.
 */
```

```
public class MovimentoFinanceiro implements Serializable {

    protected LocalDate data;
    protected TipoDeMovimento tipoMovimento;
    protected double valor;

    /**
     * Construtor.
     * Permite criar uma nova instância de MovimentoFinanceiro.
     * Obtém a data atual e coloca dentro do atributo data.
     *
     * @param tipoMovimento corresponde a um dos tipos contido dentro do enumerador TipoDeMovimento.
     * @param valor corresponde ao valor que vai ser transacionado.
     */

    public MovimentoFinanceiro(TipoDeMovimento tipoMovimento, double valor) {
        this.data = java.time.LocalDate.now();
        this.tipoMovimento = tipoMovimento;
    }
}
```

```

    this.valor = valor;
}

/**
 * Getter do atributo data.
 * @return retorna a data em que o movimento financeiro foi realizado.
 */
public LocalDate getData() {
    return data;
}

/**
 * Getter do atributo tipoMovimento.
 * @return retorna o tipo de movimento financeiro realizado, que se encontra estipulado no enumerador
TipoDeMovimento.
 */
public TipoDeMovimento getTipoMovimento() {
    return tipoMovimento;
}

/**
 * Getter do atributo valor.
 * @return retorna o valor que foi transacionado aquando do movimento financeiro.
 */
public double getValor() {
    return valor;
}
}

```

10.8 Classe Program (Main)

```

package programa;
import gui.FrameUnica;

//Anna Helena Drumond - 2021151911
//Joana Valente - 2021152666

/**
 * Classe principal.
 * Instancia um objeto do tipo EmpresaAorWords e um objeto do tipo FrameUnica.
 * Permite iniciar a leitura dos ficheiros bem como a interface gráfica.
 */

public class Program {

    public static void main(String[] args) {

        EmpresaAorWords empresaObjeto = new EmpresaAorWords();
        FrameUnica frameUnica = new FrameUnica(empresaObjeto, empresaObjeto.getFicheiroDeTexto());

    }
}

```

10.9 Classe Programador

```

package programa;

import java.io.Serializable;

```



```
import java.util.ArrayList;
```

```
//Anna Helena Drumond - 2021151911
```

```
//Joana Valente - 2021152666
```

```
/**  
 * Extends Utilizador.  
 * Permite criar um tipo de Utilizador, tendo o objeto a mesma informação que a classe Utilizador, todavia,  
 * permite diferenciar o tipo de interface a que aquele utilizador concreto é exposto.  
 * É instanciada na classe EcraAddUtilizador sempre o intermediário seleciona a opção 'Programador' da JComboBox  
 * e clica em criar.  
 */
```

```
public class Programador extends Utilizador implements Serializable {
```

```
    protected ArrayList<Aplicacao> listaAplicacoes;
```

```
    /**  
     * Construtor.  
     * Permite criar uma nova instância de Utilizador do tipo Programador.  
     * Inicializa o Arraylist listaAplicacoes.  
     *  
     * @param nome corresponde ao nome do utilizador a ser criado que o intermediário introduziu no EcraAddUtilizador.  
     * @param morada corresponde à morada do utilizador a ser criado que o intermediário introduziu no EcraAddUtilizador.  
     * @param email corresponde ao e-mail do utilizador a ser criado que o intermediário introduziu no EcraAddUtilizador.  
     * @param telefone corresponde ao número de telefone do utilizador a ser criado que o intermediário introduziu no  
     EcraAddUtilizador.  
     * @param password corresponde à password da conta do utilizador a ser criado que o intermediário introduziu no  
     EcraAddUtilizador.  
     */
```

```
    public Programador(String nome, String morada, String email, String telefone, String password) {  
        super(nome, morada, email, telefone, password);  
        this.listaAplicacoes = new ArrayList<> (10);  
    }
```

```
    /**  
     * Getter do atributo Arraylist listaAplicacoes.  
     * @return retorna a lista de aplicações daquele programador.  
     */  
    public ArrayList<Aplicacao> getListaAplicacoes() {  
        return listaAplicacoes;  
    }
```

```
    /**  
     * Adiciona uma nova aplicação à listaAplicacoes daquele programador.  
     * @param aplicacao corresponde à aplicação criada no EcraAddAplicacao sempre o Intermediário/Programador clica no  
     botão 'adicionar'.  
     */  
    public void adicionarApps(Aplicacao aplicacao) {  
        if (aplicacao != null) {  
            listaAplicacoes.add(aplicacao);  
        }  
        System.out.println(listaAplicacoes);  
    }  
}
```

10.10 Classe Publicitario

```
package programa;
```

```

import java.io.Serializable;
import java.util.ArrayList;

//Anna Helena Drumond - 2021151911
//Joana Valente - 2021152666

/**
 * Extends Utilizador.
 * Permite criar um tipo de Utilizador, tendo o objeto a mesma informação que a classe Utilizador, todavia,
 * permite diferenciar o tipo de interface a que aquele utilizador concreto é exposto.
 * É instanciada na classe EcraAddUtilizador sempre o intermediário seleciona a opção 'Publicitário' da JComboBox
 * e clica em criar.
 */
public class Publicitario extends Utilizador implements Serializable {
    private static final long serialVersionUID = 1L;

    protected ArrayList<Frase> listaFrases;

    /**
     * Construtor.
     * Permite criar uma nova instância de Utilizador do tipo Publicitario.
     * Inicializa o ArrayList listaFrases.
     *
     * @param nome corresponde ao nome do utilizador a ser criado que o intermediário introduziu no EcraAddUtilizador.
     * @param morada corresponde à morada do utilizador a ser criado que o intermediário introduziu no EcraAddUtilizador.
     * @param email corresponde ao e-mail do utilizador a ser criado que o intermediário introduziu no EcraAddUtilizador.
     * @param telefone corresponde ao número de telefone do utilizador a ser criado que o intermediário introduziu no
     EcraAddUtilizador.
     * @param password corresponde à password da conta do utilizador a ser criado que o intermediário introduziu no
     EcraAddUtilizador.
     */
    public Publicitario(String nome, String morada, String email, String telefone, String password) {
        super(nome, morada, email, telefone, password);
        this.listaFrases = new ArrayList<> (10);
    }

    /**
     * Getter do atributo ArrayList listaFrases.
     * @return retorna a lista de frases daquele publicitário.
     */
    public ArrayList<Frase> getListaFrases() {
        return listaFrases;
    }

    /**
     * Adiciona uma nova frase à listaFrases daquele publicitário.
     * @param frase corresponde à frase criada na classe EcraAddFrase sempre que o Publicitário/Intermediário clica no botão
     'adicionar'.
     */
    public void adicionarFrases(Frase frase) {
        if (frase != null) {
            listaFrases.add(frase);
        }
        System.out.println(listaFrases);
    }
}

```

10.11 Enumerador TipoDeMovimento

```

package programa;

//Anna Helena Drumond - 2021151911
//Joana Valente - 2021152666

/**
 * Enumerador usado pela classe MovimentoFinanceiro.
 */
public enum TipoDeMovimento {

    DEPOSITO,
    LEVANTAMENTO,
    PAGAMENTO
}

```

10.12 Classe Abstrata Utilizador

```

package programa;

import java.io.Serializable;

//Anna Helena Drumond - 2021151911
//Joana Valente - 2021152666

/**
 * Superclasse de Publicitário, Programador e Intermediário. Permite criar uma lista de Utilizadores na classe
 * EmpresaAorWords.
 * Contém os atributos comuns a todos os utilizadores, incluindo um número de identificação único para cada utilizador.
 */
public abstract class Utilizador implements Serializable {

    protected String nome;
    protected String morada;
    protected String telefone;
    protected int idUtilizador;
    protected String email;
    protected String password;
    protected static int geradorId = 1000;

    /**
     * Construtor.
     * Contém os dados comuns a todos os tipos de utilizadores.
     * Implementa o iterador do gerador de 'id'.
     *
     * @param nome corresponde ao nome do utilizador a ser criado que o intermediário introduziu no EcraAddUtilizador.
     * @param morada corresponde à morada do utilizador a ser criado que o intermediário introduziu no EcraAddUtilizador.
     * @param email corresponde ao e-mail do utilizador a ser criado que o intermediário introduziu no EcraAddUtilizador.
     * @param telefone corresponde ao número de telefone do utilizador a ser criado que o intermediário introduziu no
     * EcraAddUtilizador.
     * @param password corresponde à password da conta do utilizador a ser criado que o intermediário introduziu no
     * EcraAddUtilizador.
     */

    public Utilizador(String nome, String morada, String telefone, String email, String password) {
        this.nome = nome;
        this.morada = morada;
        this.telefone = telefone;
        this.email = email;
        this.idUtilizador = geradorId++;
        this.password = password;
    }
}

```

```

/**
 * Getter do atributo nome.
 * @return retorna o nome do utilizador.
 */
public String getNome() {
    return nome;
}

/**
 * Setter do atributo nome.
 * @param nome modifica o nome do utilizador ativo.
 */
public void setNome(String nome) {
    this.nome = nome;
}

/**
 * Getter do atributo morada.
 * @return retorna a morada do utilizador.
 */
public String getMorada() {
    return morada;
}

/**
 * Setter do atributo morada.
 * @param morada modifica o morada do utilizador ativo.
 */
public void setMorada(String morada) {
    this.morada = morada;
}

/**
 * Getter do atributo telefone.
 * @return retorna o número de telefone do utilizador.
 */
public String getTelefone() {
    return telefone;
}

/**
 * Setter do atributo telefone.
 * @param telefone modifica o número de telefone do utilizador ativo.
 */
public void setTelefone(String telefone) {
    this.telefone = telefone;
}

/**
 * Getter do atributo email.
 * @return retorna o email do utilizador.
 */
public String getEmail() {
    return email;
}

```

```

/**
 * Getter do atributo password.
 * @return retorna a palavra-chave do utilizador.
 */
public String getPassword() {
    return password;
}

/**
 * Setter do atributo password.
 * @param password modifica a palavra-chave do utilizador ativo.
 */
public void setPassword(String password){
    this.password=password;
}

/**
 * Getter do atributo idUtilizador.
 * @return retorna o número de identificação único do utilizador.
 */
public int getIdUtilizador() {
    return idUtilizador;
}

/**
 * Recebe o valor do iterador atualizado, para poder gerar novos lds, a partir do número
 * do último ld gerado.
 * @param novoValor referência do valor atualizado a ser usado pelo geradorId.
 */
public static void buscarValorAtualizadoParaAlterador(int novoValor){
    geradorId = novoValor;
}
}

```