

Relatório POO

11 de Junho de 2020

Universidade do Minho

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA

TRAZAQUI!

PROGRAMAÇÃO ORIENTADA A OBJECTOS

2º ANO, 2º SEMESTRE, 2019/2020



Grupo 40

FRANCISCO PEIXOTO A84668

RENATO GOMES A84696

INTRODUÇÃO

Este relatório contém a apresentação do projecto desenvolvido ao longo do semestre para a disciplina de Programação Orientada a Objectos. Serão expostas as soluções implementadas para a construção da plataforma TrazAqui!.

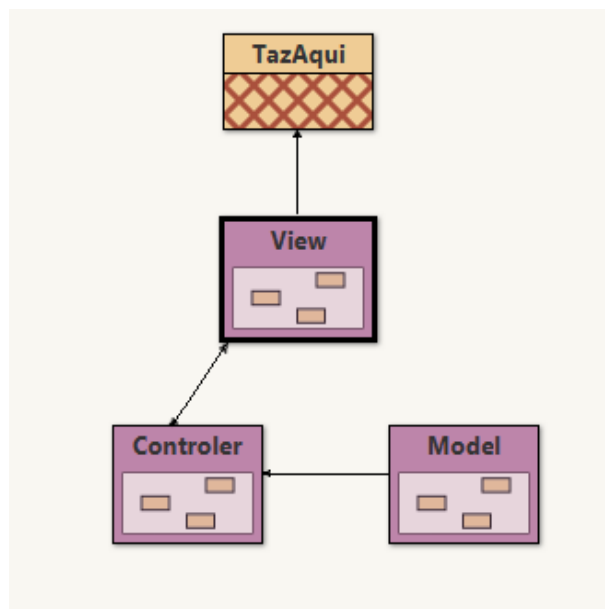
O trabalho prático tem como objectivo a criação de uma plataforma em Java que permite conjugar as actuais necessidades de entregar encomendas a pessoas que estão confinadas nas suas habitações, para tal desenvolvemos todas as funcionalidades, desde a criação de utilizadores, transportadoras, lojas e voluntários á simulação do processo de encomendar um produto até este ser entregue. Para cumprir as metas do projecto foram desenvolvidas as funcionalidades básicas da aplicação requeridas no enunciado que permitem ter voluntários que façam entregas de encomendas a um utilizador (que se encontra na sua habitação), ter empresas que façam entregas de encomendas, ter lojas que adiram a este serviço para poderem ser entregues as encomendas aos utilizadores.

Finalmente, a aplicação permite guardar registo de todas as operações efectuadas e possui mecanismos para aceder a essa informação.

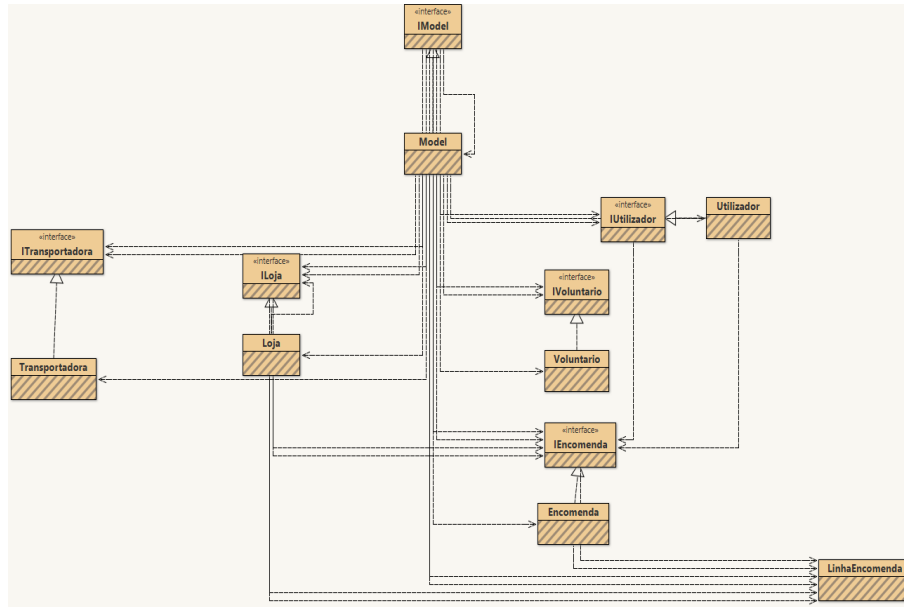
DESCRIÇÃO DO PROJECTO

No início do projeto criamos as classes iniciais como o Utilizador, Transportadora, Loja e Voluntário. Mais tarde fomos adicionando mais classes como as classes que representa a arquitetura de software MVC, dividido por packages Model, View e Controller.

Abaixo está o diagrama de classes do nosso projeto.



MODEL



1 Utilizador

A classe UTILIZADOR permite-nos criar a identificação de um utilizador. Aqui criamos todas as variáveis necessárias para que o cliente seja identificado. Para tal, temos nesta classe definidos o email do utilizador e a sua password, que são essenciais para que o utilizador possa fazer login na aplicação. Possui ainda o nome, a histórico de encomendas, ID, numero de encomendas realizadas e a localização do cliente.

```
1 public class Utilizador implements Serializable, IUtilizador
2 {
3     private String id;
4     private String nome;
5     private String email;
6     private String pwd;
7     private int acessos;
8     private double localizacaoX;
9     private double localizacaoY;
10    private int estado;
11    private Set<IEncomenda> historico;
```

2 Transportadora

A classe TRANSPORTADORA permite-nos a identificação de uma transportadora. Aqui criamos todas as variáveis necessárias para que a transportadora seja identificado. Para tal, temos nesta classe definidos todas as instâncias necessárias no registo da transportadora como o seu email, password, alcance, nome e preço por km . Possui ainda o um histórico de encomendas e a sua facturação.

3 Voluntario

A classe VOLUNTARIO permite-nos a identificação de um voluntario. Aqui, como na transportadora, criamos todas as variáveis necessárias para que o voluntario seja identificado. Para tal, temos nesta classe definidos todas as instâncias necessárias no registo do voluntario como o seu email, password, alcance e nome . Possui ainda um histórico de encomendas.

4 Loja

A classe LOJA é também á semelhança das classes acima apresentadas, uma classe que caracteriza uma loja. Como tal tem um nome, email e password necessários no registo e também um inventario de produtos disponíveis para o utilizador encomendar. Tem também uma subclasse FILA que guarda informações sobre as filas nessa loja, como numero máximo de pessoas, tempo médio de espera e o numero de pessoas á espera.

5 Encomenda

A classe ENCOMENDA é uma entidade criada quando é realizada uma encomenda por parte de utilizador numa loja, guardando assim tudo o que é necessário para que todas as entidades envolvidas na encomenda tenham as informações necessárias para poder realizar a compra e entrega da mesma. Para isso a classe possui os ID da loja onde foi comprada e do utilizador que a encomendou, bem como um Set de ID dos estafetas, sejam eles voluntários ou transportadora, que a aceitaram entregar e que será mais tarde apresentada ao utilizador para escolha final. Possui também uma lista de produtos criados pela classe LinhaEncomenda.

```
12 public class Encomenda implements Serializable, IEncomenda {
13     private String id;
14     private String loja;
15     private String userId;
16     private Set<String> estafeta;
17     private ArrayList<LinhaEncomenda> produtos;
18     private int tempo;
```

6 LinhaEncomenda

Esta classe é usada para criar produtos com todos os parâmetros necessários, como ID, descrição peso e preço. É usada quando uma encomenda é criada.

```
19 public class LinhaEncomenda implements Serializable {
20
21     private String codProd;
22     private String descricao;
23     private double preco;
24     private double peso;
25     private String tipo;
```

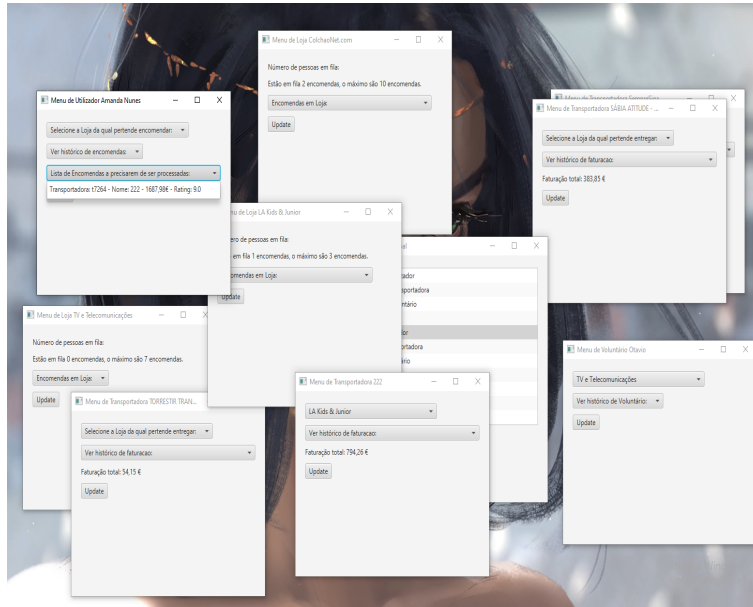
7 Model

Esta é a principal classe deste package, onde se cria e preenche os Maps com as diversas entidades. Estes são essencialmente a estrutura base desta aplicação. É aqui também onde os métodos para guardar e carregar o estado da aplicação estão, bem como os que fazem o carregamento inicial dos dados que o professor nos forneceu. É aqui também que estão os métodos para a ordenação das listas top 10 usando expressões lambda.

```
26 public class Model implements Serializable, IModel {
27
28     private HashMap<String,ITransportadora> transMap;
29     private HashMap<String,IVoluntario> volMap;
30     private TreeMap<String,IUtilizador> userMap;
31     private HashMap<String,ILoja> lojaMap;
32     private HashMap<String,IEncomenda> encMap;
33
34     private List<ITransportadora> get_all_trans(){
35         List<ITransportadora> transportadoras =
36             new ArrayList<>(transMap.values());
37         return transportadoras.stream()
38             .sorted(Comparator.comparing(ITransportadora::
39                 getDistancia)
40                 .reversed()).collect(Collectors.toList());
41     }
42     public List<String> top10Distancias() {
43         List<String> lista = new ArrayList<>();
44         List<ITransportadora> l = get_all_trans();
45         for(ITransportadora t:l){
46             lista.add("Transportadora: " + t.getNome()
47                 + " | Distancia percorrida: " + t.
48                 getDistancia());
49         }
50         return lista.stream().limit(10).collect(Collectors.
51             toList());
52     }
53 }
```

VIEW

Neste package que contem apenas uma classe de mesmo nome e a sua interface, é onde toda a interface de usuário é construído. Para tal usamos a Framework de Java, Javafx, que nos permite ter um GUI (Graphic User Interface) de varias janelas simultâneas, o que facilita bastante quer nos testes quer na navegação da mesma, ficando com um aspeto bem mais apelativo também.



CONTROLLER

Este package, semelhante ao View, também só contem uma classe de mesmo nome, Controller, e a sua interface. É aqui que estão implementados os métodos de acesso aos dados carregados e filtrados na classe Model. É aqui que se faz o registo e validação das entidades recebidas na View bem como a gestão dos pedidos de recolha e finalização das encomendas. No fim da classe Controller encontramos duas classes que entendem TimerTasks que nos permitem simular o tempo de entrega/filas em loja por parte das entidades voluntario e transportadora.

```

51 public void finalizar_encomenda(IUtilizador u, String
    estafeta, String value){
52     for(IEncomenda enc : model.getEncMap().values()){
53         (...)
54         if(value.startsWith("v")){
55             IVoluntario v = model.voluntario(value);
56             v.not_available();
57             long tempo = (long) (l.f_time() * 100 *
58                 distancia(u.getId(), l.getId(), v.getId()));
59             enc.setTempo((int) tempo );
60             timer.schedule(new Finalizar_rv(v, enc, u, l
                ), tempo);
61         }
62         (...)

```

No exemplo podemos ver como `timer.schedule` implementa um tempo de espera baseado no tempo médio em fila da loja e na distancia entre a loja e o utilizador.

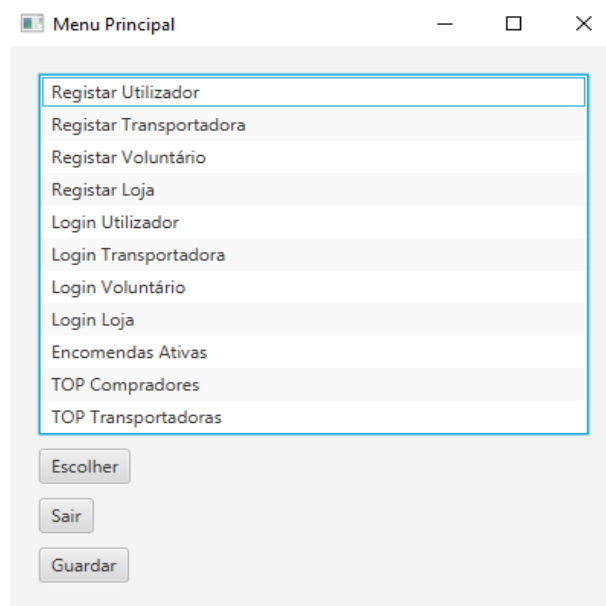
FUNCIONAMENTO

Para este projetos decidimos usar Javafx uma Framework de Java, que nos permite construir janelas, para que assim a interface de usuário se torne mais simples e convidativa.

Como tal achamos que deveríamos introduzir uma pequena guia para utilizar a aplicação.

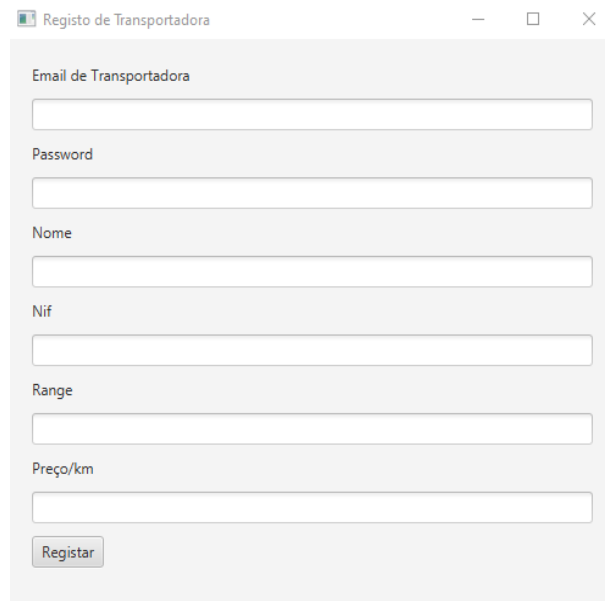
Menu Principal

No menu principal podemos optar se queremos fazer LogIn ou Registrar com qualquer uma das entidades, bem como ter acesso aos top 10 utilizadores que mais compraram e transportadoras que mais distancia percorreram, e ainda verificar que encomendas estão ativas, ou seja foram encomendadas mas ainda não entregues ou seleccionadas para serem entregues por um voluntario ou transportadora. É também possível guardar o estado da aplicação, algo que se deve fazer para guarda as alterações feitas na aplicação, e sair da aplicação neste menu.



Menu Registo

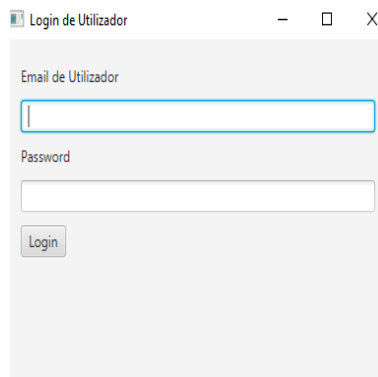
Este menu, é como o nome indica, o menu onde se registam cada uma das entidades, havendo um menu de registo específico para cada uma delas, com as informações necessárias ao seu registo.



The screenshot shows a web application window titled "Registo de Transportadora". It contains a registration form with the following fields: "Email de Transportadora", "Password", "Nome", "Nif", "Range", and "Preço/km". Each field is represented by a text input box. At the bottom of the form is a button labeled "Registar".

Menu LogIn

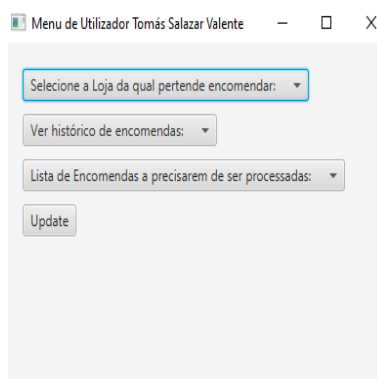
Este menu, serve para efectuar LogIn das entidades, havendo um para cada entidade.



The screenshot shows a web application window titled "Login de Utilizador". It contains a login form with two fields: "Email de Utilizador" and "Password". Each field is represented by a text input box. At the bottom of the form is a button labeled "Login".

Menu Utilizador

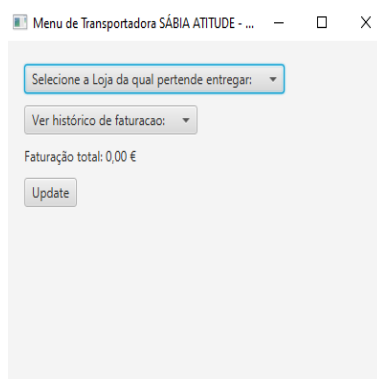
Este menu, serve para que o utilizador possa encomendar um produto de uma determinada loja, e mais tarde quando disponível uma opção de escolha de entidade que realiza entrega, é aqui que pode escolher um dessas entidades para realizar a entrega da sua encomenda. Pode também aceder ao histórico de encomendas realizadas. Ainda possui um botão Update que serve para atualizar os dados.



The screenshot shows a window titled "Menu de Utilizador Tomás Salazar Valente". It contains four dropdown menus and one button. The first dropdown menu is labeled "Selecione a Loja da qual pretende encomendar:". The second dropdown menu is labeled "Ver histórico de encomendas:". The third dropdown menu is labeled "Lista de Encomendas a precisarem de ser processadas:". The fourth dropdown menu is labeled "Update".

Menu Transportadora

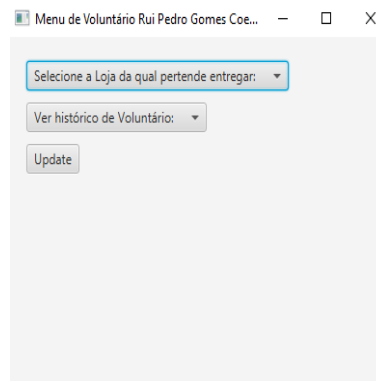
Este menu, serve para que a transportadora possa sinalizar a sua intenção de entrega de uma encomenda de uma determinada loja, desde que esta se encontre dentro do seu alcance. Pode também aceder ao histórico de faturação de encomendas, uma a uma, bem como a faturação total. Ainda possui um botão Update que serve para atualizar os dados.



The screenshot shows a window titled "Menu de Transportadora SÁBIA ATITUDE - ...". It contains two dropdown menus, a text label, and one button. The first dropdown menu is labeled "Selecione a Loja da qual pretende entregar:". The second dropdown menu is labeled "Ver histórico de faturacao:". Below the second dropdown menu, there is a text label "Faturação total: 0,00 €". The third dropdown menu is labeled "Update".

Menu Voluntario

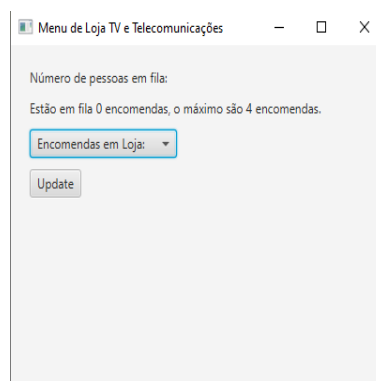
Este menu, muito semelhante ao menu da transportadora, serve para que o voluntario possa sinalizar a sua intenção de entrega de uma encomenda de uma determinada loja, desde que esta se encontre dentro do seu alcance. Pode também aceder ao histórico de encomendas entregues. Ainda possui um botão Update que serve para atualizar os dados.



The screenshot shows a window titled "Menu de Voluntário Rui Pedro Gomes Coe...". Inside the window, there is a dropdown menu labeled "Selecione a Loja da qual pretende entregar:". Below this, there is another dropdown menu labeled "Ver histórico de Voluntário:". At the bottom of the menu area, there is a button labeled "Update".

Menu Loja

Este menu mostra o numero de pessoas em fila de uma loja bem como a sua ocupação máxima, e as encomendas ativas efetuadas nessa loja. Ainda possui um botão Update que serve para atualizar os dados.



The screenshot shows a window titled "Menu de Loja TV e Telecomunicações". Inside the window, there is a section titled "Número de pessoas em fila:" followed by the text "Estão em fila 0 encomendas, o máximo são 4 encomendas." Below this, there is a dropdown menu labeled "Encomendas em Loja:". At the bottom of the menu area, there is a button labeled "Update".

TESTE

Juntamente com o projeto e este relatório vai também um ficheiro texto, com as credenciais de algumas entidades, para que seja possível testar a aplicação.

CONCLUSÃO

Em suma, o grupo considera que o projeto foi concluído com sucesso. Permitiu-nos compreender como a optimização e o processamento de grandes volumes de dados não é igual em todos os paradigmas da programação, tal como os seus cuidados, tivemos a possibilidade de por em prática os conhecimentos adquiridos na unidade curricular. Tivemos de ter noções sobre classes, tendo sempre cuidado com o encapsulamento dos dados, e respeitando a arquitetura MVC.

O processo de desenvolvimento da nossa aplicação realizou-se de forma progressiva. Começamos por definir as classes chave, de seguida implementamos os requisitos básicos.

Foi necessário decidir o tipo de dados a utilizar ao definir conjuntos, tendo sido implementados sempre as estruturas que achamos que mais se adequavam, por exemplo usamos TreeMaps para armazenar os utilizadores e as transportadoras para que se tornasse mais fácil a execução dos top 10 utilizadores e transportadoras, para que pudéssemos usar um Comparable, usando um HashMap nas restantes entidades.

Para além disso, tivemos de ter noções sobre iteradores externos e internos. A escolha destes iteradores teve sempre em atenção a situação onde estavam a ser aplicados e a sua eficiência. A título de exemplo temos a utilização da Stream para ordenar os top 10 referidos acima.

Foi utilizado também o Abstract, as interfaces, por exemplo Serializable, e Comparable, tornando assim o código modular, e a captura de erros com Exceptions.

Além de mais, tivemos também, por opção própria, que aprender a usar a framework Javafx, que foi desafiante e arriscado, mas no fim recompensador pela forma como acabou por ser realizado.