Universidade do Minho
Licenciatura em Ciências da Computação
2º Ano

Programação Orientada a Objetos

2º Semestre - 2019/2020

Trabalho Prático - JAVA – Traz Aqui!

Realizado por:

Luís Francisco Mendes Lopes - A85367

David João Oliveira Gonçalves – A81820

de 2020

Índice

1. **Introdução.....3**
2. **Arquitectura da Aplicação.....4**
    - **2.1. Visão Geral.....4**
    - **2.2. Classes.....4**
        - **I. Menu (View).....4**
        - **II. Controller.....5**
        - **III. trazAqui (Model).....6**
        - **IV. Métodos Extra.....7**
    - **2.3. Diagrama de Classes.....7**
3. **Conclusão.....8**

1. Introdução

No âmbito da UC de POO, foi-nos proposto pela equipa docente a realização de um projecto em JAVA que envolve-se a criação de uma plataforma semelhente ao projecto www.posso-ir.com.

Isto é, uma plataforma onde que permita satisfazer as actuais necessidades de entregar encomendas a pessoas/clientes que estão confinadas nas sua habitações e necessitam de um serviço que permita:

- ter voluntários/estafetas que façam entregas de encomendas a um utilizador/cliente;
- ter empresas que realizam serviços de entregas;
- ter lojas que adiram a este serviço para poderem ser entregues as encomendas aos utilizadores.

Para a sua realização, esta plataforma necessitará de:

- abrangir mecanismos de criação de utilizadores, voluntários, empresas transportadoras e lojas;
- guardar o registo de todas as operações efectuadas e as informação relevantes aos diversos perfis, e que o sistema possuía de mecanismos para as disponibilizar;
- e finalmente processos para que cada perfil apenas consiga aceder às informações e funcionalidades respectivas a si.

Com este relatório iremos também expor a estrutra, código, funcionamento, como realizamos e algumas dificuldades deste trabalho.

Esperamos que o relatório esteje do seu agrado caro leitor.

2. Arquitetura da Aplicação

2.1 Visão Geral

Para a proposta de resolução do trabalho implementamos uma estrutura do tipo **MVC**. Desta forma dividimos o trabalho em 3 componentes distintas: **Model,View e Controller.**

O Model consiste na classe **"trazAqui"** e toda a restante estrutura de dados necessárias, clientes, lojas e etc.

A view consiste numa classe, **"Menu"**, que vai guardar todos os métodos necessários para apresentar e recolher informação.

O controller (classe associada tem o mesmo nome) acaba por ser o cérebro por tras do programa. Tem no mínimo duas variaveis de instancia: Model e View.

2.2 Classes

I. Menu (View)

Efetivamente onde a Interface da aplicação é produzida. Estabelece ligação com o **Controller** quando o utilizador introduz dados (Input).

A classe começa com a definição de Variáveis Globais que estarão presentes nos metódos a seguir descritos:

- `public int InicioApp()` - é aqui onde o utilizador pode fazer Login, Registar uma conta e terminatar a aplicação ;
- `public int menuInicial()` - onde o utilizador entra nos vários serviços (Loja, Transportadoras, Voluntariado e área para Clientes) ;
- `public String Login[L/T/V/C]()` - Ecrãs de Login nas contas dos respectivos serviços ;
- `public int menuRegistoGeral()` - Registo de contas, o utilizador escolhe qual dos serviços quer registar uma conta e aí é "enviado" para uma dos seguintes métodos `public String RegL/RegT/RegV/RegC()` que realmente irá lidar com o resgisto do respetivo serviço;

- Menus do Utilizador - `public int menuGeralUser(String nome) / public String showStores(List<Store> lojas) / public ArrayList<String> showProducts(Set<Product> produtos) / public void showPastOrders(Set<Order> orders, Map<String,Store> stores) / public void showTop10Transporters(Set<Transporter> transporterSet) / public void showTop10Users(Set<User> userSet);`
- Menus da Loja - `public int menuGeralLoja(String nome) / public List<String> showOrders(Set<Order> ordersToSetReady,String title) / public String showOrdersString(Set<Order> ordersToSetReady,String title);`
- Menus dos Voluntários - `public int menuGeralVoluntario(String nome) / public int menuGeralEmpresa(String nome)`

II. Controller

É responsável pela interação com o utilizador da aplicação, é por isso a classe que lida com o Input do utilizador enviado para esta classe por via da View ("Menu")

Estabelece ligação com o **trazAqui** para este guardar os dados inseridos pelo utilizdor. É também a classe que "diz" ao **Menu** que Output imprimir.

O controller é constituído por 4 variáveis de instância:

- **LoadState**
- **SaveState**
- **trazAqui » Model**
- **Menu » View**

A loadstate acaba por ser uma classe que vai ler do ficheiro **.obj,** se este existir ou não estiver corrompido. Caso isto aconteça opta por fazer Parse dos logs fornecidos. É desta forma, então, que carregamos o estado da aplicação.

A savestate, como indica o nome será a classe usada para guardar o estado da aplicação. Tem um int como variável de instancia que diz se o programa esta em modo de teste, se for o caso, além de gravar em .obj, guarda também em JSON.

A trazAqui como já dito várias vezes é o modelo do programa, e o Menu a view.

O Controller, como Menu, começa por definir algumas variáveis globais seguidas de vários métodos de instancia que permitem deste modo o pleno funcionamento da mesma:

- `public Controller(int tesmodeSelected)` - Método construtor;
- `public static void main(String args[])` – Método MAIN;
- `public void setInicioApp ()` - Este método é o primeiro a ser executado após a criação do Controller e carregamento de dados, chama os metodos da view que imprimem o menu principal, que permite escolher autenticar ou registar;
- `public void setMenuLogin()` - Se o utilizador decidir autenticar, será chamado este método que imprime, através a view, as opções de contas ;
- `public void Login[Loja/Transportadora/Voluntario/Cliente]()` - Os 4 métodos que permitem o login nas contas, se errarem alguma credencial dará erro de conta nao existente;
- `public void setMenuRegisto()` + `public void Registo[Loja/Transportadora/Voluntario/Cliente]()` - Métodos para registo.

Os métodos a seguir são para cada tipo de utilizador:

- Loja - `public void MenuLoja(String code) / public void processOrders(String code);`
- Voluntário - `public void MenuVoluntario(String code);`

- Transportadora - `public void MenuEmpresa(String code) / public void deliveryDone(String code,int type) / public void totalFaturado(String code);`
- Cliente - `public void MenuCliente(String code) / public void classificarEntrega(String codeUser) / public void showPastOrders(String code,int type) / public void criarEncomenda(String codeUser) / private String selectTransporter(String codeStore,ArrayList<String> products).`

III. trazAqui (Model)

Esta classe é o modelo do programa.

Envolve toda a estruturação e estado do programa. Composta por 7 maps e uma list de strings, esta classe armazena transportadores, voluntários, lojas e clientes. A restantes variaveis de instancia armazenam desde encomendas aceites a encomendas entregues.

Métodos incluíem:

- os vários `public trazAqui()` - métodos de Construção da Classe por omissão, passagem de argumentos e passagem de Objeto.
- métodos para o registo de Utilizadore do programa (sendo eles Loja, Voluntário, Transportadoras e Clientes), cada um recebe como argumento um array de Strings que contem a informação necessária - `public String createStore(String[] input) / public String createTransporter(String[] input) / public String createVolunteer(String[] input) / public String createUser(String[] input) / etc...`
- `public String validaLogin(int type,String nome_code)` – tal como o nome indica verifica as credênciais do Login. Faz isto recebendo o tipo de conta, bem como uma string no formato Nome#Codigo, que posteriormente é divida e passada por argumento na chamada à validateLogin existente em cada classe Store,User,Volunteer.
- `public String randomCodeGenerator(int type)` - Método para gerar um código. Maximo é 3000, valor arbitrario. Limita o numero de contas de cada tipo a 3000 bem como encomendas.
- `public boolean codeInUse(int type, String code)` - recebe o tipo de codigo e o novo codigo, verificando se o mesmo ja esta em uso.
- `public String getCloserTransporterToStore` e `getCloserVolunteerToStore` servem o mesmo propósito, encontram a Transportadora e Voluntário mais próximos de uma loja.
- `public Set<Order> getPastOrders(String code, int type)` – encontra entregas anteriores.
- `public double calculateDistance(Coordinates trans,Coordinates store)` - determina a distancia entre uma transportadora/Voluntario e loja desejada.
- `public double getPriceT(String trans,String store, ArrayList<String> product)` - calcula o preço de entrega de uma encomenda consoante o preço e a distancia.
- `public Set<Transporter>/<User> listaEmpresasMaisUsam()/listaUtilizadoresMaisUsam()` – determina a listangens das 10 empresas/clientes que mais utilizam o sistema.
- Os restantes métodos são praticamente funcionalidades da aplicação para vários serviços.

IV. Métodos Extra

Existem obviamente vários outros métodos, mas a sua maioria tem um funcionamente e propósito bastante simples e de pouca importância em comparação com as classes MVC. Como tal não serão elaborados. No entanto aqui estão algumas da qual a sua operação não é tão aparente:

randomOrderTime

Esta classe da extend a Random recebe a queue eo avgTime de serviço duma loja, com estes dados decide aleatoriamente o tempo de processamento da entrega.

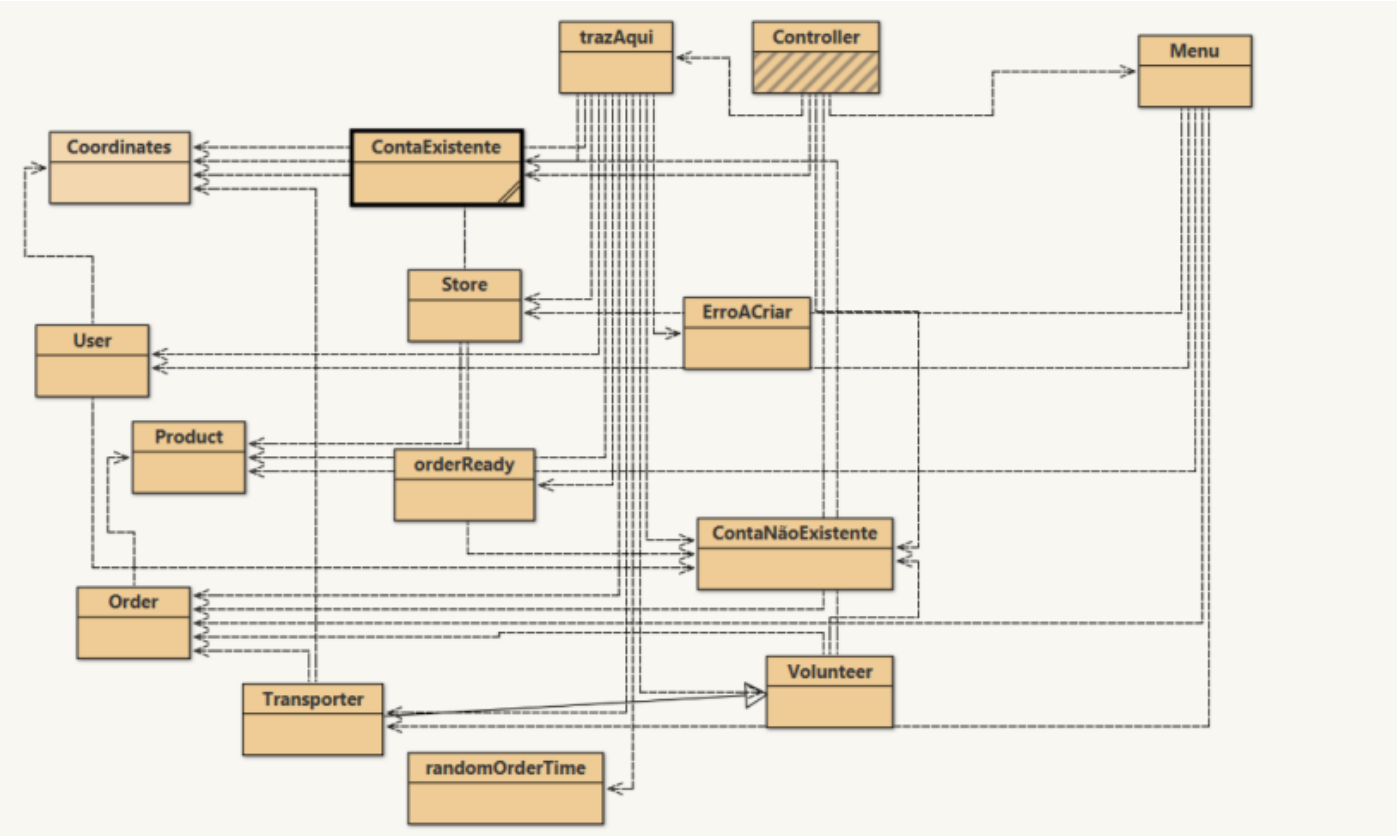
saveState

A classe saveState extends thread.

Utiliza reentrantLock para que não seja possível escrever ou ler ao mesmo tempo de um ficheiro por outras threads, desta forma não ocorre nenhum erro. Visto que pode haver uma falha de energia ou um súbito fecho da aplicação, sempre que existe alguma alteração de dados guardamos o estado.

Com a implementação em threads faz com que não ocorra qualquer degradação na performance de todas as outras funcionalidades da aplicação.

2.3 Diagrama de Classes



3. Conclusão

Fazendo uma avaliação geral da nossa jornada com este projeto, o grupo concluiu que a experiência foi deveras aliciante e que pôs em prática todos os nossos conhecimentos da linguagem JAVA.

Produzimos na nossa capacidade, e opinião, um programa acessível e adequadamente construído. Embora encontra-mos dificuldade com a organização do Model (classe Menu), no nosso ponto de vista, consideramos ter cumprido com sucesso os objetivos deste trabalho prático. Aliás não diríamos ter encontrado contra-tempos significativos visto que o tempo disponível para a resolução de tanto os requisitos básicos e avançados tenha sido suficiente.