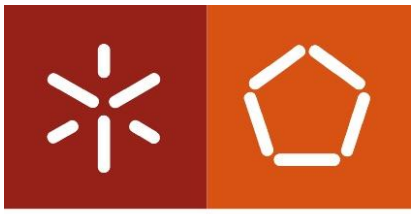

Universidade do Minho

Departamento de Informática

Traz Aqui

Grupo Nº 42

Filipe Felício (A85983)

Henrique Ribeiro (A89582)

Luis Araújo (A86772)

June 8,2020

Conteúdo

Capítulo 1

Introdução

As seguintes páginas servem o propósito de rever em profundidade a sofisticação da nossa implementação de software para um sistema de Gestão de Encomendas, com tipos diferentes utilizadores, capaz de suportar todas as funcionalidades atribuidas a cada e responder aos desafios de comunicação de informação entre eles com soluções e arquitetura ótimas.

Sendo a palavra chave do último paragrafo "sofisticação", no âmbito do Paradigma Orientado aos Objetos e da linguagem de programação JAVA, fazendo recurso à abstração e a hierarquia de Classes procuramos atingir um código modular, reutilizável cuja a implementação de novas "features", sejam elas qual forem, sejam o mais fácil possível desde que respeitem o seu correto lugar dentro da hierarquia imposta, algo em que o projeto assentou desde a genese e que tornou o seu desenvolvimento extremamente agradavel.

De realçar que graças ao uso de classes mais abstratas como a Class Object, a Class Class, a Class Method, a Class Field e a Class Constructor, entre outras, foi nos permitido construir uma aplicação no terminal que, enquadrada numa arquitetura MVC, consegue adptar se instantaneamente a alterações no Model sem necessitar de incluir diretamente as novas funcionalidades no Controller e na View. Isto permitiu um desenvolvimento rápido, escalavel e permitiu corrigir erros de intrepertação do Enunciado ou adicionar novas ideias,como catalogo nas lojas por exemplo, quando extrema naturalidade demonstrando, a nosso ver, boa percepçao do Paradigma e intimidade com a API da linguagem.

Nas próximas secções deste relatório iremos explicar as estruturas de dados escolhidas para armazenar as diferentes informações, as razões que nos levaram a escolhe-las, a estrategia escolhida para elaborar esta aplicação e alguns comentarios conclusivos à cerca deste projeto.

Capítulo 2

Arquitetura

A aplicação foi desvolvida tendo sempre em mente o modelo de desenvolvimento MVC(Model - View - Controller). Desta forma, consiguimos destinguir três grandes bases neste trabalho: o *Model* que agrega todas as classes que sao utilizadas no armazenamento das estruturas de dados, e a parte algorítima do programa; a *View* que efetua todas as operações correspondente à comunicação do utilizador; o *Controller* serve como um meio de comunicação entre o *Model* e a *View*, fazendo com que este fluxo tenha sempre de passar primeiro pelo *Controller*.
Com base na hierarquia de classes, decidimos produzir uma arquitetura que nos permitisse nao só a reutilização código, como támbem uma arquitetura que nos permitisse uma gestão bastante prática do mesmo.
Começamos por definir classes abstratas que nos ajudariam a definir esta hierarquia, uma vez que as diferentes classes partilhavam algumas variáveis de instância. Ficou-nos claro então que esta estrutura deveria tendo uma classe (geral) abstrata *Entitie* que seria estendida pelas diferentes classes de utilizadores do sistema (*User*,*Carrier*,*Transporter*,*Volunteer*,*Store*).

Capítulo 3

Classes

3.1 Model

3.1.1 Entitie

private String ID;
private String name;
private String email;
private String password;
private Double x;
private Double y;

Esta classe possui todas as variáveis de instâncias que serão parte de todas as outras entidades do sistemas. Além disso, apontamos esta classe como abstrata visto que nao será necessário, nem permitido o seu instânciamento. Vamos agora examinar cada uma das variáveis e ver como serão necessárias em todas entidades:

- **ID** Identificador de cada entidade
- **name** Nome de cada entidade
- **email** Email de cada entidade
- **password** Password de cada entidade
- **x** Posição x do GPS
- **y** Posicao y do GPS

3.1.2 User

private String storeToBuyFromID;
private Map<Double,Order>_transportOffers;
private Map<String,Integer>_cart;

Esta classe estende a classe *Entitie* contendo todas suas variáveis de instância.
Além disso, são acrescentados mais algumas variáveis ao *User*:

- **storeToBuyFromID** Loja a qual o *User* irá fazer uma encomenda
- **_transportOffers** Lista de ofertas das transportadoras perante uma encomenda do *User*
- **_cart** Carrinho de produtos do *User*

3.1.3 Carrier

private Double radius;

À semelhança da classe anterior, esta estende a classe *Entitie* pelo que herda as variáveis de instância da mesma. Porém, é-lhe adicionada o raio de operação de um *Carrier*. Optamos por deixar esta classe como abstrata, visto que esta será estendida por todas as classes que envolvem transporte de encomendas: Além das já existentes (*Transporter*, *Volunteer*), também as futuras classes de transporte de encomendas.

- **radius** Raio de operação de um *Carrier*

3.1.4 Transporter

private String TIN;
private Double fee;
private Integer maxOrders;

Esta classe estende a classe anterior (*Carrier*). Além de obter todas as suas variáveis de instância, foram-lhe acrescentadas mais algumas variáveis necessárias, uma vez que uma transportadora necessita de um NIF, de uma taxa cobrada por KM, e um numero máximo de encomendas que pode entregar.

- **TIN** Número de Identificação Fiscal
- **fee** Taxa cobrada por KM
- **maxOrders** Número máximo de encomendas de um *Transporter*

3.1.5 Volunteer

Tal como a classe anterior, esta estende a classe *Carrier* e recebe todas as suas variáveis de instância.

3.1.6 Store

private Map<String,Map<String,Integer>>_userRequests;

Esta classe estende a classe *Entitie* recebendo todas as suas variáveis de instância.
Foi-lhe adicionada uma nova variável de instância que lhe permite ter uma lista de solicidações de encomendas por um *User*.

- **_userRequests** Encomendas pendentes a aceitação da *Store*

3.1.7 Model

private Map<String,Entitie>_entities;
private Map<String,List<Record>>_records;
private Map<String,List<Order>>_orders;

Nesta classe agrupamos e armazenamos todos os itens atrás referidos e que são necessários para que o sistema proposto pelos docentes funcione.

3.2 Controller

Como já foi referido anteriormente, esta classe é responsável pelo controle da interação entre o *Utilizador* e o *Model*.

3.3 View

Classe responsável pela interação com o *Utilizador*.

Capítulo 4

Manual de Utilização

Ao ser inicializada a aplicação é apresentado o menu principal,que permite ao utilizador executar algumas funcionalidades do programa. Optamos por escolher uma navegação baseada nas linhas de comando do terminal (prompt), pois este encaixa-se bem com a nossa aplicação.

Figura 4.1: Menu Principal.

Uma vez que as entidades presentes nos logs não possuiam nem email nem password, e querendo deixar o login como uma possibilidade global, decidimos que para fazer o login o utilizador necessitasse de introduzir o seu ID ou seu Email. Caso o utilizador esteja registado no sistema e não tenha password, o mesmo pode correr o comando "Login [ID do Utilizador]"e o programa irá ser redirecionado para um menu que permiterá o registo destes dados. Apresentamos aqui também as opções de criar uma nova conta, carregar um novo ficheiro de dados, gravar o estado da aplicação, carregar o ficheiro "logs"dos professores e por fim as queries requisitadas pelos docentes.

Capítulo 5

Conclusão

Neste projeto foram correspondidos os requisitos necessários para a resolução, visto que, por sua vez, foi implementada respeitando os conceitos de modularidade, encapsulamento de dados, reutilização de código e o modelo MVC.
A modularidade desta aplicação possibilita-nos a introdução de novas entidades e funcionalidades no sistema com relativa facilidade.
A partir dos clones presentes em todos os métodos de comunicação entre o *Controller* e o *Model* controlamos o encapsulamento de dados do sistema.
Com a hierarquia presente no *Model* consiguimos permitir a reutilização de código.
O modelo Module - View - Controller permite-nos um fluxo organizado entre o *Utilizador* e a *Base de Dados*.
Por fim, avaliamos este projeto como um sucesso perante o que foi sugerido pelos docentes desta unidade curricular.

Capítulo 6

Diagrams

6.1 Packages Diagram

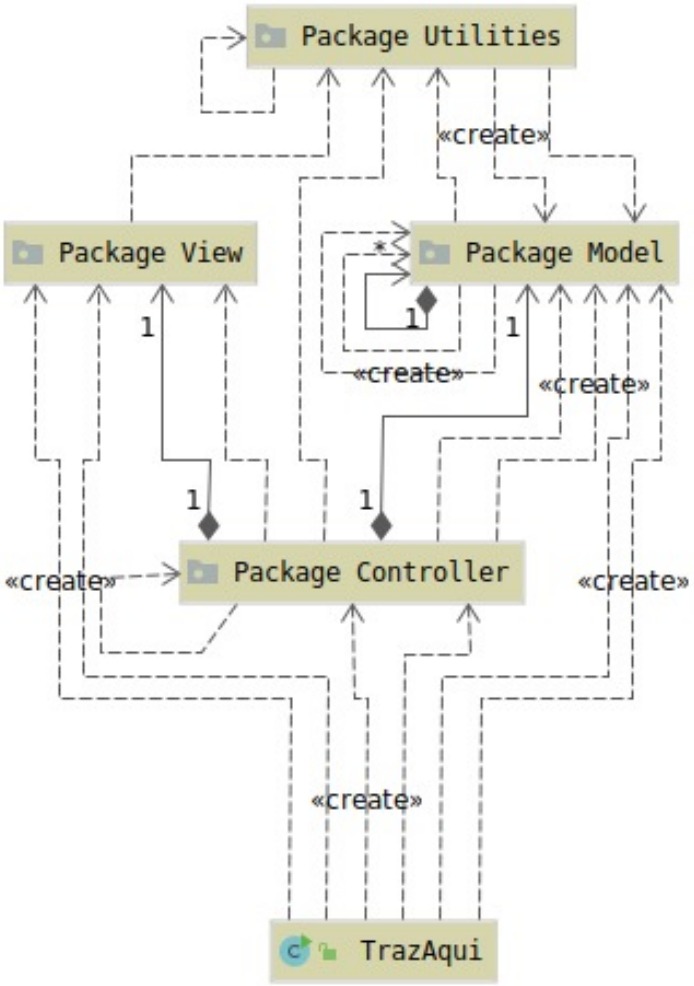

Figura 6.1: Diagram of the Model module.

6.2 Model Diagram

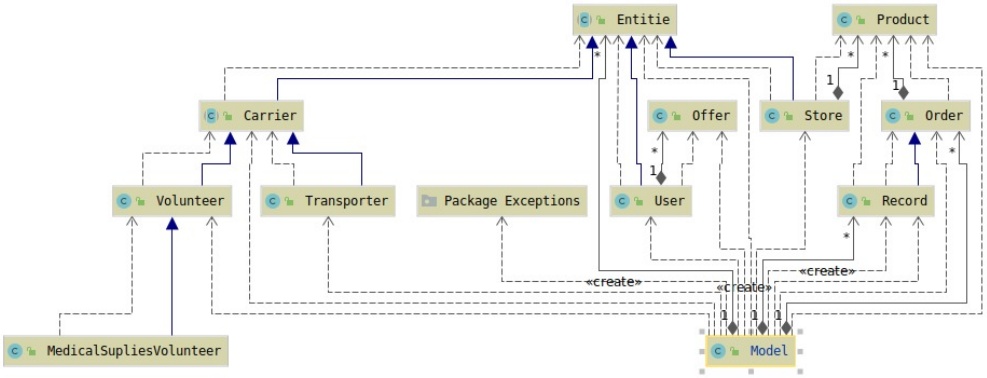

Figura 6.2: Diagram of the Model module.