

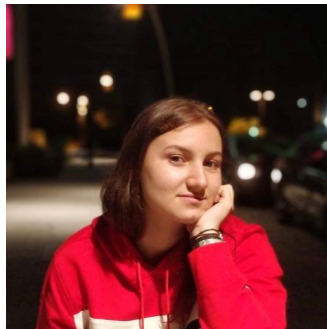
UNIVERSIDADE DO MINHO

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA

POO-Sistema de Gestão de Encomendas
Grupo 33

Sofia Santos (A89615) Ana Filipa Pereira (A89589)
Carolina Santejo (A89500)

Ano Letivo 2019/2020



Conteúdo

1	Introdução e principais desafios	4
2	Classes	5
2.1	Main	5
2.2	TrazAqui	5
2.3	Estado	5
2.4	Controller	6
2.5	ControllerUtilizador	6
2.6	ControllerVoluntario	6
2.7	ControllerLoja	7
2.8	ControllerTransportadora	7
2.9	Conta	7
2.10	Contas	8
2.11	Encomenda	8
2.12	LinhaEncomenda	8
2.13	Utilizador	9
2.14	Voluntario	9
2.15	Loja	9
2.16	Transportadora	10
2.17	Menu	10
2.18	Interface Randoms	10
2.19	Interface TranspVolunt	10
2.20	TipoConta	11

2.21 Point	11
3 Estrutura do projeto	12
4 Conclusão	13
A Diagrama de Classes	14

Capítulo 1

Introdução e principais desafios

Este projeto consistiu no desenvolvimento de um Sistema de Gestão de Encomendas na linguagem de programação Java, de forma a pôr em prática os conhecimentos adquiridos ao longo do semestre.

Consideramos que o maior desafio tenha sido gerir várias encomendas, de diferentes naturezas, e várias entidades em simultâneo, tendo em conta fatores aleatórios, tais como atrasos nas filas de espera das lojas, ou atrasos nos deslocamentos das entidades Voluntários e Transportadoras.

Capítulo 2

Classes

2.1 Main

Classe responsável por arrancar com o programa. Para isto o método *main* chama o método *main* da classe *controler*;

2.2 TrazAqui

```
private Conta contaLoggedIn;  
private Estado estado;
```

TrazAqui é uma das várias classes controladoras do sistema; A classe TrazAqui é a classe que associa a conta com que se fez login a todo o estado atual do programa. Decidimos que a relação entre esta classe e Estado seria de agregação, pois ao partilhar o apontador da conta, caso esta seja alterada no Estado, o atributo *contaLoggedIn* será automaticamente alterado.

2.3 Estado

```
private Contas utilizadores;  
private Contas voluntarios;  
private Contas transportadoras;  
private Contas lojas;  
private Encomendas encomendas;  
private Set<AbstractMap.SimpleEntry<String, String>> pedidosTransporte; // ->
```

Estado é uma das várias classes controladoras do sistema. A classe Estado é a classe responsável por gerir toda a informação do sistema. Como podemos ver, esta possui a informação de todas as contas registadas no sistema e todas as encomendas registadas na aplicação. O atributo *pedidosDeTransporte* consiste num Set em que cada elemento é um "par" constituído pelo código da Transportadora e pelo código da Encomenda que aquela deseja transportar. Este pedido é retirado do Set se o pedido for aprovado ou outra entidade tenha ficado responsável pelo transporte da encomenda. A única relação de agregação entre classes em todo o programa é entre o Estado e

a TrazAqui(falaremos disso novamente).

2.4 Controller

Classe, que juntamente com outras, gere o fluxo da aplicação. Esta é a classe responsável por gerir o menu inicial, disponível para todas as entidades, além de interagir com outras classes controladoras que gerem os menus de cada entidade em particular (*login*); O método run de cada controller das entidades recebe um objecto de TrazAqui, de forma a que tenham acesso à informação que necessitam que se encontra, por exemplo, no estado.

2.5 ControllerUtilizador

Um das várias classes controladoras do sistema. Esta em particular é a que gere o menu do utilizador;

Neste menu temos as seguintes opções:

1-Efetuar uma compra: Esta será adicionada à fila de espera da Loja.

2-Solicitar Encomenda: Só após seleccionar esta opção e escolher a encomenda que quer solicitar, é que essa encomenda estará disponível para ser escolhida pelas entidades de transporte.

3-Verificar ofertas da Transportadora: Caso alguma transportadora tenha desejado transportar uma das encomendas do utilizador, este terá acesso ao código da transportadora, ao tempo que (em teoria) demorará o transporte e ao custo (que poderá variar dependendo da rota do transportador). Com isto o utilizador pode aceitar ou recusar o pedido.

4-Historico: Devolve o histórico das encomendas transportadas entre duas datas.

5-Classificar Entidade: Permite que o utilizador classifique uma entidade, mas com a condição de que esta tenha que já transportado para o dito utilizador.

6-Numero de encomendas Transportadas: Devolve o número de encomendas transportadas ao utilizador.

2.6 ControllerVoluntario

Um das várias classes controladoras do sistema. Esta em particular é a que gere o menu do voluntário;

Neste menu temos as seguintes opções:

1-Selecionar encomenda para transportar: Caso o voluntário esteja disponível (ou seja, não tenha mais nenhuma encomenda na sua posse), aparecerão todas as encomendas solicitadas e dentro dos seus limites de escolha (ex. raio). Este pode aceitar ou não.

2-Fazer Entrega: O voluntário incrementa o número de encomendas transportadas do utilizador, é mostrado o tempo que efetivamente demorou, contando com atrasos e filas de espera.

3-Verificar classificacao: Devolve a média de todas as classificações atribuídas.

4-Historico: Devolve o histórico das encomendas transportadas entre duas datas.

2.7 ControllerLoja

Um das várias classes controladoras do sistema. Esta em particular é a que gere o menu da Loja;

Neste menu temos as seguintes opções:

1-Ver lista de encomenda: Mostra todas as encomendas na fila de espera. *1-Despachar Encomenda* : Encomenda deixa de estar na fila de espera e passa para a lista de encomendas despachadas;

2-Quanto falta até encomenda estar disponível:Caso a encomenda se encontre na loja, devolve o tempo teórico que demorará até ficar despachada(relembrando que poderá haver atrasos ou a encomenda se despachar mais rápido);

2.8 ControllerTransportadora

Um das várias classes controladoras do sistema. Esta em particular é a que gere o menu da Transportadora;

Neste menu temos as seguintes opções:

1-Preço de transporte de uma encomenda: Caso a transportadora possa transportar a encomenda, é devolvido o preço do transporte, tendo em conta a distância e o peso da mesma.

2-Fazer Entrega:A transportadora define uma rota e vai distribuindo as encomendas guardando as distâncias parciais percorridas.(Ela primeiro vai a todas as lojas e depois a todos os utilizadores.)

3-Total faturado num período:Dado que as encomendas guardam quem as transportou, a distância para a ir buscar e entregar e a data, foi apenas necessário buscar as encomendas que satisfazem os requisitos e somar o custo de cada uma.

4-Selecionar encomenda para transportar: Caso a transportadora esteja disponível(ou seja,o número de encomendas é inferior ao máximo suportado), aparecerão todas as encomendas solicitadas e dentro dos seus limites de escolha(ex. raio). Este pode aceitar ou não.Caso aceite tem de aguardar aprovação do utilizador.

5-Verificar classificacao: Devolve a média de todas as classificações atribuídas.

6-Historico: Devolve o histórico das encomendas transportadas entre duas datas.

2.9 Conta

```
private String codigo;  
private String nome;  
private Point2D gps;  
private String email;
```

```
private String password;
```

Esta classe faz parte do Modelo da aplicação. Conta é superclasse de todas as outras classes que representam as entidades do sistema. Os seus atributos são aqueles comuns a essas mesmas entidades.

2.10 Contas

```
private Map<String, Conta> mapContas;
```

Esta classe faz parte do Modelo da aplicação. A classe Contas é responsável por armazenar todas as contas do Sistema. Para tal escolhemos uma estrutura Map, na qual cada conta fica associada ao seu código;

2.11 Encomenda

```
private String codEnc;  
private String codUtil;  
private String codLoja;  
private double peso;  
private Map<String, LinhaEncomenda> produtos;  
private LocalDateTime data;  
private boolean foiSolicitada;  
private boolean foiEntregue;  
private String quemTransportou;  
private boolean encMedica;  
private double distPercorrida;  
};
```

A classe Encomenda possui todos os atributos mínimos exigidos pelos docentes, no entanto tomámos a liberdade de adicionar 4 novos atributos:

LocalDateTime data - data do Transporte da encomenda

boolean foiSolicitada - Indica se a encomenda foi solicitada pelo utilizador

boolean foiEntregue - Indica se a encomenda chegou a ser entregue

String quemTransportou - código de quem a transportou;

boolean encMedica - Indica se a encomenda é médica ou não

distPercorrida - distância percorrida pela entidade de transporte para entregar essa encomenda;

2.12 LinhaEncomenda

```
private String codProduto;  
private String descricao;  
private double quantidade;  
private double valorUnitario;  
};
```


Os atributos desta classe são em tudo semelhante aos atributos mínimos exigidos.

2.13 Utilizador

Uma das subclasses da classe Conta, e que faz parte do Modelo do Sistema.

Cada utilizador, além de possuir os atributos da classe conta, possui também o atributo:

```
private int encTransportadas;
```

que indica quantas encomendas foram transportadas para o utilizador (cada vez que um voluntário ou transportadora efetua o transporte um método incrementa o valor de encTransportadas);

2.14 Voluntario

Uma das subclasses da classe Conta, e que faz parte do Modelo do Sistema;

Cada voluntario, além de possuir os atributos da classe conta, possui também os atributos:

```
private double raio;  
private String encAceite; //Encomenda que aceitou  
private boolean disponivel;  
private List<Integer> classificacao;  
private boolean medicamentos;  
private double velocidade;
```

Destas, aquelas que foram acrescentadas por decisão do grupo foram as seguintes:

String encAceite - Indica qual a encomenda na posse do voluntário para entregar

boolean disponivel - Indica se o voluntário pode receber uma nova encomenda (se já tiver aceitado uma encomenda, então não está disponível)

List<Integer> classificacao - Todas as classificações atribuídas ao voluntário.

boolean medicamentos - Indica se o voluntario pode transportar encomendas médicas

double velocidade - velocidade média a que o voluntário se desloca;

2.15 Loja

Uma das subclasses da classe Conta, e que faz parte do Modelo do Sistema;

Cada loja, além de possuir os atributos da classe conta, possui também os atributos:

```
private Queue<Encomenda> filaEspera;  
private double tempoEsperaIndividual;  
private List<Encomenda> encProntas;
```

Queue<Encomenda> filaEspera - fila com os códigos de encomendas à espera para serem despachados;

List<Encomenda> encProntas - fila com todas as encomendas despachadas pela loja. Não há tempo de espera neste caso.

double tempoEsperaIndividual - tempo médio de espera, por pessoa, na fila da loja, sendo que este tempo pode ser superior caso haja atrasos. Falaremos disso daqui a pouco.

2.16 Transportadora

Uma das subclasses da classe Conta, e que faz parte do Modelo do Sistema;

Cada transportadora, além de possuir os atributos da classe conta, possui também os atributos:

```
private String nif;  
private double raio;  
private double precoKm;  
private double precoKg;  
private List<Integer> classificacao;
```

List<Integer> classificacao - Todas as classificações atribuídas à transportadora.

2.17 Menu

O Menu é a Vista do programa. Comunica apenas com o Controller e os Controllers das entidades. Possui vários métodos, nomeadamente para receber algum tipo de informação da entidade que usa a app, para mostrar algum tipo de informação ou mostrar mensagens de erro.

2.18 Interface Randoms

De forma a tentar resolver o problema da gestão de tempos de fila de espera e os atrasos no deslocamento das entidades de transporte, criamos a interface Random, apenas com métodos default, que se baseiam essencialmente em probabilidades. De forma resumida acontece o seguinte:

- Quando entidade de transporte chega à loja para pegar a encomenda, a sua entrega pode ser despachada mais rápido, logo o tempo na fila de espera será menor do que o inicialmente pensado, ou a sua entrega pode ser atrasada e assim o tempo na fila de espera será maior.

- Quando entidade de transporte está a realizar a entrega existe a probabilidade de haver um atraso. Caso isto se verifique, implementamos ainda a ideia de que há 3 tipos de atraso: um pequeno atraso, um atraso médio, ou um grande atraso.

Em ambos os casos os tempos são gerados aleatoriamente.

2.19 Interface TranspVolunt

Interface com métodos comuns a ambos os voluntários e transportadoras.

2.20 TipoConta

A classe Enum TipoConta permite que os vários tipos de contas sejam encarados como "constantes" e foi-nos útil para situações que dependendo da conta uma condição diferente teria de ser realizada. Por exemplo, ao fazer login, será devolvido um TipoConta, sendo que dependendo deste será invocado o método run em uma das classes controladoras das entidades.

2.21 Point

Classe com métodos que transforma as latitudes e longitudes em distâncias em KM.

Capítulo 3

Estrutura do projeto

O nosso projeto segue a estrutura *Model View Controller* (MVC), estando por isso organizado em três camadas:

- A camada de dados (o modelo) é composta pelas Classes da Entidades e pelas classes Main, Conta, Contas, Encomendas, Encomenda, LinhaEncomenda e pelas interfaces.
- A camada de interação com o utilizador (a vista, ou apresentação) é composta unicamente pela classe Menu.
- A camada de controlo do fluxo do programa (o controlador) é composta pela classe Controller, pelo Controller das Entidades, e pelas classes Estado e TrazAqui, sendo que destas apenas a Controller e Controller das Entidades acede á vista diretamente.

Como foi referido anteriormente, todo o projeto baseia-se na ideia de encapsulamento, á exceção das classes Estado e TrazAqui cuja relação é de agregação. Escolhemos est relação para estas duas classes especificamente pelo simples motivo de que ao partilhar, o apontador da conta com que se fez login, se este for mudado no Estado, a contaLoggedIn em TrazAqui é mudada automaticamnte, algo que seria mais trabalhoso caso a relação fosse de composição.

Capítulo 4

Conclusão

A nível geral, e tendo em conta o que foi explicado nos capítulos anteriores, podemos afirmar que temos um projeto bem conseguido. Pensamos que respondémos de forma correta ao problema da gestão de factores de aleatoriedade, que era a principal dificuldade do trabalho, além de termos implementado as regras de encapsulamento (tirando a exceção já referida) e a possibilidade do nosso projeto crescer de forma controlada.

Apêndice A

Diagrama de Classes