



**Universidade do Minho**  
Escola de Engenharia

# PROGRAMAÇÃO ORIENTADA AOS OBJETOS

## RELATÓRIO PROJETO JAVA

TRAZ AQUI

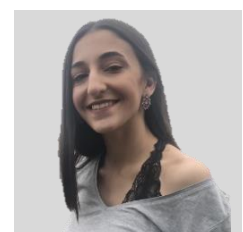
Grupo 18



Luís Miguel Pinto A89506



Ana Luísa Carneiro A89533



Ana Rita Peixoto A89612

# ÍNDICE

---

INTRODUÇÃO .....	2
ARQUITETURA DE CLASSES .....	3
MVC - DIAGRAMA DE CLASSES.....	3
ENTIDADES & DECISÕES TOMADAS .....	4
DADOS APP .....	4
UTILIZADOR .....	4
VOLUNTÁRIO .....	5
LOJA .....	5
ENCOMENDA .....	5
TRANSPORTADORA.....	6
GPS.....	6
PARSE.....	6
ARQUITETURA DA APLICAÇÃO .....	7
CRIAÇÃO DE UMA ENCOMENDA.....	7
OUTRAS FUNCIONALIDADES .....	8
CONCLUSÃO .....	9

# INTRODUÇÃO

---

O projeto “Traz Aqui” propõe a implementação de um programa que seja capaz de processar pedidos e entregas de encomendas, tentando respeitar os princípios de modularidade, encapsulamento e programação orientada aos objetos.

No decorrer do desenvolvimento deste projeto, foram surgindo alguns desafios, os quais passamos a enumerar:

- ➔ O primeiro desafio surgiu durante a interpretação do enunciado e, com base no que entendemos ser o pretendido com o projeto, tentar desenvolver uma aplicação de fácil utilização.
- ➔ Um outro desafio foi perceber quais as variáveis/atributos que faziam sentido acrescentar a cada entidade. Assim como, resolver o hiato entre as variáveis implementadas no sistema e as disponíveis no ficheiro de *logs*, já que nem todas as variáveis se encontravam neste ficheiro.
- ➔ Também consideramos desafiante implementar a noção de fila de espera nas lojas e a noção de rota nas transportadoras, devido às diferentes extrapolações que podiam advir do enunciado.
- ➔ Finalmente, foi desafiante a criação de funcionalidades extra que permitem gerar fatores de aleatoriedade e outras capazes de diferenciar estados de encomenda.

# ARQUITETURA DE CLASSES

## MVC - DIAGRAMA DE CLASSES

O projeto encontra-se inserido num *package* inicial “TrazAqui” que foi dividido em três *packages* (“Model”, “View”, “Controller”) de modo a respeitar o padrão MVC.

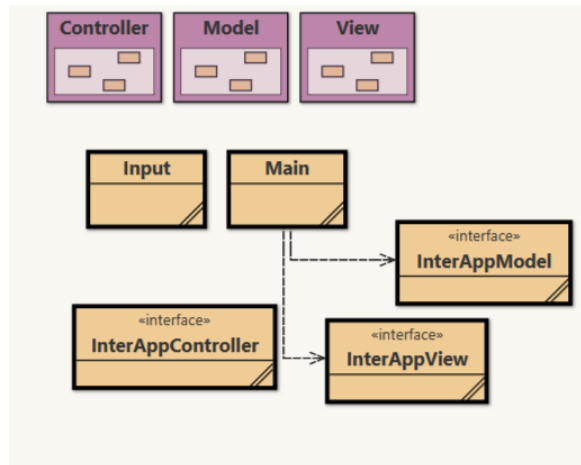


Figura 1: Package TrazAqui

Assim, o *package* model, que contém as camadas de dados e algoritmos, está organizado de acordo com o seguinte diagrama de classes<sup>1</sup>.

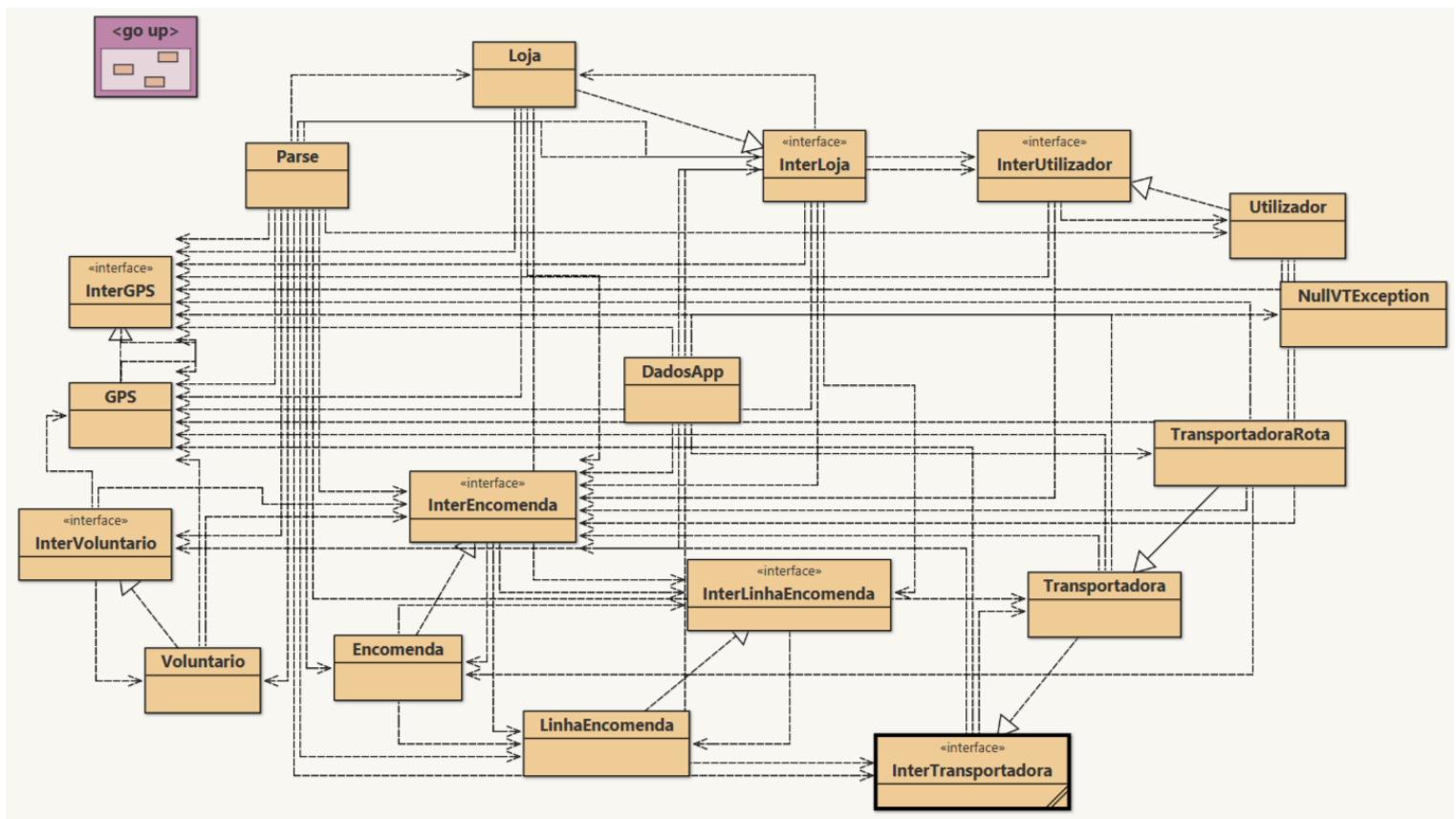


Figura 2: Package Model

<sup>1</sup> Obtidos através do BlueJ.

E, por fim, ilustramos os *packages* View (que se encarrega da apresentação ao utilizador) e Controller (controlo de fluxo) nos seguintes diagramas<sup>2</sup>:

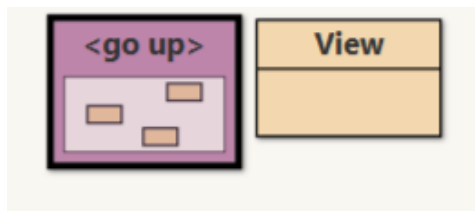


Figura 3: Package View

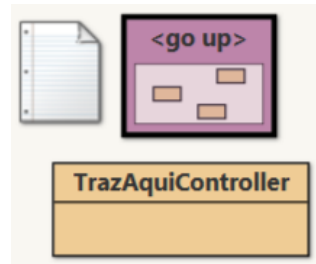


Figura 4: Package Controller

## ENTIDADES & DECISÕES TOMADAS

De modo a melhor respeitar os princípios do encapsulamento, decidimos implementar interfaces referentes às classes do nosso projeto.

Um ponto comum às entidades que utilizam o programa (transportadoras, voluntários, utilizadores e lojas), trata-se do código de utilizador, email e password de cada uma destas. O código é um valor do tipo String, em que o primeiro carácter do mesmo remete à entidade correspondente e permite diferenciá-la das restantes que utilizam o programa, e os seguintes dígitos são um número gerado aleatoriamente com o uso da classe Random. Deste modo, cada entidade irá ter um código diferente. Em complemento ao mesmo, decidimos adicionar uma outra variável, o *email* que é criado através da concatenação do código de identificação com “@trazaqui.com”. A palavra passe, é generalizada para todas as entidades de modo a facilitar o fluxo de encomendas e do programa, e adotamos como convenção “1234”.

## DADOS APP

Na classe DadosApp é constituída pelos utilizadores, voluntários, transportadoras e lojas que fazem parte do sistema assim como todas as encomendas feitas pelos utilizadores. Para além disto também é composta por uma entidade log que diz respeito ao código da entidade que está atualmente a utilizar o sistema.

DadosApp	
utilizadores	Map<String, InterUtilizador>
voluntarios	Map<String, InterVoluntario>
transportadoras	Map<String, InterTransportadora>
lojas	Map<String, InterLoja>
encomendas	Map<String, InterEncomenda>
log	String

Esta classe incorpora todos os métodos necessários para responder as funcionalidades da aplicação.

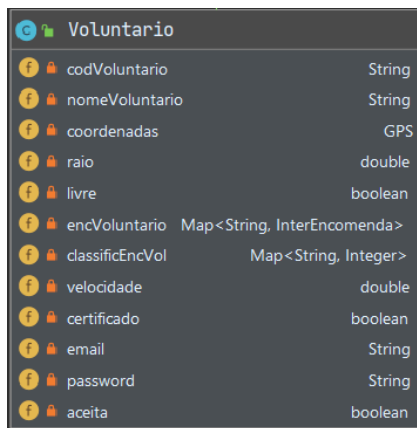
## UTILIZADOR

Utilizador	
codUtilizador	String
nomeUtilizador	String
coordenadas	InterGPS
encUtilizador	Map<String, InterEncomenda>
email	String
password	String

Entidade “motora” da aplicação, usa a aplicação para pedir entregas de encomendas. Para a ativação da conta é usado um email, nome, coordenadas GPS e password. Será depois caracterizado internamente por um código de utilizador (ex.: “u40”). Tem ainda associado ao próprio um registo de todas as encomendas que fez na aplicação.

<sup>2</sup> Obtidos através do BlueJ.

## VOLUNTÁRIO

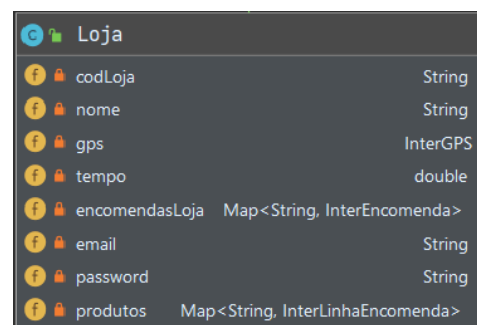


Voluntario	
codVoluntario	String
nomeVoluntario	String
coordenadas	GPS
raio	double
livre	boolean
encVoluntario	Map<String, InterEncomenda>
classificEncVol	Map<String, Integer>
velocidade	double
certificado	boolean
email	String
password	String
aceita	boolean

A classe voluntário é constituída por toda a informação respetiva a uma das entidades responsáveis pelo transporte de encomendas, ou seja, informação sobre o código, nome, email e password. Para além disto, a classe também possui informação sobre as coordenadas, raio de ação e velocidade de deslocamento da entidade que vão servir para o cálculo de tempo da encomenda que lhe foi atribuída. Tal como a transportadora, o voluntário também possui ou não certificado medico. A entidade também tem a capacidade de determinar se esta livre para aceitar encomendas e se aceita o transporte de encomendas medicas. Finalmente, o voluntario possui um histórico de encomendas realizadas assim como as múltiplas avaliações feitas pelos utilizadores.

## LOJA

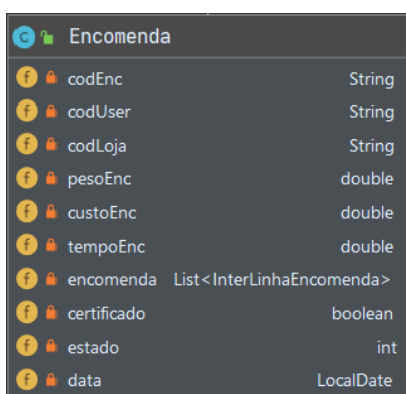
Na classe loja é constituída por todas as informações em relação á entidade loja, isto é, informações sobre o seu nome, as suas coordenadas, o seu código no sistema, email e password para o login na aplicação. Para além desta informação, encontra-se também presentes estruturas como a *encomendasLoja* que guarda as encomendas feitas á loja assim como *produtos* que possui o inventario de produtos da loja.



Loja	
codLoja	String
nome	String
gps	InterGPS
tempo	double
encomendasLoja	Map<String, InterEncomenda>
email	String
password	String
produtos	Map<String, InterLinhaEncomenda>

Cada classe possui uma variável, *tempo*, que determina o tempo médio de atendimento nessa loja. Este tempo é determinado recorrendo á classe Random, sendo que se o tempo for zero então a loja não possui fila de espera. Esta variável é usada para determinar o tempo estimado de que cada encomenda.

## ENCOMENDA



Encomenda	
codEnc	String
codUser	String
codLoja	String
pesoEnc	double
custoEnc	double
tempoEnc	double
encomenda	List<InterLinhaEncomenda>
certificado	boolean
estado	int
data	LocalDate

A classe Encomenda possui toda a informação relacionada com encomendas, isto é, os códigos da encomenda, do utilizador que realizou o serviço e da loja que foi encomendado. Para além disso também possui a data em que foi criada assim como o estado da mesma (0 – Esta em loja, 1 – Esta com voluntário/transportadora, 2 – Entregue ao utilizador). Esta classe também possui uma variável *encomenda* que é composta por várias linhas de encomenda, isto é, os vários produtos encomendados da loja.

Na nossa aplicação há 2 tipos de encomenda, médica ou não médica, sendo essa característica dada pela existência ou não do *certificado* (true – medica, false – não medica). Esta qualidade terá repercussões na atribuição de um voluntario/transportador, pois este também tem de ter certificado e estar disponível para entregar encomendas médicas.

O peso da encomenda é dado partir da geração de um número aleatório através da classe Random. O custo é determinado a partir do preço unitário de cada produto e pela

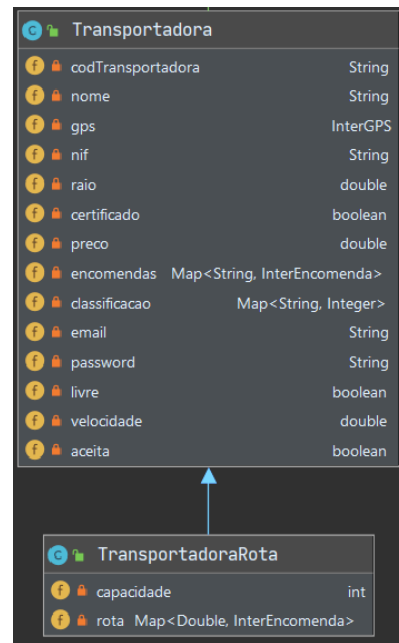
quantidade pedida pelo utilizador. Para além disso se a essa encomenda for atribuída a uma transportadora terá de ser adicionada uma taxa por km que depende de cada transportadora assim como uma taxa sobre o peso da encomenda. Finalmente, o tempo é atribuído através do somatório entre tempo medio da loja e o tempo que leva o voluntário ou transportadora a irem até á loja e da loja até ao utilizador, para isso é necessário saber a velocidade a que voluntários e transportadoras andam.

## TRANSPORTADORA

Para esta classe, que denota uma das entidades de transporte suportadas pelo nosso programa, adotamos a sua implementação através de hierarquia de classes, de modo a ser possível a criação de uma *TransportadoraRota*, que se trata de uma especificação da classe *Transportadora* propriamente dita.

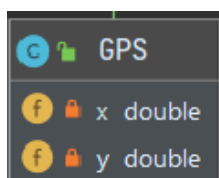
Assim, tal como é visível na figura, foram criadas variáveis de instância de identificação de um objeto desta classe, das quais destacamos o código da transportadora, que permite a exclusividade da mesma.

Em adição ao referido, implementamos também várias outras variáveis de modo a responder aos requisitos propostos. O raio (que permite averiguar quais os utilizadores e lojas abrangidos por uma transportadora), o certificado (informa se está apta para transportar encomendas médicas), o preço (preço por km), a velocidade média a que uma transportadora se desloca, e por fim, as variáveis *aceita* e *livre*, que denotam o caso em que se trata de aceitar ou não encomendas médicas (para os casos em que há certificado) e se a entidade está disposta para recolher encomendas, respetivamente. Para complementar, decidimos implementar uma estrutura onde fosse possível guardar as encomendas transportadas e a classificação de cada serviço, por parte do utilizador correspondente, com auxílio a coleções do Java, neste caso o *Map*.



No caso da subclasse *TransportadoraRota*, esta herdou todas as variáveis referentes à *Transportadora* referidas anteriormente. Além disso, também lhe foi acrescido variáveis que a especificam e diferenciam de uma transportadora normal, as quais: mapeamento com a rota (conjunto de encomendas que estão prontas a ser entregues) e a capacidade (isto é, o número de encomendas máximo que uma transportadora pode entregar, ao mesmo tempo).

## GPS



A classe GPS engloba todas as funcionalidades ligadas às coordenadas de cada entidade. Esta classe define métodos que vem distâncias entre duas coordenadas assim como se estas se encontram dentro de um determinado raio. Nota: as coordenadas (latitude e longitude) são vista como pontos num referencial cartesiano.

## PARSE

Esta classe é aquela que efetua a leitura do ficheiro inicial, que irá popular as estruturas e inicializar o nosso programa. Aqui, módulo em que a leitura é feita com base na classe *Files*, é onde são tratados os dados lidos inicialmente.

Assim, a nossa ação perante o ficheiro fornecido, foi adicionar às nossas entidades as informações concretas presentes no mesmo. Dados tais como: certificado médico, velocidade, data em que foi efetuada uma encomenda, tempo que demorou a entrega de uma encomenda e o tempo médio de fila de espera numa loja, são variáveis que constituem as nossas classes, mas que não são fornecidas no ficheiro. Deste modo, de maneira a contornar este obstáculo, decidimos preencher estas variáveis com valores aleatórios (em que definimos um valor mínimo e máximo para cada), de modo que pudéssemos tirar partido de todas as funcionalidades do programa.

A classe *Parse* não possui variáveis de instância, mas sim um método que recebe a nossa estrutura principal do programa, *DadosApp*, e que a preenche quando o ficheiro é lido.

## ARQUITETURA DA APLICAÇÃO

---

A nossa aplicação esta contruída com base em menus dedicados a cada uma das entidades assim, o menu inicial do nosso programa tem uma opção login, que permite aceder as funcionalidades de cada entidade. Nesta opção o acesso á aplicação faz através do uso do email e passwords validos. Para além disto é também possível registar uma nova entidade (utilizador, voluntario, loja, ...) no sistema assim como ler e guardar num ficheiro objeto. Finalmente, neste menu principal também é possível ver o top dos utilizadores que mais utilizaram o sistema e as transportadoras com mais quilómetros feitos.

### CRIAÇÃO DE UMA ENCOMENDA

Tudo começa com o login de um utilizador, que de seguida efetua um pedido de encomenda numa loja a sua escolha. Neste momento, o sistema encarrega-se de tentar encontrar um voluntario ou uma empresa transportadora capaz de fazer o transporte da encomenda, desde a loja até a localização do utilizador.

Para isso, o sistema baseia-se num sistema de prioridades e tem em conta alguns requisitos , i.e, irá sempre procurar primeiro se existe algum voluntario capaz de realizar a entrega antes de procurar por uma empresa transportadora (uma vez que voluntários não cobram o transporte e o objetivo da app é ajudar as pessoas durante a pandemia)e, em segundo lugar, irá sempre fazer uma procura tendo em conta o transportador com menor deslocamento necessário (visando desta forma minimizar o tempo de entrega das encomendas). Em relação aos requisitos mencionados anteriormente, uma encomenda do tipo médica só poderá ser entregue por um transportador certificado.

Caso seja selecionado um voluntario, a encomenda é diretamente encaminhada para a loja, no entanto, caso seja uma empresa transportadora, o utilizador será informado dos custos associados e terá a opção de aceitar ou recusar o transporte por parte da mesma.

Na loja, mal o pedido esteja pronto, esta notifica o transportador que pode levantar a encomenda. Uma vez entregue ao transportador e feita a viagem até ao destino final, basta apenas este sinalizar ao utilizador que pode recolher a encomenda.

Finalmente, caso deseje, o utilizador pode avaliar o serviço prestado numa escala de 0 a 5. Nota: Durante o processo de entrega o utilizador pode acompanhar o estado da encomenda através da aplicação, receberá a informação “Em loja” ou “A caminho”.



## OUTRAS FUNCIONALIDADES

Além as funcionalidades que permitem a execução base do programa, de modo a proporcionar uma melhor experiência ao utilizador, foram implementadas funcionalidades extra.

Deste modo, no menu referente a cada uma das entidades, é possível verificar o histórico de encomendas. Além disso, no caso dos voluntários e das transportadoras, é possível verificar a classificação efetuada pelos utilizadores, a cada encomenda previamente transportada pelos mesmos. Para estas entidades, também é possível declarar fim de turno, isto é, a própria pessoa alterar o seu estado para “não livre”. O mesmo acontece para o estado de aceitação de encomendas médicas, que é decidido pela própria entidade de transporte, e pode ser alterado a qualquer momento.

## CONCLUSÃO

---

Dado por concluído o nosso projeto, apresentamos uma reflexão sobre possíveis melhorias, assim como aspetos a valorizar.

No desenvolvimento da aplicação “Traz Aqui” consideramos positivos os seguintes pontos:

- ➔ Implementação de funcionalidades extras como o histórico de encomendas de cada entidade, a verificação do estado das encomendas ainda não entregues ao utilizador, criação de tops sobre o sistema e apresentar as classificações de cada transporte.
- ➔ Implementação de fatores gerados aleatoriamente como a velocidade de transporte, peso da encomenda e tempo da loja, que terão consequência no tempo e custo da entrega.
- ➔ Para além destas implementações, também achamos positivo o facto de termos satisfeito os requisitos básicos proposto no enunciado.

Por outro lado, consideramos que existem pontos a melhorar no nosso projeto:

- ➔ No que toca à capacidade de a aplicação evoluir de forma controlada, notamos que seria mais apropriado a implementação de uma hierarquia de *Encomenda* e de *Transporte*, que englobasse todo o tipo de encomendas e entidades que fazem entregas, respetivamente.
- ➔ Em relação à divisão do trabalho no modo MVC, achamos que seria mais benéfico a criação de mini-MVC's para os vários menus das entidades. Assim o controller ficaria menos sobrecarregado e a aplicação mais organizada.

De um modo geral, o trabalho cumpriu com os requisitos propostos e apesar de haver melhorias, achamos que o balanço é positivo.