Universidade do Minho

Mestrado Integrado em Engenharia Informática

Universidade do Minho

TrazAqui

Grupo 47

Carlos Preto (A89587)

Pedro Veloso (A89557)

José Pereira (A89596)

Índice

Introdução

Como método de avaliação da disciplina de POO, foi pedido que se elaborasse um projeto em linguagem Java que simulasse um serviço de entrega de encomendas em casa. Foi-nos também pedido que utilizássemos a estrutura Model, View, Controller, nunca esquecendo de garantir o encapsulamento de dados.

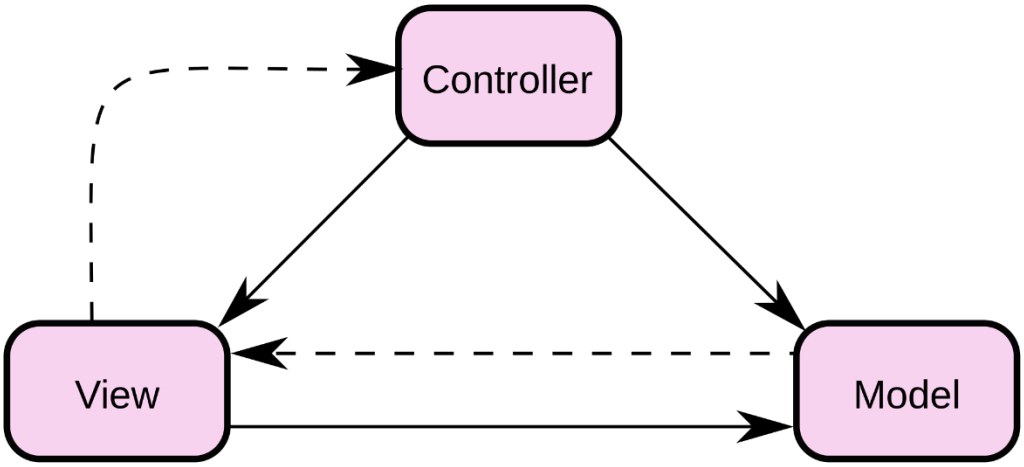
Para o funcionamento correto do projeto, seria necessário que fosse possível pedir uma encomenda a uma loja, por parte de um Utilizador. Posteriormente a loja sinalizava que teria uma encomenda pronta para uma Transportadora ou Voluntario vir transportá-la até casa do Utilizador. Caso se tratasse de uma Transportadora, teria de haver uma posterior confirmação por parte do Utilizador de que realmente aceitava a despesa associada ao transporte da sua Encomenda. Quando transporte fosse concluído, o utilizador poderia avaliar o transportador e posteriormente iria se repetir este ciclo para qualquer outro Utilizador que entrasse no Sistema.

É também importante realçar que de maneira a poder controlar qualquer tipo de ação, será necessário fazer login no Sistema com o tipo especificado, ou seja, quando se pretender tratar alguma coisa da Loja, essa Loja terá de fazer o seu login e controlar as diferentes situações.

Assim, neste Relatório, iremos abordar a estratégias que decidimos adotar, bem como outras que decidimos não optar, apresentado a devida razão para cada uma delas, usando como auxílio os Diagramas de Classe.

MVC

Antes de falarmos das Estruturas em si, falaremos da maneira como estas foram organizadas no Projeto de maneira a seguir o funcionamento desejado pelos Docentes.



Como podemos observar na imagem, o padrão MVC de auxílio ao utilizador, de maneira a poder interagir de diferentes maneiras com o Programa. Este padrão também permite uma melhor organização do Projeto, pois cada um dos componentes realizará uma função diferente. Falaremos então de cada um deles em específico.

- ✓ Model

  O Model contém todas as classes, nas quais a informação presente é usada para o tratamento de dados apenas. As classes nelas presentes não podem interagir com o Usuário, apenas sendo usadas para serem invocadas numa fase seguinte, quando houver dados para tratar.

- ✓ Controller

  O Controller processa as informações vindas da View. Uma vez que há diferentes Views, poderá ser o Controller a invocar cada uma delas, com os seus devidos dados recebidos do Model.

- ✓ View

  A View serve para apresentar no ecrã informação útil para o Utilizador se orientar no programa, e mostrando também o resultado das diferentes opções. Diferentes contextos requerem diferentes View, sendo que cada View terá de ter um Controller associado a si, de maneira a poder tratar todos os inputs. Uma vez que existem vários tipos de padrões MVC, adotamos um que permite a View conhecer também o Model, que lhe passa informação a mostrar ao utilizador.

- ✓ Exceptions

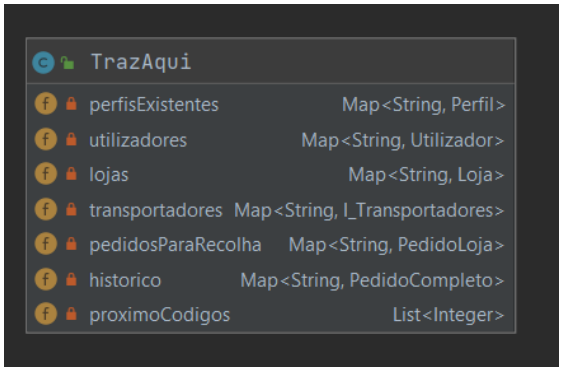
  Para além do padrão MVC, o grupo criou também um Padrão para que quando houvesse ocorrências de exceções, como a inexistência de um produto ou de uma encomenda, o programa simplesmente não “morresse” e passa-se a informar o utilizador do erro que originou essa exceção.

Classes

TrazAqui e TrazAquiApp

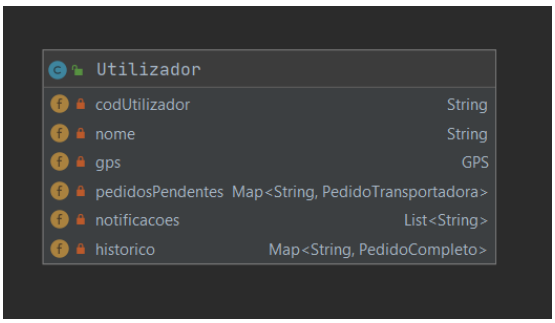
A classe **TrazAqui** irá conter todas as classes principais do nosso trabalho (lojas, utilizadores, transportadores, etc.), sendo por isso a principal classe do Programa. Contém variáveis correspondentes às lojas, utilizadores, transportadores, histórico, quais os perfis existentes no programa, quais os pedidos para recolher e uma lista que contém os próximos códigos a serem gerados.

A classe **TrazAquiApp** trata de carregar o sistema do formato binário, permitindo recuperar o estado num ponto de paragem. Caso haja algum erro carregam-se os dados default do programa do ficheiro Logs. Para além disso, é responsável por inicializar o programa, pondo assim a primeira view a correr.

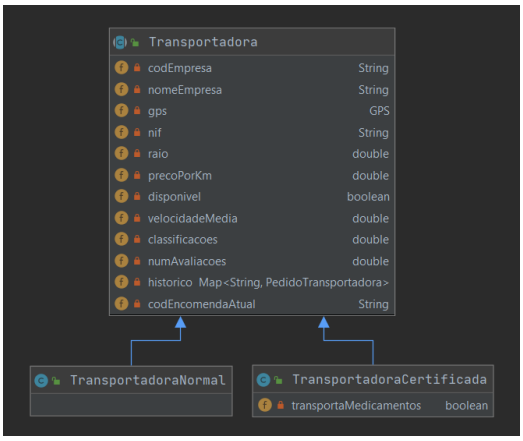


Utilizador

Um Utilizador representa o Cliente no Sistema. Quando se pretende pedir um Encomenda é com este que se tem de fazer Login e dar início ao ciclo. Cada Utilizador contém informações como: **código, nome, GPS, pedidos Pendentes, notificações e histórico,** sabendo que o necessário para gerar um pedido de encomenda seriam apenas as 3 primeiras. É importante ter um Map com os **Pedidos Pendentes**, uma vez que sempre que um Utilizador efetuar uma encomenda e esta seja aceite por uma transportadora, é necessário o utilizador aceitar a mesma, desta forma é guardado no map todos os pedidos para entrega feitos ao utilizador por uma transportadora e que o preço ainda não foi aceite. As **notificações** avisam o Utilizador de que uma Transportadora aceitou transportar a encomenda e que agora cabe a este decidir se aceita ou não que o transporte seja realizado, caso seja um voluntario as notificações apenas informam que a encomenda irá ser entregue. Por fim adicionamos um **histórico** para que ficassem registadas todas as Encomendas que o Utilizador fez.



Transportadora



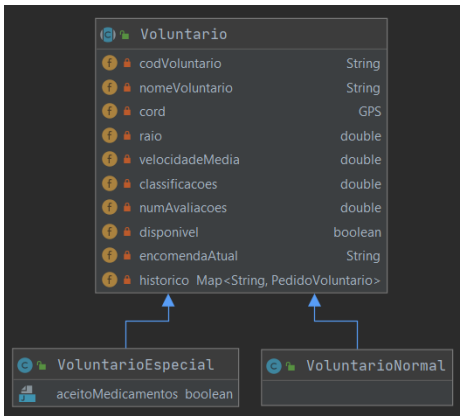
Uma Transportadora fica encarregue de se deslocar até uma Loja e recolher uma Encomenda, e posteriormente fazer o seu transporte até aos Utilizadores. De maneira a ser apta para transportar um Encomenda terá de se encontrar quer no raio da Loja, quer no raio do Utilizador, terá de se encontrar disponível e, conforme as características das Encomendas, verificar se está classificada para fazer o seu transporte. Falemos agora no porque da escolha de certas variáveis instância. É importante saber qual a encomenda que está atualmente atribuída a uma Transportadora, por isso esta terá o código da encomenda que atualmente a está a ocupar. À semelhança do Utilizador, atribuímos um Histórico que armazena todos os Pedidos que uma Transportadora já finalizou. Outros fatores como **precoPorKm** e **velocidadeMedia** são usados para futuros cálculos de tempo e preço associado ao transporte de uma Encomenda, além do preço associado a cada um dos produtos presentes nessa.

Uma vez que uma Encomenda pode conter medicamentos, seria necessário distinguir entre **TransportadorasCertificadas** para transporte de medicamentos e **TransportadorasNormais** (sendo que estas últimas não implementam nenhum método extra, logo é idêntica em tudo à classe Transportadora)

A TransportadoraCertificada é capaz de fazer transporte de encomendas com medicamentos, sendo que por isso lhe adicionamos uma variável denominada de **transportaMedicamentos** que sinalizará se nesse momento aceita transportar medicamentos (o facto de poder transportar medicamentos não significa que a qualquer momento possa fazer o transporte destes).

Por fim é necessário referir 3 estados diferentes que uma transportadora pode ter. O primeiro ocorre quando a transportadora se encontra disponível e não ocupada, estando desta forma livre para escolher encomendas para entrega. Contudo pode estar também indisponível e ocupada, estando assim em fase de entrega, e pode estar disponível e ocupada, ou seja, pediu para entregar uma encomenda e está à espera pelo aval do utilizador.

Voluntario

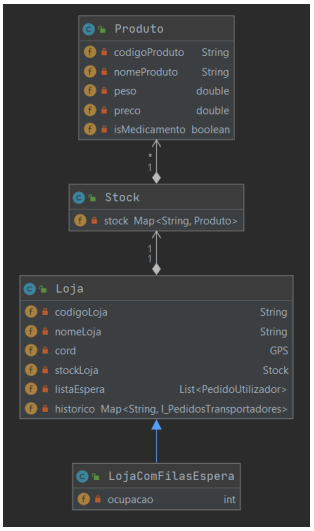


À semelhança da Transportadora, um Voluntario trata de transportar as encomendas de um Utilizador, sendo que a única diferença seja o facto de um Voluntario não ter um custo associado a esse mesmo transporte, logo sempre que um Voluntario fique encarregue de Transportar uma Encomenda, o Utilizador apenas terá de se preocupar em pagar o preço dos produtos que encomendou.

Em termos de variáveis de instância, a principal diferença em relação à Transportadora é a inexistência da variável precoPorKm. Também dividimos este em **VoluntarioEspecial** e **VoluntarioNormal**, de maneira representar que pode ou não transportar medicamentos.

Não havendo a necessidade de o utilizador aceitar o transporte de uma encomenda realizado por um voluntario, ficamos assim com 2 estados diferentes possíveis para o voluntario. O primeiro ocorre quando o voluntario se encontra disponível e não ocupado, estando desta forma livre para escolher encomendas para entrega. O segundo quando o voluntario está em fase de entrega, ficando assim indisponível e ocupado.

Loja



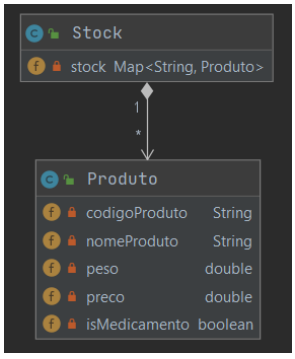
A Loja contém o Stock de Produtos que um Utilizador poderá optar por comprar. Para além disso indica qual o seu **código, nome, GPS, uma lista de Espera e Histórico**. O histórico apresenta todos os Pedidos que já foram aceites por essa Loja e também aceites para transporte.

É parte da loja também uma lista com todos os pedidos que a loja ainda não aceitou. Contudo na hora de escolha de aceitar um pedido a loja pode escolher qualquer um dos que se encontram na lista, não sendo obrigada a respeitar a ordem de chegada dos pedidos.

Uma loja além de aceitar pedidos dos utilizadores pode também adicionar/remover produtos ao/do seu stock e verificar o seu histórico de vendas, que é apenas constituído pelas encomendas que estão realmente concluídas, ou seja, encomendas que foram aceites pela loja e já existe algum transportador encarregue da mesma.

Por fim, existem **Lojas com filas de espera**, que implementava todos os métodos de uma Loja e para além desses acrescentava um inteiro correspondente à **ocupação** da Loja. Posteriormente, com base nessa ocupação, é nos possível dizer qual o estado da Fila de Espera nessa Loja, para que assim se saiba quanto tempo terá de se esperar na Loja até poder tratar de uma Encomenda. Esse tempo irá, obviamente, afetar o tempo e preço de transporte de uma Encomenda.

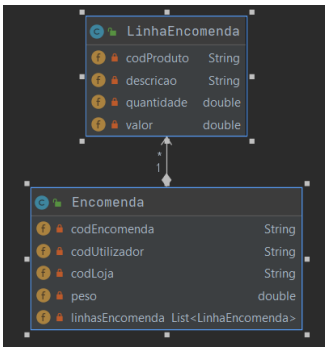
Stock e Produto



Um **Stock** é representado por um Map onde cada Key corresponde ao **código do Produto** e o Value ao **Produto** associado a esse código. Cada Produto é constituído por um **código**, pelo seu **nome**, **peso**, **preço** e a sinalização se é ou não um **medicamento.** Estas informações do Produto permitem numa faze futura calcular o preço da encomenda e do transporte deste, sabendo que caso seja medicamento terá de ter um tratamento especial, daí a existência de Transportadoras e Voluntários certificados.

Encomenda

Uma **Encomenda** é definida pelo seu **código**, o **código do Utilizador** que a realizou, o **código da Loja** onde foi pedida, o seu **peso** e uma lista de **LinhasEncomenda.** Cada linha de Encomenda será constituída pelo **código do Produto** que transporta, uma **descrição** deste, bem como a sua **quantidade** e **valor.** Assim, é possível ao Utilizador fazer uma Encomenda onde deseje mais que um Produto.

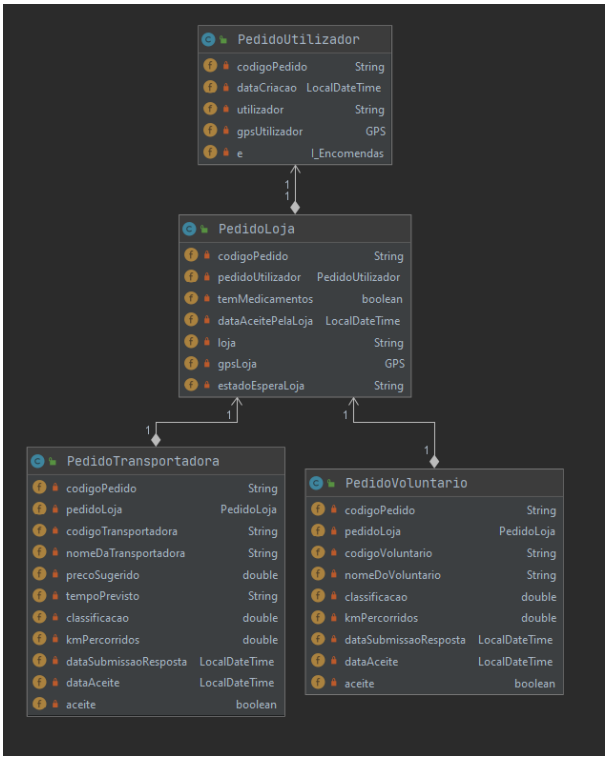


Pedido

Uma vez que é possível haver 4 tipo de Usuários no nosso Programa, achamos por bem que houvesse um Pedido específico para cada um destes, tendo criado para isso os seguintes Pedidos:

- ✓ **PedidoUtilizador**
- ✓ **PedidoVoluntario**
- ✓ **PedidoTransportadora**
- ✓ **PedidoLoja**
- ✓ **PedidoCompleto**

A informação presente em cada um destes é bastante similar, porém dependendo do tipo, haverá informações adicionais que achamos por bem colocar. Começando pelo **PedidoUtilizador**, este irá ter o código do Pedido, a sua data de criação, o código do Utilizador que o realizou, e a Encomenda nele presente. Já no caso do **PedidoLoja**, também haverá o código do Pedido, porém a restante informação é diferente. Optamos por incluir neste o PedidoUtilizador, a sinalização da existência ou não de medicamentos, a data de aceitação do Pedido por Parte da Loja, o código da Loja, a sua Localização e o estado de espera na Loja onde o Pedido foi realizado.



Seguidamente, o **PedidoVoluntario**, é constituído pelo seu Código, o PedidoLoja onde foi tratada a Encomenda, o código do Voluntário que transportará o Pedido, bem como o seu nome, classificação, número total de Kilometros percorridos, a data de Submissão da Resposta do Voluntario, a data em que

efetivamente foi aceite e se realmente já foi aceite. A única diferença entre o **PedidoTransportadora** e o Pedido anterior encontra-se nos códigos e nome do Transportador, que passa a ser transportadora e não voluntário e em mais duas instâncias. A primeira sendo o preço sugerido, uma vez que tem um preço associado ao transporte da encomenda e dos Produtos que esta contém. A segunda refere-se ao tempo previsto da Viagem até ao Utilizador.

Por fim, criamos um **PedidoCompleto,** que representa o Pedido finalizado. Neste apenas temos informação acerca do código do Pedido, o Pedido que foi finalizado, a data em que foi entregue e a sinalização se o Transportador desse Pedido já foi ou não avaliado.

Restantes Classes

Além das Principais Classes do nosso Programa referidas anteriormente, temos classes que tratam de fatores de Aleatoriedade, ler Ficheiros e verificar quais os Tipos de Utilizadores que se encontram atualmente no programa.

Aleatoriedade

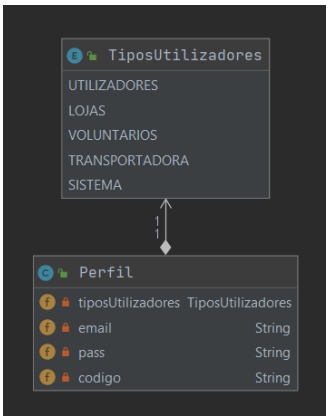
De maneira a tornar o projeto mais interessante, decidimos criar fatores de Aleatoriedade, que afetassem o tempo de transporte de encomendas e que se aproximassem o mais possível da realidade do dia-a-dia de uma Pessoa. Assim, usamos **Enums** para enumerar esses diferentes fatores. O Enum **CondicoesAtmosfericas** indica-nos como estará o tempo na altura do transporte de uma Encomenda, e tal como na vida real, quando está a chover há que conduzir com prudência, aumentando o tempo de transporte.

Outro fator importante será **EstadoEstrada,** pois pior o pavimento da condução, mais cuidado será necessário e dai mais tempo será necessário para o transporte. Por último, e talvez o mais óbvio, definimos o **Transito,** onde quantos mais veículos houver na estrada, mais tempo será desperdiçado.

A classe de Aleatoriedade tratará de gerar um inteiro de 0 a 3 que corresponderá a um estado dos diferentes Enums, e onde será calculada uma taxa de acréscimo ao tempo de transporte do Pedido.

Perfil

De maneira a poder navegar pelo sistema, um usuário terá de ter um perfil registado. Informação como email, password, código e tipo de Utilizador são necessários para preencher o perfil de um Usuário. Para saber que tipos de Utilizadores existem, criamos um **Enum TiposUtilizadores** que contem os tipos **UTILIZADORES**, **LOJAS**, **VOLUNTARIOS**, **TRANSPORTADORA** e **SISTEMA**. É importante ter estes tipos pois assim quando for necessário ir para o menu de um dado usuário, sabendo qual o seu tipo podemos facilmente saber qual a View que carregar.



LeituraFicheiros

Realiza a leitura do ficheiro Logs e cria um TrazAqui com a informação retirada desse, para que assim haja dados no sistema. Também permite carregar um estado através da leitura de um Ficheiro com dados sobre o estado anterior do Programa, podendo assim retomar-se a um estado anterior que não o atual.

Primeiramente, utiliza-se o FileReader para ler o ficheiro “Logs” fornecido, fazendo posteriormente a separação da informação presente em cada linha. Conforme o código de Hash das palavras iniciais, ou seja, “Utilizador”, “Loja”, “Aceite”, “Encomenda”, “Transportadora” e “Voluntario”, irá decidir como guardar as diferentes informações.

Realizada essa tarefa, trata de registar quais os códigos que irão ser gerados para as próximas encomendas, para que assim não haja códigos de encomendas repetidas.

Interfaces

Uma Interface é vista como uma classe totalmente abstrata, onde são agrupados métodos que serão partilhados por classes que implementem essa Interface, ou seja, sempre que uma classe implemente uma Interface irá de ter implementado os métodos que estão definidos nessa Interface.

Assim, sempre que achamos oportuno, implementamos uma Interface que contivesse todos os métodos que essa classe precisava de ter implementado, para que quando fosse necessário aceder a uma dada classe acedermos antes à sua Interface.

Com esta implementação de interfaces torna possível alterar/acrescentar novos tipos de classes sem ter de alterar todo o conteúdo do projeto, podendo assim acrescentar, por exemplo, mais tipos diferentes de transportadores, sem ter de alterar em grande parte o nosso código. Utilizando esta estratégia tentamos que o código fique mais aberto para mudanças e evolução.

Descrição do Programa

Iremos então descrever como funciona o nosso Programa simulando o pedido de uma Encomenda e a Posterior entrega desta ao Utilizador (utilizador já existente no Sistema).

1. Quando um utilizador entrar no nosso programa é lhe imediatamente perguntado se já tem conta ou se pretende criar uma. Depois de passada essa etapa o Utilizador decide o que pretende fazer no nosso programa. Neste caso, como queremos pedir uma Encomenda, selecionamos a opção "Pedir Encomenda".

2. Depois é necessário selecionar a loja onde queremos realizar a encomenda. Posteriormente ser-nos-á apresentado no ecrã a lista de Produtos que estão disponíveis no Stock da Loja. É possível encomendar mais que um Produto e pedir a quantidade desejada de cada um.

3. Assim que finalizar a encomenda, o Utilizador é informado se esta foi concluída com sucesso, e caso tal se verifique, a encomenda é enviada para a Loja e o Utilizador é retornado ao menu Principal (onde se desejar poderá selecionar outras opções).

4. Quando a Loja aceder ao seu Perfil, irá reparar que tem um Pedido Pendente que poderá aceitar. Após aceitar a encomenda, as Transportadoras e Voluntários que estejam quer no raio do Utilizador, quer no raio da Loja poderão notar de que há uma Encomenda que está à espera de ser transportada.

5. Uma vez feito o login com um Transportadora que cumpra os requisitos, é possível pedir uma Encomenda para Entregar. Caso seja essa a opção escolhida, ser-lhe-ão apresentadas todas as encomendas que pode transportar, bastando escrever o código da encomenda que pretender transportar.

6. Após tudo isto, o Utilizador será notificado que uma transportadora se ofereceu para transportar a sua encomenda, podendo este optar por aceitar ou não os preços associados ao transporte desta. Caso aceite, a Transportadora é informada de que pode começar a fazer o transporte.

7. Cabe por último há transportadora finalizar a Entrega para que assim esteja concluído o ciclo a entrega. Ficando depois disponível para ser avaliada por parte do utilizador.

Para além do que foi referido há bastantes outras opções funcionais no nosso programa que poderão ser experimentadas pelos docentes.

Conclusão

Após finalizar o projeto, o grupo ficou contente com o resultado obtido. Sentimos que apesar do conceito parecer um pouco confuso ao início, pois nunca tínhamos realizado um trabalho onde houvesse tantos tipos de utilizadores (Transportadora, Voluntario, Lojas, Utilizador) diferentes que pudessem intervir na aplicação, conseguimos orientar-nos e realizar um trabalho que vai de acordo com o pedido, pois tentamos cumprir todos os requisitos do enunciado do Projeto e incluir, tal como pedido, fatores de aleatoriedade que distinguissem pela positiva o Trabalho.

Apesar disso, temos noção que, a nível de código, há algumas partes que poderíamos ter melhorado. Por exemplo, quando tratamos de Pedidos, há alguma informação repetida, que não simboliza uma boa prática de Programação. Tal não foi implementado pois quando pensamos nas estruturas destes pensamos neles separadamente e não em conjunto, acabando por reparar no final que podíamos ter feito uma melhor gestão de código. Tentamos respeitar ao máximo a Composição, fazendo clones sempre que possível para haver encapsulamento de dados. Aplicamos enums, para tornar em alguns aspetos o nosso código mais percetível, e exceções de modo a que um utilizador não observasse o programa “morrer” no meio de uma execução. Também criamos uma classe que lê do standard input e garante que os dados lidos vão de acordo com o qual vão ser usados, impedindo, por exemplo, que o utilizador escreva letras numa opção que requeria números levando assim à “morte” do programa.

Em geral, aprendemos bastante sobre os padrões de escrita em Java e de como a língua funciona, e tivemos bastante prazer ao fazer o trabalho.

Faltou apenas referir ao longo do relatório que para poder aceder às contas dos utilizadores dados no ficheiro logs, basta inserir o código da mesma na zona do email e utilizar 1234 como password. Para aceder ao sistema em geral, que permite ver o top 10 utilizadores ou transportadoras, bem como outras estatísticas, basta apenas escrever pedroVeloso como email e de novo 1234 como password.

Diagramas de Classe MVC

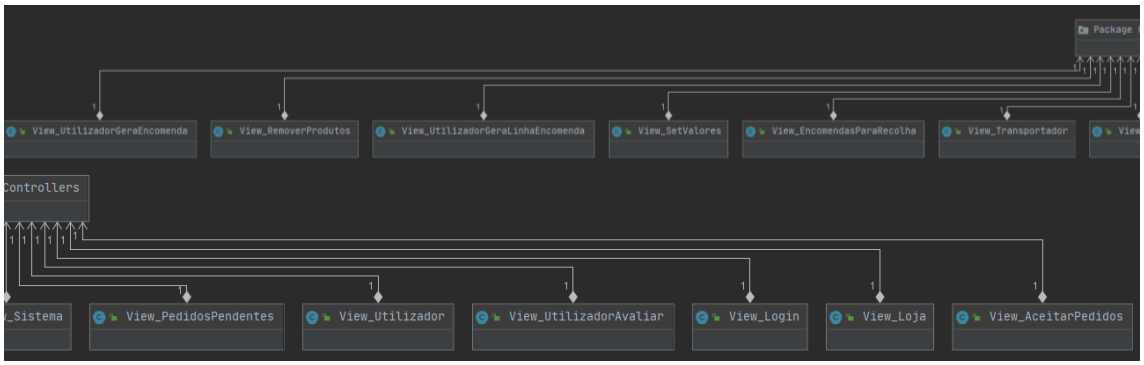


Figura 1: Diagrama de Classe da View

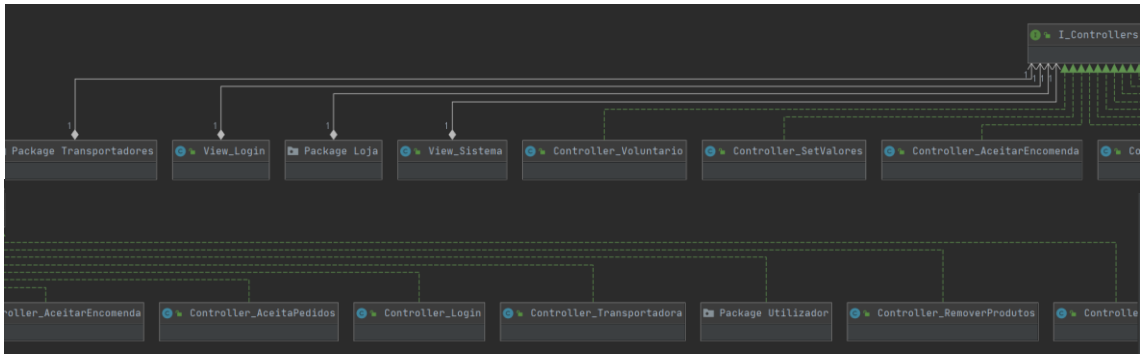


Figura 2: Diagrama de Classe do Controller

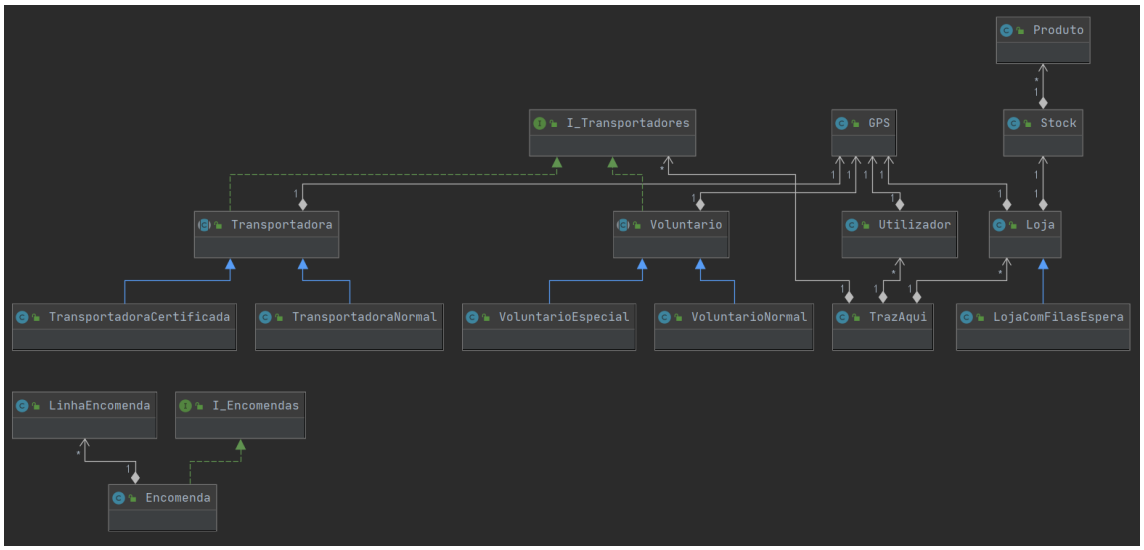


Figura 3: Diagrama de Classe Models