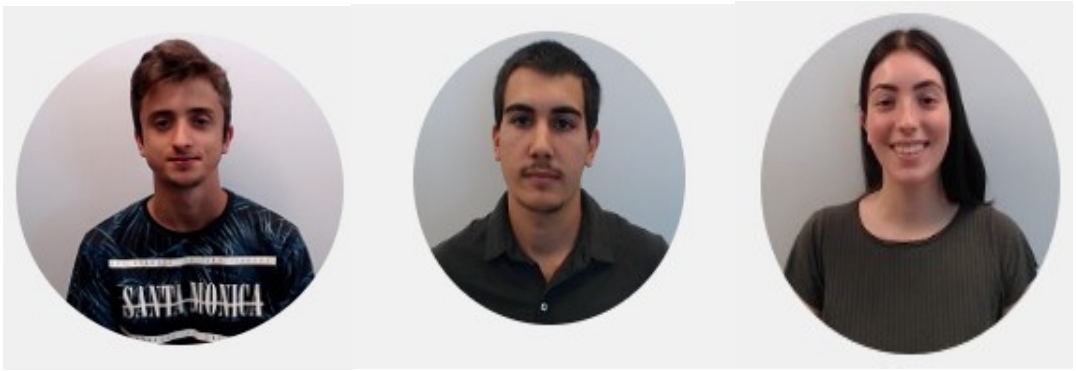
Universidade do Minho
Escola de Engenharia

Mestrado Integrado em Engenharia Informática
2º ano

Programação Orientada aos Objetos
Ano Letivo de 2019/2020



TrazAqui!

Rita Gomes (a87960)

José Rafael Reis (a87980)

José Pedro Manso (a87961)

Junho, 2020

Índice

Introdução

Este relatório serve de apoio ao projeto *TrazAqui!*, desenvolvido por Rita Gomes (a87960), José Rafael Reis (a87980) e José Pedro Manso (a87961).

No âmbito da disciplina de Programação Orientada aos Objetos, foi proposto aos alunos que desenvolvessem uma aplicação, em linguagem Java, que permitisse conjugar as atuais necessidades de entrega de encomendas a pessoas que estão confinadas nas suas habitações. Para tal, começamos por usar o IDE BlueJ e, com o avançar do projeto, decidimos mudar para o IntelliJ.

Como principal objetivo, pretendemos criar uma interação eficiente entre o utilizador e as restantes entidades envolvidas no processo de entrega de uma encomenda. Para isso, o grupo desenvolveu um software devidamente estruturado em várias classes.

Aqui iremos expor uma boa parte do nosso projeto, evidenciando quais as estratégias usadas de modo a permitir o bom funcionamento da aplicação. Iremos também enumerar as dificuldades que sentimos no decorrer do trabalho.

Segue-se então o desenvolvimento, onde iremos expor as nossas decisões.

Estruturação do projeto

Começamos por fazer um esboço (diagrama de classes) com o objetivo de facilitar a visualização do projeto, simplificando assim o desenvolvimento das classes e respetivas variáveis de instância. A partir daqui, verificamos quais as prioridades do projeto, definindo quais as classes gerais e quais as classes hereditárias.

Desta forma, definimos, primeiramente, as seguintes classes: Voluntário, Empresa, Loja, Utilizador, Encomenda e Linha de Encomenda. Reparamos de imediato que as quatro primeiras estavam relacionadas através de várias variáveis de instância. Sendo assim, criamos uma classe abstrata 'Entidade' à qual estas quatro classes já referidas vão herdar e onde teremos como variáveis de instância o id, password, nome, localização e uma lista de encomendas. Esta lista de encomendas difere de entidade para entidade. Vejamos:

- No caso dos utilizadores, corresponde às encomendas que ele solicitou.
- No caso das lojas, diz respeito à lista de encomendas prontas a serem entregues.
- No caso dos voluntários e empresas, representa as encomendas que o voluntario ou empresa em questão entregou.

Como forma de tratamento da localização, optamos por criar uma classe 'GPS' com o propósito de ter todos os métodos que servem para calcular as distâncias entre dois pontos de entidades distintas e obter as devidas localizações de cada entidade conjugados numa só classe, facilitando assim os cálculos.

Com o decorrer da realização do projeto, chegamos à conclusão que uma empresa poderia ser uma extensão de voluntário, sendo que acresce como variável de instância a taxa de transporte e o nif. Posto isto, a classe 'Voluntário' tem o objetivo de manipular os dados referentes a qualquer trabalhador, seja ele um voluntário ou funcionário de uma empresa. Sendo assim, temos agora três classes a herdar de 'Entidade': Loja, Utilizador e Voluntário.

A classe 'Loja' foi criada com o intuito de vender produtos aos utilizadores da aplicação, podendo ainda fornecer dados relativos ao tamanho da fila de espera e ao tempo médio de atendimento.

Por fim, o 'Utilizador' tem a capacidade de criar uma encomenda através da escolha de produtos de uma determinada loja, à sua escolha. Com esta informação, a aplicação terá de escolher o primeiro voluntário que aprove a intenção de efetuar a entrega desta encomenda, sendo que o sistema apenas notifica os voluntários que se encontrem disponíveis e cuja encomenda em questão

se encontre dentro do seu raio de ação. Caso isto não se suceda, a encomenda ficará ao cargo das empresas. Por sua vez, a classe ‘Empresa’, herdada da ‘Voluntário’, tem a peculiaridade de escolher, das encomendas que não foram aceites por um voluntário, quais é que pretende aceitar. Após a seleção das entregas que pretendem efetuar, as empresas propõem um orçamento aos utilizadores que efetuaram o pedido da respetiva encomenda. Posteriormente, os utilizadores em questão podem confirmar ou rejeitar a taxa sugerida.

Convém ainda destacar que uma ‘Encomenda’ é um conjunto de ‘LinhaDeEncomenda’, sendo que uma linha de encomenda contém a informação necessária para encomendar um produto, ou seja, peso, nome, preço unitário, quantidade, etc. Já na ‘Encomenda’, para além dos vários produtos que possam estar numa encomenda, também estarão as informações relativas ao processamento da encomenda, isto é, o id da loja, id da transportadora, id do utilizador, etc.

Finalmente, decidimos implementar a arquitetura Model-View-Controller lecionada nas aulas teóricas da UC, visto que se trata de um modelo que facilita a interação entre o utilizador e a camada computacional. Com isso, passámos à criação de um Model, de nome *TrazAqui*, ou seja, uma classe responsável por juntar os dados das diferentes entidades existentes em uma só. Depois, para cada entidade, foram criados um Controller e uma View, sendo que o Controller tem como função disponibilizar aquilo que a View necessita (podendo recorrer a métodos do Model) e, por sua vez, esta imprime as repostas obtidas. Finalmente, para fazer a gestão das contas, ou seja, eliminar ou introduzir novas entidades na aplicação, recorreu-se à criação de um Controller e de uma StartView, que têm também a finalidade de encaminhar para as restantes View’s e Controller’s, anteriormente mencionadas.

Discussão

Uma das primeiras decisões tomadas foi recorrer ao uso de herança, porque se tornou evidente que todas as entidades tinham certos aspetos em comum. Assim, era impensável criar várias classes com variáveis e métodos de instância iguais, sendo que havia a possibilidade de armazenar tudo numa única classe, para que assim haja um maior reaproveitamento de código.

Posteriormente, optamos por adotar a arquitetura por composição, visto que contribuímos para um programa mais seguro no que toca à partilha de endereços. Desta forma, sempre que haja a necessidade de solicitar um determinado objeto, temos a garantia de que não recebemos o objeto original.

Dado ser um dos pilares fundamentais da Programação Orientada a Objetos, decidimos também aplicar o conceito de abstração através, por exemplo, do uso de Collection ao invés de List. No entanto, este conceito acabou por não ser muito utilizado, uma vez que optámos por diferenciar as distintas entidades através de diferentes Map's na classe *TrazAqui* (Model). De certa forma, ao utilizar os vários Map's considerámos que os registos das entidades ficaram armazenados de uma forma mais estruturada. Contudo, caso optássemos por usar apenas um Map como variável de instância, tirávamos bastante mais partido da abstração.

A justificação para o uso de Map's reside no facto de procurarmos sucessivas vezes pela "key" dos mesmos, correspondente ao id da entidade em questão. Desta forma, é possível aumentar a eficiência da procura de uma entidade, ainda que o tempo de inserção da mesma seja um pouco mais elevado do que o de um ArrayList, por exemplo. No entanto, não será de todo uma desvantagem, já que o número de entidades nunca chega a ser suficientemente grande para justificar o uso de outro tipo de estrutura de dados.

A separação de View's e Controller's deve-se particularmente ao facto de tornar o código mais leve e organizado, para que os métodos usados sejam devidamente diferenciados e de fácil compreensão.

Já no final da realização do projeto, achamos que seria uma boa ideia introduzir o envio de notificações às diversas entidades que usam a nossa aplicação. Deste modo, o cliente irá receber um aviso quando:

- a sua encomenda for entregue;
- receber um orçamento por parte da empresa, para posteriormente o aceitar ou rejeitar, e neste mesmo aviso informamos ainda a estimativa do tempo de entrega;

- houver um voluntário que aceitou fazer a entrega da sua encomenda, informando qual o id do mesmo e qual o tempo previsto para a entrega.

Após a loja informar que existe uma encomenda pronta a ser entregue, todos os voluntários que se encontram no raio de ação dessa mesma loja irão receber um aviso a perguntar se aceitam fazer a entrega desta. A empresa vai receber um aviso a informar da resposta do utilizador face ao orçamento proposto.

Por último, de modo a minimizarmos a ocorrência de erros na nossa aplicação, introduzimos ainda três exceções:

1. EntidadeNaoExisteException

   Nesta situação, escolhemos uma loja que não estava na lista de lojas apresentadas.

```
Lojas disponiveis:
    ID: l13 -> LA Kids & Junior
    ID: l29 -> TV e Telecomunicações
    ID: l58 -> Primark
    ID: l8 -> Punt Roma
    ID: l83 -> ColchaoNet.com
Escolha uma loja (escreva o id da loja pretendida)
l10
Ups! Loja Inválida
```

2. ListaVaziaException

   Neste caso específico tentamos ver o histórico de um novo utilizador.

```
O que pretende fazer?
    1 -> Criar uma nova Encomenda.
    2 -> Ver o histórico de Encomendas
    3 -> Ver os 10 clientes que mais usaram esta APP
    4 -> Classificar Voluntario/Empresa
    5 -> Definiçoes
    6 -> Apagar a conta
    0 -> Logout
2
Historico de Encomendas:
Ups! Histórico de encomendas vazio!
```

3. IdRepetidoException

   Esta exceção serve para quando queremos adicionar uma nova entidade com um id já existente, por exemplo. Este caso é bastante improvável, visto que os id's atribuídos às novas entidades são calculados tendo em conta a quantidade de entidades já existentes.

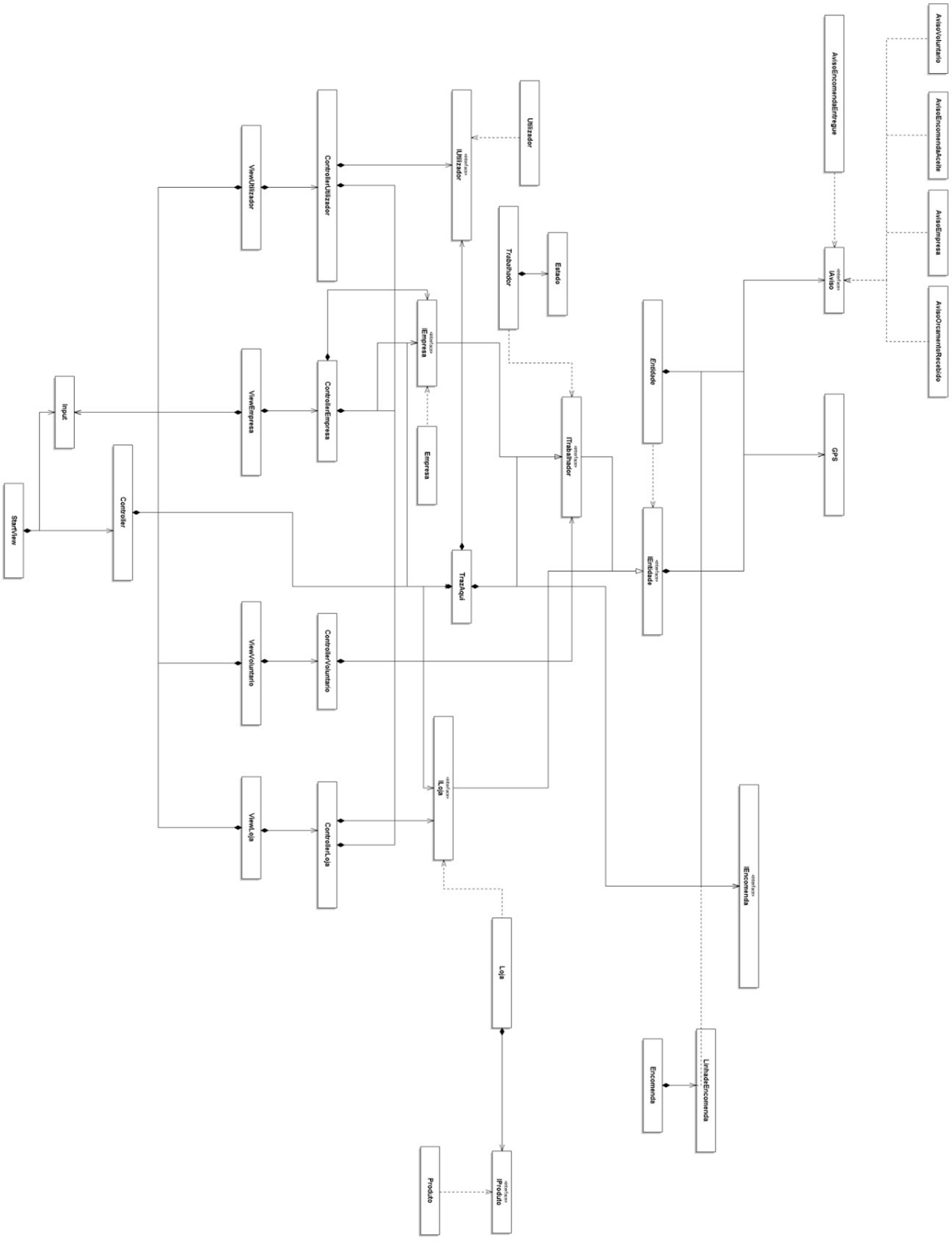
Quanto à gravação do ficheiro objeto, este processa-se de forma automática, isto é, sempre que há uma atualização de dados, seja introdução ou alteração dos mesmos, o ficheiro guarda sucessivamente estas alterações.

Conclusão

Após a resolução de todo o projeto, o grupo concluiu que este foi deveras desafiante e pôs em prática todos os conhecimentos adquiridos nesta UC, desde o uso do encapsulamento à introdução de conceitos como streams e interfaces. Com a realização deste trabalho desenvolvemos a nossa capacidade de programação em objetos e aprendemos, de forma construtiva, a consolidar os seus principais conceitos.

Tínhamos ainda algumas ideias que pretendíamos implementar, tais como a introdução de um veículo para o voluntário, sendo que ele podia escolher qual o tipo de veículo que pretendia usar para efetuar a entrega. E, para além disso, tínhamos ainda em mente ajustar a taxa sugerida pela empresa conforme as condições climatéricas, isto é, por exemplo, se estivesse um dia chuvoso, a taxa seria um pouco mais alta.

Todos os elementos do grupo se empenharam e deram o seu melhor no decorrer do trabalho ajudando sempre com os conhecimentos que possuíam. O grupo concluiu que o projeto final está positivo, visto que cumpre todos os requisitos básicos pedidos no enunciado e está funcional. Verificamos ainda que o tempo disponível para a resolução do projeto foi o ideal.



Nota: Caso a imagem seja impercetível, enviamos em anexo a imagem original e o ficheiro que foi feito no programa Nclass.