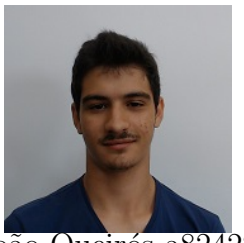
Programação Orientada aos Objetos (2º ano de Mestrado Integrado em Engenharia Informática)

Trabalho Prático

Grupo 31 - Relatório


João Queirós a82422


Pedro Gomes a82422


José Santos a84288

23 de Julho de 2020

Conteúdo

Capítulo 1

Especificação do Enunciado

O programa, "UMCarroJa", desenvolvido no âmbito da Unidade Curricular de Programação Orientada aos Objetos, deve permitir que proprietários de veículos registem os mesmos na aplicação, para estes poderem sem posteriormente alugados por clientes. Após efetuado um pedido de alugar por um cliente, este necessita ainda da confirmação por parte do proprietário, sendo que este pode, ou não, aceitar o pedido.

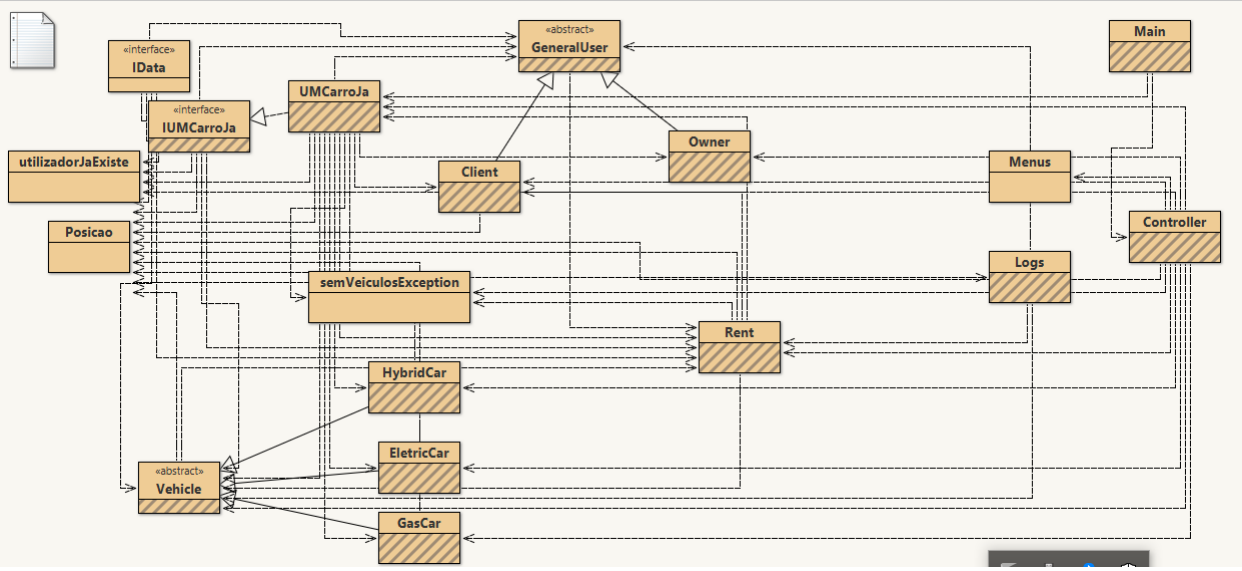
O programa deve manter registos dos alugueres realizados e da informação relativa aos mesmos, para que ambos, clientes e proprietários, tenham informação á sua disposição aquando da realização do aluguer e da sua confirmação.

Sendo que os veículos são particulares, não se encontram agregados em localizações específicas, mas sim estacionados na rua, estando as suas coordenadas GPS disponíveis para os clientes poderem selecionar o veículo que querem alugar.

Capítulo 2

Classes

Para além dos atributos necessários para o funcionamento de cada classe, as classes cujas informações são guardadas entre instâncias do programa possuem ainda um atributo *serialVerionUID*.



2.1 Main

Classe que inicializa o funcionamento normal do programa.

2.2 Menus

Classe que contém variáveis e métodos para simplificar a apresentação de menus, assim como aumentar a claridade do código e a sua modularidade.

- `List<String> menuOptions`
- int choice

2.3 UMCarroJa

Classe que guarda a maioria da informação do programa. Decidimos utilizar como chave o nif dos utilizadores, no entanto, como as credenciais de login são o email e password, decidimos definir um map que associa os email aos nifs dos utilizadores. A classe possui ainda o map com todos os users, veículos e alugueres com ratings por atribuir.

- `Map<String, String> emailToNif`
- `Map<String, GeneralUser> users`
- `Map<String, Vehicle> allVehicles`
- GeneralUser loggedInUser
- `Map<String, List<Rent>> pendingRating`
- Logs log
- boolean backupDataRead

2.4 IUMCarroJa

Interface que pussui os métodos necessários para implementar a classe *UMCarroJa*. Como esta (UMCarroJa) se trata de uma classe bastante extensa, a implementação desta interface facilitou bastante a sua compreensão de modo a permitir uma melhor colaboração entre os elementos do grupo.

2.5 Controller

Classe central em termos de gerência do programa.

- Menus menuPrincipal
- Menus client
- Menus owner
- Menus aluguer
- Menus registar
- boolean running
- UMCarroJa mUMCarroJa
- Scanner sn

2.6 GeneralUser

Classe Abstrata que contém variáveis e métodos comuns aos vários tipos de utilizadores do programa.

- String NIF
- String Nome
- String Email
- String Password
- String Morada
- LocalDate Data de Nascimento
- double Rating
- `List<Rent> Alugueres`

2.7 Client

O cliente, para além do que herda da classe GeneralUser, possui também informação relativa á sua posição atual.

Este pode alugar carros, consulta o histórico de alugueres, visualizar o preço da sua última viagem, dar rating aos seus alugueres realizados assim como definir a sua posição.

- Posicao pos

2.8 Owner

Tal como com o Cliente, o Proprietário possui as variáveis e métodos do GeneralUser, no entanto este possui ainda a lista dos carros que este registou para serem alugados. O proprietário

- `List<String> listCar`

2.9 Vehicle

Assim como o GeneralUser, esta trata-se de um classe abstrata que contém variáveis e métodos comuns aos vários veículos existentes.

- int averageSpeed
- double pricePerKm
- double consumptionPerKm
- double rating
- Posicao pos

- String matricula
- String marca
- String nifOwner
- boolean needFuel
- `List<Rent> alugueres`

2.10 HybridCar, EletricCar, GasCar

Classes que implementam as várias nuances de cada tipo de veículo presente no programa não abrangidas na classe *Vehicle*, nomeadamente o modo de funcionamento e abastecimento do combustível.

2.11 Posicao

Representa a posição de um Cliente ou Veículo num espaço 2d (eixos X e Y), e contém os métodos necessários para a sua utilização, nomeadamente o método *distancia*, que recebe outra posição e calcula a distância entre as duas.

- double x
- double y

2.12 Rent

Classe que representa um Aluguer realizado. Possui a informação associada ao mesmo, nomeadamente:

- LocalDateTime date
- Duration rentTime
- double price
- Posicao price
- String nif
- String matricula

Possuí ainda os métodos relativos aos vários filtros de seleção de carro, i.e. O carro mais próximo, o mais barato, etc.

2.13 Logs

Classe responsável por escrever para ficheiro os Logs respetivos ás várias ações do programa, tais como o registo de novos utilizadores/veículos e a realização de alugueres.

- PrintWriter printWriter
- String fileName

2.14 semVeiculosException

Exceção resultante de tentar realizar ações sobre veículos quando estes não existem.

2.15 utilizadorJaExiste

Exceção resultante de tentar inserir um utilizador que já existe.

Capítulo 3

Requisitos e Funcionalidades

3.1 Estado Pré-Guardado

3.2 Registo de Clientes/Proprietários

Quando é selecionada a opção de registo, o utilizador pode escolher entre fazer o registo como cliente ou proprietário, sendo depois requisitadas as informações necessárias para o registo, e após uma breve validação das mesmas, é criada a entrada em UMCarroJa com a informação fornecida.

3.3 Validar o acesso á aplicação (login)

Para realizar o login, o utilizador insere o seu email e password, e é verificado se o seu email existe nos dados do programa. Caso exista, é verificado se a password providenciada corresponde é válida, e, caso seja, o *loggedInUser* da instância de *UMCarroJa* existente no controller passa a ser o utilizador a quem correspondem os dados.

3.4 Criar e Inserir Viaturas

Se o utilizador estiver logged in como proprietário, pode selecionar a opção para registar um veículo. Após ser selecionada, o utilizador seleciona qual o tipo de veículo a inserir e introduz as suas propriedades. O veículo é associado ao utilizador e adicionado á lista de carros do mesmo. Caso a inserção seja bem sucedida, o veículo é adicionado ao Map *allVehicles* existente na instância de *UMCarroJa* do controller.

3.5 Solicitar uma viagem

A solicitação de uma viagem por parte de um cliente pode ser feita segundo os seguintes critérios:

- Selecionar o carro mais próximo
- Selecionar o carro mais barato
- Selecionar um carro em particular
- Selecionar um carro com base na sua autonomia

Para fazer a seleção do carro, é chamado o método da classe *Rent* correspondente á opção selecionada, sendo que é passado como parâmetro a lista que contém todos os carros do sistema. Esta é posteriormente percorrida e é devolvido o carro que melhor se adequa á opção selecionada. No caso de o cliente selecionar a opção de escolher um carro em particular, são-lhe apresentados todos os carros válidos para escolher. Decidimos ainda que não faria sentido ser possível selecionar um carro cuja distância entre ele e o cliente fosse maior do que a distância entre o cliente e a sua posição destino.

3.6 Classificar a viatura após a viagem

Estando o cliente logged in no programa, este possui a opção de atribuir a classificação á viatura. Para tal temos um Map que contém os alugueres sem classificação de todo o programa, e, após ser atribuída a classificação por parte do utilizador, o aluguer é removido do Map.

3.7 Ter acesso, no perfil de cliente, à listagem dos alugueres efectuados (entre datas)

3.8 Ter acesso, no perfil de proprietário, à listagem das alugueres efectuados (entre datas)

3.9 Indicar o total facturado por uma viatura num determinado período

3.10 Gravar o estado da aplicação em ficheiro, para que seja possível retomar mais tarde a execução

Todas as interfaces

Capítulo 4

Escalabilidade - Adição de novos tipos de veículos ao programa

Para adicionar novos tipos de veículos basta criar uma nova class que dê extend á classe vehicle, sendo necessário que esta nova classe defina os métodos e variáveis necessários para o abastecimento do veículo.

Capítulo 5

Conclusões

No geral, estamos satisfeitos com o trabalho, apesar de este poder ser melhorado. Uma das opções que poderia ter sido implementada, e que ajudaria em termos de adições futuras seria a implementação de uma interface que todos os tipos de veículos teriam de implementar, de modo a servir de standard para futuras adições. Para além disso o relatório do mesmo poderia estar mais completo, com descrições mais detalhadas e mais aprofundadas, nomeadamente no que toca ás decisões por parte do grupo.