



Universidade do Minho  
Departamento de Informática  
Mestrado Integrado em Engenharia Informática

Programação Orienta a Objetos  
2018/2019

# UM CARRO JÁ!

Grupo 36



Gonçalo Pinto  
83732



Luís Lopes  
85367



Rui Chaves  
83693

## Índice:

1. <u>Capítulo1</u> .....	página 3
1.1 - Introdução	
1.2 - Objetivos Gerais	
2. <u>Capítulo2</u> .....	página 4
2.1 - Classe Coordenadas	
2.2 - Classe Aluguer	
2.3 - Classe Veiculos	
2.3.1 – Classe CarroGasolina	
2.3.2 – Classe CarroHibrido	
2.3.3 – Classe CarroElectrico	
2.4 - Classe Cliente	
2.5 - Classe Proprietario	
2.6 - Classe RedeProprietarios	
2.7 - Classe RedeClientes	
2.8 - Classe Acoes	
2.9 - Classes Menu	
2.10 – Classe AcoesCliente	
2.11 – Classe MenuCliente	
2.12 – Classe AcoesProprietario	
2.13 – Classe MenuProprietario	
2.14 – Classe SISTEMA	
3. <u>Capítulo3</u> .....	página 10
3.1 Funcionamento da aplicação	
4. <u>Capítulo4</u> .....	página 14
5. <u>Capítulo5</u> .....	página 14

# **Capítulo 1**

## **1.1 Introdução**

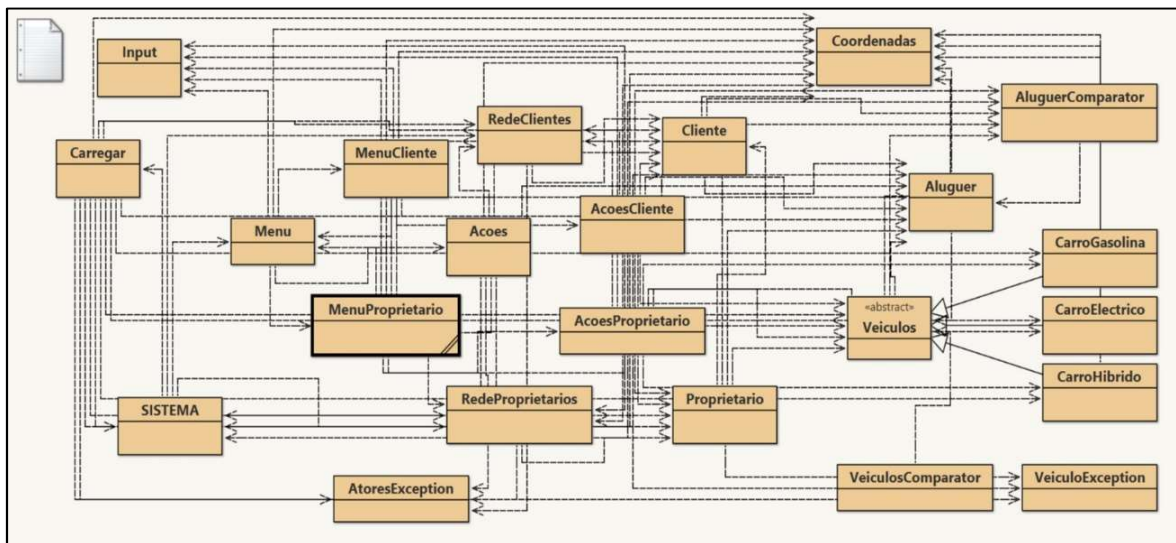
No 2º ano do Mestrado Integrado em Engenharia Informática, no ano letivo 2018/2019, a unidade curricular Programação Orientada aos Objetos apresenta como proposta, na componente prática, a realização de um programa que crie um serviço de aluguer de veículos particulares. Neste programa um proprietário de um automóvel pode registar o seu veículo de momento pode ser a gasolina, elétrico e híbrido aqui e posteriormente pode ser alugado por um cliente inscrito. Este aluguer é efetuado mediante a escolha do cliente onde um veículo registado vai ser lhe atribuído e de seguida é efetuado um pedido de aluguer ao proprietário que detém este veículo podendo este aceitar ou rejeitar o pedido.

Este programa além do processo de aluguer e de aceitação ou não de pedidos. Este programa também abrange a criação de clientes e proprietários. O cliente pode alterar os dados pessoais, pedir um veículo mediante diversas opções, classificar apenas uma vez a viagem e ver viagens que efetuou entre datas. O proprietário por sua vez pode alterar os seus dados pessoais, registar carros, ver os carros que tem registado, sinalizar veículos como disponíveis e indisponíveis, abastecer veículos, alterar o preço por km de um dado veículo, ver o total faturado por um dado veículo entre datas, ver a sua lista de pedidos e automaticamente é gerado um aluguer, ver viagens de um dado veículo entre datas e ver os 10 clientes que utilizam mais esta aplicação.

## **1.2 Objetivos Gerais**

Este projeto tem como objetivo principal consolidar e colocar em prática os conceitos fundamentais transmitidos durante a UC de Programação Orientada aos Objetos. Com o presente trabalho pretendesse desenvolver uma aplicação que respeite as normas da programação orientada aos objetos, nomeadamente o encapsulamento, a abstração de implementação e capacidade de a aplicação evoluir de forma controlada, como por exemplo a integração de novos tipos de veículos.

## Capítulo 2



### 2.1 Classe Coordenadas

Esta classe tem como variáveis de instância dois *doubles* que representam as coordenadas no espaço 2D de um dado objeto. Esta classe além da declaração dos construtores por omissão, parametrizado e de cópia, dos métodos de acesso e alteração do seu estado interno e do método *clone*, *toString* e *equals*, possui os métodos de mover um ponto para uma dada coordenada e calcular a distância entre duas coordenadas.

### 2.2 Classe Aluguer

Esta classe é constituída por uma *String* que representa o nome do cliente que efectuou o aluguer, pelo veículo utilizado, pela data do aluguer, pelas coordenadas iniciais da viagem e pelas coordenadas finais. Esta classe possui ainda como variáveis a distância, o preço, o tempo da viagem. Detém uma variável *boolean* utilizada para indicar se o cliente já classificou essa viagem como também permite identificar se o proprietário já registou o preço e o tempo de um dado aluguer. Estão presentes também os construtores por omissão, parametrizado e de cópia, os métodos de acesso e alteração do seu estado interno e o método *clone*, *toString* e *equals*.

### 2.3 Classe Veiculos

Esta classe é uma classe especial porque é abstrata, ou seja, serve como “modelo” para outras classes que pretendem herdar desta classe. Uma das

outras características que esta classe possui é o facto de não poder ser instanciada por si só, isto é, as classes que vão herdar desta classe tem que a instanciar desta forma é possível criar vários tipos de veículos. Para termos um objeto desta classe é necessário que uma classe mais especializada a herda e então instanciar essa nova classe. Como uma classe abstrata pode ter variáveis decidimos implementar desta forma e não como interface. Assim sendo as variáveis desta classe são uma descrição do veículo, a matrícula que permite identificar os diferentes veículos, o estado do veículo como *boolean*, a velocidade média, o preço por km, total faturado, a classificação atribuída pelos clientes, as coordenadas onde se encontra e um conjunto de Alugueres que representa o histórico de viagens de um dado veículo este conjunto é ordenado e testado pelo *AluguerComparator*. Para além da declaração dos construtores por omissão, parametrizado e de cópia, dos métodos de acesso e alteração do seu estado interno e método *hashCode*, *toString* e *equals*. Possui os métodos que permitem adicionar e remover um aluguer, adicionar um valor a sua faturação, alterar a sua classificação e abastecer o veículo. Como esta classe é abstrata implementa métodos que não possuem implementação os designados métodos abstratos cujo implementação tem de ser necessariamente feita na classe que a herda. Estes métodos são de abastecimento de veículos e acesso/alteração de um limite como um consumo por parte de um veículo.

### 2.3.1 – Classe CarroGasolina

Esta classe é uma especialização da classe Veículos assim sendo herda todos os métodos da classe e variáveis. Esta classe tem duas variáveis consumo e autonomia. Possui a declaração dos construtores por omissão, parametrizado e de cópia, dos métodos de acesso e alteração do seu estado interno e método *hashCode*, *toString* e *equals*. Como é uma extensão da classe Veículos tem de implementar os métodos de abastecimento e acesso e alteração de um limite como um consumo por parte de um veículo como acima foi descrito.

### 2.3.2 – Classe CarroHibrido

Esta classe é uma outra especialização da classe Veículos assim sendo herda todos os métodos da classe e variáveis. Esta classe tem duas variáveis capacidade e consumo. Possui a declaração dos construtores por omissão, parametrizado e de cópia, dos métodos de acesso e alteração do seu estado

interno e método *hashCode*, *toString* e *equals*. Como é uma extensão da classe Veículos tem de implementar os métodos de abastecimento e acesso e alteração de um limite como um consumo por parte de um veículo como acima foi descrito.

### 2.3.3 – Classe CarroElectrico

Esta classe é uma especialização da classe Veículos assim sendo herda todos os métodos da classe e variáveis. Esta classe tem duas variáveis consumo e autonomia. Possui a declaração dos construtores por omissão, parametrizado e de cópia, dos métodos de acesso e alteração do seu estado interno e método *hashCode*, *toString* e *equals*. Como é uma extensão da classe Veículos tem de implementar os métodos de abastecimento e acesso e alteração de um limite como um consumo por parte de um veículo como acima foi descrito.

## **2.4 Classe Cliente**

A classe Cliente apresenta como variáveis de instância o nome, o email que é o seu identificador, a password e a morada. Possui ainda a sua data de nascimento, as coordenadas onde se encontra, as coordenadas para onde pretende ir e possui ainda um conjunto mais propriamente um *TreeSet* de *Aluguer* que representa o seu histórico de viagens este conjunto é ordenado e testado pelo *AluguerComparator*. Estão presentes também os construtores por omissão, parametrizado e de cópia, os métodos de acesso e alteração do seu estado interno, o método *clone*, *hashCode*, *toString* e *equals*. Além destes métodos possui um outro para adicionar um aluguer ao seu histórico.

## **2.5 Classe Proprietario**

A classe Proprietário possui como variáveis o email que é o identificador, nome, password, morada declaradas como *Strings*, a data de nascimento declarada como *LocalDate*, a classificação dada pelos clientes a este proprietário, uma associação de *String* (objeto chave que é a matrícula de um Veiculo) a um dado Veiculo (objeto valor) que representa os veículos associados a um proprietário, uma associação de *String* a um dado Cliente que representa a lista de pedidos onde a chave é a matrícula do veículo requisitado e o valor o cliente que efetuou o pedido. Por fim detém ainda como variável um conjunto de alugueres que representa o seu histórico este conjunto é ordenado e testado pelo *AluguerComparator*. Para além da declaração dos construtores por omissão, parametrizado e de cópia, dos métodos de acesso e alteração do seu

estado interno e do método *clone*, *toString*, *hashCode* e *equals*, possui os métodos para adicionar e remover veículos à primeira associação referida acima caso não consiga lança a exceção *VeiculoException*, como também de obter um dado Veiculo, trocar um veiculo por um outro, ver se um dado veiculo existe e quantos veículos este proprietário tem. Em relação a segunda associação encontramos métodos de adicionar, remover, obter e saber na totalidade o número de pedidos existentes. Em relação ao conjunto temos métodos para adicionar e remover um dado Aluguer. Para finalizar temos um método que atualiza a classificação do proprietário.

## 2.6 Classe RedeProprietários

Esta classe como variável de instância um *Map* representa uma associação de Strings (objeto chave) a um dado Proprietário (objeto valor). Como cada email do proprietário é único decidimos utilizá-lo como chave para aceder ao cliente em causa. Esta classe além da declaração dos construtores por omissão, parametrizado e de cópia, dos métodos de acesso e alteração do seu estado interno e do método *clone*, *toString* e *equals*, possui os métodos para adicionar um proprietário à associação caso o email exista é lançada uma exceção *AtoresException* que é uma extensão de *Exception* que indica que esse email já está a ser utilizado como chave. Por outro lado, também possui o método para remover um proprietário caso este não existe lança a mesma exceção indicando que o email do cliente que se pretende eliminar não existe. Nesta classe podemos também encontrar um método denominado login que recebe um email e uma palavra-passe onde vai procurar o email na associação que recebeu caso não exista lança novamente a exceção *AtoresException* caso exista compara a palavra que recebeu com o valor que o email recebido mapeia, valor este sendo um Proprietário efetuando o acesso devido á variável password como acima descrito. Possui o método de substituir um dado Proprietário. Detém o método de dado um determinado email retorna o Proprietário a ele associado. Possui um método que retorna se um dado email existe ou não. Para finalizar tem ainda mais dois métodos que escrevem e guardam em ficheiro de objetos esta classe.

## 2.7 Classe RedeClientes

Esta classe como variável de instância um *Map* representa uma associação de *Strings* (objeto chave) a um dado *Cliente* (objeto valor). Como cada email do cliente é único decidimos utilizá-lo como chave para aceder ao cliente em causa. Esta classe além da declaração dos construtores por omissão, parametrizado e de cópia, dos métodos de acesso e alteração do seu estado interno e do método *clone*, *toString* e *equals*, possui os métodos para adicionar um cliente à variável caso o email exista é lançada uma exceção *AtoresException* que é uma extensão da classe *Exception* que indica que esse email já está a ser utilizado como chave. Por outro lado, também possui o método para remover um cliente caso este não existe lança a mesma exceção indicando o email do cliente que se pretende eliminar não existe. Nesta classe podemos também encontrar um método denominado login que recebe um email e uma palavra-passe onde vai procurar o email que recebeu caso não exista lança novamente a exceção *AtoresException* caso exista compara a palavra que recebeu com o valor que o email recebido mapeia, valor este sendo um *Cliente* efetua-se o acesso devido á variável password como acima descrito. Possui o método de substituir um dado *Cliente*. Detém o método de dado um determinado email retorna o *Cliente* a ele associado. Possui um método que retorna se um dado email existe ou não. Para finalizar tem ainda mais dois métodos que escrevem e guardam em ficheiro de objetos.

## 2.8 Classes Acoes

Esta classe tem duas variáveis *static* sendo uma do tipo *RedeClientes* e outra *RedeProprietarios*. Esta classe tem os métodos de fazer login de *Cliente*, login de *Proprietário*, criação de um cliente como também a criação de um *proprietário*. Possui ainda o método de gravar os dados.

## 2.9 Classe Menu

Esta classe não possui nenhuma variável de instância tendo apenas definido métodos *static*, métodos que permitem ser referidos através da classe e que fazem a apresentação do menu inicial, de login ou de cliente ou de *proprietário* como também a criação dos mesmos. Assim, apresenta as diversas opções ao utilizador, interage com este e posteriormente em cada método cria



uma instância de classe *Acoes* onde executa o método correspondente enviando tudo o que utilizador inseriu, independente do método executado, retornam sempre uma resposta de sucesso ou insucesso. Se o utilizador tiver feito um qualquer login e posteriormente o método respetivo tiver dado sucesso. Esta classe cria uma instância do menu respetivo e invoca o menu inicial do mesmo.

## **2.10 Classe AcoesCliente**

Esta classe tem duas variáveis *static* sendo uma do tipo *RedeClientes* e outra *RedeProprietarios*. Possui os métodos de editar password, editar o nome, editar morada, editar a data de nascimento, editar o local, ver o perfil, efetuar um pedido mais próximo, mais barato, mais barato dentro de uma distância, um determinado veículo, com uma determinada autonomia. Possui os métodos de classificar uma viagem e ver viagens dentro de um espaço de tempo.

## **2.11 Classe MenuCliente**

Esta classe também não possui variáveis tal como a classe *Menu* esta classe apenas faz a interação com o cliente neste caso contudo os seus métodos não são *static*. Apresenta as diversas opções implementadas e interage com este pedindo os valores mediante a escolha de opção. Ao fim desta interação quase todos os métodos desta classe criam uma instância de *AcoesCliente* para aceder aos métodos correspondentes enviando os valores recebidos e recebendo a resposta por parte dos métodos respetivos em *AcoesCliente*.

## **2.12 Classe AcoesProprietario**

Esta classe tem duas variáveis *static* sendo uma do tipo *RedeClientes* e outra *RedeProprietarios*. Possui os métodos de editar password, editar o nome, editar morada, editar a data de nascimento, ver perfil, criar tipos de veículos, devolver a lista de veículos de um dado proprietário, apagar um veiculo, marcar como indisponível ou disponível um dado Veiculo, abastecer cada tipo existente, alterar o preço por km de um dado veiculo, ver a faturação entre duas datas de um determinado veiculo, devolver a lista de pedidos, devolver a informação de quanto tempo demorara um dado cliente como o veiculo que pretende. Aceitar ou rejeitar pedidos. Ver viagens entre datas. E por fim devolver a lista geral dos 10 clientes que mais utilizam a aplicação.

## 2.13 Classe MenuProprietario

Esta classe Menu é diferente das outras anteriores pois esta possui variáveis de instância, variáveis estas que representam a quantidade de veículos e pedidos que um utilizador tem decidimos criar estas variáveis de forma a evitar a interação com o utilizador mais rápida assim se um dado proprietário não tiver veículos esta classe escusa de fazer a interação e invocar o método respetivo de AcoesProprietario. Estas variáveis tem os respetivos métodos de acesso e alteração. De resto esta classe tem o mesmo comportamento que as outras classes menus, apresentar as diversas opções e interagir com o utilizador. Após a interação cria instâncias de AcoesProprietario e invoca os métodos respetivos de forma a dar a satisfazer o pedido do utilizador. Posteriormente é apresentado ao utilizador os dados obtidos.

## 2.14 Classe SISTEMA

Esta classe possui duas variáveis de instância static que permite referenciar através da classe do tipo RedeClientes e do tipo RedeProprietarios. Estas classes tem os respetivos métodos de acesso e alteração. Esta classe é onde se encontra a main do programa.

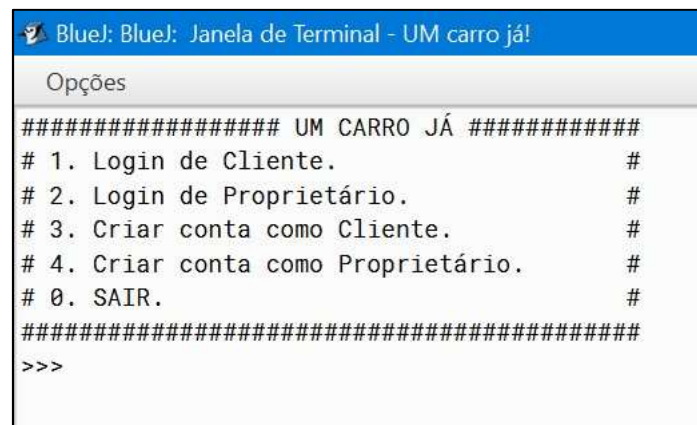
# Capítulo 3

## 3.1 Funcionamento

Como foi referido a main do programa encontra-se na classe SISTEMA aqui se for a primeira utilização o programa vai ler o ficheiro de *logs* fornecido pelos docentes. No ficheiro de logs de forma a facilitar a apresentação os alugueres todos os alugueres registados foram efetuados por apenas 3 tipos de veículos. Assim existe um proprietário cuja o email é *prop@admin.pt* e palavra passe *prop* que tem estes veículos e que tem a informação de todos os alugueres. Contudo o cliente que efetuou o pedido encontra o aluguer no seu histórico indicando que o veículo utilizado foi criado para o efeito.

Caso contrário é lido os ficheiros objetos estadoC.obj e estadoP.obj respetivamente vão inicializar as variáveis desta classe com estes dois ficheiros o primeiro vai carregar a variável do tipo RedeClientes e o segundo carregar a variável do tipo RedeProprietarios. Todas estas leituras e carregamentos encontram-se previamente cauteladas quanto as exceções possíveis.

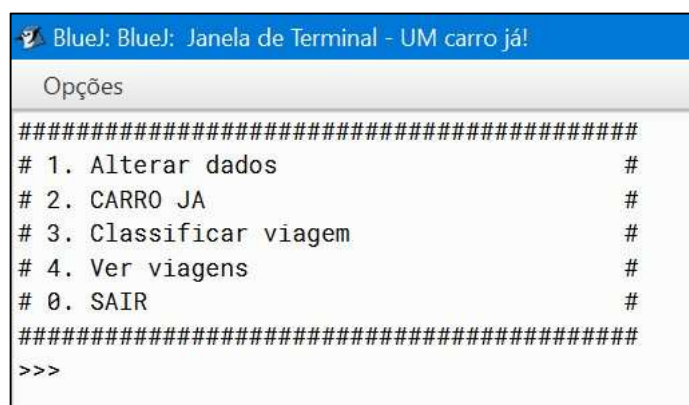
Posteriormente chama através da classe porque é static o método menuInicial onde apresenta as diversas opções.



```
BlueJ: BlueJ: Janela de Terminal - UM carro já!
Opções
##### UM CARRO JÁ #####
# 1. Login de Cliente. #
# 2. Login de Proprietário. #
# 3. Criar conta como Cliente. #
# 4. Criar conta como Proprietário. #
# 0. SAIR. #
#####
>>>
```

Por exemplo, se o utilizador pretende fazer um login como cliente. O menu inicial chama o método da mesma classe loginC onde aqui é pedido ao utilizador que insira o seu email e a sua password. Após receber estas duas variáveis este método cria uma instância de Acoes onde executa o método loginCliente onde envia como parâmetros aquilo que recebeu. Este método cabe inicializar a sua variável de instância necessária para responder ao pedido, neste caso vai a sua variável do tipo RedeClientes inicializar os valores desta com os mesmos da classe SISTEMA utilizando o método de acesso devido pois esta foi carregada anteriormente pelos ficheiros de objetos. Executa o método que permite efetuar um login na classe RedeClientes posteriormente é retornado um valor. Valor este que é interpretado e retorna uma palavra (*String*). O menu recebe esta palavra caso a palavra neste caso seja de sucesso invoca um método menu da classe MenuCliente enviando como parâmetro o email do utilizador. Desta forma sabemos sempre qual é o utilizador que está a utilizar o programa.

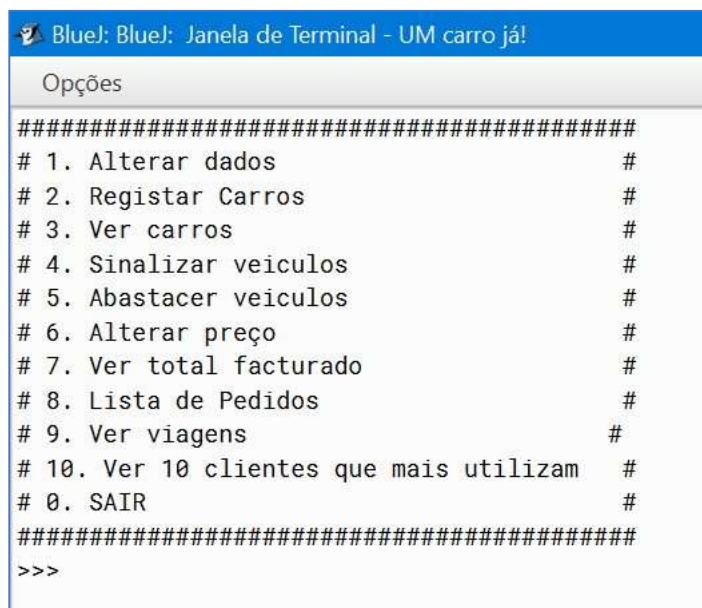
De seguida é apresentado o menu do Cliente ao utilizador.



```
BlueJ: BlueJ: Janela de Terminal - UM carro já!
Opções
#####
# 1. Alterar dados #
# 2. CARRO JA #
# 3. Classificar viagem #
# 4. Ver viagens #
# 0. SAIR #
#####
>>>
```

Posteriormente o utilizador pode escolher qualquer uma das opções acima apresentadas. Em cada uma destas é chamado o menu de interação respetivo que recebe sempre o email do utilizador de momento como parâmetro depois é efetuado a interação. Após isso tal como foi abordado anteriormente o método em causa cria uma instância de AcoesCliente enviando o email do utilizador sempre e os parâmetros recebidos. É satisfeito o pedido do utilizador retornando sempre uma mensagem para o menu apresentar contudo ao ver viagens retorna uma lista com as viagens pretendidas cabendo ao menu apresentar ao utilizador. O cliente apenas pode classificar uma viagem após o proprietário aceitar e registar o pedido da mesma, viagem esta que o cliente tem de saber exatamente todas as características do aluguer entre elas matrícula, coordenadas iniciais e finais pois o teste de existência do aluguer é feito através da comparação destes parâmetros. Ficou em falta da nossa parte implementar uma forma de impedir o cliente não efetuar mais pedidos enquanto não forem aceites pelo proprietário os anteriores.

Por outro lado, quanto ao menu do proprietário é acedido da mesma forma após um login de proprietário é chamado as Acoes. Estas acedem as variáveis de SISTEMA e retornam o resultado. Caso o resultado seja válido a tentativa de login, é instanciado um MenuProprietario onde tem como argumentos o número de veículos e número de pedidos. Posteriormente invoca-se o método para apresentar o menu recebendo como parâmetro o email do proprietário. De seguida apresenta-se o menu de Proprietário:



```
Blue: Blue: Janela de Terminal - UM carro já!
Opções
#####
# 1. Alterar dados                                     #
# 2. Registar Carros                                  #
# 3. Ver carros                                       #
# 4. Sinalizar veiculos                              #
# 5. Abastacer veiculos                              #
# 6. Alterar preço                                    #
# 7. Ver total facturado                              #
# 8. Lista de Pedidos                                #
# 9. Ver viagens                                     #
# 10. Ver 10 clientes que mais utilizam              #
# 0. SAIR                                             #
#####
>>>
```

Tal como foi referenciado anteriormente a classe MenuProprietario invoca os métodos dentro da classe de forma a satisfazer ao pedido do utilizador em cada uma delas é criado uma instância de AcoesProprietario onde utiliza-se os métodos designados. Todos estes métodos devolvem uma mensagem ou uma coleção por vezes que cabe ao menu que o invocou apresentar o utilizador.

De forma geral descrevemos de seguida o ato de alugar, o cliente encontra-se no seu menu, escolhe pedir um carro tem diversas opções a sua escolha entre elas o mais próximo da sua localização, o mais barato, o mais barato dentro de uma distância, um carro específico e um com uma autonomia desejada. Posteriormente é perguntado a preferência do tipo de veículo, as coordenadas finais e mais parâmetros dependentes da sua escolha. É executado o método de AcoesCliente respetivo retornando existência ou não de um veículo com essas propriedades, mensagem que indica o tempo e o preço base, caso encontre previamente é feito o teste se o veículo tem autonomia suficiente. É inserido também anteriormente na lista de pedidos do proprietário. O proprietário ao executar no seu menu o menu de ver pedidos aparece o email do cliente que pretende alugar o veículo indicando a matrícula deste. Juntamente também aparece quanto tempo demora o cliente a chegar. O proprietário pode aceitar ou rejeitar se aceitar é registado o aluguer sendo inserido no seu histórico, do veículo em causa e do cliente. Futuramente o cliente passa a poder classificar esta viagem apenas uma vez.

De uma forma geral o programa funciona como anteriormente foi referido. A parte de interação e apresentação cabe as classes Menus e a criação ou alteração de dados cabe as Acoes. Onde a classe SISTEMA é importante pois tem sempre as redes tanto de proprietários como de clientes atualizada. Permitindo cada método da classe Acoes inicializar as suas variáveis com essas redes. Desta forma pretendemos respeitar o modelo MVC utilizado na programação orientada aos objetos.

## **Capítulo 4**

De forma a adicionar outros tipos de veículos ao sistema bastava apenas criar a classe respetiva que seria uma especificação de veículos, isto é, herdaria todos as variáveis e métodos de veículos. A partir daí era criar as variáveis que fossem ao encontro do tipo de veículo em causa e claro os métodos que fossem necessários. Contudo é obrigatório definir método de abastecimento, de um limite de km para esse novo veículo como também os métodos de consumo. Caso o veículo em causa não tenha consumo como por exemplo um skate basta definir os métodos e criar um variável consumo sendo zero e o programa funcionaria com esse novo tipo de Veículos.

## **Capítulo 5**

O presente relatório descreveu, de forma sucinta, as principais componentes da aplicação desenvolvida para a avaliação prática da Unidade Curricular de POO.

Considera-se que os principais objetivos foram cumpridos com a exceção de um pequeno problema nas coordenadas que não encontramos explicação para tal que é a coordenada x desaparece dando lugar a y e no lugar de y fica 0.

Este projeto ajudou na consolidação dos conhecimentos da língua JAVA, bem como na metodologia de programação com objetos.