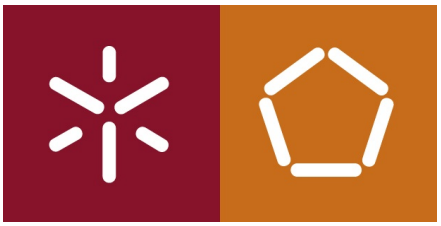


Universidade do Minho

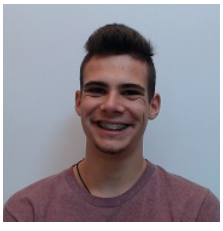
Mestrado Integrado em Engenharia Informática

Programação Orientada aos Objetos

Grupo 60

Ana Rita Rosendo
A84475

Gonçalo Esteves
A85731



Rui Oliveira
A83610

Conteúdo

1 Introdução

No âmbito da Unidade Curricular de Programação Orientada aos Objetos, foi-nos proposta a elaboração de um projeto em linguagem de programação Java, cuja finalidade seria a criação de uma aplicação, denominada UMCarroJá!, que simule um serviço de aluguer de veículos particulares utilizando a internet. Na aplicação referida, um proprietário poderá registar o seu veículo e o mesmo poderá ser alugado por um cliente também registado. Tal como nos serviços Uber e Airbnb, após o cliente efetuar o pedido de aluguer, o proprietário terá de aceitar (ou não) o pedido .Para tal, é necessária a criação de um histórico de feeback. Esse aluguer apresenta um determinado custo para o cliente consoante o seu período de tempo. Após o perído de tempo do aluguer chegar ao fim, é dada a opção ao cliente de classificar o veículo e ao proprietário de classificar o cliente.

2 Descrição do Problema

Esta aplicação tem como principal objetivo o suporte de todas as funcionalidades que permitam um utilizador alugar e realizar uma viagem num dos veículos da UMCarroJá!. Sendo assim, o programa deverá cumprir os seguintes requisitos básicos:

- O estado da aplicação deverá estar pré-carregado com um conjunto de dados significativos, que permita testar toda a aplicação;
- Registar um utilizador, quer cliente quer proprietário;
- Validar o acesso à aplicação utilizando as credenciais (email e password), por parte dos clientes e dos proprietários;
- Criar e inserir viaturas;
- Solicitar, por parte de um cliente, uma viagem de ponto P(x,y) para o ponto R(x,y), escolhendo uma viatura ou então solicitando a viatura mais próxima. Caso a viatura seja das que possuem lista de espera, inserir na lista de espera;
- Classificar a viatura, após a viagem;
- Ter acesso, no perfil de cliente, à listagem dos alugueres efetuados (entre datas);
- Ter acesso, no perfil de proprietário, à listagem dos alugueres efetuados (entre datas);
- Indicar o total faturado por uma viatura num determinado período;
- Determinar as listagens dos 10 clientes que mais utilizam o sistema (em número de vezes e em kms percorridos);
- Gravar o estado da aplicação em ficheiro, para que seja possível retomar mais tarde a execução.

Para além destes foram também sugeridos outros aspetos como o cálculo do tempo da viagem, logo o seu custo, depender não só da distância entre os locais mas também das condições atmosféricas, dos condicionantes do trânsito e da destreza do condutor, e a implementação de outros tipos de veículos como, por exemplo, bicicletas, trotinetes e pranchas de surf elétricas.

2.1 Atores do Sistema

O ecossistema da UMCarroJá! contempla 2 tipos de atores de sistema, os clientes e os proprietários, em que ambos possuem informação relativa somente assim e partilham a seguinte informação:

- email (que identifica o utilizador);
- nome;
- password;
- morada;
- data de nascimento;

Cada um possui um papel diferente na aplicação. Ou seja, enquanto que os clientes apenas podederão solicitar um aluguer, os proprietários apenas poderão sinalizar que um dos seus carros está disponível para aluguer, abastecer o veículo, alterar o preço por km, aceitar/rejeitar o aluguer de um determinado cliente e registar quanto custou a viagem.

Existem diferentes tipos de aluguer. Estes são: o aluguer do carro mais próximo, o aluguer do carro mais barato, o aluguer do carro mais barato dentro de uma distância que o cliente está disposto a percorrer a pé, o aluguer de um carro específico e o aluguer de um carro com uma autonomia desejada.

2.2 Tipos de Veículos

Para além disso, o ecossistema da UMCarroJá! contempla diferentes tipos de veículos que são: os carros a gasolina, os carros elétricos, os carros a gasóleo e os carros híbridos. Cada um dos três tipos de viaturas tem associada:

- Uma velocidade média por km;
- Uma autonomia;
- Uma localização;
- Um preço base por km;
- Um consumo de gasolina/bateria por km percorrido;
- Acesso a um histórico de alugueres realizados;
- Classificação do carro, dado numa escala de 0 a 100, calculada com base na classificação dada pelos clientes no final do aluguer;

3 Resolução do Problema

Inicialmente, foram definidas classes necessárias para o funcionamento do projeto. Após essa tarefa, desenvolvemos a aplicação criando os menus e definindo as funções essenciais para a sua execução.

3.1 Classes Definidas

3.1.1 Dados

```
1 public class Dados implements Serializable{
2     private Map<String, Cliente> clientes; //email, cliente
3     private Map<String, Proprietario> proprietarios; /*email,
      proprietario*/
4     private Map<String, Veiculo> veiculos; //matricula, veiculo
5 }
```

A superclasse **Dados** foi implementada com o intuito de armazenar a informação necessária para o bom funcionamento da aplicação. Esta contém várias variáveis de instância que representam os clientes, os proprietários e os veículos. Para representar os clientes foi usada uma coleção Map que possui todos os clientes e a sua chave é o email do cliente. Para representar os proprietários foi usada uma coleção Map que possui todos os proprietários e a sua chave é o email do proprietário. E para representar os veículos foi usada uma coleção Map que possui todos os veículos e a sua chave é a matrícula do veículo.

Para comunicar devidamente com a **Aplicacao** sobre o conteúdo da estrutura, alguns métodos existentes nesta classe lançam as seguintes excecões:

- EmailInvalidoException - utilizada caso o email introduzida não cumpra os requisitos de um email válido;
- EmailInexistenteException - utilizada caso o email introduzido não conste na estrutura;
- NomeInvalidoException - utilizada caso o nome introduzido não conste na estrutura;
- PasswordInvalidaException - utilizada caso a password de confirmação não coincida com a inserida durante o processo de registo de um proprietário/cliente;
- VeiculoInexistenteException - utilizada caso o veículo não exista.

3.1.1.1 Proprietário

```
public class Proprietario extends Dados{
    private String email;
    private String nome;
    private String password;
    private String morada;
    private LocalDate ddn;
    private int nif;
    private double classificacao;
    private List<Aluguer> alugueres;
    private List<Veiculo> veiculos;
}
```

A subclasse **Proprietário** contém a informação necessária para a representação de um proprietário, uma lista relativa aos alugueres realizados e uma lista relativa aos veículos que o mesmo possui. Além disso, contém uma variável de instância que representa a classsificação do mesmo, dada numa escala de 0 a 100, calculada com base na classificação dada pelos clientes que alugaram os seus carros.

3.1.1.2 Cliente

```
public class Cliente extends Dados{
    private String email;
    private String nome;
    private String password;
    private String morada;
    private LocalDate ddn;
    private int nif;
    private double localX;
    private double localY;
    private List <Aluguer> alugueres;
}
```

A subclasse **Cliente** contém a informação necessária para a representação de um cliente, ou seja, a pessoa que solicita e efetua o aluguer de um carro. Além do mais, contém uma lista relativa aos alugueres efetuados, uma variável de instância que representa a classsificação do mesmo em relação à viatura e duas variáveis de instância que representam a sua localização.

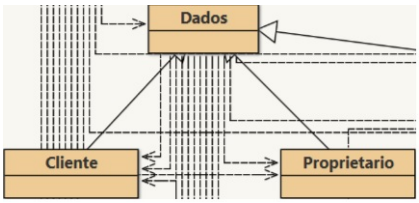


Figura 1: Esquema da superclasse **Dados** e as suas subclasses **Proprietário** e **Cliente**

Para além dos clientes e dos proprietários, a aplicação UMCarroJá! contempla diferentes tipos de viaturas de aluguer com certas características. Sendo assim, foi necessária a criação de classes para os diferentes tipos.

3.1.2 Veículo

```
1  public abstract class Veiculo extends Dados{
2      private String matricula;
3      private double velocidade;
4      private double preco;
5      private List<Aluguer> alugueres;
6      private double classificacao;
7      private double totalFaturado;
8      private double localX;
9      private double localY;
10     private boolean disponivel;
11 }
```

A superclasse **Veículo** foi implementada com o intuito de conter a informação comum às subclasses correspondentes aos diferentes tipos de veículos, o que permitiu evitar a repetição das mesmas variáveis nas subclasses referidas. Esta classe foi definida como abstrata uma vez que não deverão ser criados veículos do tipo **Veículo** mas sim dos tipos **Carro Híbrido**, **Carro Gasóleo**, **Carro Gasolina** e **Carro Elétrico**.

3.1.2.1 Carro Híbrido

```
public class CarroHibrido extends Veiculo{
    private double consumoBat;
    private double bateria;
    private double bateriaMax;
    private double consumoComb;
    private double combustivel;
    private double combustivelMax;
}
```

A subclasse **Carro Híbrido** foi criada de modo a adicionar a informação que apenas diz respeito a este tipo de veículo sendo ela o consumo da bateria, a quantidade de bateria que o veículo tem num determinado momento e a quantidade máxima do mesmo, o consumo quando é utilizado combustível como fonte de alimentação, a quantidade de combustível que o veículo tem num determinado momento e a respetiva quantidade máxima.

3.1.2.2 Carro Gasóleo

```
public class CarroGasoleo extends Veiculo{
    private double consumo;
    private double combustivel;
    private double combustivelMax;
}
```

Assim como a subclasse **Carro Híbrido**, a subclasse **Carro Gasóleo** foi criada de modo a adicionar a informação que apenas diz respeito a este tipo de veículo sendo esta o seu consumo, a quantidade de combustível que o mesmo tem num determinado momento e a sua quantidade de combustível máxima.

3.1.2.3 Carro Gasolina

```
public class CarroGasolina extends Veiculo{
    private double consumo;
    private double combustivel;
    private double combustivelMax;
}
```

A subclasse **Carro Gasolina** foi criada de modo a adicionar a informação que apenas diz respeito a este tipo de veículo sendo esta o seu consumo, a quantidade de combustível que o mesmo tem num determinado momento e a sua quantidade de combustível máxima.

3.1.2.4 Carro Elétrico

```
1  public class CarroEletrico extends Veiculo{
2      private double consumo;
3      private double bateria;
4      private double bateriaMax;
5  }
```

Assim como a subclasse **Carro Gasolina**, a subclasse **Carro Elétrico** foi criada de modo a adicionar a informação que apenas diz respeito a este tipo de veículo sendo esta o seu consumo, a quantidade de bateria que o mesmo tem num determinado momento e a quantidade de bateria máxima.

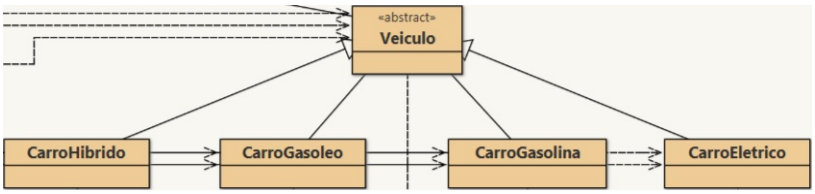


Figura 2: Esquema da superclasse **Veículo** e as suas subclasses **Carro Híbrido**, **Carro Gasóleo**, **Carro Gasolina** e **Carro Elétrico**

3.1.3 AluguerCarro

```
1  public class AluguerCarro implements Aluguer{
2      private String emailCli;
3      private String emailPro;
4      private String matricula;
5      private double km;
6      private double custo;
7      private double localX;
8      private double localY;
9      private double pendente; /*estado do aluguer, se está pendente, já
       foi efetuado...*/
10 }
```

A classe **AluguerCarro** foi implementada com o intuito de conter a informação de um aluguer de um veículo. Esta contém várias variáveis de instância que representam o email do cliente que vai realizar o aluguer, o email do proprietário do veículo em questão, a matrícula do veículo, a distância da viagem e o custo da mesma, a localização do destino final e o estado do aluguer.

Como a classe **AluguerCarro** contém a informação de um aluguer esta necessita de ser partilhada por outras classes, no entanto, tal não é possível pois, em Java, uma classe não pode ser partilhada. Sendo assim, foi necessária a criação de uma interface.

```
1  public interface Aluguer{
2      public String getEmailCli();
3      public void setEmailCli(String em);
4      public String getEmailPro();
5      public void setEmailPro(String em);
6      public String getMatricula();
7      public void setMatricula(String m);
8      public double getKm();
9      public void setKm(double k);
10     public double getCusto();
11     public void setCusto(double c);
12     public Aluguer clone();
13     public boolean equals(Object o);
14     public String toString();
15     public int hashCode();
16     public int compareTo(AluguerCarro c);
17 }
```

3.1.4 Menu

```
1  public class Menu{
2      // variáveis de instância
3      private List<String> opcoes;
4  }
```

A classe **Menu** representa o menu da aplicação em modo texto. Este é um factor necessário para permitir a comunicação entre a aplicação e o utilizador.

3.1.5 Aplicação

```
1  public class Aplicacao {
2      // Menus da aplicação
3      private Menu menuInicial;
4      private Menu menuRegisto;
5      private Menu menuCliente;
6      private Menu menuProprietario;
7      private Menu menuViagem;
8      private Menu menuCarro;
9      private Menu menuPosViagem;
10
11     private Dados d = new Dados();
12 }
```

A classe **Aplicacao** contém os principais métodos. Nesta classe, os menus são carregados, os logins e registos submetidos, as viagens realizadas e as exceções lançadas pelos métodos definidos em **Dados** são geridas, uma vez que esta lida diretamente com o input submetido pelo utilizador. Está é, portanto, responsável pelo funcionamento da aplicação UMCarroJá!.

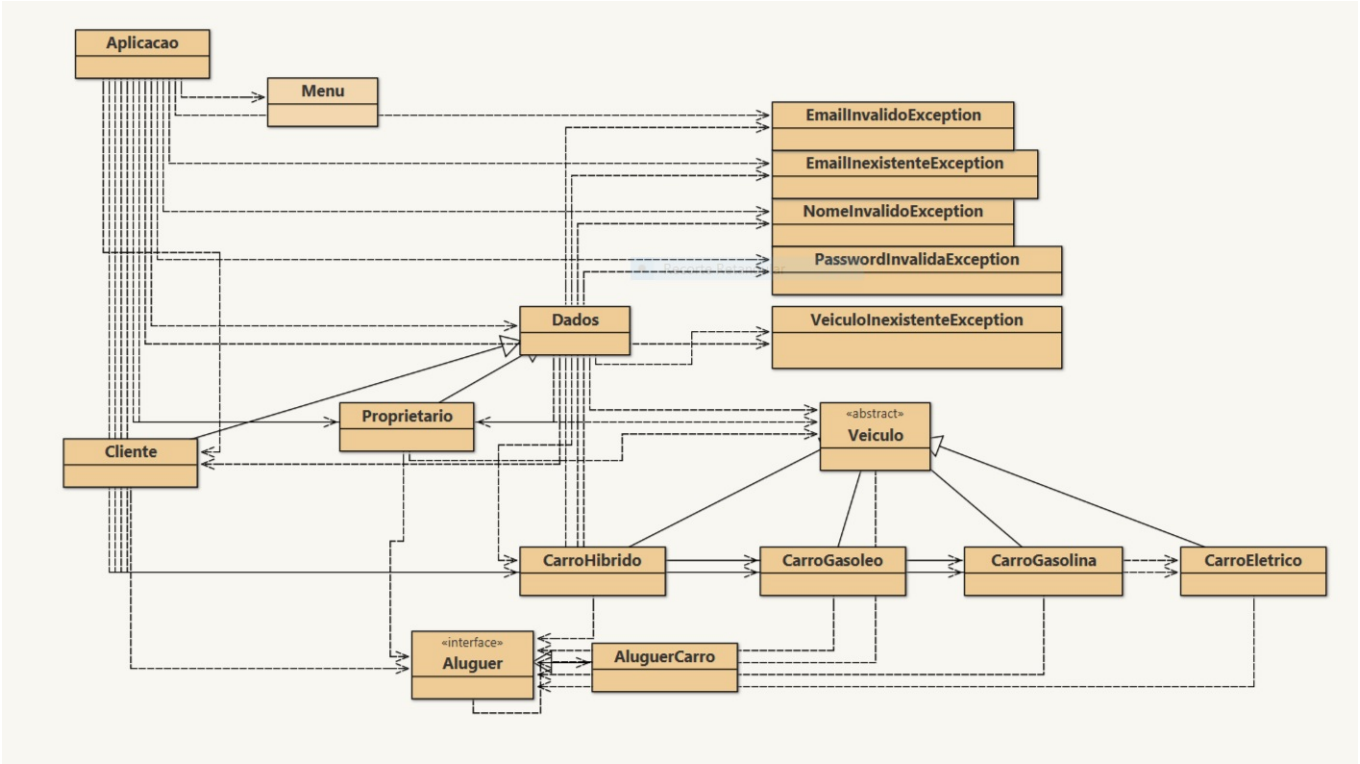


Figura 3: Esquema geral do projeto

4 Considerações

4.1 Manual de utilização da aplicação

Nesta secção, iremos dar uma breve explicação de como utilizar esta aplicaçãoo, de modo a facilitar o uso da mesma.

4.1.1 Login e Registo

Quando a aplicação é iniciada é dada a opção a opção ao utilizador de fazer login caso o mesmo já tenha uma conta ou de se registar caso tal não se verifique.

Figura 4: Menu Principal

No login, o utilizador deve inserir o email e a password correspondentes à sua conta. Após a confirmação dos dados inseridos, é apresentado um novo menu consoante o estatuto do ultizador (cliente ou proprietário).

- **Login do Cliente**

Depois de inserir os dados da sua conta, o utilizador vai ter acesso a um conjunto de opções.

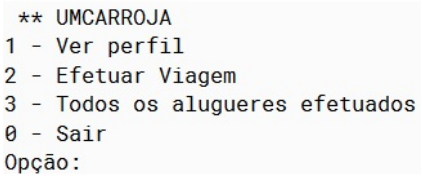


Figura 5: Menu do Cliente

1 - Imprime no ecrã o perfil do cliente
2 - Faz o pedido da viagem
3 - Imprime no ecrã todos os alugueres que o cliente realizou
0 - Sair do menu

- **Login do Proprietário**

Depois de inserir os dados da sua conta, o utilizador vai ter acesso a um conjunto de opções.

```
 ** UMCARROJA
1 - Ver Perfil
2 - Novo veiculo
3 - Abastecer veiculo
4 - Alterar o preço por km
5 - Lista dos 10 clientes que mais utilizam o sistema(em numero de vezes)
6 - Lista dos 10 clientes que mais utilizam o sistema(em kms)
7 - Alterar Disponibilidade de um veiculo
8 - Todos os alugueres efetuados
9 - Total faturado por um veiculo
0 - Sair
Opção:
```

Figura 6: Menu do Proprietário

1 - Imprime no ecrã o perfil do proprietário
2 - Insere novo veículo
3 - Abastece um veículo
4 - Altera o preço por km
5 - Imprime no ecrã a lista dos 10 clientes que mais utilizam o sistema, sendo o número de vezes que já usaram o sistema o critério de seleção)
6 - Devolve a lista dos 10 clientes que mais utilizam o sistema, sendo o número de kms que já procuraram usando a aplicação o critério de seleção
7 - Altera a disponibilidade de um veículo
8 - Imprime no ecrã todos os alugueres efetuados
9 - Imprime no ecrã o total faturado de um veículo

Por outro lado, no registo, aparece a opção de registar como cliente ou proprietário e, dentro dessas opções, existe um conjunto de informação a ser preenchida pelo utilizador. Após o registo, o utilizador tem de fazer login.

```
 ** UMCARROJA
1 - Registar como cliente
2 - Registar como proprietario
0 - Sair
Opção:
```

Figura 7: Menu do registo

- **Registo do Cliente**

No registo de um cliente, o utilizador deve seguir as instruções que aparecem no ecrã e preencher todos os campos.

```
*** REGISTO
Email: cliente@email.com
Nome: Cliente
Password: cl
Confirmação da Password: cl
Morada: Rua Bairro Novo
Data de Nascimento
Exemplo: 05/06/1998
01/03/1995
NIF:
234456678
```

Figura 8: Registo de um cliente

- **Registo do Proprietário**

No registo de um proprietário, o utilizador deve seguir as instruções que aparecem no ecrã e preencher todos os campos.

```
*** REGISTO
Email: proprietario@email.com
Nome: Proprietario
Password: pt
Confirmação da Password: pt
Morada: Rua Santa Casa
Data de Nascimento
Exemplo: 05/06/1998
01/02/19990
NIF:
123345567
```

Figura 9: Registo de um proprietário

4.1.2 Efetuar Viagem

Tal como indica o nome, esta opção permite ao cliente solicitar uma viatura à sua escolha que o transporte para o local pretendido. Após a seleção desta opção no menu do cliente, é apresentado ao cliente um menu com os diferentes tipos de viaturas existentes. O cliente deve escolher aquela que satisfaz as suas condições.

```
 ** UMCARROJA
1 - Carro a gasolina
2 - Carro hibrido
3 - Carro eletrico
4 - Carro a gasoleo
0 - Sair
Opção:
```

Figura 10: Tipo de veículos

Após a seleção da opção correspondente ao tipo de carro pretendido, será apresentado um menu que permite ao cliente escolher entre vários tipos de alugueres.

```
 ** UMCARROJA
1 - Solicitar veiculo mais próximo
2 - Solicitar veiculo mais barato
3 - Solicitar carro mais barato dentro de uma determinada distância a percorrer a pé
4 - Solicitar um veiculo específico
5 - Solicitar veiculo com determinada autonomia
0 - Sair
Opção:
```

Figura 11: Tipos de alugueres

Posto isto, o cliente terá de preencher os campos com a informação pedida.

5 Conclusão

Após a elaboração deste projeto, podemos afirmar que aumentamos o nosso nível de conhecimento, não só no que toca ao domínio da linguagem de programação Java, mas também à compreensão de determinados conceitos, nomeadamente o encapsulamento de dados e a otimização de código. Com o desenvolvimento deste trabalho, fomo-nos deparando com diversos obstáculos que se colocaram à nossa frente, sendo necessário não só fazer uso de conhecimentos previamente adquiridos, mas também de outros novos, conseguidos através de uma procura nas fontes disponíveis.