# Projeto UMeR

A78416 - Francisco Oliveira A79617 - Raul Vilas Boas
A79175 - Vitor Peixoto

**Grupo 73**

Braga, 3 de Junho de 2017

**Programação Orientada aos Objetos**

**Mestrado Integrado em Engenharia Informática**

**Universidade do Minho**

# Conteúdo

# Capítulo 1

# Introdução

Foi-nos proposto criar um projeto que implementasse um sistema de gestão de táxis, direcionado a clientes e a motoristas, em *Java*.

O sistema deve ser capaz de ter clientes e motoristas com as suas respetivas contas e registar novos se assim for necessário, efetuar viagens com determinada viatura ou simplesmente a que se encontrar mais próxima, verificar dados estatísticos e guardar o estado da aplicação num ficheiro.

Imediatamente após a inicialização deste projeto, os principais problemas com que nos deparamos foram:

- Manter um código coerente, estruturado e de fácil reutilização, dado que algumas áreas do trabalho sejam um pouco mais difíceis de modularizar;
- Guardar o estado da aplicação num ficheiro binário e recuperá-lo quando inicarmos a aplicação.

Este relatório está orientado por três secções: os problemas detetados já falados nesta introdução, uma concepção da solução onde será explorada a nossa abordagem para as resoluções dos problemas detetados e uma breve conclusão sobre o resultado final.

# Capítulo 2

# Concepção da Solução

## 2.1 Criação das classes

Numa fase inicial do projeto, começou-se por criar as classes que iriam ser necessárias para o projeto. Dentro dessas classes foram inicializadas as variáveis de instância, os seus construtores, os `getters` e `setters`, `equals` e `clones`.

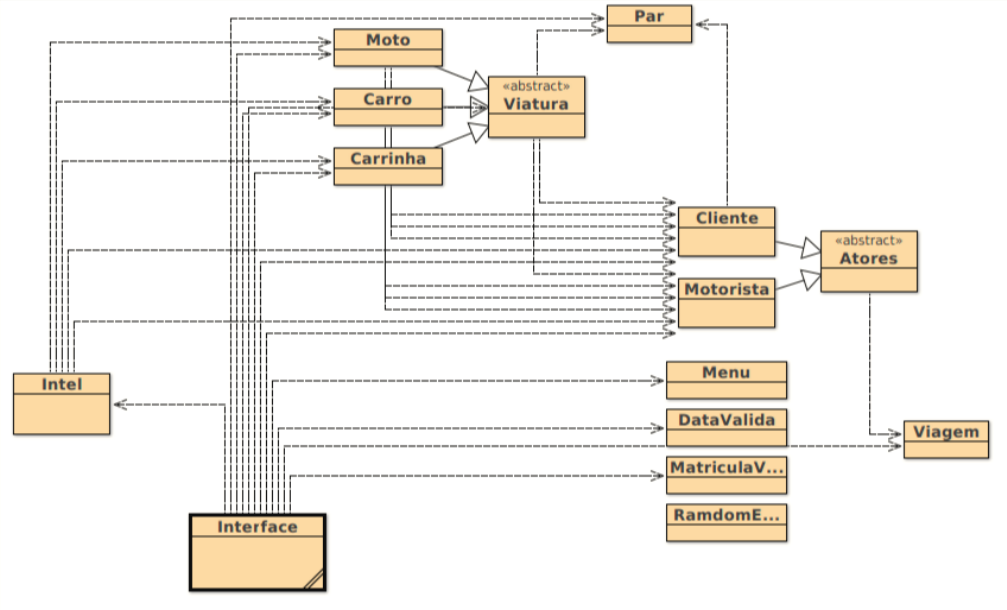


Figura 2.1: *Organização das classes vista no BlueJ.*

A classe abstrata `Atores` tem como subclasses a `Clientes` e a `Motoristas`. Esta classe serve de base para a criação de clientes e motoristas contendo toda a informação do utilizador e relatório de viagens. A sua subclasse `Cliente` tem as coordenadas e os seus gastos acumulados. A sua outra subclasse `Motorista` tem a disponibilidade deste, número de quilómetros percorridos e medidas de avaliação, tais como classificação e grau de condução.

Outra classe abstrata criada foi a `Viatura`. Esta classe serve de base para a criação de carros, motos e carrinhas, contendo toda a informação relativa ao veículo, motorista

que o ocupa, coordenadas e fiabilidade da viatura e o total faturado pela mesma. Contém também dois métodos que alteram a fiabilidade da viatura. Esta classe abstrata tem 3 subclasses, `Carrinha`, `Moto` e `Carro`.

Foi implementada também a classe `Par` que implementa um sistema de coordenadas e um cálculo de distancia entre coordenadas em linha reta.

A classe `Viagem` define a informação relativa às viagens efetuadas na rede *UMeR*.

A classe `Menu` é a classe que implementa uma interface de *input/output*, recebendo uma lista de *strings* de opções.

`DataValida` é a classe que verifica se o formato da data introduzida é válido (AAAA/MM/DD) e permite convertê-la para um inteiro (AAAAMMDD).

`MatriculaValida` é a classe que verifica se a matrícula introduzida tem um formato válido, idêntico ao sistema de matrículas português (AA-00-00 ou 00-00-AA ou 00-AA-00).

## 2.2 Classe Interface

A classe `Interface` é o ”coração”da aplicação. Esta classe tem como principal função executar os diversos menus, no entanto alberga também alguns dos métodos necessários para fazer viagens, adicionar clientes, motoristas, etc.

O Menu Principal permite efetuar o login e registo de um cliente ou motorista, adicionar uma viatura nova ou ver dados estatísticos, como os clientes que mais gastam ou os motoristas com mais desvios no preço.

Os métodos de login utilizam as listas na classe `Intel` que armazenam toda a informação e verificam se o utilizador inserido existe e se as palavras-passe coincidem e caso as credenciais sejam corretamente inseridas, acede ao menu de cliente/motorista.

Os métodos de registo de utilizador pedem para inserir os seus dados e cria uma nova instância de Cliente/Motorista que é adicionada à lista respetiva da classe `Intel`.

O método de adicionar viaturas pergunta qual o tipo de viatura que está a ser adicionada e faz exatamente o mesmo que os métodos de registo de utilizador, adicionado no fim a viatura à sua listam em `Intel`.

Para calcular os dados estatísticos dos clientes que mais gastam criamos uma *HashMap* para armazenar o nome do cliente *(Key)* e o total de gastos *(Value)*. Esta *HashMap* é depois ordenada pelos clientes com mais gastos e limitada aos 10 primeiros através de uma *Stream*. O mesmo foi feito para calcular os motoristas com mais desvios no preço estimado e final.

Depois temos o menu de cliente. Este menu permite ver a informação do cliente, incluindo a sua posição no momento e os gastos efetuados até agora. Permite também ver um relatório das viagens efetuadas entre duas datas. Para além disso o cliente pode pedir uma viagem ao carro, moto, carrinha ou a qualquer viatura que esteja mais próxima, ou então a uma viatura específica, através da sua matrícula.

Os métodos que efetuam as viagens calculam a distância da viatura até ao cliente e do cliente até ao destino pedido. É calculado o tempo e preço (depende do preço base da viatura) estimado antes da viagem. No entanto é provável que vão divergir do tempo e preço real, sendo influenciados por fatores aleatórios e pela fiabilidade da viatura. Tal como foi pedido se a diferença entre o tempo real e o estipulado for maior que 25% do tempo estipulado o cliente paga o preço estipulado e a viatura perde fiabilidade. Caso contrário o cliente paga o preço real e a viatura aumenta a sua fiabilidade. As coordenadas da viatura e do cliente são alteradas para o destino e é criada uma instância de `Viagem` que é adicionada à lista de viagens da viatura e do cliente.

O menu do motorista permite visualizar informações sobre si próprio, visualizar o relatório das viagens efetuadas entre datas, alterar a sua disponibilidade, mudar para outra viatura e visualizar o ganho total da sua viatura.

## 2.3 Funcionamento da Aplicação

Ao executar a função `main` da classe `Interface` temos o menu principal que nos apresenta diversas opções numeradas. Para selecionar a opção temos de escolher o número correspondente.

```
 --- MENU ---
1 - Login Cliente
2 - Login Motorista
3 - Registar Cliente
4 - Registar Motorista
5 - Adicionar Viatura
6 - Clientes que mais gastam
7 - Motoristas com mais desvios no preço
0 - Sair e salvar estado
Opção: 4
```

Figura 2.2: *Menu principal.*

Escolhendo por exemplo a opção de registar um novo motorista, é-nos apresentado os dados que devemos inserir para registar o novo motorista que após ser registado nos leva de novo ao menu principal.

```
 --- MENU ---
1 - Login Cliente
2 - Login Motorista
3 - Registar Cliente
4 - Registar Motorista
5 - Adicionar Viatura
6 - Clientes que mais gastam
7 - Motoristas com mais desvios no preço
0 - Sair e salvar estado
Opção: 4
Email: tiago@taxis.pt
Nome: Tiago
Password: tiagovski
Morada: Rua das Tilias
Nascimento (AAAA/MM/DD): 1990/12/10
Motorista criado com sucesso!
```

Figura 2.3: *Registo de um motorista.*

Por outro lado, escolhendo a opção 1 e inserindo as nossas credenciais somos levados para o menu do cliente. Neste menu podemos ver as nossas informações, ver o nosso relatorio de viagens entre datas e efetuar viagens.

Note-se que as datas e as matriculas têm um formato especial e só serão consideradas válidas se respeitarem esse formato.

No menu do motorista (opção 2 no menu principal) é possível ver as nossas informações, ver o nosso relatório de viagens, alterar disponibilidade, mudar de viatura e ver o total faturado pela mesma.

```
Data inicial (AAAA/MM/DD): 1997/09/30
Data final (AAAA/MM/DD): 30/09/1997
30/09/1997 nao e uma data valida!
```

```
Matricula do veiculo: 06GP30
06GP30 nao e uma matricula valida!
```

Figura 2.4: *Formatação das datas e matrículas.*

Temos ainda dados estatísticos nas opções 6 e 7 do menu principal. Estes dados revelam os clientes mais gastadores e os motoristas com maior desvio no preço final.

```
CLIENTES QUE MAIS GASTAM:
Gil=216.59
Harry Potter=210.73
Vitor=125.14
Kiko=114.71
Raul=103.71
Filipa=73.87
Marcio=67.55
Eva=36.95
Adao=33.24
Joao=31.02
```

Figura 2.5: *Clientes mais gastadores.*

Para sair a opção é o número 0. Esta opção grava o estado da aplicação e quando a volta a iniciar o estado é carregado novamente. A aplicação já tem clientes, motoristas, viaturas e viagens efetuadas para efeitos de teste. As credenciais encontram-se no ficheiro *README.TXT*.

# Capítulo 3

# Conclusão

Em geral fazemos um balanço positivo deste trabalho apesar de termos noção de que muitas coisas poderiam ter sido melhoradas se houvesse mais tempo para dedicar.

Entre as coisas que nos deveriamos ter debruçado mais é o facto de não existir uma classe para datas para melhor comparar anos com anos, meses com meses, dias com dias. Assim obteríamos uma melhor organização e controlo.

Um *Array* único de viaturas seria mais vantajoso do que 3 para cada viatura visto que não aproveitamos bem as possibilidade de exploração de cada viatura e acabaram todas por ser bastante similares.

A modularidade na classe `Interface` poderia certamente ser melhorada.

Sentimos também que poderiam ter sido criadas exceções para melhor compreensão de erros que possam eventualmente surgir.

Numa vista abrangente o nosso maior inimigo neste trabalho foi sem dúvida o tempo. Certamente com mais alguns dias de trabalho conseguiríamos por este projeto num nível superior, embora o nível atual seja já satisfatório.