# Projeto de POO - UMER
# Grupo 52

Vítor Castro (A77870), Marcos Pereira (A79116), Sérgio Jorge (A77730)

(da esquerda para a direita)

**Resumo**

Neste relatório faremos uma análise do projeto de Programação Orientada aos Objetos, no qual o objetivo era desenvolver um programa, em Java, que fizesse a gestão da *UMER*, uma empresa de transporte de passageiros. Assim, este documento apresenta detalhadamente a abordagem tomada ao problema proposto pela equipa docente da UC.

# Conteúdo

# 1 Introdução

Este projeto foi realizado com o objetivo de desenvolver um programa responsável por toda a gestão de uma empresa de taxis. Foram, então, propostas pelos professores algumas funcionalidades com contexto real enquadradas no tema, às quais o programa deve responder com sucesso. A implementação destas permitiram consolidar e adquirir conhecimentos ao nível da sintaxe de programação em Java e também incentivaram à exploração de estruturas de dados e de APIs características desta linguagem. Assim, de modo a facilitar a compreensão do projeto, o relatório está dividido da seguinte forma:

**Secção 2 :** Problema;

**Secção 3 :** Solução;

**Secção 4 :** Conclusão.

# 2 Problema

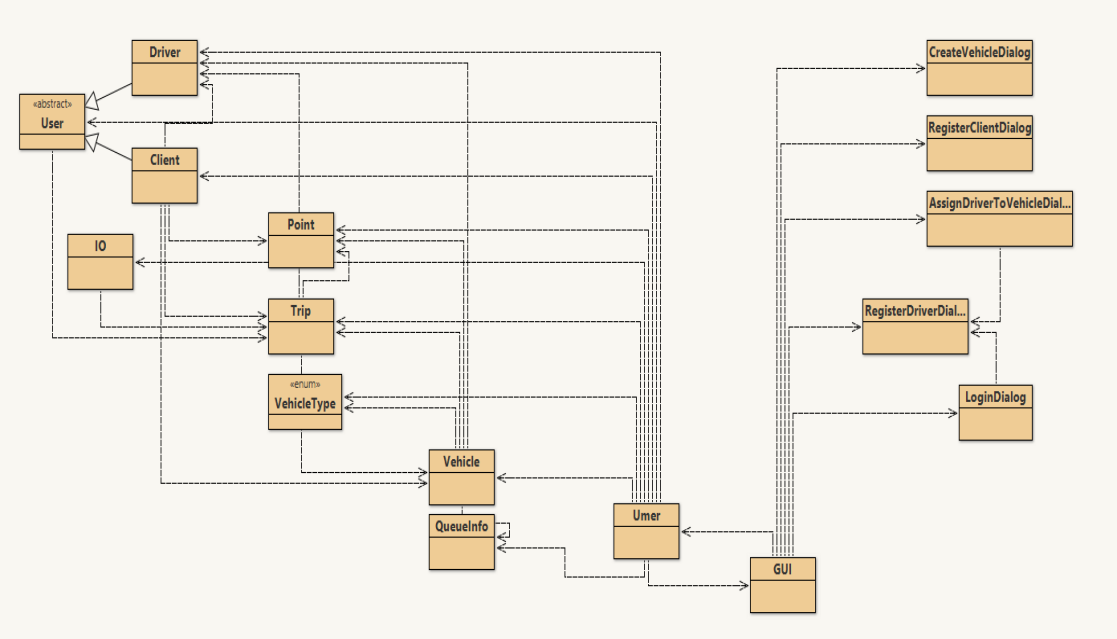
Neste projeto de POO pede-se para desenvolver um programa capaz de auxiliar na gestão de uma empresa de transporte de pessoas. Assim, este deve ser capaz de:

- Registar um utilizador (cliente ou motorista);
- Implementar login no sistema;
- Criar viaturas;
- Associar motoristas a viaturas;
- Um cliente pode solicitar uma viagem escolhendo uma viatura específica ou a mais próxima de si;
- Classificar o motorista, após a viagem;
- Registar um utilizador (cliente ou motorista);
- Possibilidade do cliente conseguir ver as viagens que já fez;
- Possibilidade do motorista conseguir ver as viagens que já fez;
- Indicar o total faturado pela viatura ou pela empresa;
- Listar os 10 clientes que mais gastam;
- Listar os 5 motoristas que apresentam mais desvios entre valores previstos para a viagem e o valor final faturado;
- Gravar o estado do programa em ficheiro.

# 3 Solução

A nossa solução foi implementada com base em diferentes classes dos quais destacamos:

- User;
  - Driver;
  - Client;
- Vehicle;
- Trip;
- IO.



### 3.1 User

Esta é uma superclasse que define variáveis como: email, nome, password, morada e data de nascimento do utilizador. A razão para a implementação de uma hierarquia é porque todos os campos referidos são comuns a motoristas e a clientes pelo que não há a necessidade de repetir código em diferentes classes. Está definida como abstrata porque não haverá necessidade de instanciar Users.

**Driver** A classe referente aos motoristas inclui as variáveis definidas na superclasse mas também o grau de cumprimento de horário, a classificação do motorista, o total de quilómetros feitos por este na empresa e um booleano que informa sobre a disponibilidade do motorista.

**Client** Na classe de clientes são também incluídas as variáveis definidas em User. Além disso, define-se o ponto/localização do cliente e o dinheiro total gasto por este. Nesta classe,

### 3.2 Vehicle

Em Vehicle, classe relativa aos objetos táxis da empresa, são definidas variáveis como: finanças do veículo, posição, ID, tipo de veículo, motorista atual e dois booleanos relativos a fila de espera e ocupação.

### 3.3 Trip

Na classe Trip estão as variáveis relativas às viagens e, a cada viagem, deve estar associado um veículo, um motorista, dois pontos (origem e destino), duração estimada, duração real, custo da viagem , hora de partida e chegada.

### 3.4 IO

Esta é a classe responsável pelas operações de Input/Output do programa. Assim, através dela, é-nos possível guardar o estado da aplicação em determinado momento e é também possível recuperar esse estado posteriormente. O estado é guardado em ficheiros presentes na diretoria do programa.

### 3.5 Implementação - UMER

Na implementação deste programa, optamos por criar uma interface gráfica do utilizador (GUI) de modo a facilitar a interação da aplicação com as funcionalidades exigidas. Usamos três *hashmaps* para guardar a informação relativa a motoristas, clientes e veículos. A escolha recaiu no uso deste tipo de estrutura de dados por nos possibilitar ganhos de performance relativamente a listas. Por sua vez, as listas são usadas, no nosso programa, para guardar a informação relativa a viagens já completadas e viagens a decorrer.

Primeiramente, esta classe é responsável por métodos relativos a *login* e *logout*, instancia ou cria objetos das classes motoristas, clientes e veículos e adiciona-os às estruturas de dados respetivas e trata de associar um condutor a um veículo. É, então, que se torna possível a criação ou o começo de uma viagem.

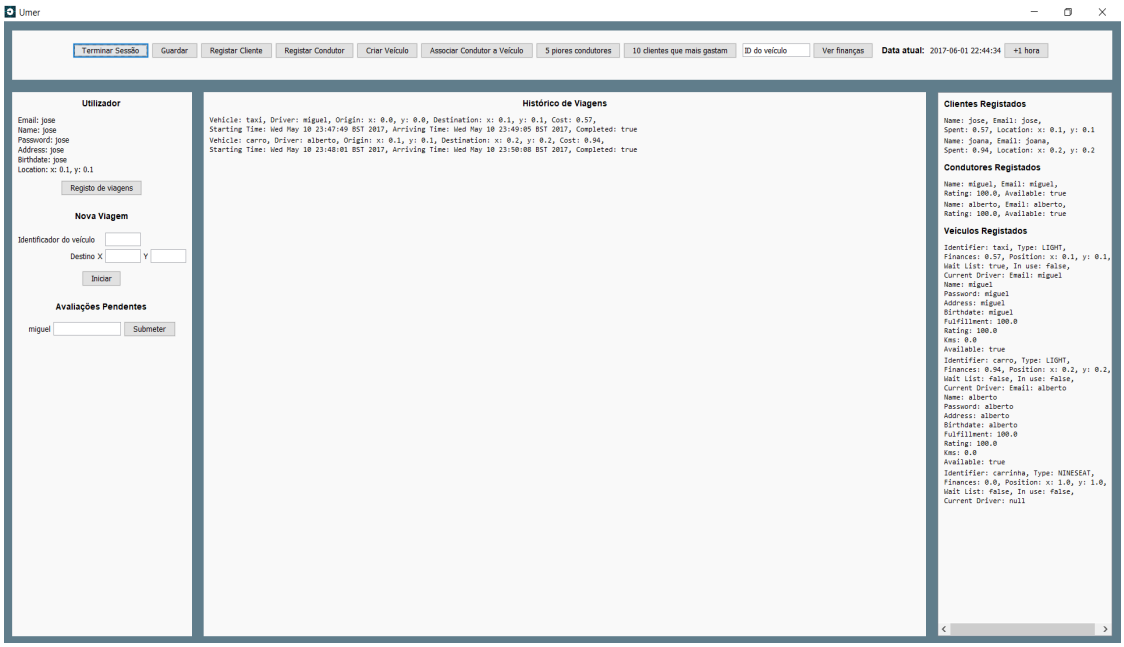
No que diz respeito à criação de viagens, é verificado se há algum veículo disponível ou se o que foi pedido pelo cliente está disponível. Em caso afirmativo, a viagem é adicionada às listas de viagens, as posições dos objetos são atualizadas, as finanças e o custo são adicionadas ao veículo e ao cliente e o cliente avalia o condutor. Em caso negativo, se existir algum veículo com fila de espera, o cliente é adicionado à sua fila de espera. É importante referir que a nossa aplicação faz uso de um relógio que nos permite marcar a hora de início e de fim da viagem. Assim, é-nos possível saber quando a viagem acaba pelo que, nesse momento, podemos tirar a viagem da lista de viagens a decorrer. Está também implementado o método *fastforward* que adianta o relógio.

Para cliente e motorista terem acesso às viagens que já fizeram percorre-se a lista do historial de viagens e vê-se em que viagens é que estes participaram e é apenas feito um ToString dessa informação.

Em relação aos tops, decidimos converter as *hashmaps* de clientes e de motoristas para *arraylists* que, posteriormente, são ordenados. A seguir, passamos os N primeiros elementos dos *arraylists* (já ordenados) para *linkedhashmaps* aos quais damos toString.

O estado da aplicação é guardado a partir de um botão na GUI que chama os métodos responsáveis por esse efeito, presentes na classe IO. Relativamente a carregar o estado, é feita a chamada aos métodos, automaticamente, quando a GUI é aberta.

### 3.6 Resultado Final




# 4 Conclusões

Este projeto serviu para aprofundarmos o conhecimento da linguagem JAVA, assim como as APIs que lhe estão associadas. Achamos que a realização de um trabalho deste tipo permite uma consolidação proveitosa da linguagem, não só em termos teóricos como também em termos práticos. Permite também melhorar as habilidades na resolução de problemas. No entanto, apesar de nos termos proposto a fazer todos os pontos do trabalho, acabamos por não ter tempo para implementar as empresas de taxis e o random dos condutores. Não temos dúvidas de que o conseguimos fazer, mas tivemos de dedicar tempo precioso a outras UCs. Fomos além do pedido e fizemos uma GUI, o que nos poderá ter custado tempo útil para fazermos os pontos pedidos que faltaram, mas acreditamos que ficamos a conhecer melhor outra parte da programação que até ao momento desconhecíamos e, assim sendo, cremos ter sido vantajoso este esforço que sabemos não contar para avaliação.