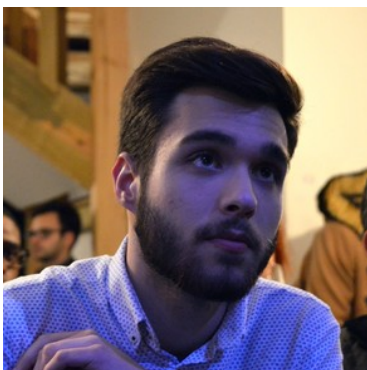


Relatório do Trabalho Prático de Programação Orientada a Objetos Grupo11 POO2017

Elísio Freitas Fernandes {55617}, Daniel Gonçalves Martins {73175}, and Nuno José Ribeiro da Silva {78879}

Universidade do Minho, Departamento de Informática, 4710-057 Braga, Portugal
e-mail: {a55617, a73175, a78879}@uminho.pt



Daniel Gonçalves Martins {73175}



Elísio Freitas Fernandes {55617}



Nuno José R. da Silva {78879}

Resumo Serve o presente como relatório do projeto elaborado no âmbito da unidade curricular de Programação Orientada a Objetos. Projeto esse no qual se previa a elaboração de um programa para a empresa UMeR, o qual fosse capaz de garantir a prestação continua do seu serviço. Os requisitos de tal programa incluem a criação e manutenção da base de dados que inclui os dados dos clientes, bem como as informações dos seus colaboradores e das suas viaturas, utilizadas para a prestação do serviço em questão. Mais ainda, está incluída a criação de uma interface de interação com os utilizadores com várias opções, como: criação de conta; posterior acesso e atualização dos dados da conta; consulta de histórico de serviços requeridos/prestados.

1 Introdução

Com o presente documento pretende-se apresentar resumidamente o trabalho prático elaborado pelo grupo, como proposta de resolução do enunciado apresentado, no seguimento da UC de programação Orientada aos Objetos. As necessidades da empresa UMeR foram abordadas e tratadas usando Java, temos recorrido maioritariamente ao Atom como editor de texto e ao Terminal como compilador, bem como o Visual Studio Code e a consola do linux para Windows. Irão ser apresentadas as classes elaboradas e a várias escolhas específicas feitas em cada uma delas, a sua hierarquia e sua razão de ser.

2 Tipos de dados usados

No geral, as estruturas de dados usadas para o armazenamento dos dados de clientes, veículos e viagens foram os *Maps* - verificar, por exemplo, **Figura 1**. Dada a sua eficácia em termos de armazenamento ordenado de

dados/rapidez na procura, à sua conveniência, uma vez que permite identificar cada objecto unicamente através de um dado único. Queríamos garantir a unicidade de dados, pois não haveria razão para termos duas entradas de registo para o mesmo utilizador, assim, usamos o email do mesmo como *key* de identificação no *Map* e assim garantimos que o mesmo *User* não se regista duas vezes. Pouparamos memória e reduzimos dados duplicados nas bases de dados da empresa. O mesmo se dá para os *Vehicle*, *Trips* e para a associação de *Driver/Vehicle*.

```
public class UMeR implements Serializable
{
    private boolean isLoggedIn;
    private String loggedUserEmail;
    private Map<String, User> userList;
    private Map<String, Vehicle> vehicleList;
    private Map<Integer, Trip> tripList;
    private Map<String, String> driverVehicle;
    private int tripNumber;
```

Figura 1. Variáveis da classe UMeR

Por outro lado, usamos também dados do tipo *List* em situações como na classe *User*, para guardar os ids das viagens efetuadas, uma vez que o tipo de dados é só um tipo de dados primitivo.

2.1 Metodologias de implementação dos tipos de dados

A nível da declaração das variáveis de instância foi usado sempre um tipo de dados com um grande nível de generalização. Com tal pretendemos facilitar a alteração dos tipos de dados específicos usados. Por exemplo, ao usar a declaração do tipo *Map* invés de *HashMap* permitimos que no futuro fosse fácil alterar esse tipo para um *TreeMap* por exemplo, mantendo no entanto a mesma intenção de permitir uma gestão eficiente e organizada da informação.

Pela mesma razão declaramos as variáveis de instância como *List*, permitindo depois a escolha do tipo específico de *List* pretendida - verificar, por exemplo, **Figura 2** .

```
public abstract class User implements Comparable<User>, Serializable {
    private String name;
    private Address address;
    private LocalDate birthday;
    private String email;
    private String password;
    private List<Integer> tripHistory;
    private double totalTripCost;
```

Figura 2. Variáveis da classe User

Observando a **Figura 3**, é visível a uniformização dos métodos das classes a esta metodologia de abstração do tipo de dados.

3 Métodos

Em cada classe existem métodos que permitem a interação da empresa com a mesma, permitindo a criação, de instâncias de classe, atribuição de valores às várias variáveis através dos métodos *set<nome da variável>*, bem como obtenção dos valores contidos nas mesmas, através dos métodos *get<nome da variável>*.

```

/**
 * Gets copy of this instance User's trip history
 * @return Returns List that's a copy of this instance of User's Trip History
 */
public List<Integer> getTripHistory(){
    List<Integer> aux = new LinkedList<Integer>();

    ((LinkedList<Integer>) this.tripHistory).forEach(trip -> {aux.add(trip)});

    return aux;
}

```

Figura 3. Métodos com abstração de dados assegurada

Cada classe tem em comum os métodos `clone()`, `equals()`, `compareTo()`, `hashCode()` e `toString()`. O método `toString()` e o `equals()` é definido na classe abstrata e redefinido nas subclasses. O método é definido na classe abstrata porque várias das variáveis são definidas nesta classe e, assim, evita-se a duplicação de código, no entanto, existem também variáveis declaradas na classe *client*, pelo que é necessário chamar o *super* da super classe para tratar da execução do método sobre as variáveis lá definidas e só depois é aplicado o código local ao resto das variáveis de instância. O método `clone()` é um método abstrato e, portanto, é totalmente definido em cada subclasse, uma vez que apenas a subclasse sabe como definir cada um dos parâmetros que compõe. O método `compareTo()` é o responsável pela "ordem natural" definida em cada classe para a ordenação das suas instâncias. Já o método `hashCode()` é um método que todas as classes têm apenas na sua superclasse abstrata, uma vez que é nela que se encontra o identificador único inerente a cada uma das subclasses que a ampliam.

Salienta-se o método *setPassword* (*String password*) e *public String getPassword* () que na sua simplicidade, apenas guardam a password entregue pela empresa, sendo da responsabilidade e controlo da mesma garantir a segurança da mesma - nesta proposta tal método não foi implementado. Destacam-se também métodos construtores vazios da classe *User*, *Vehicle* e *Trip* que são privados, uma vez que não pretendido que estes sejam criadas instâncias destas classes sem dados concretos. No entanto, dada a filosofia do Java, foi necessário criar os construtores, para que o Java em si não desse automaticamente essa possibilidade aos utilizadores.

Para além destes métodos, cada classe dispõe dos seus métodos próprios, os quais podem ser consultados em anexo, necessários para efetuar as suas operações.

4 Expressão escrita

Ao longo do projeto foi usado um estilo restrito de regras para a criação dos seus ficheiros componentes. Para a declaração das variáveis foi definido que as mesmas deveriam ter nomes sugestivos que tornassem fácil compreender aquilo que as mesmas representavam. Mais ainda, para variáveis compostas por várias palavras, foi adotado o método `lowerCamelCase`, em que cada letra de cada palavra é iniciada por letra maiúscula.

5 Arquitetura

Nesta proposta foram utilizados vários tipos de classes com diferentes atributos. Desde classes concretas a classes abstratas, classes com e sem implementações de interfaces e subclasses de classes abstratas.

Por questões de necessidade de gravação do estado da aplicação, todas as classes tem em comum a implementação da interface *Serializable*, oferecendo, assim, um método simples de gravação dos objetos com todos os seus estados atuais em ficheiro, para posterior consulta.

5.1 Classe UMeR

Utiliza `Maps`, e tipos primitivos para guardar os seus dados. Possui um registo de toda a informação necessária para o bom funcionamento da empresa. Contém quatro estruturas de dados `Map` onde se encontram guardados os dados de: 1. *User* como *value* e *email* como *key*, para guardar todos os utilizadores registados na empresa, quer clientes, quer condutores ; 2. *Vehicle* como *value* e *licensePlate* como *key*, para guardar a informação da

viaturas ao serviço da empresa; 3. Trip como *value* e id de viagem como *key*, para manter um histórico de todas as viagens efetuadas através da empresa; 4. Email como *value* e licensePlate como *key*, para manter um registo do veículo associado a cada condutor. Existe ainda a variável *isLogged*, do tipo *boolean*, que indica se existe um utilizador "logado". Se sim, então o seu e-mail estará presente na variável *loggedUserEmail*, do tipo *String*. Por último, existe na classe uma variável *tripNumber*, do tipo *Integer*, responsável por garantir a sequencialização dos identificadores únicos da viagem, utilizados no Map das Trips.

Foram aglomerados todos os dados do tipo *User*, respetivamente, todos os dados de tipo *Trip* e *Vehicle*, num só *Map*, uma vez que eram compatíveis, de maneira a ser possível efetuar operações sobre todos os utilizadores de uma só vez, bem como facilitar a adição de novos tipo de utilizadores.

5.2 User

Não são permitidas instanciações desta classe, uma vez que se trata duma classe abstrata, que implementa comparadores. Define os dados dos utilizadores que são comuns entre as suas subclasses e pressupões a criação de subclasses que permitirão a instanciação de objetos deste tipo. Contém variáveis de tipos primitivos e uma *List* para guardar os registos da viagens efetuadas pelo utilizador.

Client A classe *Client* é uma subclasse de *User*, tem a particularidade de possuir uma variável do tipo personalizado *Coordinates* que permite guardar a localização do cliente num plano cartesiano. Esta é atualizada após ter efetuado uma viagem.

Driver A classe *Driver* é uma subclasse de *User*, tem a particularidade de possuir variáveis referentes ao seu performance como motorista bem como um total de km já percorridos ao serviço da empresa e a indicação da sua disponibilidade atual para efetuar um serviço. A sua disponibilidade é alterada conforme o mesmo está ou não a efetuar uma viagem.

A classificação do condutor é determinada pela média das suas classificações. A performance é um valor de 0 a 100. Este valor começa por ter valor 100, sendo que caso a sua primeira viagem tenha um desvio superior a 25% do tempo estimado, ele passa automaticamente para 50. Após estas situações concretas, este valor é alterado segundo a subtração ou soma ao valor atual de 50 / (<numero de viagens efetuadas anteriormente> + 1), dependendo de a viagem efetuada ter ou não excedido 25% do tempo estimado. Ou seja, a sua performance vai aumentado, ou diminuindo, um valor incrementalmente pequeno conforme faz mais viagens.

5.3 Vehicle

Não são permitidas instanciações desta classe, uma vez que se trata duma classe abstrata, que implementa comparadores. Define os dados de utilização dos veículos utilizados ao serviço da empresa, bem como o custo da viagem por quilómetro, *fare*, a fiabilidade, *reliability*, e ainda a sua localização, *location*, do tipo *Coordinates*, que é atualizada no fim de cada viagem. Possui os métodos abstratos *clone()* e *setReliability()*.

Car, Motorcycle e Van Estas classes são subclasses de *Vehicle*, são idênticas e não possuem variáveis próprias. Cada uma das classes possui uma definição dos métodos abstratos da sua superclasse.

5.4 Trip

A classe *Trip* é a responsável pela definição do parâmetros das viagens efetuadas, incluindo: o id da viagem, o id do cliente e do condutor, a data em que foi efetuada, a matrícula do carro que efetuou a viagem, a localização do veículo, do cliente e do destino, o custo/tempo estimado da viagem e o custo/tempo real da mesma. Esta classe possui também comparadores para que as viagens possam ser ordenadas segundo mais do que um critério.

5.5 Coordinates

A classe *Coordinates* é a responsável pela definição da localização de algum objeto e tratamento de do cálculo da distância entre dois pontos.

5.6 Address

A classe *Address* é a responsável pela definição da cidade e país de um dado utilizador, contendo apenas *Strings*. Implementa exceções.

5.7 EmailValidator

Classe responsável por analisar o formato de um dado email, e verificar se tem um formato válido.

5.8 Interface

A classe *Interface* é a responsável pela impressão dos menus no ecrã, contendo apenas métodos que imprimem conjuntos de *Strings* no ecrã. Cada conjunto de *Strings* imprimidas é diferente de acordo com o menu que se pretende imprimir.

5.9 Main

A classe *Main* é a principal responsável pelo funcionamento da aplicação, uma vez que é ela que continuamente recebe, interpreta e valida os comandos introduzidos pelo utilizador, imprimindo os menus solicitados e chamando as funções da UMeR necessárias para efetuar as opções solicitadas. A classe *Main*, a classe *UMeR* e o utilizador têm uma relação de repetida comunicação para solicitar, obter, introduzir e validar dados. Acima de tudo, esta classe é a responsável pelo tratamento de erros.

5.10 Comparadores

Para além das classes mencionadas, esta proposta dispõe também de um conjunto de comparadores, os quais podem ser usados para ordenar as listas e os maps por uma ordem diferente da ordem natural determinada pelo método `compareTo()`.

6 Hierarquia

Podemos ver abaixo na **Figura 4** a hierarquia das classes do projeto e a maneira como todas elas estão ligadas entre si. Podemos também verificar na **Figura 5** uma representação gráfica das exceções previstas no código.

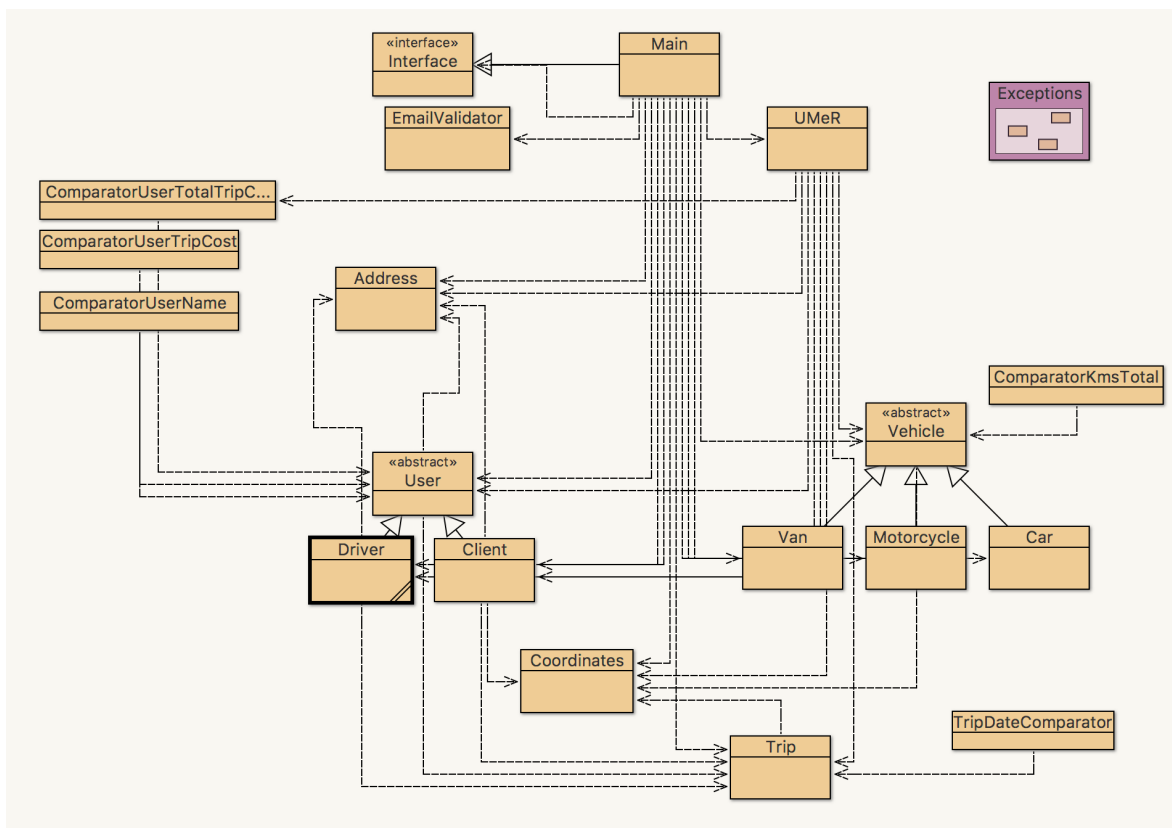


Figura 4. Hierarquia de classes na UMeR

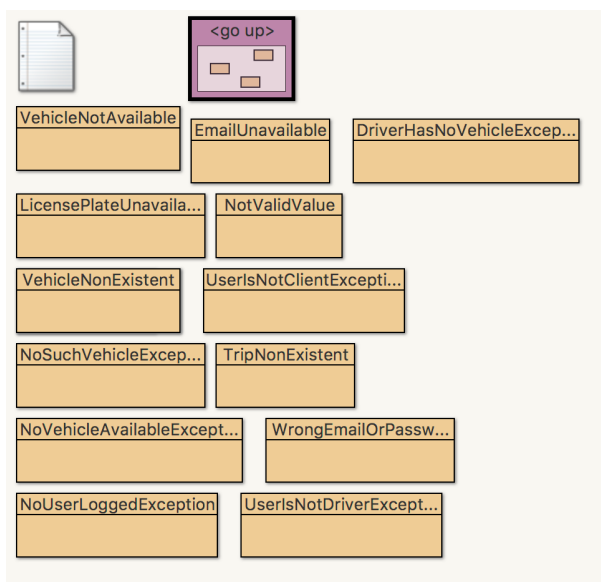


Figura 5. Exceções previstas no código

7 Manual de utilização

Ao longo da execução do programa, o próprio programa vai indicando o Input que está à espera. Caso não seja um input valido o programa reconhece como inválido e pede novamente input. A qualquer ponto da utilização é possível voltar atrás e cancelar a operação do momento, excepto após uma viagem ter sido efetuada. Nesse caso é possível voltar atrás apenas depois de dar classificação ao motorista.

7.1 Menus

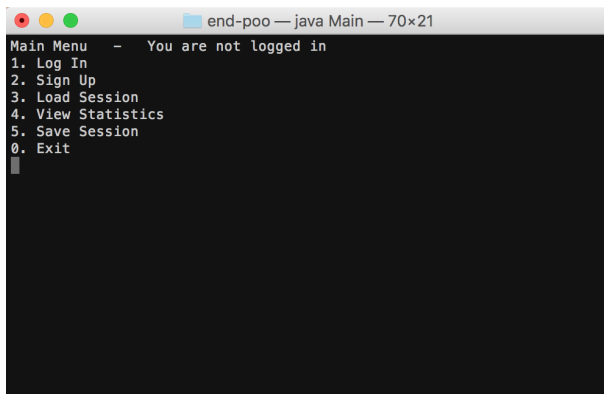


Figura 6. Menu principal antes do login

Menu Sem Utilizador Este menu é apresentado o utilizador não tem *log in* feito. A partir deste menu é possível: iniciar sessão (1. Log In); registar no serviço (2. Sign up); fazer load a partir de um ficheiro (3. Load Session); ver estatísticas (4. View Statistics) tais como, 10 clientes que mais gastaram e 5 condutores com mais desvios de tempo previsto nas viagens; e guardar para ficheiro a sessão (5. Save Session); sair da aplicação (0. Exit).

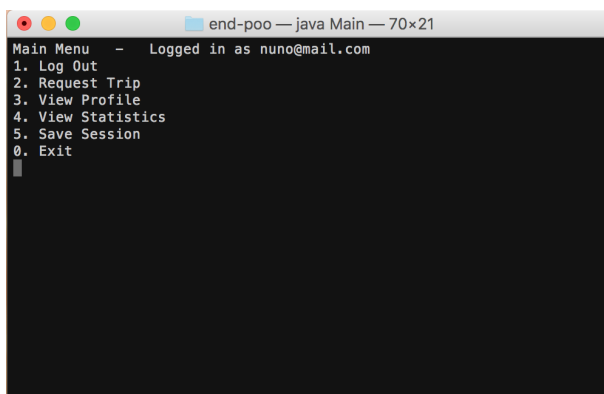


Figura 7. Menu principal de um cliente

Menu de Cliente Este menu é apresentado quando um utilizador tem *log in* feito como cliente. A partir deste menu é possível: sair da sessão como utilizador (1. Log Out); efetuar uma viagem (2. Request Trip); ver o

perfil de utilizador (3. View Profile); ver estatísticas (4. View Statistics); guardar sessão para ficheiro (5. Save Session); sair da aplicação (0. Exit).



Figura 8. Menu principal de um condutor

Menu de Condutor Este menu é apresentado quando um utilizador tem *log in* feito como condutor. A partir deste menu é possível: sair da sessão como utilizador (1. Log Out); ver o perfil de utilizador e informação sobre veiculo (2. View Profile); registar um novo veiculo, caso ainda não tenha um associado, (3. Register Vehicle); mudar localização atual (4. Set Location); alterar a disponibilidade (5. Switch Availability); ver estatísticas (6. View Statistics); guardar sessão para ficheiro (7. Save Session); sair da aplicação (0. Exit). A partir do menu de condutor é possível mudar a disponibilidade do condutor. Quando não está disponível não será possível fazer viagens com este condutor e o seu veiculo.

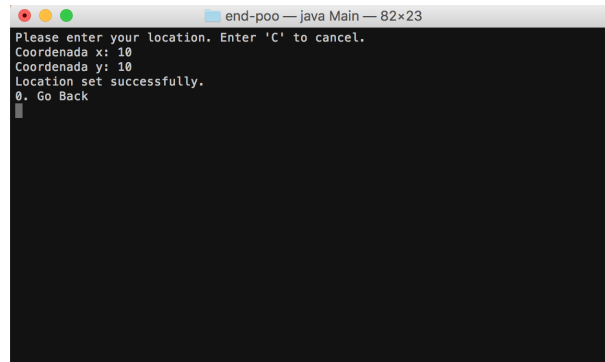


Figura 9. Informação sobre a localização

A partir do menu de condutor é possível mudar a localização atual do condutorveiculo.

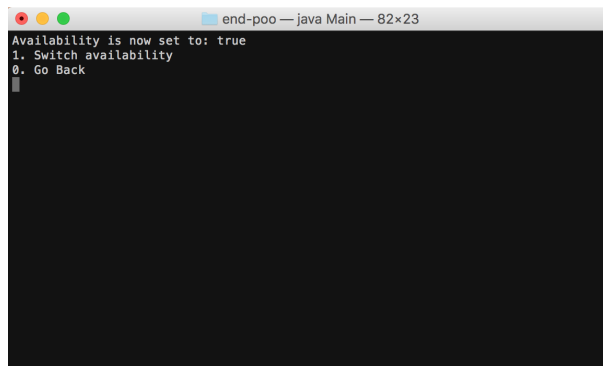


Figura 10. Mudança de disponibilidade

7.2 Iniciar Sessão de Utilizador

No menu de *log in* é pedido ao utilizador um email e password relativa à conta à qual está a tentar aceder. É apresentada uma opção para voltar atrás e cancelar a operação (0. Go Back). Quando é introduzido o email, se o formato do email não for valido a aplicação indica o email como inválido e pede-o novamente. Caso o *log in* tenha ocorrido com sucesso é apresentada a seguinte mensagem:

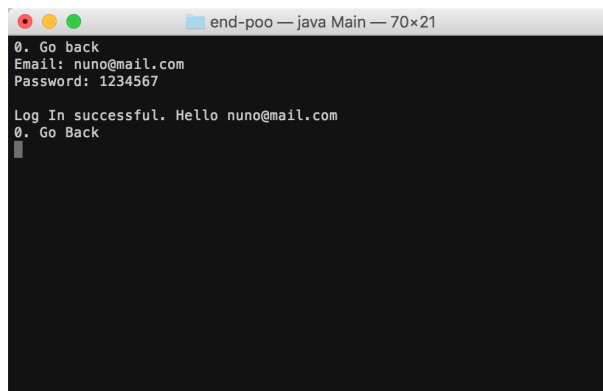


Figura 11. Login com sucesso

Caso contrário é pedido novamente a informação para *log in*:

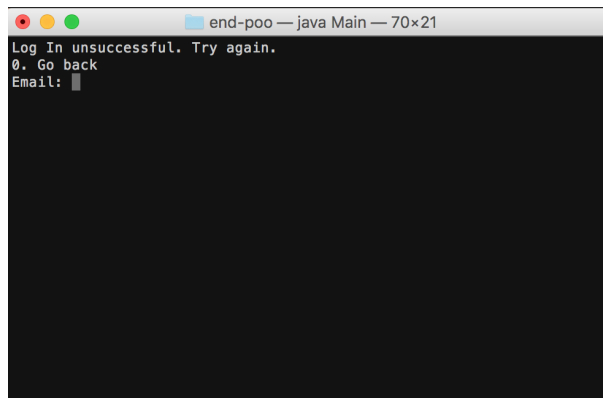


Figura 12. Tentativa falhada de login

7.3 Registrar Utilizador

No menu de *Sign up*, é pedido ao utilizador para indicar se deseja se registar no serviço como cliente ou como condutor. Independentemente da escolha, os dados a serem pedidos sobre o utilizador são os mesmos. A aplicação vai validando o *input* a cada passo pedindo esse pedaço de informação novamente caso o *input* anterior não seja válido. A qualquer altura no processo de registo é possível cancelar a operação e voltar ao menu anterior (0. Go Back).

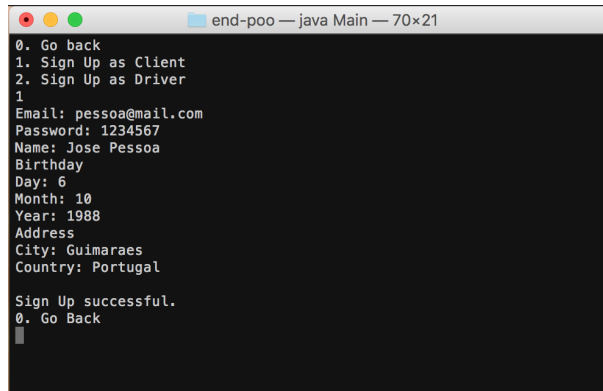
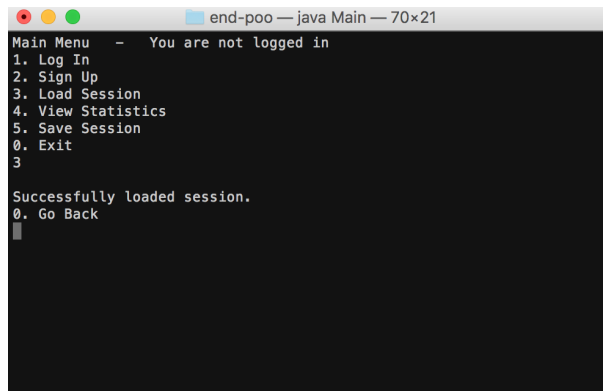


Figura 13. Processo de login de um utilizador na plataforma

Uma vez que os dados introduzidos vão sendo validados ao longo do processo de registo é improvável que o o registo ocorra sem sucesso, no entanto, nesse caso, os dados serão pedidos novamente, voltando ao início.

7.4 Carregar Sessão

Caso tenha sido escolhida a opção de carregar uma sessão a partir do ficheiro, o programa substituirá a informação da UMeR pelo que se encontra no ficheiro. É apenas apresentada uma mensagem a indicar se o *load* foi efetuado com sucesso ou não.

A terminal window titled "end-poo — java Main — 70x21" with a dark background. It displays a menu with options 1 to 5 and 0. Option 3, "Load Session", has been selected, and the terminal shows the message "Successfully loaded session." followed by a sub-menu with option 0, "Go Back".

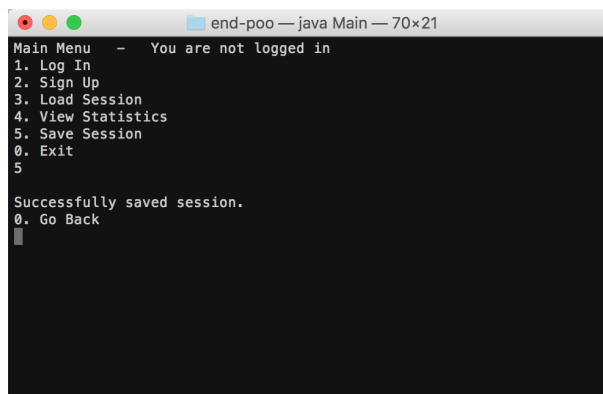
```
end-poo — java Main — 70x21
Main Menu - You are not logged in
1. Log In
2. Sign Up
3. Load Session
4. View Statistics
5. Save Session
0. Exit
3

Successfully loaded session.
0. Go Back
```

Figura 14. Notificação de login com sucesso

7.5 Guardar Sessão

Caso tenha sido escolhida a opção de guardar um sessão para um ficheiro. O programa escreve a informação sobre a UMeR e escreve para um ficheiro em binário. É apenas apresentada uma mensagem a indicar se o *save* foi efetuado com sucesso ou não.

A terminal window titled "end-poo — java Main — 70x21" with a dark background. It displays the same menu as Figure 14. Option 5, "Save Session", has been selected, and the terminal shows the message "Successfully saved session." followed by a sub-menu with option 0, "Go Back".

```
end-poo — java Main — 70x21
Main Menu - You are not logged in
1. Log In
2. Sign Up
3. Load Session
4. View Statistics
5. Save Session
0. Exit
5

Successfully saved session.
0. Go Back
```

Figura 15. Possibilidade de guardar a sessão atual

7.6 Ver Estatísticas

Neste menu é possível ver estatísticas sobre a UMeR. Nomeadamente: 10 clientes que mais gastam em viagens e 5 condutores com mais desvio entre tempo esperado de uma viagem e tempo que realmente levou.

```
end-poo — java Main — 70x21
View Statistics
1. 10 Clients who spend the most
2. 5 Drivers with worst performance
0. Go Back
█
```

Figura 16. Consulta de estatísticas de clientes e condutores

```
end-poo — java Main — 70x21
Name:  Elisio  Fernandes
Total Spent:  115.54999999999998

Name:  Nuno Silva
Total Spent:   60.75

Name:  Daniel Martins
Total Spent:   42.1

0. Go back
█
```

Figura 17. Visualização dos 10 clientes que mais gastam

```
end-poo — java Main — 70x21
Name: Anabela Cardoso
Email: anabela@mail.com
Number of deviations: 9

Name: Sofia
Email: sofia@mail.com
Number of deviations: 5

Name: Joao Silva
Email: joao@mail.com
Number of deviations: 4

0. Go back
█
```

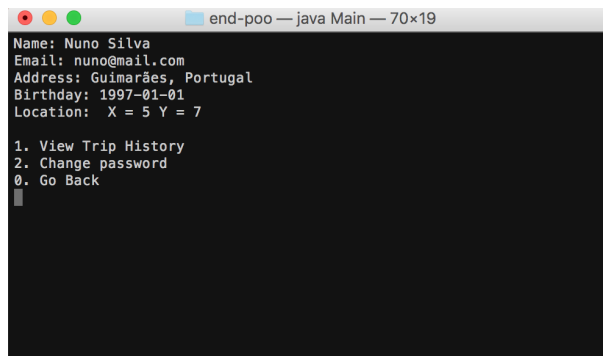
Figura 18. Visualização dos perfis dos condutores e respetivos desvios

7.7 Fechar sessão

A opção *log out* simplesmente sai do menu de utilizador, reiniciando as variáveis relativas ao *log in* na UMeR e volta a o menu sem utilizador.

7.8 Ver Perfil

Perfil de Cliente Quando um cliente escolhe a opção *View Profile*, é apresentada varia informação sobre o seu perfil. É aqui que é possível alterar a password e ver o histórico de viagens

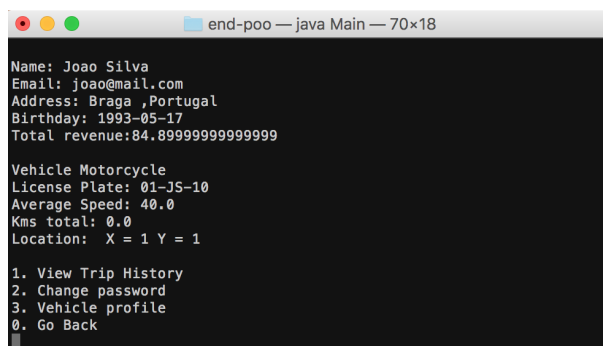


```
end-poo — java Main — 70x19
Name: Nuno Silva
Email: nuno@mail.com
Address: Guimarães, Portugal
Birthday: 1997-01-01
Location: X = 5 Y = 7

1. View Trip History
2. Change password
0. Go Back
```

Figura 19. Perfil de um cliente

Perfil de Condutor No perfil de Condutor é apresentada a mesma a informação sobre o utilizador que é apresentada no perfil de cliente. Para além disso é possível ver a informação relativa ao veículo ao qual o condutor está associado. A opção (3. Vehicle Profile) permite ver informação mais detalhada sobre o veículo e alterar alguma informação sobre o veículo.



```
end-poo — java Main — 70x18
Name: Joao Silva
Email: joao@mail.com
Address: Braga, Portugal
Birthday: 1993-05-17
Total revenue: 84.89999999999999

Vehicle Motorcycle
License Plate: 01-JS-10
Average Speed: 40.0
Kms total: 0.0
Location: X = 1 Y = 1

1. View Trip History
2. Change password
3. Vehicle profile
0. Go Back
```

Figura 20. Perfil pessoal de um condutor

```
end-poo — java Main — 82x23
Vehicle profile
-----
Car
License Plate: 10-AC-10
Average speed: 70.0
Total Kms:0.0
Fare: 2.5
Reliability: 0.6168584176913557
Seats: 4
Location: X = 90 Y = 90
1. Change Average Speed
2. Change Fare
0. Go Back
```

Figura 21. Perfil de um veículo

Ver Histórico de Viagens A partir do perfil é possível consultar ao histórico de viagens. É pedida uma data inicial e uma data final que constitui o espaço de tempo sobre o qual queremos ver as viagens efetuadas.

```
end-poo — java Main — 70x57
View trip history.
Choose range of dates to view your trip history.
0. Go Back

From what date do you want to see your trip history?
Day: 1
Month: 1
Year: 1990

Up to what date do you want to see your trip history?
Day: 1
Month: 12
Year: 2020
Trip History:

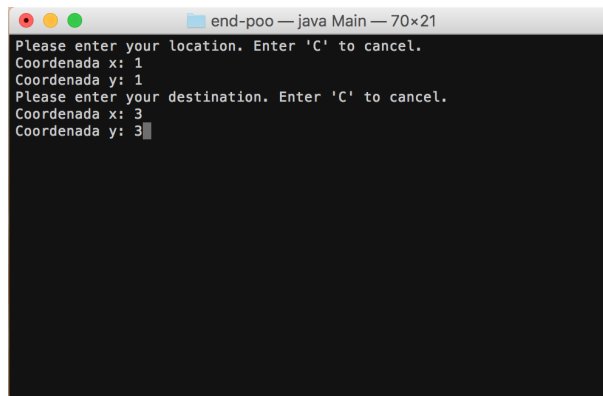
Date:
2017-06-03T19:41:29.517
Client: nuno@mail.com
Driver: anabela@mail.com
Destination: X = 5 Y = 7
Trip duration: 29
Cost: 7.25

Date:
2017-06-03T19:40:14.975
Client: nuno@mail.com
Driver: joao@mail.com
Destination: X = 3 Y = 3
Trip duration: 4
Cost: 1.2
```

Figura 22. Visualização das viagens feitas entre determinadas datas

7.9 Efetuar Viagem

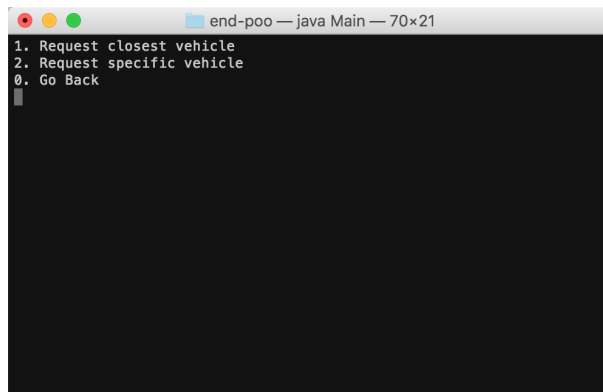
A partir do menu de cliente é possível efetuar uma viagem. Inicialmente serão pedidas as coordenadas atuais do utilizador e as coordenadas do seu destino.

A terminal window titled "end-poo — java Main — 70x21" with a dark background. It displays the following text: "Please enter your location. Enter 'C' to cancel.", "Coordenada x: 1", "Coordenada y: 1", "Please enter your destination. Enter 'C' to cancel.", "Coordenada x: 3", and "Coordenada y: 3" with a cursor at the end of the last line.

```
end-poo — java Main — 70x21
Please enter your location. Enter 'C' to cancel.
Coordenada x: 1
Coordenada y: 1
Please enter your destination. Enter 'C' to cancel.
Coordenada x: 3
Coordenada y: 3
```

Figura 23. Processo de pedido de uma viagem na aplicação - Localização e destino

De seguida, o utilizador pode escolher entre simplesmente requisitar o veiculo mais proximo de si ou requisitar um veiculo à sua escolha.

A terminal window titled "end-poo — java Main — 70x21" with a dark background. It displays the following text: "1. Request closest vehicle", "2. Request specific vehicle", and "0. Go Back" with a cursor at the end of the last line.

```
end-poo — java Main — 70x21
1. Request closest vehicle
2. Request specific vehicle
0. Go Back
```

Figura 24. Processo de pedido de uma viagem na aplicação - Escolha do veículo

Caso o utilizador pretenda requisitar o veiculo mais proximo ser-lhe-à dada a escolha do tipo de veiculo.

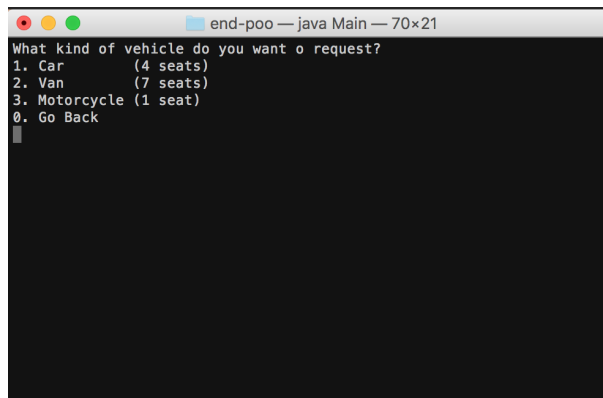


Figura 25. Processo de pedido de uma viagem na aplicação - Tipo de veículos disponíveis

Caso o utilizador pretenda escolher um veiculo em específico, será apresentada uma lista de condutores e veículos por ordem decrescente de distância, finalmente deve escrever o email do condutor que quer requisitar.

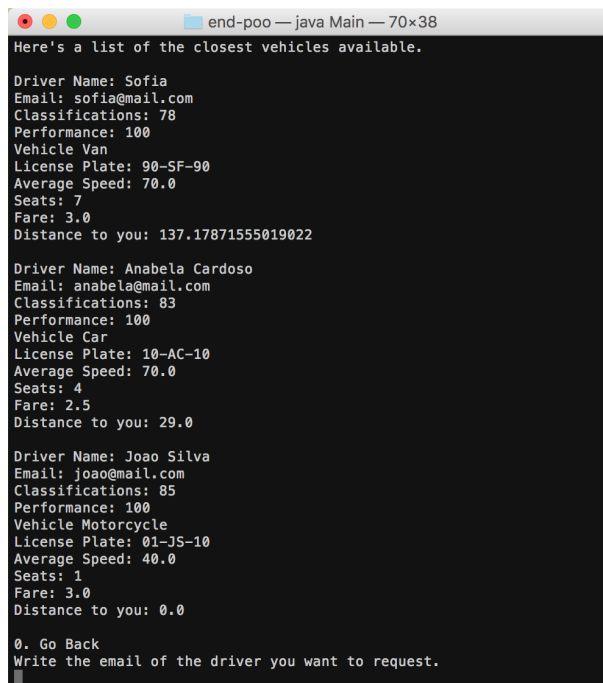
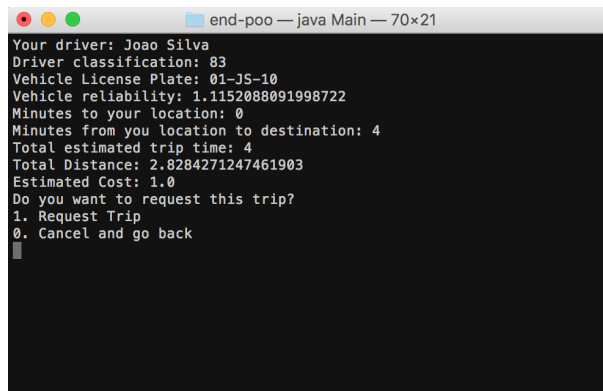


Figura 26. Possibilidade de escolha de condutor específico

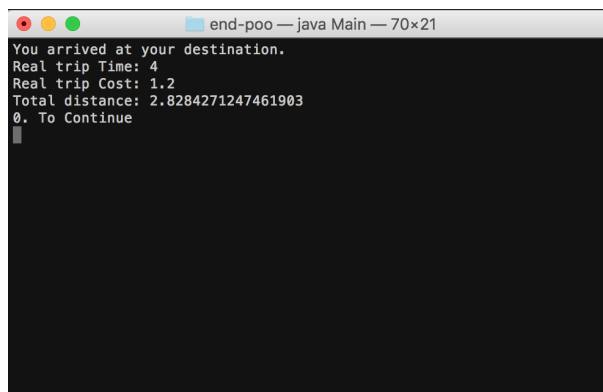
Após a escolha de veiculo será apresentado finalmente o pedido, com informação sobre condutor, veiculo, tempo e custo esperado para a viagem. Nesta altura o cliente pode escolher efetuar a viagem, ou cancelar o pedido, voltando ao menu principal.

A terminal window titled "end-poo — java Main — 70x21" with a dark background and light text. It displays the following information: "Your driver: Joao Silva", "Driver classification: 83", "Vehicle License Plate: 01-JS-10", "Vehicle reliability: 1.1152088091998722", "Minutes to your location: 0", "Minutes from you location to destination: 4", "Total estimated trip time: 4", "Total Distance: 2.8284271247461903", and "Estimated Cost: 1.0". It then asks "Do you want to request this trip?" and provides two options: "1. Request Trip" and "0. Cancel and go back". A cursor is visible at the end of the second option.

```
end-poo — java Main — 70x21
Your driver: Joao Silva
Driver classification: 83
Vehicle License Plate: 01-JS-10
Vehicle reliability: 1.1152088091998722
Minutes to your location: 0
Minutes from you location to destination: 4
Total estimated trip time: 4
Total Distance: 2.8284271247461903
Estimated Cost: 1.0
Do you want to request this trip?
1. Request Trip
0. Cancel and go back
█
```

Figura 27. Processo de pedido de uma viagem na aplicação - Dados da viagem e confirmação final

Caso tenha escolhido seguir com a viagem, a viagem é efetuada e o cliente é notificado sobre os dados da viagem. Se a viagem demorou perto do tempo esperado o cliente é simplesmente notificado do tempo real da viagem e do preço que terá de pagar. Caso a viagem tenha demorado mais que 125% do tempo esperado, será dada a possibilidade de negociar um preço a pagar.

A terminal window titled "end-poo — java Main — 70x21" with a dark background and light text. It displays the following information: "You arrived at your destination.", "Real trip Time: 4", "Real trip Cost: 1.2", and "Total distance: 2.8284271247461903". It then prompts the user with "0. To Continue". A cursor is visible at the end of the prompt.

```
end-poo — java Main — 70x21
You arrived at your destination.
Real trip Time: 4
Real trip Cost: 1.2
Total distance: 2.8284271247461903
0. To Continue
█
```

Figura 28. Final de uma viagem

Após efetuado o pagamento é pedido ao cliente dar uma classificação ao condutor, de 0 a 100.

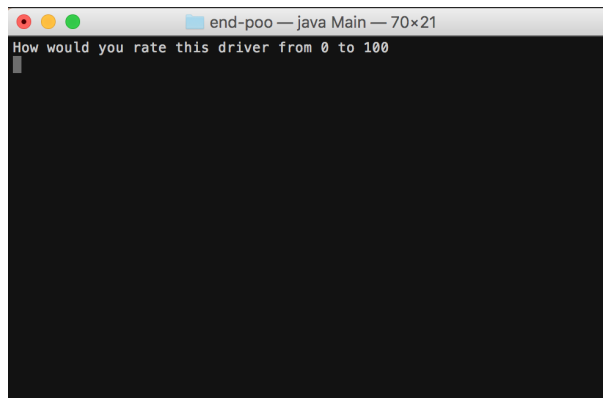


Figura 29. Classificação do condutor

Finalmente, o cliente volta ao menu principal.

7.10 Sair da aplicação

A opção (0. Exit), disponível nos menus principais, simplesmente termina a sessão. É importante ter em conta que esta opção termina o programa sem guardar qualquer informação que não tenha sido guardada anteriormente com a opção *Save Session*.

8 Conclusões

Após a conclusão deste trabalho verificamos que do que ficou implementado, segundo os parâmetros que apresentamos anteriormente, tudo aparenta estar a funcionar, recebendo apenas números quando deve receber números, rejeitando letras, rejeitando linhas em branco, efetuando transições adequadas entre menus, tal como solicitado pelo utilizador e executando as funcionalidades devidas de acordo com os pedidos recebidos. Salvo algum caso não testado, no geral, a aplicação aparenta estar operacional e consideravelmente robusta. No entanto, não conseguimos otimizar o código, nomeadamente em questões como métodos de pesquisa na classe UMeR, que poderiam estar mais concisos/eficientes e, possivelmente, até de mais fácil compreensão; os veículos poderiam ser mais diferenciados, sendo que neste momento são muito parecidos exceto apenas o número de lugares e velocidade média atribuída pelo construtor mais usado.

Mais do que as melhorias indicadas, ficaram por implementar algumas funcionalidades avançadas. Questões como veículos com fila de espera poderiam ser implementadas criando novas classes, por exemplo *CarWithQueue.java*, para os mesmos que em tudo se assemelhariam às classes existentes, sendo inclusive extensões das classes abstratas, no entanto, incluindo uma *List*, possivelmente *ArrayList* com um limite de 10, para garantir que existe uma fluidez no serviço e não existem tempos exagerados de espera. Juntamente à variável seriam necessários os métodos de verificação se um dado veículo é do tipo que suporta fila de espera, adição de clientes à fila de espera e remoção, caso fosse dada a possibilidade de desistir da reserva. Outra funcionalidade não implementada foi as Empresas de Taxis. Para estas, poderíamos implementar uma classe *EmpresasTaxi.java*, que teria apenas o nome da empresa, o seu NIF, lista dos seus veículos e lista de motoristas. Estas empresas iriam permitir que os motoristas trocassem de veículo, efetuando a troca da associação dos *key* e *values* do *Map*, onde se encontram associados os veículos e os motoristas. Cada empresa constaria numa nova estrutura de dados da UMeR, para manter o registo das mesmas. Externamente às funcionalidades solicitadas pretenderíamos dar mais liberdade aos utilizadores para alterarem os seus dados, bem como facilitar obter informação sobre veículos, mais especificamente, implementaríamos uma lista de favoritos para que o cliente pudesse guardar os seus motoristas preferidos.