Programação Orientada aos Objetos

Projeto UMeR - Grupo 60

Diogo Araújo
A78485
a78485@alunos.uminho.pt

Diogo Nogueira
A78957
a78957@alunos.uminho.pt

03/06/2017

Conteúdo

Introdução

Este projeto foi feito no âmbito da Unidade Curricular de Programação Orientada de Objetos, tomando como ideia principal um serviço de transporte de passageiros autónomo, intitulado de UMeR, com utilizadores/clientes e diversas viaturas, como Táxis, Monovolumes e até Motociclos, em que os tais serão conduzidos por Motoristas, que é a outra base do serviço UMeR.

Baseia-se num serviço através da localização do cliente e a localização das viaturas que calcula a partir daí o tempo, preço e distância que a viagem feita através do UMeR irá demorar/custar, dado que cada tipo de Viatura tem diferentes valores estimados para *velocidade por km* e também *preço por km*.

Neste relatório iremos descrever o trabalho que foi realizado para desenvolver o serviço UMeR, mais concretamente uma descrição da arquitetura de classes que utilizamos, incluindo algumas das decisões que foram tomadas para desenvolver a aplicação da maneira que está apresentada. Também irá acontecer uma descrição da aplicação com ilustrações de algumas das funcionalidades para exemplificar melhor as várias formas de interagir com todo o serviço e aplicação UMeR.

Arquitetura das Classes

Todo este projeto foi pensado de forma a que tivesse em mente todo o paradigma de *Programação Orientada a Objetos* dado que usamos uma estratégia para que o nosso código esteja modularizado e invocável pelo exterior de forma a dividir tarefas e procedimentos consoante a classe em si. Isso fez com que o código mantivesse um escapsulamento de todas as variáveis de instância, incluindo *Maps e Lists* que estão dentro de classes que contêm os métodos necessários para controlar, criar e modificar estas instâncias do **Java Collections Framework**.

2.1 IDGeral

A classe **IDGeral** é a classe abstrata que contém a informação genérica dos utilizadores do serviço UMeR. Utilizamos uma herança de classes em que a *superclasse* é abstrata, uma vez que não temos como objetivo instanciar um objeto da mesma.

Esta classe é apenas usada como ”molde”para as suas *subclasses* **Cliente** e **Motorista** que herdam da mesma todo o seu comportamento e entidade, facilitando assim toda a transferência de informação entre a *superclasse* e as suas *subclasses*. Sendo assim, os métodos **clone()**, **toString()** e **equals()** desta *superclasse* são definidos como abstratos, obrigando as subclasses a implementar estes métodos para estarem disponíveis nas suas instâncias futuras.

Esta decisão visa dar uma maior consistência ao nosso código, contornando a desnecessária repetição do mesmo.

2.1.1 Cliente

A classe **Cliente** é uma das subclasses da classe **IDGeral** e define o Cliente do serviço UMeR.

Herda portanto todos os métodos da classe **IDGeral** adicionando as seguintes variáveis de instância:

- **Localização** - para esta variável recorremos a uma classe da API do Java. A classe é a **Point2D.Float** (`https://docs.oracle.com/javase/8/docs/api/java/awt/geom/Point2D.html`) que permite a criação de um Point2D com coordenadas específicas. Procedemos a esta escolha pois esta classe já traz consigo métodos que acabaram por nos facilitar em alguns requisitos como por exemplo no **cálculo das distâncias**, reduzindo, dessa

forma, linhas de código, tornando-o mais compacto e mais fácil de se compreender.

- **Histórico** - para o histórico das Viagens do Cliente usamos um *ArrayList* que reúne todas as Viagens efetuadas pelo mesmo até ao momento.

2.1.2 Motorista

A classe **Motorista** é outra das *subclasses* da classe **IDGeral** e define o Motorista do serviço UMeR.

Herda igualmente todos os métodos da classe **IDGeral** adicionando também um histórico das Viagens do Motorista em questão, tendo ainda outras variáveis e métodos essenciais para a gestão de alguns dos requisitos do projeto e característicos deste tipo de serviço, como por exemplo a disponibilidade do Motorista.

2.2 Viatura

A classe **Viatura** é a classe que define a Viatura do serviço UMeR. Ao contrário do que acontece com a classe **IDGeral**, esta classe não é abstrata. A classe não é abstrata porque no código cria-se instâncias de Viatura para facilitar a programação, mesmo que na prática existirá verificações com o *instance of* para poder fazer o *cast* necessário das suas subclasses. Assim, nesta *superclasse* **Viatura** temos apenas definidos os métodos e atributos que são comuns a todas as Viaturas.

Deste modo, temos uma hierarquia de classes entre a *superclasse* e as *subclasses* (que iremos citar a seguir), criando classes mais específicas.

2.2.1 Taxi, MonoVolume e Moto

As classes **Taxi, MonoVolume e Moto** são *subclasses* da classe **Viatura** e definem o tipo de Viatura associado ao serviço UMeR. São classes mais específicas que detalham o preço por cada km, bem como a velocidade média, de modo a que o Cliente possa tomar uma decisão baseada neste parâmetros no momento de requisitar uma Viagem.

Estas variáveis de instância adicionais irão-nos ajudar no processo de calcular o preço e tempo que cada viagem toma, consoante o tipo/*subclasse* de **Viatura** que têm valores diferentes.

2.3 ConjuntoCliente e ConjuntoMotorista

As classes **ConjuntoCliente** e **ConjuntoMotorista** são classes que agrupam num *HashMap* todos os Clientes e Motoristas (respetivamente) inscritos no serviço UMeR. Estas classes têm como principal utilidade fazer o agrupamento de todos o que se vão inscrevendo na App, associando a cada um deles o seu email de modo a que todos possam ser facilmente identificados.

Estas duas classes têm métodos gerais para gerenciamento dos *HashMaps*, mas também métodos mais específicos consoante a necessidade da aplicação e a parte visual/*front-end* da aplicação UMeR.

Um desses métodos encontra-se escrito na classe **ConjuntoCliente** e é responsável por verificar se a password relativa a um determinado Cliente se encontra correta. Este método foi necessário para verificar se o Cliente está inscrito no serviço UMeR.

Eis o excerto do mesmo:

```
// ConjuntoCliente.java

    public boolean passwordCorreta(String eemail, String epass) {
        if (existeCliente(eemail))
            return (osclientes.get(eemail).clone()).getPassword().equals(epass);
        else return false;
    }
```

Este método recebe o email do Cliente bem como a password por si introduzida aquando do seu login, verificando numa primeira fase se existe algum Cliente com esse email e, finalmente, caso esse Cliente de facto exista, se possui a referida password.

2.4 ConjuntoViatura

A classe **ConjuntoViatura** é a classe que agrupa num *ArrayList* todas as Viaturas existentes no serviço UMeR. A única diferença em relação aos conjuntos supracitados é no tipo de coleção que usa, pois não tem a necessidade de agrupar as Viaturas consoante algo. Em relação aos métodos, implementa também os básicos e outros que são necessários para o normal funcionamento da App.

2.5 Viagem

A classe **Viagem** é a classe que define uma Viagem no serviço UMeR. Sendo assim, esta classe usa todas as classes que acima foram discutidas. Sem elas, a classe **Viagem** seria inútil e disfuncional uma vez que seria completamente impossível um determinado Cliente requisitar uma Viagem numa Viatura associada a um Motorista.

Por assim ser, tem contida nela métodos cruciais para a Viagem requisitada pelo Cliente, mais concretamente quando essa Viagem é solicitada requerindo uma Viatura em específico.

Nesta classe recorremos à classe **Duration** (`https://docs.oracle.com/javase/8/docs/api/java/time/Duration.html`) proveniente da API do Java que nos auxiliou no cálculo do tempo total da Viagem, convertendo o mesmo em minutos.

Veja-se o método que calcula o tempo da Viagem:

```
// Viagem.java

    public void calculaTempo(){
        if (viatura instanceof Taxi) {
            long temp = (long)((distanciaviagem/((Taxi) viatura).getVelocMedia())*60);
            this.tempo = Duration.ofMinutes(temp);
        }
        if (viatura instanceof Moto) {
            long temp = (long)((distanciaviagem/((Moto) viatura).getVelocMedia())*60);
            this.tempo = Duration.ofMinutes(temp);
        }
        if (viatura instanceof MonoVolume) {
            long temp = (long)((distanciaviagem/((MonoVolume) viatura).getVelocMedia())*60);
            this.tempo = Duration.ofMinutes(temp);
        }
    }
```

Este método verifica se a Viatura em questão é um Taxi, uma Moto ou uma MonoVolume através do operador **instanceof** e, após isso, calcula o tempo da Viagem tendo em conta a velocidade de cada Viatura.

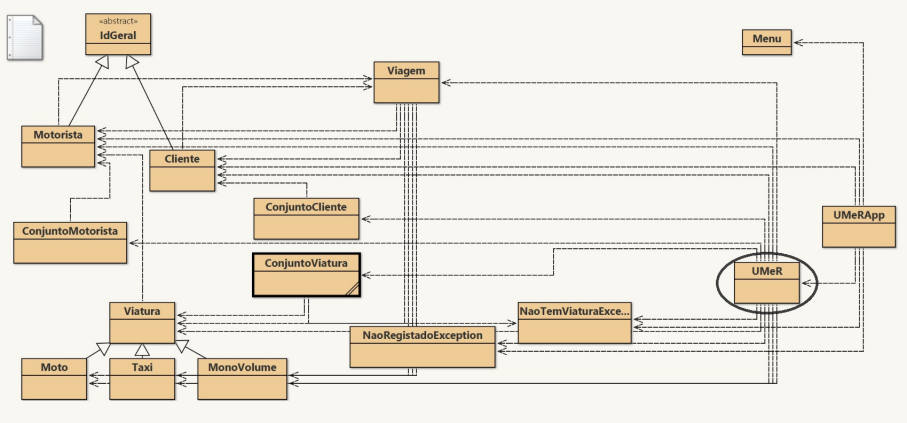
A classe **Duration** é invocada neste método quando queremos converter o tempo para minutos, usando para esse fim o método **ofMinutes(long seconds)**.

2.6 UMeR e UMeRApp

2.6.1 UMeR

A classe **UMeR** é a classe *back-end* que permite que todas as instâncias das diversas classes que acima referimos sejam efetivamente criadas. Esta classe possui métodos para alterar as mais diversas opções, sendo a responsável por conter métodos que dão respostas a todos os menus da classe **UMeRApp**, quer por parte do Cliente ou do Motorista.

Através do screenshot abaixo anexado é possível verificar-se a conexão desta classe UMeR com todas as outras existentes no nosso projeto (incluindo as exceções):



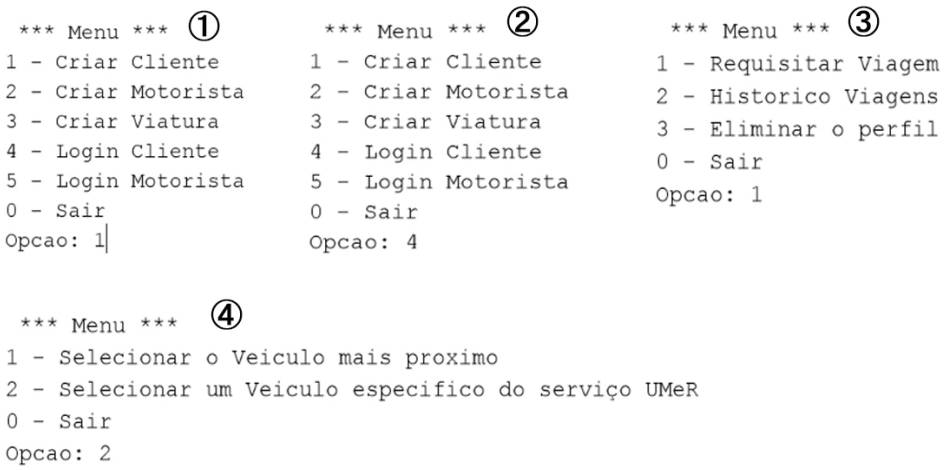
2.6.2 UMeRApp

A classe **UMeRApp** é a classe *front-end* que chama todos os métodos que a classe **UMeR** criou. Esta classe foi modificada tendo como base a classe disponibilizada pelo professor ***José Creissac Campos***.

Ilustração de algumas funcionalidades

Após todo o processo de criação e desenvolvimento das classes necessárias para o projeto **UMeR** e da sua integração com todos os menus, tornando todo o serviço da **UMeR** funcional apenas é preciso escolher, perante todas as opções do menu principal o que se pretende fazer. O menu principal contém as opções base para a possível conexão entre Cliente/Motorista.

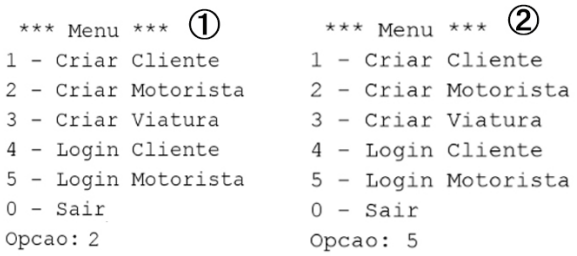
Vejamos assim algumas imagens que exemplificam o processo de registo e posterior login na App.
A imagem abaixo anexada mostra os passos a efetuar por parte de um Cliente recém-chegado ao serviço de modo a proceder à sua inscrição:

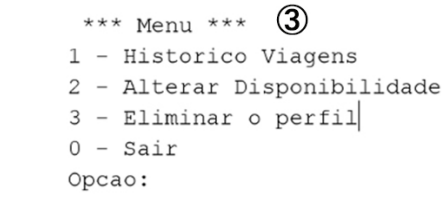


Estando no menu principal, o Ciente terá de selecionar a opção *"Criar Cliente"*. Serão pedidas as informações básicas para que seja possível o correto registo no serviço. Após o perfil do Cliente estar criado, o mesmo será redirecionado novamente para o menu principal onde poderá agora ingressar na opção

"Login Cliente". Introduzindo os dados pedidos corretamente terá acesso ao seu perfil onde terá a possibilidade de *"Requisitar uma Viagem"*, ver o *"Histórico Viagens"* já feitas e até *"Eliminar o perfil"*.

Caso o utilizador da App seja um Motorista, os passos para proceder à sua inscrição no serviço serão os mesmo que para o Cliente, variando apenas nas opções do menu corrospondente seu perfil:





Após se registar no serviço, precisará de efetuar o login na aplicação inserindo os seus dados corretamente de modo a ser redirecionado para o seu perfil, podendo aí ver o *"Histórico Viagens"*, *"Alterar Disponibilidade"* no momento, bem como *"Eliminar o perfil"*.

3.1 Conclusão

Fazendo uma análise crítica do projeto bem como do trabalho por nós efetuado para a conclusão do mesmo, podemos dizer que, perante toda a dificuldade que foi surgindo ao longo do desenvolver das classes, foi possível criar uma aplicação funcional e onde todos os métodos criados manobram bem entre si.

Além disso, conseguimos desenvolver um código modularizado, preservando o encapsulamento de todas as variáveis de instância.