Universidade do Minho

Escola de Engenharia

Licenciatura em Engenharia Informática

Unidade Curricular de

Programação Orientada a Objectos

Ano Letivo de 2016/2017

Trabalho Prático

UMer

Grupo 6

João Reis – 65194

Flávio Martins - 65277

Índice

1. Introdução e Contextualização

Este trabalho prático propõe uma primeira experiência ao mundo do java fazendo uma aplicação que pudesse ser usada numa empresa parecidíssima com uma bem conhecida de todos nós, a UBER. O projeto enquadra-se no seguinte contexto, retirado do enunciado fornecido:

"Uma empresa de alunos de POO pretende criar um serviço de transporte de passageiros que faça concorrência a um serviço muito conhecido (e que tem um nome muito parecido com UMeR...). Pretende-se que a aplicação a ser desenvolvida dê suporte a toda a funcionalidade que permita que um utilizador realize uma viagem num dos táxis da UMeR. O processo deve abranger todos os mecanismos de criação de utilizadores, motoristas, automóveis e posteriormente a marcação das viagens, a realização das mesmas e respetiva imputação do preço. Pretende-se também que o sistema guarde registo de todas as operações efetuadas e que depois tenha mecanismos para as disponibilizar (exemplo: viagens de um utilizador, extrato de viagens de um táxi num determinado período, valor faturado por um táxi num determinado período, etc.). Cada perfil de utilizador deve apenas conseguir aceder às informações e funcionalidades respetivas.

- Os clientes dos táxis UMeR poderão:
  - solicitar uma viagem ao táxi mais próximo das suas coordenadas;
  - solicitar uma viagem a um táxi específico;
  - fazer uma reserva para um táxi específico que, de momento, não está disponível.
- Os motoristas poderão:
  - sinalizar que estão disponíveis para serem requisitados;
  - registar uma viagem para um determinado cliente;
  - registar o preço que custou determinada viagem."

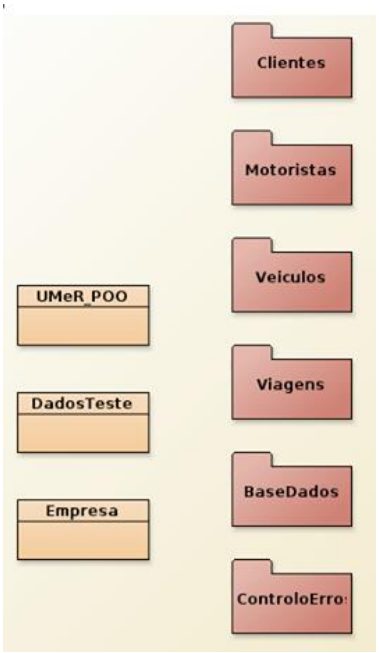
2. Arquitetura de Classes

Nesta secção vamos apresentar a arquitetura de classes da aplicação, sendo que fizemos uma subdivisão em packages para uma melhor organização e leitura da arquitetura. Em cada package contem as classes que lhe dizem respeito.

2.1 Pagina inicial BlueJ

Na pagina inicial encontra-se a classe principal UMerPOO, que também pode ser chamada main, visto que esta classe é responsável pela interface utilizador/aplicação. Esta importa os packages necessários para o desenrolar da aplicação, e como mostra na figura, podemos ver os seguintes packages:

- BaseDados;
- Clientes;
- Motoristas;
- Veiculos;
- ControloErro;
- Viagens;

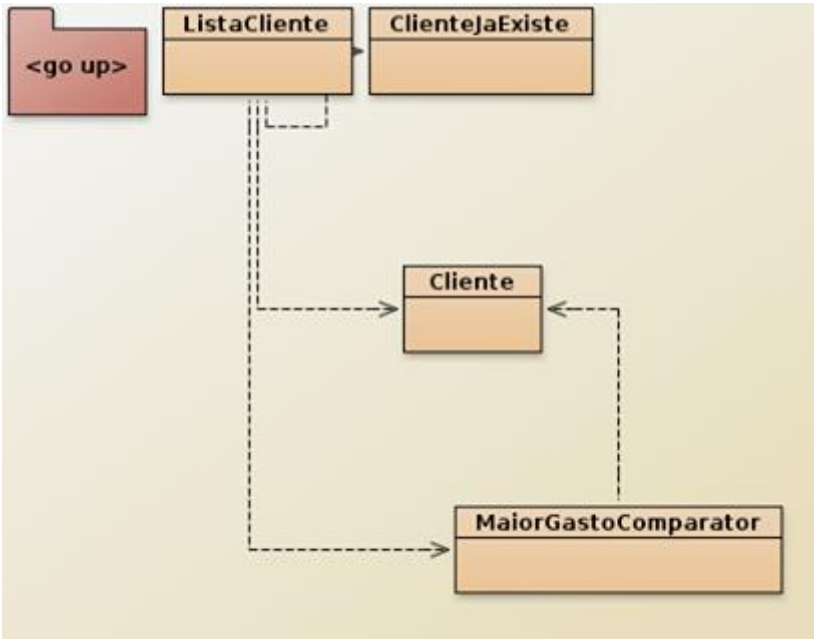


2.2 Package BaseDados

Este package aloca apenas a classe BaseDeDados que guarda todos os dados da aplicação, clientes, motoristas e veículos, bem como as viagens efetuadas.

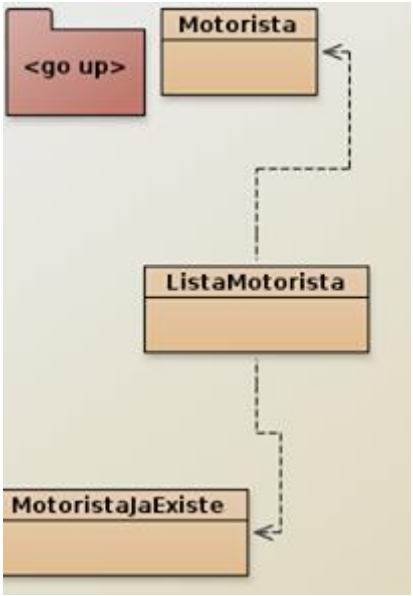
2.3 Package Clientes

Este package contem as classes referentes aos clientes. A classe Cliente aloca toda a informação e métodos relacionados com as várias ações do cliente. Os clientes e a sua informação são guardadas numa Lista de Clientes, e essa lista é guardada na BaseDeDados.



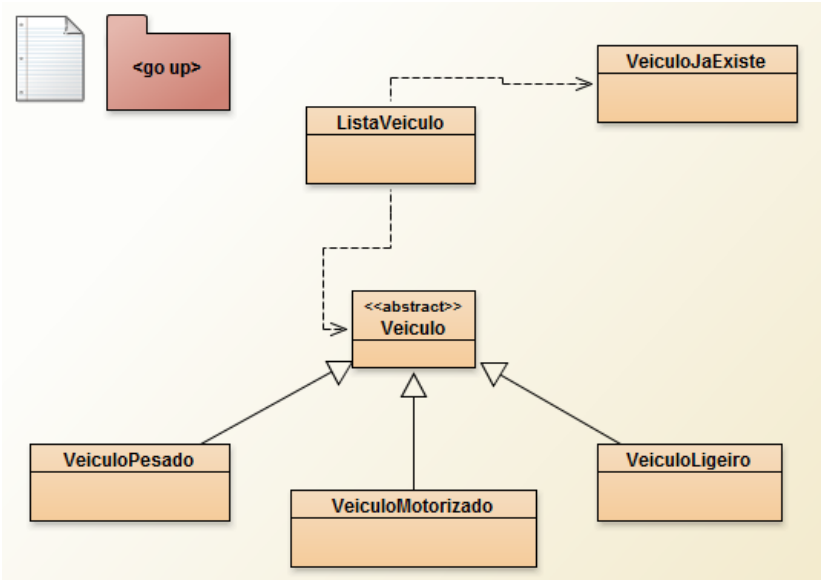
2.4 Package Motoristas

Neste package contem as classes referentes aos motoristas. Este package é muito parecido ao do cliente. Existe uma classe motorista que aloca a informação e ações de um motorista e outra classe Lista Motorista que guarda os motoristas e suas informações, que por sua vez é guardada na classe BaseDeDados.



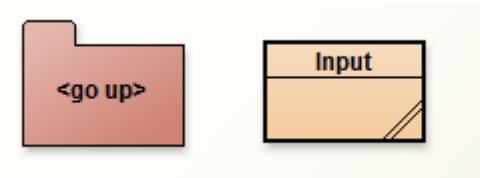
2.5 Package Veiculos

Neste package é alocado toda a informação relacionada com os veículos da aplicação. Neste caso, irão existir 3 tipos de veículos possíveis de registar na aplicação(Veiculo Motorizado, Veiculo Ligeiro, Veiculo Pesado). A informação dos veículos são guardados numa classe Lista Veiculos que posteriormente se integra na BaseDeDados.



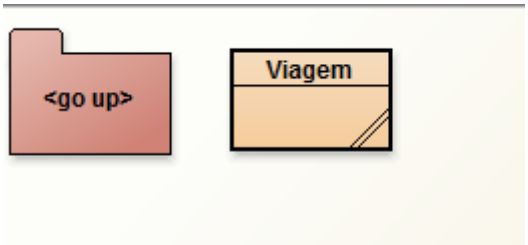
2.6 Package ControloErro

Este package tem a classe Input que é responsável pelos métodos de ler inteiros ou strings, bem como garantir que o campo data seja introduzido de forma correta pelo utilizador.



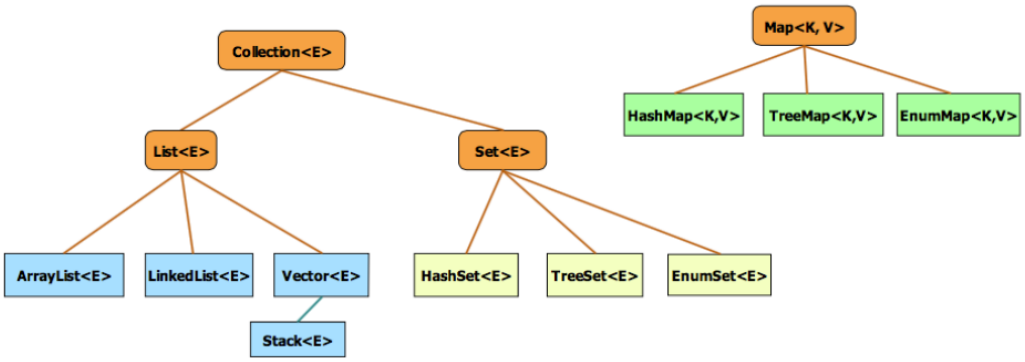
2.7 Package Viagens

Neste package encontra-se a classe Viagem, responsável por definir uma viagem de um cliente/motorista.



3. Decisões relativamente à estrutura das classes

Nesta seção iremos abordar as decisões tomadas relativamente as estruturas de dados implementadas nas classes.



3.1 Cliente

De seguida, apresentamos as variáveis de instância pertencentes à classe Cliente:

```
public class Cliente implements Serializable
{
    private String email;
    private String nome;
    private String password;
    private String morada;
    private GregorianCalendar dataNascimento;
    private int latitude;
    private int longitude;
    private TreeMap<GregorianCalendar, Viagem> registoViagens;
```

Utilizamos um TreeMap para guardar o histórico das viagens realizadas. A chave identificadora é a data da viagem, e no campo do valor é guardado as informações da viagem.

3.2 ListaCliente

```
public class ListaCliente implements Serializable{

    private HashMap<String,Cliente> clientes;
```

Os clientes são guardados numa estrutura do tipo HashMap em que a chave é o email do cliente.

3.3 Motorista

```
public class Motorista implements Serializable
{
    private String email;
    private String nome;
    private String password;
    private String morada;
    private GregorianCalendar dataNascimento;
    private int latitude;
    private int longitude;
    private int grauDeComprimentoHorario;
    private int classificacao;
    private int kmsRealizados;
    private boolean isFree;
    private TreeMap<GregorianCalendar, Viagem> registoViagens;
    private Veiculo veiculoQueEstaAConduzirActualmente;
```

Nesta classe utilizou-se um TreeMap para guardar o histórico das viagens e uma variável do tipo Veiculo para saber qual o táxi conduzido pelo motorista.

3.4 ListaMotorista

```
public class ListaMotorista implements Serializable {

    private HashMap<String,Motorista> motoristas;
```

Do mesmo género à classe ListaCliente.

3.5 ListaVeiculo

Semelhante ao ultimo ponto.

```
public class ListaVeiculo implements Serializable {

    private HashMap<String,Veiculo> veiculos;
```

3.6 Viagem

De seguida apresentamos as variáveis da classe Viagem.

```
public class Viagem implements Serializable
{
    private int duracao; // em minutos
    private int valor;
    private GregorianCalendar data;
    private int latitudeInicial;
    private int longitudeInicial;
    private int latitudeFinal;
    private int longitudeFinal;
```

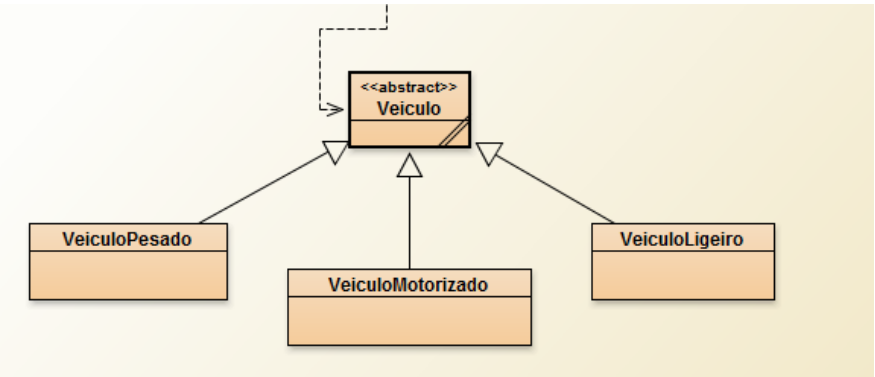
4. Outras Decisões

4.1 Hierarquia de Classes

Existe apenas uma hierarquia presente na nossa arquitetura é a dos veículos. Consideramos os seguintes atributos gerais a todos os veículos, apresentados na próxima imagem, desta forma há menos replicação de código e ganha uma melhor organização.
De seguida apresentamos os atributos gerais dos veículos:

```
public abstract class Veiculo implements Serializable
{

    private double velMedia;
    private double precoBasePorKm;
    private double factorFiabilidade;
    private String matricula;
```



5. ScreenShots da Aplicação

Neste capítulo iremos apresentar alguns screenshots da aplicação. Como se tornava numa missão muito difícil e trabalhosa não iremos mostrar todos os estados da aplicação apenas os mais importantes.

Opções

```
======================================================================
----------------------- UMeR  POO --------------------------------
                  Bem-vindo caro Cliente!
======================================================================


--------------------------
1 - Registar Novo Cliente
2 - Registar Novo Motorista
3 - Registar Novo Veiculo
4 - Efectuar Login Cliente
5 - Efectuar Login Motorista
6 - Lista Clientes Mais Gasto
0 - Sair
--------------------------
Opção:
0
Obrigado, volte sempre
```

```
-----------------------------
MENU VIAGEM
-------------------------
1 - Chamar taxi mais proximo
2 - Chamar taxi por um motorista em particular
0 - Sair
------------------
Opção:
1
----------------------
REALIZAR VIAGEM
----------------------
Insira a sua posição atual:
Latitude:
100
Longitude:
100
Insira a posição final:
Latitude:
83
Longitude:
89
Quantos passageiros?
3
-------------------------------
Deseguida apresentamos o valor da viagem:
Valor a pagar ate o taxi chegar a si:556.0
Valor a pagar desde a sua posição até ao destino:80.99382692526635
VALOR TOTAL:636
-----------------------------------
-----------------------------------
Deseja realizar a sua viagem?
1 - Sim
2 - Não
1
Viagem registada
```

6. Conclusão

De seguida a apresentação inicial, foi-nos proposto pelos docentes realizar este projeto com uma dimensão média/grande utilizando os princípios da Programação Orientada a Objetos, reutilização, encapsulamento, herança, etc. A parte fundamental da realização do projeto foi planear a estrutura da aplicação antes do grupo por a mãos, ou seja, escrever o código. Apesar do grupo não estar totalmente satisfeito com resultado final, cremos que aplicamos os conceitos lecionados em Programação Orientada a Objetos e, por consequência, desenvolvemos as nossas capacidades a programar neste paradigma em Java.