Escola de Engenharia
Universidade do Minho

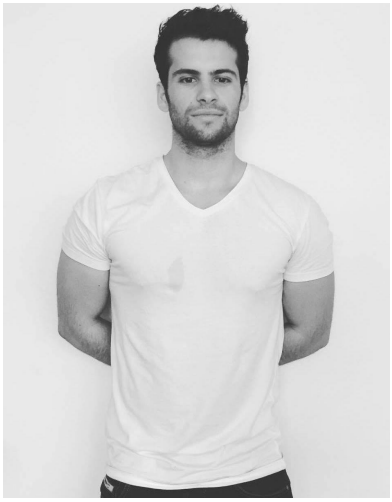
DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
Mestrado Integrado em Engenharia Informática
Programação Orientada aos Objetos

UMeR

Serviço de transporte de passageiros

Grupo 07

Célia Figueiredo
a67637



José Carlos Faria
a67638

Márcia Costa
a67672

Braga, 28 de Maio de 2017

Conteúdo

Resumo

O presente relatório descreve o trabalho efetuado para a realização do projeto, onde foi pedida a implementação de um serviço de transporte de passageiros (UMeR) com o uso da linguagem *JAVA* esta que é orientada aos objetos.

1. Introdução

No âmbito da Unidade Curricular de Programação Orientada aos Objectos pertencente ao plano de estudos do 2º ano do Mestrado Integrado em Engenharia Informática foi proposto o desenvolvimento de um serviço de transporte de passageiros.

2. Descrição geral do projeto

2.1 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 respectiva imputação do preço. Pretende-se também que o sistema guarde registo de todas as operações efectuadas e que depois tenha mecanismos para as disponibilizar (exemplo: viagens de um utilizador, extracto de viagens de um taxi num determinado período, valor facturado por um taxi num determinado período, etc.).

Cada perfil de utilizador deve apenas conseguir aceder às informações e funcionalidades respectivas.

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

2.1.1 Actores do sistema

Existirão dois tipos distintos de actores no sistema, que partilham a seguinte informação:

- email (que identifica o utilizador);
- nome;
- password;
- morada;
- data de nascimento.

Cliente

O Cliente representa a pessoa que solicita e efectua uma viagem de táxi. O cliente está sempre numa determinada localização (expressa em x e y, isto é, num espaço 2D) e escolhe um táxi específico ou então solicita o táxi mais perto que esteja disponível. O cliente tem também uma relação de todas as viagens que fez, com toda a informação relativa à viagem.

Motorista - colaborador da UMeR

O motorista conduz o táxi e além da informação atrás referida tem também dados relativos a:

- grau de cumprimento de horário estabelecido com o cliente, dado por um factor entre 0 e 100;
- classificação do motorista, dado numa escala de 0 a 100, calculada com base na classificação dada pelo cliente no final da viagem;
- histórico das viagens realizadas;
- número de kms já realizados na UMeR;
- informação sobre se está ou não disponível em determindado momento, isto é, se está ou não a trabalhar.

2.1.2 Os táxis UMeR

O ecossistema do UMeR contempla diferentes tipos de viaturas de aluguer (táxis). Neste momento estão em funcionamento os seguintes tipos de viaturas:

- carros ligeiros;
- carrinhas de nove lugares;
- motos.

Cada um destes tipos de viaturas tem associada:

- uma velocidade média por km;
- um preço base por km;
- um factor de fiabilidade, que determina a capacidade da viatura cumprir o tempo acordado com o cliente. Sempre que se realiza uma viagem é calculado (através da invocação de um random()) a capacidade de o veículo cumprir com o tempo acordado com o cliente. Este factor tem um efeito multiplicador sobre o tempo fornecido ao cliente.

Existem ainda alguns tipos de viaturas que possibilitam a existência de uma fila de espera de marcações. Quando o táxi não está disponível (por exemplo, pelo facto do condutor estar fora do horário de trabalho) é possível para essas viaturas aceitarem reservas de clientes. As reservas serão satisfeitas por ordem de chegada. Uma viatura sabe sempre a localização (em x e y) onde está. Quando realiza um serviço desloca-se para as coordenadas indicadas pelo cliente e fica aí parado até que seja solicitado um novo serviço.

2.1.3 Fazer uma viagem no UMeR

O processo de fazer uma viagem no UMeR segue as seguintes regras:

1. o cliente indica as coordenadas x e y em que se encontra;
2. o cliente decide se pretende chamar um táxi específico ou então solicitar o que está mais próximo;
3. por uma questão de simplificação, os UMeR deslocam-se sempre em linha recta pelo que o cálculo da distância entre o cliente e o táxi é feito pela cálculo da distância euclidiana (exemplo: se o cliente estiver em (0,0) e o táxi em (2,2) a distância é de 2.8284 kms);
4. após ser calculada a distância consegue-se saber, dadas as características do táxi, quanto tempo demora a chegar ao cliente e depois ao destino que o cliente solicita;
5. o táxi indica ao cliente qual o custo estimado da viagem, tendo em conta o deslocamento que é necessário efectuar, e o tempo total de viagem;
6. de acordo com a fiabilidade do carro (e de outros factores que pode considerar: a destreza do condutor, as condições metereológicas, etc.) é calculado o tempo real da viagem. Se a diferença for superior a 25% do tempo estimado, então o preço a cobrar é o combinado com o cliente. Se a diferença for igual ou inferior a 25% o valor é ajustado para o valor real em função do tempo decorrido;
7. o táxi fica no ponto definido como fim da viagem à espera de nova solicitação de serviço;
8. após a viagem o cliente pode dar uma nota ao motorista e fica com o documento relativo à viagem guardado na sua área pessoal.

2.1.4 Motoristas individuais vs Empresas de Táxi

Numa primeira fase o UMeR foi pensado para condutores que conduziam a sua viatura e fazem serviço de transporte de clientes. No entanto, e devido ao sucesso do negócio, foram criadas empresas que possuem várias viaturas (em teoria de diversos tipos) e que empregam vários motoristas. A gestão que é feita destas empresas implica que após a criação da empresa se possam adicionar viaturas e motoristas, e que uma mesma viatura possa ser conduzida por motoristas diferentes (em tempos diferentes).

2.1.5 Viaturas com Fila de Espera

O UMeR possibilita que algumas viaturas possam ser definidas como possuindo uma fila de espera, que possibilita que quando a viatura está indisponível os pedidos sejam adicionados a uma fila de espera. Quando o veículo fica disponível são executadas todas as viagens inseridas na fila de espera, pela ordem de chegada. Os veículos com fila de espera possibilitam este comportamento, sendo que os demais não exibem este comportamento e nessa situação se a viatura estiver indisponível não será candidata a efectuar viagens.

3. Arquitetura das Classes

Neste capítulo falaremos do esqueleto da aplicação UMeR, serão abordadas as classes presentes na aplicação, assim como os atributos e funcionamento de cada uma.

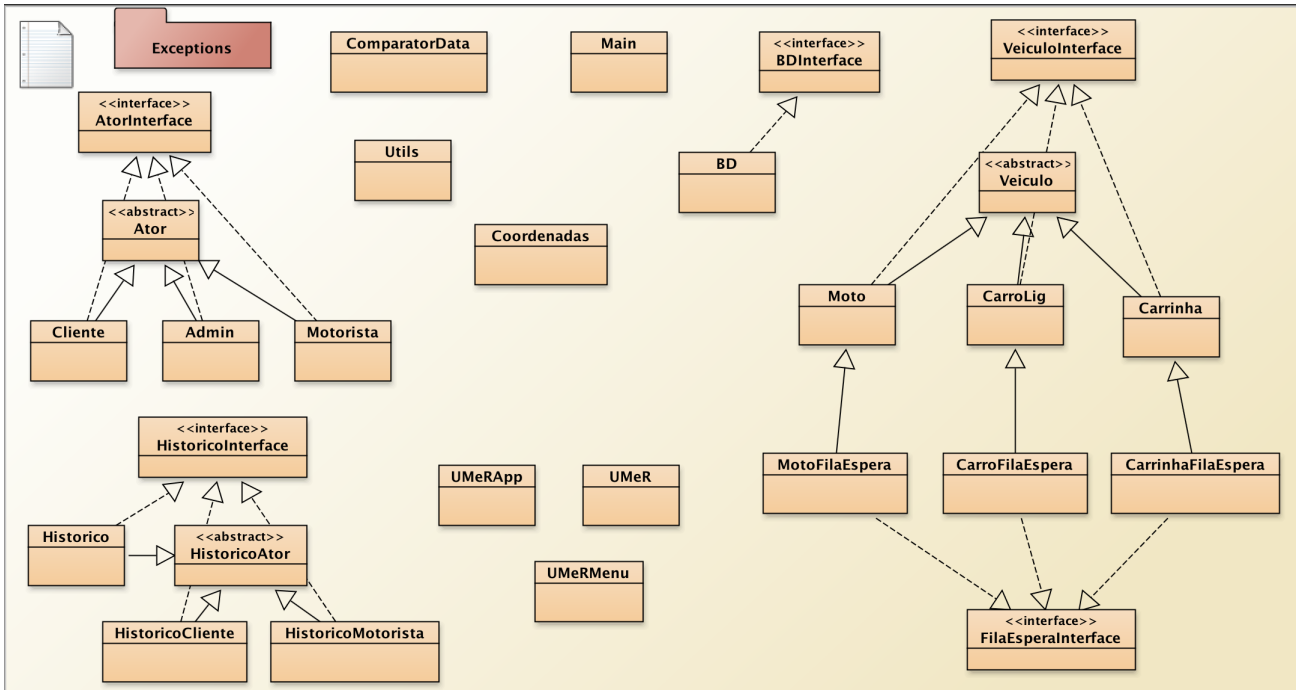


Figura 3.1: Classes

3.1 Ator

A classe *Ator* é uma classe abstrata e sevirá como "modelo" para outras classes que dela herdem, não podendo ser instanciada por si só. Para ter um objeto de uma classe abstrata é necessário criar uma classe mais especializada que herda dela e então instanciar essa nova classe. Neste caso foram criadas as classes *Admin*, *Cliente* e *Motorista*.

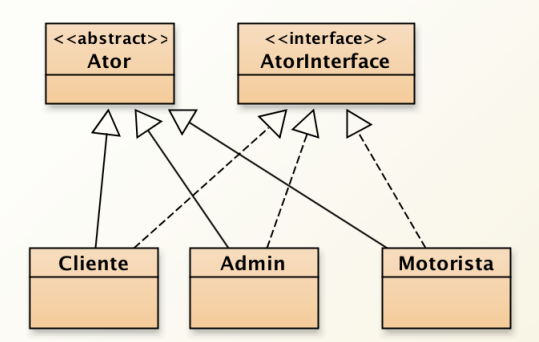


Figura 3.2: Classes

As variáveis de instância da classe abstrata *Ator* são apresentadas de seguida:

```
private String email;
private String nome;
private String password;
private String morada;
private LocalDate dataNascimento;
```

3.1.1 Admin

Na classe *Admin* estarão todos os dados herdados da classe *Ator*.

3.1.2 Cliente

Na classe *Cliente* estarão todos os dados herdados da classe *Ator*.

```
private Coordenadas loc; //localização atual do cliente
private HistoricoCliente histClie;
```

3.1.3 Motorista

Na classe *Motorista* estarão todos os dados herdados da classe *Ator*.

```
private int grauCumprimentoHorario; //0-100
private int classificacao; //0-100
private double totalKms;
private boolean disponivel;//verifica se está disponivel ou não
private boolean horarioTrabalho; //verificar se está no horário de trabal
private double destreza; //valor entre 0,5 e 1.9
private VeiculoInterface veiculo;
private HistoricoMotorista histMoto;
```

Decidimos que a destreza do motorista seria atribuida através da invocação de um random() que gera valores entre 0,5 e 1,9 afim de gerar alguma aleatoriedade nos tempos obtidos das viagens efetuadas.

```
this.destreza = Utils.generateRandom(0.5f, 1.9f);
```

3.2 Veiculo

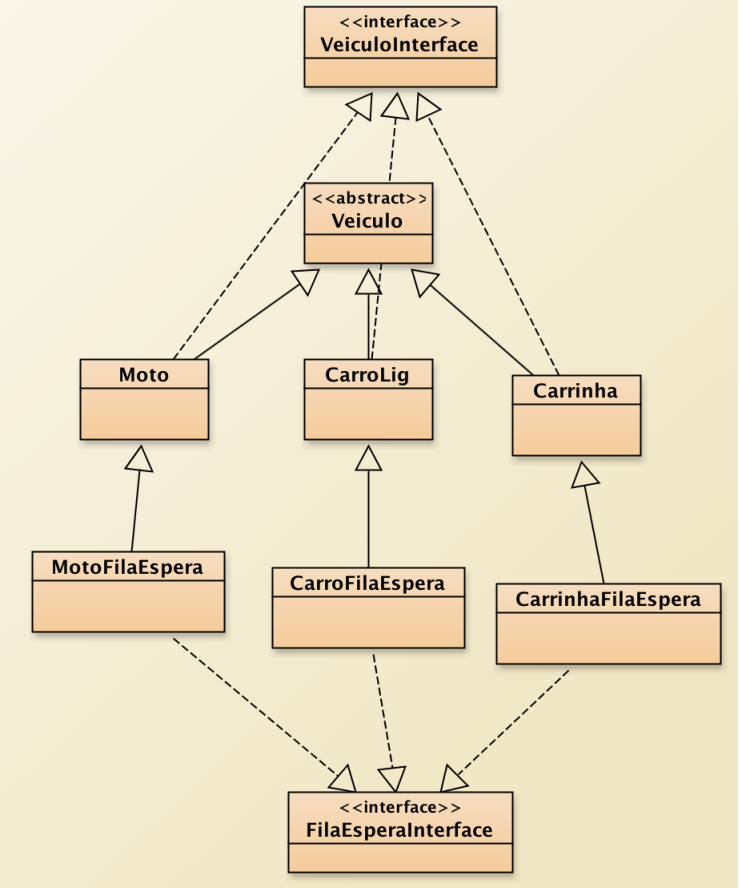


Figura 3.3: Classes

```
private String matricula;
private String marca; //Acrescentou-se a variável de instância marca,
para o cliente poder escolher um carro com base na marca do veiculo

private float fiabilidade;//0 a 2 randon()
private Coordenadas loc;
```

3.3 Moto

```
private static final int lugaresLivres = 1;
private static final double vm = 40.5;
private static final double precoPorKm = 3.1;
```

3.3.1 MotoFilaEspera

```
private List<Cliente> filaClientes;
```

3.4 CarroLig

```
private static final int lugaresLivres = 4;
private static final double vm = 65;
private static final double precoPorKm = 4.1;
```

3.4.1 CarroFilaEspera

```
private List<Cliente> filaClientes;
```

3.5 Carrinha

```
private static int lugaresLivres = 8;
private static final double vm = 55;
private static final double precoPorKm = 5.1;
```

3.5.1 CarrinhaFilaEspera

```
private List<Cliente> filaClientes;
```

3.6 Historico

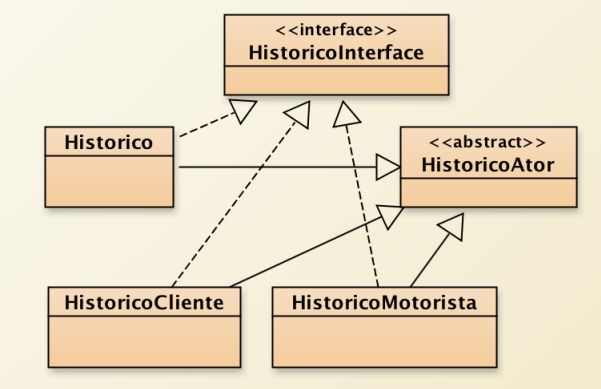


Figura 3.4: Classes

3.7 Utils

Adicionalmente a classe *Utils* tem implementado um método que encripta a password. E um método que gera números random com intervalos de 0.1.

3.7.1 Meteorologia

```
public static final String sol = "Sol";
public static final String nevoeiro  = "Nevoeiro";
public static final String granizo = "Granizo";
public static final String chuva = "Chuva";
public static final String neve = "Neve";
```

3.7.2 Trânsito

```
public static final String st = "Sem Transito";
public static final String tn = "Transito Normal";
public static final String mt  = "Muito Transito";
```

3.8 Coordenadas

```
private double x;
private double y;
```

As cordenadas também são iniciadas com o método random().

```
this.x=Utils.generateRandom(0f, 100f);
this.y=Utils.generateRandom(0f, 100f);
```

O método getDistancia() calcula a distancia euclidiana, este será um método importante na execução da simulação de uma viagem.

```
public double getDistancia (Coordenadas c){
    double distancia=0;
    distancia = Math.sqrt( Math.pow( (this.x - c.getX()),2 ) +
                                        Math.pow( (this.y - c.getY()),2 )
return distancia;
}
```

3.9 BD

```
private Map<String, AtorInterface> clientes;
private Map<String, AtorInterface> motoristas;
private Map<String, AtorInterface> admins;
private Map<String,VeiculoInterface> veiculos;
private Set<Historico> historico;
```

3.10 UMeRMenu

```
private String titulo;
private List<String> opcoes;
private int op;
```

3.11 UMeR

```
private BDInterface baseDeDados;
private AtorInterface atorLoggado;
private int tentativasDeLoginFalhadas;
private Map<String, AtorInterface> atores;
```

3.12 UMeRApp

```
private static UMeR umer;
private static UMeRMenu menu_principal;
private static UMeRMenu menu_registar_atores;
private static UMeRMenu menu_motorista;
private static UMeRMenu menu_cliente;
private static UMeRMenu menu_dados_pessoais;
private static UMeRMenu menu_cliente_efetuarViagem;
private static UMeRMenu menu_admin;
private static UMeRMenu menu_registar_veiculos;
private static UMeRMenu menu_solicitarViagem;
private static UMeRMenu menu_inserir_coord_destino;
private static UMeRMenu menu_terminar_viagem;
private static UMeRMenu menu_terminar_horario_trabalho;
private static UMeRMenu menu_iniciar_horario_trabalho;
private static UMeRMenu menu_proposta_viagem;
```

4. Funcionamento da Aplicação UMeR

Esta é uma aplicação com uma interface para o utilizador muito simples, foi pensada de maneira a que o utilizador pudesse tirar o maior proveito da mesma, com comandos simples, tendo em conta que todos os menus funcionam à base de opções por números. Quando um utilizador executa a aplicação o primeiro menu a que estás sujeito é o seguinte:

5. Conclusão