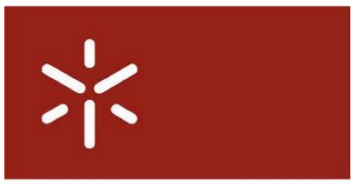


Universidade do Minho

Universidade do Minho

Relatório do Trabalho Prático POO

Mestrado Integrado em Engenharia Informática

Grupo 46

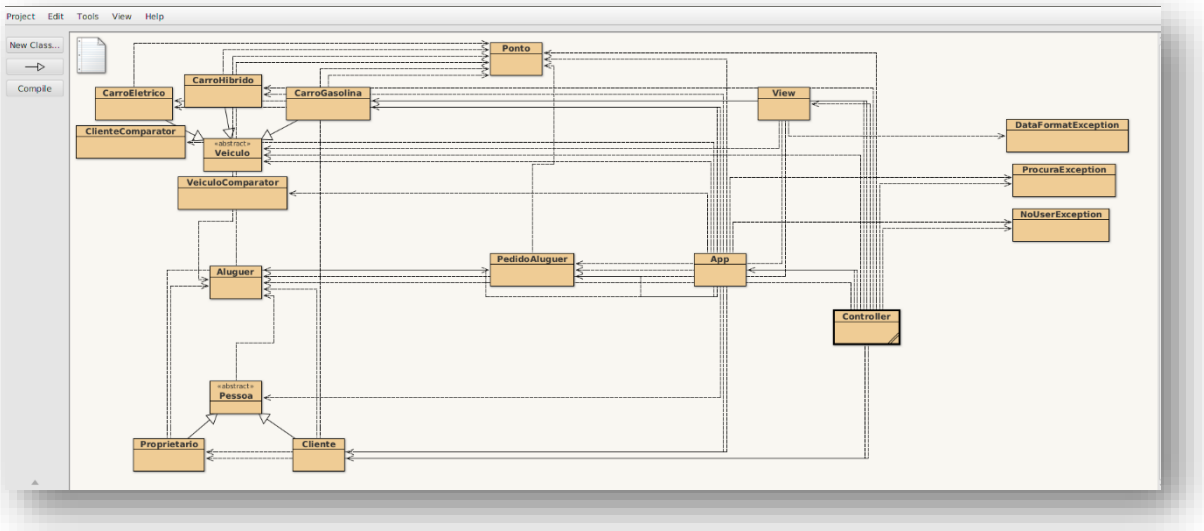
André Morais	A83899
Luís Ribeiro	A85954
Pedro Rodrigues	A84783

Introdução

O trabalho realizado foi desenvolvido em linguagem *Java* e teve como principal objetivo a gestão de uma Empresa de aluguer de carros. Para a concretização do objetivo, tivemos que definir várias Classes e métodos, as quais vamos abordas mais profundamente.

Inicialmente, começamos por criar as SuperClasses Veículo e Pessoa, por sua vez, abstratas. Mais tarde, definimos as suas subClasses. Ao longo do tempo fomos melhorando o programa, tendo em conta as necessidades deste.

O objetivo deste relatório é justificar as nossas escolhas e também explicar as estratégias usadas para o desenvolvimento do programa.



Esquema das nossas Classes

Pessoa

Inicialmente, definimos os *get's* e os *set's* das variáveis de instância que a classe Pessoa contêm. Posteriormente, declaramos alguns métodos abstratos que vão ser implementados e definidos nas suas subclasses.

```java
public abstract Pessoa clone ();

public abstract void registaAluguer(Aluguer a);

public abstract void classifica(int classi);

public abstract List<Aluguer> getHistorial ();

public abstract List<Aluguer> alugueresEntreDatas (LocalDate inicio, LocalDate fim);
```

A classe Pessoa tem como subclasses:

- Cliente – Para além dos métodos usuais, *get's* e *set's,* definimos novas variáveis de instâncias que vão definir o Cliente e os métodos abstratos antes criados.
- Proprietário – Na classe Proprietário, a abordagem é idêntica à do Cliente visto que têm quase as mesmas variáveis de instância, diferindo apenas nos métodos "*aceitaAluguer*" e "*registaAluguer*".

Veículo

A superclasse Veículo não difere muito da classe Pessoa, pois este apenas serve para definir as variáveis e respetivos construtores. Este, contempla vários tipos de veículos de aluguer:

- Veículo a Gasolina
- Veículo Híbrido
- Veículo Elétrico

O desenvolvimento destas subclasses foi semelhante, visto que, apenas se definem construtores*, equals*, *toString* e o *clone*

```java
public class CarroEletrico extends Veiculo
{
    public CarroEletrico(){
        super();
    }

    public CarroEletrico(String marca, String matricula, int nif,double velomed, double precokm,double consumokm, Ponto posicao, double autonomiaMax,double autonomia)
    {
        super(marca,matricula,nif,velomed,precokm,consumokm,posicao,autonomiaMax,autonomia);
    }

    public CarroEletrico(CarroEletrico e){
        super(e);
    }

    public CarroEletrico clone ()
    {
        CarroEletrico c = new CarroEletrico (this);
        return c;
    }

    public boolean equals(Object obj)
    {
        if (obj==this) return true;
        if (obj==null || obj.getClass()!=this.getClass()) return false;
        CarroEletrico c = (CarroEletrico) obj;
        return this.getMatricula().equals(c.getMatricula());
    }

    public String toString()
    {
        StringBuilder sb = new StringBuilder ();
        sb.append("Tipo de Veiculo: Carro Eletrico").append("\nMarca:").append(this.getMarca()).append("\nMatricula:")
            .append(this.getMatricula()).append("\nNif:").append(this.getNif()).append("\nVelocidade media: ")
            .append(this.getVelomed()).append("\nPreço por km: ").append(this.getPrecokm()).append("\nConsumo per km: ")
            .append(this.getConsumokm()).append("\nPosiçao: ").append(this.getPos().toString()).append("\nPercentagem do deposito: ")
            .append(this.getPercDep()).append("%").append("\nClassificacao: ").append(this.getClassificacao()).append("\nNumero de votos: ").append(this.getVotos());
        return sb.toString();
    }
}
```

Aluguer

As classes *Aluguer* e *PedidoAluguer* implementam as variáveis necessárias para um aluguer. Estas duas são classes simples, onde apenas servem para criar as variáveis, tendo em conta os seus construtores, os *get's* e *set's.*

```java
public class Aluguer implements java.io.Serializable
{
    private int nifCliente;
    private int nifProp;
    private String matricula;
    private Ponto origem;
    private Ponto destino;
    private double custo;
    private double distancia;
    private double duracao;
    private LocalDate data;
    private boolean clienteClassificou;


    public Aluguer (){
        this.nifCliente = 0;
        this.nifProp = 0;
        this.matricula = "";
        this.origem = new Ponto ();
        this.destino = new Ponto ();
        this.custo=0;
        this.distancia=0;
        this.duracao=0;
        this.data=LocalDate.now();
        this.clienteClassificou=false;
    }
```

```java
public class PedidoAluguer implements java.io.Serializable
{
    private int nifCliente;
    private Ponto partida;
    private Ponto destino;
    private String combustivel;
    private String preferencia;

    /**
     * Contstrutores da classe PedidoAluguer
     */

    /**
     * Construtor de inicializaçao
     */
    public PedidoAluguer()
    {
        // initialise instance variables
        this.nifCliente=0;
        this.partida=new Ponto ();
        this.destino= new Ponto();
        this.combustivel="";
        this.preferencia="";
    }
```

Menu (VIEW)

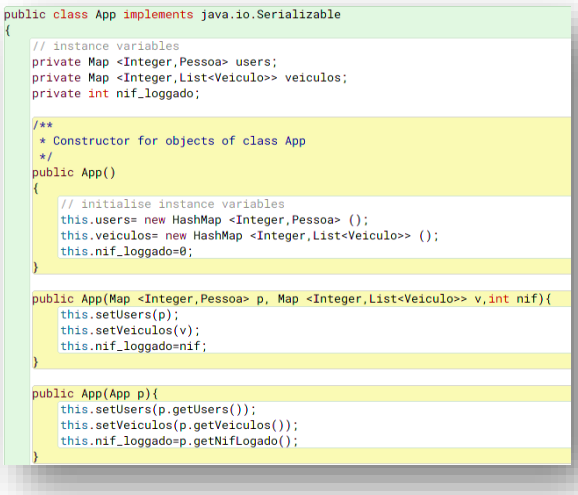
Esta classe teve como objetivo criar a interface gráfica textual do programa, onde fica mais fácil para o recetor poder interagir com o sistema. Todos os métodos implementados nesta classe, usam a classe *Scanner* para ler do *stndin* e facilitar a comunicação utilizador-sistema.
Para além da utilização do *scanner*, houve um uso recorrente de prints para fornecer as opções de escolha ao utilizador.

```
    ==> UmCarroJa! <==


[1] Entrar
[2] Registar


[0] Sair
```

```
        ==> Signup <==

Insira o tipo de conta:

  [1] Cliente
  [2] Proprietario
```

APP

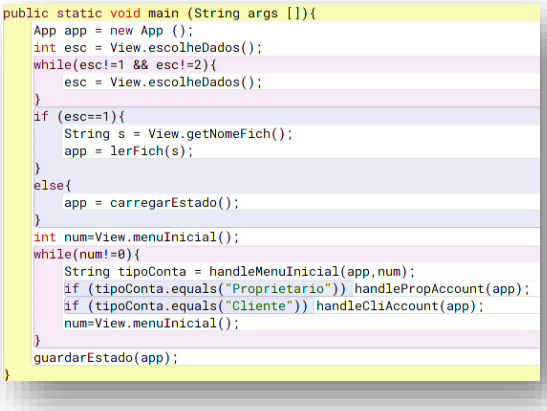
A Classe *App* é onde a magia acontece. Definimos bastantes métodos para manipular a base de dados.
No modelo MVC representa o model do projeto. É onde guardamos o estado da aplicação bem como todos os utilizadores registados, todos os veículos, todos os alugueres e todos os pedidos de alugueres no sistema.
É também nesta classe que executamos as linhas do ficheiro de *logs.*



```
public class App implements java.io.Serializable
{
    // instance variables
    private Map <Integer,Pessoa> users;
    private Map <Integer,List<Veiculo>> veiculos;
    private int nif_loggado;

    /**
     * Constructor for objects of class App
     */
    public App()
    {
        // initialise instance variables
        this.users= new HashMap <Integer,Pessoa> ();
        this.veiculos= new HashMap <Integer,List<Veiculo>> ();
        this.nif_loggado=0;
    }

    public App(Map <Integer,Pessoa> p, Map <Integer,List<Veiculo>> v,int nif){
        this.setUsers(p);
        this.setVeiculos(v);
        this.nif_loggado=nif;
    }

    public App(App p){
        this.setUsers(p.getUsers());
        this.setVeiculos(p.getVeiculos());
        this.nif_loggado=p.getNifLogado();
    }
```

Controller

É a classe que implementa a ligação entre o View e o Model, neste caso o *APP*.
Recebe a instrução requisitada pelo View e recorre ao Model para poder responder ao utilizador, enviando a resposta novamente para o View para que este a apresente posteriormente.



```
public static void main (String args []){
    App app = new App ();
    int esc = View.escolheDados();
    while(esc!=1 && esc!=2){
        esc = View.escolheDados();
    }
    if (esc==1){
        String s = View.getNomeFich();
        app = lerFich(s);
    }
    else{
        app = carregarEstado();
    }
    int num=View.menuInicial();
    while(num!=0){
        String tipoConta = handleMenuInicial(app,num);
        if (tipoConta.equals("Proprietario")) handlePropAccount(app);
        if (tipoConta.equals("Cliente")) handleCliAccount(app);
        num=View.menuInicial();
    }
    guardarEstado(app);
}
```

A função *main* inicializa o programa, carregando os dados, dando login e envia o tipo de conta para o respetivo controlador desse tipo de conta. No final guarda o estado da aplicação num ficheiro, para que depois possa ser retomado.

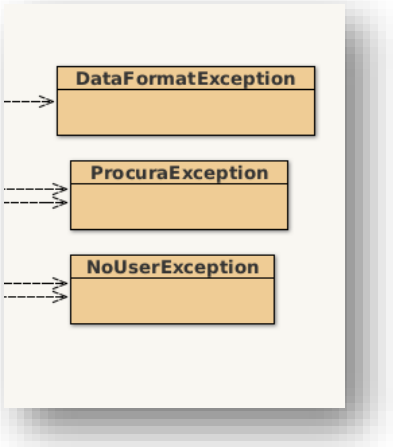
Comparators

Criamos duas Classes Comparators: o *ClienteComparator* e o *VeiculoComparator,* para ajudar a fazer uma Lista de Top 10 Clientes e uma Lista de Carros disponíveis.
O raciocínio em ambos os comparators são análogos. Definimos uma variável que pode tomar dois valores que nos permite distinguir os dois diferentes tipos de comparação.
Na Classe *ClienteComparator* podemos comparar os Clientes, ou por número de alugueres, ou por número de km's percorridos.
Na Classe *VeiculoComparator* podemos comparar os Veículos, por preço por km ou por distância veículo/cliente.

Exceptions

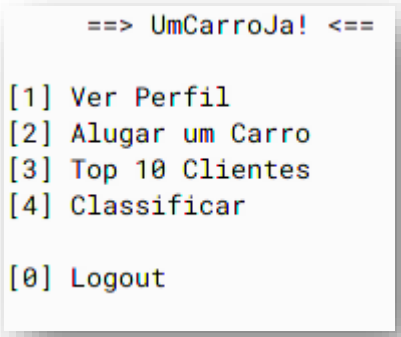
Criamos algumas Exceptions, como podemos observar na imagem em baixo. Estas servem para um maior controlo de erros e contínua funcionabilidade do programa.
Por exemplo, se não tivessemos a Classe *DataFormatException,* se nos fosse pedido uma data e não escrevessemos do tipo dd-mm-aaaa, o programa simplesmente fechava, daí a necessidade de criar este exception.

```
public class DataFormatException extends Exception
{
    public DataFormatException(){
        super();
    }

    public DataFormatException(String s){
        super(s);
    }
}
```



Conclusão

Concluído o trabalho, entendemos que houve uma aprendizagem notória relativamente aos conhecimentos adquiridos nesta Unidade Curricular. Utilizamos todos os elementos que nos foram lecionados nas aulas práticas, dos *getters e setters* até ao uso das Exceptions.



Quanto as funcionalidades do nosso programa, está tudo bem estruturado, iniciando o programa com um menu inicial onde o utilizador se pode registar, ou dar login. Depois temos um pequeno menu, quando já foi dado o login, onde tem várias opções como apresenta a imagem à direita.

Quanto à possibilidade de inserção de novas viaturas, basta ter o cuidado que nós tivemos em criar uma superclasse Veículos, onde cada tipo de veículo se insere numa subclasse deste.

No geral faz-se uma avaliação muito positiva das diversas métricas consideradas e entende-se que a abordagem ao problema foi concisa e direta.