|  |  |
| --- | --- |
| EENG | **Escola de Engenharia**  Departamento de Engenharia Informática  Mestrado Integrado em Engenharia Informática  Programação Orientada aos Objetos |

Relatório de Programação Orientadas aos Objetos

**ImOObiliaria**

**(Trabalho Prático de Java)**

Grupo 12:

João Martins (A68646)



João Pereira (A75273)



Índice

[Introdução 3](#_Toc450857841)

[API e arquitetura de classes 4](#_Toc450857842)

[Utilizador (classe abstrata) 4](#_Toc450857843)

[Atributos 4](#_Toc450857844)

[Construtores 4](#_Toc450857845)

[Métodos 5](#_Toc450857846)

[Comprador (especialização de Utilizador) 5](#_Toc450857847)

[Atributos 6](#_Toc450857848)

[Construtores 6](#_Toc450857849)

[Métodos 6](#_Toc450857850)

[Vendedor (especialização de Utilizador) 7](#_Toc450857851)

[Atributos 7](#_Toc450857852)

[Construtores 7](#_Toc450857853)

[Métodos 8](#_Toc450857854)

[Imovel (classe abstrata) 9](#_Toc450857855)

[Atributos 9](#_Toc450857856)

[Construtores 9](#_Toc450857857)

[Métodos 9](#_Toc450857858)

[Moradia (especialização de Imovel) 11](#_Toc450857859)

[Atributos 11](#_Toc450857860)

[Construtores 11](#_Toc450857861)

[Métodos 12](#_Toc450857862)

[Apartamento (especialização de Imovel) 13](#_Toc450857863)

[Atributos 13](#_Toc450857864)

[Construtores 13](#_Toc450857865)

[Métodos 13](#_Toc450857866)

[Loja (especialização de Imovel) 14](#_Toc450857867)

[Atributos 14](#_Toc450857868)

[Construtores 14](#_Toc450857869)

[Métodos 15](#_Toc450857870)

[LojaHabitavel (especialização de Loja) 15](#_Toc450857871)

[Atributos 16](#_Toc450857872)

[Construtores 16](#_Toc450857873)

[Métodos 16](#_Toc450857874)

[Terreno (especialização de Imovel) 17](#_Toc450857875)

[Atributos 17](#_Toc450857876)

[Construtores 17](#_Toc450857877)

[Métodos 18](#_Toc450857878)

[Módulo principal 18](#_Toc450857879)

# Introdução

O projeto **ImOObiliaria** foi desenvolvido em Java8 e consiste numa aplicação de gestão de imóveis. A aplicação admite 2 tipos de utilizadores: **vendedores** e **compradores**. Cada tipo de utilizador tem um conjunto de opções a que pode aceder. Um vendedor autenticado pode registar, consultar e alterar o estado de um imóvel (em venda, reservado ou vendido), tendo também acesso a opções que lhe permitem a obtenção de informações relativas às consultas dos imóveis que tem para venda. Um comprador pode pesquisar imóveis com base nas suas características ou pelo seu identificador, sem ter que estar registado na aplicação. No entanto, se um comprador pretender marcar um imóvel como favorito e posteriormente consultar a lista dos seus imóveis favoritos, deverá estar registado e autenticado na aplicação. Além das funcionalidades referidas, a aplicação ainda permite que um vendedor coloque o seu imóvel em leilão e defina a duração do leilão. Uma vez iniciado um leilão, qualquer comprador pode participar no mesmo, bastando para isso indicar a quantia limite que está disposto a investir, o intervalo de tempo (em minutos) entre licitações e os incrementos que tenciona realizar.

# API e arquitetura de classes

Antes de falarmos sobre cada uma das classes e da sua arquitetura, é importante referirmos que todas as variáveis de instância foram declaradas com o modificador de acesso **private** de forma a serem acessíveis de forma direta apenas dentro da sua própria classe. Assim, o acesso e modificação, a partir do exterior, das variáveis de instância de uma classe só são possíveis através dos *getters* e *setters* disponibilizados por essa classe.

## Utilizador (classe abstrata)

Todos os utilizadores registados na aplicação têm que especificar, aquando do seu registo, o seu e-mail, nome, password, morada e data de nascimento. Existem ainda campos específicos consoante o tipo de utilizador, no entanto, qualquer utilizador registado tem os 5 campos acima enumerados.

### Atributos

Pelos motivos referidos, o grupo definiu a classe abstrata **Utilizador**, com as seguintes variáveis de instância:



**Nota:** a data de nascimento de cada utilizador é armazenada numa instância de **LocalDate**. As instâncias desta classe são imutáveis e não armazenam qualquer informação relativa ao tempo (i.e.: horas, minutos e segundos), pelo que são adequadas para guardar datas de nascimento.

### Construtores

* private **Utilizador**();

Construtor por omissão (declarado como privado para não ser possível criar um utilizador sem especificar os seus dados).

* public **Utilizador** (

String email,

String nome,

String password,

String morada,

LocalDate dataNascimento

);

Construtor parametrizado. Constrói um utilizador com os campos especificados nos argumentos.

* public **Utilizador**(Utilizador original);

Construtor de cópia. Constrói uma cópia do utilizador original.

### Métodos

#### Getters

* public String **getEmail**();
* public String **getNome**();
* public String **getPassword**();
* public String **getMorada**();
* public LocalDate **getDataNascimento**();

#### Setters

* public void **setEmail**(String email);
* public void **setNome**(String nome);
* public void **setPassword**(String password);
* public void **setMorada**(String morada);
* public void **setDataNascimento**(LocalDate dataNascimento);

#### Restantes

* public boolean **validaPassword**(String password);

Testa se a *password* passada como parâmetro é igual à *password* deste Utilizador.

**Retorna:**

*true* se a *password* passada como parâmetro for a deste Utilizador.

* public boolean **equals**(Object o)

Testa se este Utilizador é igual ao objeto passado como parâmetro.

**Retorna:**

*true* se este Utilizador for igual ao objeto passado como parâmetro.

* abstract public **Utilizador** clone();

**Retorna:**

*Clone* deste Utilizador.

* public String **toString**();

**Retorna:**

Representação textual deste Utilizador.

## Comprador (especialização de Utilizador)

Um comprador tem todos os dados de um Utilizador, tendo ainda um conjunto de ids dos seus imóveis favoritos. Por esse motivo, o grupo definiu a classe Comprador como uma especialização da classe Utilizador.

### Atributos

Tal como já foi referido, um Comprador é um Utilizador que tem um conjunto de ids dos seus imóveis favoritos. Assim, temos apenas uma declaração de v.i. na classe Comprador, que é a seguinte:



### Construtores

* public **Comprador**(

String email,

String nome,

String password,

String morada,

LocalDate dataNascimento

);

Construtor parametrizado. Constrói um Comprador com os dados especificados nos parâmetros e com um conjunto vazio de imóveis favoritos.

* public **Comprador**(Comprador comprador);

Construtor de cópia. Cria uma cópia do Comprador passado como parâmetro.

### Métodos

#### Getters

* public TreeSet<String> **getFavoritos**();

**Retorna:**

Uma *shallow copy* do conjunto de imóveis favoritos deste Comprador.

**Nota:** é realizada uma *shallow copy* porque as instâncias de String são imutáveis, logo a cópia das Strings do conjunto de ids dos imóveis favoritos é desnecessária.

#### Setters

* public void **setFavoritos**(Set<String> favoritos);

Define os imóveis favoritos deste Comprador.

* public void **setFavorito**(String idImovel);

Adiciona idImovel ao conjunto de ids dos imóveis favoritos deste vendedor.

#### Restantes

* public Comprador **clone**();

**Retorna:**

Clone deste Comprador.

* public boolean **equals**(Object o);

Testa se este Comprador é igual ao objeto passado como parâmetro.

**Retorna:**

*true* se este Comprador for igual ao objeto passado como parâmetro.

* public String **toString**();

**Retorna:**

Representação textual deste Comprador.

* public int **hashCode**();

**Retorna:**

O valor do *hash code* deste Comprador.

## Vendedor (especialização de Utilizador)

Cada vendedor tem os dados de um Utilizador e tem ainda um conjunto com os ids dos imóveis que tem à venda e outro conjunto com os ids dos imóveis vendidos. Assim, a classe Vendedor é (tal como Comprador) uma especialização da classe Utilizador.

### Atributos

Segue-se a declaração das duas variáveis de instância que um Vendedor acrescenta às variáveis de instância de Utilizador:



### Construtores

* public **Vendedor**(

String email,

String nome,

String password,

String morada,

LocalDate dataNascimento

);

Construtor parametrizado. Constrói um Vendedor com os dados especificados nos parâmetros e com conjuntos vazios de imóveis em venda e vendidos.

* public **Vendedor**(Vendedor vendedor);

Construtor de cópia. Cria uma cópia do Vendedor passado como parâmetro.

### Métodos

#### Getters

* public Set<String> **getEmVenda**();

**Retorna:**

Uma *shallow copy* do conjunto de ids dos imóveis que este Vendedor tem à venda.

**Nota:** é realizada uma *shallow copy* porque as instâncias de String são imutáveis, logo a cópia das Strings do conjunto de ids dos imóveis em venda é desnecessária.

* public Set<String> **getVendidos**();

**Retorna:**

Uma *shallow copy* do conjunto de ids dos imóveis que este Vendedor já vendeu.

#### Setters

* public void **setEmVenda**(Set<String> emVenda);

Define o conjunto dos ids dos imóveis que este Vendedor tem para vender.

* public void **setVendidos**(Set<String> vendidos);

Define o conjunto de ids dos imóveis que este Vendedor já vendeu.

#### Restantes

* public Vendedor **clone**();

**Retorna:**

Clone deste Vendedor.

* public boolean **equals**(Object o);

Testa se este Vendedor é igual ao objeto passado como parâmetro.

**Retorna:**

*true* se este Vendedor for igual ao objeto passado como parâmetro.

* public String **toString**();

**Retorna:**

Representação textual deste Vendedor.

* public int **hashCode**();

**Retorna:**

O valor do *hash code* deste Vendedor.

## Imovel (classe abstrata)

Todos os imóveis têm a si associado o seu id, a rua onde se situam, o seu estado (em venda, reservado ou vendido), o preço pedido, o preço mínimo aceite pelo proprietário, o número de consultas e uma lista de consultas. Além destes atributos, cada tipo de imóvel tem um ou mais atributos específicos, no entanto todos os imóveis têm os campos acima referidos, pelo que o grupo definiu a classe abstrata Imovel (Cada um dos tipos de imóvel que serão apresentados é uma especialização desta classe abstrata).

### Atributos



**Nota:** EstadoImovel é um *enum* cujos valores são: EM\_VENDA, RESERVADO e VENDIDO.

### Construtores

* private **Imovel**();

Construtor por omissão (declarado como privado para não ser possível construir imóveis sem especificar os seus dados).

* public **Imovel**(String id, String rua, int precoPedido, int precoMinimo);

Construtor parametrizado. Constrói um Imovel com os dados especificados nos argumentos e com o estado EM\_VENDA, o número de consultas a 0 e uma lista de consultas vazia.

* public **Imovel**(Imovel imv)

Construtor de cópia. Cria uma cópia do Imovel passado como parâmetro.

### Métodos

#### Getters

* public String **getId**();
* public String **getRua**();
* public EstadoImovel **getEstado**();
* public int **getPrecoPedido**();
* private int **getPrecoMinimo**();

**Nota:** O método **getPrecoMinimo()** foi declarado como *private* para que o preço pedido por um imóvel não possa ser acedido noutras classes, nomeadamente na classe Comprador.

* public int **getQuantasConsultas**();

**Retorna:**

Número de consultas deste Imovel.

* public ArrayList<Consulta> **getConsultas**();

**Retorna:**

*Deep copy* da lista de consultas deste Imovel.

#### Setters

* public void **setId**(String id);

Define o id deste Imovel.

* public void **setRua**(String rua);

Define a rua deste Imovel.

* public void **setEstado**(EstadoImovel estado);

Altera o estado de este Imovel.

* public void **setPrecoPedido**(int precoPedido);

Define o preço pedido deste Imovel.

* private void **setPrecoMinimo**(int precoMinimo);

Define o preço mínimo deste imóvel (declarado como *private* para não ser possível alterar o preço mínimo de um Imovel fora da classe Imovel).

#### Restantes

* public void **registaConsulta**(Consulta c);

Regista uma consulta deste Imovel.

* public int **compareTo**(Imovel imv);

Compara o número de consultas deste Imovel como o número de consultas do Imovel passado como parâmetro.

**Retorna:**

0 se o Imovel passado como argumento tiver o mesmo número de consultas que este Imovel; 1 se o Imovel passado como parâmetro tiver mais consultas que este Imovel; -1 caso contrário.

* public abstract **clone**();

**Retorna:**

Clone deste Imovel.

* public boolean **equals**(Object o);

Testa se este Imovel é igual ao objeto passado como parâmetro.

**Retorna:**

*true* se este Imovel for igual ao objeto passado como parâmetro.

* public String **toString**();

**Retorna:**

Representação textual deste Imovel.

* public int **hashCode**();

**Retorna:**

Valor do *hash code* deste Imovel.

## Moradia (especialização de Imovel)

Uma moradia caracteriza-se por cada um dos atributos de Imovel, tendo também a si associado o seu tipo (isolada, geminada, banda ou gaveto), área de implantação, área total, área envolvente, número de quartos, WCs e número da porta. Assim, a classe Moradia é uma especialização da classe Imovel.

### Atributos



**Nota:** TipoMoradia é um *enum* cujos valores são ISOLADA, GEMINADA, BANDA, GAVETO.

### Construtores

* public **Moradia**(

String id,

String rua,

int precoPedido,

int precoMinimo,

TipoMoradia tipo,

int areaTotal,

int areaEnv,

int numQuartos,

int numWCs,

int numDaPorta

);

Construtor por omissão. Constrói uma Moradia com cada um dos dados passados como parâmetros.

* public **Moradia**(Moradia m);

Construtor de cópia. Cria uma cópia da Moradia passada como parâmetro.

### Métodos

#### Getters

* public TipoMoradia **getTipo**();
* public int **getAreaTotal**();
* public int **getAreaEnv**();
* public int **getNumQuartos**();
* public int **getNumWCs**();
* public int **getNumDaPorta**();

#### Setters

* public void **setTipo**(TipoMoradia tipo);
* public void **setAreaTotal**(int areaTotal);
* public void **setAreaEnv**(int areaEnv);
* public void **setNumQuartos**(int numQuartos);
* public void **setNumWCs**(int numWCs);
* public void **setNumDaPorta**(int numDaPorta);

#### Restantes

* public Moradia **clone**();

**Retorna:**

Clone desta Moradia.

* public boolean **equals**(Object o);

Testa se esta Moradia é igual ao objeto passado como parâmetro.

**Retorna:**

*true* se esta Moradia for igual ao objeto passado como parâmetro.

* public String **toString**();

**Retorna:**

Representação textual desta Moradia.

* public int **hashCode**();

**Return:**

Valor do *hash code* desta Moradia.

## Apartamento (especialização de Imovel)

Um apartamento é um Imovel que tem um tipo (Simples, Duplex ou Triplex), uma determinada área total, um certo número de quartos e de WCs, um número da porta e do andar, podendo também ter garagem.

### Atributos



**Nota:** TipoApartamento é um *enum* cujo domínio é: SIMPLES, DUPLEX, TRIPLEX.

### Construtores

* public **Apartamento**(

String id,

String rua,

int precoPedido,

int precoMinimo,

TipoApartamento tipo,

int numQuartos,

int numWCs,

int numDaPorta,

int andar,

boolean temGaragem

);

Construtor parametrizado. Constrói um Apartamento com cada um dos dados passados como parâmetro.

* public **Apartamento**(Apartamento a);

Construtor de cópia. Cria uma cópia do Apartamento passado como parâmetro.

### Métodos

#### Getters

* public TipoAparamento **getTipo**();
* public int **getAreaTotal**();
* public int **getNumQuartos**();
* public int **getNumWCs**();
* public int **getNumDaPorta**();
* public int **getAndar**();
* public boolean **getTemGaragem**();

#### Setters

* public void **setTipo**(TipoApartamento tipo);
* public void **setAreaTotal**(int areaTotal);
* public void **setNumQuartos**(int numQuartos);
* public void **setNumWCs**(int numWCs);
* public void **setNumDaPorta**(int numDaPorta);
* public void **setAndar**(int andar);
* public void **setTemGaragem**(boolean temGaragem);

#### Restantes

* public Apartamento **clone**();

**Retorna:**

Clone deste Apartamento.

* public boolean **equals**(Object o);

Testa se este Apartamento é igual ao objeto passado como parâmetro.

**Retorna:**

*true* se este Apartamento for igual ao objeto passado como parâmetro.

* public String **toString**();

**Retorna:**

Representação textual deste Apartamento.

* public int **hashCode**();

**Retorna:**

Valor do *hash code* deste Apartamento.

## Loja (especialização de Imovel)

A Loja é um Imovel que tem uma determinada área, um certo número da porta, pode ter ou não WC e tem um determinado tipo de negócio viável. Além disso, uma loja poderá ainda ter parte habitacional, incorporando um Apartamento, pelo que Loja é ainda especializada pela classe LojaHabitavel (a descrever a seguir à classe Loja).

### Atributos



### Construtores

* public **Loja**(

String id,

String rua,

int precoPedido,

int precoMinimo,

int area,

boolean temWC,

String tipoNegocio,

int numDaPorta

);

Construtor parametrizado. Constrói uma Loja com os valores passados como parâmetros.

* public **Loja**(Loja loja);

Construtor de cópia. Cria uma cópia da Loja passada como parâmetro.

### Métodos

#### Getters

* public int **getArea**();
* public boolean **getTemWC**();
* public String **getTipoNegocio**();
* public int **getNumDaPorta**();

#### Setters

* public void **setArea**(int area);
* public void **setTemWC**(boolean temWC);
* public void **setTipoNegocio**(String tipoNegocio);
* public void **setNumDaPorta**(int numDaPorta);

#### Restantes

* public Loja **clone**();

**Retorna:**

Clone desta Loja.

* public Loja **equals**(Object o);

Testa se esta Loja é igual ao objeto passado como parâmetro.

**Retorna:**

*true* se esta Loja for igual ao objeto o.

* public String **toString**();

**Retorna:**

Representação textual desta Loja.

* public int **hashCode**();

**Retorna:**

Valor do *hash code* desta Loja.

## LojaHabitavel (especialização de Loja)

Uma LojaHabitavel não é mais do que uma Loja com um Apartamento.

### Atributos

O único atributo que a LojaHabitavel acrescenta a uma Loja é o apartamento, logo temos:



### Construtores

* public **LojaHabitavel**(

String id,

String rua,

int precoPedido,

int precoMinimo,

int area,

boolean temWC,

String tipoNegocio,

int numDaPorta,

Apartamento apartamento

);

Construtor parametrizado. Constrói uma LojaHabitavel com os dados passados como parâmetro.

* public **LojaHabitavel**(Loja l, Apartamento a);

Constrói uma LojaHabitavel a partir de uma Loja e de um Apartamento.

* public **LojaHabitavel**(LojaHabitavel lh);

Construtor de cópia. Cria uma cópia da LojaHabitavel passada como parâmetro.

### Métodos

#### Getters

* public Apartamento **getApartamento**();

**Retorna:**

Uma cópia do Apartamento desta LojaHabitavel.

#### Setters

* public void **setApartamento**(Apartamento apartamento);

Define o apartamento desta LojaHabitavel como uma cópia do Apartamento passado como parâmetro.

#### Restantes

* public LojaHabitavel **clone**();

**Retorna:**

Clone desta LojaHabitavel.

* public boolean **equals**(Object o);

Testa se esta LojaHabitavel é igual ao objeto passado como parâmetro.

**Retorna:**

*true* se esta LojaHabitavel for igual a o.

* public String **toString**();

**Retorna:**

Representação textual desta LojaHabitavel.

* public int **hashCode**();

**Retorna:**

Valor do *hash code* desta LojaHabitavel.

## Terreno (especialização de Imovel)

Um Terreno é um Imovel com uma determinada área que poderá ser adequado para construção de habitação ou para construção de armazéns e que tem informações adicionais como o diâmetro das suas canalizações, os kWh. Além disso, um Terreno poderá ter também rede de esgotos.

### Atributos



### Construtores

* public **Terreno**(

String id,

String rua,

int precoPedido,

int precoMinimo,

int area,

boolean terrenoHab,

boolean terrenoArm,

double diamCanalizacoes,

double maxKWh,

boolean temRedeEsgotos

);

Construtor parametrizado. Constrói um Terreno com os dados especificados nos parâmetros.

* public **Terreno**(Terreno terr);

Construtor de cópia. Cria uma cópia do Terreno passado como parâmetro.

### Métodos

#### Getters

* public int **getArea**();
* public boolean **getTerrenoHab**();
* public boolean **getTerrenoArm**();
* public double **getDiamCanalizacoes**();
* public double **getMaxKWh**();
* public boolean **getTemRedeEsgotos**();

#### Setters

* public void **setArea**(int area);
* public void **setTerrenoHab**(boolean terrenoHab);
* public void **setTerrenoArm**(boolean terrenoArm);
* public void **setDiamCanalizacoes**(double diamCanalizacoes);
* public void **setMaxKWh**(double maxKWh);
* public void **setTemRedeEsgotos**(boolean temRedeEsgotos);

#### Restantes

* public Terreno **clone**();

**Retorna:**

Um clone deste Terreno.

* public boolean **equals**(Object o);

Testa se este Terreno é igual ao objeto passado como parâmetro.

**Retorna:**

*true* se este Terreno for igual ao objeto o.

* public String **toString**();

**Retorna:**

Representação textual deste terreno.

* public int **hashCode**();

**Retorna:**

Valor do *hash code* deste Terreno.

# Módulo principal