|  |  |
| --- | --- |
| 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 2](#_Toc451126634)

[API e arquitetura de classes 3](#_Toc451126635)

[Utilizador (classe abstrata) 3](#_Toc451126636)

[Comprador (especialização de Utilizador) 4](#_Toc451126637)

[Vendedor (especialização de Utilizador) 6](#_Toc451126638)

[Imovel (classe abstrata) 7](#_Toc451126639)

[Moradia (especialização de Imovel) 10](#_Toc451126640)

[Apartamento (especialização de Imovel) 12](#_Toc451126641)

[Loja (especialização de Imovel) 13](#_Toc451126642)

[LojaHabitavel (especialização de Loja) 15](#_Toc451126643)

[Terreno (especialização de Imovel) 16](#_Toc451126644)

[Consulta (classe de objetos imutáveis) 18](#_Toc451126645)

[Módulo principal 18](#_Toc451126646)

# 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 e embora não existam utilizadores que tenham apenas os 5 campos enumerados, qualquer utilizador registado tem necessariamente cada um desses campos.

### Atributos

Como os campos acima referidos são a base de qualquer utilizador, o grupo optou por definir **Utilizador** como uma classe abstrata 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

* public **Utilizador**();

Construtor por omissão.

* 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. Cria 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 métodos

* 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 acrescenta apenas um conjunto de ids de imóveis favoritos aos dados de um Utilizador, pelo que optámos por definir a classe Comprador como uma especialização da classe Utilizador.

### Atributos

Ao especializar Utilizador, a classe Comprador herda cada um dos atributos de Utilizador, permitindo assim que na especificação das variáveis de instância da classe Comprador tenhamos apenas que declarar o conjunto de ids dos seus imóveis favoritos, i.e.:



**Nota:** A utilização de um Set<String> em vez de uma List<String> justifica-se pela necessidade de garantirmos que não ocorrem repetições de ids de imóveis favoritos, mesmo que um utilizador adicione o mesmo Imovel aos favoritos mais do que uma vez.

### 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 confere-lhe 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 métodos

* 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)

Um Vendedor tem cada um dos atributos de um Utilizador e tem ainda um conjunto de ids de imóveis que estão à venda e um conjunto de ids de imóveis vendidos, pelo que optámos por definir a classe Vendedor como uma especialização da classe Utilizador.

### Atributos

Na declaração das variáveis de instância de Vendedor temos apenas os conjuntos de ids de imóveis em venda e vendidos, uma vez que os restantes atributos já são herdados da classe Utilizador. Assim, na declaração das variáveis de instância de Vendedor temos:



**Nota:** A utilização de Set<String> e não de List<String> deve-se ao facto de não poderem existir repetições de ids de imóveis, tanto na coleção de imóveis em venda, como na dos vendidos.

### 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 confere-lhe 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 métodos

* 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. É importante notar que qualquer imóvel tem os atributos referidos, mas um imóvel concreto não fica completamente definido pelos mesmos, pelo que definimos a classe Imovel como uma classe abstrata.

### Atributos

Os atributos da classe Imovel são aqueles foram acima enumerados, logo temos:



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

### Construtores

* public **Imovel**();

Construtor por omissão.

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

Construtor parametrizado. Constrói um Imovel com os dados especificados nos argumentos e coloca-o 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**();
* public int **getPrecoMinimo**();
* 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 métodos

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

Regista uma consulta deste Imovel.

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

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

**Retorna:**

0 se o Imovel passado como parâmetro 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 tem cada um dos atributos da classe Imovel e tem também os campos: 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

Os atributos de Moradia correspondem a cada um dos campos que esta acrescenta a um Imovel, ou seja:



**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 métodos

* 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 além dos atributos de Imovel tem os campos: tipo (Simples, Duplex ou Triplex), área total, número de quartos, número de WCs, número da porta e do andar e pode ter ou não garagem. Assim sendo, definimos Apartamento como uma especialização de Imovel.

### 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 classe Loja especializa Imovel, acrescentando-lhe informações relativamente à área da loja, número da porta, tipo de negócio viável e existência de WC. 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

Os atributos de Loja são:



### 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 métodos

* 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, pelo que LojaHabitavel especializa Loja.

### Atributos

O único atributo que uma LojaHabitavel acrescenta a uma Loja é o apartamento, logo na declaração das v.i. de LojaHabitavel temos apenas:



### 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 uma especialização de Imovel na medida em que acrescenta aos atributos de Imovel informação relativa a: área, adequação para construção de habitação ou para construção de armazéns, diâmetro das canalizações (mm), kWh máximos e existência ou não de 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 métodos

* 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.

## Consulta (classe de objetos imutáveis)

Consulta é um objeto imutável que guarda a data em que um determinado imóvel foi consultado, bem como o seu id. Se o imóvel em questão tiver sido consultado por um utilizador autenticado, também é guardado o email desse utilizador.

### Atributos

Todos os atributos de Consulta foram declarados com o modificador *final*, de forma a garantir que as instâncias de Consulta são imutáveis. Na declaração das variáveis de instância de Consulta temos então:



### Construtores

* public **Consulta**();

Construtor por omissão.

* public **Consulta**(String email, String idImovel);

Cria uma Consulta do Imovel identificado por idImovel e associa-a a email.

* public **Consulta**(Utilizador usr, String idImovel);

Cria uma Consulta do Imovel identificado por idImovel e associa-a ao email de usr.

* public **Consulta**(Consulta c);

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

### Métodos

#### Getters

* public LocalDate **getData**();
* public String **getEmail**();
* public String **getIdImovel**();

#### Setters

(não definimos *setters* para a classe Consulta, de forma a garantirmos a imutabilidade das suas instâncias.)

#### Restantes métodos

* public boolean **feitaPorUtilizadorRegistado**();

**Retorna:**

*true* se esta Consulta foi feita por um utilizador registado.

* public Consulta **clone**();

**Retorna:**

Clone desta Consulta.

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

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

**Retorna:**

*true* se esta Consulta for igual a o.

* public int **compareTo**(Consulta c);

Compara a data desta Consulta com a data da consulta passada como parâmetro.

**Retorna:**

-1 se a data desta Consulta for menor que a de c; 0 se as datas forem iguais; 1 caso contrário.

* public String **toString**();

**Retorna:**

Representação textual desta Consulta.

* public int **hashCode**();

**Retorna:**

Valor do *hash code* desta Consulta.

## Licitador

## Leilão

## Imoobiliaria

A classe **Imoobiliaria** é composta por objetos das classes que foram descritas até agora, nomeadamente das classes: Utilizador, Imovel e Leilao. Cada instância de Imoobiliaria contém uma coleção de utilizadores, uma coleção de imóveis, o utilizador autenticado (se este existir) e um leilão que tanto pode estar encerrado como a decorrer.

### Atributos

Tanto as instâncias de Imovel como as instâncias de Utilizador têm a si associada uma chave pela qual são pesquisadas em grande parte das operações da Imoobiliaria. Dada esta necessidade de mapear chaves em valores de forma rápida e eficiente, o grupo optou por guardar um *Map* de email para Utilizador e um *Map* de ID de Imovel para Imovel. Além desses 2 *Maps*, cada instância de Imoobiliaria tem o utilizador autenticado e um objeto da classe Leilao, pelo que os atributos de Imoobiliaria são os seguintes:



### Construtores

# Módulo principal