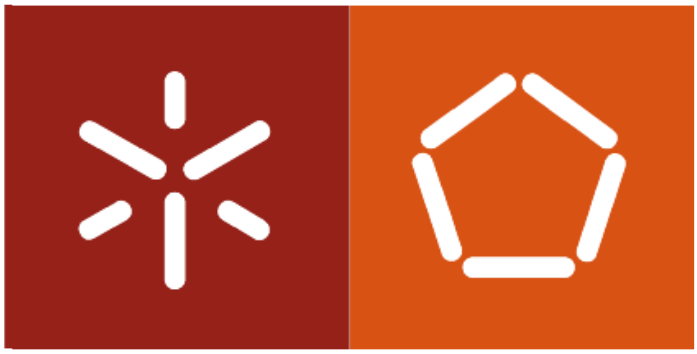


# RELATÓRIO DE POO

## POO - 2º Semestre 2016

**Grupo de Trabalho 46**


Alexandre Teixeira
(A73547)

Bruno Sousa
(A74330)

Rafael Silva
(A74264)

# POO - IMOOBILIÁRIA

## Índice

# POO - IMOOBILIÁRIA

## Breve descrição do projeto

Neste projeto foi-nos proposto a criação de uma aplicação capaz de fazer uma gestão de imóveis e utilizadores, mais concretamente, a venda e compra de imóveis, para esse efeito sendo que, utilizamos uma linguagem orientada a objetos - JAVA.

Esta aplicação tem como funcionalidade a gestão de uma imobiliária, isto é, uma aplicação, capaz e precisa, responsável pela venda e compra de imóveis. A mesma tem como propósito dar aos utilizadores, **vendedores** e **compradores**, a informação necessária para efetuar, respetivamente, a venda e compra de um imóvel.

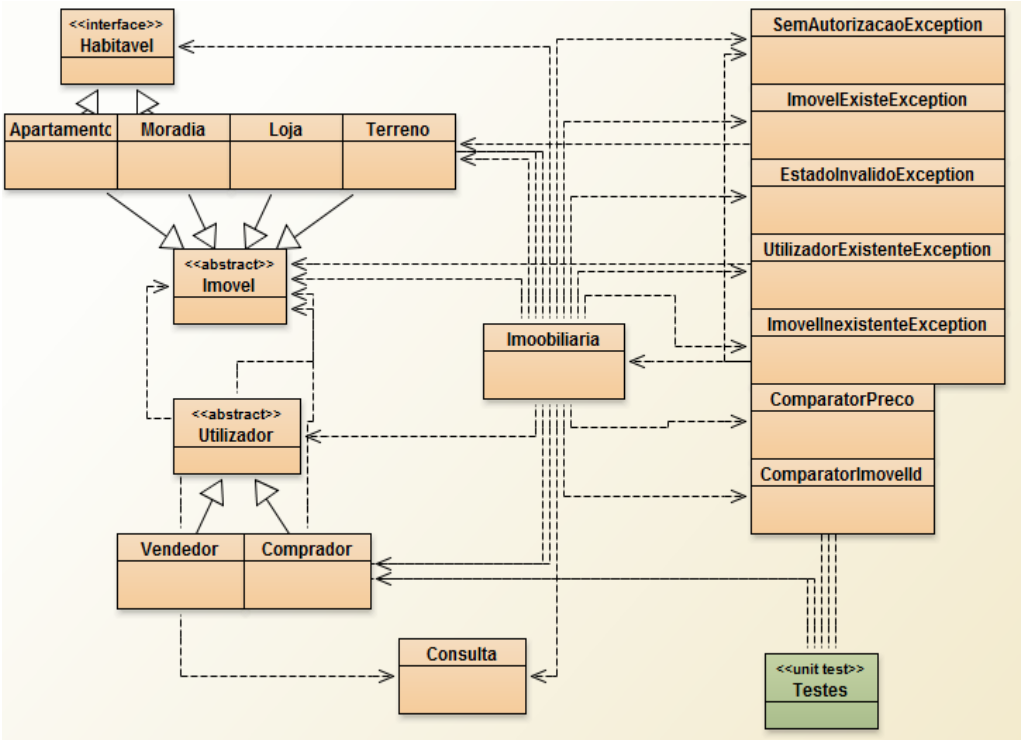
Todos os **utilizadores** têm a capacidade de realizar uma pesquisa de imóveis a fim de encontrarem o pretendido, de modo a suprir as suas necessidades.

Por sua vez, os **vendedores** têm a permissão de inserir, consultar e remover os imóveis, inseridos pelos mesmos.

E somente os **compradores**, que estão registados, têm a possibilidade de criar uma lista de imóveis favoritos.

# 1- Estrutura da aplicação

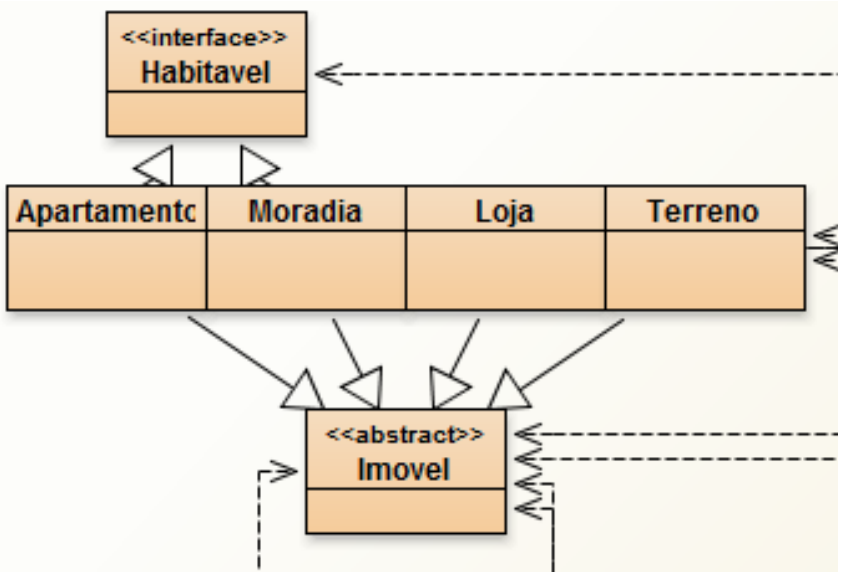
## 1.1- Classe Imoobliaria




A classe **Imoobiliria** contém as variáreis de instância, pedidas no enunciado do projeto, que permitem a funcionalidade da aplicação, sendo esta a classe principal da mesma pois, controla tudo o que é executado.

Esta contem a lista de imoveis que são acedidos pelo utilizador e uma lista com todos os utilizadores. Nesta classe, encontramos também, um **utilizador** e um **int**, sendo estes responsáveis, respetivamente, pelo login e controlo da sessão. Alem disso, esta possui os construtores "get", "set", "equals", "clone" e "toString". Podemos encontrar também as funções "saveState" e "loadState", responsáveis, na devida ordem, por guardar o estado da aplicação e ler o estado da aplicação a partir de um ficheiro.

## 1.2- Classe Abstrata Imovel




A classe Imovel é abstrata, isto é, responsável pelos controladores dos diferentes tipos dos imóveis. Estes são ligados à classe de forma a partilharem variáveis, sendo estas, preço mínimo, preço pedido, estado e identificação. Dentro das suas subclasses é usada a função "super", para aceder a estas variáveis.

### 1.2.1- Classe Apartamento

Esta subclasse contem os dados de um apartamento, estado representados pelo tipo (Simples, Duplex, Triplex, …), área total, total de quartos e WCs, número da porta, andar e garagem (caso esta exista).

### 1.2.2- Classe Moradia

A subclasse Moradia tem como representação o tipo (Germinada, Isolada, ...), área de implementação, área de cobertura, área do terreno, número de quartos e WCs e o número de porta.

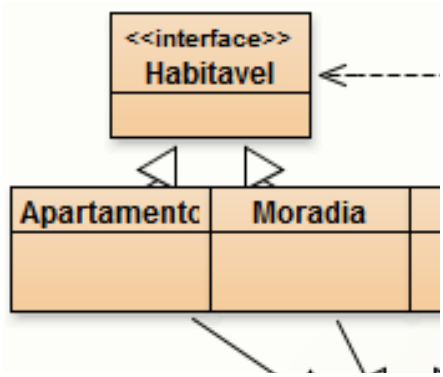
### 1.2.3- Classe Loja

Esta subclasse representa uma loja, contendo assim os seus dados, sendo estes: área total, WC (boolean), negócio viável (Roupa, Comida, ...) e o número da porta.

### 1.2.4- Classe Terreno

A subclasse terreno contém os dados para a sua identificação, sendo estes, representados por: área de construção, tipo de construção (Habitação, Comercial, ...), diâmetro da canalização, kWh máximos suportados, acesso a rede de esgotos (boolean).

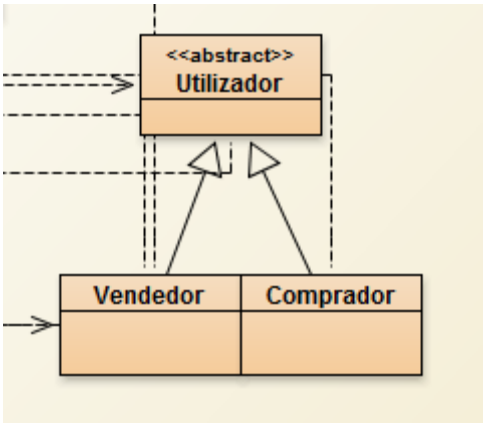
## 1.3- Interface Habitavel




A interface Habitavel implementa métodos e propriedades comuns ás duas Subclasses, **apartamento** e **moradia**. Esta implementação é responsável pelos controladores comuns das duas subclasses sendo estes:

```
1   public interface Habitavel
2   {
3       public String getTipo ();
4       public double getAreaT ();
5       public int getNQuartos ();
6       public int getNWc();
7       public int getNPorta();
8       public void setTipo (String tipo);
9       public void setAreaT (double areaT);
10      public void setNQuartos (int nQuartos);
11      public void setNWc (int nWc);
12      public void setNPorta (int nPorta);
13  }
```

## 1.4- Classe Abstrata Utilizador




A classe Utilizador é abstrata, responsável pelos controladores dos seus dois tipos, **vendedor** e **comprador**. Estes são ligados à classe de forma a partilharem variáveis, como: email, nome, password, morada e data de nascimento. Dentro das subclasses é usada a função "super", com o intuito de aceder a estas variáveis.

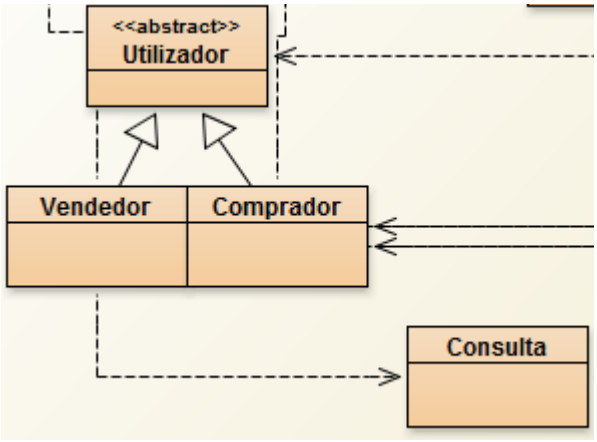
### 1.4.1- Classe Vendedor

A subclasse vendedor representa os dados de um vendedor, tais como: lista de imóveis e lista de consultas. Esta subclasse é responsável pelos controladores de um vendedor, isto é, a implementação das funções que o complementam.

### 1.4.2- Classe Comprador

Subclasse comprador contem os dados respetivos de um cliente, sendo este uma lista de imóveis favoritos. Esta subclasse implementa controladores, para um utilizador com a capacidade de compra de imóveis, isto é, utiliza controladores específicos para a sua finalidade.

## 1.5- Classe Consulta




A Classe consulta é responsável pela implementação e gestão de consultas ao imóvel realizadas pelos utilizadores e acedidas apenas pelos vendedores. Esta implementação controla as seguintes variáveis: email, identificação do imóvel, lista de datas de consulta e número de consultas.

## 1.6- Classes Exception

As classes Exception são responsáveis por evitar erros durante a execução do programa, por exemplo, estas evitam a sobreposição de dados, a introdução de dados incorretos e problemas de acesso.

### 1.6.1- Classe SemAutorizacaoExeption

Subclasse responsável por verificar se um certo utilizador tem autorização para aceder a certos dados da aplicação.

### 1.6.2- Classe ImovelExisteException

Subclasse responsável por verificar se um dado imóvel existe na aplicação, evitando assim, a sobreposição de dados.

### 1.6.3- Classe EstadoInvalidoException

Subclasse responsável por verificar se o estado de um certo imóvel é valido.

### 1.6.4- Classe UtilizadorExistenteException

Subclasse responsável por verificar se o utilizador existe para a execução de determinada operação.

### 1.6.5- Classe ImovelInexistenteException

Subclasse responsável por verificar se o imóvel em estudo existe.

## 1.7- Classes Comparator

A classe **Comparator** têm como finalidade comparar dados pretendidos para uma melhor consulta e inserção de dados na aplicação.

### 1.7.1- Classe ComparatorPreco

Subclasse **ComparatorPreco** tem como função comparar os preços de dois imóveis.

### 1.7.2- Classe ComparatorImovelId

Por sua vez, esta subclasse vai comparar se dois imóveis possuem a mesma identificação, ou seja, compara se dois imóveis são iguais.

## 2- Extensão da aplicação

Temos duas classes abstratas, Imovel e Utilizador, a primeira é extensível à introdução de novos tipos de imóveis, podendo assim ser possível adicionar imóveis à aplicação que for preciso, a segunda classe também é extensível pelo facto de ser possível introduzir outro tipo diferente de utilizador.

Esta aplicação tem a capacidade de permitir a inserção de novos utilizadores, bem como, a inserção de novos imóveis, permitindo assim ao utilizar que gere esta aplicação a possibilidade de introdução de dados caso seja preciso.

É de salientar que, para poder ser feita uma nova inserção de uma Classe de Imovel esta tem de respeitar o tipo de dados que são representados na mesma, sendo também de referir que tal, diz respeito a uma nova inserção de uma Classe de Utilizador que tem de respeitar o tipo de dados da Classe Abstrata utilizador.

# 3- Conclusão

Após a realização deste projeto, conseguimos perceber o funcionamento de uma aplicação que tem como objetivo a gestão de vendas e compras de imóveis.

Para a concretização do mesmo, utilizámos a linguagem de programação - JAVA, sendo esta uma linguagem nova para os diferentes elementos do grupo.

De modo a desenvolvermos este projeto de forma correta e ordenada, foi essencial a partição do grupo nas aulas práticas, que proporcionaram uma aprendizagem ampla e detalhada sobre a utilização desta linguagem, sendo de salientar a preocupação e empenho posterior por parte do grupo.

Pudemos com este trabalho perceber, que a linguagem em JAVA tem semelhanças, relativamente á sintaxe, com a linguagem em C, sendo que, na primeira há certos detalhes os quais não necessitam da nossa preocupação, como: memória, processamento, lixo, etc enquanto que, em C são detalhes que necessitam de uma consideração diferente.

Este projeto permitiu o desenvolvimento do nosso conhecimento acerca da linguagem e da sua utilidade na criação de aplicações como esta.