

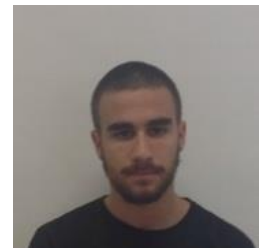
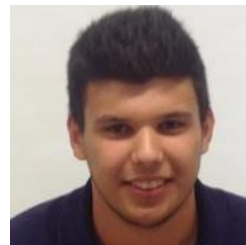


**Universidade do Minho**  
Escola de Engenharia  
Mestrado Integrado em Engenharia Informática

## **Unidade Curricular de Programação Orientada aos Objetos**

Ano Lectivo de 2015/2016

### **Grupo 70**



**Ricardo Guerra Leal (A75411)**

**João André Aleixo (A74098)**

**Bruno Filipe Ferreira (A74155)**

Maio, 2016

## Índice:

Introdução.....	
Funções definidas.....	
Interface.....	
Acréscimo de novos tipos de imoveis ...	

## **Introdução:**

O projeto lançado na cadeira de POO consiste no desenvolvimento de uma aplicação destinada a uma imobiliária que permita a gestão de diversos tipos de imóveis. Para além disto permite ainda que o utilizador tenha acesso a vários dados relativos a um certo imóvel, moradia, apartamento, loja e terreno, variando as permissões de acordo com o seu tipo, Vendedor ou Comprador.

# Funções definidas

Neste projeto foi inicialmente definida a classe **ImobiliariaApp** que posteriormente estaria ligada com todas as ou classes definidas. Para permitir a completa utilização do programa foi necessária a implementação das seguintes funções disponibilizadas, presentes na classe **ImobiliariaApp**:

- ✓ O estado da aplicação deverá estar pré-populado com um conjunto de dados significativos, que permita testar toda a aplicação no dia da entrega.  
`public static Imobiliaria initApp ()`
- ✓ Registrar um utilizador, quer vendedor, quer comprador:  
`public void registarUtilizador ( Utilizador utilizador )`  
`throws UtilizadorExistenteException`
- ✓ Validar o acesso à aplicação utilizando as credenciais (email e password);  
`public void iniciaSessao ( String email , String password )`  
`throws SemAutorizacaoException` `public void fechaSessao ()`

## Vendedores

- ✓ Colocar um imóvel à venda;  
`public void registaImovel ( Imovel im )`  
`throws ImovelExisteException`  
`SemAutorizacaoException`
- ✓ Visualizar uma lista com as datas (e emails, caso exista essa informação) das 10 últimas consultas aos imóveis que tem para venda;  
`public List < Consulta > getConsultas ()`  
`throws SemAutorizacaoException`
- ✓ Alterar o estado de um imóvel, de acordo com as ações feitas sobre ele;  
`public void setEstado (String idImovel , String estado )`  
`throws ImovelInexistenteException,`  
`SemAutorizacaoException,`  
`EstadoInvalidoException`
- ✓ Obter um conjunto com os códigos dos seus imóveis mais consultados (ou seja, com mais de N consultas).  
`public Set < String > getTopImoveis ( int n)`

## Todos os utilizadores

- ✓ Consultar a lista de todos os imóveis de um dado tipo (Terreno, Moradia, etc.) e até um certo preço.

`public List < Imovel > getImovel ( String classe , int preco )`

- ✓ Consultar a lista de todos os imóveis habitáveis (até um certo preço)

`public List < Habitavel > getHabitaveis ( int preco )`

- ✓ Obter um mapeamento entre todos os imóveis e respectivos vendedores.

`public Map < Imovel , Vendedor > getMapeamentoImoveis ()`

## Compradores registados

- ✓ Marcar um imóvel como favorito.

`public void setFavorito (String idImovel ) throws ImovellnexistenteException , SemAutorizacaoException`

- ✓ Consultar imóveis favoritos ordenados por preço.

`public TreeSet < Imovel > getFavoritos () throws SemAutorizacaoException`

# Classes definidas

- ✓ **(abstract)Utilizador:** Classe onde estão definidas as funções que recebem a informação de todos os utilizadores, email, nome, password, morada, data de nascimento, sendo estes depois classificados como Vendedores ou Compradores.
  - **Vendedor:** Classe onde estão definidas todas as funções características dos vendedores, como adicionar/remover e consultar anúncios, bem como os imóveis vendidos.
  - **Comprador:** Classe onde estão definidas todas as funções características dos compradores, como marcar um anúncio como favorito e pesquisar os anúncios em venda.
  
- ✓ **(abstract)Imóvel:** Classe onde estão definidas todas as funções que recebem a informação de todos os imóveis.
  - **Morada:** Classe que recebe as informações, referentes a uma moradia, tipo, área de implantação, área total coberta, área do terreno envolvente, número de quartos/wc's, número da porta.
  - **Apartamento:** Classe que recebe as informações referentes a um apartamento, tipo, área, número de quartos/wc's, número de andar e porta e a existência ou não de garagem.
  - **Loja:** Classe que recebe as informações referentes a uma loja, área, existência ou não de wc, tipo de negócio viável para o tipo de loja e número de porta.
  - **Terreno:** Classe que recebe as informações referentes a um terreno, área disponível, tipo (habitação ou industrial), diâmetro das canalizações, potência máxima suportada pela rede elétrica e acesso a rede de esgotos.

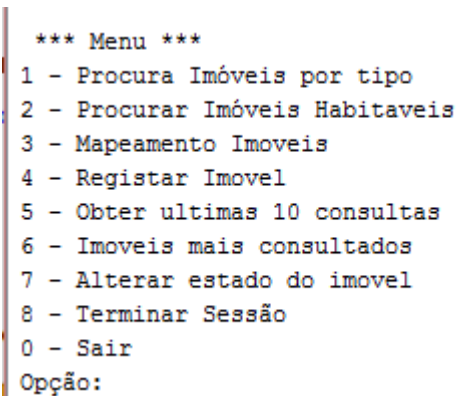
# Menu Principal



```
BlueJ: BlueJ: Janela de Terminal - src
Opções

*** Menu ***
1 - Iniciar Sessão
2 - Registar Utilizador
3 - Procura Imóveis por tipo
4 - Procurar Imóveis Habitaveis
5 - Mapeamento Imoveis
0 - Sair
Opção:
|
```

# Menu Vendedores



```
*** Menu ***
1 - Procura Imóveis por tipo
2 - Procurar Imóveis Habitaveis
3 - Mapeamento Imoveis
4 - Registar Imovel
5 - Obter ultimas 10 consultas
6 - Imoveis mais consultados
7 - Alterar estado do imovel
8 - Terminar Sessão
0 - Sair
Opção:
```

## Menu Compradores

```
*** Menu ***
1 - Procura Imóveis por tipo
2 - Procurar Imóveis Habitaveis
3 - Mapeamento Imoveis
4 - Adicionar imovel a favoritos
5 - Lista de imoveis favoritos
6 - Terminar Sessão
0 - Sair
Opção:
```

## Menu Registrar Utilizador

```
*** Menu ***
1 - Registrar Comprador
2 - Registrar Vendedor
0 - Sair
Opção:
```

## Menu Registrar Imóvel

```
*** Menu ***
1 - Moradia
2 - Terreno
3 - Loja
4 - Loja Habitável
5 - Apartamento
0 - Sair
Opção:
```



## **Acréscimo de novos tipos de imóveis**

Para adicionar um novo tipo de imóvel seria necessária a definição de uma nova classe, onde estivessem contidas todas as funções que trabalhassem com toda a informação que o utilizador pretendesse para um novo imóvel, utilizando funções semelhantes as utilizadas nas classes anteriormente definidas facilitando assim a implementação da nova classe