ImOObiliaria

Universidade do Minho

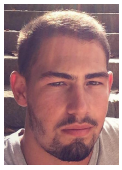
Licênciatura em Ciências da Computação

Grupo 6

21 de Maio de 2016

 João Gomes, a70400

 Luís Ventuzelos, a73002

 Paulo Barbosa, a70073

Resumo

Este relatório foi realizado no âmbito da unidade curricular de Programação Orientada aos Objetos, no curso Ciências da Computação da Universdade do Minho, sendo o trabalho proposto a componente prática da mesma, que envolve o desenvolvimento de uma aplicação que permite o registo e a venda de imoveis. Pretende-se, assim, dar uma ideia de todo o trabalho envolvido, nomeadamente da preparação, do modo de abordagem do problema e de alguns conteúdos, de modo a auxiliarem a compreensão deste trabalho.

Conteúdo

1 Introdução

Este trabalho foi realizado no âmbito da unidade curricular de Programação Orientada aos Objetos, do 2º ano da Licenciatura em Ciências da Computação da Universidade do Minho, com o objetivo de se desenvolver uma aplicação similar à "Imovirtual". Esta aplicação, desenvolvida na linguagem Java™ usando todos os métodos disponíveis até à versão 1.8.0, teria que realizar as seguintes tarefas:

- Registo de utilizadores
- Pesquisa de imóveis
- Venda de imóveis
- Favoritos
- Histórico

2 Estruturas de dados

2.1 Arquitetura de classes

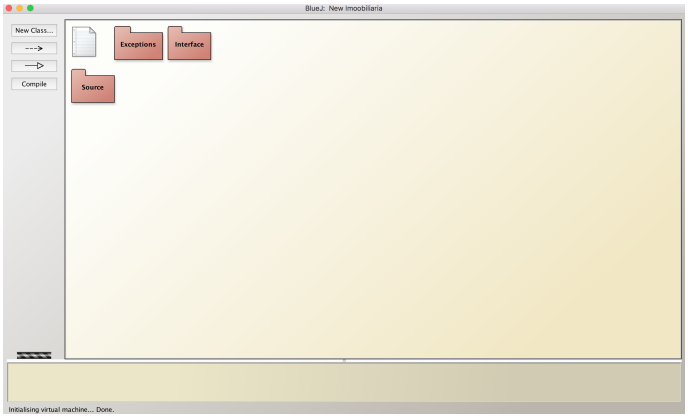


Figura 1: Print Screen BlueJ

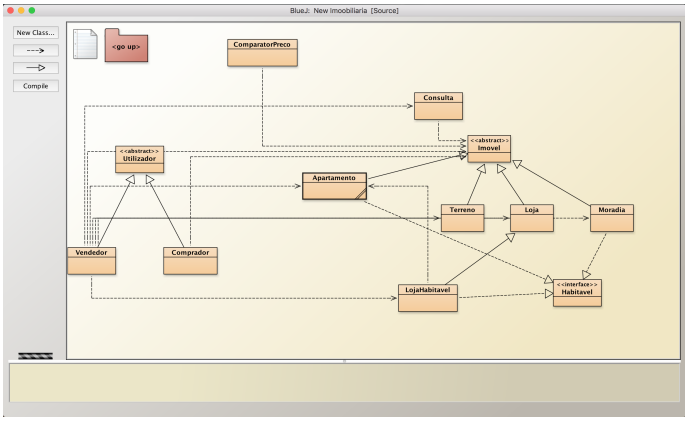


Figura 2: Print Screen BlueJ - Source Package

2.1.1 Classe Utilizador

Esta classe permite criar utilizadores,com os seguintes dados:

- E-mail
- Nome
- Password

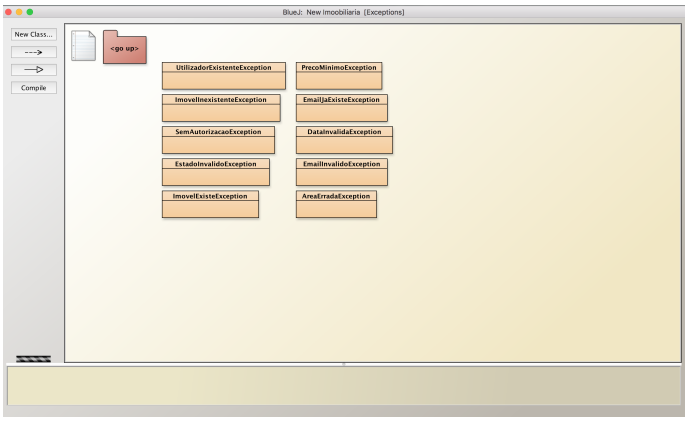


Figura 3: Print Screen BlueJ - Exceptions Package

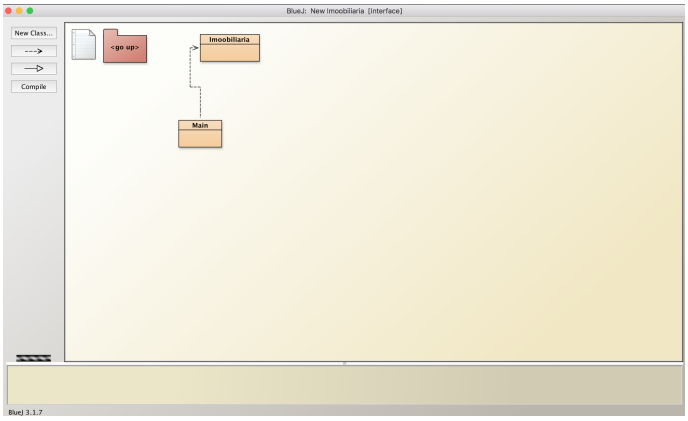


Figura 4: Print Screen BlueJ - Interface Package

- Morada
- Username
- Data de nascimento

Nesta classe não há a distinção entre comprador e vendedor pois essa distinção vai ser feita nas classes Vendedor e Comprador. Essa duas classe herdam tudo o que é feito na classe Utilizador, apenas acrescentando certos dados e permissões ao perfil de utilizador, tais como:

- Lista de imóveis vendidos (no caso de Vendedor)
- Lista de imóveis em venda (no caso de Vendedor)
- Lista de imóveis favoritos (no caso de Comprador)

- Lista de imóveis pesquisados (no caso de Comprador)

2.1.2 Classe Imoobiliaria

É nesta classe que definimos as estruturas de dados da aplicação, e que temos todos os métodos necessários para registar um utilizador, iniciar sessão, registar imóveis, etc.

Por quê usar um HashMap para guardar todos os utilizadores ?

Para guardar os utiliadores escolhemos usar um HashMap onde cada e-mail corresponde ao utilizador respetivo. Achamos que esta seria a melhor hipotese de guardar os utilizadores da aplicação.
Cada tipo de utilizador (Vendedor/Comprador) está guardado num ArrayList de Vendedores/Compradores, ou seja, o ArrayList de Vendedores e o ArrayList de Compradores são os dois um ArrayList de HashMaps. Os imóveis existentes na aplicação são guardados num HashMap, no qual as chaves correspondem ao ID do imóvel em questão.

Variáveis de Instância:

```
public HashMap<String,Utilizador> utilizadores; //Email -> utilizador
public ArrayList<Vendedor> vendedores;
public ArrayList<Comprador> compradores;
public HashMap<String,Imovel> imoveis; //ID do imovel Lista de todos
    os imoveis publicados
private String email;
```

2.1.3 Classe Imovel

Nesta classe estão definidos todos os construtores para podermos criar imóveis. Cada imóvel tem os seguintes dados:

- Id
- Rua
- Estado (em venda, vendido, reservado)
- Preço pedido
- Preço minimo (o comprador não tem permissão para o saber)
- Distrito

Por outro lado temos também vários tipos de imóveis que, além de terem as variáveis mencionadas acima, cada um tem as suas especificações, sendo eles:

- Moradia
- Apartamento

- Loja
- Terreno
- Loja Habitavel

2.1.3.1 Moradia

Na classe Moradia o objetivo é implementar os dados associados a esta subclasse, que neste caso serão:

- Tipo (geminada, banda, gaveto)
- Rua
- Área de implantação
- Área total coberta
- Área do terreno
- Número de quartos
- Número de WC
- Numero da porta

Exemplo de moradia:

```
----- Moradia -----
ID: 34
Rua: Rixa
Preço Pedido: 200000
Estado do Imóvel: Em Venda
Distrito: Porto
Tipo: geminada
Area de Implantação: 120
Area Total Coberta: 100
Area do Terreno Envolvente: 20
Número de Quartos: 6
Número de WC's: 3
Número da porta: 18
```

Figura 5: Print Screen BlueJ - Moradia

2.1.3.2 Apartamento

Na classe Apartamento o objetivo é implementar os dados associados a esta subclasse, que neste caso serão:

- Tipo
- Área Total

- Número de quartos
- Número de WC
- Numero da porta
- Andar
- Garagem (se tem ou não)

Exemplo de apartamento:

```
----- Apartamento -----
ID: 6
Rua: rua do Local
Preço Pedido: 75000
Estado do Imóvel: Em Venda
Distrito: Braga
Tipo:simples
Area Total50
Número de quartos:3
Número de WCs:2
Porta:7
Andar:2
Garagem:Sim
```

Figura 6: Print Screen BlueJ - Classe Apartamento

2.1.3.3 Loja

Na classe Loja o objetivo é implementar os dados associados a esta subclasse, que neste caso serão:

- Área total
- WC (se tem ou não)
- Tipo (tipo de negócio praticado na loja)
- Número da porta

2.1.3.3.1 Loja Habitavel Na classe Loja Habitavel o objetivo é fazer uma união das classes Loja e Apartamento, implementando os dados associados a esta subclasse de Loja, que neste caso serão:

- ID
- Rua
- Preço Pedido

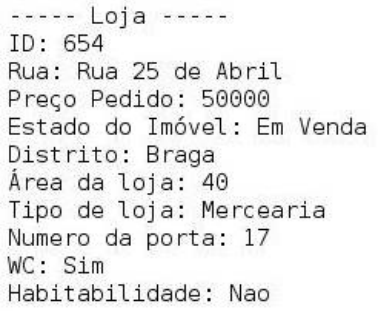


Figura 7: Print Screen BlueJ - Classe Loja

- Estado do Imóvel (Em venda/Vendido/Reservado)
- Distrito
- Área da loja
- WC
- Tipo de negócio
- Número da porta
- Habitabilidade
- Tipo (Simples/Duplex/Trilex)
- Número de quartos
- Número de WCs
- Garagem (se tem ou não)

2.1.3.4 Terreno

Na classe Terreno o objetivo é implementar os dados associados a esta subclasse, que neste caso serão:

- Área total
- Diâmetro da canalização
- Rede Elétrica (se tem ou não)
- Capacidade máxima suportada pela rede elétrica (caso instalada)
- Tipo (contrução de armazém ou de habitação)

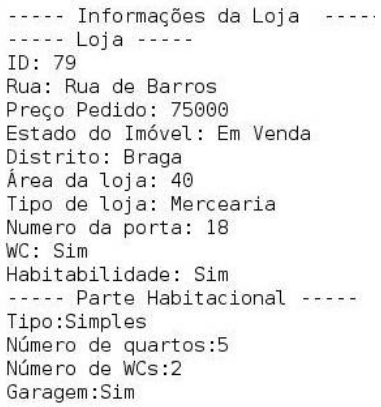


Figura 8: Print Screen BlueJ - Classe Loja Habitavel

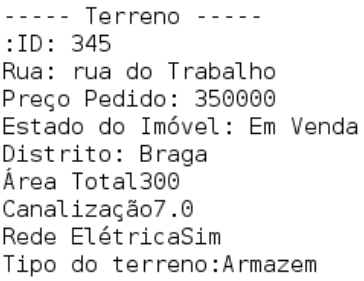


Figura 9: Print Screen BlueJ - Classe Terreno

2.1.4 Classe Consulta

Nesta classe, o objetivo é implementar os dados relativamente a uma consulta de um utilizador. Assim, um Vendedor conseguia saber quais os Compradores que consultaram os seus imóveis em venda.

3 Exceptions

Tivemos a necessidade de criar as seguintes exceções:

- SemAutorizacaoException
- ImovelInexistenteException
- UtilizadorExistenteException
- EstadoInvalidoException
- ImovelExisteException

- DataInvalidaException
- EmailInvalidoException
- EmailJaExisteException

3.1 SemAutorizacaoException

Esta **exception** é responsável garantir que um certo utilizador tenha, ou não, permissão para executar um determinado método.

3.2 ImovelInexistenteException e ImovelExisteException

Esta **exception** é responsável garantir que um certo imóvel está, ou não, presente nas estruturas de dados implementadas(HashMap e ArrayList).

3.3 UtilizadorExistenteException

Esta **exception** é responsável garantir que um certo utilizador está, ou não, presente nas estruturas de dados implementadas(HashMap e ArrayList).

3.4 EstadoInvalidoException

Esta **exception** é responsável garantir que um certo imóvel que já tenha sido vendido não apareça na lista de imóveis em venda.

3.5 DataInvalidaException

Esta **exception** testa se uma data está correta, isto é:

- Um dia está entre 1 e 31;
- Um mês está entre 1 e 12;
- Um ano tem que ter 4 algarismos

3.6 EmailInvalidoException

Esta **exception** é responsável garantir que um certo email tem o caracter '@' e o caracter '.'.

3.7 EmailJaExisteException

Esta **exception** é responsável por verificar se algum dos utilizadores resgistados tem um dado email.

4 Interface Habitável

Esta interface implementa os métodos em comum com todo o tipo de imóveis que são habitáveis (**Apartamento**,**Moradia** e **LojaHabitavel**). Utilizamos apenas os métodos **getQuartos()** e **getWcs()** pois são estes os que distinguem um imóvel habitável de um imóvel não habitável. Assim, temos de adicionar estes dois métodos às nossas classes habitáveis, **Apartamento**, **Moradia** e **LojaHabitavel**.

5 Conclusão

Neste trabalho conseguimos por em prática os conhecimentos adquiridos nas aulas, de forma a fortalecer o esse conhecimento para uma melhor preparação para o teste final da disciplina. Um trabalho desta dimensão claro que ainda tem alguns pontos a melhorar, pois não conseguimos cumprir todos os objetivos que nos foram propostos, tais como os métodos da classe **Consulta**(que no nosso caso está a imprimir todas as consultas feitas na aplicação), mais exceções de modo a evitar alguns erros de apresentação, optaríamos por implementar uma classe **Menus**, de modo a não sobrecarregarmos tanto a **Main**, como também a classe **Leilões**. No entanto, achamos que sem este trabalho ia ser muito mais complicado fazermos um bom estudo para esta unidade curricular.