Universidade do Minho
Escola de Engenharia

Programação Orientada aos Objetos, POO

Relatório do projeto prático

IMOOBILIARIA

Grupo de Trabalho 34


Ana Esmeralda Fernandes A74321


Diogo Alexandre Machado A75399


Miguel Dias Miranda A74726


Rui Filipe Castro Leite A75551

Mestrado Integrado em Engenharia Informática

Maio de 2016

Índice

Breve descrição

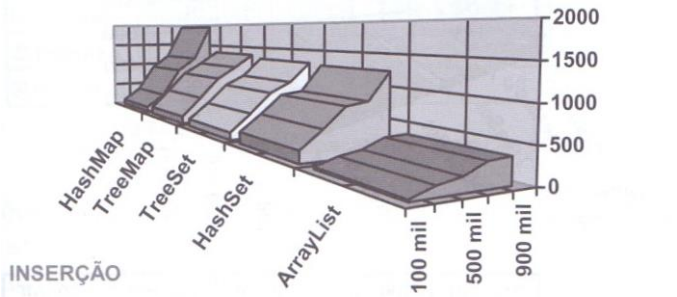
Foi proposto a elaboração de uma aplicação em JAVA™ que permita a simulação e gestão de uma Imobiliária, através do registo de Vendedores e Compradores e de vários tipos de Imóveis (nomeadamente, Moradias, Apartamentos, Lojas e Terrenos).

Para cada utilizador existem funcionalidades que vão de acordo com o seu estatuto (comprador/vendedor), possibilitando a criação de um processo que descreve o ciclo de vida de um imóvel, desde o seu anúncio até à sua venda.

No desenvolvimento da aplicação foram tidos em conta alguns dos princípios fundamentais do JAVA™, nomeadamente:

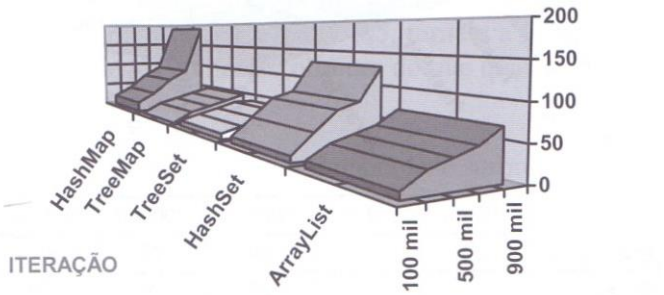
- Encapsulamento, através da criação de métodos que permitem o acesso e modificação dos atributos de uma classe;
- Herança, através do uso de superclasses que engloba métodos comuns a determinadas classes;
- Polimorfismo, invocando os mesmos métodos para diferentes subclasses.

Foi necessário também escolher algumas estruturas de dados para guardar os utilizadores, os imóveis, entre outros dados. Teve-se em conta a eficiência na inserção e consulta, conforme mostram as figuras seguintes.



	100 mil	300 mil	500 mil	700 mil	900 mil
ArrayList	30	140	200	340	360
HashSet	190	461	770	820	1250
TreeSet	130	430	770	950	1280
TreeMap	140	500	880	1100	1350
HashMap	100	370	930	1200	1950

Figura 2 – Medidas de desempenho na inserção



	100 mil	300 mil	500 mil	700 mil	900 mil
ArrayList	10	30	40	50	70
HashSet	10	40	70	80	130
TreeSet	10	20	30	40	40
TreeMap	0	20	30	40	50
HashMap	20	40	70	90	190

Figura 1 - Medidas de desempenho na consulta

Imagens retiradas do livro JAVA6 e programação orientada pelos objectos / F. Mário Martins

Arquitetura das classes utilizadas

ImoobiliariaAPP

A ImoobiliariaAPP é a classe responsável por tratar da interação entre o utilizador e a parte computacional da aplicação. Foi aqui que se implementaram os menus apresentados no Terminal e os interpretadores de comandos, bem como os métodos que permitem pré-popular a Imoobiliaria (quer através da leitura de um ficheiro de dados (`ObjectStream`) ou através do registo de determinados utilizadores e imóveis declarados no método `initApp()`).

Imoobiliaria

A classe Imoobiliaria é quem realmente define o objeto de uma imobiliária. Existem três variáveis de instância essenciais:

- `Map<String, Utilizador> utilizadores` – mapeamento entre *email* e utilizador (contém todos os utilizadores na aplicação);
- `Map<String, Imovel> imoveis` – mapeamento entre ID (referência) e imóvel (contém todos os imóveis na aplicação);
- `Map<String, Leilao> leiloes` – mapeamento entre ID do imóvel e o leilão (contém todos os leilões registados).

Além destas variáveis existem outras de controle, como `String atualUser` (*email* do utilizador com sessão iniciada), `boolean online` (indica se o utilizador está online), `int count` (indica quantos imóveis foram até ao momento inscritos – necessário para a criação de um ID de imóvel), `Vendedor admin` (administrador da aplicação, quem a pré-popula).

Nesta classe existem os métodos essenciais de computação, que permitem aceder a informações da imobiliária, tais como:

- Aceder aos utilizadores, imóveis, leilões e ao *email* do utilizador com sessão iniciada;
- Dado um email/ID devolver o Utilizador/Imóvel correspondente;
- Iniciar/fechar sessão e registar um utilizador;
- Aceder à lista dos imóveis com determinadas especificações (até um preço, até N consultas ou que implementam a interface `Habitavel`);
- Aceder à lista dos favoritos do `(Comprador)atualUser`;
- Gravar o estado do programa;

Imóvel

A classe Imóvel é uma superclasse que contém variáveis de instância importantes para todos os tipos de imóveis e métodos de acesso e modificação dessas mesmas variáveis. São elas:

- `String ref` – referência (ID) do imóvel;
- `String proprietário` – *email* do proprietário do anúncio do imóvel;
- `String rua`;
- `float precoPedido, precoMin` – preço pedido e preço mínimo aceitável pelo proprietário;

- `int consultas` – número de consultas ao imóvel;
- `String estado` – estado do imóvel (`"À venda"`, `"Reservado"`, `"Vendido"`);

Foi implementado um método, nesta classe, que permite gerar uma referência (um ID) para o imóvel em questão. Essa referência é gerada tendo em conta o tipo do imóvel e quantos imóvel foram registado até ao momento (exemplo de uma referência de uma Moradia: `Mor-0`; Apartamento: `Apar-1`; Loja: `Loj-2`; Loja Habitável: `LojH-3`; Terreno: `Terr-4`).

Foram criadas 4 classes correspondentes a cada tipo de imóvel: `Moradia`, `Apartamento`, `Loja` e `Terreno`, cada uma com variáveis de instância necessárias ao registo, conforme dito no enunciado do projeto. Foi considerada uma quinta classe, a `LojaHabitavel`, que é uma subclasse de `Loja` e que contém, existindo, a parte habitável de uma loja, i.e., o apartamento a que está associada.

Interface `Habitavel`

De forma a ser possível classificar os diferentes tipos de imóveis, foi criada uma interface `Habitavel` que, por opção, não contém qualquer variável de instância nem método. Apenas serve como indicador da habitabilidade de um imóvel. São considerados habitáveis os imóveis Moradia, Apartamento e Loja com parte habitável.

Utilizador

A classe Utilizador trata-se de uma superclasse abstrata cujas variáveis de instância são comuns a Vendedores e Compradores, como o Nome, *Email*, *Password*, Morada e Data de Nascimento. Permite, portanto, que estas duas classes sejam tratadas em comum, estabelecendo a relação de herança entre elas.

Comprador

De forma a dar resposta ao que foi solicitado, a classe Comprador possui um `TreeSet<String>` com os ID's dos imóveis marcados como favorito. A escolha desta estrutura de dados foi devida ao facto de não ser possível a existência de ID's iguais, pela facilidade e rapidez de acesso do `TreeSet<>` e pelo facto de não ser necessária uma correspondência entre chave-valor.

O Comprador pode:

- Definir um imóvel como favorito;
- Consultar os seus favoritos ordenados por preço (foi implementado um `ComparatorImovelPreco` para tal);
- Participar num leilão.

Vendedor

A classe Vendedor possui um `TreeSet<String>` com os ID's dos imóveis em portfólio (i.e., imóveis à venda) e no histórico (imóveis vendidos). Existe também um `ArrayList<Consulta>` com todas as consultas feitas por todos os utilizadores da aplicação aos imóveis em portfólio do respetivo Vendedor. A escolha do `ArrayList<>` deve-se ao facto de se tornar fácil a inserção, aquando de uma qualquer consulta, e da impressão ordenada no ecrã.

O Vendedor tem acesso a:

- Definir o estado de um determinado imóvel (`"À venda"`, `"Reservado"`, `"Vendido"`);

- Registar um imóvel;
- Aceder a todas as consultas feitas aos seus imóveis em venda;
- Aceder a um conjunto dos imóveis com mais de N consultas;
- Lançar um leilão.

Consulta

Para a implementação das consultas aos imóveis de um dado Vendedor foi necessário criar uma classe que representa de forma estruturada cada consulta. Para tal, implementou-se a classe `Consulta`, que contém a data a que a consulta foi realizada (`GregorianCalendar`), o *email* do utilizador que a fez (caso tenha sessão iniciada) e o ID do imóvel consultado.

Sempre que são chamados os métodos `getImovel(String classe, int preco)`, `getHabitaveis(int preco)` e `getMapeamentoImoveis()` é chamado um método auxiliar que regista a consulta (i.e., adiciona-a à lista do Vendedor proprietário do anuncio).

Leilão

A simulação dos leilões na aplicação é feita da seguinte forma: um vendedor regista um leilão de um determinado imóvel na sua posse, indicado por quanto tempo estará aberto. Depois de iniciado; os compradores que pretendem participar registam-se, depois de autenticados no programa; o vendedor, tendo participantes, pode iniciar a simulação do leilão.

A classe `Leilao` guarda informações como: data de encerramento do leilão, o ID do imóvel a leiloar, o *email* do vencedor (i.e., quem tem a maior licitação até ao momento), a duração do leilão e um mapeamento com todos os licitadores inscritos.

Quando um vendedor decide "arrancar" com o leilão, o programa entra num ciclo até à hora de encerramento e para cada licitador é-lhe "perguntado" se pretende licitar (i.e., se está na hora de licitar e se "vale a pena" fazê-lo). Em caso afirmativo, verifica-se se a licitação feita ultrapassa a maior até agora e atualizam-se os campos `licitacaoMaior` e `idVencedor`.

No final do leilão, tem-se o *email* do licitador com a maior aposta (obtido pelo método `encerraLeilao()`). Caso essa aposta satisfaça o preço mínimo exigido pelo vendedor do imóvel então é alterado o estado do imóvel leiloado para `"Reservado"`, caso contrário permanece inalterado.

Licitador

Para a implementação desta simulação foi criada uma classe `Licitador`, que representa um participante no leilão. Nesta classe existem as seguintes variáveis de instância e os seguintes métodos:

- `String idLicitador` – *email* do participante;
- `double limite` – valor limite, em €, que o participante está disposto a pagar;
- `double incrementos` – valor de cada incremento;
- `double minutos` – intervalo entre incrementos;
- `double licitacao` – valor atual da licitação (começa em 0);
- `GregorianCalendar ultLicitacao` – data da última licitação feita (ideal para determinar quando deverá ser a próxima);

- `Licitar(double licitacaoMaior, String idVencedor)` – responsável por fazer uma nova licitação. É usado um tratamento de erros: se retorna `-2`: ainda não está na hora de licitar; se `-1`: o licitador está na frente do leilão e, portanto, não interessa licitar; se `0`: ocorreu licitação.
  Caso a licitação atual do participante esteja "longe" da licitação maior no leilão, são feitos sucessivos incrementos até se cobrir a maior proposta (nunca ultrapassando o limite definido pelo participante);

Descrição da aplicação

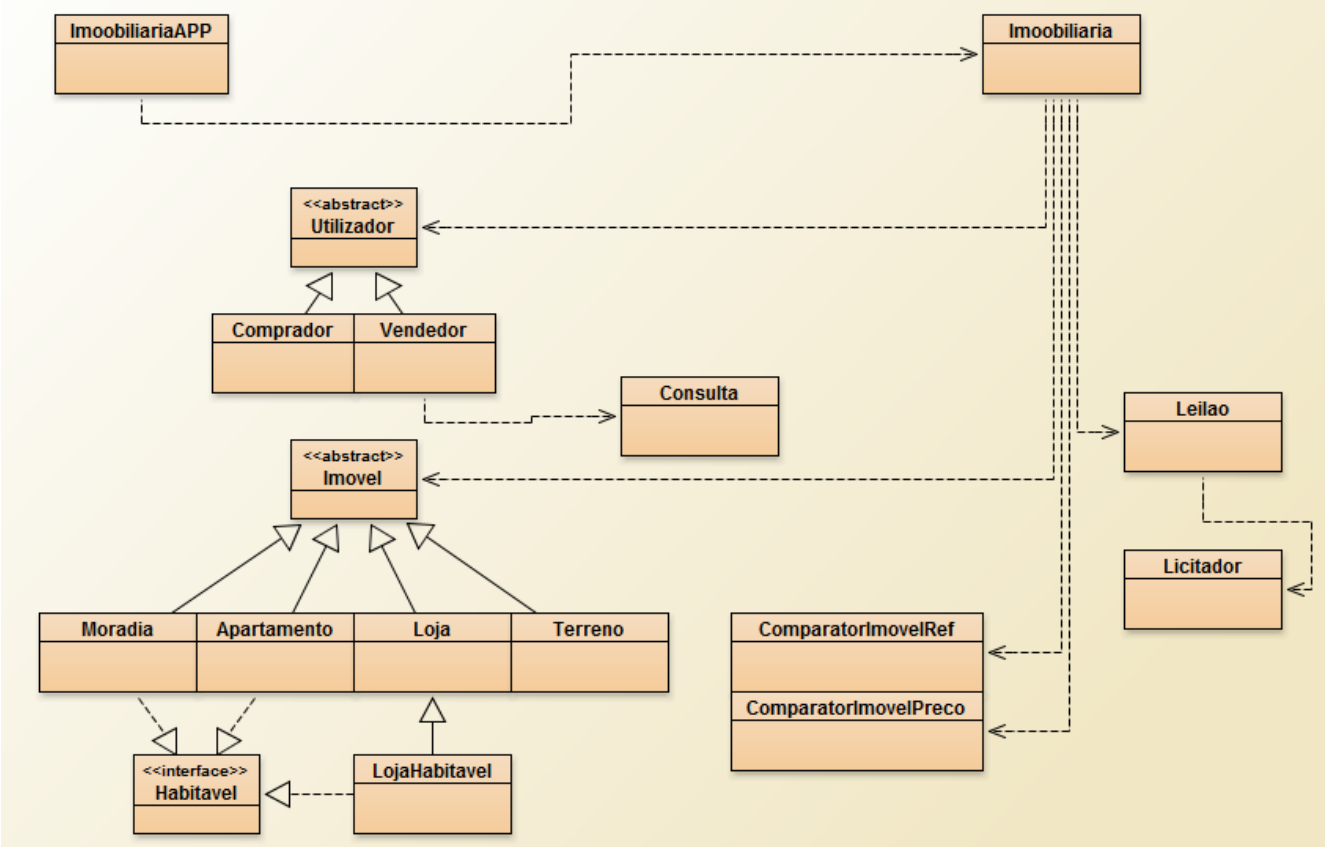


Figura 3 - Esquema representativo das classes da aplicação

A aplicação desenvolvida foi pensada de forma a ser interativa para com o utilizador que usufrui dela. Para tal, fez-se uso de menus e interpretadores de comandos.

Inicialmente é mostrado ao utilizador um menu principal onde é possível iniciar sessão, fazer um registo, aceder a funcionalidades gerais ou gravar o atual estado do programa.

```
- - - - - - - - - - - - - - - - - - - - - -
|                                         |
|              IMOOBILIARIA               |
|                                         |
- - - - - - - - - - - - - - - - - - - - - -

(1) - Iniciar sessão
(2) - Fazer registo

(3) - Mais opções

(4) - Guardar estado do programa
```

Tendo a sessão iniciada, dependendo se é Comprador ou Vendedor, tem-se acesso a diferentes funcionalidades

```
Bem-vindo(a) (Nome do Vendedor)
(1) - Registar imóvel
(2) - Listar 10 últimas consultas aos imóveis em venda
(3) - Alterar o estado de um imóvel
(4) - Apresentar os códigos dos seus imóveis com mais de N consultas
(5) - Criar leilão
(6) - Inicar leilao
(7) - Mais opções

(0) - Fechar sessão
```

Um Vendedor tem acesso à gestão, propriamente dita, da Imobiliária como: o registo de um Imóvel, acesso a informações estatísticas e a venda num Leilão. Em qualquer parte da aplicação é possível aceder a funcionalidades passíveis de serem usadas por qualquer utilizador (autenticado ou não). No menu do Vendedor, é possível aceder a estas opções através do comando `7`.

```
Bem-vindo(a) Nira Fernandes
(1) - Registar um imóvel como favorito
(2) - Consultar imóveis favoritos ordenados por preço
(3) - Participar num leilão
(4) - Mais opções

(0) - Fechar sessão
```

É permitido a um Comprador marcar imóveis como favoritos, consultá-los e, ainda, participar num qualquer leilão lançado por um Vendedor. Além disso, tem acesso às opções gerais.

Utilizadores e imóveis para teste

Para o teste da aplicação foram declarados alguns compradores e vendedores e também alguns imóveis, de forma a que seja possível testar todas as funcionalidades numa primeira abordagem.

	Email	*Password*
Vendedores	admin@email.com	123
	esmeralda@email.com	esmeralda
	diogo@email.com	diogo
	miguel@email.com	miguel
	rui@email.com	rui
Compradores	maria@email.com	maria
	anamarques@email.com	ana
	nadine@email.com	nadine
	nira@email.com	nira

	Vendedor	*Referência*
Imóveis	admin@email.com	Mor-0
	admin@email.com	Mor-1
	admin@email.com	Apar-2
	admin@email.com	Apar-3
	admin@email.com	Loj-4
	admin@email.com	Loj-5
	admin@email.com	LojH-6
	admin@email.com	Terr-7

Inclusão de novos tipos de imóveis

O desenvolvimento do projeto foi pensado de forma a que fosse possível a inclusão de novos tipos de imóveis, através do principio da modularidade. Para tal, teve-se em conta os seguintes aspetos:

- Não repetir partes do código em diferentes métodos, de forma a que uma alteração no código não implique múltiplas modificações;
- Utilização de métodos auxiliares que permitam melhor legibilidade por quem tiver que rever ou acrescentar código ao projeto;
- Separação do projeto em diferentes e sucintas classes. Havendo necessidade de acrescentar novas classes, torna-se simples a integração na aplicação (principalmente pelo uso das superclasses e pelos métodos que podem ser usados pelas várias subclasses).

Assim, a inclusão de novos tipos de imóveis pode ser feita através da criação das respetivas classes, que serão extensão da superclasse `Imóvel` (abstrata), sendo necessário poucas alterações, como por exemplo, criar um novo "formulário" de registo e a referência nos menus da aplicação.

Conclusão

Através da realização deste projeto, tivemos a oportunidade de fortalecer os nossos conhecimentos na linguagem JAVA™, bem como nos princípios de encapsulamento, polimorfismo, modularidade e criação de diferentes objetos.

Em particular, conseguimos assimilar conhecimentos no principio de herança e no desenvolvimento de superclasses e de subclasses, e no uso do operador `super()`.

Em relação à parte complementar do projeto, os Leilões, consideramos a tarefa relativamente bem conseguida.