Relatório do Projeto de Programação Orientada aos Objetos: ImOObiliária



Grupo 44:

- A73580, Maria Ana de Brito
- A75135, Bruno Pereira
- A76407, Diogo Silva

2015/2016

Introdução

O trabalho prático ImOObiliária consiste no desenvolvimento de uma aplicação que permite simular uma agência imobiliária e todos os processos que a compõe, desde o armazenamento de imóveis às atividades sobre elas efetuadas pelos diversos atores.

Primeiramente, existem os imóveis. Os imóveis podem ser de diversos tipos, tais como uma moradia, um apartamento, uma loja (habitável ou não) e um terreno. Cada um destes imóveis têm particularidades intrínsecas, no entanto, todos partilham o facto de serem definidos pela rua onde se situam, o preço pedido por eles, e o preço mínimo aceite pelo proprietário.

Prosseguindo para os atores, estes podem ser definidos como três tipos: os vendedores, os compradores registados, e os utilizadores não registados. Os primeiros dois estão registados no sistema através de características pessoais, enquanto que o último não se encontra registado no sistema. Ainda relativamente aos atores, temos que estes partilham também do uso de certas funcionalidades, sendo elas a consulta de imóveis (exemplo: obter imóveis de um certo tipo, obter todos os imóveis, obter imóveis habitáveis).

Relativamente aos compradores (registados), estes podem, além de efetuar consultas sobre os imóveis, registar um certo tipo de imóvel como favorito para, mais tarde, poder ver o conjunto dos seus imóveis favoritos.

Relativamente aos vendedores, estes podem, tal como os compradores, efetuar consultas sobre os imóveis e ainda são definidos pelo seu portfólio de imóveis em venda e pelo histórico de imóveis vendidos. Estes atores são capazes de registar um imóvel na imobiliária, obter uma lista das últimas dez consultas dos seus imóveis, alterar o estado de um imóvel, e obter os códigos dos imóveis mais consultados.

Tudo atrás referido chega para implementar uma imobiliária. No entanto, ainda é pedido, como valorização, a realização de uma simulação de leilões.

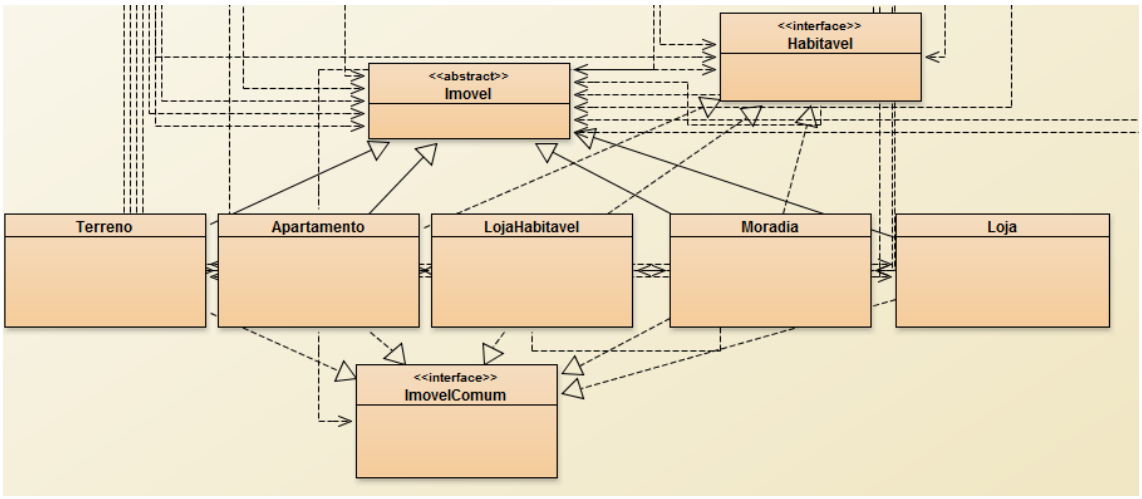
A simulação de leilões consiste no vendedor colocar um imóvel à venda. Durante determinado intervalo de tempo os compradores poderão licitar sobre esse imóvel. Se aquando o fim do tempo do leilão, qualquer das licitações (deve-se atender à maior, obviamente) superarem o preço mínimo pedido, o imóvel passa a estar reservado, faltando unicamente efetuar a compra. Se a maior licitação for menor que o preço mínimo pedido, o imóvel não é vendido.

Resumindo, este foi o trabalho que nos foi proposto, que visa cimentar os conhecimentos da Programação Orientada aos Objetos obtidos ao longo do semestre.

Descrição das Classes

No projeto ImOObiliária temos diversas classes e uma hierarquia definida entre elas. Perante este facto, vamos considerar certas classes como um todo para demonstrar as dependências e as semelhanças entre elas.

Classes que implementam um imóvel



As várias classes que implementam um imóvel

Neste caso, temos uma classe abstrata que é implementada pelas diferentes classes representativas dos vários tipos de imóveis. A nossa escolha para uma classe abstrata Imovel, baseou-se na necessidade de obter métodos gerais à classe, mas diferentes em cada classe pela qual é implementada. Ou seja, existem métodos cuja assinatura está na classe abstrata, mas que estão definidos de maneira diferente para cada subclasse. Um exemplo de um destes métodos é o método *toString* que tem diferentes implementações para cada subclasse, pois o que define um apartamento não define (necessariamente) uma moradia.

As variáveis de instância da classe Imovel são a rua do imóvel, o preço pedido pelo imóvel, e o preço mínimo pedido pelo imóvel. O acesso a esta última variável vai ser restrito, tornando-se mais forte a escolha de uma classe abstrata.

Subclasses da classe Imóvel

- **Classe Moradia:** classe que implementa uma moradia com a área de implantação, a área coberta, a área envolvente, o número de quartos, o número de WCs, o número da porta, e o tipo da moradia.

- **Classe Apartamento:** classe que implementa um apartamento. Esta classe contem o tipo, o número de quartos e WCs, o número da porta, o andar, a área total, e se possui garagem.

- **Classe Loja:** classe que define uma loja não habitável. Esta classe contém o tipo de negócio, a área, se possui WC e o número da porta.

- **Classe LojaHabitavel:** classe que implementa uma loja habitável que difere da loja apenas num sentido: é possível habitar a loja, logo vai herdar as características dos apartamentos.

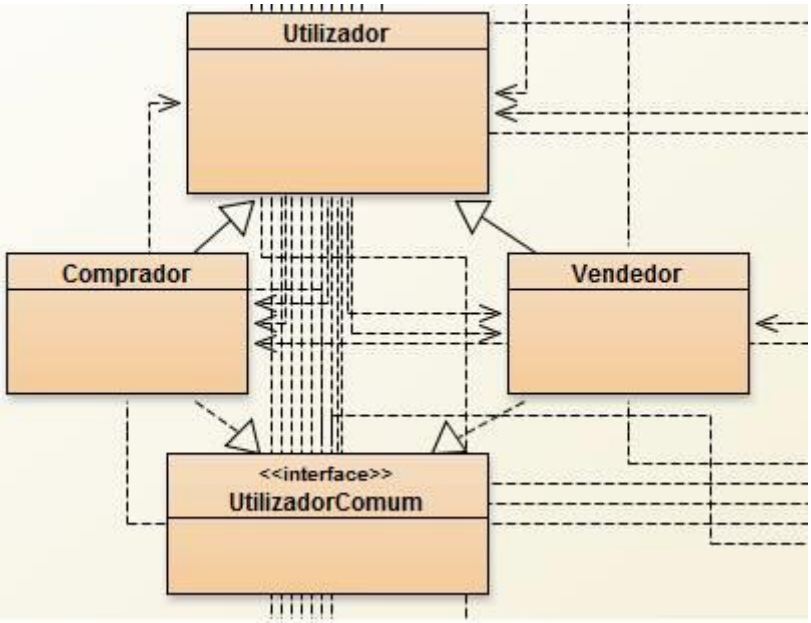
- **Classe Terreno:** classe que implementa um terreno em que se conhece se o terreno é próprio para construção, o diâmetro das canalizações (em milímetros), os kWh máximos suportados pela rede elétrica (se instalados), e se existe o acesso à rede de esgotos.

Estas subclasses estendem uma interface Habitavel, que define se elas são habitáveis. Também possui um método para cópia de imóveis habitáveis.

Finalmente, todas as subclasses estendem uma interface, ImovelComum, que contém métodos iguais a todas essas subclasses, tais como, passar o imóvel para uma *string*, construir a identificação referente a cada imóvel, e um verificar a igualdade entre dois imóveis.

É de referir que o ID de cada um dos imóveis é sempre diferente e é o número total de imóveis registados no sistema, tanto estejam nos portfólios como nos históricos. Logo, pode-se pensar nele como o número de ordem de entrada no sistema.

Classes que implementam um utilizador



Classes que implementam um utilizador

Estas classes implementam um utilizador. A nossa escolha para dividir o Utilizador em duas classes mais específicas foi feita no facto de existirem dois tipos diferentes de atores neste projetos, os compradores e os vendedores. Deste modo, estas duas classes são uma especificação da classe Utilizador, na medida que contêm informação própria não partilhada.

Subclasses da classe Utilizador

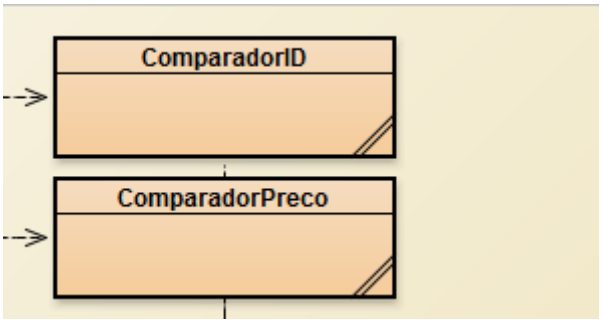
- **Classe Utilizador:** temos a classe mais geral (Utilizador) que contém os parâmetros comuns aos dois tipos de utilizadores, nomeadamente o e-mail, o nome, a password, a morada e a data de nascimento.

- **Classe Comprador:** implementa um tipo específico de utilizadores: os compradores, que têm como característica específica uma lista de imóveis favoritos.

- **Classe Vendedor:** implementa o outro tipo específico de utilizadores: os vendedores, que têm como característica própria uma lista de imóveis em venda, imóveis vendidos e consultas dos imóveis.

- **Interface UtilizadorComum**: declara todos os métodos comum aos compradores e aos vendedores.

Comparadores de Imóveis

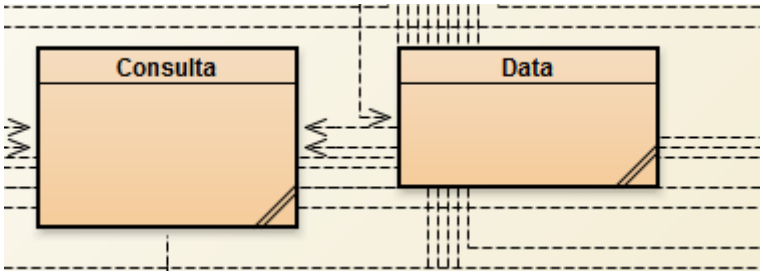


Comparadores de imóveis

Os comparadores de imóveis são importantes na organização dos imóveis nas classes que os implementam. Os comparadores possuem um método que compara certos parâmetros da classe Imovel, tanto o preço como a identificação, e retorna um valor dependendo da comparação entre imóveis.

Estas classes são importantes quando queremos, por exemplo, implementar um TreeSet e o queremos organizar por, no nosso caso, ou pelo preço, ou pela identificação.

Classes Consulta e Data



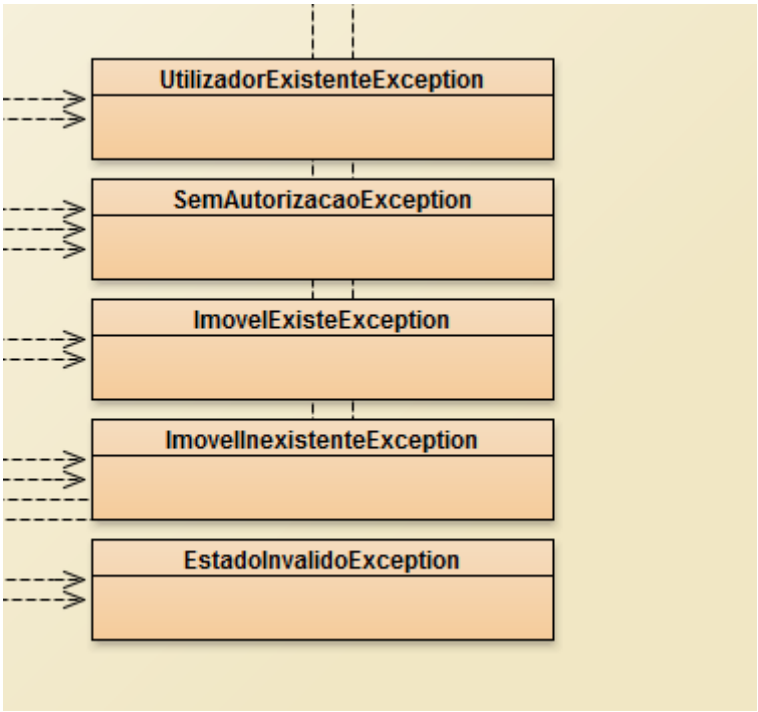
Consulta e Data

A classe Data, conforme o próprio nome, implementa uma data. Possui unicamente uma string que indica uma data (com ano, mês, dia, horas, minutos e segundos) e métodos que implementam essa classe. Também possui um método importante que calcula a data atual e a converte para uma string.

A classe Consulta é bem mais complexa que a anterior. Possui uma identificação do comprador que a visualizou, a data da visualização, o imóvel visualizado, e o número de vezes que foi visualizado.

Esta classe será importante para o vendedor, pois com esta classe ele terá o acesso às consultas dos seus imóveis.

Exceções

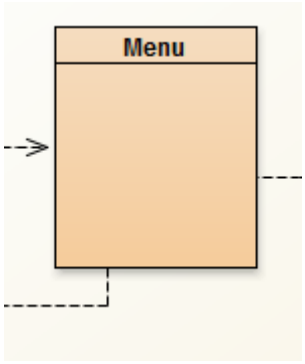


Classes com exceções

Estas classes permitem implementar as eventuais exceções que podem acontecer ao longo da execução da aplicação. Eis as várias classes que implementam uma exceção e a descrição de cada:

- **UtilizadorExistenteException:** O utilizador já existe e não pode ser registado.
- **SemAutorizacaoException:** O utilizador não tem autorização para executar tal método.
- **ImovelExisteException**: O vendedor está a tentar inserir um imóvel existente.
- **ImovelInexistenteException:** O utilizador está a tentar aceder a um imóvel que não existe.
- **EstadoInvalidoException:** O vendedor está a tentar definir um estado do imóvel que não é válido.

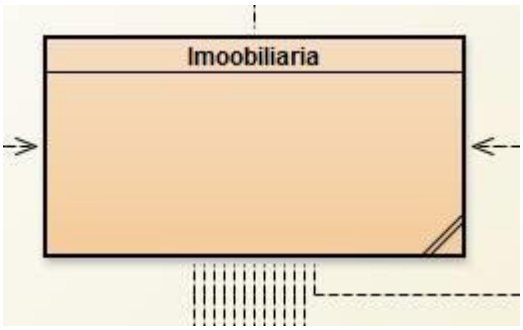
Classe que implementa um menu



Classe que implementa o menu

Esta classe vai ser uma classe muito importante para a aplicação. É nesta classe que estão os métodos necessários a compor um menu que será executado pela aplicação. Esta classe contém os métodos que permitem criar, mostrar e executar, o menu, que permitem ler opções do menu, e que permitem obter opções do menu. As variáveis de instância desta classe são uma lista de strings (correspondente às várias opções do menu) e um inteiro para as operações.

Classes que implementam a Imoobiliaria



Classe que implementa a Imoobiliaria

Esta classe é a que vai chamar todas as outras classes para desenvolver os métodos solicitados no enunciado do projeto, como, por exemplo, a classe Comprador, a classe Vendedor, a classe Consulta, entre outras. Deste modo, todas as estruturas construídas nessas classes refletir-se-ão nesta classe.

Assim, temos um Set de Compradores, um Set de Vendedores, um Comprador, um Vendedor, uma associação sob a forma Map entre os Ids dos imóveis e as suas consultas atualizadas, um número inteiro log que indica qual o tipo de utilizador que está dentro do sistema naquele momento (0 indica que é um utilizador anónimo, 1 indica que é um vendedor, 2 indica que é um comprador) e, finalmente, outro número inteiro quantos que indica o total de imóveis registados no sistema.

Ainda se teve o cuidado de declarar os métodos acessíveis por outras classes como *public* e os métodos puramente auxiliares como *private*.

É de referir que o grupo decidiu usar duas variáveis para identificar os diferentes tipos de utilizadores, em vez de uma única variável Utilizador u, para facilitar a compreensão do código e evitar o uso de castings.

Métodos relevantes da classe Imoobiliaria

- **iniciaSessao:** este método inicia a sessão de um comprador ou vendedor, que têm de estar registados no sistema. Caso contrário, o método lança uma exceção que indica que o utilizador não tem autorização para iniciar a sessão, o mesmo acontecendo para utilizadores anónimos. Quando o utilizador é um comprador (com e-mail e password corretos), o valor de log passa a ser 1 e a variável Vendedor v toma o valor do dito vendedor. Quando o utilizador é um comprador, o log passa a ser 2 e a variável Comprador c toma o valo do dito comprador.

- **fechaSessao:** garante que no sistema nunca vão estar um vendedor e um comprador ligados ao mesmo tempo. Quando um utilizador decide encerrar a sessão, a sua variável passar a ser genérica e o valor de log passa a corresponder a um utilizador anónimo.

- **registarUtilizador:** este método adiciona um utilizador, caso este não exista no sistema, à lista de compradores se este for um comprador ou à lista de vendedores se este for um vendedor.

- **registaImovel:** este método só pode ser acedido pelos vendedores e só deixa acrescentar um imóvel, caso este não exista já no sistema. O imóvel será adicionado à lista de imóveis em venda (portfólio) do respetivo vendedor.

- **setEstado:** este método, à semelhança do anterior, só pode ser acedido por vendedores. Existem dois estados disponíveis: **Vende-se** (quando se pretende passar um imóvel Vendido a Vende-se) e **Vendido** (quando se pretende passar um imóvel Vende-se a Vendido). Desta forma, quando um imóvel passar a Vende-se, o seu estado é alterado, ele é adicionado no portfólio do vendedor e é removido do histórico do vendedor, para evitar duplicações. Por outro lado, quando um imóvel passar a Vendido, o seu estado é alterado, é adicionado ao histórico do vendedor e removido do portfólio do mesmo vendedor.

- **getMapeamentoImoveis:** este método cria uma associação entre cada imóvel e o respetivo vendedor. Só se consideram imóveis que esteja em venda, ou seja, que estejam nos portfólios dos vendedores. Antes do término deste método, é feita a atualização (ou adição) das consultas com a associação que foi formada.

- **getHabitaveis:** este método forma uma lista com todos imóveis que são considerados como habitáveis (neste caso, são as moradias, os apartamentos e as lojas habitáveis) até um determinado preço, não inclusive. Um imóvel é considerado habitável se implementar a interface Habitavel, ou seja, se for instância desta interface. Antes do término deste método, é feita a atualização (ou adição) das consultas com a lista que foi formada.

- **getImovel:** este método forma uma lista com todos os imóveis de uma determinada classe até um certo preço, não inclusive. Se a classe não for válido(isto é se não for igual a uma das seguintes classes: Moradia, Apartamento, Loja, LojaHabitavel, Terreno), é devolvida uma lista vazia. Antes do término deste método, é feita a atualização (ou adição) das consultas com a lista que foi formada.

- **setFavorito:** este método só pode ser acedido por compradores, se não é lançada uma exceção que indica que o utilizador não tem permissão para fazer tal operação. Também é lançada uma exceção se o utilizador inseriu um Id não existente. Deste modo, quando se tiver a confirmação que o imóvel existe e o utilizador tem permissão, o imóvel é adicionado à lista de favoritos que chamou este método.

- **getFavoritos:** este método só pode ser acedido por compradores, se não é lançada uma exceção que indica que o utilizador não tem permissão para fazer tal operação. É, então, criado um Set com todos os imóveis favoritos do comprador que o chamou este método. Antes do término deste método, é feita a atualização (ou adição) das consultas com o conjunto que foi formado.

- **atualizaConsultasgetImovel:** este método vai a cada imóvel da lista recebida e, se o encontrar na lista de consultas, incrementa o contador e atualiza a data com a data atual. Caso contrário, é criada uma nova associação com a data atual e o contador com o valor 1.
- **atualizaConsultasgetHabitaveis:** este método vai a cada imóvel habitável da lista recebida, converte-o para um imóvel e, se o encontrar na lista de consultas, incrementa o contador e atualiza a data com a data atual. Caso contrário, é criada uma nova associação com a data atual e o contador com o valor 1.

- **atualizaConsultasgetFavoritos:** este método vai a cada imóvel do conjunto recebido e, se o encontrar na lista de consultas, incrementa o contador e atualiza a data com a data atual. Caso contrário, é criada uma nova associação com a data atual e o contador com o valor 1.

- **atualizaConsultasMapeamento:** este método vai a cada entrada da associação recebida e retira o id do imóvel, se o encontrar na lista de consultas, incrementa o contador e atualiza a data com a data atual. Caso contrário, é criada uma nova associação com a data atual e o contador com o valor 1.

  Nota: É de referir que estes métodos de atualização das consultas estão a operar sob uma associação entre o Id do imóvel e a respetiva consulta, por isso quando se diz que se cria uma nova associação com a data atual e o contador com o valor 1, os outros elementos da associação são também adicionados, apenas decidiu-se não os referir na definição desses métodos, pois tal referência está implícita.

  Refere-se ainda que o grupo primeiramente estava a utilizar um ciclo for, do tipo for(Map.Entry<String, Consulta> e: l.entrySet()), porém foi detetado o erro "java.util.ConcurrentModificationException: null" e, após uma breve pesquisa sobre a origem do erro, foi decidido usar um iterador para fazer a travessia da associação.

- **getConsultas:** este método cria uma lista com 10 consultas (ou menos, caso o tamanho da lista de consultas seja menor do que 10) com as consultas mais recentes aos imóveis do vendedor que chamou este método. É, por tanto, feita uma comparação entre as datas das consultas da lista.

- **getTopImoveis:** este método devolve a lista de todos Ids dos imóveis com mais de N consultas (não inclusive) do vendedor em questão. É devolvida uma lista vazia, caso nenhum imóvel corresponda ao requisito referido.

Descrição da aplicação ImoobiliariaAPP

A classe ImoobiliariaAPP é a classe que estabelece a interação entre o utilizador (a pessoa a executá-la) e todos os métodos referidos anteriormente.

Primeiramente, é necessário demonstrar o menu das opções para depois se fazer uma correta análise destas.

```
 *** Menu ***
1 - Registar Utilizador
2 - Iniciar Sessão
3 - Consultar a lista de todos os imóveis de um certo tipo até um dado preço
4 - Consultar a lista de todos os imóveis habitáveis até um dado preço
5 - Obter um mapeamento entre todos os imóveis e os seus vendedores
6 - Colocar um imóvel à venda
7 - Visualizar a lista das 10 últimas consultas aos imóveis para venda
8 - Alterar o estado de um imóvel
9 - Marcar um imóvel como favorito
10 - Consultar a lista dos imóveis preferidos ordenados pelo seu preço
11 - Obter o top de imóveis
12 - Encerrar Sessão
0 - Sair
Opção:
```

Menu principal

Este é o menu principal que pode visto por qualquer utilizador. No entanto, há opções exclusivas de certo tipo de utilizadores que se tentarem ser acedidas por utilizadores sem permissão imprimem mensagens de erro. Esta leitura das opções é feita escrevendo um número (através do teclado) que será interpretado e executado pela aplicação.

Opções do Menu Principal

Registar Utilizador

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

Menu de registo de um utilizador

Esta opção permite registar um utilizador na imobiliária dando a escolha de se registar um vendedor ou comprador. Independentemente da escolha, o menu que surgirá a seguir irá pedir ao utilizador para escrever no teclado o seu email, o seu nome, a password, a morada, e a sua data de nascimento.

Iniciar Sessão

```
Opção: 2
Insira o e-mail:
mailgenerico@enderecogenerico.com
Insira a password:
1234
```

Menu para iniciar a sessão

Esta opção permite ao utilizador iniciar a sessão, podendo aceder assim às características que o identificam. A partir deste momento há métodos que passarão a ser exclusivos ao tipo de utilizador que a eles acede.

Consultar a lista de todos os imóveis de um certo tipo e até um certo preço

```
Opção: 3
Insira a classe:
Moradia
Insira o preço:
1200000
```

Menu da opção 3

Esta opção a qualquer utilizador, registado ou não, visualizar uma lista de todos os imóveis de um certo tipo e até um determinado preço. Os resultados desta opção serão:

```
Rua: Rua de 9 de Julho
Estado: Vende-se
ID: 11
Preço Pedido: 400
Preço Mínimo: 390


Rua: Rua de S.Miguel
Estado: Vende-se
ID: 12
Preço Pedido: 398
Preço Mínimo: 360
```

Resultados da opção 3

Consultar a lista de todos os imóveis habitáveis até um certo preço

```
Opção: 4

Insira o preço:
4000
```

Menu da opção 4

Esta opção permite a qualquer utilizador escrever um número no teclado e receber uma lista de todos os imóveis habitáveis. Eis o resultado desta opção:

```
Rua: Avenida da França
Estado: Vende-se
ID: 13
Preço Pedido: 300
Preço Mínimo: 100


Rua: Rua dos Clérigos
Estado: Vende-se
ID: 16
Preço Pedido: 590
Preço Mínimo: 500


Rua: Rua das Carmelitas
Estado: Vende-se
ID: 17
Preço Pedido: 1500
Preço Mínimo: 1250
```

Resultados da opção 4

Obter um mapeamento entre todos os imóveis e os seus vendedores

Esta opção permite a qualquer utilizador obter um mapeamento com todos os imóveis e os respetivos vendedores. Os resultados desta opção serão:

```
-----

Imóvel: Rua: Rua do Rosário
Estado: Vende-se
ID: 9
Preço Pedido: 300
Preço Mínimo: 290

Vendedor: E-mail: helena@gmail.com
Nome: Helena Dias
Password: helena
Morada: Rua de Sant'Ana
Data de Nascimento: 1990/07/01


-----
```

Resultado da opção 5

Colocar um imóvel à venda

Esta opção permite ao vendedor colocar um imóvel à venda. Este terá de escrever os parâmetros que classificam o imóvel no teclado e, se o imóvel não existir, colocá-lo nos sítios certo. Qualquer utilizador que não seja um vendedor que tente aceder a este método terá uma mensagem de erro imprimida. Eis um exemplo do registo de um imóvel à venda:

```
Opção: 6
Insira a rua:
Rua Genérica
Insira o preço pedido:
400
Insira o preço mínimo:
350
Insira o tipo do imóvel:

1 - Moradia
2 - Apartamento
3 - Terreno
4 - Loja
5 - Loja Habitável

Insira o tipo de negócio viável na loja:

 *** Menu ***
1 - Alimentação
2 - Vestuário
3 - Bem-estar e Saúde
4 - Lazer
5 - Outro
0 - Sair
Opção: 1
Insira a área:
1200
Indique se a loja possui WC ou não:

 *** Menu ***
1 - Tem WC
2 - Não tem WC
0 - Sair
Opção: 2
Insira o número da porta:
13

O imóvel foi registado com sucesso!
```

Resultados da opção 6

Visualizar a lista das últimas dez consultas aos imóveis para venda

Este método permite ao vendedor aceder às suas últimas dez consultas dos imóveis que tem para venda. Será imprimida uma mensagem de erro se um utilizador que não seja vendedor tente aceder a este método. (Nota: Se um e-mail surge como "notvalid", então foi um utilizador anónimo a acedê-lo). Eis um exemplo da opção:

```
Email : notvalid
Data : 2016/05/20 12:59:32
Consultas: 1
Imovel : Rua: Avenida da França
Estado: Vende-se
ID: 13
Preço Pedido: 300
Preço Mínimo: 100

Email : jose@gmail.com
Data : 2016/05/20 11:03:12
Consultas: 1
Imovel : Rua: Rua dos Caldeireiros
Estado: Vende-se
ID: 14
Preço Pedido: 600
Preço Mínimo: 500
```

Resultados da opção 7

Alterar o estado do imóvel

Esta opção permite ao vendedor inserir um estado no teclado e o identificador do imóvel e tentar alterar o estado desse imóvel. Eis um resultado:

```
Opção: 8

Insira o ID do imóvel:
15

Insira o novo estado do imóvel:

 *** Menu ***
1 - Vende-se
2 - Vendido
3 - Outro
0 - Sair
Opção: 2
estado: Vendido

O estado do imóvel foi alterado com sucesso!
```

Resultado da opção 8

Marcar um imóvel como favorito

Esta opção permite ao comprador registar um imóvel como favorito através da leitura do identificador deste do teclado. Eis um resultado desta opção:

```
Opção: 9

Insira o ID do imóvel:
3

 O imóvel foi marcado como favorito com sucesso!
```

Resultados da opção 9

Consultar a lista dos imóveis preferidos ordenados pelo seu preço

Esta opção permite ao comprador visualizar os seus imóveis favoritos. Eis uns resultados para esta opção.

Resultados da opção 10

```
Opção: 10

Lista dos imóveis favoritos ordenados pelo preço:

Rua: Rua do Almada
Estado: Vende-se
ID: 2
Preço Pedido: 300

Rua: Rua das Flores
Estado: Vende-se
ID: 3
Preço Pedido: 340

Rua: Rua de Belomonte
Estado: Vende-se
ID: 0
Preço Pedido: 345

Rua: Rua do Breiner
Estado: Vende-se
ID: 1
Preço Pedido: 450

Rua: Rua de Passos Manuel
Estado: Vende-se
ID: 4
Preço Pedido: 600
```

Obter o Top de imóveis

Esta opção obriga o vendedor a ler um número do teclado e escreve no ecrã todos os imóveis consultados mais vezes que esse número. Eis um exemplo.

```
Opção: 11

Insira o limite correspondente ao número de imóveis que deseja obter:
0

Top 0 de imóveis mais consultados

ID: 13

ID: 14

ID: 15
```

Resultados da opção 11

Encerrar sessão

Deve ser efetuada cada vez que se sai da aplicação. Escreve os dados todos num ficheiro para serem lidos noutra altura. Eis um exemplo da opção.

```
Opção: 12

A encerrar sessão...
```

Resultado da opção 12

Discussão: Como seria possível incluir novos tipos de imóveis na aplicação?

Neste momento, a nossa aplicação inclui os seguintes tipos de imóveis: Moradia, Apartamento, Loja, Terreno e Loja Habitável. Se, porventura, nos fosse proposto acrescentar novos tipos de imóveis, como, por exemplo, o tipo Quinta, teríamos, primeiramente, de criar uma nova classe com o nome do tipo que estendesse a superclasse Imóvel, pois seria, certamente, uma especificação desta. Implementaríamos na classe os métodos usuais, get's, set's, clone, toString e equals, bem como os construtores de classe.

De seguida, se a quinta permitisse que fosse habitada, teríamos de implementar a interface Habitavel, que marcaria o tipo Quinta como habitável.

Como os nossos compradores e vendedores têm como tipo de Set a superclasse Imóvel, não teríamos de fazer mudanças nessas classes.

Por outro lado, teríamos certamente de mudar o método que regista um imóvel, na classe ImoobiliariaApp, acrescentando o campo Quinta, caso o vendedor quisesse pôr uma quinta à venda.

Deste modo, por exemplo, o método getHabitaveis() definido na classe Imoobiliaria incluiria desde já o tipo Quinta. Desta maneira, poderíamos, então, acrescentar novos tipos de imóveis à aplicação desenvolvida.

Leilões

A simulação de leilões proposta no enunciado como valorização trouxe bastantes problemas ao grupo. Apesar do código estar feito, este não funciona. O grupo teve problemas em vários aspetos, sendo a cronometragem o principal. Apesar de não ter sido feita, esta parte foi bastante ponderada pela grupo e, durante muito tempo, trabalhada, logo o grupo achou que se devia mencionar a tática utilizada, complementando esta explicação com pedaços do código original, visto este não fazer parte da aplicação final.

Além das classes utilizadas anteriormente, a parte dos leilões acrescentou duas novas: a classe Leilao e a classe Licitacao. A classe Leilao continha um imóvel, o preço mínimo desse imóvel, a maior oferta por esse imóvel, o tempo em que devia acabar, a lista de compradores que queriam licitar, e um mapeamento entre a identificação do comprador e a sua licitação. A classe Licitacao implementava uma licitação, contendo intrinsecamente um número representante do valor da licitação.

Relativamente ao método iniciaLeilao, a estratégia usada passou pelos seguintes passos:

- Percorrer todos os compradores, verificar quais queriam leiloar, e adicioná-los ao leilão através do método adicionaComprador. Este último método adiciona simplesmente um comprador à lista da classe Leilao.

- Obter a data atual e calcular a data em que o leilão vai terminar. Como sabemos o tempo de duração do leilão (é inserido pelo vendedor), podemos calcular a hora em que ele termina.

- Inicia-se o leilão com todos estes parâmetros.

O método encerraLeilão limitava-se a devolver o comprador com a maior oferta e a eliminar os parâmetros do leilão. Se o comprador retornado fosse nulo, isto significava que nenhuma licitação superou o preço mínimo pedido.

Apesar de haver outros métodos, tais como getMaiorOferta que dava o valor da maior oferta no fim do leilão e setEstado para mudar o estado do imóvel se este fosse vendido, há um método que se destaca pela importância. O método desenvolvimentoLeilao. Este método simulava o leilão, contando o tempo passado, alterando o valor das licitações e licitando quando a oportunidade fosse a correta. Eis um excerto desse código juntamente com um método auxiliar igualmente importante.

```
public void desenvolvimentoLeilao()
{
    double menor = getMinIncrementoI();
    int j;
    double licita;
    for(j = 0; j < listaComprador.size(); j++)
    {
        Licitacao li = new Licitacao(listaComprador.get(j).getValor());
        mapLicitacao.put(listaComprador.get(j).getEmail(), li);
    }

    while(System.currentTimeMillis() < minutos)

    {
        System.out.println("Entrei aqui");
        try
        {
            Thread.sleep(1000 *(long) menor);//1000ms corresponde a 1s * menor intervalo
        }
         catch(InterruptedException e) {
            Thread.currentThread().interrupt();
        }
        for(j = 0; j < listaComprador.size(); j++)
        {
            licita = atualizaLicitacao(listaComprador.get(j),menor);
            if(licita > oferta)
                oferta = licita;
        }

    }
}
```

Função desenvolvimentoLeilao

```
public double atualizaLicitacao(Comprador co, double min)
{
    double licitacao = 0;
    for(Map.Entry<String,Licitacao> e : mapLicitacao.entrySet())
    {
        if(e.getKey().equals(co.getEmail()) == true)
        {
            co.adicionaIndAtual(min);
            if(co.getIndAtual() >= min)
            {

                licitacao = e.getValue().adicionaLicitacao(co.getValor());
                if( licitacao > co.getLimite())
                    licitacao = e.getValue().setLicitacao(co.getLimite());
            }
        }
    }
    return licitacao;
}
```

O método desenvolvimentoLeilao executava um ciclo que, durante certo intervalo de tempo, ia fazendo as atualizações da licitação de determinado comprador. No fim de cada iteração, verificava-se se a licitação era superior à maior oferta atual. Se sim, essa licitação passaria a ser a maior oferta atual.

O método atualizaLicitacao percorria o mapeamento dos compradores e da sua licitação e procurava o comprador atual. Quando encontrado adicionava o tempo ao tempo que já tinha passado desde a última licitação (nova variável de instância do Comprador) e se fosse maior ou igual, alterava-se a licitação. Se a nova licitação for superior ao preço máximo que o comprador pode dar, esta licitação torna-se no valor máximo.

Resumindo, o grupo, apesar de convicto no seu trabalho relativamente aos leilões, não conseguiu fazer este aspeto de valorização para desilusão do mesmo.

Conclusão

Terminando o desenvolvimento deste projeto de Java de Programação Orientada aos Objetos, podemos concluir que, embora o grupo não tenha conseguido concretizar a tarefa dos Leilões com sucesso (a nossa tentativa ficou registada no relatório para que o nosso esforço não tenha sido em vão e para provar que, efetivamente, tentamos fazê-la), o grupo conseguiu finalizar, com sucesso, todas as outras tarefas propostas aos alunos mencionadas no enunciado do projeto disponibilizado aos alunos desta Unidade Curricular.

O grupo encontrou algumas dificuldades em resolver alguns problemas, devido, em parte, ao conhecimento escasso que tínhamos em relação aos erros da linguagem de programação Java, contudo, conseguiu perceber a sua origem e razão, tendo, portanto, ultrapassado tais dificuldades. Possuímos, então, um conhecimento mais abrangente relativamente a estas situações. Destaca-se o erro "java.util.ConcurrentModificationException: null", que foi resolvido usando um *iterator* para percorrer uma associação *Map*.

Foram utilizadas diversas estruturas que foram lecionadas tanto nas aulas práticas como nas teóricas, nomeadamente os *Sets*, os *Maps*, as *Collections*, os *Iterators* e as *Lists*, bem como os Comparadores *(compareTo*), as superclasses e as datas (biblioteca *Date*). Ainda pusemos em prática as interfaces, as classes abstratas e as exceções.

Assim, podemos dizer, com toda a certeza, que o desenvolvimento deste projeto consolidou o nosso conhecimento de Java, pois trabalhámos com várias estruturas diferentes, aprendemos a implementá-las e, por isso, sentimo-nos muito mais à vontade para trabalhar com elas no futuro.