Trabalho Prático II - Programação Orientada a Objetos

👦 **Alunos** - Fabio Henrique Alves Fernandes | 19.1.4128
Igor Santiago Almeida Paixão | 19.1.4033
Pedro Luís Ribeiro de Souza Marra | 19.2.4076

🎓 **Professor** - Guillermo Camara Chavez

🏫 **Instituição** - Universidade Federal de Ouro Preto | Departamento de Computação

📅 **Data De Entrega** - 3 de dezembro de 2021

Vídeo de Apresentação

O vídeo de apresentação foi hospedado no Google Drive. Pelo tamanho do vídeo, não será possível enviá-lo junto do trabalho.

https://drive.google.com/file/d/1hi__6uomC8uWMo5Dj_7IONam39mgfnQt/view

Criação das Classes

Para a implementação do nosso trabalho, criamos a classe principal Imóvel, e suas classes polimórficas Apartamento, Casa e Chácara

Imóvel

A classe imóvel tem como atributos:

- Proprietário;
- Valor;
- Rua;
- Bairro;
- Cidade;
- Número;
- Quantidade de banheiros;
- Quantidade de quartos.

A classe, além de seu construtor, de seu destrutor e de seus getters e setters, conta também com a sobrecarga do método `toString()` para a saída de seus atributos. Também temos a sobrecarga do operador compareTo para ajudar durante a ordenação.

Apartamento

A classe Apartamento tem como atributos, além de todos os pertencentes a Imóvel:

- Andar em que está localizado;
- Taxa de condomínio;
- A existência ou não de elevador;

- A existência ou não de sacada.

Além do construtor, do destrutor e dos getters e setters, temos a sobrecarga que completa a saída feita na classe base Imóvel.

Casa

A classe Casa tem como atributos, além de todos os pertencentes a Imóvel:

- Quantidade de andares;
- A existência ou não de sala de jantar.

Além do construtor, do destrutor e dos getters e setters, temos a sobrecarga que completa a saída feita na classe base Imóvel.

Chácara

A classe Chácara tem como atributos, além de todos os pertencentes a Imóvel:

- A existência ou não de salão de festas;
- A existência ou não de salão de jogos;
- A existência ou não de campo de futebol;
- A existência ou não de churrasqueira;
- A existência ou não de piscina.

Além do construtor, do destrutor e dos getters e setters, temos a sobrecarga que completa a saída feita na classe base Imóvel.

Funções Uteis

Esse trabalho contém uma classe específica com todas as funções úteis para o seu funcionamento.

ArrayList<Imovel> leArquivo()

Essa função lê a database de imóveis, pegando cada linha do arquivo texto, fazendo um split da String formada em um Array de String contendo todos os itens necessários. A partir disso, criamos objetos das subclasses Apartamento, Casa ou Chacara e os guardamos em um ArrayList, retornado no final do arquivo. Caso ocorra algum erro de abertura do arquivo, existe um tratamento de excessões usando try.

boolean proprietario_tem_imovel(ArrayList <Imovel> imoveis, String proprietario)

Função que, a partir de um ArrayList já inicializado e completo com todos os imóveis, procura pelo proprietário a partir de seu nome que foi passado por valor na função, percorrendo o ArrayList, usando o metodo equals da classe String, para saber se o proprietário é realmente dono de algum imóvel. Se a função encontra um imóvel que seja desse proprietário, retorna `true`, senão, retorna `false`.

ArrayList <Imovel> conjunto_valor(ArrayList <Imovel> imoveis, double valor_max)

Função que, a partir de um ArrayList já inicializado e completo com todos os imóveis, procura por imóveis que tem um valor igual ou menor ao passado para a função. Um novo ArrayList é inicializado dentro dessa função, para que os imóveis encontrados sejam guardados nele. O ArrayList original é percorrido, e um if valida os imóveis que tem um valor menor ou igual ao definido. Caso um imóvel seja selecionado, ele é passado para o ArrayList criado no inicio. No final, estando com itens ou não, o ArrayList é retornado.

ArrayList <Imovel> conjunto_quartos (ArrayList <Imovel> imoveis, int numMaxQuartos)

Função que, a partir de um ArrayList já inicializado e completo com todos os imóveis, procura por imóveis que tem um número de quartos igual ou menor ao passado para a função. Um novo ArrayList é inicializado dentro dessa função, para que os imóveis encontrados sejam guardados nele. O ArrayList original é percorrido, e um if valida os imóveis que tem o número de quartos menor ou igual ao definido. Caso um imóvel seja selecionado, ele é passado para o ArrayList criado no inicio. No final, estando com itens ou não, o ArrayList é retornado.

ArrayList <Imovel> conjunto_cidade (ArrayList <Imovel> imoveis, String cidade)

Função que, a partir de um ArrayList já inicializado e completo com todos os imóveis, procura por imóveis que são de uma cidade igual à passada para a função. Um novo ArrayList é inicializado dentro dessa função, para que os imóveis encontrados sejam guardados nele. O ArrayList original é percorrido, e um if valida os imóveis que são da cidade definida. Caso um imóvel seja selecionado, ele é passado para o ArrayList criado no inicio. No final, estando com itens ou não, o ArrayList é retornado.

ArrayList <Imovel> conjunto_apartamento (ArrayList <Imovel> imoveis)

Função que, a partir de um ArrayList já inicializado e completo com todos os imóveis, procura por imóveis que sejam Apartamentos. Um novo ArrayList é inicializado dentro dessa função, para que os imóveis encontrados sejam guardados nele. O ArrayList original é percorrido, e um if, usando um `instanceof` (método que define se determinado objeto é uma instancia de alguma classe). Caso um imóvel seja selecionado, ele é passado para o ArrayList criado no inicio. No final, estando com itens ou não, o ArrayList é retornado.

ArrayList <Imovel> conjunto_casa (ArrayList <Imovel> imoveis)

Função que, a partir de um ArrayList já inicializado e completo com todos os imóveis, procura por imóveis que sejam Casas. Um novo ArrayList é inicializado dentro dessa função, para que os imóveis encontrados sejam guardados nele. O ArrayList original é percorrido, e um if, usando um `instanceof` (método que define se determinado objeto é uma instancia de alguma classe). Caso um imóvel seja selecionado, ele é passado para o ArrayList criado no inicio. No final, estando com itens ou não, o ArrayList é retornado.

ArrayList <Imovel> conjunto_chacara (ArrayList <Imovel> imoveis)

Função que, a partir de um ArrayList já inicializado e completo com todos os imóveis, procura por imóveis que sejam Chacaras. Um novo ArrayList é inicializado dentro dessa função, para que os imóveis encontrados sejam guardados nele. O ArrayList original é percorrido, e um if, usando um `instanceof` (método que define se determinado objeto é uma instancia de alguma classe). Caso um imóvel seja selecionado, ele é passado para o ArrayList criado no inicio. No final, estando com itens ou não, o ArrayList é retornado.

ArrayList <Imovel> conjunto_numero_imovel (ArrayList <Imovel> imoveis, int numero)

Função que, a partir de um ArrayList já inicializado e completo com todos os imóveis, procura por imóveis que tem um número de endereço do imóvel igual ao passado para a função. Um novo ArrayList é inicializado dentro dessa função, para que os imóveis encontrados sejam guardados nele. O ArrayList original é percorrido, e um if valida os imóveis que tem o número de endereço de imóvel igual ao definido. Caso um imóvel seja selecionado, ele é passado para o ArrayList criado no inicio. No final, estando com itens ou não, o ArrayList é retornado.

ArrayList <Integer> conjunto_iteradores(ArrayList <Imovel> imoveis, String proprietario)

Função que, a partir de um ArrayList já inicializado e completo com todos os imóveis, procura pelo proprietário a partir de seu nome que foi passado por valor na função. Um ArrayList de inteiros é incicializado, onde ficarão os iteradores e também iniciamos um contador, que começa de 0, e vai sendo atualizado a cada iteração do for que percorre o ArrayList de imóveis. Usamos inteiros como iteradores, pois é um método mais fácil de se trabalhar com isso. Percorremos o ArrayList e, usando um if com o metodo equals da classe String, sabemos se o proprietário é realmente dono de algum imóvel. Se for, passamos o conteúdo atual do contador para o ArrayList de retorno, que mostra em que posição do ArrayList de imóveis está o imóvel. Ao final, esse ArrayList de retorno é retornado.

void salva_aquivo_texto(ArrayList <Imovel> imoveis)

Essa função percorre o ArrayList passado, imprimindo atributo por atributo de cada imóvel, a partir da formatação pedida. Para isso, usamos os getters de cada atributo, e, quando tivermos que imprimir os atributos das subclasses, usaremos downcast, para que consigamos utilizar dos métodos presentes nelas.

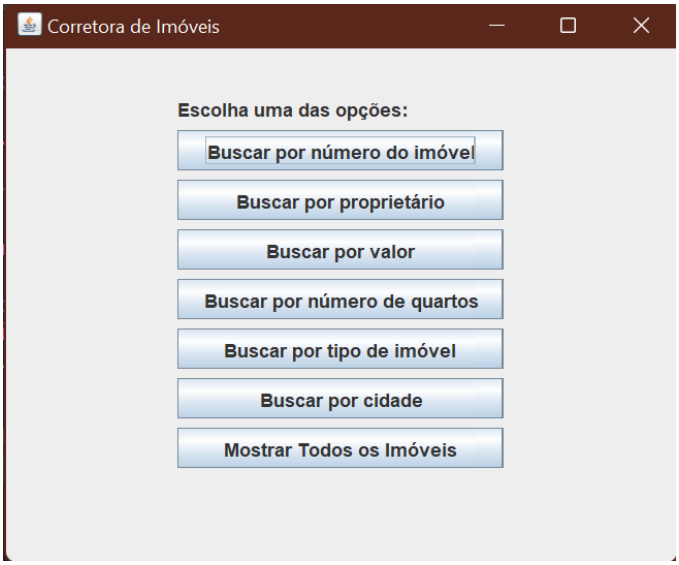
Telas

As telas são interfaces implementadas com o intuito de mostrar os dados de cada questão exigida no trabalho prático.

TelaInicial

A interface da tela inicial é responsável pela seleção das opções desejadas no trabalho prático, ela é responsável por chamar as outras telas que serão responsáveis por usar as funções definidas na Classe `FuncoesUteis` . O construtor da Classe `TelaInicial` já inicia com um `ArrayList` de Imóveis que foi passado pelo main. Temos 7 botões que realiza a operação desejada com o `ArrayList`, quando acionamos algum botão ele realiza duas operações, `this.setEnable(false)` que vai fazer com que a `TelaInicial` fica indisponível para acesso, mas continuará visível e logo em seguida inicializamos uma nova tela, das quais iremos tratar logo em seguida, de acordo com a opção que acionamos, fazendo com que ela também fique visível, exemplo:

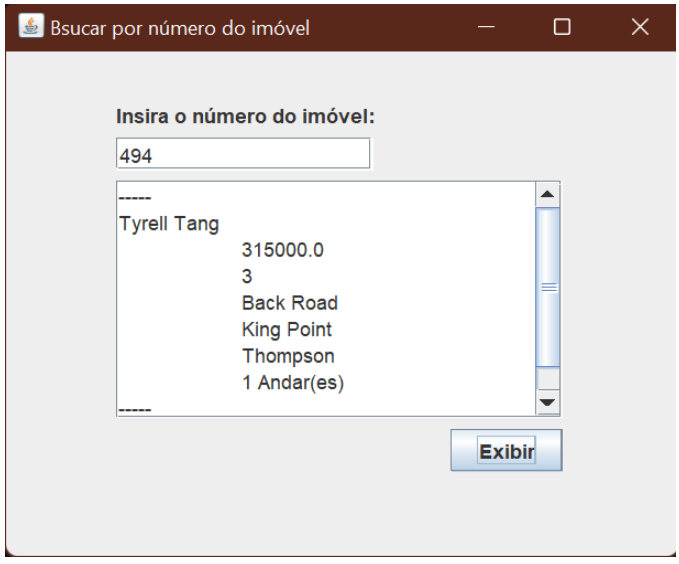
```
new opcaoAcionada(this).setVisible(True);
```



Tela TelaInicial

NumeroImovel

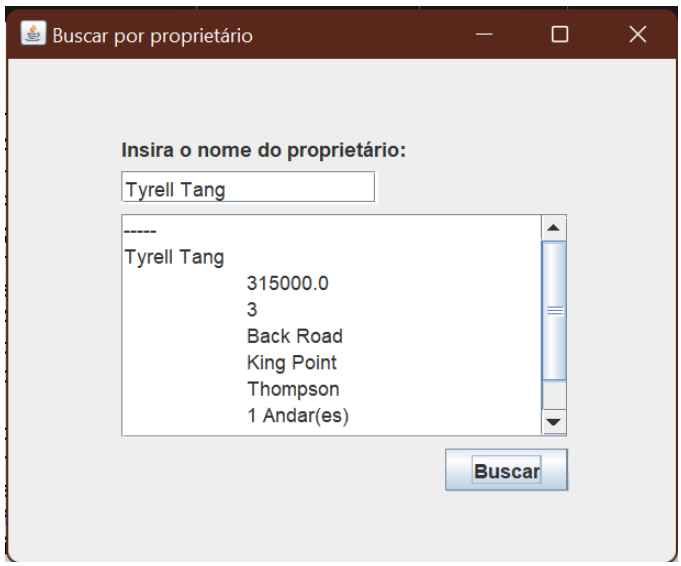
A tela `NumeroImovel` ela em essencial utiliza a função `conjunto_numero_imovel`, presente na Classe `FuncoesUteis` , uma função que dado um número, pesquisa na base de dados, um imóvel com numero igual ao desejado. Na tela passamos como uma string que representará o numero, pegamos ela por um `jTextField` , transformamos a string em um inteiro e chamamos a função que pesquisa o item desejado, logo em seguida, a função retornara um `ArrayList` com todas as ocorrências de um imóvel com numero desejado. E assim, usamos a função `toString()` para mostrar na tela através de um `jTextArea`, a lista de imóveis presentes no `ArrayList` retornado como resultado da busca. A seguir temos o exemplo de uma busca por algum imóvel com número **494**:



Tela NumeroImovel

Proprietario

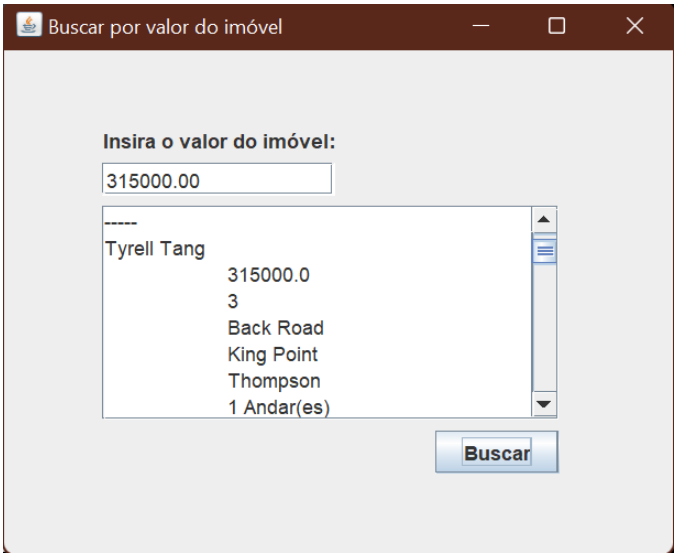
A tela `Proprietario`, ela utiliza a função `proprietário_tem_imovel` e `conjunto_iteradores`, como o intuito de verificar a existência de algum imóvel em propriedade do nome pesquisado. Caso não encontre nenhuma propriedade no nome pesquisado, retornamos uma caixa de dialogo, alertando que não existe imóvel do proprietário em questão. Caso contrário, a função `proprietário_tem_imovel` retornará verdadeiro, garantindo a existência de imóveis no nome desejado, e a função `conjunto_iteradores`, irá retornar um `ArrayList` de imóveis com todos os imóveis pertencentes a tal proprietário, passado como parâmetro em um `jTextField`, mostrando em um `jTextArea` o `ArrayList` utilizando a função `toString()`, a seguir temos um exemplo de busca por imóvel do proprietário **Tyrel Tang**:



Tela Proprietario

Valor

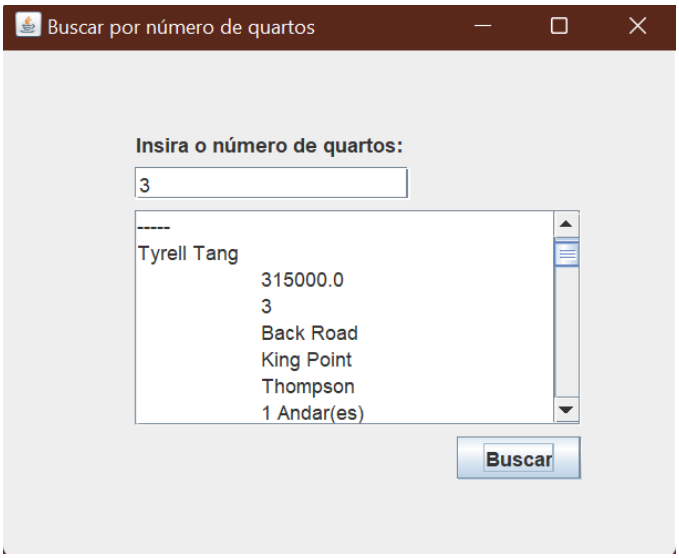
A tela `Valor`, utiliza a função `conjunto_valor` que retorna todos os imóveis, em um `ArrayList`, que possuem valor igual ou abaixo do especificado na entrada. Utilizamos um `jTextField` para captar a entrada, transformamos ele em double e passamos como parâmetro na função `conjunto_valor`, retornando assim a lista com os imóveis dentro do especificado acima, e mostramos dentro de um `jTextArea` todos os imóveis através da função `toString()`, a seguir temos um exemplo de busca:



Tela Valor

NumeroQuartos

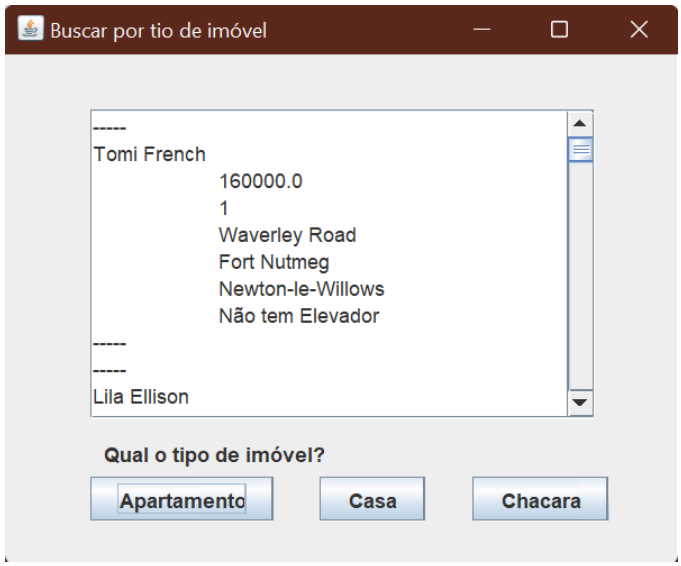
A tela `NumeroQuartos`, utiliza a função `conjunto_quartos` que retorna todos os imóveis, em um `ArrayList` , que possuem o número de quartos igual ou abaixo do especificado na entrada. Utilizamos um `jTextField` para captar a entrada, transformamos ele em um `int` e passamos como parâmetro na função `conjunto_quartos` , retornando assim a lista com os imóveis dentro do especificado acima, e mostramos dentro de um `jTextArea` todos os imóveis através da função `toString()` , a seguir temos um exemplo de busca:



Tela NumeroQuartos

TipoImovel

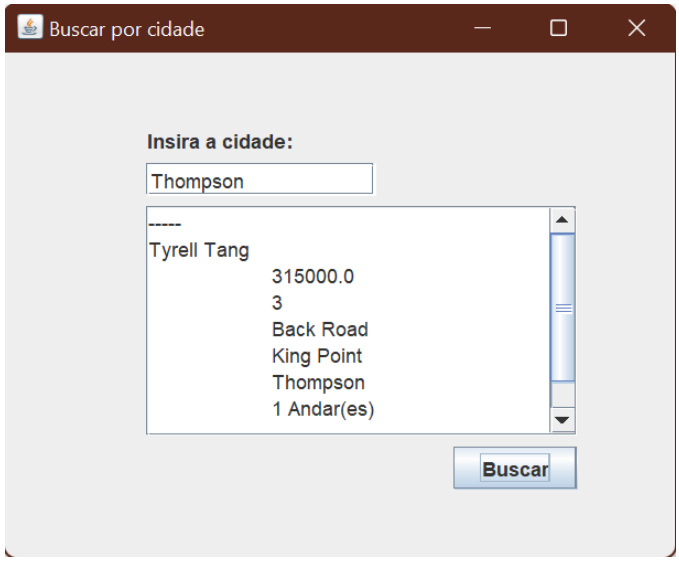
A tela `TipoImovel`, utiliza `conjunto_apartamentos`, `conjunto_casa`, `conjunto_chacara` que retornam um `ArrayList` com todos os imóveis dos determinados tipos. Para escolha do tipo usaremos 3 botões com acionamento, ao acionarmos 1 opção de tipo de imóvel utilizamos 1 das funções citadas acima, equivalente ao tipo desejado, organizamos o `ArrayList` em ordem crescente pelo valor do imóvel, e mostramos na tela através de um `jTextArea` utilizando a função `toString()` , a seguir um exemplo de acionamento do tipo **Apartamento**:



Tela TipoImovel

Cidade

A tela `Cidade`, utiliza a função `conjunto_cidade` que retorna todos os imóveis, em um `ArrayList` , que são da cidade especificada na entrada. Utilizamos um `jTextField` para captar a entrada, e passamos como parâmetro na função `conjunto_cidade` , retornando assim a lista com os imóveis dentro do especificado acima, logo após ordenamos o `ArrayList` em ordem crescente com a função sort, e logo em seguida, invertemos para o `ArrayList` para ficar em ordem decrescente, e mostramos dentro de um `jTextArea` todos os imóveis através da função `toString()` , a seguir temos um exemplo de busca pela cidade `Thompson`:



Tela Cidade

TodosImoveis

A tela `TodosImoveis`, utiliza o `ArrayList` e vai percorrendo e escrevendo todos os elementos no `jTextArea`, para cada tipo diferente faremos um `DownCast`, para verificar a existência de atributos dos determinados tipos e printar corretamente, fazemos isso com cada imóvel , usando seus determinados getters. Temos dois botões para acionamento, o botão para `Salvar em Arquivo` e `Mostrar na Tela`, na primeira opção salvamos em um arquivo de saida.txt, já na segunda opção mostramos no `jTextArea` como citado anteriormente.

Tela TodosImoveis

UML

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/82e6bad5-63b5-40ed-ae4d-b9b5825cf1ff/UML.pdf


FuncoesUteis
+ leArquivo(): ArrayList<Imovel>
+ proprietario_tem_imovel(ArrayList <Imovel> , String): boolean
+ conjunto_valor(ArrayList <Imovel>, double): ArrayList <Imovel>
+ conjunto_quartos (ArrayList <Imovel>, int): ArrayList <Imovel>
+ conjunto_cidade (ArrayList <Imovel>, String): ArrayList <Imovel>
+ conjunto_apartamento (ArrayList <Imovel>): ArrayList <Imovel>
+ conjunto_casa (ArrayList <Imovel>): ArrayList <Imovel>
+ conjunto_chacara (ArrayList <Imovel>): ArrayList <Imovel>
+ conjunto_numero_imovel (ArrayList <Imovel>, int): ArrayList <Imovel>
+ conjunto_iteradores(ArrayList <Imovel>, String): ArrayList <Integer>
+ salva_aquivo_texto(ArrayList <Imovel>): void
Apartamento
- andar: int
- taxa_condominio: float
- elevador: bool
- sacada: bool
+ getAndar(): return int
+ setAndar(int): void
+ getTaxaCondominio(): return float
+ setTaxaCondominio(float): void
+ getElevador(): return bool
+ setElevadorbool): void
+ getSacada(): return bool
+ setSacada(bool): void
+ toString(): String
Casa
- andares: int
- sala_jantar: bool
+ getAndares(): return int
+ setAndares(int): void
+ getSalaJantar(): return bool
+ setSalaJantar(bool): void
+ toString(): String
Chácara
- salao_festa: bool
- salao_jogos: bool
- campo_futebol: bool
- churrasqueira: bool
- piscina: bool
+ getSalaoFesta(): return bool
+ setSalaoFesta(bool): void
+ getSalaoJogos(): return bool
+ setSalaoJogos(bool): void
+ getcCampoFutebol(): return bool
+ setCampoFutebol(bool): void
+ getChurrasqueira(): return bool
+ setChurrasqueira(bool): void
+ getPiscina(): return bool
+ setPiscina(bool): void
+ getSacada(): return bool
+ setSacada(bool): void
+ toString(): String
TelaInicial
- imoveis: ArrayList<Imovel>
- javax.swing.JButton jButton1;
- javax.swing.JButton jButton2;
- javax.swing.JButton jButton3;
- javax.swing.JButton jButton4;
- javax.swing.JButton jButton5;
- javax.swing.JButton jButton6;
- javax.swing.JButton jButton7;
- javax.swing.JLabel jLabel1;
- javax.swing.JPanel jPanel1;
+ getImoveis(): ArrayList<Imovel>
+ jButton1ActionPerformed(java.awt.event.ActionEvent): void
+ jButton2ActionPerformed(java.awt.event.ActionEvent): void
+ jButton3ActionPerformed(java.awt.event.ActionEvent): void
+ jButton4ActionPerformed(java.awt.event.ActionEvent): void
+ jButton5ActionPerformed(java.awt.event.ActionEvent): void
+ jButton6ActionPerformed(java.awt.event.ActionEvent): void
+ jButton7ActionPerformed(java.awt.event.ActionEvent): void
Imóvel
- valor: float
- proprietário: string
- rua: string
- bairro: string
- cidade: string
- número: int
- quartos: int
- banheiros: int
+ getValor(): return float
+ setValor(float): void
+ getProprietario(): return string
+ setProprietario(string): void
+ getRua(): return string
+ setRua(string): void
+ getBairro(): return string
+ setBairro(string): void
+ getCidade(): return string
+ setCidade(string): void
+ getNumero(): return int
+ setNumero(int): void
+ getQuartos(): return int
+ setQuartos(int): void
+ getBanheiros(): return int
+ setBanheiros(int): void
+ toString(): String
+ compareTo(Imovel): int
<<tela>>
NumeroImovel
- tela: TelaInicial
- javax.swing.JButton jButton1;
- javax.swing.JLabel jLabel1;
- javax.swing.JLabel jLabel2;
- javax.swing.JScrollPane jScrollPane1;
- javax.swing.JTextArea jTextArea1;
- javax.swing.JTextField jTextField1;
+ jButton1ActionPerformed(java.awt.event.ActionEvent): void
+ formWindowClosed(java.awt.event.WindowEvent): void
<<tela>>
Proprietario
- tela: TelaInicial
- javax.swing.JButton jButton1;
- javax.swing.JLabel jLabel1;
- javax.swing.JScrollPane jScrollPane1;
- javax.swing.JTextArea jTextArea1;
- javax.swing.JTextField jTextField1;
+ jButton1ActionPerformed(java.awt.event.ActionEvent): void
+ formWindowClosed(java.awt.event.WindowEvent): void
<<tela>>
Valor
- tela: TelaInicial
- javax.swing.JButton jButton1;
- javax.swing.JLabel jLabel1;
- javax.swing.JScrollPane jScrollPane1;
- javax.swing.JTextArea jTextArea1;
- javax.swing.JTextField jTextField1;
+ jButton1ActionPerformed(java.awt.event.ActionEvent): void
+ formWindowClosed(java.awt.event.WindowEvent): void
<<tela>>
NumeroQuartos
- tela: TelaInicial
- javax.swing.JButton jButton1;
- javax.swing.JLabel jLabel1;
- javax.swing.JScrollPane jScrollPane1;
- javax.swing.JTextArea jTextArea1;
- javax.swing.JTextField jTextField1;
+ jButton1ActionPerformed(java.awt.event.ActionEvent): void
+ formWindowClosed(java.awt.event.WindowEvent): void
<<tela>>
TipoImovel
- tela: TelaInicial
- javax.swing.JButton jButton2;
- javax.swing.JButton jButton3;
- javax.swing.JButton jButton4;
- javax.swing.JLabel jLabel1;
- javax.swing.JScrollPane jScrollPane1;
- javax.swing.JTextArea jTextArea1;
+ jButton2ActionPerformed(java.awt.event.ActionEvent): void
+ jButton3ActionPerformed(java.awt.event.ActionEvent): void
+ jButton4ActionPerformed(java.awt.event.ActionEvent): void
+ formWindowClosed(java.awt.event.WindowEvent): void
<<tela>>
Cidade
- tela: TelaInicial
- javax.swing.JButton jButton1;
- javax.swing.JLabel jLabel1;
- javax.swing.JScrollPane jScrollPane1;
- javax.swing.JTextArea jTextArea1;
- javax.swing.JTextField jTextField1;
+ jButton1ActionPerformed(java.awt.event.ActionEvent): void
+ formWindowClosed(java.awt.event.WindowEvent): void
<<tela>>
TodosImoveis
- tela: TelaInicial
- javax.swing.JButton jButton1;
- javax.swing.JButton jButton2;
- javax.swing.JScrollPane jScrollPane1;
- javax.swing.JTextArea jTextArea1;
+ jButton1ActionPerformed(java.awt.event.ActionEvent): void
+ jButton2ActionPerformed(java.awt.event.ActionEvent): void
+ formWindowClosed(java.awt.event.WindowEvent): void

Compilação e Execução

Para esse projeto em Java, por dificuldades na geração do arquivo .jar (executável), a melhor forma de compilação e execução é usando alguma IDE (projeto feito no NetBeans). Durante a criação do projeto, não conseguimos fazer com que a classe principal fosse explicitada automaticamente dentro do arquivo MANIFEST.mf, o que impossibilita a execução do .jar criado. Até poderíamos enviar um .jar criado já com um MANIFEST.mf atualizado, porém, a partir de outra compilação, esse arquivo seria desatualizado.

Uso da Java Collections Framework no trabalho

Para o nosso sistema, usamos ArrayList para todas as questões para que não precisemos de conversões entre as outras e pela facilidade em percorrê-lo. Além de a função de ordenação funcionar bem.

Uso geral da Java Collections Framework

Acessar uma posição específica de um contêiner

Para o acesso a uma posição específica podemos usar um Array, List ou ArrayList, já que podemos usar o operador `[]` ou outros métodos específicos que retornam o elemento esperado.

Adicionar um elemento e manter somente elementos únicos no contêiner

Um dos contêineres que permitem a inserção somente de elementos únicos é o Set, que tem essa característica desde sua concepção.

Inserção ou remoção no final

Um dos contêineres que melhor faz a inserção ou a remoção no final é o Array, já que o acesso a esse elemento é muito rápido (O(1))

Retornar um valor baseado em uma chave específica (não necessariamente inteiros)

Nesse caso, Multiset é uma opção bem viável.

Inserção e remoção no início

O Deque é uma ótima opção, não só para a remoção/inserção no inicio, mas também para no final.

Busca por um elemento

Multimap é uma opção bem interessante quando queremos buscar um elemento.

Contêiner com o comportamento de primeiro a entrar e o último a sair

O contêiner com essa característica é basicamente a Stack (pilha), já que ela vai empilhando itens, de forma que o primeiro item que entrou ficou no final da pilha, já o último fica no topo.

Contêiner com o comportamento de primeiro a entrar e o primeiro a sair

O contêiner com essa característica é a Queue (fila), onde sempre que o primeiro elemento entrar, ele vai ser sempre o primeiro a sair.