



FACULDADE ESTÁCIO DE SÁ
CURSO: DESENVOLVIMENTO FULL STACK
3º SEMESTRE – MATRÍCULA 202302595341

ALAIM ALMEIDA DE OLIVEIRA

Criação das Entidades e Sistema de Persistência

Salvador – BA

2024

1º Procedimento | Criação das Entidades e Sistema de Persistência

Objetivo da Prática

De acordo com o passo-a-passo instruído pelo curso, é mostrar a forma de criar um aplicativo de cadastro, um CRUD (Create, Read, Update, Delete) com o Java de uma forma simples e objetiva, fazendo com que os alunos coloquem em prática todo o assunto estudado no nível 1.

Com essas instruções e seguindo os passos, ao final estaremos aptos a elaborar um sistema de cadastro que cria, ler, altera e deleta o que foi cadastrado.

Herança

A herança é um conceito importante e fundamental na programação orientada a objetos, pois é amplamente utilizada na linguagem de programação Java.

É um mecanismo pelo qual uma classe(subclasse) pode herdar atributos e métodos de outra classe (classepai ou superclasse). A subclasse pode então, estender e modificar o comportamento da superclasse, permitindo a reutilização de código e a criação de hierarquias de classes.

Na utilização da herança no Java, uma das maiores vantagens seria no quesito de reutilização de código, pois a superclasse compartilha todo o seu comportamento, atributos para as outras classes, evitando a repetição de código, promovendo a reutilização e a modularidade do mesmo.

Com a herança, temos o polimorfismo, onde um objeto de uma subclasse pode ser tratado como um objeto da superclasse. Isso facilita a escrita de código genérico e flexível. Além do reaproveitamento de código, todas as classes herdadas, ficaram organizadas em uma hierarquia, representando relações de especialização e generalização entre objetos do mundo real. Isso facilita a compreensão e a manutenção do código.

Desvantagens da Herança em Java

Ao utilizar a herança, é criado um acoplamento forte entre a subclasse e superclasse, o que pode tornar o código mais difícil de entender e manter. Mudanças na superclasse podem afetar as subclasses e vice-versa.

Quando se trata de herança múltiplas, o Java não tem suporte.

Devido a hierarquias profundas ao ser utilizado a herança, o código pode se tornar complexo e difícil de gerenciar, pois pode levar a uma estrutura excessivamente complicada e difícil de entender.

A herança em Java é uma poderosa ferramenta de programação orientada a objetos que oferece vantagens significativas, como reutilização de código,

extensibilidade e polimorfismo. No entanto, é importante usar a herança com cuidado e considerar suas potenciais desvantagens, como acoplamento forte e herança frágil, ao projetar sistemas de software.

Interface Serializable

A interface Serializable em Java indica que uma classe pode ser convertida em uma sequência de bytes para armazenamento ou transmissão. Essa conversão é essencial para persistir objetos em arquivos binários, permitindo que o Java os armazene e reconstrua quando necessário.

Paradigma Funcional e API Stream

A API Stream em Java, desde o Java 8, utiliza conceitos do paradigma funcional para operar em coleções de dados de maneira mais eficiente e concisa. Ela oferece operações de alto nível, como map, filter e reduce, que simplificam a manipulação dos elementos da coleção de forma declarativa, sem a necessidade de iteração manual. Além disso, as operações em Stream são geralmente imutáveis, o que significa que não modificam o estado original da coleção, tornando o código mais seguro e concorrente. A API Stream também suporta a composição de funções, permitindo encadear várias operações em uma única expressão para criar pipelines de processamento de dados eficientes e legíveis.

Padrão de Desenvolvimento na Persistência de Dados em Arquivos em Java

Para persistir dados em arquivos em Java, é comum seguir o padrão de desenvolvimento chamado Object Serialization. Este padrão envolve a serialização de objetos em bytes para armazenamento em arquivos e inclui os seguintes passos:

Implementar a interface Serializable nas classes desejadas.

Usar classes como ObjectOutputStream e ObjectInputStream para serializar e desserializar objetos para e a partir de arquivos.

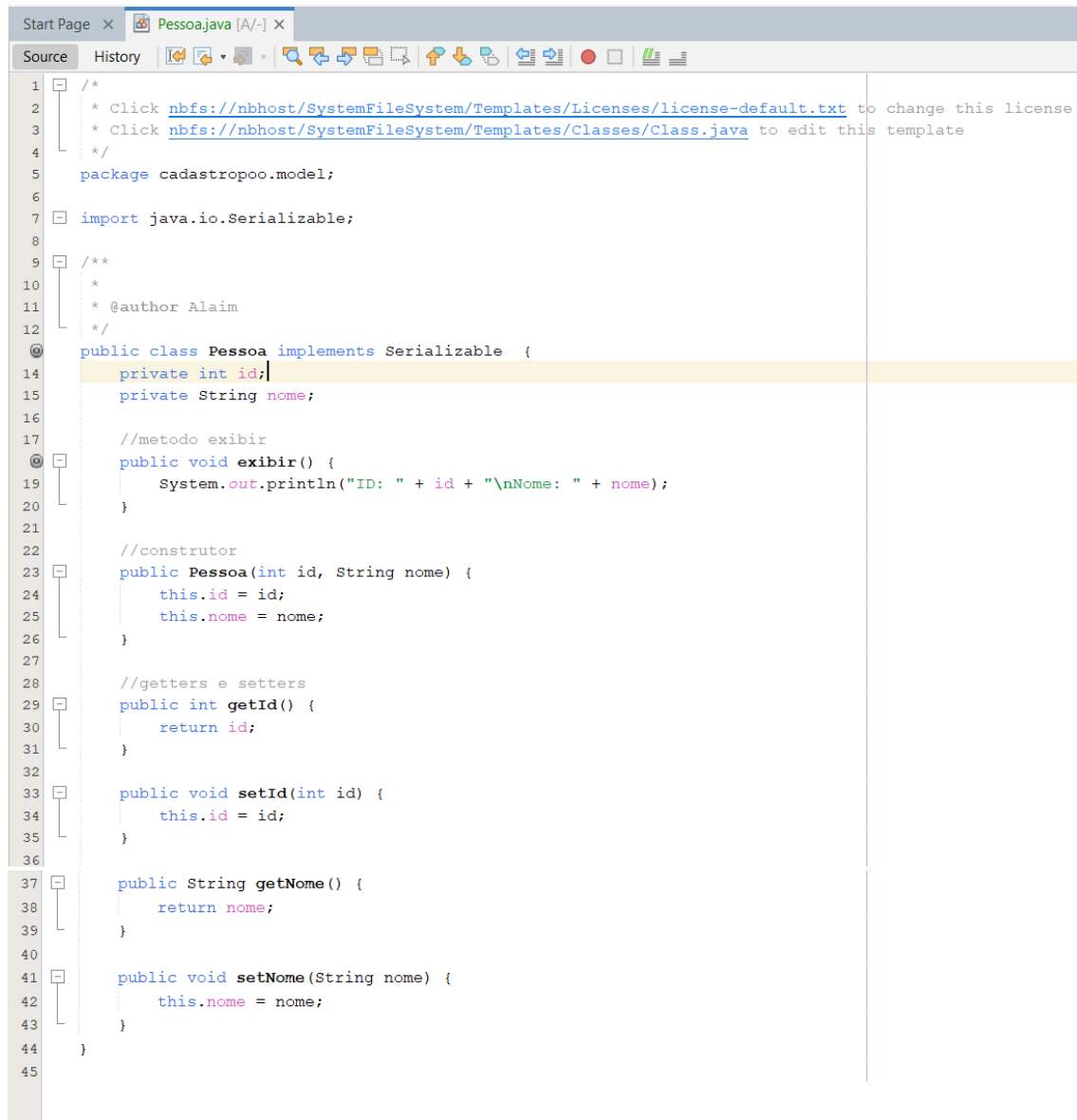
Lidar com exceções apropriadas ao trabalhar com operações de entrada e saída de dados.

Garantir que todas as classes usadas na serialização tenham uma versão única para evitar problemas de compatibilidade.

Esse padrão oferece uma maneira eficiente e flexível de persistir objetos em arquivos binários em Java.

Segue abaixo os códigos utilizados para criar o sistema de cadastro:

Logo de início, criamos um pacote chamado “model”, para armazenar as entidades e os gerenciadores. No pacote model, criamos as entidades Pessoa, PessoaFisica e PessoaJuridica como pode ver abaixo:



```
1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4   */
5   package cadastrapoo.model;
6
7   import java.io.Serializable;
8
9   /**
10    *
11    * @author Alaim
12    */
13   @
14   public class Pessoa implements Serializable {
15       private int id;
16       private String nome;
17
18       //metodo exibir
19       @
20       public void exibir() {
21           System.out.println("ID: " + id + "\nNome: " + nome);
22       }
23
24       //construtor
25       public Pessoa(int id, String nome) {
26           this.id = id;
27           this.nome = nome;
28       }
29
30       //getters e setters
31       public int getId() {
32           return id;
33       }
34
35       public void setId(int id) {
36           this.id = id;
37       }
38
39       public String getNome() {
40           return nome;
41       }
42
43       public void setNome(String nome) {
44           this.nome = nome;
45       }
46   }
```

Imagem 01 – classe Pessoa

```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5  package cadastrapoo.model;
6
7  /**
8   *
9   * @author Alaim
10  */
11  public class PessoaFisica extends Pessoa{
12      private String cpf;
13      private int idade;
14
15      //construtores
16      public PessoaFisica(int idade, String cpf, String nome, int id) {
17          super(id, nome);
18          this.cpf = cpf;
19          this.idade = idade;
20      }
21
22      //metodo exibir
23      @Override
24      public void exibir() {
25          super.exibir();
26          System.out.println("CPF: " + cpf + "\nIdade: " + idade);
27      }
28
29      //getters e setters
30      public String getCpf() {
31          return cpf;
32      }
33
34      public void setCpf(String cpf) {
35          this.cpf = cpf;
36      }
37
38      public int getIdade() {
39          return idade;
40      }
41
42      public void setIdade(int idade) {
43          this.idade = idade;
44      }
45  }
46
47  }
```

Imagem 02 – classe PessoaFisica

Depois de ter criado as duas classes Pessoa e PessoaFisica, foi criado a parte das classes gerenciadores com os nomes PessoaFisicaRepo e PessoaJuridicaRepo:



```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5  package cadastrapoo.model;
6
7  import java.io.FileInputStream;
8  import java.io.FileOutputStream;
9  import java.io.IOException;
10 import java.io.ObjectInputStream;
11 import java.io.ObjectOutputStream;
12 import java.util.ArrayList;
13 import java.util.List;
14
15 /**
16 *
17 * @author Alaim
18 */
19 public class PessoaFisicaRepo {
20     //arrayList
21     private List<PessoaFisica> pessoasFisicas = new ArrayList<>();
22
23     //método inserir
24     public void inserir(PessoaFisica pessoaFisica){
25         pessoasFisicas.add(pessoaFisica);
26     }
27
28     //método alterar
29     public void alterar(PessoaFisica pessoaFisica){
30
31     }
32
33     //método excluir
34     public void excluir(int id){
35         pessoasFisicas.removeIf(pessoa -> pessoa.getId() == id);
36     }
37
38     //método obter
39     public PessoaFisica obter(int id){
40         return pessoasFisicas.stream()
41             .filter(pessoa -> pessoa.getId() == id)
42             .findFirst()
43             .orElse(null);
44     }
45
46     //método obterTodos
47     public List<PessoaFisica> obterTodos(){
48         return new ArrayList<>(pessoasFisicas);
49     }
50
51     //método persistir
52     public void persistir(String nomeArquivo) throws IOException {
53         try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream(nomeArquivo))) {
54             outputStream.writeObject(pessoasFisicas);
55         }
56     }
57
58     //metodo recuperar
59     @SuppressWarnings("unchecked")
60     public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {
61         try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(nomeArquivo))) {
62             pessoasFisicas = (List<PessoaFisica>) inputStream.readObject();
63         }
64     }
65 }
66
```

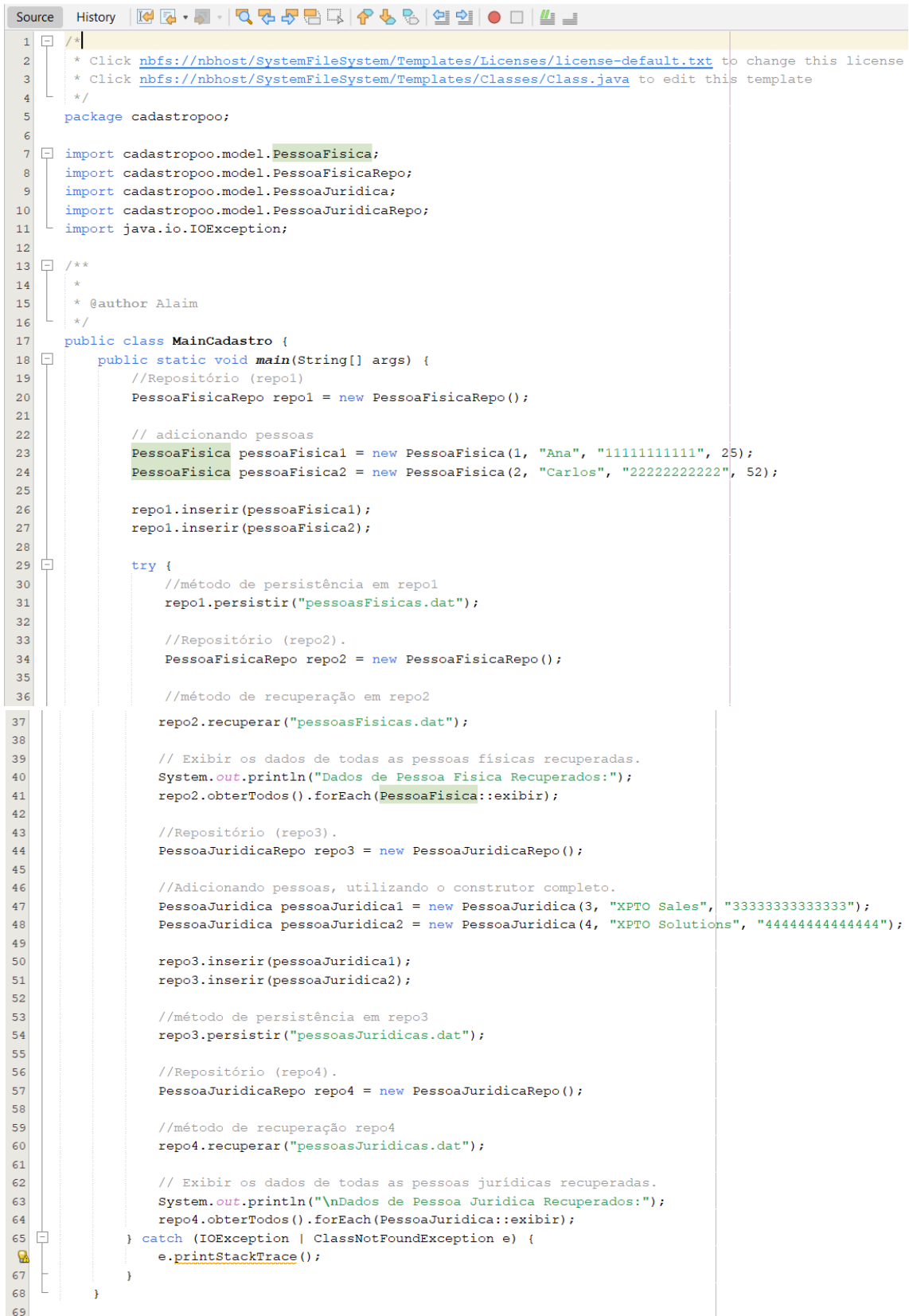
Imagem 03 – Classe PessoaFisicaRepo



```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5  package cadastropoo.model;
6
7  import cadastropoo.model.PessoaJuridica;
8  import java.io.FileInputStream;
9  import java.io.FileOutputStream;
10 import java.io.IOException;
11 import java.io.ObjectInputStream;
12 import java.io.ObjectOutputStream;
13 import java.util.ArrayList;
14 import java.util.List;
15
16 /**
17 *
18 * @author Alaim
19 */
20 public class PessoaJuridicaRepo {
21     public List<PessoaJuridica> pessoasJuridicas = new ArrayList();
22
23     //metodo inserir
24     public void inserir(PessoaJuridica pessoaJuridica){
25         pessoasJuridicas.add(pessoaJuridica);
26     }
27
28     //metodo alterar
29     public void alterar(PessoaJuridica pessoaJuridica){
30
31     }
32
33     //metodo excluir
34     public void excluir(int id){
35         pessoasJuridicas.removeIf(pessoa -> pessoa.getId() == id);
36     }
37
38     //metodo obter
39     public PessoaJuridica obter(int id){
40         return pessoasJuridicas.stream()
41             .filter(pessoa -> pessoa.getId() == id)
42             .findFirst()
43             .orElse(null);
44     }
45
46     //metodo obter todos
47     public List<PessoaJuridica> obterTodos(){
48         return new ArrayList <>(pessoasJuridicas);
49     }
50
51     //metodo persistir
52     public void persistir(String nomeArquivo) throws IOException {
53         try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream(nomeArquivo))) {
54             outputStream.writeObject(pessoasJuridicas);
55         }
56     }
57
58     //metodo recuperar
59     @SuppressWarnings("unchecked")
60     public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {
61         try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(nomeArquivo))) {
62             pessoasJuridicas = (List<PessoaJuridica>) inputStream.readObject();
63         }
64     }
65 }
```

Imagem 04 – Classe PessoaJuridicaRepo

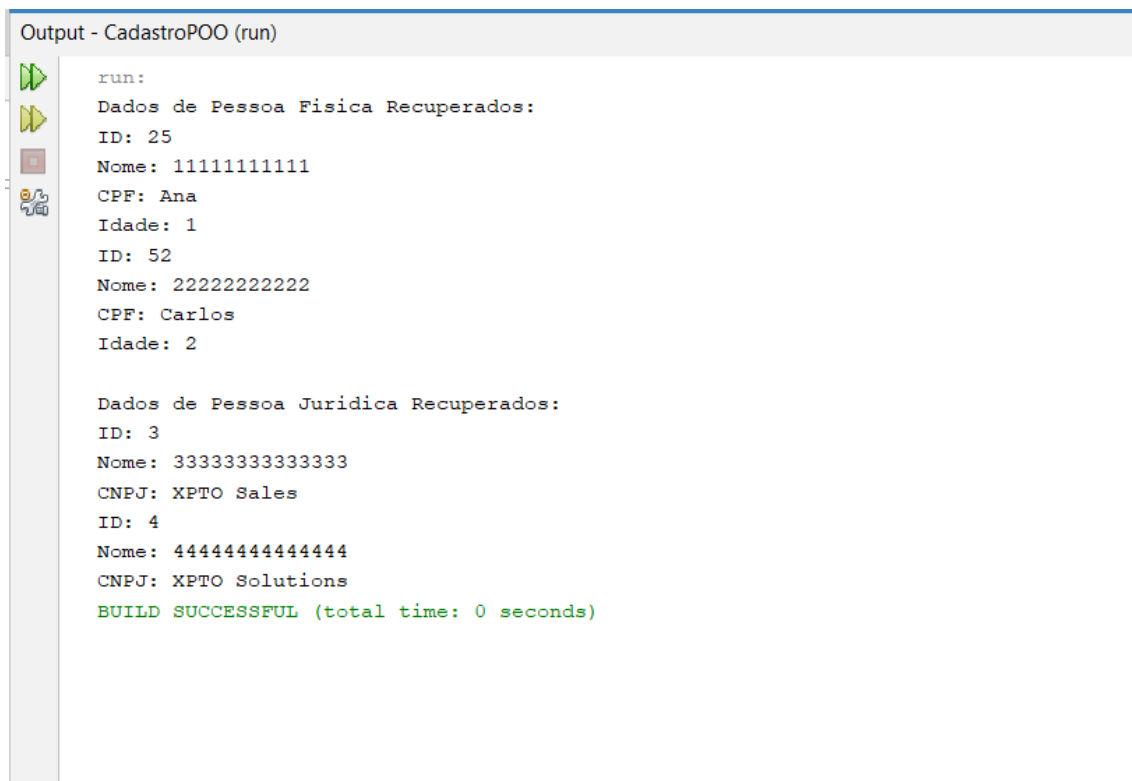
Com isso, foi criada duas classes, uma que foi a “MainCadastro” com cadastros já definidos para termos uma ideia que o código esteja todo funcionando sem erros como mostra abaixo o código:



```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5  package cadastropoo;
6
7  import cadastropoo.model.PessoaFisica;
8  import cadastropoo.model.PessoaFisicaRepo;
9  import cadastropoo.model.PessoaJuridica;
10 import cadastropoo.model.PessoaJuridicaRepo;
11 import java.io.IOException;
12
13 /**
14 *
15 * @author Alaim
16 */
17 public class MainCadastro {
18     public static void main(String[] args) {
19         //Repositório (repo1)
20         PessoaFisicaRepo repo1 = new PessoaFisicaRepo();
21
22         // adicionando pessoas
23         PessoaFisica pessoaFisica1 = new PessoaFisica(1, "Ana", "1111111111", 25);
24         PessoaFisica pessoaFisica2 = new PessoaFisica(2, "Carlos", "2222222222", 52);
25
26         repo1.inserir(pessoaFisica1);
27         repo1.inserir(pessoaFisica2);
28
29         try {
30             //método de persistência em repo1
31             repo1.persistir("pessoasFisicas.dat");
32
33             //Repositório (repo2).
34             PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
35
36             //método de recuperação em repo2
37
38             repo2.recuperar("pessoasFisicas.dat");
39
40             // Exibir os dados de todas as pessoas físicas recuperadas.
41             System.out.println("Dados de Pessoa Fisica Recuperados:");
42             repo2.obterTodos().forEach(PessoaFisica::exibir);
43
44             //Repositório (repo3).
45             PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
46
47             //Adicionando pessoas, utilizando o construtor completo.
48             PessoaJuridica pessoaJuridica1 = new PessoaJuridica(3, "XPTO Sales", "33333333333333");
49             PessoaJuridica pessoaJuridica2 = new PessoaJuridica(4, "XPTO Solutions", "44444444444444");
50
51             repo3.inserir(pessoaJuridica1);
52             repo3.inserir(pessoaJuridica2);
53
54             //método de persistência em repo3
55             repo3.persistir("pessoasJuridicas.dat");
56
57             //Repositório (repo4).
58             PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
59
60             //método de recuperação repo4
61             repo4.recuperar("pessoasJuridicas.dat");
62
63             // Exibir os dados de todas as pessoas jurídicas recuperadas.
64             System.out.println("\nDados de Pessoa Juridica Recuperados:");
65             repo4.obterTodos().forEach(PessoaJuridica::exibir);
66         } catch (IOException | ClassNotFoundException e) {
67             e.printStackTrace();
68         }
69     }
70 }
```

Imagem 05 – Classe MainCadastro

E logo em seguida, temos o resultado esperado do:



```
run:
Dados de Pessoa Fisica Recuperados:
ID: 25
Nome: 11111111111
CPF: Ana
Idade: 1
ID: 52
Nome: 22222222222
CPF: Carlos
Idade: 2

Dados de Pessoa Juridica Recuperados:
ID: 3
Nome: 33333333333333
CNPJ: XPTO Sales
ID: 4
Nome: 44444444444444
CNPJ: XPTO Solutions
BUILD SUCCESSFUL (total time: 0 seconds)
```

Imagem 06 – resultado do MainCadastro

2º Procedimento | Criação do Cadastro em Modo Texto

Finalizado essa parte do cadastro com informações definidas do primeiro procedimento, foi criada outra classe para poder solicitar ao usuário para preencher o mesmo e obter as informações do cadastro, com as opções necessárias de criação, alteração, leitura e exclusão de cada item cadastrado.

Logo em seguida, tem uma breve explicação do significado de cada parte do processo.

Elementos estáticos

Elementos estáticos no Java são aqueles que “existem” sem a necessidade de serem instanciados. Eles estão disponíveis para uso diretamente, sem a criação de objetos.

Exemplos de elementos estáticos incluem métodos estáticos e variáveis estáticas.

Por que o método main é estático?

O método “main” é declarado como estático porque é o ponto de entrada de um programa Java e precisa ser invocado pelo sistema Java antes da criação de qualquer instância da classe que o contém.

Classe Scanner

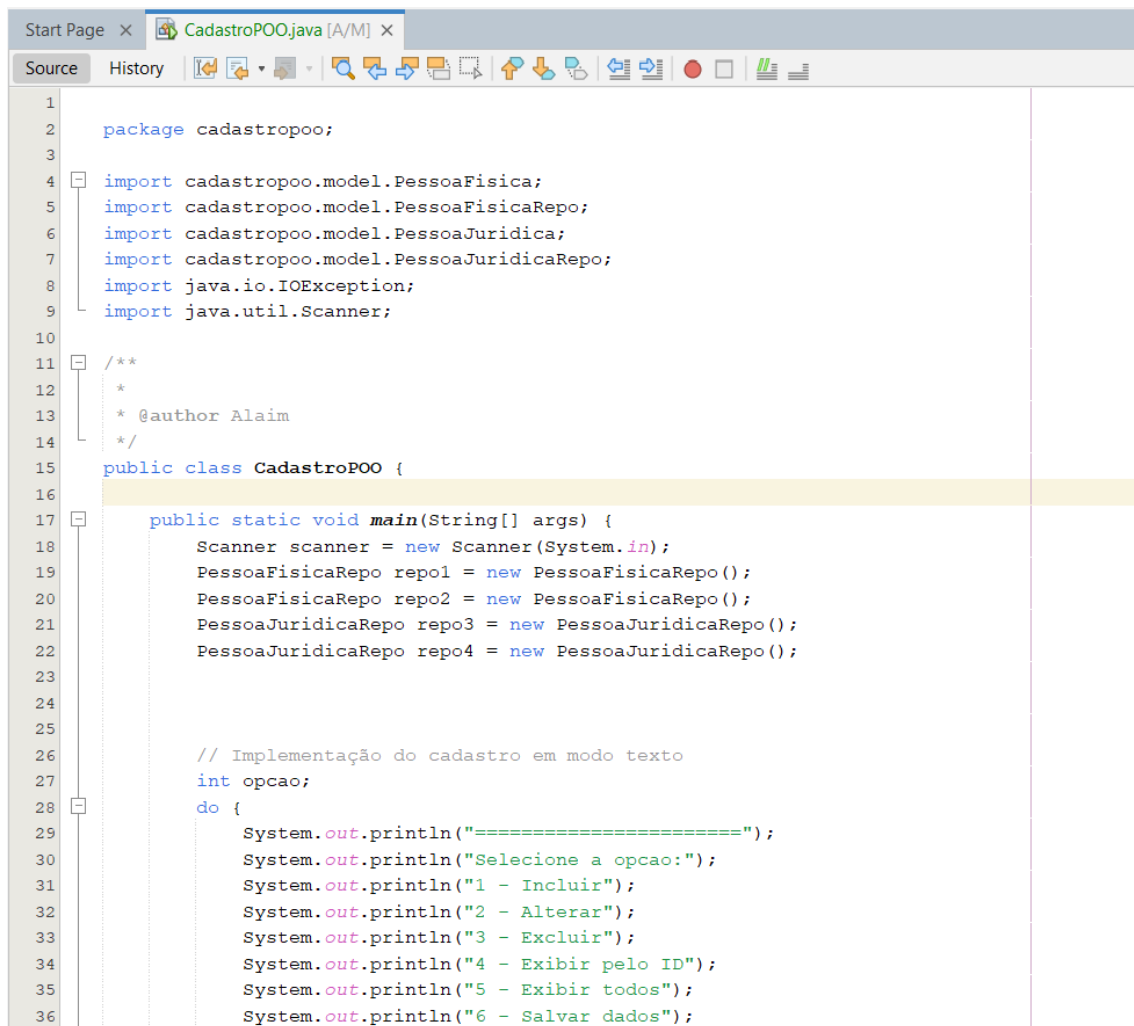
A classe Scanner no Java é utilizada para facilitar a entrada de dados a partir do console.

Ela permite ler valores de diferentes tipos primitivos (inteiros, números de ponto flutuante, strings etc.) a partir do teclado.

Impacto das Classes de Repositório na Organização do Código

O uso de classes de repositório, como **PessoaFisicaRepo** e **PessoaJuridicaRepo**, proporcionou uma organização mais modular e clara do código. Essas classes encapsulam a lógica de gerenciamento de dados relacionados a entidades específicas, separando a lógica de negócios da lógica de persistência. Isso facilita a manutenção do código, torna mais fácil adicionar novas funcionalidades e promove uma organização mais eficiente dos dados.

Segue abaixo os códigos da classe CadastroPOO:



```
1
2 package cadastrpoo;
3
4 import cadastrpoo.model.PessoaFisica;
5 import cadastrpoo.model.PessoaFisicaRepo;
6 import cadastrpoo.model.PessoaJuridica;
7 import cadastrpoo.model.PessoaJuridicaRepo;
8 import java.io.IOException;
9 import java.util.Scanner;
10
11 /**
12  *
13  * @author Alaim
14  */
15 public class CadastroPOO {
16
17     public static void main(String[] args) {
18         Scanner scanner = new Scanner(System.in);
19         PessoaFisicaRepo repo1 = new PessoaFisicaRepo();
20         PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
21         PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
22         PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
23
24
25
26         // Implementação do cadastro em modo texto
27         int opcao;
28         do {
29             System.out.println("=====");
30             System.out.println("Selecione a opcao:");
31             System.out.println("1 - Incluir");
32             System.out.println("2 - Alterar");
33             System.out.println("3 - Excluir");
34             System.out.println("4 - Exibir pelo ID");
35             System.out.println("5 - Exibir todos");
36             System.out.println("6 - Salvar dados");
```

Imagem 07

```

37 System.out.println("7 - Recuperar dados");
38 System.out.println("0 - Finalizar execucao");
39 System.out.println("=====");
40
41 opcao = scanner.nextInt();
42 scanner.nextLine(); // Limpar o buffer
43
44 switch (opcao) {
45     case 1:
46         System.out.println("Escolha o tipo (F -> Fisica ou J -> Juridica):");
47         String tipo = scanner.nextLine();
48         if (tipo.equalsIgnoreCase("F")) {
49             System.out.println("Digite o ID:");
50             int id = scanner.nextInt();
51             scanner.nextLine(); // Limpar o buffer
52             System.out.println("Digite o nome:");
53             String nome = scanner.nextLine();
54             System.out.println("Digite o CPF:");
55             String cpf = scanner.nextLine();
56             System.out.println("Digite a idade:");
57             int idade = scanner.nextInt();
58             scanner.nextLine(); // Limpar o buffer
59             repo1.inserir(new PessoaFisica(id, nome, cpf, idade));
60         } else if (tipo.equalsIgnoreCase("J")) {
61             System.out.println("Digite o ID:");
62             int id = scanner.nextInt();
63             scanner.nextLine(); // Limpar o buffer
64             System.out.println("Digite o nome:");
65             String nome = scanner.nextLine();
66             System.out.println("Digite o CNPJ:");
67             String cnpj = scanner.nextLine();
68             repo3.inserir(new PessoaJuridica(id, nome, cnpj));
69         } else {
70             System.out.println("Tipo inválido.");
71         }
72         break;
73     case 2:
74         System.out.println("Escolha o tipo (F -> Fisica ou J -> Juridica):");
75         tipo = scanner.nextLine();
76         System.out.println("Digite o ID:");
77         int id = scanner.nextInt();
78         scanner.nextLine(); // Limpar o buffer
79         if (tipo.equalsIgnoreCase("F")) {
80             PessoaFisica pessoaFisica = repo1.obter(id);
81             if (pessoaFisica != null) {
82                 System.out.println("Dados atuais:");
83                 pessoaFisica.exibir();
84                 System.out.println("Digite o novo nome:");
85                 String novoNome = scanner.nextLine();
86                 pessoaFisica.setNome(novoNome);
87                 System.out.println("Digite o novo CPF:");
88                 String novoCpf = scanner.nextLine();
89                 pessoaFisica.setCpf(novoCpf);
90                 System.out.println("Digite a nova idade:");
91                 int novaIdade = scanner.nextInt();
92                 scanner.nextLine(); // Limpar o buffer
93                 pessoaFisica.setIdade(novaIdade);
94                 repo1.alterar(pessoaFisica);
95                 System.out.println("Pessoa fisica alterada com sucesso.");
96             } else {
97                 System.out.println("Pessoa fisica não encontrada.");
98             }
99         } else if (tipo.equalsIgnoreCase("J")) {
100             PessoaJuridica pessoaJuridica = repo3.obter(id);
101             if (pessoaJuridica != null) {
102                 System.out.println("Dados atuais:");

```

Imagem 08

```

103         pessoaJuridica.exibir();
104         System.out.println("Digite o novo nome:");
105         String novoNome = scanner.nextLine();
106         pessoaJuridica.setNome(novoNome);
107         System.out.println("Digite o novo CNPJ:");
108         String novoCnpj = scanner.nextLine();
109         pessoaJuridica.setCnpj(novoCnpj);
110         repo3.alterar(pessoaJuridica);
111         System.out.println("Pessoa juridica alterada com sucesso.");
112     } else {
113         System.out.println("Pessoa juridica não encontrada.");
114     }
115 } else {
116     System.out.println("Tipo invalido.");
117 }
118 break;
119 case 3:
120     System.out.println("Escolha o tipo (F -> Fisica ou J -> Juridica):");
121     tipo = scanner.nextLine();
122     System.out.println("Digite o ID:");
123     id = scanner.nextInt();
124     scanner.nextLine(); // Limpar o buffer
125     if (tipo.equalsIgnoreCase("F")) {
126         repo1.excluir(id);
127         System.out.println("Pessoa fisica excluida com sucesso.");
128     } else if (tipo.equalsIgnoreCase("J")) {
129         repo3.excluir(id);
130         System.out.println("Pessoa juridica excluida com sucesso.");
131     } else {
132         System.out.println("Tipo invalido.");
133     }
134     break;
135 case 4:
136     System.out.println("Escolha o tipo (F -> Fisica ou J -> Juridica):");
137     tipo = scanner.nextLine();
138     System.out.println("Digite o ID:");
139     id = scanner.nextInt();
140     scanner.nextLine(); // Limpar o buffer
141     if (tipo.equalsIgnoreCase("F")) {
142         PessoaFisica pessoaFisica = repo1.obter(id);
143         if (pessoaFisica != null) {
144             System.out.println("Dados da pessoa fisica:");
145             pessoaFisica.exibir();
146         } else {
147             System.out.println("Pessoa fisica nao encontrada.");
148         }
149     } else if (tipo.equalsIgnoreCase("J")) {
150         PessoaJuridica pessoaJuridica = repo3.obter(id);
151         if (pessoaJuridica != null) {
152             System.out.println("Dados da pessoa juridica:");
153             pessoaJuridica.exibir();
154         } else {
155             System.out.println("Pessoa juridica não encontrada.");
156         }
157     } else {
158         System.out.println("Tipo invalido.");
159     }
160     break;
161 case 5:
162     System.out.println("Escolha o tipo (F -> Fisica ou J -> Juridica):");
163     tipo = scanner.nextLine();
164     if (tipo.equalsIgnoreCase("F")) {
165         System.out.println("Pessoas fisicas cadastradas:");
166         for (PessoaFisica pessoa : repo1.obterTodos()) {
167             pessoa.exibir();
168             System.out.println();
169         }
170     } else if (tipo.equalsIgnoreCase("J")) {
171         System.out.println("Pessoas juridicas cadastradas:");

```

```

172         for (PessoaJuridica pessoa : repo3.obterTodos()) {
173             pessoa.exibir();
174             System.out.println();
175         }
176     } else {
177         System.out.println("Tipo invalido.");
178     }
179     break;
180 case 6:
181     try {
182         repo1.persistir("pessoas_fisicas_cadastro.bin");
183         repo3.persistir("pessoas_juridicas_cadastro.bin");
184         System.out.println("Dados salvos com sucesso.");
185     } catch (IOException e) {
186         e.printStackTrace();
187     }
188     break;
189 case 7:
190     try {
191         repo2.recuperar("pessoas_fisicas_cadastro.bin");
192         repo4.recuperar("pessoas_juridicas_cadastro.bin");
193         System.out.println("Dados recuperados com sucesso.");
194     } catch (IOException | ClassNotFoundException e) {
195         e.printStackTrace();
196     }
197     break;
198 case 0:
199     System.out.println("Finalizando execucao.");
200     break;
201 default:
202     System.out.println("Opção invalida.");
203 }
204 } while (opcao != 0);
205 }
206
207 }

```

Imagem 10

Logo abaixo segue o resultado do cadastro:

```
Output - CadastroPOO (run)

run:
=====
Selecione a opcao:
1 - Incluir
2 - Alterar
3 - Excluir
4 - Exibir pelo ID
5 - Exibir todos
6 - Salvar dados
7 - Recuperar dados
0 - Finalizar execucao
=====
1
Escolha o tipo (F -> Fisica ou J -> Juridica):
F
Digite o ID:
1290
Digite o nome:
Alaim Almeida
Digite o CPF:
44455566678
Digite a idade:
34
=====
```

Imagem 11 – resultado do cadastro

Depois do cadastro salvo, foi solicitado a exibição do cadastro armazenado:

```
=====
Selecione a opcao:
1 - Incluir
2 - Alterar
3 - Excluir
4 - Exibir pelo ID
5 - Exibir todos
6 - Salvar dados
7 - Recuperar dados
0 - Finalizar execucao
=====
6
Dados salvos com sucesso.
=====
Selecione a opcao:
1 - Incluir
2 - Alterar
3 - Excluir
4 - Exibir pelo ID
5 - Exibir todos
6 - Salvar dados
7 - Recuperar dados
0 - Finalizar execucao
=====
5
Escolha o tipo (F -> Fisica ou J -> Juridica):
F
Pessoas fisicas cadastradas:
ID: 34
Nome: 44455566678
CPF: Alaim Almeida
Idade: 1290
=====
```

Imagem 12 – resultado do cadastro armazenado

Com isso, foi finalizado todo os dois procedimentos na criação do cadastro.

No exercício solicitado, foi utilizado o Apache NetBeans IDE 20 e a versão do JDK do Java foi a 21.0.2.

Segue abaixo o repositório no GitHub de todo o projeto:

[alaimalmeida/CadastroPOO \(github.com\)](https://github.com/alaimalmeida/CadastroPOO)