



Missão Prática do Nível 1 - Iniciando pelo caminho do Java

Lorena Rosa Borges Sanches - 202204376067

Campus Uberlândia/MG

Iniciando pelo caminho do Java – 2023.2 – 3º Semestre

Desenvolvimento Full Stack

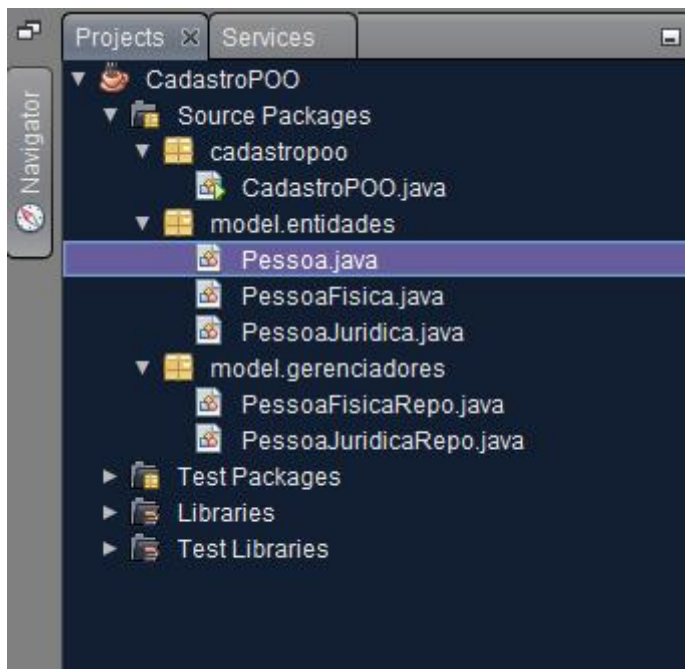
Link do repositório Git: <https://github.com/LorenaBorgesSanches/Mundo3-Pratica1>

Objetivo da Prática

O presente trabalho tem como objetivo abordar conceitos utilizados no desenvolvimento da Missão Prática do Nível 1 (parte 1 e 2), do curso de Desenvolvimento Full Stack. De forma, responder questões e elucidar os códigos e conceitos utilizados na prática. As imagens a seguir correspondem aos códigos da presente prática e a seguir o relatório discente de acompanhamento.

1º Procedimento – Criação das Entidades e Sistema de Persistência

1. O projeto é iniciado com a criação de um projeto do tipo Ant..Java Application no Net Beans, utilizando o nome CadastroPOO para o projeto;
2. Um pacote com o nome “model”, para as entidades e gerenciadores;



3. No pacote model criar as **entidades**, com as seguintes características:

a. Classe Pessoa, com os campos **id** (inteiro) e **nome** (texto), método **exibir**, para impressão dos dados, construtor padrão e completo, além de getters e setters para todos os campos.

```
1 package model.entidades;
2
3 import java.io.Serializable;
4
5 public class Pessoa implements Serializable {
6     private int id;
7     private String nome;
8
9     public Pessoa (int id, String nome){
10         this.id = id;
11         this.nome = nome;
12     }
13
14     public int getId(){
15         return id;
16     }
17
18     public void setId(int id) {
19         this.id = id;
20     }
21
22     public String getNome() {
23         return nome;
24     }
25
26     public void setNome(String nome) {
27         this.nome = nome;
28     }
29
30     public String exibir() {
31         return "Id: "
32             + id
33             + "\nNome: "
34             + nome;
35     }
36 }
37
38
39
40
41
```

b. Classe **PessoaFisica**, herdando de Pessoa, com o acréscimo dos campos **cpf** (texto) e **idade** (inteiro), método **exibir** polimórfico, construtores, getters e setters.

```
1 package model.entidades;
2 import java.io.Serializable;
3
4 public class PessoaFisica extends Pessoa implements Serializable {
5     private String cpf;
6     private int idade;
7
8     public PessoaFisica(int id, String nome, String cpf, int idade) {
9         super(id, nome);
10         this.cpf = cpf;
11         this.idade = idade;
12     }
13
14     public String getCpf() {
15         return cpf;
16     }
17
18     public void setCpf(String cpf) {
19         this.cpf = cpf;
20     }
21
22     public int getIdade() {
23         return idade;
24     }
25
26     public void setIdade(int idade) {
27         this.idade = idade;
28     }
29
30     @Override
31     public String exibir() {
32         return super.exibir() + "\nCPF: "
33             + cpf
34             + "\nIdade: "
35             + idade;
36     }
37 }
38
39
40
41
```

c. Classe **PessoaJuridica**, herdando de **Pessoa**, com o acréscimo do campo **cnpj** (texto), método **exibir** polimórfico, construtores, getters e setters.

```
1 package model.entidades;
2 import java.io.Serializable;
3
4 public class PessoaJuridica extends Pessoa implements Serializable {
5
6     private String cnpj;
7
8     public PessoaJuridica(int id, String nome, String cnpj) {
9         super(id, nome);
10        this.cnpj = cnpj;
11    }
12
13    public String getCnpj() {
14        return cnpj;
15    }
16
17    public void setCnpj(String cnpj){
18        this.cnpj = cnpj;
19    }
20
21    @Override
22    public String exibir() {
23        return super.exibir() + "\nCnpj: "
24            + cnpj;
25    }
26 }
```

d. Adicionar a interface **Serializable** em todas as classes.

Foram adicionadas no momento da criação das classes e conforme demonstra as imagens dos itens a, b e c, usando a palavra-chave reservada “implements”.

4. No pacote **model** criar os gerenciadores, com as seguintes características:

a. Classe **PessoaFisicaRepo**, contendo um **ArrayList** de **PessoaFisica**, nível de acesso privado, e métodos públicos **inserir**, **alterar**, **excluir**, **obter** e **obterTodos**, para gerenciamento das entidades contidas no **ArrayList**.

```
12 public class PessoaFisicaRepo {
13
14     private ArrayList<PessoaFisica> repo;
15
16     public PessoaFisicaRepo() {
17         repo = new ArrayList<>();
18     }
19
20     public void inserir(PessoaFisica pessoa) {
21         repo.add(pessoa);
22     }
23
24     public void alterar(PessoaFisica pessoa) {
25         for (int i = 0; i < repo.size(); i++) {
26             if (pessoa.getId() == repo.get(i).getId()) {
27                 repo.set(i, pessoa);
28                 break;
29             }
30         }
31     }
32
33     public void excluir(int Id) {
34         Iterator<PessoaFisica> iterator = repo.iterator();
35         while (iterator.hasNext()) {
36             PessoaFisica pessoa = iterator.next();
37             if (pessoa.getId() == Id) {
38                 iterator.remove();
39                 break;
40             }
41         }
42     }
43
44     public PessoaFisica obter(int Id) {
45         for (PessoaFisica pessoa : repo) {
46             if (pessoa.getId() == Id) {
47                 return pessoa;
48             }
49         }
50         return null;
51     }
52
53     public ArrayList<PessoaFisica> obterTodos() {
54         return repo;
55     }
56 }
```

b. Classe **PessoaJuridicaRepo**, com um **ArrayList** de **PessoaJuridica**, nível de acesso privado, e métodos públicos **inserir**, **alterar**, **excluir**, **obter** e **obterTodos**, para gerenciamento das entidades contidas no **ArrayList**.

```
13 public class PessoaJuridicaRepo {
14
15     private ArrayList<PessoaJuridica> repo;
16
17     public PessoaJuridicaRepo() {
18         repo = new ArrayList<>();
19     }
20
21     public void inserir(PessoaJuridica pessoa) {
22         repo.add(pessoa);
23     }
24
25     public void alterar(PessoaJuridica pessoa) {
26         for (int i = 0; i < repo.size(); i++) {
27             if (pessoa.getId() == repo.get(i).getId()) {
28                 repo.set(i, pessoa);
29                 break;
30             }
31         }
32     }
33
34     public void excluir(int Id) {
35         Iterator<PessoaJuridica> iterator = repo.iterator();
36         while (iterator.hasNext()) {
37             PessoaJuridica pessoa = iterator.next();
38             if (pessoa.getId() == Id) {
39                 iterator.remove();
40                 break;
41             }
42         }
43     }
44
45     public PessoaJuridica obter(int Id) {
46         for (PessoaJuridica pessoa : repo) {
47             if (pessoa.getId() == Id) {
48                 return pessoa;
49             }
50         }
51         return null;
52     }
53
54     public ArrayList<PessoaJuridica> obterTodos() {
55         return repo;
56     }
57 }
```

c. Em ambos os gerenciadores adicionar o método público **persistir**, com a recepção do nome do arquivo, para armazenagem dos dados no disco.

Método de persistir de PessoaFisicaRepo:

```
57
58 public void persistir(String caminho) throws FileNotFoundException, IOException {
59     FileOutputStream fout = new FileOutputStream(name: caminho);
60     ObjectOutputStream oos = new ObjectOutputStream(out: fout);
61     oos.writeObject(obj: repo);
62 }
```

Método de persistir de PessoaJuridicaRepo:

```
57
58 public void persistir(String caminho) throws FileNotFoundException, IOException {
59     FileOutputStream fout = new FileOutputStream(name: caminho);
60     ObjectOutputStream oos = new ObjectOutputStream(out: fout);
61     oos.writeObject(obj: repo);
62 }
```

d. Em ambos os gerenciadores adicionar o método público **recuperar**, com a recepção do nome do arquivo, para recuperação dos dados do disco.

Método de recuperar de PessoaFisicaRepo:

```
63
64 public void recuperar(String caminho) throws FileNotFoundException, IOException, ClassNotFoundException {
65     FileInputStream fout = new FileInputStream(name: caminho);
66     ObjectInputStream oos = new ObjectInputStream(in: fout);
67     repo = (ArrayList<PessoaFisica>) oos.readObject();
68 }
69 }
```

Método de recuperar de PessoaJuridicaRepo:

```
63
64 public void recuperar(String caminho) throws FileNotFoundException, IOException, ClassNotFoundException {
65     FileInputStream fout = new FileInputStream(name: caminho);
66     ObjectInputStream oos = new ObjectInputStream(in: fout);
67     repo = (ArrayList<PessoaJuridica>) oos.readObject();
68 }
69 }
```

e. Os métodos persistir e recuperar devem ecoar (**throws**) exceções.

Referente a PessoaFisicaRepo:

```
57
58 public void persistir(String caminho) throws FileNotFoundException, IOException {
59     FileOutputStream fout = new FileOutputStream(name: caminho);
60     ObjectOutputStream oos = new ObjectOutputStream(out: fout);
61     oos.writeObject(obj: repo);
62 }
63
64 public void recuperar(String caminho) throws FileNotFoundException, IOException, ClassNotFoundException {
65     FileInputStream fout = new FileInputStream(name: caminho);
66     ObjectInputStream oos = new ObjectInputStream(in: fout);
67     repo = (ArrayList<PessoaFisica>) oos.readObject();
68 }
69 }
```

Referente a PessoaJuridicaRepo:

```
58 public void persistir(String caminho) throws FileNotFoundException, IOException {
59     FileOutputStream fout = new FileOutputStream(name: caminho);
60     ObjectOutputStream oos = new ObjectOutputStream(fout);
61     oos.writeObject(repo);
62 }
63
64 public void recuperar(String caminho) throws FileNotFoundException, IOException, ClassNotFoundException {
65     FileInputStream fin = new FileInputStream(name: caminho);
66     ObjectInputStream ois = new ObjectInputStream(fin);
67     repo = (ArrayList<PessoaJuridica>) ois.readObject();
68 }
69 }
```

f. O método obter deve retornar uma entidade a partir do id.

Referente a PessoaFisicaRepo:

```
44
45 public PessoaFisica obter(int Id) {
46     for (PessoaFisica pessoa : repo) {
47         if (pessoa.getId() == Id) {
48             return pessoa;
49         }
50     }
51     return null;
52 }
53 }
```

Referente a PessoaJuridicaRepo:

```
44
45 public PessoaFisica obter(int Id) {
46     for (PessoaFisica pessoa : repo) {
47         if (pessoa.getId() == Id) {
48             return pessoa;
49         }
50     }
51     return null;
52 }
53 }
```

g. Os métodos inserir e alterar devem ter entidades como parâmetros.

Referente a PessoaFisicaRepo:

```
21 public void inserir(PessoaFisica pessoa) {
22     repo.add(pessoa);
23 }
24
25 public void alterar(PessoaFisica pessoa) {
26     for (int i = 0; i < repo.size(); i++) {
27         if (pessoa.getId() == repo.get(i).getId()) {
28             repo.set(i, pessoa);
29             break;
30         }
31     }
32 }
33 }
```


Referente a PessoaJuridicaRepo:

```
21 public void inserir(PessoaJuridica pessoa) {
22     repo.add(=: pessoa);
23 }
24
25 public void alterar(PessoaJuridica pessoa) {
26     for (int i = 0; i < repo.size(); i++) {
27         if (pessoa.getId() == repo.get(index: i).getId()) {
28             repo.set(index: i, element: pessoa);
29             break;
30         }
31     }
32 }
```

h. O método excluir deve receber o id da entidade para exclusão.

Referente a PessoaFisicaRepo:

```
34 public void excluir(int Id) {
35     Iterator<PessoaFisica> iterator = repo.iterator();
36     while (iterator.hasNext()) {
37         PessoaFisica pessoa = iterator.next();
38         if (pessoa.getId() == Id) {
39             iterator.remove();
40             break;
41         }
42     }
43 }
```

Referente a PessoaJuridicaRepo:

```
34 public void excluir(int Id) {
35     Iterator<PessoaJuridica> iterator = repo.iterator();
36     while (iterator.hasNext()) {
37         PessoaJuridica pessoa = iterator.next();
38         if (pessoa.getId() == Id) {
39             iterator.remove();
40             break;
41         }
42     }
43 }
```

i. O método obterTodos deve retornar o conjunto completo de entidades.

Referente a PessoaFisicaRepo:

```
54 public ArrayList<PessoaFisica> obterTodos() {  
55     return repo;  
56 }
```

Referente a PessoaJuridicaRepo:

```
54 public ArrayList<PessoaJuridica> obterTodos() {  
55     return repo;  
56 }
```

5. Alterar o método **main** da classe principal para testar os repositórios:

a. Instanciar um repositório de pessoas físicas (**repo1**):

```
12  
13 PessoaFisicaRepo repo1 = new PessoaFisicaRepo();
```

b. Adicionar duas pessoas físicas, utilizando o construtor completo:

```
14 PessoaFisica pessoa1 = new PessoaFisica(id: 1, nome: "Ana", cpf: "11111111111", idade: 25);  
15 PessoaFisica pessoa2 = new PessoaFisica(id: 2, nome: "Carlos", cpf: "22222222222", idade: 52);
```

c. Invocar o método de persistência de repo1, fornecendo um nome de arquivo fixo, através do código:

```
17 repo1.inserir(pessoa: pessoa1);  
18 repo1.inserir(pessoa: pessoa2);  
19  
20 try {  
21     repo1.persistir(caminho: "./repoPessoasFisicas.db");  
22 } catch (IOException ex) {  
23 }  
24 System.out.println("Dados de Pessoa Fisica Armazenados.");
```

d. Instanciar outro repositório de pessoas físicas (**repo2**):

```
26 PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
```

e. Invocar o método de recuperação em repo2, fornecendo o mesmo nome de arquivo utilizado anteriormente:

```
27 try {  
28     repo2.recuperar(caminho: "./repoPessoasFisicas.db");  
29 } catch (IOException | ClassNotFoundException ex) {  
30 }  
31 System.out.println("Dados de Pessoa Fisica Recuperados.");  
32
```

f. Exibir os dados de todas as pessoas físicas recuperadas:

```
33 repo2.obterTodos();  
34 for (PessoaFisica pessoa : repo2.obterTodos()) {  
35     System.out.println(":" + pessoa.exibir());  
36 }
```


g. Instanciar um repositório de pessoas jurídicas (**repo3**):

```
38 PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
```

h. Adicionar duas pessoas jurídicas, utilizando o construtor completo:

```
39 PessoaJuridica pessoa3 = new PessoaJuridica(id: 3, nome: "XPTO Sales", cnpj: "333333333333333333");
40 PessoaJuridica pessoa4 = new PessoaJuridica(id: 4, nome: "XPTO Solutions", cnpj: "444444444444444444");
```

i. Invocar o método de persistência de repo3, fornecendo um nome de arquivo fixo, através do código:

```
42 repo3.inserir(pessoa: pessoa3);
43 repo3.inserir(pessoa: pessoa4);
44
45 try {
46     repo3.persistir(caminho: "./repoPessoasJuridicas.db");
47 } catch (IOException ex) {
48 }
49 System.out.println("Dados de Pessoa Juridica Armazenados.");
50
```

j. Instanciar outro repositório de pessoas jurídicas (**repo4**):

```
51 PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
```

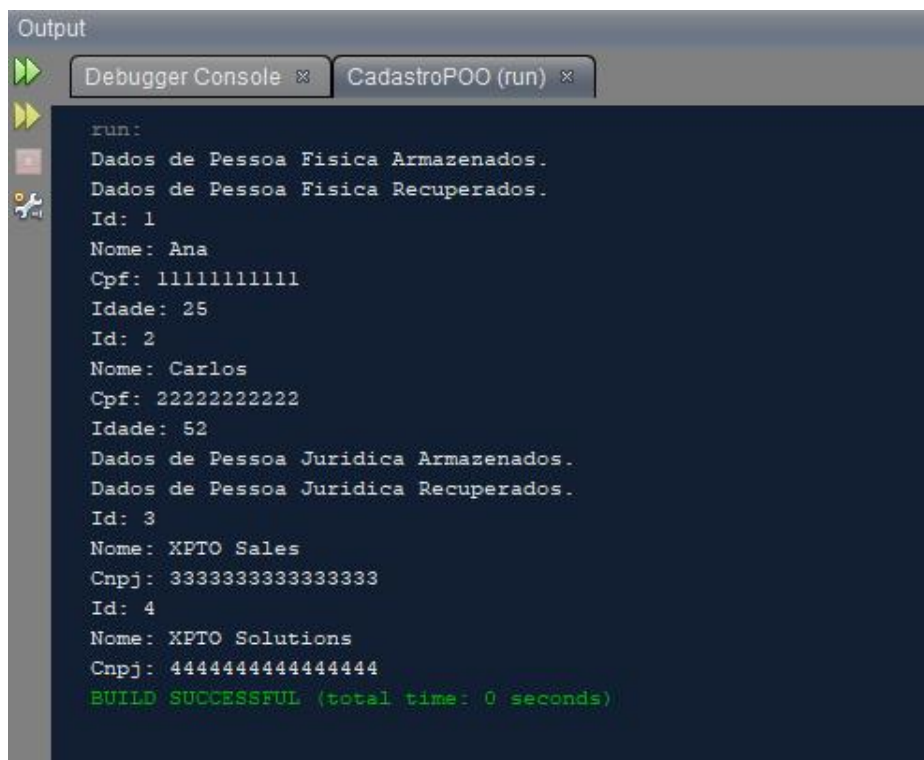
k. Invocar o método de recuperação de repo4, fornecendo o mesmo nome de arquivo utilizado anteriormente:

```
52 try {
53     repo4.recuperar(caminho: "./repoPessoasJuridicas.db");
54 } catch (IOException | ClassNotFoundException ex) {
55 }
56 System.out.println("Dados de Pessoa Juridica Recuperados.");
57
```

l. Exibir os dados de todas as pessoas jurídicas recuperadas:

```
57
58 repo4.obterTodos();
59 for (PessoaJuridica pessoa : repo4.obterTodos()) {
60     System.out.println(":" + pessoa.exibir());
61 }
62
```

6. Ajustar as características para obter uma execução como a seguinte:
Esse é o resultado da execução do código apresentado acima.



```
run:
Dados de Pessoa Fisica Armazenados.
Dados de Pessoa Fisica Recuperados.
Id: 1
Nome: Ana
Cpf: 111111111111
Idade: 25
Id: 2
Nome: Carlos
Cpf: 222222222222
Idade: 52
Dados de Pessoa Juridica Armazenados.
Dados de Pessoa Juridica Recuperados.
Id: 3
Nome: XPTO Sales
Cnpj: 3333333333333333
Id: 4
Nome: XPTO Solutions
Cnpj: 4444444444444444
BUILD SUCCESSFUL (total time: 0 seconds)
```

7. Relatório discente de acompanhamento - Análise e conclusão (parte 1)

a. Vantagens e desvantagens do uso de Herança:

Em Java e em outras linguagens de programação, se utiliza das vantagens do uso de herança em programação orientada a objetos. O principal proveito da herança é a possibilidade de se reutilizar uma classe com seus atributos e métodos (classe pai) para criar classes especializadas (classes filhas). O reuso do código otimiza tempo e esforço de desenvolvimento.

Outra vantagem da herança é a possibilidade de criar classes derivadas que estendem ou especializam o comportamento da classe pai. Dessa forma, pode-se criar novos atributos e métodos nas classes filhas e não alterar a classe original, que pode servir de base para outras classes derivadas.

A herança também serve de base para o polimorfismo, que permite tratar objetos de diferentes classes derivadas da mesma forma. Aumentando a flexibilidade e a capacidade de extensão do código.

Para usufruir da herança em Java utiliza-se a palavra reservada “extends”. Por exemplo: `public class Animais extends Cachorro`. A classe pai é genérica e serve de “forma” para a criação da classe que especializa.

A classe que deriva as outras classes chamamos de superclasse, enquanto a classe que é derivada denomina subclasse.

Existem ainda classes que não podem ser herdadas, assim determinadas no momento da sua criação, são aquelas criadas com a palavra reservada “final”. Convenientemente, analisaremos as desvantagens do uso da herança.

Quando se impede o uso da herança garante maior segurança a classes que desempenham um papel crítico no sistema e não devem ser alteradas ou estendidas. Portanto, não é toda classe que convém ser herdada.

Outra vantagem do uso de “final” é a otimização da performance, uma vez que o compilador entende que aquela classe não será herdada e não precisará de todas as verificações associadas a ela. Além disso, criar uma classe imutável garante consistência e previne alterações indesejadas os objetos.

Portanto, o uso da herança deve ser ponderada para a finalidade do sistema. Sendo incrivelmente útil em alguns casos e outros nem tanto.

b. Por que a Interface Serializable é necessária ao efetuar persistência em arquivos binários?

A interface Serializable em java é uma interface de marcação que tem como função serializar uma classe. Ao implementa-la em uma classe, os objetos serão convertidos em uma sequência de bytes, permitindo que seja persistida em arquivos binários, transmissão por rede ou armazenamento em memória.

A serialização é um processo de converter os objetos em uma representação binária, que é necessária para a persistência de dados. Por esse motivo a interface em questão é tão importante, pois através da conversão em sequência de bytes os dados podem ser gravados em disco ou transmitido pela rede e posteriormente reconstituídos no estado original.

A interface Serializable não possui métodos para implementar. No momento da criação da classe define-se a interface, e ela, juntamente com o java, possui um mecanismo de verificação para assegurar que um objeto pode ser convertido em bytes antes de ser gravado em um arquivo.

c. Como o paradigma funcional é utilizado pela API stream no Java?

O paradigma funcional é um estilo de programação que se baseia no conceito de funções matemáticas puras. Ele foca na avaliação de expressões e na transformação de dados por meio de funções. A API Stream introduzida no java 8 utiliza conceitos do paradigma funcional para operar em coleções de elementos de forma declarativa e expressiva.

A programação funcional no âmbito da API Stream envolve o uso de funções lambda, expressões lambda e métodos de ordem superior, para realizar operações em uma sequência de elementos.

As funções lambda são utilizadas para aplicar comportamentos específicos a serem executados em cada elemento da função. Por exemplo o map, filter, forEach e o reduce.

As expressões lambda são usadas para criar funções anônimas compactas e concisas. Elas podem ser usadas em métodos como o filter, map e reduce, para especificar as condições de filtro, transformação de dados e operações de redução.

Os métodos de ordem superior permitem executar operações em uma coleção de elementos de forma declarativa, sem precisar iterar manualmente sobre os dados. Esses métodos aceitam funções lambda como argumentos, moldando o comportamento desejado.

d. Padrão de persistência de dados de arquivo em Java:

O padrão de desenvolvimento adotado em java para a persistência de arquivos é o DAO (Data Access Object). O padrão DAO separa a lógica de acesso aos dados da lógica de negócios , em razão dessa separação, o código fica mais organizado, modular, fácil de testar e reutilizar.

2º Procedimento – Criação do Cadastro em Modo Texto:

1. Alterar o método **main** da classe principal do projeto, para implementação do cadastro em modo texto:

a. Apresentar as opções do programa para o usuário, sendo 1 para incluir, 2 para alterar, 3 para excluir, 4 para exibir pelo id, 5 para exibir todos, 6 para salvar dados, 7 para recuperar dados e 0 para finalizar a execução.

```
25         do {
26             System.out.println(x: "=====");
27             System.out.println(x: "1 - Incluir Pessoa");
28             System.out.println(x: "2 - Alterar Pessoa");
29             System.out.println(x: "3 - Excluir Pessoa");
30             System.out.println(x: "4 - Buscar pelo Id");
31             System.out.println(x: "5 - Exibir Todos");
32             System.out.println(x: "6 - Persistir Dados");
33             System.out.println(x: "7 - Recuperar Dados");
34             System.out.println(x: "0 - Finalizar Programa");
35             System.out.println(x: "=====");
36         }
```

b. Selecionada a opção **incluir**, escolher o tipo (Física ou Jurídica), receber os dados a partir do teclado e adicionar no repositório correto.

```
58     public static void incluir() {
59         System.out.println(x: "F - Pessoa Fisica | J - Pessoa Juridica ");
60         char tipoPessoa = sc.nextLine().charAt(index: 0);
61         if (tipoPessoa != 'F' && tipoPessoa != 'f' && tipoPessoa != 'J' && tipoPessoa != 'j') {
62             System.out.println(x: "Tipo de pessoa invalido");
63             return;
64         }
65
66         System.out.println(x: "Digite o id da pessoa: ");
67         int id = sc.nextInt();
68         sc.nextLine();
69
70         System.out.println(x: "Digite o nome: ");
71         String nome = sc.nextLine();
72
73         if (tipoPessoa == 'F' || tipoPessoa == 'f') {
74             System.out.println(x: "Digite o CPF: ");
75             String cpf = sc.nextLine();
76             System.out.println(x: "Digite a idade: ");
77             int idade = sc.nextInt();
78             sc.nextLine();
79             repoPF.inserir(new PessoaFisica(id, nome, cpf, idade));
80         } else {
81             System.out.println(x: "Digite o CNPJ: ");
82             String cnpj = sc.nextLine();
83             repoPJ.inserir(new PessoaJuridica(id, nome, cnpj));
84         }
85
86         System.out.println(x: "Pessoa incluida com sucesso! ");
87     }
88 }
```


c. Selecionada a opção **alterar**, escolher o tipo (Física ou Jurídica), receber o **id** a partir do teclado, apresentar os dados atuais, solicitar os novos dados e alterar no repositório correto.

```
89 public static void alterar() {
90     System.out.println("F - Pessoa Fisica | J - Pessoa Juridica ");
91     char tipoPessoa = sc.nextLine().charAt(index: 0);
92     if (tipoPessoa != 'F' && tipoPessoa != 'f' && tipoPessoa != 'J' && tipoPessoa != 'j') {
93         System.out.println("Tipo de pessoa invalido");
94         return;
95     }
96     System.out.println("Digite o id da pessoa: ");
97     int id = sc.nextInt();
98     sc.nextLine();
99
100     if (tipoPessoa == 'F' || tipoPessoa == 'f') {
101         PessoaFisica pf = repoPF.obter(id: id);
102         if (pf == null) {
103             System.out.println("Pessoa não encontrada. ");
104             return;
105         }
106         System.out.println(pf.exibir());
107
108         System.out.println("Digite o nome: ");
109         String nomePf = sc.nextLine();
110         System.out.println("Digite o CPF: ");
111         String cpfPf = sc.nextLine();
112         System.out.println("Digite a idade: ");
113         int idadePf = sc.nextInt();
114         sc.nextLine();
115         pf.setNome(nome: nomePf);
116         pf.setCpf(cpf: cpfPf);
117         pf.setIdade(idade: idadePf);
118         repoPF.alterar(pessoa: pf);
119     } else {
120         PessoaJuridica pj = repoPJ.obter(id: id);
121         if (pj == null) {
122             System.out.println("Pessoa não encontrada. ");
123             return;
124         }
125
126         System.out.println(pj.exibir());
127         System.out.println("Digite o nome: ");
128         String nomePj = sc.nextLine();
129         System.out.println("Digite o CNPJ. ");
130         String cnpjPj = sc.nextLine();
131         pj.setNome(nome: nomePj);
132         pj.setCnpj(cnpj: cnpjPj);
133         repoPJ.alterar(pessoa: pj);
134     }
135     System.out.println("Alterado com sucesso");
136 }
```

d. Seleccionada a opção **excluir**, escolher o tipo (Física ou Jurídica), receber o **id** a partir do teclado e remover do repositório correto.

```
138 public static void excluir() {
139     System.out.println("F - Pessoa Fisica | J - Pessoa Juridica ");
140     char tipoPessoa = sc.nextLine().charAt(index: 0);
141     if (tipoPessoa != 'F' && tipoPessoa != 'f' && tipoPessoa != 'J' && tipoPessoa != 'j') {
142         System.out.println("Tipo de pessoa invalido");
143         return;
144     }
145     System.out.println("Digite o id da pessoa: ");
146     int id = sc.nextInt();
147     sc.nextLine();
148
149     if (tipoPessoa == 'F' || tipoPessoa == 'f') {
150         repoPF.excluir(id: id);
151     } else {
152         repoPJ.excluir(id: id);
153     }
154     System.out.println("Excluido com sucesso. ");
155 }
156
```

e. Seleccionada a opção **obter**, escolher o tipo (Física ou Jurídica), receber o **id** a partir do teclado e apresentar os dados atuais para a entidade.

```
157 public static void obter() {
158     System.out.println("F - Pessoa Fisica | J - Pessoa Juridica ");
159     char tipoPessoa = sc.nextLine().charAt(index: 0);
160     if (tipoPessoa != 'F' && tipoPessoa != 'f' && tipoPessoa != 'J' && tipoPessoa != 'j') {
161         System.out.println("Tipo de pessoa invalido");
162         return;
163     }
164     System.out.println("Digite o id da pessoa: ");
165     int id = sc.nextInt();
166     sc.nextLine();
167
168     if (tipoPessoa == 'F' || tipoPessoa == 'f') {
169         PessoaFisica pf = repoPF.obter(id: id);
170         System.out.println(pf.exibir());
171     } else {
172         PessoaJuridica pj = repoPJ.obter(id: id);
173         System.out.println(pj.exibir());
174     }
175 }
176
```

f. Seleccionada a opção **obterTodos**, escolher o tipo (Física ou Jurídica) e apresentar os dados de todas as entidades do repositório correto.

```
177 public static void obterTodos() {
178     System.out.println("F - Pessoa Fisica | J - Pessoa Juridica ");
179     char tipoPessoa = sc.nextLine().charAt(index: 0);
180     if (tipoPessoa != 'F' && tipoPessoa != 'f' && tipoPessoa != 'J' && tipoPessoa != 'j') {
181         System.out.println("Tipo de pessoa invalido");
182         return;
183     }
184
185     if (tipoPessoa == 'F' || tipoPessoa == 'f') {
186         ArrayList<PessoaFisica> pfs = repoPF.obterTodos();
187         for (PessoaFisica pf : pfs) {
188             System.out.println(pf.exibir());
189             System.out.println("-----");
190         }
191     } else {
192         ArrayList<PessoaJuridica> pjs = repoPJ.obterTodos();
193         for (PessoaJuridica pj : pjs) {
194             System.out.println(pj.exibir());
195             System.out.println("-----");
196         }
197     }
198 }
199
```

g. Selecionada a opção **salvar**, solicitar o **prefixo** dos arquivos e persistir os dados nos arquivos **[prefixo].fisica.bin** e **[prefixo].juridica.bin**.

```
200 public static void salvar() {
201     System.out.println("Digite o nome do arquivo a ser salvo: ");
202     String prefixo = sc.nextLine();
203
204     try {
205         repoPF.persistir(prefixo + "fisica.bin");
206     } catch (IOException ex) {
207         System.out.println("Não foi possível salvar o repositório de pessoa física.");
208         return;
209     }
210
211     try {
212         repoPJ.persistir(prefixo + "juridica.bin");
213     } catch (IOException ex) {
214         System.out.println("Não foi possível salvar o repositório de pessoa jurídica.");
215         return;
216     }
217
218     System.out.println("Arquivo salvo com sucesso! ");
219 }
220
```

h. Selecionada a opção **recuperar**, solicitar o **prefixo** dos arquivos e obter os dados a partir dos arquivos **[prefixo].fisica.bin** e **[prefixo].juridica.bin**.

```
221 public static void recuperar() {
222     System.out.println("Digite o nome do arquivo a ser carregado: ");
223     String prefixo = sc.nextLine();
224     try {
225         repoPF.recuperar(prefixo + "fisica.bin");
226     } catch (IOException | ClassNotFoundException ex) {
227         System.out.println("Não foi possível ler o arquivo de pessoa física informado. ");
228         return;
229     }
230
231     try {
232         repoPJ.recuperar(prefixo + "juridica.bin");
233     } catch (IOException | ClassNotFoundException ex) {
234         System.out.println("Não foi possível ler o arquivo de pessoa jurídica informado. ");
235         return;
236     }
237
238     System.out.println("Arquivo recuperado com sucesso! ");
239 }
240
241
```

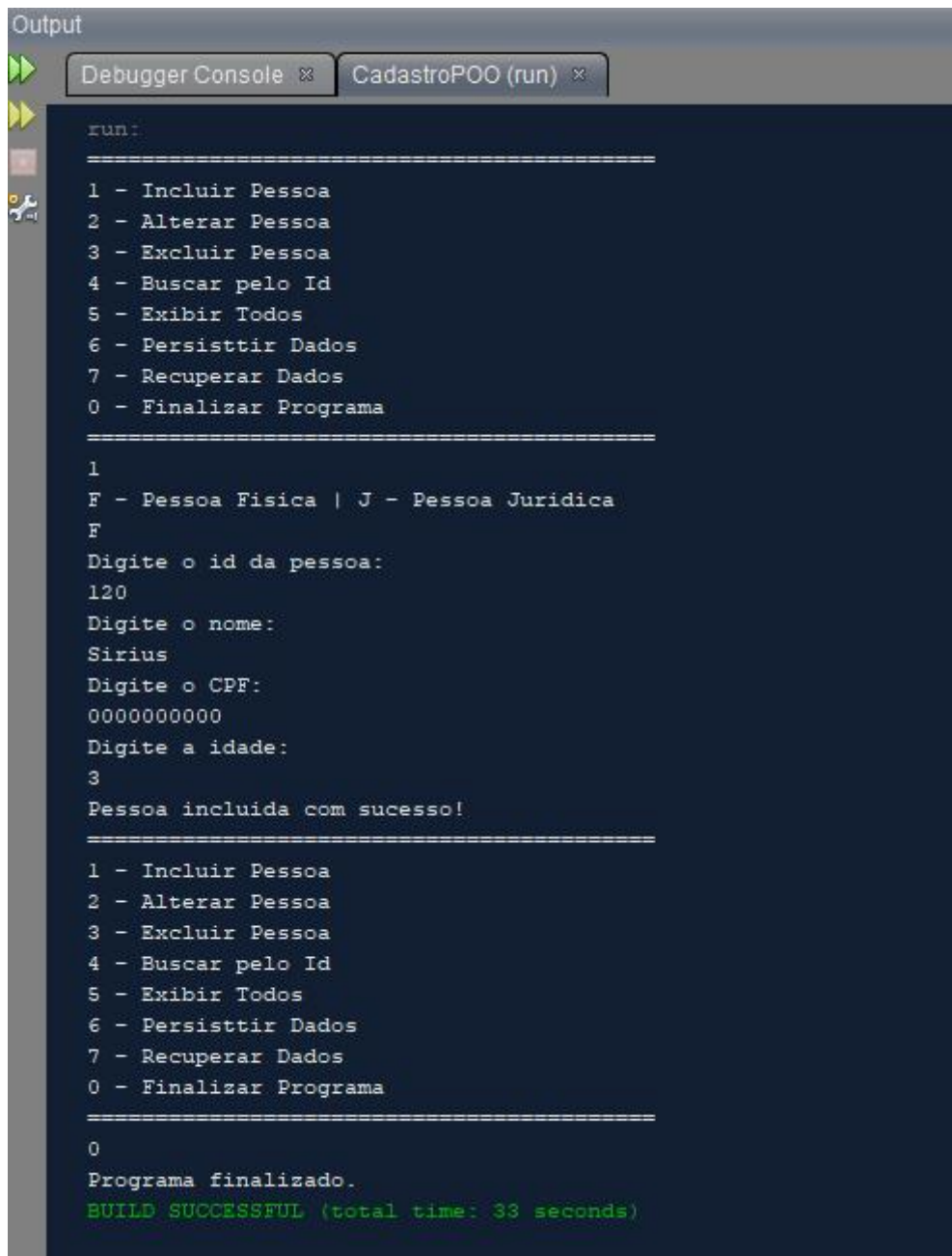
i. Nas opções **salvar** e **recuperar** devem ser tratadas as exceções.

Foram tratadas conforme as imagens dos itens g e h.

j. Selecionada a opção sair, finalizar a execução do sistema.

```
48 case 0 -> System.out.println("Programa finalizado. ");
```

2. Ajustar as características para obter uma exceção como a seguinte:



```
Output
Debugger Console x CadastroPOO (run) x

run:
=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo Id
5 - Exibir Todos
6 - Persisttir Dados
7 - Recuperar Dados
0 - Finalizar Programa
=====

1
F - Pessoa Fisica | J - Pessoa Juridica
F
Digite o id da pessoa:
120
Digite o nome:
Sirius
Digite o CPF:
0000000000
Digite a idade:
3
Pessoa incluída com sucesso!
=====

1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo Id
5 - Exibir Todos
6 - Persisttir Dados
7 - Recuperar Dados
0 - Finalizar Programa
=====

0
Programa finalizado.
BUILD SUCCESSFUL (total time: 33 seconds)
```

8. Relatório discente de acompanhamento - Análise e conclusão (parte 2)

a. O que são elementos estáticos e qual o motivo para o método main adotar esse modificador?

Elementos estáticos em Java, são variáveis ou métodos de uma classe que são declaradas com a palavra-chave “static”, e pertencem a classe em si, não a instâncias individuais da classe. Um objeto estático possui apenas uma instância e uma cópia na memória, mesmo que exista mais uma de uma instância da classe ao qual ele pertence.

Os elementos estáticos podem ser acessados diretamente pela classe, sem precisar criar uma instância da classe. Mas também podem ser acessados através de uma instância da classe. Além disso, são inicializados apenas uma vez, quando a classe é carregada pela primeira vez e são compartilhados por todas as instâncias da classe.

As variáveis estáticas são adequadas quando é útil armazenar um valor comum a todas as instâncias da classe, pois todas elas terão acesso a mesma cópia da variável estática. E os métodos estáticos, são utilizados para efetuar ações que não dependem do estado de uma instância específica da classe.

O motivo de adotar o “static” como padrão no main do Java, é em razão da execução sem instância, ou seja, quando inicia a execução do sistema, o main será chamado diretamente pela classe e não há a necessidade de instanciar a classe. É uma convenção do Java que se utilize o “static” no main, e facilita a execução do programa.

b. Para que serve a classe Scanner?

O Scanner é uma classe do pacote “java.util”, que tem como propósito interagir com o usuário recebendo dados através do teclado, além disso, ela pode ler dados de arquivos de outras fontes de entrada.

Dentre as funcionalidades fornecida pela classe Scanner, há o métodos para ler tipos inteiros: “nextInt()”; números com casas decimais: “nextDouble()”; strings: “nextLine()”; caracteres: “nextChar()”, dentre outros.

Uma importante informação é que essa classe deve ser fechada ao final do uso, utilizando o “close()”.

A classe Scanner tem um importante papel de receber e interpretar os dados para que possam ser utilizados na execução do programa. Seus métodos são práticos e eficientes para a finalidade que se propõe.

c. Como o uso de classes de repositório impactou na organização do código?

O uso de classes de repositório são de grande valia em quesitos de organização, uma vez que separam a lógica de acesso, as regras de negocio das outras partes do código. Visando principalmente a organização, mas também refletindo em outros aspectos, como por exemplo, sendo útil na visualização da responsabilidade de cada classe, de simples testabilidade, manutenção, reuso do código, dentre outras.

Na prática, fica mais simples encontrar os objetos para alterar suas funcionalidade, abstrair métodos