

Desenvolvimento Full Stack

Nível 1: Iniciando o Caminho Pelo Java

2023.1

Mundo 3 Período 2024.1

Breno Félix de Souza

1. Título da Prática;

Missão Prática | Nível 1 | Mundo 3

1º Procedimento | Criação das Entidades e Sistema de Persistência

2. Objetivo da Prática;

Utilizar herança e polimorfismo na definição de entidades.

Utilizar persistência de objetos em arquivos binários.

Implementar uma interface cadastral em modo texto.

Utilizar o controle de exceções da plataforma Java.

3. Todos os códigos solicitados neste roteiro de aula;

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.

```
public class Pessoa implements Serializable {
    private int id;
    private String nome;

    //construtores
    public Pessoa(int id, String nome) {
        this.id = id;
        this.nome = nome;
    }

    //Getters
    public int getId() {
        return id;
    }

    public String getNome() {
        return nome;
    }

    //setters
    public void setId(int id) {
        this.id = id;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    //Metodo Exibir
    public void exibir(){
        System.out.println("ID: " + getId());
        System.out.println("Nome: " + getNome());
    };
}
```

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

```
public class PessoaFisica extends Pessoa
implements Serializable{

    private String cpf;

    private int idade;

    //Contrutores

    public PessoaFisica(int id, String nome, String
    cpf, int idade) {

        super(id, nome);

        this.cpf = cpf;

        this.idade = idade;

    }

    //getters

    public String getCpf() {

        return cpf;

    }

    public int getIdade() {

        return idade;

    }

    //setters

    public void setCpf(String cpf) {

        this.cpf = cpf;

    }

    public void setIdade(int idade) {

        this.idade = idade;

    }

    @Override

    public void exibir(){

        System.out.println("ID: " + getId());

        System.out.println("Nome: " + getNome());

        System.out.println("CPF: " + getCpf());

        System.out.println("Idade: " + getIdade());

    }

}
```

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

```
public class PessoaJuridica extends
Pessoa implements Serializable{

    private String cnpj;

    //construtores

    public PessoaJuridica(int id, String
nome, String cnpj) {
        super(id, nome);
        this.cnpj = cnpj;
    }

    public String getCnpj() {
        return cnpj;
    }

    public void setCnpj(String cnpj) {
        this.cnpj = cnpj;
    }

    @Override
    public void exibir(){
        super.exibir();

        System.out.println("CNPJ: " +
```

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.

```
public class PessoaFisicaRepo {  
    private ArrayList<PessoaFisica> pessoas = new ArrayList<>();  
  
    public void inserir(PessoaFisica pessoa) {  
        pessoas.add(pessoa);  
    }  
  
    public void alterar(PessoaFisica pessoa) {  
        // Implementar altera  o  
    }  
  
    public void excluir(int id) {  
        // Implementar exclus  o  
    }  
  
    public PessoaFisica obter(int id) {  
        // Implementar obten  o  
        return null;  
    }  
  
    public ArrayList<PessoaFisica> obterTodos() {  
        return pessoas;  
    }  
}
```

- 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 .

```
public class PessoaJuridicaRepo {  
    private ArrayList<PessoaJuridica> pessoas = new ArrayList<>();  
  
    public void inserir(PessoaJuridica pessoa) {  
        pessoas.add(pessoa);  
    }  
  
    public void alterar(PessoaJuridica pessoa) {  
        // Implementar alteração  
    }  
  
    public void excluir(int id) {  
        // Implementar exclusão  
    }  
  
    public PessoaJuridica obter(int id) {  
        // Implementar obtenção  
        return null;  
    }  
  
    public ArrayList<PessoaJuridica> obterTodos() {  
        return pessoas;  
    }  
}
```

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

PessoaFisicaRepo.java

```
public void persistir(String nomeArquivo) throws IOException {  
    try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(nomeArquivo))) {  
        out.writeObject(pessoas);  
    }  
}  
  
public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {  
    try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(nomeArquivo))) {  
        pessoas = (ArrayList<PessoaFisica>) in.readObject();  
    }  
}
```

PessoaJuridicaRepo.java

```
public void persistir(String nomeArquivo) throws IOException {  
    try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(nomeArquivo))) {  
        out.writeObject(pessoas);  
    }  
}  
  
public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {  
    try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(nomeArquivo))) {  
        pessoas = (ArrayList<PessoaJuridica>) in.readObject();  
    }  
}
```


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

- a. Instanciar um repositório de pessoas físicas (**repo1**).
- b. Adicionar duas pessoas físicas, utilizando o construtor completo.
- c. Invocar o método de persistência em repo1, fornecendo um nome de arquivo fixo, através do código.
- d. Instanciar outro repositório de pessoas físicas (**repo2**).
- e. Invocar o método de recuperação em repo2, fornecendo o mesmo nome de arquivo utilizado anteriormente.
- f. Exibir os dados de todas as pessoas físicas recuperadas.

```
public class CadastroPOO {  
  
    public static void main(String[] args) {  
  
        PessoaFisicaRepo repo1 = new PessoaFisicaRepo();  
  
        PessoaFisica pessoa1 = new PessoaFisica(1, "Ana", "11111111111", 30);  
        PessoaFisica pessoa2 = new PessoaFisica(2, "Carlos", "22222222222", 52);  
  
        repo1.inserir(pessoa1);  
        repo1.inserir(pessoa2);  
  
        try {  
            repo1.persistir("pessoas_fisicas.dat");  
  
            PessoaFisicaRepo repo2 = new PessoaFisicaRepo();  
            repo2.recuperar("pessoas_fisicas.dat");  
  
            for (PessoaFisica pessoa : repo2.obterTodos()) {  
                pessoa.exibir();  
            }  
        }  
    }  
}
```

- g. Instanciar um repositório de pessoas jurídicas (**repo3**).
- h. Adicionar duas pessoas jurídicas, utilizando o construtor completo.
- i. Invocar o método de persistência em repo3, fornecendo um nome de arquivo fixo, através do código.
- j. Instanciar outro repositório de pessoas jurídicas (**repo4**).
- k. Invocar o método de recuperação em repo4, fornecendo o mesmo nome de arquivo utilizado anteriormente.

```
PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();

PessoaJuridica empresa1 = new PessoaJuridica(3, "XPTO Sales", "3333333333333333");
PessoaJuridica empresa2 = new PessoaJuridica(4, "XPTO Solutions", "4444444444444444");

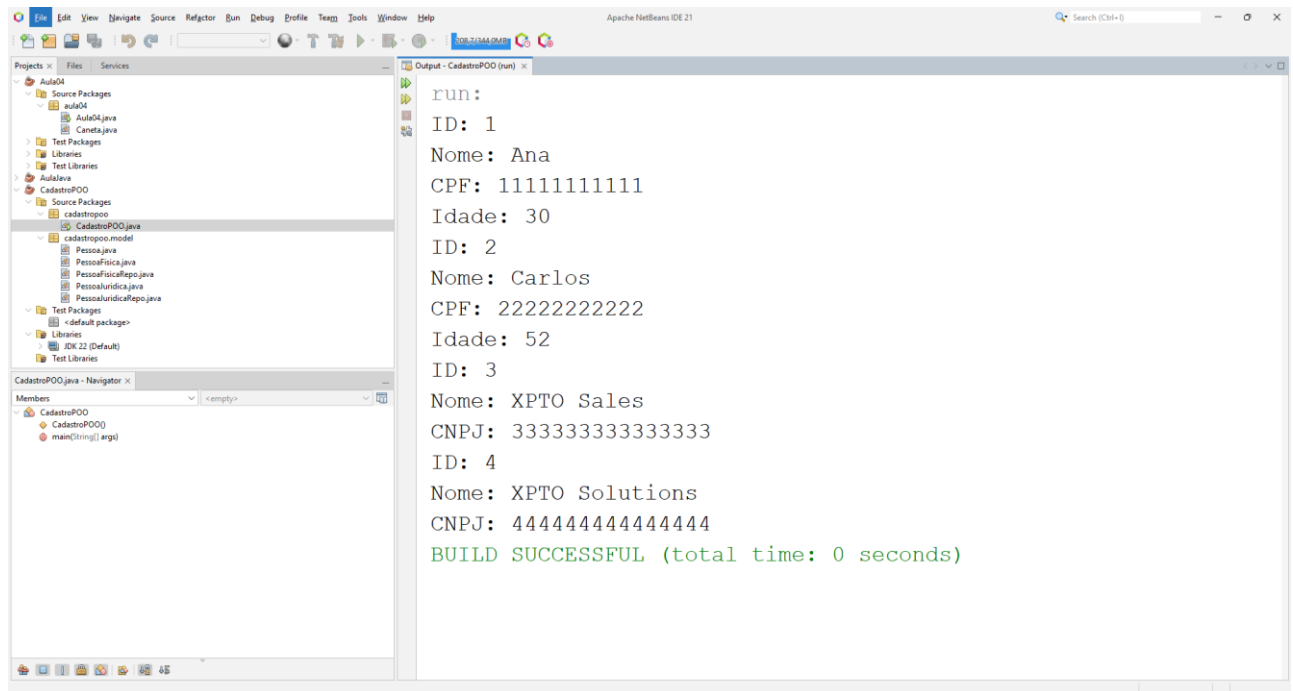
repo3.inserir(empresa1);
repo3.inserir(empresa2);

repo3.persistir("pessoas_juridicas.dat");

PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
repo4.recuperar("pessoas_juridicas.dat");

for (PessoaJuridica empresa : repo4.obterTodos()) {
    empresa.exibir();
}
} catch (IOException | ClassNotFoundException e) {
    e.printStackTrace();
}
```

4. Os resultados da execução dos códigos também devem ser apresentados;



```
run:
ID: 1
Nome: Ana
CPF: 111111111111
Idade: 30
ID: 2
Nome: Carlos
CPF: 222222222222
Idade: 52
ID: 3
Nome: XPTO Sales
CNPJ: 33333333333333
ID: 4
Nome: XPTO Solutions
CNPJ: 4444444444444444
BUILD SUCCESSFUL (total time: 0 seconds)
```

5. Análise e Conclusão:

- Quais as vantagens e desvantagens do uso de herança?
- Por que a interface `Serializable` é necessária ao efetuar persistência em arquivos binários?
- Como o paradigma funcional é utilizado pela API `stream` no Java?
- Quando trabalhamos com Java, qual padrão de desenvolvimento é adotado na persistência de dados em arquivos?

Endereço do projeto no GITHUB

<https://github.com/BrenoSouza2023/Miss-o-Pr-tica-N-vel-1-Mundo-3-Java.git>