



**Campus:** POLO VILA DOS REMÉDIOS – OSASCO – SP

**Curso:** Desenvolvimento Full Stack

**Nome Disciplina:** Nível 1: Iniciando o Caminho Pelo Java

**Matrícula:** 2023 0397 9797

**Semestre Letivo:** 3º semestre

**Nome:** Anderson Barbosa Almeida

**Repositório no GIT:** <https://github.com/andydevbarbosa/andydevbarbosa/tree/main/RPG0014%20-%20Iniciando%20o%20caminho%20pelo%20Java>

**Título Prática:** Missão Prática | Nível 1 | Mundo 3 - 1º Procedimento | Criação das Entidades e Sistema de Persistência

#### **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.
- No final do projeto, o aluno terá implementado um sistema cadastral em Java, utilizando os recursos da programação orientada a objetos e a persistência em arquivos binários.

**Todos os códigos solicitados nesse roteiro:**

#### **Criação das Entidades**

##### **Classe Pessoa:**

```
package model;
import java.io.Serializable;
public class Pessoa implements Serializable {
    private int id;
    private String nome;
    public Pessoa() {}
    public Pessoa(int id, String nome) {
        this.id = id;
        this.nome = nome;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public void exibir() {
        System.out.println("ID: " + id + ", Nome: " + nome);
    }
}
```



### PessoaFisica.java:

```
package model;
import java.io.Serializable;
public class PessoaFisica extends Pessoa {
    private String cpf;
    private int idade;
    public PessoaFisica() {}
    public PessoaFisica(int id, String nome, String cpf, int idade) {
        super(id, nome);
        this.cpf = cpf;
        this.idade = idade;
    }
    public String getCpf() {
        return cpf;
    }
    public void setCpf(String cpf) {
        this.cpf = cpf;
    }
    public int getIdade() {
        return idade;
    }
    public void setIdade(int idade) {
        this.idade = idade;
    }
    @Override
    public void exibir() {
        super.exibir();
        System.out.println("CPF: " + cpf + ", Idade: " + idade);
    }
}
```

### PessoaFisicaRepo.java:

```
package model;
import java.io.*;
import java.util.*;
public class PessoaFisicaRepo {
    private ArrayList<PessoaFisica> pessoas = new ArrayList<>();
    public void inserir(PessoaFisica pessoa) {
        pessoas.add(pessoa);
    }
    public void alterar(PessoaFisica pessoa) {
        for (int i = 0; i < pessoas.size(); i++) {
            if (pessoas.get(i).getId() == pessoa.getId()) {
                pessoas.set(i, pessoa);
                return;
            }
        }
    }
    public void excluir(int id) {
        pessoas.removeIf(p -> p.getId() == id);
    }
    public PessoaFisica obter(int id) {
        for (PessoaFisica p : pessoas) {
            if (p.getId() == id) {
                return p;
            }
        }
        return null;
    }
    public ArrayList<PessoaFisica> obterTodos() {
        return pessoas;
    }
    public void persistir(String nomeArquivo) throws IOException {
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(nomeArquivo))) {
            oos.writeObject(pessoas);
        }
    }
    public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(nomeArquivo))) {

```



```
        pessoas = (ArrayList<PessoaFisica>) ois.readObject();
    }
}
```

### **PessoaJuridica.java:**

```
package model;
import java.io.Serializable;
public class PessoaJuridica extends Pessoa {
    private String cnpj;
    public PessoaJuridica() {}
    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: " + cnpj);
    }
}
```

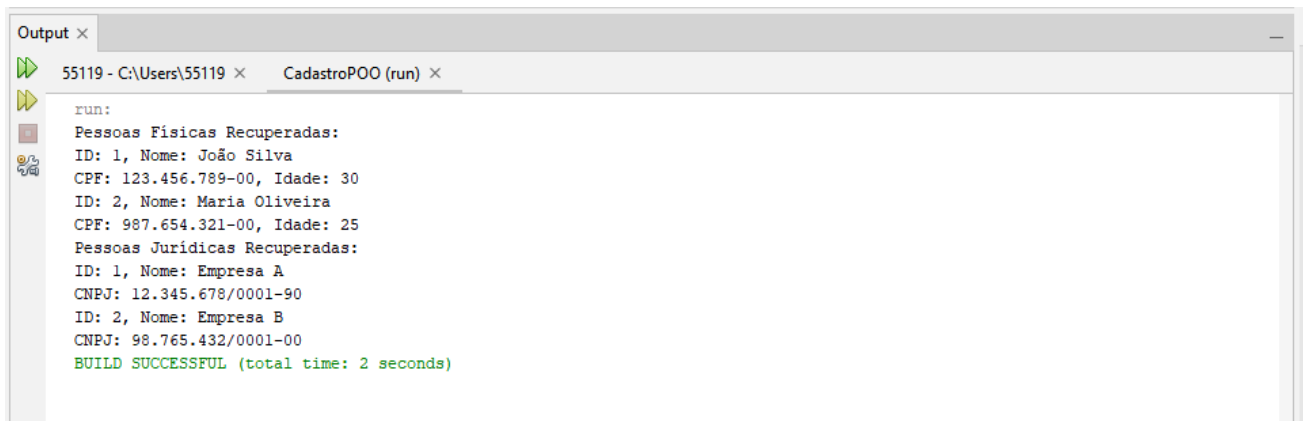
### **PessoaJuridicaRepo.java:**

```
package model;
import java.io.*;
import java.util.ArrayList;
public class PessoaJuridicaRepo {
    private ArrayList<PessoaJuridica> pessoas = new ArrayList<>();
    public void inserir(PessoaJuridica pessoa) {
        pessoas.add(pessoa);
    }
    public void alterar(PessoaJuridica pessoa) {
        for (int i = 0; i < pessoas.size(); i++) {
            if (pessoas.get(i).getId() == pessoa.getId()) {
                pessoas.set(i, pessoa);
                return;
            }
        }
    }
    public void excluir(int id) {
        pessoas.removeIf(p -> p.getId() == id);
    }
    public PessoaJuridica obter(int id) {
        for (PessoaJuridica p : pessoas) {
            if (p.getId() == id) {
                return p;
            }
        }
        return null;
    }
    public ArrayList<PessoaJuridica> obterTodos() {
        return pessoas;
    }
    public void persistir(String nomeArquivo) throws IOException {
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(nomeArquivo))) {
            oos.writeObject(pessoas);
        }
    }
    public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(nomeArquivo))) {
            pessoas = (ArrayList<PessoaJuridica>) ois.readObject();
        }
    }
}
```

### CadastroPOO.java:

```
package model;
public class CadastroPOO {
    public static void main(String[] args) {
        try {
            // Teste para PessoaFisica
            PessoaFisicaRepo repo1 = new PessoaFisicaRepo();
            repo1.inserir(new PessoaFisica(1, "João Silva", "123.456.789-00", 30));
            repo1.inserir(new PessoaFisica(2, "Maria Oliveira", "987.654.321-00", 25));
            repo1.persistir("pessoas_fisicas.dat");
            PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
            repo2.recuperar("pessoas_fisicas.dat");
            System.out.println("Pessoas Físicas Recuperadas:");
            for (PessoaFisica pf : repo2.obterTodos()) {
                pf.exibir();
            }
            // Teste para PessoaJuridica
            PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
            repo3.inserir(new PessoaJuridica(1, "Empresa A", "12.345.678/0001-90"));
            repo3.inserir(new PessoaJuridica(2, "Empresa B", "98.765.432/0001-00"));
            repo3.persistir("pessoas_juridicas.dat");
            PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
            repo4.recuperar("pessoas_juridicas.dat");
            System.out.println("Pessoas Jurídicas Recuperadas:");
            for (PessoaJuridica pj : repo4.obterTodos()) {
                pj.exibir();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

### Resultado da Execução dos Códigos:



```
Output ×
55119 - C:\Users\55119 × CadastroPOO (run) ×

run:
Pessoas Físicas Recuperadas:
ID: 1, Nome: João Silva
CPF: 123.456.789-00, Idade: 30
ID: 2, Nome: Maria Oliveira
CPF: 987.654.321-00, Idade: 25
Pessoas Jurídicas Recuperadas:
ID: 1, Nome: Empresa A
CNPJ: 12.345.678/0001-90
ID: 2, Nome: Empresa B
CNPJ: 98.765.432/0001-00
BUILD SUCCESSFUL (total time: 2 seconds)
```

**Análise e Conclusão:**

Vantagens e Desvantagens do Uso de Herança

**Vantagens:**

1. Reuso de Código: A herança permite que classes filhas reutilizem métodos e atributos de classes pai, reduzindo a duplicação de código.
2. Organização e Estrutura: Facilita a organização do código em uma hierarquia lógica, tornando-o mais fácil de entender e manter.
3. Polimorfismo: Permite que objetos de diferentes classes sejam tratados de forma uniforme, aumentando a flexibilidade do código.

**Desvantagens:**

1. Acoplamento: A herança pode criar um acoplamento forte entre classes, tornando mudanças em uma classe pai impactantes em suas classes filhas.
2. Complexidade: Hierarquias de herança muito profundas podem dificultar a compreensão do código e a localização de bugs.
3. Fragilidade: Mudanças na classe pai podem afetar todas as subclasses, levando a erros inesperados.

**Importância da Interface Serializable**

A interface Serializable é necessária para a persistência em arquivos binários porque:

1. Marca os Objetos: Ao implementar Serializable, você indica que os objetos dessa classe podem ser serializados, ou seja, convertidos em um formato que pode ser facilmente armazenado em um arquivo ou transmitido através de uma rede.
2. Armazenamento de Estado: A serialização permite que você armazene o estado de um objeto, incluindo seus atributos, para que possam ser recuperados posteriormente.
3. Compatibilidade: Permite a troca de dados entre diferentes sistemas e versões de classes, contanto que a estrutura dos dados permaneça compatível.

**Paradigma Funcional e API Stream no Java**

O paradigma funcional é utilizado na API Stream do Java da seguinte forma:

1. Operações Funcionais: A API Stream permite que você processe coleções de dados de maneira declarativa usando funções, como ``map``, ``filter``, e ``reduce``. Isso promove um estilo de programação mais expressivo e conciso.
2. Imutabilidade: Streams não alteram a fonte de dados original, o que é um princípio do paradigma funcional. Ao invés disso, eles produzem novos fluxos de dados.
3. Lazy Evaluation: A avaliação de operações em streams é feita de forma preguiçosa, ou seja, as operações são realizadas apenas quando necessário, o que pode melhorar a performance.

**Padrão de Desenvolvimento para Persistência de Dados em Java**

Quando se trabalha com Java, um padrão comum para a persistência de dados em arquivos é o Data Access Object (DAO). Esse padrão:

1. Isola a Lógica de Acesso a Dados: O DAO separa a lógica de negócios da lógica de acesso a dados, facilitando a manutenção e os testes dos códigos.
2. Abstrai Detalhes de Implementação: Os desenvolvedores podem trabalhar com objetos de domínio sem se preocupar com os detalhes de como os dados são armazenados e recuperados.
3. Facilita a Mudança de Persistência: Se, por exemplo, você decidir mudar a forma de persistência (de arquivos para um banco de dados), isso pode ser feito na implementação do DAO sem impactar a lógica de negócios.



**Campus:** POLO VILA DOS REMÉDIOS – OSASCO – SP

**Curso:** Desenvolvimento Full Stack

**Nome Disciplina:** Nível 1: Iniciando o Caminho Pelo Java

**Matrícula:** 2023 0397 9797

**Semestre Letivo:** 3º semestre

**Nome:** Anderson Barbosa Almeida

**Repositório no GIT:** <https://github.com/andydevbarbosa/andydevbarbosa/tree/main/RPG0014%20-%20Iniciando%20o%20caminho%20pelo%20Java>

**Título Prática:** Missão Prática | Nível 1 | Mundo 3 - 2º Procedimento | Criação do Cadastro em Modo Texto

#### **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.
- No final do projeto, o aluno terá implementado um sistema cadastral em Java, utilizando os recursos da programação orientada a objetos e a persistência em arquivos binários.

#### **Todos os códigos solicitados nesse roteiro:**

```
package model;
import java.util.Scanner;
public class CadastroPOO {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        PessoaFisicaRepo repoFisica = new PessoaFisicaRepo();
        PessoaJuridicaRepo repoJuridica = new PessoaJuridicaRepo();
        int opcao;
        do {
            System.out.println("Selecione uma opção:");
            System.out.println("1 - Incluir");
            System.out.println("2 - Alterar");
            System.out.println("3 - Excluir");
            System.out.println("4 - Exibir pelo ID");
            System.out.println("5 - Exibir todos");
            System.out.println("6 - Salvar dados");
            System.out.println("7 - Recuperar dados");
            System.out.println("0 - Sair");
            opcao = scanner.nextInt();
            scanner.nextLine(); // Consumir a nova linha
            switch (opcao) {
                case 1:
                    incluir(scanner, repoFisica, repoJuridica);
                    break;
                case 2:
                    alterar(scanner, repoFisica, repoJuridica);
                    break;
                case 3:
                    excluir(scanner, repoFisica, repoJuridica);
                    break;
                case 4:
                    exibirPorId(scanner, repoFisica, repoJuridica);
                    break;
                case 5:
                    exibirTodos(scanner, repoFisica, repoJuridica);
                    break;
                case 6:
                    salvarDados(scanner, repoFisica, repoJuridica);
```

```
        break;
    case 7:
        recuperarDados(scanner, repoFisica, repoJuridica);
        break;
    case 0:
        System.out.println("Saindo do sistema...");
        break;
    default:
        System.out.println("Opção inválida. Tente novamente.");
    }
} while (opcao != 0);
scanner.close();
}

private static void incluir(Scanner scanner, PessoaFisicaRepo repoFisica, PessoaJuridicaRepo repoJuridica) {
    System.out.println("Escolha o tipo (1 - Física, 2 - Jurídica):");
    int tipo = scanner.nextInt();
    scanner.nextLine(); // Consumir a nova linha
    if (tipo == 1) {
        // Incluir Pessoa Física
        System.out.print("ID: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // Consumir a nova linha
        System.out.print("Nome: ");
        String nome = scanner.nextLine();
        System.out.print("CPF: ");
        String cpf = scanner.nextLine();
        System.out.print("Idade: ");
        int idade = scanner.nextInt();
        repoFisica.inserir(new PessoaFisica(id, nome, cpf, idade));
    } else if (tipo == 2) {
        // Incluir Pessoa Jurídica
        System.out.print("ID: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // Consumir a nova linha
        System.out.print("Nome: ");
        String nome = scanner.nextLine();
        System.out.print("CNPJ: ");
        String cnpj = scanner.nextLine();
        repoJuridica.inserir(new PessoaJuridica(id, nome, cnpj));
    } else {
        System.out.println("Tipo inválido.");
    }
}

private static void alterar(Scanner scanner, PessoaFisicaRepo repoFisica, PessoaJuridicaRepo repoJuridica) {
    System.out.println("Escolha o tipo (1 - Física, 2 - Jurídica):");
    int tipo = scanner.nextInt();
    scanner.nextLine(); // Consumir a nova linha
    System.out.print("ID para alteração: ");
    int id = scanner.nextInt();
    scanner.nextLine(); // Consumir a nova linha
    if (tipo == 1) {
        PessoaFisica pf = repoFisica.obter(id);
        if (pf != null) {
            System.out.println("Dados atuais: ");
            pf.exibir();
            System.out.print("Novo nome: ");
            String nome = scanner.nextLine();
            System.out.print("Novo CPF: ");
            String cpf = scanner.nextLine();
            System.out.print("Nova idade: ");
            int idade = scanner.nextInt();
            pf.setNome(nome);
            pf.setCpf(cpf);
            pf.setIdade(idade);
        } else {
            System.out.println("Pessoa Física não encontrada.");
        }
    } else if (tipo == 2) {
        PessoaJuridica pj = repoJuridica.obter(id);
        if (pj != null) {
            System.out.println("Dados atuais: ");
            pj.exibir();
            System.out.print("Novo nome: ");
        }
    }
}
```



```
String nome = scanner.nextLine();
System.out.print("Novo CNPJ: ");
String cnpj = scanner.nextLine();
pj.setNome(nome);
pj.setCnpj(cnpj);
} else {
    System.out.println("Pessoa Jurídica não encontrada.");
}
} else {
    System.out.println("Tipo inválido.");
}
}

private static void excluir(Scanner scanner, PessoaFisicaRepo repoFisica, PessoaJuridicaRepo repoJuridica) {
    System.out.println("Escolha o tipo (1 - Física, 2 - Jurídica:");
    int tipo = scanner.nextInt();
    scanner.nextLine(); // Consumir a nova linha
    System.out.print("ID para exclusão: ");
    int id = scanner.nextInt();
    if (tipo == 1) {
        repoFisica.excluir(id);
    } else if (tipo == 2) {
        repoJuridica.excluir(id);
    } else {
        System.out.println("Tipo inválido.");
    }
}

private static void exibirPorId(Scanner scanner, PessoaFisicaRepo repoFisica, PessoaJuridicaRepo repoJuridica) {
    System.out.println("Escolha o tipo (1 - Física, 2 - Jurídica:");
    int tipo = scanner.nextInt();
    scanner.nextLine(); // Consumir a nova linha
    System.out.print("ID: ");
    int id = scanner.nextInt();
    if (tipo == 1) {
        PessoaFisica pf = repoFisica.obter(id);
        if (pf != null) {
            pf.exibir();
        } else {
            System.out.println("Pessoa Física não encontrada.");
        }
    } else if (tipo == 2) {
        PessoaJuridica pj = repoJuridica.obter(id);
        if (pj != null) {
            pj.exibir();
        } else {
            System.out.println("Pessoa Jurídica não encontrada.");
        }
    } else {
        System.out.println("Tipo inválido.");
    }
}

private static void exibirTodos(Scanner scanner, PessoaFisicaRepo repoFisica, PessoaJuridicaRepo repoJuridica) {
    System.out.println("Escolha o tipo (1 - Física, 2 - Jurídica:");
    int tipo = scanner.nextInt();
    scanner.nextLine(); // Consumir a nova linha
    if (tipo == 1) {
        for (PessoaFisica pf : repoFisica.obterTodos()) {
            pf.exibir();
        }
    } else if (tipo == 2) {
        for (PessoaJuridica pj : repoJuridica.obterTodos()) {
            pj.exibir();
        }
    } else {
        System.out.println("Tipo inválido.");
    }
}

private static void salvarDados(Scanner scanner, PessoaFisicaRepo repoFisica, PessoaJuridicaRepo repoJuridica) {
    System.out.print("Prefixo do arquivo: ");
    String prefixo = scanner.nextLine();
    try {
        repoFisica.persistir(prefixo + ".fisica.bin");
        repoJuridica.persistir(prefixo + ".juridica.bin");
        System.out.println("Dados salvos com sucesso.");
    }
```





```
} catch (Exception e) {  
    System.out.println("Erro ao salvar dados: " + e.getMessage());  
}  
}  
  
private static void recuperarDados(Scanner scanner, PessoaFisicaRepo repoFisica, PessoaJuridicaRepo repoJuridica) {  
    System.out.print("Prefixo do arquivo: ");  
    String prefixo = scanner.nextLine();  
    try {  
        repoFisica.recuperar(prefixo + ".fisica.bin");  
        repoJuridica.recuperar(prefixo + ".juridica.bin");  
        System.out.println("Dados recuperados com sucesso.");  
    } catch (Exception e) {  
        System.out.println("Erro ao recuperar dados: " + e.getMessage());  
    }  
}
```

## Análise e Conclusão

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

### Elementos Estáticos:

Elementos estáticos em Java são aqueles que pertencem à classe, e não a instâncias individuais da classe. Isso significa que você pode acessar um membro estático sem criar um objeto da classe. Os elementos estáticos incluem variáveis (campos) e métodos.

### Motivo para o método `main` ser estático:

O método `main` é o ponto de entrada de uma aplicação Java e é declarado como `static` para que a JVM possa invocá-lo sem a necessidade de criar uma instância da classe. Isso é essencial para o funcionamento do programa, pois a JVM não possui informações sobre a instância de qualquer classe até que o programa comece a ser executado.

### Para que serve a classe `Scanner`?

A classe `Scanner` em Java é utilizada para ler dados de várias fontes, como a entrada padrão (teclado), arquivos ou strings. Ela fornece métodos convenientes para ler diferentes tipos de dados (como `int`, `String`, `double`, etc.) de forma interativa.

### Principais usos da classe `Scanner`:

1. Leitura de Dados do Usuário: Permite que o programa receba entrada do usuário de forma dinâmica.
2. Facilidade de Uso: A classe oferece métodos simples para converter entradas em tipos primitivos.
3. Flexibilidade: Pode ler dados de várias fontes, não apenas do teclado.

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

### O uso de classes de repositório traz várias vantagens para a organização do código:

1. Separação de Preocupações: As classes de repositório isolam a lógica de acesso a dados da lógica de negócios. Isso facilita a manutenção e a compreensão do código.
2. Reuso de Código: As operações comuns de CRUD (Criar, Ler, Atualizar, Deletar) são encapsuladas em métodos de repositório, permitindo que sejam reutilizadas em diferentes partes da aplicação.
3. Facilidade de Testes: Com a lógica de acesso a dados separada, é mais fácil criar testes unitários para a lógica de negócios, pois você pode simular o comportamento dos repositórios.
4. Flexibilidade para Mudanças: Se houver necessidade de mudar a forma de persistência (por exemplo, de arquivos para um banco de dados), essa mudança pode ser feita apenas na implementação do repositório, sem afetar a lógica de negócios.
5. Melhoria da Legibilidade: O código se torna mais legível e organizado, com cada classe e método tendo uma responsabilidade clara.