Estácio	**Universidade Estácio de Sá** Desenvolvimento Full Stack Nível 1 – Iniciando o Caminho Pelo Java Turma 2022.3 – 3º Semestre
Nome:	Luan Augusto Vieira Bandeira
Professor:	Rodrigo Augusto
Repositório:	https://github.com/luanguto/Nivel1-Mundo3

Implementação de um cadastro de clientes em modo texto, com persistência em arquivos, baseado na tecnologia Java.

Objetivos da prática

1. Utilizar herança e polimorfismo na definição de entidades.
2. Utilizar persistência de objetos em arquivos binários.
3. Implementar uma interface cadastral em modo texto.
4. Utilizar o controle de exceções da plataforma Java.
7. em arquivos binários.

1º Procedimento – Criação das entidades e sistemas de persistência.

Códigos solicitados:

Pessoa.class

```
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.class

```
package model;

import java.io.Serializable;

public class PessoaFisica extends Pessoa implements Serializable {
    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;
    }

    @Override
    public void exibir() {
        super.exibir();
        System.out.println("CPF: " + cpf + ", 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;
    }


}
```

PessoaFisicaRepo.class

```
package model;

import java.io.*;
import java.util.ArrayList;
import java.util.Optional;

public class PessoaFisicaRepo {
    private ArrayList<PessoaFisica> pessoasFisicas = new
ArrayList<>();

    public void inserir(PessoaFisica pessoaFisica) {
        pessoasFisicas.add(pessoaFisica);
    }

    public void alterar(PessoaFisica pessoaFisica) {
        int index = -1;
        for (int i = 0; i < pessoasFisicas.size(); i++) {
            if (pessoasFisicas.get(i).getId() == pessoaFisica.getId())
{
                index = i;
                break;
            }
        }
        if (index != -1) {
            pessoasFisicas.set(index, pessoaFisica);
        }
    }

    public void excluir(int id) {
        for (int i = 0; i < pessoasFisicas.size(); i++) {
            if (pessoasFisicas.get(i).getId() == id) {
                pessoasFisicas.remove(i);
                return;
            }
        }
    }

    public PessoaFisica obter(int id) {
        for (PessoaFisica pessoaFisica : pessoasFisicas) {
            if (pessoaFisica.getId() == id) {
                return pessoaFisica;
            }
        }
        return null;
    }

    public ArrayList<PessoaFisica> obterTodos() {
        return pessoasFisicas;
    }

    public void persistir(String fileName) throws IOException {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(fileName))) {
            oos.writeObject(pessoasFisicas);
        }
    }

    public void recuperar(String fileName) throws IOException,
ClassNotFoundException {
```

```
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(fileName))) {
            pessoasFisicas = (ArrayList<PessoaFisica>)
ois.readObject();
        }
    }
}
```

PessoaJuridica.class

```
package model;

import java.io.Serializable;

public class PessoaJuridica extends Pessoa implements Serializable{
    private static final long serialVersionUID = 1L;
    private String cnpj;

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

    @Override
    public void exibir() {
        super.exibir();
        System.out.println("CNPJ: " + cnpj);
    }

    public String getCnpj() {
        return cnpj;
    }

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

}
```

PessoaJuridicaRepo.class

```
package model;

import java.io.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

public class PessoaJuridicaRepo {
    private final List<PessoaJuridica> pessoaJuridicas = new
ArrayList<>();

    public void inserir(PessoaJuridica pessoaJuridica) {
        pessoaJuridicas.add(pessoaJuridica);
    }

    public void alterar(PessoaJuridica pessoaJuridica) {
        pessoaJuridicas.replaceAll(p -> p.getId() ==
pessoaJuridica.getId() ? pessoaJuridica : p);
    }
```

```
    public void excluir(int id) {
        pessoaJuridicas.removeIf(p -> p.getId() == id);
    }

    public PessoaJuridica obter(int id) {
        return pessoaJuridicas.stream()
                .filter(p -> p.getId() == id)
                .findFirst()
                .orElse(null);
    }

    public List<PessoaJuridica> obterTodos() {
        return new ArrayList<>(pessoaJuridicas);
    }

    public void persistir(String nomeArquivo) throws IOException {
        try (ObjectOutputStream outputStream = new
ObjectOutputStream(new FileOutputStream(nomeArquivo))) {
            outputStream.writeObject(pessoaJuridicas);
        }
    }

    public void recuperar(String nomeArquivo) throws IOException,
ClassNotFoundException {
        try (ObjectInputStream inputStream = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
            pessoaJuridicas.clear();
            List<PessoaJuridica> listaRecuperada =
(List<PessoaJuridica>) inputStream.readObject();
            pessoaJuridicas.addAll(listaRecuperada);
        }
    }
}
```

Main

```
package main;

import java.io.IOException;
import model.PessoaFisica;
import model.PessoaFisicaRepo;
import model.PessoaJuridica;
import model.PessoaJuridicaRepo;


public class Main {
    public static void main(String[] args) {

        PessoaFisicaRepo repo1 = new PessoaFisicaRepo();


        repo1.inserir(new PessoaFisica(1, "João", "123.456.789-01",
25));
        repo1.inserir(new PessoaFisica(2, "Maria", "987.654.321-09",
30));


        try {
            repo1.persistir("pessoasFisicas.dat");
        } catch (IOException e) {
```

```
            e.printStackTrace();
        }


        PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
        try {
            repo2.recuperar("pessoasFisicas.dat");
            for (PessoaFisica pf : repo2.obterTodos()) {
                pf.exibir();
            }
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }


        PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();


        repo3.inserir(new PessoaJuridica(3, "Empresa A",
"12.345.678/0001-90"));
        repo3.inserir(new PessoaJuridica(4, "Empresa B",
"98.765.432/0001-10"));


        try {
            repo3.persistir("pessoasJuridicas.dat");
        } catch (IOException e) {
            e.printStackTrace();
        }


        PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
        try {
            repo4.recuperar("pessoasJuridicas.dat");
            for (PessoaJuridica pj : repo4.obterTodos()) {
                pj.exibir();
            }
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Resultado:

```
ID: 1, Nome: João
CPF: 123.456.789-01, Idade: 25
ID: 2, Nome: Maria
CPF: 987.654.321-09, Idade: 30
ID: 3, Nome: Empresa A
CNPJ: 12.345.678/0001-90
ID: 4, Nome: Empresa B
CNPJ: 98.765.432/0001-10

Process finished with exit code 0
```

. Análise e Conclusão:

a. Quais as vantagens e desvantagens do uso de herança?

Vantagens:
Reuso de Código: Permite a adoção de campos e métodos de classes existentes.
Organização: Facilita a organização hierárquica das classes.
Polimorfismo: Facilita a implementação de operações gerais com objetos de diferentes subclasses.

Desvantagens:
Acoplamento: Dependência forte entre classes derivadas e sua classe base.
Hierarquia Rígida: Mudanças na superclasse podem afetar todas as subclasses.
Complexidade: Pode resultar em uma estrutura de classes complexa e confusa.

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

Indica que objetos podem ser convertidos em bytes para armazenamento ou transmissão.

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

Permite operações encadeadas e processamento de dados com expressões lambda, promovendo código limpo e expressivo.

d. Quando trabalhamos com Java, qual padrão de desenvolvimento é adotado na persistência de dados em arquivos?

Utiliza fluxos de entrada e saída para leitura e escrita, com o padrão Decorator para adicionar funcionalidades como bufferização. Inclui o padrão DAO para abstrair acessos aos dados.

2º Procedimento – Criação do cadastro em modo texto

Códigos solicitados:

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

```
package model;

import java.io.Serializable;

public class PessoaFisica extends Pessoa implements Serializable {
    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;
    }
```

```
    @Override
    public void exibir() {
        super.exibir();
        System.out.println("CPF: " + cpf + ", 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;
    }

    public String toString() {
        return "ID: " + this.getId() + ", Nome: " + this.getNome() +
", CPF: " + this.getCpf() + ", Idade: " + this.getIdade();
    }

}
```

PessoaFisicaRepo

```
package model;

import java.io.*;
import java.util.ArrayList;
import java.util.List;

public class PessoaFisicaRepo {
    private ArrayList<PessoaFisica> pessoasFisicas = new
ArrayList<>();

    public void inserir(PessoaFisica pessoaFisica) {
        pessoasFisicas.add(pessoaFisica);
    }

    public void alterar(PessoaFisica pessoaFisica) {
        int index = -1;
        for (int i = 0; i < pessoasFisicas.size(); i++) {
            if (pessoasFisicas.get(i).getId() == pessoaFisica.getId())
{
                index = i;
                break;
            }
        }
        if (index != -1) {
            pessoasFisicas.set(index, pessoaFisica);
        }
    }

    public boolean excluir(int id) {
        for (int i = 0; i < pessoasFisicas.size(); i++) {
            if (pessoasFisicas.get(i).getId() == id) {
```

```
                pessoasFisicas.remove(i);
                return false;
            }
        }
        return false;
    }

    public PessoaFisica obter(int id) {
        for (PessoaFisica pessoaFisica : pessoasFisicas) {
            if (pessoaFisica.getId() == id) {
                return pessoaFisica;
            }
        }
        return null;
    }

    public ArrayList<PessoaFisica> obterTodos() {
        return pessoasFisicas;
    }

    public void persistir(String fileName) throws IOException {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(fileName))) {
            oos.writeObject(pessoasFisicas);
        }
    }

    public void recuperar(String fileName) throws IOException,
ClassNotFoundException {
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(fileName))) {
            pessoasFisicas = (ArrayList<PessoaFisica>)
ois.readObject();
        }
    }

    public PessoaFisica obterPorId(int id) {
        for (PessoaFisica pessoaFisica : pessoasFisicas) {
            if (pessoaFisica.getId() == id) {
                return pessoaFisica;
            }
        }
        return null;
    }

}
```

PessoaJuridica

```
package model;

import java.io.Serializable;

public class PessoaJuridica extends Pessoa implements Serializable{
    private static final long serialVersionUID = 1L;
    private String cnpj;

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

```
    @Override
    public void exibir() {
        super.exibir();
        System.out.println("CNPJ: " + cnpj);
    }

    public String getCnpj() {
        return cnpj;
    }

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

    public String toString() {
        return "ID: " + this.getId() + ", Nome: " + this.getNome() +
", CNPJ: " + this.getCnpj();
    }
}
```

PessoaJuridicaRepo

```
package model;

import java.io.*;
import java.util.ArrayList;
import java.util.List;


public class PessoaJuridicaRepo {
    private final List<PessoaJuridica> pessoaJuridicas = new
ArrayList<>();

    public void inserir(PessoaJuridica pessoaJuridica) {
        pessoaJuridicas.add(pessoaJuridica);
    }

    public void alterar(PessoaJuridica pessoaJuridica) {
        pessoaJuridicas.replaceAll(p -> p.getId() ==
pessoaJuridica.getId() ? pessoaJuridica : p);
    }

    public boolean excluir(int id) {
        pessoaJuridicas.removeIf(p -> p.getId() == id);
        return false;
    }

    public PessoaJuridica obter(int id) {
        return pessoaJuridicas.stream()
                .filter(p -> p.getId() == id)
                .findFirst()
                .orElse(null);
    }

    public List<PessoaJuridica> obterTodos() {
        return new ArrayList<>(pessoaJuridicas);
    }

    public void persistir(String nomeArquivo) throws IOException {
        try (ObjectOutputStream outputStream = new
ObjectOutputStream(new FileOutputStream(nomeArquivo))) {
            outputStream.writeObject(pessoaJuridicas);
```

```
        }
    }

    public void recuperar(String nomeArquivo) throws IOException,
ClassNotFoundException {
        try (ObjectInputStream inputStream = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
            pessoaJuridicas.clear();
            List<PessoaJuridica> listaRecuperada =
(List<PessoaJuridica>) inputStream.readObject();
            pessoaJuridicas.addAll(listaRecuperada);
        }
    }

    public PessoaJuridica obterPorId(int id) {
        for (PessoaJuridica pessoaJuridica : pessoaJuridicas) {
            if (pessoaJuridica.getId() == id) {
                return pessoaJuridica;
            }
        }
        return null;
    }


}
```

Main

```
import model.PessoaFisica;
import model.PessoaFisicaRepo;
import model.PessoaJuridica;
import model.PessoaJuridicaRepo;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;

public class Main {

    private static BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));
    private static PessoaFisicaRepo repoFisica = new
PessoaFisicaRepo();
    private static PessoaJuridicaRepo repoJuridica = new
PessoaJuridicaRepo();

    public static void main(String[] args) {
        String opcao;
        do {
            exibirMenu();
            opcao = lerInput();
            tratarOpcaoMenu(opcao);
        } while (!"0".equals(opcao));
        System.out.println("Programa finalizado.");
    }

    private static void exibirMenu() {

System.out.println("============================================");
        System.out.println("1 - Incluir pessoa");
        System.out.println("2 - Alterar Pessoa");
```

```
        System.out.println("3 - Excluir Pessoa");
        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 - Finalizar Programa");

System.out.println("============================================");
        System.out.print("Escolha uma opção: ");
    }

    private static String lerInput() {
        try {
            return reader.readLine();
        } catch (IOException e) {
            System.out.println("Erro ao ler a entrada: " +
e.getMessage());
            return "";
        }
    }

    private static void tratarOpcaoMenu(String opcao) {
        try {
            switch (opcao) {
                case "1":
                    incluirPessoa();
                    break;
                case "2":
                    alterarPessoa();
                    break;
                case "3":
                    excluirPessoa();
                    break;
                case "4":
                    obterPessoa();
                    break;
                case "5":
                    obterTodos();
                    break;
                case "6":
                    salvarDados();
                    break;
                case "7":
                    recuperarDados();
                    break;
                case "0":
                    // Finalizar Programa
                    break;
                default:
                    System.out.println("Opção inválida.");
                    break;
            }
        } catch (Exception e) {
            System.out.println("Ocorreu um erro: " + e.getMessage());
        }
    }

    private static void incluirPessoa() throws IOException {
        System.out.println("Escolha o tipo de pessoa para incluir:");
        System.out.println("1 - Pessoa Física");
        System.out.println("2 - Pessoa Jurídica");
```

```
        System.out.print("Digite sua opção: ");
        String tipo = lerInput();

        if ("1".equals(tipo)) {
            PessoaFisica pessoaFisica = lerDadosPessoaFisica();
            repoFisica.inserir(pessoaFisica);
            System.out.println("Pessoa Física inserida com sucesso.");
        } else if ("2".equals(tipo)) {
            PessoaJuridica pessoaJuridica = lerDadosPessoaJuridica();
            repoJuridica.inserir(pessoaJuridica);
            System.out.println("Pessoa Jurídica inserida com
sucesso.");
        } else {
            System.out.println("Tipo de pessoa inválido. Tente
novamente.");
        }
    }

    private static PessoaFisica lerDadosPessoaFisica() throws
IOException {
        System.out.print("Digite o ID da pessoa física: ");
        int id = Integer.parseInt(lerInput());
        System.out.print("Digite o nome da pessoa física: ");
        String nome = lerInput();
        System.out.print("Digite o CPF da pessoa física: ");
        String cpf = lerInput();
        System.out.print("Digite a idade da pessoa física: ");
        int idade = Integer.parseInt(lerInput());

        return new PessoaFisica(id, nome, cpf, idade);
    }

    private static PessoaJuridica lerDadosPessoaJuridica() throws
IOException {
        System.out.print("Digite o ID da pessoa jurídica: ");
        int id = Integer.parseInt(lerInput());
        System.out.print("Digite o nome da pessoa jurídica: ");
        String nome = lerInput();
        System.out.print("Digite o CNPJ da pessoa jurídica: ");
        String cnpj = lerInput();

        return new PessoaJuridica(id, nome, cnpj);
    }
    private static void alterarPessoa() throws IOException {
        System.out.println("Escolha o tipo de pessoa para alterar:");
        System.out.println("1 - Pessoa Física");
        System.out.println("2 - Pessoa Jurídica");
        System.out.print("Digite sua opção: ");
        String tipo = lerInput();

        System.out.print("Digite o ID da pessoa: ");
        int id = Integer.parseInt(lerInput());

        if ("1".equals(tipo)) {
            PessoaFisica pf = repoFisica.obter(id);
            if (pf != null) {
                System.out.println("Dados atuais: " + pf);
                PessoaFisica novaPf = lerDadosPessoaFisica();
                novaPf.setId(id); // Mantém o mesmo ID
                repoFisica.alterar(novaPf);
                System.out.println("Pessoa Física alterada com
```

```
sucesso.");
            } else {
                System.out.println("Pessoa Física não encontrada.");
            }
        } else if ("2".equals(tipo)) {
            PessoaJuridica pj = repoJuridica.obter(id);
            if (pj != null) {
                System.out.println("Dados atuais: " + pj);
                PessoaJuridica novaPj = lerDadosPessoaJuridica();
                novaPj.setId(id); // Mantém o mesmo ID
                repoJuridica.alterar(novaPj);
                System.out.println("Pessoa Jurídica alterada com
sucesso.");
            } else {
                System.out.println("Pessoa Jurídica não encontrada.");
            }
        } else {
            System.out.println("Tipo de pessoa inválido.");
        }
    }
    private static void excluirPessoa() throws IOException {
        System.out.println("Escolha o tipo de pessoa para excluir:");
        System.out.println("1 - Pessoa Física");
        System.out.println("2 - Pessoa Jurídica");
        System.out.print("Digite sua opção: ");
        String tipo = lerInput();

        System.out.print("Digite o ID da pessoa: ");
        int id = Integer.parseInt(lerInput());

        if ("1".equals(tipo)) {
            boolean excluido = repoFisica.excluir(id);
            if (excluido) {
                System.out.println("Pessoa Física excluída com
sucesso.");
            } else {
                System.out.println("Pessoa Física não encontrada ou
erro na exclusão.");
            }
        } else if ("2".equals(tipo)) {
            boolean excluido = repoJuridica.excluir(id);
            if (excluido) {
                System.out.println("Pessoa Jurídica excluída com
sucesso.");
            } else {
                System.out.println("Pessoa Jurídica não encontrada ou
erro na exclusão.");
            }
        } else {
            System.out.println("Tipo de pessoa inválido.");
        }
    }
    private static void obterPessoa() throws IOException {
        System.out.println("Escolha o tipo de pessoa para obter os
dados:");
        System.out.println("1 - Pessoa Física");
        System.out.println("2 - Pessoa Jurídica");
        System.out.print("Digite sua opção: ");
        String tipo = lerInput();

        System.out.print("Digite o ID da pessoa: ");
```

```
        int id = Integer.parseInt(lerInput());

        if ("1".equals(tipo)) {
            PessoaFisica pessoaFisica = repoFisica.obterPorId(id);
            if (pessoaFisica != null) {
                System.out.println("Dados da Pessoa Física:");
                System.out.println(pessoaFisica); // Assume que o
método toString() está implementado
            } else {
                System.out.println("Pessoa Física não encontrada.");
            }
        } else if ("2".equals(tipo)) {
            PessoaJuridica pessoaJuridica =
repoJuridica.obterPorId(id);
            if (pessoaJuridica != null) {
                System.out.println("Dados da Pessoa Jurídica:");
                System.out.println(pessoaJuridica); // Assume que o
método toString() está implementado
            } else {
                System.out.println("Pessoa Jurídica não encontrada.");
            }
        } else {
            System.out.println("Tipo de pessoa inválido.");
        }
    }
    private static void obterTodos() throws IOException {
        System.out.println("Escolha o tipo de pessoa para listar todos
os registros:");
        System.out.println("1 - Pessoa Física");
        System.out.println("2 - Pessoa Jurídica");
        System.out.print("Digite sua opção: ");
        String tipo = lerInput();

        if ("1".equals(tipo)) {
            List<PessoaFisica> pessoasFisicas =
repoFisica.obterTodos();
            System.out.println("Lista de todas as Pessoas Físicas:");
            for (PessoaFisica pf : pessoasFisicas) {
                System.out.println(pf); // Assume que PessoaFisica tem
um método toString() implementado
            }
        } else if ("2".equals(tipo)) {
            List<PessoaJuridica> pessoasJuridicas =
repoJuridica.obterTodos();
            System.out.println("Lista de todas as Pessoas
Jurídicas:");
            for (PessoaJuridica pj : pessoasJuridicas) {
                System.out.println(pj); // Assume que PessoaJuridica
tem um método toString() implementado
            }
        } else {
            System.out.println("Tipo de pessoa inválido.");
        }
    }
    private static void salvarDados() {
        System.out.print("Digite o prefixo para os arquivos onde os
dados serão salvos: ");
        String prefixo = lerInput();

        try {
            String arquivoPessoaFisica = prefixo + ".fisica.bin";
```

```
            String arquivoPessoaJuridica = prefixo + ".juridica.bin";

            repoFisica.persistir(arquivoPessoaFisica);
            repoJuridica.persistir(arquivoPessoaJuridica);

            System.out.println("Dados salvos com sucesso nos
arquivos:");
            System.out.println(arquivoPessoaFisica);
            System.out.println(arquivoPessoaJuridica);
        } catch (IOException e) {
            System.out.println("Erro ao salvar os dados: " +
e.getMessage());
        }
    }

    private static void recuperarDados() {
        System.out.print("Digite o prefixo para os arquivos de onde os
dados serão recuperados: ");
        String prefixo = lerInput();

        try {
            String arquivoPessoaFisica = prefixo + ".fisica.bin";
            String arquivoPessoaJuridica = prefixo + ".juridica.bin";

            repoFisica.recuperar(arquivoPessoaFisica);
            repoJuridica.recuperar(arquivoPessoaJuridica);

            System.out.println("Dados recuperados com sucesso dos
arquivos:");
            System.out.println(arquivoPessoaFisica);
            System.out.println(arquivoPessoaJuridica);
        } catch (IOException e) {
            System.out.println("Erro ao ler os dados do arquivo: " +
e.getMessage());
        } catch (ClassNotFoundException e) {
            System.out.println("Erro ao recuperar os dados: uma das
classes não foi encontrada.");
        }
    }

}
```

Resultado:

```
==========================================
1 - Incluir pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Exibir pelo ID
5 - Exibir Todos
6 - Salvar Dados
7 - Recuperar Dados
0 - Finalizar Programa
==========================================
Escolha uma opção: 1
Escolha o tipo de pessoa para incluir:
1 - Pessoa Física
2 - Pessoa Jurídica
Digite sua opção: 1
Digite o ID da pessoa física: 10
Digite o nome da pessoa física: Luan
Digite o CPF da pessoa física: 1234567890
Digite a idade da pessoa física: 25
Pessoa Física inserida com sucesso.
```

```
==========================================
Escolha uma opção: 5
Escolha o tipo de pessoa para listar todos os registros:
1 - Pessoa Física
2 - Pessoa Jurídica
Digite sua opção: 1
Lista de todas as Pessoas Físicas:
ID: 10, Nome: Luan, CPF: 1234567890, Idade: 25
==========================================
1 - Incluir pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Exibir pelo ID
5 - Exibir Todos
6 - Salvar Dados
7 - Recuperar Dados
0 - Finalizar Programa
==========================================
Escolha uma opção:
```

Análise e Conclusão:

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

São membros de uma classe que existem independentemente de instâncias daquela classe. O método main é estático porque ele pode ser chamado pela JVM para iniciar o programa sem a necessidade de criar um objeto da classe.

b. Para que serve a classe Scanner?

É usada para ler entrada de dados do usuário, de um arquivo ou de qualquer outra fonte de entrada.

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

As classes de repositório centralizam o gerenciamento de dados, promovendo a organização e a manutenção do código ao separar a lógica de acesso aos dados da lógica de negócios. Alguns benefícios dessas classes são: **Encapsulamento, manutenção, reutilização e flexibilidade.**