



Universidade Estácio de Sá

Polo Bangu - Rio de Janeiro

-
- **Curso:** Desenvolvimento Full-Stack
 - **Disciplina:** Iniciando o Caminho pelo Java
 - **Semestre:** 3
 - **Turma:** 2024.2
 - **Aluna:** Clara Martins Azevedo
-

1. Missão Prática - Nível 1

2. Objetivos da Missão 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.

3. Códigos do Projeto:

- Pessoa.java:

```
package model;
import java.io.Serializable;

/**
 *
 * @author okidata
 */
public class Pessoa implements Serializable {

    private int id;
    private String nome;

    public Pessoa() {
        id = 0;
        nome = "";
    }

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

    public void exibir() {
        System.out.println("id: " + getId());
        System.out.println("nome: " + getNome());
    }

    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;
    }

}
```

• PessoaFisica.java:

```
package model;
import java.io.Serializable;

/**
 *
 * @author okidata
 */
public class PessoaFisica extends Pessoa implements Serializable{

    private String cpf;
    private int idade;

    @Override
    public void exibir() {
        super.exibir();
        System.out.println("cpf: " + getCpf());
        System.out.println("idade: " + getIdade());
    }

    public PessoaFisica() {
        cpf = "";
        idade = 0;
    }

    public PessoaFisica(int id, String nome, String cpf, int idade) {
        setId(id);
        setNome(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;
    }
}
```

• PessoaFisicaRepo.java:

```
package model;
import java.util.ArrayList;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.io.FileOutputStream;
import java.io.FileInputStream;

/**
 *
 * @author okidata
 */
public class PessoaFisicaRepo {

    private ArrayList<PessoaFisica> pessoasFisicas;

    public PessoaFisicaRepo() {
        this.pessoasFisicas = new ArrayList<>();
    }

    public boolean inserir(PessoaFisica pessoaF) {
        return pessoasFisicas.contains(pessoaF) ? false : pessoasFisicas.add(pessoaF);
    }

    public boolean alterar(PessoaFisica pessoaF) {
        int position = pessoasFisicas.indexOf(pessoaF);
        if (position != -1) {
            pessoasFisicas.set(position, pessoaF);
            return true;
        }
        return false;
    }

    public boolean excluir(int id) {
        return pessoasFisicas.remove(obter(id));
    }

    public PessoaFisica obter(int id) {
        return pessoasFisicas.stream()
            .filter(pessoaF -> pessoaF.getId() == id)
            .findFirst()
            .orElse(null);
    }

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

    public void persistir(String arquivo) throws IOException {
        try (ObjectOutputStream oss =
            new ObjectOutputStream(new FileOutputStream(arquivo))) {
            oss.writeObject(pessoasFisicas);
            System.out.println(arquivo);
        }
    }

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

- PessoaJuridica.java:

```
package model;
import java.io.Serializable;

/**
 *
 * @author okidata
 */
public class PessoaJuridica extends Pessoa implements Serializable {

    private String cnpj;

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

    public PessoaJuridica() {
        cnpj = "";
    }

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

    public PessoaJuridica(String cnpj) {
        this.cnpj = cnpj;
    }

    public String getCnpj() {
        return cnpj;
    }

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

}
```

• PessoaJuridicaRepo.java:

```
package model;

import java.util.ArrayList;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.io.FileOutputStream;
import java.io.FileInputStream;

/**
 *
 * @author okidata
 */
public class PessoaJuridicaRepo {

    private ArrayList<PessoaJuridica> pessoasJuridicas;

    public PessoaJuridicaRepo() {
        this.pessoasJuridicas = new ArrayList<>();
    }

    public boolean inserir(PessoaJuridica pessoaJ) {
        return pessoasJuridicas.contains(pessoaJ) ? false : pessoasJuridicas.add(pessoaJ);
    }

    public boolean alterar(PessoaJuridica pessoaJ) {
        int position = pessoasJuridicas.indexOf(pessoaJ);
        if (position != -1) {
            pessoasJuridicas.set(position, pessoaJ);
            return true;
        }
        return false;
    }

    public boolean excluir(int id) {
        return pessoasJuridicas.remove(obter(id));
    }

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

    public ArrayList<PessoaJuridica> obterTodos() {
        return pessoasJuridicas;
    }

    public void persistir(String arquivo) throws IOException {
        try (ObjectOutputStream outputStream =
            new ObjectOutputStream(new FileOutputStream(arquivo))) {
            outputStream.writeObject(pessoasJuridicas);
            System.out.println(arquivo);
        }
    }

    public void recuperar(String arquivo) throws IOException, ClassNotFoundException {
        try (ObjectInputStream inputStream =
            new ObjectInputStream(new FileInputStream(arquivo))) {
            pessoasJuridicas = (ArrayList<PessoaJuridica>) inputStream.readObject();
            System.out.println(arquivo);
        }
    }
}
```

• CadastroPOO.java:

```
package cadastropoo;

import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import model.PessoaFisica;
import model.PessoaFisicaRepo;
import model.PessoaJuridica;
import model.PessoaJuridicaRepo;

/**
 *
 * @author okidata
 */
public class CadastroPOO {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        String arquivo1;
        String arquivo2;

        arquivo1 = "pf.txt";
        arquivo2 = "pj.txt";

        PessoaFisicaRepo repo1 = new PessoaFisicaRepo();

        PessoaFisica pf1 = new PessoaFisica(001, "lucia", "212.212.212-21", 56);
        PessoaFisica pf2 = new PessoaFisica(002, "maria", "313.313.313-31", 70);

        repo1.inserir(pf1);
        repo1.inserir(pf2);
        try {
            repo1.persistir(arquivo1);
        } catch (IOException ex) {
            Logger.getLogger(CadastroPOO.class.getName()).log(Level.SEVERE, null, ex);
        }

        System.out.println("Dados de Pessoa Fisica Armazenados.");

        PessoaFisicaRepo repo2 = new PessoaFisicaRepo();

        try {
            repo2.recuperar(arquivo1);
        } catch (IOException ex) {
            Logger.getLogger(CadastroPOO.class.getName()).log(Level.SEVERE, null, ex);
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(CadastroPOO.class.getName()).log(Level.SEVERE, null, ex);
        }

        System.out.println("Dados de Pessoa Fisica Recuperados.");
        for (PessoaFisica pessoaF : repo2.obterTodos()) {
            pessoaF.exibir();
        }

        PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();

        PessoaJuridica pj1 = new PessoaJuridica(003, "xpto Sales", "3333333333");
        PessoaJuridica pj2 = new PessoaJuridica(004, "xpto Solutions", "444444444444");

        repo3.inserir(pj1);
```

```

repo3.inserir(pj2);
try {
    repo3.persistir(arquivo2);
} catch (IOException ex) {
    Logger.getLogger(CadastroPOO.class.getName()).log(Level.SEVERE, null, ex);
}

System.out.println("Dados de Pessoa Juridica Armazenados.");

PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();

try {
    repo4.recuperar(arquivo2);
} catch (IOException ex) {
    Logger.getLogger(CadastroPOO.class.getName()).log(Level.SEVERE, null, ex);
} catch (ClassNotFoundException ex) {
    Logger.getLogger(CadastroPOO.class.getName()).log(Level.SEVERE, null, ex);
}

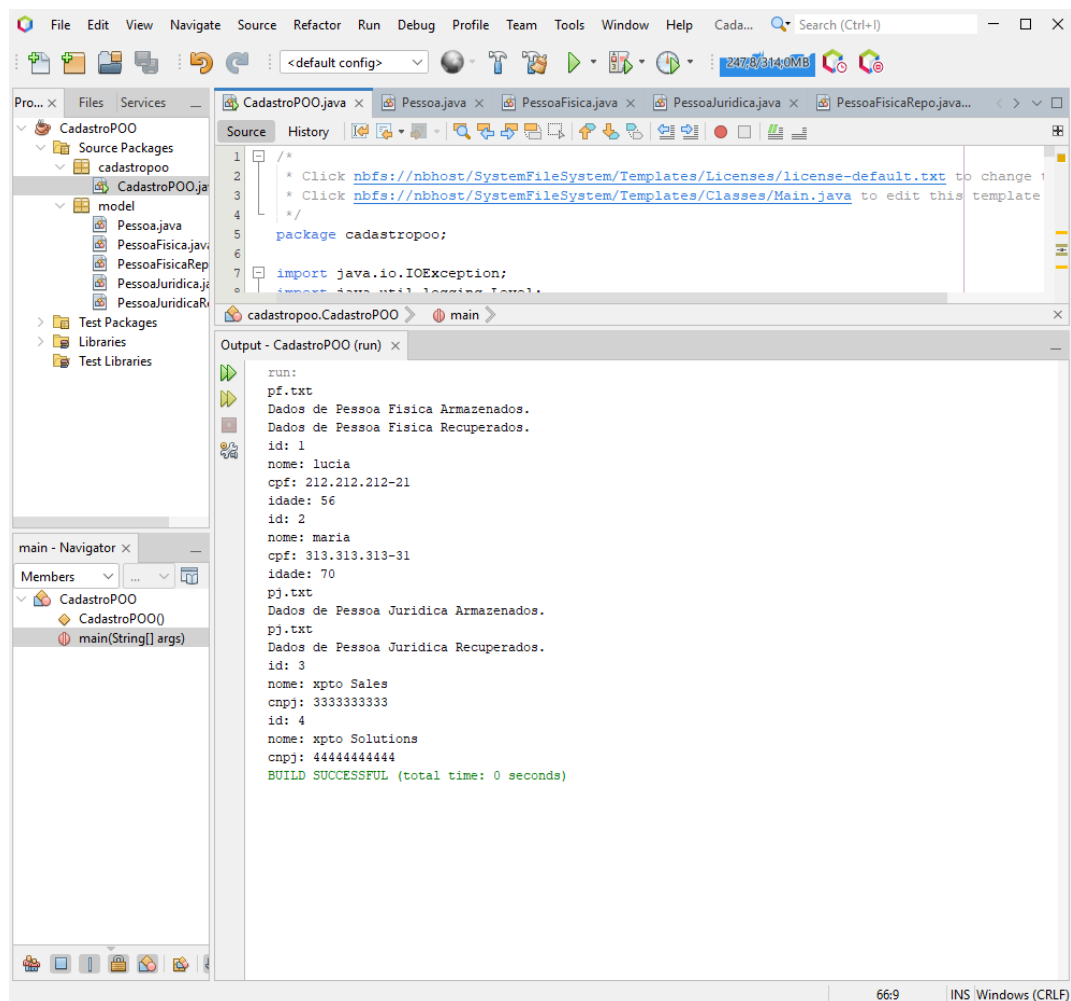
System.out.println("Dados de Pessoa Juridica Recuperados.");
for (PessoaJuridica pessoaJ : repo4.obterTodos()) {
    pessoaJ.exibir();
}

}

}

```

4. Resultado da execução do código:



5. Análise e Conclusão:

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

Vantagem: Reaproveitamento do código e consequentemente otimização do tempo, afinal através da herança a classe pai é capaz de compartilhar métodos e atributos com a classe filha, otimizando o processo de desenvolvimento do código.

Desvantagem: O acoplamento entre classe pai e filha, apesar de vantajoso, tem seu lado negativo, já que pode tornar mais complexa a manutenção do código.

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

Pois a interface Serializable converte os objetos em sequências de bytes, para que possam ser mais facilmente manipulados através do transporte pela rede.

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

Através da viabilização da utilização de Classes Funcionais, Lambdas, Streams (map, filter, reduce), entre outras funcionalidades da programação funcional.

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

O uso de métodos como “writeObject” e “readObject” das classes “ObjectOutputStream” e “ObjectInputStream” respectivamente. A fim de passar como parâmetro e posteriormente recuperar o objeto.

1. Missão Prática - Nível 1

2. Objetivos da Missão 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.

3. Códigos do Projeto:

```
package cadastropoo;

/**
 *
 * @author okidata
 */
public class CadastroPOO2 {

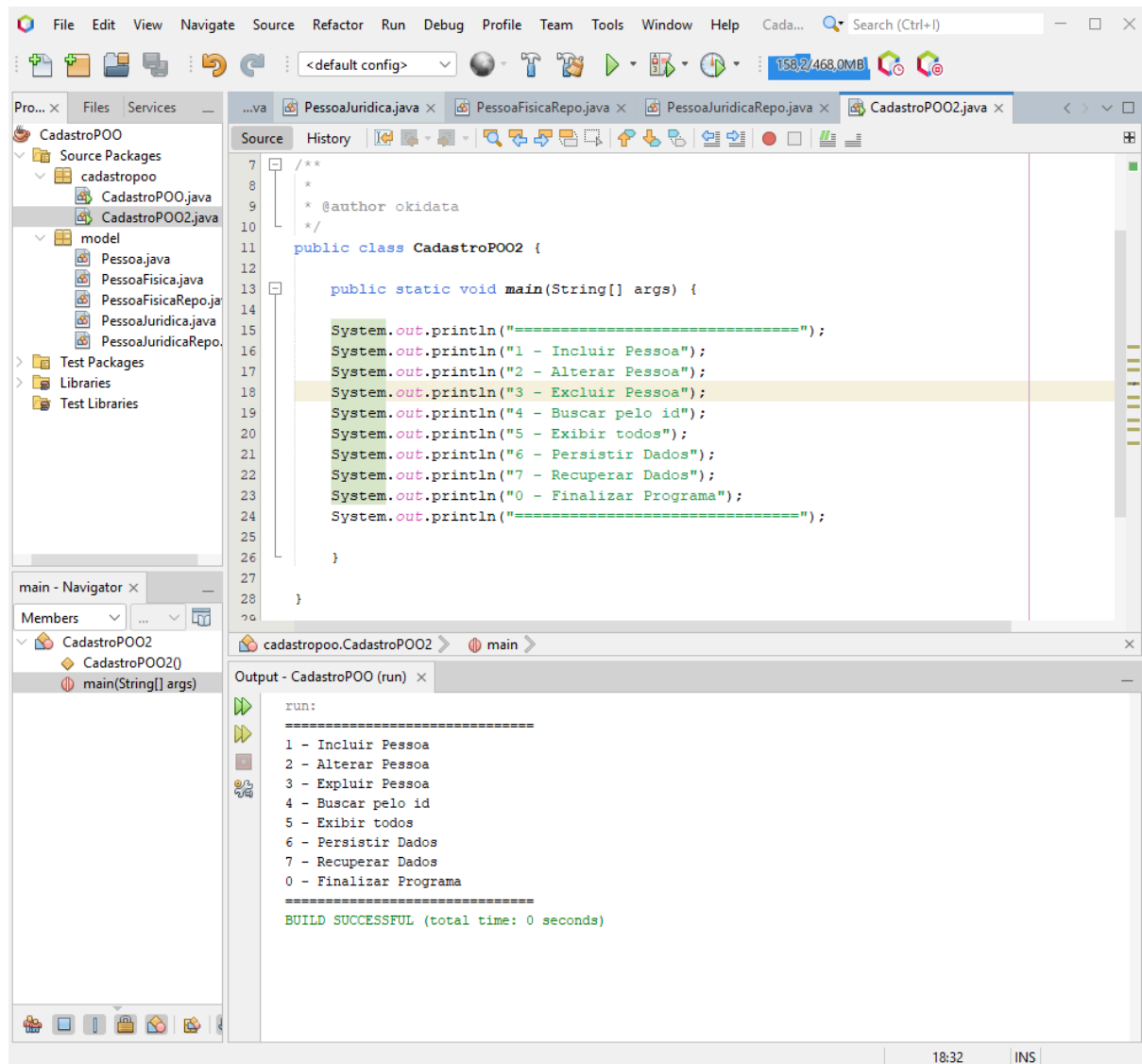
    public static void main(String[] args) {

        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 - Buscar pelo id");
        System.out.println("5 - Exibir todos");
        System.out.println("6 - Persistir Dados");
        System.out.println("7 - Recuperar Dados");
        System.out.println("0 - Finalizar Programa");
        System.out.println("=====");

    }

}
```

4. Resultado da execução do código:



```
7  /**
8   *
9   * @author okidata
10  */
11  public class CadastroPOO2 {
12
13      public static void main(String[] args) {
14
15          System.out.println("=====");
16          System.out.println("1 - Incluir Pessoa");
17          System.out.println("2 - Alterar Pessoa");
18          System.out.println("3 - Excluir Pessoa");
19          System.out.println("4 - Buscar pelo id");
20          System.out.println("5 - Exibir todos");
21          System.out.println("6 - Persistir Dados");
22          System.out.println("7 - Recuperar Dados");
23          System.out.println("0 - Finalizar Programa");
24          System.out.println("=====");
25
26      }
27
28  }
```

run:

```
=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo id
5 - Exibir todos
6 - Persistir Dados
7 - Recuperar Dados
0 - Finalizar Programa
=====
BUILD SUCCESSFUL (total time: 0 seconds)
```

5. Análise e Conclusão:

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

Elementos estáticos são aqueles que estão diretamente relacionados à classe, através do modificador do modificador “static”.

b. Para que serve a classe Scanner?

Scanner é uma classe de entrada utilizada para a leitura de dados.