Campus Virtual (SIA)

Curso - Desenvolvimento Full Stack

Disciplina - Programação Back-end Com Java

Turma - DGT2821

Semestre - 2025.4

Aluno - José Cristiano Chaves Dias

Criação das Entidades e Sistema de Persistência

Objetivos do trabalho

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.
5. 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.

Códigos:

Classe Pessoa

```
package model;

import java.io.Serializable;

public class Pessoa implements Serializable {

  private int id;
  private String nome;

  // Construtor padrão
  public Pessoa() {
  }

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

  // Método exibir
  public void exibir() {
    System.out.println("ID: " + id);
    System.out.println("Nome: " + nome);
  }

  // Getters e Setters
  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;
    }
}
```

Classe PessoaFisica

```
package model;

import java.io.Serializable;

public class PessoaFisica extends Pessoa implements Serializable {

  private String cpf;
  private int idade;

  // Construtor padrão
  public PessoaFisica() {
    super();
  }

  // Construtor completo
  public PessoaFisica(int id, String nome, String cpf, int idade) {
    super(id, nome);
    this.cpf = cpf;
    this.idade = idade;
  }

  // Método exibir polimórfico
  @Override
  public void exibir() {
    super.exibir();
    System.out.println("CPF: " + cpf);
    System.out.println("Idade: " + idade);
  }

  // Getters e Setters
  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;
  }
}
```

Classe PessoaJuridica

```
package model;

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

public class PessoaJuridicaRepo {

  private ArrayList<PessoaJuridica> lista = new ArrayList<>();

  // Inserir
  public void inserir(PessoaJuridica pj) {
    lista.add(pj);
  }

  // Alterar
  public void alterar(PessoaJuridica pj) {
    for (int i = 0; i < lista.size(); i++) {
      if (lista.get(i).getId() == pj.getId()) {
        lista.set(i, pj);
        return;
      }
    }
  }

  // Excluir
  public void excluir(int id) {
    lista.removeIf(p -> p.getId() == id);
  }

  // Obter por ID
  public PessoaJuridica obter(int id) {
    for (PessoaJuridica pj : lista) {
      if (pj.getId() == id) {
        return pj;
      }
    }
    return null;
  }

  // Obter todos
  public ArrayList<PessoaJuridica> obterTodos() {
    return lista;
  }

  // Persistir
  public void persistir(String nomeArquivo) throws Exception {
    ObjectOutputStream out = new ObjectOutputStream(
        new FileOutputStream(nomeArquivo));
```

```
        out.writeObject(lista);
        out.close();
    }

    // Recuperar
    @SuppressWarnings("unchecked")
    public void recuperar(String nomeArquivo) throws Exception {
        ObjectInputStream in = new ObjectInputStream(
                new FileInputStream(nomeArquivo));
        lista = (ArrayList<PessoaJuridica>) in.readObject();
        in.close();
    }
}
```

Classe PessoaJuridicaRepo

```
package model;

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

public class PessoaJuridicaRepo {

  private ArrayList<PessoaJuridica> lista = new ArrayList<>();

  // Inserir
  public void inserir(PessoaJuridica pj) {
    lista.add(pj);
  }

  // Alterar
  public void alterar(PessoaJuridica pj) {
    for (int i = 0; i < lista.size(); i++) {
      if (lista.get(i).getId() == pj.getId()) {
        lista.set(i, pj);
        return;
      }
    }
  }

  // Excluir
  public void excluir(int id) {
    lista.removeIf(p -> p.getId() == id);
  }

  // Obter por ID
  public PessoaJuridica obter(int id) {
```

```
        for (PessoaJuridica pj : lista) {
            if (pj.getId() == id) {
                return pj;
            }
        }
        return null;
    }

    // Obter todos
    public ArrayList<PessoaJuridica> obterTodos() {
        return lista;
    }

    // Persistir
    public void persistir(String nomeArquivo) throws Exception {
        ObjectOutputStream out = new ObjectOutputStream(
                new FileOutputStream(nomeArquivo));
        out.writeObject(lista);
        out.close();
    }

    // Recuperar
    @SuppressWarnings("unchecked")
    public void recuperar(String nomeArquivo) throws Exception {
        ObjectInputStream in = new ObjectInputStream(
                new FileInputStream(nomeArquivo));
        lista = (ArrayList<PessoaJuridica>) in.readObject();
        in.close();
    }
}
```

Classe PessoaFisicaRepo

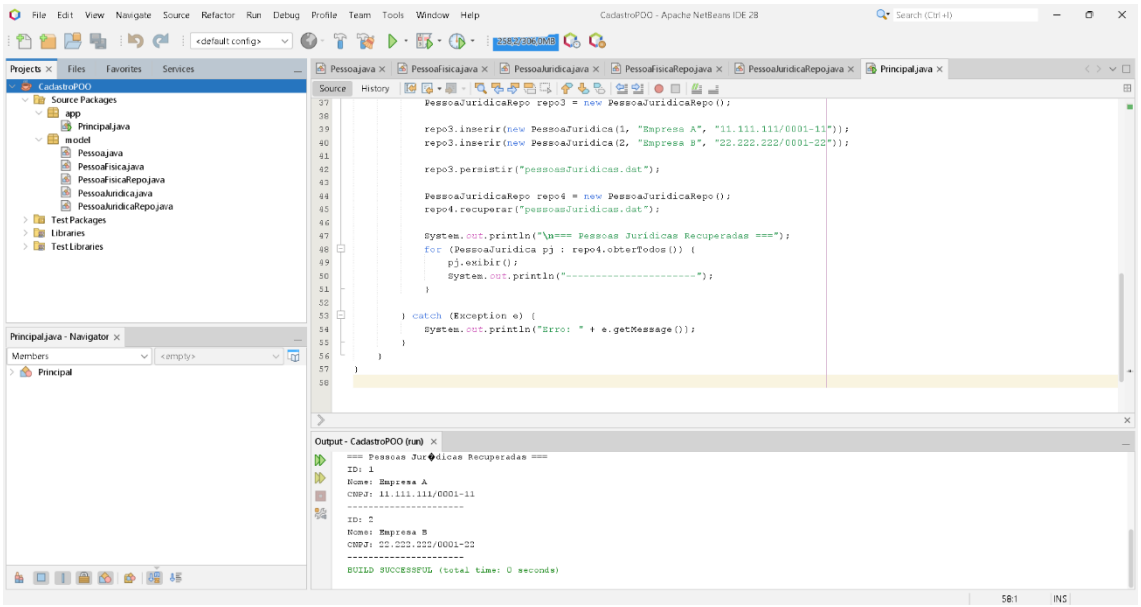
```
package model;

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

public class PessoaFisicaRepo {

  private ArrayList<PessoaFisica> lista = new ArrayList<>();

  // Inserir
  public void inserir(PessoaFisica pf) {
    lista.add(pf);
  }

  // Alterar
  public void alterar(PessoaFisica pf) {
    for (int i = 0; i < lista.size(); i++) {
      if (lista.get(i).getId() == pf.getId()) {
        lista.set(i, pf);
        return;
      }
    }
  }

  // Excluir
  public void excluir(int id) {
    lista.removeIf(p -> p.getId() == id);
  }

  // Obter por ID
  public PessoaFisica obter(int id) {
```

```
        for (PessoaFisica pf : lista) {
            if (pf.getId() == id) {
                return pf;
            }
        }
        return null;
    }

    // Obter todos
    public ArrayList<PessoaFisica> obterTodos() {
        return lista;
    }

    // Persistir
    public void persistir(String nomeArquivo) throws Exception {
        ObjectOutputStream out = new ObjectOutputStream(
                new FileOutputStream(nomeArquivo));
        out.writeObject(lista);
        out.close();
    }

    // Recuperar
    @SuppressWarnings("unchecked")
    public void recuperar(String nomeArquivo) throws Exception {
        ObjectInputStream in = new ObjectInputStream(
                new FileInputStream(nomeArquivo));
        lista = (ArrayList<PessoaFisica>) in.readObject();
        in.close();
    }
}
```

Classe Principal

```
package app;

import model.*;

public class Principal {

    public static void main(String[] args) {

        try {

            PessoaFisicaRepo repo1 = new PessoaFisicaRepo();

            repo1.inserir(new PessoaFisica(1, "Cristiano", "111.111.111-11", 43));
            repo1.inserir(new PessoaFisica(2, "Maria", "222.222.222-22", 30));

            repo1.persistir("pessoasFisicas.dat");

            PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
            repo2.recuperar("pessoasFisicas.dat");

            System.out.println("=== Pessoas Físicas Recuperadas ===");
            for (PessoaFisica pf : repo2.obterTodos()) {
                pf.exibir();
                System.out.println("----------------------");
            }

            PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();

            repo3.inserir(new PessoaJuridica(1, "Empresa A", "11.111.111/0001-11"));
            repo3.inserir(new PessoaJuridica(2, "Empresa B", "22.222.222/0001-22"));

            repo3.persistir("pessoasJuridicas.dat");

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

            System.out.println("\n=== Pessoas Jurídicas Recuperadas ===");
            for (PessoaJuridica pj : repo4.obterTodos()) {
                pj.exibir();
                System.out.println("----------------------");
            }

        } catch (Exception e) {
            System.out.println("Erro: " + e.getMessage());
        }
    }
}
```

Tela do resultado da execução:



Vantagens e desvantagens do uso de herança

Ao desenvolver este trabalho na prática foi possível ver como a herança ajuda a organizar o código. Ao criar a classe Pessoa como base para PessoaFisica e PessoaJuridica, evitou-se repetir informações que são comuns, como o id e o nome. Isso deixou o código mais limpo e fácil de entender.

A principal vantagem foi a reutilização. Quando algo precisa ser alterado na classe principal, não é necessário modificar várias partes do sistema, Além disso, o uso do polimorfismo permitiu que cada tipo de pessoa tivesse sua própria forma de exibir os dados, mesmo mantendo uma estrutura comum.

Por outro lado, também ficou claro que é preciso cuidado ao usar herança. Se a classe principal sofrer muitas mudanças, todas as classes que dependem dela podem ser afetadas. Por isso, é importante planejar bem a estrutura antes de começar a programar.

De modo geral, a herança ajudou bastante na organização e deixou o sistema mais estruturado.

Por que a interface Serializable é necessária?

Durante o desenvolvimento, foi necessário salvar os dados das pessoas em arquivos para que as informações não se perdessem ao encerrar o programa. Para isso, foi utilizada a interface Serializable.

Ela permite que os objetos sejam convertidos em um formato que pode ser armazenado em arquivo. Sem essa interface, o Java não permite que os dados sejam gravados em formato binário.

Na prática, isso significa que o sistema consegue salvar as informações no disco e depois recuperá-las quando o programa for executado novamente. Isso torna o sistema mais completo e próximo de uma aplicação real.

Como o paradigma funcional aparece na API Stream?

Embora o foco principal do trabalho tenha sido a programação orientada a objetos, o Java também oferece recursos que seguem uma abordagem mais moderna, como a API Stream.

Essa API permite trabalhar com listas e coleções de forma mais simples e direta, usando expressões mais curtas e objetivas. Em vez de escrever vários comandos para percorrer uma lista, é possível usar métodos que tornam o código mais limpo e fácil de ler.

Mesmo não sendo o foco principal deste projeto, entender que o Java também permite essa forma de programação amplia a visão sobre as possibilidades da linguagem.

Qual padrão foi utilizado na persistência de dados?

No projeto, os dados foram organizados utilizando uma estrutura semelhante ao padrão de repositório. As classes responsáveis por gerenciar as pessoas físicas e jurídicas ficaram separadas das classes que representam as entidades.

Essa organização facilitou bastante o entendimento do código, pois cada parte do sistema ficou responsável por uma função específica. As entidades representam os dados, enquanto os repositórios cuidam das operações como inserir, alterar, excluir e salvar em arquivo.

Essa separação torna o sistema mais organizado e mais fácil de manter.

Conclusão Final

A realização deste trabalho foi muito importante para consolidar os conhecimentos sobre programação orientada a objetos em Java. Foi possível aplicar conceitos como herança, polimorfismo, encapsulamento e tratamento de exceções de forma prática.

Também foi possível compreender como funciona a persistência de dados em arquivos binários, permitindo que as informações sejam armazenadas e recuperadas posteriormente.

Além disso, a organização em pacotes e a separação das responsabilidades deixaram o projeto mais estruturado e profissional.

De maneira geral, a prática contribuiu para um melhor entendimento da linguagem Java e mostrou como esses conceitos são aplicados no desenvolvimento de sistemas reais.

Criação do Cadastro em Modo Texto

Objetivo da Prática

O objetivo desta prática foi desenvolver um sistema simples em Java para cadastrar e organizar informações de pessoas físicas e jurídicas. A ideia principal foi colocar em prática os conceitos vistos em aula, principalmente organização em classes, uso de pacotes, entrada de dados pelo teclado e salvamento das informações em arquivo.

Além disso, a atividade ajudou a entender melhor como estruturar um projeto de forma organizada, separando as responsabilidades de cada parte do sistema.

Códigos:

Classe Pessoa

```
package model;

import java.io.Serializable;

public abstract class Pessoa implements Serializable {

  private String id;
  private String nome;

  public Pessoa() {}

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

  public String getId() {
    return id;
  }

  public void setId(String id) {
    this.id = id;
  }

  public String getNome() {
    return nome;
  }

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

  public abstract void exibir();
}
```

Classe PessoaFisica

```
package model;

public class PessoaFisica extends Pessoa {

  private String cpf;

  public PessoaFisica() {}

  public PessoaFisica(String id, String nome, String cpf) {
    super(id, nome);
    this.cpf = cpf;
  }

  public String getCpf() {
    return cpf;
  }

  public void setCpf(String cpf) {
    this.cpf = cpf;
  }

  @Override
  public void exibir() {
    System.out.println("Pessoa Física");
    System.out.println("ID: " + getId());
    System.out.println("Nome: " + getNome());
    System.out.println("CPF: " + cpf);
    System.out.println("----------------------------");
  }
}
```

Classe PessoaJuridica

```
package model;

public class PessoaJuridica extends Pessoa {

  private String cnpj;

  public PessoaJuridica() {}

  public PessoaJuridica(String 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() {
    System.out.println("Pessoa Jurídica");
    System.out.println("ID: " + getId());
    System.out.println("Nome: " + getNome());
    System.out.println("CNPJ: " + cnpj);
    System.out.println("----------------------------");
  }
}
```

Classe PessoaFisicaRepo

```
package model;

import model.PessoaFisica;
import java.io.*;
import java.util.ArrayList;

public class PessoaFisicaRepo {

  private ArrayList<PessoaFisica> lista = new ArrayList<>();

  public void inserir(PessoaFisica pf) {
    lista.add(pf);
  }

  public PessoaFisica obter(String id) {
    for (PessoaFisica pf : lista) {
      if (pf.getId().equals(id)) {
        return pf;
      }
    }
    return null;
  }

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

  public void salvar(String prefixo) throws Exception {
    ObjectOutputStream oos = new ObjectOutputStream(
        new FileOutputStream(prefixo + ".fisica.bin"));
    oos.writeObject(lista);
    oos.close();
  }

  public void recuperar(String prefixo) throws Exception {
    ObjectInputStream ois = new ObjectInputStream(
        new FileInputStream(prefixo + ".fisica.bin"));
    lista = (ArrayList<PessoaFisica>) ois.readObject();
    ois.close();
  }
}
```

Classe PessoaJuridicaRepo

```
package model;

import model.PessoaJuridica;
import java.io.*;
import java.util.ArrayList;

public class PessoaJuridicaRepo {

  private ArrayList<PessoaJuridica> lista = new ArrayList<>();

  public void inserir(PessoaJuridica pj) {
    lista.add(pj);
  }

  public PessoaJuridica obter(String id) {
    for (PessoaJuridica pj : lista) {
      if (pj.getId().equals(id)) {
        return pj;
      }
    }
    return null;
  }

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

  public void salvar(String prefixo) throws Exception {
    ObjectOutputStream oos = new ObjectOutputStream(
        new FileOutputStream(prefixo + ".juridica.bin"));
    oos.writeObject(lista);
    oos.close();
  }

  public void recuperar(String prefixo) throws Exception {
    ObjectInputStream ois = new ObjectInputStream(
        new FileInputStream(prefixo + ".juridica.bin"));
    lista = (ArrayList<PessoaJuridica>) ois.readObject();
    ois.close();
  }
}
```

Classe Principal

```
package app;

import model.*;
import java.util.Scanner;

public class Principal {

  public static void main(String[] args) {

    Scanner sc = new Scanner(System.in);
    PessoaFisicaRepo repoFisica = new PessoaFisicaRepo();
    PessoaJuridicaRepo repoJuridica = new PessoaJuridicaRepo();

    int opcao;

    do {
      System.out.println("===== MENU =====");
      System.out.println("1 - Incluir");
      System.out.println("2 - Exibir por ID");
      System.out.println("3 - Exibir todos");
      System.out.println("4 - Salvar");
      System.out.println("5 - Recuperar");
      System.out.println("0 - Sair");
      System.out.print("Escolha uma opção: ");

      opcao = Integer.parseInt(sc.nextLine());

      switch (opcao) {

        case 1:
          System.out.print("Pessoa Física (1) ou Jurídica (2)? ");
          int tipo = Integer.parseInt(sc.nextLine());

          System.out.print("ID: ");
          String id = sc.nextLine();

          System.out.print("Nome: ");
          String nome = sc.nextLine();

          if (tipo == 1) {
            System.out.print("CPF: ");
            String cpf = sc.nextLine();
            repoFisica.inserir(new PessoaFisica(id, nome, cpf));
          } else {
            System.out.print("CNPJ: ");
            String cnpj = sc.nextLine();
            repoJuridica.inserir(new PessoaJuridica(id, nome, cnpj));
          }
```

```
            break;

        case 2:
            System.out.print("Pessoa Física (1) ou Jurídica (2)? ");
            tipo = Integer.parseInt(sc.nextLine());

            System.out.print("ID: ");
            id = sc.nextLine();

            if (tipo == 1) {
                PessoaFisica pf = repoFisica.obter(id);
                if (pf != null) pf.exibir();
                else System.out.println("Não encontrada.");
            } else {
                PessoaJuridica pj = repoJuridica.obter(id);
                if (pj != null) pj.exibir();
                else System.out.println("Não encontrada.");
            }
            break;

        case 3:
            for (PessoaFisica pf : repoFisica.obterTodos())
                pf.exibir();

            for (PessoaJuridica pj : repoJuridica.obterTodos())
                pj.exibir();
            break;

        case 4:
            try {
                System.out.print("Prefixo do arquivo: ");
                String prefixo = sc.nextLine();
                repoFisica.salvar(prefixo);
                repoJuridica.salvar(prefixo);
                System.out.println("Dados salvos com sucesso!");
            } catch (Exception e) {
                System.out.println("Erro ao salvar.");
            }
            break;

        case 5:
            try {
                System.out.print("Prefixo do arquivo: ");
                String prefixo = sc.nextLine();
                repoFisica.recuperar(prefixo);
                repoJuridica.recuperar(prefixo);
                System.out.println("Dados recuperados com sucesso!");
            } catch (Exception e) {
                System.out.println("Erro ao recuperar.");
```

```
                }
                break;

            }

        } while (opcao != 0);

        System.out.println("Programa finalizado.");
    }
}
```

Resultados da Execução

run:

===== MENU =====

1 - Incluir

2 - Exibir por ID

3 - Exibir todos

4 - Salvar

5 - Recuperar

0 - Sair

Escolha uma op□□o: 1

Pessoa F□sica (1) ou Jur□dica (2)? 1

ID: 1

Nome: dias

CPF: 96385274100

===== MENU =====

1 - Incluir

2 - Exibir por ID

3 - Exibir todos

4 - Salvar

5 - Recuperar

0 - Sair

Escolha uma op□□o: 4

Prefixo do arquivo: 1

Dados salvos com sucesso!

===== MENU =====

1 - Incluir

2 - Exibir por ID

3 - Exibir todos

4 - Salvar

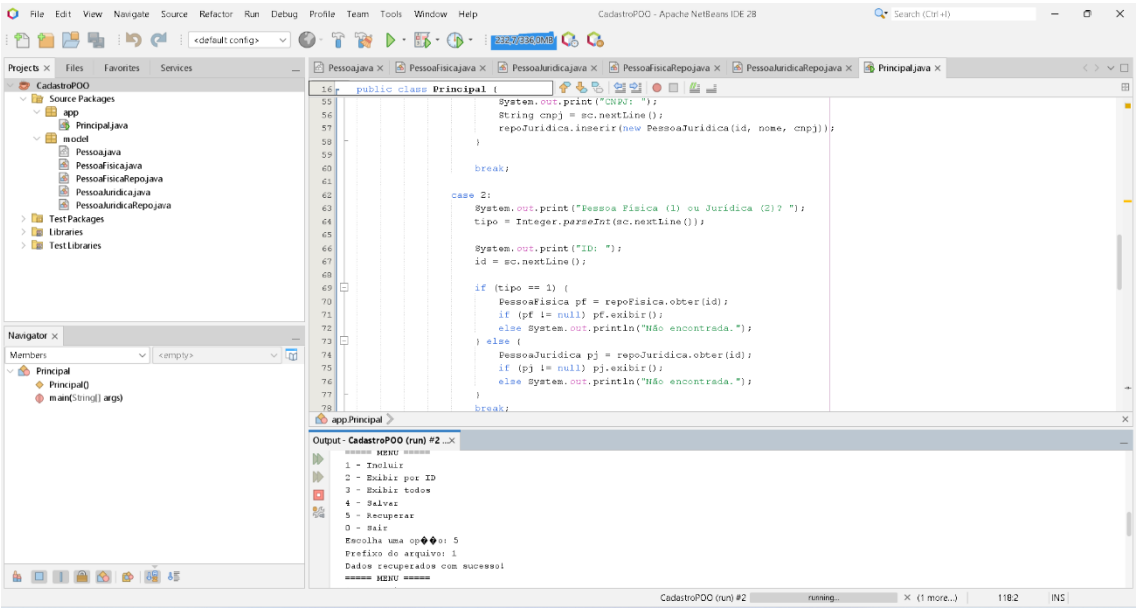
5 - Recuperar

0 - Sair

Escolha uma op□□o: 5

Prefixo do arquivo: 1

Dados recuperados com sucesso!
===== MENU =====
1 - Incluir
2 - Exibir por ID
3 - Exibir todos
4 - Salvar
5 - Recuperar
0 - Sair
Escolha uma op□□o: 3
Pessoa F□sica
ID: 1
Nome: dias
CPF: 96385274100



Análise e Conclusão

O que são elementos estáticos e por que o método main é static?

Pertencem à classe e não a um objeto específico e podem ser usados sem precisar criar uma instância da classe.

Para que serve a classe Scanner?

A classe Scanner serve para ler dados digitados pelo usuário no teclado e permite que o usuário digite informações como ID, nome, CPF e CNPJ. Sem ela, o programa não conseguiria receber dados externos durante a execução.

Como o uso das classes de repositório ajudou na organização?

As classes de repositório foram importantes para manter o código mais organizado. Elas ficaram responsáveis por guardar, buscar, salvar e recuperar os dados, isso evitou que a classe principal ficasse muito grande e confusa.

Considerações Finais

A prática foi importante para consolidar o entendimento sobre programação orientada a objetos em Java. Foi possível aplicar conceitos como organização em pacotes, criação de classes, uso de herança e manipulação de arquivos.

O sistema desenvolvido atendeu ao que foi solicitado na atividade, funcionando corretamente em todos os testes realizados.

ENDEREÇO NO GITHUB:

https://github.com/CristianoCDias/Programacao-Back-end-Com-Java