



Missão Prática - Nível 1 Mundo 3

Campus : POLO COPACABANA

Curso : DESENVOLVIMENTO FULL STACK

Disciplina : RPG0014 INICIANDO O CAMINHO PELO JAVA

Turma : 9003

Semestre: 2023.3 FLEX

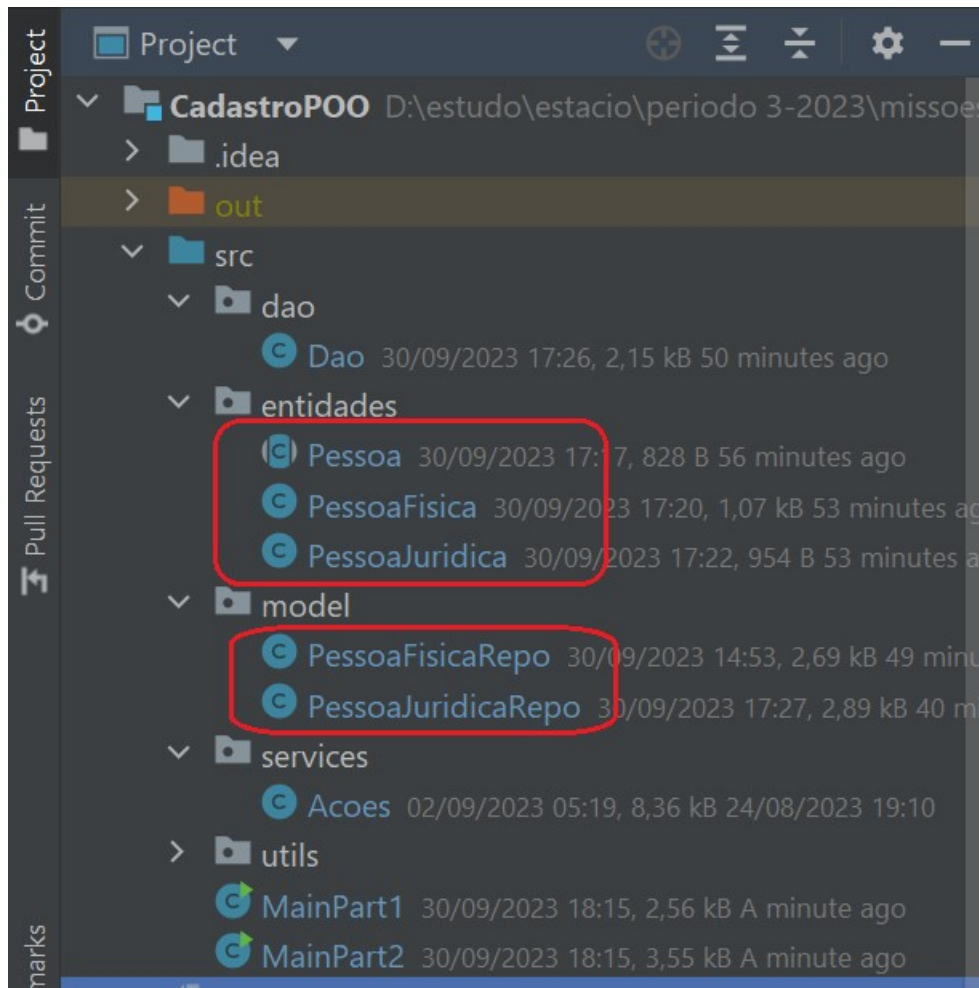
Nome : MARCO SERGIO ALBINO VITTORIO BAROZZI

1º Procedimento | Criação das Entidades e Sistema de Persistência

Objetivos:

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 desenvolvidos :



Classe MainPart1.java

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

public class MainPart1 {
    public static void main(String[] args) {
        // a - Instanciar um repositório de pessoas físicas
        (repol).
        PessoaFisicaRepo repol= new PessoaFisicaRepo();

        // b - Adicionar duas pessoas físicas, utilizando o
```

```

construtor
    //completo.

    repo1.inserir(new PessoaFisica("Matheus",25,"1222611"))
;

    repo1.inserir(new PessoaFisica("Marcia",22,"1223211")) ;
    repo1.inserir(new PessoaFisica("Carlos",19,"122222")) ;

    System.out.println("");
    System.out.println("");

    // c -Invocar o método de persistência em repo1,
fornecendo
    //um nome de arquivo fixo, através do código.
    repo1.persistir("listaPessoafisica");

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

    //d - Instanciar outro repositório de pessoas físicas
(repo2).
    PessoaFisicaRepo repo2= new PessoaFisicaRepo();

    // e- Invocar o método de recuperação em repo2,
fornecendo o
    //mesmo nome de arquivo utilizado anteriormente.

    repo2.recuperar("listaPessoafisica");

    // f- Exibir os dados de todas as pessoas físicas
recuperadas.
    System.out.println("Listar Pessoas Fisicas
Recuperadas");
    repo2.ListarTodas();

    // g - Instanciar um repositório de pessoas jurídicas
(repo3).
    PessoaJuridicaRepo repo3= new PessoaJuridicaRepo();

    // h- Adicionar duas pessoas jurídicas, utilizando o
construtor
    //completo.
    repo3.inserir(new PessoaJuridica("BB1","1222611")) ;
    repo3.inserir(new PessoaJuridica("M.LUIZA","1223211")) ;
    repo3.inserir(new PessoaJuridica("LOJAS S/A","122222"))
;

```

```

        // i - Invocar o método de persistência em repo3,
fornecendo
        //um nome de arquivo fixo, através do código.
        System.out.println("Dados Pessoa Juridicas
Aramzenados");
        repo3.persistir("listaPessoaJuridica");

        // j - Instanciar outro repositório de pessoas jurídicas
(repo4).
        PessoaJuridicaRepo repo4= new PessoaJuridicaRepo();

        // k - Invocar o método de recuperação em repo4,
fornecendo o
        //mesmo nome de arquivo utilizado anteriormente.
        repo4.recuperar("listaPessoaJuridica");

        // l - Exibir os dados de todas as pessoas jurídicas
//recuperadas.
        System.out.println("Dados Pessoa Juridicas
Recuperados");
        repo4.ListarTodas();
    }
}

```

Classe Pessoa.java

```

package entidades;

import utils.IDControle;

import java.io.Serializable;

public abstract class Pessoa implements Serializable {
    private static final long serialVersionUID = 1L;
    private static Integer id = 0;
    private Integer individualId;
    private String nome;

    public Pessoa() {

    }

    public Pessoa( String nome) {

```

```

        //this.id += 1;
        this.id= IDControle.getID();
        this.individualId=id;

        this.nome = nome;
    }

    public Integer getId() {
        return individualId;
    }

    public String getNome() {
        return nome;
    }

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

    public abstract String exibir();
}

```

Classe PessoaFisica.java

```

package entidades;

import java.io.Serializable;

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

    private Integer idade;
    private String cpf;
    public PessoaFisica() {
    }
    public PessoaFisica(Integer idade, String cpf) {
        this.idade = idade;
        this.cpf = cpf;
    }
}

```

```

    public PessoaFisica( String nome, Integer idade, String cpf)
    {
        super( nome);
        this.idade = idade;
        this.cpf = cpf;

    }
    public Integer getIdade() {
        return idade;
    }

    public void setIdade(Integer idade) {
        this.idade = idade;
    }

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

    @Override
    public String exhibir() {
        return
            "id : " + getId() + "\n" +
            "nome : " + getNome() + "\n" +
            "idade : " + idade + "\n" +
            "cpf : " + cpf + "\n" ;
    }
}

```

Classe PessoaJuridica.java

```

package entidades;

import java.io.Serializable;

import utils.IDControle;

```

```

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

    private String cnpj;

    public PessoaJuridica() {

    }
    public PessoaJuridica(String cnpj) {
        this.cnpj = cnpj;
    }
    public PessoaJuridica(String nome, String cnpj) {
        super(nome);
        this.cnpj = cnpj;
    }
    public PessoaJuridica(Integer id, String nome, String cnpj)
{
        super(nome);
        this.cnpj = cnpj;
    }
    public String getCnpj() {
        return cnpj;
    }
    public void setCnpj(String cnpj) {
        this.cnpj = cnpj;
    }
    @Override
    public String exhibir() {
        return
            "id : " + getId() + "\n" +
            "empresa : " + getNome() + "\n" +
            "cnpj : " + getCnpj() + "\n";
    }
}
}

```

Classe PessoaFisicaRepo.java

```
package model;

import entidades.PessoaFisica;

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

public class PessoaFisicaRepo {

    private List<PessoaFisica> listaPessoasFisicas;

    // Construtor
    public PessoaFisicaRepo() {

        this.listaPessoasFisicas = new ArrayList<>();
    }

    // Método para inserir uma pessoa física
    public void inserir(PessoaFisica pessoaFisica) {

        listaPessoasFisicas.add(pessoaFisica);
    }

    // Método para alterar uma pessoa física

    public boolean alterar(PessoaFisica pessoaFisica) {
        for (int i = 0; i < listaPessoasFisicas.size(); i++) {
            if (listaPessoasFisicas.get(i).getId() ==
pessoaFisica.getId()) {
                listaPessoasFisicas.set(i, pessoaFisica);
                return true; // Retorna true indicando sucesso
na alteração
            }
        }

        return false; // Retorna false se a pessoa física não
foi encontrada na lista
    }

    // Método para excluir uma pessoa física por ID

    public boolean excluir(int id) {
```



```

        for (PessoaFisica p:listaPessoasFisicas) {
            if(p.getId()==id){
                listaPessoasFisicas.remove(p);
                return true;
            }
        }
        return false; // Retorna false se a pessoa física não
foi encontrada na lista
    }

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

    // Método para obter todas as pessoas físicas

    public List<PessoaFisica> obterTodos() {
        return listaPessoasFisicas ;
    }

    public void persistir(String nomeArquivo) {

        if(listaPessoasFisicas.isEmpty()){
            System.out.println("Não existem registros a serem
persistidos");
            return;
        }

        try (ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(nomeArquivo))) {
            out.writeObject(listaPessoasFisicas);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void recuperar(String nomeArquivo) {
        try (ObjectInputStream in = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
            listaPessoasFisicas = (ArrayList<PessoaFisica>)
in.readObject();

```

```

        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    public void ListarTodas(){
        for (PessoaFisica pf: listaPessoasFisicas) {
            System.out.println(pf.exibir());
        }
    }
}

```

Classe PessoaJuridicaRepo.java

```

package model;

import entidades.PessoaJuridica;

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

public class PessoaJuridicaRepo {

    private List<PessoaJuridica> listaPessoasJuridicas;

    // Construtor
    public PessoaJuridicaRepo() {

        this.listaPessoasJuridicas = new
ArrayList<PessoaJuridica>();
    }

    // Método para inserir uma pessoa física
    public void inserir(PessoaJuridica pessoaJuridica) {
        listaPessoasJuridicas.add(pessoaJuridica);
    }

    // Método para alterar uma pessoa física

    public boolean alterar(PessoaJuridica pessoaJuridica) {

```

```

        for (int i = 0; i < listaPessoasJuridicas.size(); i++) {
            if (listaPessoasJuridicas.get(i).getId() ==
pessoaJuridica.getId()) {
                listaPessoasJuridicas.set(i, pessoaJuridica);
                return true; // Retorna true indicando sucesso
na alteração
            }
        }

        return false; // Retorna false se a pessoa não foi
encontrada na lista
    }

    // Método para excluir uma pessoa física por ID
    public boolean excluir(int id) {
        for (PessoaJuridica p: listaPessoasJuridicas) {
            if(p.getId()==id){
                listaPessoasJuridicas.remove(p);
                return true;
            }
        }
        return false; // Retorna false se a pessoa física não
foi encontrada na lista
    }

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

    // Método para obter todas as pessoas físicas
    public ArrayList<PessoaJuridica> obterTodos_1() {
        return new ArrayList<>(listaPessoasJuridicas);
    }

    public List<PessoaJuridica> obterTodos() {
        return listaPessoasJuridicas;
    }

    public void persistir(String nomeArquivo) {
        if(listaPessoasJuridicas.isEmpty()){
            System.out.println("Não existem registros a serem
persistidos");
            return;
        }
    }

```

```

        try (ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(nomeArquivo))) {
            out.writeObject(listaPessoasJuridicas);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void recuperar(String nomeArquivo) {
        try (ObjectInputStream in = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
            listaPessoasJuridicas = (ArrayList<PessoaJuridica>)
in.readObject();

            } catch (IOException | ClassNotFoundException e) {
                e.printStackTrace();
            }
        }

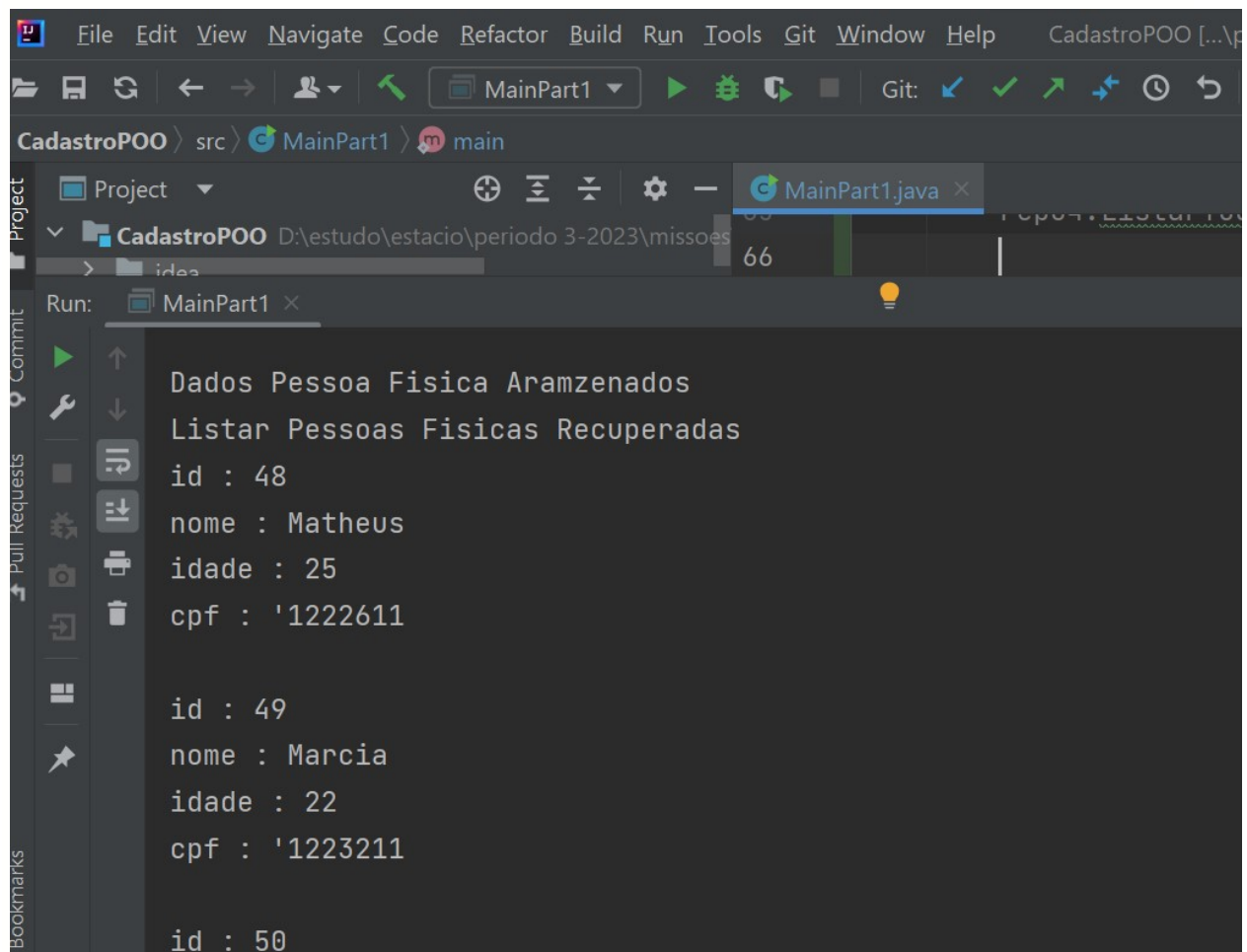
        public void ListarTodas(){
            for (PessoaJuridica pf: listaPessoasJuridicas) {
                System.out.println(pf.exibir());
            }
        }
    }
}

```

Resultados da execução dos códigos

- 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.



```
File Edit View Navigate Code Refactor Build Run Tools Git Window Help CadastroPOO [...\p
MainPart1
CadastroPOO > src > MainPart1 > main
Project
CadastroPOO D:\estudo\estacio\periodo 3-2023\missoes
Run: MainPart1
Dados Pessoa Fisica Aramzenados
Listar Pessoas Fisicas Recuperadas
id : 48
nome : Matheus
idade : 25
cpf : '1222611

id : 49
nome : Marcia
idade : 22
cpf : '1223211

id : 50
```

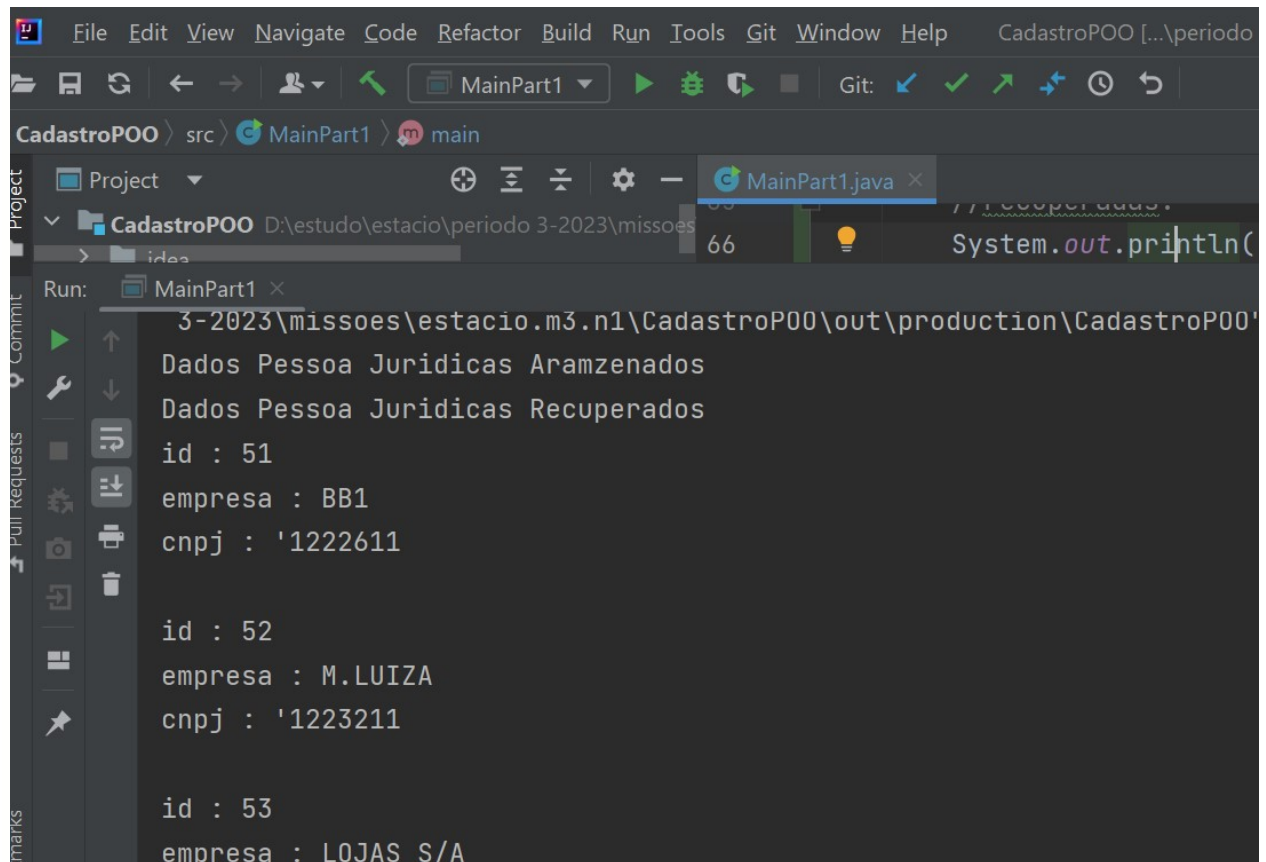
h- Adicionar duas pessoas jurídicas, utilizando o construtor completo.

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.

l - Exibir os dados de todas as pessoas jurídicas recuperadas.



```
File Edit View Navigate Code Refactor Build Run Tools Git Window Help CadastroPOO [...\periodo
MainPart1
CadastroPOO > src > MainPart1 > main
Project
CadastroPOO D:\estudo\estacio\periodo 3-2023\missoes
MainPart1.java
System.out.println(
Run: MainPart1
3-2023\missoes\estacio.m3.n1\CadastroPOO\out\production\CadastroPOO
Dados Pessoa Juridicas Aramzenados
Dados Pessoa Juridicas Recuperados
id : 51
empresa : BB1
cnpj : '1222611
id : 52
empresa : M.LUIZA
cnpj : '1223211
id : 53
empresa : LOJAS S/A
```

Análise e Conclusão:

Quais as vantagens e desvantagens do uso de herança?

r) Reutilização de código, polimorfismo no sentido de utilizar classes como base para outras, extensibilidade ou seja adicionar funcionalidades ou modificar alguma existente, encapsulamento, relacionamentos etc...

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

r) Devido a interface indicar que a classe em questão poderá ter uma serialização e desserialização, no caso de recuperação de dados em um arquivo binário.

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

r) Através de operação de alto nível como map, filter e reduce para processar coleções de dados.
Tambem permitem "Expressões Lambda"

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

r) Acredito que sejam os ORMs (Object-Relational Mapping) como Hibernate ou JPA são mais associados a bancos de dados relacionais, eles também podem ser usados para persistir dados em arquivos, especialmente quando bancos de dados integrados são usados.

Eles permitem que os desenvolvedores interajam com bancos de dados usando objetos Java em vez de SQL direto.