Estácio

POLO PRINC. IZABEL - CACHOEIRINHA - RS

DESENVOLVIMENTO FULL STACK

RPG0014 -Iniciando o caminho pelo Java

Turma 9001

Primeiro semestre de 2024

Integrantes: Fagner Luiz Gimenez Mendes.

1º Procedimento | Criação das Entidades e Sistema de Persistência

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.
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.

Todos os códigos solicitados neste roteiro de aula;

```
package model;

import java.io.Serializable;


public class Pessoa implements Serializable{

   private static final long serialVersionUID = 1L;
   int id;
   String nome;

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

   }

   public void exibir() {
           System.out.println("ID:" + id);
           System.out.println("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;

    }
}

package model;

import java.io.Serializable;

public class PessoaFisica extends Pessoa implements Serializable{

    private static final long serialVersionUID = 1L;
    String cpf;
    int idade;

    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);
        System.out.println("Idade: " + idade +"\n");
        }

    public String getCpf() {
        return cpf;
        }

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

        public int getIdade() {
        return idade;
```

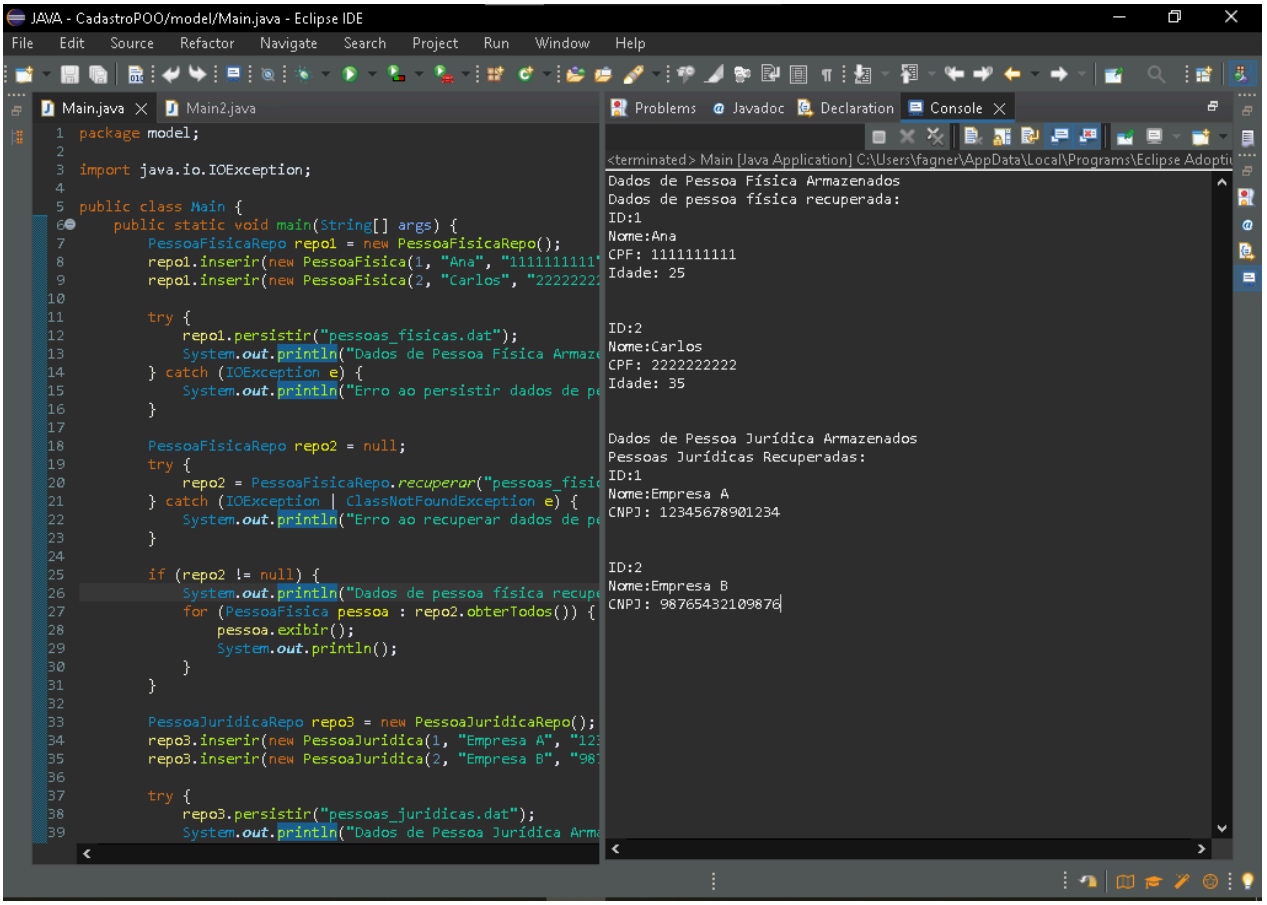
```
        }

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


package model;

import java.io.Serializable;

public class PessoaJuridica extends Pessoa implements Serializable{

   private static final long serialVersionUID = 1L;
   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+"\n");
        }

   public String getcnpj() {
        return cnpj;
        }

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


package model;

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

public class PessoaFisicaRepo implements Serializable {

   private static final long serialVersionUID = 1L;
   private List<PessoaFisica> pessoasFisicas = new ArrayList<>();
```

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

        public void alterar(PessoaFisica pessoa) {
        int index = obterIndexPorId(pessoa.getId());
        if (index != -1) {
        pessoasFisicas.set(index, pessoa);
        }
        }

        public void excluir(int id) {
        PessoaFisica pessoa = obter(id);
        if (pessoa != null) {
        pessoasFisicas.remove(pessoa);
        }
        }

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

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

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

        public static PessoaFisicaRepo recuperar(String nomeArquivo) throws IOException,
ClassNotFoundException {
        try (ObjectInputStream inputStream = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
        return (PessoaFisicaRepo) inputStream.readObject();
        }
        }

        private int obterIndexPorId(int id) {
```

```
        for (int i = 0; i < pessoasFisicas.size(); i++) {
        if (pessoasFisicas.get(i).getId() == id) {
        return i;
        }
        }
        return -1;
        }
}


package model;

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

public class PessoaJuridicaRepo implements Serializable {

   private static final long serialVersionUID = 1L;
   private List<PessoaJuridica> pessoasJuridicas = new ArrayList<>();

        public void inserir(PessoaJuridica pessoa) {
        pessoasJuridicas.add(pessoa);
        }

        public void alterar(PessoaJuridica pessoa) {
        int index = obterIndexPorId(pessoa.getId());
        if (index != -1) {
        pessoasJuridicas.set(index, pessoa);
        }
        }

        public void excluir(int id) {
        PessoaJuridica pessoa = obter(id);
        if (pessoa != null) {
        pessoasJuridicas.remove(pessoa);
        }
        }

        public PessoaJuridica obter(int id) {
        for (PessoaJuridica pessoa : pessoasJuridicas) {
        if (pessoa.getId() == id) {
        return pessoa;
        }
        }
        return null;
        }
```

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

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

	public static PessoaJuridicaRepo recuperar(String nomeArquivo) throws IOException,
ClassNotFoundException {
	try (ObjectInputStream inputStream = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
	return (PessoaJuridicaRepo) inputStream.readObject();
	}
	}

	private int obterIndexPorId(int id) {
	for (int i = 0; i < pessoasJuridicas.size(); i++) {
	if (pessoasJuridicas.get(i).getId() == id) {
	return i;
	}
	}
	return -1;
	}
}

package model;

import java.io.IOException;

public class Main {
	public static void main(String[] args) {
	PessoaFisicaRepo repo1 = new PessoaFisicaRepo();
	repo1.inserir(new PessoaFisica(1, "Ana", "1111111111", 25));
	repo1.inserir(new PessoaFisica(2, "Carlos", "2222222222", 35));

	try {
	repo1.persistir("pessoas_fisicas.dat");
	System.out.println("Dados de Pessoa Física Armazenados");
	} catch (IOException e) {
	System.out.println("Erro ao persistir dados de pessoas físicas: " + e.getMessage());
	}

	PessoaFisicaRepo repo2 = null;
	try {
```

```
        repo2 = PessoaFisicaRepo.recuperar("pessoas_fisicas.dat");
        } catch (IOException | ClassNotFoundException e) {
        System.out.println("Erro ao recuperar dados de pessoas físicas: " + e.getMessage());
        }

        if (repo2 != null) {
        System.out.println("Dados de pessoa física recuperada:");
        for (PessoaFisica pessoa : repo2.obterTodos()) {
        pessoa.exibir();
        System.out.println();
        }
        }

        PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
        repo3.inserir(new PessoaJuridica(1, "Empresa A", "12345678901234"));
        repo3.inserir(new PessoaJuridica(2, "Empresa B", "98765432109876"));

        try {
        repo3.persistir("pessoas_juridicas.dat");
        System.out.println("Dados de Pessoa Jurídica Armazenados");
        } catch (IOException e) {
        System.out.println("Erro ao persistir dados de pessoas jurídicas: " + e.getMessage());
        }

        PessoaJuridicaRepo repo4 = null;
        try {
        repo4 = PessoaJuridicaRepo.recuperar("pessoas_juridicas.dat");
        } catch (IOException | ClassNotFoundException e) {
        System.out.println("Erro ao recuperar dados de pessoas jurídicas: " + e.getMessage());
        }

        if (repo4 != null) {
        System.out.println("Pessoas Jurídicas Recuperadas:");
        for (PessoaJuridica pessoa : repo4.obterTodos()) {
        pessoa.exibir();
        System.out.println();
        }
        }
        }
}
```



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

Vantagens:

- **Reutilização de código:** Com a herança, você pode criar uma nova classe que é baseada em uma classe existente. Isso permite a reutilização do código já escrito na classe base.
- **Extensibilidade:** A herança permite estender o comportamento de uma classe existente, adicionando novos métodos e propriedades à subclasse.
- **Polimorfismo:** Classes derivadas podem ser tratadas como a classe base em determinadas situações, o que facilita o polimorfismo e a flexibilidade do código.

Desvantagens:

- **Acoplamento forte:** Muita herança pode levar a um acoplamento forte entre classes, o que torna o código mais difícil de entender e manter.
- **Herança múltipla não suportada:** Algumas linguagens, como Java, não suportam a herança múltipla de classes. Isso pode limitar a flexibilidade do design.
- **Complexidade:** Uma hierarquia de classes muito profunda pode levar à complexidade do código e dificultar a compreensão.

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

A interface Serializable em Java é necessária para permitir que objetos Java sejam convertidos em uma sequência de bytes, que podem ser gravados em um arquivo binário. Aqui estão algumas razões:

- **Persistência de Objetos:** A interface Serializable em Java é usada para persistir objetos em um fluxo de bytes. Quando uma classe implementa a interface Serializable, os objetos dessa classe podem ser salvos em arquivos binários.
- **Comunicação de Rede:** A serialização é importante quando se trata de enviar objetos pela rede. Uma vez que os objetos são serializados, podem ser transferidos pela rede e recriados em outra máquina.
- **Armazenamento de Objetos:** A serialização permite que os objetos sejam armazenados em arquivos, para que possam ser recuperados posteriormente.

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

A API Stream no Java aproveita os conceitos do paradigma funcional para fornecer um conjunto de operações para processamento de coleções. Aqui estão algumas características-chave:

- **Imutabilidade:** Os streams são imutáveis. Quando uma operação é realizada em um stream, ela não altera o stream original, mas cria um novo stream com o resultado da operação.
- **Operações de Alta Ordem:** As operações de stream são de alta ordem, o que significa que podem receber funções como parâmetros e retornar funções.
- **Lazy Evaluation:** As operações em um stream são avaliadas somente quando necessário. Isso permite uma maior eficiência e permite trabalhar com conjuntos de dados muito grandes.
- **Expressividade:** As operações da API Stream são expressivas e permitem escrever código mais limpo e conciso.

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

Em Java, para a persistência de dados em arquivos, é comum adotar o padrão de desenvolvimento DAO (Data Access Object). O padrão DAO é utilizado para isolar a lógica de acesso aos dados do restante da aplicação, fornecendo uma abstração para a manipulação de dados. Aqui estão alguns pontos sobre o padrão DAO:

- **Separação de Responsabilidades:** O padrão DAO separa a lógica de acesso aos dados do restante da aplicação. Isso permite que as classes de negócios se concentrem em lógica de negócios específica, enquanto as classes DAO cuidam da interação com o banco de dados ou, no seu caso, com os arquivos.
- **Reutilização:** Como o acesso aos dados é encapsulado nas classes DAO, elas podem ser reutilizadas em toda a aplicação. Isso promove a coesão e a reutilização do código.
- **Manutenção Simples:** Com o DAO, é mais fácil manter e atualizar o código, pois as mudanças na lógica de acesso aos dados são isoladas das outras partes do sistema.