Desenvolvimento FullStack
Missão Prática Nível 1 - Mundo3

Darlyson Cavalcanti Nery de Souza

Informações

Curso

Curso: Estácio Desenvolvimento Full Stack

Universidade: Estácio de Sá - Campus São Lourenço da Mata

Tutora: Maria Manso

Período: 3º

Turma: 9001

Matéria: Iniciando o caminho pelo Java

Aluno

Nome: Darlyson Cavalcanti Nery de Souza

Matrícula: 202301453471

Data: 18/04/2024

GitHub: https://github.com/DarlysonCavalcanti/DesenvolvimentoFullStack-MissaoPratica-Nivel1-Mundo3

Projeto

Cadastro de Clientes em Java

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

1. Utilizar herança e polimorfismo na definição de entidades.

2. Utilizar persistência de objetos em arquivos binários.Implementar uma interface cadastral em modo texto. Utilizar o controle de exceções da plataforma Java.

3. Implementar um sistema cadastral em Java, utilizando os recursos da programação orientada a objetos e a persistência em arquivos binários.

Procedimento de Número 1 - Criação das Entidades e Sistema de

Persistência

Códigos

App.java

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

public class App {
 public static void main(String[] args) {
 try {
 PessoaFisicaRepo repo1 = new PessoaFisicaRepo();
 repo1.inserir(new PessoaFisica(1, "Ana", "11111111111", 25)); repo1.inserir(new PessoaFisica(2,
"Carlos", "22222222222", 52)); repo1.persistir("pessoasFisicas.dat");
 System.out.println("Dados de Pessoa Fisica Armazenados.");

 PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
 repo2.recuperar("pessoasFisicas.dat");
 System.out.println("Dados de Pessoa Fisica Recuperados."); for (PessoaFisica pf :
repo2.obterTodos()) {
 pf.exibir();
 }

 PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
 repo3.inserir(new PessoaJuridica(3, "XPTO Sales", "33333333333333")); repo3.inserir(new PessoaJuridica(4,
"XPTO Solutions", "44444444444444")); repo3.persistir("pessoasJuridicas.dat");
 System.out.println("Dados de Pessoa Juridica Armazenados.");

 PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
 repo4.recuperar("pessoasJuridicas.dat");
 System.out.println("Dados de Pessoa Juridica Recuperados."); for (PessoaJuridica pj :
repo4.obterTodos()) {
 pj.exibir();
 }
 } catch (Exception e) {
 e.printStackTrace();
 }
 }
}
```

Pessoa.java

```
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 void exibir() {
System.out.println("ID: " + id + ", 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;
}
}
```

PessoaFisica.java

```
package model;

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

PessoaJuridica.Java

```
package model;

public class PessoaJuridica extends Pessoa {
 private String cnpj;

 public PessoaJuridica() {
 }

 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;
}
}
```

PessoaFisicaRepo.java

```
package model;


import java.util.ArrayList;
import java.io.*;
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) {
for (int i = 0; i < pessoasFisicas.size(); i++) {
if (pessoasFisicas.get(i).getId() == pessoaFisica.getId()) {
pessoasFisicas.set(i, pessoaFisica);
return;
}
}
}

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

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

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

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

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

PessoaJuridicaRepo.java

```
package model;

import java.util.ArrayList;
import java.io.*;
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) {
for (int i = 0; i < pessoasFisicas.size(); i++) {
if (pessoasFisicas.get(i).getId() == pessoaFisica.getId()) {
pessoasFisicas.set(i, pessoaFisica);
return;
}
}
}

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

}

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

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

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

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

Execuções

```
PS C:\Users\darly\OneDrive\Área de Trabalho\Missao Pratica-Nivel1- Iniciando o Caminho Pelo Java-\Procedimento de Numero 1> & 'C:\Program
Files\Java\jdk-22\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp'
'C:\Users\darly\AppData\Roaming\Code\User\workspaceStorage\02c667aaaf0420e5050b2cec3aca34a2\redhat.java\jdt_ws\Procedimento de Numero
1_af911361\bin' 'App'
Dados de Pessoa Fisica Armazenados.
Dados de Pessoa Fisica Recuperados.
ID: 1, Nome: Ana
CPF: 11111111111, Idade: 25
ID: 2, Nome: Carlos
CPF: 22222222222, Idade: 52
Dados de Pessoa Juridica Armazenados.
Dados de Pessoa Juridica Recuperados.
ID: 3, Nome: XPTO Sales
CNPJ: 33333333333333
ID: 4, Nome: XPTO Solutions
CNPJ: 44444444444444
PS C:\Users\darly\OneDrive\Área de Trabalho\Missao Pratica-Nivel1- Iniciando o Caminho Pelo Java-\Procedimento de Numero 1>
```

Análise e Conclusões

Quais as vantagens e desvantagens do uso de herança?

Vantagens:

Uma das grandes vantagens de usar o recurso da herança é na reutilização do código. Esse reaproveitamento pode ser acionado quando se identifica que o atributo ou método de uma classe será igual para as outras.

Facilita a organização e manutenção do código

Desvantagens

Java não suporta herança múltipla, o que limita o uso de uma única superclasse por subclasse

Em Sistemas mais complexos é difícil de modificar a estrutura

Por que a interface Serializable é necessária ao efetuar persistência em arquivos binários? A serialização é uma forma de avisar a JVM que aquela informações podem ser salvam em binário e assim convertendo o estado do objeto em um fluxo de bytes, que podem ser armazenados em um arquivo.

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

Essa fuga do paradigma do POO em Java na API Stream é utilizado para declarar de forma mais pratica toda manipulação de dados sequencial, como filter, reduce, map. Permitindo uma performance maior, sem complexidade e sem modificar os dados dos objetos originais, agindo em paralelo ao funcionamento padrão do programa.
No entanto ela ainda opera sobre objetos e classes, processando os dados de maneira funcional dentro da orientação por objetos de Java.

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

A persistência de dados em arquivos no desenvolvimento com Java é o DAO.