

Curso - Desenvolvimento full Stack Disciplina – Iniciando o caminho pelo Java

Thiago dos Santos Moreira Matrícula – 2022 0804 2279

Campus - Polo Betânia BH – MG

Mundo 3 Período 2023.2

Criação das entidades e sistemas de persistência

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

Link do repositório no Github: <https://github.com/Thiagomoreira97/MP-java1>

Vantagens e desvantagens do uso de herança:

Vantagens:

1. Reutilização de código: A herança permite que você crie classes derivadas (subclasses) a partir de classes existentes (superclasses), reutilizando os atributos e métodos da classe pai.
2. Organização hierárquica: A herança ajuda a modelar relações hierárquicas entre classes, refletindo a estrutura da realidade de forma mais clara e intuitiva.
3. Polimorfismo: A herança permite que objetos de subclasses sejam tratados como objetos da classe pai, facilitando a substituição de objetos em diferentes contextos.

Desvantagens:

1. Acoplamento forte: A herança cria um acoplamento forte entre as classes pai e filhas. Mudanças na classe pai podem afetar as subclasses e vice-versa.
2. Herança múltipla problemática: Algumas linguagens, incluindo Java, não suportam herança múltipla direta, o que pode limitar a reutilização de código em situações complexas.
3. Dificuldade de manutenção: Uma hierarquia de classes muito profunda pode se tornar difícil de entender e manter.

Interface Serializable e persistência em arquivos binários:

A interface `Serializable` é necessária ao efetuar persistência em arquivos binários porque permite que objetos sejam serializados em uma sequência de bytes, que pode ser escrita em um arquivo e posteriormente lida e desserializada. Isso é essencial para salvar objetos em disco ou transferi-los

pela rede. A interface `Serializable` marca a classe como apta a ser serializada, e ao implementá-la, a classe define como seus campos devem ser convertidos em bytes.

Paradigma funcional na API Stream do Java:

A API Stream do Java utiliza o paradigma funcional fornecendo operações de alto nível para processar coleções de dados de forma declarativa. Em vez de usar loops explícitos, você pode usar operações como `map`, `filter`, `reduce` e outras para processar e transformar os elementos de uma coleção. Isso promove um código mais conciso, legível e expressivo, além de permitir paralelismo mais eficiente e otimização por parte da JVM.

Padrão de desenvolvimento para persistência de dados em arquivos:

No contexto do Java, um padrão comum de desenvolvimento para persistência de dados em arquivos é o uso de classes de entrada e saída (I/O) para ler e gravar dados em arquivos. O padrão envolve a criação de objetos `FileInputStream` ou `FileOutputStream` para leitura e gravação de bytes, e também pode envolver o uso de classes `ObjectInputStream` e `ObjectOutputStream` para a serialização e desserialização de objetos. Em conjunto com a interface `Serializable`, essa abordagem é usada para persistir e recuperar objetos em formato binário. Além disso, em cenários mais complexos, é comum usar padrões como o padrão DAO (Data Access Object) para abstrair a camada de persistência e facilitar a manutenção do código.

Códigos usados nessa aplicação:

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);
    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;
}
}
```

PessoaFisica.java:

```
package model;
```

```
import java.io.Serializable;
```

```
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);  
        System.out.println("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.io.*;
import java.util.ArrayList;
import java.util.List;

public class PessoaFisicaRepo {
    private List<PessoaFisica> pessoasFisicas;
    private final String arquivo;

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

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

    public void alterar(PessoaFisica pessoaFisicaNova) {
        for (int i = 0; i < pessoasFisicas.size(); i++) {
            PessoaFisica pessoaFisica = pessoasFisicas.get(i);
```

```
        if (pessoaFisica.getId() == pessoaFisicaNova.getId()) {  
            pessoasFisicas.set(i, pessoaFisicaNova);  
            break;  
        }  
    }  
}
```

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

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

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

```
public void persistir() throws IOException {  
    try (ObjectOutputStream outputStream = new ObjectOutputStream(new  
FileOutputStream(arquivo))) {  
        outputStream.writeObject(pessoasFisicas);  
    }  
}
```

```
public void recuperar() throws IOException, ClassNotFoundException {
```

```

        try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(arquivo))) {
            pessoasFisicas = (List<PessoaFisica>) inputStream.readObject();
        }
    }
}

```

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;
    }
}
```

PessoaJuridicaRepo.java:

```
package model;

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

public class PessoaJuridicaRepo {
    private List<PessoaJuridica> pessoasJuridicas;
    private final String arquivo;

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

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

    public void alterar(PessoaJuridica pessoaJuridicaNova) {
        for (int i = 0; i < pessoasJuridicas.size(); i++) {
            PessoaJuridica pessoaJuridica = pessoasJuridicas.get(i);
            if (pessoaJuridica.getId() == pessoaJuridicaNova.getId()) {
                pessoasJuridicas.set(i, pessoaJuridicaNova);
                break;
            }
        }
    }
}
```



```
    }  
}  
}
```

```
public void excluir(int id) {  
    pessoasJuridicas.removeIf(pessoaJuridica -> pessoaJuridica.getId() == id);  
}
```

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

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

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

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

```
}  
}
```

Resultado do Código:

run:

Pessoas Fisicas recuperadas:

ID: 1

Nome: João

CPF: 12345678901

Idade: 30

ID: 2

Nome: Maria

CPF: 98765432101

Idade: 25

Pessoas Juridicas recuperadas:

ID: 1

Nome: Empresa A

CNPJ: 12345678901234

ID: 2

Nome: Empresa B

CNPJ: 98765432109876

CONSTRUÍDO COM SUCESSO (tempo total: 2 segundos)



Curso - Desenvolvimento full Stack Disciplina – Iniciando o caminho pelo Java

Thiago dos Santos Moreira Matrícula – 2022 0804 2279

Campus - Polo Betânia BH – MG

Mundo 3 Período 2023.2

Criação das entidades e sistemas de persistência

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

Link do repositório no Github: <https://github.com/Thiagomoreira97/MP-java1>

Análise e Conclusão:

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

Elementos estáticos em Java são membros de uma classe que pertencem à classe em si, não a instâncias específicas da classe. O método `main` é definido como estático porque precisa ser chamado diretamente na classe antes de criar instâncias, permitindo que o programa comece a ser executado sem criar objetos da classe principal. Isso torna o método `main` acessível desde o início da execução do programa.

Para que serve a classe Scanner?

A classe Scanner em Java é usada para ler dados de entrada, como texto digitado pelo usuário no teclado ou informações de arquivos. Ela é frequentemente utilizada para permitir a interação do usuário com programas, facilitando a leitura e análise de dados formatados. Em resumo, o Scanner é uma ferramenta importante para a entrada de dados interativa em programas Java.

Como o uso de classes de repositório impactou na organização do código?

O uso de classes de repositório em um projeto Java contribui para a organização eficiente do código, promovendo a separação nítida da lógica de acesso a dados. Isso resulta em uma melhor reutilização de código, abstração da camada de armazenamento, maior testabilidade, clareza e manutenção do código, além da capacidade de organizar hierarquicamente as operações de dados para refletir a estrutura de dados do projeto, simplificando o desenvolvimento e a escalabilidade do software.

Código usado na classe principal:

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
 */
package cadastropoo;

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

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

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws Exception {
        // TODO code application logic here
        PessoaFisicaRepo repo1 = new PessoaFisicaRepo();
    }
}
```

```
repo1.inserir(new PessoaFisica("00111122200", 35, 001, "Maria Luiza da Silva"));
repo1.inserir(new PessoaFisica("00122233301", 40, 002, "José da Silva Gonçalves"));
```

```
repo1.persistir("pessoa-fisicaRepo1");
```

```
PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
repo2.recuperar("pessoa-fisicaRepo1");
```

```
PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
repo3.inserir(new PessoaJuridica("00111222/0001-99", 003,"Jose enterprise"));
repo3.inserir(new PessoaJuridica("00222333/0001-88", 004,"Maria enterprise"));
repo3.persistir("pessoa-juridicaRepo3");
```

```
PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
repo4.recuperar("pessoa-juridicaRepo3");
```

```
}
```

```
}
```

Resultado da classe:

run:

- 1 - Incluir pessoa
- 2 - Alterar Pessoa
- 3 - Excluir Pessoa
- 4 - Buscar pelo ID
- 5 - Exibir Todos

6 - Persistir Datos

7 - Recuperar Datos

8 - Finalizar Programa
