



Desenvolvimento Full Stack

Aluno: Victor de A. Costa

Missão Prática | Nível 1 | Mundo 3 - 2024.3
INICIANDO O CAMINHO PELO JAVA

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

1º Procedimento | Criação das Entidades e Sistema de Persistência

---> CÓDIGOS UTILIZADOS

GIT: github.com/VictorDSGNR/missao_mundo03_nivel01

CadastroPOO.java

```
package model;
import java.io.IOException;
/**
 *
 * @author victoracosta
 */
public class CadastroPOO {
    public static void main(String[] args) {
        PessoaFisicaRepo repo1 = new PessoaFisicaRepo();
        PessoaFisica pessoa1 = new PessoaFisica(01, "Victor", "123453175-98", 37);
        PessoaFisica pessoa2 = new PessoaFisica(02, "Yumi", "125443175-68", 32);
        repo1.inserir(pessoa1);
        repo1.inserir(pessoa2);
        System.out.println("Dados de Pessoa Física Armazenados.");
        try {
            repo1.persistir("pessoasFisicas.dat");
        } catch (IOException e) {
            e.printStackTrace();
        }
        PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
        try {
            repo2.recuperar("pessoasFisicas.dat");
            System.out.println("Dados de Pessoa Física Recuperados.");
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
        for (PessoaFisica pf : repo2.obterTodos()) {
            pf.exibir();
        }
        PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
        PessoaJuridica pessoa3 = new PessoaJuridica(3, "CREATEE BRASIL", "12.345.678/0001-10");
        PessoaJuridica pessoa4 = new PessoaJuridica(4, "CREATEE STORE", "12.346.978/0001-11");
        repo3.inserir(pessoa3);
        repo3.inserir(pessoa4);
        System.out.println("Dados de Pessoa Jurídica Armazenados.");
        try {
            repo3.persistir("pessoasJuridicas.dat");
        } catch (IOException e) {
            e.printStackTrace();
        }
        PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
        try {
            repo4.recuperar("pessoasJuridicas.dat");
            System.out.println("Dados de Pessoa Jurídica Recuperados.");
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
        for (PessoaJuridica pf : repo4.obterTodos()) {
            pf.exibir();
        }
    }
}
```

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);
    }
    // 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;
    }
}
```

PessoaFisica.java

```
package model;
import java.io.Serializable;

public class PessoaFisica extends Pessoa implements Serializable {
    private String cpf;
    private int idade;
    public PessoaFisica() {
        super();
    }
    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);
    }
}
```

PessoaFisicaRepo.java

```
package model;
import java.io.*;
import java.util.ArrayList;
import java.util.List;

public class PessoaFisicaRepo {
    private List<PessoaFisica> pessoasFisicas = new ArrayList<>();
    public void inserir(PessoaFisica pessoaFisica) {
        pessoasFisicas.add(pessoaFisica);
    }
    public void alterar(PessoaFisica pessoaFisica) {
    }
    public void excluir(int id) {
    }
    public PessoaFisica obter(int id) {
        return null;
    }
    public List<PessoaFisica> obterTodos() {
        return new ArrayList<>(pessoasFisicas);
    }
    public void persistir(String fileName) throws IOException {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(fileName))) {
            oos.writeObject(pessoasFisicas);
        }
    }
    public void recuperar(String fileName) throws IOException, ClassNotFoundException {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(fileName))) {
            pessoasFisicas = (List<PessoaFisica>) ois.readObject();
        }
    }
}
```

PessoaJuridica.java

```
package model;
import java.io.Serializable;

public class PessoaJuridica extends Pessoa implements Serializable {
    private String cnpj;
    public PessoaJuridica() {
        super();
    }
    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);
    }
}
```

PessoaJuridicaRepo.java

```
package model;
import java.io.*;
import java.util.ArrayList;
import java.util.List;

public class PessoaJuridicaRepo {
    private List<PessoaJuridica> pessoasJuridicas = new ArrayList<>();
    public void inserir(PessoaJuridica pessoaJuridica) {
        pessoasJuridicas.add(pessoaJuridica);
    }
    public void alterar(PessoaJuridica pessoaJuridica) {
    }
    public void excluir(int id) {
    }
    public PessoaJuridica obter(int id) {
        return null;
    }
    public List<PessoaJuridica> obterTodos() {
        return new ArrayList<>(pessoasJuridicas);
    }
    public void persistir(String fileName) throws IOException {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(fileName))) {
            oos.writeObject(pessoasJuridicas);
        }
    }
    public void recuperar(String fileName) throws IOException, ClassNotFoundException {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(fileName))) {
            pessoasJuridicas = (List<PessoaJuridica>) ois.readObject();
        }
    }
}
```

---> RESULTADO



The screenshot shows the Eclipse IDE's Output window titled "Output - CadastroPOO (run)". The window displays the execution results of the application. It starts with the message "run:" followed by "Dados de Pessoa Física Armazenados." and "Dados de Pessoa Física Recuperados.". Below these, two sets of data for physical persons are listed, each with Id, Nome, and CPF. Then, it shows "Dados de Pessoa Jurídica Armazenados." and "Dados de Pessoa Jurídica Recuperados.", followed by two entries for legal entities with Id, Nome, and CNPJ. The output concludes with "BUILD SUCCESSFUL (total time: 0 seconds)".

```
run:
Dados de Pessoa Física Armazenados.
Dados de Pessoa Física Recuperados.
Id: 1
Nome: Victor
CPF: 123453175-98
Idade: 37
Id: 2
Nome: Yumi
CPF: 125443175-68
Idade: 32
Dados de Pessoa Jurídica Armazenados.
Dados de Pessoa Jurídica Recuperados.
Id: 3
Nome: CREATEE BRASIL
CNPJ: 12.345.678/0001-10
Id: 4
Nome: CREATEE STORE
CNPJ: 12.346.978/0001-11
BUILD SUCCESSFUL (total time: 0 seconds)
```

---> ANÁLISE E CONCLUSÃO

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

Algumas vantagens seriam: a reutilização do código, maior coerência e consistência, melhor abstração e generalização e a extensibilidade, facilitando a extensão de uma classe existente para criar novas classes.

Já, algumas desvantagens seriam: acoplamento rígido, dificultando a substituição das classes pai por outras que não sejam diretamente compatíveis, problemas de manutenção e escalabilidade, risco de quebra do princípio de substituição de Liskov, complexidade adicional.

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

Serialização seria o armazenamento do estado atual dos objetos em arquivos em formato binário para o seu computador, e isso ocorre através do uso da interface Serializable, fazendo com que esse estado (no momento da sua serialização) possa ser recuperado posteriormente recriando o objeto em memória.

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

As expressões lambda permitem que você passe comportamentos (funções) como argumentos para métodos. Isso é fundamental para a API de Streams, pois permite operações como filtragem, mapeamento e redução de forma concisa e legível.

A API de Streams também oferece operações de alto nível que abstraem o controle de fluxo e loops. Por exemplo, métodos como filter, map e reduce permitem que você descreva o que deseja fazer com os dados,

As Streams também são imutáveis, o que significa que cada operação em uma Stream retorna uma nova Stream. Isso é um princípio fundamental da programação funcional, que ajuda a evitar efeitos colaterais e torna o código mais previsível.

Facilitando o processamento paralelo de dados. Com o método parallelStream, é possível paralelizar operações de forma simples, aproveitando melhor os recursos do sistema.

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

Ao trabalhar com Java e a necessidade de reter dados em arquivos, vários padrões e técnicas são adotados, dependendo do contexto e dos requisitos específicos do projeto. Mas trabalhando com dados em arquivos, podem ser citadas: o uso de bibliotecas como Jackson ou Gson para manipulação de arquivos JSON, ou Apache Commons CSV para arquivos CSV. Essas bibliotecas ajudam a serializar e desserializar objetos Java para formatos de arquivo comuns.