Campus Londrina - Desenvolvimento Full Stack
Disciplina: Iniciando o Caminho Pelo Java
Turma 9001, 3o Semestre

Aluna: Gabriela Garcia Mortean
GitHub: https://github.com/gabimortean/Mundo3-Nivel1.git

1. **Título da Prática: 1º Procedimento | Criação das Entidades e Sistema de Persistência**

2. **Objetivo da Prática:** Utilizar herança e polimorfismo na definição de entidades; utilizar persistência de objetos em arquivos binários; implementar uma interface cadastral em modo texto; utilizar o controle de exceções da plataforma Java.

Estrutura de diretórios do projeto "CadastroPOO":
CadastroPOO > Source Packages > model > Pessoa.java
CadastroPOO > Source Packages > model > PessoaFisica.java
CadastroPOO > Source Packages > model > PessoaJuridica.java
CadastroPOO > Source Packages > model > PessoaFisicaRepo.java
CadastroPOO > Source Packages > model > PessoaJuridicaRepo.java
CadastroPOO > Source Packages > MainClass.java

3. **Todos os códigos solicitados neste roteiro de aula:**

Código 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;
    }
}
```

Código PessoaFisica.java:

```
package model;

import java.io.Serializable;

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

Código PessoaJuridica.java:

```
package model;

import java.io.Serializable;

public class PessoaJuridica extends Pessoa implements Serializable {
    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;
    }
}
```

Código PessoaFisicaRepo.java:

```
package model;

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

public class PessoaFisicaRepo {
```

```
    private ArrayList<PessoaFisica> pessoasFisicas = new ArrayList<>();

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

    public void alterar(int id, PessoaFisica pessoa) {
        for (int i = 0; i < pessoasFisicas.size(); i++) {
            if (pessoasFisicas.get(i).getId() == id) {
                pessoasFisicas.set(i, pessoa);
                return;
            }
        }
    }

    public void excluir(int id) {
        for (int i = 0; i < pessoasFisicas.size(); i++) {
            if (pessoasFisicas.get(i).getId() == id) {
                pessoasFisicas.remove(i);
                return;
            }
        }
    }

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

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

Código PessoaJuridicaRepo.java:

```
package model;

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

public class PessoaJuridicaRepo {
   private ArrayList<PessoaJuridica> pessoasJuridicas = new ArrayList<>();

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

   public void alterar(int id, PessoaJuridica pessoa) {
      for (int i = 0; i < pessoasJuridicas.size(); i++) {
         if (pessoasJuridicas.get(i).getId() == id) {
            pessoasJuridicas.set(i, pessoa);
            return;
         }
      }
   }

   public void excluir(int id) {
      for (int i = 0; i < pessoasJuridicas.size(); i++) {
         if (pessoasJuridicas.get(i).getId() == id) {
            pessoasJuridicas.remove(i);
            return;
         }
      }
   }

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

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

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

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

Código MainClass.java:

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

import java.io.IOException;

public class MainClass {
    public static void main(String[] args) {
        PessoaFisicaRepo repo1 = new PessoaFisicaRepo();

        PessoaFisica pessoaFisica1 = new PessoaFisica(1, "Carlos", "11111111111", 30);
        PessoaFisica pessoaFisica2 = new PessoaFisica(2, "Ana", "22222222222", 25);
        repo1.inserir(pessoaFisica1);
        repo1.inserir(pessoaFisica2);

        try {
            repo1.persistir("pessoasfisicas.dat");

            PessoaFisicaRepo repo2 = new PessoaFisicaRepo();

            repo2.recuperar("pessoasfisicas.dat");

            System.out.println("Pessoas Físicas Recuperadas:");
            for (PessoaFisica pessoa : repo2.obterTodos()) {
                pessoa.exibir();
            }

        } catch (IOException | ClassNotFoundException e) {
        }

        PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();

        PessoaJuridica pessoaJuridica1 = new PessoaJuridica(3, "XPTO Sales", "33333333333333");
        PessoaJuridica pessoaJuridica2 = new PessoaJuridica(4, "XPTO Solutions", "44444444444444");
        repo3.inserir(pessoaJuridica1);
        repo3.inserir(pessoaJuridica2);

        try {
            repo3.persistir("pessoasjuridicas.dat");

            PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
```

```
            repo4.recuperar("pessoasjuridicas.dat");

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

        } catch (IOException | ClassNotFoundException e) {
        }
    }
}
```

4. **Os resultados da execução dos códigos:**

```
run:
Pessoas F�sicas Recuperadas:
ID: 1
Nome: Carlos
CPF: 11111111111
Idade: 30
ID: 2
Nome: Ana
CPF: 22222222222
Idade: 25
Pessoas Jur�dicas Recuperadas:
ID: 3
Nome: XPTO Sales
CNPJ: 33333333333333
ID: 4
Nome: XPTO Solutions
CNPJ: 44444444444444
BUILD SUCCESSFUL (total time: 0 seconds)
```

5. **Análise e Conclusão:**
    a. **Quais as vantagens e desvantagens do uso de herança?**

Vantagens:
- Evita a duplicação de código: O uso da herança permite que uma classe herde atributos e métodos de outra classe, o código de uma classe pai em classes filhas. Isso leva a um código mais limpo e mais eficiente.
- Facilita a implementação do polimorfismo: Permite que objetos de classes diferentes sejam tratados de maneira uniforme. Isso é fundamental para a flexibilidade e a extensibilidade do código.
- Ajuda a organizar e estruturar o código: Cria hierarquias de classes que representam conexões entre objetos.

- Facilita a manutenção: Alterações feitas na classe pai afetam automaticamente todas as classes filhas. Isso pode simplificar a manutenção do código.

Desvantagens:
- Componentes fortemente interligados: As alterações na classe pai podem afetar todas as classes filhas, o que pode tornar o código mais frágil.
- Pode dificultar a substituição de uma classe específica: Cria uma forte dependência entre classes pai e filhas.
- Uso excessivo da Herança pode ser prejudicial: Pode tornar o código difícil de entender e manter.

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

A interface Serializable é essencial para a persistência em arquivos binários em Java, pois permite que objetos sejam transformados em uma sequência de bytes que pode ser armazenada de forma permanente. Isso é fundamental para gravar objetos em arquivos, transmiti-los pela rede ou armazená-los em bancos de dados, tornando possível recriar esses objetos posteriormente. A serialização é a base da persistência de dados em Java e desempenha um papel crucial em aplicativos que precisam manter o estado de objetos entre execuções, compartilhá-los com outras partes do código ou entre diferentes aplicativos. Portanto, a presença da interface Serializable é necessária para viabilizar a persistência de dados de forma eficaz em formato binário.

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

A API Stream no Java utiliza o paradigma funcional de várias maneiras. Ela permite encadear operações em uma sequência, como map, filter, reduce, e outras, para manipular os elementos da coleção de forma eficiente e declarativa. As operações Stream são imutáveis, o que evita alterações indesejadas nos dados originais. Além disso, as operações são avaliadas preguiçosamente, o que economiza tempo e recursos. A API Stream também oferece suporte ao processamento paralelo, melhorando o desempenho em sistemas multicore. É possível fornecer funções como argumentos, promovendo a reutilização de código, e a programação se torna mais declarativa, o que melhora a legibilidade do código. A API Stream aproveita os princípios do paradigma funcional para simplificar a manipulação de dados em Java, tornando o código mais eficiente e legível.

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

Em Java, um padrão adotado na persistência de dados em arquivos é o uso da serialização, que permite que objetos sejam armazenados em formato binário e recuperados posteriormente. A interface Serializable é usada para marcar classes como serializáveis, permitindo que os objetos sejam gravados e lidos de arquivos.