Disciplina:	RPG0014 – Iniciando o Caminho pelo Java
Nome:	Iggor Motta de Oliveira
Turma:	2024.3
Campus:	EAD
Curso:	Desenvolvimento Full stack
Link:	https://github.com/Iggor-motta/Miss-oPr-tica-N-vel-1-Parti1--Mundo-3.git

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

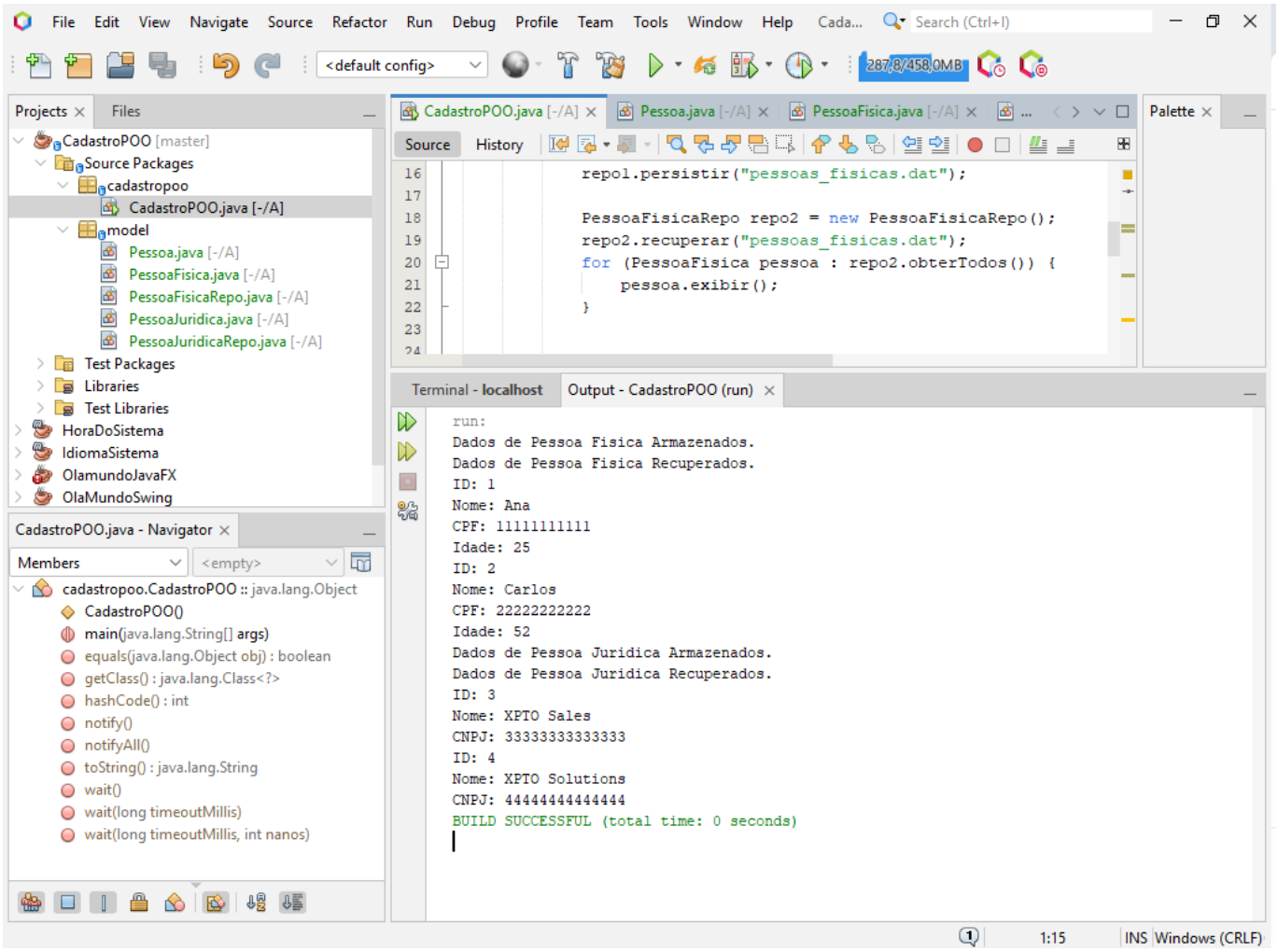
1. **Título da Prática**: “Procedimento | Criação das Entidades e Sistema de Persistência”.

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.
   - Utilizar os recursos da programação orientada a objetos e a persistência em arquivos binários.

3. **Todos os códigos solicitados neste roteiro serão anexados no final do relatório.**

4. Resultado da execução dos códigos:



5. Análise e Conclusão

- Quais as vantagens e desvantagens do uso de herança?

Uma das maiores vantagem e a fácil manutenção**,** pois com Alterações na classe base podem facilmente ser refletidas em todas as subclasses, ajudando a ser mais rápido na manutenção.
Herança permite que uma classe herde atributos e métodos de outra classe, promovendo a reutilização de código e evitando duplicação.

Desvantagens é a Fragilidade ao alterar a superclasse qualquer alteração na classe base podem gerar impactos inesperados nas subclasses.
Subclasses são fortemente acopladas à superclasse, o que pode dificultar a manutenção, especialmente se houver necessidade de mudanças na classe base.

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

A serialização é o processo de converter um objeto em uma sequência de bytes, que podem ser armazenados em um arquivo, transmitidos pela rede. Sem implementar Serializable, o Java não pode garantir que o estado interno de um objeto possa ser representado de forma segura em uma sequência de bytes.

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

Fornecendo uma forma funcional de processar coleções de dados de maneira declarativa. A API Stream permite passar funções como parâmetros.
No paradigma funcional, a ideia é que funções não devem modificar variáveis ou objetos fora de seu escopo. As operações com Stream são projetadas para minimizar ou eliminar efeitos colaterais.

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

No Java, o padrão adotado para persistência de dados em arquivos é o DAO (Data Access Object)**.** Esse padrão é usado para separar a lógica de negócios da lógica de persistência. Fornecendo métodos como inserir, alterar, excluir, obter, e persistir para abstrair o acesso aos arquivos binários que armazenam os objetos.

Todos os códigos solicitados neste roteiro esta no link abaixo:

https://github.com/Iggor-motta/Miss-oPr-tica-N-vel-1-Parti1--Mundo-3

Códigos:

Projeto CadastoPOO.java:

```
package cadastropoo;

import model.*;
import java.io.IOException;

public class CadastroPOO {

    public static void main(String[] args) {
        try {

            PessoaFisicaRepo repo1 = new PessoaFisicaRepo();
            System.out.println("Dados de Pessoa Fisica Armazenados.");
            System.out.println("Dados de Pessoa Fisica Recuperados.");
            repo1.inserir(new PessoaFisica(1, "Ana", "11111111111", 25));
            repo1.inserir(new PessoaFisica(2, "Carlos", "22222222222", 52));
            repo1.persistir("pessoas_fisicas.dat");

            PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
            repo2.recuperar("pessoas_fisicas.dat");
            for (PessoaFisica pessoa : repo2.obterTodos()) {
                pessoa.exibir();
            }


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

            PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
            repo4.recuperar("pessoas_juridicas.dat");
            for (PessoaJuridica pessoa : repo4.obterTodos()) {
                pessoa.exibir();
            }
```

```
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();

        }

    }}
```

Entidades do Pacote Model:

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

    }
```

```
}
```

PessoaFisica.java

```
package model;

public class PessoaFisica extends Pessoa {
    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;
    }

    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;
    }

    @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;

public PessoaFisicaRepo() {
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(pessoa -> pessoa.getId() == id);
}

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

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

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

```
@SuppressWarnings("unchecked")

public void recuperar(String nomeArquivo) throws IOException,
ClassNotFoundException {

try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {

pessoasFisicas = (List<PessoaFisica>) ois.readObject();

}

}

}
```

PessoaJuridica.java

```
package model;


public class PessoaJuridica extends Pessoa {

private String cnpj;


public PessoaJuridica() {

super();

}


public PessoaJuridica(int id, String nome, String cnpj) {

super(id, nome);

this.cnpj = cnpj;

 }
```

```
        public String getCnpj() {
        return cnpj;
        }

        public void setCnpj(String cnpj) {
         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;
```

```
public PessoaJuridicaRepo() {
    pessoasJuridicas = new ArrayList<>();
}

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

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

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

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

```
                return pessoaJuridica;
            }
        }
        return null;
    }

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

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

    @SuppressWarnings("unchecked")
    public void recuperar(String nomeArquivo) throws IOException,
ClassNotFoundException {
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
            pessoasJuridicas = (List<PessoaJuridica>) ois.readObject();
        }
    }
}
```