	<p align="center"><b>SOCIEDADE DE ENSINO SUPERIOR ESTÁCIO DE SÁ</b></p> <p align="center"><b>POLO SETOR IPIRANGA - APARECIDA DE GOIÂNIA – GO</b></p> <p align="center"><b>DESENVOLVIMENTO FULL STACK 2023.2 FLEX</b></p> <p align="center">Relatório da Missão Prática   Nível 1   Mundo 3</p>
Aluno:	Teovânio Santos Moreira
Tutor:	Rodrigo Dias
Repositório GIT	<a href="https://github.com/Teovanio/CadastroPoo">https://github.com/Teovanio/CadastroPoo</a>

## 1º Procedimento | Criação das Entidades e Sistema de Persistência

### 1. Título da Prática

RPG0014 - Iniciando o caminho pelo Java;

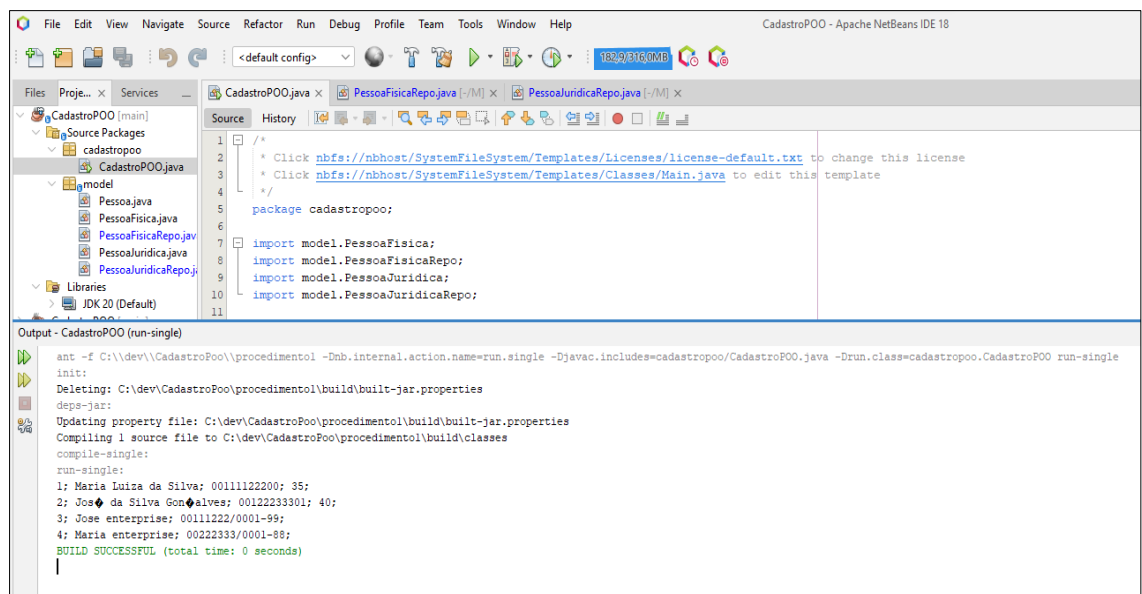
### 2. Objetivos da prática;

- 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.
- No final do projeto, o aluno terá implementado um sistema cadastral em Java, utilizando os recursos da programação orientada a objetos e a persistência em arquivos binários.

### 3. Códigos deste roteiro de aula;

<https://github.com/Teovanio/CadastroPoo>

### 4. Resultado da execução dos códigos;



The screenshot shows the Apache NetBeans IDE interface. The main editor displays the source code of `CadastroPoo.java`, which includes package declarations and imports for `model.PessoaFisica`, `model.PessoaFisicaRepo`, `model.PessoaJuridica`, and `model.PessoaJuridicaRepo`. The left sidebar shows the project structure with `CadastroPoo` as the main project, containing `cadastroPoo` and `model` packages. The bottom output window shows the command `ant -f C:\dev\CadastroPoo\procedimentol -Dnb.internal.action.name=run.single -Djavac.includes=cadastroPoo\CadastroPoo.java -Drun.class=cadastroPoo.CadastroPoo run-single` and its successful execution, displaying a list of test data and the message `BUILD SUCCESSFUL (total time: 0 seconds)`.

## 5. Análise e Conclusão:

### a. Quais as vantagens e desvantagens do uso de herança?

A herança é um dos conceitos fundamentais da programação orientada a objetos e é amplamente utilizada na linguagem Java. Ela permite que uma classe herde características (atributos e métodos) de outra classe, o que pode trazer vantagens e desvantagens para o design e a manutenção do código.

Como vantagens identificamos:

**Reutilização de código:** que permite que criemos classes baseadas em outras já existentes, o que significa que podemos aproveitar a funcionalidade já implementada nessas, evitando a duplicação de código.

**Flexibilidade:** a herança pode dar mais flexibilidade ao desenvolver nosso código, pois permite que criemos classes que se adaptem às nossas necessidades específicas.

**Polimorfismo:** que é a capacidade de tratar objetos de classes diferentes de maneira uniforme. Possibilitando criar uma lista de objetos de uma classe pai e adicionar objetos de suas classes filhas, o que permite tratar todos eles de forma polimórfica.

Temos como desvantagens os seguintes itens:

**Acoplamento rígido:** a herança cria um acoplamento entre a classe pai e suas classes filhas. Mudanças na classe pai podem afetar as classes filhas e vice-versa, tornando o código mais difícil de manter e modificar.

**Complexidade:** a herança pode tornar nosso código mais complexo, pois precisaremos entender as relações entre as classes base e derivadas.

Falta de suporte a **Herança múltipla**, Em Java, uma classe não pode herdar de múltiplas classes. Isso pode limitar a flexibilidade na organização do código e forçar o uso de interfaces em vez de herança, em alguns casos.

**Vício:** a herança pode tornar o código mais dependente de classes base, o que pode dificultar a mudança para novas classes base no futuro.

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

A interface `Serializable` é necessária ao efetuar persistência em arquivos binários porque ela fornece um contrato para objetos que podem ser gravados e lidos em arquivos binários. Quando um objeto implementa a interface `Serializable`, ele está indicando que pode ser serializado para um fluxo de bytes. Esse processo inclui a gravação dos dados do objeto, bem como as referências para outros objetos que o objeto contém.

A interface `Serializable` fornece um conjunto de métodos que são usados para serializar e desserializar objetos.

Ao implementar a interface `Serializable`, um desenvolvedor está garantindo que seu objeto possa ser serializado para um arquivo binário. Isso pode ser útil para armazenar dados de objetos em um arquivo, ou para transferir dados de um aplicativo para outro.

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

O paradigma funcional é um estilo de programação que se concentra na utilização de funções puras e expressões lambda. As funções puras são aquelas que não têm efeitos colaterais, ou seja, não alteram o estado do mundo exterior.

A API stream no Java é baseada no paradigma funcional e fornece uma maneira de trabalhar com dados de forma declarativa. Fazendo com que o código seja escrito de uma maneira a apresentar o que deve ser feito com os dados, em vez de como isso deve ser feito.

Como exemplos do uso do paradigma funcional na API stream, temos a função `filter()`, função `map()`, função `reduce()`.

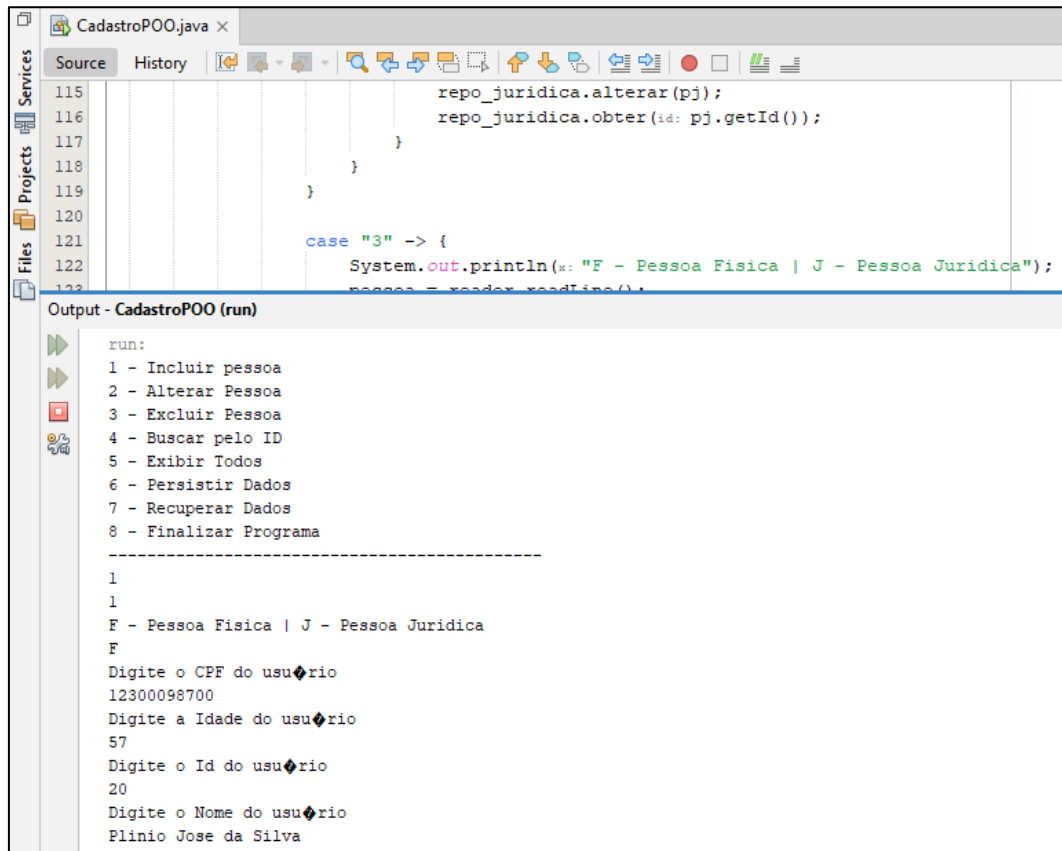
### **d. Quando trabalhamos com Java, qual padrão de desenvolvimento é adotado na persistência de dados em arquivos?**

Em Java, um padrão de desenvolvimento comum para a persistência de dados em arquivos é o padrão DAO (Data Access Object). O padrão DAO é um padrão de projeto de software que separa a lógica de acesso aos dados da lógica de negócios da aplicação.

No Padrão DAO, a lógica de negócios é responsável por obter e manipular os dados, enquanto a lógica de persistência de dados é responsável por acessar e armazenar os dados em um arquivo.

## 2º Procedimento | Criação do Cadastro em Modo Texto

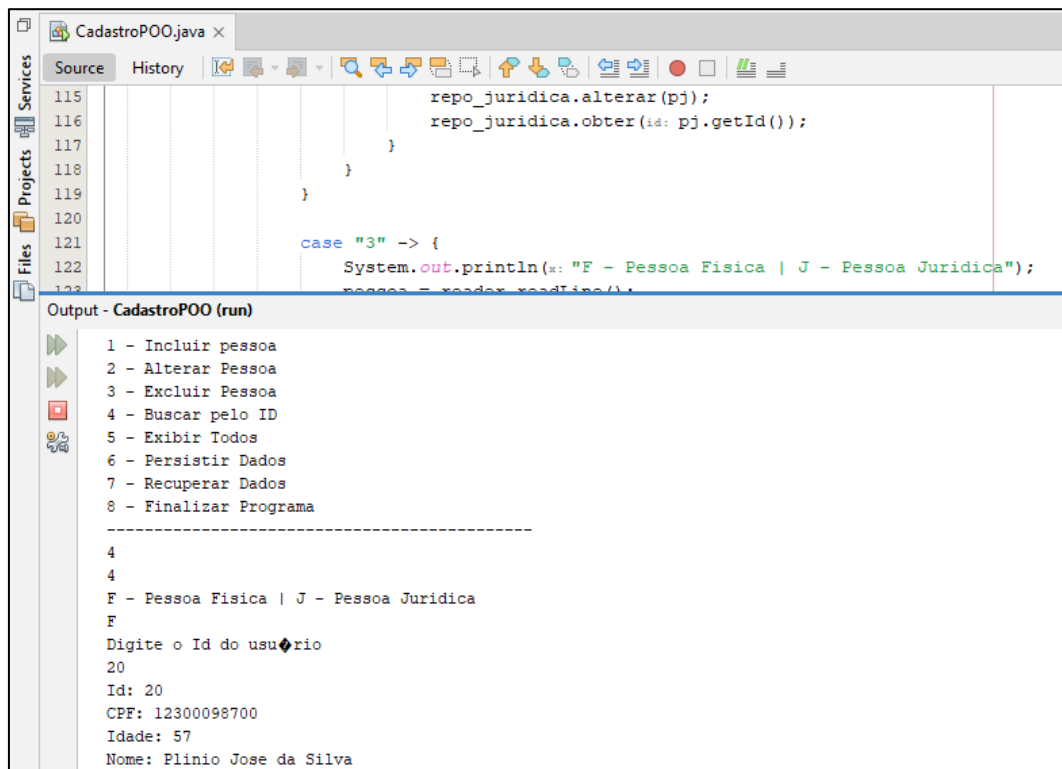
### 1. Resultado da execução dos códigos;



```
115         repo_juridica.alterar(pj);
116         repo_juridica.obter(id: pj.getId());
117     }
118 }
119 }
120
121 case "3" -> {
122     System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
123     pessoa = reader.readLine();
```

Output - CadastroPOO (run)

```
run:
1 - Incluir pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo ID
5 - Exibir Todos
6 - Persistir Dados
7 - Recuperar Dados
8 - Finalizar Programa
-----
1
1
F - Pessoa Fisica | J - Pessoa Juridica
F
Digite o CPF do usuário
12300098700
Digite a Idade do usuário
57
Digite o Id do usuário
20
Digite o Nome do usuário
Plinio Jose da Silva
```



```
115         repo_juridica.alterar(pj);
116         repo_juridica.obter(id: pj.getId());
117     }
118 }
119 }
120
121 case "3" -> {
122     System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
123     pessoa = reader.readLine();
```

Output - CadastroPOO (run)

```
1 - Incluir pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo ID
5 - Exibir Todos
6 - Persistir Dados
7 - Recuperar Dados
8 - Finalizar Programa
-----
4
4
F - Pessoa Fisica | J - Pessoa Juridica
F
Digite o Id do usuário
20
Id: 20
CPF: 12300098700
Idade: 57
Nome: Plinio Jose da Silva
```

```

129
130
131         case "J" -> {
132             System.out.println("Digite o Id do usuário");
133             int id = Integer.parseInt(reader.readLine());
134             repo_juridica.excluir(id);
135         }
136     }

```

Output - CadastroPOO (run)

```

8 - Finalizar Programa
-----
2
2
F - Pessoa Fisica | J - Pessoa Juridica
F
Digite o Id que deseja alterar:
20
CPF Antigo: 12300098700
Digite o novo CPF do usuário
12300098700
Idade Antigo: 57
Digite a nova Idade do usuário
20
Nome Antigo: Plinio Jose da Silva
Digite o novo Nome do usuário
Plinio Jose da Silva Gomes

```

Files Services Proje... x

CadastroPOO.java x

Source Packages

- CadastroPOO [main]
  - cadastropoo
    - model
  - Libraries
- CadastroPOO [main]
  - cadastropoo
    - CadastroPOO.java
    - model
      - Pessoa.java
      - PessoaFisica.java

Source History

```

52 String pessoa = "";
53
54 switch (opcao) {
55
56     case "1" -> {
57         System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
58         pessoa = reader.readLine();
59         switch (pessoa) {
60             case "F" -> {
61                 PessoaFisica p = new PessoaFisica();
62                 System.out.println("Digite o CPF do usuário");

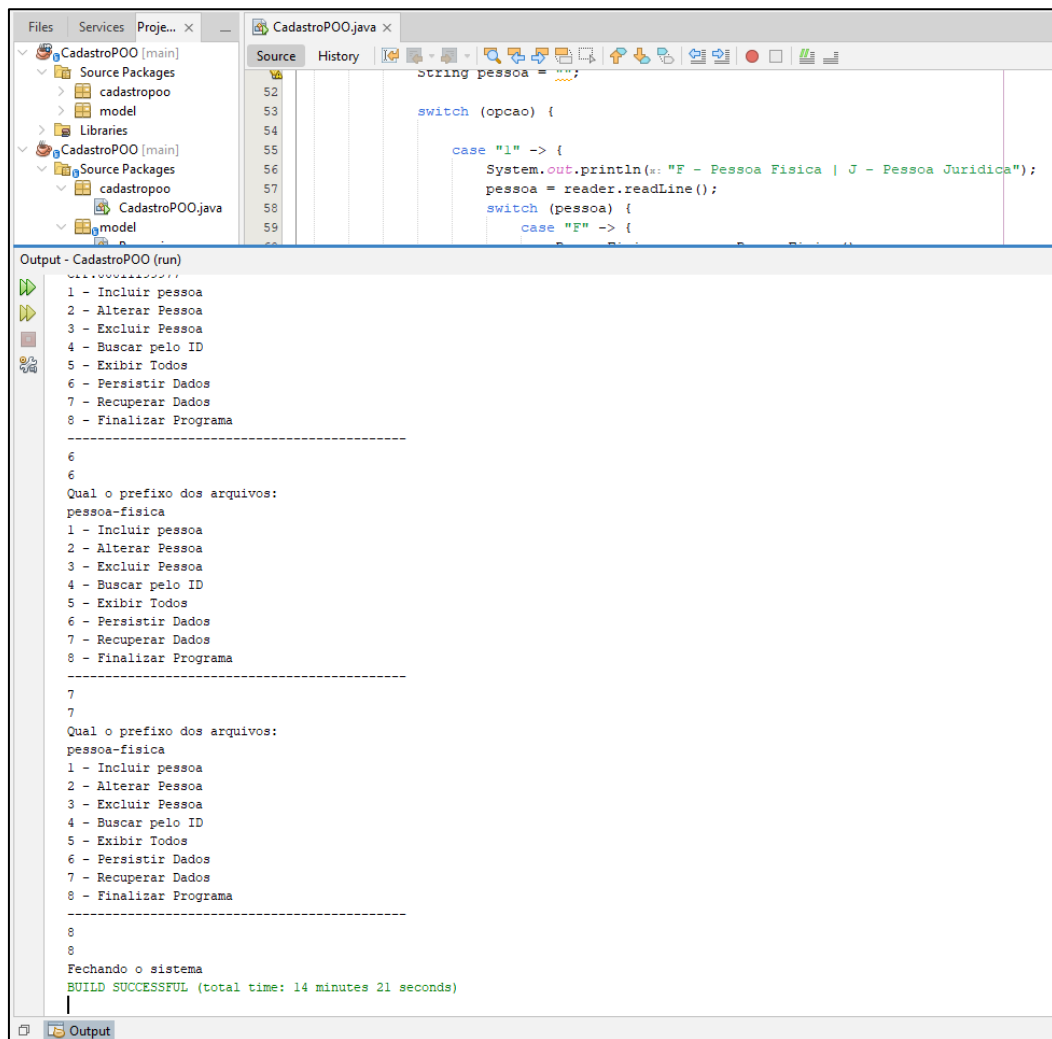
```

Output - CadastroPOO (run)

```

1 - Incluir pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo ID
5 - Exibir Todos
6 - Persistir Dados
7 - Recuperar Dados
8 - Finalizar Programa
-----
3
3
F - Pessoa Fisica | J - Pessoa Juridica
F
Digite o Id do usuário
20
1 - Incluir pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo ID
5 - Exibir Todos
6 - Persistir Dados
7 - Recuperar Dados
8 - Finalizar Programa
-----
5
5
F - Pessoa Fisica | J - Pessoa Juridica
F
id:1
Nome:Marcos Vinicios Rodrigues
CPF:00011122299
id:2
Nome:Paulo Silva Ribeiro
CPF:00011199977

```



## 2. Análise e Conclusão:

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

Elementos estáticos são aqueles que pertencem à classe e não a uma instância da classe. Isso significa que eles são compartilhados por todas as instâncias da classe.

O método main é um método estático porque ele é usado para iniciar a execução de um programa. Ele precisa ser estático para que todas as instâncias da classe possam acessá-lo.

Se o método main não fosse estático, teríamos que chamá-lo em cada instância da classe. Isso seria ineficiente, pois cada instância teria que criar seu próprio objeto do método main.

```
public class CadastroPOO {
    /**...3 lines */
    public static void main(String[] args) throws Exception { ...187 lines }
}
```

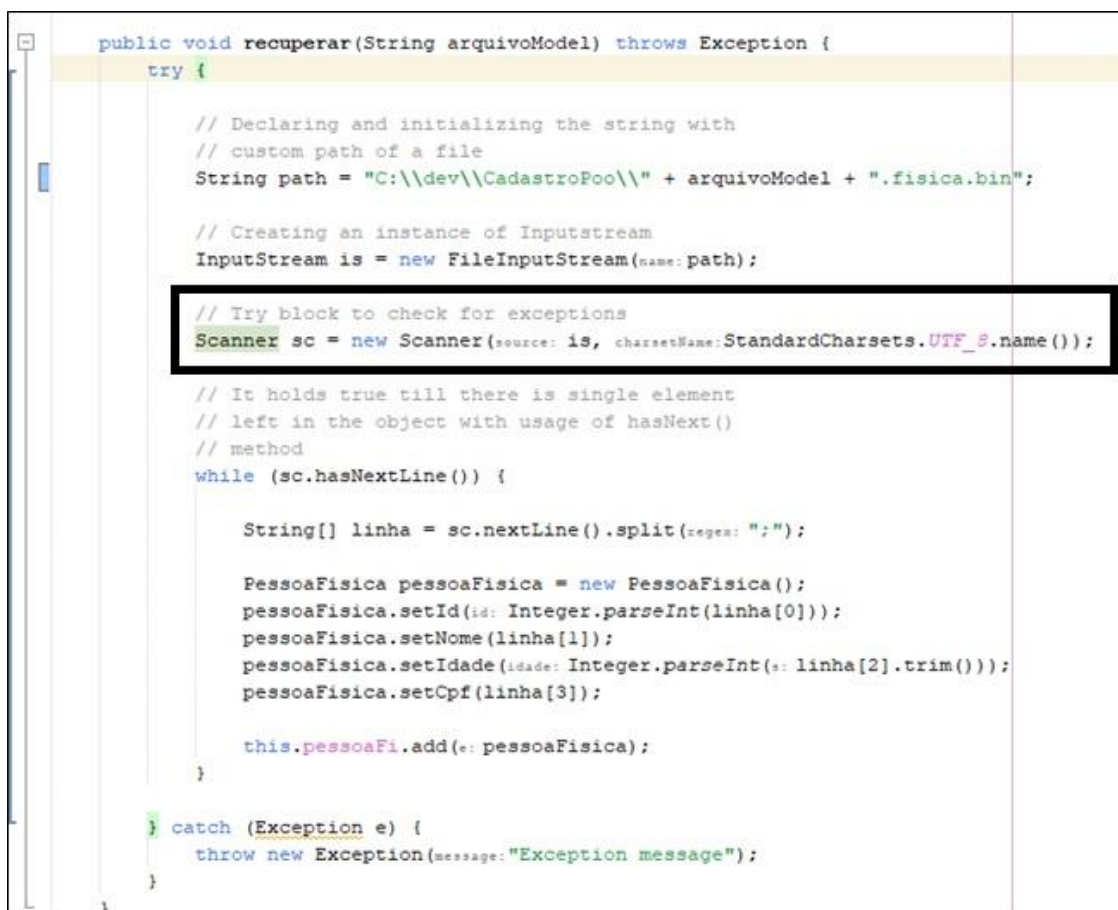
Na imagem acima, o método main é estático, porque está declarado com o modificador static.

### b. Para que serve a classe Scanner?

A classe Scanner em Java é uma classe da biblioteca padrão que é amplamente utilizada para ler entrada de dados do usuário ou de arquivos. Ela fornece vários métodos para ler tipos de dados primitivos, strings e outros tipos de objetos.

A classe Scanner é um objeto que pode ser usado para ler dados de várias fontes, incluindo o console, um arquivo e uma rede. Para ler dados do console, basta criar um objeto Scanner e passar o fluxo de entrada do console para o construtor.

Na prática desse relatório, foi utilizado a classe Scanner para criar um novo objeto Scanner e associá-lo o fluxo de entrada is ao objeto Scanner. O segundo argumento do construtor Scanner é o nome do conjunto de caracteres a ser usado para interpretar a entrada. Nesse caso, o conjunto de caracteres é UTF-8.



```
public void recuperar(String arquivoModel) throws Exception {
    try {
        // Declaring and initializing the string with
        // custom path of a file
        String path = "C:\\dev\\CadastroPoo\\" + arquivoModel + ".fisica.bin";

        // Creating an instance of InputStream
        InputStream is = new FileInputStream(name: path);

        // Try block to check for exceptions
        Scanner sc = new Scanner(source: is, charsetName: StandardCharsets.UTF_8.name());

        // It holds true till there is single element
        // left in the object with usage of hasNext()
        // method
        while (sc.hasNextLine()) {

            String[] linha = sc.nextLine().split(sep: ",");

            PessoaFisica pessoaFisica = new PessoaFisica();
            pessoaFisica.setId(id: Integer.parseInt(linha[0]));
            pessoaFisica.setNome(linha[1]);
            pessoaFisica.setIdade(idade: Integer.parseInt(linha[2].trim()));
            pessoaFisica.setCpf(linha[3]);

            this.pessoaFi.add(e: pessoaFisica);
        }

    } catch (Exception e) {
        throw new Exception(message: "Exception message");
    }
}
```

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

O uso de classes de repositório impactou na organização do código de várias maneiras:

Ajudando a separar a lógica de negócios da lógica de acesso a dados. Tornando o código mais fácil de entender e manter;

Melhorando a reusabilidade do código, já que podem ser usadas por várias classes de negócios, o que pode ajudar a reduzir a duplicação de código;

Melhorando a testabilidade do código. pois podem ser testadas independentemente das classes de negócios, o que pode ajudar a garantir que o código de acesso a dados esteja funcionando corretamente.

No entanto, é importante notar que a introdução de classes de repositório também pode adicionar uma camada de complexidade à aplicação. Portanto, é essencial equilibrar os benefícios de organização e abstração com a necessidade de manter o código claro e simples.

Em resumo, o uso de classes de repositório pode ter um impacto positivo na organização do código, tornando-o mais modular, reutilizável e fácil de manter, especialmente em sistemas que envolvem acesso a dados complexos.