



Estácio

UNIVERSIDADE ESTÁCIO DE SÁ DE RIBEIRÃO PRETO

POLO PARQUE ANDORINHAS

DESENVOLVIMENTO FULL STACK

2023.3 FULL STACK ALUNO

NIVEL 1: INICIANDO O CAMINHO PELO JAVA

ALESSANDRO SENDI SHIGEMATSU

1. Título da Prática

RPG0014 - Iniciando o caminho pelo Java

2. Objetivos da prática

1. Utilizar herança e polimorfismo na definição de entidades.
2. Utilizar persistência de objetos em arquivos binários.
3. Implementar uma interface cadastral em modo texto.
4. Utilizar o controle de exceções da plataforma Java.
5. 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. Todos os códigos solicitados neste roteiro de aula

```
package model;
```

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

```
public class Pessoa implements Serializable {
```

```
    private int id;
```

```
    private String nome;
```

```
    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;  
        }  
    }  
  
    package model;  
    import java.io.Serializable;  
    /**  
     *  
     * @author sendi  
     */  
    public class PessoaFisica extends Pessoa {  
        private String cpf;  
        private int idade;  
  
        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);  
}  
}
```

```

package model;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.stream.Collectors;

public class PessoaFisicaRepo {

    private ArrayList<PessoaFisica> pessoas = new ArrayList<>();

    public void inserir( PessoaFisica pessoa ) {

        pessoas.add( pessoa );
//    pessoasFisicas.add(pessoa);
    }

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

        return pessoas.stream().filter(pessoa -> pessoa.getId() == id ).findFirst().orElse(null);

    }

    public void alterar( PessoaFisica pessoaAlvo, PessoaFisica pessoaDestino ) {

        int index = pessoas.indexOf( pessoaAlvo );

        pessoas.set( index, pessoaDestino);
    }

```

```
}
```

```
public void excluir( int id ) {
```

```
    pessoas = (ArrayList<PessoaFisica>) pessoas.stream().filter(pessoa -> pessoa.getId() != id  
) .collect(Collectors.toList());
```

```
}
```

```
public ArrayList<PessoaFisica> obterTodos() {
```

```
    return pessoas;
```

```
}
```

```
public void persistir(String nomeArquivo) throws IOException {
```

```
    try (ObjectOutputStream outputStream = new ObjectOutputStream(new  
FileOutputStream(nomeArquivo))) {
```

```
        outputStream.writeObject(pessoas);
```

```
    }
```

```
}
```

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

```
    try (ObjectInputStream inputStream = new ObjectInputStream(new  
FileInputStream(nomeArquivo))) {
```

```
        pessoas = (ArrayList<PessoaFisica>) inputStream.readObject();
```

```
    }
```

```
}
```

```
}
```

```
package model;

import java.io.Serializable;

/**
 * @author sendi
 */
public class PessoaJuridica extends Pessoa implements Serializable {

    private String cnpj;

    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);
    }
}
```

```

package model;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.stream.Collectors;

public class PessoaFisicaRepo {

    private ArrayList<PessoaFisica> pessoas = new ArrayList<>();

    public void inserir( PessoaFisica pessoa ) {

        pessoas.add( pessoa );
//    pessoasFisicas.add(pessoa);
    }

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

        return pessoas.stream().filter(pessoa -> pessoa.getId() == id ).findFirst().orElse(null);

    }

    public void alterar( PessoaFisica pessoaAlvo, PessoaFisica pessoaDestino ) {

        int index = pessoas.indexOf( pessoaAlvo );

        pessoas.set( index, pessoaDestino);
    }

```



```
}
```

```
public void excluir( int id ) {  
    pessoas = (ArrayList<PessoaFisica>) pessoas.stream().filter(pessoa -> pessoa.getId() != id  
) .collect(Collectors.toList());  
}
```

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

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

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

```
package model;
```

```
import java.util.ArrayList;
```

```
/**
```

```
*
```

```
* @author sendi
```

```
*/
```

```
public class main {
```

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

```
        try {
```

```
            final String NOME_ARQUIVO_PESSOA_FISICA = "exemploPessoaFisica";
```

```
            final String NOME_ARQUIVO_PESSOA_JURIDICA = "exemploPessoaJuridica";
```

```
            PessoaFisicaRepo repo1 = new PessoaFisicaRepo( );
```

```
            repo1.inserir( new PessoaFisica( 1,"Ana", "1111111111", 25) );
```

```
            repo1.inserir( new PessoaFisica(2, "Carlos", "2222222222", 52) );
```

```
            repo1.persistir( NOME_ARQUIVO_PESSOA_FISICA );
```

```
            System.out.println("Dados de Pessoas Fisicas armazenados.");
```

```
            PessoaFisicaRepo repo2 = new PessoaFisicaRepo( );
```

```
            repo2.recuperar( NOME_ARQUIVO_PESSOA_FISICA );
```

```
            System.out.println("Dados de Pessoas Fisicas recuperados.");
```

```
            ArrayList<PessoaFisica> pessoasFisicas = repo2.obterTodos();
```

```
            pessoasFisicas.stream().forEach( pessoa -> pessoa.exibir() );
```

```
            PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo( );
```

```
repo3.inserir( new PessoaJuridica( 3,"Ana", "1111111111" ) );  
repo3.inserir( new PessoaJuridica(4, "Carlos", "2222222222" ) );
```

```
repo3.persistir( NOME_ARQUIVO_PESSOA_JURIDICA );  
System.out.println("Dados de Pessoas Juridicas armazenados.");
```

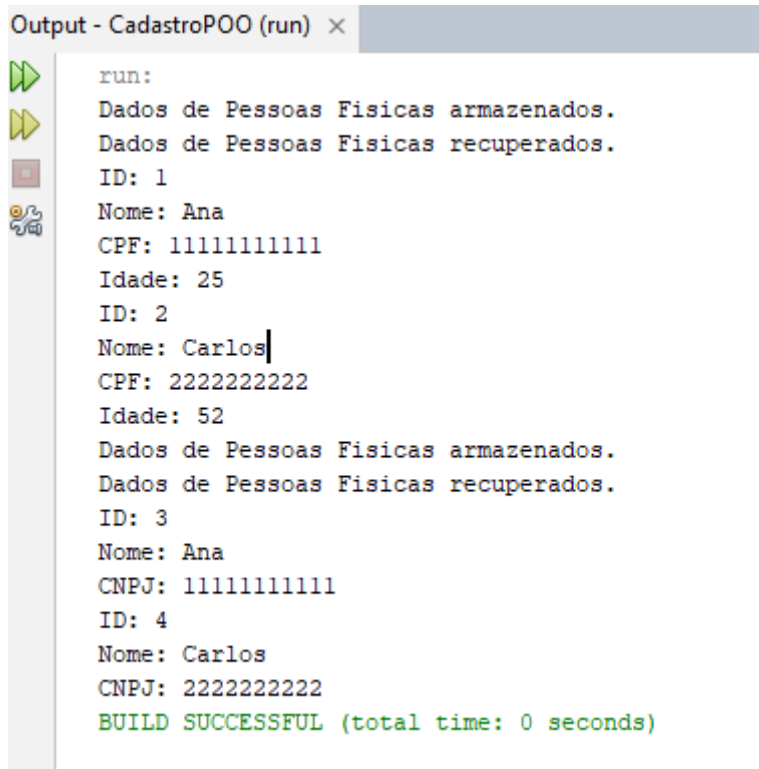
```
PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo( );  
repo4.recuperar( NOME_ARQUIVO_PESSOA_JURIDICA );  
System.out.println("Dados de Pessoas Juridicas recuperados.");
```

```
ArrayList<PessoaJuridica> pessoasJuridicas = repo4.obterTodos();
```

```
pessoasJuridicas.stream().forEach( pessoa -> pessoa.exibir() );
```

```
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}  
}
```

4. Os resultados da execução dos códigos também devem ser apresentados.



```
run:
Dados de Pessoas Fisicas armazenados.
Dados de Pessoas Fisicas recuperados.
ID: 1
Nome: Ana
CPF: 11111111111
Idade: 25
ID: 2
Nome: Carlos
CPF: 22222222222
Idade: 52
Dados de Pessoas Fisicas armazenados.
Dados de Pessoas Fisicas recuperados.
ID: 3
Nome: Ana
CNPJ: 11111111111
ID: 4
Nome: Carlos
CNPJ: 22222222222
BUILD SUCCESSFUL (total time: 0 seconds)
```

5. Análise e Conclusão

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

As vantagens são:

- As vantagens são redução de código duplicado.
- Desenvolvimento acelerado devida utilização de código existente.
- Consistência no código, padronizando em classes em métodos.
- Otimização dos recursos usados como tempo de CPU e memórias usadas.
- Escalamento fácil.

As desvantagens são:

- Qualquer mudança na classe pai irá refletir na classe filha afetando funcionalidades que não eram para haver mudanças.
- Problemas de Fragilidade na Classe base.
- Problemas de Múltiplas herança (Diamond Inheritance Problem) .
- Usar a herança desnecessariamente acarretando no aumento de complexidade no código.
- Inflexibilidade nas classes filhas.

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

Serialização é a técnica que permite transformar o estado de um objeto em uma sequência bytes. Uma vez serializado, um objeto pode ser salvo em arquivo e recuperado a partir do arquivo e desserializado para recriar o objeto na memória.

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

Através de expressões Lambda que são pequenos blocos de códigos que podem haver parâmetros e retornam algum valor. São similares aos métodos.

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

Java Persistence Api (JPA)