



Estácio

Estácio - Mundo 3 - Missão Nível 1

Faculdade Estácio - Polo Itaipava - Petrópolis/RJ.

Curso: Desenvolvimento Full Stack.

Disciplina: Nível 1: Iniciando o Caminho Pelo Java.

RPG0014.

Semestre Letivo: 3.

Integrante: Jeferson Jones Smith da Rocha.

Repositório: <https://github.com/JefersonSmith/estacio-mundo3-nivel1>

IDE Utilizada: Apache NetBeans.

Título da Prática

Iniciando o caminho pelo Java

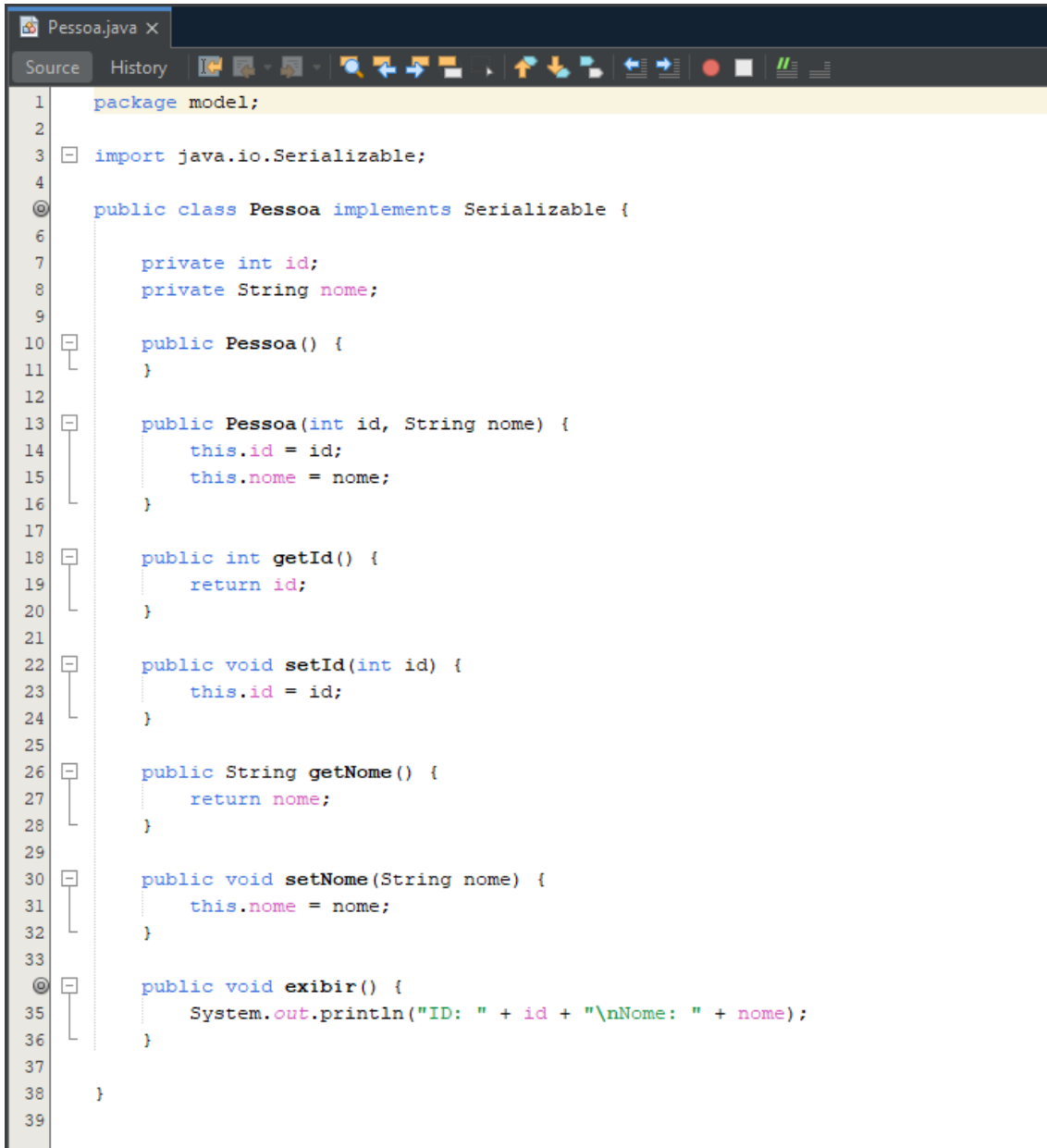
Implementação de um cadastro de clientes em modo texto, com persistência em arquivos, baseado na tecnologia Java.

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.

Arquivos do Projeto

Pessoa.java



```
1 package model;
2
3 import java.io.Serializable;
4
5 public class Pessoa implements Serializable {
6
7     private int id;
8     private String nome;
9
10    public Pessoa() {
11    }
12
13    public Pessoa(int id, String nome) {
14        this.id = id;
15        this.nome = nome;
16    }
17
18    public int getId() {
19        return id;
20    }
21
22    public void setId(int id) {
23        this.id = id;
24    }
25
26    public String getNome() {
27        return nome;
28    }
29
30    public void setNome(String nome) {
31        this.nome = nome;
32    }
33
34    public void exibir() {
35        System.out.println("ID: " + id + "\nNome: " + nome);
36    }
37
38 }
39
```

PessoaFisica.java

```
PessoaFisica.java X
Source History
1 package model;
2
3 import java.io.Serializable;
4 import model.Pessoa;
5
6 public class PessoaFisica extends Pessoa implements Serializable {
7
8     private String cpf;
9     private int idade;
10
11     public PessoaFisica() {
12         super();
13     }
14
15     public PessoaFisica(int id, String nome, String cpf, int idade) {
16         super(id, nome);
17         this.cpf = cpf;
18         this.idade = idade;
19     }
20
21     public String getCpf() {
22         return cpf;
23     }
24
25     public void setCpf(String cpf) {
26         this.cpf = cpf;
27     }
28
29     public int getIdade() {
30         return idade;
31     }
32
33     public void setIdade(int idade) {
34         this.idade = idade;
35     }
36
37     @Override
38     public void exibir() {
39         // Chama o método exibir da classe pai
40         super.exibir();
41         // Mostra as informações do objeto
42         System.out.println("CPF: " + cpf + "\nIdade: " + idade);
43     }
44 }
45
```

PessoaJuridica.java

```
PessoaJuridica.java X
Source History
1 package model;
2
3 import java.io.Serializable;
4
5 public class PessoaJuridica extends Pessoa implements Serializable {
6
7     private String cnpj;
8
9     public PessoaJuridica() {
10         super();
11     }
12
13     public PessoaJuridica(int id, String nome, String cnpj) {
14         super(id, nome);
15         this.cnpj = cnpj;
16     }
17
18     public String getCnpj() {
19         return cnpj;
20     }
21
22     public void setCnpj(String cnpj) {
23         this.cnpj = cnpj;
24     }
25
26     @Override
27     public void exibir() {
28         super.exibir();
29         // Exibe as informações do objeto
30         System.out.println("CNPJ: " + cnpj);
31     }
32 }
33
34
```

PessoaFisicaRepo.java

```
PessoaFisicaRepo.java X
Source History
1 package model;
2
3 import java.io.FileInputStream;
4 import java.io.FileOutputStream;
5 import java.io.IOException;
6 import java.io.ObjectInputStream;
7 import java.io.ObjectOutputStream;
8 import java.util.ArrayList;
9 import java.util.List;
10
11 public class PessoaFisicaRepo {
12
13     private List<PessoaFisica> pessoasFisicas;
14
15     public PessoaFisicaRepo() {
16         this.pessoasFisicas = new ArrayList<>();
17     }
18
19     public void inserir(PessoaFisica pessoaFisica) {
20         pessoasFisicas.add(pessoaFisica);
21     }
22
23     public void alterar(PessoaFisica pessoaFisica) {
24         for (int i = 0; i < pessoasFisicas.size(); i++) {
25             if (pessoasFisicas.get(i).getId() == pessoaFisica.getId()) {
26                 pessoasFisicas.set(i, pessoaFisica);
27                 return;
28             }
29         }
30     }
31
32     public void excluir(int id) {
33         pessoasFisicas.removeIf(pessoaFisica -> pessoaFisica.getId() == id);
34     }
35
36     public PessoaFisica obter(int id) {
37         for (PessoaFisica pf : pessoasFisicas) {
38             if (pf.getId() == id) {
39                 return pf;
40             }
41         }
42         return null;
43     }
44
45     public List<PessoaFisica> obterTodos() {
46         return new ArrayList<>(pessoasFisicas);
47     }
48
49     public void persistir(String nomeArquivo) throws IOException {
50         try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(nomeArquivo))) {
51             oos.writeObject(pessoasFisicas);
52         }
53     }
54
55     public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {
56         try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(nomeArquivo))) {
57             pessoasFisicas = (ArrayList<PessoaFisica>) ois.readObject();
58         }
59     }
60
61     public void setLista(List<PessoaFisica> listaFisica) {
62         this.pessoasFisicas = listaFisica;
63     }
64
65 }
66
```

PessoaJuridicaRepo.java

```
PessoaJuridicaRepo.java X
Source History
1 package model;
2
3 import java.io.FileInputStream;
4 import java.io.FileOutputStream;
5 import java.io.IOException;
6 import java.io.ObjectInputStream;
7 import java.io.ObjectOutputStream;
8 import java.util.ArrayList;
9 import java.util.List;
10
11 /**
12  *
13  * @author Smith
14  */
15 public class PessoaJuridicaRepo {
16
17     private List<PessoaJuridica> pessoasJuridicas;
18
19     public PessoaJuridicaRepo() {
20         this.pessoasJuridicas = new ArrayList<>();
21     }
22
23     public void inserir(PessoaJuridica pessoaJuridica) {
24         pessoasJuridicas.add(pessoaJuridica);
25     }
26
27     public void alterar(PessoaJuridica pessoaJuridica) {
28         for (int i = 0; i < pessoasJuridicas.size(); i++) {
29             if (pessoasJuridicas.get(i).getId() == pessoaJuridica.getId()) {
30                 pessoasJuridicas.set(i, pessoaJuridica);
31                 return;
32             }
33         }
34     }
35
36     public void excluir(int id) {
37         pessoasJuridicas.removeIf(pessoaJuridica -> pessoaJuridica.getId() == id);
38     }
39
40     public PessoaJuridica obter(int id) {
41         for (PessoaJuridica pj : pessoasJuridicas) {
42             if (pj.getId() == id) {
43                 return pj;
44             }
45         }
46         return null;
47     }
48
49     public List<PessoaJuridica> obterTodos() {
50         return new ArrayList<>(pessoasJuridicas);
51     }
52
53     public void persistir(String nomeArquivo) throws IOException {
54         try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(nomeArquivo))) {
55             oos.writeObject(pessoasJuridicas);
56         }
57     }
58
59     public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {
60         try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(nomeArquivo))) {
61             pessoasJuridicas = (ArrayList<PessoaJuridica>) ois.readObject();
62         }
63     }
64
65     public void setLista(List<PessoaJuridica> listaJuridica) {
66         this.pessoasJuridicas = listaJuridica;
67     }
68
69 }
70
```

Main.java

```
1 package cadastrapoo;
2
3 /**
4  *
5  * @author Smith
6  */
7
8 import java.io.IOException;
9 import model.PessoaFisica;
10 import model.PessoaFisicaRepo;
11 import model.PessoaJuridica;
12 import model.PessoaJuridicaRepo;
13
14
15 public class Main {
16
17     public static void main(String[] args) {
18         PessoaFisicaRepo repol = new PessoaFisicaRepo();
19
20         PessoaFisica pf1 = new PessoaFisica(1, "Jeferson Smith", "123.456.789-00", 18);
21         PessoaFisica pf2 = new PessoaFisica(2, "Daniela Smith", "111.111.111-11", 18);
22
23         repol.inserir(pf1);
24         repol.inserir(pf2);
25
26         String nomeArquivo = "pessoasFisicas.txt";
27         try {
28             repol.persistir(nomeArquivo);
29         } catch (IOException e) {
30             System.err.println("Erro: " + e);
31         }
32
33         PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
34
35         try {
36             repo2.recuperar(nomeArquivo);
37             System.out.println("Dados de Pessoas Fisicas Armazenados.");
38         } catch (IOException | ClassNotFoundException e) {
39             System.err.println("Erro: " + e);
40         }
41
42         System.out.println("Dados de Pessoas Fisicas Recuperados.");
43         for (PessoaFisica pf : repo2.obterTodos()) {
44             pf.exibir();
45         }
46
47         PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
48
49         PessoaJuridica pj1 = new PessoaJuridica(3, "Empresa 1", "11.111.111/0001-01");
50         PessoaJuridica pj2 = new PessoaJuridica(4, "Empresa 2", "11.111.111/0001-02");
51
52         repo3.inserir(pj1);
53         repo3.inserir(pj2);
54
55         String nomeArquivoPJ = "pessoasJuridicas.txt";
56         try {
57             repo3.persistir(nomeArquivoPJ);
58             System.out.println("\nDados de Pessoas Juridicas Armazenados.");
59         } catch (IOException e) {
60             System.err.println("Erro Encontrado: " + e);
61         }
62
63         PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
64
65         String nomeArquivoPJ2 = "pessoasJuridicas.txt";
66         try {
67             repo4.recuperar(nomeArquivoPJ2);
68         } catch (IOException | ClassNotFoundException e) {
69             System.err.println("Erro Encontrado: " + e);
70         }
71
72         System.out.println("Dados de Pessoas Juridicas Recuperados.");
73         for (PessoaJuridica pj : repo4.obterTodos()) {
74             pj.exibir();
75         }
76     }
77 }
78 }
```


Resultados da execução dos códigos:

```
Output - CadastroPOO (run) x
run:
Dados de Pessoas Fisicas Armazenados.
Dados de Pessoas Fisicas Recuperados.
ID: 1
Nome: Jeferson Smith
CPF: 123.456.789-00
Idade: 18
ID: 2
Nome: Daniela Smith
CPF: 111.111.111-11
Idade: 18

Dados de Pessoas Juridicas Armazenados.
Dados de Pessoas Juridicas Recuperados.
ID: 3
Nome: Empresa 1
CNPJ: 11.111.111/0001-01
ID: 4
Nome: Empresa 2
CNPJ: 11.111.111/0001-02
BUILD SUCCESSFUL (total time: 0 seconds)
```

Análise e Conclusão:

1) Quais as vantagens e desvantagens do uso de herança?

A herança é um conceito fundamental em programação orientada a objetos (POO), onde uma classe pode herdar atributos e métodos de outra classe.

Vantagens:

1. **Reutilização de código:** Uma das principais vantagens da herança é a capacidade de reutilizar o código existente. Uma classe filha pode herdar todos os métodos e atributos da classe pai, evitando a necessidade de reescrever o mesmo código.
2. **Facilita a manutenção:** Ao usar herança, as mudanças feitas na classe pai são automaticamente refletidas em todas as classes filhas. Isso facilita a manutenção do código, já que as alterações precisam ser feitas apenas em um único lugar.
3. **Promove a consistência:** Ao definir um conjunto de comportamentos em uma classe pai, você pode garantir que todas as classes filhas terão funcionalidades consistentes. Isso ajuda a manter a integridade e a consistência do sistema.
4. **Encapsulamento:** Herança permite encapsular comportamentos comuns em uma classe pai, o que pode tornar o código mais organizado e fácil de entender.

Desvantagens:

1. **Acoplamento forte:** A herança pode levar a um acoplamento forte entre as classes pai e filhas. Mudanças na classe pai podem ter um impacto inesperado em todas as classes filhas, o que pode tornar o sistema mais difícil de manter e modificar.
2. **Hierarquia complexa:** À medida que mais classes são adicionadas à hierarquia de herança, ela pode se tornar complexa e difícil de entender. Isso pode dificultar a manutenção e a extensão do código.
3. **Violação do princípio de substituição de Liskov:** Se as classes filhas não puderem ser usadas de forma transparente no lugar da classe pai, isso pode violar o princípio de substituição de Liskov, causando comportamento inesperado no código.
4. **Possíveis problemas de design:** A herança mal planejada pode levar a problemas de design, como classes muito grandes ou hierarquias profundas, o que pode prejudicar a legibilidade e a eficiência do código.

Em resumo, enquanto a herança pode oferecer muitas vantagens, é importante usá-la com cuidado e considerar as possíveis desvantagens para garantir um design de código robusto e flexível. Em alguns casos, outras técnicas de programação, como composição ou interfaces, podem ser mais adequadas.

2) 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 marca uma classe como sendo serializável, ou seja, objetos dessa classe podem ser convertidos em uma sequência de bytes, que podem ser posteriormente gravados em um arquivo binário. Isso é essencial para a persistência de objetos em Java.

3) Como o paradigma funcional é utilizado pela API `Stream` no Java?

A API `Stream` em Java aproveita conceitos do paradigma funcional para fornecer uma maneira mais eficiente e concisa de lidar com operações em coleções de dados. Como exemplo temos: operações de alta ordem, expressões `lambda`, imutabilidade, `lazy evaluation`...

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

Ao trabalhar com Java e persistência de dados em arquivos, um padrão comum é o padrão de projeto `DAO` (`Data Access Object`), muitas vezes combinado com o padrão de `Serialização`.

1. **Padrão `DAO` (`Data Access Object`):** O padrão `DAO` é usado para separar a lógica de acesso aos dados da lógica de negócios da aplicação. Ele define uma interface que fornece métodos para acessar e manipular dados, sem expor detalhes de implementação. Isso permite que você substitua facilmente a fonte de dados subjacente sem afetar o restante do código.
2. **Serialização:** A Serialização em Java é o processo de salvar o estado de um objeto em uma sequência de bytes, que pode ser posteriormente gravada em um arquivo. Isso permite que os objetos sejam persistidos em arquivos e posteriormente recuperados de forma fácil e eficiente. A Serialização é frequentemente usada em conjunto com o padrão `DAO` para persistir objetos em arquivos.

Ao usar o padrão `DAO` em conjunto com a Serialização, você pode criar classes `DAO` que encapsulam a lógica de acesso aos dados e fornecem métodos para salvar e carregar objetos em arquivos. Isso ajuda a manter um código organizado, modular e flexível, facilitando a manutenção e a extensão da aplicação.