Prof. Evandro César Freiberger

Exercício de Manipulação de Banco de Dados com JDBC Implementação em Camadas

Brasil

10 de outubro de 2018

Sumário

1 Projeto Implementação Modelo

Apresenta uma implementação que pode ser usada como base para o desenvolvimento de aplicações de manipulação de informações em banco de dados.

O Projeto é organizado em três camadas: visão, negócio e persistência. Usa uma classe VO para transporte de dados entre as camadas (AlunoVO).

- Crie um projeto do tipo Aplicativo Java com o nome **ProjetoImplementacaoModelo01**;
- Crie os pacotes de código-fonte: *execucao, visao, negocio, persistencia e vo*;

1.1 Estrutura do Banco de Dados

Crie uma tabela com o nome **aluno** em uma base de dados chamada **academico**, conforme script:

```
1  CREATE TABLE aluno
2  (
3    matricula serial NOT NULL,
4    nome character(50) NOT NULL,
5    nomemae character(50) NOT NULL,
6    nomepai character(50) NOT NULL,
7    sexo integer NOT NULL,
8    logradouro character(50),
9    numero integer,
10   bairro character(40),
11   cidade character(40),
12   uf character(2),
13   CONSTRAINT aluno_pkey PRIMARY KEY (matricula)
14 )
```

Código 1 – Tabela aluno

- Estrutura não normalizada para facilitar a implementação da primeira versão.

2 Classes VOs e Enumerações

2.1 Enum EnumUF

Crie uma enumeração Java com o nome EnumUF.java e codifique o seguinte código:

```
1  package vo;
2
3  public enum EnumUF {
4
5      MT,
6      MS,
7      PR,
8      SP
9
10 }
```

Código 2 – EnumUF.java

- Usado para representar o domínio de unidade de federação.
- As enumerações fortalecem a tipificação das informações.
- O compilador não permite erros tipográficos, como podem acontecer com literais de strings.
- O compilador não permite valores que estejam fora do conjunto enumerado.
- Não é necessário escrever pré-condições, ou testes manuais, para assegurar que o argumento de um método está dentro da gama de valores aceite.
- Podem ser usadas em switch-case.

2.2 Enum EnumSexo

Crie uma enumeração Java com o nome EnumSexo.java e codifique o seguinte código:

```
1  package vo;
2
3  public enum EnumSexo {
4
5      MASCULINO,
6      FEMININO
7
8  }
```

Código 3 – EnumSexo.java

- Usado para representar o domínio de sexo.
- Fortalecem a tipificação pois evita o uso de valores simbólicos ao invés de tipo.

2.3 Classe EnderecoVO

Crie uma classe Java com o nome EnderecoVO.java e codifique o seguinte código:

```java
package vo;

public class EnderecoVO {

    private String logradouro;
    private int numero;
    private String bairro;
    private String cidade;
    private EnumUF uf;

    public EnderecoVO() {
        this.logradouro = "";
        this.numero = 0;
        this.bairro = "";
        this.cidade = "";
        this.uf = EnumUF.MT;
    }

    public EnderecoVO(String logradouro, int numero, String bairro, String cidade, EnumUF uf) {
        this.logradouro = logradouro;
        this.numero = numero;
        this.bairro = bairro;
        this.cidade = cidade;
        this.uf = uf;
    }

    public String getLogradouro() {
        return logradouro;
    }

    public void setLogradouro(String logradouro) {
        this.logradouro = logradouro;
    }

    public int getNumero() {
        return numero;
    }

    public void setNumero(int numero) {
        this.numero = numero;
    }

    public String getBairro() {
        return bairro;
    }

    public void setBairro(String bairro) {
        this.bairro = bairro;
    }

    public String getCidade() {
        return cidade;
    }

    public void setCidade(String cidade) {
        this.cidade = cidade;
    }

    public EnumUF getUf() {
        return uf;
    }

    public void setUf(EnumUF uf) {
        this.uf = uf;
    }

    @Override
    public String toString() {
        return this.logradouro + ", " + this.numero+ ", " +this.bairro + " ," +
                this.cidade+ "-" +this.uf;
    }
}
```

Código 4 – EnderecoVO.java

- Usado para representar a instância de endereço, que é agregada em AlunoVO.
- Representa o conceito endereço que pode ser reusado em outras agregações.

2.4 Classe AlunoVO

Crie uma classe Java com o nome AlunoVO.java e codifique o seguinte código:

```java
package vo;

public class AlunoVO {

    private int matricula;
    private String nome;
    private String nomeMae;
    private String nomePai;
    private EnumSexo sexo;
    private EnderecoVO endereco;

    public AlunoVO() {
        this.endereco = new EnderecoVO();
        this.matricula = 0;
        this.nome = "";
        this.nomeMae = "";
        this.nomePai = "";
        this.sexo = EnumSexo.FEMININO;
    }

    public AlunoVO(int matricula, String nome, EnumSexo sexo) {
        this();
        this.matricula = matricula;
        this.nome = nome;
        this.sexo = sexo;
    }

    public int getMatricula() {
        return matricula;
    }

    public void setMatricula(int matricula) {
        this.matricula = matricula;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getNomeMae() {
        return nomeMae;
    }

    public void setNomeMae(String nomeMae) {
        this.nomeMae = nomeMae;
    }

    public String getNomePai() {
        return nomePai;
    }

    public void setNomePai(String nomePai) {
        this.nomePai = nomePai;
    }

    public EnumSexo getSexo() {
        return sexo;
    }

    public void setSexo(EnumSexo sexo) {
        this.sexo = sexo;
    }

    public EnderecoVO getEndereco() {
        return endereco;
    }

    public void setEndereco(EnderecoVO endereco) {
        this.endereco = endereco;
    }

    @Override
    public String toString() {
        return matricula +", "+ nome +", "+ sexo +", residente em: "+ endereco;
    }
}
```

Código 5 – AlunoVO.java

- Usado para transporte de dados entre as camadas.

- Representa a entidade Aluno.
- Possui um objeto EnderecoVO agregado para representar os dados de endereço do aluno.

3 Classes de Persistência

3.1 PersistenciaException

Crie uma classe Java com o nome PersistenciaException.java e codifique o seguinte código:

```java
package persistencia;

public class PersistenciaException extends Exception {

    public PersistenciaException() {
        super("Erro ocorrido na conexÃ£o do banco de dados");
    }

    public PersistenciaException(String msg) {
        super(msg);
    }
}
```

Código 6 – PersistenciaException.java

- Especialização da classe Exception para captura e tratamento de exceções ocorridas na camada de persitência.

3.2 ConexaoBD

Crie uma classe Java com o nome ConexaoBD.java e codifique o seguinte código:

```java
package persistencia;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class ConexaoBD {

    private Connection con;
    private static ConexaoBD instancia;

    private ConexaoBD() throws PersistenciaException {
        try {
            Class.forName("org.postgresql.Driver");
            String url =  "jdbc:postgresql://localhost:5432/academico";
            con = DriverManager.getConnection(url,"postgres","postgres");
        } catch (SQLException ex) {
            throw new PersistenciaException("Erro ao conectar o banco de dados - "+ex.toString());
        } catch (ClassNotFoundException ex) {
            throw new PersistenciaException("Driver do banco de dados nÃ£o localizado - "+ex.
                toString());
        }
    }

    public static ConexaoBD getInstancia() throws PersistenciaException {
        if (instancia == null) {
            instancia = new ConexaoBD();
        }
        return instancia;
    }

    public Connection getConexao() {
        return con;
    }

    public void desconectar() throws PersistenciaException {
        try {
            con.close();
        } catch (SQLException ex) {
            throw new PersistenciaException("Erro ao desconectar o banco de dados - " +ex.toString()
                );
        }
    }
}
```

Código 7 – ConexaoBD.java

- Classe responsável por criar e retornar uma instância de conexão com o banco de dados.

3.3 DAO

Crie uma classe Java com o nome DAO.java e codifique o seguinte código:

```
1  package persistencia;
2
3  public class DAO {
4
5      protected ConexaoBD conexao;
6
7      public DAO(ConexaoBD conexao) throws PersistenciaException {
8          this.conexao = conexao;
9      }
10
11     public ConexaoBD getConexao() {
12         return conexao;
13     }
14
15     public void setConexao(ConexaoBD conexao) {
16         this.conexao = conexao;
17     }
18
19     public void desconectarBD() throws PersistenciaException {
20         conexao.desconectar();
21     }
22 }
```

Código 8 – DAO.java

- Superclasse para as classes de persistência (DAO).

3.4 AlunoDAO

Crie uma classe Java com o nome AlunoDAO.java e codifique o seguinte código:

```
1  package persistencia;
2
3  import java.sql.PreparedStatement;
4  import java.sql.ResultSet;
5  import java.sql.SQLException;
6  import java.util.ArrayList;
7  import java.util.List;
8  import vo.AlunoVO;
9  import vo.EnumSexo;
10 import vo.EnumUF;
11
12 public class AlunoDAO extends DAO {
13
14     private static PreparedStatement comandoIncluir;
15     private static PreparedStatement comandoAlterar;
16     private static PreparedStatement comandoExcluir;
17     private static PreparedStatement comandoBuscaMatricula;
18
19     public AlunoDAO(ConexaoBD conexao) throws PersistenciaException {
20         super(conexao);
21         try {
22             comandoIncluir = conexao.getConexao().prepareStatement("INSERT INTO Aluno ( nome,
                   nomemae, nomepai, sexo, " +
23                     "logradouro, numero, bairro, cidade, uf )VALUES (?, ?, ?, ?, ?, ?, ? ,?, ?)");
24             comandoAlterar = conexao.getConexao().prepareStatement(
25                     "UPDATE Aluno SET nome=?, nomemae=?, nomepai=?, sexo=?," +
26                     "logradouro=?, numero=?, bairro=?, cidade=?, uf=? WHERE matricula=?");
27             comandoExcluir = conexao.getConexao().prepareStatement("DELETE FROM Aluno WHERE
                   matricula=?");
28             comandoBuscaMatricula = conexao.getConexao().prepareStatement("SELECT * FROM Aluno WHERE
                    matricula = ?");
29         } catch (SQLException ex) {
30             throw new PersistenciaException("Erro ao incluir novo aluno - "+ex.getMessage());
31         }
32     }
33
34     public int incluir(AlunoVO alunoVO) throws PersistenciaException {
35         int retorno = 0;
36         try {
37             comandoIncluir.setString(1, alunoVO.getNome());
38             comandoIncluir.setString(2, alunoVO.getNomeMae());
39             comandoIncluir.setString(3, alunoVO.getNomePai());
40             comandoIncluir.setInt(4, alunoVO.getSexo().ordinal());
41             comandoIncluir.setString(5, alunoVO.getEndereco().getLogradouro());
42             comandoIncluir.setInt(6, alunoVO.getEndereco().getNumero());
43             comandoIncluir.setString(7, alunoVO.getEndereco().getBairro());
44             comandoIncluir.setString(8, alunoVO.getEndereco().getCidade());
45             comandoIncluir.setString(9, alunoVO.getEndereco().getUf().name());
46             retorno = comandoIncluir.executeUpdate();
47         } catch (SQLException ex) {
48             throw new PersistenciaException("Erro ao incluir novo aluno - "+ex.getMessage());
49         }
50         return retorno;
51     }
52
53     public int alterar(AlunoVO alunoVO) throws PersistenciaException {
54         int retorno = 0;
55         try {
56             comandoAlterar.setString(1, alunoVO.getNome());
57             comandoAlterar.setString(2, alunoVO.getNomeMae());
58             comandoAlterar.setString(3, alunoVO.getNomePai());
59             comandoAlterar.setInt(4, alunoVO.getSexo().ordinal());
60             comandoAlterar.setString(5, alunoVO.getEndereco().getLogradouro());
61             comandoAlterar.setInt(6, alunoVO.getEndereco().getNumero());
62             comandoAlterar.setString(7, alunoVO.getEndereco().getBairro());
63             comandoAlterar.setString(8, alunoVO.getEndereco().getCidade());
64             comandoAlterar.setString(9, alunoVO.getEndereco().getUf().name());
65             comandoAlterar.setInt(10, alunoVO.getMatricula());
66             retorno = comandoAlterar.executeUpdate();
67         } catch (SQLException ex) {
68             throw new PersistenciaException("Erro ao alterar o aluno - "+ex.getMessage());
69         }
70         return retorno;
71     }
72
73     public int excluir(int matricula) throws PersistenciaException {
74         int retorno = 0;
75         try {
76             comandoExcluir.setInt(1, matricula);
77             retorno = comandoExcluir.executeUpdate();
78         } catch (SQLException ex) {
79             throw new PersistenciaException("Erro ao excluir o aluno - "+ex.getMessage());
80         }
81         return retorno;
82     }
83
84     public AlunoVO buscarPorMatricula(int matricula) throws PersistenciaException {
```

```
85
86          AlunoVO alu = null;
87
88          try {
89              comandoBuscaMatricula.setInt(1, matricula);
90              ResultSet rs = comandoBuscaMatricula.executeQuery();
91              if (rs.next()) {
92                  alu = new AlunoVO();
93                  alu.setMatricula(rs.getInt("matricula"));
94                  alu.setNome(rs.getString("Nome").trim());
95                  alu.setNomeMae(rs.getString("nomemae"));
96                  alu.setNomePai(rs.getString("nomepai"));
97                  alu.setSexo(EnumSexo.values()[rs.getInt("sexo")]);
98                  alu.getEndereco().setLogradouro(rs.getString("logradouro"));
99                  alu.getEndereco().setNumero(rs.getInt("numero"));
100                 alu.getEndereco().setBairro(rs.getString("bairro"));
101                 alu.getEndereco().setCidade(rs.getString("cidade"));
102                 alu.getEndereco().setUf(EnumUF.valueOf(rs.getString("uf")));
103             }
104         } catch (Exception ex) {
105             throw new PersistenciaException("Erro na seleÃ§Ã£o por codigo - "+ex.getMessage());
106         }
107         return alu;
108     }
109
110     public List<AlunoVO> buscarPorNome(String nome) throws PersistenciaException {
111         List<AlunoVO> listaAluno = new ArrayList();
112         AlunoVO alu = null;
113
114         String comandoSQL = "SELECT * FROM Aluno WHERE UPPER(nome) LIKE '" + nome.trim().toUpperCase
                ()) + "%' ORDER BY NOME LIMIT 10";
115
116         try {
117             PreparedStatement comando = conexao.getConexao().prepareStatement(comandoSQL);
118             ResultSet rs = comando.executeQuery();
119             while (rs.next()) {
120                 alu = new AlunoVO();
121                 alu.setMatricula(rs.getInt("matricula"));
122                 alu.setNome(rs.getString("Nome").trim());
123                 alu.setNomeMae(rs.getString("nomemae"));
124                 alu.setNomePai(rs.getString("nomepai"));
125                 alu.setSexo(EnumSexo.values()[rs.getInt("sexo")]);
126                 alu.getEndereco().setLogradouro(rs.getString("logradouro"));
127                 alu.getEndereco().setNumero(rs.getInt("numero"));
128                 alu.getEndereco().setBairro(rs.getString("bairro"));
129                 alu.getEndereco().setCidade(rs.getString("cidade"));
130                 alu.getEndereco().setUf(EnumUF.valueOf(rs.getString("uf")));
131                 listaAluno.add(alu);
132             }
133             comando.close();
134         } catch (Exception ex) {
135             throw new PersistenciaException("Erro na selecaoo por nome - "+ex.getMessage());
136         }
137         return listaAluno;
138     }
139 }
```

Código 9 – AlunoDAO.java

- Subclasse de DAO, especializada nas operações de persistência para a entidade Aluno.

4 Classes de Negócio

4.1 NegocioException

Crie uma classe Java com o nome NegocioException.java e codifique o seguinte código:

```
1  package negocio;
2
3  public class NegocioException extends Exception {
4
5      public NegocioException() {
6          super("Erro ocorrido na camada de negocio");
7      }
8
9      public NegocioException(String msg) {
10         super(msg);
11     }
12 }
```

Código 10 – NegocioException.java

4.2 AlunoNegocio

Crie uma classe Java com o nome AlunoNegocio.java e codifique o seguinte código:

```
1  package negocio;
2
3  import java.util.List;
4  import persistencia.AlunoDAO;
5  import persistencia.ConexaoBD;
6  import persistencia.PersistenciaException;
7  import vo.AlunoVO;
8
9  public class AlunoNegocio {
10
11     private AlunoDAO alunoDAO;
12
13     public AlunoNegocio() throws NegocioException {
14         try {
15             this.alunoDAO = new AlunoDAO(ConexaoBD.getInstancia());
16         } catch (PersistenciaException ex) {
17             throw new NegocioException("Erro ao iniciar a Persistencia - " + ex.getMessage());
18         }
19     }
20
21     public void inserir(AlunoVO alunoVO) throws NegocioException {
22
23         String mensagemErros = this.validarDados(alunoVO);
24
25         if (!mensagemErros.isEmpty()) {
26             throw new NegocioException(mensagemErros);
27         }
28
29         try {
30             if (alunoDAO.incluir(alunoVO) == 0) {
31                 throw new NegocioException("InclusÃ£o nao realizada!!");
32             }
33         } catch (PersistenciaException ex) {
34             throw new NegocioException("Erro ao incluir o produto - " + ex.getMessage());
35         }
36     }
37
38     public void alterar(AlunoVO alunoVO) throws NegocioException {
39         String mensagemErros = this.validarDados(alunoVO);
40         if (!mensagemErros.isEmpty()) {
41             throw new NegocioException(mensagemErros);
42         }
43         try {
44             if (alunoDAO.alterar(alunoVO) == 0) {
45                 throw new NegocioException("AlteraÃ§Ã£o nao realizada!!");
46             }
47         } catch (PersistenciaException ex) {
48             throw new NegocioException("Erro ao alterar o aluno - " + ex.getMessage());
49         }
50     }
51
52     public void excluir(int codigo) throws NegocioException {
53         try {
54             if (alunoDAO.excluir(codigo) == 0) {
55                 throw new NegocioException("AlteraÃ§Ã£o nÃ£o realizada!!");
56             }
57         } catch (PersistenciaException ex) {
58             throw new NegocioException("Erro ao excluir o aluno - " + ex.getMessage());
59         }
60     }
61
```

```
62      public List<AlunoVO> pesquisaParteNome(String parteNome) throws NegocioException {
63          try {
64              return alunoDAO.buscarPorNome(parteNome);
65          } catch (PersistenciaException ex) {
66              throw new NegocioException("Erro ao pesquisar aluno pelo nome - " + ex.getMessage());
67          }
68      }
69
70      public AlunoVO pesquisaMatricula(int matricula) throws NegocioException {
71          try {
72              return alunoDAO.buscarPorMatricula(matricula);
73          } catch (PersistenciaException ex) {
74              throw new NegocioException("Erro ao pesquisar aluno pela matricula - " + ex.getMessage()
                    );
75          }
76      }
77
78      private String validarDados(AlunoVO alunoVO) {
79
80          String mensagemErros = "";
81
82          if (alunoVO.getNome() == null || alunoVO.getNome().length() == 0) {
83              mensagemErros += "Nome do aluno nao pode ser vazio";
84          }
85
86          if (alunoVO.getNomeMae() == null || alunoVO.getNomeMae().length() == 0) {
87              mensagemErros += "\nNome da mae nao pode ser vazio";
88          }
89
90          if (alunoVO.getNomePai() == null || alunoVO.getNomePai().length() == 0) {
91              mensagemErros += "\nNome do pai nao pode ser vazio";
92          }
93
94          if (alunoVO.getSexo() == null) {
95              mensagemErros += "\nSexo nao pode ser nulo";
96          }
97
98          if (alunoVO.getEndereco().getLogradouro() == null || alunoVO.getEndereco().getLogradouro().
                length() == 0) {
99              mensagemErros += "\nLogradouro nao pode ser vazio";
100         }
101
102         if (alunoVO.getEndereco().getNumero() <= 0) {
103             mensagemErros += "\nNumero deve ser maior que zero";
104         }
105
106         if (alunoVO.getEndereco().getBairro() == null || alunoVO.getEndereco().getBairro().length()
                == 0) {
107             mensagemErros += "\nBairro nao pode ser vazio";
108         }
109
110         if (alunoVO.getEndereco().getCidade() == null || alunoVO.getEndereco().getCidade().length()
                == 0) {
111             mensagemErros += "\nCidade nao pode ser vazio";
112         }
113         if (alunoVO.getEndereco().getUf() == null) {
114             mensagemErros += "\nUF nao pode ser vazio";
115         }
116
117         return mensagemErros;
118     }
119 }
```

Código 11 – AlunoNegocio.java

5 Classes de Execução

5.1 Enum EnumMenuAluno

Crie uma enumeração Java com o nome EnumMenuAluno.java e codifique o seguinte código:

```
1  package execucao;
2
3  public enum EnumMenuAluno {
4
5      IncluirAluno,
6      AlterarAluno,
7      ExcluirAluno,
8      PesqMatricula,
9      PesqNome,
10     Sair
11 }
```

Código 12 – EnumMenuAluno.java

5.2 Principal

Crie uma classe Java com o nome Principal.java e codifique o seguinte código:

```
1  package execucao;
2
3  import java.util.List;
4  import javax.swing.JOptionPane;
5  import negocio.AlunoNegocio;
6  import negocio.NegocioException;
7  import vo.AlunoVO;
8  import vo.EnumSexo;
9  import vo.EnumUF;
10
11 public class Principal {
12
13     private static AlunoNegocio alunoNegocio;
14
15     public static void main(String[] args) {
16         try {
17             alunoNegocio = new AlunoNegocio();
18         } catch (NegocioException ex) {
19             System.out.println("Camada de negocio e persistencia nao iniciada - " + ex.getMessage())
                   ;
20         }
21
22         if (alunoNegocio != null) {
23             EnumMenuAluno opcao = EnumMenuAluno.Sair;
24             do {
25                 try {
26                     opcao = exibirMenu();
27                     switch (opcao) {
28                         case IncluirAluno:
29                             incluirAluno();
30                             break;
31                         case AlterarAluno:
32                             alterarAluno();
33                             break;
34                         case ExcluirAluno:
35                             excluirAluno();
36                             break;
37                         case PesqMatricula:
38                             pesquisarPorMatricula();
39                             break;
40                         case PesqNome:
41                             pesquisarPorNome();
42                     }
43                 } catch (NegocioException ex) {
44                     System.out.println("Operacao nao realizada corretamente - " + ex.getMessage());
45                 }
46             } while (opcao != EnumMenuAluno.Sair);
47         }
48         System.exit(0);
49     }
50
51     /**
52      * Inclui um novo aluno na base de dados
53      *
54      * @throws NegocioException
55      */
56     private static void incluirAluno() throws NegocioException {
57
58         AlunoVO alunoTemp = lerDados();
59         alunoNegocio.inserir(alunoTemp);
60     }
61
```

```
62      /**
63       * Permite a alteracao dos dados de um aluno por meio da matricula
64       * fornecida.
65       *
66       * @throws NegocioException
67       */
68      private static void alterarAluno() throws NegocioException {
69          int matricula = 0;
70          try {
71              matricula = Integer.parseInt(JOptionPane.showInputDialog(null, "Forneca a matricula do
                    Aluno", "Leitura de Dados", JOptionPane.QUESTION_MESSAGE));
72          } catch (Exception ex) {
73              JOptionPane.showMessageDialog(null, "Digitacao inconsistente - " + ex.getMessage());
74          }
75
76          AlunoVO alunoVO = alunoNegocio.pesquisaMatricula(matricula);
77          if (alunoVO != null) {
78              AlunoVO alunoTemp = lerDados(alunoVO);
79              alunoNegocio.alterar(alunoTemp);
80          } else {
81              JOptionPane.showMessageDialog(null, "Aluno nao localizado");
82          }
83      }
84
85      /**
86       * Exclui um aluno por meio de uma matricula fornecida.
87       *
88       * @throws NegocioException
89       */
90      private static void excluirAluno() throws NegocioException {
91          int matricula = 0;
92          try {
93              matricula = Integer.parseInt(JOptionPane.showInputDialog(null, "Forneca a matricula do
                    Aluno", "Leitura de Dados", JOptionPane.QUESTION_MESSAGE));
94          } catch (Exception ex) {
95              JOptionPane.showMessageDialog(null, "Digitacao inconsistente - " + ex.getMessage());
96          }
97
98          AlunoVO alunoVO = alunoNegocio.pesquisaMatricula(matricula);
99          if (alunoVO != null) {
100             alunoNegocio.excluir(alunoVO.getMatricula());
101         } else {
102             JOptionPane.showMessageDialog(null, "Aluno nao localizado");
103         }
104     }
105
106     /**
107      * Pesquisa um aluno por meio da matricula.
108      *
109      * @throws NegocioException
110      */
111     private static void pesquisarPorMatricula() throws NegocioException {
112         int matricula = 0;
113         try {
114             matricula = Integer.parseInt(JOptionPane.showInputDialog(null, "ForneÃ§a a matricula do
                    Aluno", "Leitura de Dados", JOptionPane.QUESTION_MESSAGE));
115         } catch (Exception ex) {
116             JOptionPane.showMessageDialog(null, "Digitacao inconsistente - " + ex.getMessage());
117         }
118
119         AlunoVO alunoVO = alunoNegocio.pesquisaMatricula(matricula);
120         if (alunoVO != null) {
121             mostrarDados(alunoVO);
122         } else {
123             JOptionPane.showMessageDialog(null, "Aluno nao localizado");
124         }
125     }
126
127     /**
128      * Le um nome ou parte de um nome de um aluno e busca no banco de dados
129      * alunos que possuem esse nome, ou que iniciam com a parte do nome
130      * fornecida. Caso nao seja fornecido nenhum valor de entrada sera retornado
131      * os 10 primmeiros alunos ordenados pelo nome.
132      *
133      * @throws NegocioException
134      */
135     private static void pesquisarPorNome() throws NegocioException {
136
137         String nome = JOptionPane.showInputDialog(null, "ForneÃ§a o nome do Aluno", "Leitura de
                Dados", JOptionPane.QUESTION_MESSAGE);
138         if (nome != null) {
139             List<AlunoVO> listaAlunoVO = alunoNegocio.pesquisaParteNome(nome);
140
141             if (listaAlunoVO.size() > 0) {
142                 for (AlunoVO alunoVO : listaAlunoVO) {
143                     mostrarDados(alunoVO);
144                 }
145             } else {
146                 JOptionPane.showMessageDialog(null, "Aluno nao localizado");
147             }
148         }else{
```

```
149             JOptionPane.showMessageDialog(null, "Nome nao pode ser nulo");
150         }
151     }
152 
153     /**
154      * Exibe no console da aplicacao os dados dos alunos recebidos pelo
155      * parametro alunoVO.
156      *
157      * @param alunoVO
158      */
159     private static void mostrarDados(AlunoVO alunoVO) {
160         if (alunoVO != null) {
161             System.out.println("Matricula: " + alunoVO.getMatricula());
162             System.out.println("Nome: " + alunoVO.getNome());
163             System.out.println("Nome da Mae: " + alunoVO.getNomeMae());
164             System.out.println("Nome da Pai: " + alunoVO.getNomePai());
165             System.out.println("Sexo: " + alunoVO.getSexo().name());
166             if (alunoVO.getEndereco() != null) {
167                 System.out.println("Logradouro: " + alunoVO.getEndereco().getLogradouro());
168                 System.out.println("Numero: " + alunoVO.getEndereco().getNumero());
169                 System.out.println("Bairro: " + alunoVO.getEndereco().getBairro());
170                 System.out.println("Cidade: " + alunoVO.getEndereco().getCidade());
171                 System.out.println("UF: " + alunoVO.getEndereco().getUf());
172                 System.out.println("----------------------------------------");
173             }
174         }
175     }
176 
177     /**
178      * Le os dados de um aluno exibindo os dados atuais recebidos pelo parametro
179      * alunoTemp. Na alteracao permite que os dados atuais do alunos sejam
180      * visualizados. Na inclusÃ£o sÃ£o exibidos os dados inicializados no AlunoVO.
181      *
182      * @param alunoTemp
183      * @return
184      */
185     private static AlunoVO lerDados(AlunoVO alunoTemp) {
186 
187         String nome, nomeMae, nomePai, logradouro, bairro, cidade;
188         int numero;
189         EnumSexo sexo;
190         EnumUF uf;
191 
192         try {
193             nome = JOptionPane.showInputDialog("Forneca o nome do Aluno", alunoTemp.getNome().trim()
                    );
194             alunoTemp.setNome(nome);
195             nomeMae = JOptionPane.showInputDialog("Forneca o nome da mae do Aluno", alunoTemp.
                    getNomeMae().trim());
196             alunoTemp.setNomeMae(nomeMae);
197 
198             nomePai = JOptionPane.showInputDialog("Forneca o nome do pai do Aluno", alunoTemp.
                    getNomePai().trim());
199             alunoTemp.setNomePai(nomePai);
200 
201             sexo = (EnumSexo) JOptionPane.showInputDialog(null, "Escolha uma Opcaoo", "Leitura de
                    Dados",
202                     JOptionPane.QUESTION_MESSAGE, null, EnumSexo.values(), alunoTemp.getSexo());
203             alunoTemp.setSexo(sexo);
204 
205             logradouro = JOptionPane.showInputDialog("ForneÃ§a o logradouro do endereco", alunoTemp.
                    getEndereco().getLogradouro().trim());
206             alunoTemp.getEndereco().setLogradouro(logradouro);
207 
208             numero = Integer.parseInt(JOptionPane.showInputDialog("ForneÃ§a o numero no endereco",
                    alunoTemp.getEndereco().getNumero()));
209             alunoTemp.getEndereco().setNumero(numero);
210 
211             bairro = JOptionPane.showInputDialog("ForneÃ§a o bairro no endereco", alunoTemp.
                    getEndereco().getBairro().trim());
212             alunoTemp.getEndereco().setBairro(bairro);
213 
214             cidade = JOptionPane.showInputDialog("ForneÃ§a a cidade no endereco", alunoTemp.
                    getEndereco().getCidade().trim());
215             alunoTemp.getEndereco().setCidade(cidade);
216 
217             uf = (EnumUF) JOptionPane.showInputDialog(null, "Escolha uma Opcao", "Leitura de Dados",
218                     JOptionPane.QUESTION_MESSAGE, null, EnumUF.values(), alunoTemp.getEndereco().
                            getUf());
219             alunoTemp.getEndereco().setUf(uf);
220         } catch (Exception ex) {
221             System.out.println("Digitacao inconsistente - " + ex.getMessage());
222         }
223         return alunoTemp;
224     }
225 
226     /**
227      * Cria uma nova intancia de AlunoVO e chama o metodo lerdados(AlunoVO
228      * alunoVO).
229      *
230      * @return
```

```
231         */
232        private static AlunoVO lerDados() {
233            AlunoVO alunoTemp = new AlunoVO();
234            return lerDados(alunoTemp);
235        }
236
237        /**
238         * Exibe as opcoes por meio de uma tela de dialogo.
239         *
240         * @return
241         */
242        private static EnumMenuAluno exibirMenu() {
243            EnumMenuAluno opcao;
244
245            opcao = (EnumMenuAluno) JOptionPane.showInputDialog(null, "Escolha uma Opcao", "Menu",
246                    JOptionPane.QUESTION_MESSAGE, null, EnumMenuAluno.values(), EnumMenuAluno.values()
                        [0]);
247            if (opcao == null) {
248                JOptionPane.showMessageDialog(null, "Nenhuma Opcao Escolhida");
249                opcao = EnumMenuAluno.Sair;
250            }
251            return opcao;
252        }
253    }
```

Código 13 – Principal.java

6 Questões sobre a Implementação

1. Qual a justificativa/vantagem do uso de enumerações, ao invés de usar valores literais ou constantes do tipo String ou Inteiro?
2. Qual o papel das classes VO's na implementação fornecida?
3. Caso não seja usado VO, como seriam passados valores entre as classes da visão, negócio e persistência?
4. Que tipo de relação existe entre o VO de Aluno e Endereço?
5. O que justifica o uso do EnderecoVO na implementação, haja visto que no banco de dados existe apenas uma tabela (Aluno)?
6. Aponte exemplos de uso da classe PersistenciaException.
7. Qual o benefício de criar uma classe para tratar a conexão com o banco de dados, como a classe ConexaoBD?
8. Qual o papel/responsabilidade da classe DAO?
9. Qual o papel/responsabilidade da classe AlunoNegocio?
10. Caso tenha que inserir uma nova entidade/conceito na aplicação, quais seriam as classes a serem implementadas?