Relatório de Atividade Prática Nível 3

Idelmar Fernando de Souza - 202302114431

Campus Santa Rosa - RS
Backend sem banco não tem – 3º Semestre

Objetivo da Prática

Desenvolver uma aplicação em Java, com o objetivo de inserir cadastro de pessoas com atributos como pessoa física ou jurídica, com métodos de inserir, alterar, excluir e recuperar os dados existentes, persistindo em banco de dados;

1º Procedimento | Mapeamento Objeto-Relacional e DAO

Classe CadastroBDTeste

Nessa primeira parte a classe main, cria objeto de pessoa física e jurídica setando valores fixos. Realiza operações de inclusão, alteração, recuperação e exclusão, e imprime os resultados no console, para isso usa os métodos incluir, alterar, getPessoas e excluir. SE conecta com o banco através da classe ConectorDB;

Classe Pessoa

Cria o objeto pessoa, com atributos idpessoa, nome, logradouro, cidade, estado, telefone e e-mail;

Traz os métodos públicos exibir(), e os get's e set's de todos os atributos da classe.

```
package cadastrobd.model;

public class Pessoa {

    private int idpessoa;

    private String nome;

    private int logradouro;

    private int cidade;

    private int estado;

    private String telefone;

    private String email;
```

```
    public Pessoa() {
    }
     public Pessoa(int idpessoa, String nome, int logradouro, int cidade, int estado, String telefone, String email) {
        this.idpessoa = idpessoa;
        this.nome = nome;
        this.logradouro = logradouro;
        this.cidade = cidade;
        this.estado = estado;
        this.telefone = telefone;
        this.email = email;
    }
    public void exibir() {
        System.out.println("ID: " + idpessoa);
        System.out.println("Nome: " + nome);
        System.out.println("Logradouro: " + logradouro);
        System.out.println("Cidade: " + cidade);
        System.out.println("Estado: " + estado);
        System.out.println("Telefone: " + telefone);
        System.out.println("Email: " + email);
    }
    public int getId() {
        return idpessoa;
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
```

```
}
public int getLogradouro() {
    return logradouro;
}
public void setLogradouro(int logradouro) {this.logradouro = logradouro;}
public int getCidade() {
    return cidade;
}
public void setCidade(int cidade) {
    this.cidade = cidade;
}
public int getEstado() {
    return estado;
}
public void setEstado(int estado) {
    this.estado = estado;
}
public String getTelefone() {
    return telefone;
}
public void setTelefone(String telefone) {
    this.telefone = telefone;
}
public String getEmail() {
    return email;
}
public void setEmail(String email) {
    this.email = email;
}
```

}

Classe PessoaFisica

A classe é herdade de Pessoa, e adicionando o atributo cpf;

Traz os métodos set e get (cpf);

```
package cadastrobd.model;

public class PessoaFisica extends Pessoa {

    private String cpf;

    public PessoaFisica() {

    }

     public PessoaFisica(int idpessoa, String nome, int logradouro, int cidade, int estado, String telefone, String email, String cpf) {

        super(idpessoa, nome, logradouro, cidade, estado, telefone, email);

        this.cpf = cpf;

    }

    @Override

    public void exibir() {

        super.exibir();

        System.out.println("CPF: " + cpf);

    }


    public void setCpf(String cpf) {

        this.cpf = cpf;

    }

    public String getCpf() {

        return cpf;

    }

}
```

Classe PessoaJuridica

A classe é herdade de Pessoa, e adicionando o atributo cnpj;

Traz os métodos set e get (cnpj);

```
package cadastrobd.model;

public class PessoaJuridica extends Pessoa {

    private String cnpj;

    public PessoaJuridica() {

    }

     public PessoaJuridica(int id, String nome, int logradouro, int cidade, int estado, String telefone, String email, String cnpj) {

        super(id, nome, logradouro, cidade, estado, telefone, email);

        this.cnpj = cnpj;

    }

    @Override

    public void exibir() {

        super.exibir();

        System.out.println("CNPJ: " + cnpj);

    }

    public String getCnpj() {

        return cnpj;

    }

    public void setCnpj(String cnpj) {

        this.cnpj = cnpj;

    }

}
```

Classe ConectorDB

A classe responsável por gerenciar a conexão com o banco de dados; possui métodos para executar SQL. O PreparedStatement inicia a conexão e o ResultSet devolve o resultado; Com a conexão centralizada o código fica mais organizado e facilita a manutenção;

```
package cadastrobd.model.util;

import java.sql.*;

public class ConectorBD {

                                    private static final String URL =
"jdbc:sqlserver://localhost:1433;databaseName=loja;encrypt=true;trustServerCertificate=true;";

   private static final String USUARIO = "sa";

   private static final String SENHA = "loja";

   public static Connection getConnection() throws SQLException {

      return DriverManager.getConnection(URL, USUARIO, SENHA);

   }

   public static PreparedStatement getPrepared(String sql) throws SQLException {

      Connection conn = getConnection();

      return conn.prepareStatement(sql);

   }

   public static ResultSet getSelect(String sql) throws SQLException {

      PreparedStatement ps = getPrepared(sql);

      return ps.executeQuery();

   }

   public static void main(String[] args) {

      try {

         ResultSet rs = getSelect("SELECT * FROM tabela_exemplo");

         while (rs.next()) {

            String nome = rs.getString("nome");

            System.out.println("Nome: " + nome);

         }

         rs.close();

      } catch (SQLException e) {

         e.printStackTrace();

      }
```

```
}

public static void close(PreparedStatement stmt) {
    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
public static void close(Connection conn, PreparedStatement stmt, ResultSet rs) {
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    if (conn != null) {
        try {
            conn.close();
```

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

Classe SequenceManager

Como o nome já diz, a classe gerencia a sequencia de id da tabela. Se conecta com o banco e faz uma consulta SQL obtendo o id e adicionando a sequencia;

```
import cadastrobd.model.util.ConectorBD;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class SequenceManager {
   public static int getNextId(String tableName) {
      Connection conn = null;
      PreparedStatement stmt = null;
      ResultSet rs = null;
      int nextId = -1;

      ty {
         conn = ConectorBD.getConnection();
         String sql = "SELECT IDENT_CURRENT('" + tableName + "') + 1 AS nextId";
         stmt = conn.prepareStatement(sql);
         rs = stmt.executeQuery();
         if (rs.next()) {
```

```
                nextId = rs.getInt("nextId");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            close(conn, stmt, rs);
        }
        return nextId;
    }
    private static void close(Connection conn, PreparedStatement stmt, ResultSet rs) {
        try {
            if (rs != null) {
                rs.close();
            }
            if (stmt != null) {
                stmt.close();
            }
            if (conn != null) {
                conn.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Classe PessoaFisicaDao

É a classe utiliza responsável por acessar o banco, fazendo CRUD, para a classe PessoaFisica, além dos métodos incluir, alterar e excluir ela possui o método getPessoa() que recuperar um objeto com base no id passado e getPessoas que recupera todos os dados cadastrados.

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import cadastrobd.model.PessoaFisica;
import cadastrobd.model.util.ConectorBD;

public class PessoaFisicaDAO {

    public static PessoaFisica getPessoa(int idpessoa) {
        Connection conn = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;
        PessoaFisica pessoa = null;

        try {
            conn = ConectorBD.getConnection();
            String sql = """
                    SELECT pes.nome, pes.logradouro, pes.cidade, pes.estado, pes.telefone, pes.email, pef.cpf\s
                    FROM pessoafisica pef\s
                    INNER JOIN pessoa pes ON (pes.idpessoa = pef.idpessoa)
                    WHERE pes.idpessoa=?""";

            stmt = conn.prepareStatement(sql);
            stmt.setInt(1, idpessoa);
            rs = stmt.executeQuery();
```

```
        if (rs.next()) {
            pessoa = new PessoaFisica(
                    idpessoa,
                    rs.getString("nome"),
                    rs.getInt("logradouro"),
                    rs.getInt("cidade"),
                    rs.getInt("estado"),
                    rs.getString("telefone"),
                    rs.getString("email"),
                    rs.getString("cpf")
            );
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        ConectorBD.close(conn, stmt, rs);
    }

    return pessoa;
}

public static List<PessoaFisica> getPessoas() {
    Connection conn = null;
    PreparedStatement stmt = null;
    ResultSet rs = null;
    List<PessoaFisica> pessoas = new ArrayList<>();

    try {
```

```
            conn = ConectorBD.getConnection();
            String sql = """
                        SELECT pes.nome, pes.logradouro, pes.cidade, pes.estado, pes.telefone, pes.email,
pef.cpf\s
                        FROM pessoafisica pef\s
                        INNER JOIN pessoa pes ON (pes.idpessoa = pef.idpessoa)
                       \s""";
            stmt = conn.prepareStatement(sql);
            rs = stmt.executeQuery();
             while (rs.next()) {
               PessoaFisica pessoa = new PessoaFisica(
                       rs.getInt("idpessoa"),
                       rs.getString("nome"),
                       rs.getInt("logradouro"),
                       rs.getInt("cidade"),
                       rs.getInt("estado"),
                       rs.getString("telefone"),
                       rs.getString("email"),
                       rs.getString("cpf")
               );
               pessoas.add(pessoa);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            ConectorBD.close(conn, stmt, rs);
        }

        return pessoas;
```

```
    }
        public static void incluir(PessoaFisica pessoa) {
        Connection conn = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;

        try {
            conn = ConectorBD.getConnection();
              String sqlPessoa = "INSERT INTO Pessoa (nome, logradouro, cidade, estado, telefone, email,
tipo) VALUES (?, ?, ?, ?, ?, ?, 'F')";
            stmt = conn.prepareStatement(sqlPessoa, Statement.RETURN_GENERATED_KEYS);
            stmt.setString(1, pessoa.getNome());
            stmt.setInt(2, pessoa.getLogradouro());
            stmt.setInt(3, pessoa.getCidade());
            stmt.setInt(4, pessoa.getEstado());
            stmt.setString(5, pessoa.getTelefone());
            stmt.setString(6, pessoa.getEmail());
            stmt.executeUpdate();

            rs = stmt.getGeneratedKeys();
            int idPessoa = 0;
            if (rs.next()) {
               idPessoa = rs.getInt(1);
            }

            String sqlPessoaFisica = "INSERT INTO PessoaFisica (idpessoa, cpf) VALUES (?, ?)";
            stmt = conn.prepareStatement(sqlPessoaFisica);
            stmt.setInt(1, idPessoa);
            stmt.setString(2, pessoa.getCpf());
```

```
            stmt.executeUpdate();

            System.out.println("Pessoa física inserida com sucesso:");
            System.out.println("Nome: " + pessoa.getNome());
            System.out.println("Logradouro: " + pessoa.getLogradouro());
            System.out.println("Cidade: " + pessoa.getCidade());
            System.out.println("Estado: " + pessoa.getEstado());
            System.out.println("Telefone: " + pessoa.getTelefone());
            System.out.println("Email: " + pessoa.getEmail());
            System.out.println("CPF: " + pessoa.getCpf());
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            ConectorBD.close(conn, stmt, rs);
        }
    }

    public static void alterar(PessoaFisica pessoa) {
        Connection conn = null;
        PreparedStatement stmt = null;

        try {
            conn = ConectorBD.getConnection();
            String sqlPessoaFisica = "UPDATE PessoaFisica SET cpf=? WHERE idpessoa=?";
            PreparedStatement stmt2 = conn.prepareStatement(sqlPessoaFisica);
            stmt2.setString(1, pessoa.getCpf());
            stmt2.setInt(2, pessoa.getId());
            stmt2.executeUpdate();
```

```
            String sqlPessoa = "UPDATE Pessoa SET nome=?, logradouro=?, cidade=?, estado=?,
telefone=?, email=? WHERE idpessoa=?";
            stmt = conn.prepareStatement(sqlPessoa);
            stmt.setString(1, pessoa.getNome());
            stmt.setInt(2, pessoa.getLogradouro());
            stmt.setInt(3, pessoa.getCidade());
            stmt.setInt(4, pessoa.getEstado());
            stmt.setString(5, pessoa.getTelefone());
            stmt.setString(6, pessoa.getEmail());
            stmt.setInt(7, pessoa.getId());
            stmt.executeUpdate();

            ConectorBD.close(stmt2);
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            ConectorBD.close(conn, stmt, null);
        }
    }

    public static void excluir(int idpessoa) {
        Connection conn = null;
        PreparedStatement stmt = null;

        try {
            conn = ConectorBD.getConnection();
            String sqlPessoaFisica = "DELETE FROM PessoaFisica WHERE idpessoa=?";
            stmt = conn.prepareStatement(sqlPessoaFisica);
            stmt.setInt(1, idpessoa);
```

```
            stmt.executeUpdate();

            String sqlPessoa = "DELETE FROM Pessoa WHERE idpessoa=?";
            stmt = conn.prepareStatement(sqlPessoa);
            stmt.setInt(1, idpessoa);
            stmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            ConectorBD.close(conn, stmt, null);
        }
    }
}
```

Classe PessoaJuridicaDao

Da mesma forma que PessoaFisicaDao, PessoaJuridicaDao é a classe utiliza responsável por acessar o banco, fazendo CRUD, para a classe PessoaJuridica, além dos métodos incluir,alterar e excluir ela possui o método getPessoa() que recuperar um objeto com base no id passado e getPessoas que recupera todos os dados cadastrados.

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import cadastrobd.model.PessoaJuridica;
import cadastrobd.model.util.ConectorBD;

public class PessoaJuridicaDAO {

    public static PessoaJuridica getPessoa(int idpessoa) {
```

```
        Connection conn = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;
        PessoaJuridica pessoa = null;

        try {
            conn = ConectorBD.getConnection();
            String sql = """
                        SELECT pes.nome, pes.logradouro, pes.cidade, pes.estado, pes.telefone, pes.email,
pej.cnpj
                    FROM pessoa pes
                    INNER JOIN PessoaJuridica pej ON (pes.idpessoa = pej.idpessoa)
                    WHERE pes.idpessoa=?""";

            stmt = conn.prepareStatement(sql);
            stmt.setInt(1, idpessoa);
            rs = stmt.executeQuery();

            if (rs.next()) {
                pessoa = new PessoaJuridica(
                        idpessoa,
                        rs.getString("nome"),
                        rs.getInt("logradouro"),
                        rs.getInt("cidade"),
                        rs.getInt("estado"),
                        rs.getString("telefone"),
                        rs.getString("email"),
                        rs.getString("cnpj")
                );
```

```
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        ConectorBD.close(conn, stmt, rs);
    }

    return pessoa;
}

public static List<PessoaJuridica> getPessoas() {
    Connection conn = null;
    PreparedStatement stmt = null;
    ResultSet rs = null;
    List<PessoaJuridica> pessoas = new ArrayList<>();

    try {
        conn = ConectorBD.getConnection();
        String sql = """
                SELECT pes.idpessoa, pes.nome, pes.logradouro, pes.cidade, pes.estado, pes.telefone,
pes.email, pej.cnpj
                FROM pessoa pes
                INNER JOIN PessoaJuridica pej ON (pes.idpessoa = pej.idpessoa)
                WHERE pes.tipo = 'J'""";
        stmt = conn.prepareStatement(sql);
        rs = stmt.executeQuery();

        while (rs.next()) {
            PessoaJuridica pessoa = new PessoaJuridica(
```

```
                rs.getInt("idpessoa"),
                rs.getString("nome"),
                rs.getInt("logradouro"),
                rs.getInt("cidade"),
                rs.getInt("estado"),
                rs.getString("telefone"),
                rs.getString("email"),
                rs.getString("cnpj")
            );
            pessoas.add(pessoa);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        ConectorBD.close(conn, stmt, rs);
    }

    return pessoas;
}

public static void incluir(PessoaJuridica pessoa) {
    Connection conn = null;
    PreparedStatement stmt = null;
    ResultSet rs = null;

    try {
        conn = ConectorBD.getConnection();
            String sqlPessoa = "INSERT INTO Pessoa (nome, logradouro, cidade, estado, telefone, email,
tipo) VALUES (?, ?, ?, ?, ?, ?, 'J')";
```

```
stmt = conn.prepareStatement(sqlPessoa, Statement.RETURN_GENERATED_KEYS);
stmt.setString(1, pessoa.getNome());
stmt.setInt(2, pessoa.getLogradouro());
stmt.setInt(3, pessoa.getCidade());
stmt.setInt(4, pessoa.getEstado());
stmt.setString(5, pessoa.getTelefone());
stmt.setString(6, pessoa.getEmail());
stmt.executeUpdate();

rs = stmt.getGeneratedKeys();
int idPessoa = 0;
if (rs.next()) {
    idPessoa = rs.getInt(1);
}

String sqlPessoaJuridica = "INSERT INTO PessoaJuridica (idpessoa, cnpj) VALUES (?, ?)";
stmt = conn.prepareStatement(sqlPessoaJuridica);
stmt.setInt(1, idPessoa);
stmt.setString(2, pessoa.getCnpj());
stmt.executeUpdate();

System.out.println("Pessoa jurídica inserida com sucesso:");
System.out.println("Nome: " + pessoa.getNome());
System.out.println("Logradouro: " + pessoa.getLogradouro());
System.out.println("Cidade: " + pessoa.getCidade());
System.out.println("Estado: " + pessoa.getEstado());
System.out.println("Telefone: " + pessoa.getTelefone());
System.out.println("Email: " + pessoa.getEmail());
System.out.println("CNPJ: " + pessoa.getCnpj());
```

```
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            ConectorBD.close(conn, stmt, rs);
        }
    }

    public static void alterar(PessoaJuridica pessoa) {
        Connection conn = null;
        PreparedStatement stmt = null;

        try {
            conn = ConectorBD.getConnection();
            String sqlPessoa = "UPDATE Pessoa SET nome=?, logradouro=?, cidade=?, estado=?, telefone=?,
email=? WHERE idpessoa=?";
            stmt = conn.prepareStatement(sqlPessoa);
            stmt.setString(1, pessoa.getNome());
            stmt.setInt(2, pessoa.getLogradouro());
            stmt.setInt(3, pessoa.getCidade());
            stmt.setInt(4, pessoa.getEstado());
            stmt.setString(5, pessoa.getTelefone());
            stmt.setString(6, pessoa.getEmail());
            stmt.setInt(7, pessoa.getId());
            stmt.executeUpdate();

            String sqlPessoaJuridica = "UPDATE PessoaJuridica SET cnpj=? WHERE idpessoa=?";
            PreparedStatement stmt2 = conn.prepareStatement(sqlPessoaJuridica);
            stmt2.setString(1, pessoa.getCnpj());
            stmt2.setInt(2, pessoa.getId());
```

```
            stmt2.executeUpdate();


            ConectorBD.close(stmt2);
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            ConectorBD.close(conn, stmt, null);
        }

    }

    public static void excluir(int idpessoa) {
        Connection conn = null;
        PreparedStatement stmt = null;

        try {
            conn = ConectorBD.getConnection();
            String sqlPessoaJuridica = "DELETE FROM PessoaJuridica WHERE idpessoa=?";
            stmt = conn.prepareStatement(sqlPessoaJuridica);
            stmt.setInt(1, idpessoa);
            stmt.executeUpdate();

            String sqlPessoa = "DELETE FROM Pessoa WHERE idpessoa=?";
            stmt = conn.prepareStatement(sqlPessoa);
            stmt.setInt(1, idpessoa);
            stmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
```

```
            ConectorBD.close(conn, stmt, null);
        }
    }
}
```

a) Qual a importância dos componentes de middleware, como o JDBC?

Permite interação com bancos de dados, aumenta a segurança tendo autenticação e facilita a manutenção de aplicações maiores pois centraliza as movimentações;

b) Qual a diferença no uso de *Statement* ou *PreparedStatement* para a manipulação de dados?

O Statement é usado para consultas que não usam parâmetros, mais simples e desempenho pode ser afetado se houver muitas consultas; O PreparedStatement é usado para fazer consultas com parâmetros tornando a consulta dinâmica e a consulta é compilado uma vez, com desempenho melhor.

c) Como o padrão DAO melhora a manutenibilidade do software?

No padrão DAO, cada classe tem sua classe DAO, que centraliza o acesso de dados melhorando a legibilidade e a manutenção, principalmente em projetos maiores;

d) Como a herança é refletida no banco de dados, quando lidamos com um modelo estritamente relacional?

A herança é representado por chaves estrangeiras (FK's) nas tabelas do banco relacional.

2º Procedimento | Alimentando a Base

Classe CadastroBDTeste

Classe main, trazendo pra o usuário um scanner para as opções no console onde pode escolher entre incluir, alterar, excluir, exibir escolhendo o id, exibir todos os cadastros e cancelar. O switch faz a chamada das respectivas funções conforme escolha do usuário.

A função incluirOpçao() deriva-se em incluirPessoaFisica() e incluirPessoaJuridica(); ambas criando um novo objeto de pessoa e setando os valores correspondentes;

A função alterarOpcao() recupera o cadastro através do id, e possibilita alterar os dados recuperados.

A função excluirOpcao() recupera o cadastro através do id e possibilita excluir os dados recuperados.

A função exibirPorIdOpcao() recupera o cadastro através do id e exibe no console.

A Função exibirTodosOpcao() recupera todos os dados cadastrados em banco.

```
package cadastrobd.model;


public class Pessoa {
   private int idpessoa;
   private String nome;
   private int logradouro;
   private int cidade;
   private int estado;
   private String telefone;
   private String email;

   public Pessoa() {
   }

    public Pessoa(int idpessoa, String nome, int logradouro, int cidade, int estado, String telefone, String email) {
        this.idpessoa = idpessoa;
        this.nome = nome;
        this.logradouro = logradouro;
        this.cidade = cidade;
        this.estado = estado;
        this.telefone = telefone;
```

```
    this.email = email;
}

public void exibir() {
    System.out.println("ID: " + idpessoa);
    System.out.println("Nome: " + nome);
    System.out.println("Logradouro: " + logradouro);
    System.out.println("Cidade: " + cidade);
    System.out.println("Estado: " + estado);
    System.out.println("Telefone: " + telefone);
    System.out.println("Email: " + email);
}

public int getId() {
    return idpessoa;
}
public String getNome() {
    return nome;
}
public void setNome(String nome) {
    this.nome = nome;
}
public int getLogradouro() {
    return logradouro;
}
public void setLogradouro(int logradouro) {this.logradouro = logradouro;}
public int getCidade() {
    return cidade;
}
```

```
    public void setCidade(int cidade) {
        this.cidade = cidade;
    }
    public int getEstado() {
        return estado;
    }
    public void setEstado(int estado) {
        this.estado = estado;
    }
    public String getTelefone() {
        return telefone;
    }
    public void setTelefone(String telefone) {
        this.telefone = telefone;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
```

As outras classes praticamente se mantiveram como na primeira parte;

Respostas:

a) Quais as diferenças entre a persistência em arquivo e a persistência em banco de dados?

A persistência em arquivo é usada para armazenar os dados no arquivo escolhido, por exemplo: json; No banco de dados a persistência é feita através de tabelas no modelo relacional e o não relacional se usa outras formas como chave e valor.

b) Como o uso de operador *lambda* simplificou a impressão dos valores contidos nas entidades, nas versões mais recentes do Java?

A declaração lambda deixa o código mais limpo e facilita o visualizar das funções, a estrutura fica mais performática.

c) Por que métodos acionados diretamente pelo método main, sem o uso de um objeto, precisam ser marcados como *static*?

Porque o método main que inicia a aplicação, não tendo nenhum objeto para referenciar, e o static serve para isso pois pode ser chamado sem classe

Conclusão

A aplicação usa-se de ferramentas disponíveis da linguam como o scanner, do while, faz tratamento de exceções com try catch. Possui as classes e métodos bem definidos. As funcionalidades da aplicação funcionam perfeitamente, então o usuário consegue cadastrar, alterar e exibir os dados existentes no banco de dados; Traz um visual amigável na medida do possível, mostrando mensagens ao usuário para informar as ações sendo executadas. Um ponto de melhoria seria a validação dos dados passados do usuário, para evitar ´problemas quando for persistir os dados no banco.