Faculdade Estácio - Polo Curitiba - Centro

Curso: Desenvolvimento Full Stack
Disciplina: Vamos manter as informações
Número da Turma: RPG0016
Semestre Letivo: 3
Integrante: Mariana Lucas Fernandes Onório
Repositório: https://github.com/MariLFO/estacio-mundo3-missao-nivel-3

Sumário:

1. Título da Prática:

RPG0016 - BackEnd sem banco não tem

Criação de aplicativo Java, com acesso ao banco de dados
SQL Server através do middleware JDBC.

2. Objetivos da Prática:

1. Implementar persistência com base no middleware JDBC.
2. Utilizar o padrão DAO (Data Access Object) no manuseio de dados.
3. Implementar o mapeamento objeto-relacional em sistemas Java.
4. Criar sistemas cadastrais com persistência em banco relacional.
5. No final do exercício, o aluno terá criado um aplicativo cadastral com uso do SQL Server na persistência de dados.

3. Códigos do roteiro:

Arquivo: Pessoa.java

```
package cadastrobd.model;

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

   public Pessoa() {
       this.id = 0;
       this.nome = "";
       this.logradouro = "";
       this.cidade = "";
       this.estado = "";
       this.telefone = "";
       this.email = "";
   }

   public Pessoa(int id, String nome, String logradouro, String cidade, String estado, String telefone,
String email) {
       this.id = id;
       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: " + this.id);
        System.out.println("Nome: " + this.nome);
        System.out.println("Logradouro: " + this.logradouro);
        System.out.println("Cidade: " + this.cidade);
        System.out.println("Estado: " + this.estado);
        System.out.println("Telefone: " + this.telefone);
        System.out.println("Email: " + this.email);
    }

    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;
    }

    public String getLogradouro() {
        return logradouro;
    }

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

    public String getCidade() {
        return cidade;
    }

    public void setCidade(String cidade) {
        this.cidade = cidade;
    }
    public String getEstado() {
        return estado;
    }
```

```
    public void setEstado(String 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;
    }
}
```

Arquivo: PessoaFisica.java

```
package cadastrobd.model;

public class PessoaFisica extends Pessoa {
    private String cpf;

    public PessoaFisica() {
        super();
        this.cpf = "";
    }

    public PessoaFisica(int id, String nome, String logradouro, String cidade, String estado, String
telefone, String email, String cpf) {
        super(id, nome, logradouro, cidade, estado, telefone, email);
        this.cpf = cpf;
    }

    @Override
    public void exibir() {
        super.exibir();
        System.out.println("CPF: " + this.cpf);
    }

    public String getCpf() {
        return cpf;
    }
```

```
    public void setCpf(String cpf) {
        this.cpf = cpf;
    }
}
```

Arquivo: PessoaJuridica.java

```
package cadastrobd.model;

public class PessoaJuridica extends Pessoa {
    private String cnpj;

    public PessoaJuridica() {
        super();
        this.cnpj = "";
    }

    public PessoaJuridica(int id, String nome, String logradouro, String cidade, String 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: " + this.cnpj);
    }
    public String getCnpj() {
        return cnpj;
    }
    public void setCnpj(String cnpj) {
        this.cnpj = cnpj;
    }
}
```

Arquivo: ConectorBD.java

```
package cadastrobd.model.util;

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

public class ConectorBD {
    private static final String DRIVER = "com.microsoft.sqlserver.jdbc.SQLServerDriver";
```

```
    private static final String URL =
"jdbc:sqlserver://localhost\\MSSQL:1433;databaseName=loja;encrypt=true;trustServerCertificate=true;";
    private static final String USER = "loja";
    private static final String PASSWORD = "loja";

    public static Connection getConnection() {
        try {
            // Carrega o driver JDBC na memória
            Class.forName(DRIVER).newInstance();
            // Retorna uma conexão com o banco de dados
            return DriverManager.getConnection(URL, USER, PASSWORD);
        } catch (ClassNotFoundException | SQLException e) {
            System.out.println("Erro ao conectar com o banco de dados: " + e.getMessage());
            return null;
        } catch (InstantiationException e) {
            throw new RuntimeException(e);
        } catch (IllegalAccessException e) {
            throw new RuntimeException(e);
        }
    }

    public static PreparedStatement getPrepared(Connection conexao, String sql) {
        try {
            return conexao.prepareStatement(sql);
        } catch (SQLException e) {
            System.out.println("Erro ao preparar o SQL: " + e.getMessage());
            return null;
        }
    }

    public static ResultSet getSelect(PreparedStatement consulta) {
        try {
            return consulta.executeQuery();
        } catch (SQLException e) {
            System.out.println("Erro ao executar a consulta: " + e.getMessage());
            return null;
        }
    }

    public static void close(PreparedStatement statement) {
        try {
            if (statement != null) {
                statement.close();
            }
        } catch (SQLException e) {
            System.out.println("Erro ao fechar o Statement: " + e.getMessage());
        }
    }

    public static void close(ResultSet resultado) {
        try {
            if (resultado != null) {
```

```
                resultado.close();
            }
        } catch (SQLException e) {
            System.out.println("Erro ao fechar o ResultSet: " + e.getMessage());
        }
    }

    public static void close(Connection con) {
        try {
            if (con != null) {
                con.close();
            }
        } catch (SQLException e) {
            System.out.println("Erro ao fechar a conexão: " + e.getMessage());
        }
    }
}
```

Arquivo: SequenceManager.java

```
package cadastrobd.model.util;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class SequenceManager {

    public static int getValue(String sequence) {
        try {
            Connection conexao = ConectorBD.getConnection();

            // Verifica se a conexão é válida
            if (conexao == null) {
                // Se a conexão for nula, retorna -1
                return -1;
            }

            // Cria um SQL para consultar o próximo valor da sequência
            String sql = "SELECT NEXT VALUE FOR dbo." + sequence;
            PreparedStatement consulta = ConectorBD.getPrepared(conexao, sql);
            ResultSet resultado = ConectorBD.getSelect(consulta);

            // Verifica se o ResultSet é válido e contém algum dado
            if (resultado == null || !resultado.next()) {
                // Se o ResultSet for nulo ou não contiver dados, retorna -1
                ConectorBD.close(conexao);
                return -1;
            }
```

```
            // Obtém o próximo valor da sequência como um inteiro
            int value = resultado.getInt(1);

            // Fecha os objetos ResultSet, PreparedStatement e Connection
            ConectorBD.close(resultado);
            ConectorBD.close(consulta);
            ConectorBD.close(conexao);

            return value;
        } catch (SQLException e) {
            System.out.println("Erro ao obter o valor da sequência: " + e.getMessage());
            return -1;
        }
    }
}
```

Arquivo: PessoaFisicaDAO.java

```
package cadastrobd.model;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import cadastrobd.model.util.ConectorBD;
import cadastrobd.model.util.SequenceManager;

public class PessoaFisicaDAO {
    public PessoaFisica getPessoa(int id) {
        try {
            Connection conexao = ConectorBD.getConnection();

            // Verifica se a conexão é válida
            if (conexao == null) {
                // Se a conexão for nula, retorna null
                return null;
            }

            // Cria um SQL para consultar os dados da pessoa física pelo id
            String sql = "SELECT * FROM Pessoa p INNER JOIN PessoaFisica pf ON p.idPessoa = pf.idPessoa
WHERE p.idPessoa = ?";

            // Obtém um objeto PreparedStatement com o SQL criado e o id fornecido como parâmetro
            PreparedStatement prepared = ConectorBD.getPrepared(conexao, sql);
            prepared.setInt(1, id);

            // Executa a consulta e obtém um objeto ResultSet com o resultado
```

```
            ResultSet resultSet = ConectorBD.getSelect(prepared);

            // Verifica se o ResultSet é válido e contém algum dado
            if (resultSet != null && resultSet.next()) {
                // Cria um objeto PessoaFisica com os dados obtidos do ResultSet
                PessoaFisica pessoaFisica = criaPessoaFisica(resultSet);

                // Fecha os objetos ResultSet, PreparedStatement e Connection
                ConectorBD.close(resultSet);
                ConectorBD.close(prepared);
                ConectorBD.close(conexao);
                return pessoaFisica;
            }

            // Se o ResultSet for nulo ou vazio, fecha os objetos PreparedStatement e Connection e
retorna null
            ConectorBD.close(prepared);
            ConectorBD.close(conexao);
            return null;
        } catch (SQLException e) {
            // Se ocorrer algum erro, imprime a mensagem no console e retorna null
            System.out.println("Erro ao obter a pessoa física pelo id: " + e.getMessage());
            return null;
        }
    }

    public List<PessoaFisica> getPessoas() {
        try {
            // Obtém uma conexão com o banco de dados
            Connection conexao = ConectorBD.getConnection();

            // Verifica se a conexão é válida
            if (conexao == null) {
                // Se a conexão for nula, retorna null
                return null;
            }

            // Cria um SQL para consultar todos os dados das pessoas físicas
            String sql = "SELECT * FROM Pessoa p INNER JOIN PessoaFisica pf ON p.idPessoa = pf.idPessoa";

            // Obtém um objeto PreparedStatement com o SQL criado
            PreparedStatement prepared = conexao.prepareStatement(sql);

            // Executa a consulta e obtém um objeto ResultSet com o resultado
            ResultSet resultSet = ConectorBD.getSelect(prepared);

            // Cria uma lista de objetos PessoaFisica para armazenar os dados obtidos
            List<PessoaFisica> pessoas = new ArrayList<>();

            // Percorre o ResultSet enquanto houver dados
            while (resultSet != null && resultSet.next()) {
                // Cria um objeto PessoaFisica com os dados obtidos do ResultSet
```

```
                PessoaFisica pessoaFisica = criaPessoaFisica(resultSet);
                pessoas.add(pessoaFisica);
            }

            // Fecha os objetos ResultSet, PreparedStatement e Connection
            ConectorBD.close(resultSet);
            ConectorBD.close(prepared);
            ConectorBD.close(conexao);

            // Retorna a lista de objetos PessoaFisica criada
            return pessoas;
        } catch (SQLException e) {
            // Se ocorrer algum erro, imprime a mensagem no console e retorna null
            System.out.println("Erro ao obter todas as pessoas físicas: " + e.getMessage());
            return null;
        }
    }

    public boolean incluir(PessoaFisica pessoaFisica) {
        try {
            Integer nextId = SequenceManager.getValue("PessoaSequence");

            if (nextId == -1) {
                // Se não foi possível obter o próximo id da sequência, retorna false
                return false;
            }

            pessoaFisica.setId(nextId);
            Connection conexao = ConectorBD.getConnection();

            // Verifica se a conexão é válida
            if (conexao == null) {
                // Se a conexão for nula, retorna false
                return false;
            }

            // Cria um SQL para inserir os dados da pessoa na tabela Pessoa
            String sql = "INSERT INTO Pessoa (idPessoa, nome, telefone, email, logradouro, cidade,
estado) VALUES (?, ?, ?, ?, ?, ?, ?)";

            // Obtém um objeto PreparedStatement com o SQL criado e os dados da pessoa física fornecida
como parâmetro
            PreparedStatement prepared = ConectorBD.getPrepared(conexao, sql);
            prepared.setInt(1, pessoaFisica.getId());
            prepared.setString(2, pessoaFisica.getNome());
            prepared.setString(3, pessoaFisica.getTelefone());
            prepared.setString(4, pessoaFisica.getEmail());
            prepared.setString(5, pessoaFisica.getLogradouro());
            prepared.setString(6, pessoaFisica.getCidade());
            prepared.setString(7, pessoaFisica.getEstado());

            // Executa a inserção e verifica se foi bem sucedida
```

```
            if (prepared.executeUpdate() <= 0) {
                // Se a inserção na tabela Pessoa falhou, fecha os objetos PreparedStatement e Connection
e retorna false
                ConectorBD.close(prepared);
                ConectorBD.close(conexao);
                return false;
            }

            // Se a inserção na tabela Pessoa foi bem sucedida, cria um SQL para inserir os dados da
pessoa física na tabela PessoaFisica
            sql = "INSERT INTO PessoaFisica (idPessoa, cpf) VALUES (?, ?)";

            // Obtém um objeto PreparedStatement com o SQL criado e os dados da pessoa física fornecida
como parâmetro
            prepared = ConectorBD.getPrepared(conexao, sql);
            prepared.setInt(1, nextId);
            prepared.setString(2, pessoaFisica.getCpf());

            // Executa a inserção e verifica se foi bem sucedida
            if (prepared.executeUpdate() <= 0) {
                // Se a inserção na tabela PessoaFisica falhou, fecha os objetos PreparedStatement e
Connection e retorna false
                ConectorBD.close(prepared);
                ConectorBD.close(conexao);
                return false;
            }

            // Se a inserção na tabela PessoaFisica foi bem sucedida, fecha os objetos PreparedStatement
e Connection e retorna true
            ConectorBD.close(prepared);
            ConectorBD.close(conexao);
            return true;
        } catch (SQLException e) {
            // Se ocorrer algum erro, imprime a mensagem no console e retorna false
            System.out.println("Erro ao incluir a pessoa física: " + e.getMessage());
            return false;
        }
    }

    public boolean alterar(PessoaFisica pessoaFisica) {
        try {
            Connection conexao = ConectorBD.getConnection();

            // Verifica se a conexão é válida
            if (conexao == null) {
                // Se a conexão for nula, retorna false
                return false;
            }

            // Cria um SQL para atualizar os dados da pessoa na tabela Pessoa
            String sql = "UPDATE Pessoa SET nome = ?, telefone = ?, email = ?, logradouro = ?, cidade =
?, estado = ? WHERE idPessoa = ?";
```

```
            // Obtém um objeto PreparedStatement com o SQL criado e os dados da pessoa física fornecida
como parâmetro
            PreparedStatement prepared = ConectorBD.getPrepared(conexao, sql);
            prepared.setString(1, pessoaFisica.getNome());
            prepared.setString(2, pessoaFisica.getTelefone());
            prepared.setString(3, pessoaFisica.getEmail());
            prepared.setString(4, pessoaFisica.getLogradouro());
            prepared.setString(5, pessoaFisica.getCidade());
            prepared.setString(6, pessoaFisica.getEstado());
            prepared.setInt(7, pessoaFisica.getId());

            // Executa a atualização e verifica se foi bem sucedida
            if (prepared.executeUpdate() <= 0) {
                // Se a atualização na tabela Pessoa falhou, fecha os objetos PreparedStatement e
Connection e retorna false
                ConectorBD.close(prepared);
                ConectorBD.close(conexao);
                return false;
            }

            // Se a atualização na tabela Pessoa foi bem sucedida, cria um SQL para atualizar os dados da
pessoa física na tabela PessoaFisica
            sql = "UPDATE PessoaFisica SET cpf = ? WHERE idPessoa = ?";

            // Obtém um objeto PreparedStatement com o SQL criado e os dados da pessoa física fornecida
como parâmetro
            prepared = ConectorBD.getPrepared(conexao, sql);
            prepared.setString(1, pessoaFisica.getCpf());
            prepared.setInt(2, pessoaFisica.getId());

            // Executa a atualização e verifica se foi bem sucedida
            if (prepared.executeUpdate() <= 0) {
                // Se a atualização na tabela PessoaFisica falhou, fecha os objetos PreparedStatement e
Connection e retorna false
                ConectorBD.close(prepared);
                ConectorBD.close(conexao);
                return false;
            }

            // Se a atualização na tabela PessoaFisica foi bem sucedida, fecha os objetos
PreparedStatement e Connection e retorna true
            ConectorBD.close(prepared);
            ConectorBD.close(conexao);
            return true;
        } catch (SQLException e) {
            // Se ocorrer algum erro, imprime a mensagem no console e retorna false
            System.out.println("Erro ao alterar a pessoa física: " + e.getMessage());
            return false;
        }
    }
```

```java
    public boolean excluir(int id) {
        try {
            Connection conexao = ConectorBD.getConnection();

            // Verifica se a conexão é válida
            if (conexao == null) {
                // Se a conexão for nula, retorna false
                return false;
            }

            // Cria um SQL para excluir os dados da pessoa física na tabela PessoaFisica
            String sql = "DELETE FROM PessoaFisica WHERE idPessoa = ?";

            // Obtém um objeto PreparedStatement com o SQL criado e o id fornecido como parâmetro
            PreparedStatement prepared = ConectorBD.getPrepared(conexao, sql);
            prepared.setInt(1, id);

            // Executa a exclusão e verifica se foi bem sucedida
            if (prepared.executeUpdate() <= 0) {
                // Se a exclusão na tabela PessoaFisica falhou, fecha os objetos PreparedStatement e
Connection e retorna false
                ConectorBD.close(prepared);
                ConectorBD.close(conexao);
                return false;
            }

            // Se a exclusão na tabela PessoaFisica foi bem sucedida, cria um SQL para excluir os dados
da pessoa na tabela Pessoa
            sql = "DELETE FROM Pessoa WHERE idPessoa = ?";

            // Obtém um objeto PreparedStatement com o SQL criado e o id fornecido como parâmetro
            prepared = ConectorBD.getPrepared(conexao, sql);
            prepared.setInt(1, id); // Substitua 1 pelo índice do campo id na tabela Pessoa

            // Executa a exclusão e verifica se foi bem sucedida
            if (prepared.executeUpdate() <= 0) {
                // Se a exclusão na tabela Pessoa falhou, fecha os objetos PreparedStatement e Connection
e retorna false
                ConectorBD.close(prepared);
                ConectorBD.close(conexao);
                return false;
            }

            // Se a exclusão na tabela Pessoa foi bem sucedida, fecha os objetos PreparedStatement e
Connection e retorna true
            ConectorBD.close(prepared);
            ConectorBD.close(conexao);
            return true;
        } catch (SQLException e) {
            // Se ocorrer algum erro, imprime a mensagem no console e retorna false
            System.out.println("Erro ao excluir a pessoa física: " + e.getMessage());
            return false;
```

```
        }
    }

    private static PessoaFisica criaPessoaFisica(ResultSet resultSet) throws SQLException {
        PessoaFisica pessoaFisica = new PessoaFisica();
        pessoaFisica.setId(resultSet.getInt("idPessoa"));
        pessoaFisica.setNome(resultSet.getString("nome"));
        pessoaFisica.setTelefone(resultSet.getString("telefone"));
        pessoaFisica.setEmail(resultSet.getString("email"));
        pessoaFisica.setLogradouro(resultSet.getString("logradouro"));
        pessoaFisica.setCidade(resultSet.getString("cidade"));
        pessoaFisica.setEstado(resultSet.getString("estado"));
        pessoaFisica.setCpf(resultSet.getString("cpf"));
        return pessoaFisica;
    }
}
```

Arquivo: PessoaJuridicaDAO.java

```
package cadastrobd.model;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import cadastrobd.model.util.ConectorBD;
import cadastrobd.model.util.SequenceManager;

public class PessoaJuridicaDAO {
    public PessoaJuridica getPessoa(int id) {
        try {
            Connection conexao = ConectorBD.getConnection();

            // Verifica se a conexão é válida
            if (conexao == null) {
                // Se a conexão for nula, retorna null
                return null;
            }

            // Cria um SQL para consultar os dados da pessoa jurídica pelo id
            String sql = "SELECT * FROM Pessoa p INNER JOIN PessoaJuridica pj ON p.idPessoa = pj.idPessoa
WHERE p.idPessoa = ?";

            // Obtém um objeto PreparedStatement com o SQL criado e o id fornecido como parâmetro
            PreparedStatement prepared = ConectorBD.getPrepared(conexao, sql);
            prepared.setInt(1, id);
```

```
            // Executa a consulta e obtém um objeto ResultSet com o resultado
            ResultSet resultSet = ConectorBD.getSelect(prepared);

            // Verifica se o ResultSet é válido e contém algum dado
            if (resultSet != null && resultSet.next()) {
                // Cria um objeto PessoaJuridica com os dados obtidos do ResultSet
                PessoaJuridica pessoaJuridica = criaPessoaJuridica(resultSet);

                // Fecha os objetos ResultSet, PreparedStatement e Connection
                ConectorBD.close(resultSet);
                ConectorBD.close(prepared);
                ConectorBD.close(conexao);
                return pessoaJuridica;
            }

            // Se o ResultSet for nulo ou vazio, fecha os objetos PreparedStatement e Connection e
retorna null
            ConectorBD.close(prepared);
            ConectorBD.close(conexao);
            return null;
        } catch (SQLException e) {
            // Se ocorrer algum erro, imprime a mensagem no console e retorna null
            System.out.println("Erro ao obter a pessoa jurídica pelo id: " + e.getMessage());
            return null;
        }
    }

    public List<PessoaJuridica> getPessoas() {
        try {
            // Obtém uma conexão com o banco de dados
            Connection conexao = ConectorBD.getConnection();

            // Verifica se a conexão é válida
            if (conexao == null) {
                // Se a conexão for nula, retorna null
                return null;
            }

            // Cria um SQL para consultar todos os dados das pessoas jurídicas
            String sql = "SELECT * FROM Pessoa p INNER JOIN PessoaJuridica pf ON p.idPessoa =
pf.idPessoa";

            // Obtém um objeto PreparedStatement com o SQL criado
            PreparedStatement prepared = conexao.prepareStatement(sql);

            // Executa a consulta e obtém um objeto ResultSet com o resultado
            ResultSet resultSet = ConectorBD.getSelect(prepared);

            // Cria uma lista de objetos PessoaJuridica para armazenar os dados obtidos
            List<PessoaJuridica> pessoas = new ArrayList<>();

            // Percorre o ResultSet enquanto houver dados
```

```
            while (resultSet != null && resultSet.next()) {
                // Cria um objeto PessoaJuridica com os dados obtidos do ResultSet
                PessoaJuridica pessoaJuridica = criaPessoaJuridica(resultSet);
                pessoas.add(pessoaJuridica);
            }

            // Fecha os objetos ResultSet, PreparedStatement e Connection
            ConectorBD.close(resultSet);
            ConectorBD.close(prepared);
            ConectorBD.close(conexao);

            // Retorna a lista de objetos PessoaJuridica criada
            return pessoas;
        } catch (SQLException e) {
            // Se ocorrer algum erro, imprime a mensagem no console e retorna null
            System.out.println("Erro ao obter todas as pessoas jurídicas: " + e.getMessage());
            return null;
        }
    }

    public boolean incluir(PessoaJuridica pessoaJuridica) {
        try {
            Integer nextId = SequenceManager.getValue("PessoaSequence");

            if (nextId == -1) {
                // Se não foi possível obter o próximo id da sequência, retorna false
                return false;
            }

            pessoaJuridica.setId(nextId);
            Connection conexao = ConectorBD.getConnection();

            // Verifica se a conexão é válida
            if (conexao == null) {
                // Se a conexão for nula, retorna false
                return false;
            }

            // Cria um SQL para inserir os dados da pessoa na tabela Pessoa
            String sql = "INSERT INTO Pessoa (idPessoa, nome, telefone, email, logradouro, cidade,
estado) VALUES (?, ?, ?, ?, ?, ?, ?)";

            // Obtém um objeto PreparedStatement com o SQL criado e os dados da pessoa jurídica fornecida
como parâmetro
            PreparedStatement prepared = ConectorBD.getPrepared(conexao, sql);
            prepared.setInt(1, pessoaJuridica.getId());
            prepared.setString(2, pessoaJuridica.getNome());
            prepared.setString(3, pessoaJuridica.getTelefone());
            prepared.setString(4, pessoaJuridica.getEmail());
            prepared.setString(5, pessoaJuridica.getLogradouro());
            prepared.setString(6, pessoaJuridica.getCidade());
            prepared.setString(7, pessoaJuridica.getEstado());
```

```
            // Executa a inserção e verifica se foi bem sucedida
            if (prepared.executeUpdate() <= 0) {
                // Se a inserção na tabela Pessoa falhou, fecha os objetos PreparedStatement e Connection
e retorna false
                ConectorBD.close(prepared);
                ConectorBD.close(conexao);
                return false;
            }

            // Se a inserção na tabela Pessoa foi bem sucedida, cria um SQL para inserir os dados da
pessoa jurídica na tabela PessoaJuridica
            sql = "INSERT INTO PessoaJuridica (idPessoa, cnpj) VALUES (?, ?)";

            // Obtém um objeto PreparedStatement com o SQL criado e os dados da pessoa jurídica fornecida
como parâmetro
            prepared = ConectorBD.getPrepared(conexao, sql);
            prepared.setInt(1, nextId);
            prepared.setString(2, pessoaJuridica.getCnpj());

            // Executa a inserção e verifica se foi bem sucedida
            if (prepared.executeUpdate() <= 0) {
                // Se a inserção na tabela PessoaJuridica falhou, fecha os objetos PreparedStatement e
Connection e retorna false
                ConectorBD.close(prepared);
                ConectorBD.close(conexao);
                return false;
            }

            // Se a inserção na tabela PessoaJuridica foi bem sucedida, fecha os objetos
PreparedStatement e Connection e retorna true
            ConectorBD.close(prepared);
            ConectorBD.close(conexao);
            return true;
        } catch (SQLException e) {
            // Se ocorrer algum erro, imprime a mensagem no console e retorna false
            System.out.println("Erro ao incluir a pessoa jurídica: " + e.getMessage());
            return false;
        }
    }

    public boolean alterar(PessoaJuridica pessoaJuridica) {
        try {
            Connection conexao = ConectorBD.getConnection();

            // Verifica se a conexão é válida
            if (conexao == null) {
                // Se a conexão for nula, retorna false
                return false;
            }

            // Cria um SQL para atualizar os dados da pessoa na tabela Pessoa
```

```
            String sql = "UPDATE Pessoa SET nome = ?, telefone = ?, email = ?, logradouro = ?, cidade =
?, estado = ? WHERE idPessoa = ?";

            // Obtém um objeto PreparedStatement com o SQL criado e os dados da pessoa jurídica fornecida
como parâmetro
            PreparedStatement prepared = ConectorBD.getPrepared(conexao, sql);
            prepared.setString(1, pessoaJuridica.getNome());
            prepared.setString(2, pessoaJuridica.getTelefone());
            prepared.setString(3, pessoaJuridica.getEmail());
            prepared.setString(4, pessoaJuridica.getLogradouro());
            prepared.setString(5, pessoaJuridica.getCidade());
            prepared.setString(6, pessoaJuridica.getEstado());
            prepared.setInt(7, pessoaJuridica.getId());

            // Executa a atualização e verifica se foi bem sucedida
            if (prepared.executeUpdate() <= 0) {
                // Se a atualização na tabela Pessoa falhou, fecha os objetos PreparedStatement e
Connection e retorna false
                ConectorBD.close(prepared);
                ConectorBD.close(conexao);
                return false;
            }

            // Se a atualização na tabela Pessoa foi bem sucedida, cria um SQL para atualizar os dados da
pessoa jurídica na tabela PessoaJuridica
            sql = "UPDATE PessoaJuridica SET cnpj = ? WHERE idPessoa = ?";

            // Obtém um objeto PreparedStatement com o SQL criado e os dados da pessoa jurídica fornecida
como parâmetro
            prepared = ConectorBD.getPrepared(conexao, sql);
            prepared.setString(1, pessoaJuridica.getCnpj());
            prepared.setInt(2, pessoaJuridica.getId());

            // Executa a atualização e verifica se foi bem sucedida
            if (prepared.executeUpdate() <= 0) {
                // Se a atualização na tabela PessoaJuridica falhou, fecha os objetos PreparedStatement e
Connection e retorna false
                ConectorBD.close(prepared);
                ConectorBD.close(conexao);
                return false;
            }

            // Se a atualização na tabela PessoaJuridica foi bem sucedida, fecha os objetos
PreparedStatement e Connection e retorna true
            ConectorBD.close(prepared);
            ConectorBD.close(conexao);
            return true;
        } catch (SQLException e) {
            // Se ocorrer algum erro, imprime a mensagem no console e retorna false
            System.out.println("Erro ao alterar a pessoa jurídica: " + e.getMessage());
            return false;
        }
```

```java
    }

    public boolean excluir(int id) {
        try {
            Connection conexao = ConectorBD.getConnection();

            // Verifica se a conexão é válida
            if (conexao == null) {
                // Se a conexão for nula, retorna false
                return false;
            }

            // Cria um SQL para excluir os dados da pessoa jurídica na tabela PessoaJuridica
            String sql = "DELETE FROM PessoaJuridica WHERE idPessoa = ?";

            // Obtém um objeto PreparedStatement com o SQL criado e o id fornecido como parâmetro
            PreparedStatement prepared = ConectorBD.getPrepared(conexao, sql);
            prepared.setInt(1, id);

            // Executa a exclusão e verifica se foi bem sucedida
            if (prepared.executeUpdate() <= 0) {
                // Se a exclusão na tabela PessoaJuridica falhou, fecha os objetos PreparedStatement e
Connection e retorna false
                ConectorBD.close(prepared);
                ConectorBD.close(conexao);
                return false;
            }

            // Se a exclusão na tabela PessoaJuridica foi bem sucedida, cria um SQL para excluir os dados
da pessoa na tabela Pessoa
            sql = "DELETE FROM Pessoa WHERE idPessoa = ?";

            // Obtém um objeto PreparedStatement com o SQL criado e o id fornecido como parâmetro
            prepared = ConectorBD.getPrepared(conexao, sql);
            prepared.setInt(1, id); // Substitua 1 pelo índice do campo id na tabela Pessoa

            // Executa a exclusão e verifica se foi bem sucedida
            if (prepared.executeUpdate() <= 0) {
                // Se a exclusão na tabela Pessoa falhou, fecha os objetos PreparedStatement e Connection
e retorna false
                ConectorBD.close(prepared);
                ConectorBD.close(conexao);
                return false;
            }

            // Se a exclusão na tabela Pessoa foi bem sucedida, fecha os objetos PreparedStatement e
Connection e retorna true
            ConectorBD.close(prepared);
            ConectorBD.close(conexao);
            return true;
        } catch (SQLException e) {
            // Se ocorrer algum erro, imprime a mensagem no console e retorna false
```

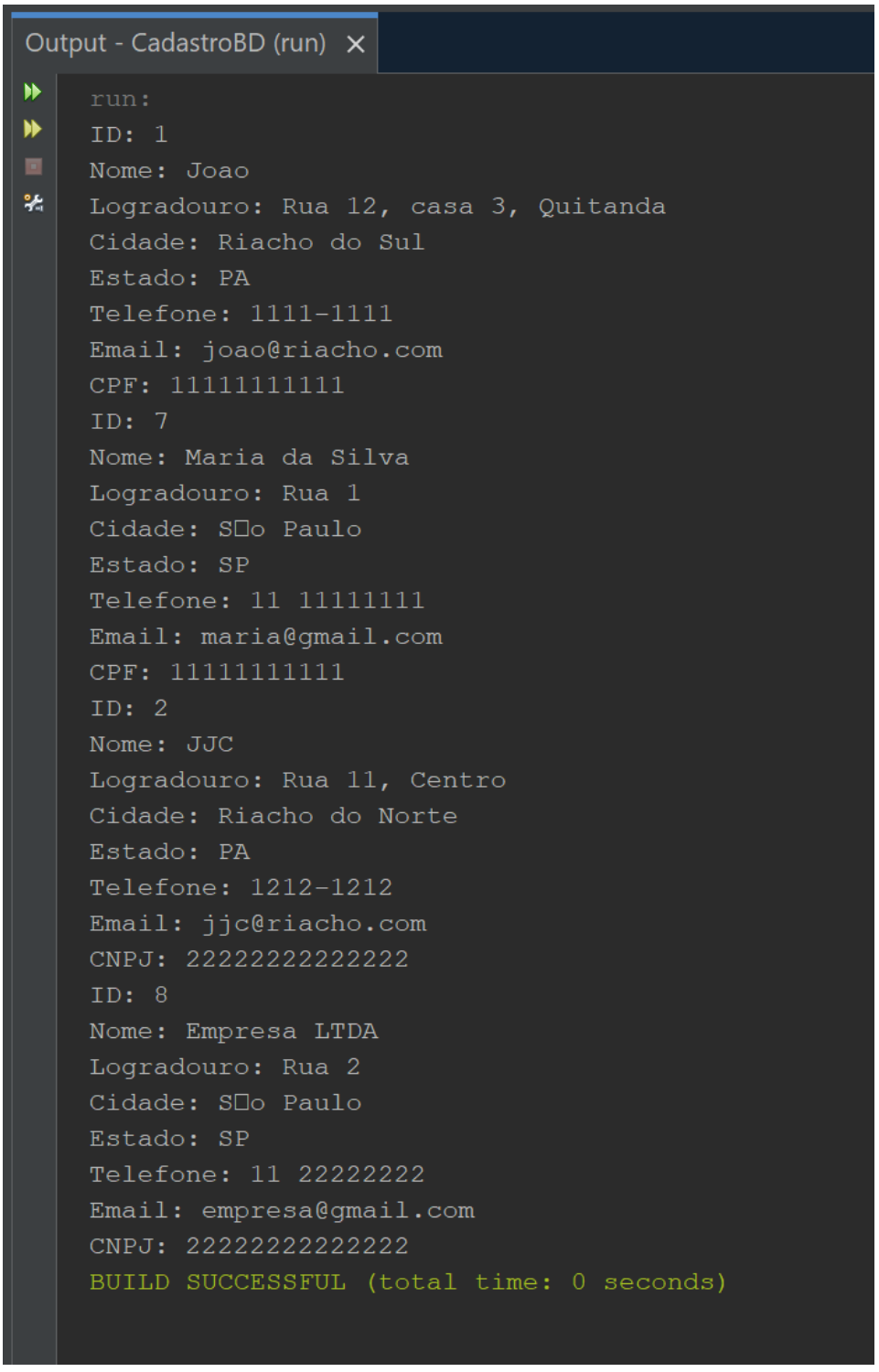
```
            System.out.println("Erro ao excluir a pessoa jurídica: " + e.getMessage());
            return false;
        }
    }

    private static PessoaJuridica criaPessoaJuridica(ResultSet resultSet) throws SQLException {
        PessoaJuridica pessoaJuridica = new PessoaJuridica();
        pessoaJuridica.setId(resultSet.getInt("idPessoa"));
        pessoaJuridica.setNome(resultSet.getString("nome"));
        pessoaJuridica.setTelefone(resultSet.getString("telefone"));
        pessoaJuridica.setEmail(resultSet.getString("email"));
        pessoaJuridica.setLogradouro(resultSet.getString("logradouro"));
        pessoaJuridica.setCidade(resultSet.getString("cidade"));
        pessoaJuridica.setEstado(resultSet.getString("estado"));
        pessoaJuridica.setCnpj(resultSet.getString("cnpj"));
        return pessoaJuridica;
    }
}
```

Arquivo: CadastroBD.java

```
package cadastrobd;

import cadastrobd.model.PessoaFisica;
import cadastrobd.model.PessoaFisicaDAO;
import cadastrobd.model.PessoaJuridica;
import cadastrobd.model.PessoaJuridicaDAO;

public class CadastroBD {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // a. Instanciar uma pessoa física e persistir no banco de dados.
        PessoaFisica pessoaFisica = new PessoaFisica(1, "Maria", "Rua 1", "São Paulo", "SP", "11
11111111", "maria@gmail.com", "11111111111");
        PessoaFisicaDAO pessoaFisicaDAO = new PessoaFisicaDAO();
        pessoaFisicaDAO.incluir(pessoaFisica);

        //b. Alterar os dados da pessoa física no banco.
        pessoaFisica.setNome("Maria da Silva");
        pessoaFisicaDAO.alterar(pessoaFisica);

        // c. Consultar todas as pessoas físicas do banco de dados e listar no console.
        pessoaFisicaDAO.getPessoas().forEach((pessoa) -> {
            pessoa.exibir();
        });
```

```
        // d. Excluir a pessoa física criada anteriormente no banco.
        pessoaFisicaDAO.excluir(pessoaFisica.getId());

        // e. Instanciar uma pessoa jurídica e persistir no banco de dados.
        PessoaJuridica pessoaJuridica = new PessoaJuridica(2, "Empresa", "Rua 2", "São Paulo", "SP", "11
22222222", "empresa@gmail.com", "22222222222222");
        PessoaJuridicaDAO pessoaJuridicaDAO = new PessoaJuridicaDAO();
        pessoaJuridicaDAO.incluir(pessoaJuridica);

        // f. Alterar os dados da pessoa jurídica no banco.
        pessoaJuridica.setNome("Empresa LTDA");
        pessoaJuridicaDAO.alterar(pessoaJuridica);

        // g. Consultar todas as pessoas jurídicas do banco e listar no console.
        pessoaJuridicaDAO.getPessoas().forEach((pessoa) -> {
            pessoa.exibir();
        });

        // h. Excluir a pessoa jurídica criada anteriormente no banco.
        pessoaJuridicaDAO.excluir(pessoaJuridica.getId());
    }
}
```

4. Resultados da execução dos códigos



5. Análise e Conclusão

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

Resposta: Estes tipos de componentes ajudam e aceleram o desenvolvimento proporcionando uma forma mais simples e integrada de conectar diferentes funcionalidades. No caso do JDBC, esse componente é responsável por facilitar a integração do Java à Bancos de Dados.

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

Resposta: A principal diferença é que no Statement, você normalmente utiliza instruções fixas de SQL e no PreparedStatement, você pode utilizar instruções parametrizadas.

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

Resposta: O padrão DAO (Data Access Object) encapsula e abstrai o acesso aos dados, podendo estes dados estar armazenados em um Banco de Dados, por exemplo, sem expor os detalhes dessa implementação para as demais camadas da aplicação.

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

Resposta: A herança é traduzida através de uma tabela para cada classe, sendo que cada tabela representa uma classe e as tabelas são relacionadas por meio de chaves estrangeiras. A tabela da classe mãe contém os atributos comuns a todas as classes filhas, enquanto as tabelas das classes filhas contém os atributos específicos de cada classe.