Estácio - Unidade São pedro
RJ 140 Km 2, 512 loja 1, São Pedro da Aldeia - RJ, 28941-182

Desenvolvimento Full Stack
Classe: Missão Prática | Nível 3 | Mundo 3
3º Semestre
Marvin de Almeida Costa

Título da Prática: 1º Procedimento | Mapeamento Objeto-Relacional e DAO

Objetivos da prática:

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

Todos os códigos solicitados neste roteiro de aula:

CadastroBD.java

```
package cadastrobd;

/**
 *
 * @author Marvin
 */
import java.util.*;
import cadastro.model.util.ConectorBD;
import cadastrobd.model.PessoaFisica;
```

```
import cadastrobd.model.PessoaFisicaDAO;
import cadastrobd.model.PessoaJuridica;
import cadastrobd.model.PessoaJuridicaDAO;
import cadastro.model.util.SequenceManager;

public class CadastroBD {

    public static void main(String[] args) {
        // Instanciando ConectorBD e SequenceManager (simulando a inicialização)
        ConectorBD conectorBD = new ConectorBD();
        SequenceManager sequenceManager = new SequenceManager();

        // Instanciando DAOs
        PessoaFisicaDAO pessoaFisicaDAO = new PessoaFisicaDAO(conectorBD,
sequenceManager);
        PessoaJuridicaDAO pessoaJuridicaDAO = new PessoaJuridicaDAO(conectorBD,
sequenceManager);

        // Operações com Pessoa Física
        int idPessoa = 0;
        String nome = "João";
        String logradouro = "Rua Exemplo";
        String cidade = "São Paulo";
        String estado = "SP";
        String telefone = "(11) 99999-9999";
        String email = "joao@example.com";
        String cpf = "1111666617";

        PessoaFisica pessoaFisica = new PessoaFisica(idPessoa, nome, logradouro, cidade,
estado, telefone, email, cpf);
        idPessoa = pessoaFisicaDAO.incluir(pessoaFisica);
        System.out.println("Pessoa Física incluída com sucesso.");

        pessoaFisica.setId(idPessoa);
        pessoaFisica.setNome("Alterando nome");
        pessoaFisica.setCpf("E111666617");

        pessoaFisicaDAO.alterar(pessoaFisica); // Alterando os dados da pessoa física
        System.out.println("Dados da pessoa física alterados com sucesso.");

        List<PessoaFisica> todasPessoasFisicas = pessoaFisicaDAO.getPessoas();
        for (PessoaFisica pf : todasPessoasFisicas) {
            pf.exibir();
        }
```

```
        System.out.println(pessoaFisica.getId());
        pessoaFisicaDAO.excluir(pessoaFisica.getId()); // Excluindo a pessoa física
        System.out.println("Pessoa Física excluída com sucesso.");

        idPessoa = 21;
        nome = "Empresa XYZ";
        logradouro = "Rua Exemplo";
        cidade = "São Paulo";
        estado = "SP";
        telefone = "(11) 99999-9999";
        email = "XYZ@example.com";
        String cnpj = "000000000010";

        // Operações com Pessoa Jurídica
        PessoaJuridica pessoaJuridica = new PessoaJuridica(idPessoa, nome, logradouro,
cidade, estado, telefone, email, cnpj);

        idPessoa = pessoaJuridicaDAO.incluir(pessoaJuridica);
        System.out.println("Pessoa Jurídica incluída com sucesso.");

        pessoaJuridica.setId(idPessoa);
        pessoaJuridica.setNome("Alterando Empresa");
        pessoaJuridica.setCnpj("E000000000010");

        pessoaJuridicaDAO.alterar(pessoaJuridica); // Alterando os dados da pessoa jurídica
        System.out.println("Dados da pessoa jurídica alterados com sucesso.");

        List<PessoaJuridica> todasPessoasJuridicas = pessoaJuridicaDAO.getPessoas();
        for (PessoaJuridica pj : todasPessoasJuridicas) {
            pj.exibir();
        }

        pessoaJuridicaDAO.excluir(pessoaJuridica.getId()); // Excluindo a pessoa jurídica
        System.out.println(pessoaFisica.getId());
        System.out.println("Pessoa Jurídica excluída com sucesso.");
    }
}
```

Pessoa.java

```
package cadastrobd.model;
```

```java
/**
 *
 * @author marvin
 */
public class Pessoa {
    protected int idPessoa;
    protected String nome;
    protected String logradouro;
    protected String cidade;
    protected String estado;
    protected String telefone;
    protected String email;

    // Construtor padrão
    public Pessoa() {}

    // Construtor completo
    public Pessoa(int idPessoa, String nome, String logradouro, String cidade, String 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 setId(int id) {
        this.idPessoa = id;
    }

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

    // Método para exibir dados
    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 String getLogradouro() {
        return logradouro;
    }

    public String getCidade() {
        return cidade;
    }

    public String getEstado() {
        return estado;
    }

    public String getTelefone() {
        return telefone;
    }

    public String getEmail() {
        return email;
    }

}
```

PessoaFisica.java

```
package cadastrobd.model;

/**
 *
 * @author marvin
 */
public class PessoaFisica extends Pessoa {
    private String cpf;
```

```
    // Construtor padrão
    public PessoaFisica() {
        super();
    }

    // Construtor completo
    public PessoaFisica(int idPessoa, String nome, String logradouro, String cidade, String
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 String getCpf() {
        return cpf;
    }

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

PessoaFisicaDAO.java

```
package cadastrobd.model;

/**
 *
 * @author marvin
 */
import java.util.*;
import java.util.List;
import cadastro.model.util.ConectorBD;
import cadastro.model.util.SequenceManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.ResultSet;
```

```
import java.sql.Connection;

public class PessoaFisicaDAO {
    private ConectorBD conector;
    private SequenceManager sequenceManager;

    public PessoaFisicaDAO(ConectorBD conector, SequenceManager sequenceManager) {
        this.conector = conector;
        this.sequenceManager = sequenceManager;
    }

    // Método para obter uma pessoa física pelo ID
    public PessoaFisica getPessoa(int id) {
        String sql = "SELECT * FROM Pessoa JOIN PessoaFisica ON Pessoa.idPessoa =
PessoaFisica.idPessoa WHERE Pessoa.id = ?";
        try (
                Connection conn = ConectorBD.getConnection();
                PreparedStatement pstmt = conn.prepareStatement(sql);
            ) {
            pstmt.setInt(1, id);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                String nome = rs.getString("nome");
                String logradouro = rs.getString("logradouro");
                String cidade = rs.getString("cidade");
                String estado = rs.getString("estado");
                String telefone = rs.getString("telefone");
                String email = rs.getString("email");
                String cpf = rs.getString("cpf");
                return new PessoaFisica(id, nome, logradouro, cidade, estado, telefone, email, cpf);
            }
            while (rs.next()) {

            }
        } catch (SQLException e) { // Catch SQLException specifically first
            e.printStackTrace();
        } catch (Exception e) { // Then catch the more general Exception
            e.printStackTrace();
        }
        return null;
    }

    // Método para obter todas as pessoas físicas
    public List<PessoaFisica> getPessoas() {
```

```
        List<PessoaFisica> pessoas = new ArrayList<>();
        String sql = "SELECT * FROM Pessoa JOIN PessoaFisica ON Pessoa.idPessoa =
PessoaFisica.idPessoa";
        try (
                Connection conn = ConectorBD.getConnection();
                PreparedStatement pstmt = conn.prepareStatement(sql);
                ResultSet rs = pstmt.executeQuery()
            ) {
            while (rs.next()) {
                int id = rs.getInt("idPessoa");
                String nome = rs.getString("nome");
                String logradouro = rs.getString("logradouro");
                String cidade = rs.getString("cidade");
                String estado = rs.getString("estado");
                String telefone = rs.getString("telefone");
                String email = rs.getString("email");
                String cpf = rs.getString("cpf");
                pessoas.add(new PessoaFisica(id, nome, logradouro, cidade, estado, telefone, email,
cpf));
            }
        } catch (SQLException e) { // Catch SQLException specifically first
            e.printStackTrace();
        } catch (Exception e) { // Then catch the more general Exception
            e.printStackTrace();
        }
        return pessoas;
    }

    // Método para incluir uma nova pessoa física
    public int incluir(PessoaFisica pessoaFisica) {
        String sqlInsertPessoa = "INSERT INTO Pessoa (nome, logradouro, cidade, estado,
telefone, email) VALUES (?, ?, ?, ?, ?, ?)";
        String sqlInsertPessoaFisica = "INSERT INTO PessoaFisica (idPessoa, cpf) VALUES (?,
?)";
        String sqlMaxIdPessoa = "SELECT MAX(idPessoa) AS MaxId FROM Pessoa";

        try {
            // Insert Pessoa
            try (Connection conn = ConectorBD.getConnection();
                 PreparedStatement pstmt = conn.prepareStatement(sqlInsertPessoa)) {

                pstmt.setString(1, pessoaFisica.getNome());
                pstmt.setString(2, pessoaFisica.getLogradouro());
                pstmt.setString(3, pessoaFisica.getCidade());
```

```
            pstmt.setString(4, pessoaFisica.getEstado());
            pstmt.setString(5, pessoaFisica.getTelefone());
            pstmt.setString(6, pessoaFisica.getEmail());
            pstmt.executeUpdate();


            PreparedStatement pstmtMaxId = conn.prepareStatement(sqlMaxIdPessoa);
            ResultSet rsMaxId = pstmtMaxId.executeQuery();
            if (rsMaxId.next()) {
                int lastInsertedId = rsMaxId.getInt("MaxId");
                // Now use this ID to insert PessoaFisica
                try (PreparedStatement pstmt2 = conn.prepareStatement(sqlInsertPessoaFisica)) {
                    pstmt2.setInt(1, lastInsertedId);
                    pstmt2.setString(2, pessoaFisica.getCpf());
                    pstmt2.executeUpdate();

                    return lastInsertedId;
                }
            } else {
                System.out.println("No se encontró el último ID insertado.");
            }

        }
    } catch (SQLException e) {
        e.printStackTrace();
        // Handle exception, such as logging the error or informing the user
    }
    return 0;
}

// Método para alterar os dados de uma pessoa física
public void alterar(PessoaFisica pessoaFisica) {
    String sql = "UPDATE Pessoa SET nome=?, logradouro=?, cidade=?, estado=?,
telefone=?, email=? WHERE idPessoa=?";
    String sql2 = "UPDATE PessoaFisica SET cpf=? WHERE idPessoa=?";

    try {
        // Insert Pessoa
        try (Connection conn = ConectorBD.getConnection();
            PreparedStatement pstmt = conn.prepareStatement(sql);
            PreparedStatement pstmt2 = conn.prepareStatement(sql2)) {

            pstmt.setString(1, pessoaFisica.getNome());
            pstmt.setString(2, pessoaFisica.getLogradouro());
```

```
            pstmt.setString(3, pessoaFisica.getCidade());
            pstmt.setString(4, pessoaFisica.getEstado());
            pstmt.setString(5, pessoaFisica.getTelefone());
            pstmt.setString(6, pessoaFisica.getEmail());
            pstmt.setInt(7, pessoaFisica.getId());
            pstmt.executeUpdate();

            pstmt2.setString(1, pessoaFisica.getCpf());
            pstmt2.setInt(2, pessoaFisica.getId());
            pstmt2.executeUpdate();
        }
    } catch (SQLException e) {
        e.printStackTrace();
        // Handle exception, such as logging the error or informing the user
    }
  }

  // Método para excluir uma pessoa física
  public void excluir(int id) {
    String sql = "DELETE FROM PessoaFisica WHERE idPessoa=?";
    String sql2 = "DELETE FROM Pessoa WHERE idPessoa=?";

    try (
        Connection conn = ConectorBD.getConnection();
        PreparedStatement pstmt = conn.prepareStatement(sql);
        PreparedStatement pstmt2 = conn.prepareStatement(sql2)) {
        pstmt.setInt(1, id);
        pstmt.executeUpdate();

        pstmt2.setInt(1, id);
        pstmt2.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
  }
}
```

PessoaJuridica.java

```
package cadastrobd.model;

/**
 *
 * @author marvin
```

```
 */
public class PessoaJuridica extends Pessoa {
    private String cnpj;

    // Construtor padrão
    public PessoaJuridica() {
        super();
    }

    // Construtor completo
    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: " + cnpj);
    }

    public String getCnpj() {
        return cnpj;
    }

    public void setCnpj(String cnpj) {
        this.cnpj = cnpj;
    }
}
```

PessoaJuridicaDAO.java

```
package cadastrobd.model;

/**
 *
 * @author marvin
 */
import java.util.*;
import java.util.List;
import cadastro.model.util.ConectorBD;
import cadastro.model.util.SequenceManager;
import java.sql.PreparedStatement;
```

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

public class PessoaJuridicaDAO {
    private ConectorBD conector;
    private SequenceManager sequenceManager;

    public PessoaJuridicaDAO(ConectorBD conector, SequenceManager sequenceManager) {
        this.conector = conector;
        this.sequenceManager = sequenceManager;
    }

    // Método para obter uma pessoa física pelo ID
    public PessoaJuridica getPessoa(int id) {
        String sql = "SELECT * FROM Pessoa JOIN PessoaJuridica ON Pessoa.idPessoa =
PessoaJuridica.idPessoa WHERE Pessoa.id = ?";
        try (
                Connection conn = ConectorBD.getConnection();
                PreparedStatement pstmt = conn.prepareStatement(sql);
            ) {
            pstmt.setInt(1, id);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                String nome = rs.getString("nome");
                String logradouro = rs.getString("logradouro");
                String cidade = rs.getString("cidade");
                String estado = rs.getString("estado");
                String telefone = rs.getString("telefone");
                String email = rs.getString("email");
                String cnpj = rs.getString("cnpj");
                return new PessoaJuridica(id, nome, logradouro, cidade, estado, telefone, email,
cnpj);
            }
            while (rs.next()) {

            }
        } catch (SQLException e) { // Catch SQLException specifically first
            e.printStackTrace();
        } catch (Exception e) { // Then catch the more general Exception
            e.printStackTrace();
        }
        return null;
    }
```

```
    // Método para obter todas as pessoas físicas
    public List<PessoaJuridica> getPessoas() {
        List<PessoaJuridica> pessoas = new ArrayList<>();
        String sql = "SELECT * FROM Pessoa JOIN PessoaJuridica ON Pessoa.idPessoa =
PessoaJuridica.idPessoa";
        try (
                Connection conn = ConectorBD.getConnection();
                PreparedStatement pstmt = conn.prepareStatement(sql);
                ResultSet rs = pstmt.executeQuery()
            ) {
            while (rs.next()) {
                int id = rs.getInt("idPessoa");
                String nome = rs.getString("nome");
                String logradouro = rs.getString("logradouro");
                String cidade = rs.getString("cidade");
                String estado = rs.getString("estado");
                String telefone = rs.getString("telefone");
                String email = rs.getString("email");
                String cnpj = rs.getString("cnpj");
                pessoas.add(new PessoaJuridica(id, nome, logradouro, cidade, estado, telefone,
email, cnpj));
            }
        } catch (SQLException e) { // Catch SQLException specifically first
            e.printStackTrace();
        } catch (Exception e) { // Then catch the more general Exception
            e.printStackTrace();
        }
        return pessoas;
    }

    // Método para incluir uma nova pessoa física
    public int incluir(PessoaJuridica pessoaJuridica) {
        String sqlInsertPessoa = "INSERT INTO Pessoa (nome, logradouro, cidade, estado,
telefone, email) VALUES (?, ?, ?, ?, ?, ?)";
        String sqlInsertPessoaJuridica = "INSERT INTO PessoaJuridica (idPessoa, cnpj) VALUES
(?, ?)";
        String sqlMaxIdPessoa = "SELECT MAX(idPessoa) AS MaxId FROM Pessoa";

        try {
            // Insert Pessoa
            try (Connection conn = ConectorBD.getConnection();
                 PreparedStatement pstmt = conn.prepareStatement(sqlInsertPessoa)) {
```

```
            pstmt.setString(1, pessoaJuridica.getNome());
            pstmt.setString(2, pessoaJuridica.getLogradouro());
            pstmt.setString(3, pessoaJuridica.getCidade());
            pstmt.setString(4, pessoaJuridica.getEstado());
            pstmt.setString(5, pessoaJuridica.getTelefone());
            pstmt.setString(6, pessoaJuridica.getEmail());
            pstmt.executeUpdate();


            PreparedStatement pstmtMaxId = conn.prepareStatement(sqlMaxIdPessoa);
            ResultSet rsMaxId = pstmtMaxId.executeQuery();
            if (rsMaxId.next()) {
                int lastInsertedId = rsMaxId.getInt("MaxId");
                // Now use this ID to insert PessoaJuridica
                try (PreparedStatement pstmt2 = conn.prepareStatement(sqlInsertPessoaJuridica))
{
                    pstmt2.setInt(1, lastInsertedId);
                    pstmt2.setString(2, pessoaJuridica.getCnpj());
                    pstmt2.executeUpdate();

                    return lastInsertedId;
                }
            } else {
                System.out.println("No se encontró el último ID insertado.");
            }

        }
    } catch (SQLException e) {
        e.printStackTrace();
        // Handle exception, such as logging the error or informing the user
    }
    return 0;
}

// Método para alterar os dados de uma pessoa física
public void alterar(PessoaJuridica pessoaJuridica) {
    String sql = "UPDATE Pessoa SET nome=?, logradouro=?, cidade=?, estado=?,
telefone=?, email=? WHERE idPessoa=?";
    String sql2 = "UPDATE PessoaJuridica SET cnpj=? WHERE idPessoa=?";

    try {
        // Insert Pessoa
        try (Connection conn = ConectorBD.getConnection();
            PreparedStatement pstmt = conn.prepareStatement(sql);
```

```
            PreparedStatement pstmt2 = conn.prepareStatement(sql2)) {

            pstmt.setString(1, pessoaJuridica.getNome());
            pstmt.setString(2, pessoaJuridica.getLogradouro());
            pstmt.setString(3, pessoaJuridica.getCidade());
            pstmt.setString(4, pessoaJuridica.getEstado());
            pstmt.setString(5, pessoaJuridica.getTelefone());
            pstmt.setString(6, pessoaJuridica.getEmail());
            pstmt.setInt(7, pessoaJuridica.getId());
            pstmt.executeUpdate();

            pstmt2.setString(1, pessoaJuridica.getCnpj());
            pstmt2.setInt(2, pessoaJuridica.getId());
            pstmt2.executeUpdate();
        }
    } catch (SQLException e) {
        e.printStackTrace();
        // Handle exception, such as logging the error or informing the user
    }
  }

  // Método para excluir uma pessoa física
  public void excluir(int id) {
    String sql = "DELETE FROM PessoaJuridica WHERE idPessoa=?";
    String sql2 = "DELETE FROM Pessoa WHERE idPessoa=?";

    try (
        Connection conn = ConectorBD.getConnection();
        PreparedStatement pstmt = conn.prepareStatement(sql);
        PreparedStatement pstmt2 = conn.prepareStatement(sql2)) {
        pstmt.setInt(1, id);
        pstmt.executeUpdate();

        pstmt2.setInt(1, id);
        pstmt2.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
  }
}
```

ConectorBD.java

```
package cadastro.model.util;

/**
 *
 * @author marvin
 */
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;
import java.sql.SQLException;

public class ConectorBD {

    // Método para obter uma conexão com o banco de dados
    public static Connection getConnection() throws SQLException {
        String url =
"jdbc:sqlserver://localhost:1433;databaseName=Loja;trustServerCertificate=true";

        String usuario = "sa";
        String senha = "1234";

        return DriverManager.getConnection(url, usuario, senha);
    }

    // Método para obter um PreparedStatement
    public static PreparedStatement getPrepared(String sql) throws Exception {
        try (Connection connection = getConnection()) {
            return connection.prepareStatement(sql);
        }
    }

    // Método para obter um ResultSet
    public static ResultSet getSelect(String sql) throws Exception {
        try (Connection connection = getConnection(); Statement statement =
connection.createStatement()) {
            return statement.executeQuery(sql);
        }
    }

    // Métodos sobrecarregados para fechar Statement, ResultSet e Connection
    public static void close(Statement statement) {
        if (statement != null) {
```

```
            try {
                statement.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    public static void close(ResultSet resultSet) {
        if (resultSet != null) {
            try {
                resultSet.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    public static void close(Connection connection) {
        if (connection != null) {
            try {
                connection.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

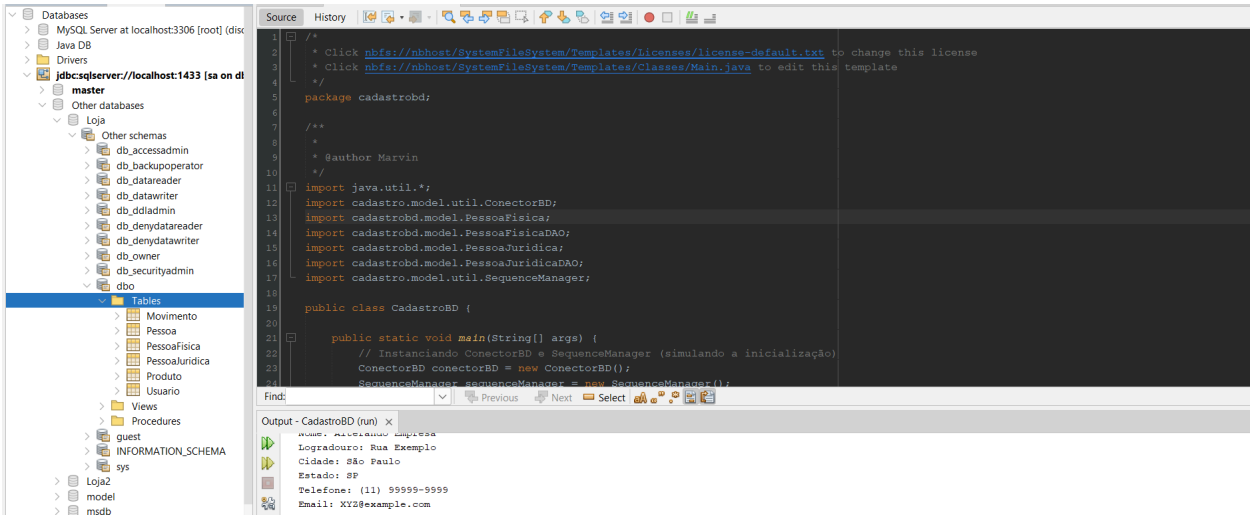
SequenceManager.java

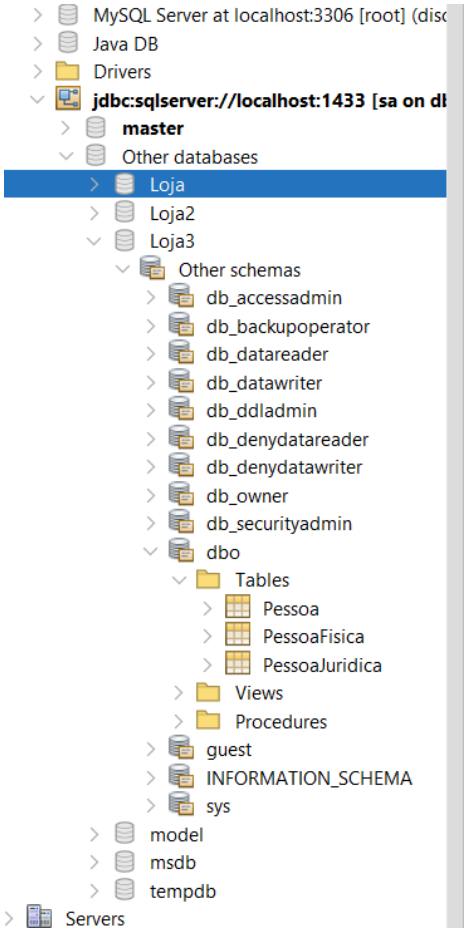
```
package cadastro.model.util;

/**
 *
 * @author marvin
 */
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.ResultSet;

public class SequenceManager {
```

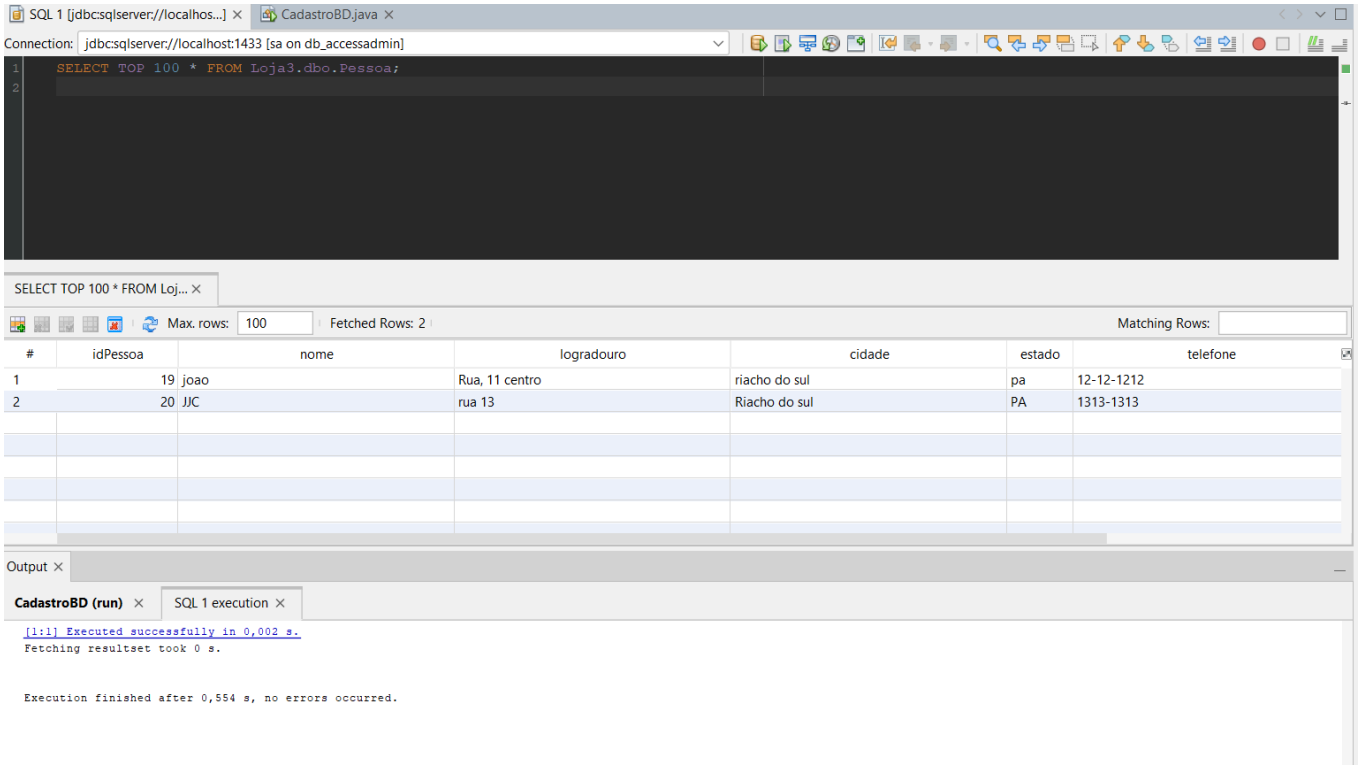
```
    // Método para obter o próximo valor de uma sequência
    public static long getValue(String sequenceName) throws SQLException {
        try (Connection connection = ConectorBD.getConnection();
             PreparedStatement preparedStatement = connection.prepareStatement("SELECT
nextval('" + sequenceName + "')")) {
            try (ResultSet resultSet = preparedStatement.executeQuery()) {
                if (resultSet.next()) {
                    long result = resultSet.getLong(1);
                    return result;
                } else {
                    throw new SQLException("No result found");
                }
            }
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Os resultados da execução dos códigos também devem ser apresentados;

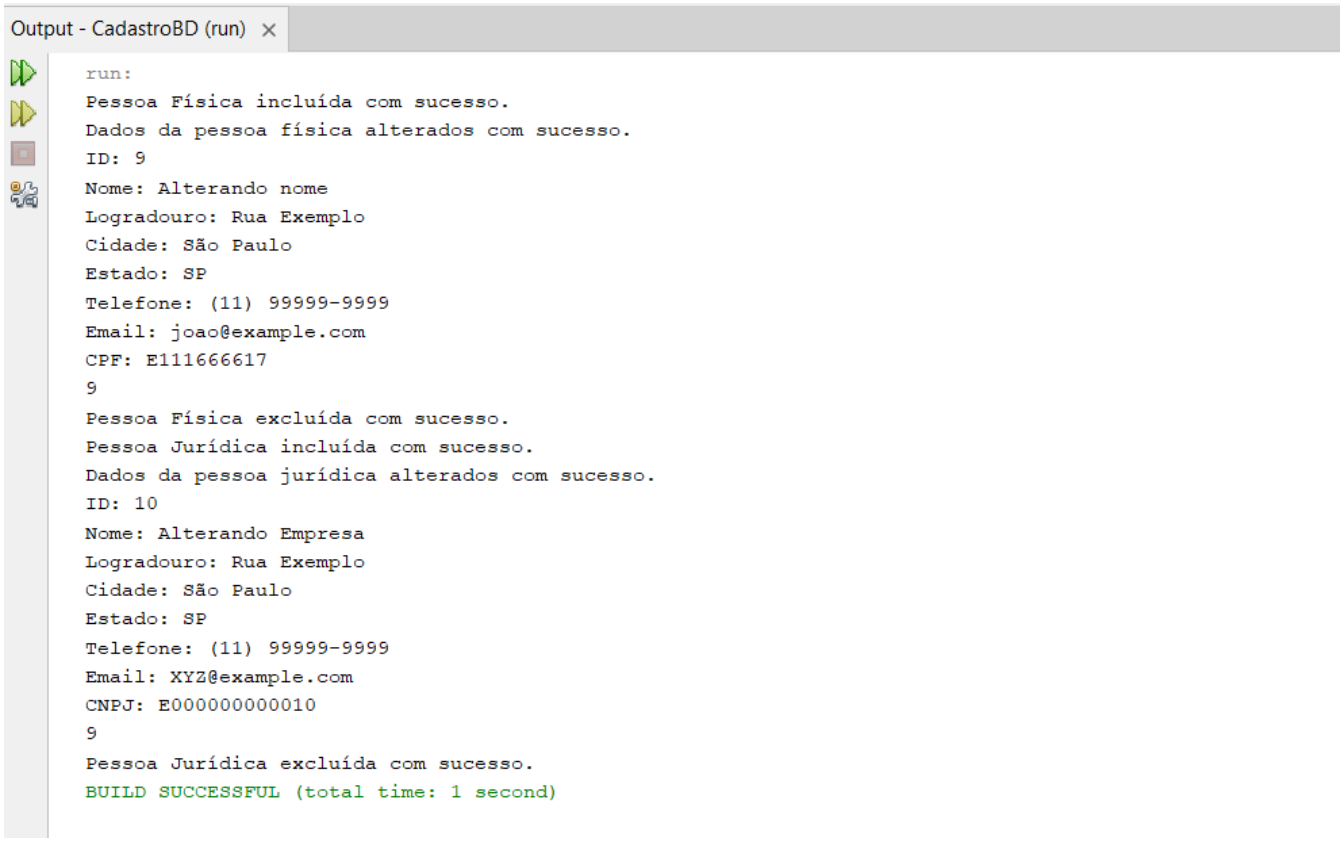
SQL Server Driver




MySQL Server at localhost:3306 [root] (disc
Java DB
Drivers
jdbc:sqlserver://localhost:1433 [sa on dl
master
Other databases
Loja
Loja2
Loja3
Other schemas
db_accessadmin
db_backupoperator
db_datareader
db_datawriter
db_ddladmin
db_denydatareader
db_denydatawriter
db_owner
db_securityadmin
dbo
Tables
Pessoa
PessoaFisica
PessoaJuridica
Views
Procedures
guest
INFORMATION_SCHEMA
sys
model
msdb
tempdb
Servers


SQL 1 [jdbc:sqlserver://localhos...]
CadastroBD.java
Connection: jdbc:sqlserver://localhost:1433 [sa on db_accessadmin]
SELECT TOP 100 * FROM Loja3.dbo.Pessoa;
SELECT TOP 100 * FROM Loj...
Max. rows: 100
Fetched Rows: 2
Matching Rows:
idPessoa nome logradouro cidade estado telefone
1 19 joao Rua, 11 centro riacho do sul pa 12-12-1212
2 20 JJC rua 13 Riacho do sul PA 1313-1313
Output
CadastroBD (run)
SQL 1 execution
[1:1] Executed successfully in 0,002 s.
Fetching resultset took 0 s.
Execution finished after 0,554 s, no errors occurred.

- Instanciar uma pessoa física e persistir no banco de dados.
- Alterar os dados da pessoa física no banco.
- Consultar todas as pessoas físicas do banco de dados e listar no console.
- Excluir a pessoa física criada anteriormente no banco.
- Instanciar uma pessoa jurídica e persistir no banco de dados.
- Alterar os dados da pessoa jurídica no banco.
- Consultar todas as pessoas jurídicas do banco e listar no console.
- Excluir a pessoa jurídica criada anteriormente no banco.



Análise e Conclusão:

Quais as diferenças no uso de sequence e identity?

Os componentes de middleware, como o JDBC (Java Database Connectivity), desempenham um papel crucial no desenvolvimento de aplicações, especialmente quando se trata de conectar e interagir com bancos de dados. A importância do JDBC e outros componentes de middleware pode ser entendida através de vários aspectos:

1. Abstração de Banco de Dados

O JDBC oferece uma abstração de banco de dados que permite aos desenvolvedores escrever código independente do sistema de gerenciamento de banco de dados (DBMS). Isso significa que você pode mudar seu DBMS sem ter que alterar significativamente o código da sua aplicação, desde que o novo DBMS suporte JDBC.

2. Facilita a Interação com Bancos de Dados

O JDBC fornece uma API para executar consultas SQL, manipular resultados e gerenciar transações. Isso simplifica a interação entre a aplicação Java e o banco de dados, permitindo operações complexas como joins, subconsultas, inserções em massa, atualizações e exclusões.

3. Suporte a Transações

O JDBC suporta o controle de transações, o que é essencial para garantir a integridade dos dados em aplicações que realizam várias operações de banco de dados simultaneamente. Você pode iniciar, confirmar ou reverter transações usando o JDBC, garantindo que todas as operações dentro de uma transação sejam concluídas com sucesso antes de serem efetivadas no banco de dados.

4. Segurança

O JDBC inclui mecanismos para autenticação e autorização, ajudando a proteger os dados sensíveis armazenados no banco de dados. Além disso, ele suporta criptografia de conexão, o que ajuda a proteger os dados transmitidos entre a aplicação e o servidor de banco de dados.

5. Desempenho

O JDBC é otimizado para fornecer alto desempenho na execução de consultas e manipulação de dados. Ele utiliza técnicas como pooling de conexões para reduzir o overhead de abrir e fechar conexões frequentemente, melhorando assim a eficiência geral das operações de banco de dados.

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

A diferença entre Statement e PreparedStatement em Java, especialmente quando se trata da manipulação de dados em bancos de dados, está relacionada à forma como eles preparam e executam consultas SQL. Ambos são interfaces do JDBC (Java Database Connectivity) que permitem a interação com bancos de dados, mas têm características distintas que os tornam adequados para diferentes cenários.

Statement

- Uso Simples: O Statement é uma interface simples que permite enviar instruções SQL ao banco de dados. É útil para operações simples onde não há necessidade de reutilização de consultas ou parâmetros.

- Desempenho: Para consultas simples, o desempenho do Statement pode ser suficiente. No entanto, para operações complexas ou repetitivas, ele pode ser menos eficiente devido à falta de pré-compilação das consultas.

- Segurança: Não oferece segurança contra ataques de injeção SQL, pois os valores dos parâmetros são inseridos diretamente na consulta SQL.

PreparedStatement

- Preparação de Consultas: A principal vantagem do PreparedStatement é sua capacidade de preparar consultas SQL com placeholders (?) para parâmetros. Isso melhora significativamente o desempenho para consultas repetidas, pois a consulta SQL é compilada apenas uma vez e armazenada no cache do banco de dados.

- Segurança: Oferece proteção contra ataques de injeção SQL, pois os parâmetros são separados da consulta SQL, evitando a concatenação direta de strings que poderiam conter código malicioso.

- Flexibilidade: Permite a definição de tipos de dados para os parâmetros, o que ajuda a evitar erros de tipo de dados.

Como o padrão DAO melhora a manutenibilidade do software?

O padrão DAO (Data Access Object) é uma abordagem de design que separa a lógica de acesso aos dados da lógica de negócios do aplicativo. Isso significa que os detalhes específicos de como os dados são armazenados e recuperados são isolados dos demais componentes do sistema. Essa separação promove a manutenibilidade do software de várias maneiras:

1. Facilita a Manutenção do Código

Ao encapsular a lógica de acesso aos dados em classes DAO, você pode fazer alterações na forma como os dados são persistidos ou recuperados sem afetar diretamente o código de negócios. Isso reduz o risco de erros durante a manutenção e facilita a atualização do sistema para suportar novas tecnologias de banco de dados.

2. Melhora a Reutilização de Código

Os objetos DAO podem ser reutilizados em diferentes partes do aplicativo, evitando a duplicação de código. Por exemplo, se você precisar acessar um recurso de banco de dados em vários lugares no seu aplicativo, pode criar um objeto DAO para esse recurso e usá-lo sempre que necessário.

3. Promove a Segregação de Responsabilidades

Cada classe DAO é responsável por uma única operação de acesso ao banco de dados, como inserir, atualizar ou excluir registros. Isso segue o princípio de responsabilidade única, tornando o código mais fácil de entender e manter.

4. Fácil de Testar

Como a lógica de acesso aos dados está separada do restante do aplicativo, é mais fácil escrever testes unitários para as classes DAO. Você pode simular o comportamento do banco de dados usando mocks, permitindo testar a lógica de negócios sem depender de um banco de dados real.

5. Flexibilidade Tecnológica

Usar o padrão DAO permite mudar facilmente entre diferentes tecnologias de banco de dados ou até mesmo entre bancos de dados SQL e NoSQL, sem ter que modificar significativamente o código de negócios. Basta implementar novas classes DAO para a nova tecnologia.

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

Na programação orientada a objetos (POO), a herança é um mecanismo que permite criar novas classes a partir de classes existentes, herdando seus atributos e métodos. No contexto de bancos de dados relacionais (RDBMS) e modelos estritamente relacionais, a herança não é uma característica nativa como em linguagens de programação orientadas a objetos. No entanto, existem várias abordagens para simular ou implementar heranças em bancos de dados relacionais.

Abordagem de Tabela Pai/Filho

Uma maneira comum de simular herança é através da criação de tabelas pai/filho. A tabela "pai" contém os campos comuns a todas as suas "filhas", enquanto cada tabela "filha" inclui campos específicos além dos herdados. Isso pode ser feito usando chaves estrangeiras para estabelecer relações entre as tabelas.

Abordagem de Herança Múltipla

Em alguns casos, uma entidade pode precisar herdar de mais de uma classe base. Embora o SQL padrão não suporte diretamente herança múltipla, algumas extensões do SQL, como o PostgreSQL, oferecem suporte a essa funcionalidade.