# CadastroBD

**Matheus José Ribeiro de Moura 2023 0713 6158**

**Polo Rua Tereza - Petrópolis - RJ**
**BackEnd sem banco não tem – Turma 9001 – 2024.3 FLEX**

## Objetivo da Prática

A prática tem como objetivo implementar a persistência de dados no banco de dados através do middleware JDBC, utilizar o padrão DAO ao manusear os dados, implementar o mapeamento objeto-relacional em Java criando, assim, um sistema cadastral com persistência de dados em um banco relacional.

No final do exercício, será criado um aplicativo cadastral com uso do SQL Server na persistência de dados.

## 1º Procedimento | Mapeamento Objeto-Relacional e DAO

- Classe Endereco

```
package cadastrobd.model;
public class Endereco {
private int id;
private String logradouro;
private String bairro;
private String numero;
private String cidade;
private String estado;
private String complemento;
private String cep;

public Endereco() {
}
```

```java
public Endereco(int id, String logradouro, String bairro,
String numero, String cidade, String estado, String
complemento, String cep) {
    this.id = id;
    this.logradouro = logradouro;
    this.bairro = bairro;
    this.numero = numero;
    this.cidade = cidade;
    this.estado = estado;
    this.complemento = complemento;
    this.cep = cep;
}

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getLogradouro() {
    return logradouro;
}

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

public String getBairro() {
    return bairro;
}

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

public String getNumero() {
    return numero;
}
```

```java
public void setNumero(String numero) {
    this.numero = numero;
}

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 getComplemento() {
    return complemento;
}

public void setComplemento(String complemento) {
    this.complemento = complemento;
}

public String getCep() {
    return cep;
}

public void setCep(String cep) {
    this.cep = cep;
}

}
```

- ENUM TipoPessoa

```
package cadastrobd.enums;

public enum TipoPessoa {
    FISICA(1),
    JURIDICA(2);

    private final int tipo;

    TipoPessoa(int tipo) {
        this.tipo = tipo;
    }

    public int getTipo() {
        return tipo;
    }
}
```

- Classe Pessoa

```
package cadastrobd.model;

import cadastrobd.enums.TipoPessoa;

public class Pessoa {

    private int id;
    private String nome;
    private String telefone;
    private Endereco endereco;
    private TipoPessoa tipoPessoa;

    public Pessoa() {
    }
```

```
    public Pessoa(int id, String nome, String
telefone, Endereco endereco, TipoPessoa tipoPessoa)
{
        this.id = id;
        this.nome = nome;
        this.telefone = telefone;
        this.endereco = endereco;
        this.tipoPessoa = tipoPessoa;
    }

    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 getTelefone() {
        return telefone;
    }

    public void setTelefone(String telefone) {
        this.telefone = telefone;
    }

    public Endereco getEndereco() {
```

```
        return endereco;
    }

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

    public TipoPessoa getTipoPessoa() {
        return tipoPessoa;
    }

    public void setTipoPessoa(TipoPessoa
tipoPessoa)
    {
        this.tipoPessoa = tipoPessoa;
    }
}
```

- Classe PessoaFisica

```
package cadastrobd.model;

import cadastrobd.enums.TipoPessoa;

public class PessoaFisica extends Pessoa {

    private String cpf;

    public PessoaFisica() {
    }
```

```
    public PessoaFisica(int id, String nome, String
telefone, Endereco endereco, String cpf) {
        super(id, nome, telefone, endereco,
TipoPessoa.FISICA);
        this.cpf = cpf;
    }

    public String getCpf() {
        return cpf;
    }

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

- Classe PessoaJuridica

```
package cadastrobd.model;

import cadastrobd.enums.TipoPessoa;

public class PessoaJuridica extends Pessoa {

    private String cnpj;

    public PessoaJuridica() {
    }

    public PessoaJuridica(int id, String nome,
String telefone, Endereco endereco, String cnpj) {
        super(id, nome, telefone, endereco,
TipoPessoa.JURIDICA);
        this.cnpj = cnpj;
```

```
    }

    public String getCnpj() {
        return cnpj;
    }

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

- Classe ConectorDB

```
package cadastro.model.util;

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

public class ConectorBD {

    private static final String URL =
"jdbc:sqlserver://localhost:1433;databaseName=loja;
encrypt=false;trustServerCertificate=true";
    private static final String USERNAME = "loja";
    private static final String PASSWORD =
"1234567890";

    public Connection getConnection() throws
SQLException {
        return DriverManager.getConnection(URL,
```

```
USERNAME, PASSWORD);
    }

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

    public PreparedStatement
getPreparedStatement(Connection connection, String
sql, Object... params) throws SQLException {
        PreparedStatement preparedStatement =
connection.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS);
        for (int i = 0; i < params.length; i++) {
            preparedStatement.setObject(i + 1,
params[i]);
        }
        return preparedStatement;
    }

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

```
            }
        }
    }

    public ResultSet getSelect(Connection
connection, String sql, Object... params) throws
SQLException {
        PreparedStatement preparedStatement =
getPreparedStatement(connection, sql, params);
        return preparedStatement.executeQuery();
    }

    public void close(ResultSet resultSet) {
        if (resultSet != null) {
            try {
                resultSet.close();
            } catch (SQLException e) {
                System.err.println("Erro ao fechar
o ResultSet: " + e.getMessage());
            }
        }
    }
}
```

- Classe PessoaFisicaDAO

```
package cadastro.model;

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

import java.util.ArrayList;
```

```
import java.util.List;

import cadastro.model.util.ConectorBD;

import cadastrobd.model.PessoaFisica;
import cadastrobd.model.Endereco;

import cadastrobd.enums.TipoPessoa;

public class PessoaFisicaDAO {

    private final ConectorBD conectorBD;

    public PessoaFisicaDAO(ConectorBD conectorBD) {
        this.conectorBD = conectorBD;
    }

    public PessoaFisica getPessoaFisica(int id)
throws SQLException {
        PessoaFisica pessoaFisica = null;
        Connection connection =
conectorBD.getConnection();
        try {
            PreparedStatement preparedStatement =
conectorBD.getPreparedStatement(connection, "SELECT
\n"
                    + "  p.*,\n"
                    + "  pf.*,\n"
                    + "  e.*\n"
                    + "FROM \n"
                    + "  Pessoa p\n"
                    + "  INNER JOIN Pessoa_Fisica
pf ON p.id = pf.pessoa_id\n"
                    + "  INNER JOIN Endereco e ON
p.id_endereco = e.id\n"
```

```
                    + "WHERE \n"
                    + "  p.id = ?;", id);
            ResultSet resultSet =
preparedStatement.executeQuery();
            if (resultSet.next()) {
                pessoaFisica = new PessoaFisica(
                        resultSet.getInt("id"),

resultSet.getString("nome"),

resultSet.getString("telefone"),
                        new
Endereco(resultSet.getInt("id_endereco"),
resultSet.getString("logradouro"),
resultSet.getString("bairro"),
resultSet.getString("numero"),
resultSet.getString("cidade"),
resultSet.getString("estado"),
resultSet.getString("complemento"),
resultSet.getString("cep")),

resultSet.getString("cpf"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
            System.out.println("getPessoaFisica ->
" + e);
        } finally {
            conectorBD.close(connection);
        }
        return pessoaFisica;
    }

    public List<PessoaFisica> getPessoasFisicas()
throws SQLException {
```

```
        List<PessoaFisica> pessoasFisicas = new
ArrayList<>();
        Connection connection =
conectorBD.getConnection();
        try {
            PreparedStatement preparedStatement =
conectorBD.getPreparedStatement(connection, "SELECT
* FROM Pessoa p INNER JOIN Pessoa_Fisica pf ON p.id
= pf.pessoa_id INNER JOIN Endereco e ON
p.id_endereco = e.id WHERE p.id_tipo_pessoa = ?",
TipoPessoa.FISICA.getTipo()
            );
            ResultSet resultSet =
preparedStatement.executeQuery();
            while (resultSet.next()) {
                PessoaFisica pessoaFisica = new
PessoaFisica(
                        resultSet.getInt("id"),

resultSet.getString("nome"),

resultSet.getString("telefone"),
                        new
Endereco(resultSet.getInt("id_endereco"),
resultSet.getString("logradouro"),
resultSet.getString("bairro"),
resultSet.getString("numero"),
resultSet.getString("cidade"),
resultSet.getString("estado"),
resultSet.getString("complemento"),
resultSet.getString("cep")),
                        resultSet.getString("cpf")
                );
                pessoasFisicas.add(pessoaFisica);
            }
```

```
        } catch (SQLException e) {
            System.out.println("getPessoasFisicas
->  " + e);
        } finally {
            conectorBD.close(connection);
        }
        return pessoasFisicas;
    }

    public void incluirPessoaFisica(PessoaFisica
pessoaFisica) throws SQLException {
        Connection connection =
conectorBD.getConnection();
        try {
            connection.setAutoCommit(false);

            int idEnderecoCadastrado =
cadastrarEndereco(pessoaFisica.getEndereco(),
connection);

            PreparedStatement
preparedStatementPessoa =
conectorBD.getPreparedStatement(connection, "INSERT
INTO Pessoa (nome, telefone, id_endereco,
id_tipo_pessoa) VALUES (?, ?, ?, ?)",
pessoaFisica.getNome(), pessoaFisica.getTelefone(),
idEnderecoCadastrado, TipoPessoa.FISICA.getTipo());

preparedStatementPessoa.executeUpdate();

            if
(preparedStatementPessoa.getGeneratedKeys().next())
{
                int id =
preparedStatementPessoa.getGeneratedKeys().getInt(1
```

```
);
                pessoaFisica.setId(id);
                PreparedStatement
preparedStatementPessoaFisica =
conectorBD.getPreparedStatement(connection, "INSERT
INTO Pessoa_Fisica  (pessoa_id, cpf) VALUES (?,
?)", id, pessoaFisica.getCpf());

preparedStatementPessoaFisica.executeUpdate();
                connection.commit();
                System.out.println("Pessoa Fisica
cadastrada com sucesso!");
            } else {
                System.out.println("A instrução
INSERT não gerou uma chave.");
            }
        } catch (SQLException e) {
            e.printStackTrace();
            try {
                connection.rollback();
            } catch (SQLException err) {

System.out.println("incluirPessoaFisica ->  " +
err);
            }
        } finally {
            conectorBD.close(connection);
        }
    }

    private int cadastrarEndereco(Endereco
endereco, Connection connection) throws
SQLException {
        PreparedStatement preparedStatementEndereco
= conectorBD.getPreparedStatement(connection,
```

```
"INSERT INTO Endereco (logradouro, numero, bairro,
cidade, estado, complemento, cep) VALUES (?, ?, ?,
?, ?, ?, ?)",
                endereco.getLogradouro(),
endereco.getNumero(), endereco.getBairro(),
endereco.getCidade(), endereco.getEstado(),
endereco.getComplemento(), endereco.getCep());
        preparedStatementEndereco.executeUpdate();

        if
(preparedStatementEndereco.getGeneratedKeys().next(
)) {
            int id =
preparedStatementEndereco.getGeneratedKeys().getInt
(1);
            endereco.setId(id);
            System.out.println("Endereco cadastrado
com sucesso!");
            return id;
        } else {
            System.out.println("A instrução INSERT
não gerou uma chave.");
            return 0;
        }
    }

    public void alterarPessoaFisica(PessoaFisica
pessoaFisica) throws SQLException {
        Connection connection =
conectorBD.getConnection();
        try {
            connection.setAutoCommit(false);
            PreparedStatement
preparedStatementPessoa =
conectorBD.getPreparedStatement(connection, "UPDATE
```

```
Pessoa SET nome = ?, telefone = ?, id_endereco = ?
WHERE id = ?", pessoaFisica.getNome(),
pessoaFisica.getTelefone(),
pessoaFisica.getEndereco().getId(),
pessoaFisica.getId());

preparedStatementPessoa.executeUpdate();
            PreparedStatement
preparedStatementPessoaFisica =
conectorBD.getPreparedStatement(connection, "UPDATE
Pessoa_Fisica SET cpf = ? WHERE pessoa_id = ?",
pessoaFisica.getCpf(), pessoaFisica.getId());

preparedStatementPessoaFisica.executeUpdate();
            connection.commit();
        } catch (SQLException e) {
            e.printStackTrace();
            try {
                connection.rollback();
            } catch (SQLException err) {

System.out.println("alterarPessoaFisica ->  " +
err);
            } finally {
                conectorBD.close(connection);
            }
        }
    }

    public void excluirPessoaFisica(int id) throws
SQLException {
        Connection connection =
conectorBD.getConnection();
        try {
            connection.setAutoCommit(false);
```

```
            PreparedStatement
preparedStatementMovimentacaoCompra =
conectorBD.getPreparedStatement(connection, "DELETE
FROM Movimentacao_Compra WHERE id_pessoa_fisica =
?", id);

preparedStatementMovimentacaoCompra.executeUpdate()
;
            PreparedStatement
preparedStatementPessoaFisica =
conectorBD.getPreparedStatement(connection, "DELETE
FROM Pessoa_Fisica WHERE pessoa_id = ?", id);

preparedStatementPessoaFisica.executeUpdate();
            PreparedStatement
preparedStatementPessoa =
conectorBD.getPreparedStatement(connection, "DELETE
FROM Pessoa WHERE id = ? AND id_tipo_pessoa = 1",
id);

preparedStatementPessoa.executeUpdate();
            connection.commit();
        } catch (SQLException e) {
            e.printStackTrace();
            try {
                connection.rollback();
            } catch (SQLException err) {

System.out.println("excluirPessoaFisica ->  " +
err);
            } finally {
                conectorBD.close(connection);
            }
        }
    }
```

```
}
```

- Classe PessoaJuridicaDAO

```
package cadastro.model;

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

import java.util.ArrayList;
import java.util.List;

import cadastro.model.util.ConectorBD;

import cadastrobd.model.PessoaJuridica;
import cadastrobd.model.Endereco;

import cadastrobd.enums.TipoPessoa;

public class PessoaJuridicaDAO {

    private final ConectorBD conectorBD;

    public PessoaJuridicaDAO(ConectorBD conectorBD)
{
        this.conectorBD = conectorBD;
    }

    public PessoaJuridica getPessoaJuridica(int id)
throws SQLException {
        PessoaJuridica PessoaJuridica = null;
        Connection connection =
```

```
conectorBD.getConnection();
        try {
            PreparedStatement preparedStatement =
conectorBD.getPreparedStatement(connection, "SELECT
\n"
                    + "  p.*,\n"
                    + "  pj.*,\n"
                    + "  e.*\n"
                    + "FROM \n"
                    + "  Pessoa p\n"
                    + "  INNER JOIN Pessoa_Juridica
pj ON p.id = pj.pessoa_id\n"
                    + "  INNER JOIN Endereco e ON
p.id_endereco = e.id\n"
                    + "WHERE \n"
                    + "  p.id = ?;", id);
            ResultSet resultSet =
preparedStatement.executeQuery();
            if (resultSet.next()) {
                PessoaJuridica = new
PessoaJuridica(
                        resultSet.getInt("id"),

resultSet.getString("nome"),

resultSet.getString("telefone"),
                        new
Endereco(resultSet.getInt("id_endereco"),
resultSet.getString("logradouro"),
resultSet.getString("bairro"),
resultSet.getString("numero"),
resultSet.getString("cidade"),
resultSet.getString("estado"),
resultSet.getString("complemento"),
resultSet.getString("cep")),
```

```
resultSet.getString("cnpj"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
            System.out.println("getPessoaJuridica
->  " + e);
        } finally {
            conectorBD.close(connection);
        }
        return PessoaJuridica;
    }

    public List<PessoaJuridica>
getPessoaJuridicas() throws SQLException {
        List<PessoaJuridica> pessoaJuridicas = new
ArrayList<>();
        Connection connection =
conectorBD.getConnection();
        try {
            PreparedStatement preparedStatement =
conectorBD.getPreparedStatement(connection, "SELECT
* FROM Pessoa p INNER JOIN Pessoa_Juridica pj ON
p.id = pj.pessoa_id INNER JOIN Endereco e ON
p.id_endereco = e.id WHERE p.id_tipo_pessoa = ?",
TipoPessoa.JURIDICA.getTipo()
            );
            ResultSet resultSet =
preparedStatement.executeQuery();
            while (resultSet.next()) {
                PessoaJuridica pessoaJuridica = new
PessoaJuridica(
                        resultSet.getInt("id"),

resultSet.getString("nome"),
```

```
resultSet.getString("telefone"),
                        new
Endereco(resultSet.getInt("id_endereco"),
resultSet.getString("logradouro"),
resultSet.getString("bairro"),
resultSet.getString("numero"),
resultSet.getString("cidade"),
resultSet.getString("estado"),
resultSet.getString("complemento"),
resultSet.getString("cep")),
                        resultSet.getString("cnpj")
                );

pessoaJuridicas.add(pessoaJuridica);
            }
        } catch (SQLException e) {
            System.out.println("getPessoaJuridicas
->  " + e);
        } finally {
            conectorBD.close(connection);
        }
        return pessoaJuridicas;
    }

    public void
incluirPessoaJuridica(PessoaJuridica
PessoaJuridica) throws SQLException {
        Connection connection =
conectorBD.getConnection();
        try {
            connection.setAutoCommit(false);

            int idEnderecoCadastrado =
cadastrarEndereco(PessoaJuridica.getEndereco(),
```

```
connection);

            PreparedStatement
preparedStatementPessoa =
conectorBD.getPreparedStatement(connection, "INSERT
INTO Pessoa (nome, telefone, id_endereco,
id_tipo_pessoa) VALUES (?, ?, ?, ?)",
PessoaJuridica.getNome(),
PessoaJuridica.getTelefone(), idEnderecoCadastrado,
TipoPessoa.JURIDICA.getTipo());

preparedStatementPessoa.executeUpdate();

            if
(preparedStatementPessoa.getGeneratedKeys().next())
{
                int id =
preparedStatementPessoa.getGeneratedKeys().getInt(1
);
                PessoaJuridica.setId(id);
                PreparedStatement
preparedStatementPessoaJuridica =
conectorBD.getPreparedStatement(connection, "INSERT
INTO Pessoa_Juridica  (pessoa_id, cnpj) VALUES (?,
?)", id, PessoaJuridica.getCnpj());

preparedStatementPessoaJuridica.executeUpdate();
                connection.commit();
                System.out.println("Pessoa Juridica
cadastrada com sucesso!");
            } else {
                System.out.println("A instrução
INSERT não gerou uma chave.");
            }
        } catch (SQLException e) {
```

```
            e.printStackTrace();
            try {
                connection.rollback();
            } catch (SQLException err) {

System.out.println("incluirPessoaJuridica ->  " +
err);
            }
        } finally {
            conectorBD.close(connection);
        }
    }

    private int cadastrarEndereco(Endereco
endereco, Connection connection) throws
SQLException {
        PreparedStatement preparedStatementEndereco
= conectorBD.getPreparedStatement(connection,
"INSERT INTO Endereco (logradouro, numero, bairro,
cidade, estado, complemento, cep) VALUES (?, ?, ?,
?, ?, ?, ?)",
                endereco.getLogradouro(),
endereco.getNumero(), endereco.getBairro(),
endereco.getCidade(), endereco.getEstado(),
endereco.getComplemento(), endereco.getCep());
        preparedStatementEndereco.executeUpdate();

        if
(preparedStatementEndereco.getGeneratedKeys().next(
)) {
            int id =
preparedStatementEndereco.getGeneratedKeys().getInt
(1);
            endereco.setId(id);
            System.out.println("Endereco cadastrado
```

```
com sucesso!");
            return id;
        } else {
            System.out.println("A instrução INSERT
não gerou uma chave.");
            return 0;
        }
    }

    public void
alterarPessoaJuridica(PessoaJuridica
PessoaJuridica) throws SQLException {
        Connection connection =
conectorBD.getConnection();
        try {
            connection.setAutoCommit(false);
            PreparedStatement
preparedStatementPessoa =
conectorBD.getPreparedStatement(connection, "UPDATE
Pessoa SET nome = ?, telefone = ?, id_endereco = ?
WHERE id = ?", PessoaJuridica.getNome(),
PessoaJuridica.getTelefone(),
PessoaJuridica.getEndereco().getId(),
PessoaJuridica.getId());

preparedStatementPessoa.executeUpdate();
            PreparedStatement
preparedStatementPessoaJuridica =
conectorBD.getPreparedStatement(connection, "UPDATE
Pessoa_Juridica SET cnpj = ? WHERE pessoa_id = ?",
PessoaJuridica.getCnpj(), PessoaJuridica.getId());

preparedStatementPessoaJuridica.executeUpdate();
            connection.commit();
        } catch (SQLException e) {
```

```
            e.printStackTrace();
            try {
                connection.rollback();
            } catch (SQLException err) {

System.out.println("alterarPessoaJuridica ->  " +
err);
            } finally {
                conectorBD.close(connection);
            }
        }
    }

    public void excluirPessoaJuridica(int id)
throws SQLException {
        Connection connection =
conectorBD.getConnection();
        try {
            connection.setAutoCommit(false);
            PreparedStatement
preparedStatementMovimentacaoVenda =
conectorBD.getPreparedStatement(connection, "DELETE
FROM Movimentacao_Venda WHERE id_pessoa_Juridica =
?", id);

preparedStatementMovimentacaoVenda.executeUpdate();
            PreparedStatement
preparedStatementPessoaJuridica =
conectorBD.getPreparedStatement(connection, "DELETE
FROM Pessoa_Juridica WHERE pessoa_id = ?", id);

preparedStatementPessoaJuridica.executeUpdate();
            PreparedStatement
preparedStatementPessoa =
conectorBD.getPreparedStatement(connection, "DELETE
```

```
FROM Pessoa WHERE id = ? AND id_tipo_pessoa = 2",
id);

preparedStatementPessoa.executeUpdate();
            connection.commit();
        } catch (SQLException e) {
            e.printStackTrace();
            try {
                connection.rollback();
            } catch (SQLException err) {

System.out.println("excluirPessoaJuridica ->  " +
err);
            } finally {
                conectorBD.close(connection);
            }
        }
    }
}
```

- Classe CadastroBDTeste
    - Incluir Pessoa Física ao banco

```
package cadastrobd;

import cadastro.model.PessoaFisicaDAO;
import cadastro.model.PessoaJuridicaDAO;
import cadastro.model.util.ConectorBD;
import cadastrobd.enums.TipoPessoa;

import cadastrobd.model.Endereco;
import cadastrobd.model.PessoaFisica;
import java.sql.SQLException;
```

```
public class CadastroBDTeste {

    public static void main(String[] args) throws
SQLException {
        ConectorBD conectorBD = new ConectorBD();
        PessoaFisicaDAO pfDAO = new
PessoaFisicaDAO(conectorBD);

        Endereco endereco = new Endereco();
        endereco.setId(3);

        PessoaFisica pf = new PessoaFisica();
        pf.setNome("Isabela Arruda Braz");
        pf.setTelefone("24998375604");
        pf.setCpf("11245863521");
        pf.setEndereco(endereco);
        pf.setTipoPessoa(TipoPessoa.FISICA);
        pfDAO.incluirPessoaFisica(pf);
    }
}
```

| # | id | nome | telefone | id_endereco | id_tipo_pessoa |
|---|---|---|---|---|---|
| ▶1 | 1008 | Isabela Arruda Braz | 24998375604 | 3 | 1 |

| # | pessoa_id | id_tipo_pessoa | cpf |
|---|---|---|---|
| 1 | 1008 | 1 | 11245863521 |

- Alteração dos dados de Pessoa Física ao banco

```
package cadastrobd;

import cadastro.model.PessoaFisicaDAO;
import cadastro.model.PessoaJuridicaDAO;
import cadastro.model.util.ConectorBD;
```

```java
import cadastrobd.model.Endereco;
import cadastrobd.model.PessoaFisica;
import java.sql.SQLException;

public class CadastroBDTeste {

    public static void main(String[] args) throws
SQLException {
        ConectorBD conectorBD = new ConectorBD();
        PessoaFisicaDAO pfDAO = new
PessoaFisicaDAO(conectorBD);

        Endereco endereco = new Endereco();
        endereco.setId(2);

        PessoaFisica pf = new PessoaFisica();
        pf.setId(1008);
        pf.setNome("Zé");
        pf.setTelefone("24998375604");
        pf.setCpf("03304405566");
        pf.setEndereco(endereco);
        pfDAO.alterarPessoaFisica(pf);
}
          }
```

| # | id | nome | telefone | id_endereco | id_tipo_pessoa |
|---|---|---|---|---|---|
| 1 | 1008 | Zé | 24998375604 | 2 | 1 |

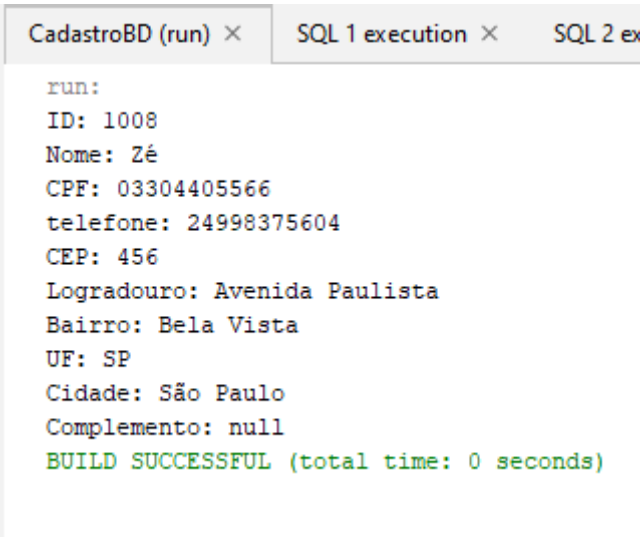
| # | pessoa_id | id_tipo_pessoa | cpf |
|---|---|---|---|
| 1 | 1008 | 1 | 03304405566 |

- Consulta de todas as pessoas físicas no banco

```java
package cadastrobd;
```

```java
import cadastro.model.PessoaFisicaDAO;
import cadastro.model.PessoaJuridicaDAO;
import cadastro.model.util.ConectorBD;


import cadastrobd.model.PessoaFisica;

import java.sql.SQLException;
import java.util.List;

public class CadastroBDTeste {

    public static void main(String[] args) throws
SQLException {
        ConectorBD conectorBD = new ConectorBD();
        PessoaFisicaDAO pfDAO = new
PessoaFisicaDAO(conectorBD);
        PessoaJuridicaDAO pjDAO = new
PessoaJuridicaDAO(conectorBD);


        List<PessoaFisica> pf =
pfDAO.getPessoasFisicas();
        for (int i = 0; i < pf.size(); i++) {
            System.out.println("ID: " +
pf.get(i).getId());
            System.out.println("Nome: " +
pf.get(i).getNome());
            System.out.println("CPF: " +
pf.get(i).getCpf());
            System.out.println("telefone: " +
pf.get(i).getTelefone());
            System.out.println("CEP: " +
pf.get(i).getEndereco().getNumero());
```

```
            System.out.println("Logradouro: " +
pf.get(i).getEndereco().getLogradouro());
            System.out.println("Bairro: " +
pf.get(i).getEndereco().getBairro());
            System.out.println("UF: " +
pf.get(i).getEndereco().getEstado());
            System.out.println("Cidade: " +
pf.get(i).getEndereco().getCidade());
            System.out.println("Complemento: " +
pf.get(i).getEndereco().getComplemento());
        }
    }
}
```




- Exclusão da pessoa física

```
package cadastrobd;

import cadastro.model.PessoaFisicaDAO;
import cadastro.model.PessoaJuridicaDAO;
import cadastro.model.util.ConectorBD;

import java.sql.SQLException;
```

```
public class CadastroBDTeste {

    public static void main(String[] args) throws
SQLException {
        ConectorBD conectorBD = new ConectorBD();
        PessoaFisicaDAO pfDAO = new
PessoaFisicaDAO(conectorBD);


        pfDAO.excluirPessoaFisica(1008);
    }
}
```

| # | id | nome | telefone | id_endereco | id_tipo_pessoa |
|---|---|---|---|---|---|

| # | pessoa_id | id_tipo_pessoa | cpf |
|---|---|---|---|

- Incluir Pessoa Jurídica ao banco

```
package cadastrobd;

import cadastro.model.PessoaJuridicaDAO;
import cadastro.model.util.ConectorBD;

import cadastrobd.enums.TipoPessoa;

import cadastrobd.model.Endereco;
import cadastrobd.model.PessoaJuridica;

import java.sql.SQLException;

public class CadastroBDTeste {
```

```
    public static void main(String[] args) throws
SQLException {
        ConectorBD conectorBD = new ConectorBD();
        PessoaJuridicaDAO pjDAO = new
PessoaJuridicaDAO(conectorBD);

        Endereco endereco = new Endereco();
        endereco.setId(3);
        PessoaJuridica pj = new PessoaJuridica();
        pj.setNome("Best2Bee LTDA");
        pj.setTelefone("24998989898");
        pj.setCnpj("03290750000133");
        pj.setEndereco(endereco);
        pj.setTipoPessoa(TipoPessoa.JURIDICA);
        pjDAO.incluirPessoaJuridica(pj);
    }
}
```

| # | id | nome | telefone | id_endereco | id_tipo_pessoa |
|---|---|---|---|---|---|
| 1 | 1009 | Best2Bee LTDA | 24998989898 | 3 | 2 |

| # | pessoa_id | id_tipo_pessoa | cnpj |
|---|---|---|---|
| 1 | 1009 | 2 | 03290750000133 |

- Alteração dos dados de Pessoa Jurídica ao banco

```
package cadastrobd;

import cadastro.model.PessoaJuridicaDAO;
import cadastro.model.util.ConectorBD;

import cadastrobd.model.Endereco;
import cadastrobd.model.PessoaJuridica;

import java.sql.SQLException;
```

```
public class CadastroBDTeste {

    public static void main(String[] args) throws
SQLException {
        ConectorBD conectorBD = new ConectorBD();
        PessoaJuridicaDAO pjDAO = new
PessoaJuridicaDAO(conectorBD);

        Endereco endereco = new Endereco();
        endereco.setId(2);
        PessoaJuridica pj = new PessoaJuridica();
        pj.setId(1009);
        pj.setNome("B2B");
        pj.setTelefone("24998375604");
        pj.setCnpj("03290750000134");
        pj.setEndereco(endereco);
        pjDAO.alterarPessoaJuridica(pj);
    }
}
```

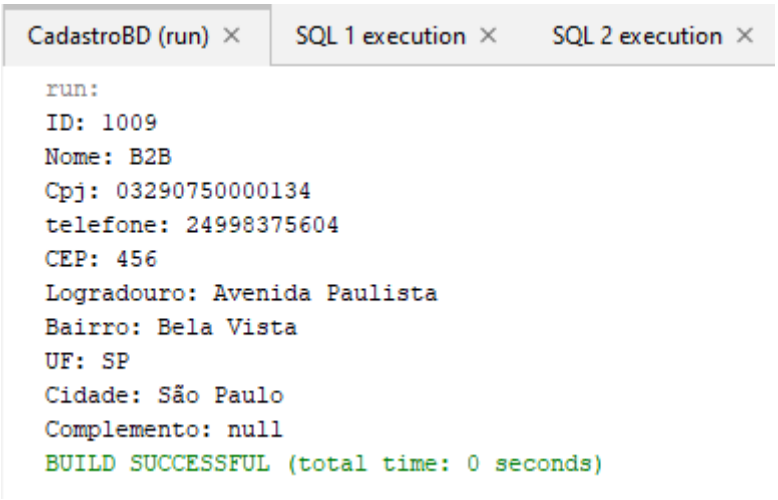
| # | id | nome | telefone | id_endereco | id_tipo_pessoa |
|---|---|---|---|---|---|
| 1 | 1009 | B2B | 24998375604 | 2 | 2 |

| # | pessoa_id | id_tipo_pessoa | cnpj |
|---|---|---|---|
| 1 | 1009 | 2 | 03290750000134 |

- Consulta de todas as pessoas físicas no banco

```
package cadastrobd;

import cadastro.model.PessoaJuridicaDAO;
import cadastro.model.util.ConectorBD;

import cadastrobd.model.PessoaJuridica;

import java.util.List;

import java.sql.SQLException;
```

```
public class CadastroBDTeste {

    public static void main(String[] args) throws
SQLException {
        ConectorBD conectorBD = new ConectorBD();
        PessoaJuridicaDAO pjDAO = new
PessoaJuridicaDAO(conectorBD);

        List<PessoaJuridica> pj =
pjDAO.getPessoaJuridicas();
        for (int i = 0; i < pj.size(); i++) {
            System.out.println("ID: " +
pj.get(i).getId());
            System.out.println("Nome: " +
pj.get(i).getNome());
            System.out.println("Cpj: " +
pj.get(i).getCnpj());
            System.out.println("telefone: " +
pj.get(i).getTelefone());
            System.out.println("CEP: " +
pj.get(i).getEndereco().getNumero());
            System.out.println("Logradouro: " +
pj.get(i).getEndereco().getLogradouro());
            System.out.println("Bairro: " +
pj.get(i).getEndereco().getBairro());
            System.out.println("UF: " +
pj.get(i).getEndereco().getEstado());
            System.out.println("Cidade: " +
pj.get(i).getEndereco().getCidade());
            System.out.println("Complemento: " +
pj.get(i).getEndereco().getComplemento());
        }
   }
}
```




- Exclusão da pessoa jurídica

```
package cadastrobd;

import cadastro.model.PessoaJuridicaDAO;
import cadastro.model.util.ConectorBD;

import java.sql.SQLException;

public class CadastroBDTeste {

    public static void main(String[] args) throws
SQLException {
        ConectorBD conectorBD = new ConectorBD();
        PessoaJuridicaDAO pjDAO = new
PessoaJuridicaDAO(conectorBD);

        pjDAO.excluirPessoaJuridica(1009);
    }

}
```

| # | id | nome | telefone | id_endereco | id_tipo_pessoa |
|---|---|---|---|---|---|

| # | pessoa_id | id_tipo_pessoa | cnpj |
|---|---|---|---|

## Análises

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

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

R: *A principal diferença entre eles reside em sua funcionalidade e desempenho. Um objeto Statement é usado para executar uma instrução SQL estática, a instrução SQL é compilada a cada vez que é executada, o que pode levar a uma perda de desempenho.*

*Um objeto PreparedStatement é usado para executar uma instrução SQL pré-compilada, a instrução SQL é compilada apenas uma vez, o que melhora o desempenho. É mais seguro do que Statement, pois os parâmetros são tratados como valores e não como parte da instrução SQL, o que reduz a chance de ataques de injeção de SQL.*

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

R: *Ele separa a lógica de acesso a dados da lógica de negócios. Isso facilita a manutenção do software, permite reutilizar o código de acesso a dados em diferentes partes do sistema e torna mais fácil mudar a tecnologia de acesso a dados, pois a lógica de acesso a dados está encapsulada em uma camada separada.*

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

R: *A herança é refletida no banco de dados por meio de tabelas separadas para cada classe.*

*Uma classe genérica, como por exemplo* ***Pessoa*** *e outra classe* ***Pessoa_Fisica.*** *A classe* ***Pessoa_Fisica*** *teria uma chave estrangeira referenciando a tabela* ***Pessoa****, representando a relação de herança.*

**2º Procedimento | Alimentando a Base**

- Classe CadastroBD

```
package cadastrobd;

import cadastro.model.PessoaFisicaDAO;
import cadastro.model.PessoaJuridicaDAO;
import cadastrobd.model.PessoaFisica;
import cadastrobd.model.PessoaJuridica;
import cadastro.model.util.ConectorBD;
import cadastrobd.model.Endereco;
import java.sql.SQLException;
import java.util.List;
import java.util.Scanner;

public class CadastroBD {

    public static void main(String[] args) throws
SQLException {
        try (Scanner scanner = new
Scanner(System.in)) {
            ConectorBD conectorBD = new
ConectorBD();
            PessoaFisicaDAO pessoaFisicaDAO = new
PessoaFisicaDAO(conectorBD);
            PessoaJuridicaDAO pessoaJuridicaDAO =
new PessoaJuridicaDAO(conectorBD);
            int option;

            do {

System.out.println("==============================="
);
```

```
                System.out.println("1 - Incluir
Pessoa");
                System.out.println("2 - Alterar
Pessoa");
                System.out.println("3 - Excluir
Pessoa");
                System.out.println("4 - Buscar pelo
Id");
                System.out.println("5 - Exibir
Todos");
                System.out.println("0 - Finalizar
Programa");

System.out.println("==============================="
);
                option = scanner.nextInt();
                scanner.nextLine();

                switch (option) {
                    case 1 ->
                        incluirPessoa(scanner,
pessoaFisicaDAO, pessoaJuridicaDAO);
                    case 2 ->
                        alterarPessoa(scanner,
pessoaFisicaDAO, pessoaJuridicaDAO);
                    case 3 ->
                        excluirPessoa(scanner,
pessoaFisicaDAO, pessoaJuridicaDAO);
                    case 4 ->
                        buscarPorId(scanner,
pessoaFisicaDAO, pessoaJuridicaDAO);
                    case 5 ->
                        exibirTodos(scanner,
pessoaFisicaDAO, pessoaJuridicaDAO);
                    case 0 ->
```

```
System.out.println("Finalizando o programa...");
                    default ->
                        System.out.println("Opção
inválida. Tente novamente.");
                }
            } while (option != 0);
        }
    }

    private static void incluirPessoa(Scanner
scanner, PessoaFisicaDAO pfDAO, PessoaJuridicaDAO
pjDAO) {
        System.out.println("F - Pessoa Física | J -
Pessoa Jurídica");
        String tipo =
scanner.nextLine().toUpperCase();
        try {
            if (!tipo.equals("F")) {
                if (tipo.equals("J")) {
                    System.out.println("Nome: ");
                    String nome =
scanner.nextLine();
                    if (nome.isEmpty()) {
                        System.out.println("Nome é
obrigatório.");
                        return;
                    }
                    System.out.println("CNPJ(APENAS
DIGITOS): ");
                    String cnpj =
scanner.nextLine();
                    if (cnpj.isEmpty()) {
                        System.out.println("CNPJ é
obrigatório.");
```

```
                        return;
                    }
                    if (!cnpj.matches("\\d{14}")) {
                        System.out.println("CNPJ
invalido! Deve conter apenas 14 digitos.");
                        return;
                    }
                    System.out.println("Telefone:
");
                    String telefone =
scanner.nextLine();

                    if (telefone.isEmpty()) {

System.out.println("Telefone é obrigatório.");
                        return;
                    }
                    System.out.println("CEP(APENAS
DIGITOS): ");
                    String cep =
scanner.nextLine();
                    if (cep.isEmpty()) {
                        System.out.println("CEP é
obrigatório.");
                        return;
                    }
                    if (!cep.matches("\\d{8}")) {
                        System.out.println("CEP
invalido! Deve conter apenas 8 digitos.");
                        return;
                    }
                    System.out.println("Logradouro:
");
                    String logradouro =
scanner.nextLine();
```

```
                    if (logradouro.isEmpty()) {
System.out.println("Logradouro é obrigatório.");
                        return;
                    }
                    System.out.println("Bairro: ");
                    String bairro =
scanner.nextLine();
                    if (bairro.isEmpty()) {
                        System.out.println("Bairro
é obrigatório.");
                        return;
                    }
                    System.out.println("Número: ");
                    String numero =
scanner.nextLine();
                    if (numero.isEmpty()) {
                        System.out.println("Número
é obrigatório.");
                        return;
                    }
                    System.out.println("Cidade: ");
                    String cidade =
scanner.nextLine();
                    if (cidade.isEmpty()) {
                        System.out.println("Cidade
é obrigatório.");
                        return;
                    }
                    System.out.println("UF: ");
                    String uf = scanner.nextLine();
                    if (uf.isEmpty()) {
                        System.out.println("UF é
obrigatório.");
                        return;
```

```
                    }
                    if (!uf.matches("[a-zA-Z]{2}"))
{
                        System.out.println("UF
invalido! Deve conter apenas 2 letras.");
                        return;
                    }

System.out.println("Complemento: ");
                    String complemento =
scanner.nextLine();
                    Endereco endereco = new
Endereco();
                    endereco.setCep(cep);

endereco.setLogradouro(logradouro);
                    endereco.setBairro(bairro);
                    endereco.setNumero(numero);
                    endereco.setCidade(cidade);
                    endereco.setEstado(uf);

endereco.setComplemento(complemento);
                    PessoaJuridica pj = new
PessoaJuridica();
                    pj.setNome(nome);
                    pj.setTelefone(telefone);
                    pj.setEndereco(endereco);
                    pj.setCnpj(cnpj);

pjDAO.incluirPessoaJuridica(pj);
                } else {
                    System.out.println("Tipo
inválido.");
                }
            } else {
```

```
                System.out.println("Nome: ");
                String nome = scanner.nextLine();
                if (nome.isEmpty()) {
                    System.out.println("Nome é
obrigatório.");
                    return;
                }
                if
(!nome.matches("[A-Za-z\\s\\p{Punct}]+")) {
                    System.out.println("Nome
invalido!");
                    return;
                }
                System.out.println("CPF(APENAS
DIGITOS): ");
                String cpf = scanner.nextLine();
                if (cpf.isEmpty()) {
                    System.out.println("CPF
invalido! Deve ser uma String nao vazia.");
                    return;
                }
                if (!cpf.matches("\\d{11}")) {
                    System.out.println("CPF
invalido! Deve conter apenas 11 digitos.");
                    return;
                }
                System.out.println("Telefone: ");
                String telefone =
scanner.nextLine();
                if (telefone.isEmpty()) {
                    System.out.println("Telefone é
obrigatório.");
                    return;
                }
                System.out.println("CEP(APENAS
```

```
DIGITOS): ");
                String cep = scanner.nextLine();
                if (cep.isEmpty()) {
                    System.out.println("CEP é
obrigatório.");
                    return;
                }
                if (!cep.matches("\\d{8}")) {
                    System.out.println("CEP
invalido! Deve conter apenas 8 digitos.");
                    return;
                }
                System.out.println("Logradouro: ");
                String logradouro =
scanner.nextLine();
                if (logradouro.isEmpty()) {
                    System.out.println("Logradouro
é obrigatório.");
                    return;
                }
                System.out.println("Bairro: ");
                String bairro = scanner.nextLine();
                if (bairro.isEmpty()) {
                    System.out.println("Bairro é
obrigatório.");
                    return;
                }
                System.out.println("Número: ");
                String numero = scanner.nextLine();
                if (numero.isEmpty()) {
                    System.out.println("Número é
obrigatório.");
                    return;
                }
                System.out.println("Cidade: ");
```

```
                String cidade = scanner.nextLine();
                if (cidade.isEmpty()) {
                    System.out.println("Cidade é
obrigatório.");
                    return;
                }
                System.out.println("UF: ");
                String uf = scanner.nextLine();
                if (uf.isEmpty()) {
                    System.out.println("UF é
obrigatório.");
                    return;
                }
                if (!uf.matches("[a-zA-Z]{2}")) {
                    System.out.println("UF
invalido! Deve conter apenas 2 letras.");
                    return;
                }
                System.out.println("Complemento:
");
                String complemento =
scanner.nextLine();
                Endereco endereco = new Endereco();
                endereco.setCep(cep);
                endereco.setLogradouro(logradouro);
                endereco.setBairro(bairro);
                endereco.setNumero(numero);
                endereco.setCidade(cidade);
                endereco.setEstado(uf);

endereco.setComplemento(complemento);
                PessoaFisica pf = new
PessoaFisica();
                pf.setNome(nome);
                pf.setTelefone(telefone);
```

```
                pf.setEndereco(endereco);
                pf.setCpf(cpf);
                pfDAO.incluirPessoaFisica(pf);
            }
        } catch (Exception e) {
            System.out.println("Erro ao incluir
pessoa: " + e.getMessage());
        }
    }

    private static void alterarPessoa(Scanner
scanner, PessoaFisicaDAO pfDAO, PessoaJuridicaDAO
pjDAO) {
        System.out.println("F - Pessoa Física | J -
Pessoa Jurídica");
        String tipo =
scanner.nextLine().toUpperCase();

        try {
            if (!tipo.equals("F")) {
                if (tipo.equals("J")) {
                    System.out.println("Informe o
ID da Pessoa Jurídica que deseja alterar: ");
                    int id =
Integer.parseInt(scanner.nextLine());

                    PessoaJuridica pj =
pjDAO.getPessoaJuridica(id);
                    if (pj != null) {
                        System.out.println("Dados
atuais:");
                        System.out.println("Nome: "
+ pj.getNome());
                        System.out.println("CNPJ: "
+ pj.getCnpj());
```

```
System.out.println("Telefone: " +
pj.getTelefone());
                        System.out.println("CEP: "
+ pj.getEndereco().getCep());

System.out.println("Logradouro: " +
pj.getEndereco().getLogradouro());
                        System.out.println("Bairro:
" + pj.getEndereco().getBairro());
                        System.out.println("Número:
" + pj.getEndereco().getNumero());
                        System.out.println("Cidade:
" + pj.getEndereco().getCidade());
                        System.out.println("UF: " +
pj.getEndereco().getEstado());

System.out.println("Complemento: " +
pj.getEndereco().getComplemento());

                        System.out.println("Informe
o novo nome (Ou aperte ENTER para manter): ");
                        String nome =
scanner.nextLine();
                        if (!nome.isEmpty()) {
                            pj.setNome(nome);
                        }
                        System.out.println("Informe
o novo CNPJ com APENAS DIGITOS (Ou aperte ENTER
para manter): ");
                        String cnpj =
scanner.nextLine();
                        if (!cnpj.isEmpty() &&
!cnpj.matches("\\d{14}")) {
```

```
System.out.println("CNPJ invalido! Deve conter
apenas 14 digitos.");
                            return;
                        }
                        if (!cnpj.isEmpty()) {
                            pj.setCnpj(cnpj);
                        }
                        System.out.println("Informe
o novo telefone (Ou aperte ENTER para manter): ");
                        String telefone =
scanner.nextLine();
                        if (!telefone.isEmpty()) {

pj.setTelefone(telefone);
                        }
                        System.out.println("Informe
o novo CEP (Ou aperte ENTER para manter)(APENAS
DIGITOS): ");
                        String cep =
scanner.nextLine();
                        if (!cep.isEmpty() &&
!cep.matches("\\d{8}")) {
                            System.out.println("CEP
invalido! Deve conter apenas 8 digitos.");
                            return;
                        }
                        if (!cep.isEmpty()) {

pj.getEndereco().setCep(cep);
                        }
                        System.out.println("Informe
o novo logradouro (Ou aperte ENTER para manter):
");
                        String logradouro =
scanner.nextLine();
```

```
                        if (!logradouro.isEmpty())
{

pj.getEndereco().setLogradouro(logradouro);
                        }
                        System.out.println("Informe
o novo bairro (Ou aperte ENTER para manter): ");
                        String bairro =
scanner.nextLine();
                        if (!bairro.isEmpty()) {

pj.getEndereco().setBairro(bairro);
                        }
                        System.out.println("Informe
o novo número (Ou aperte ENTER para manter): ");
                        String numero =
scanner.nextLine();
                        if (!numero.isEmpty()) {

pj.getEndereco().setNumero(numero);
                        }
                        System.out.println("Informe
o novo cidade (Ou aperte ENTER para manter): ");
                        String cidade =
scanner.nextLine();
                        if (!cidade.isEmpty()) {

pj.getEndereco().setCidade(cidade);
                        }
                        System.out.println("Informe
a nova UF (Ou aperte ENTER para manter): ");
                        String uf =
scanner.nextLine();
                        if (!uf.isEmpty() &&
!uf.matches("[a-zA-Z]{2}")) {
```

```
                            System.out.println("UF
invalido! Deve conter apenas 2 letras.");
                            return;
                        }
                        if (!uf.isEmpty()) {

pj.getEndereco().setEstado(uf);
                        }
                        System.out.println("Informe
o novo complemento (Ou aperte ENTER para manter):
");
                        String complemento =
scanner.nextLine();
                        if (!complemento.isEmpty())
{

pj.getEndereco().setComplemento(complemento);
                        }

pjDAO.alterarPessoaJuridica(pj);
                        System.out.println("Pessoa
Jurídica atualizada com sucesso!");
                    } else {
                        System.out.println("Nenhuma
pessoa jurídica com o ID informado foi
encontrada.");
                    }
                } else {
                    System.out.println("Tipo
inválido.");
                }
            } else {
                System.out.println("Informe o ID da
Pessoa Física que deseja alterar: ");
                int id =
```

```
Integer.parseInt(scanner.nextLine());

                PessoaFisica pf =
pfDAO.getPessoaFisica(id);
                if (pf != null) {
                    System.out.println("Dados
atuais: ");
                    System.out.println("Nome: " +
pf.getNome());
                    System.out.println("CPF: " +
pf.getCpf());
                    System.out.println("Telefone: "
+ pf.getTelefone());
                    System.out.println("CEP: " +
pf.getEndereco().getCep());
                    System.out.println("Logradouro:
" + pf.getEndereco().getLogradouro());
                    System.out.println("Bairro: " +
pf.getEndereco().getBairro());
                    System.out.println("Número: " +
pf.getEndereco().getNumero());
                    System.out.println("Cidade: " +
pf.getEndereco().getCidade());
                    System.out.println("UF: " +
pf.getEndereco().getEstado());

System.out.println("Complemento: " +
pf.getEndereco().getComplemento());

                    System.out.println("Informe o
novo nome (Ou aperte ENTER para manter): ");
                    String nome =
scanner.nextLine();
                    if
(!nome.matches("[A-Za-z\\s\\p{Punct}]+")) {
```

```
                        System.out.println("Nome
invalido!");
                        return;
                    }
                    if (!nome.isEmpty()) {
                        pf.setNome(nome);
                    }
                    System.out.println("Informe o
novo CPF com APENAS DIGITOS (Ou aperte ENTER para
manter): ");
                    String cpf =
scanner.nextLine();
                    if (!cpf.isEmpty() &&
!cpf.matches("\\d{11}")) {
                        System.out.println("CPF
invalido! Deve conter apenas 8 digitos.");
                        return;
                    }
                    if (!cpf.isEmpty()) {
                        pf.setCpf(cpf);
                    }
                    System.out.println("Informe o
novo telefone (Ou aperte ENTER para manter): ");
                    String telefone =
scanner.nextLine();
                    if (!telefone.isEmpty()) {
                        pf.setTelefone(telefone);
                    }
                    System.out.println("Informe o
novo CEP (Ou aperte ENTER para manter)(APENAS
DIGITOS): ");
                    String cep =
scanner.nextLine();
                    if (!cep.isEmpty() &&
!cep.matches("\\d{8}")) {
```

```
                        System.out.println("CEP
invalido! Deve conter apenas 8 digitos.");
                        return;
                    }
                    if (!cep.isEmpty()) {

pf.getEndereco().setCep(cep);
                    }
                    System.out.println("Informe o
novo logradouro (Ou aperte ENTER para manter): ");
                    String logradouro =
scanner.nextLine();
                    if (!logradouro.isEmpty()) {

pf.getEndereco().setLogradouro(logradouro);
                    }
                    System.out.println("Informe o
novo bairro (Ou aperte ENTER para manter): ");
                    String bairro =
scanner.nextLine();
                    if (!bairro.isEmpty()) {

pf.getEndereco().setBairro(bairro);
                    }
                    System.out.println("Informe o
novo número (Ou aperte ENTER para manter): ");
                    String numero =
scanner.nextLine();
                    if (!numero.isEmpty()) {

pf.getEndereco().setNumero(numero);
                    }
                    System.out.println("Informe o
novo cidade (Ou aperte ENTER para manter): ");
                    String cidade =
```

```
scanner.nextLine();
                    if (!cidade.isEmpty()) {

pf.getEndereco().setCidade(cidade);
                    }
                    System.out.println("Informe a
nova UF (Ou aperte ENTER para manter): ");
                    String uf = scanner.nextLine();
                    if (!uf.isEmpty() &&
!uf.matches("[a-zA-Z]{2}")) {
                        System.out.println("UF
invalido! Deve conter apenas 2 letras.");
                        return;
                    }
                    if (!uf.isEmpty()) {

pf.getEndereco().setEstado(uf);
                    }
                    System.out.println("Informe o
novo complemento (Ou aperte ENTER para manter): ");
                    String complemento =
scanner.nextLine();
                    if (!complemento.isEmpty()) {

pf.getEndereco().setComplemento(complemento);
                    }
                    pfDAO.alterarPessoaFisica(pf);
                    System.out.println("Pessoa
Física atualizada com sucesso!");

                } else {
                    System.out.println("Nenhuma
pessoa física com o ID informado foi encontrada.");
                }
            }
```

```
        } catch (NumberFormatException e) {
            System.out.println("ID inválido. Por
favor, informe um número.");
        } catch (Exception e) {
            System.out.println("Erro ao alterar
pessoa: " + e.getMessage());
        }
    }

    private static void excluirPessoa(Scanner
scanner, PessoaFisicaDAO pfDAO, PessoaJuridicaDAO
pjDAO) {
        System.out.println("F - Pessoa Física | J -
Pessoa Jurídica");
        String tipo =
scanner.nextLine().toUpperCase();

        try {
            if (!tipo.equals("F")) {
                if (tipo.equals("J")) {
                    System.out.println("Informe o
ID da Pessoa Jurídica que deseja excluir: ");
                    int id =
Integer.parseInt(scanner.nextLine());

                    PessoaJuridica pessoa =
pjDAO.getPessoaJuridica(id);
                    if (pessoa != null) {
                        System.out.println("Tem
certeza que deseja excluir a Pessoa Jurídica com ID
" + id + "? (S/N)");
                        String confirmacao =
scanner.nextLine().toUpperCase();
                        if
(confirmacao.equals("S")) {
```

```
pjDAO.excluirPessoaJuridica(id);

System.out.println("Pessoa Jurídica excluída com
sucesso!");
                        } else {

System.out.println("Operação de exclusão
cancelada.");
                        }
                    } else {
                        System.out.println("Nenhuma
pessoa jurídica com o ID informado foi
encontrada.");
                    }
                } else {
                    System.out.println("Tipo
inválido.");
                }
            } else {
                System.out.println("Informe o ID da
Pessoa Física que deseja excluir: ");
                int id =
Integer.parseInt(scanner.nextLine());

                PessoaFisica pessoa =
pfDAO.getPessoaFisica(id);
                if (pessoa != null) {
                    System.out.println("Tem certeza
que deseja excluir a Pessoa Física com ID " + id +
"? (S/N)");
                    String confirmacao =
scanner.nextLine().toUpperCase();
                    if (confirmacao.equals("S")) {
```

```
pfDAO.excluirPessoaFisica(id);
                        System.out.println("Pessoa
Física excluída com sucesso!");
                    } else {

System.out.println("Operação de exclusão
cancelada.");
                    }
                } else {
                    System.out.println("Nenhuma
pessoa física com o ID informado foi encontrada.");
                }
            }
        } catch (NumberFormatException e) {
            System.out.println("ID inválido. Por
favor, informe um número.");
        } catch (Exception e) {
            System.out.println("Erro ao excluir
pessoa: " + e.getMessage());
        }
    }

    private static void buscarPorId(Scanner
scanner, PessoaFisicaDAO pfDAO, PessoaJuridicaDAO
pjDAO) throws SQLException {
        System.out.println("F - Pessoa Física | J -
Pessoa Jurídica");
        String tipo =
scanner.nextLine().toUpperCase();

        if (!"F".equals(tipo)) {
            if ("J".equals(tipo)) {
                System.out.println("Digite o ID da
pessoa que deseja buscar:");
                int id = scanner.nextInt();
```

```
                scanner.nextLine();
                PessoaJuridica pj =
pjDAO.getPessoaJuridica(id);
                if (pj != null) {
                    System.out.println("Exibindo
dados de Pessoa Jurídica...");

System.out.println("=========================");
                    System.out.println("Id: " +
pj.getId());
                    System.out.println("Nome: " +
pj.getNome());
                    System.out.println("CNPJ: " +
pj.getCnpj());
                    System.out.println("Telefone: "
+ pj.getTelefone());
                    System.out.println("Logradouro:
" + pj.getEndereco().getLogradouro());
                    System.out.println("Bairro: " +
pj.getEndereco().getBairro());
                    System.out.println("Numero: " +
pj.getEndereco().getNumero());
                    System.out.println("Cidade: " +
pj.getEndereco().getCidade());
                    System.out.println("Estado: " +
pj.getEndereco().getEstado());

System.out.println("Complemento: " +
pj.getEndereco().getComplemento());
                    System.out.println("CEP: " +
pj.getEndereco().getCep());
                } else {
                    System.out.println("Pessoa
Jurídica com ID " + id + " não encontrada.");
                }
```

```
            } else {
                System.out.println("Tipo inválido.
Por favor, digite 'F' para Pessoa Física ou 'J'
para Pessoa Jurídica.");
            }
        } else {
            System.out.println("Digite o ID da
pessoa que deseja buscar:");
            int id = scanner.nextInt();
            scanner.nextLine();
            PessoaFisica pf =
pfDAO.getPessoaFisica(id);
            if (pf != null) {
                System.out.println("Exibindo dados
de Pessoa Física...");

System.out.println("=========================");
                System.out.println("Id: " +
pf.getId());
                System.out.println("Nome: " +
pf.getNome());
                System.out.println("CPF: " +
pf.getCpf());
                System.out.println("Telefone: " +
pf.getTelefone());
                System.out.println("Logradouro: " +
pf.getEndereco().getLogradouro());
                System.out.println("Bairro: " +
pf.getEndereco().getBairro());
                System.out.println("Numero: " +
pf.getEndereco().getNumero());
                System.out.println("Cidade: " +
pf.getEndereco().getCidade());
                System.out.println("Estado: " +
pf.getEndereco().getEstado());
```

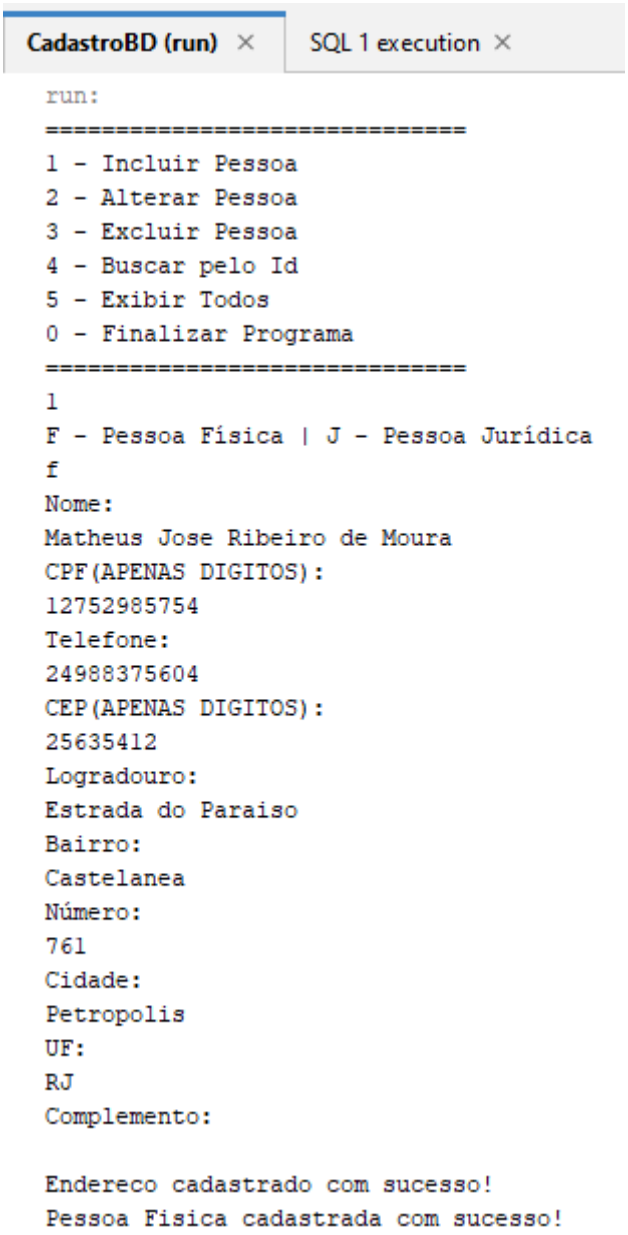
```
                System.out.println("Complemento: "
+ pf.getEndereco().getComplemento());
                System.out.println("CEP: " +
pf.getEndereco().getCep());
            } else {
                System.out.println("Pessoa Física
com ID " + id + " não encontrada.");
            }
        }
    }

    private static void exibirTodos(Scanner
scanner, PessoaFisicaDAO pfDAO, PessoaJuridicaDAO
pjDAO) throws SQLException {
        System.out.println("F - Pessoa Física | J -
Pessoa Jurídica");
        String tipo =
scanner.nextLine().toUpperCase();

        if (!"F".equals(tipo)) {
            if ("J".equals(tipo)) {
                List<PessoaJuridica>
pessoasJuridicas = pjDAO.getPessoaJuridicas();
                if (pessoasJuridicas.isEmpty()) {
                    System.out.println("Nenhuma
Pessoa Jurídica encontrada.");
                } else {
                    System.out.println("Exibindo
todas as Pessoas Jurídicas...");

System.out.println("=========================");
                    for (PessoaJuridica pj :
pessoasJuridicas) {
                        System.out.println("Id: " +
pj.getId());
```

```
                        System.out.println("Nome: "
+ pj.getNome());
                        System.out.println("CNPJ: "
+ pj.getCnpj());

System.out.println("Telefone: " +
pj.getTelefone());
                        System.out.println("CEP: "
+ pj.getEndereco().getCep());

System.out.println("Logradouro: " +
pj.getEndereco().getLogradouro());
                        System.out.println("Bairro:
" + pj.getEndereco().getBairro());
                        System.out.println("Numero:
" + pj.getEndereco().getNumero());
                        System.out.println("Cidade:
" + pj.getEndereco().getCidade());
                        System.out.println("Estado:
" + pj.getEndereco().getEstado());

System.out.println("Complemento: " +
pj.getEndereco().getComplemento());

System.out.println("-------------------------");
                    }
                }
            } else {
                System.out.println("Tipo
inválido.");
            }
        } else {
            List<PessoaFisica> pessoasFisicas =
pfDAO.getPessoasFisicas();
            if (pessoasFisicas.isEmpty()) {
```

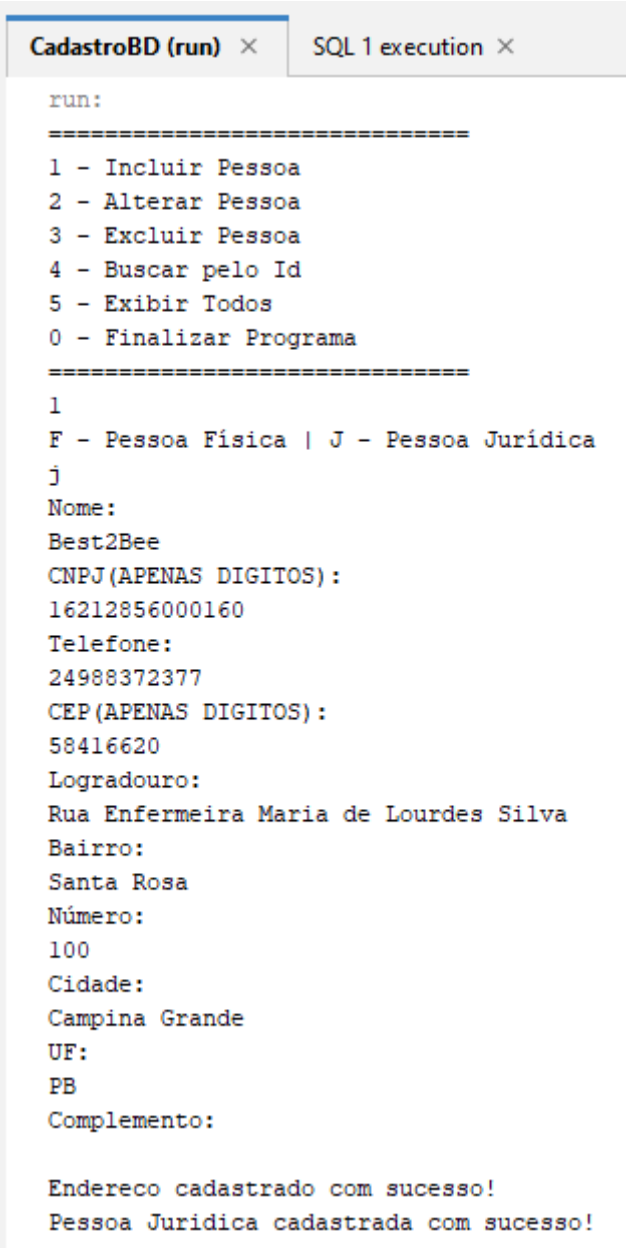
```
                System.out.println("Nenhuma Pessoa
Física encontrada.");
            } else {
                System.out.println("Exibindo todas
as Pessoas Físicas...");

System.out.println("=========================");
                for (PessoaFisica pf :
pessoasFisicas) {
                    System.out.println("Id: " +
pf.getId());
                    System.out.println("Nome: " +
pf.getNome());
                    System.out.println("CPF: " +
pf.getCpf());
                    System.out.println("Telefone: "
+ pf.getTelefone());
                    System.out.println("CEP: " +
pf.getEndereco().getCep());
                    System.out.println("Logradouro:
" + pf.getEndereco().getLogradouro());
                    System.out.println("Bairro: " +
pf.getEndereco().getBairro());
                    System.out.println("Numero: " +
pf.getEndereco().getNumero());
                    System.out.println("Cidade: " +
pf.getEndereco().getCidade());
                    System.out.println("Estado: " +
pf.getEndereco().getEstado());

System.out.println("Complemento: " +
pf.getEndereco().getComplemento());

System.out.println("-------------------------");
                }
```

```
            }
        }
    }
}
```

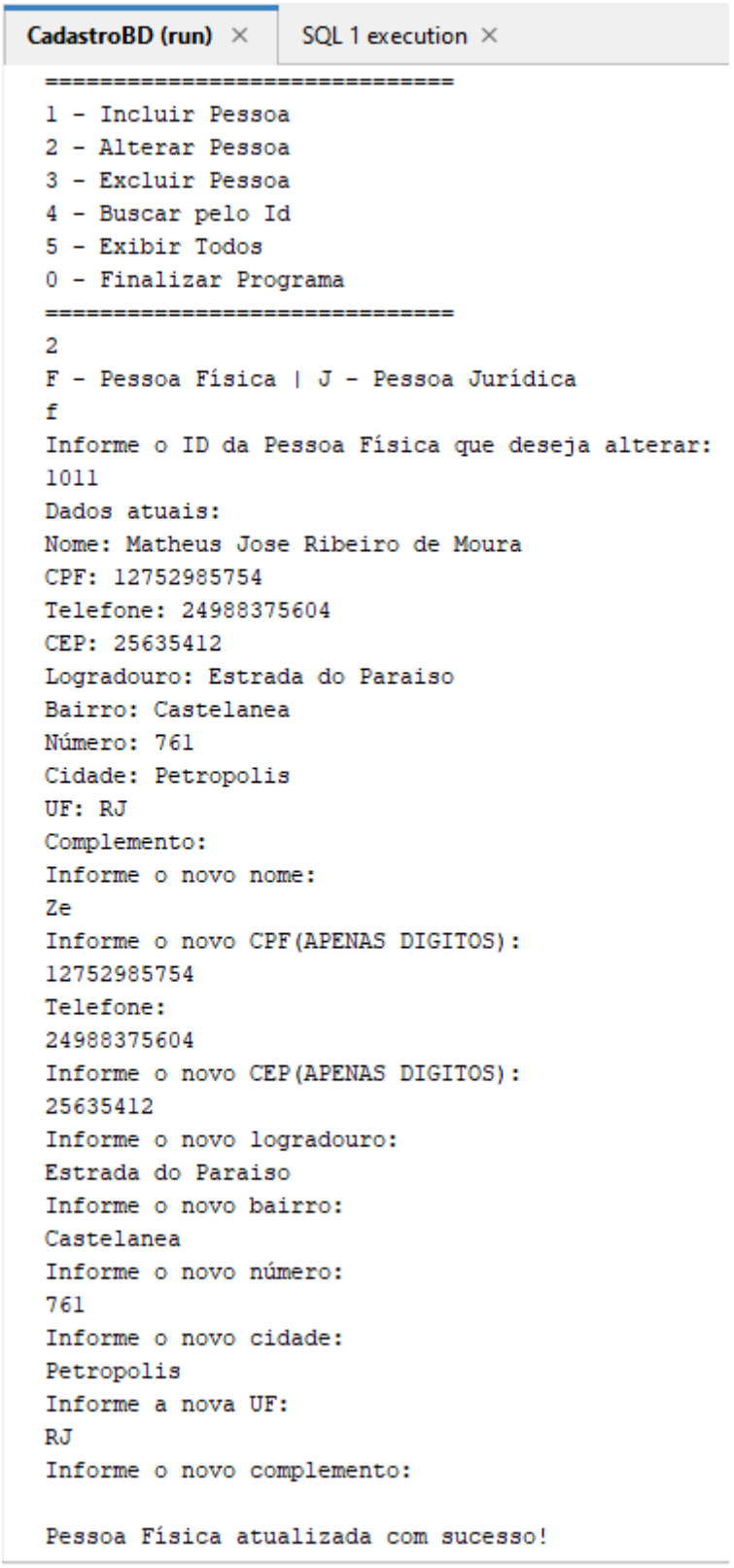
- Resultados da execução do código
    - Incluir Pessoa Física




- Incluir Pessoa Jurídica




- Alterar Pessoa Física




- Alterar Pessoa Jurídica

```
run:
==============================
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo Id
5 - Exibir Todos
0 - Finalizar Programa
==============================
2
F - Pessoa Física | J - Pessoa Jurídica
j
Informe o ID da Pessoa Jurídica que deseja alterar:
1013
Dados atuais:
Nome: Best2Bee
CNPJ: 16212856000160
Telefone: 24988372377
CEP: 58416620
Logradouro: Rua Enfermeira Maria de Lourdes Silva
Bairro: Santa Rosa
Número: 100
Cidade: Campina Grande
UF: PB
Complemento:
Informe o novo nome (Ou aperte ENTER para manter):
B2B
Informe o novo CNPJ com APENAS DIGITOS (Ou aperte ENTER para manter):

Informe o novo telefone (Ou aperte ENTER para manter):

Informe o novo CEP (Ou aperte ENTER para manter)(APENAS DIGITOS):

Informe o novo logradouro (Ou aperte ENTER para manter):

Informe o novo bairro (Ou aperte ENTER para manter):

Informe o novo número (Ou aperte ENTER para manter):

Informe o novo cidade (Ou aperte ENTER para manter):

Informe a nova UF (Ou aperte ENTER para manter):

Informe o novo complemento (Ou aperte ENTER para manter):

Pessoa Jurídica atualizada com sucesso!
```

- Buscar Pessoa Física por id

```
==============================
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo Id
5 - Exibir Todos
0 - Finalizar Programa
==============================
4
F - Pessoa Física | J - Pessoa Jurídica
f
Digite o ID da pessoa que deseja buscar:
1011
Exibindo dados de Pessoa Física...
==========================
Id: 1011
Nome: Ze
CPF: 12752985754
Telefone: 24988375604
Logradouro: Estrada do Paraiso
Bairro: Castelanea
Numero: 761
Cidade: Petropolis
Estado: RJ
Complemento:
CEP: 25635412
```

- Buscar Pessoa Jurídica por id

```
==============================
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo Id
5 - Exibir Todos
0 - Finalizar Programa
==============================
4
F - Pessoa Física | J - Pessoa Jurídica
j
Digite o ID da pessoa que deseja buscar:
1013
Exibindo dados de Pessoa Jurídica...
==========================
Id: 1013
Nome: B2B
CNPJ: 16212856000160
Telefone: 24988372377
Logradouro: Rua Enfermeira Maria de Lourdes Silva
Bairro: Santa Rosa
Numero: 100
Cidade: Campina Grande
Estado: PB
Complemento:
CEP: 58416620
```

- Exibir todas Pessoas Físicas

```
==============================
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo Id
5 - Exibir Todos
0 - Finalizar Programa
==============================
5
F - Pessoa Física | J - Pessoa Jurídica
f
Exibindo todas as Pessoas Físicas...
=========================
Id: 1011
Nome: Ze
CPF: 12752985754
Telefone: 24988375604
CEP: 25635412
Logradouro: Estrada do Paraiso
Bairro: Castelanea
Numero: 761
Cidade: Petropolis
Estado: RJ
Complemento:
-------------------------
```

- Exibir todas Pessoas Jurídicas

```
==============================
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo Id
5 - Exibir Todos
0 - Finalizar Programa
==============================
5
F - Pessoa Física | J - Pessoa Jurídica
j
Exibindo todas as Pessoas Jurídicas...
========================
Id: 1013
Nome: B2B
CNPJ: 16212856000160
Telefone: 24988372377
CEP: 58416620
Logradouro: Rua Enfermeira Maria de Lourdes Silva
Bairro: Santa Rosa
Numero: 100
Cidade: Campina Grande
Estado: PB
Complemento:
------------------------
```

- Excluir Pessoa Física

```
==============================
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo Id
5 - Exibir Todos
0 - Finalizar Programa
==============================
3
F - Pessoa Física | J - Pessoa Jurídica
f
Informe o ID da Pessoa Física que deseja excluir:
1011
Tem certeza que deseja excluir a Pessoa Física com ID 1011? (S/N)
s
Pessoa Física excluída com sucesso!
```

- Excluir Pessoa Jurídica

```
==============================
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo Id
5 - Exibir Todos
0 - Finalizar Programa
==============================
3
F - Pessoa Física | J - Pessoa Jurídica
j
Informe o ID da Pessoa Jurídica que deseja excluir:
1013
Tem certeza que deseja excluir a Pessoa Jurídica com ID 1013? (S/N)
s
Pessoa Jurídica excluída com sucesso!
```

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

R: *A persistência em arquivos geralmente envolve a gravação de dados em arquivos de texto, binários, ou formatos como JSON ou XML. É mais simples de implementar e não requer um sistema de gerenciamento de banco de dados, Os dados em arquivos são, geralmente, acessados de forma linear, o que pode ser menos eficiente para grandes volumes de dados. Não há otimizações para buscas complexas como índices. Já a persistência em banco de dados Utiliza um SGBD para armazenar dados de maneira estruturada em tabelas, o que facilita a aplicação de regras de integridade, como restrições e relacionamentos, oferecem otimizações, como índices, para acesso rápido a grandes volumes de dados, e permitem operações avançadas como consultas SQL para filtragem e junções.*

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

R: *Permite escrever funções anônimas de forma concisa, facilitando o processamento de coleções de dados, especialmente ao trabalhar com a api* ***stream****. Essa abordagem reduz a verbosidade do código, deixando-o mais limpo e fácil de ler. Além disso, permite aplicar operações mais complexas de forma declarativa, como filtragem, mapeamento e ordenação, facilitando a manipulação de listas e coleções.*

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

R: *Qualquer método que seja chamado diretamente pelo main também precisa ser static, já que métodos estáticos só podem acessar outros métodos ou variáveis estáticas. Isso facilita a organização de funcionalidades que não dependem de dados de instância, evitando a necessidade de criar objetos apenas para invocar esses métodos.*

## Conclusão

A atividade proposta permitiu a compreensão e a aplicação prática dos principais conceitos relacionados à persistência de dados em Java, utilizando o JDBC para a integração com um banco de dados relacional e consolidou conhecimentos sobre como conectar e realizar operações CRUD (Create, Read, Update, Delete).

A implementação do padrão DAO trouxe uma importante estruturação ao código, separando as responsabilidades de acesso e manipulação dos dados, garantindo uma maior organização e manutenibilidade do sistema.

Além disso, o ORM realizado em Java possibilitou a conversão de objetos do domínio em registros no banco de dados, facilitando a interação entre o modelo orientado a objetos e a estrutura relacional do banco.