

Universidade Estácio de Sá

Curso: Desenvolvimento Full Stack

Disciplina: Back-end Sem Banco Não Tem

Turma: 01.23

Semestre Letivo: 3°

Aluno: Lucas de Oliveira dos Santos

Repositório: <https://github.com/Lucasph3/Estacio/tree/main/mundo%20-%20miss%C3%A3o%20>

1. Título da Prática:

RPG0016 - BackEnd sem banco não tem

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

2. Objetivos da Prática:

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

3. Códigos do roteiro:

Arquivo: [Pessoa.java](#)

```
package cadastrbd.model;

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

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

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

```
        this.estado = estado;
        this.telefone = telefone;
        this.email = email;
    }
}
```

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

```
public int getId() {
    return id;
}
```

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

```
public String getNome() {
    return nome;
}
```

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

public String getLogradouro() {
    return logradouro;
}

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

public String getCidade() {
    return cidade;
}

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

public String getEstado() {
    return estado;
}

public void setEstado(String estado) {
    this.estado = estado;
}

public String getTelefone() {
    return telefone;
}

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

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}
}
```

Arquivo: [PessoaFisica.java](#)

```
package cadastrobd.model;

public class PessoaFisica extends Pessoa {
    private String cpf;

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

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

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

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

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

Arquivo: [PessoaJuridica.java](#)

```

package cadastrobd.model;

public class PessoaJuridica extends Pessoa {
    private String cnpj;

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

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

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

    public String getCnpj() {
        return cnpj;
    }

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

```

Arquivo: [ConectorBD.java](#)

```

package cadastrobd.model.util;

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

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

```

```

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

    public static Connection getConnection() {
        try {
            // Carrega o driver JDBC na memória
            Class.forName(DRIVER).newInstance();
            // Retorna uma conexão com o banco de dados

```

```

        return DriverManager.getConnection(URL, USER, PASSWORD);
    } catch (ClassNotFoundException | SQLException e) {
        System.out.println("Erro ao conectar com o banco de dados: " +
            e.getMessage()); return null;
    } catch (InstantiationException e) {
        throw new RuntimeException(e);
    } catch (IllegalAccessException e) {
        throw new RuntimeException(e);
    }
}

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

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

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

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

```

```

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

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

```

Arquivo: [SequenceManager.java](#)

```

package cadastrbd.model.util;

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

public class SequenceManager {

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

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

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

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

```

```

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

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

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

```

Arquivo: [PessoaFisicaDAO.java](#)

```

package cadastrobd.model;

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

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

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

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

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

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

            ResultSet resultSet = ConectorBD.getSelect(prepared);

```



```

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

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

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

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

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

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

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

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

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

        // Percorre o ResultSet enquanto houver dados
        while (resultSet != null && resultSet.next()) {
            // Cria um objeto PessoaFisica com os dados obtidos do ResultSet
            PessoaFisica pessoaFisica = criaPessoaFisica(resultSet);
            pessoas.add(pessoaFisica);
        }
    }
}

```

```

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

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

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

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

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

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

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

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

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

```

```

    }

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

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

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

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

```

```

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

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

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

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

```

```

        prepared.setInt(7, pessoaFisica.getId());

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

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

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

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

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

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

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

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

        // Obtém um objeto PreparedStatement com o SQL criado e o id fornecido como parâmetro

```

```

        PreparedStatement prepared = ConectorBD.getPrepared(conexao, sql);
        prepared.setInt(1, id);

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

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

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

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

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

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

```

```

package cadastrobd.model;

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

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

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

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

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

```

```

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

```

```

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

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

```

```

// Se o ResultSet for nulo ou vazio, fecha os objetos PreparedStatement e Connection
e retorna null
ConectorBD.close(prepared);
ConectorBD.close(conexao);
return null;
} catch (SQLException e) {
    // Se ocorrer algum erro, imprime a mensagem no console e retorna null

```

```

        System.out.println("Erro ao obter a pessoa jurídica pelo id: " + e.getMessage());
        return null;
    }
}

```

```

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

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

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

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

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

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

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

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

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

```

```

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

        if (nextId == -1) {

```

```

        // Se não foi possível obter o próximo id da sequência, retorna false
        return false;
    }

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

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

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

    // Obtém um objeto PreparedStatement com o SQL criado e os dados da pessoa jurídica fornecida
    como parâmetro
    PreparedStatement prepared = ConectorBD.getPrepared(conexao, sql);
    prepared.setInt(1, pessoaJuridica.getId());
    prepared.setString(2, pessoaJuridica.getNome());
    prepared.setString(3, pessoaJuridica.getTelefone());
    prepared.setString(4, pessoaJuridica.getEmail());
    prepared.setString(5, pessoaJuridica.getLogradouro());
    prepared.setString(6, pessoaJuridica.getCidade());
    prepared.setString(7, pessoaJuridica.getEstado());
    // Executa a inserção e verifica se foi bem sucedida
    if (prepared.executeUpdate() <= 0) {
        // Se a inserção na tabela Pessoa falhou, fecha os objetos PreparedStatement e
        Connection e retorna false
        ConectorBD.close(prepared);
        ConectorBD.close(conexao);
        return false;
    }

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

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

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

```



```

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

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

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

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

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

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

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

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

```

```

        prepared.setInt(2, pessoaJuridica.getId());

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

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

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

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

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

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

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

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

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

```

```

        prepared.setInt(1, id); // Substitua 1 pelo índice do campo id na tabela Pessoa

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

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

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

```

Arquivo: [CadastroBD.java](#)

```

package cadastrobd;

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

import java.util.List;
import java.util.Scanner;

public class CadastroBD {
    private static Scanner scanner = new Scanner(System.in);
    private static PessoaFisicaDAO pessoaFisicaDAO = new PessoaFisicaDAO();
    private static PessoaJuridicaDAO pessoaJuridicaDAO = new PessoaJuridicaDAO();

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        int opcao = -1;
        while (opcao != 0) {
            System.out.println("=====");
            System.out.println("Selecione uma opção:");
            System.out.println("1 - Incluir Pessoa");
            System.out.println("2 - Alterar Pessoa");
            System.out.println("3 - Excluir Pessoa");
            System.out.println("4 - Exibir pelo id");

            System.out.println("5 - Exibir todos");
            System.out.println("0 - Finalizar a execução");

            opcao = Integer.parseInt(scanner.nextLine());

            System.out.println("=====");
            switch (opcao) {
                case 1:
                    inserirPessoa();
                    break;
                case 2:
                    alterarPessoa();
                    break;
                case 3:
                    excluirPessoa();
                    break;
                case 4:
                    exibirPessoaPeloId();
                    break;
                case 5:
                    exibirTodasAsPessoas();
                    break;
            }

            System.out.println();
        }
    }
}

```

```

    }

    private static String lerTipoDePessoa() {
        System.out.println("Escolha o tipo:\n\tPara Pessoa Física digite F\n\tPara Pessoa Jurídica digite J");
        String tipo = scanner.nextLine();

        System.out.println("=====\n");

        if (tipo.equalsIgnoreCase("F") || tipo.equalsIgnoreCase("J")) {
            return tipo;
        } else {
            System.out.println("Opção inválida, tente novamente.");
            return lerTipoDePessoa();
        }
    }

    private static PessoaFisica definirDadosPessoaFisica(PessoaFisica pessoaFisica)
    { try {
        System.out.println("Digite o nome: ");
        pessoaFisica.setNome(scanner.nextLine());
        System.out.println("Digite o cpf: ");
        pessoaFisica.setCpf(scanner.nextLine());
        System.out.println("Digite o telefone: ");
        pessoaFisica.setTelefone(scanner.nextLine());
        System.out.println("Digite o email: ");
        pessoaFisica.setEmail(scanner.nextLine());
        System.out.println("Digite o logradouro: ");
        pessoaFisica.setLogradouro(scanner.nextLine());
        System.out.println("Digite a cidade: ");
        pessoaFisica.setCidade(scanner.nextLine());
        System.out.println("Digite o estado: ");
        pessoaFisica.setEstado(scanner.nextLine());
        return pessoaFisica;
    } catch (Exception e) {
        System.out.println("Erro ao inserir os dados da Pessoa física:");
        e.printStackTrace();
        System.out.println("Por favor, tente novamente.");
        return null;
    }
}

    private static PessoaJuridica definirDadosPessoaJuridica(PessoaJuridica pessoaJuridica)
    { try {
        System.out.println("Digite o nome: ");
        pessoaJuridica.setNome(scanner.nextLine());
        System.out.println("Digite o cpf: ");
        pessoaJuridica.setCnpj(scanner.nextLine());
        System.out.println("Digite o telefone: ");
        pessoaJuridica.setTelefone(scanner.nextLine());
        System.out.println("Digite o email: ");
        pessoaJuridica.setEmail(scanner.nextLine());
        System.out.println("Digite o logradouro: ");
        pessoaJuridica.setLogradouro(scanner.nextLine());
        System.out.println("Digite a cidade: ");
        pessoaJuridica.setCidade(scanner.nextLine());
    }
}

```

```

        System.out.println("Digite o estado: ");
        pessoaJuridica.setEstado(scanner.nextLine());
        return pessoaJuridica;
    } catch (Exception e) {
        System.out.println("Erro ao inserir os dados da Pessoa Jurídica:");
        e.printStackTrace();
        System.out.println("Por favor, tente novamente.");
        return null;
    }
}

private static void inserirPessoa() {
    String tipo = lerTipoDePessoa();

    if (tipo.equalsIgnoreCase("F")) {
        PessoaFisica pessoaFisica = definirDadosPessoaFisica(new PessoaFisica());
        if (pessoaFisica == null) {
            System.out.println("Falha ao incluir Pessoa Física.");
            return;
        }
        if (pessoaFisicaDAO.incluir(pessoaFisica) == false) {
            System.out.println("Falha ao incluir Pessoa Física.");
            return;
        }
        System.out.println("Pessoa Física incluída com sucesso.");
        return;
    }
    if (tipo.equalsIgnoreCase("J")) {
        PessoaJuridica pessoaJuridica = definirDadosPessoaJuridica(new PessoaJuridica());
        if (pessoaJuridica == null) {
            System.out.println("Falha ao incluir Pessoa Jurídica.");
            return;
        }
        if (pessoaJuridicaDAO.incluir(pessoaJuridica) == false) {
            System.out.println("Falha ao incluir Pessoa Jurídica.");
            return;
        }
        System.out.println("Pessoa Jurídica incluída com sucesso.");
    }
}

private static void alterarPessoa() {
    String tipo = lerTipoDePessoa();

    if (tipo.equalsIgnoreCase("F")) {
        System.out.println("Digite o id da Pessoa Física que deseja alterar:");
        int idPessoaFisica = Integer.parseInt(scanner.nextLine());
        PessoaFisica pessoaFisica = pessoaFisicaDAO.getPessoa(idPessoaFisica);

        if (pessoaFisica == null) {
            System.out.println("Pessoa Física não encontrada.");
            return;
        }
        pessoaFisica.exibir();
        pessoaFisica = definirDadosPessoaFisica(pessoaFisica);
    }
}

```

```

        if (pessoaFisica == null) {
            System.out.println("Falha ao alterar Pessoa Física.");
            return;
        }

        if (pessoaFisicaDAO.alterar(pessoaFisica) == false) {
            System.out.println("Falha ao alterar Pessoa Física.");
            return;
        }
        System.out.println("Pessoa Física alterada com sucesso.");
        return;
    }

    if (tipo.equalsIgnoreCase("J")) {
        System.out.println("Digite o id da Pessoa Jurídica que deseja alterar:");
        int idPessoaJuridica = Integer.parseInt(scanner.nextLine());
        PessoaJuridica pessoaJuridica = pessoaJuridicaDAO.getPessoa(idPessoaJuridica);
        if (pessoaJuridica == null) {
            System.out.println("Pessoa Jurídica não encontrada.");
            return;
        }

        pessoaJuridica.exibir();
        pessoaJuridica = definirDadosPessoaJuridica(pessoaJuridica);

        if (pessoaJuridica == null) {
            System.out.println("Falha ao alterar Pessoa Jurídica.");
            return;
        }

        if (pessoaJuridicaDAO.alterar(pessoaJuridica) == false) {
            System.out.println("Falha ao alterar Pessoa Jurídica.");
            return;
        }

        System.out.println("Pessoa Jurídica alterada com sucesso.");
    }
}

private static void excluirPessoa() {
    String tipo = lerTipoDePessoa();

    if (tipo.equalsIgnoreCase("F")) {
        System.out.println("Digite o id da Pessoa Física que deseja excluir:");
        int idPessoaFisica = Integer.parseInt(scanner.nextLine());
        PessoaFisica pessoaFisica = pessoaFisicaDAO.getPessoa(idPessoaFisica);

        if (pessoaFisica == null) {
            System.out.println("Pessoa Física não encontrada.");
            return;
        }

        if (pessoaFisicaDAO.excluir(idPessoaFisica) == false) {
            System.out.println("Falha ao excluir Pessoa Física.");
        }
    }
}

```

```

        return;
    }

    System.out.println("Pessoa Física excluída com sucesso.");
    return;
}

if (tipo.equalsIgnoreCase("J")) {
    System.out.println("Digite o id da Pessoa Jurídica que deseja excluir:");
    int idPessoaJuridica = Integer.parseInt(scanner.nextLine());
    PessoaJuridica pessoaJuridica = pessoaJuridicaDAO.getPessoa(idPessoaJuridica);

    if (pessoaJuridica == null) {
        System.out.println("Pessoa Jurídica não encontrada.");
        return;
    }

    if (pessoaJuridicaDAO.excluir(idPessoaJuridica) == false) {
        System.out.println("Falha ao excluir Pessoa Jurídica.");
        return;
    }

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

}

private static void exibirPessoaPeloId() {
    String tipo = lerTipoDePessoa();

    if (tipo.equalsIgnoreCase("F")) {
        System.out.println("Digite o id da Pessoa Física que deseja exibir: ");
        int idPessoaFisica = Integer.parseInt(scanner.nextLine());
        PessoaFisica pessoaFisica = pessoaFisicaDAO.getPessoa(idPessoaFisica);

        if (pessoaFisica == null) {
            System.out.println("Pessoa Física não encontrada.");
            return;
        }
        pessoaFisica.exibir();
        return;
    }

    if (tipo.equalsIgnoreCase("J")) {
        System.out.println("Digite o id da Pessoa Jurídica que deseja exibir: ");
        int idPessoaJuridica = Integer.parseInt(scanner.nextLine());
        PessoaJuridica pessoaJuridica = pessoaJuridicaDAO.getPessoa(idPessoaJuridica);

        if (pessoaJuridica == null) {
            System.out.println("Pessoa Jurídica não encontrada.");
            return;
        }
        pessoaJuridica.exibir();
    }
}
}

```



```

private static void exibirTodasAsPessoas() {
    String tipo = lerTipoDePessoa();

    if (tipo.equalsIgnoreCase("F")) {
        List<PessoaFisica> pessoasFisicas = pessoaFisicaDAO.getPessoas();
        if (pessoasFisicas == null) {
            System.out.println("Não existem Pessoas Físicas cadastradas.");
            return;
        }

        System.out.println("Exibindo todos os registros de Pessoas
Físicas:\n"); for (PessoaFisica pessoaFisica : pessoasFisicas) {
            System.out.println("-----");
            pessoaFisica.exibir();
        }
        return;
    }

    if (tipo.equalsIgnoreCase("J")) {
        List<PessoaJuridica> pessoasJuridicas = pessoaJuridicaDAO.getPessoas();
        if (pessoasJuridicas == null) {
            System.out.println("Não existem Pessoas Jurídicas cadastradas.");
            return;
        }

        System.out.println("Exibindo todos os registros de Pessoas
Jurídicas."); for (PessoaJuridica pessoaJuridica : pessoasJuridicas) {
            System.out.println("-----");
            pessoaJuridica.exibir();
        }
    }
}
}

```

4. Resultados da execução dos códigos

```
Run: CadastroBD x
"C:\Program Files\Java\jdk1.8.0_251\bin\java.exe" ..
=====
Selecione uma opção:
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Exibir pelo id
5 - Exibir todos
0 - Finalizar a execução
1
=====
Escolha o tipo:
--- Para Pessoa Fisica digite F
--- Para Pessoa Juridica digite J
J
=====
Exibindo todos os registros de Pessoas Fisicas:
-----
ID: 1
Nome: Joao
Logradouro: Rua 12, casa 3, Quitanda
Cidade: Riacho do Sul
Estado: PA
Telefone: 1111-1111
Email: joao@riacho.com
CPF: 11111111111
-----
ID: 12
Nome: Maria
Logradouro: Rua das Casas
Cidade: Cidade 1
Estado: AC
Telefone: 123456789
Email: maria@gmail.com
CPF: 123456789
-----
Selecione uma opção:
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Exibir pelo id
5 - Exibir todos
0 - Finalizar a execução
5
=====
Escolha o tipo:
--- Para Pessoa Fisica digite F
--- Para Pessoa Juridica digite J
J
=====
Exibindo todos os registros de Pessoas Juridicas.
-----
```

```

ID: 2
Nome: JJC
Logradouro: Rua 11, Centro
Cidade: Riacho do Norte
Estado: PA
Telefone: 1212-1212
Email: jjc@riacho.com
CNPJ: 22222222222222
=====
Selecione uma opção:
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Exibir pelo id
5 - Exibir todos
0 - Finalizar a execução
1
=====
Escolha o tipo:
--- Para Pessoa Fisica digite F
--- Para Pessoa Juridica digite J
J
=====
Digite o nome:
Comunidade das coisas
Digite o cpf:
11111111111
Digite o telefone:
123456789
Digite o email:
comunidade@gmail.com
Digite o logradouro:
Rua 11
Digite a cidade:
Cidade Novo
Digite o estado:
PA
Pessoa Juridica incluida com sucesso.
=====
Selecione uma opção:
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Exibir pelo id
5 - Exibir todos
0 - Finalizar a execução
1
=====
Escolha o tipo:
--- Para Pessoa Fisica digite F
--- Para Pessoa Juridica digite J
J
=====
```

Exibindo todos os registros de Pessoas Juridicas.

ID: 2

Nome: JJC

Logradouro: Rua 11, Centro

Cidade: Riacho do Norte

Estado: PA

Telefone: 1212-1212

Email: jjc@riacho.com

CNPJ: 22222222222222

ID: 13

Nome: Companhia das coisas

Logradouro: Rua 11

Cidade: Cidade Nova

Estado: PA

Telefone: 123456789

Email: companhia@gmail.com

CNPJ: 123456789

=====

Selecione uma opção:

1 - Incluir Pessoa

2 - Alterar Pessoa

3 - Excluir Pessoa

4 - Exibir pelo id

5 - Exibir todos

0 - Finalizar a execução

4

=====

Escolha o tipo:

—— Para Pessoa Física digite F

—— Para Pessoa Juridica digite J

J

=====

Digite o id da Pessoa Juridica que deseja exibir:

13

ID: 13

Nome: Companhia das coisas

Logradouro: Rua 11

Cidade: Cidade Nova

Estado: PA

Telefone: 123456789

Email: companhia@gmail.com

CNPJ: 123456789

```

Digite o id da Pessoa Fisica que deseja alterar:
12
ID: 12
Nome: Maria
Logradouro: Rua das Casas
Cidade: Cidade 1
Estado: AC
Telefone: 123456789
Email: maria@gmail.com
CPF: 123456789
Digite o nome:
Maria da Silva
Digite o cpf:
123456789
Digite o telefone:
987654321
Digite o email:
maria.da.silva@gmail.com
Digite o logradouro:
Rua das Muitas Casas
Digite a cidade:
Cidade 2
Digite o estado:
AC
Pessoa Fisica alterada com sucesso.

=====
Selecione uma opção:
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Exibir pelo id
5 - Exibir todos
8 - Finalizar a execução
5
=====
Escolha o tipo:
—— Para Pessoa Fisica digite F
—— Para Pessoa Juridica digite J
F
=====

Exibindo todos os registros de Pessoas Fisicas:

-----
ID: 1
Nome: Joao
Logradouro: Rua 12, casa 3, Quitanda
Cidade: Riacho do Sul
Estado: PA
Telefone: 1111-1111
Email: joao@riacho.com
CPF: 11111111111
-----
ID: 12
Nome: Maria da Silva
Logradouro: Rua das Muitas Casas
Cidade: Cidade 2
Estado: AC

```

```
-- Lista dados completos de pessoas físicas:
SELECT * FROM Pessoa p JOIN PessoaFisica pf ON p.idPessoa = pf.idPessoa;
GO

-- Lista dados completos de pessoas jurídicas:
SELECT * FROM Pessoa p JOIN PessoaJuridica pj ON p.idPessoa = pj.idPessoa;
GO
```

	idPessoa	nome	logradouro	cidade	estado	telefone	email	idPessoa	CPF
1	1	Joao	Rua 12, casa 3, Quitanda	Riacho do Sul	PA	1111-1111	joao@riacho.com	1	11111111111
2	12	Maria da Silva	Rua das Mulas Casas	Cidade 2	AC	987654321	maria.da.silva@gmail.com	12	123456789

	idPessoa	nome	logradouro	cidade	estado	telefone	email	idPessoa	CNPJ
1	2	JJC	Rua 11, Centro	Riacho do Norte	PA	1212-1212	jjc@riacho.com	2	22222222222222
2	13	Companhia das coisas	Rua 11	Cidade Nova	PA	123456789	companhia@gmail.com	13	123456789

5. Análise e Conclusão

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

Resposta: O armazenamento em arquivos normalmente é utilizada apenas para dados locais onde uma ou poucas pessoas irão acessar o mesmo arquivo ao mesmo tempo e quando não há a necessidade de alta performance nas buscas e consultas. O armazenamento em um banco de dados permite o acesso de múltiplas aplicações ao mesmo tempo e garante uma melhor performance, maior capacidade de armazenamento, maior integridade e distribuição de dados.

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

Resposta: Através desse operador é possível realizar operações que antes precisavam de múltiplas linhas, em apenas uma linha, permitindo assim, um código mais legível.

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

Resposta: Eles precisam ser marcados como static pois o método main também é static, significando que eles pertencem à classe e não à uma instância específica da classe, portanto não há como acessar uma instância da classe contendo o método main para poder chamar os demais métodos.