Etec
Bento Quirino
Campinas

Tutorial: Persistência de Dados em Java com JDBC

Justificativa

No mundo do desenvolvimento de sistemas, salvar informações de forma permanente (persistência de dados) é essencial. Compreender como conectar uma aplicação Java a um banco de dados é uma habilidade fundamental para quem está aprendendo programação, especialmente em cursos técnicos.

Objetivos

- Entender o que é persistência de dados e como usá-la com Java.
- Aprender o que é JDBC e como utilizá-lo para se conectar a um banco de dados MySQL.
- Desenvolver um projeto CRUD (Criar, Ler, Atualizar, Deletar) usando a linguagem Java.
- Utilizar o Visual Studio Code (VSCode) como ambiente de desenvolvimento.

Índice

---

1. O que é persistência de dados?

Persistência de dados significa guardar as informações de um sistema de forma que elas não se percam quando o programa for fechado. Isso normalmente é feito com o uso de bancos de dados, como o **MySQL**.

---

2. O que é JDBC?

JDBC (Java Database Connectivity) é uma API do Java que permite a comunicação entre a linguagem Java e bancos de dados relacionais, como o MySQL.

---

3. Requisitos para o projeto

- Java JDK instalado (recomenda-se JDK 17)
- MySQL instalado
- VSCode com a extensão **"Extension Pack for Java"**
- Driver JDBC para MySQL (arquivo .jar)
- Terminal ou CMD

---

4. Criando o banco de dados MySQL

Abra o MySQL e crie um banco de dados com a tabela produtos:

```
CREATE DATABASE loja;

USE loja;

CREATE TABLE produtos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100),
    preco DECIMAL(10,2)
);
```

4.1 Projeto Java simples: Conectando ao MySQL e listando produtos no console

Antes de criarmos o projeto completo com múltiplas classes, vamos fazer um teste simples para garantir que conseguimos **nos conectar ao banco de dados MySQL e listar os produtos da tabela**.

Esse exemplo é feito **em um único arquivo chamado Main.java**, com código reduzido e explicado linha por linha.

Passo a passo:

1. Certifique-se de que o banco de dados loja e a tabela produtos já estão criados (ver seção 4).
2. Coloque o driver mysql-connector-j-8.x.x.jar na pasta lib/.
3. Crie o arquivo Main.java com o seguinte conteúdo:

Main.java (versão simples)

```java
import java.sql.*;

public class Main {
    @SuppressWarnings("ConvertToTryWithResources")
    public static void main(String[] args) {
        // Dados de conexão
        String url = "jdbc:mysql://localhost:3306/loja";
        String usuario = "root";
        String senha = ""; // Substitua pela sua senha do MySQL

        try {
            // Estabelece a conexão com o banco de dados
            Connection conexao = DriverManager.getConnection(url, usuario, senha);
            System.out.println("Conexão estabelecida com sucesso!");

            // Cria e executa uma consulta SQL
            String sql = "SELECT * FROM produtos";
            Statement stmt = conexao.createStatement();
            ResultSet resultado = stmt.executeQuery(sql);

            // Exibe os produtos no console
            System.out.println("Lista de Produtos:");
            while (resultado.next()) {
                int id = resultado.getInt("id");
                String nome = resultado.getString("nome");
                double preco = resultado.getDouble("preco");

                System.out.println(id + " - " + nome + " - R$" + preco);
            }

            // Fecha a conexão
            resultado.close();
            stmt.close();
            conexao.close();

        } catch (SQLException e) {
            System.out.println("Erro ao conectar: " + e.getMessage());
        }
    }
}
```

Descrição:

Este programa Java realiza a **conexão com o banco de dados MySQL** e **lista todos os produtos da tabela produtos**. Veja o que cada parte faz:

- String url, usuario, senha: são os dados para se conectar ao banco de dados. Você deve trocar "sua_senha" pela senha do seu MySQL.
- DriverManager.getConnection(...): estabelece a conexão com o banco.
- Statement stmt: cria o comando que será enviado ao banco de dados.

- ResultSet resultado: guarda os dados retornados pela consulta SELECT * FROM produtos.
- O while (resultado.next()): percorre todos os registros retornados.
- System.out.println(...): exibe no console o ID, nome e preço de cada produto.
- resultado.close(), stmt.close(), conexao.close(): encerram a conexão com o banco e liberam os recursos.

Como compilar e executar:

No terminal, navegue até o diretório do projeto e execute:

```
javac -cp "lib/*" Main.java
java -cp "lib/*:." Main
```

No Windows, use ; em vez de ::

```
java -cp "lib/*;." Main
```

---

5. Opção 1 (Recomendada): Criar projeto Java com VSCode

1. No VSCode, pressione Ctrl + Shift + P → selecione Java: Create Java Project
2. Escolha No build tools
3. Nomeie o projeto como CrudJDBC
4. Dentro do projeto, crie a pasta lib e coloque o **driver JDBC MySQL** (mysql-connector-j-x.x.xx.jar)
5. Clique com o botão direito no projeto → Properties → Java Build Path → Add JARs... → adicione o JAR do driver
6. Crie a classe Produto.java, Conexao.java e ProdutoDAO.java e a Main.java.

---

6. Opção 2: Criar projeto via linha de comando

```
mkdir CrudJDBC
cd CrudJDBC
mkdir src
mkdir lib
cd src
code .
```

Coloque os arquivos .java dentro da pasta src, e o driver .jar dentro de lib. Para compilar:

```
javac -cp "lib/*" src/*.java
java -cp "lib/*:src" Main
```

---

7. Estrutura do projeto CRUD

```
CrudJDBC/
│
├── lib/
│   └── mysql-connector-j-8.x.x.jar
├── src/
│   ├── Produto.java
│   ├── Conexao.java
│   ├── ProdutoDAO.java
│   └── Main.java
```

---

8. Códigos comentados

Produto.java

```java
public class Produto {
    private int id;
    private String nome;
    private double preco;

    public Produto(String nome, double preco) {
        this.nome = nome;
        this.preco = preco;
    }

    // Getters e Setters
    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 double getPreco() { return preco; }
    public void setPreco(double preco) { this.preco = preco; }
}
```

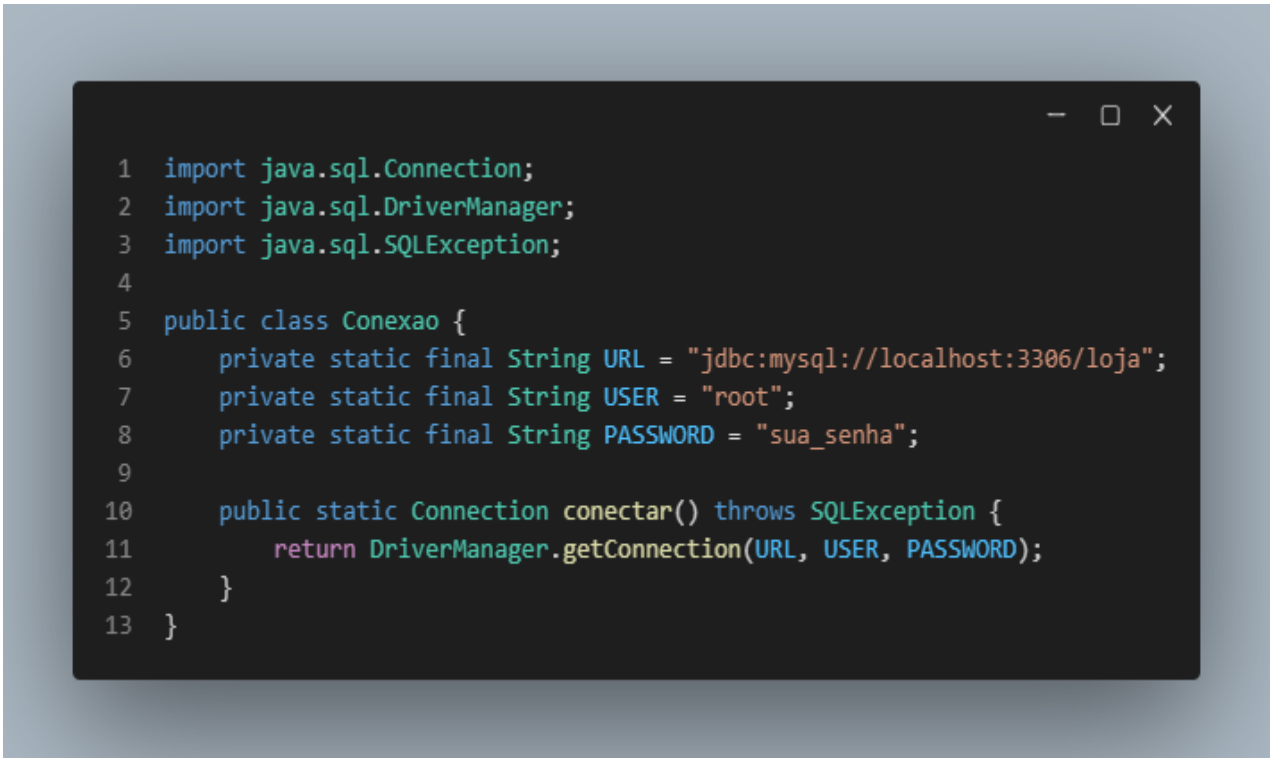
Descrição:

A classe Produto representa um produto da nossa tabela no banco de dados. Ela possui três atributos:

- id: é o identificador único do produto (gerado automaticamente pelo MySQL).
- nome: representa o nome do produto.
- preco: representa o valor do produto.

Também temos um construtor que recebe nome e preço, e os métodos getters e setters, que servem para acessar e alterar os valores dos atributos.

Conexao.java



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

public class Conexao {
    private static final String URL = "jdbc:mysql://localhost:3306/loja";
    private static final String USER = "root";
    private static final String PASSWORD = "sua_senha";

    public static Connection conectar() throws SQLException {
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }
}
```

Descrição:

A classe Conexao é responsável por criar a conexão com o banco de dados MySQL.

- O método conectar() retorna um objeto Connection, que é usado para enviar comandos SQL.
- O DriverManager.getConnection(...) utiliza a URL do banco, o usuário (root) e a senha configurada.
- Essa classe ajuda a evitar repetição de código toda vez que precisamos conectar com o banco.

ProdutoDAO.java

```java
import java.sql.*;
import java.util.*;

public class ProdutoDAO {

    @SuppressWarnings("CallToPrintStackTrace")
    public void inserir(Produto p) {
        String sql = "INSERT INTO produtos (nome, preco) VALUES (?, ?)";

        try (Connection conn = Conexao.conectar();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setString(1, p.getNome());
            stmt.setDouble(2, p.getPreco());
            stmt.executeUpdate();
            System.out.println("Produto inserido com sucesso!");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @SuppressWarnings("CallToPrintStackTrace")
    public List<Produto> listar() {
        List<Produto> lista = new ArrayList<>();
        String sql = "SELECT * FROM produtos";

        try (Connection conn = Conexao.conectar();
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {

            while (rs.next()) {
                Produto p = new Produto(rs.getString("nome"),
                                        rs.getDouble("preco"));
                p.setId(rs.getInt("id"));
                lista.add(p);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return lista;
    }
```

```java

    @SuppressWarnings("CallToPrintStackTrace")
    public void atualizar(Produto p) {
        String sql = "UPDATE produtos SET nome=?, preco=? WHERE id=?";

        try (Connection conn = Conexao.conectar();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setString(1, p.getNome());
            stmt.setDouble(2, p.getPreco());
            stmt.setInt(3, p.getId());
            stmt.executeUpdate();
            System.out.println("Produto atualizado!");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @SuppressWarnings("CallToPrintStackTrace")
    public void deletar(int id) {
        String sql = "DELETE FROM produtos WHERE id=?";

        try (Connection conn = Conexao.conectar();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, id);
            stmt.executeUpdate();
            System.out.println("Produto deletado!");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Descrição:

A classe ProdutoDAO (DAO significa *Data Access Object*) contém os métodos para acessar e manipular os dados da tabela produtos no banco MySQL. Ela realiza as 4 operações do CRUD:

- inserir(): adiciona um novo produto ao banco.
- listar(): retorna todos os produtos da tabela.
- atualizar(): altera nome e preço de um produto, dado o ID.
- deletar(): remove um produto do banco, usando seu ID.

Todos os métodos usam try-with-resources, o que garante que as conexões sejam fechadas corretamente ao final do uso.

Main.java

```java
import java.util.*;

public class Main {
    @SuppressWarnings("ConvertToTryWithResources")
    public static void main(String[] args) {
        ProdutoDAO dao = new ProdutoDAO();
        Scanner sc = new Scanner(System.in);

        System.out.println(
            "1 - Inserir | 2 - Listar | 3 - Atualizar | 4 - Deletar"
        );
        int opcao = sc.nextInt();
        sc.nextLine(); // Limpa buffer

        switch (opcao) {
            case 1:
                System.out.print("Nome: ");
                String nome = sc.nextLine();
                System.out.print("Preço: ");
                double preco = sc.nextDouble();
                Produto p = new Produto(nome, preco);
                dao.inserir(p);
                break;
            case 2:
                for (Produto prod : dao.listar()) {
                    System.out.println(
                        prod.getId() + " - " +
                        prod.getNome() + " - R$" +
                        prod.getPreco());
                }
                break;
```

```
32              case 3:
33                  System.out.print("ID do produto: ");
34                  int id = sc.nextInt();
35                  sc.nextLine();
36                  System.out.print("Novo nome: ");
37                  nome = sc.nextLine();
38                  System.out.print("Novo preço: ");
39                  preco = sc.nextDouble();
40                  p = new Produto(nome, preco);
41                  p.setId(id);
42                  dao.atualizar(p);
43                  break;
44              case 4:
45                  System.out.print("ID do produto a deletar: ");
46                  id = sc.nextInt();
47                  dao.deletar(id);
48                  break;
49              default:
50                  System.out.println("Opção inválida.");
51          }
52          sc.close();
53      }
54  }
55
56
```

Descrição:

Esse é o programa principal que o usuário executa. Ele funciona como um **menu no console**, permitindo escolher uma das quatro ações:

1. Inserir um novo produto.
2. Listar todos os produtos.
3. Atualizar um produto já existente.
4. Deletar um produto pelo ID.

- Usa a classe Scanner para ler as entradas do teclado.
- Cria objetos da classe Produto e chama os métodos da ProdutoDAO.
- É um exemplo de **interação básica com o usuário** via terminal.

Conclusão

Aprender a persistir dados com Java e MySQL utilizando JDBC é um passo importante para quem deseja trabalhar com desenvolvimento de software. O projeto apresentado mostra de forma prática como criar, ler, atualizar e deletar registros no banco de dados. Dominar essa conexão entre backend e banco de dados é essencial no mercado de trabalho técnico.

Referências

- Oracle. *Introdução ao JDBC*. Disponível em: https://docs.oracle.com/javase/tutorial/jdbc/
- MySQL Brasil. *Documentação oficial do MySQL*. Disponível em: https://dev.mysql.com/doc/
- Visual Studio Code. *Extensão para desenvolvimento em Java*. Disponível em: https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-java-pack
- Loiane Groner. *Curso de Java e Banco de Dados*. YouTube. Disponível em: https://www.youtube.com/user/loianeg