

Implementação de CRUD com DAO e MVC usando Interface Gráfica em Java

Integrando Model, View, Controller, DAO e JDBC em uma aplicação Swing completa

1. Introdução

Aplicações reais de cadastro combinam vários conceitos vistos isoladamente em outros momentos: o padrão **MVC** (Model-View-Controller), o padrão **DAO** (*Data Access Object*) para acesso a banco de dados via **JDBC**, e uma **interface gráfica** construída com Swing. Este texto apresenta uma implementação completa de **CRUD** (Create, Read, Update, Delete) para o cadastro de produtos, organizando o código em camadas bem definidas e mostrando como elas se comunicam.

2. Arquitetura em Camadas

A aplicação é dividida em quatro camadas principais: a **View** (interface gráfica Swing), o **Controller** (recebe ações da View e coordena o fluxo), o **DAO** (executa os comandos SQL via JDBC) e o **Model** (representa os dados, tanto em memória quanto na tabela do banco de dados).

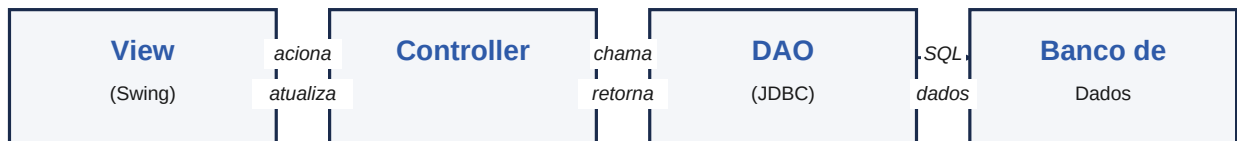


Figura 1 — Fluxo entre as camadas: View aciona o Controller, que utiliza o DAO para acessar o banco de dados e devolve o resultado para atualizar a View.

Essa organização segue o mesmo espírito do MVC tradicional, com o DAO atuando como uma extensão do Model dedicada exclusivamente à persistência dos dados, mantendo o Controller livre de comandos SQL.

3. Camada Model

O **Model** representa a entidade **Produto**, com seus atributos e validações básicas, sem nenhuma referência à interface gráfica ou ao banco de dados.

```

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

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

    public Produto(String nome, double preco) { // para produtos novos (sem id)
        this(0, nome, preco);
    }

    public int getId() { return id; }
    public String getNome() { return nome; }
    public double getPreco() { return preco; }

    public void setId(int id) { this.id = id; }
    public void setNome(String nome) { this.nome = nome; }
    public void setPreco(double preco) { this.preco = preco; }

    @Override
    public String toString() {
        return id + " | " + nome + " | R$ " + preco;
    }
}

```

4. Camada DAO

O ProdutoDAO concentra toda a comunicação com o banco de dados por meio de JDBC, implementando as quatro operações do CRUD. Nenhuma outra camada da aplicação executa comandos SQL diretamente.

```

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class ProdutoDAO {
    private static final String URL = "jdbc:mysql://localhost:3306/loja";
    private static final String USUARIO = "root";
    private static final String SENHA = "senha123";

    private Connection conectar() throws SQLException {
        return DriverManager.getConnection(URL, USUARIO, SENHA);
    }

    // CREATE
    public void inserir(Produto produto) throws SQLException {
        String sql = "INSERT INTO produto (nome, preco) VALUES (?, ?)";
        try (Connection conn = conectar();
            PreparedStatement stmt = conn.prepareStatement(sql)) {
            stmt.setString(1, produto.getNome());
            stmt.setDouble(2, produto.getPreco());
            stmt.executeUpdate();
        }
    }

    // READ
    public List<Produto> listarTodos() throws SQLException {
        String sql = "SELECT id, nome, preco FROM produto ORDER BY id";
        List<Produto> produtos = new ArrayList<>();

        try (Connection conn = conectar();
            PreparedStatement stmt = conn.prepareStatement(sql);
            ResultSet rs = stmt.executeQuery()) {

            while (rs.next()) {
                produtos.add(new Produto(
                    rs.getInt("id"), rs.getString("nome"), rs.getDouble("preco")
                ));
            }
        }
        return produtos;
    }
}

```

Continuação da mesma classe ProdutoDAO, com as operações de atualização e remoção:

```

public class ProdutoDAO { // continuação: mesma classe do bloco anterior

    // UPDATE
    public void atualizar(Produto produto) throws SQLException {
        String sql = "UPDATE produto SET nome = ?, preco = ? WHERE id = ?";
        try (Connection conn = conectar();
            PreparedStatement stmt = conn.prepareStatement(sql)) {
            stmt.setString(1, produto.getNome());
            stmt.setDouble(2, produto.getPreco());
            stmt.setInt(3, produto.getId());
            stmt.executeUpdate();
        }

    // DELETE
    public void remover(int id) throws SQLException {
        String sql = "DELETE FROM produto WHERE id = ?";
        try (Connection conn = conectar();
            PreparedStatement stmt = conn.prepareStatement(sql)) {
            stmt.setInt(1, id);
            stmt.executeUpdate();
        }
    }
}

```

5. Camada View

A `ProdutoView` é responsável apenas por exibir os componentes gráficos (campos de texto, botões e uma tabela) e expor métodos para que o `Controller` leia ou atualize essas informações — sem conter nenhuma lógica de negócio ou acesso a dados.

```

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;

public class ProdutoView extends JFrame {
    private JTextField campoNome = new JTextField(15);
    private JTextField campoPreco = new JTextField(8);
    private JTable tabela;
    private DefaultTableModel modeloTabela;

    public JButton botaoSalvar = new JButton("Salvar");
    public JButton botaoAtualizar = new JButton("Atualizar");
    public JButton botaoRemover = new JButton("Remover");

    public ProdutoView() {
        super("Cadastro de Produtos");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout(10, 10));

        JPanel formulario = new JPanel(new GridLayout(2, 4, 5, 5));
        formulario.add(new JLabel("Nome:"));
        formulario.add(campoNome);
        formulario.add(new JLabel("Preço:"));
        formulario.add(campoPreco);
        formulario.add(botaoSalvar);
        formulario.add(botaoAtualizar);
        formulario.add(botaoRemover);

        modeloTabela = new DefaultTableModel(
            new String[]{"ID", "Nome", "Preço"}, 0
        );
        tabela = new JTable(modeloTabela);

        add(formulario, BorderLayout.NORTH);
        add(new JScrollPane(tabela), BorderLayout.CENTER);

        pack();
    }
}

```

Continuação da mesma classe ProdutoView, com os métodos usados pelo Controller para ler e atualizar a interface:

```

public class ProdutoView extends JFrame { // continuação da classe anterior

    public void atualizarTabela(java.util.List<Produto> produtos) {
        modeloTabela.setRowCount(0);
        for (Produto p : produtos) {
            Object[] linha = {p.getId(), p.getNome(), p.getPreco()};
            modeloTabela.addRow(linha);
        }
    }

    public String getNomeDigitado() { return campoNome.getText().trim(); }
    public double getPrecoDigitado() {
        return Double.parseDouble(campoPreco.getText());
    }
    public void limparCampos() { campoNome.setText(""); campoPreco.setText(""); }

    public int getIdSelecionado() {
        int linha = tabela.getSelectedRow();
        if (linha == -1) return -1;
        return (int) modeloTabela.getValueAt(linha, 0);
    }

    public void exibirErro(String mensagem) {
        JOptionPane.showMessageDialog(
            this, mensagem, "Erro", JOptionPane.ERROR_MESSAGE
        );
    }
}

```

6. Camada Controller

O ProdutoController conecta os botões da View aos métodos do DAO, tratando eventuais erros (como SQLException) e atualizando a View com os dados mais recentes após cada operação.

```

import java.sql.SQLException;

public class ProdutoController {
    private ProdutoView view;
    private ProdutoDAO dao;

    public ProdutoController(ProdutoView view, ProdutoDAO dao) {
        this.view = view;
        this.dao = dao;

        this.view.botaoSalvar.addActionListener(e -> salvar());
        this.view.botaoAtualizar.addActionListener(e -> atualizar());
        this.view.botaoRemover.addActionListener(e -> remover());

        carregarProdutos(); // exibe os dados assim que a aplicação inicia
    }

    private void carregarProdutos() {
        try {
            view.atualizarTabela(dao.listarTodos()); // READ
        } catch (SQLException ex) {
            view.exibirErro("Erro ao carregar produtos: " + ex.getMessage());
        }
    }

    private void salvar() {
        try {
            String nome = view.getNomeDigitado();
            double preco = view.getPrecoDigitado();

            dao.inserir(new Produto(nome, preco)); // CREATE
            view.limparCampos();
            carregarProdutos();
        } catch (SQLException | NumberFormatException ex) {
            view.exibirErro("Erro ao salvar produto: " + ex.getMessage());
        }
    }
}

```

Continuação da mesma classe ProdutoController, com os métodos de atualização e remoção:

```

public class ProdutoController { // continuação: mesma classe do bloco anterior

    private void atualizar() {
        int id = view.getIdSelecionado();
        if (id == -1) {
            view.exibirErro("Selecione um produto na tabela.");
            return;
        }
        try {
            String nome = view.getNomeDigitado();
            double preco = view.getPrecoDigitado();
            Produto produto = new Produto(id, nome, preco);
            dao.atualizar(produto); // UPDATE
            view.limparCampos();
            carregarProdutos();
        } catch (SQLException | NumberFormatException ex) {
            view.exibirErro("Erro ao atualizar produto: " + ex.getMessage());
        }
    }

    private void remover() {
        int id = view.getIdSelecionado();
        if (id == -1) {
            view.exibirErro("Selecione um produto na tabela.");
            return;
        }
        try {
            dao.remover(id); // DELETE
            carregarProdutos();
        } catch (SQLException ex) {
            view.exibirErro("Erro ao remover produto: " + ex.getMessage());
        }
    }
}

```

7. Integrando Tudo na Classe Main

A classe principal apenas cria uma instância de cada camada e as conecta, sem conter nenhuma lógica própria — seu papel é exclusivamente montar a aplicação.

```

import javax.swing.SwingUtilities;

public class Main {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            ProdutoView view = new ProdutoView();
            ProdutoDAO dao = new ProdutoDAO();
            new ProdutoController(view, dao); // conecta View e DAO

            view.setVisible(true);
        });
    }
}

```

8. Fluxo Passo a Passo de uma Operação CRUD

Ao clicar no botão "Salvar", por exemplo, a aplicação percorre o seguinte caminho: (1) o JButton dispara o evento configurado com `addActionListener`; (2) o método `salvar()` do Controller lê os dados digitados na View; (3) o Controller cria um objeto `Produto` (Model) e

o repassa ao ProdutoDAO; (4) o DAO executa o comando INSERT via JDBC; e (5) o Controller solicita à View que se atualize, exibindo a lista mais recente vinda do banco de dados.

9. Boas Práticas

- Mantenha comandos SQL **exclusivamente** na camada DAO — nem a View, nem o Controller devem conhecer detalhes de JDBC.
- Trate exceções de banco de dados (SQLException) no Controller, exibindo mensagens amigáveis na View em vez de deixar o erro "vazar" para o usuário.
- Use PreparedStatement em todas as operações do DAO para evitar SQL Injection, mesmo em campos preenchidos por um formulário controlado.
- Sempre inicie a interface Swing dentro de SwingUtilities.invokeLater(), garantindo que ela seja construída na Event Dispatch Thread.
- Evite lógica de negócio na View: ela deve apenas capturar entradas e exibir saídas.

10. Conclusão

Combinar MVC, DAO e Swing em uma única aplicação mostra, na prática, o valor de separar responsabilidades em camadas: a View cuida apenas da apresentação, o Controller orquestra o fluxo da aplicação, o DAO isola todo o acesso ao banco de dados, e o Model representa os dados de forma independente de qualquer tecnologia de persistência ou interface. Essa organização torna a aplicação de cadastro de produtos mais fácil de testar, manter e evoluir — por exemplo, trocar o banco de dados ou a biblioteca gráfica exigiria alterações concentradas em uma única camada, sem impactar as demais.